



VISUAL
ENGINEERING
ADVANCED MOBILE
TECHNOLOGY

Fixed Scene 3D Reconstruction for Mobile Applications

Alberto Irurueta

Acknowledgments

Etiam sit amet est

Eset eiusmod tempor incididunt et labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exerc. Irure dolor in reprehend incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit.

Lorem ipsum dolor sit amet, ligula

Suspendisse nulla pretium, rhoncus tempor placerat fermentum, enim integer ad vestibulum volutpat. Nisl rhoncus turpis est, vel elit, congue wisi enim nunc ultricies sit, magna tincidunt. Maecenas aliqua maecenas ligula nostra, accumsan. Fermentum pede fringilla urna semper, pede quam scelerisque et enim in commodo, dictum a consequatur arcu.

Notations

Etiam sit amet est

Eset eiusmod tempor incididunt et labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exerc. Irure dolor in reprehend incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit.

Lorem ipsum dolor sit amet, ligula

Suspendisse nulla pretium, rhoncus tempor placerat fermentum, enim integer ad vestibulum volutpat. Nisl rhoncus turpis est, vel elit, congue wisi enim nunc ultricies sit, magna tincidunt. Maecenas aliqua maecenas ligula nostra, accumsan. Fermentum pede fringilla urna semper, pede quam scelerisque et enim in commodo, dictum a consequatur arcu.

Table of Contents

PART 1: Camera geometry and structure	1
1.- Projective Geometry	2
1.1.- Introduction	2
1.2.- 2D Projective Geometry	2
1.2.1.- Homogeneous coordinates on the image plane	2
1.2.2.- The image point	2
1.2.3.- The projective line	3
1.2.4.- Minimum distance between 2D points and lines	4
1.2.5.- Conics	6
1.2.6.- Dual Conics	8
1.2.6.1.- Line-conic intersections	8
1.2.6.2.- Line-conic tangency	10
1.2.6.3.- Relation between conic and dual conic	11
1.2.7.- Transforming conics and dual conics	11
1.3.- 3D Projective Geometry	12
1.3.1.- Homogeneous coordinates in the projective “real world”	13
1.3.2.- The projective “real world” point	13
1.3.3.- The projective plane	13
1.3.4.- The point and plane duality	14
1.3.5.- Quadrics	15

1.3.6.- Dual Quadrics	17
1.3.6.1.- Plane-quadric intersections	17
1.3.6.2.- Plane-quadric tangency	19
1.3.6.3.- Relation between quadric and dual quadric	20
1.3.7.- Transforming quadrics and dual quadrics	20
1.4.- The stratification of 3D geometry	21
1.4.1.- The projective stratum	22
1.4.1.1.- The plane at infinity	23
1.4.1.2.- The line at infinity	23
1.4.2.- The affine stratum	24
1.4.2.1.- Upgrading from projective to affine stratum	25
1.4.3.- The metric stratum	25
1.4.3.1.- Introduction to the dual absolute conic and the dual absolute quadric.	28
1.4.3.2.- Orthogonality in metric stratum	29
1.4.3.3.- Dual absolute conic and dual absolute quadric in metric stratum	32
1.4.3.4.- Invariance of the absolute conic and the dual absolute quadric respect to similarity transformations	35
1.4.3.5.- Upgrading from projective or affine to metric stratum	37
1.4.4.- The euclidean stratum	41
1.4.5.- Summary of projective geometry strata	41
2.- Camera models	44
2.1.- Introduction	44
2.2.- The projective canonical camera	44
2.2.1.- The retinal plane, principal point and focal length on the canonical camera	46
2.2.2.- The intrinsic camera parameters	47

2.2.2.1.- <i>Focal length</i>	47
2.2.2.2.- <i>Principal point offset</i>	48
2.2.2.3.- <i>CCD cameras</i>	49
2.2.3.- The extrinsic camera parameters	51
2.2.4.- Deviations from the camera model	52
2.3.- Camera decomposition	54
2.3.1.- Camera center	54
2.3.1.1.- <i>Computing the camera projection center by means of SVD</i>	55
2.3.1.2.- <i>Algebraically computing the camera projection center</i>	57
2.3.1.3.- <i>Computing the camera projection center for finite cameras</i>	58
2.3.2.- Vanishing points and image of the world origin	59
2.3.3.- The principal plane and axes planes	59
2.3.3.1.- <i>Axes planes</i>	59
2.3.3.2.- <i>The principal plane</i>	61
2.3.4.- The principal point	61
2.3.5.- The principal axis vector and camera sign	62
2.3.5.1.- <i>Camera sign normalization</i>	63
2.3.6.- Finding the camera orientation and intrinsic parameters	64
2.4.- Depth and cheirality	64
2.4.1.- Computing depth of world points respect to camera	64
2.4.2.- Computing cheirality of world points respect to camera	66
2.5.- Camera estimation	68
2.5.1.- The DLT algorithm for camera estimation	68
2.5.1.1.- <i>Least Mean Squares Error (LMSE) solution to the camera estimation</i>	71
2.5.1.2.- <i>Other considerations</i>	72

2.5.2.- Robust camera estimation using RANSAC	74
2.6.- Point triangulation	77
2.6.1.- The DLT algorithm for point triangulation	77
2.6.1.1.- Triangulating by means of inhomogeneous method.	80
2.6.2.- Point triangulation resolution	80
2.6.3.- Relation between depth and disparity	81
2.7.- Back-projection of points and lines	82
2.8.- Camera matrices, the geometry strata and their relation to conics and quadrics	83
2.9.- Camera projection matrices and image homographies	86
3.- Multiple view geometry	89
3.1.- Introduction	89
3.2.- Two-view geometry: The epipolar constraints	90
3.2.1.- The Fundamental matrix	91
3.2.1.1.- Obtaining the Fundamental matrix from pairs of cameras	92
3.2.1.2.- Obtaining the epipoles	93
3.2.2.- Computation of the Fundamental matrix	95
3.2.2.1.- The 8-point algorithm	96
3.2.2.2.- The 7-point algorithm	98
3.2.3.- Robust Fundamental matrix computation using RANSAC	104
3.2.4.- Initial projective pair of camera matrices and their relation to image homographies	106
3.2.4.1.- Proving that the initial pair of projective cameras is not unique	109
3.2.5.- Improving point correspondences	109
3.2.5.1.- The Sampson approximation	111
3.2.5.2.- The Gold Standard method	116
3.2.6.- Comparison between Fundamental matrices	124

3.2.6.1.- Frobenius norm of difference	125
3.2.6.2.- Mean Epipolar distance	125
3.2.7.- The Essential matrix	126
3.2.7.1.- Properties of the Essential matrix	129
3.2.7.2.- Initial metric structure	129
3.2.8.- Degenerate cases	130
3.2.9.- Uncertainty of the Epipolar geometry	132
3.2.9.1.- Computing the covariance matrix of the fundamental matrix	132
3.2.9.2.- The envelope of epipolar lines	137
3.2.9.3.- The uncertainty of epipoles and matched image points	142
3.2.9.4.- The uncertainty of projective 3D reconstruction	144
3.3.- Three-view geometry	147
4.- Structure and motion	149
4.1.- Introduction	149
4.2.- Initial structure and motion	149
4.2.1.- Obtaining an initial quasi-euclidean reconstruction	150
4.3.- Adding more views	152
4.3.1.- Refining and extending structure	153
4.4.- Building a mesh of 3D points	154
4.5.- Texture computation	156
4.5.1.- Reducing glares and shadows	156
4.5.2.- Computing super-resolution textures	157
4.6.- Closure	159
5.- Self-calibration	161
5.1.- Introduction	161

5.2.- The absolute dual quadric and the dual image of the absolute conic	161
5.3.- Self-calibration methods	163
5.3.1.- Calibration using the absolute dual quadric	164
5.3.1.1.- Assuming constant intrinsic parameters	170
5.3.1.2.- Assuming variable intrinsic parameters	171
5.3.2.- The Kruppa equations	171
5.4.- Critical motion sequences	175
5.5.- Closure	175
PART 2: Photogrammetry	176
Introduction	177
6.- Image pre-processing	179
6.1.- Introduction	179
6.2.- Blurring	179
6.3.- Birchfield and Tomasi dissimilarity measure	179
6.4.- Nonparametric Image Transformations	181
6.4.1.- RANK transform	181
6.4.2.- Census transform	182
6.4.2.1.- Sparse Census Transform	183
6.5.- Closure	183
7.- Match measures	185
7.1.- Introduction	185
7.2.- Distances of image regions	185
7.2.1.- Matching distances for intensity images	185
7.2.2.- Matching distances for multichannel images	188
7.3.- Distances for bit strings	188

7.4.- Statistical Distances	189
7.5.- Distances based on information theory	190
7.6.- Improving matching values	190
7.6.1.- Truncation	191
7.6.2.- Birchfield-Tomasi dissimilarity measure	191
7.7.- Aggregation of match measures	191
7.7.1.- Gaussian windowing	191
7.7.2.- Adaptive windowing	192
7.7.2.1.- Window size based on pixel similarity	192
7.7.2.2.- Window size based on coherence of structural tensor	193
7.8.- Efficient Computation of Distances	194
7.9.- A robust match measure	196
7.9.1.- The Log-polar transformation	196
7.9.2.- Matching of the log-polar transformation	198
7.10.- Closure	199
8.- Feature extraction and matching	200
8.1.- Introduction	200
8.2.- Gradient estimation of an image	200
8.2.1.- Simoncelli filters	201
8.2.2.- Savitzky-Golay filters	203
8.3.- The structural tensor	206
8.3.1.- Computation of Orientation	207
8.3.2.- The structural tensor on intensity images	210
8.3.2.1.- Augmented Structural Tensor and Coherence	214
8.3.3.- Extending the structural tensor to multichannel images	215

8.3.4.- Converting the structural tensor for visualization	215
8.4.- Corner detectors	216
8.4.1.- Harris corner detector	217
8.4.2.- Corner detector using the structural tensor	218
8.4.3.- SIFT	218
8.4.4.- Sub-pixel accuracy on corner detection	219
8.4.4.1.- <i>One dimensional polynomial interpolation</i>	219
8.4.4.2.- <i>Two dimensional polynomial interpolation</i>	222
8.5.- Closure	224
9.- Dense depth estimation	225
9.1.- Introduction	225
9.2.- Image rectification	226
9.2.1.- Hartley's rectification	226
9.2.1.1.- <i>Mapping the epipole to infinity</i>	226
9.2.1.2.- <i>Making epipolar lines coincident</i>	228
9.2.1.3.- <i>Constraints</i>	233
9.2.2.- Pollefe's polar rectification	235
9.2.2.1.- <i>Oriented geometry</i>	235
9.2.2.2.- <i>Determining the common region</i>	238
9.2.2.3.- <i>Determining the distance between epipolar lines</i>	239
9.2.2.4.- <i>Building the rectified images</i>	241
9.2.2.5.- <i>Hartley's vs polar rectification</i>	242
9.3.- Cost function	243
9.3.1.- Grey value constancy assumption	243
9.3.2.- Gradient constancy assumption	245

9.3.3.- Defining an energy functional	246
9.3.4.- Smoothness restriction	248
9.3.5.- Epipolar restriction	248
9.3.5.1.- <i>Epipolar restriction vs. image rectification</i>	250
9.4.- DS minimization	250
9.4.1.- Local DS minimization	250
9.4.1.1.- <i>Winner Takes All</i>	250
9.4.1.2.- <i>Evolutionary algorithms</i>	251
9.4.2.- Global DS minimization	262
9.4.2.1.- <i>Dynamic Programming</i>	263
9.4.2.2.- <i>Multiresolution</i>	269
9.5.- Dense disparity map post-processing	286
9.5.1.- Obtaining sub-pixel accuracy	286
9.5.2.- Uniqueness constraint	288
9.5.3.- Dense disparity correction	289
9.6.- Closure	290
References	295

PART 1 : Camera geometry and structure

1.- Projective Geometry

1.1.- Introduction

In our daily life we are used to the presence of common geometric entities such as points, lines, circles, etc. which even though they have a 2D nature, they can also be represented inside a 3D world. Other entities such as common surfaces like spheres or conics are also commonly seen in a 3D world, however when represented in an image, they all need to be converted to a 2D representation.

In this chapter we'll see how to mathematically represent many of these entities, and in the following chapters we will see how to convert their representation from 3D to 2D and vice-versa.

1.2.- 2D Projective Geometry

2D projective geometry (also known as $\mathbf{P}^2$) refers to the entities that are represented on a plane in 2D (i.e. image coordinates). This is usually what happens when the 3D world is projected by means of a camera into an image or "retinal" plane.

The simplest entity that can be represented on an image is a single point, which is commonly located by means of (x,y) coordinates representing its horizontal and vertical location respectively. This kind of coordinates to represent the location of a point are called *inhomogeneous coordinates*, however, it's not the only way to represent a point as will be seen next.

1.2.1.- Homogeneous coordinates on the image plane

For simplicity of computations, coordinates are also usually given in homogeneous coordinates. As will be seen in the following chapters, homogeneous coordinates help to linearize camera projections, and the conversion from inhomogeneous coordinates is simply obtained by adding one more component, which is typically one.

Because homogeneous coordinates helps to simplify many computations in matrix form, they are commonly used.

Notice that because conversion from homogeneous to inhomogeneous coordinates is done by dividing by the last homogeneous coordinate, then many entities given in homogeneous coordinates are only given up to scale, because any scale factor different from zero yields the same inhomogeneous result. An example of this situation will be seen on the next section.

1.2.2.- The image point

Points are among the simplest entities that can be represented either in 2D or 3D. However, in the 2D case, a point can be seen as a single location in one image. For digital images this usually implies that a point has the size of one pixel, and hence its location (x,y) must be integer. However, mathematically, points are infinitesimally small and can be arbitrarily located at any (x,y) pair of values. Indeed, they can even be located at infinity (when the last homogeneous coordinate is $w = 0$), as will be shown next.

Hence, point coordinates can be represented in homogeneous coordinates as follows:

$$\mathbf{m} = [x, y, w] \tag{1.1}$$

where $\mathbf{m}$ represents a point in an image and x, y, w its homogeneous coordinates. It must be noticed that homogeneous coordinates are related to the most common inhomogeneous ones as follows:

$$\tilde{\mathbf{m}} = [\tilde{x}, \tilde{y}] = \left[\frac{x}{w}, \frac{y}{w} \right] \quad (1.2)$$

In other words, we can convert homogeneous coordinates into inhomogeneous ones by dividing each component by the last homogeneous component. Conversely, we can convert inhomogeneous coordinates into homogeneous ones by assuming that $w = 1$, and hence just annexing to the inhomogeneous coordinates one further coordinate which is equal to one. The latter is true because expression (1.1) is given only up to scale. Indeed expression (1.1) could be given instead as:

$$\mathbf{m} \sim (x, y, w)^T \equiv \alpha \cdot (x, y, w)^T \equiv (x, y, 1)^T \quad (1.3)$$

where α is just any scale factor different from zero, and w can be arbitrarily set to be one.

Then conversion of (1.3) to inhomogeneous coordinates becomes:

$$\tilde{\mathbf{m}} = [\tilde{x}, \tilde{y}] = \left[\frac{\alpha x}{\alpha w}, \frac{\alpha y}{\alpha w} \right] = \left[\frac{x}{w}, \frac{y}{w} \right] \quad (1.4)$$

which is the same regardless of the scale factor α . For that reason w can be assumed part of such scale factor and be arbitrarily set to $w = 1$ for simplicity.

As a curiosity also notice that when $w = 0$, then the point represented using inhomogeneous coordinates is located at infinity.

1.2.3.- The projective line

As will be seen here, there exists a duality between points and lines in 2D. This can be seen next taking into account the equation of a line as the points (x, y) seen below:

$$y = k(x - x_0) + y_0 \quad (1.5)$$

where y stands for the vertical location, x for the horizontal location and (x_0, y_0) for a central point where the line is passing through. Expression (1.5) can also be written in homogeneous form as follows:

$$k \cdot x - y + (y_0 - k \cdot x_0) = 0 \quad (1.6)$$

This expression can be seen as the following product:

$$(x, y, 1)^T \cdot (k, -1, y_0 - k \cdot x_0) = 0 \rightarrow \mathbf{m}^T \cdot \mathbf{l} = 0 \quad (1.7)$$

where $\mathbf{m}$ is a point in 2D and $\mathbf{l}$ represents a line in 2D.

It can be seen that both, points and lines have 3 homogeneous components, however, their meaning is completely different as seen in (1.7). This is commonly called the *point and line duality*. And as we'll see in subsequent chapters it will extend to the 3D world and will affect many operations that will be related by their inverse thanks to the duality property.

Notice as well that a line passing through 2 homogeneous image points ($\mathbf{m}_1$ and $\mathbf{m}_2$) can also be defined as their cross product, just as shown below:

$$\mathbf{l} = \mathbf{m}_1 \times \mathbf{m}_2 = [\mathbf{m}_1]_{\times} \cdot \mathbf{m}_2 = [\mathbf{m}_2]_{\times}^T \cdot \mathbf{m}_1 \quad (1.8)$$

where $[\mathbf{m}]_x$ is the skew-symmetric matrix of a 3-component vector that can be used for cross product computation. Such matrix is defined as follows for a given vector $\mathbf{m} = (m_1, m_2, m_3)^T$:

$$[\mathbf{m}]_x = \begin{bmatrix} 0 & -m_3 & m_2 \\ m_3 & 0 & -m_1 \\ -m_2 & m_1 & 0 \end{bmatrix} \quad (1.9)$$

Analogously, by means of duality, the point where the intersection of two lines occurs can also be found as the cross product of the line vectors $\mathbf{m} = \mathbf{l}_1 \times \mathbf{l}_2$

1.2.4.- Minimum distance between 2D points and lines

When looking at the right side of expression (1.7), we notice that indeed such expression indicates that each point $\mathbf{m}$ within a given line $\mathbf{l}$ is located at a distance of zero from such line $\mathbf{l}$. Hence, in order to find the minimum (perpendicular) distance between a given point $\mathbf{m}$ and a given line $\mathbf{l}$, it seems logical to notice that such distance is proportional to expression (1.7) as follows:

$$d(\mathbf{m}, \mathbf{l}) \sim \mathbf{m}^T \cdot \mathbf{l} \quad (1.10)$$

where $d(\mathbf{m}, \mathbf{l})$ stands for the *signed* distance between point $\mathbf{m}$ and line $\mathbf{l}$ both expressed using homogeneous coordinates as we have explained in previous sections. Notice that provided distance expression in (1.10) has sign, which indicates whether point $\mathbf{m}$ is located at one side or the other of line $\mathbf{l}$, depending on orientation defined on line $\mathbf{l}$ (i.e. its sign).

Indeed, the minimum euclidean distance between a point and line as shown in figure 1.1. is expressed as follows:

$$d(\mathbf{m}, \mathbf{l}) = \frac{\mathbf{m}^T \cdot \mathbf{l}}{\sqrt{l_1^2 + l_2^2}} \quad (1.11)$$

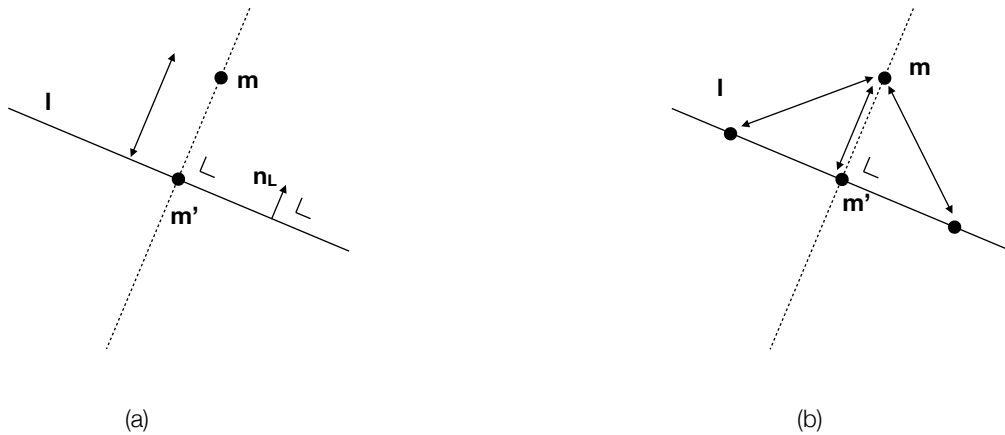


Fig. 1.1. Shortest signed distance between a point $\mathbf{m}$ and a line $\mathbf{l}$. (a) Geometrically shows that the shortest distance from $\mathbf{m}$ to $\mathbf{l}$ goes through point $\mathbf{m}'$ generating a perpendicular vector respect to line $\mathbf{l}$, or a parallel one respect to director vector $\mathbf{n}_l$. (b) Geometrically shows that other options instead of $\mathbf{m}'$ would result in longer distances, only the option depicted in (a) is the shortest one.

where l_1 and l_2 stand for the first and second component of line $\mathbf{l} = (l_1, l_2, l_3)^T$. Proof for expression (1.11) is given next.

Proof 1.1

From figure 1.1 we know that any point $\mathbf{m}$ can be expressed either in homogeneous (up to scale) or inhomogeneous coordinates as follows:

$$\mathbf{m} \equiv (x, y, 1)^T, \quad \tilde{\mathbf{m}} \equiv (x, y)^T \quad (1.12)$$

And consequently, a second point $\mathbf{m}'$ can be expressed as:

$$\mathbf{m}' \equiv (x', y', 1)^T, \quad \tilde{\mathbf{m}}' \equiv (x', y')^T \quad (1.13)$$

We also know that for any line $\mathbf{l}$ expressed in homogeneous coordinates (up to scale) as shown below:

$$\mathbf{l} \equiv (l_1, l_2, 1)^T \quad (1.14)$$

Its director vector $\mathbf{n}_L$, which is perpendicular to the locus of line $\mathbf{l}$, is expressed as:

$$\mathbf{n}_L = (l_1, l_2)^T \quad (1.15)$$

Because we are interested in the shortest distance from point $\mathbf{m}$ to line $\mathbf{l}$, which happens to be generated by vector $(\mathbf{m} - \mathbf{m}')$ which is perpendicular to line $\mathbf{l}$, and consequently, parallel to director vector $\mathbf{n}_L$. We can use the definition of dot product over the inhomogeneous coordinates of $\mathbf{m}$ and $\mathbf{m}'$ as follows:

$$\begin{aligned} \mathbf{n}_L \parallel (\tilde{\mathbf{m}} - \tilde{\mathbf{m}}') &\Rightarrow \\ \mathbf{n}_L^T (\tilde{\mathbf{m}} - \tilde{\mathbf{m}}') &= |\mathbf{n}_L| |\tilde{\mathbf{m}} - \tilde{\mathbf{m}}'| \cos 0 = |\mathbf{n}_L| |\tilde{\mathbf{m}} - \tilde{\mathbf{m}}'| \end{aligned} \quad (1.16)$$

Because vectors $\mathbf{n}_L$ and $(\mathbf{m} - \mathbf{m}')$ are parallel we can consider that their separation angle is either 0° or 180° , consequently, expression (1.16) can also be rewritten as:

$$\mathbf{n}_L^T (\tilde{\mathbf{m}} - \tilde{\mathbf{m}}') = |\mathbf{n}_L| |\tilde{\mathbf{m}} - \tilde{\mathbf{m}}'| \cos \pi = -|\mathbf{n}_L| |\tilde{\mathbf{m}} - \tilde{\mathbf{m}}'| \quad (1.17)$$

Hence, using both (1.16) and (1.17) the dot product is defined up to sign as follows:

$$\mathbf{n}_L^T (\tilde{\mathbf{m}} - \tilde{\mathbf{m}}') = \pm |\mathbf{n}_L| |\tilde{\mathbf{m}} - \tilde{\mathbf{m}}'| \quad (1.18)$$

Because the modulus of vector $(\mathbf{m} - \mathbf{m}')$ is exactly the distance we are looking for $d(\mathbf{m}, \mathbf{l})$, we can rewrite (1.18) as:

$$\begin{aligned} \mathbf{n}_L^T (\tilde{\mathbf{m}} - \tilde{\mathbf{m}}') &= \pm |\mathbf{n}_L| d(\mathbf{m}, \mathbf{l}) \Rightarrow \\ d(\mathbf{m}, \mathbf{l}) &= \pm \frac{\mathbf{n}_L^T (\tilde{\mathbf{m}} - \tilde{\mathbf{m}}')}{|\mathbf{n}_L|} \end{aligned} \quad (1.19)$$

And by expanding (1.19) we obtain:

$$d(\mathbf{m}, \mathbf{l}) = \pm \frac{\mathbf{n}_L^T (\tilde{\mathbf{m}} - \tilde{\mathbf{m}}')}{|\mathbf{n}_L|} = \pm \frac{l_1 \cdot x + l_2 \cdot y - l_1 \cdot x' - l_2 \cdot y'}{\sqrt{l_1^2 + l_2^2}} \quad (1.20)$$

Because we know that point $\mathbf{m}'$ is located within line $\mathbf{l}$, we add the following constraint using homogeneous coordinates:

$$\mathbf{l}^T \cdot \mathbf{m}' = 0 \Rightarrow l_1 \cdot x' + l_2 \cdot y' + 1 = 0 \Rightarrow 1 = -l_1 \cdot x' - l_2 \cdot y' \quad (1.21)$$

And by substituting constraint (1.21) into (1.20) we obtain:

$$d(\mathbf{m}, \mathbf{l}) = \pm \frac{l_1 \cdot x + l_2 \cdot y + 1}{\sqrt{l_1^2 + l_2^2}} = \pm \frac{\mathbf{m}^T \cdot \mathbf{l}}{\sqrt{l_1^2 + l_2^2}} \quad (1.22)$$

Where expression (1.22) is an unsigned measured of distance which is always positive. If we dropped the $\pm$ from expression (1.22) we would allow for negative distances due to negative dot products between point $\mathbf{m}$ and line $\mathbf{l}$ which would simply indicate whether point $\mathbf{m}$ is located at one side or the other of line $\mathbf{l}$. Hence, the signed shortest distance between point $\mathbf{m}$ and line $\mathbf{l}$ becomes:

$$d(\mathbf{m}, \mathbf{l}) = \frac{\mathbf{m}^T \cdot \mathbf{l}}{\sqrt{l_1^2 + l_2^2}} \quad (1.23)$$

As shown in (1.23) the signed shortest distance between a point and a line is proportional to their dot product, just as stated in (1.10).

1.2.5.- Conics

A conic is a second degree expression on $\mathbf{P}^2$ that defines the locus of all image points $\mathbf{m} = (m_1, m_2, m_3)^T$ satisfying the following homogeneous quadratic equation:

$$S(\mathbf{m}) = \mathbf{m}^T \cdot \mathbf{C} \cdot \mathbf{m} = 0 \quad (1.24)$$

where $\mathbf{C}$ is a 3x3 symmetric matrix defined up to scale that represents a given conic. Hence, a conic only depends of five independent parameters, as shown below:

$$\mathbf{C} \sim \begin{bmatrix} a & b/2 & d/2 \\ b/2 & c & e/2 \\ d/2 & e/2 & 1 \end{bmatrix} \quad (1.25)$$

where the bottom right element has been set to one to remove the scale factor dependency.

Because homogeneous image points $\mathbf{m}$ can also be expressed in terms of image coordinates (x, y) as follows:

$$\mathbf{m} = (x, y, 1)^T \quad (1.26)$$

Then, expression (1.24) can be expanded as:

$$\begin{aligned}
S(\mathbf{m}) &= \begin{bmatrix} x & y & 1 \end{bmatrix} \begin{bmatrix} a & b/2 & d/2 \\ b/2 & c & e/2 \\ d/2 & e/2 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \\
&= \begin{bmatrix} x & y & 1 \end{bmatrix} \begin{bmatrix} a \cdot x + \frac{b \cdot y}{2} + \frac{d}{2} \\ \frac{b \cdot x}{2} + c \cdot y + \frac{e}{2} \\ \frac{d \cdot x}{2} + \frac{e \cdot y}{2} + 1 \end{bmatrix} \Rightarrow \\
S(\mathbf{m}) &= a \cdot x^2 + b \cdot x \cdot y + c \cdot y^2 + d \cdot x + e \cdot y + 1 = 0
\end{aligned} \tag{1.27}$$

And depending on the values of parameters a , b , c , d and e in expression (1.27), we have the following cases:

- If $b^2 - 4ac < 0$, expression (1.27) represents an **ellipse** (unless the conic is degenerate, as in $x^2 + y^2 + 10 = 0$).
 - More specifically, if $a = c$, and $b = 0$, the equation represents a **circle**.
- If $b^2 - 4ac = 0$, expression (1.27) represents a **parabola**.
- If $b^2 - 4ac > 0$, expression (1.27) represents an **hyperbola**.
 - If also $a + c = 0$, the equation represents a **rectangular hyperbola**.

Indeed, all conic shapes stated above can be geometrically fitted within a cone as shown in figure 1.2, hence their name of conics. Their standard expressions are also given in table 1.1 for ease of understanding.

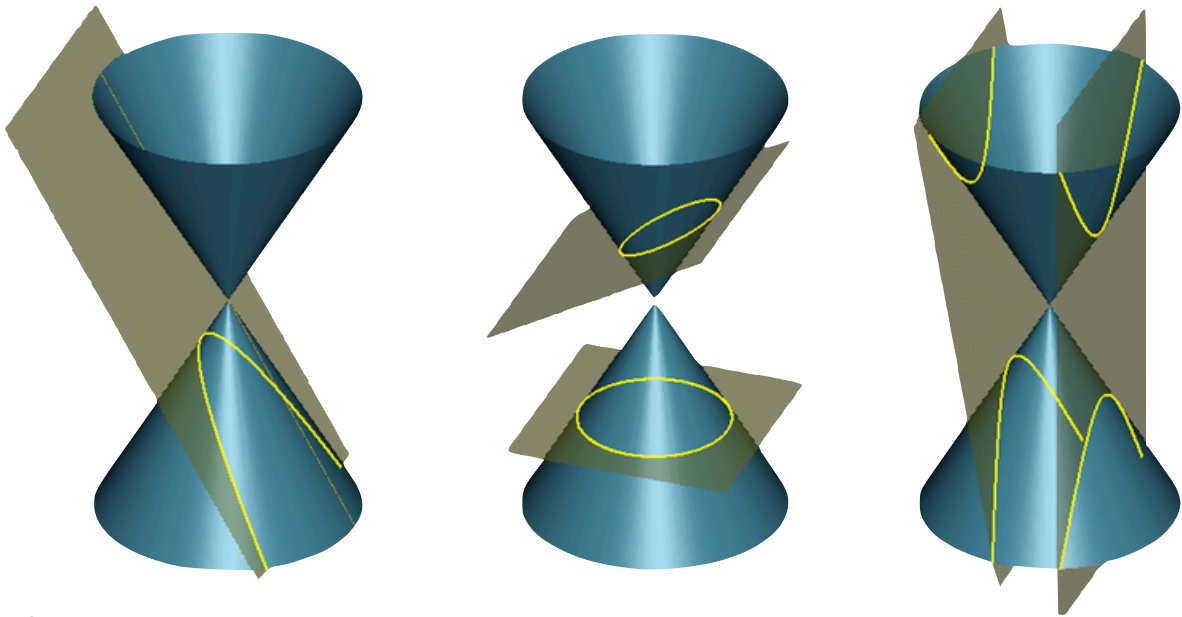


Fig. 1.2. Conic sections obtained geometrically from the intersection of a plane with a 3D cone shape. Left: Parabola (cutting plane parallel to side of cone). Middle: Circle and Ellipse. Right: Hyperbolas

Circle	$x^2 + y^2 = r^2$	(1.28)
Ellipse	$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1, \quad \frac{x^2}{b^2} + \frac{y^2}{a^2} = 1$	(1.29)
Parabola	$y^2 = 4ax, \quad x^2 = 4ay$	(1.30)
Hyperbola	$\frac{x^2}{a^2} - \frac{y^2}{b^2} = 1, \quad \frac{x^2}{b^2} - \frac{y^2}{a^2} = -1$	(1.31)
Rectangular Hyperbola	$xy = c^2$	(1.32)

Table 1.1. Summary of standard conic expressions

1.2.6.- Dual Conics

Starting from expression (1.24) that defines the homogeneous conic equation, and taking into account the duality concept between points and lines, we can also define a *conic envelope* or *dual conic* as the locus of all lines $\mathbf{l}$ satisfying the following homogeneous quadratic equation:

$$\mathbf{l}^T \cdot \mathbf{C}^* \cdot \mathbf{l} = 0 \quad (1.33)$$

where $\mathbf{C}^*$ is a 3x3 symmetric matrix defined up to scale that represents the dual conic. Hence, as in the case of the conic, a dual conic also depends on five independent parameters.

1.2.6.1.- Line-conic intersections

As depicted in figure 1.3, we can generally assume that a line $\mathbf{l}$ intersects a conic in two points $\mathbf{m}$ and $\mathbf{m}'$. Assuming that both $\mathbf{m}$ and $\mathbf{m}'$ lie within line $\mathbf{l}$, we can state from expression (1.7)) that:

$$\begin{aligned} \mathbf{l}^T \cdot \mathbf{m} &= 0 \\ \mathbf{l}^T \cdot \mathbf{m}' &= 0 \end{aligned} \quad (1.34)$$

Hence, any point of the form

$$\mathbf{m}'' = \mathbf{m} + \lambda \mathbf{m}' \quad (1.35)$$

where λ is just an arbitrary scalar value, will also lie within the same line $\mathbf{l}$ as shown below.

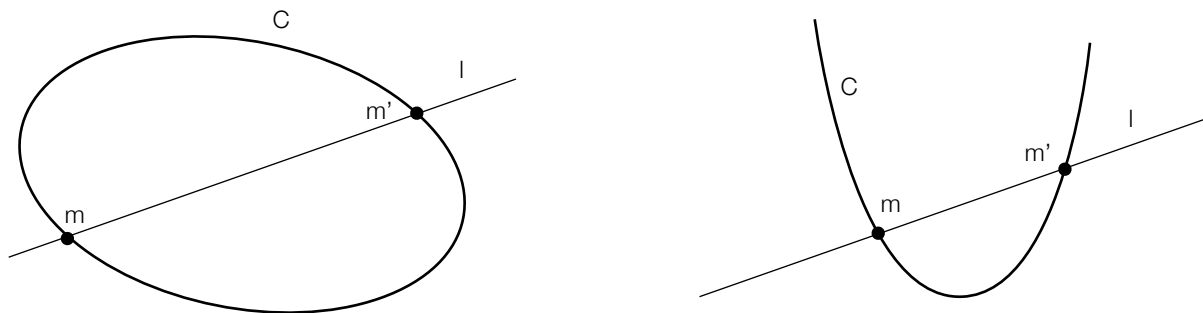


Fig. 1.3. General intersections of lines with different conics. Lines $\mathbf{l}$ typically intersect conics in two points $\mathbf{m}$ and $\mathbf{m}'$.

We know that any point $\mathbf{m}''$ that lies on line $\mathbf{l}$ must fulfill:

$$\mathbf{l}^T \cdot \mathbf{m}'' = 0 \quad (1.36)$$

By using expression (1.35) on (1.36) we obtain

$$\mathbf{l}^T \cdot \mathbf{m}'' = \mathbf{l}^T \cdot (\mathbf{m} + \lambda \mathbf{m}') = \mathbf{l}^T \cdot \mathbf{m} + \lambda \mathbf{l}^T \cdot \mathbf{m}' \quad (1.37)$$

And taking into account (1.34) on (1.37) ,

$$\mathbf{l}^T \cdot \mathbf{m}'' = \mathbf{l}^T \cdot \mathbf{m} + \lambda \mathbf{l}^T \cdot \mathbf{m}' = 0 + \lambda \cdot 0 = 0 \quad (1.38)$$

we have proved that any point $\mathbf{m}''$ lies on line $\mathbf{l}$

Then, assuming that such point $\mathbf{m}''$ lies within a conic $\mathbf{C}$, we can use expression (1.24) as follows:

$$S(\mathbf{m}'') = S(\mathbf{m} + \lambda \mathbf{m}') = 0 \quad (1.39)$$

which can be expanded as follows:

$$\begin{aligned} S(\mathbf{m}'') &= S(\mathbf{m} + \lambda \mathbf{m}') = (\mathbf{m} + \lambda \mathbf{m}')^T \cdot \mathbf{C} \cdot (\mathbf{m} + \lambda \mathbf{m}') = \\ &= \mathbf{m}^T \cdot \mathbf{C} \cdot (\mathbf{m} + \lambda \mathbf{m}') + \lambda \mathbf{m}'^T \cdot \mathbf{C} \cdot (\mathbf{m} + \lambda \mathbf{m}') = \\ &= \mathbf{m}^T \cdot \mathbf{C} \cdot \mathbf{m} + \lambda \mathbf{m}^T \cdot \mathbf{C} \cdot \mathbf{m}' + \lambda \mathbf{m}'^T \cdot \mathbf{C} \cdot \mathbf{m} + \lambda^2 \mathbf{m}'^T \cdot \mathbf{C} \cdot \mathbf{m}' = \\ &= S(\mathbf{m}) + \lambda (\mathbf{m}^T \cdot \mathbf{C} \cdot \mathbf{m}' + \mathbf{m}'^T \cdot \mathbf{C} \cdot \mathbf{m}) + \lambda^2 \cdot S(\mathbf{m}') \end{aligned} \quad (1.40)$$

Because $\mathbf{C}$ is a 3x3 symmetric matrix, then $\mathbf{C} = \mathbf{C}^T$. Consequently:

$$(\mathbf{m}^T \cdot \mathbf{C} \cdot \mathbf{m}')^T = \mathbf{m}'^T \cdot \mathbf{C}^T \cdot (\mathbf{m}^T)^T = \mathbf{m}'^T \cdot \mathbf{C} \cdot \mathbf{m} \quad (1.41)$$

Because expression (1.41) results in a scalar value, then it is also true that:

$$(\mathbf{m}^T \cdot \mathbf{C} \cdot \mathbf{m}')^T = \mathbf{m}^T \cdot \mathbf{C} \cdot \mathbf{m}' \quad (1.42)$$

Hence, expression (1.40) can be further simplified by using expressions (1.41) and (1.42) to obtain:

$$\begin{aligned} S(\mathbf{m}'') &= S(\mathbf{m} + \lambda \mathbf{m}') = \\ &= S(\mathbf{m}) + 2 \cdot \lambda \mathbf{m}^T \cdot \mathbf{C} \cdot \mathbf{m}' + \lambda^2 \cdot S(\mathbf{m}') = \\ &= S(\mathbf{m}) + 2 \cdot \lambda \cdot S(\mathbf{m}, \mathbf{m}') + \lambda^2 \cdot S(\mathbf{m}') = 0 \end{aligned} \quad (1.43)$$

Hence,

$$S(\mathbf{m}) + 2 \cdot \lambda \cdot S(\mathbf{m}, \mathbf{m}') + \lambda^2 \cdot S(\mathbf{m}') = 0 \quad (1.44)$$

where,

$$S(\mathbf{m}, \mathbf{m}') = \mathbf{m}^T \cdot \mathbf{C} \cdot \mathbf{m}' \quad (1.45)$$

We can observe that expression (1.44) is a quadratic expression in terms of λ . This means that at least two lambda values exist that fulfill such equation, which by means of (1.35) means that at least two points $\mathbf{m}''$ exist on the same line $\mathbf{l}$ that lie within conic $\mathbf{C}$. In other words, a line $\mathbf{l}$ intersects a conic $\mathbf{C}$ on two points, as depicted in fig. 1.3.

1.2.6.2.- Line-conic tangency

For the special case where the two intersection points of a line with a conic coincide, then we can easily observe geometrically that such line becomes tangent to the conic. This fact will only happen when expression (1.44) yields a single λ solution.

Because the solution of (1.44) over lambda is:

$$\lambda = \frac{-2 \cdot S(\mathbf{m}, \mathbf{m}') \pm \sqrt{4 \cdot S^2(\mathbf{m}, \mathbf{m}') - 4 \cdot S(\mathbf{m}') \cdot S(\mathbf{m})}}{2 \cdot S(\mathbf{m}')} \quad (1.46)$$

Then, expression (1.46) produces a single solution when the discriminant is zero, which can be written as,

$$S^2(\mathbf{m}, \mathbf{m}') - S(\mathbf{m}') \cdot S(\mathbf{m}) = 0 \quad (1.47)$$

Assuming that $\mathbf{m}$ point is fixed and belongs to the conic $\mathbf{C}$ (i.e. $S(\mathbf{m}) = 0$), then expression (1.47) becomes

$$S^2(\mathbf{m}, \mathbf{m}') = 0 \Rightarrow S(\mathbf{m}, \mathbf{m}') = \mathbf{m}^T \cdot \mathbf{C} \cdot \mathbf{m}' = 0 \quad (1.48)$$

Because point $\mathbf{m}$ is fixed, we need to look for the location of the other point $\mathbf{m}'$ that also intersects conic $\mathbf{C}$. We can observe that expression (1.48) is linear in terms of $\mathbf{m}'$, which indicates that there is only one possible solution for $\mathbf{m}'$. Indeed, because we desired a single intersection, $\mathbf{m}'$ must be equal to $\mathbf{m}$. This is depicted in figure 1.4.

By using expression (1.7) and taking into account duality, we can also observe that expression (1.48) can also be written as:

$$\mathbf{m}^T \cdot \mathbf{C} \cdot \mathbf{m}' = \mathbf{l}^T \cdot \mathbf{m}' = 0 \quad (1.49)$$

This latter expression is very important as it indicates that line $\mathbf{l}$ to which $\mathbf{m}'$ belongs can be expressed (up to scale) as:

$$\mathbf{l}^T \sim \mathbf{m}^T \cdot \mathbf{C} \Rightarrow \mathbf{l} \sim \mathbf{C}^T \cdot \mathbf{m} \quad (1.50)$$

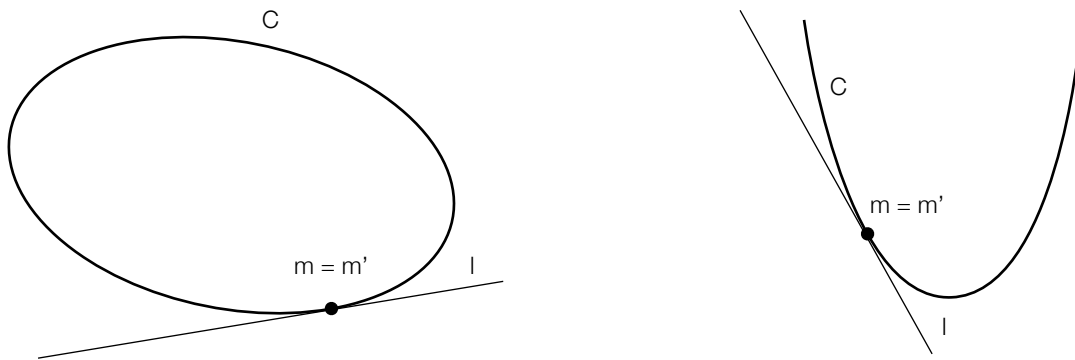


Fig. 1.4. Line tangency to conics

where $\mathbf{l}$ is a line tangent to conic $\mathbf{C}$ at point $\mathbf{m} = \mathbf{m}'$. Because conic $\mathbf{C}$ is symmetric, then expression (1.50) can also be written as:

$$\mathbf{l} \sim \mathbf{C}^T \cdot \mathbf{m} = \mathbf{C} \cdot \mathbf{m} \quad (1.51)$$

1.2.6.3.- Relation between conic and dual conic

Using and expanding expression (1.24) that relates a conic $\mathbf{C}$ and all points belonging to such conic as:

$$S(\mathbf{m}) = \mathbf{m}^T \cdot \mathbf{C} \cdot \mathbf{m} = \mathbf{m}^T \cdot \mathbf{C} \cdot \mathbf{C}^{-1} \cdot \mathbf{C} \cdot \mathbf{m} = 0 \quad (1.52)$$

And by using expressions (1.50) and (1.51) over (1.52) we obtain that:

$$\mathbf{m}^T \cdot \mathbf{C} \cdot \mathbf{C}^{-1} \cdot \mathbf{C} \cdot \mathbf{m} = \mathbf{l}^T \cdot \mathbf{C}^{-1} \cdot \mathbf{l} = 0 \quad (1.53)$$

By looking at expression (1.33), we observe that both expressions coincide as long as,

$$\mathbf{C}^* = \mathbf{C}^{-1} \quad (\text{assuming } \mathbf{C} \text{ is of full rank}) \quad (1.54)$$

We have then proved that the dual conic is the inverse matrix of a conic $\mathbf{C}$, where all the lines laying within such dual conic are tangent to conic $\mathbf{C}$. This is depicted on figure 1.5.

1.2.7.- Transforming conics and dual conics

On previous sections we have shown what conics and dual conics are, we are now going to show how transformations under a given homography $\mathbf{H}$ affect such entities.

Homographies can be seen as simple linear applications ($\mathbf{P}^2 \rightarrow \mathbf{P}^2$) that transform a given image point $\mathbf{m}$ into another image point $\mathbf{m}'$. Hence, given an image point $\mathbf{m}$ using homogeneous coordinates, an homography $\mathbf{H}$ can be defined as any 3x3 invertible matrix so that:

$$\mathbf{m}' = \mathbf{H} \cdot \mathbf{m} \quad (1.55)$$

On the other hand, by taking into account expression (1.7) that relates a point and a line as:

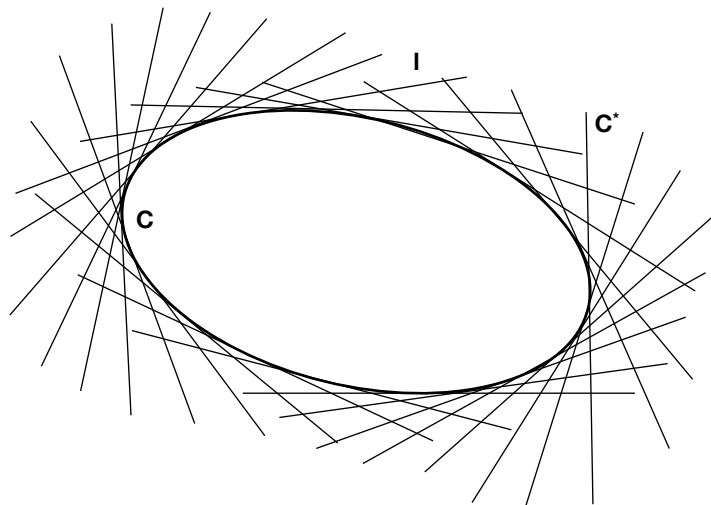


Fig. 1.5. Representation of a dual conic $\mathbf{C}^*$, which contains the locus of all lines tangent to a given conic $\mathbf{C}$.

$$\mathbf{m}^T \cdot \mathbf{l} = 0 \Rightarrow (\mathbf{H}^{-1} \cdot \mathbf{m}')^T \cdot \mathbf{l} = \mathbf{m}'^T \cdot \mathbf{H}^{-T} \cdot \mathbf{H}^T \cdot \mathbf{l} = \mathbf{m}'^T \cdot \mathbf{l} = 0 \quad (1.56)$$

We can infer that:

$$\mathbf{l} = \mathbf{H}^T \cdot \mathbf{l}' \Rightarrow \mathbf{l}' = \mathbf{H}^{-T} \cdot \mathbf{l} \quad (1.57)$$

That said, by using expression (1.55), we can transform expression (1.24) so that it becomes:

$$\mathbf{m}'^T \cdot \mathbf{C}' \cdot \mathbf{m}' = \mathbf{m}^T \cdot \mathbf{H}^T \cdot \mathbf{C}' \cdot \mathbf{H} \cdot \mathbf{m} = \mathbf{m}^T \cdot \mathbf{H}^T \cdot \mathbf{H}^{-T} \cdot \mathbf{C} \cdot \mathbf{H}^{-1} \cdot \mathbf{H} \cdot \mathbf{m} = \mathbf{m}^T \cdot \mathbf{C} \cdot \mathbf{m} = 0 \quad (1.58)$$

Then, from expression (1.58) we can observe that the transformed conic $\mathbf{C}'$ is equal to (up to scale):

$$\mathbf{C}' \sim \mathbf{H}^{-T} \cdot \mathbf{C} \cdot \mathbf{H}^{-1} \quad (1.59)$$

In a similar way the same analysis can be made over dual conics, taking into account expression (1.57), so that

$$\mathbf{l}'^T \cdot \mathbf{C}^* \cdot \mathbf{l}' = \mathbf{l}^T \cdot \mathbf{H}^{-1} \cdot \mathbf{C}^* \cdot \mathbf{H}^T \cdot \mathbf{l} = \mathbf{l}^T \cdot \mathbf{H}^{-1} \cdot \mathbf{H} \cdot \mathbf{C}^* \cdot \mathbf{H}^T \cdot \mathbf{H}^{-T} \cdot \mathbf{l} = \mathbf{l}^T \cdot \mathbf{C}^* \cdot \mathbf{l} = 0 \quad (1.60)$$

Then, from expression (1.60) we obtain that the transformed dual conic $\mathbf{C}^{*'} is (up to scale):$

$$\mathbf{C}^{*'} \sim \mathbf{H} \cdot \mathbf{C}^* \cdot \mathbf{H}^T \quad (1.61)$$

Finally, it must be noticed that transformation by an homography and duality are exchangeable, as shown below,

$$(\mathbf{C}')^* = (\mathbf{C}^*)' \quad (1.62)$$

Statement (1.62) is proved next.

Proof 1.3

We know from expression (1.54) that the dual conic is the inverse of a conic $\mathbf{C}$. Then, by using expressions (1.59) and (1.61) we obtain that:

$$(\mathbf{C}')^* = (\mathbf{C}')^{-1} = (\mathbf{H}^{-T} \cdot \mathbf{C} \cdot \mathbf{H}^{-1})^{-1} = (\mathbf{H}^{-1})^{-1} \cdot (\mathbf{C})^{-1} \cdot (\mathbf{H}^{-T})^{-1} = \mathbf{H} \cdot \mathbf{C}^{-1} \cdot \mathbf{H}^T = \mathbf{H} \cdot \mathbf{C}^* \cdot \mathbf{H}^T = \mathbf{C}^{*'} \quad (1.63)$$

which demonstrates expression (1.62) where both the transformation and duality are exchangeable.

1.3.- 3D Projective Geometry

Up to this point we have seen how to deal with 2D image points on a projective world. We have seen entities such as points, lines, conics and dual conics, as well as their relations, mostly due to duality.

It is time to add another dimension in order to deal with 3D “real world” coordinates, just as we do in our daily live.

When working with 3D projective geometry ($\mathbf{P}^3$) we refer to entities that can be represented in 3D just as any shape on the real world, hence we will usually refer to coordinates in 3D projective geometry as “real world” coordinates.

Again, the simplest entity that can be represented on a real 3D world is a single point, which is commonly located by means of (x,y,z) inhomogeneous coordinates representing its horizontal, vertical and depth location respectively. Again, as in the 2D case, we'll see that 3D real world points can also be represented using homogeneous coordinates in a similar way.

Aside from that, as in the 2D case we will see the dual of the point in 3D, which in this case is the plane, and we'll extend conics into 3D which will lead us to quadrics and dual quadrics.

1.3.1.- Homogeneous coordinates in the projective “real world”

Similarly to the 2D case, for simplicity of computations, coordinates of geometric entities are also commonly given in homogeneous coordinates.

Again, as in the 2D case, conversion from homogeneous to inhomogeneous coordinates is done by dividing by the last homogeneous coordinates, except that now, as we will see a 3D point is represented using 3 inhomogeneous coordinates and 4 homogeneous coordinates. We will also see, that as usually happened in the 2D case, homogeneous entities will only be given up to scale.

1.3.2.- The projective “real world” point

Typically, when we represent a point in 3D we usually give its coordinates using inhomogeneous coordinates as follows:

$$\tilde{\mathbf{M}} = (\tilde{X}, \tilde{Y}, \tilde{Z})^T \quad (1.64)$$

As in the 2D case, 3D real world points can also be expressed using homogeneous coordinates,

$$\mathbf{M} = (X, Y, Z, W)^T \quad (1.65)$$

Again, entities given using homogeneous coordinates will only be defined up to scale, and hence, the following is equivalent to expression (1.65),

$$\mathbf{M} \sim (X, Y, Z, W)^T \equiv \alpha (X, Y, Z, W)^T \equiv (X, Y, Z, 1)^T \quad (1.66)$$

where α is just a scale factor different from zero.

Hence, it is clear again from expression (1.66) that conversion from inhomogeneous coordinates to homogeneous ones can simply be done by adding one further coordinate which is equal to one.

The opposite conversion, from homogeneous coordinates to inhomogeneous ones can be achieved by dividing by the last homogeneous coordinate, just as follows,

$$\tilde{\mathbf{M}} = (\tilde{X}, \tilde{Y}, \tilde{Z})^T = \left(\frac{\alpha X}{\alpha W}, \frac{\alpha Y}{\alpha W}, \frac{\alpha Z}{\alpha W} \right)^T = \left(\frac{X}{W}, \frac{Y}{W}, \frac{Z}{W} \right)^T \quad (1.67)$$

where again α is just the scale factor, and as in the 2D case, W can be assumed to be part of such scale factor and can be arbitrarily set to $W = 1$ for simplicity.

Analogously to the 2D case, we must also notice that when $W = 0$, then the real world 3D point represented using inhomogeneous coordinates will be located at infinity.

1.3.3.- The projective plane

Planes in euclidean 3D world are typically given using their homogeneous equation, this equation can be obtained by taking into account that any set of 3 points $\mathbf{M}_1, \mathbf{M}_2, \mathbf{M}_3$ define a plane, and hence, any other additional point $\mathbf{M}$ that belongs to the same plane will be linear combination of the set of 3 points. Hence,

$$\mathbf{M} = \lambda_1 \cdot \mathbf{M}_1 + \lambda_2 \cdot \mathbf{M}_2 + \lambda_3 \cdot \mathbf{M}_3 \quad (1.68)$$

Consequently, the 4x4 matrix formed by column vectors $\mathbf{M}$, $\mathbf{M}_1$, $\mathbf{M}_2$ and $\mathbf{M}_3$, which are linearly dependent, will have a determinant equal to zero.

$$\mathbf{H} = \begin{bmatrix} \mathbf{M} & \mathbf{M}_1 & \mathbf{M}_2 & \mathbf{M}_3 \end{bmatrix} = \begin{bmatrix} X & X_1 & X_2 & X_3 \\ Y & Y_1 & Y_2 & Y_3 \\ Z & Z_1 & Z_2 & Z_3 \\ W & W_1 & W_2 & W_3 \end{bmatrix} \Rightarrow \quad (1.69)$$

$$\det(\mathbf{H}) = 0$$

And such determinant can be expressed by means of *Laplace's formula* in terms of the determinants of $\mathbf{H}$ matrix minors (D_{234} , D_{134} , D_{124} and D_{123}),

$$\det(H) = X \cdot D_{234} - Y \cdot D_{134} + Z \cdot D_{124} - W \cdot D_{123} = 0 \quad (1.70)$$

where D_{jkl} is the determinant of minor formed by the jkl rows of $\mathbf{H}$, just as follows:

$$D_{234} = \begin{vmatrix} Y_1 & Y_2 & Y_3 \\ Z_1 & Z_2 & Z_3 \\ W_1 & W_2 & W_3 \end{vmatrix}, \quad D_{134} = \begin{vmatrix} X_1 & X_2 & X_3 \\ Z_1 & Z_2 & Z_3 \\ W_1 & W_2 & W_3 \end{vmatrix}, \quad (1.71)$$

$$D_{124} = \begin{vmatrix} X_1 & X_2 & X_3 \\ Y_1 & Y_2 & Y_3 \\ W_1 & W_2 & W_3 \end{vmatrix}, \quad D_{123} = \begin{vmatrix} X_1 & X_2 & X_3 \\ Y_1 & Y_2 & Y_3 \\ Z_1 & Z_2 & Z_3 \end{vmatrix}$$

Then, taking into account (1.67), expression (1.70) can be rewritten as the following widely known expression:

$$A \cdot x + B \cdot y + C \cdot z + D = 0 \quad (1.72)$$

where $A = D_{234}$, $B = -D_{134}$, $C = D_{124}$ and $D = -D_{123}$ are the scalar parameters that define the plane, and x , y , z are the inhomogeneous coordinates of points within such plane. Also notice that vector

$$n_{\Pi} = (A, B, C)^T \quad (1.73)$$

is the director vector of such plane, which defines its direction and is perpendicular to the plane surface.

1.3.4.- The point and plane duality

By looking at expression (1.73) and analogously to the 2D case, we can define the following equation:

$$\Pi^T \cdot \mathbf{M} = 0 \quad (1.74)$$

where Π stands for the plane, and $\mathbf{M}$ for the point laying in such plane, both expressed using 4 homogeneous coordinates.

It is clear now, that analogously to the 2D case, the plane Π will be the dual of any 3D real world point $\mathbf{M}$, which as in the 2D case will allow us to define the dual quadrics as will be shown in the following sections.

As in section 1.2.4, expression (1.74) is also proportional to the signed shortest distance between a 3D point $\mathbf{M}$ and a plane Π , however, for the sake of brevity this will not be demonstrated as we will not need it. However, the expression of signed shortest distance between a point and a plane can be obtained analogously to the 2D case.

We can then infer that expression (1.74) defines all the points $\mathbf{M}$ laying inside plane Π , or in other words, all the points $\mathbf{M}$ located at distance zero from plane Π .

1.3.5.- Quadrics

As in the 2D case, a Quadric is a second degree expression, but this time on $\mathbf{P}^3$. A Quadric defines the locus of all 3D world points $\mathbf{M} = (M_1, M_2, M_3, M_4)^T$ satisfying the following homogeneous quadratic equation:

$$S(\mathbf{M}) = \mathbf{M}^T \cdot \mathbf{Q} \cdot \mathbf{M} = 0 \quad (1.75)$$

where $\mathbf{Q}$ is a 4x4 symmetric matrix defined up to scale that represents a quadric. Hence, a quadric only depends on 9 independent parameters, just as show below:

$$\mathbf{Q} = \begin{bmatrix} a & d/2 & f/2 & g/2 \\ d/2 & b & e/2 & h/2 \\ f/2 & e/2 & c & i/2 \\ g/2 & h/2 & i/2 & 1 \end{bmatrix} \quad (1.76)$$

where the bottom right element has been set to one to remove the scale factor dependency.

Taking into account $\mathbf{M}$ points in homogeneous coordinates as in (1.66), we can expand expression (1.75) as follows:

$$\begin{aligned} S(\mathbf{M}) &= \begin{bmatrix} X & Y & Z & 1 \end{bmatrix} \begin{bmatrix} a & d/2 & f/2 & g/2 \\ d/2 & b & e/2 & h/2 \\ f/2 & e/2 & c & i/2 \\ g/2 & h/2 & i/2 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \\ &= \begin{bmatrix} X & Y & Z & 1 \end{bmatrix} \begin{bmatrix} a \cdot X + \frac{d \cdot Y}{2} + \frac{f \cdot Z}{2} + \frac{g}{2} \\ \frac{d \cdot X}{2} + b \cdot Y + \frac{e \cdot Z}{2} + \frac{h}{2} \\ \frac{f \cdot X}{2} + \frac{e \cdot Y}{2} + c \cdot Z + \frac{i}{2} \\ \frac{g \cdot X}{2} + \frac{h \cdot Y}{2} + \frac{i \cdot Z}{2} + 1 \end{bmatrix} \Rightarrow \\ S(\mathbf{M}) &= a \cdot X^2 + b \cdot Y^2 + c \cdot Z^2 + d \cdot X \cdot Y + e \cdot Y \cdot Z + f \cdot X \cdot Z + g \cdot X + h \cdot Y + i \cdot Z + 1 = 0 \end{aligned} \quad (1.77)$$

Expression (1.77) provides 9 parameters ($a, b, c, d, e, f, g, h, i$) that allow us to control the quadric $\mathbf{Q}$ shape. And as in the 2D case, depending of their values, we can have different specific cases, which are summarized in table 1.2 and table 1.3 using their respective normalized equations.

Ellipsoid	$\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} = 1$	(1.78)
Spheroid (special case of ellipsoid)	$\frac{x^2}{a^2} + \frac{y^2}{a^2} + \frac{z^2}{b^2} = 1$	(1.79)
Sphere (special case of spheroid)	$\frac{x^2}{a^2} + \frac{y^2}{a^2} + \frac{z^2}{a^2} = 1$	(1.80)
Elliptic Paraboloid	$\frac{x^2}{a^2} + \frac{y^2}{b^2} - z = 0$	(1.81)
Circular Paraboloid (special case of elliptic paraboloid)	$\frac{x^2}{a^2} + \frac{y^2}{a^2} - z = 0$	(1.82)
Hyperbolic paraboloid	$\frac{x^2}{a^2} - \frac{y^2}{b^2} - z = 0$	(1.83)
Hyperboloid of one sheet	$\frac{x^2}{a^2} + \frac{y^2}{b^2} - \frac{z^2}{c^2} = 1$	(1.84)
Hyperboloid of two sheets	$\frac{x^2}{a^2} + \frac{y^2}{b^2} - \frac{z^2}{c^2} = -1$	(1.85)

Table 1.2. Summary of non-degenerate standard quadric expressions

Cone	$\frac{x^2}{a^2} + \frac{y^2}{b^2} - \frac{z^2}{c^2} = 0$	(1.86)
Circular Cone (special case of cone)	$\frac{x^2}{a^2} + \frac{y^2}{a^2} - \frac{z^2}{c^2} = 0$	(1.87)
Elliptic cylinder	$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$	(1.88)
Circular cylinder (special case of elliptic cylinder)	$\frac{x^2}{a^2} + \frac{y^2}{a^2} = 1$	(1.89)
Hyperbolic cylinder	$\frac{x^2}{a^2} - \frac{y^2}{b^2} = 1$	(1.90)
Parabolic cylinder	$x^2 + 2ay = 0$	(1.91)

Table 1.3. Summary of degenerate standard quadric expressions

Notice that we will be mainly interested on the non-degenerate cases (table 1.2) because as we will see later, and analogously to the 2D case, the dual quadrics will be defined as the inverse quadric matrix $\mathbf{Q}$. For that inverse matrix to exist, we will then need to ensure that quadrics are non-degenerate, and hence their matrices are invertible.

On figure 1.6 some of the main quadric cases are depicted for ease of understanding.

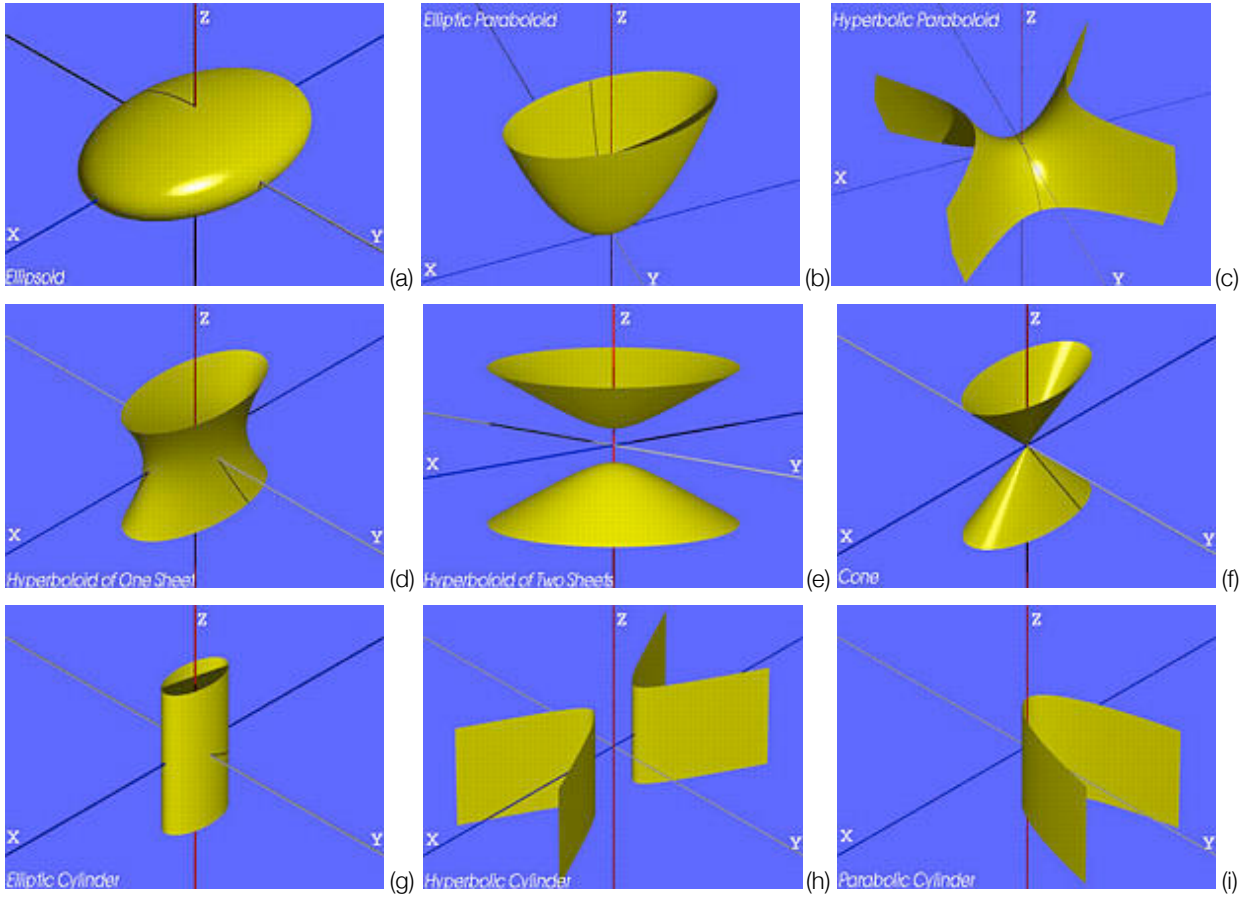


Fig. 1.6. Some quadric surfaces depicted on 3D. (a) Ellipsoid. (b) Elliptic paraboloid. (c) Hyperbolic Paraboloid. (d) Hyperboloid of one sheet. (e) Hyperboloid of two sheets. (f) Cone. (g) Elliptic cylinder. (h) Hyperbolic cylinder. (i) Parabolic cylinder.

1.3.6.- Dual Quadrics

Analogously to the 2D case, by means of expression (1.75) and taking into account the point and plane duality, we can define the following equation,

$$\Pi^T \cdot \mathbf{Q}^* \cdot \Pi = 0 \quad (1.92)$$

That defines a *dual quadric* $\mathbf{Q}^*$ as the locus of all planes satisfying expression (1.92), and where $\mathbf{Q}^*$ is, as in the quadric case, a 4x4 symmetric matrix defined up to scale and with 9 independent parameters.

1.3.6.1.- Plane-quadric intersections

As depicted in figure 1.2, the intersection of any plane Π with a quadric Q , usually results in a set of points (i.e. a conic), which all of them belong to the same plane. Because a plane can be obtained from any set of three points, using expression (1.74), we define 3 points that belong to a given plane Π .

$$\begin{aligned} \Pi^T \cdot \mathbf{M} &= 0 \\ \Pi^T \cdot \mathbf{M}' &= 0 \\ \Pi^T \cdot \mathbf{M}'' &= 0 \end{aligned} \quad (1.93)$$

Hence, any linear combination of such 3 points,

$$\mathbf{M}''' = \mathbf{M} + \lambda \mathbf{M}' + \mu \mathbf{M}'' \quad (1.94)$$

also belongs to the same plane, as proved next.

Proof 1.4

We assume that any point $\mathbf{M}'''$ as defined on (1.94) also belongs to plane Π , hence:

$$\Pi^T \cdot \mathbf{M}''' = 0 \quad (1.95)$$

and by using expressions (1.93) and (1.94) we obtain,

$$\begin{aligned} \Pi^T \cdot \mathbf{M}''' &= \Pi^T \cdot (\mathbf{M} + \lambda \mathbf{M}' + \mu \mathbf{M}'') = \Pi^T \cdot \mathbf{M} + \lambda \Pi^T \cdot \mathbf{M}' + \mu \Pi^T \cdot \mathbf{M}'' = \\ &= 0 + \lambda \cdot 0 + \mu \cdot 0 = 0 \end{aligned} \quad (1.96)$$

which fulfills equation (1.95), and consequently proves that any point $\mathbf{M}'''$ belongs to plane Π .

Assuming that the set of points $\mathbf{M}'''$ is the set of points on plane Π that intersect with quadric $\mathbf{Q}$, then such set of points $\mathbf{M}'''$ also belongs to quadric $\mathbf{Q}$, and consequently they must fulfill expression (1.75) as follows:

$$S(\mathbf{M}''') = S(\mathbf{M} + \lambda \mathbf{M}' + \mu \mathbf{M}'') = 0 \quad (1.97)$$

By expanding expression (1.97) we obtain,

$$\begin{aligned} S(\mathbf{M}''') &= S(\mathbf{M} + \lambda \mathbf{M}' + \mu \mathbf{M}'') = \\ &= (\mathbf{M} + \lambda \mathbf{M}' + \mu \mathbf{M}'')^T \cdot \mathbf{Q} \cdot (\mathbf{M} + \lambda \mathbf{M}' + \mu \mathbf{M}'') = \\ &= \mathbf{M}^T \cdot \mathbf{Q} \cdot (\mathbf{M} + \lambda \mathbf{M}' + \mu \mathbf{M}'') + \lambda \mathbf{M}'^T \cdot \mathbf{Q} \cdot (\mathbf{M} + \lambda \mathbf{M}' + \mu \mathbf{M}'') + \\ &+ \mu \mathbf{M}''^T \cdot \mathbf{Q} \cdot (\mathbf{M} + \lambda \mathbf{M}' + \mu \mathbf{M}'') = \\ &= \mathbf{M}^T \cdot \mathbf{Q} \cdot \mathbf{M} + \lambda \mathbf{M}^T \cdot \mathbf{Q} \cdot \mathbf{M}' + \mu \mathbf{M}^T \cdot \mathbf{Q} \cdot \mathbf{M}'' + \\ &+ \lambda \mathbf{M}'^T \cdot \mathbf{Q} \cdot \mathbf{M} + \lambda^2 \mathbf{M}'^T \cdot \mathbf{Q} \cdot \mathbf{M}' + \lambda \mu \mathbf{M}'^T \cdot \mathbf{Q} \cdot \mathbf{M}'' + \\ &+ \mu \mathbf{M}''^T \cdot \mathbf{Q} \cdot \mathbf{M} + \lambda \mu \mathbf{M}''^T \cdot \mathbf{Q} \cdot \mathbf{M}' + \mu^2 \mathbf{M}''^T \cdot \mathbf{Q} \cdot \mathbf{M}'' \end{aligned} \quad (1.98)$$

Analogously to expressions (1.41) and (1.42), taking into account that quadric $\mathbf{Q}$ matrices are symmetric ($\mathbf{Q} = \mathbf{Q}^T$), and taking into account that expressions like (1.92) yield a scalar value, we can safely state that,

$$S(\mathbf{M}, \mathbf{M}') = \mathbf{M}^T \cdot \mathbf{Q} \cdot \mathbf{M}' = \mathbf{M}'^T \cdot \mathbf{Q} \cdot \mathbf{M} \quad (1.99)$$

Then, expression (1.98) becomes,

$$\begin{aligned} S(\mathbf{M}''') &= S(\mathbf{M} + \lambda \mathbf{M}' + \mu \mathbf{M}'') = \\ &= \mathbf{M}^T \cdot \mathbf{Q} \cdot \mathbf{M} + \lambda^2 \mathbf{M}'^T \cdot \mathbf{Q} \cdot \mathbf{M}' + \mu^2 \mathbf{M}''^T \cdot \mathbf{Q} \cdot \mathbf{M}'' + \\ &+ 2\lambda \mathbf{M}^T \cdot \mathbf{Q} \cdot \mathbf{M}' + 2\mu \mathbf{M}^T \cdot \mathbf{Q} \cdot \mathbf{M}'' + 2\lambda \mu \mathbf{M}'^T \cdot \mathbf{Q} \cdot \mathbf{M}'' = \\ &= S(\mathbf{M}) + \lambda^2 S(\mathbf{M}') + \mu^2 S(\mathbf{M}'') + \\ &+ 2\lambda S(\mathbf{M}, \mathbf{M}') + 2\mu S(\mathbf{M}, \mathbf{M}'') + 2\lambda \mu S(\mathbf{M}', \mathbf{M}'') = 0 \end{aligned} \quad (1.100)$$

We can summarize expression (1.100) as,

$$S(\mathbf{M}) + \lambda^2 \cdot S(\mathbf{M}') + \mu^2 \cdot S(\mathbf{M}'') + 2\lambda \cdot S(\mathbf{M}, \mathbf{M}') + 2\mu \cdot S(\mathbf{M}, \mathbf{M}'') + 2\lambda\mu \cdot S(\mathbf{M}', \mathbf{M}'') = 0 \quad (1.101)$$

If we look closely at (1.101), we can see that it resembles to (1.27), but in terms of λ and μ instead of x and y . Hence we can safely state that the intersection of a plane with a quadric results in a conic.

1.3.6.2.- Plane-quadric tangency

As we stated in (1.97), the set of points $\mathbf{M}'''$ belong to the quadric $\mathbf{Q}$, consequently $\mathbf{M}$, $\mathbf{M}'$ and $\mathbf{M}''$ also belong to such quadric, then we can assert that:

$$S(\mathbf{M}) = S(\mathbf{M}') = S(\mathbf{M}'') = 0 \quad (1.102)$$

Besides, when a plane is tangent to a given quadric, analogously to the conic case, will only intersect the quadric in a single point. Or in other words, $\mathbf{M} = \mathbf{M}' = \mathbf{M}''$, and consequently,

$$S(\mathbf{M}, \mathbf{M}') = S(\mathbf{M}, \mathbf{M}'') = S(\mathbf{M}', \mathbf{M}'') = S(\mathbf{M}) = 0 \quad (1.103)$$

This can also be seen from another point of view. We could say instead that we are interested on a single solution to expression (1.101), then it could be seen as if the conic in (1.101) became a circle with radius equal to zero, in such case, both $S(\mathbf{M}')$ and $S(\mathbf{M}'')$ would be equal to the square of the radius, which is zero as shown in (1.102), and the remaining parameters $S(\mathbf{M}, \mathbf{M}')$, $S(\mathbf{M}, \mathbf{M}'')$ and $S(\mathbf{M}', \mathbf{M}'')$ must also be zero from expression (1.28). Consequently, in order to fulfill (1.101), $S(\mathbf{M})$ will also be zero, which is true because $\mathbf{M}$ belongs to the quadric $\mathbf{Q}$.

Then, as in the conic case, assuming that point $\mathbf{M}$ is fixed, taking into account the duality between points and planes and also expressions (1.103) and (1.99), we can state that any other point $\mathbf{M}'$ intersecting both the plane π and the quadric $\mathbf{Q}$ fulfills,

$$S(\mathbf{M}, \mathbf{M}') = \mathbf{M}'^T \cdot \mathbf{Q} \cdot \mathbf{M}' = \Pi^T \cdot \mathbf{M}' = 0 \quad (1.104)$$

This last assertion allows us to state that point $\mathbf{M} = \mathbf{M}' = \mathbf{M}''$ belongs to plane Π , and such point comprises the whole set of points that intersect quadric $\mathbf{Q}$. Consequently, such plane Π as shown in (1.104) is the plane tangent to quadric $\mathbf{Q}$, as shown next:

$$\Pi \sim \mathbf{Q} \cdot \mathbf{M} \quad (1.105)$$

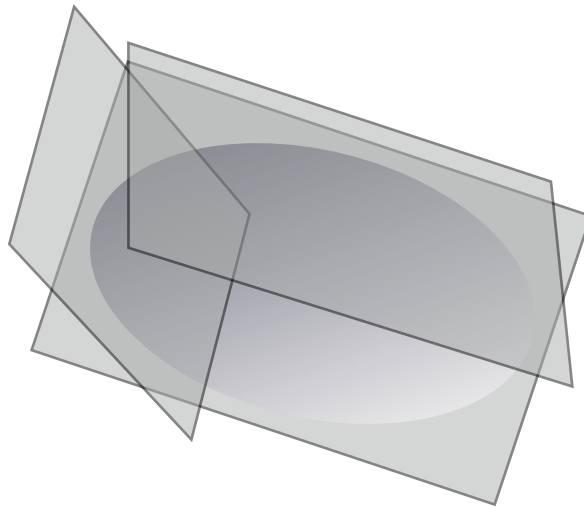


Fig. 1.7. Representation of a dual quadric $\mathbf{Q}^*$, which contains the locus of all planes tangent to a given quadric $\mathbf{Q}$.

where, we have taken into account that $\mathbf{Q}$ is symmetric ($\mathbf{Q} = \mathbf{Q}^T$), and as whenever we use homogeneous coordinates, plane Π is defined up to scale.

1.3.6.3.- Relation between quadric and dual quadric

As in section 1.2.6.3, we can use expression (1.75) as follows:

$$S(\mathbf{M}) = \mathbf{M}^T \cdot \mathbf{Q} \cdot \mathbf{M} = \mathbf{M}^T \cdot \mathbf{Q} \cdot \mathbf{Q}^{-1} \cdot \mathbf{Q} \cdot \mathbf{M} = 0 \quad (1.106)$$

And by using expression (1.105) and its transpose, then expression (1.106) becomes

$$\mathbf{M}^T \cdot \mathbf{Q} \cdot \mathbf{Q}^{-1} \cdot \mathbf{Q} \cdot \mathbf{M} = \Pi^T \cdot \mathbf{Q}^{-1} \cdot \Pi = 0 \quad (1.107)$$

Consequently, as in the conic case, by looking at expression (1.92) we can state that the dual quadric of $\mathbf{Q}$, which contains the locus of all planes tangent to quadric $\mathbf{Q}$ is equal to:

$$\mathbf{Q}^* = \mathbf{Q}^{-1} \quad (1.108)$$

where we assume that $\mathbf{Q}$ is of full rank, and hence it is a non-degenerate quadric case because $\mathbf{Q}$ is invertible, just as we saw in section 1.3.5.

Analogously to figure 1.5, we could then depict a dual quadric as the set of planes tangent to quadric $\mathbf{Q}$. This is shown in figure 1.7.

1.3.7.- Transforming quadrics and dual quadrics

Up to this point, we have seen what quadrics and dual quadrics are. Now, as in section 1.2.7, we'll see how transformations affect such entities.

Similarly to the 2D case, a 3D projective transformation on a given world point $\mathbf{M}$ into another point $\mathbf{M}'$ can be defined as any linear application ($\mathbf{P}^3 \rightarrow \mathbf{P}^3$) formed by any 4x4 invertible matrix $\mathbf{T}$ so that,

$$\mathbf{M}' = \mathbf{T} \cdot \mathbf{M} \quad (1.109)$$

Then, taking into account expression (1.71) that relates a world point and a plane we can state the following:

$$\mathbf{M}^T \cdot \Pi = 0 \Rightarrow (\mathbf{T}^{-1} \cdot \mathbf{M}')^T \cdot \Pi = \mathbf{M}'^T \cdot \mathbf{T}^{-T} \cdot \mathbf{T}^T \cdot \Pi' = \mathbf{M}'^T \cdot \Pi' = 0 \quad (1.110)$$

As in section 1.2.7, we can then state that:

$$\Pi = \mathbf{T}^T \cdot \Pi' \Rightarrow \Pi' = \mathbf{T}^{-T} \cdot \Pi \quad (1.111)$$

Then, using both expression (1.109) and (1.75), we obtain,

$$\mathbf{M}'^T \cdot \mathbf{Q}' \cdot \mathbf{M}' = \mathbf{M}^T \cdot \mathbf{T}^T \cdot \mathbf{Q}' \cdot \mathbf{T} \cdot \mathbf{M} = \mathbf{M}^T \cdot \mathbf{T}^T \cdot \mathbf{T}^{-T} \cdot \mathbf{Q} \cdot \mathbf{T}^{-1} \cdot \mathbf{T} \cdot \mathbf{M} = \mathbf{M}^T \cdot \mathbf{Q} \cdot \mathbf{M} = 0 \quad (1.112)$$

Consequently, we can observe that the transformed quadric $\mathbf{Q}'$ is equal to (up to scale):

$$\mathbf{Q}' \sim \mathbf{T}^{-T} \cdot \mathbf{Q} \cdot \mathbf{T}^{-1} \quad (1.113)$$

We can also obtain similar results on dual quadrics. Hence, using expression (1.92) and (1.111), we obtain:

$$\Pi'^T \cdot \mathbf{Q}^* \cdot \Pi' = \Pi^T \cdot \mathbf{T}^{-1} \cdot \mathbf{Q}^* \cdot \mathbf{T}^{-T} \cdot \Pi = \Pi^T \cdot \mathbf{T}^{-1} \cdot \mathbf{T} \cdot \mathbf{Q}^* \cdot \mathbf{T}^T \cdot \mathbf{T}^{-T} \cdot \Pi = \Pi^T \cdot \mathbf{Q}^* \cdot \Pi = 0 \quad (1.114)$$

Then, from expression (1.114) we can infer that the transformed dual quadric $\mathbf{Q}''$ is (up to scale):

$$\mathbf{Q}'' \sim \mathbf{T} \cdot \mathbf{Q}^* \cdot \mathbf{T}^T \quad (1.115)$$

Again, as in the conic case, we can also assert that the transformation and duality are exchangeable. Hence, similarly to expression (1.62) we can state the following:

$$(\mathbf{Q}'')^* = (\mathbf{Q}^*)' \quad (1.116)$$

where proof of expression (1.116) is analogous to Proof 1.3.

1.4.- The stratification of 3D geometry

We usually perceive the world as an Euclidean 3D space, where we know the exact 3D shape of objects. When attempting to retrieve 3D shape from images it is desirable to recover such Euclidean structure, although not always is possible if not enough references are available.

For instance, if we take pictures of an accurate model of a given object (i.e. a doll's house), it is usually impossible to determine whether the pictures were taken to a doll's house or an actual house. In such cases the real scale of the object is lost, and hence we say that shape is only recovered up to metric stratum.

Similarly, when we take pictures of a given scene, usually pictures do not store the actual orientation of the scene. In Layman terms, most cameras don't have an integrated compass or GPS, and hence orientation or location (i.e. respect to Earth), is also lost. For that reason, in subsequent chapters we will consider that the camera where we took the first picture of a given sequence will be considered the reference camera and will be located at the origin pointing towards the z's axis. That way, by means of the first camera, axis of coordinates will be set, and the coordinates of the remaining cameras will be set respect to the reference camera.

Besides of what we've said, there are further strata in 3D geometry aside from the euclidean and metric ones. Depending on the application, it will not always be needed to recover the actual 3D shape of an object, and maybe just other invariant measures (as we will see later) will be needed. In such cases, using the affine or projective stratum might just be enough.

Indeed the projective 3D space can be seen as if it was stratified in different layers or subspaces, each of which is included in the previous one. That way, the euclidean subspace would be included in the metric one, which at the same time would

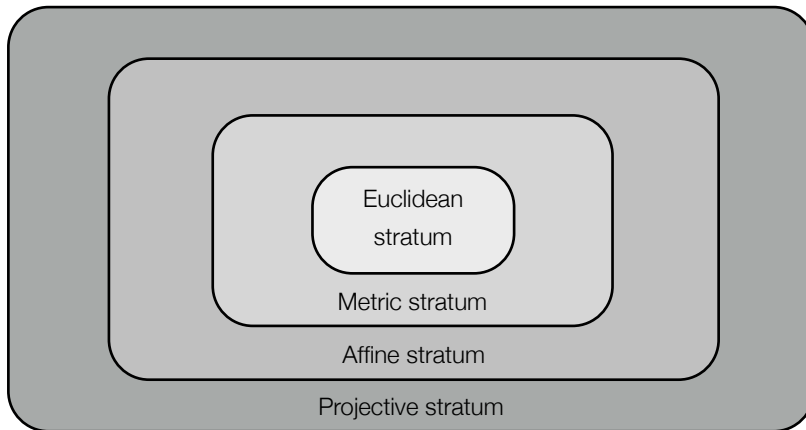


Figure 1.8. Stratification of 3D geometry

be included in the affine subspace, which finally would be included in the most general one, the projective subspace. This is depicted on figure 1.8.

Why the projective 3D space is stratified that way is out of the scope of this chapter, but will be explained with further detail on chapter 2, when projective cameras are introduced.

Next we will explain with further detail each geometry stratum, we will indicate their invariants, their properties and the transformations that can be applied to them.

1.4.1.- The projective stratum

The projective stratum is the most general among all strata in 3D projective geometry. It is the less structured one, and therefore the one that has the least number of invariants. Consequently, the group of projective transformation is the most general group and the one that has less constraints.

Hence, any 4x4 invertible matrix, as we saw in section 1.3.7, can be seen as a $\mathbf{P}^3 \rightarrow \mathbf{P}^3$ linear application that allows transformation between two geometric entities. Then, any projective transformation will have the following form:

$$\mathbf{T}_P \sim \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \\ p_{41} & p_{42} & p_{43} & p_{44} \end{bmatrix} \quad (1.117)$$

where matrix $\mathbf{T}_P$ is defined only up to scale (p_{44} could be arbitrarily set to one), and consequently only has 15 degrees of freedom.

It must be observed that within the projective stratum only relations of incidence, collinearity and tangency are projectively invariant. In Layman terms, for instance, if an object is tangent to another one, it will continue to be tangent no matter the projective transformation that we apply.

We must notice however that other properties such as parallelism, angles or scale are not preserved. If we wish to preserve any of such properties when recovering 3D shape of scene, we'll need to upgrade our geometry to the appropriate stratum.

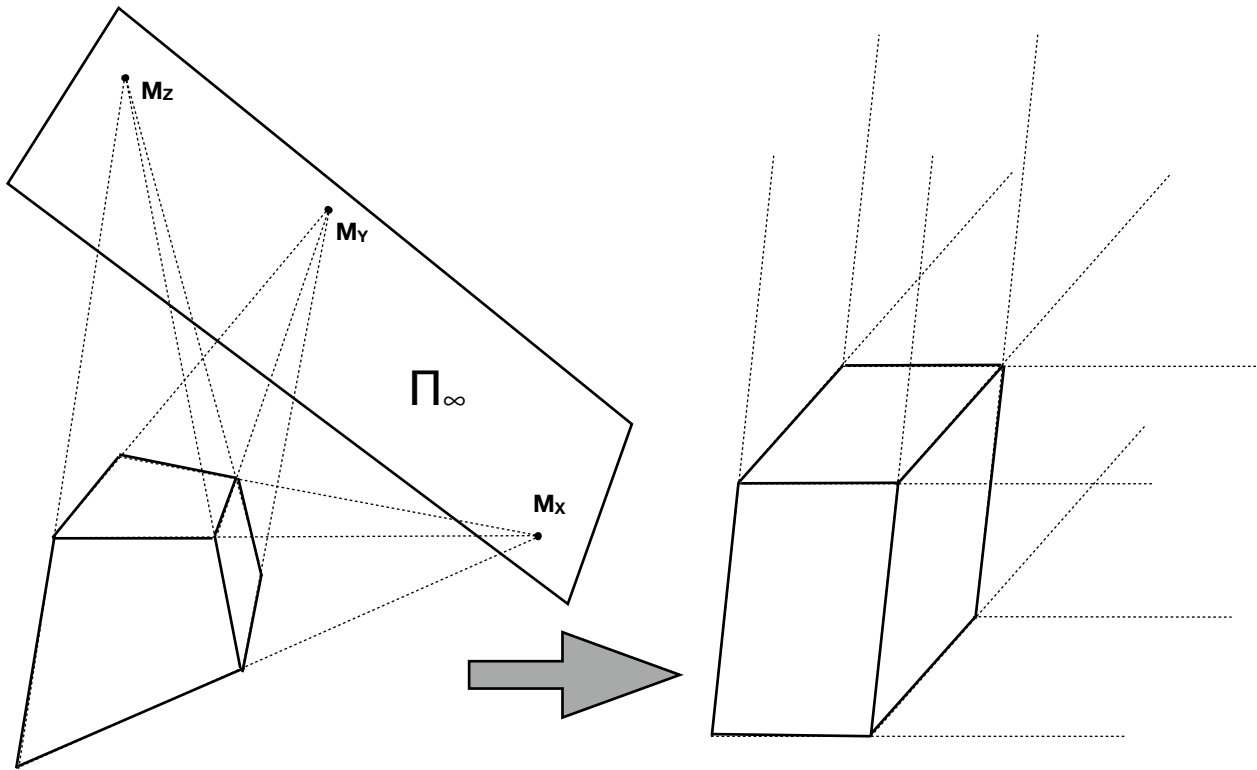


Figure 1.9. Left: Arbitrary projective representation of a cube where 3 vanishing points are identified, which at the same time they form the plane at infinity. Right: Affine representation of the cube after transformation from projective to affine. Notice that parallelism is preserved, and hence the vanishing where parallel lines intersect have been moved to infinity.

1.4.1.1.- The plane at infinity

This is a special plane that lays within the projective space and outside of any of the other stratified subspaces of figure 1.8. Taking into account expression (1.65), the plane at infinity can be defined as $W = 0$, which in Layman terms means that this plane is formed by all points located at infinity (i.e. when using inhomogeneous coordinates and $W = 0$). Hence, using expression (1.74) we can infer that the plane at infinity has the following expression:

$$\Pi_{\infty} \sim (0,0,0,1)^T \quad (1.118)$$

Expression (1.118) will always be true as long as we are already working on the canonical euclidean space, or at least on the affine stratum, because as we will see on section 1.4.2, it leaves unchanged the plane at infinity.

However, when working on any other projective space, expression (1.118) might not hold any longer. Indeed, by using expression (1.111) with any arbitrary projective transformation as in (1.117), the plane at infinity will change and will become the last row of $\mathbf{T_P}$ in (1.117).

Hence, in any arbitrary projective representation, the plane at infinity can be anywhere. In this case, when attempting to upgrade geometry from projective to affine, we will need to find the appropriate transformation that transforms the plane at infinity on any arbitrary projective representation into its canonical form in (1.118). For that reason we will first need to identify the plane at infinity on any arbitrary projective representation.

As it will also be explained in section 1.4.2, given some affine properties the plane at infinity can be identified. For instance, since we know that parallel lines only intersect at infinity, then, by identifying any 3 vanishing points where such parallel lines intersect, we can find the plane at infinity. This is depicted on figure 1.9.

1.4.1.2.- The line at infinity

Because the real 3D world can be projected into 2D images, it also exists another entity which is called the line at infinity $\mathbf{l}_{\infty}$. Such line, is more usually known as the line of horizon, and is a line that lies within the plane at infinity Π_{∞} .

The line at infinity can be seen as the intersection of any pair of general parallel planes, which only intersect at the plane at infinity, and such intersection is formed by a set of collinear vanishing points formed by lines which are parallel to such planes. Typically, the line at infinity of a given scene is found as the intersection of a plane parallel the scene plane, which goes through the camera center, as depicted on figure 1.10. Hence, it can be observed that the line at infinity only depends on the orientation of the scene plane, but not on its actual position.

More details respect to the line at infinity will be given on section 1.4.3

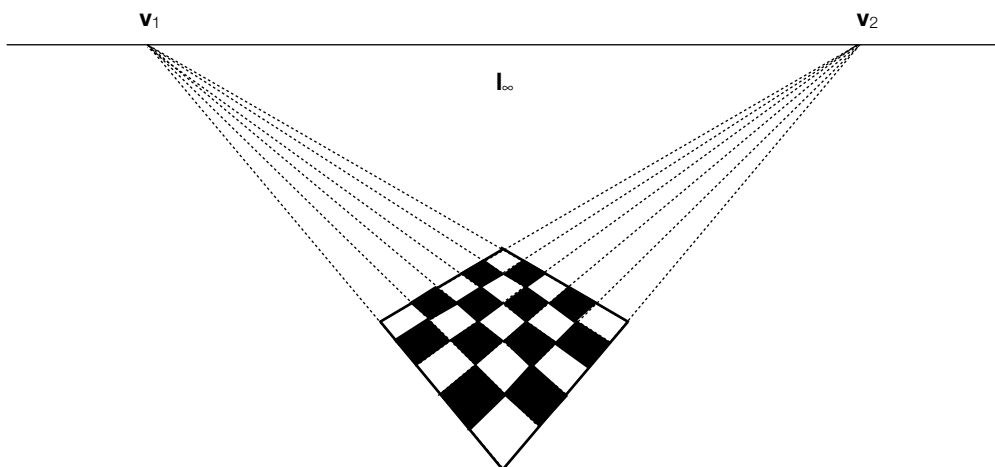


Fig. 1.10. Vanishing points ($\mathbf{v_1}$, $\mathbf{v_2}$) and the line at infinity ($\mathbf{l_{\infty}}$).

1.4.2.- The affine stratum

The affine stratum is located between the projective and the metric group. Obtaining 3D reconstructions up to this level, allows recovery of more structured scenes than in the projective stratum because parallelism is also preserved (although angles might still look skewed).

Affine geometry differs from projective geometry by identifying the *plane at infinity*, which lays outside of the affine stratum as well as the points that lay within such plane. This is true because points at infinity cannot be expressed as 3-vectors using inhomogeneous coordinates as it is usually done in the affine, metric and euclidean subspaces.

An affine transformation is usually expressed as follows:

$$\begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + \begin{bmatrix} a_{14} \\ a_{24} \\ a_{34} \end{bmatrix} \quad \text{with } \det(a_{ij}) \neq 0 \quad (1.119)$$

And by using homogeneous coordinates, expression (1.119) can be rewritten as $\mathbf{M}' \sim \mathbf{T}_A \cdot \mathbf{M}$, where

$$\mathbf{T}_A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1.120)$$

We can then observe that an affine transformation accounts for 12 independent degrees of freedom.

It can also be verified that the plane at infinity Π_∞ is one of the invariants of affine stratum, because as we can observe, any affine transformation like the one in (1.120) leaves the plane at infinity unchanged. Indeed,

$$\mathbf{T}_A^T \cdot \Pi_\infty = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} a_{14} \\ a_{24} \\ a_{34} \\ 1 \end{bmatrix} = \Pi_\infty \quad (1.121)$$

Hence we can state that:

$$\mathbf{T}_A^T \cdot \Pi_\infty \sim \Pi_\infty \quad (1.122)$$

Or by taking the inverse expression, which is analogous to (1.111),

$$\Pi_\infty \sim \mathbf{T}_A^{-T} \cdot \Pi_\infty \quad (1.123)$$

Notice that although the plane at infinity remains unchanged, the points within such plane can change under an affine transformation, although they will always remain within the plane at infinity.

Finally, on the affine stratum we can observe that besides all the invariants in the projective space that we saw in section 1.4.2, a new invariant appears which is related to the preservation of parallelism. This is also related to the fact that the plane at infinity remains unchanged.

By definition, two lines are called *parallel* if they never intersect. This is true except at the plane at infinity, where parallel lines will always intersect. In other words, parallel lines do not intersect in any point except at infinity. Hence, because the plane at infinity remains invariant in the affine stratum, so it will *parallelism* between lines.

1.4.2.1.- Upgrading from projective to affine stratum

As we explained in section 1.4.1.1, once the plane at infinity Π_∞ has been identified on any arbitrary projective space, we can find a transformation $\mathbf{T}_{PA}$ that upgrades projective geometry to the affine stratum by bringing the plane at infinity that has been identified into its canonical representation as shown in (1.118). Hence, using (1.122) or (1.123) we obtain:

$$\mathbf{T}^T \cdot \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \sim \Pi_\infty \quad \text{or} \quad \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \sim \mathbf{T}^T \cdot \Pi_\infty \quad (1.124)$$

Where $(0, 0, 0, 1)^T$ is the canonical representation of the plane at infinity.

Consequently, as we also mentioned in section 1.4.1.1, the fourth row of $\mathbf{T}$ is forced to be the plane at infinity Π_∞ that has been identified on a projective space. Since, no other constraints are imposed at this level, in order to have a simple transformation matrix that can be easily inverted, we chose the following matrix to upgrade projective stratum to the affine one:

$$\mathbf{T}_{PA} \sim \begin{bmatrix} \mathbf{I}_{3 \times 3} & \mathbf{0}_3 \\ \boldsymbol{\pi}_\infty^T & 1 \end{bmatrix} \quad (1.125)$$

where $\mathbf{I}_{3 \times 3}$ is a 3x3 identity matrix, $\mathbf{0}_3$ is a 3-vector of zeros and the last row contains the plane at infinity in homogeneous coordinates where the last coordinate has been scaled to 1. That is,

$$\Pi_\infty \sim \begin{bmatrix} \boldsymbol{\pi}_\infty & 1 \end{bmatrix}^T \quad (1.126)$$

It must also be noticed, that any other transformation of the following form would also bring the plane at infinity to its canonical representation $\Pi_\infty = (0, 0, 0, 1)^T$.

$$\mathbf{T}_{PA} \sim \begin{bmatrix} \mathbf{A} & \mathbf{0}_3 \\ \boldsymbol{\pi}_\infty^T & 1 \end{bmatrix} \quad \text{with } \det(\mathbf{A}) \neq 0 \quad (1.127)$$

where $\mathbf{A}$ is any non-singular 3x3 matrix.

Hence, (1.127) will also be valid, however, we will typically use (1.125) for simplicity in many computations.

1.4.3.- The metric stratum

The metric stratum corresponds to the group of similarities. In such group, as we stated in section 1.4, we will be able to recover 3D shape of objects up to a given scale, and a given rotation or translation due to the camera that has been chosen as reference (i.e. where the origin of coordinates is located).

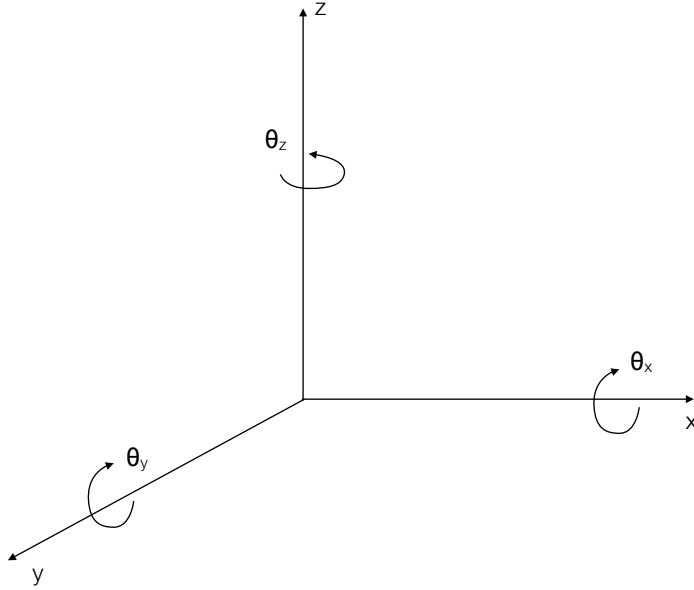


Fig. 1.11. Rotation on x, y, z axes.

$$\begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix} = \sigma \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + \begin{bmatrix} t_{14} \\ t_{24} \\ t_{34} \end{bmatrix} \quad (1.128)$$

where t_{ij} correspond to the components of a translation vector, and r_{ij} correspond to the coefficients of matrix $\mathbf{R}$, which is a rotation matrix. Hence, such matrix must be orthonormal and fulfills:

$$\mathbf{R}^T \cdot \mathbf{R} = \mathbf{R} \cdot \mathbf{R}^T = \mathbf{I} \Rightarrow \mathbf{R}^{-1} = \mathbf{R}^T \quad (1.129)$$

and also, because $\mathbf{R}$ is a rotation matrix, then its determinant is one, as it is proved in Proof 1.5:

$$\det(\mathbf{R}) = 1 \quad (1.130)$$

Proof 1.5

Expression (1.130) is true since any rotation matrix in 2D can be expressed in terms of the rotation angle. Hence, in 3D we can defined rotations by a given angle for each of the 3 axis that spans the 3D space. In other words, we can define a rotation matrix $\mathbf{R}_x$ that rotates the y-axis towards the z-axis by an angle θ_x , a second matrix $\mathbf{R}_y$ that rotates the z-axis towards the x-axis by an angle θ_y , and a third matrix $\mathbf{R}_z$ that rotates the x-axis towards the y-axis by an angle θ_z . This is depicted in figure 1.11.

Such matrices can be defined as follows:

However, even with this limitations, recovered objects at this stratum already look very much like their counterparts in the original scene, because besides recovery of parallelism in the affine stratum, also angles are preserved in the metric stratum. Hence, recovered shapes can be seen as a scaled models of a given scene, which can already be used in many applications, such as special effects in movies or artificial intelligence among others.

A metric transformation is usually represented as follows:

$$\begin{aligned}
\mathbf{R}_x(\theta_x) &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \theta_x & -\sin \theta_x \\ 0 & \sin \theta_x & \cos \theta_x \end{bmatrix} \\
\mathbf{R}_y(\theta_y) &= \begin{bmatrix} \cos \theta_y & 0 & \sin \theta_y \\ 0 & 1 & 0 \\ -\sin \theta_y & 0 & \cos \theta_y \end{bmatrix} \\
\mathbf{R}_z(\theta_z) &= \begin{bmatrix} \cos \theta_z & -\sin \theta_z & 0 \\ \sin \theta_z & \cos \theta_z & 0 \\ 0 & 0 & 1 \end{bmatrix}
\end{aligned} \tag{1.131}$$

where each of those matrices only has a single degree of freedom, namely their corresponding angle of rotation.

Then, the combined overall rotation can be defined as the product of those three matrices, where the order of such product will determine the order in which each rotation on each axis is done, starting from the rightmost matrix. Hence,

$$\mathbf{R} \sim \mathbf{R}_z(\theta_z) \cdot \mathbf{R}_y(\theta_y) \cdot \mathbf{R}_x(\theta_x) \tag{1.132}$$

Consequently, by means of (1.132) we can assert that any general rotation will have 3 degrees of freedom corresponding to the 3 angles of rotation.

Hence, by using expression (1.131), we can observe that the determinants of all three matrices are equal to one. Therefore, by means of expression (1.132), the determinant of $\mathbf{R}$ (which is the product of the determinants of all three matrices) will also be one and the proof will be finished.

Notice as well that in expression (1.131) we have given rotation on the 3 axis of the 3D space, but rotation can also be directly defined against any arbitrary axis of rotation by taking into account that the vector $\mathbf{u}$ containing such axis fulfills,

$$\mathbf{R} \cdot \mathbf{u} = \mathbf{u} \tag{1.133}$$

In other words, points located in axis of rotation $\mathbf{u}$, do not change after rotation. Also notice that expression (1.133) can be seen as if $\mathbf{u}$ is an eigenvector of $\mathbf{R}$ corresponding to the eigenvalue $\lambda = 1$. ($\mathbf{R} \cdot \mathbf{u} = \lambda \cdot \mathbf{u}$).

After this dissertation on rotation matrices, we can then assert that expression (1.128) can also be expressed as:

$$\tilde{\mathbf{M}}' = \sigma \cdot \mathbf{R} \cdot \tilde{\mathbf{M}} + \mathbf{T} \tag{1.134}$$

where points $\mathbf{M}$ and $\mathbf{M}'$ are expressed using inhomogeneous coordinates, σ is a scale factor, $\mathbf{R}$ a given rotation matrix and $\mathbf{T}$ the translation vector.

Then, using homogeneous coordinates, expression (1.134) can be rewritten as in (1.109):

$$\mathbf{M}' \sim \mathbf{T}_M \cdot \mathbf{M} \tag{1.135}$$

where $\mathbf{T}_M$ is a 4x4 metric transformation that has the following form:

$$\mathbf{T}_M \sim \begin{bmatrix} \sigma \cdot r_{11} & \sigma \cdot r_{12} & \sigma \cdot r_{13} & t_x \\ \sigma \cdot r_{21} & \sigma \cdot r_{22} & \sigma \cdot r_{23} & t_y \\ \sigma \cdot r_{31} & \sigma \cdot r_{32} & \sigma \cdot r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1.136)$$

Therefore, we can observe that any metric transformation has 7 degrees of freedom (3 for rotation as we have seen, 3 for translation vector components t_x , t_y and t_z , and the last one that accounts for scale σ).

Next, we will explain the invariants of the metric stratum, which are based on the *absolute conic* and the *absolute dual quadric* in order to ensure that angles are preserved.

1.4.3.1.- Introduction to the dual absolute conic and the dual absolute quadric.

The *absolute conic* Ω_∞ is a point conic $\mathbf{C}$ (like the ones in section 1.2.5) that lies on the *plane at infinity* Π_∞ and it is formed by the intersection of the *absolute quadric* $\mathbf{Q}_\infty$ with such plane. Because the *absolute conic* lies on the *plane at infinity*, then its points expressed in 3D homogeneous world coordinates $\mathbf{M} = (X, Y, Z, W)^T$ will be located at infinity as well ($W = 0$).

By means of duality, then the *dual absolute conic* Ω_∞^* can be seen as the set of lines that lie on the *plane at infinity* Π_∞ and are tangent to the *absolute conic*. Therefore, also the *dual absolute quadric* $\mathbf{Q}_\infty^*$ can be defined as the set of planes tangent to the *absolute quadric* $\mathbf{Q}_\infty$, so that the intersection of this set of planes with the plane at infinity yields the dual absolute conic Ω_∞^* . All these entities are shown on figure 1.12 for the sake of clarity.

After this dissertation, let's put all this into equations. First of all, as we saw in section 1.3.6 any tangent plane to the *dual absolute quadric* must fulfill:

$$\Pi^T \cdot \mathbf{Q}_\infty^* \cdot \Pi = 0 \quad (1.137)$$

We also know that each of these planes Π will intersect with the *plane of infinity*, producing for each plane a linear system of equations like the following one:

$$\begin{cases} \Pi^T \cdot \mathbf{M} = 0 \\ \Pi_\infty^T \cdot \mathbf{M} = 0 \end{cases} \quad (1.138)$$

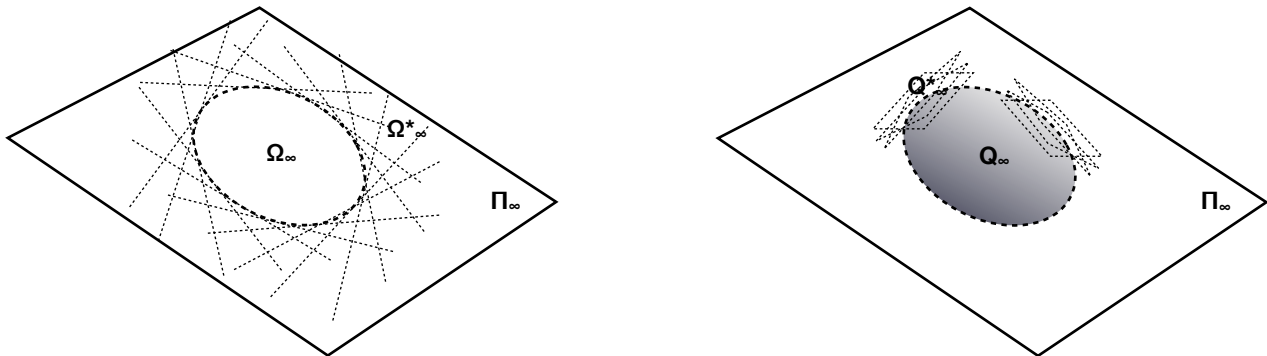


Fig. 1.12. Left: The absolute conic and the dual absolute conic laying on the plane at infinity in a 3D projective world. Right: The absolute quadric and the dual absolute quadric intersecting on the plane at infinity at the absolute conic and dual absolute conic in a 3D projective world.

The solution $\mathbf{M}$ to each of these systems of equations will be a line $\mathbf{l}$ that will lie at the *plane of infinity*. Indeed, by expanding expression (1.138), and taking into account that as in (1.118) the canonical form of the plane of infinity is $\mathbf{\Pi}_\infty = (0,0,0,1)^T$, then (1.138) becomes:

$$\begin{aligned} \begin{cases} (A,B,C,D)^T \cdot (X,Y,Z,W) = 0 \\ (0,0,0,1)^T \cdot (X,Y,Z,W) = 0 \end{cases} &\Rightarrow \begin{cases} A \cdot X + B \cdot Y + C \cdot Z + D \cdot W = 0 \\ W = 0 \end{cases} \Rightarrow \\ &\Rightarrow A \cdot X + B \cdot Y + C \cdot Z = 0 \end{aligned} \quad (1.139)$$

Then, taking into account expression (1.7), we can state that the set of points that belong to line (1.139) can be expressed as:

$$\mathbf{l}^T \cdot \mathbf{m} = 0 \Rightarrow (A,B,C) \cdot (X,Y,Z)^T = 0 \quad (1.140)$$

And then the line $\mathbf{l}$ of a plane $\mathbf{\Pi} = (A,B,C,D)^T$ that is tangent to the *dual absolute quadric* and also that intersects on the *plane at infinity* $\mathbf{\Pi}_\infty$ is:

$$\mathbf{l} = (A,B,C)^T \quad (1.141)$$

which is equal to the director vector of the plane as we stated in (1.73).

Because the *absolute conic* $\mathbf{\Omega}_\infty$ also lies on the *plane at infinity* $\mathbf{\Pi}_\infty$ and is the intersection of the *absolute quadric* $\mathbf{Q}_\infty$ with the *plane at infinity* $\mathbf{\Pi}_\infty$, and because each plane $\mathbf{\Pi}$ is also tangent to the *absolute quadric* $\mathbf{Q}_\infty$ as well, then the lines $\mathbf{l}$ in (1.141) will also be tangent to the *absolute conic* $\mathbf{\Omega}_\infty$ as depicted on figure 1.12. Hence, this set of lines $\mathbf{l}$ will be the dual absolute conic $\mathbf{\Omega}_\infty^*$, and will follow next expression:

$$\mathbf{l}^T \cdot \mathbf{\Omega}_\infty^* \cdot \mathbf{l} = 0 \quad (1.142)$$

In following sections and chapters, and when we introduce the camera models, we will observe the relations between the dual absolute conic and the dual absolute quadric.

Next, we will discuss how orthogonality is preserved in the metric stratum.

1.4.3.2.- Orthogonality in metric stratum

As we have previously mentioned, within the metric stratum angles are preserved so that 3D shapes are represented up to scale. In order to demonstrate how angles are preserved, we will take into account to concept of orthogonality, first in 2D and then in 3D.

First of all, by definition of the dot product of any pair of vectors of any dimension (i.e. 2D points lines or also 3D points or planes), we know that the dot product of those vectors $\mathbf{v}_1$ and $\mathbf{v}_2$ is:

$$\mathbf{v}_1^T \cdot \mathbf{v}_2 = \|\mathbf{v}_1\| \cdot \|\mathbf{v}_2\| \cdot \cos \theta \quad (1.143)$$

where θ is the angle of separation between $\mathbf{v}_1$ and $\mathbf{v}_2$, and $\|\cdot\|$ is their norm.

Hence, for any pair of orthogonal vectors ($\theta = 90^\circ$), we obtain that (1.43) becomes:

$$\mathbf{v}_1^T \cdot \mathbf{v}_2 = 0 \quad (1.144)$$

In a metric 2D world, expression (1.144) would hold for any pair of points $\mathbf{m}_1$, $\mathbf{m}_2$. However, whenever a 3x3 homography $\mathbf{H}$ is applied to such points in order to obtain points $\mathbf{m}'_1$, $\mathbf{m}'_2$ as shown below:

$$\begin{aligned}\mathbf{m}'_1 &= \mathbf{H} \cdot \mathbf{m}_1 \\ \mathbf{m}'_2 &= \mathbf{H} \cdot \mathbf{m}_2\end{aligned}\tag{1.145}$$

Then, expression (1.144) for a pair of orthogonal points $\mathbf{m}_1, \mathbf{m}_2$ in the metric stratum becomes:

$$\mathbf{m}_1^T \cdot \mathbf{m}_2 = (\mathbf{H}^{-1} \cdot \mathbf{m}'_1)^T \cdot \mathbf{H}^{-1} \cdot \mathbf{m}'_2 = \mathbf{m}'_1{}^T \cdot \mathbf{H}^{-T} \cdot \mathbf{H}^{-1} \cdot \mathbf{m}'_2 = 0\tag{1.146}$$

We know that any matrix multiplied by its transpose is a symmetric matrix $\mathbf{A}^T \cdot \mathbf{A} = \mathbf{A} \cdot \mathbf{A}^T$. Hence $\mathbf{H}^{-T} \cdot \mathbf{H}^{-1}$ is a 3x3 symmetric matrix. From section 1.2.5, we know that any 3x3 symmetric matrix is a conic, hence we can assume that

$$\mathbf{C}' = \mathbf{H}^{-T} \cdot \mathbf{H}^{-1}\tag{1.147}$$

where $\mathbf{C}'$ is a conic. Therefore, expression (1.146) can also be expressed as:

$$\mathbf{m}_1^T \cdot \mathbf{m}_2 = \mathbf{m}'_1{}^T \cdot \mathbf{C}' \cdot \mathbf{m}'_2 = 0\tag{1.148}$$

And therefore, by means of expression (1.143) the angle of separation between two vectors becomes:

$$\cos \theta = \frac{\mathbf{m}_1^T \cdot \mathbf{m}_2}{\sqrt{\mathbf{m}_1^T \cdot \mathbf{m}_1} \cdot \sqrt{\mathbf{m}_2^T \cdot \mathbf{m}_2}} = \frac{\mathbf{m}'_1{}^T \cdot \mathbf{C}' \cdot \mathbf{m}'_2}{\sqrt{\mathbf{m}'_1{}^T \cdot \mathbf{C}' \cdot \mathbf{m}'_1} \cdot \sqrt{\mathbf{m}'_2{}^T \cdot \mathbf{C}' \cdot \mathbf{m}'_2}}\tag{1.149}$$

We can observe that by means of (1.51) expression (1.148) can be seen as:

$$\mathbf{m}_1^T \cdot \mathbf{m}_2 = \mathbf{m}'_1{}^T \cdot \mathbf{C}' \cdot \mathbf{m}'_2 = \mathbf{m}'_1{}^T \cdot \mathbf{l}'_2 = 0\tag{1.150}$$

where $\mathbf{l}'_2 = \mathbf{C}' \cdot \mathbf{m}'_2$ is a line tangent to conic $\mathbf{C}'$ at point $\mathbf{m}'_2$, and consequently $\mathbf{m}'_1$ is a point perpendicular to the line $\mathbf{l}'_2$. Indeed, we can also assume that $\mathbf{m}'_1$ is also a point where line $\mathbf{l}'_1$ tangent to conic $\mathbf{C}'$ meets the conic. This is depicted with more detail on figure 1.13.

Hence, similarly to (1.53), we can write (1.150) by means of duality as:

$$\mathbf{l}_1^T \cdot \mathbf{l}_2 = \mathbf{l}_1^T \cdot \mathbf{C}'^{-1} \cdot \mathbf{l}_2 = \mathbf{l}_1^T \cdot \mathbf{C}^* \cdot \mathbf{l}_2 = 0\tag{1.151}$$

where $\mathbf{C}^*$ is the dual conic, and $\mathbf{l}'_1, \mathbf{l}'_2$ are two orthogonal lines in an arbitrary projective stratum, and $\mathbf{l}_1, \mathbf{l}_2$ are two orthogonal lines in the metric stratum.

We know from expressions (1.59) and (1.60) how a given homography $\mathbf{H}$ affects a conic or a dual conic. Hence, by looking at (1.148), we can infer that the conic in a metric stratum $\mathbf{C}_M$ must be the 3x3 identity. In other words:

$$\mathbf{C}_M = \mathbf{I}_{3 \times 3} \Rightarrow \mathbf{C}' \sim \mathbf{H}^{-T} \cdot \mathbf{C}_M \cdot \mathbf{H}^{-1} = \mathbf{H}^{-T} \cdot \mathbf{H}^{-1}\tag{1.152}$$

which fulfills expression (1.147).

Consequently, because the dual conic in metric stratum is the inverse of such conic, then:

$$\mathbf{C}_M^* = \mathbf{C}_M^{-1} = \mathbf{I}_{3 \times 3}\tag{1.153}$$

And hence, also expression (1.151) is also fulfilled.

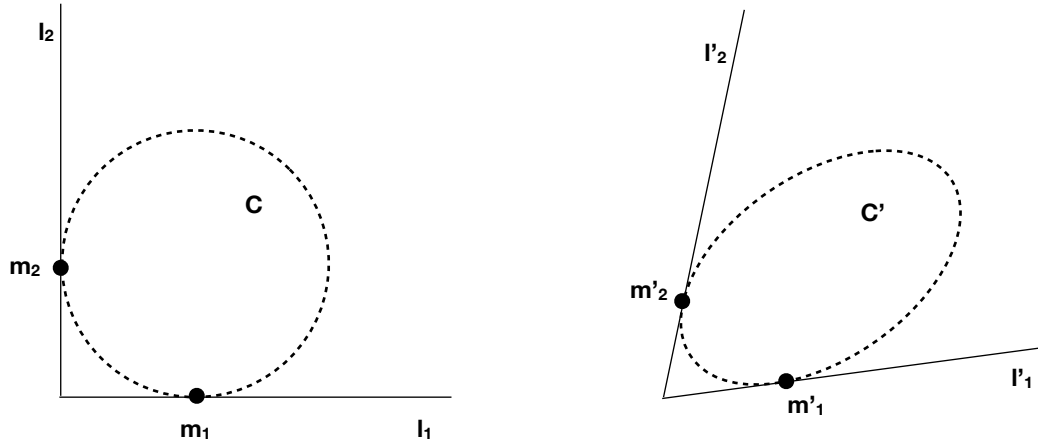


Fig. 1.13. Left: Orthogonal points and lines in a 2D metric stratum. Orthogonal lines l_1, l_2 are tangent to conic C at points m_1, m_2 . Right: Orthogonal points and lines in an arbitrary 2D projective stratum where points m_1, m_2 have been transformed by means of homography H resulting in points m'_1, m'_2 , lines l'_1, l'_2 and a transformed conic C'

On the other hand, in a metric 3D world, determining whether two planes are orthogonal, must be done in terms of their director vectors as in (1.73). Then, given two planes $\Pi_1 = (A_1, B_1, C_1, D_1)^T$, $\Pi_2 = (A_2, B_2, C_2, D_2)^T$, we can say they are orthogonal if their director vectors $\mathbf{n}_1 = (A_1, B_1, C_1)^T$, $\mathbf{n}_2 = (A_2, B_2, C_2)^T$ are also orthogonal. In other words:

$$\Pi_1^T \cdot \Pi_2 = 0 \Rightarrow \mathbf{n}_1^T \cdot \mathbf{n}_2 = 0 \quad (1.154)$$

By analogy with the 2D case, any pair or perpendicular planes can be seen as two perpendicular planes tangent to a given quadric, as follows:

$$\Pi_1^T \cdot \Pi_2 = \Pi_1^T \cdot \mathbf{Q}^* \cdot \Pi_2 = 0 \quad (1.155)$$

Then, by analogy with the 2D case, there exists a given quadric in the metric stratum $\mathbf{Q}_M$ that makes two planes perpendicular, so that:

$$\begin{aligned} \Pi_1^T \cdot \mathbf{Q}_M^* \cdot \Pi_2 &= 0 \Rightarrow \\ \begin{bmatrix} \mathbf{n}_1 & D_1 \end{bmatrix}^T \begin{bmatrix} \mathbf{A} & \mathbf{b} \\ \mathbf{b}^T & c \end{bmatrix} \begin{bmatrix} \mathbf{n}_2 \\ D_2 \end{bmatrix} &= \begin{bmatrix} \mathbf{n}_1 & D_1 \end{bmatrix}^T \begin{bmatrix} \mathbf{A} \cdot \mathbf{n}_2 + D_2 \cdot \mathbf{b} \\ \mathbf{b}^T \cdot \mathbf{n}_2 + c \cdot D_2 \end{bmatrix} = \\ &= \mathbf{n}_1^T \cdot \mathbf{A} \cdot \mathbf{n}_2 + D_2 \cdot \mathbf{n}_1^T \cdot \mathbf{b} + D_1 \cdot \mathbf{b}^T \cdot \mathbf{n}_2 + c \cdot D_1 \cdot D_2 = 0 \end{aligned} \quad (1.156)$$

where the dual quadric in metric stratum $\mathbf{Q}_M^*$ is a 4x4 symmetric matrix, and then $\mathbf{A}$ is a 3x3 symmetric matrix that could be considered to be a conic, $\mathbf{b}$ is a 3-vector and c is a scalar value. On the other hand, planes are expressed in terms of their director vectors $\mathbf{n}_1, \mathbf{n}_2$, and their last component D_1, D_2 .

Then, from (1.148), we know that the dot product of two orthogonal vectors in a metric stratum is zero, and hence,

$$\mathbf{n}_1^T \cdot \mathbf{A} \cdot \mathbf{n}_2 = 0, \quad \mathbf{A} = \mathbf{I}_{3 \times 3} \quad (1.157)$$

Consequently, in a **metric** stratum expression (1.156) is only fulfilled for any arbitrary pair of planes $\Pi_1 = (A_1, B_1, C_1, D_1)^T = (\mathbf{n}_1, D_1)^T$, and $\Pi_2 = (A_2, B_2, C_2, D_2)^T = (\mathbf{n}_2, D_2)^T$, if $\mathbf{b} = (0,0,0)^T$ and $c = 0$

Therefore, the dual quadric in the metric stratum has the following form:

$$\mathbf{Q}_M^* = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad (1.158)$$

which is a degenerate dual quadric, and hence its inverse cannot be computed.

1.4.3.3.- Dual absolute conic and dual absolute quadric in metric stratum

So far, we have seen how orthogonality of points, lines and planes is referred to a given conic or quadric, let's now see how this is related to the *absolute conic* and the dual *absolute quadric*.

First of all we must notice that the *absolute conic* lies on a 3D metric projective space, hence, two orthogonal 2D points or lines, although they can look orthogonal in a 2D space, they might no longer look orthogonal on 3D after rotations and translations and projection to 2D are applied. Hence, instead of the *absolute conic* Ω_∞ , we define the *image of the absolute conic* (IAC) ω_∞ , which is obtained when projecting the absolute conic laying on the *plane at infinity* Π_∞ to the retinal plane, as in figure 1.14.left. Indeed, in any projected image we will observe just IAC or the DIAC, not the absolute conic. Both the IAC and its dual the DIAC will be explained with further detail on chapter 2 and 5.

Because the IAC might not preserve orthogonality, we need to move to the plane at infinity Π_∞ so that the absolute conic looks completely as a 2D figure, as in figure 1.14.right.

We can then observe that any pair of orthogonal points or orthogonal lines in the *plane at infinity* Π_∞ will follow expressions (1.148) and (1.151), consequently for any pair of orthogonal points $\mathbf{m}_1, \mathbf{m}_2$ in a metric stratum:

$$\mathbf{m}_1^T \cdot \Omega_\infty \cdot \mathbf{m}_2 = \mathbf{m}_1^T \cdot \mathbf{m}_2 = 0 \quad (1.159)$$

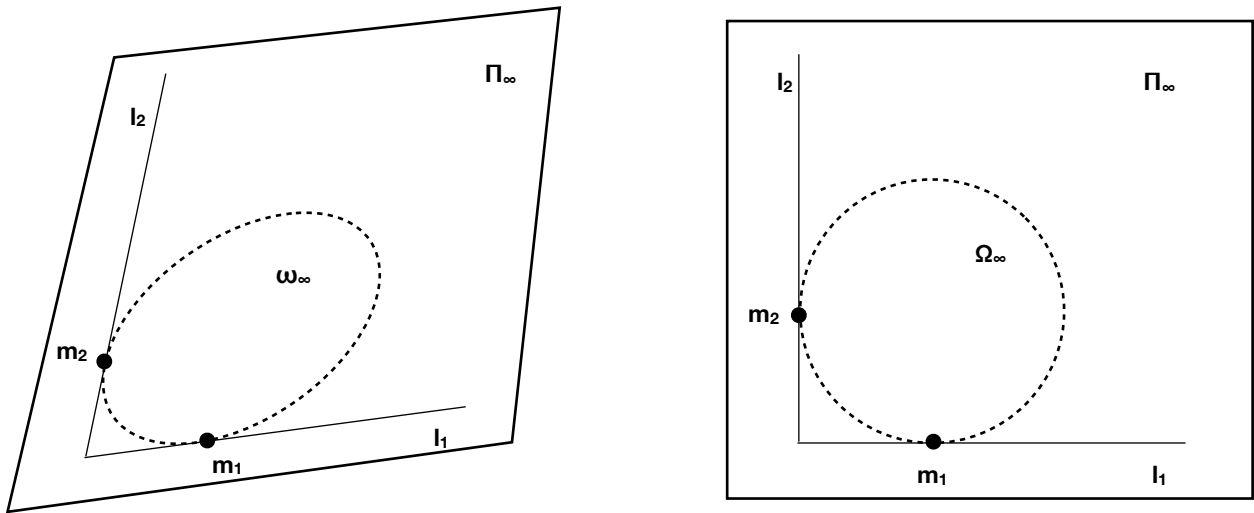


Fig. 1.14. Left: Orthogonal points and lines in 3D laying on the *plane at infinity* Π_∞ and tangent to what is called the *image absolute conic* (IAC) ω_∞ . As always, the dual image of the absolute conic (DIAC) ω_∞^* is the locus of all tangent lines to the IAC ω_∞ . Right: After setting the retinal plane on the *plane at infinity* Π_∞ , orthogonal points and lines are projected in 2D, and this time the tangent conic is *the absolute conic* Ω_∞ .

We then infer that in a **metric** stratum the absolute conic Ω_∞ has to be the 3x3 identity matrix as we saw in (1.153):

$$\Omega_\infty = \mathbf{I}_{3 \times 3} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (1.160)$$

Therefore, by means of duality, a pair of lines $\mathbf{l}_1, \mathbf{l}_2$ tangent to the absolute conic Ω_∞ is:

$$\mathbf{l}_1^T \cdot \Omega_\infty^* \cdot \mathbf{l}_2 = \mathbf{l}_1^T \cdot \mathbf{l}_2 = 0 \quad (1.161)$$

where we know that $\mathbf{l}_1, \mathbf{l}_2$ are the intersection of planes tangent to the *absolute quadric* $\mathbf{Q}_\infty$ with the *plane at infinity* Π_∞ , and the *dual absolute conic* is the inverse of the *absolute conic*. Hence,

$$\Omega_\infty^* = (\Omega_\infty)^{-1} = \mathbf{I}_{3 \times 3} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (1.162)$$

Therefore, we then know then that the absolute conic is the 3x3 identity, which by means of (1.27) is equivalent to the following equation:

$$x^2 + y^2 + 1 = 0 \quad (1.163)$$

which by looking at table 1.1, it can be seen as a circle of complex points with radius $i = \sqrt{-1}$.

As we saw in (1.141) any plane in a projective stratum can be expressed in terms of the line intersecting the plane at infinity as shown below:

$$\Pi = (A, B, C, D)^T = (\mathbf{l}, D)^T \quad (1.164)$$

Then, for any pair of orthogonal planes $\Pi_1 = (A_1, B_1, C_1, D_1)^T = (\mathbf{l}_1, D_1)^T$, $\Pi_2 = (A_2, B_2, C_2, D_2)^T = (\mathbf{l}_2, D_2)^T$ tangent to the *dual absolute quadric* $\mathbf{Q}_\infty^*$ in the metric stratum, we obtain that:

$$\begin{aligned} \Pi_1^T \cdot \Pi_2 &= \Pi_1^T \cdot \mathbf{Q}_\infty^* \cdot \Pi_2 = \begin{bmatrix} \mathbf{l}_1^T & D_1 \end{bmatrix} \begin{bmatrix} \mathbf{A} & \mathbf{b} \\ \mathbf{b}^T & c \end{bmatrix} \begin{bmatrix} \mathbf{l}_2 \\ D_2 \end{bmatrix} = \\ &= \begin{bmatrix} \mathbf{l}_1^T & D_1 \end{bmatrix} \begin{bmatrix} \mathbf{A} \cdot \mathbf{l}_2 + D_2 \cdot \mathbf{b} \\ \mathbf{b}^T \cdot \mathbf{l}_2 + c \cdot D_2 \end{bmatrix} = \mathbf{l}_1^T \cdot \mathbf{A} \cdot \mathbf{l}_2 + D_2 \cdot \mathbf{l}_1^T \cdot \mathbf{b} + D_1 \cdot \mathbf{b}^T \cdot \mathbf{l}_2 + c \cdot D_1 \cdot D_2 = 0 \end{aligned} \quad (1.164)$$

where again $\mathbf{A}$ must be a 3x3 symmetric matrix because the *dual absolute quadric* $\mathbf{Q}_\infty^*$ is a 4x4 symmetric matrix. Hence, the first term indicates that $\mathbf{l}_1$ and $\mathbf{l}_2$, which we know that are two orthogonal lines corresponding to the intersection with the *plane at infinity* Π_∞ . Hence, $\mathbf{l}_1, \mathbf{l}_2$ lie on the plane at infinity and consequently so does the dual conic tangent to both lines. We can then infer that $\mathbf{A}$ corresponds to such dual conic. And by means of (1.161) we can say besides that $\mathbf{A}$ must correspond to the dual absolute conic Ω_∞^* . And hence,

$$\mathbf{l}_1^T \cdot \mathbf{A} \cdot \mathbf{l}_2 = \mathbf{l}_1^T \cdot \Omega_\infty^* \cdot \mathbf{l}_2 = 0 \quad (1.165)$$

And the matrix of the *dual absolute quadric* can be expressed in terms of the dual absolute conic Ω_∞^* as:

$$\mathbf{Q}_\infty^* = \begin{bmatrix} \Omega_\infty^* & \mathbf{b} \\ \mathbf{b}^\top & c \end{bmatrix} \quad (1.166)$$

In a metric stratum, we know that the dual absolute conic Ω_∞^* is equal to the 3x3 identity as we saw in (1.162). Also as we saw in (1.156), we must notice that $\mathbf{b} = (0,0,0)^\top$ and $c = 0$. Consequently in a metric stratum the dual absolute quadric becomes:

$$\mathbf{Q}_\infty^* = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad (1.167)$$

which is a degenerate dual quadric that has no inverse as we saw in (1.158).

Therefore, the *absolute quadric* can be interpreted as follows using the following limit:

$$\mathbf{Q}_\infty = \lim_{k \rightarrow \infty} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & k \end{bmatrix} \quad (1.168)$$

Then the dual absolute quadric is:

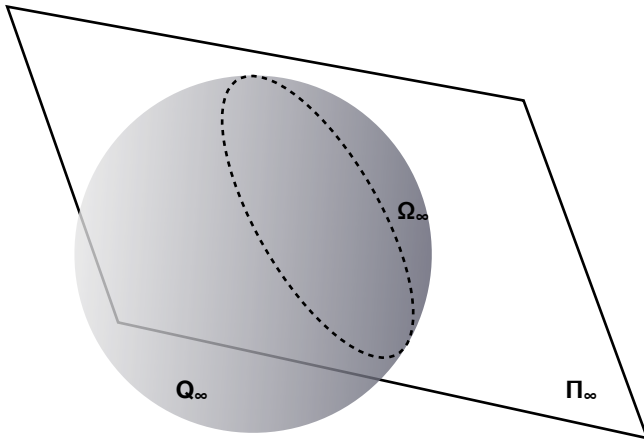


Fig. 1.15. The *absolute quadric* $\mathbf{Q}_\infty$ as a sphere of complex points intersecting the *plane at infinity* Π_∞ to form the *absolute conic* Ω_∞ , which is a circle of complex points in the laying in a 3D metric stratum. Because the *absolute conic* lies on a 2D plane the IAC ω_∞ is defined as depicted in fig. 1.14 by moving the *plane at infinity* Π_∞ to the retinal plane.

$$\mathbf{Q}_\infty^* = \mathbf{Q}_\infty^{-1} = \lim_{k \rightarrow \infty} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1/k \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad (1.169)$$

Consequently, by analogy with the absolute conic case, the *absolute quadric* $\mathbf{Q}_\infty$ in **metric** stratum can be interpreted as a sphere of complex numbers with infinite radius (when k tends to infinity) that meets with the *plane at infinity* Π_∞ forming the *absolute conic* Ω_∞ which is a circle of complex points. This is depicted at figure 1.15.

As shown in figure 1.14, it must also be noticed that the *absolute conic* Ω_∞ becomes the *image of the absolute conic* (IAC) ω_∞ when it is projected to the retinal plane by means of a camera as will be shown on chapter 2.

1.4.3.4.- Invariance of the absolute conic and the dual absolute quadric respect to similarity transformations

By using expression (1.136), we can define any similarity transformation as:

$$\mathbf{T} \sim \begin{bmatrix} \sigma \mathbf{R} & \mathbf{t} \\ \mathbf{0}_3^T & 1 \end{bmatrix} \quad (1.170)$$

where, as we already saw, $\mathbf{R}$ is a 3×3 rotation matrix, so that $\mathbf{R} \cdot \mathbf{R}^{-1} = \mathbf{R} \cdot \mathbf{R}^T = \mathbf{I}_{3 \times 3}$ and $\det(\mathbf{R}) = 1$, $\mathbf{t}$ is a 3-vector indicating translation, and σ is a scalar value to indicate the scale of the transformation.

Then, by means of expression (1.115) in a metric stratum where the dual absolute quadric has the form in (1.169) we obtain that:

$$\begin{aligned} \mathbf{Q}^{*'} &\sim \mathbf{T} \cdot \mathbf{Q}^* \cdot \mathbf{T}^T = \begin{bmatrix} \sigma \mathbf{R} & \mathbf{t} \\ \mathbf{0}_3^T & 1 \end{bmatrix} \begin{bmatrix} \mathbf{I}_{3 \times 3} & \mathbf{0}_3 \\ \mathbf{0}_3^T & 0 \end{bmatrix} \begin{bmatrix} \sigma \mathbf{R} & \mathbf{t} \\ \mathbf{0}_3^T & 1 \end{bmatrix}^T = \\ &= \begin{bmatrix} \sigma \cdot \mathbf{R} \cdot \mathbf{I}_{3 \times 3} + \mathbf{t} \cdot \mathbf{0}_3^T & \sigma \cdot \mathbf{R} \cdot \mathbf{0}_3 + 0 \cdot \mathbf{t} \\ \mathbf{0}_3^T \cdot \mathbf{I}_{3 \times 3} + 1 \cdot \mathbf{0}_3^T & \mathbf{0}_3^T \cdot \mathbf{0}_3 + 1 \cdot 0 \end{bmatrix} \begin{bmatrix} \sigma \mathbf{R} & \mathbf{t} \\ \mathbf{0}_3^T & 1 \end{bmatrix}^T = \\ &= \begin{bmatrix} \sigma \cdot \mathbf{R} \cdot \mathbf{I}_{3 \times 3} & \mathbf{0}_3 \\ \mathbf{0}_3^T & 0 \end{bmatrix} \begin{bmatrix} \sigma \mathbf{R}^T & \mathbf{0}_3 \\ \mathbf{t}^T & 1 \end{bmatrix} = \\ &= \begin{bmatrix} \sigma^2 \cdot \mathbf{R} \cdot \mathbf{I}_{3 \times 3} \cdot \mathbf{R}^T + \mathbf{0}_3 \cdot \mathbf{t}^T & \sigma \cdot \mathbf{R} \cdot \mathbf{I}_{3 \times 3} \cdot \mathbf{0}_3 + 1 \cdot \mathbf{0}_3 \\ \sigma \cdot \mathbf{0}_3^T \cdot \mathbf{R}^T + 0 \cdot \mathbf{t}^T & \mathbf{0}_3^T \cdot \mathbf{0}_3 + 0 \cdot 1 \end{bmatrix} = \\ &= \begin{bmatrix} \sigma^2 \cdot \mathbf{R} \cdot \mathbf{I}_{3 \times 3} \cdot \mathbf{R}^T & \mathbf{0}_3 \\ \mathbf{0}_3^T & 0 \end{bmatrix} = \begin{bmatrix} \sigma^2 \cdot \mathbf{R} \cdot \mathbf{R}^T & \mathbf{0}_3 \\ \mathbf{0}_3^T & 0 \end{bmatrix} \end{aligned} \quad (1.171)$$

And because $\mathbf{R}$ is a rotation matrix and $\mathbf{R} \cdot \mathbf{R}^T = \mathbf{I}_{3 \times 3}$, then (1.171) becomes:

$$\mathbf{Q}^{*'} \sim \mathbf{T} \cdot \mathbf{Q}^* \cdot \mathbf{T}^T = \begin{bmatrix} \sigma^2 \cdot \mathbf{R} \cdot \mathbf{R}^T & \mathbf{0}_3 \\ \mathbf{0}_3^T & 0 \end{bmatrix} = \sigma^2 \begin{bmatrix} \mathbf{I}_{3 \times 3} & \mathbf{0}_3 \\ \mathbf{0}_3^T & 0 \end{bmatrix} \sim \begin{bmatrix} \mathbf{I}_{3 \times 3} & \mathbf{0}_3 \\ \mathbf{0}_3^T & 0 \end{bmatrix} \quad (1.172)$$

Consequently, the similarity transformed *dual absolute quadric* remains invariant up to scale, or in other words:

$$\mathbf{Q}^{*'} \sim \mathbf{Q}^*, \quad \mathbf{T} \sim \begin{bmatrix} \sigma \mathbf{R} & \mathbf{t} \\ \mathbf{0}_3^T & 1 \end{bmatrix} \quad (1.173)$$

Inversely, it can also be proven that the only kind of transformation that leaves the *dual absolute quadric* $\mathbf{Q}^*$ unchanged is the group of similarity transformations.

Proof 1.6

Given any arbitrary 4x4 projective transformation of the form:

$$\mathbf{T} = \begin{bmatrix} \mathbf{A} & \mathbf{b} \\ \mathbf{c}^T & d \end{bmatrix} \quad (1.174)$$

Then, by means of (1.115) the transformed dual absolute quadric in the metric stratum becomes

$$\begin{aligned} \mathbf{Q}^{*'} \sim \mathbf{T} \cdot \mathbf{Q}^* \cdot \mathbf{T}^T &= \begin{bmatrix} \mathbf{A} & \mathbf{b} \\ \mathbf{c}^T & d \end{bmatrix} \begin{bmatrix} \mathbf{I}_{3 \times 3} & \mathbf{0}_3 \\ \mathbf{0}_3^T & 0 \end{bmatrix} \begin{bmatrix} \mathbf{A} & \mathbf{b} \\ \mathbf{c}^T & d \end{bmatrix}^T = \\ &= \begin{bmatrix} \mathbf{A} \cdot \mathbf{I}_{3 \times 3} + \mathbf{b} \cdot \mathbf{0}_3^T & \mathbf{A} \cdot \mathbf{0}_3 + 0 \cdot \mathbf{b} \\ \mathbf{c}^T \cdot \mathbf{I}_{3 \times 3} + d \cdot \mathbf{0}_3^T & \mathbf{c}^T \cdot \mathbf{0}_3 + d \cdot 0 \end{bmatrix} \begin{bmatrix} \mathbf{A} & \mathbf{b} \\ \mathbf{c}^T & d \end{bmatrix}^T = \\ &= \begin{bmatrix} \mathbf{A} & \mathbf{0}_3 \\ \mathbf{c}^T & 0 \end{bmatrix} \begin{bmatrix} \mathbf{A}^T & \mathbf{c} \\ \mathbf{b}^T & d \end{bmatrix} = \\ &= \begin{bmatrix} \mathbf{A} \cdot \mathbf{A}^T + \mathbf{0}_3 \cdot \mathbf{b}^T & \mathbf{A} \cdot \mathbf{c} + d \cdot \mathbf{0}_3 \\ \mathbf{c}^T \cdot \mathbf{A}^T + 0 \cdot \mathbf{b}^T & \mathbf{c}^T \cdot \mathbf{c} + 0 \cdot d \end{bmatrix} = \begin{bmatrix} \mathbf{A} \cdot \mathbf{A}^T & \mathbf{A} \cdot \mathbf{c} \\ \mathbf{c}^T \cdot \mathbf{A}^T & \mathbf{c}^T \cdot \mathbf{c} \end{bmatrix} \end{aligned} \quad (1.175)$$

Because we want the transformed dual absolute quadric $\mathbf{Q}^{*'}$ to remain invariant in the metric stratum, then:

$$\mathbf{Q}^{*'} \sim \mathbf{Q}^* = \begin{bmatrix} \mathbf{I}_{3 \times 3} & \mathbf{0}_3 \\ \mathbf{0}_3^T & 0 \end{bmatrix} = \begin{bmatrix} \mathbf{A} \cdot \mathbf{A}^T & \mathbf{A} \cdot \mathbf{c} \\ \mathbf{c}^T \cdot \mathbf{A}^T & \mathbf{c}^T \cdot \mathbf{c} \end{bmatrix} \quad (1.176)$$

and consequently

$$\begin{aligned}
\mathbf{A} \cdot \mathbf{A}^T &= \mathbf{I}_{3 \times 3} \\
\mathbf{A} \cdot \mathbf{c} &= \mathbf{0}_3 \\
\mathbf{c}^T \cdot \mathbf{c} &= 0 \Rightarrow \mathbf{c} = 0
\end{aligned} \tag{1.177}$$

Therefore, $\mathbf{c}$ must be a null 3-vector, and $\mathbf{A}$ must be a rotation matrix ($\mathbf{A} = \mathbf{R}$) that fulfills the condition $\mathbf{A} \cdot \mathbf{A}^T = \mathbf{I}_{3 \times 3}$. Besides, d can arbitrarily be set to one ($d = 1$) because the transformation is only defined up to scale. These conditions leave the arbitrary transformation $\mathbf{T}$ as:

$$\mathbf{T} \sim \begin{bmatrix} \mathbf{R} & \mathbf{b} \\ \mathbf{0}_3^T & 1 \end{bmatrix} \tag{1.178}$$

which is a similarity transformation.

Therefore we have proved that the only kind of transformation that leaves the *dual absolute quadric* $\mathbf{Q}_\infty^*$ unchanged is the group of similarity transformations.

Finally, we must notice that since the dual absolute conic $\mathbf{\Omega}_\infty^*$ is included in the top-right 3x3 minor of the *dual absolute quadric* $\mathbf{Q}_\infty^*$, as we saw in (1.166), and the *dual absolute quadric* $\mathbf{Q}_\infty^*$ remains unchanged by any **similarity** transformation $\mathbf{T}$. Then, also the dual absolute conic $\mathbf{\Omega}_\infty^*$, and consequently the absolute conic $\mathbf{\Omega}_\infty = (\mathbf{\Omega}_\infty^*)^{-1}$, remain unchanged by any of such similarity transformations $\mathbf{T}$.

1.4.3.5.- Upgrading from projective or affine to metric stratum

Up to this point we have seen the expressions of the *dual absolute quadric* $\mathbf{Q}_\infty^*$ in a metric stratum (1.169). We know that the *dual absolute quadric* $\mathbf{Q}_\infty^*$ can be linearly solved by means of (1.137) if enough orthogonal planes are known on a given image (orthogonal planes can also be obtained from 3 coplanar points or coplanar lines and points). And we also know how any given **projective** transformation $\mathbf{T}$ behaves on such dual quadric by means of (1.115).

So far, we also know how to convert a metric stratum into an affine one by locating the *plane at infinity* $\mathbf{\Pi}_\infty$ as we saw in section 1.4.2.1.

Then, taking all this information into account, we know that the *dual absolute quadric* $\mathbf{Q}_\infty^*$ in a metric stratum is:

$$\mathbf{Q}_\infty^* = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{1.179}$$

and after applying a given affine transformation $\mathbf{T}_A$ as in (1.120),

$$\mathbf{T}_A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{A} & \mathbf{a} \\ \mathbf{0}_3^T & 1 \end{bmatrix} \tag{1.180}$$

where $\mathbf{A}$ is any arbitrary non-singular 3x3 matrix and $\mathbf{a}$ any arbitrary 3-vector. We obtain from (1.174) and (1.175) that the *dual absolute quadric* $\mathbf{Q}_\infty^*$ becomes in the **affine** stratum:

$$\mathbf{Q}_{\infty}^* \sim \mathbf{T}_A \cdot \mathbf{Q}_{\infty}^* \cdot \mathbf{T}_A^T = \begin{bmatrix} \mathbf{A} \cdot \mathbf{A}^T & \mathbf{0}_3 \\ \mathbf{0}_3^T & 0 \end{bmatrix} \quad (1.181)$$

where we know that the the top-left 3x3 minor $\mathbf{A} \cdot \mathbf{A}^T$ is a conic where $\mathbf{A}$ can be obtained from Cholesky decomposition.

In order to reverse the effect of the affine transformation $\mathbf{T}_A$, we need to find another transformation $\mathbf{T}_{AM}$ that restores the affine stratum into a metric one. Hence,

$$\mathbf{Q}_{\infty}^* \sim \mathbf{T}_{AM} \cdot \mathbf{Q}_{\infty}^* \cdot \mathbf{T}_{AM}^T \quad (1.182)$$

where $\mathbf{Q}_{\infty}^*$ is the dual absolute quadric in the **metric** stratum and has the form in (1.179), and $\mathbf{Q}_{\infty}^*$ is the dual absolute quadric in the **affine** stratum that has the form shown in (1.181).

Hence, such transformation $\mathbf{T}_{AM}$ can be any transformation like the following one:

$$\mathbf{T}_{AM} = \begin{bmatrix} \mathbf{A}^{-1} & \mathbf{0}_3 \\ \mathbf{0}_3^T & k \end{bmatrix}, \quad \forall k \in \mathbb{R} \quad (1.183)$$

where k is any arbitrary scalar value, but for simplicity we might just choose $k = 1$.

It can then be proved that (1.183) is valid by expanding (1.182) as follows:

$$\begin{aligned} \mathbf{Q}_{\infty}^* \sim \mathbf{T}_{AM} \cdot \mathbf{Q}_{\infty}^* \cdot \mathbf{T}_{AM}^T &= \begin{bmatrix} \mathbf{A}^{-1} & \mathbf{0}_3 \\ \mathbf{0}_3^T & k \end{bmatrix} \begin{bmatrix} \mathbf{A} \cdot \mathbf{A}^T & \mathbf{0}_3 \\ \mathbf{0}_3^T & 0 \end{bmatrix} \begin{bmatrix} \mathbf{A}^{-T} & \mathbf{0}_3 \\ \mathbf{0}_3^T & k \end{bmatrix} = \\ &= \begin{bmatrix} \mathbf{A}^{-1} & \mathbf{0}_3 \\ \mathbf{0}_3^T & k \end{bmatrix} \begin{bmatrix} \mathbf{A} \cdot \mathbf{A}^T \cdot \mathbf{A}^{-T} + \mathbf{0}_3 \cdot \mathbf{0}_3^T & \mathbf{A} \cdot \mathbf{A}^T \cdot \mathbf{0}_3 + k \cdot \mathbf{0}_3 \\ \mathbf{0}_3^T \cdot \mathbf{A}^{-T} + 0 \cdot \mathbf{0}_3^T & \mathbf{0}_3^T \cdot \mathbf{0}_3 + 0 \cdot k \end{bmatrix} = \\ &= \begin{bmatrix} \mathbf{A}^{-1} & \mathbf{0}_3 \\ \mathbf{0}_3^T & k \end{bmatrix} \begin{bmatrix} \mathbf{A} & \mathbf{0}_3 \\ \mathbf{0}_3^T & 0 \end{bmatrix} = \begin{bmatrix} \mathbf{A}^{-1} \cdot \mathbf{A} + \mathbf{0}_3 \cdot \mathbf{0}_3^T & \mathbf{A}^{-1} \cdot \mathbf{0}_3 + 0 \cdot \mathbf{0}_3 \\ \mathbf{0}_3^T \cdot \mathbf{A} + k \cdot \mathbf{0}_3^T & \mathbf{0}_3^T \cdot \mathbf{0}_3 + 0 \cdot k \end{bmatrix} = \\ &= \begin{bmatrix} \mathbf{I}_{3 \times 3} & \mathbf{0}_3 \\ \mathbf{0}_3^T & 0 \end{bmatrix} \end{aligned} \quad (1.184)$$

which fulfills the condition that the dual absolute quadric is restored into its canonical form as in (1.179).

Hence, by combining expressions (1.125) and (1.183), we can obtain a transformation that directly upgrades the projective stratum into the metric as follows:

$$\begin{aligned}
\mathbf{T}_{\mathbf{PM}} &= \mathbf{T}_{\mathbf{AM}} \mathbf{T}_{\mathbf{PA}} = \begin{bmatrix} \mathbf{A}^{-1} & \mathbf{0}_3 \\ \mathbf{0}_3^T & k \end{bmatrix} \begin{bmatrix} \mathbf{I}_{3 \times 3} & \mathbf{0}_3 \\ \boldsymbol{\pi}_\infty^T & 1 \end{bmatrix} = \\
&= \begin{bmatrix} \mathbf{A}^{-1} \cdot \mathbf{I}_{3 \times 3} + \mathbf{0}_3 \cdot \boldsymbol{\pi}_\infty^T & \mathbf{A}^{-1} \cdot \mathbf{0}_3 + 1 \cdot \mathbf{0}_3 \\ \mathbf{0}_3^T \cdot \mathbf{I}_{3 \times 3} + k \cdot \boldsymbol{\pi}_\infty^T & \mathbf{0}_3^T \cdot \mathbf{0}_3 + k \cdot 1 \end{bmatrix} = \begin{bmatrix} \mathbf{A}^{-1} & \mathbf{0}_3 \\ k \cdot \boldsymbol{\pi}_\infty^T & k \end{bmatrix}
\end{aligned} \tag{1.185}$$

and because k is can be arbitrarily set to one, as we saw before, then (1.185) can be further simplified as:

$$\mathbf{T}_{\mathbf{PM}} = \begin{bmatrix} \mathbf{A}^{-1} & \mathbf{0}_3 \\ \boldsymbol{\pi}_\infty^T & 1 \end{bmatrix} \tag{1.186}$$

Finally, it must be noticed that expression (1.182) can be rewritten for the projective to metric transformation matrix $\mathbf{T}_{\mathbf{PM}}$ as:

$$\mathbf{Q}_\infty^* \sim \mathbf{T}_{\mathbf{PM}} \cdot \mathbf{Q}_\infty^* \cdot \mathbf{T}_{\mathbf{PM}}^T \tag{1.187}$$

where now $\mathbf{Q}_\infty^*$ is the dual absolute quadric in the **metric** stratum and has the canonic form in (1.179), and $\mathbf{Q}_\infty^{*'} is the dual absolute quadric in the **projective** stratum. Then, by the inverse of (1.187) we can obtain the *dual absolute quadric* $\mathbf{Q}_\infty^{*'}$ in any projective stratum as:$

$$\mathbf{Q}_\infty^{*' \sim \mathbf{T}_{\mathbf{PM}}^{-1} \cdot \mathbf{Q}_\infty^* \cdot \mathbf{T}_{\mathbf{PM}}^T \tag{1.188}$$

where we need to find the expression of $\mathbf{T}_{\mathbf{PM}}^{-1}$. But taking into account that blockwise inversion of any matrix can be done as:

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix}^{-1} = \begin{bmatrix} \mathbf{A}^{-1} + \mathbf{A}^{-1} \mathbf{B} (\mathbf{D} - \mathbf{C} \mathbf{A}^{-1} \mathbf{B})^{-1} \mathbf{C} \mathbf{A}^{-1} & -\mathbf{A}^{-1} \mathbf{B} (\mathbf{D} - \mathbf{C} \mathbf{A}^{-1} \mathbf{B})^{-1} \\ -(\mathbf{D} - \mathbf{C} \mathbf{A}^{-1} \mathbf{B})^{-1} \mathbf{C} \mathbf{A}^{-1} & (\mathbf{D} - \mathbf{C} \mathbf{A}^{-1} \mathbf{B})^{-1} \end{bmatrix} \tag{1.189}$$

if $\mathbf{A}$ and $\mathbf{D} - \mathbf{C} \mathbf{A}^{-1} \mathbf{B}$ are nonsingular.

Therefore using (1.185) as the expression of $\mathbf{T}_{\mathbf{PM}}$, its inverse $\mathbf{T}_{\mathbf{PM}}^{-1}$ can be expressed as:

$$\begin{aligned}
\mathbf{T}_{\mathbf{PM}}^{-1} &= \begin{bmatrix} \mathbf{A}^{-1} + \mathbf{A}^{-1} \mathbf{b} (d - \mathbf{c} \mathbf{A}^{-1} \mathbf{b})^{-1} \mathbf{c} \mathbf{A}^{-1} & -\mathbf{A}^{-1} \mathbf{b} (d - \mathbf{c} \mathbf{A}^{-1} \mathbf{b})^{-1} \\ -(\mathbf{d} - \mathbf{c} \mathbf{A}^{-1} \mathbf{b})^{-1} \mathbf{c} \mathbf{A}^{-1} & (\mathbf{d} - \mathbf{c} \mathbf{A}^{-1} \mathbf{b})^{-1} \end{bmatrix} = \\
&= \begin{bmatrix} \mathbf{A} + \mathbf{A} \cdot \mathbf{0}_3 (k - k \cdot \boldsymbol{\pi}_\infty^T \cdot \mathbf{A} \cdot \mathbf{0}_3)^{-1} \cdot k \cdot \boldsymbol{\pi}_\infty^T \cdot \mathbf{A} & -\mathbf{A} \cdot \mathbf{0}_3 (k - k \cdot \boldsymbol{\pi}_\infty^T \cdot \mathbf{A} \cdot \mathbf{0}_3)^{-1} \\ -(k - k \cdot \boldsymbol{\pi}_\infty^T \cdot \mathbf{A} \cdot \mathbf{0}_3)^{-1} \cdot k \cdot \boldsymbol{\pi}_\infty^T \cdot \mathbf{A} & (k - k \cdot \boldsymbol{\pi}_\infty^T \cdot \mathbf{A} \cdot \mathbf{0}_3)^{-1} \end{bmatrix} = \\
&= \begin{bmatrix} \mathbf{A} & \mathbf{0}_3 \\ -\boldsymbol{\pi}_\infty^T \cdot \mathbf{A} & k^{-1} \end{bmatrix}
\end{aligned} \tag{1.190}$$

Where we can prove that the inverse in (1.190) is valid as follows:

$$\begin{aligned}
 \mathbf{T}_{\text{PM}} \cdot \mathbf{T}_{\text{PM}}^{-1} &= \begin{bmatrix} \mathbf{A}^{-1} & \mathbf{0}_3 \\ k \cdot \boldsymbol{\pi}_\infty^T & k \end{bmatrix} \begin{bmatrix} \mathbf{A} & \mathbf{0}_3 \\ -\boldsymbol{\pi}_\infty^T \cdot \mathbf{A} & k^{-1} \end{bmatrix} = \\
 &= \begin{bmatrix} \mathbf{A}^{-1} \cdot \mathbf{A} - \mathbf{0}_3 \cdot \boldsymbol{\pi}_\infty^T \cdot \mathbf{A} & \mathbf{A}^{-1} \cdot \mathbf{0}_3 + \mathbf{0}_3 \cdot k^{-1} \\ k \cdot \boldsymbol{\pi}_\infty^T \cdot \mathbf{A} - k \cdot \boldsymbol{\pi}_\infty^T \cdot \mathbf{A} & k \cdot \boldsymbol{\pi}_\infty^T \cdot \mathbf{0}_3 + k \cdot k^{-1} \end{bmatrix} = \\
 &= \begin{bmatrix} \mathbf{I}_{3 \times 3} & \mathbf{0}_3 \\ \mathbf{0}_3^T & 1 \end{bmatrix} = \mathbf{I}_{4 \times 4}
 \end{aligned} \tag{1.191}$$

Notice, that for simplicity we have chosen $k = 1$ (although any k different from zero would be valid as well to compute the inverse). Then, (1.190) simplifies into:

$$\mathbf{T}_{\text{PM}}^{-1} = \begin{bmatrix} \mathbf{A} & \mathbf{0}_3 \\ -\boldsymbol{\pi}_\infty^T \cdot \mathbf{A} & 1 \end{bmatrix} \tag{1.192}$$

And so, expression (1.188) becomes:

$$\begin{aligned}
 \mathbf{Q}_\infty^{*'} \sim \mathbf{T}_{\text{PM}}^{-1} \cdot \mathbf{Q}_\infty^* \cdot \mathbf{T}_{\text{PM}}^T &= \begin{bmatrix} \mathbf{A} & \mathbf{0}_3 \\ -\boldsymbol{\pi}_\infty^T \cdot \mathbf{A} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{I}_{3 \times 3} & \mathbf{0}_3 \\ \mathbf{0}_3^T & 0 \end{bmatrix} \begin{bmatrix} \mathbf{A}^T & -\mathbf{A}^T \cdot \boldsymbol{\pi}_\infty \\ \mathbf{0}_3^T & 1 \end{bmatrix} = \\
 &= \begin{bmatrix} \mathbf{A} & \mathbf{0}_3 \\ -\boldsymbol{\pi}_\infty^T \cdot \mathbf{A} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{I}_{3 \times 3} \cdot \mathbf{A}^T + \mathbf{0}_3 \cdot \mathbf{0}_3^T & -\mathbf{I}_{3 \times 3} \cdot \mathbf{A}^T \cdot \boldsymbol{\pi}_\infty + 1 \cdot \mathbf{0}_3 \\ \mathbf{0}_3^T \cdot \mathbf{A}^T + 0 \cdot \mathbf{0}_3^T & -\mathbf{0}_3^T \cdot \mathbf{A}^T \cdot \boldsymbol{\pi}_\infty + 0 \cdot 1 \end{bmatrix} = \\
 &= \begin{bmatrix} \mathbf{A} & \mathbf{0}_3 \\ -\boldsymbol{\pi}_\infty^T \cdot \mathbf{A} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{A}^T & -\mathbf{A}^T \cdot \boldsymbol{\pi}_\infty \\ \mathbf{0}_3^T & 0 \end{bmatrix} = \begin{bmatrix} \mathbf{A} \cdot \mathbf{A}^T + \mathbf{0}_3 \cdot \mathbf{0}_3^T & -\mathbf{A} \cdot \mathbf{A}^T \cdot \boldsymbol{\pi}_\infty + 0 \cdot \mathbf{0}_3 \\ -\boldsymbol{\pi}_\infty^T \cdot \mathbf{A} \cdot \mathbf{A}^T + 1 \cdot \mathbf{0}_3^T & \boldsymbol{\pi}_\infty^T \cdot \mathbf{A} \cdot \mathbf{A}^T \cdot \boldsymbol{\pi}_\infty + 1 \cdot 0 \end{bmatrix} = \\
 &= \begin{bmatrix} \mathbf{A} \cdot \mathbf{A}^T & -\mathbf{A} \cdot \mathbf{A}^T \cdot \boldsymbol{\pi}_\infty \\ -\boldsymbol{\pi}_\infty^T \cdot \mathbf{A} \cdot \mathbf{A}^T & \boldsymbol{\pi}_\infty^T \cdot \mathbf{A} \cdot \mathbf{A}^T \cdot \boldsymbol{\pi}_\infty \end{bmatrix}
 \end{aligned} \tag{1.193}$$

In following chapters we will also observe that for any **projective** stratum, the 3×3 top-left minor of the *dual absolute quadric* $\mathbf{Q}_\infty^*$ is the dual image of the absolute conic (DIAC) $\boldsymbol{\omega}_\infty^*$, hence:

$$\boldsymbol{\omega}_\infty^* = \mathbf{A} \cdot \mathbf{A}^T \tag{1.194}$$

And so, expression (1.193) becomes in any **projective** stratum:

$$\mathbf{Q}_\infty^* \sim \begin{bmatrix} \mathbf{A} \cdot \mathbf{A}^T & -\mathbf{A} \cdot \mathbf{A}^T \cdot \boldsymbol{\pi}_\infty \\ -\boldsymbol{\pi}_\infty^T \cdot \mathbf{A} \cdot \mathbf{A}^T & \boldsymbol{\pi}_\infty^T \cdot \mathbf{A} \mathbf{A}^T \cdot \boldsymbol{\pi}_\infty \end{bmatrix} = \begin{bmatrix} \boldsymbol{\omega}_\infty^* & -\boldsymbol{\omega}_\infty^* \cdot \boldsymbol{\pi}_\infty \\ -\boldsymbol{\pi}_\infty^T \cdot \boldsymbol{\omega}_\infty^* & \boldsymbol{\pi}_\infty^T \cdot \boldsymbol{\omega}_\infty^* \cdot \boldsymbol{\pi}_\infty \end{bmatrix} \quad (1.195)$$

where $\boldsymbol{\pi}_\infty$ refers to the first 3 coordinates of the *plane at infinity* Π_∞ in any projective stratum as in (1.126). Notice that since $\boldsymbol{\pi}_\infty$ contains the first 3 coordinates of the plane at infinity Π_∞ , then $\boldsymbol{\pi}_\infty$ it is equal to the director vector of such plane.

Hence, we can observe in (1.195), that the *dual absolute quadric* $\mathbf{Q}_\infty^*$ in any arbitrary projective space contains both the *dual image of the absolute conic* (DIAC) $\boldsymbol{\omega}_\infty^*$ and the direction of the plane at infinity $\boldsymbol{\pi}_\infty$. For that reason, finding the *dual absolute quadric* is the most direct way to upgrade a given projective stratum into a metric one, as it contains the two fundamental pieces needed for such upgrade.

1.4.4.- The euclidean stratum

The euclidean stratum does not differ much from the metric one, because as we mentioned in previous sections, the metric stratum was equal to the euclidean one except for some scale. Therefore, any euclidean transformation will equal to a similarity transformation like in (1.136), except for the scale factor σ , which is known ($\sigma = 1$):

$$\mathbf{T}_E \sim \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_X \\ r_{21} & r_{22} & r_{23} & t_Y \\ r_{31} & r_{32} & r_{33} & t_Z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}_3^T & 1 \end{bmatrix} \quad (1.196)$$

where r_{ij} are the components of rotation matrix $\mathbf{R}$, (3 degrees of freedom) and $\mathbf{t}$ is a 3-vector that contains translation respect the origin of coordinates (3 degrees of freedom). Consequently any euclidean transformation has 6 degrees of freedom

We can then observe that any euclidean transformed point $\mathbf{M}' = \mathbf{T}_E \cdot \mathbf{M}$, will differ from the true point by a given translation and rotation, and as we explained at the beginning of section 1.4, this is inherent to the reference camera, which is always located at the origin of coordinates.

Consequently, any given 3D reconstruction can be reconstructed as much, in a euclidean stratum where it will differ by some rotation and translation from the true 3D shape.

1.4.5.- Summary of projective geometry strata

The different projective strata are summarized on table 1.4, where we show the different strata that have been shown in previous sections, their degrees of freedom, their typical transformations and their invariants. Also a depiction of a cube in the different strata is shown in figure 1.16 for convenience.

Ambiguity	DOF	Transformation	Invariants
Projective	15	$\mathbf{T}_P \sim \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \\ p_{41} & p_{42} & p_{43} & p_{44} \end{bmatrix} \quad (1.197)$	Cross-ratio

Affine	12	$\mathbf{T}_A \sim \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{A} & \mathbf{a} \\ \mathbf{0}_3^T & 1 \end{bmatrix} \quad (1.198)$	Relative distances Along direction Parallelism <i>Plane at infinity</i>
Metric	7	$\mathbf{T}_M \sim \begin{bmatrix} \sigma r_{11} & \sigma r_{12} & \sigma r_{13} & t_x \\ \sigma r_{21} & \sigma r_{22} & \sigma r_{23} & t_y \\ \sigma r_{31} & \sigma r_{32} & \sigma r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \sigma \mathbf{R} & \mathbf{t} \\ \mathbf{0}_3^T & 1 \end{bmatrix} \quad (1.199)$	Relative distances Angles <i>Absolute Conic</i> <i>Dual Absolute Quadric</i>
Euclidean	6	$\mathbf{T}_E \sim \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}_3^T & 1 \end{bmatrix} \quad (1.200)$	Absolute distances

Table 1.4. Summary of different 3D geometry strata

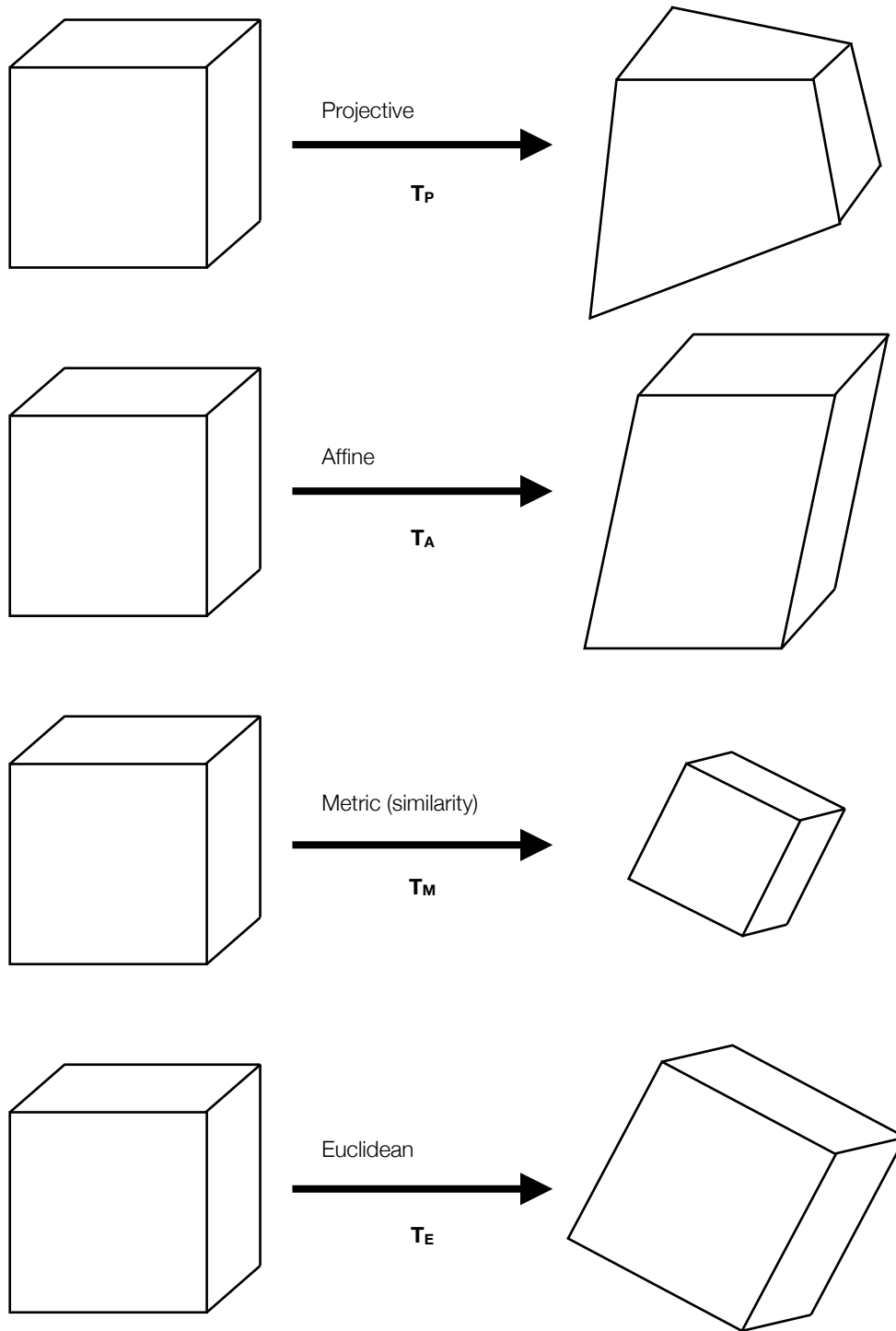


Fig. 1.16. Summary of transformations of a cube on different 3D geometry strata.

2.- Camera models

2.1.- Introduction

In order to obtain 3D information from images, it is important to know first how images in a camera are formed by projecting 3D information into a 2D image plane (i.e. the “retinal” plane).

In this chapter we will deal with the pinhole camera model, which for simplification assumes that no radial distortion is present. That way, the camera can be modeled as a simple linear $\mathbf{P}^3 \rightarrow \mathbf{P}^2$ application

Radial distortion (see section 2.2.4) is a non-linear distortion that becomes more evident at image borders and is due to camera lenses. In images with high radial distortion levels straight objects or lines tend to appear curved as we get further

from image center, hence the term radial distortion. Because it is due to camera lenses it is usually hard to model, specially if the camera can change its intrinsic parameters such as amount of zoom or focal length.

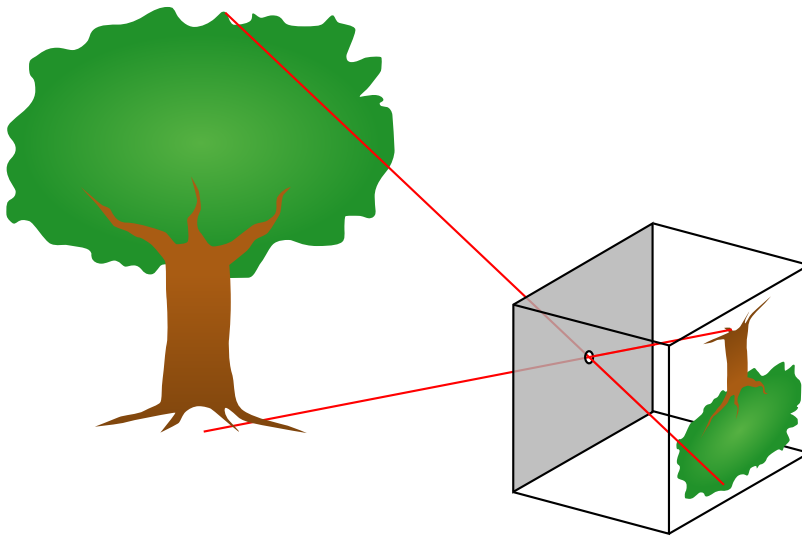


Fig. 2.1. Depiction of a pinhole camera. Rays of light traverse the small whole without distortion and the inverse of the image is captured in the photographic plate.

Because radial distortion is more evident in low quality camera models (i.e. surveillance cameras), we will assume that it can be mostly neglected. Hence, it is assumed that the camera has no lens, and for that reason the camera model is called a pinhole camera model, which is an old type of camera that instead of a lens has a small hole to capture light within a dark box. This kind of camera is depicted in figure 2.1.

2.2.- The projective canonical camera

In an euclidean world, as we saw in previous chapter, orthogonality is preserved as well as parallelism. In other words, parallel lines or planes never intersect (except at infinity).

However, in our daily life we do not see objects in a pure euclidean world. Instead, we have vanishing points so that parallel lines or planes appear to intersect at such points or at the horizon (the line at infinity). Also, as objects are located further away from the camera or our eyes, they appear smaller. This kind of effects are particularly evident in situations like a long road that extends to the horizon as shown in figure 2.2.



Fig. 2.2. Real life photograph of a road extending towards horizon. Parallel lines in the road intersect in the vanishing point in the middle of the image, and the earth surface form the horizon or plane at infinity. Also cars located further away look smaller than those located closer.

It must be noticed that the image captured by a given pinhole camera will depend on the location (i.e. projection center) of the camera, and the direction it points to (i.e. rotation). Therefore, it is obvious that if the camera is translated to a new position or points to a new direction, the captured image will change.

Looking at figure 2.2, assuming that the origin of coordinates is located at the camera projection center $\mathbf{C} = (0,0,0,1)^T$ and that the camera points towards the inhomogeneous z-axis of coordinates (i.e. the retinal plane is $z = 1$, which is perpendicular to the direction the camera is pointing to), we can model the projection process using *inhomogeneous* coordinates simply as:

$$\tilde{x} = \frac{\tilde{X}}{\tilde{Z}}, \quad \tilde{y} = \frac{\tilde{Y}}{\tilde{Z}} \quad (2.1)$$

where lowercase (x, y) are inhomogeneous image coordinates, and uppercase (X, Y, Z) are inhomogeneous 3D world coordinates. This projection is usually referred as the **canonical** one, and its associated camera as the **canonical camera**, which is depicted with more detail on figure 2.3. In such figure we can observe how a 3D *world point* $\mathbf{M}$ is projected into the *retinal plane* $\mathbf{R}$ as *image point* $\mathbf{m}$. We can also observe that the optical axis passes through the *camera projection center* $\mathbf{C}$ indicating camera direction, and its intersection with the retinal plane forms the *principal point* $\mathbf{c}$ at a distance from the camera center equal to the *focal length* f of the camera, which for the canonical camera is $f = 1$ as we will see later.

We must notice that expression (2.1) can also be represented using *homogeneous* coordinates as:

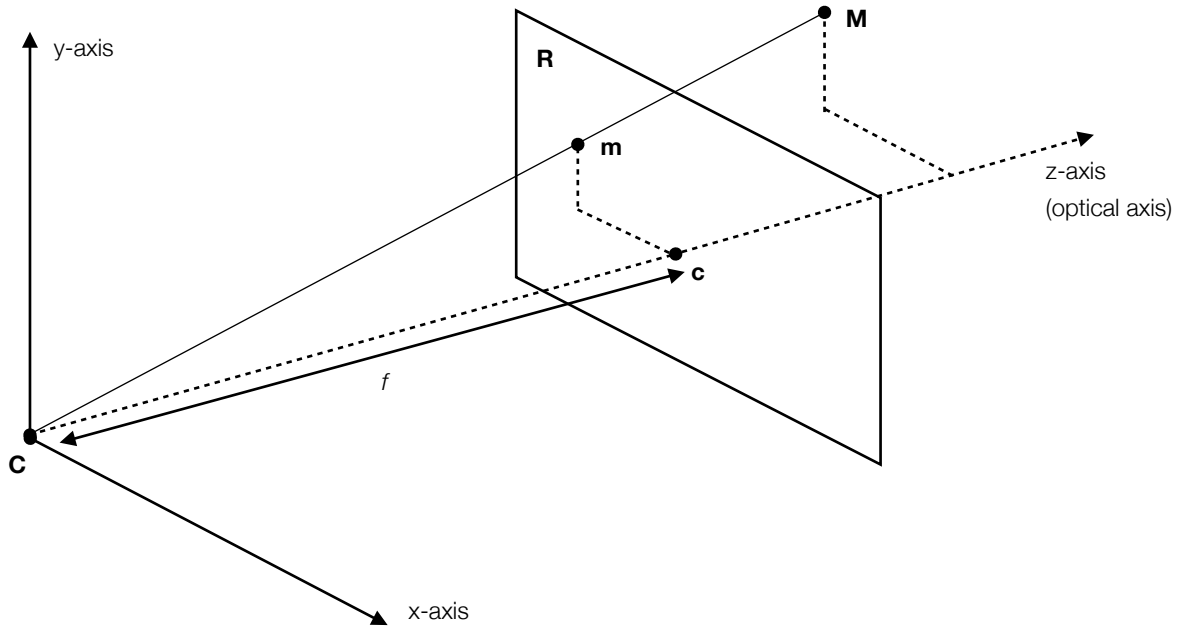


Fig. 2.3. Perspective projection. **C**: camera projection center, **c**: principal point, **M**: 3D world point, **m**: projected 3D world point on retinal plane, f : focal length of camera, **R**: retinal plane

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} \sim \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix} \quad (2.2)$$

where $\mathbf{m} = (x, y, w)^T$ is a 2D image point as defined in (1.1) and $\mathbf{M} = (X, Y, Z, W)^T$ is a 3D world point as defined in (1.65). It must be noticed that because both the image point and the 3D world point are defined using homogeneous coordinates, then expression (2.2) is defined up to scale. This will be useful as we'll see later.

Looking at expression (2.2) we can see that it can also be expressed in matrix form as:

$$\mathbf{m} = \mathbf{P} \cdot \mathbf{M} \quad (2.3)$$

where $\mathbf{m}$ is the homogeneous 2D image point, $\mathbf{M}$ is the homogeneous 3D world point and $\mathbf{P}$ is a $\mathbf{P}^3 \rightarrow \mathbf{P}^2$ linear application that contains the pinhole camera model in a 3x4 matrix.

2.2.1.- The retinal plane, principal point and focal length on the canonical camera

We have mentioned before that for the canonical camera, the retinal plane would be located at $Z = 1$. The retinal plane, also known as the image plane, is the plane where points are projected, as it is shown in figure 2.3.

We can show that for the canonical camera, 3D world points located in this plane ($Z = 1$) will project into the same (x, y) location on the retinal plane as it is on the real world. In short, for $Z = 1$, then (2.2) becomes:

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} \sim \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \\ W \end{bmatrix} = \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} \quad (2.4)$$

And consequently (2.1) can be rewritten as:

$$\tilde{x} = X, \quad \tilde{y} = Y \quad (2.5)$$

And therefore we have proved that the retinal plane is on $Z = 1$ for the canonical camera.

We can also observe that in the canonical camera the principal point $\mathbf{c}$ lies on the z-axis, which is also the optical axis. Because the retinal plane $\mathbf{R}$ is parallel to the x-y plane, then the intersection of the retinal plane and the optical axis will be located at $Z = 1$, or in other words $\mathbf{M}_c = (0,0,1,1)^T$. Therefore, the distance between the principal point expressed in inhomogeneous world coordinates, and the camera projection center $\mathbf{C} = (0,0,0,1)^T$ also expressed using inhomogeneous coordinates is:

$$f = \|\tilde{\mathbf{M}}_c - \tilde{\mathbf{C}}\| = \|(0,0,1)^T - (0,0,0)^T\| = 1 \quad (2.6)$$

which is the *focal length* as depicted in figure 2.3. From (2.6) we can also infer that any retinal plane parallel to x-y plane and located at a distance f from the camera projection center can be expressed as $Z = f$.

Also notice that the principal point $\mathbf{c}$ coordinates in the image plane after projection is:

$$\mathbf{c} = \mathbf{P} \cdot \mathbf{M}_c = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad (2.7)$$

which, expressed using inhomogeneous coordinates, is:

$$\tilde{\mathbf{c}} = (0,0)^T \quad (2.8)$$

Therefore, the principal point is located at the origin of coordinates in the retinal plane, which will be typically located in the middle of any captured image. We warn the reader to take care with this fact, as in most computer systems the origin of coordinates in an image is located in the top-left corner, whereas for the purpose of 3D point projection we will assume it is located in the middle of the image.

Hence, if using the top-left corner as the origin of coordinates, the principal point $\mathbf{c}$ can be considered to be located at the following coordinates $\mathbf{c} = (w/2, h/2, 1)^T$, where w is image width, and h is image height.

2.2.2.- The intrinsic camera parameters

The intrinsic camera parameters are a set of parameters in a camera that usually depend on the internal configuration of the camera such as the quality of the sensor or how it is located, the camera lens, etc. In the following subsections we will discuss the main intrinsic camera parameters that we will take into account in the pinhole camera model.

2.2.2.1.- Focal length

Up to this point we have seen how the canonical camera projects 3D world points into an image or retinal plane assuming that the camera projection center is located at the origin of coordinates, that the camera points towards the z-axis and that the retinal plane is located at distance $f = 1$ from the camera projection center, which in other words can be expressed as a focal length equal to one.

We must notice however, that any general camera can have any arbitrary focal length, and as we have just mentioned in previous section, any retinal plane parallel to the x-y plane at distance f from can be expressed as $Z = f$.

Therefore, if we rewrite (2.1) to define the following mapping using inhomogeneous coordinates:

$$(\tilde{X}, \tilde{Y}, \tilde{Z})^T \rightarrow (f\tilde{X}/\tilde{Z}, f\tilde{Y}/\tilde{Z})^T \quad (2.9)$$

Then any point located at $Z = f$ (using inhomogeneous coordinates), will be projected at the same (x, y) coordinates as the world point. In other words, the retinal plane is located at $Z = f$ as desired.

As in the previous case, expression (2.9) can be expressed in homogeneous matrix form as (assuming $W = 1$):

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \rightarrow \begin{pmatrix} fX \\ fY \\ Z \end{pmatrix} = \begin{bmatrix} f & 0 & 0 \\ & f & 0 \\ & & 1 & 0 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \quad (2.10)$$

Hence, from (2.3) the 3x4 camera matrix has now become:

$$\mathbf{P} = \begin{bmatrix} f & 0 \\ & f & 0 \\ & & 1 & 0 \end{bmatrix} = \text{diag}(f, f, 1) \cdot [\mathbf{I}_{3 \times 3} | \mathbf{0}] \quad (2.11)$$

2.2.2.2- Principal point offset

In expression (2.10) it is still assumed that the principal point is located at the origin of coordinates in the image plane. However, this might not always be true, as it is shown in figure 2.4. Therefore, the following mapping can also be defined:

$$(\tilde{X}, \tilde{Y}, \tilde{Z})^T \rightarrow (f\tilde{X}/\tilde{Z} + p_x, f\tilde{Y}/\tilde{Z} + p_y)^T \quad (2.12)$$

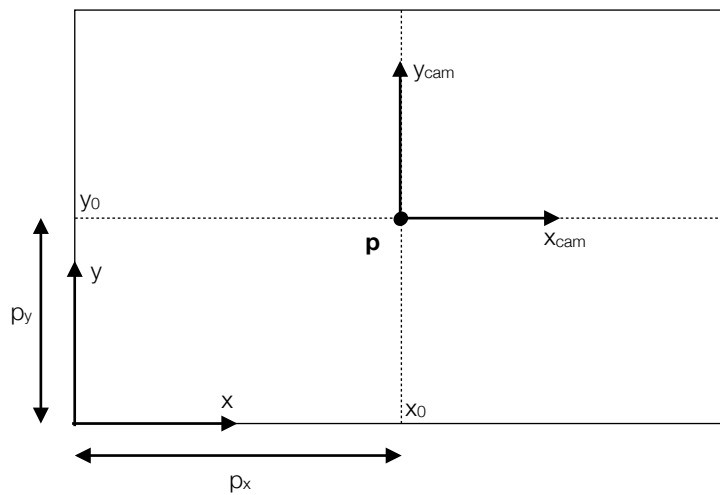


Fig. 2.4. The principal point location respect to image (x, y) coordinates located at left-bottom corner and canonical camera (x_0, y_0) coordinates located at image center. Image coordinates (x, y) can be obtained by summation of camera coordinates $(x_{cam}, y_{cam}) = (fX/Z, fY/Z)$ to the principal point location $p = (p_x, p_y)$.

where $(p_x, p_y)^T$ are the coordinates of the principal point as depicted in figure 2.4. As before, mapping in (2.12) can be expressed in matrix form using homogeneous coordinates as:

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \rightarrow \begin{pmatrix} fX + Z \cdot p_x \\ fY + Z \cdot p_y \\ Z \end{pmatrix} = \begin{bmatrix} f & p_x & 0 \\ & f & p_y \\ & & 1 & 0 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \quad (2.13)$$

Hence, as in (2.11) we can express the camera matrix as:

$$\mathbf{P} = \begin{bmatrix} f & p_x & 0 \\ & f & p_y \\ & & 1 & 0 \end{bmatrix} = \mathbf{K} \cdot [\mathbf{I}_{3 \times 3} | \mathbf{0}_3] \quad (2.14)$$

where $\mathbf{K}$ is called the camera calibration matrix and contains all the intrinsic parameters of the camera. Up to this point $\mathbf{K}$ is an upper triangular 3x3 matrix that has the following form:

$$\mathbf{K} = \begin{bmatrix} f & & p_x \\ & f & p_y \\ & & 1 \end{bmatrix} \quad (2.15)$$

Notice however, that most common cameras have the sensor located respect to the lens in such a way that the principal point is located at the origin of coordinates in the image plane (i.e. the middle of the image), or at least very close to it. As we will see, this fact will help us to put some constraints when attempting to self-calibrate the intrinsic parameters of the camera in chapter 5.

2.2.2.3.- CCD cameras

Up to this point we have not given units in 3D world point coordinates and 2D image coordinates. Typical coordinates for images are pixels, as it is the smallest entity that can be represented on a digital image. Hence, because we will attempt to obtain 3D shape from images in a metric stratum without any further reference to the real scale of pictured objects, then we will always work in **pixel units**. For that reason real word coordinates need to be converted to pixels by taking into account the number of pixels per unit distance m .

It must be noticed however that CCD sensors do not always have squared pixels, the number of pixels per unit distance m might be different in the horizontal x and vertical y directions in the image plane, hence we now have two number of pixels per unit distance values (m_x, m_y) , or in other words the horizontal size of a pixel is $1/m_x$ and the vertical size of a pixel is $1/m_y$. Consequently, the transformation from world coordinates to pixel units is obtained by multiplying (2.15) by $\text{diag}(m_x, m_y, 1)$. Hence, the general form of the calibration matrix $\mathbf{K}$ for a CC camera in **pixel units** is:

$$\mathbf{K} = \text{diag}(m_x, m_y, 1) \cdot \begin{bmatrix} f & & p_x \\ & f & p_y \\ & & 1 \end{bmatrix} = \begin{bmatrix} \alpha_x & & x_0 \\ & \alpha_y & y_0 \\ & & 1 \end{bmatrix} \quad (2.16)$$

where $\alpha_x = f \cdot m_x$, and $\alpha_y = f \cdot m_y$ are the focal length of the camera in terms of pixel dimension in the x and y image directions respectively. Similarly, (x_0, y_0) are now the coordinates of the principal point in terms of pixel dimensions. Notice that typically

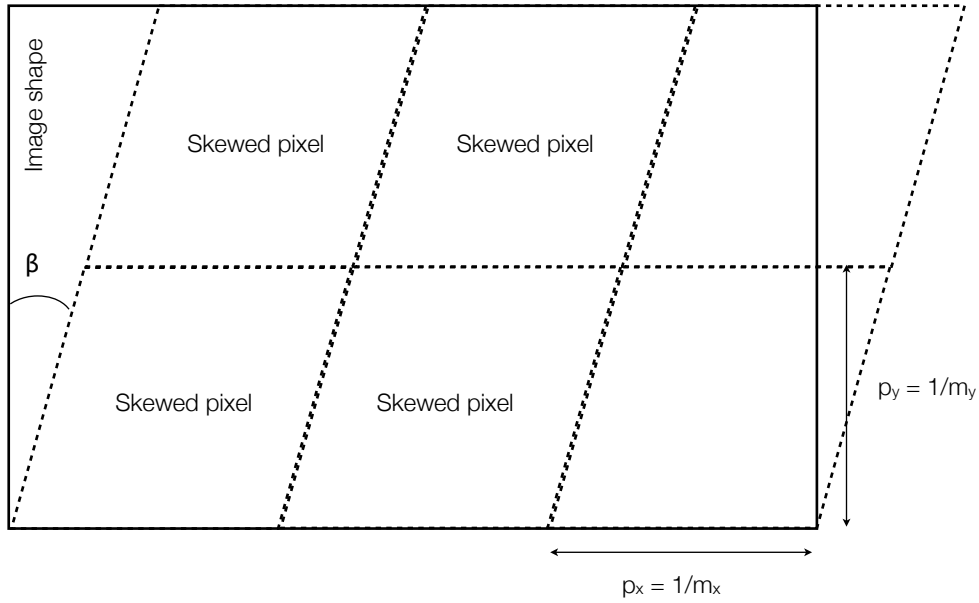


Fig. 2.5. The skew angle β respect to pixel size $(p_x, p_y) = (1/m_x, 1/m_y)$

the principal point will be set around the image center, which will set the origin of coordinates in pixels at the bottom-left corner.

Finally, it must be taken into account that a CCD sensor might not have all the pixels perfectly aligned in a squared (or rectangular) grid. For most cameras pixels are correctly aligned, but in low quality cameras this might not be exactly true. Hence, the calibration matrix $\mathbf{K}$ can be defined with further generality as:

$$\mathbf{K} = \begin{bmatrix} \alpha_x & s & x_0 \\ & \alpha_y & y_0 \\ & & 1 \end{bmatrix} \quad (2.17)$$

where s is the skew parameter which defines the amount of slope between consecutive pixels in a grid. It can be further defined as:

$$s = \alpha_y \tan \beta \quad (2.18)$$

where β is the skew angle as depicted in figure 2.5.

Notice that because the skew angle will typically be small or close to zero if the camera has enough quality, then the skew parameter will also be close to zero. Also notice that the factor α_y/α_x is commonly known as the *aspect ratio*.

In summary, any 3D world point respect to a camera located at the origin and pointing towards the z-axis can be expressed as:

$$\mathbf{m} = \mathbf{K} \cdot \begin{bmatrix} \mathbf{I}_{3 \times 3} & \mathbf{0}_3 \end{bmatrix} \cdot \mathbf{M}_{\text{cam}} = \mathbf{P} \cdot \mathbf{M}_{\text{cam}} \quad (2.19)$$

where $\mathbf{M}_{\text{cam}}$ refers to a 3D world point in such coordinates (camera center at origin and pointing towards z-axis) and $\mathbf{P}$ would be the 3x4 camera projection matrix which is equal to $\mathbf{P} = \mathbf{K} \cdot \begin{bmatrix} \mathbf{I}_{3 \times 3} & \mathbf{0}_3 \end{bmatrix}$, taking into account that $\mathbf{K}$ follows expression (2.17).

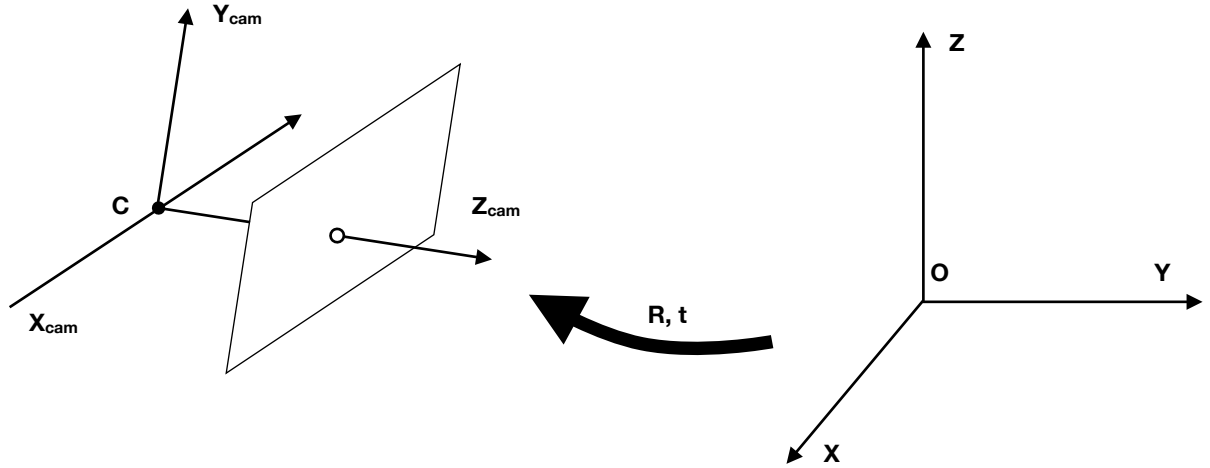


Fig. 2.6. Euclidean transformation between world and camera coordinate frames. After moving the origin $\mathbf{O}$ to the camera center $\mathbf{C}$ and applying a given rotation $\mathbf{R}$, we can transform points $\mathbf{M}$ into the camera coordinate frame $\mathbf{M}_{\text{cam}}$.

2.2.3.- The extrinsic camera parameters

The extrinsic camera parameters is a set of parameters that are external to the camera. These settings are basically related to camera rotation and location, because cameras can be moved and rotated freely in a 3D space.

If $\tilde{\mathbf{M}}$ is an inhomogeneous 3-vector representing the coordinates of a 3D world point respect to the world coordinates frame (i.e. the origin of coordinates is located on the reference camera center $\tilde{\mathbf{C}}$, and $\tilde{\mathbf{M}}_{\text{cam}}$ represents a 3D world point laying in the intersection of the ray of light that goes from the camera center to the $\tilde{\mathbf{M}}$ point with the retinal plane $\mathbf{R}$, as depicted in figure 2.6. Therefore $\tilde{\mathbf{M}}_{\text{cam}}$ can be expressed as:

$$\tilde{\mathbf{M}}_{\text{cam}} = \mathbf{R}(\tilde{\mathbf{M}} - \tilde{\mathbf{C}}) \quad (2.20)$$

where $\mathbf{R}$ is a 3x3 rotation matrix as the one we explained in (1.132).

Geometrically, expression (2.19) can be seen as if we set the origin of coordinates in the *camera projection center* by removing $\tilde{\mathbf{C}}$ from $\tilde{\mathbf{M}}$ and then we applied a certain rotation $\mathbf{R}$ that would be centered in the camera projection center because it is now the origin of coordinates.

As always, expression (2.19) can also be expressed in matrix form using homogeneous coordinates as follows:

$$\mathbf{M}_{\text{cam}} = \begin{bmatrix} \mathbf{R} & -\mathbf{R}\tilde{\mathbf{C}} \\ \mathbf{0}_3^T & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R} & -\mathbf{R}\tilde{\mathbf{C}} \\ \mathbf{0}_3^T & 1 \end{bmatrix} \mathbf{M} \quad (2.21)$$

Then, by combining both (2.21) and (2.19) we obtain that any 3D world point is projected into the image plane by means of the pinhole camera model as:

$$\begin{aligned}
\mathbf{m} &= \mathbf{K} \cdot \left[\mathbf{I}_{3 \times 3} \mid \mathbf{0}_3 \right] \cdot \begin{bmatrix} \mathbf{R} & -\mathbf{R} \cdot \tilde{\mathbf{C}} \\ \mathbf{0}_3^T & 1 \end{bmatrix} \cdot \mathbf{M} = \\
&= \mathbf{K} \cdot \begin{bmatrix} \mathbf{I}_{3 \times 3} \cdot \mathbf{R} + \mathbf{0}_3 \cdot \mathbf{0}_3^T & -\mathbf{I}_{3 \times 3} \cdot \mathbf{R} \cdot \tilde{\mathbf{C}} + 1 \cdot \mathbf{0}_3 \end{bmatrix} \cdot \mathbf{M} = \\
&= \mathbf{K} \cdot \begin{bmatrix} \mathbf{R} & -\mathbf{R} \cdot \tilde{\mathbf{C}} \end{bmatrix} \cdot \mathbf{M} = \mathbf{K} \cdot \mathbf{R} \cdot \left[\mathbf{I}_{3 \times 3} \mid -\tilde{\mathbf{C}} \right] \cdot \mathbf{M}
\end{aligned} \tag{2.22}$$

Hence, the 3x4 camera projection matrix can be expressed as:

$$\mathbf{P} = \mathbf{K} \cdot \mathbf{R} \cdot \left[\mathbf{I}_{3 \times 3} \mid -\tilde{\mathbf{C}} \right] \tag{2.23}$$

where $\mathbf{K}$ is the 3x3 camera calibration matrix that contains the intrinsic camera parameters, $\mathbf{R}$ is a 3x3 rotation matrix so that $\mathbf{R} \cdot \mathbf{R}^{-1} = \mathbf{R} \cdot \mathbf{R}^T = \mathbf{I}_{3 \times 3}$ and $\det(\mathbf{R}) = 1$, $\mathbf{I}_{3 \times 3}$ is the 3x3 identity matrix, and $\tilde{\mathbf{C}}$ is the camera center expressed using inhomogeneous coordinates.

Sometimes, the camera projection matrix $\mathbf{P}$ is more conveniently expressed as:

$$\mathbf{P} = \mathbf{K} \cdot \left[\mathbf{R} \mid \mathbf{t} \right], \quad \mathbf{t} = -\mathbf{R} \cdot \tilde{\mathbf{C}} \tag{2.24}$$

where the camera projection center $\mathbf{C}$ is no longer explicit and $\mathbf{t}$ can be seen as an amount of translation, as shown in figure 2.6, when using the following world to image transformation:

$$\tilde{\mathbf{M}}_{\text{cam}} = \mathbf{R} \tilde{\mathbf{M}} + \mathbf{t} \tag{2.25}$$

In section 2.3 we will also see that $\mathbf{t}$, the last column of $\mathbf{P}$, can also be interpreted as the image of the world origin on the retinal plane.

2.2.4.- Deviations from the camera model

As we have seen in previous sections, we have modeled the camera projection of 3D world points into 2D image points by means of a linear application using the pinhole camera model.

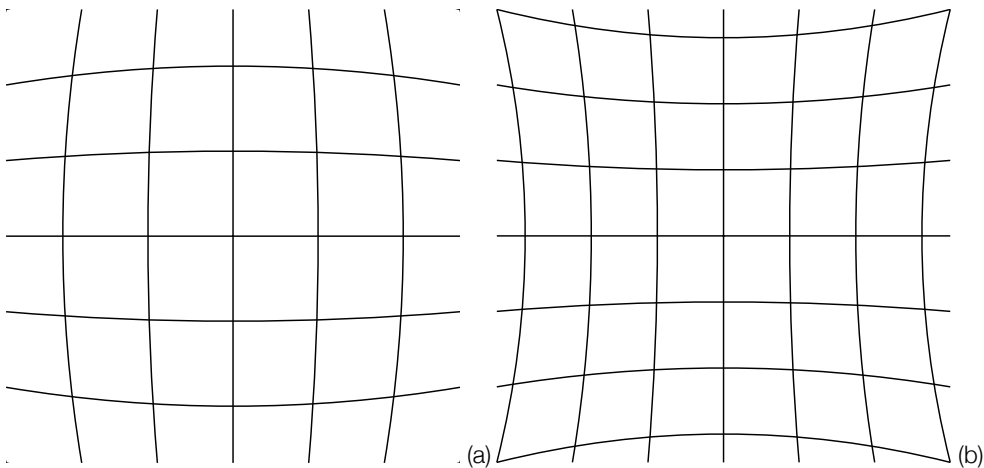


Fig. 2.7. Examples of radial distortion. (a) Barrel distortion. (b) Pincushion distortion

However, in the real world, the projection process is not completely linear. Instead different kinds of distortions or effects can be taken into account. Most of them are due to failures in the optical system, which produce different kinds of aberrations like astigmatism, chromatic aberrations or spherical aberrations when rays of light are not properly collimated in all light wavelengths, and cannot be modeled in a simple way as in the pinhole camera model.

Many of the previous deviations from the linear model are negligible for most cameras nowadays, and only the radial distortion might be noticeable for low quality cameras. Otherwise, if the camera is assumed to be good enough, radial distortion can be neglected as well.

The **radial distortion** can be seen as a linear displacement of image points radially from the center of the image, caused by the fact that objects at different angular distance from the lens axis (i.e. the principal point) undergo different magnifications. This produces a distortion such that straight lines or planes appear curved, specially as they appear closer to image borders, where distortion is more noticeable than at image center. This is depicted for ease of understanding on figure 2.7.

Therefore, the inhomogeneous coordinates in the undistorted image plane (x_u, y_u) can be obtained from the observed distorted image coordinates (x_d, y_d) by means of Brown's distortion model using the following equations:

$$\begin{aligned} x_u &= x_d + (x_d - x_c) \left(K_1 r^2 + K_2 r^4 + \dots \right) + \left(P_1 (r^2 + 2(x_d - x_c)^2) + 2P_2 (x_d - x_c)(y_d - y_c) \right) \left(1 + P_3 r^2 + \dots \right) \\ y_u &= y_d + (y_d - y_c) \left(K_1 r^2 + K_2 r^4 + \dots \right) + \left(P_1 (r^2 + 2(y_d - y_c)^2) + 2P_2 (x_d - x_c)(y_d - y_c) \right) \left(1 + P_3 r^2 + \dots \right) \end{aligned} \quad (2.26)$$

where (x_u, y_u) is an undistorted inhomogeneous image point coordinates, (x_d, y_d) is its corresponding distorted (observed) inhomogeneous image point coordinates, (x_c, y_c) is the center of distortion (i.e. typically is the principal point which is located around the image center), K_n is the n^{th} radial distortion coefficient, P_n is the n^{th} tangential distortion coefficient and r is defined as the distance from the distorted (observed) image point respect to the center of distortion. Therefore r can be defined as:

$$r = \sqrt{(x_d - x_c)^2 + (y_d - y_c)^2} \quad (2.27)$$

It must be noticed that in Brown's distortion model shown in (2.26), when K_1 is positive, then we obtain a barrel radial distortion as shown in fig. 2.7.a, whereas when K_1 is negative, then distortion has a pincushion shape, as in fig. 2.7.b.

In most situations tangential distortion coefficients P_n can be neglected respect to radial ones K_n , therefore, expression (2.26) can be further simplified into:

$$\begin{aligned} x_u &= x_d + (x_d - x_c) \left(K_1 r^2 + K_2 r^4 + \dots \right) \\ y_u &= y_d + (y_d - y_c) \left(K_1 r^2 + K_2 r^4 + \dots \right) \end{aligned} \quad (2.28)$$

Where we can observe that radial distortion coefficients K_1, K_2 in (2.28) can be linearly expressed as follows:

$$\begin{aligned} (x_d - x_c) r^2 K_1 + (x_d - x_c) r^4 K_2 + \dots &= x_u - x_d \\ (y_d - y_c) r^2 K_1 + (y_d - y_c) r^4 K_2 + \dots &= y_u - y_d \end{aligned} \quad (2.29)$$

which can be expressed in matrix form as:

$$\begin{bmatrix} (x_{1d}-x_c) \cdot r^2 & (x_{1d}-x_c) \cdot r^4 & \cdots & (x_{1d}-x_c) \cdot r^{2j} \\ (y_{1d}-y_c) \cdot r^2 & (y_{1d}-y_c) \cdot r^4 & \cdots & (y_{1d}-y_c) \cdot r^{2j} \\ \vdots & \vdots & \ddots & \vdots \\ (x_{id}-x_c) \cdot r^2 & (x_{id}-x_c) \cdot r^4 & \cdots & (x_{id}-x_c) \cdot r^{2j} \\ (y_{id}-y_c) \cdot r^2 & (y_{id}-y_c) \cdot r^4 & \cdots & (y_{id}-y_c) \cdot r^{2j} \end{bmatrix} \begin{bmatrix} K_1 \\ K_2 \\ \vdots \\ K_j \end{bmatrix} = \begin{bmatrix} x_{1u} - x_{1d} \\ y_{1u} - y_{1d} \\ \vdots \\ x_{iu} - x_{id} \\ y_{iu} - y_{id} \end{bmatrix} \quad (2.30)$$

where (2.30) can be solved as long as the number of points i is smaller than $j/2$, where j is the number of radial distortion coefficients to be solved.

Therefore, if only K_1 and K_2 are needed, then only the correspondence between the distorted and undistorted horizontal and vertical coordinates of a **single point** are enough to solve the linear system as follows:

$$\begin{bmatrix} (x_{1d}-x_c) \cdot r^2 & (x_{1d}-x_c) \cdot r^4 \\ (y_{1d}-y_c) \cdot r^2 & (y_{1d}-y_c) \cdot r^4 \end{bmatrix} \begin{bmatrix} K_1 \\ K_2 \end{bmatrix} = \begin{bmatrix} x_{1u} - x_{1d} \\ y_{1u} - y_{1d} \end{bmatrix} \quad (2.31)$$

Notice that if more than one correspondence between the coordinates of distorted and undistorted points is known, then (2.31) can be solved in a Least Mean Squared Error (LMSE) fashion by means of the Moore-Penrose pseudo-inverse. Also a LMSE solution can be found in (2.30) by means of the pseudo-inverse when the number of points i is greater than $j/2$, or in other words, when $2i > j$.

2.3.- Camera decomposition

We saw in (2.24) how the pinhole camera matrix is formed by taking into account a series of parameters such as rotation, translation or other internal parameters such as focal length, principal point or skewness. Our goal now is to attempt to determine such parameters from any provided 3x4 P matrix. We will see that indeed the camera matrix P can be decomposed in such a way that geometric properties can be retrieved.

It must be noticed that if we take a camera matrix P in the form:

$$\mathbf{P} = \mathbf{K} \cdot \mathbf{R} \left[\mathbf{I}_{3 \times 3} \mid -\tilde{\mathbf{C}} \right] = \left[\mathbf{M}_P \mid \mathbf{p}_4 \right] \quad (2.32)$$

where $\mathbf{M}_P$ is the 3x3 top-left minor of $\mathbf{P}$ and $\mathbf{p}_4$ is the 4th column of $\mathbf{P}$.

Then, we will assume that the 3x3 top-left minor $\mathbf{M}_P = \mathbf{K} \cdot \mathbf{R}$ is not singular (i.e. $\det(\mathbf{M}_P) = \det(\mathbf{K} \cdot \mathbf{R}) \neq 0$). Otherwise the camera is called an infinite camera because its center is located on the plane at infinity. The demonstration of this fact is out of the scope of this document, however, we will see in section 2.4 that in finite projective cameras, where $\mathbf{M}_P$ is not singular, its determinant is related to depth of 3D world points respect to the cameras as well as to the camera orientation itself.

Notice that in the following subsections we will provide the decomposition for the case of finite cameras only.

2.3.1.- Camera center

The camera matrix $\mathbf{P}$ is a 3x4 matrix. Because we have assumed that $\mathbf{P}$ belongs to a finite camera where $\mathbf{M}_P$ is not singular, then the rank of $\mathbf{P}$ is 3, which is the number of rows that $\mathbf{P}$ has. However, $\mathbf{P}$ has 4 columns, which means that there can be some vector $\mathbf{C}$ different from zero that fulfills:

$$\mathbf{P} \cdot \mathbf{C} = \mathbf{0}_3 \quad (2.33)$$

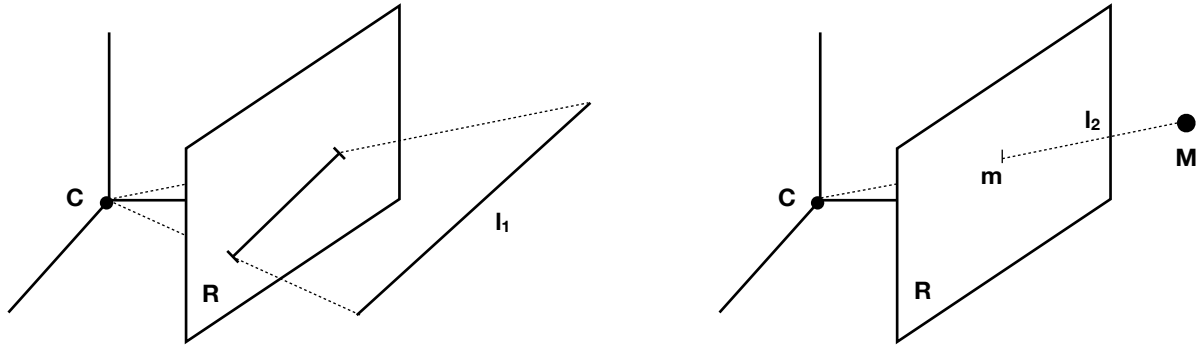


Fig. 2.8. Left: Projection of a general line $\mathbf{l}_1$ on the retinal plane $\mathbf{R}$. Right: Projection of a ray of light $\mathbf{l}_2$ passing through the camera center $\mathbf{C}$.

Then, $\mathbf{C}$ is a 4-vector representing the right null-space of $\mathbf{P}$. It will be proved next that $\mathbf{C}$ is actually the camera projection center expressed using homogeneous coordinates.

Proof 2.1

It can be shown geometrically that in general the projection of any 3D line $\mathbf{l}_1$ into the retinal plane $\mathbf{R}$ results in another line on the projected image; however, if a line $\mathbf{l}_2$ passes through the camera projection center $\mathbf{C}$ and another 3D world point $\mathbf{M}$, then the projection of such 3D line (i.e. also known as ray of light) becomes in this case a single point $\mathbf{m}$ projected into the image plane, which is indeed the projection of the world point $\mathbf{M}$, as can be seen on figure 2.8. Therefore, any point $\mathbf{M}'$ on line $\mathbf{l}_2$ that passes through $\mathbf{C}$ and $\mathbf{M}$ can be expressed as:

$$\mathbf{M}' = \mathbf{A} + \lambda \cdot \mathbf{C} \quad (2.34)$$

where λ is an arbitrary scalar value.

And consequently the projection of any of those points $\mathbf{M}'$ is:

$$\mathbf{m}' = \mathbf{P} \cdot \mathbf{M}' = \mathbf{P} \cdot (\mathbf{A} + \lambda \cdot \mathbf{C}) = \mathbf{P} \cdot \mathbf{A} + \lambda \cdot \mathbf{P} \cdot \mathbf{C} \quad (2.35)$$

where $\mathbf{m}'$ is an image point.

As we have seen in figure 2.8, if $\mathbf{l}_1$ was a general 3D line, then the projection of world points $\mathbf{M}'$ laying on such line would result on a line on the image plane and hence $\mathbf{m}'$ would not be unique (i.e. would depend on lambda) as it would represent all projected points in the line. However, because we assumed that the 3D line was a ray of light passing through the camera center $\mathbf{C}$, then $\mathbf{m}'$ has to be unique (i.e. the projection is the same for all points laying in the ray of light) and therefore independent of λ , which can have any arbitrary value. Consequently the $\mathbf{P} \cdot \mathbf{C}$ term in (2.35) has to be zero, and we have proved that the camera projection center $\mathbf{C}$ is the right null-space of camera projection matrix $\mathbf{P}$.

Looking at (2.33) we can then observe that the projection of the camera center $\mathbf{C}$ results in image point $(0,0,0)^T = \mathbf{P} \cdot \mathbf{C}$, which if we attempted to represent it using inhomogeneous coordinates we would see it is not defined (i.e. because 0/0 is not defined). Therefore we can assert that the camera center is the unique point in space for which the image is undefined.

2.3.1.1.- Computing the camera projection center by means of SVD

In order to compute the camera projection center $\mathbf{C}$, since we now know that it is equal to the right null-space of $\mathbf{P}$, we can use its singular value decomposition (SVD) to retrieve $\mathbf{C}$. Therefore, we can express the SVD of $\mathbf{P}$ as:

$$\mathbf{P} = \mathbf{U} \cdot \mathbf{S} \cdot \mathbf{V}^T \quad (2.36)$$

where $\mathbf{U}$ is a 3x4 orthonormal matrix ($\mathbf{U} \cdot \mathbf{U}^T = \mathbf{I}_{3 \times 3}$), $\mathbf{S}$ is a 4x4 diagonal matrix containing the singular values ordered from largest one on top-left element of $\mathbf{S}$ to the smallest one on bottom-right element of $\mathbf{S}$, and finally $\mathbf{V}$ is a 4x4 orthonormal matrix ($\mathbf{V} \cdot \mathbf{V}^T = \mathbf{I}_{4 \times 4}$). Notice that the SVD is not unique but we assume that it is provided this way as most typical implementations do.

Because the rank of matrix $\mathbf{P}$ is 3, then the 4th singular value located on the 4th column of $\mathbf{S}$ will be zero, therefore $\mathbf{S}$ has the following form:

$$\mathbf{S} = \begin{bmatrix} s_1 & & & \\ & s_2 & & \\ & & s_3 & \\ & & & 0 \end{bmatrix}, \quad s_1 \geq s_2 \geq s_3 \quad (2.37)$$

Also, because $\mathbf{V}$ is orthonormal and assuming that $\mathbf{v}_i$ is the i^{th} column of $\mathbf{V}$, then:

$$\begin{aligned} \mathbf{V} \cdot \mathbf{V}^T &= \mathbf{V}^T \cdot \mathbf{V} = \mathbf{I}_{4 \times 4} \Rightarrow \\ \mathbf{V}^T \cdot \mathbf{V} &= \begin{bmatrix} \mathbf{v}_1^T \\ \mathbf{v}_2^T \\ \mathbf{v}_3^T \\ \mathbf{v}_4^T \end{bmatrix} \begin{bmatrix} \mathbf{v}_1 & \mathbf{v}_2 & \mathbf{v}_3 & \mathbf{v}_4 \end{bmatrix} = \\ &= \begin{bmatrix} \mathbf{v}_1^T \cdot \mathbf{v}_1 & \mathbf{v}_1^T \cdot \mathbf{v}_2 & \mathbf{v}_1^T \cdot \mathbf{v}_3 & \mathbf{v}_1^T \cdot \mathbf{v}_4 \\ \mathbf{v}_2^T \cdot \mathbf{v}_1 & \mathbf{v}_2^T \cdot \mathbf{v}_2 & \mathbf{v}_2^T \cdot \mathbf{v}_3 & \mathbf{v}_2^T \cdot \mathbf{v}_4 \\ \mathbf{v}_3^T \cdot \mathbf{v}_1 & \mathbf{v}_3^T \cdot \mathbf{v}_2 & \mathbf{v}_3^T \cdot \mathbf{v}_3 & \mathbf{v}_3^T \cdot \mathbf{v}_4 \\ \mathbf{v}_4^T \cdot \mathbf{v}_1 & \mathbf{v}_4^T \cdot \mathbf{v}_2 & \mathbf{v}_4^T \cdot \mathbf{v}_3 & \mathbf{v}_4^T \cdot \mathbf{v}_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned} \quad (2.38)$$

where we can observe that only the product of a column of $\mathbf{V}$ with itself is equal to one, and otherwise it is zero (i.e. a Kronecker's delta), just as follows:

$$\delta_{ij} = \mathbf{v}_i^T \cdot \mathbf{v}_j = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases} \quad (2.39)$$

And then, by expanding (2.36), and using (2.37) and (2.39) we obtain that:

$$\begin{aligned}
\mathbf{P} \cdot \mathbf{v}_4 &= \mathbf{U} \cdot \mathbf{S} \cdot \mathbf{V}^T \cdot \mathbf{v}_4 = \mathbf{U} \cdot \begin{bmatrix} s_1 & 0 & 0 & 0 \\ 0 & s_2 & 0 & 0 \\ 0 & 0 & s_3 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{v}_1^T \\ \mathbf{v}_2^T \\ \mathbf{v}_3^T \\ \mathbf{v}_4^T \end{bmatrix} \cdot \mathbf{v}_4 = \\
&= \mathbf{U} \cdot \begin{bmatrix} s_1 \cdot \mathbf{v}_1^T \cdot \mathbf{v}_4 \\ s_2 \cdot \mathbf{v}_2^T \cdot \mathbf{v}_4 \\ s_3 \cdot \mathbf{v}_3^T \cdot \mathbf{v}_4 \\ 0 \cdot \mathbf{v}_4^T \cdot \mathbf{v}_4 \end{bmatrix} = \mathbf{U} \cdot \begin{bmatrix} s_1 \cdot 0 \\ s_2 \cdot 0 \\ s_3 \cdot 0 \\ 0 \cdot 1 \end{bmatrix} = \mathbf{U} \cdot \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = (0,0,0)^T = \mathbf{0}_3
\end{aligned} \tag{2.40}$$

Therefore, we can observe that the product of $\mathbf{P}$ with the last column of $\mathbf{V}$ results in the null vector, and then the 4th column of $\mathbf{V}$, which corresponds to the null singular value of $\mathbf{S}$, is the right null-space of $\mathbf{P}$. Because SVD decomposition is not unique and singular values could be given in a different order, then we could also assert that the column of $\mathbf{V}$ corresponding to the null singular value of $\mathbf{S}$ is the right null-space of $\mathbf{P}$.

Consequently, as we saw in Proof 2.1, we can infer that the column of $\mathbf{V}$ corresponding to the null singular value of $\mathbf{S}$ is also the camera projection center $\mathbf{C}$ of $\mathbf{P}$ using homogeneous coordinates.

2.3.1.2.- Algebraically computing the camera projection center

We have already seen that the camera center $\mathbf{C}$ can be obtained by means of SVD decomposition of the camera matrix $\mathbf{P}$ as $\mathbf{C}$ is the null-vector of $\mathbf{P}$, or in other words as we saw in (2.33) $\mathbf{P} \cdot \mathbf{C} = \mathbf{0}_3$. However, instead of computing the whole SVD decomposition, the same results can be obtained in a more computationally efficient way by means of the following expression:

$$\begin{aligned}
X &= \det([\mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4]) & Y &= -\det([\mathbf{p}_1, \mathbf{p}_3, \mathbf{p}_4]) \\
Z &= \det([\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_4]) & W &= -\det([\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3])
\end{aligned} \tag{2.41}$$

where the camera projection center $\mathbf{C}$ is expressed in homogenous coordinates as $\mathbf{C} = (X, Y, Z, W)^T$. The proof is shown next.

Proof 2.2

Notice that expression (2.41) has been obtained similarly to (1.69) and (1.70) where any co-planar set of 4 points have null determinant and the expression of the plane is obtained as the determinants of the minors. By means of duality, then the intersection of a set of 3 planes generally is a point, and as we will see in section 2.3.3, the first row of the camera matrix $\mathbf{P}$, namely $\mathbf{p}^1$ contains the vertical image axis plane, the second row of the camera matrix $\mathbf{p}^2$ contains the horizontal image axis plane, and the 3rd row of the camera matrix, $\mathbf{p}^3$ contains the principal plane, which is parallel to the image plane. The camera center $\mathbf{C}$ is contained in all of those 3 planes.

Because a point is obtained by the intersection of three planes, adding another plane intersecting on the same point is redundant. Therefore, setting those three planes $\mathbf{p}^1$, $\mathbf{p}^2$, $\mathbf{p}^3$ along with another 4th plane $\Pi = (A, B, C, D)^T$ that also intersects in the same point $\mathbf{C}$ on the same matrix results on rank-3 matrix with determinant equal to zero. Just as follows:

$$\mathbf{H} = \begin{bmatrix} \mathbf{P} \\ \Pi \end{bmatrix} = \begin{bmatrix} \mathbf{p}^{1T} \\ \mathbf{p}^{2T} \\ \mathbf{p}^{3T} \\ \Pi \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \\ A & B & C & D \end{bmatrix} = \begin{bmatrix} \mathbf{p}_1 & \mathbf{p}_2 & \mathbf{p}_3 & \mathbf{p}_4 \\ A & B & C & D \end{bmatrix} \Rightarrow \quad (2.42)$$

$$\det(\mathbf{H}) = 0$$

where $\mathbf{p}^{iT}$ is i^{th} row of $\mathbf{P}$ and $\mathbf{p}_j$ is j^{th} column of $\mathbf{P}$.

Then, the determinant of $\mathbf{H}$ can be expressed as in (1.70) by means of *Laplace's formula* in terms of the determinants of the minors as follows:

$$\begin{aligned} \det(H) = & A \cdot \det \begin{bmatrix} p_{12} & p_{13} & p_{14} \\ p_{22} & p_{23} & p_{24} \\ p_{32} & p_{33} & p_{34} \end{bmatrix} - B \cdot \det \begin{bmatrix} p_{11} & p_{13} & p_{14} \\ p_{21} & p_{23} & p_{24} \\ p_{31} & p_{33} & p_{34} \end{bmatrix} + \\ & + C \cdot \det \begin{bmatrix} p_{11} & p_{12} & p_{14} \\ p_{21} & p_{22} & p_{24} \\ p_{31} & p_{32} & p_{34} \end{bmatrix} - D \cdot \det \begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{bmatrix} = 0 \end{aligned} \quad (2.43)$$

Resulting in the following plane equation:

$$\Pi^T \cdot \mathbf{C} = A \cdot \det \begin{bmatrix} \mathbf{p}_2 & \mathbf{p}_3 & \mathbf{p}_4 \end{bmatrix} - B \cdot \det \begin{bmatrix} \mathbf{p}_1 & \mathbf{p}_3 & \mathbf{p}_4 \end{bmatrix} + C \cdot \det \begin{bmatrix} \mathbf{p}_1 & \mathbf{p}_2 & \mathbf{p}_4 \end{bmatrix} - D \cdot \det \begin{bmatrix} \mathbf{p}_1 & \mathbf{p}_2 & \mathbf{p}_3 \end{bmatrix} = 0 \quad (2.44)$$

And consequently to complete the proof, by means of (1.74) point $\mathbf{C}$ must follow expression (2.41)

2.3.1.3.- Computing the camera projection center for finite cameras

Notice that there are other ways of computing the camera projection center. For instance, by looking at expression (2.32) we can observe that:

$$\mathbf{p}_4 = -\mathbf{K} \cdot \mathbf{R} \cdot \tilde{\mathbf{C}} = -\mathbf{M}_p \cdot \tilde{\mathbf{C}} \quad (2.45)$$

where $\mathbf{p}_4$ is the 4th column of $\mathbf{P}$.

Therefore, because we have assumed a finite camera model where $\mathbf{M}_p = \mathbf{K} \cdot \mathbf{R}$ is not singular, then the camera projection center using *inhomogeneous* coordinates can be computed as:

$$\tilde{\mathbf{C}} = -\mathbf{M}_p^{-1} \cdot \mathbf{p}_4 = -(\mathbf{K} \cdot \mathbf{R})^{-1} \cdot \mathbf{p}_4 \quad (2.46)$$

We remark that expression (2.46) is only valid for finite cameras, while results in sections 2.3.1.1 and 2.3.1.2 are valid for any camera.

2.3.2.- Vanishing points and image of the world origin

By expressing the camera projection matrix $\mathbf{P}$ in terms of its columns $\mathbf{p}_i$, $i = 1, \dots, 4$, we can observe that such vectors $\mathbf{p}_1$, $\mathbf{p}_2$, $\mathbf{p}_3$, $\mathbf{p}_4$ have a geometric meaning. Indeed we will prove that $\mathbf{p}_1$, $\mathbf{p}_2$, $\mathbf{p}_3$ are the vanishing points of the world coordinate X , Y and Z axes respectively, and that $\mathbf{p}_4$ is the image of the world origin.

Hence, taking into account that $\mathbf{M}_x$, $\mathbf{M}_y$, $\mathbf{M}_z$ are vanishing points which are located at infinity in the x , y , and z axes respectively, with the following expressions:

$$\begin{aligned}\mathbf{M}_x &= (1, 0, 0, 0)^T \\ \mathbf{M}_y &= (0, 1, 0, 0)^T \\ \mathbf{M}_z &= (0, 0, 1, 0)^T\end{aligned}\tag{2.47}$$

Then, their respective projected images are:

$$\begin{aligned}\mathbf{p}_1 &= \mathbf{P} \cdot \mathbf{M}_x \\ \mathbf{p}_2 &= \mathbf{P} \cdot \mathbf{M}_y \\ \mathbf{p}_3 &= \mathbf{P} \cdot \mathbf{M}_z\end{aligned}\tag{2.48}$$

therefore, as it was commented above, the former 3 columns of $\mathbf{P}$ correspond to the vanishing points on the image plane.

Finally, we can observe that if we take the origin of the world coordinates as $\mathbf{M}_o = (0, 0, 0, 1)^T$ using homogeneous coordinates. Then its projection on the image plane is:

$$\mathbf{p}_4 = \mathbf{P} \cdot \mathbf{M}_o\tag{2.49}$$

and consequently the last column of $\mathbf{P}$ corresponds to the projection of the world origin $(0, 0, 0)^T$ expressed in inhomogeneous coordinates.

2.3.3.- The principal plane and axes planes

We have seen that the columns of $\mathbf{P}$ have a meaning, however, its row vectors can also be interpreted geometrically, as depicted in figure 2.9.

Hence, given the camera matrix as:

$$\mathbf{P} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} = \begin{bmatrix} \mathbf{p}^{1T} \\ \mathbf{p}^{2T} \\ \mathbf{p}^{3T} \end{bmatrix}\tag{2.50}$$

where $\mathbf{p}^{iT}$ is the i^{th} row of the 3×4 camera matrix $\mathbf{P}$. We will see the meaning of each of these vectors $\mathbf{p}^{1T}$, $\mathbf{p}^{2T}$ and $\mathbf{p}^{3T}$.

2.3.3.1.- Axes planes

Assuming that the first row vector of $\mathbf{p}^{1T}$ of the camera matrix $\mathbf{P}$ is a plane, we can observe by means of (1.74) that the set of world points $\mathbf{M}$ belonging to such plane fulfill:

$$\mathbf{p}^{1T} \cdot \mathbf{M} = 0\tag{2.51}$$

Therefore, the projection of $\mathbf{M}$ by means of the camera projection matrix $\mathbf{P}$ is as follows:

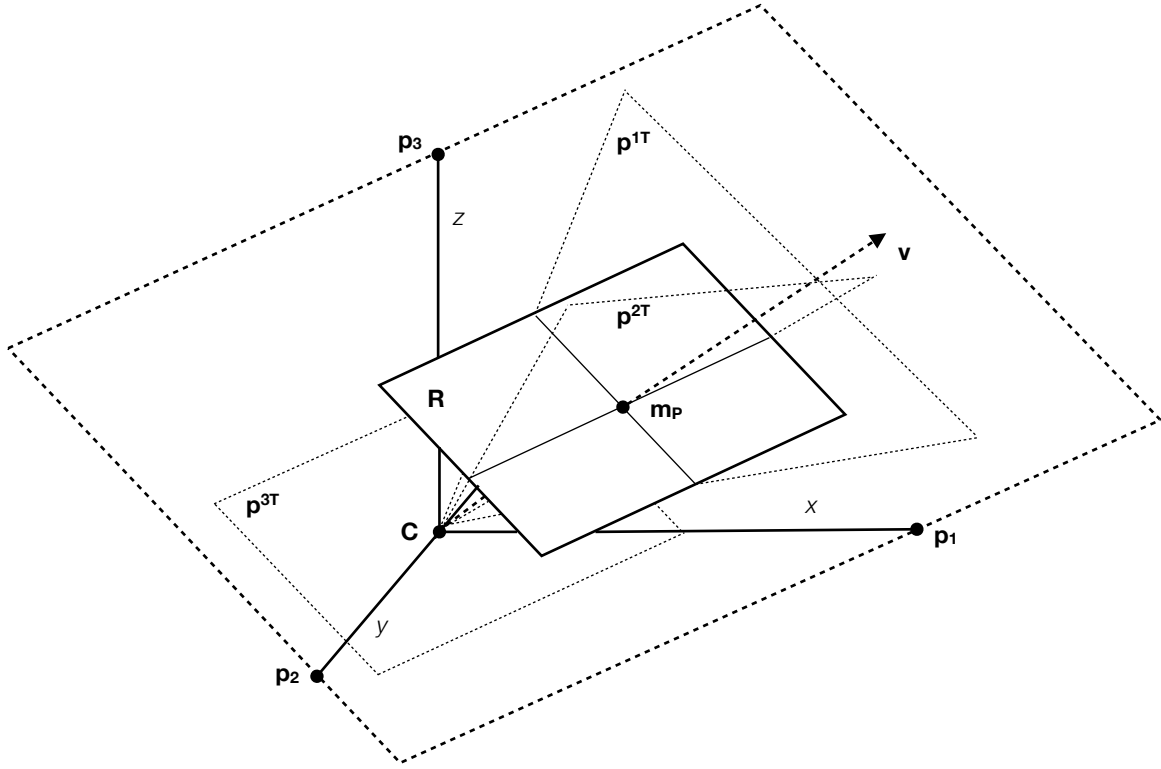


Fig. 2.9. The principal plane and axes planes, where **C** is the camera center, **R** is the retinal plane, x, y, z are the axes of the world coordinate system, **p₁** is the x vanishing point, **p₂** is the y vanishing point, **p₃** is the z vanishing point, **p₁^T** is the vertical image axis plane, **p₂^T** is the horizontal image axis plane, **p₃^T** is the principal plane, **m_p** is the principal point and **v** is the principal axis vector. Notice that the origin of image coordinates is located at image center where in this case is also located the principal point.

$$\mathbf{m} = \mathbf{P} \cdot \mathbf{M} = \begin{bmatrix} \mathbf{p}^{1T} \\ \mathbf{p}^{2T} \\ \mathbf{p}^{3T} \end{bmatrix} \cdot \mathbf{M} = \begin{bmatrix} \mathbf{p}^{1T} \cdot \mathbf{M} \\ \mathbf{p}^{2T} \cdot \mathbf{M} \\ \mathbf{p}^{3T} \cdot \mathbf{M} \end{bmatrix} = \begin{bmatrix} 0 \\ y \\ w \end{bmatrix} \quad (2.52)$$

and by simple inspection of (2.52) we can then observe that the projected points **m** all lay on the line forming the vertical axis of the image plane respect to the image coordinate system.

Hence, the plane **p₁^T** is the vertical axis plane respect to the image coordinate system, as depicted in figure 2.9.

We must observe that by means of (2.33) $\mathbf{P} \cdot \mathbf{C} = \mathbf{0}_3$, then we can also assert that:

$$\mathbf{p}^{1T} \cdot \mathbf{C} = 0 \quad (2.53)$$

which by looking at expression (1.74) it indicates that the camera center **C** lies within the vertical axis plane **p₁^T**.

On the other, assuming that the second row **p₂^T** is another plane, then all the world points **M** laying on such plane must follow:

$$\mathbf{p}^{2T} \cdot \mathbf{M} = 0 \quad (2.54)$$

And then, the projection of world points **M** into the image plane is now:

$$\mathbf{m} = \mathbf{P} \cdot \mathbf{M} = \begin{bmatrix} \mathbf{p}^{1T} \\ \mathbf{p}^{2T} \\ \mathbf{p}^{3T} \end{bmatrix} \cdot \mathbf{M} = \begin{bmatrix} \mathbf{p}^{1T} \cdot \mathbf{M} \\ \mathbf{p}^{2T} \cdot \mathbf{M} \\ \mathbf{p}^{3T} \cdot \mathbf{M} \end{bmatrix} = \begin{bmatrix} x \\ 0 \\ w \end{bmatrix} \quad (2.55)$$

where now, we can observe in (2.55) that all projected points $\mathbf{m}$ lay on the line forming the horizontal axis on the image plane respect to the image coordinate system.

Hence, now the plane $\mathbf{p}^{2T}$ is the horizontal axis plane respect to the image coordinate system, which is also shown in figure 2.9.

By analogy to the previous case, we can also infer that the camera center $\mathbf{C}$ also lies within the horizontal axis plane, because by expression (2.33) we know that $\mathbf{P} \cdot \mathbf{C} = \mathbf{0}_3$, and therefore

$$\mathbf{p}^{2T} \cdot \mathbf{C} = 0 \quad (2.56)$$

2.3.3.2.- The principal plane

Lastly, we can assume that the third row $\mathbf{p}^{3T}$ of the camera projection matrix $\mathbf{P}$ is another plane. As in previous cases, then all the world points $\mathbf{M}$ laying on such plane fulfill:

$$\mathbf{p}^{3T} \cdot \mathbf{M} = 0 \quad (2.57)$$

And then, their respective projections $\mathbf{m}$ on the image plane is:

$$\mathbf{m} = \mathbf{P} \cdot \mathbf{M} = \begin{bmatrix} \mathbf{p}^{1T} \\ \mathbf{p}^{2T} \\ \mathbf{p}^{3T} \end{bmatrix} \cdot \mathbf{M} = \begin{bmatrix} \mathbf{p}^{1T} \cdot \mathbf{M} \\ \mathbf{p}^{2T} \cdot \mathbf{M} \\ \mathbf{p}^{3T} \cdot \mathbf{M} \end{bmatrix} = \begin{bmatrix} x \\ y \\ 0 \end{bmatrix} \quad (2.58)$$

which is the set of points $\mathbf{m}$ that lie on the line at infinity.

Hence, plane $\mathbf{p}^{3T}$ is the plane that contains all the points on the line at infinity. This plane is parallel to the image plane, as it is depicted in figure 2.9.

Finally, we must observe that as in the previous two cases the camera center $\mathbf{C}$ also belongs to the plane $\mathbf{p}^{3T}$. As before, taking expression (2.33) where $\mathbf{P} \cdot \mathbf{C} = \mathbf{0}_3$, we can then assert:

$$\mathbf{p}^{3T} \cdot \mathbf{C} = 0 \quad (2.59)$$

and by means of (1.74) it indicates that the camera center $\mathbf{C}$ lies within the principal plane $\mathbf{p}^{3T}$.

2.3.4.- The principal point

We will see in section 2.3.5 that the principal axis is the line going through the camera center $\mathbf{C}$ that is perpendicular to the principal plane $\mathbf{p}^{3T}$, and consequently to the retinal plane $\mathbf{R}$, as they are both parallel. For convenience, this is also depicted on figure 2.9.

We know from expression (1.73) that the director vector of a plane $\Pi = (A, B, C, D)^T$ is $\mathbf{n}_\Pi = (A, B, C)^T$, and we know that this director vector is perpendicular to the plane surface. Then, by means of the director vector we can assert that the set of points perpendicular to a given plane Π can be expressed as:

$$\mathbf{M} = (A, B, C, \lambda)^T, \quad \lambda \in \mathbb{R} \quad (2.60)$$

where λ can be any arbitrary value, because homogeneous coordinates are defined only up to scale. Then, a point at *infinity* in the direction of the director vector $\mathbf{n}_n$ would be:

$$\mathbf{M}_\infty = (A, B, C, 0)^T \quad (2.61)$$

Hence, if we assumed that the plane in this case is the principal plane $\Pi = \mathbf{p}^{3T}$, then the point at infinity perpendicular to such plane would be:

$$\mathbf{M}_\infty = (p_{31}, p_{32}, p_{33}, 0)^T = (\mathbf{m}^{3T}, 0)^T \quad (2.62)$$

where $\mathbf{m}^{3T}$ is the 3rd row of $\mathbf{M}_P$, which we saw in (2.32) that is the 3x3 top-left minor of $\mathbf{P}$.

And the projection of the point in (2.62) by means of the camera projection matrix $\mathbf{P}$ would result in the following expression:

$$\mathbf{m}_P = \mathbf{P} \cdot \mathbf{M}_\infty = [\mathbf{M}_P | \mathbf{p}_4] \cdot \begin{bmatrix} \mathbf{m}^{3T} \\ 0 \end{bmatrix} = \mathbf{M}_P \cdot \mathbf{m}^{3T} + 0 \cdot \mathbf{p}_4 = \mathbf{M}_P \cdot \mathbf{m}^{3T} \quad (2.63)$$

where $\mathbf{m}_P = \mathbf{M}_P \cdot \mathbf{m}^{3T}$ is the principal point $(p_x, p_y)^T$ that was seen in (2.15).

2.3.5.- The principal axis vector and camera sign

The principal axis vector is the vector perpendicular to the principal plane that defines the direction the camera is looking at, as shown in figure 2.9. In section 2.3.4 we saw that this vector can be defined as $\mathbf{m}^{3T}$, which contains the first 3 elements of $\mathbf{p}^{3T}$, which is the 3rd row of $\mathbf{P}$.

We also saw in section 2.3.3.2 that $\mathbf{p}^{3T}$ was the principal plane, hence the first 3 elements $\mathbf{m}^{3T}$ of such plane is its director vector, or in this case the *principal axis vector*. However, because we've been using homogeneous coordinates, the principal axis vector $\mathbf{m}^{3T}$ is only defined up to scale, which includes changes of sign. Therefore, we don't know whether $\mathbf{m}^{3T}$ points towards the front of the camera (in the *positive direction*) or not.

It is obvious then that we need to define the sign of $\mathbf{m}^{3T}$ so that it points towards the front of the camera, as in the real world, 3D points will only be located in front of the camera.

By using the camera coordinate frame that we saw in (2.20), we can then state that a projected point in the image plane can be expressed by a camera without extrinsic parameters as shown next:

$$\mathbf{m} = \mathbf{P}_{\text{cam}} \cdot \mathbf{M}_{\text{cam}} = \mathbf{K} \cdot [\mathbf{I}_{3 \times 3} | \mathbf{0}_3] \cdot \mathbf{M}_{\text{cam}} \quad (2.64)$$

where $\mathbf{M}_{\text{cam}}$ is a point in the camera coordinate frame and $\mathbf{P}_{\text{cam}}$ is its corresponding projection camera without extrinsic parameters. Notice that because $\mathbf{K}$ is a triangular matrix, its determinant is the product of the elements of the diagonal, which are all positive. Then:

$$\det(\mathbf{K}) = \alpha_x \cdot \alpha_y = \lambda > 0 \quad (2.65)$$

Then, we can observe that the determinant of the minor $\mathbf{M}_P = \mathbf{K}$ is also positive, and consequently the vector:

$$\mathbf{v} = \det(\mathbf{M}_P) \cdot \mathbf{m}^3 = \lambda \cdot (0, 0, 1)^T, \quad \lambda > 0 \quad (2.66)$$

where $\mathbf{M_P}$ is the 3x3 top-left minor of $\mathbf{P_{cam}}$, $\mathbf{m^3} = (0,0,1)^T$ is the third row of $\mathbf{M_P} = \mathbf{K}$, and $\mathbf{v}$ is a vector that points towards the front of the camera in the direction of the principal axis defined up to a **positive** scale value λ .

By scaling $\mathbf{P_{cam}}$ by any arbitrary scalar value (regardless of its sign), the result would be the same as $\mathbf{P'_{cam}} = k \cdot \mathbf{P_{cam}}$, and then $\det(\mathbf{M'_P}) = k^3 \cdot \det(\mathbf{M_P})$. Consequently, (2.66) becomes $\mathbf{v} = k^3 \cdot \det(\mathbf{M_P}) \cdot k \cdot \mathbf{m^3} = k^4 \cdot \det(\mathbf{M_P}) \cdot \mathbf{m^3}$, which does not change its sign even for negative k values.

By expressing (2.64) in the world coordinates:

$$\mathbf{m} = k \cdot \mathbf{P} \cdot \mathbf{M} = k \cdot \mathbf{K} \left[\mathbf{R} \mid -\mathbf{R} \cdot \tilde{\mathbf{C}} \right] \cdot \mathbf{M_{cam}} \quad (2.67)$$

where k is again any arbitrary scalar value that can have negative values.

Then, the 3x3 left minor of the scaled camera projection matrix $\mathbf{P'} = k \cdot \mathbf{P} = k \cdot \mathbf{K} \mathbf{R} \cdot [\mathbf{I}_{3 \times 3} \mid \mathbf{t}]$ is:

$$\mathbf{M'_P} = k \cdot \mathbf{K} \cdot \mathbf{R} = k \cdot \mathbf{M_P} \quad (2.68)$$

where $\mathbf{K}$ is the calibration matrix and $\mathbf{R}$ is the rotation matrix, and k is an arbitrary scalar value (which can also be negative). And consequently the third row of the scaled minor $\mathbf{M'_P}$ is:

$$\mathbf{m'^3} = k \cdot \mathbf{m^3} \quad (2.69)$$

We saw in (1.130) that the determinant of any rotation matrix is $\det(\mathbf{R}) = 1$, therefore the determinant of the scaled minor $\mathbf{M'_P}$ is:

$$\det(\mathbf{M'_P}) = k^3 \cdot \det(\mathbf{K}) \cdot \det(\mathbf{R}) = k^3 \cdot \det(\mathbf{K}) = k^3 \cdot \det(\mathbf{M_P}) \quad (2.70)$$

and consequently expression (2.66) on any scaled camera matrix $\mathbf{P'}$ regardless of the sign of the scale value k , by means of (2.69) and (2.70) now becomes:

$$\mathbf{v'} = \det(\mathbf{M'_P}) \cdot \mathbf{m'^3} = k^3 \cdot \det(\mathbf{M_P}) \cdot k \cdot \mathbf{m^3} = k^4 \cdot \det(\mathbf{M_P}) \cdot \mathbf{m^3} = k^4 \cdot \mathbf{v} \quad (2.71)$$

Therefore vector $\mathbf{v'}$ does not change its sign for any scale value k .

We can then assert that for any vector:

$$\mathbf{v} = \det(\mathbf{M_P}) \cdot \mathbf{m^3} \quad (2.72)$$

where $\mathbf{M_P}$ is the 3x3 left minor of the camera matrix $\mathbf{P}$ and $\mathbf{m^3}$ is the third row of $\mathbf{M_P}$. Then $\mathbf{v}$ will be a vector in the direction of the principal axis *pointing **always** towards the front of the camera*.

2.3.5.1.- Camera sign normalization

We have seen in previous section that because the camera projection matrix is defined using homogeneous coordinates, then it is only defined up to scale. Consequently both $\mathbf{P}$ or $-\mathbf{P}$ are equivalent, however one will point towards the front whereas the other will point backwards, and we know that 3D world points are always located in front of the camera. Therefore we need to determine which sign is the proper one.

As we previously saw in (2.72), the principal point direction towards the front of the camera is defined by the sign of $\det(\mathbf{M_P})$, hence the reader will see that is a good practice to use this sign to correctly set the camera matrix $\mathbf{P}$ sign so that it points towards the front as expected.

2.3.6.- Finding the camera orientation and intrinsic parameters

As we saw in (2.32), any finite camera can be decomposed as:

$$\mathbf{P} = [\mathbf{M}_p | -\mathbf{M}_p \cdot \tilde{\mathbf{C}}] = \mathbf{K} [\mathbf{R} | -\mathbf{R} \tilde{\mathbf{C}}] \quad (2.73)$$

Therefore, we would like to find a way of retrieving both the calibration matrix $\mathbf{K}$ and the rotation matrix $\mathbf{R}$ from the 3x3 left minor MP. It can be seen that by means of the RQ decomposition both of this matrices can be obtained. RQ decomposition decomposes any matrix into the product of an upper-triangular matrix (i.e. like $\mathbf{K}$) and an orthogonal matrix (i.e. like $\mathbf{R}$).

$$\mathbf{A} = \mathbf{R} \cdot \mathbf{Q} \quad (2.74)$$

where A is any arbitrary matrix, R is an upper triangular matrix, Q is orthogonal (i.e. $\mathbf{Q} \cdot \mathbf{Q}^T = \text{diag}(\lambda_1^2, \dots, \lambda_n^2) \cdot \mathbf{I}_{n \times n}$)

Notice however that RQ decomposition is not unique, and matrices can differ by scale factors. Therefore (2.74) can also be expressed as:

$$\mathbf{A} = \mathbf{R} \cdot \mathbf{Q} = \mathbf{R}' \cdot \text{diag}(\lambda_1, \dots, \lambda_n)^{-1} \cdot \text{diag}(\lambda_1, \dots, \lambda_n) \cdot \mathbf{Q}' \quad (2.75)$$

However, in our particular case we are interested in a calibration matrix $\mathbf{K}$ that has positive elements on its diagonal and its bottom right element equal to 1, as we saw in (2.17), besides $\mathbf{R}$ not only is orthogonal, but also orthonormal, as we saw in (1.129) where $\mathbf{R} \cdot \mathbf{R}^T = \mathbf{I}_{3 \times 3}$

Applying these constraints, we can then obtain a unique RQ decomposition up to scale, but because camera matrices $\mathbf{P}$ are defined up to scale as well, this will be enough to retrieve both $\mathbf{K}$ and $\mathbf{R}$.

Indeed, taking into account (2.75) and given the 3x3 minor $\mathbf{M}_p$ of camera matrix $\mathbf{P}$, we obtain its RQ decomposition as:

$$\begin{aligned} \mathbf{M}_p &= \mathbf{K}' \cdot \mathbf{R}' = \\ &= (\mathbf{K}' \cdot \text{diag}(\text{sign}(k'_{11}), \text{sign}(k'_{22}), k'_{33})) (\text{diag}(\text{sign}(k'_{11}), \text{sign}(k'_{22}), 1/k'_{33}) \cdot \mathbf{R}') \end{aligned} \quad (2.76)$$

where k'_{ij} is the element of $\mathbf{K}'$ located at i^{th} row and j^{th} column, $\text{sign}(\cdot)$ is 1 for positive values and -1 for negative ones, and $\text{diag}(\text{sign}(k'_{11}), \text{sign}(k'_{22}), 1/k'_{33})$ is a diagonal matrix that is the inverse of $\text{diag}(\text{sign}(k'_{11}), \text{sign}(k'_{22}), k'_{33})$. Therefore by means of (2.76) we can obtain the true calibration matrix $\mathbf{K}$ as:

$$\mathbf{K} = \mathbf{K}' \cdot \text{diag}(\text{sign}(k'_{11}), \text{sign}(k'_{22}), 1/k'_{33}) \quad (2.77)$$

and consequently the true rotation matrix $\mathbf{R}$ as:

$$\mathbf{R} = \text{diag}(\text{sign}(k'_{11}), \text{sign}(k'_{22}), k'_{33}) \cdot \mathbf{R}' \quad (2.78)$$

2.4.- Depth and cheirality

2.4.1.- Computing depth of world points respect to camera

Given a camera projection matrix $\mathbf{P}$ and a 3D world point, we would like to know the depth of such point respect to the principal plane of the camera (see figure 2.9). Indeed, by using the principal plane of the camera and its director vector $\mathbf{v}$, which is the principal axis vector, and also taking into account the angular separation between the vector joining $\mathbf{M}$ and $\mathbf{C}$ and $\mathbf{v}$, we can infer that the depth of any point $\mathbf{M}$ respect to the principal plane of camera $\mathbf{P}$ is:

$$\text{depth}(\mathbf{M}, \mathbf{P}) = \|\tilde{\mathbf{M}} - \tilde{\mathbf{C}}\| \cdot \cos(\alpha) \quad (2.79)$$

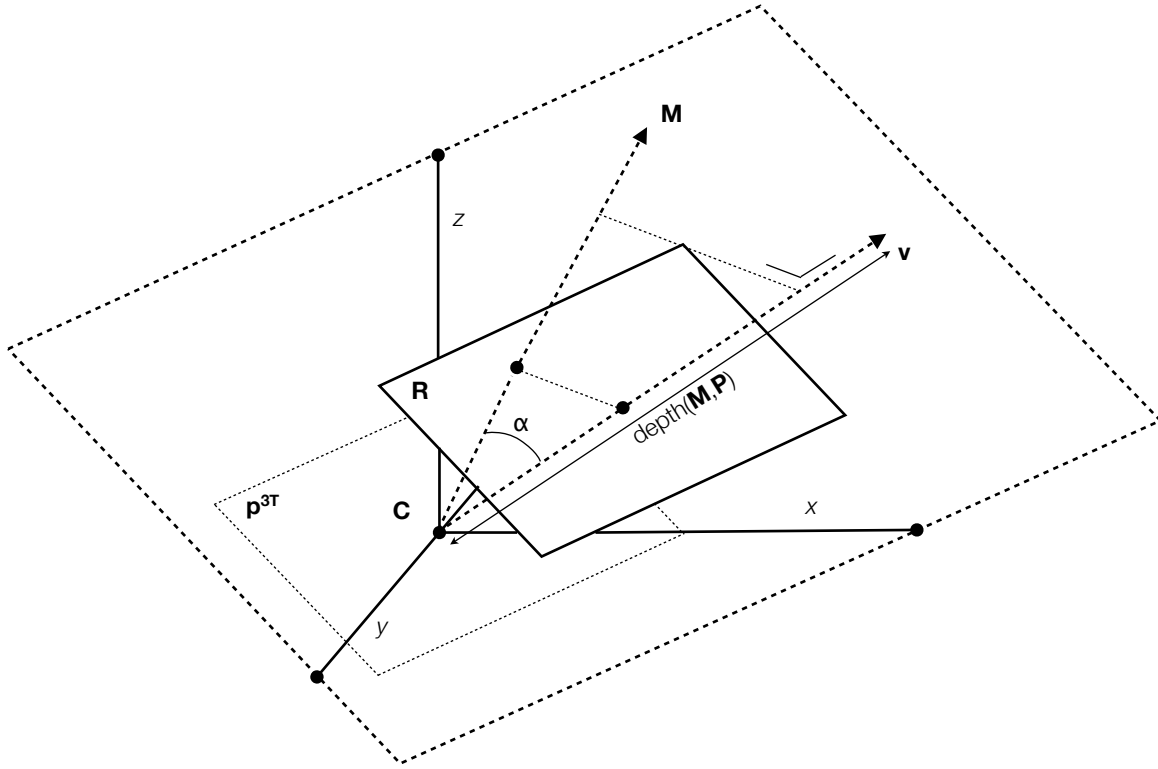


Fig. 2.10. Depth of a world point $\mathbf{M}$ respect to the principal plane $\mathbf{p}^{3T}$ of a camera $\mathbf{P}$. Where x, y, z are the axes of the world coordinate system, $\mathbf{C}$ is the camera center, $\mathbf{R}$ is the retinal plane, $\mathbf{v}$ is the principal axis vector, α is the angular separation between vector $\mathbf{M} - \mathbf{C}$ and $\mathbf{v}$, and $\text{depth}(\mathbf{M}, \mathbf{P})$ is the depth of point $\mathbf{M}$ respect to camera $\mathbf{P}$.

where $\mathbf{M}$ and $\mathbf{C}$ are the 3D world point and the camera center expressed using inhomogeneous coordinates, and α is the separation angle between point $\mathbf{M}$ and the principal axis vector $\mathbf{v}$, which is the director vector of the principal plane, as shown in figure 2.10.

By means of expression (1.143), we know then that the dot product between point world point $\mathbf{M}$ respect to camera center $\mathbf{C}$ (using inhomogeneous coordinates) and the principal axis vector $\mathbf{v}$ is:

$$\begin{aligned}
 (\tilde{\mathbf{M}} - \tilde{\mathbf{C}})^T \cdot \mathbf{v} &= \|\tilde{\mathbf{M}} - \tilde{\mathbf{C}}\| \cdot \|\mathbf{v}\| \cdot \cos(\alpha) \Rightarrow \\
 \cos(\alpha) &= \frac{(\tilde{\mathbf{M}} - \tilde{\mathbf{C}})^T \cdot \mathbf{v}}{\|\tilde{\mathbf{M}} - \tilde{\mathbf{C}}\| \cdot \|\mathbf{v}\|}
 \end{aligned} \tag{2.80}$$

and consequently, (2.79) becomes:

$$\text{depth}(\mathbf{M}, \mathbf{P}) = \|\tilde{\mathbf{M}} - \tilde{\mathbf{C}}\| \cdot \frac{(\tilde{\mathbf{M}} - \tilde{\mathbf{C}})^T \cdot \mathbf{v}}{\|\tilde{\mathbf{M}} - \tilde{\mathbf{C}}\| \cdot \|\mathbf{v}\|} = \frac{(\tilde{\mathbf{M}} - \tilde{\mathbf{C}})^T \cdot \mathbf{v}}{\|\mathbf{v}\|} \tag{2.81}$$

And finally, using (2.72) for the expression of the principal axis vector $\mathbf{v}$ then (2.81) is expressed as:

$$\begin{aligned}
depth(\mathbf{M}, \mathbf{P}) &= \frac{(\tilde{\mathbf{M}} - \tilde{\mathbf{C}})^T \cdot \mathbf{v}}{\|\mathbf{v}\|} = \frac{\det(\mathbf{M}_P) \cdot (\tilde{\mathbf{M}} - \tilde{\mathbf{C}})^T \cdot \mathbf{m}^3}{\|\det(\mathbf{M}_P)\| \cdot \|\mathbf{m}^3\|} \Rightarrow \\
depth(\mathbf{M}, \mathbf{P}) &= sign(\det(\mathbf{M}_P)) \frac{(\tilde{\mathbf{M}} - \tilde{\mathbf{C}})^T \cdot \mathbf{m}^3}{\|\mathbf{m}^3\|}
\end{aligned} \tag{2.82}$$

Where the absolute value of (2.82) will indicate the depth in pixels of a world point $\mathbf{M}$ respect to the principal plane of camera $\mathbf{P}$, and the sign of (2.82) will indicate whether such point $\mathbf{M}$ is located in front (if sign is positive) or behind of the camera (if sign is negative). Besides, if depth is infinity, it will indicate that point $\mathbf{M}$ is located on the plane at infinity Π_∞ , and if depth is zero it will mean that the point $\mathbf{M}$ is located on the principal plane $\mathbf{p}^{3T}$. Finally notice that if both the camera center and the point are at infinity, then depth is not defined.

2.4.2.- Computing cheirality of world points respect to camera

We know that projected points $\mathbf{m}$ on the retinal plane $\mathbf{R}$ correspond to 3D world points $\mathbf{M}$ that must be located in front of the camera. And we saw in (2.82) that the sign of the depth of such points respect to the camera $\mathbf{P}$ indicates whether such points are located in front or behind of the camera.

However, when we are only interested in knowing whether such points are in front or behind of the camera and not on their actual depth, we can simplify (2.82) to obtain *cheirality* of points, which is the sign of depth. Therefore,

$$\begin{aligned}
sign(depth(\mathbf{M}, \mathbf{P})) &= sign(\det(\mathbf{M}_P)) \cdot sign\left(\frac{(\tilde{\mathbf{M}} - \tilde{\mathbf{C}})^T \cdot \mathbf{m}^3}{\|\mathbf{m}^3\|}\right) = \\
&= sign(\det(\mathbf{M}_P)) \cdot sign((\tilde{\mathbf{M}} - \tilde{\mathbf{C}})^T \cdot \mathbf{m}^3)
\end{aligned} \tag{2.83}$$

From (2.33) we know that the projection of the camera center $\mathbf{C} = (X_c, Y_c, Z_c, W_c)^T$ fulfills $\mathbf{P} \cdot \mathbf{C} = \mathbf{0}_3$. Hence, representing the camera matrix in terms of its rows as in (2.50) we obtain:

$$\mathbf{P} \cdot \mathbf{C} = \begin{bmatrix} \mathbf{p}^{1T} \\ \mathbf{p}^{2T} \\ \mathbf{p}^{3T} \end{bmatrix} \cdot \mathbf{C} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{2.84}$$

therefore, by taking into account only the last row, we can assert:

$$\mathbf{p}^{3T} \cdot \mathbf{C} = 0 \tag{2.85}$$

where $\mathbf{p}^{3T}$ is the third row of the camera matrix and can be expressed in terms of the last row of the 3x3 left minor $\mathbf{M}_P$ of $\mathbf{P}$ as:

$$\mathbf{p}^{3T} = (\mathbf{m}^{3T}, p_{34})^T \tag{2.86}$$

Therefore, (2.85) can be rewritten as:

$$\mathbf{p}^{3T} \cdot \mathbf{C} = [\mathbf{m}^{3T}, p_{34}]^T \cdot \begin{bmatrix} X_C \\ Y_C \\ Z_C \\ W_C \end{bmatrix} = \mathbf{m}^{3T} \cdot \begin{bmatrix} X_C \\ Y_C \\ Z_C \end{bmatrix} + p_{34} \cdot W_C = 0 \quad (2.87)$$

And by dividing (2.87) by the last homogeneous component of the camera center W_C then we obtain:

$$\frac{\mathbf{p}^{3T} \cdot \mathbf{C}}{W_C} = \mathbf{m}^{3T} \cdot \begin{bmatrix} X_C / W_C \\ Y_C / W_C \\ Z_C / W_C \end{bmatrix} + p_{34} = \mathbf{m}^{3T} \cdot \tilde{\mathbf{C}} + p_{34} = 0 \quad (2.88)$$

Therefore,

$$p_{34} = -\mathbf{m}^{3T} \cdot \tilde{\mathbf{C}} \quad (2.89)$$

where the camera center is now expressed using inhomogeneous coordinates.

By analogy to the projection of the camera center, then the projection of a world point $\mathbf{M} = (X, Y, Z, W)^T$ can be obtained as:

$$\mathbf{m} = \mathbf{P} \cdot \mathbf{M} = \begin{bmatrix} \mathbf{p}^{1T} \\ \mathbf{p}^{2T} \\ \mathbf{p}^{3T} \end{bmatrix} \cdot \mathbf{M} = \begin{bmatrix} x \\ y \\ w \end{bmatrix} \quad (2.90)$$

Hence, as in (2.85) we can write the last homogeneous component of the projected point as

$$w = \mathbf{p}^{3T} \cdot \mathbf{M} \quad (2.91)$$

again we can use (2.86), and hence (2.91) becomes:

$$w = \mathbf{p}^{3T} \cdot \mathbf{M} = [\mathbf{m}^{3T}, p_{34}]^T \cdot \begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix} = \mathbf{m}^{3T} \cdot \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + p_{34} \cdot W \quad (2.92)$$

Again, by dividing by the last homogeneous component W of the world point $\mathbf{M}$, we obtain:

$$\frac{w}{W} = \mathbf{m}^{3T} \cdot \begin{bmatrix} X / W \\ Y / W \\ Z / W \end{bmatrix} + p_{34} = \mathbf{m}^{3T} \cdot \tilde{\mathbf{M}} + p_{34} \quad (2.93)$$

And finally, by means of (2.89), then (2.93) becomes:

$$\frac{w}{W} = \mathbf{m}^{3T} \cdot \tilde{\mathbf{M}} - \mathbf{m}^{3T} \cdot \tilde{\mathbf{C}} = \mathbf{m}^{3T} \cdot (\tilde{\mathbf{M}} - \tilde{\mathbf{C}}) = (\tilde{\mathbf{M}} - \tilde{\mathbf{C}})^T \cdot \mathbf{m}^3 \quad (2.94)$$

Consequently (2.83) then is rewritten as,

$$\begin{aligned} \text{sign}(\text{depth}(\mathbf{M}, \mathbf{P})) &= \text{sign}(\det(\mathbf{M}_p)) \cdot \text{sign}\left((\tilde{\mathbf{M}} - \tilde{\mathbf{C}})^T \cdot \mathbf{m}^3\right) \\ &= \text{sign}(\det(\mathbf{M}_p)) \cdot \text{sign}\left(\frac{w}{W}\right) = \text{sign}(\det(\mathbf{M}_p)) \cdot \text{sign}(w) \cdot \text{sign}(W) \end{aligned} \quad (2.95)$$

Assuming that the **camera matrix has been normalized** as we explained in section 2.3.5.1, then $\det(\mathbf{M}_p) > 0$, and consequently the cheirality expression in (2.95) is further simplified to finally obtain:

$$\text{sign}(\text{depth}(\mathbf{M}, \mathbf{P})) = \text{sign}(w) \cdot \text{sign}(W) \quad (2.96)$$

where $\mathbf{M}$ is any 3D world point, $\mathbf{P}$ is the camera projection matrix, w is the last homogeneous component of the projected point $\mathbf{m}$, and W is the last homogeneous component of the 3D world point $\mathbf{M}$.

Hence, as we explained before, when cheirality expression in (2.96) is positive, then the 3D world point $\mathbf{M}$ will be in front of camera $\mathbf{P}$, otherwise it will be located behind the camera.

2.5.- Camera estimation

In previous sections in this chapter we defined the pinhole camera model, indicated the intrinsic and extrinsic parameters that form such model, and provided ways to retrieve such parameters in case that only the camera matrix $\mathbf{P}$ is known. We'd like now to find techniques to obtain such camera matrix from data collected on a general world scene.

2.5.1.- The DLT algorithm for camera estimation

The DLT algorithm can be used to retrieve the camera matrix $\mathbf{P}$ from correspondences between world points $\mathbf{M}$ and their projected image points $\mathbf{m}$. Therefore, if by some reason we know the 3D world locations of some points depicted on a given image, DLT will allow us to obtain the camera matrix.

This fact is commonly used to calibrate cameras in the image processing field by means of calibration grids like the one depicted in figure 2.11. In such grid, we can assume that the world origin of coordinates is located in the intersection of all three planes, and therefore x , y , z axis are also defined.

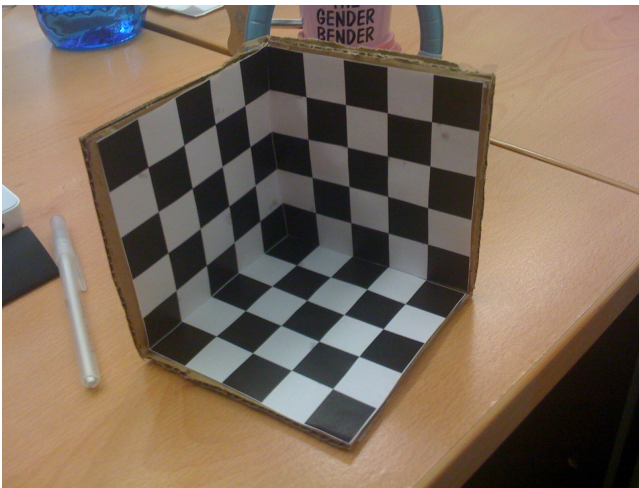


Fig. 2.11. Depiction of a calibration grid

Besides, we know that in that specific grid each square has a size of 1 cm, consequently, we can easily know the world coordinates of the points located at each square intersection. And as we will see next, by manually selecting the coordinates $\mathbf{m}_i$ of such intersections on the obtained image, and associating their corresponding 3D world coordinate coordinates $\mathbf{M}_i$, we can retrieve the camera matrix $\mathbf{P}$.

Hence, by defining the camera matrix $\mathbf{P}$ in terms of its rows $\mathbf{p}^iT$ as in (2.50), defining the i^{th} image point as:

$$\mathbf{m}_i = (x_i, y_i, w_i)^T \quad (2.97)$$

which can also be expressed in terms of inhomogeneous

coordinates as:

$$\tilde{\mathbf{m}}_i = (\tilde{x}_i, \tilde{y}_i)^T = \left(\frac{x_i}{w_i}, \frac{y_i}{w_i} \right)^T \quad (2.98)$$

and defining its corresponding i^{th} 3D world point using homogeneous coordinates as:

$$\mathbf{M}_i = (X_i, Y_i, Z_i, W_i)^T \quad (2.99)$$

Consequently, the image point $\mathbf{m}_i$ is projected on the retinal plane using a given camera $\mathbf{P}$ by means of expression (2.3), which can be written as follows:

$$\mathbf{m}_i = \mathbf{P} \cdot \mathbf{M}_i = \begin{bmatrix} \mathbf{p}^{1T} \\ \mathbf{p}^{2T} \\ \mathbf{p}^{3T} \end{bmatrix} \cdot \mathbf{M}_i = \begin{bmatrix} x_i \\ y_i \\ w_i \end{bmatrix} \Rightarrow \left. \begin{array}{l} x_i = \mathbf{p}^{1T} \cdot \mathbf{M}_i \\ y_i = \mathbf{p}^{2T} \cdot \mathbf{M}_i \\ w_i = \mathbf{p}^{3T} \cdot \mathbf{M}_i \end{array} \right\} \quad (2.100)$$

And by expressing (2.100) using inhomogeneous coordinates, we obtain:

$$\left. \begin{array}{l} \tilde{x}_i = \frac{x_i}{w_i} = \frac{\mathbf{p}^{1T} \cdot \mathbf{M}_i}{\mathbf{p}^{3T} \cdot \mathbf{M}_i} \\ \tilde{y}_i = \frac{y_i}{w_i} = \frac{\mathbf{p}^{2T} \cdot \mathbf{M}_i}{\mathbf{p}^{3T} \cdot \mathbf{M}_i} \end{array} \right\} \Rightarrow \left. \begin{array}{l} \tilde{x}_i \cdot \mathbf{p}^{3T} \cdot \mathbf{M}_i = \mathbf{p}^{1T} \cdot \mathbf{M}_i \\ \tilde{y}_i \cdot \mathbf{p}^{3T} \cdot \mathbf{M}_i = \mathbf{p}^{2T} \cdot \mathbf{M}_i \end{array} \right\} \quad (2.101)$$

And because the dot product can be commuted, then (2.101) can also be written as:

$$\left. \begin{array}{l} \tilde{x}_i \cdot \mathbf{M}_i^T \cdot \mathbf{p}^3 = \mathbf{M}_i^T \cdot \mathbf{p}^1 \\ \tilde{y}_i \cdot \mathbf{M}_i^T \cdot \mathbf{p}^3 = \mathbf{M}_i^T \cdot \mathbf{p}^2 \end{array} \right\} \quad (2.102)$$

Finally, in order to have all matrix rows in order, we could pass all the terms to the same side of the equations in (2.102) so that we obtain:

$$\left. \begin{array}{l} \mathbf{M}_i^T \cdot \mathbf{p}^1 + \mathbf{0}_4^T \cdot \mathbf{p}^2 - \tilde{x}_i \cdot \mathbf{M}_i^T \cdot \mathbf{p}^3 = 0 \\ \mathbf{0}_4^T \cdot \mathbf{p}^1 + \mathbf{M}_i^T \cdot \mathbf{p}^2 - \tilde{y}_i \cdot \mathbf{M}_i^T \cdot \mathbf{p}^3 = 0 \end{array} \right\} \quad (2.103)$$

where $\mathbf{0}_4^T$ is $(0,0,0,0)^T$.

Then, reordering matrix $\mathbf{P}$ as the following 12×1 vector $\mathbf{p}$:

$$\mathbf{p} = \begin{bmatrix} \mathbf{p}^{1T} & \mathbf{p}^{2T} & \mathbf{p}^{3T} \end{bmatrix}^T = \begin{bmatrix} \mathbf{p}^1 \\ \mathbf{p}^2 \\ \mathbf{p}^3 \end{bmatrix} = (p_{11}, p_{12}, p_{13}, p_{14}, p_{21}, p_{22}, p_{23}, p_{24}, p_{31}, p_{32}, p_{33}, p_{34})^T$$

(2.104)

where $\mathbf{p}^i$ is the i^{th} row of $\mathbf{P}$, and p_{ij} is the element of $\mathbf{P}$ located at i^{th} row and j^{th} column.

We can then express (2.103) in matrix form as:

$$\mathbf{A}_i \cdot \mathbf{p} = \mathbf{0}_2 \Rightarrow \begin{bmatrix} \mathbf{M}_i^T & \mathbf{0}_4^T & -\tilde{x}_i \cdot \mathbf{M}_i^T \\ \mathbf{0}_4^T & \mathbf{M}_i^T & -\tilde{y}_i \cdot \mathbf{M}_i^T \end{bmatrix} \begin{bmatrix} \mathbf{p}^1 \\ \mathbf{p}^2 \\ \mathbf{p}^3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad (2.105)$$

Therefore, for each image and world point correspondence we obtain 2 equations which result in the 2×12 $\mathbf{A}_i$ matrix shown in (2.105).

Because we seek for a not null solution to (2.105) defined up to scale that corresponds to matrix $\mathbf{P}$, then, we will need at least 11 equations to have a matrix $\mathbf{A}$ of rank 11 that solves the 12 unknowns from $\mathbf{p}$. Consequently, matrix $\mathbf{A}$ can be obtained by stacking matrices $\mathbf{A}_i$ defined by each image/world point correspondence,

$$\mathbf{A} \cdot \mathbf{p} = \mathbf{0}_N \Rightarrow \begin{bmatrix} \mathbf{A}_1^T & \cdots & \mathbf{A}_i^T & \cdots & \mathbf{A}_N^T \end{bmatrix}^T \cdot \mathbf{p} = \mathbf{0}_N \Rightarrow \begin{bmatrix} \mathbf{M}_1^T & \mathbf{0}_4^T & -\tilde{x}_1 \cdot \mathbf{M}_1^T \\ \mathbf{0}_4^T & \mathbf{M}_1^T & -\tilde{y}_1 \cdot \mathbf{M}_1^T \\ \vdots & \vdots & \vdots \\ \mathbf{M}_i^T & \mathbf{0}_4^T & -\tilde{x}_i \cdot \mathbf{M}_i^T \\ \mathbf{0}_4^T & \mathbf{M}_i^T & -\tilde{y}_i \cdot \mathbf{M}_i^T \\ \vdots & \vdots & \vdots \\ \mathbf{M}_M^T & \mathbf{0}_4^T & -\tilde{x}_M \cdot \mathbf{M}_M^T \\ \mathbf{0}_4^T & \mathbf{M}_M^T & -\tilde{y}_M \cdot \mathbf{M}_M^T \end{bmatrix} \begin{bmatrix} \mathbf{p}^1 \\ \mathbf{p}^2 \\ \mathbf{p}^3 \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix} \quad (2.106)$$

where $\mathbf{A}$ is and $N \times 12$ matrix, $\mathbf{p}$ is the camera vector, $\mathbf{M}_i$ is the i^{th} world point, M defines the number of point correspondences and $N = 2M$ the number of equations.

Then, assuming that each correspondence defines two independent rows ($N = 2M$), and because we need 11 independent equations to solve (2.106), then we obtain:

$$N \geq 11 \Rightarrow 2M \geq 11 \Rightarrow M \geq 5.5 \quad (2.107)$$

which means that at least 5.5 point correspondences are needed, where the half point accounts for only one of the two equations in (2.103).

As in section 2.3.1.1, the non trivial solution to the homogeneous system of equations $\mathbf{A} \cdot \mathbf{p} = \mathbf{0}_N$ can be obtained by using SVD of the $N \times 12$ matrix $\mathbf{A} = \mathbf{U} \cdot \mathbf{S} \cdot \mathbf{V}^T$ and by choosing the column of the right orthonormal 12×12 matrix $\mathbf{V}$ that corresponds to the null singular value. As previously mentioned, notice that SVD is not unique, however singular values are usually given as a positive ordered set of values:

$$s_i \geq s_j \geq 0, \quad i < j \quad (2.108)$$

2.5.1.1.- Least Mean Squares Error (LMSE) solution to the camera estimation

We've seen that by using 11 equations we can solve (2.106) by means of SVD. However, what happens when we have more equations because more than 5.5 point correspondences are available? We will prove that in the general case where $\mathbf{A}$ has rank greater than 11 (i.e. equations are linearly independent), then SVD can obtain a LMSE solution to (2.106).

Therefore, decomposing the Nx12 matrix $\mathbf{A}$ by means of SVD as:

$$\mathbf{A} = \mathbf{U} \cdot \mathbf{S} \cdot \mathbf{V}^T \quad (2.109)$$

where $\mathbf{U}$ is an orthonormal Nx12 matrix, $\mathbf{S}$ is a 12x12 diagonal matrix $\mathbf{S} = \text{diag}(s_1, s_2, \dots, s_{12})$ and $\mathbf{V}$ is an orthonormal 12x12 matrix.

By taking into account the i^{th} columns of $\mathbf{U}$ and $\mathbf{V}$ as $\mathbf{u}_i$ and $\mathbf{v}_i$ respectively,

$$\begin{aligned} \mathbf{U} &= \begin{bmatrix} \mathbf{u}_1 & \cdots & \mathbf{u}_{12} \end{bmatrix} \\ \mathbf{V} &= \begin{bmatrix} \mathbf{v}_1 & \cdots & \mathbf{v}_{12} \end{bmatrix} \end{aligned} \quad (2.110)$$

Then expression (2.109) can be rewritten as:

$$\mathbf{A} = \mathbf{U} \cdot \mathbf{S} \cdot \mathbf{V}^T = \begin{bmatrix} \mathbf{u}_1 & \cdots & \mathbf{u}_{12} \end{bmatrix} \cdot \begin{bmatrix} s_1 & & \\ & \ddots & \\ & & s_{12} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{v}_1^T \\ \vdots \\ \mathbf{v}_{12}^T \end{bmatrix} \quad (2.111)$$

where $s_1, \dots, s_{12}$ are the singular values of $\mathbf{A}$ that we assume that follow (2.108), or in Layman terms that they are ordered from the greatest to the smallest one. Therefore, because $\mathbf{S}$ is diagonal and s_i are scalar values, (2.111) becomes,

$$\mathbf{A} = \mathbf{U} \cdot \mathbf{S} \cdot \mathbf{V}^T = \begin{bmatrix} s_1 \cdot \mathbf{u}_1 & \cdots & s_{12} \cdot \mathbf{u}_{12} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{v}_1^T \\ \vdots \\ \mathbf{v}_{12}^T \end{bmatrix} = s_1 \cdot \mathbf{u}_1 \cdot \mathbf{v}_1^T + \dots + s_{12} \cdot \mathbf{u}_{12} \cdot \mathbf{v}_{12}^T \Rightarrow \quad (2.112)$$

$$\mathbf{A} = \mathbf{U} \cdot \mathbf{S} \cdot \mathbf{V}^T = \sum_{i=1}^{12} s_i \cdot \mathbf{u}_i \cdot \mathbf{v}_i^T = \sum_{i=1}^{12} s_i \cdot \mathbf{B}_i$$

where $\mathbf{B}_i = \mathbf{u}_i \cdot \mathbf{v}_i^T$ are a series of Nx12 matrices with the same size of $\mathbf{A}$ multiplied by the decreasing singular values.

Because $\mathbf{U}$ and $\mathbf{V}$ are orthonormal matrices, then the product of their columns is a Kronecker delta, as we saw in (2.39), therefore the product of two different matrices $\mathbf{B}_i, \mathbf{B}_j$ is:

$$\mathbf{B}_i \cdot \mathbf{B}_j^T = \mathbf{u}_i \cdot \mathbf{v}_i^T \cdot (\mathbf{u}_j \cdot \mathbf{v}_j^T)^T = \mathbf{u}_i \cdot \mathbf{v}_i^T \cdot \mathbf{v}_j \cdot \mathbf{u}_j^T = \delta_{ij} \cdot \mathbf{u}_i \cdot \mathbf{u}_j^T = \delta_{ij} \cdot \mathbf{C} \quad (2.113)$$

Therefore, if $i \neq j$, then matrices $\mathbf{B}_i, \mathbf{B}_j$ are orthogonal, and consequently the set of matrices $\mathbf{B}_i$, for $i = 1, \dots, 12$ constitute a basis of the subspace of Nx12 matrices.

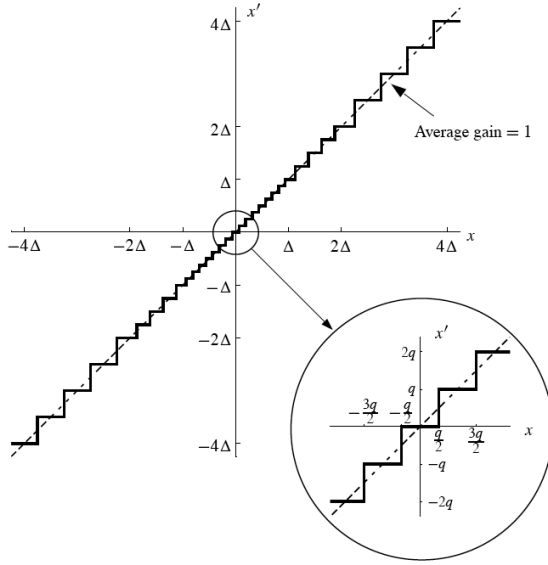


Fig. 2.12. Input/output relation for a floating-point quantizer

one, then $\mathbf{A}'$ is an approximation of $\mathbf{A}$ in LMSE terms with rank 11, as shown below:

$$\begin{aligned}\mathbf{A} &= \mathbf{U} \cdot \mathbf{S} \cdot \mathbf{V}^T = \mathbf{U} \cdot \text{diag}(s_1, s_2, \dots, s_{11}, s_{12}) \cdot \mathbf{V}^T \\ \mathbf{A}' &= \mathbf{U} \cdot \text{diag}(s_1, s_2, \dots, s_{11}, 0) \cdot \mathbf{V}^T\end{aligned}\quad (2.114)$$

where $\mathbf{A}'$ is an LMSE approximation of $\mathbf{A}$ and the smallest singular value s_{12} has been set to zero.

Therefore the solution to $\mathbf{A}' \cdot \mathbf{p} = \mathbf{0}_N$ can also be seen as an LMSE solution to $\mathbf{A} \cdot \mathbf{p} = \mathbf{0}_N$. Such solution corresponds to the column of $\mathbf{V}$ corresponding to the *smallest* singular value of $\mathbf{A}$, as we saw in section 2.3.1.1

In summary, assuming that the SVD decomposition of $\mathbf{A}$ provides ordered positive singular values, then the last column of $\mathbf{V}$ contains the LMSE estimation of the camera matrix $\mathbf{P}$.

2.5.1.2.- Other considerations

Theoretically, SVD should provide an LMSE estimation of $\mathbf{P}$ after solving (2.106), however, when computing such solution, due to machine precision and possible errors in point correspondences there are certain issues that must be considered.

First of all, because of the limitations in precision imposed by any machine, it has been suggested to normalize point coordinates before attempting to estimate a given camera.

Notice that because of the way that floating point quantizers work, points closer to zero tend to have higher precision than those located further away, as can be seen on fig. 2.12. For that reason it can be a good idea to attempt to normalize image point coordinates before estimating the camera matrix $\mathbf{P}$, although in practical situations we have found that improvements due to point normalization in most common image sizes can be neglected.

Then, starting from a given point image $\mathbf{m}$, we can convert it to another normalized point $\mathbf{m}'$ by means of the transformation matrix $\mathbf{T}$, so that,

$$\mathbf{m}' = \mathbf{T} \cdot \mathbf{m} \quad (2.115)$$

where the transformation matrix $\mathbf{T}$ can be expressed as:

As a consequence, expression (2.112) can be seen as the decomposition of matrix $\mathbf{A}$ in the basis of $\mathbf{B}_i$ matrices, where the singular values s_i are the components of $\mathbf{A}$ in the new basis of $\mathbf{B}_i$.

Hence, the singular value with the smallest absolute value (notice that singular values are usually given as positive values) contributes with the smallest amount of energy (which is related to the squared error) to the matrix $\mathbf{A}$. Therefore, by setting the smallest singular value (in absolute value) to zero we can obtain an approximation of $\mathbf{A}$ in LMSE terms.

Because $\mathbf{A}$ will in general have rank 12 if the number of equations N is $N \geq 12$, which leads to the trivial null solution $\mathbf{p} = \mathbf{0}_{12}$ of (2.106), if $\mathbf{A} \cdot \mathbf{p} = \mathbf{0}_N$ has any solution at all; then by setting one of its singular values to zero as we saw in section 2.3.1.1, we can obtain another matrix $\mathbf{A}'$ with rank 11. And as we have proven above, if the singular value that is being set to zero is the smallest

$$\mathbf{T} = \begin{bmatrix} s & 0 & -s\tilde{c}_x \\ 0 & s & -s\tilde{c}_y \\ 0 & 0 & 1 \end{bmatrix} \quad (2.116)$$

where s is a scale factor that ensures that normalized image points coordinates have a mean squared norm equal to 1, and c_x, c_y are the inhomogeneous coordinates of the centroid of coordinates on the non-normalized cloud of points.

The centroid can be simply computed as the mean coordinate values of all available image points $\mathbf{m}_i$, just as follows:

$$\tilde{\mathbf{c}} = (\tilde{c}_x, \tilde{c}_y)^T = \frac{1}{N} \sum_{i=1}^N \tilde{\mathbf{m}}_i \quad (2.117)$$

And the scale factor s can be computed as:

$$s = \frac{\sqrt{2}}{\frac{1}{N} \sum_{i=1}^N \|\tilde{\mathbf{m}}_i - \tilde{\mathbf{c}}\|} \quad (2.118)$$

which can be seen as a normalization of the mean distance of points respect to the centroid in (2.117) in order to obtain a mean squared norm equal to 1.

Because points have been normalized, then expression (2.115) can also be written by means of (2.3) as:

$$\mathbf{m}' = \mathbf{T} \cdot \mathbf{m} = \mathbf{T} \cdot \mathbf{P} \cdot \mathbf{M} = \mathbf{P}' \cdot \mathbf{M} \quad (2.119)$$

Therefore, we can consider a normalized camera matrix $\mathbf{P}'$ equal to:

$$\mathbf{P}' = \mathbf{T} \cdot \mathbf{P} \Rightarrow \mathbf{P} = \mathbf{T}^{-1} \cdot \mathbf{P}' \quad (2.120)$$

As a simplification of (2.116) we can consider that points are uniformly distributed throughout the whole image; therefore for an image of width w and height h , then the expected centroid expression approximately becomes:

$$E[\tilde{\mathbf{c}}] = \frac{1}{N} \sum_{i=1}^N E[\tilde{\mathbf{m}}_i] = \frac{1}{N} \sum_{i=1}^N \left(\frac{w}{2}, \frac{h}{2} \right)^T = \left(\frac{w}{2}, \frac{h}{2} \right)^T \quad (2.121)$$

And the scale factor s becomes:

$$E[s] = \frac{\sqrt{2}}{\frac{1}{N} \sum_{i=1}^N E[\|\tilde{\mathbf{m}}_i - \tilde{\mathbf{c}}\|]} = \frac{\sqrt{2}}{\frac{1}{2} \sqrt{\left(\frac{w}{2} \right)^2 + \left(\frac{h}{2} \right)^2}} = 4 \sqrt{\frac{2}{w^2 + h^2}} \quad (2.122)$$

And consequently (2.116) becomes:

$$\begin{aligned}
E[\mathbf{T}] &= \begin{bmatrix} E[s] & 0 & -E[s] \cdot E[\tilde{c}_x] \\ 0 & E[s] & -E[s] \cdot E[\tilde{c}_y] \\ 0 & 0 & 1 \end{bmatrix} = \\
&= \begin{bmatrix} 4\sqrt{\frac{2}{w^2 + h^2}} & 0 & -2w\sqrt{\frac{2}{w^2 + h^2}} \\ 0 & 4\sqrt{\frac{2}{w^2 + h^2}} & -2h\sqrt{\frac{2}{w^2 + h^2}} \\ 0 & 0 & 1 \end{bmatrix} \quad (2.123)
\end{aligned}$$

Therefore, for a typical image of size 640x480, then the expected $\mathbf{T}$ in (2.123) becomes:

$$E[\mathbf{T}] = \begin{bmatrix} 0.0071 & 0 & -4.5255 \\ 0 & 0.0071 & -3.3941 \\ 0 & 0 & 1 \end{bmatrix} \quad (2.124)$$

On the other hand, aside from machine precision, we can also have errors in the measures of coordinates on world or image points. Such errors, although they can be minimized when an LMSE solution is found when more than the necessary 5.5 points are provided, can lead to poor solutions if just one of the point coordinates is a completely wrong point match (with a really large error respect to other measures). In this case LMSE cannot cope with such large error and the solution found is not good enough.

One solution to this situations can be found by means of a WLMSE (Weighted Least Means Squared Error) when a certain measure of confidence is known for each image/world point match. However, in most situations this information will not be available, and hence other robust solutions such as RANSAC must be used instead, just as we show in the following section.

2.5.2.- Robust camera estimation using RANSAC

In section 2.5.1 we saw how to obtain the camera projection matrix $\mathbf{P}$ in LMSE for a given set of N correspondences between world and image points. However, as we mentioned in 2.5.1.2, the solution found by means of this technique might be poor if outliers are present in the set of corresponding points.

For that reason we provide the RANdom SAmple Consensus (RANSAC) algorithm, which is able to cope with a certain amount of outliers (typically up to 50%) while providing accurate results up to a desired amount of confidence.

In summary, supposing that the percentage (i.e. probability) of outliers p_o is known, then the RANSAC algorithm picks small subsets of $s = 6$ points (which is the minimum for camera matrix estimation with the method in section 2.5.1) within the whole set of N points, then estimates the camera matrix $\mathbf{P}$, and computes the reprojection error of all world points respect to their corresponding image points. All points having an error smaller than a given threshold t and being located in front of the camera after checking their cheirality (see section 2.4.2), are then considered to be inliers while the rest are discarded as outliers. This process is repeated keeping only the estimated camera $\mathbf{P}$ that provides the largest support of inlier points. The algorithm is stopped when it is assumed that results are accurate enough up to the desired confidence probability c . This process is depicted in algorithm 2.1.

Therefore, in order to use RANSAC there are still three issues that need to be solve. First, we need a way to determine the amount of outliers (or inliers) in a set of data, which can be unknown a priori; second, we need a way to set the threshold that determines whether points are inliers or not; and last, we need a way to obtain the confidence of the estimated result.

We will start by estimating the confidence of the results obtained by means of RANSAC.

Taking into account that the probability of a single point in the set of N points being an inlier is p_i , then its opposite is the probability of a single point in the set of N points is an outlier p_o , which can be expressed as:

$$p_o = p(\text{single point outlier}) = 1 - p(\text{single point inlier}) = 1 - p_i \quad (2.125)$$

Therefore, by choosing random subsets of $s = 6$ points, we obtain that the probability of all points in the subset being inliers is:

$$p(\text{all points in subset } s \text{ are inliers}) = p_i^s = (1 - p_o)^s \quad (2.126)$$

And consequently the probability of at least 1 point in the subset of $s = 6$ points being an outlier is:

$$p(\text{at least 1 outlier in subset } s) = 1 - p_i^s = 1 - (1 - p_o)^s \quad (2.127)$$

Expression (2.127) is true when picking a single subset of $s = 6$ points, however, RANSAC algorithm will indefinitely select random subsets of $s = 6$ points until the desired confidence is desired. When such condition is met RANSAC will have iterated M times. Consequently, the probability of having found at least 1 outlier in a random subset of $s = 6$ points after trying M times is:

$$\begin{aligned} p(\text{at least 1 outlier in } s \text{ after } M \text{ trials}) &= p(\text{at least 1 outlier in subset } s)^M = \\ &= (1 - p_i^s)^M = (1 - (1 - p_o)^s)^M \end{aligned} \quad (2.128)$$

And finally, the probability of not having a single outlier (all points being inliers) in a random subset $s = 6$ points after M trials is the opposite of (2.128), just as follows:

$$\begin{aligned} p(\text{all points inliers in } s \text{ after } M \text{ trials}) &= 1 - p(\text{at least 1 outlier in } s \text{ after } M \text{ trials}) = \\ &= 1 - p(\text{at least 1 outlier in subset } s)^M = 1 - (1 - p_i^s)^M = 1 - (1 - (1 - p_o)^s)^M \end{aligned} \quad (2.129)$$

Therefore, (2.129), can also be seen as the confidence probability c that we were looking for, as it indicates the probability of a given random subset being correct because there are no outliers. Hence,

$$c = 1 - (1 - p_i^s)^M \quad (2.130)$$

where c is the confidence probability, and p_i is the probability of a single point being an inlier, or in other words, the fraction of inliers in the whole set of N point correspondences.

It can be observed in (2.130) that the confidence probability is obtained in terms of the given amount of inlier probability and the number of iterations M . However, given a desired confidence probability and a given inlier probability, we can also find the number of iterations M that must be done to ensure that RANSAC picks the right subset of samples, just as follows:

$$c \geq 1 - (1 - p_i^s)^M \Rightarrow (1 - p_i^s)^M \geq 1 - c \Rightarrow \log(1 - p_i^s)^M \geq \log(1 - c) \Rightarrow M \cdot \log(1 - p_i^s) \geq \log(1 - c) \quad (2.131)$$

And therefore, the minimum number of iterations M to be done is:

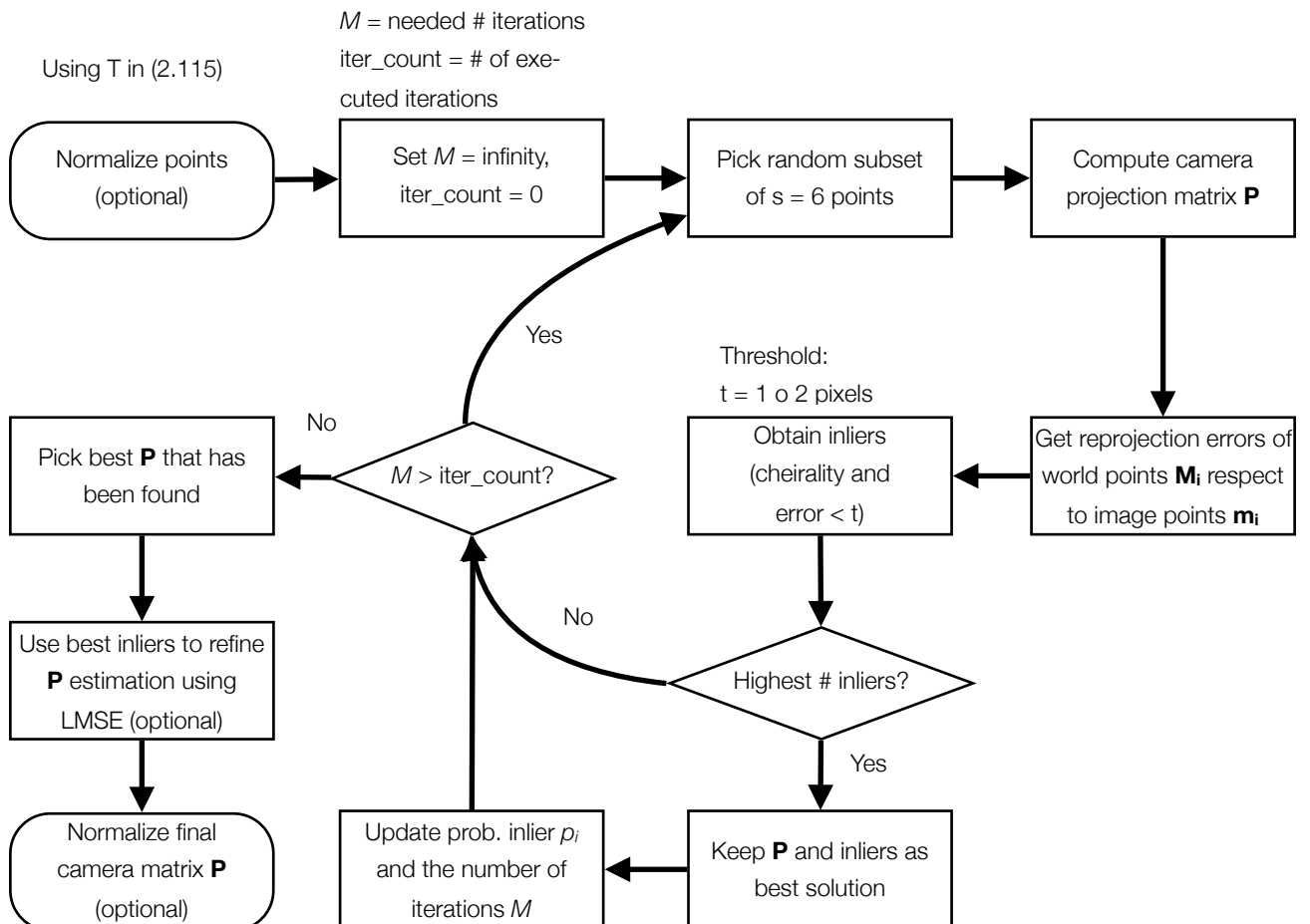
$$M \geq \frac{\log(1 - c)}{\log(1 - p_i^s)} \quad (2.132)$$

where c is desired amount of confidence (i.e. typically around 0.95), and p_i is the inlier probability, which a priori is unknown.

Notice in (2.132) that although the DLT algorithm can be used with as many s points as we want (with a minimum of $s = 6$), in order to obtain the smallest possible number of iterations M for a fixed probability of inlier p_i , we will always choose the smallest possible value for s . Therefore, s will always be $s = 6$.

We previously said that we can determine on each iteration whether a given point correspondence is an inlier or not by comparing the reprojection error respect to a given threshold t and the cheirality of world points respect to the estimated camera. Although the threshold can be expressed mathematically in terms of a chi squared distribution when we assume that the error is gaussian, in general this won't be true. Hence, in most situations the threshold t is usually set empirically. Also Notice that the probability of inliers p_i will depend on the selected threshold t , and if we choose a too strict threshold,

Algorithm 2.1. Robust camera estimation using RANSAC



then this probability p_i will decrease, making the number of needed iterations M to increase very fast (i.e. rapidly increasing the computational time). Consequently, the threshold t must be carefully chosen empirically to ensure that a good solution with small error is found in a reasonable amount of time.

In that sense, we have found experimentally that setting the threshold $t = 1$ or $t = 2$ pixels of reprojection error is good enough to correctly estimate the camera projection matrix $\mathbf{P}$ in most cases.

Finally, because the probability of inlier is unknown a priori, then we will initially set it to zero (which makes the number of iterations M equal to infinity), to take into account the worst case. And each time the RANSAC algorithm finds a new camera projection matrix $\mathbf{P}$ that increases the number of inliers respect to previous estimations on previous iterations, the probability of inlier will be updated as a fraction of the number of inliers on current iteration respect to the total number of points N . As we said, this process is depicted in algorithm 2.1.

In order to refine the obtained solution, once the M iterations are done to obtain the camera estimation with the desired amount of confidence, we will use the LMSE process described in section 2.5.1.1 using only the inlier correspondences corresponding to the camera $\mathbf{P}$ with the largest support of inliers that has been found so far.

As final note, notice that because of machine precision, camera matrices $\mathbf{P}$ are usually normalized (i.e. typically their p_{33} element is set to 1).

2.6.- Point triangulation

In section 2.5.1 we saw that the DLT algorithm can be used to estimate the camera projection matrix $\mathbf{P}$ for a given set of world an image point correspondences. However, this same algorithm can also be used to triangulate pairs of corresponding image points into 3D world points as will be shown next in section 2.6.1.

The main issue here, as will be discussed in Part II (chapters 6 to 9) will be to accurately determine such image point correspondences in order to obtain accurate triangulations.

In the following section we will also show that due to image point correspondence resolution, 3D point triangulation is also limited in resolution. In such cases we will discuss some techniques that can be used to increase such resolution, such as correction of matched points for a given epipolar geometry, or disparity estimation with sub-pixel accuracy.

Finally, on section 2.6.3 we will show how the depth of a 3D point respect to a given pair of cameras is inversely related to the disparity of its corresponding pair of image points.

2.6.1.- The DLT algorithm for point triangulation

Given a pair of corresponding image points $\mathbf{m}_1 = (x_1, y_1, w_1)^T$ and $\mathbf{m}_2 = (x_2, y_2, w_2)^T$ projected in a given pair of images by means of cameras $\mathbf{P}_1$ and $\mathbf{P}_2$, and given that such image points correspond to the same 3D world point $\mathbf{M} = (X, Y, Z, W)^T$, we can state by means of (2.3) that,

$$\begin{aligned}\mathbf{m}_1 &= \mathbf{P}_1 \cdot \mathbf{M} \\ \mathbf{m}_2 &= \mathbf{P}_2 \cdot \mathbf{M}\end{aligned}\tag{2.133}$$

And by expressing the cameras in terms of their rows as:

$$\mathbf{P}_1 = \begin{bmatrix} \mathbf{p}_1^{1T} \\ \mathbf{p}_1^{2T} \\ \mathbf{p}_1^{3T} \end{bmatrix}, \quad \mathbf{P}_2 = \begin{bmatrix} \mathbf{p}_2^{1T} \\ \mathbf{p}_2^{2T} \\ \mathbf{p}_2^{3T} \end{bmatrix}\tag{2.134}$$

we obtain that (2.133) can be written as:

$$\begin{aligned}
\mathbf{m}_1 = \mathbf{P}_1 \cdot \mathbf{M} &= \begin{bmatrix} \mathbf{p}_1^{1T} \\ \mathbf{p}_1^{2T} \\ \mathbf{p}_1^{3T} \end{bmatrix} \cdot \mathbf{M} = \begin{bmatrix} x_1 \\ y_1 \\ w_1 \end{bmatrix} \Rightarrow \left. \begin{aligned} x_1 &= \mathbf{p}_1^{1T} \cdot \mathbf{M} \\ y_1 &= \mathbf{p}_1^{2T} \cdot \mathbf{M} \\ w_1 &= \mathbf{p}_1^{3T} \cdot \mathbf{M} \end{aligned} \right\} \\
\mathbf{m}_2 = \mathbf{P}_2 \cdot \mathbf{M} &= \begin{bmatrix} \mathbf{p}_2^{1T} \\ \mathbf{p}_2^{2T} \\ \mathbf{p}_2^{3T} \end{bmatrix} \cdot \mathbf{M} = \begin{bmatrix} x_2 \\ y_2 \\ w_2 \end{bmatrix} \Rightarrow \left. \begin{aligned} x_2 &= \mathbf{p}_2^{1T} \cdot \mathbf{M} \\ y_2 &= \mathbf{p}_2^{2T} \cdot \mathbf{M} \\ w_2 &= \mathbf{p}_2^{3T} \cdot \mathbf{M} \end{aligned} \right\}
\end{aligned} \tag{2.135}$$

Therefore, by expressing image points $\mathbf{m}_1$ and $\mathbf{m}_2$ in their respective inhomogeneous coordinates,

$$\begin{aligned}
\mathbf{m}_1 &= (\tilde{x}_1, \tilde{y}_1)^T = \left(\frac{x_1}{w_1}, \frac{y_1}{w_1} \right)^T \\
\mathbf{m}_2 &= (\tilde{x}_2, \tilde{y}_2)^T = \left(\frac{x_2}{w_2}, \frac{y_2}{w_2} \right)^T
\end{aligned} \tag{2.136}$$

we obtain by means of (2.135):

$$\begin{aligned}
\tilde{x}_1 &= \frac{x_1}{w_1} = \frac{\mathbf{p}_1^{1T} \cdot \mathbf{M}}{\mathbf{p}_1^{3T} \cdot \mathbf{M}} \\
\tilde{y}_1 &= \frac{y_1}{w_1} = \frac{\mathbf{p}_1^{2T} \cdot \mathbf{M}}{\mathbf{p}_1^{3T} \cdot \mathbf{M}} \\
\tilde{x}_2 &= \frac{x_2}{w_2} = \frac{\mathbf{p}_2^{1T} \cdot \mathbf{M}}{\mathbf{p}_2^{3T} \cdot \mathbf{M}} \\
\tilde{y}_2 &= \frac{y_2}{w_2} = \frac{\mathbf{p}_2^{2T} \cdot \mathbf{M}}{\mathbf{p}_2^{3T} \cdot \mathbf{M}}
\end{aligned} \tag{2.137}$$

which can also be written as:

$$\begin{aligned}
\tilde{x}_1 \cdot \mathbf{p}_1^{3T} \cdot \mathbf{M} &= \mathbf{p}_1^{1T} \cdot \mathbf{M} \\
\tilde{y}_1 \cdot \mathbf{p}_1^{3T} \cdot \mathbf{M} &= \mathbf{p}_1^{2T} \cdot \mathbf{M} \\
\tilde{x}_2 \cdot \mathbf{p}_2^{3T} \cdot \mathbf{M} &= \mathbf{p}_2^{1T} \cdot \mathbf{M} \\
\tilde{y}_2 \cdot \mathbf{p}_2^{3T} \cdot \mathbf{M} &= \mathbf{p}_2^{2T} \cdot \mathbf{M}
\end{aligned} \tag{2.138}$$

And moving all the terms to the same side of the equations and equating to zero, we obtain the following homogeneous system of equations where the 3D world point $\mathbf{M}$ is the unknown:

$$\begin{aligned}
\tilde{x}_1 \cdot \mathbf{p}_1^{3T} \cdot \mathbf{M} - \mathbf{p}_1^{1T} \cdot \mathbf{M} &= 0 \\
\tilde{y}_1 \cdot \mathbf{p}_1^{3T} \cdot \mathbf{M} - \mathbf{p}_1^{2T} \cdot \mathbf{M} &= 0 \\
\tilde{x}_2 \cdot \mathbf{p}_2^{3T} \cdot \mathbf{M} - \mathbf{p}_2^{1T} \cdot \mathbf{M} &= 0 \\
\tilde{y}_2 \cdot \mathbf{p}_2^{3T} \cdot \mathbf{M} - \mathbf{p}_2^{2T} \cdot \mathbf{M} &= 0
\end{aligned} \tag{2.139}$$

where (2.139) can be written in matrix form as:

$$\mathbf{A} \cdot \mathbf{M} = \mathbf{0} \Rightarrow
\begin{bmatrix}
\tilde{x}_1 \cdot \mathbf{p}_1^{3T} - \mathbf{p}_1^{1T} \\
\tilde{y}_1 \cdot \mathbf{p}_1^{3T} - \mathbf{p}_1^{2T} \\
\tilde{x}_2 \cdot \mathbf{p}_2^{3T} - \mathbf{p}_2^{1T} \\
\tilde{y}_2 \cdot \mathbf{p}_2^{3T} - \mathbf{p}_2^{2T}
\end{bmatrix} \cdot \mathbf{M} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{2.140}$$

As before, the homogeneous linear system of equations in (2.140) can be solved by means of SVD of matrix A by choosing the right singular column vector corresponding to the smallest singular value as was proved for matrix estimation in section 2.5.1.

Notice that if more than 2 views of the same world point are available, then (2.140) can be written as:

$$\mathbf{A} \cdot \mathbf{M} = \mathbf{0} \Rightarrow
\begin{bmatrix}
\tilde{x}_1 \cdot \mathbf{p}_1^{3T} - \mathbf{p}_1^{1T} \\
\tilde{y}_1 \cdot \mathbf{p}_1^{3T} - \mathbf{p}_1^{2T} \\
\vdots \\
\tilde{x}_i \cdot \mathbf{p}_i^{3T} - \mathbf{p}_i^{1T} \\
\tilde{y}_i \cdot \mathbf{p}_i^{3T} - \mathbf{p}_i^{2T} \\
\vdots \\
\tilde{x}_N \cdot \mathbf{p}_N^{3T} - \mathbf{p}_N^{1T} \\
\tilde{y}_N \cdot \mathbf{p}_N^{3T} - \mathbf{p}_N^{2T}
\end{bmatrix} \cdot \mathbf{M} = \mathbf{0}_N \tag{2.141}$$

where subindex i indicates the image point coordinates and camera rows corresponding to the i^{th} view, and N is the number of total views for a given 3D world point $\mathbf{M}$.

And again, the system of equations is also solved by means of SVD, although the solution is now in LMSE terms. Notice however the solution found has minimum squared error in homogeneous coordinates, but when converting to inhomogeneous ones this might result in larger than expected errors. Therefore, when more than 2 views are available, solving by SVD should be avoided and instead we will need to minimize the following euclidean distance criterion:

$$c(\mathbf{M}) = \sum_{i=1}^N D(\mathbf{P}_i \cdot \mathbf{M}, \mathbf{m}_i) \tag{2.142}$$

How to achieve this minimization is out of the scope of this document and instead we suggest using the following method.

2.6.1.1.-Triangulating by means of inhomogeneous method.

Instead of solving (2.139) as an homogeneous linear system of equations, we could set the last homogeneous component of the world points $\mathbf{M}$ being triangulated to one, so that $\mathbf{M} = (X, Y, Z, 1)^T$. Therefore (2.139) becomes an inhomogeneous linear system of equations as the following one.

$$\begin{bmatrix} \tilde{x}_1 \cdot p_{31}^1 - p_{11}^1 & \tilde{x}_1 \cdot p_{32}^1 - p_{12}^1 & \tilde{x}_1 \cdot p_{33}^1 - p_{13}^1 \\ \tilde{y}_1 \cdot p_{31}^1 - p_{21}^1 & \tilde{y}_1 \cdot p_{32}^1 - p_{22}^1 & \tilde{y}_1 \cdot p_{33}^1 - p_{23}^1 \\ \tilde{x}_2 \cdot p_{31}^2 - p_{11}^2 & \tilde{x}_2 \cdot p_{32}^2 - p_{12}^2 & \tilde{x}_2 \cdot p_{33}^2 - p_{13}^2 \\ \tilde{y}_2 \cdot p_{31}^2 - p_{21}^2 & \tilde{y}_2 \cdot p_{32}^2 - p_{22}^2 & \tilde{y}_2 \cdot p_{33}^2 - p_{23}^2 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} p_{14}^1 - \tilde{x}_1 \cdot p_{34}^1 \\ p_{24}^1 - \tilde{y}_1 \cdot p_{34}^1 \\ p_{14}^2 - \tilde{x}_2 \cdot p_{34}^2 \\ p_{24}^2 - \tilde{y}_2 \cdot p_{34}^2 \end{bmatrix} \quad (2.143)$$

where p_{ij}^k stands for the (i,j) element of k^{th} camera matrix $\mathbf{P}_k$, and x_k, y_k , are the inhomogeneous image point coordinates on k^{th} view.

The linear system of equations in (2.143) is overdetermined and can be solved in LMSE terms by means of the Moore-Penrose pseudo-inverse. Also, if more views of the same points are provided, then more equations can be added in the linear system, as shown below,

$$\begin{bmatrix} \tilde{x}_1 \cdot p_{31}^1 - p_{11}^1 & \tilde{x}_1 \cdot p_{32}^1 - p_{12}^1 & \tilde{x}_1 \cdot p_{33}^1 - p_{13}^1 \\ \tilde{y}_1 \cdot p_{31}^1 - p_{21}^1 & \tilde{y}_1 \cdot p_{32}^1 - p_{22}^1 & \tilde{y}_1 \cdot p_{33}^1 - p_{23}^1 \\ \vdots & \vdots & \vdots \\ \tilde{x}_k \cdot p_{31}^k - p_{11}^k & \tilde{x}_k \cdot p_{32}^k - p_{12}^k & \tilde{x}_k \cdot p_{33}^k - p_{13}^k \\ \tilde{y}_k \cdot p_{31}^k - p_{21}^k & \tilde{y}_k \cdot p_{32}^k - p_{22}^k & \tilde{y}_k \cdot p_{33}^k - p_{23}^k \\ \vdots & \vdots & \vdots \\ \tilde{x}_N \cdot p_{31}^N - p_{11}^N & \tilde{x}_N \cdot p_{32}^N - p_{12}^N & \tilde{x}_N \cdot p_{33}^N - p_{13}^N \\ \tilde{y}_N \cdot p_{31}^N - p_{21}^N & \tilde{y}_N \cdot p_{32}^N - p_{22}^N & \tilde{y}_N \cdot p_{33}^N - p_{23}^N \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} p_{14}^1 - \tilde{x}_1 \cdot p_{34}^1 \\ p_{24}^1 - \tilde{y}_1 \cdot p_{34}^1 \\ \vdots \\ p_{14}^k - \tilde{x}_k \cdot p_{34}^k \\ p_{24}^k - \tilde{y}_k \cdot p_{34}^k \\ \vdots \\ p_{14}^N - \tilde{x}_N \cdot p_{34}^N \\ p_{24}^N - \tilde{y}_N \cdot p_{34}^N \end{bmatrix} \quad (2.144)$$

In this case the linear system can also be solved by means of the Moore-Penrose pseudo-inverse.

It must be noticed that using the homogeneous or inhomogeneous DLT has its advantages and disadvantages.

Usually, the inhomogeneous method returns better results in the presence of noise or when more than 2 views are provided, whereas the homogeneous method usually needs some sort of error minimization as we saw in expression (2.142).

On the other hand, the homogeneous method works even for points located at infinity, which is especially useful when working on an arbitrary projective stratum, whereas the inhomogeneous method assumes that points are finite (i.e. setting their last homogeneous component to 1), and therefore fails for points close or at infinity due to numerical instability and machine precision.

2.6.2.- Point triangulation resolution

By using expression (2.140) we can obtain 3D world points with any desired accuracy, however, in most situations coordinates of image point correspondences are given up to an accuracy of 1 pixel, as it is shown in fig. 2.13. In these cases, as can be seen on figure 2.13, for a given fixed pair of cameras, accuracy of triangulated points decreases as we get further away from cameras.

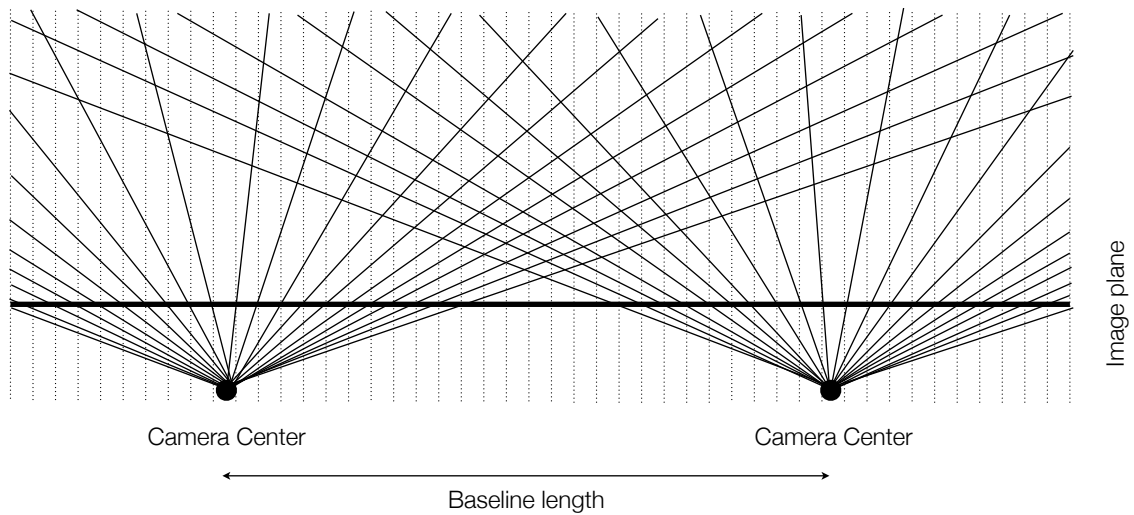


Fig. 2.13. 3D resolution vs depth and baseline length. 3D resolution gets worse as depth increases, and as the baseline decreases overall 3D resolution gets worse as well.

On the other hand, when the baseline separating the pair of cameras decreases, also resolution in 3D point triangulation is lost, although usually the quality of the estimated disparity that contains point matches increases. Therefore, there is a trade-off where we need to find the largest baseline that allows accurate disparity estimation in order to maximize resolution of 3D points. This will be explained with more detail on the second part of this document, especially on chapter 9. Indeed, will see in section 9.5 that there are several techniques that allow increasing the quality of estimated disparity (i.e. point correspondence) up to sub-pixel levels.

Finally, it must also be taken into account that when small errors (i.e. 1 or 2 pixels) are present either because of disparity estimation errors or low disparity accuracy (i.e. when no sub-pixel matches are found), but the epipolar geometry is known (i.e. the Fundamental matrix). Such errors can be optimally corrected either by using the Sampson approximation or the Gold Standard method. Both techniques will be explained with more detail in chapter 3.

2.6.3.- Relation between depth and disparity

Depth and disparity (or the modulus of optical flow) between pairs are geometrically related. Indeed by looking at figure 2.14 we can infer that as the depth of a world point respect to the cameras increases, its disparity decreases.

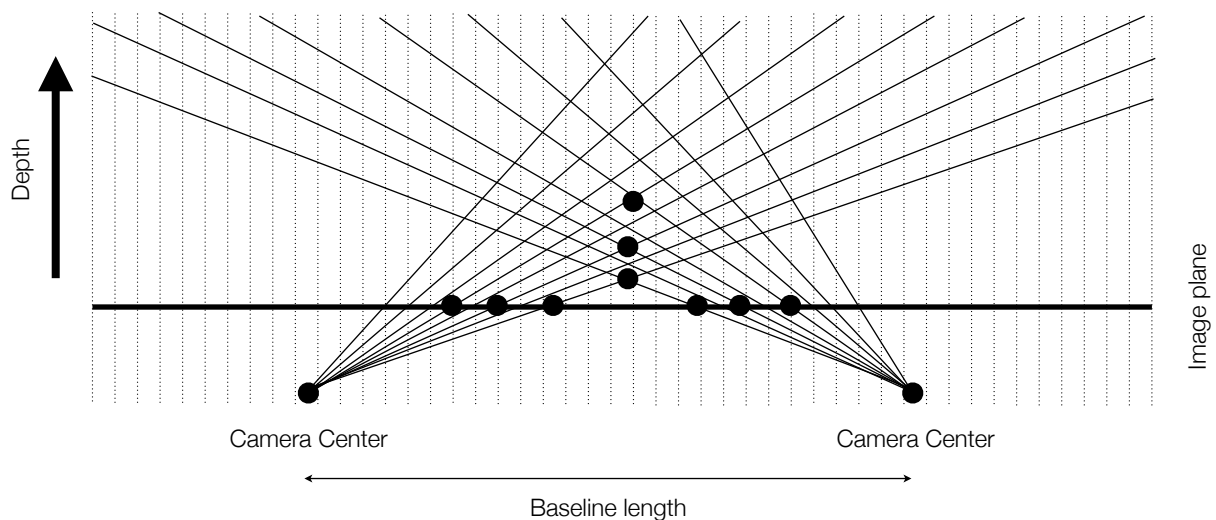


Fig. 2.14. Relation between depth and disparity.

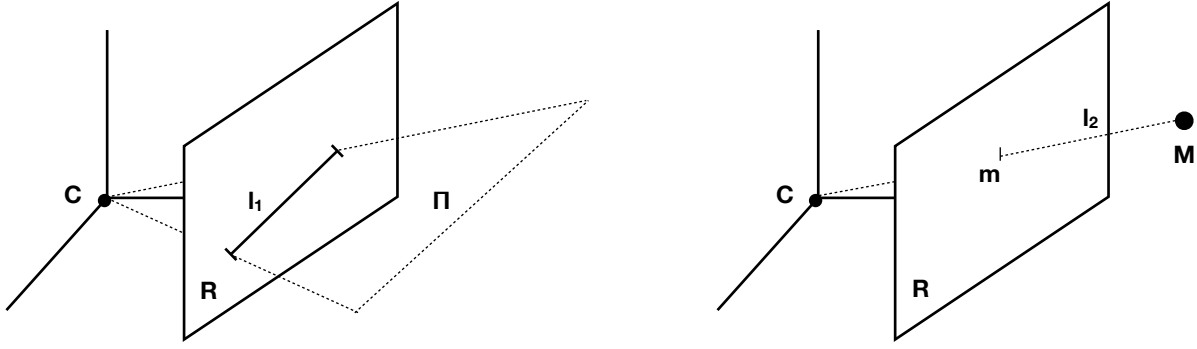


Fig. 2.15. Left: Back-projection of a general image line l_1 on the retinal plane R corresponds to a plane Π . Right: Back-projection of an image point m is a ray of light l_2 passing through the camera center C .

Indeed, for the extreme case where depth of a world point is infinite we can observe that its corresponding image points have equal coordinates (i.e. zero disparity).

Hence, from a geometric point we can state that depth is in general inversely proportional to disparity,

$$depth(\mathbf{M}, \mathbf{P}_i) \propto \frac{1}{\|\mathbf{m}_1 - \mathbf{m}_2\|}, \quad i = 1 \dots 2 \quad (2.145)$$

This assertion will be useful when attempting to compare visually the quality of a given disparity estimation.

Indeed, from this assertion, as will be seen with more detail on chapter 9, we will be able to also determine that disparity is piecewise continuous, because at object boundaries there is always a sharp change in depth which usually results in a sharp variation of disparity as well.

2.7.- Back-projection of points and lines

We've seen in (2.3) how world points are projected into the retinal plane, however another question still remains, given a single image point $\mathbf{m}$, what can we obtain if we back-project it? Geometrically, can be easily shown that the answer to this question is a ray of light (i.e. a 3D line). Just as shown below and on figure 2.15.

Therefore, given an image point $\mathbf{m}$, we want to determine the set of points in 3D space that map to this image point. Because the set of points is a ray of light, it will pass through the camera center C . Hence, any world point $\mathbf{M}'$ laying on such ray of light can be expressed as the join of two points, the camera center C and any arbitrary point $\mathbf{M}$ that projects into $\mathbf{m}$.

$$\mathbf{M}' = \mathbf{M} + \lambda \mathbf{C} \quad (2.146)$$

From expression (2.3) we now then that the world point $\mathbf{M}$ projects into image point $\mathbf{m}$ as $\mathbf{m} = \mathbf{P} \cdot \mathbf{M}$, where $\mathbf{P}$ is the 3x4 camera matrix. Because such matrix is not squared, we cannot compute its inverse, however, we can compute its Moore-Penrose pseudo-inverse assuming that its rows are independent as:

$$\mathbf{P}^\dagger = \mathbf{P}^T \cdot (\mathbf{P} \cdot \mathbf{P}^T)^{-1} \quad (2.147)$$

where in this case $\mathbf{P}^\dagger$ is a 4x3 pseudo-inverse matrix that behaves as the right inverse of $\mathbf{P}$, in other words $\mathbf{P} \cdot \mathbf{P}^\dagger = \mathbf{I}_{3 \times 3}$, as shown below:

$$\mathbf{P} \cdot \mathbf{P}^\dagger = \mathbf{P} \cdot \mathbf{P}^T \cdot (\mathbf{P} \cdot \mathbf{P}^T)^{-1} = \mathbf{I}_{3 \times 3} \quad (2.148)$$

Therefore, because of (2.147) and (2.148), and taking into account that the transposition and the pseudo-inverse can be commuted (i.e. $(\mathbf{P}^\dagger)^T = (\mathbf{P}^T)^\dagger$) we can write world point $\mathbf{M}$ as:

$$\begin{aligned}\mathbf{m} &= \mathbf{P} \cdot \mathbf{M} \Rightarrow \mathbf{m}^T = (\mathbf{P} \cdot \mathbf{M})^T \Rightarrow \\ \mathbf{m}^T \cdot (\mathbf{P}^\dagger)^T &= \mathbf{M}^T \cdot \mathbf{P}^T \cdot (\mathbf{P}^\dagger)^T = \mathbf{M}^T \cdot \mathbf{P}^T \cdot (\mathbf{P}^T)^\dagger = \mathbf{M}^T \\ \mathbf{M} &= \mathbf{P}^\dagger \cdot \mathbf{m}\end{aligned}\tag{2.149}$$

And finally, by means of (2.146), after projecting point $\mathbf{M}'$ using (2.3) we obtain:

$$\mathbf{P} \cdot \mathbf{M}' = \mathbf{P} \cdot (\mathbf{M} + \lambda \mathbf{C}) = \mathbf{P} \cdot \mathbf{P}^\dagger \cdot \mathbf{m} + \lambda \mathbf{P} \cdot \mathbf{C} = \mathbf{I}_{3 \times 3} \cdot \mathbf{m} + \lambda \cdot \mathbf{0} = \mathbf{m}\tag{2.150}$$

where we have taken into account (2.148) and (2.33). Consequently, all points $\mathbf{M}'$ laying in the right of light that joins the camera center $\mathbf{C}$ and the world point $\mathbf{M}$ project into the same image point $\mathbf{m}$. Or in other words, the back-projection of image point $\mathbf{m}$ is a ray of light.

Aside from back-projecting single points, given a line projected on the image plane, we can also show geometrically and mathematically that its back-projection corresponds to a plane. This can be obtained by means of the duality equations as follows.

Taking into account the line l in the image plane where all points $\mathbf{m}$ lie within such line, we know from expression (1.7) that $\mathbf{l}^T \cdot \mathbf{m} = 0$. Besides, we also know from (2.3) that all points $\mathbf{m}$ are projected into the image plane as $\mathbf{m} = \mathbf{P} \cdot \mathbf{M}$. Consequently, by joining both expressions, we have:

$$\mathbf{l}^T \cdot \mathbf{m} = 0 \Rightarrow \mathbf{l}^T \cdot \mathbf{P} \cdot \mathbf{M} = 0\tag{2.151}$$

However, from expression (1.74), we know that all world points $\mathbf{M}$ laying on a plane follow $\Pi^T \cdot \mathbf{M} = 0$. Therefore, expression (2.151) can also be seen as:

$$\mathbf{l}^T \cdot \mathbf{P} \cdot \mathbf{M} = 0 \Rightarrow \Pi^T \cdot \mathbf{M} = 0 \Rightarrow \Pi^T = \mathbf{l}^T \cdot \mathbf{P} \Rightarrow \Pi = \mathbf{P}^T \cdot \mathbf{l}\tag{2.152}$$

And consequently, we observe in (2.152) that the back-projection of a line on the image plane corresponds to a plane.

2.8.- Camera matrices, the geometry strata and their relation to conics and quadrics

We saw in chapter 1 that the projective 3D space can be stratified and that we could move from one stratum to another by means of transformation matrices $\mathbf{T}$. Indeed, we can observe that regardless of the transformation $\mathbf{T}$, we can obtain the same projected points $\mathbf{m}$ on the image plane assuming that both the camera projection matrix and the world points are affected by the transformation.

Indeed, we knew from (1.109) that a transformed world point $\mathbf{M}'$ can be expressed as $\mathbf{M}' = \mathbf{T} \cdot \mathbf{M}$. Therefore in order to obtain the same projected point $\mathbf{m}$ by means of (2.3) regardless of the transformation it must be true that:

$$\mathbf{m} = \mathbf{P} \cdot \mathbf{M} = (\mathbf{P} \cdot \mathbf{T}^{-1}) \cdot (\mathbf{T} \cdot \mathbf{M}) = \mathbf{P}' \cdot \mathbf{M}'\tag{2.153}$$

And as can be seen on (2.153), any transformation $\mathbf{T}$ also transforms the camera matrix as:

$$\mathbf{P}' = \mathbf{P} \cdot \mathbf{T}^{-1}\tag{2.154}$$

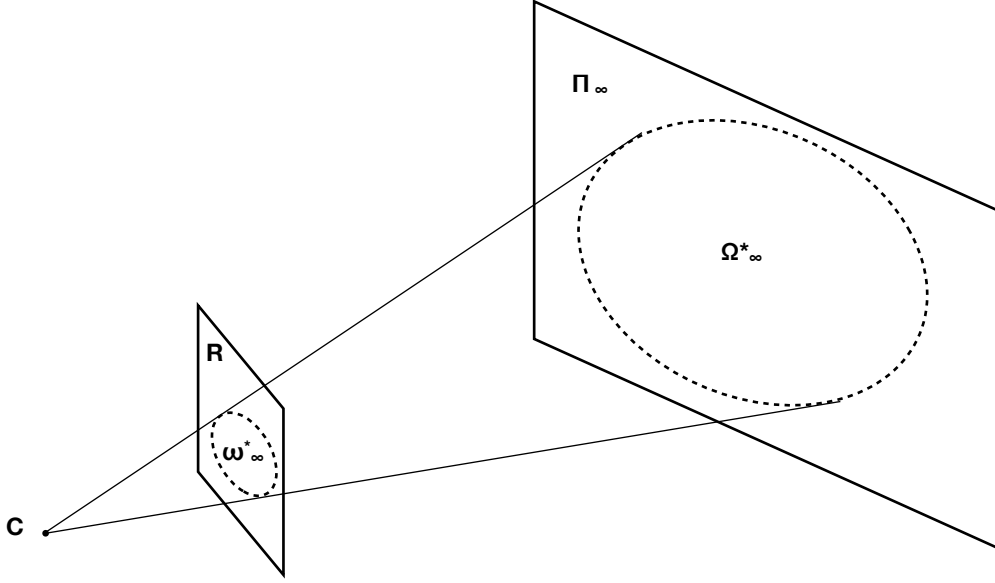


Fig. 2.16. The *dual image of the absolute conic* (DIAC) ω_{∞}^* as a projection of the *dual absolute conic* Ω_{∞}^* on the retinal plane $\mathbf{R}$ for a given camera $\mathbf{P}$ with located at camera center $\mathbf{C}$.

In chapter 5 will explain different techniques to obtain such transformation $\mathbf{T}$ so that we can obtain a 3D reconstruction in the metric stratum, because as we have seen, world points and cameras in arbitrary projective stratum can lead to the same projected image

Indeed, many of those techniques shown in chapter 5 will be based on the *dual image of the absolute conic* (DIAC) ω_{∞}^* and the *dual absolute quadric* $\mathbf{Q}_{\infty}^*$, because as we will show now, they are related by the camera projection matrix.

From expression (2.152) we know that the back-projection of a line on the image plane corresponds to a plane $\Pi = \mathbf{P}^T \cdot \mathbf{l}$. We also know from expression (1.192) that any arbitrary plane is tangent to a given quadric if it fulfills $\Pi^T \cdot \mathbf{Q}^* \cdot \Pi = 0$. Therefore, by combining both assertions we obtain:

$$\Pi^T \cdot \mathbf{Q}^* \cdot \Pi = 0 \Rightarrow \mathbf{l}^T \cdot \mathbf{P} \cdot \mathbf{Q}^* \cdot \mathbf{P}^T \cdot \mathbf{l} = 0 \quad (2.155)$$

However, in chapter 1 we also saw that any line $\mathbf{l}$ on 2D is tangent to a given conic $\mathbf{C}$ by means of expression (1.33). Then by combining both (1.33) and (2.155) we obtain:

$$\mathbf{l}^T \cdot \mathbf{C}^* \cdot \mathbf{l} = \mathbf{l}^T \cdot \mathbf{P} \cdot \mathbf{Q}^* \cdot \mathbf{P}^T \cdot \mathbf{l} = 0 \quad (2.156)$$

And consequently the dual conic $\mathbf{C}^* = \mathbf{C}^{-1}$ projected in the retinal plane in 2D by means of the camera projection matrix $\mathbf{P}$ that is also tangent to lines $\mathbf{l}$ laying on the retinal plane $\mathbf{R}$ that correspond to the projection of planes Π using the same camera matrix $\mathbf{P}$ is (up to scale):

$$\mathbf{C}^* \sim \mathbf{P} \cdot \mathbf{Q}^* \cdot \mathbf{P}^T \quad (2.157)$$

Therefore, if instead of any arbitrary dual quadric $\mathbf{Q}^*$ we use the *dual absolute quadric* $\mathbf{Q}_{\infty}^*$, we obtain that the projected conic on the retinal plane is what we call the *dual image of the absolute conic* (DIAC) ω_{∞}^* .

$$\omega_{\infty}^* \sim \mathbf{P} \cdot \mathbf{Q}_{\infty}^* \cdot \mathbf{P}^T \quad (2.158)$$

Where expression (2.158) is true for any arbitrary projective stratum, although in the metric stratum we know that the *dual absolute quadric* $\mathbf{Q}_{\infty}^*$ has the form shown in (1.179), therefore, for any arbitrary camera as in (2.24), then (2.158) becomes:

$$\begin{aligned}
\omega_{\infty}^* &\sim \mathbf{P} \cdot \mathbf{Q}_{\infty}^* \cdot \mathbf{P}^T = \\
\mathbf{K} \cdot [\mathbf{R} | \mathbf{t}] &\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{R}^T \\ \mathbf{t}^T \end{bmatrix} \mathbf{K}^T = \\
\mathbf{K} \cdot [\mathbf{R} | \mathbf{0}_3] &\begin{bmatrix} \mathbf{R}^T \\ \mathbf{t}^T \end{bmatrix} \mathbf{K}^T = \mathbf{K} \cdot \mathbf{R} \cdot \mathbf{R}^T \cdot \mathbf{K}^T + \mathbf{0}_3 \cdot \mathbf{t}^T \cdot \mathbf{K}^T \Rightarrow \\
\omega_{\infty}^* &\sim \mathbf{K} \cdot \mathbf{K}^T
\end{aligned} \tag{2.159}$$

where we can obtain the calibration matrix $\mathbf{K}$ from Cholesky decomposition once the DIAC ω_{∞}^* is known.

On the other hand, if we assume that the camera $\mathbf{P}$ is a canonical camera with the form $\mathbf{P} = [\mathbf{I}_{3 \times 3} | \mathbf{0}_3]$ and using expression (1.195), we can also observe,

$$\begin{aligned}
\omega_{\infty}^* &\sim \mathbf{P} \cdot \mathbf{Q}_{\infty}^* \cdot \mathbf{P}^T = \\
[\mathbf{I}_{3 \times 3} | \mathbf{0}_3] &\begin{bmatrix} \mathbf{A} \cdot \mathbf{A}^T & -\mathbf{A} \cdot \mathbf{A}^T \cdot \pi_{\infty} \\ -\pi_{\infty}^T \cdot \mathbf{A} \cdot \mathbf{A}^T & \pi_{\infty}^T \cdot \mathbf{A} \mathbf{A}^T \cdot \pi_{\infty} \end{bmatrix} \begin{bmatrix} \mathbf{I}_{3 \times 3} \\ \mathbf{0}_3^T \end{bmatrix} = \\
[\mathbf{I}_{3 \times 3} | \mathbf{0}_3] &\begin{bmatrix} \mathbf{A} \cdot \mathbf{A}^T \cdot \mathbf{I}_{3 \times 3} - \mathbf{A} \cdot \mathbf{A}^T \cdot \pi_{\infty} \cdot \mathbf{0}_3^T \\ -\pi_{\infty}^T \cdot \mathbf{A} \cdot \mathbf{A}^T \cdot \mathbf{I}_{3 \times 3} + \pi_{\infty}^T \cdot \mathbf{A} \mathbf{A}^T \cdot \pi_{\infty} \cdot \mathbf{0}_3^T \end{bmatrix} = \\
[\mathbf{I}_{3 \times 3} | \mathbf{0}_3] &\begin{bmatrix} \mathbf{A} \cdot \mathbf{A}^T \\ -\pi_{\infty}^T \cdot \mathbf{A} \cdot \mathbf{A}^T \end{bmatrix} = \mathbf{I}_{3 \times 3} \cdot \mathbf{A} \cdot \mathbf{A}^T - \mathbf{0}_3 \cdot \pi_{\infty}^T \cdot \mathbf{A} \cdot \mathbf{A}^T \Rightarrow \\
\omega_{\infty}^* &\sim \mathbf{A} \cdot \mathbf{A}^T
\end{aligned} \tag{2.160}$$

and hence it is also proved that the DIAC ω_{∞}^* is contained in the top-left 3x3 minor of the dual absolute quadric, as it was already assumed in chapter 1.

On the other hand, when applying any transformation $\mathbf{T}$ we know from (1.115) how a quadric $\mathbf{Q}$ is transformed, and also how it affects to a camera projection matrix $\mathbf{P}$ by means of (2.154). Combining both expressions we then obtain:

$$\omega_{\infty}^* \sim (\mathbf{P} \cdot \mathbf{T}^{-1}) \cdot (\mathbf{T} \cdot \mathbf{Q}_{\infty M}^* \cdot \mathbf{T}^T) \cdot (\mathbf{T}^{-T} \cdot \mathbf{P}^T) \tag{2.161}$$

where now $\mathbf{Q}_{\infty M}^*$ is the *dual absolute quadric* in the metric stratum as shown in (1.179), and as we mentioned before, projected points on the image plane do not change. Therefore, the DIAC remains invariant and equal to (2.159).

Indeed both expressions (2.158) and (2.161) will be used for many self-calibration techniques as will be further discussed on chapter 5.

In summary, we can then assert that the *dual absolute conic* Ω_{∞}^* is the intersection of the *dual absolute quadric* $\mathbf{Q}_{\infty}^*$ on the plane at infinity Π_{∞} , and that the projection of such conic on the plane at infinity is the *dual image of the absolute conic* (DIAC) ω_{∞}^* . This is all depicted in figure 2.16.

2.9.- Camera projection matrices and image homographies

We saw in chapter 1 that an homography can be seen as a $\mathbf{P}^2 \rightarrow \mathbf{P}^2$ linear application that transforms points from one plane to another.

Because the image is also a plane itself, then an homography of an image just looks as a distorted version of the original image. However, we will see that an homography relates planes in a 3D projective world as well, indeed we will see how the projected points laying on a plane Π into an image $\mathbf{i}$ can be described through the homography $\mathbf{H}_{\Pi\mathbf{i}}$.

Then, given a plane Π that can be expressed as $\Pi \sim (\pi^T \ 1)^T$ and a point laying on that plane $\mathbf{M}_\Pi$ expressed as $\mathbf{M}_\Pi \sim (\mathbf{m}_\Pi^T \ 1)^T$, then, because the point lays on the plane, we obtain by means of expression (1.74):

$$0 = \Pi^T \cdot \mathbf{M}_\Pi = \begin{bmatrix} \pi^T & 1 \end{bmatrix} \begin{bmatrix} \mathbf{m}_\Pi \\ 1 \end{bmatrix} = \pi^T \cdot \mathbf{m}_\Pi + 1 \Rightarrow 1 = -\pi^T \cdot \mathbf{m}_\Pi \quad (2.162)$$

Then, the world point $\mathbf{M}_\Pi$ belonging to plane Π can also be expressed as:

$$\mathbf{M}_\Pi \sim \begin{bmatrix} \mathbf{m}_\Pi \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{m}_\Pi \\ -\pi^T \cdot \mathbf{m}_\Pi \end{bmatrix} = \begin{bmatrix} \mathbf{I}_{3 \times 3} \\ -\pi^T \end{bmatrix} \cdot \mathbf{m}_\Pi \quad (2.163)$$

Then, considering the camera projection matrix $\mathbf{P}_i$ as $\mathbf{P}_i = [\mathbf{A}_i \mid \mathbf{a}_i]$, then, the i^{th} projection $\mathbf{m}_{\Pi i}$ of point $\mathbf{M}_\Pi$ on the i^{th} image plane $\mathbf{i}$ is:

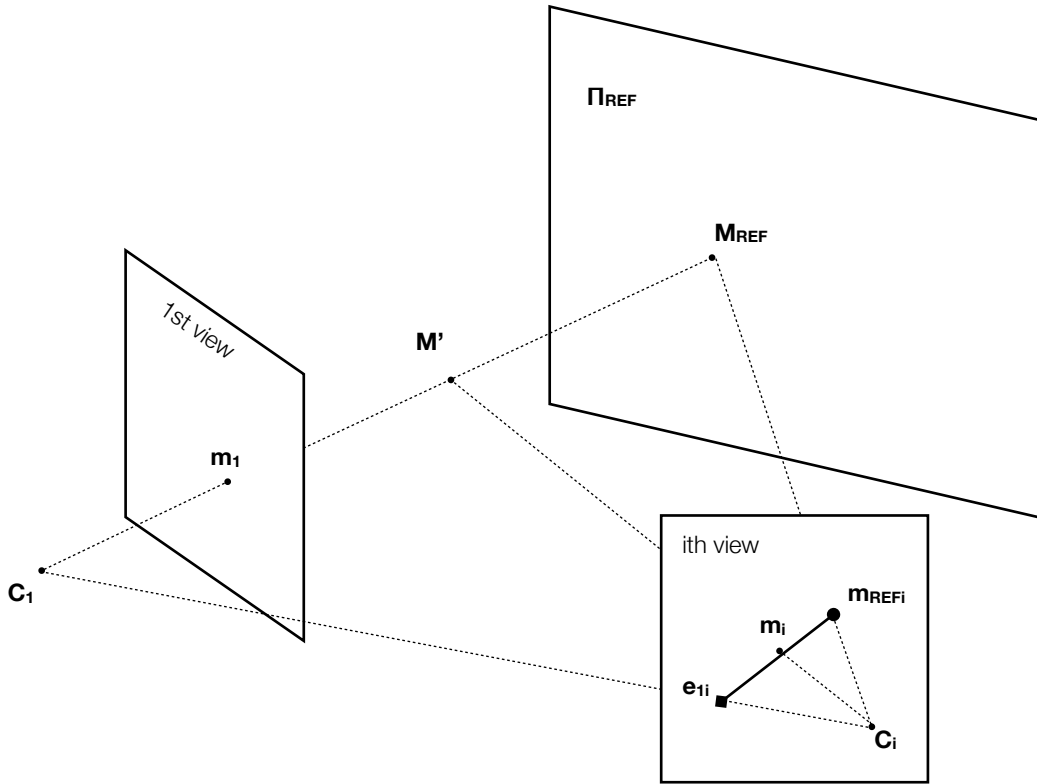


Fig. 2.17. Image points in a pair of views related by an homography. Related points lie within a line known as the epipolar line.

$$\mathbf{m}_{\Pi} \sim \mathbf{P}_i \cdot \mathbf{M}_{\Pi} = \begin{bmatrix} \mathbf{A}_i | \mathbf{a}_i \end{bmatrix} \begin{bmatrix} \mathbf{I}_{3 \times 3} \\ -\boldsymbol{\pi}^T \end{bmatrix} \cdot \mathbf{m}_{\Pi} = (\mathbf{A}_i - \mathbf{a}_i \cdot \boldsymbol{\pi}^T) \cdot \mathbf{m}_{\Pi} \quad (2.164)$$

Therefore, we can directly obtain from (2.164) that the homography relating points on plane Π respect to image plane i is:

$$\mathbf{H}_{\Pi i} \sim \mathbf{A}_i - \mathbf{a}_i \cdot \boldsymbol{\pi}^T \quad (2.165)$$

It must be noticed that usually the plane Π will be a reference plane to relate points on pairs of images, because of that, in order to simplify expression (2.165), usually the reference plane is chosen as the plane at infinity Π_{∞} , which has the following expression in the metric stratum $\Pi_{\infty} \sim (0,0,0,1)^T$ as we saw in (1.118). Therefore, the homography in (2.165) can be simplified as:

$$\mathbf{H}_{REF i} \sim \mathbf{A}_i \quad (2.166)$$

Which is an homography that relates points in the reference plane (i.e. the plane at infinity) respect to the plane of image i .

Consequently, if we wanted an homography $\mathbf{H}_{ij}^{REF}$ that related points in images i and j by means of a reference plane (in this case the plane at infinity Π_{∞}), we could first transfer the points from image i to the reference plane by means of $\mathbf{H}_{REF i}^{-1}$, and then we could transfer those points back to the image j by means of $\mathbf{H}_{REF j}$.

Therefore, the homography becomes $\mathbf{H}_{ij}^{REF}$:

$$\mathbf{H}_{ij}^{REF} = \mathbf{H}_{REF j} \cdot \mathbf{H}_{REF i}^{-1} \quad (2.167)$$

In the metric stratum we can assume that the camera $\mathbf{P}_i$ is given as in (2.32), therefore $\mathbf{A}_i$ can be expressed as $\mathbf{A}_i = \mathbf{K}_i \cdot \mathbf{R}_i^T$, and consequently (2.167) becomes:

$$\mathbf{H}_{ij}^{REF} = \mathbf{H}_{REF j} \cdot \mathbf{H}_{REF i}^{-1} = \mathbf{K}_j \cdot \mathbf{R}_j \cdot (\mathbf{K}_i \cdot \mathbf{R}_i)^{-1} = \mathbf{K}_j \cdot \mathbf{R}_j \cdot \mathbf{R}_i^{-1} \cdot \mathbf{K}_i^{-1} \quad (2.168)$$

Assuming that both cameras have the same calibration matrix, then: $\mathbf{K}_i = \mathbf{K}_j = \mathbf{K}$, and taking into account (1.129) where $\mathbf{R}^{-1} = \mathbf{R}^T$. Then (2.168) becomes:

$$\mathbf{H}_{ij}^{REF} = \mathbf{K} \cdot \mathbf{R}_j \cdot \mathbf{R}_i^T \cdot \mathbf{K}^{-1} \quad (2.169)$$

And taking into account that the rotation from the i^{th} camera to the j^{th} can be expressed as:

$$\mathbf{R}_{ij} = \mathbf{R}_i^T \cdot \mathbf{R}_j \quad (2.170)$$

Therefore, (2.169) finally becomes:

$$\mathbf{H}_{ij}^{REF} = \mathbf{K} \cdot \mathbf{R}_{ij} \cdot \mathbf{K}^{-1} \quad (2.171)$$

In the affine and projective strata, one can assume that the first camera $i = 1$ located at the origin of coordinates, pointing at the z axis and without any knowledge of its calibration matrix $\mathbf{K}_1$, can be parameterized as: $\mathbf{P}_1 = [\mathbf{I}_{3 \times 3} | \mathbf{0}_3]$, therefore the homography that transfers points from the reference plane to the first image can be represented as $\mathbf{H}_{1\Pi} \sim \mathbf{I}_{3 \times 3}$, and consequently taking into account that the last column of the camera matrix contains the image of the world origin (where in this case the first camera is located), then any camera $\mathbf{P}_i$ can be expressed as:

$$\mathbf{P}_i = \left[\mathbf{H}_{1i}^{REF} \mid \mathbf{e}_{1i} \right] \quad (2.172)$$

where $\mathbf{H}_{1i}^{REF}$ is the homography relating points from the 1st camera to the i^{th} one through the reference plane, and $\mathbf{e}_{1i}$ is the image of the first camera's center on the i^{th} camera, as will be seen with on chapter 3, which is usually known as an **epipole**.

Also notice that by combining (2.170) and (2.163), one can observe that the homography relating points from the 1st camera to the i^{th} one but using another plane Π different from the reference plane (i.e. the canonical plane at infinity) can be expressed as:

$$\mathbf{H}_{1i}^{\Pi} \sim \mathbf{H}_{1i}^{REF} - \mathbf{e}_{1i} \cdot \pi^T \quad (2.173)$$

Therefore, all homographies in a projective space are related and only differ from the term π , which has 3 parameters (i.e. degrees of freedom or DOF), that depend on the coordinates of the plane at infinity in such projective space.

Finally, from (2.172) we can observe that any world point $\mathbf{M} = (\mathbf{m}^T, 1)^T$ projected by the first camera matrix $\mathbf{P}_1$ as $\mathbf{m}_1$,

$$\mathbf{m}_1 \sim \mathbf{P}_1 \cdot \mathbf{M} = \left[\mathbf{I}_{3 \times 3} \mid \mathbf{0}_3 \right] \cdot \begin{bmatrix} \mathbf{m} \\ 1 \end{bmatrix} = \mathbf{m} \quad (2.174)$$

is also related to the projection of the same world point $\mathbf{M}$ in other views as:

$$\mathbf{m}_i \sim \mathbf{P}_i \cdot \mathbf{M} = \left[\mathbf{H}_{1i}^{REF} \mid \mathbf{e}_{1i} \right] \cdot \begin{bmatrix} \mathbf{m} \\ 1 \end{bmatrix} = \mathbf{H}_{1i}^{REF} \cdot \mathbf{m} + \mathbf{e}_{1i} \quad (2.175)$$

Because $\mathbf{m}$ and $\mathbf{m}_1$ are related up to scale (i.e. $\mathbf{m} = \lambda \cdot \mathbf{m}_1$), then (2.175) can be written as:

$$\mathbf{m}_i = \lambda \cdot \mathbf{H}_{1i}^{REF} \cdot \mathbf{m}_1 + \mathbf{e}_{1i} \quad (2.176)$$

And considering the pair of points $\mathbf{M}_1 = (\mathbf{m}_1^T, 0)^T$ and $\mathbf{M}_2 = (\mathbf{0}_3, 1)^T$, then (2.176) can be seen as the projection of another point $\mathbf{M}'$ laying on the line joining both points $\mathbf{M}_1$ and $\mathbf{M}_2$, as follows:

$$\begin{aligned} \mathbf{m}_i &= \lambda \cdot \mathbf{H}_{1i}^{REF} \cdot \mathbf{m}_1 + \mathbf{e}_{1i} = \left[\mathbf{H}_{1i}^{REF} \mid \mathbf{e}_{1i} \right] \cdot \left(\lambda \begin{bmatrix} \mathbf{m}_1 \\ 0 \end{bmatrix} + \begin{bmatrix} \mathbf{0}_3 \\ 1 \end{bmatrix} \right) = \\ &= \mathbf{P}_i \cdot \left(\lambda \begin{bmatrix} \mathbf{m}_1 \\ 0 \end{bmatrix} + \begin{bmatrix} \mathbf{0}_3 \\ 1 \end{bmatrix} \right) = \mathbf{P}_i \cdot (\lambda \mathbf{M}_1 + \mathbf{M}_2) = \mathbf{P}_i \cdot \mathbf{M}' \end{aligned} \quad (2.177)$$

where $\mathbf{M}_1 = \mathbf{M}_{REF}$ is a point laying at infinity (i.e. in the plane at infinity or the reference plane), and $\mathbf{M}_2 = \mathbf{C}_1$ is the center of the first camera located at the origin of coordinates.

As a consequence, because the point $\mathbf{M}'$ lays on a line joining $\mathbf{M}_1$ and $\mathbf{M}_2$, then its projection on the i^{th} view will also lies on a line, as depicted in figure 2.17. This line will be commonly known as an **epipolar line** and will be discussed with more detail in chapter 3. The main idea that can be gathered from figure 2.17 is that given the homography that relates pairs of images, then pairs of points in those pairs of views are related in such a way that we know their corresponding points lie in a line known as the epipolar line. This is the basis of **epipolar geometry** that will be seen in the following chapter.

3.- Multiple view geometry

3.1.- Introduction

In chapter 1 we made an introduction to projective geometry and on chapter 2 we provided a projective camera model in order to relate 3D world points with those points projected on the image plane.

At the end of chapter 2, on section 2.9 we saw that points on pairs of images are related by means of what was called the **epipolar geometry**, although it was only briefly discussed.

On this chapter we will discuss with more detail how pairs of views are related by means of homographies resulting in the epipolar geometry and the well known Fundamental matrix. We will provide algorithms to compute the Fundamental matrix, and for the cases where the calibration matrices **K** of the pair of cameras, we will provide its relation to the Essential matrix.

Finally, we will briefly discuss how errors in point correspondences affect to epipolar geometry accuracy, and will give a small insight on methods for extending the epipolar geometry to more views.

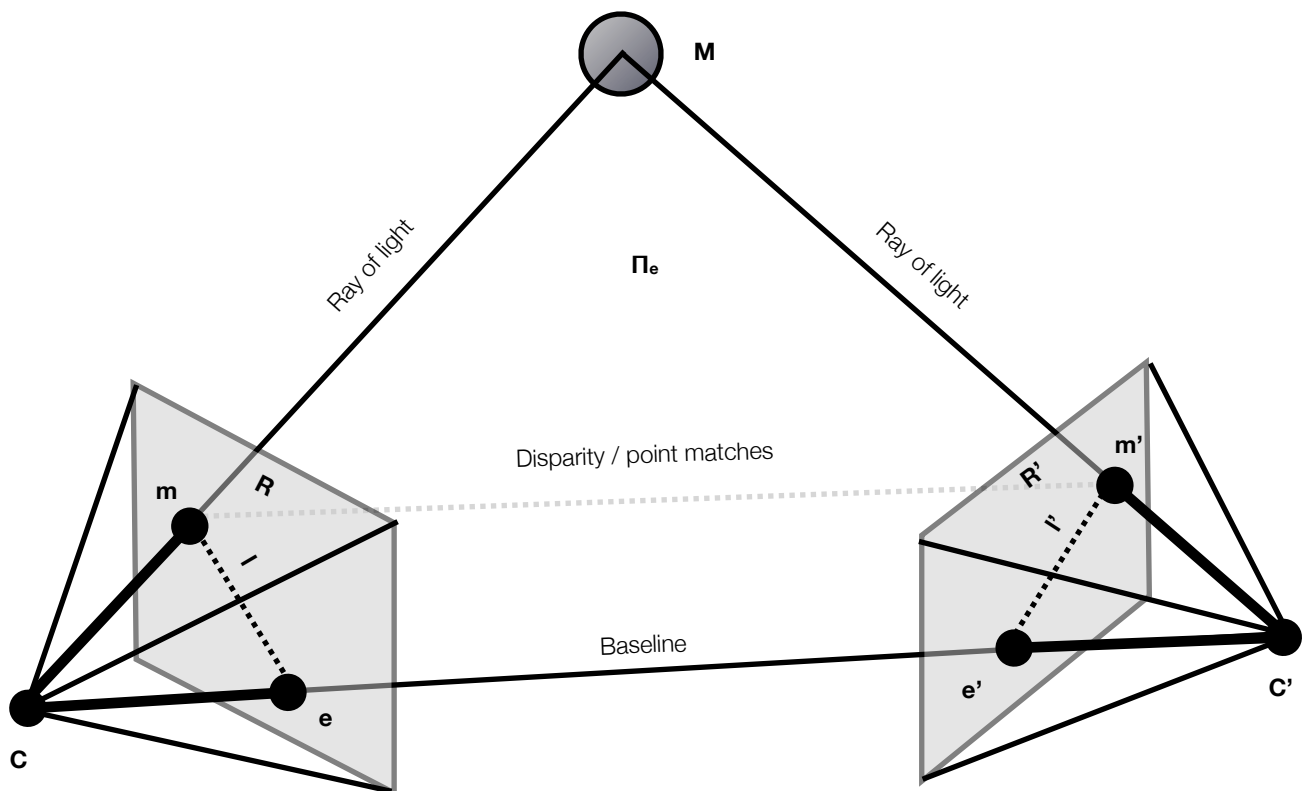


Fig. 3.1. Epipolar geometry for the two-view case. **M**: 3D point. **C**: Left camera center. **C'**: Right camera center. **l**: Left epipolar line. **l'**: Right epipolar line. **m**: left point projection. **m'**: right point projection. **e**: Left epipole. **e'**: Right epipole. **Π_e**: epipolar plane. **R**: Left Image Plane. **R'**: Right Image Plane

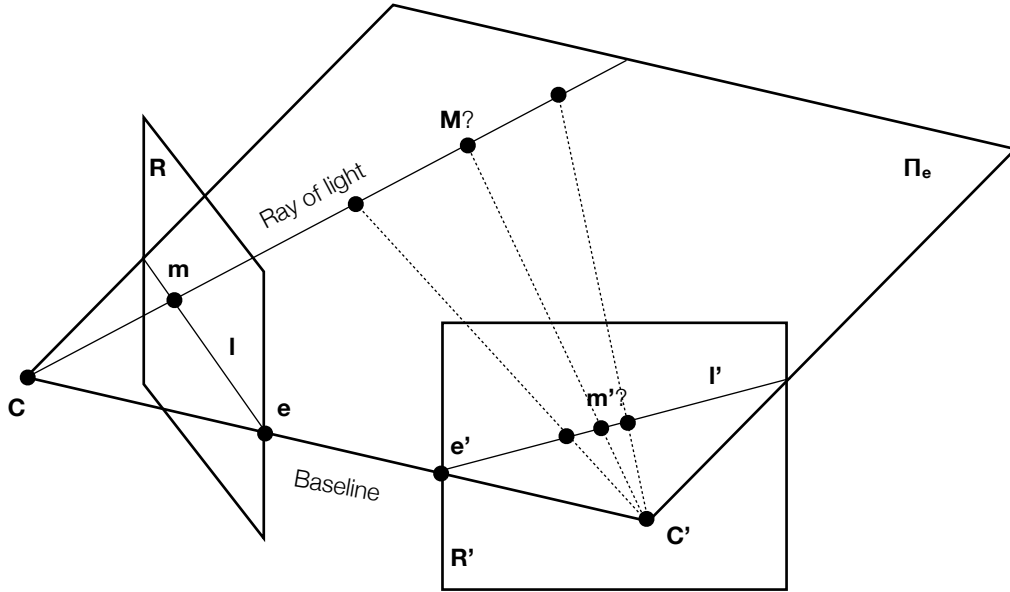


Fig. 3.2. Correspondence between two views. **C**: Left camera center. **C'**: Right camera center. **e**: Left epipole. **e'**: Right epipole. **l**: Left epipolar line. **l'**: Right epipolar line. Π_e : Epipolar plane. **R**: Left image plane. **R'**: Right image plane. **M**: 3D world point (unknown location). **m**: left image point. **m'**: right image point (unknown location or correspondence with **m**).

3.2.- Two-view geometry: The epipolar constraints

As we have already seen, epipolar geometry arises in pairs of images when trying to find point correspondences for pairs of image points **m**, **m'** that result from the projection of the same world points **M**. This is depicted in figure 3.1.

As it is depicted, it is then clear that the camera centers **C**, **C'** and any 3D world point **M** make a plane called the epipolar plane Π_e . When the world point is then projected to the respective views, such image points **m** and **m'** also lie on the epipolar plane. And analogously, the projection of the opposite camera in a pair of views also lie on the epipolar plane separated by the baseline. Such projections of camera centers are known as the epipoles **e** and **e'**. Finally, the intersection of the epipolar plane with the image planes result in the epipolar lines **l** and **l'**.

It must be noticed that when the exact location of the world point **M** is unknown, because only its projections **m** and **m'** are available in a pair of views, then in order to find such correspondence between image points **m** and **m'** we know that such search can be constrained to an epipolar line **l** or **l'**. In other words, as will be seen, an image point **m** on the left view will generate an epipolar line **l'** on the right view, where the right image point **m'** will lie. And correspondingly, an image point **m'** on the right view will generate another epipolar line **l** on the left view, where the left image point **m** will lie. This is all depicted in figures 3.1 and 3.2.

It can also be observed in figure 3.3, that for different world points **M**, then different epipolar planes are generated. All of those planes will intersect in the baseline generating a pencil of epipolar planes. Therefore, on the image planes of the pair of views **R** and **R'**, the pencil of epipolar planes will appear as a pencil of epipolar lines where all of them intersect in the corresponding epipole **e** or **e'**.

Finally, as we have already mentioned, since we want to retrieve 3D structure from images, world points **M** will be unavailable, as well as the camera projection matrices **P** and **P'**. Therefore, we will need to find some structure that allows us to obtain the epipolar constraints that forces corresponding points to lie on epipolar lines. Such structure will be the Fundamental matrix.

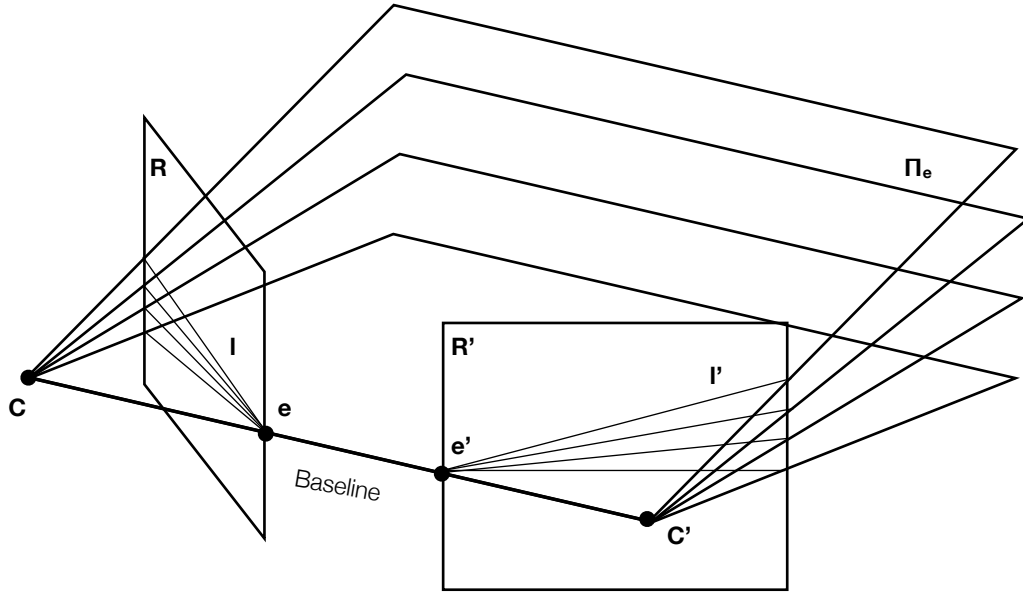


Fig. 3.3. Pencils of epipolar planes Π_e and epipolar lines l, l' . All epipolar lines intersect in the epipoles e, e' .

3.2.1.- The Fundamental matrix

The epipolar constraints that we have seen so far in section 3.2 and figures 3.1-3.3 can be expressed mathematically as follows.

First of all, because any projected image point m on the left view lies on the epipolar line l , then, because of expression (1.7) we have: $l^T \cdot m = 0$. We also know that such epipolar line l passes through the epipole e , because all epipolar lines l intersect the epipole e , and also passes through point m as we have already said. Therefore, by means of expression (1.8) we know that such line can also be expressed as:

$$l \sim [e]_{\times} \cdot m \quad (3.1)$$

where $[e]_{\times}$ is the 3x3 skew-symmetric matrix representing the cross product with vector e .

On section 2.7 we saw how lines are back-projected into planes, hence, by means of expression (2.152), and by looking at figures 3.1-3.3, we know that epipolar lines l and l' on left and right view respectively, back-project to the same epipolar plane Π_e . Therefore:

$$\Pi_e \sim P^T \cdot l = P'^T \cdot l' \quad (3.2)$$

Then the epipolar line on the right view l' can be obtained from (3.1) and (3.2) and by means of the pseudo-inverse as:

$$l' \sim (P'^T)^{\dagger} \cdot P^T \cdot l = (P'^T)^{\dagger} \cdot P^T \cdot [e]_{\times} \cdot m = F \cdot m \quad (3.3)$$

where we can define F as the 3x3 fundamental matrix, and l' is the epipolar line on the right view that is generated by the image point m on the left view by means of the fundamental matrix F .

Because corresponding image point m' on the right view lies on epipolar line l' , then it fulfills expression (1.7), and therefore:

$$m'^T \cdot l' = 0 \Rightarrow m'^T \cdot F \cdot m = 0 \quad (3.4)$$

This expression is very important as will be explained in section 3.2.2 in order to compute the fundamental matrix.

By analogy to (3.3) and using (1.7) and (3.4) we can also observe that the epipolar line $\mathbf{l}$ on the left view can be expressed as:

$$\mathbf{m}'^T \cdot \mathbf{F} \cdot \mathbf{m} = \mathbf{l}^T \cdot \mathbf{m} = 0 \Rightarrow \mathbf{l}^T = \mathbf{m}'^T \cdot \mathbf{F} \Rightarrow \mathbf{l} = \mathbf{F}^T \cdot \mathbf{m}' \quad (3.5)$$

Therefore, we can also assert that any epipolar line $\mathbf{l}$ on the left view is generated by an image point $\mathbf{m}'$ on the right view by means of the transpose of the fundamental matrix $\mathbf{F}$.

That way, we can also notice that $\mathbf{F}$ relates the left view to the right view, and oppositely $\mathbf{F}^T$ relates the right view to the left one.

It must be observed in (3.4) that because $\mathbf{m}$ and $\mathbf{m}'$ are expressed using homogeneous coordinates, then $\mathbf{F}$ is a 3×3 matrix defined up to scale. Therefore it has $8 = 3 \times 3 - 1$ unknowns.

However, because of (3.3), we also know that the fundamental matrix can be expressed as:

$$\mathbf{F} = \left(\mathbf{P}'^T \right)^\dagger \cdot \mathbf{P}^T \cdot [\mathbf{e}]_{\times} \quad (3.6)$$

Then, for any point $\mathbf{m} = \mathbf{e}$, which is the epipole on the left view, then (3.3) becomes:

$$\mathbf{l}' \sim \mathbf{F} \cdot \mathbf{e} = \left(\mathbf{P}'^T \right)^\dagger \cdot \mathbf{P}^T \cdot [\mathbf{e}]_{\times} \cdot \mathbf{e} = \left(\mathbf{P}'^T \right)^\dagger \cdot \mathbf{P}^T \cdot \mathbf{0}_3 = \mathbf{0}_3 \quad (3.7)$$

Because the cross product of the epipole with itself is zero $[\mathbf{e}]_{\times} \cdot \mathbf{e} = 0$.

Consequently, it can be observed in (3.7) that the epipole $\mathbf{e}$ is the right null-space of $\mathbf{F}$, and hence because there exist a null-space of dimension one, then the fundamental matrix $\mathbf{F}$ can have a maximum rank of 2. This also imposes the following constraint on the fundamental matrix:

$$\det(\mathbf{F}) = 0 \quad (3.8)$$

And consequently the fundamental matrix $\mathbf{F}$ has $7 = 8 - 1$ degrees of freedom due to the rank 2 constraint.

3.2.1.1.- Obtaining the Fundamental matrix from pairs of cameras

Assuming that we have a pair of calibrated cameras with their camera projection matrices $\mathbf{P}$ and $\mathbf{P}'$ available, we will show that the fundamental matrix $\mathbf{F}$ can be directly computed.

As we saw in expression (2.147), any image point $\mathbf{m}$ can be back-projected into a ray, and that any world point $\mathbf{M}$ can be expressed in terms of its projected image point $\mathbf{m}$ by means of the pseudo-inverse of the camera projection matrix $\mathbf{P}$ (i.e. $\mathbf{P} \cdot \mathbf{P}^+ = \mathbf{I}_{3 \times 3}$). Therefore the world point can be expressed as: $\mathbf{M} = \mathbf{P}^+ \cdot \mathbf{m}$.

Consequently, any point world $\mathbf{M}(\lambda)$ laying on the ray of light that projects $\mathbf{M}$ into image point $\mathbf{m}$ by means of camera matrix $\mathbf{P}$ with center $\mathbf{C}$ can be expressed as:

$$\mathbf{M}(\lambda) = \mathbf{M} + \lambda \cdot \mathbf{C} = \mathbf{P}^+ \cdot \mathbf{m} + \lambda \cdot \mathbf{C} \quad (3.9)$$

where we denote that the ray of light that contains points $\mathbf{M}(\lambda)$ passes through world point $\mathbf{M}$ and the camera center $\mathbf{C}$.

Therefore, when projecting such points with another camera $\mathbf{P}'$ corresponding to the right view, we will obtain another line $\mathbf{l}'$ that passes through the projection of world point $\mathbf{M}$ (i.e. $\mathbf{P}' \cdot \mathbf{M} = \mathbf{P}' \cdot \mathbf{P}^+ \cdot \mathbf{m}$) and the projection of the first camera center $\mathbf{C}$, which is the epipole on the right view $\mathbf{e}'$ as we already saw in previous sections (i.e. $\mathbf{e}' = \mathbf{P}' \cdot \mathbf{C}$). Such line $\mathbf{l}'$, as we saw is the right epipolar line.

Therefore, by means of expression (1.8), we know that the epipolar line on the right view $\mathbf{l}'$ can be expressed as the cross product of the two projected points that crosses. Therefore:

$$\mathbf{l}' = (\mathbf{P}' \cdot \mathbf{C}) \times (\mathbf{P}' \cdot \mathbf{M}) = (\mathbf{e}') \times (\mathbf{P}' \cdot \mathbf{P}^\dagger \cdot \mathbf{m}) = [\mathbf{e}']_{\times} \cdot (\mathbf{P}' \cdot \mathbf{P}^\dagger) \cdot \mathbf{m} = \mathbf{F} \cdot \mathbf{m} \quad (3.10)$$

And consequently the fundamental matrix can be expressed as:

$$\mathbf{F} = [\mathbf{e}']_{\times} \cdot (\mathbf{P}' \cdot \mathbf{P}^\dagger) \quad (3.11)$$

where $\mathbf{e}' = \mathbf{P}' \cdot \mathbf{C}$ is the epipole on the right view.

As will be shown later, it can be proved that although the Fundamental matrix $\mathbf{F}$ can be directly computed from a pair of cameras $\mathbf{P}$ and $\mathbf{P}'$, the opposite is not true. On section 3.2.4.1 we will see that for any given fundamental matrix $\mathbf{F}$, there exists a family of pairs of cameras $\mathbf{P}$ and $\mathbf{P}'$ defined up to a projective transformation. Therefore the solution won't be unique and we will need to correctly update the 3D geometry up to metric stratum.

3.2.1.2.- Obtaining the epipoles

We proved on section 3.2.1 that the left epipole $\mathbf{e}$ is the right-null space of the fundamental matrix $\mathbf{F}$. Therefore:

$$\mathbf{F} \cdot \mathbf{e} = \mathbf{0}_3 \quad (3.12)$$

By using the SVD of the fundamental matrix $\mathbf{F}$, and taking into account that it has rank 2 (i.e. the smallest singular value is zero). We have:

$$\begin{aligned} \mathbf{F} = \mathbf{U} \cdot \mathbf{S} \cdot \mathbf{V}^T &= \begin{bmatrix} \mathbf{u}_1 & \mathbf{u}_2 & \mathbf{u}_3 \end{bmatrix} \cdot \begin{bmatrix} s_1 & & \\ & s_2 & \\ & & 0 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{v}_1^T \\ \mathbf{v}_2^T \\ \mathbf{v}_3^T \end{bmatrix} = \\ &= \begin{bmatrix} s_1 \cdot \mathbf{u}_1 & s_2 \cdot \mathbf{u}_2 & 0 \cdot \mathbf{u}_3 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{v}_1^T \\ \mathbf{v}_2^T \\ \mathbf{v}_3^T \end{bmatrix} = s_1 \cdot \mathbf{u}_1 \cdot \mathbf{v}_1^T + s_2 \cdot \mathbf{u}_2 \cdot \mathbf{v}_2^T + 0 \cdot \mathbf{u}_3 \cdot \mathbf{v}_3^T \end{aligned} \quad (3.13)$$

where $\mathbf{U}$ is a 3x3 orthonormal matrix, $\mathbf{S}$ is a diagonal matrix containing the singular values in order from largest to smallest, $\mathbf{V}$ is another 3x3 orthonormal matrix, $\mathbf{u}_i$ is the i^{th} column of $\mathbf{U}$ and $\mathbf{v}_i$ is the i^{th} column of $\mathbf{V}$.

By taking into account expression (2.39) where we saw that the product of two different columns of an orthonormal matrix like $\mathbf{U}$ or $\mathbf{V}$ behave as a Kronecker's delta,

$$\delta_{ij} = \mathbf{v}_i^T \cdot \mathbf{v}_j = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases} \quad (3.14)$$

Then by looking at (3.13) and (3.14) we obtain that we can solve (3.12) by setting the epipole to the last column of $\mathbf{V}$, that contains the singular vector corresponding to the singular value equal to zero. Therefore,

$$\mathbf{e} = \mathbf{v}_3 \quad (3.15)$$

This result is proved next.

As a proof, (3.12) expands into the following expression by using (3.13), (3.14) and (3.15):

$$\begin{aligned}\mathbf{F} \cdot \mathbf{e} &= \mathbf{F} \cdot \mathbf{v}_3 = (s_1 \cdot \mathbf{u}_1 \cdot \mathbf{v}_1^T + s_2 \cdot \mathbf{u}_2 \cdot \mathbf{v}_2^T + 0 \cdot \mathbf{u}_3 \cdot \mathbf{v}_3^T) \cdot \mathbf{v}_3 = \\ &= s_1 \cdot \mathbf{u}_1 \cdot 0 + s_2 \cdot \mathbf{u}_2 \cdot 0 + 0 \cdot \mathbf{u}_3 \cdot 1 = \mathbf{0}_3\end{aligned}\quad (3.16)$$

By analogy, combining expression (3.5) where we showed that any epipolar line $\mathbf{l}$ can be expressed as $\mathbf{l} = \mathbf{F}^T \cdot \mathbf{m}'$, and expression (3.11) of the fundamental matrix, we obtain that any point $\mathbf{m}'$ equal to the epipole on the right view $\mathbf{m}' = \mathbf{e}'$ fulfills:

$$\mathbf{l} = \mathbf{F}^T \cdot \mathbf{e}' = ([\mathbf{e}']_{\times} (\mathbf{P}' \cdot \mathbf{P}^\dagger))^T \cdot \mathbf{e}' = (\mathbf{P}' \cdot \mathbf{P}^\dagger)^T [\mathbf{e}']_{\times} \cdot \mathbf{e}' = (\mathbf{P}' \cdot \mathbf{P}^\dagger)^T \cdot \mathbf{0}_3 = \mathbf{0}_3 \quad (3.17)$$

Therefore, the epipole on the right view $\mathbf{e}'$ is the right-null space of the transposed fundamental matrix $\mathbf{F}^T$. In other words:

$$\mathbf{F}^T \cdot \mathbf{e}' = \mathbf{0}_3 \quad (3.18)$$

By analogy with (3.12), we can then find the epipole on the right view $\mathbf{e}'$ by means of the SVD of $\mathbf{F}^T$ and choosing the singular value vector corresponding to the singular value that is equal to zero.

Notice however that the SVD of the transpose of $\mathbf{F}$ is related to the SVD of $\mathbf{F}$ as follows:

$$\mathbf{F} = \mathbf{U} \cdot \mathbf{S} \cdot \mathbf{V}^T \Rightarrow \mathbf{F}^T = (\mathbf{U} \cdot \mathbf{S} \cdot \mathbf{V}^T)^T = \mathbf{V} \cdot \mathbf{S}^T \cdot \mathbf{U}^T = \mathbf{V} \cdot \mathbf{S} \cdot \mathbf{U}^T \quad (3.19)$$

therefore, by analogy with (3.13),

$$\begin{aligned}\mathbf{F}^T &= \mathbf{V} \cdot \mathbf{S} \cdot \mathbf{U}^T = \begin{bmatrix} \mathbf{v}_1 & \mathbf{v}_2 & \mathbf{v}_3 \end{bmatrix} \cdot \begin{bmatrix} s_1 & & \\ & s_2 & \\ & & 0 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{u}_1^T \\ \mathbf{u}_2^T \\ \mathbf{u}_3^T \end{bmatrix} = \\ &= \begin{bmatrix} s_1 \cdot \mathbf{v}_1 & s_2 \cdot \mathbf{v}_2 & 0 \cdot \mathbf{v}_3 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{u}_1^T \\ \mathbf{u}_2^T \\ \mathbf{u}_3^T \end{bmatrix} = s_1 \cdot \mathbf{v}_1 \cdot \mathbf{u}_1^T + s_2 \cdot \mathbf{v}_2 \cdot \mathbf{u}_2^T + 0 \cdot \mathbf{v}_3 \cdot \mathbf{u}_3^T\end{aligned}\quad (3.20)$$

And because matrix $\mathbf{U}$ is orthonormal, the product of its column vectors also behave as a Kronecker's delta,

$$\delta_{ij} = \mathbf{u}_i^T \cdot \mathbf{u}_j = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases} \quad (3.21)$$

Therefore, by using (3.20) and (3.21), and assuming that the right epipole $\mathbf{e}'$ is $\mathbf{e}' = \mathbf{u}_3$, then (3.18) becomes:

$$\begin{aligned}\mathbf{F}^T \cdot \mathbf{e}' &= \mathbf{F}^T \cdot \mathbf{u}_3 = (s_1 \cdot \mathbf{v}_1 \cdot \mathbf{u}_1^T + s_2 \cdot \mathbf{v}_2 \cdot \mathbf{u}_2^T + 0 \cdot \mathbf{v}_3 \cdot \mathbf{u}_3^T) \cdot \mathbf{u}_3 = \\ &= s_1 \cdot \mathbf{v}_1 \cdot 0 + s_2 \cdot \mathbf{v}_2 \cdot 0 + 0 \cdot \mathbf{v}_3 \cdot 1 = \mathbf{0}_3\end{aligned}\quad (3.22)$$

Consequently

$$\mathbf{e}' = \mathbf{u}_3 \quad (3.23)$$

is the right null-space of $\mathbf{F}^T$. Where $\mathbf{u}_3$ is the last column of $\mathbf{U}$ obtained from SVD of $\mathbf{F}$ (not its transposed $\mathbf{F}^T$).

Notice that because of transposition, (3.18) can also be expressed as:

$$\mathbf{e}'^T \cdot \mathbf{F} = \mathbf{0}_3 \quad (3.24)$$

And hence, the right epipole $\mathbf{e}'$ is also the left null-space of the fundamental matrix $\mathbf{F}$.

In summary, the left epipole $\mathbf{e}$ is the *right null-space* of $\mathbf{F}$ and can be obtained as the last column of $\mathbf{V}$ in SVD decomposition of $\mathbf{F}$, and the right epipole $\mathbf{e}'$ is the *left null-space* of $\mathbf{F}$ and can be obtained as the last column of $\mathbf{U}$ in SVD decomposition of $\mathbf{F}$. As it is shown below for a given SVD of $\mathbf{F}$.

$$\begin{aligned} \mathbf{F} &= \mathbf{U} \cdot \mathbf{S} \cdot \mathbf{V}^T, \quad \mathbf{U} = (\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3), \quad \mathbf{V} = (\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3) \Rightarrow \\ \mathbf{e} &= \mathbf{v}_3 \\ \mathbf{e}' &= \mathbf{u}_3 \end{aligned} \quad (3.25)$$

3.2.2.- Computation of the Fundamental matrix

As we saw in expression (3.4), the Fundamental matrix can be derived from the following equation,

$$\mathbf{m}'^T \cdot \mathbf{F} \cdot \mathbf{m} = 0 \quad (3.26)$$

where $\mathbf{m}$ and $\mathbf{m}'$ are pairs of corresponding image points on left and right views respectively.

Therefore, expression (3.26) only depends on image point correspondences, and no knowledge is required of the camera setup (i.e. camera matrices $\mathbf{P}$ and $\mathbf{P}'$) neither on the 3D structure of the scene. Therefore, only accurate point matches are needed to correctly estimate the fundamental matrix. Notice that because accuracy on point correspondence is a difficult subject it will be explained with great detail on the second part of this document that is completely dedicated to photogrammetry (chapters 6 - 9).

Therefore, by expanding expression (3.26) we obtain:

$$\begin{aligned} \mathbf{m}'^T \cdot \mathbf{F} \cdot \mathbf{m} &= 0 \Rightarrow \\ \begin{bmatrix} \tilde{x}' & \tilde{y}' & 1 \end{bmatrix} \cdot \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix} \cdot \begin{bmatrix} \tilde{x} \\ \tilde{y} \\ 1 \end{bmatrix} &= 0 \Rightarrow \\ \begin{bmatrix} \tilde{x}' & \tilde{y}' & 1 \end{bmatrix} \cdot \begin{bmatrix} \tilde{x} \cdot f_{11} + \tilde{y} \cdot f_{12} + f_{13} \\ \tilde{x} \cdot f_{21} + \tilde{y} \cdot f_{22} + f_{23} \\ \tilde{x} \cdot f_{31} + \tilde{y} \cdot f_{32} + f_{33} \end{bmatrix} &= 0 \Rightarrow \\ \tilde{x} \cdot \tilde{x}' \cdot f_{11} + \tilde{y} \cdot \tilde{x}' \cdot f_{12} + \tilde{x}' \cdot f_{13} + \tilde{x} \cdot \tilde{y}' \cdot f_{21} + \tilde{y} \cdot \tilde{y}' \cdot f_{22} + \tilde{y}' \cdot f_{23} + \tilde{x} \cdot f_{31} + \tilde{y} \cdot f_{32} + f_{33} &= 0 \end{aligned} \quad (3.27)$$

where points $\mathbf{m}$ and $\mathbf{m}'$ are given using their inhomogeneous coordinates x, y .

Then by arranging the elements of $\mathbf{F}$ in row-major order into a single vector $\mathbf{f}$,

$$\mathbf{f} = (f_{11}, f_{12}, f_{13}, f_{21}, f_{22}, f_{23}, f_{31}, f_{32}, f_{33})^T \quad (3.28)$$

We obtain that (3.27) can be written as:

$$(\tilde{x} \cdot \tilde{x}', \tilde{y} \cdot \tilde{x}', \tilde{x}', \tilde{x} \cdot \tilde{y}', \tilde{y} \cdot \tilde{y}', \tilde{y}', \tilde{x}, \tilde{y}, 1) \cdot \mathbf{f} = 0 \quad (3.29).$$

Finally, by using multiple point correspondences stacked into a linear system of equations as the one shown below,

$$\mathbf{A} \cdot \mathbf{f} = \mathbf{0}_N \Rightarrow \begin{bmatrix} \tilde{x}_1 \cdot \tilde{x}'_1 & \tilde{y}_1 \cdot \tilde{x}'_1 & \tilde{x}'_1 & \tilde{x}_1 \cdot \tilde{y}'_1 & \tilde{y}_1 \cdot \tilde{y}'_1 & \tilde{y}'_1 & \tilde{x}_1 & \tilde{y}_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \tilde{x}_N \cdot \tilde{x}'_N & \tilde{y}_N \cdot \tilde{x}'_N & \tilde{x}'_N & \tilde{x}_N \cdot \tilde{y}'_N & \tilde{y}_N \cdot \tilde{y}'_N & \tilde{y}'_N & \tilde{x}_N & \tilde{y}_N & 1 \end{bmatrix} \cdot \mathbf{f} = \mathbf{0}_N \quad (3.30)$$

where N is the number of available point correspondences and $\mathbf{A}$ is an $N \times 9$ matrix that will generally have rank 9.

Then vector $\mathbf{f}$ containing the elements of the fundamental matrix $\mathbf{F}$ can be obtained by solving the linear system of equations in (3.30), and the fundamental matrix $\mathbf{F}$ is obtained after reordering the elements of vector $\mathbf{f}$ into a 3×3 matrix.

It must be noticed that in order to obtain a solution for $\mathbf{f}$ different than the trivial null solution, then $\mathbf{A}$ must be at least a rank 8 matrix, which is in accordance with the 8 DOF (degrees of freedom) of the fundamental matrix. Therefore 8 correspondences of points are enough to compute the fundamental matrix (i.e. $\mathbf{A}$ will be an 8×9 matrix with rank 8).

As we saw in section 2.5.1, the solution for (3.30) can be found by means of the SVD of $\mathbf{A}$,

$$\mathbf{A} = \mathbf{U} \cdot \mathbf{S} \cdot \mathbf{V}^T \quad (3.31)$$

and selecting the singular column vector of $\mathbf{V}$ corresponding to the smallest singular value. (which for the case of 8 point correspondences should be zero because $\mathbf{A}$ will be rank 8).

Notice that if **more than 8 point correspondences** are provided, then this method also solves the linear system of equations in (3.30) but in **LMSE** (Least Minimum Squared Error) terms.

3.2.2.1.- The 8-point algorithm

The 8-point algorithm for estimation of the fundamental matrix is a simple algorithm largely based on expression (3.30). However, besides of solving (3.30) the 8-point algorithm also takes into account the constraint that indicates that the fundamental matrix is a rank 2 3×3 matrix.

Therefore, after solving (3.30), the rank 2 constraint must be enforced because $\mathbf{F}$ is not ensured to be singular. For that reason, we need to search for an approximation $\mathbf{F}'$ to the fundamental matrix $\mathbf{F}$ that minimizes the Frobenius norm $\|\mathbf{F} - \mathbf{F}'\|$ (i.e. error) subject to the condition that $\det(\mathbf{F}') = 0$ to ensure the rank 2 constraint is fulfilled.

A simple method to achieve this constraint can be to use the SVD of the estimated fundamental matrix $\mathbf{F}$ after solving (3.30) to force the smallest singular value to be zero. Therefore, assuming that the estimated fundamental matrix $\mathbf{F}$ can be decomposed by means of SVD as,

$$\mathbf{F} = \mathbf{U} \cdot \mathbf{S} \cdot \mathbf{V}^T = \mathbf{U} \cdot \text{diag}(s_1, s_2, s_3) \cdot \mathbf{V}^T, \quad s_1 \geq s_2 \geq s_3 \quad (3.32)$$

Then the approximation $\mathbf{F}'$ to the fundamental matrix than minimizes the Frobenius norm while fulfilling the rank 2 constraint is:

$$\mathbf{F}' = \mathbf{U} \cdot \text{diag}(s_1, s_2, 0) \cdot \mathbf{V}^T \quad (3.33)$$

where the smallest singular value s_3 has been set to zero.

Notice that the proof that $\mathbf{F}'$ is the best approximation of $\mathbf{F}$ with smallest quadric error is shown in section 2.5.1.1, but for convenience is also shown next.

Proof 3.2

By taking into account that the SVD decomposition can be seen as a linear combination of orthogonal matrices forming a new basis $\mathbf{B}_i$, and the singular values s_i represent the components of that new basis ordered from the largest energetic component to the smallest one (i.e. $s_i \geq s_j, i < j$), just as follows,

$$\begin{aligned} \mathbf{F} = \mathbf{U} \cdot \mathbf{S} \cdot \mathbf{V}^T &= \begin{bmatrix} \mathbf{u}_1 & \mathbf{u}_2 & \mathbf{u}_3 \end{bmatrix} \cdot \begin{bmatrix} s_1 & & \\ & s_2 & \\ & & s_3 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{v}_1^T \\ \mathbf{v}_2^T \\ \mathbf{v}_3^T \end{bmatrix} = \\ &= \begin{bmatrix} s_1 \cdot \mathbf{u}_1 & s_2 \cdot \mathbf{u}_1 & s_3 \cdot \mathbf{u}_3 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{v}_1^T \\ \vdots \\ \mathbf{v}_{12}^T \end{bmatrix} = s_1 \cdot \mathbf{u}_1 \cdot \mathbf{v}_1^T + s_2 \cdot \mathbf{u}_1 \cdot \mathbf{v}_2^T + s_3 \cdot \mathbf{u}_3 \cdot \mathbf{v}_3^T \Rightarrow \\ \mathbf{F} = \mathbf{U} \cdot \mathbf{S} \cdot \mathbf{V}^T &= \sum_{i=1}^3 s_i \cdot \mathbf{u}_i \cdot \mathbf{v}_i^T = \sum_{i=1}^3 s_i \cdot \mathbf{B}_i \end{aligned} \quad (3.34)$$

Because the smallest singular value (i.e. s_3) is set to zero, then only the smallest energetic component (i.e. $\mathbf{B}_3$) is removed, and $\mathbf{F}'$ is an approximation of $\mathbf{F}$ in minimum quadratic error terms that fulfills the rank 2 constraint.

Therefore $\mathbf{F}'$ obtained in (3.33) is the estimation of the fundamental matrix provided by this algorithm, however, as in the case of the DLT algorithm for camera estimation, certain considerations regarding machine precision need to be taken into account in order to increase the accuracy of estimation.

3.2.2.1.1.- Other considerations

As in section 2.5.1.2, in order to avoid losing accuracy of estimation on the fundamental matrix $\mathbf{F}$ because of machine precision, then it is also useful to normalize corresponding image point coordinates to have an RMS equal to one, because floating-point quantizers tend to have more precision within this range of values.

The same normalization that we used in section 2.5.1.2 can also be used here for $\mathbf{F}$ estimation, therefore, any image point $\mathbf{m}$ or $\mathbf{m}'$ can be normalized, before attempting to estimate $\mathbf{F}$, by means of a transformation matrix $\mathbf{T}$ as in (2.115):

$$\begin{aligned} \hat{\mathbf{m}} &= \mathbf{T} \cdot \mathbf{m} \\ \hat{\mathbf{m}}' &= \mathbf{T}' \cdot \mathbf{m}' \end{aligned} \quad (3.35)$$

where $\mathbf{T}$ is the transformation for the left view, and $\mathbf{T}'$ is the transformation for the right view. And both transformation matrices follow expression (2.116)

$$\mathbf{T} = \begin{bmatrix} s & 0 & -s\tilde{c}_x \\ 0 & s & -s\tilde{c}_y \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{T}' = \begin{bmatrix} s' & 0 & -s'\tilde{c}'_x \\ 0 & s' & -s'\tilde{c}'_y \\ 0 & 0 & 1 \end{bmatrix} \quad (3.36)$$

where as we saw in section 2.5.1.2, (c_x, c_y) and (c'_x, c'_y) correspond to the inhomogeneous coordinates of the centroids of image points in left and right views, and s, s' correspond to the scale factor. All of them are expressed as shown below:

$$\tilde{\mathbf{c}} = (\tilde{c}_x, \tilde{c}_y)^T = \frac{1}{N} \sum_{i=1}^N \tilde{\mathbf{m}}_i, \quad \tilde{\mathbf{c}}' = (\tilde{c}'_x, \tilde{c}'_y)^T = \frac{1}{N} \sum_{i=1}^N \tilde{\mathbf{m}}'_i$$

$$s = \frac{\sqrt{2}}{\frac{1}{N} \sum_{i=1}^N \|\tilde{\mathbf{m}}_i - \tilde{\mathbf{c}}\|}, \quad s' = \frac{\sqrt{2}}{\frac{1}{N} \sum_{i=1}^N \|\tilde{\mathbf{m}}'_i - \tilde{\mathbf{c}}'\|} \quad (3.37)$$

Therefore, if points are expressed as in (3.35), then by using (3.26) on the normalized image points, we obtain:

$$\hat{\mathbf{m}}'^T \cdot \hat{\mathbf{F}} \cdot \hat{\mathbf{m}} = 0 \Rightarrow (\mathbf{T}' \cdot \mathbf{m}')^T \cdot \hat{\mathbf{F}} \cdot \mathbf{T} \cdot \mathbf{m} = \mathbf{m}'^T \cdot \mathbf{T}'^T \cdot \hat{\mathbf{F}} \cdot \mathbf{T} \cdot \mathbf{m} = \mathbf{m}'^T \cdot \mathbf{F} \cdot \mathbf{m} = 0 \quad (3.38)$$

where $\hat{\mathbf{F}}$ is the fundamental matrix estimated for the normalized matched image points.

Therefore, the fundamental matrix for the non-normalized image points is related to $\hat{\mathbf{F}}$ as follows:

$$\mathbf{F} = \mathbf{T}'^T \cdot \hat{\mathbf{F}} \cdot \mathbf{T} \Leftrightarrow \hat{\mathbf{F}} = \mathbf{T}'^T \cdot \mathbf{F} \cdot \mathbf{T}^{-1} \quad (3.39)$$

On the other hand, once the fundamental matrix $\mathbf{F}$ for the non-normalized points has been estimated, it is also usually normalized (i.e. because $\mathbf{F}$ is only defined up to scale) in order to have unitary frobenius norm to ensure that due to machine precision, epipolar lines are accurately estimated.

$$\hat{\mathbf{F}} = \frac{\mathbf{F}}{\|\mathbf{F}\|} \quad (3.40)$$

As will be seen later, this will be useful when working with robust algorithms like RANSAC, or when comparing fundamental matrices, since in both cases epipolar lines need to be compared.

3.2.2.2.- The 7-point algorithm

As in the 8-point algorithm, starting from expression (3.30), we want to find the 9x1 vector $\mathbf{f}$ containing the elements of the fundamental matrix $\mathbf{F}$, while at the same time ensuring that $\mathbf{F}$ has rank 2.

As we saw in section 3.2.2, expression (3.30) can be solved as long as 8 point correspondences are available, because then (3.30) is an homogeneous system of equations ($\mathbf{A} \cdot \mathbf{f} = 0$) where $\mathbf{A}$ has rank 8, and its solution can be found as the null-space of the matrix $\mathbf{A}$, which is the right singular vector corresponding to the null singular value.

However, because of the additional rank 2 constraint, we know then that we have 7 DOF (degrees of freedom), and consequently, with 7 point correspondences, the fundamental matrix can be found.

Therefore, if expression (3.30) is formed by 7 equations, then $\mathbf{A}$ is a 7x9 matrix which in general has rank 7. Consequently, the null-space of $\mathbf{A}$, which is the solution of (3.30) will have rank 2 because of the rank-nullity theorem. Such null-space, can be found as the 2 right singular column vectors corresponding to the two null singular values. Therefore, assuming that the SVD of $\mathbf{A}$ is:

$$\begin{aligned} \mathbf{A} = \mathbf{U} \cdot \mathbf{S} \cdot \mathbf{V}^T &= \begin{bmatrix} \mathbf{u}_1 & \cdots & \mathbf{u}_9 \end{bmatrix} \cdot \begin{bmatrix} s_1 & & \\ & \ddots & \\ & & s_9 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{v}_1^T \\ \vdots \\ \mathbf{v}_9^T \end{bmatrix} = \\ &= \begin{bmatrix} s_1 \cdot \mathbf{u}_1 & \cdots & s_9 \cdot \mathbf{u}_9 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{v}_1^T \\ \vdots \\ \mathbf{v}_9^T \end{bmatrix} = s_1 \cdot \mathbf{u}_1 \cdot \mathbf{v}_1^T + \dots + s_9 \cdot \mathbf{u}_9 \cdot \mathbf{v}_9^T = \sum_{i=1}^9 s_i \cdot \mathbf{u}_i \cdot \mathbf{v}_i^T \end{aligned} \quad (3.41)$$

where we assume that the singular values are ordered from the largest to the smallest one ($s_i \geq s_j, i < j$). Therefore, the two null singular values correspond to s_8 and s_9 , and consequently the null-space of $\mathbf{A}$ is the last two columns of $\mathbf{V}$, $\mathbf{v}_8$ and $\mathbf{v}_9$.

As in previous cases, if more than 7 equations, $\mathbf{v}_8, \mathbf{v}_9$ can also be seen as a solution for (3.30) in LMSE terms.

Therefore, we then know that the true $\mathbf{f}$ vector that contains the elements of the fundamental matrix as a solution of (3.30), must be a linear combination of $\mathbf{f}' = \mathbf{v}_8$ and $\mathbf{f}'' = \mathbf{v}_9$,

$$\mathbf{f} = \mathbf{f}' + \alpha \cdot \mathbf{f}'' = \mathbf{v}_8 + \alpha \cdot \mathbf{v}_9 \quad (3.42)$$

Consequently, the fundamental matrix $\mathbf{F}$, can also be expressed as a linear combination of matrices $\mathbf{F}'$ and $\mathbf{F}''$ which correspond to re-ordered vectors $\mathbf{f}'$ and $\mathbf{f}''$.

$$\mathbf{F} = \mathbf{F}' + \alpha \cdot \mathbf{F}'' \quad (3.43)$$

Finally, taking into account the rank 2 constraint of the fundamental matrix, which translates into null determinant as seen in (3.8)

$$\det(\mathbf{F}) = 0 \Rightarrow \det(\mathbf{F}' + \alpha \cdot \mathbf{F}'') = 0 \quad (3.44)$$

And by expanding (3.44) we obtain:

$$\begin{aligned}
\det(\mathbf{F}' + \alpha \mathbf{F}'') &= \det \begin{bmatrix} f'_{11} + \alpha f''_{11} & f'_{12} + \alpha f''_{12} & f'_{13} + \alpha f''_{13} \\ f'_{21} + \alpha f''_{21} & f'_{22} + \alpha f''_{22} & f'_{23} + \alpha f''_{23} \\ f'_{31} + \alpha f''_{31} & f'_{32} + \alpha f''_{32} & f'_{33} + \alpha f''_{33} \end{bmatrix} = \\
&= (f'_{11} + \alpha f''_{11})(f'_{22} + \alpha f''_{22})(f'_{33} + \alpha f''_{33}) + \\
&(f'_{32} + \alpha f''_{32})(f'_{21} + \alpha f''_{21})(f'_{13} + \alpha f''_{13}) + \\
&(f'_{31} + \alpha f''_{31})(f'_{12} + \alpha f''_{12})(f'_{23} + \alpha f''_{23}) - \\
&(f'_{31} + \alpha f''_{31})(f'_{22} + \alpha f''_{22})(f'_{13} + \alpha f''_{13}) - \\
&(f'_{23} + \alpha f''_{23})(f'_{32} + \alpha f''_{32})(f'_{11} + \alpha f''_{11}) - \\
&(f'_{21} + \alpha f''_{21})(f'_{12} + \alpha f''_{12})(f'_{33} + \alpha f''_{33})
\end{aligned} \tag{3.45}$$

which can be expanded as,

$$\det(\mathbf{F}' + \alpha \mathbf{F}'') = A \cdot \alpha^3 + B \cdot \alpha^2 + C \cdot \alpha + D = 0,$$

$$\begin{aligned}
A &= f''_{11} \cdot f''_{22} \cdot f''_{33} + f''_{32} \cdot f''_{21} \cdot f''_{13} + f''_{31} \cdot f''_{12} \cdot f''_{23} - \\
&f''_{31} \cdot f''_{22} \cdot f''_{13} - f''_{23} \cdot f''_{32} \cdot f''_{11} - f''_{21} \cdot f''_{12} \cdot f''_{33}
\end{aligned}$$

$$\begin{aligned}
B &= f''_{11} \cdot f''_{22} \cdot f'_{33} + f'_{11} \cdot f''_{22} \cdot f''_{33} + f''_{11} \cdot f'_{22} \cdot f''_{33} + \\
&f''_{32} \cdot f''_{21} \cdot f'_{13} + f'_{32} \cdot f''_{21} \cdot f''_{13} + f''_{32} \cdot f'_{21} \cdot f''_{13} + \\
&f''_{31} \cdot f''_{12} \cdot f'_{23} + f'_{31} \cdot f''_{12} \cdot f''_{23} + f''_{31} \cdot f'_{12} \cdot f''_{23} - \\
&f''_{31} \cdot f''_{22} \cdot f'_{13} - f'_{31} \cdot f''_{22} \cdot f''_{13} - f''_{31} \cdot f'_{22} \cdot f''_{13} - \\
&f''_{23} \cdot f''_{32} \cdot f'_{11} - f'_{23} \cdot f''_{32} \cdot f''_{11} - f''_{23} \cdot f'_{32} \cdot f''_{11} - \\
&f''_{21} \cdot f''_{12} \cdot f'_{33} - f'_{21} \cdot f''_{12} \cdot f''_{33} - f''_{21} \cdot f'_{12} \cdot f''_{33}
\end{aligned} \tag{3.46}$$

$$\begin{aligned}
C &= f'_{11} \cdot f''_{22} \cdot f'_{33} + f''_{11} \cdot f'_{22} \cdot f'_{33} + f'_{11} \cdot f'_{22} \cdot f''_{33} + \\
&f'_{32} \cdot f''_{21} \cdot f'_{13} + f''_{32} \cdot f'_{21} \cdot f'_{13} + f'_{32} \cdot f'_{21} \cdot f''_{13} + \\
&f'_{31} \cdot f''_{12} \cdot f'_{23} + f''_{31} \cdot f'_{12} \cdot f'_{23} + f'_{31} \cdot f'_{12} \cdot f''_{23} - \\
&f'_{31} \cdot f''_{22} \cdot f'_{13} - f''_{31} \cdot f'_{22} \cdot f'_{13} - f'_{31} \cdot f'_{22} \cdot f''_{13} - \\
&f'_{23} \cdot f''_{32} \cdot f'_{11} - f''_{23} \cdot f'_{32} \cdot f'_{11} - f'_{23} \cdot f'_{32} \cdot f''_{11} - \\
&f'_{21} \cdot f''_{12} \cdot f'_{33} - f''_{21} \cdot f'_{12} \cdot f'_{33} - f'_{21} \cdot f'_{12} \cdot f''_{33}
\end{aligned}$$

$$\begin{aligned}
D &= f'_{11} \cdot f'_{22} \cdot f'_{33} + f'_{32} \cdot f'_{21} \cdot f'_{13} + f'_{31} \cdot f'_{12} \cdot f'_{23} - \\
&f'_{31} \cdot f'_{22} \cdot f'_{13} - f'_{23} \cdot f'_{32} \cdot f'_{11} - f'_{21} \cdot f'_{12} \cdot f'_{33}
\end{aligned}$$

Therefore, the rank 2 constraint allows us to obtain a third degree polynomial as the one shown in (3.46), where A, B, C, D are the parameters of such polynomial in terms of the fundamental matrices belonging to the null-space of (3.41).

By finding the roots α of such polynomial, we can then find the possible estimations of the fundamental matrix by means of (3.43). Notice that because we have a third degree polynomial, we can then have 1 real root and 2 complex roots, or 3 real roots (which could also be the same root). Because the fundamental matrix is obtained from point correspondences containing real values, then the fundamental matrix must also contain real values. Therefore, the complex roots will always be discarded, and we will just pick the real one, as will generally be the case if the problem is not geometrically degenerate.

It must be noticed that if three different possible solutions can be used for the fundamental matrix, then it means that the problem is **geometrically degenerate** (i.e. not enough separation between centers of cameras as when rotating a camera on the same center, or all world points laying on a plane, which is a degenerate structure) and epipolar geometry is not uniquely defined. In those cases, it is hard to determine which is the true fundamental matrix.

As will be seen later in chapter 4, because we will usually work with image or video sequences, then in cases where a pair of images yields a geometrically degenerate solution for the fundamental matrix $\mathbf{F}$ in (3.46), we will simply ignore that pair of images, and by keeping the same reference image (i.e. left view) we will try with the next image in the sequence (i.e. video frame) that yields a non-degenerate solution.

3.2.2.2.1.- Checking the roots of a third degree polynomial

In order to check more easily whether we have more than one solution to (3.46) or not, then it might be interesting to define a scalar value in the range of 0 to 1 taking into account that the fundamental matrix $\mathbf{F}$ is defined only up to scale, then (3.44) can also be written as:

$$\det(\mathbf{F}' + \alpha \cdot \mathbf{F}'') = \det(\beta \cdot \mathbf{F}' + (1 - \beta) \cdot \mathbf{F}'') = 0 \quad (3.47)$$

Therefore α and β are related as follows,

$$\alpha = \frac{1 - \beta}{\beta} \Rightarrow \beta = \frac{1}{1 + \alpha} \quad (3.48)$$

And then expression (3.46) can also be written in terms of β as,

$$\begin{aligned} \det(\mathbf{F}' + \alpha \cdot \mathbf{F}'') &= A \cdot \alpha^3 + B \cdot \alpha^2 + C \cdot \alpha + D = \\ A \cdot \left(\frac{1 - \beta}{\beta} \right)^3 + B \cdot \left(\frac{1 - \beta}{\beta} \right)^2 + C \cdot \left(\frac{1 - \beta}{\beta} \right) + D &= \\ A \cdot \left(\frac{1 - 3\beta + 3\beta^2 - \beta^3}{\beta^3} \right) + B \cdot \left(\frac{1 - 2\beta + \beta^2}{\beta^2} \right) + C \cdot \left(\frac{1 - \beta}{\beta} \right) + D &= \\ A \cdot (1 - 3\beta + 3\beta^2 - \beta^3) + B \cdot \beta \cdot (1 - 2\beta + \beta^2) + C \cdot \beta^2 \cdot (1 - \beta) + \beta^3 \cdot D &= \\ A - 3A \cdot \beta + 3A \cdot \beta^2 - A \cdot \beta^3 + B \cdot \beta - 2B \cdot \beta^2 + B \cdot \beta^3 + C \cdot \beta^2 - C \cdot \beta^3 + D \cdot \beta^3 &= \\ (-A + B - C + D) \cdot \beta^3 + (3A - 2B + C) \cdot \beta^2 + (-3A + B) \cdot \beta + A &= \\ A' \cdot \beta^3 + B' \cdot \beta^2 + C' \cdot \beta + D' &= 0, \end{aligned} \quad (3.49)$$

where the parameters A', B', C', D' now determine the polynomial in terms of β , and are related to the parameters A, B, C and D of expression (3.46) as follows:

$$\det(\mathbf{F}' + \alpha \mathbf{F}'') = A \cdot \alpha^3 + B \cdot \alpha^2 + C \cdot \alpha + D = A' \cdot \beta^3 + B' \cdot \beta^2 + C' \cdot \beta + D = 0,$$

$$A' = -A + B - C + D$$

$$B' = 3 \cdot A - 2 \cdot B + C$$

$$C' = -3A + B$$

$$D' = A$$

(3.50)

where terms A' , B' , C' and D' won't usually be expanded (i.e. they will be typically computed by means of (3.50), because there will be no gain in computational terms.

Notice that a closed expression for solving a third degree polynomial can be hard to find, hence, in order to find the roots of the polynomial we will use a different approach by means of a **companion matrix** and **eigen-decomposition** (i.e. which is typically implemented using iterative algorithms). It must be noticed that this technique can also be used to find the roots of polynomials of any degree.

Therefore, we will start by indicating how eigen-decomposition can be obtained algebraically, by taking into account the following expression:

$$\mathbf{A} \cdot \mathbf{v} = \lambda \mathbf{v}$$

(3.51)

where $\mathbf{A}$ is an $N \times N$ squared matrix being eigen-decomposed, $\mathbf{v}$ is an eigen-vector and λ is the eigen-value corresponding to the eigen-vector $\mathbf{v}$.

Therefore, expression (3.51) can be rewritten as:

$$\mathbf{A} \cdot \mathbf{v} - \lambda \mathbf{v} = 0 \Rightarrow (\mathbf{A} - \lambda \mathbf{I}_{N \times N}) \cdot \mathbf{v} = 0$$

(3.52)

And consequently, in order to have a solution $\mathbf{v}$ for (3.52) different from the trivial null solution, then matrix $\mathbf{B} = \mathbf{A} - \lambda \mathbf{I}_{N \times N}$ needs to be rank deficient (i.e. singular). Therefore, by means of the singularity constraint on $\mathbf{B}$, we obtain the characteristic polynomial of matrix $\mathbf{A}$, which is,

$$p_A(\lambda) = \det(\mathbf{A} - \lambda \mathbf{I}_{N \times N}) = 0$$

(3.53)

where the roots of this polynomial are the eigen-values of $\mathbf{A}$.

Consequently, if we find the matrix $\mathbf{A}$ with **characteristic polynomial** coincident with the polynomial we want to solve in (3.50), we can find the roots of that polynomial in (3.50) by computing the eigenvalues of matrix $\mathbf{A}$. Such matrix $\mathbf{A}$ is called the companion matrix.

Therefore, for any polynomial of the form:

$$p(\lambda) = c_0 + c_1 \cdot \lambda + \dots + c_{n-1} \cdot \lambda^{n-1} + \lambda^n = 0$$

(3.54)

we want a matrix $\mathbf{A}$ such that (3.53) is equal to (3.54) (i.e. $p_A(\lambda) = p(\lambda)$), therefore, the companion matrix $\mathbf{A}$ can be defined as:

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & \cdots & 0 & -c_0 \\ 1 & 0 & \cdots & 0 & -c_1 \\ 0 & 1 & \cdots & 0 & -c_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & -c_{n-1} \end{bmatrix} \quad (3.55)$$

where it can be proved by the rule of Samus that the determinant in (3.53) using companion matrix $\mathbf{A}$ in (3.55) produces the same expression as in (3.54)

In our case however, we just want to solve a third degree polynomial like,

$$\begin{aligned} p(\alpha) &= \det(\mathbf{F}' + \alpha \mathbf{F}'') = A \cdot \alpha^3 + B \cdot \alpha^2 + C \cdot \alpha + D = 0 \Rightarrow \\ p(\alpha) &= \alpha^3 + \frac{B}{A} \alpha^2 + \frac{C}{A} \alpha + \frac{D}{A} \end{aligned} \quad (3.56)$$

Therefore, by means of (3.55), the companion matrix $\mathbf{C}(\alpha) = \mathbf{A}$ for the polynomial in terms of α is:

$$\mathbf{C}(\alpha) = \mathbf{A} = \begin{bmatrix} 0 & 0 & -c_0 \\ 1 & 0 & -c_1 \\ 0 & 1 & -c_2 \end{bmatrix} = \begin{bmatrix} 0 & 0 & -D/A \\ 1 & 0 & -C/A \\ 0 & 1 & -B/A \end{bmatrix} \quad (3.57)$$

which will be proved next.

Proof 3.3

To prove that this is indeed the companion matrix, we will use (3.53) and use the rule of Samus, to obtain,

$$\begin{aligned} p_A(\lambda) &= \det(\mathbf{A} - \lambda \mathbf{I}_{N \times N}) = \det \left(\begin{bmatrix} -\lambda & 0 & -c_0 \\ 1 & -\lambda & -c_1 \\ 0 & 1 & -c_2 - \lambda \end{bmatrix} \right) = \\ &= (-1)^{(1+3)} \cdot (-c_0) \cdot \begin{vmatrix} 1 & -\lambda \\ 0 & 1 \end{vmatrix} + (-1)^{(2+3)} \cdot (-c_1) \cdot \begin{vmatrix} -\lambda & 0 \\ 0 & 1 \end{vmatrix} + (-1)^{(3+3)} \cdot (-c_2 - \lambda) \cdot \begin{vmatrix} -\lambda & 0 \\ 0 & -\lambda \end{vmatrix} = \\ &= -c_0 \cdot 1 + c_1 \cdot (-\lambda) + (-c_2 - \lambda) \cdot \lambda^2 = -c_0 - c_1 \cdot \lambda - c_2 \cdot \lambda^2 - \lambda^3 = 0 \end{aligned} \quad (3.58)$$

And taking into account that $c_0 = D/A$, $c_1 = C/A$, and $c_2 = B/A$, then (3.58) becomes,

$$p_A(\lambda) = -c_0 - c_1 \cdot \lambda - c_2 \cdot \lambda^2 - \lambda^3 = -\frac{D}{A} - \frac{C}{A} \cdot \lambda - \frac{B}{A} \cdot \lambda^2 - \lambda^3 = 0 \Rightarrow$$

$$D + C \cdot \lambda + B \cdot \lambda^2 + A \cdot \lambda^3 = 0 \quad (3.59)$$

And consequently the proof is finished.

Notice that roots β in polynomial (3.50) having parameters A' , B' , C' and D' can be analogously obtained by means of the companion matrix as well, though, in this case the companion matrix will be:

$$\mathbf{C}(\beta) = \begin{bmatrix} 0 & 0 & -D'/A' \\ 1 & 0 & -C'/A' \\ 0 & 1 & -B'/A' \end{bmatrix} \quad (3.60)$$

Finally, after computing the eigenvalues of companion matrix in (3.60) by means of an iterative process, we will have the roots of polynomial (3.50), and those roots should be real and between 0 and 1. To detect whether epipolar geometry is degenerate or not, we will then need to ensure that only 1 real root (or 3 times the same root) is present, and if more roots are present then epipolar geometry should be discarded as we previously mentioned.

And finally, by using the computed root β , the fundamental matrix $\mathbf{F}$ will be obtained by means of the following expression:

$$\mathbf{F} = \beta \cdot \mathbf{F}' + (1 - \beta) \cdot \mathbf{F}'' \quad (3.61)$$

And that should be enough to estimate the Fundamental matrix with only 7 point correspondences. Notice however, that as in the case of the 8-points algorithm, considerations such as normalization of image point correspondences and Frobenius norm normalization of fundamental matrix, must be taken into account in order to obtain the best possible accuracy due to machine precision. These normalization techniques were described in section 3.2.2.1.1.

3.2.3.- Robust Fundamental matrix computation using RANSAC

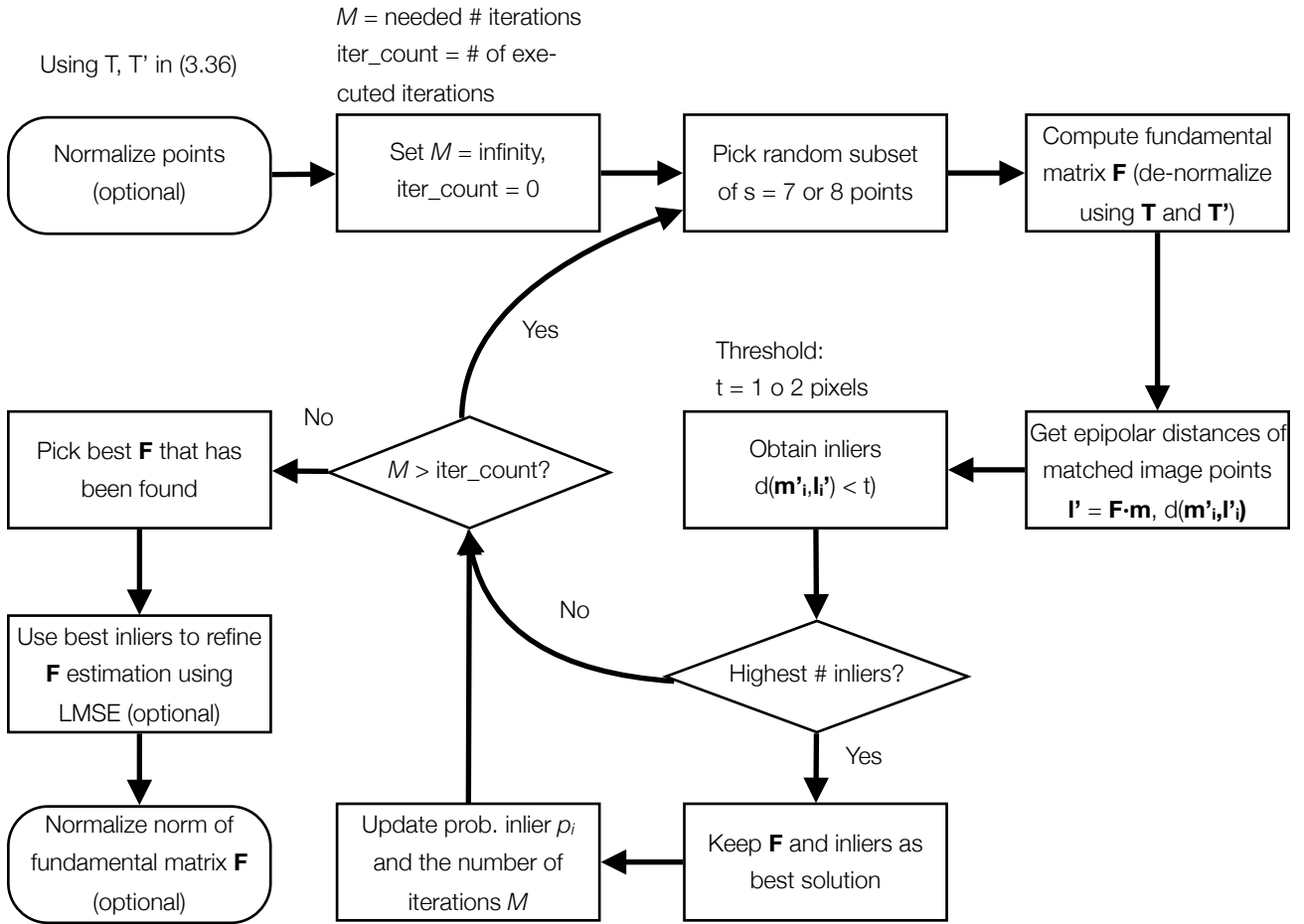
In section 2.5.2, where we explained how to use RANSAC algorithm to estimate camera matrices from world and image points correspondences. However, by using the same idea behind RANSAC, we can also estimate the fundamental matrix in a robust way. Indeed, the RANSAC algorithm can be seen as a general purpose algorithm to discard outliers (up to 50% of the total samples) in any model or structure being estimated.

Therefore, given a large set of N image point correspondences, RANSAC algorithm will randomly select small subsets of s points on each iteration to estimate the fundamental matrix. However, in this case we will need to set s either to $s = 8$ for the 8-point algorithm or to $s = 7$ for the 7-point algorithm. After the fundamental matrix is estimated, then epipolar distance of point correspondences respect to generated epipolar lines will be used to determine whether correspondences are inliers or not. This process will be repeated M iterations until the desired confidence probability c is achieved for a given probability of inlier p_i . Finally, the fundamental matrix $\mathbf{F}$ that has produced the largest support of inliers is chosen, and in order to further refine the results, then the whole set of inliers can be used to estimate $\mathbf{F}$ in LMSE terms. This is all depicted in algorithm 3.1.

Hence, assuming the probability of inlier p_i is fixed, the minimum number of iterations M to execute is expressed exactly as in (2.132):

$$M \geq \frac{\log(1 - c)}{\log(1 - p_i^s)} \quad (3.62)$$

Algorithm 3.1. Robust Fundamental matrix estimation using RANSAC



And as in the RANSAC algorithm for camera estimation, it will always be convenient to use the smallest possible s value (i.e. number of matches on each random subset, which is $s = 7$) in order to obtain the smallest possible number of iterations M .

Also notice that due to machine precision, it will always be interesting to select random points on each subset as uniformly as possible throughout the image, because if points lie too close to each other, then the estimated fundamental matrix $\mathbf{F}$ might become degenerate. In order to achieve this uniformity on point selection, techniques such as tiling or bucketing of the image can be used.

As we saw in section 2.5.2, probability of inlier p_i can be updated as the algorithm iterates in order to correctly estimate this probability, because when the algorithm starts the probability of inlier is usually unknown. However, in order to estimate this probability, we need to set a threshold t to determine whether correspondences are inliers or not for the estimated fundamental matrix $\mathbf{F}$ on a given iteration, by comparing this threshold with an error measure, which in this case is the epipolar distance.

The epipolar distance can be easily computed for a given pair of corresponding image points $\mathbf{m}, \mathbf{m}'$ and a given fundamental matrix $\mathbf{F}$ by taking into account expressions (1.11) and (3.3).

Then, given image point $\mathbf{m}$ on the left view, the generated epipolar line on the right view $\mathbf{l}'$, is by means of (3.3):

$$\mathbf{l}' \sim \mathbf{F} \cdot \mathbf{m} \quad (3.63)$$

And by using (1.11), we then obtain that the epipolar distance of matched point $\mathbf{m}'$ on the right view respect to epipolar line $\mathbf{l}'$ on the right view is:

$$d(\mathbf{m}', \mathbf{l}') = \left| \frac{\mathbf{m}'^T \cdot \mathbf{l}'}{\sqrt{l_1'^2 + l_2'^2}} \right|, \quad \mathbf{l}' = \mathbf{F} \cdot \mathbf{m} \quad (3.64)$$

where sign has been removed respect (1.11) as we don't mind whether matched points are located to one side or the other the epipolar lines.

Therefore, all points having an epipolar distance smaller than provided threshold t (i.e. $d(\mathbf{m}', \mathbf{l}') \leq t$) are considered to be inliers.

As in the case of camera estimation using RANSAC that we saw in section 2.5.2, this threshold t must be chosen empirically and with care, because setting a too strict (i.e. small) threshold will decrease the probability of inlier p_i , while rapidly increasing the number of needed iterations M (especially if the probability of inlier decreases below 50%). On the other hand, setting a too large threshold t will provide poorly estimated fundamental matrices $\mathbf{F}$ on each iteration but small number of needed iterations M , resulting on a poor fundamental matrix after RANSAC execution finishes.

Experimentally, we found that setting thresholds around $t = 1$ or $t = 2$ pixels produces good results, because as will be explained in section 3.2.4, then matches can be corrected for such small amount of errors. Besides, setting larger thresholds is discouraged because of the uncertainty introduced on epipolar geometry as will be seen on section 3.2.8.

3.2.4.- Initial projective pair of camera matrices and their relation to image homographies

In section 2.9, we saw how points laying in different 3D planes are related by homographies. Indeed, we also saw that by using a reference plane Π (which is usually the plane at infinity), points laying in pairs or images can be first transferred by an homography to the reference plane, and then transferred back to the other image in the pair by another homography as in (2.165). Finally, by means of (2.170) and (2.171), which expressed a camera $\mathbf{P}_i$ respect to the first canonical camera $\mathbf{P}_1$, and how affects the change of the reference plane on the homography relating cameras $\mathbf{P}_i$ and $\mathbf{P}_1$, we were able to determine that pairs of corresponding image points in a pair of images lie on epipolar lines, which allowed us to determine the basis for epipolar geometry.

Therefore, given an homography $\mathbf{H}_{ij}^\Pi$ relating points in i^{th} and j^{th} views through the reference plane Π , then we know that such points are related as follows:

$$\mathbf{m}_j \sim \mathbf{H}_{ij}^\Pi \cdot \mathbf{m}_i \quad (3.65)$$

Therefore, by means of expression (3.26) defining epipolar geometry and expression (3.65), we obtain:

$$\mathbf{m}_j^T \cdot \mathbf{F}_{ij} \cdot \mathbf{m}_i = 0 \Rightarrow (\mathbf{H}_{ij}^\Pi \cdot \mathbf{m}_i)^T \cdot \mathbf{F}_{ij} \cdot \mathbf{m}_i = 0 \quad (3.66)$$

where $\mathbf{F}_{ij}$ refers to the fundamental matrix relating view i^{th} to view j^{th} , and $\mathbf{m}_i$, $\mathbf{m}_j$ are image points on views i^{th} and j^{th} respectively.

Also, from expression (3.3) we know that the generated epipolar lines on j^{th} view are:

$$\mathbf{l}_j = \mathbf{F}_{ij} \cdot \mathbf{m}_i \quad (3.67)$$

And such epipolar lines pass through both $\mathbf{m}_j$ and the epipole $\mathbf{e}_{ij}$ (projection of center of camera $\mathbf{P}_i$ on j^{th} image, for two cameras it is usually written as $\mathbf{e}'$). Therefore, such epipolar lines $\mathbf{l}_j$ can also be expressed as:

$$\mathbf{l}_j \sim \mathbf{e}_{ij} \times \mathbf{m}_j = \begin{bmatrix} \mathbf{e}_{ij} \end{bmatrix}_\times \cdot \mathbf{m}_j \quad (3.68)$$

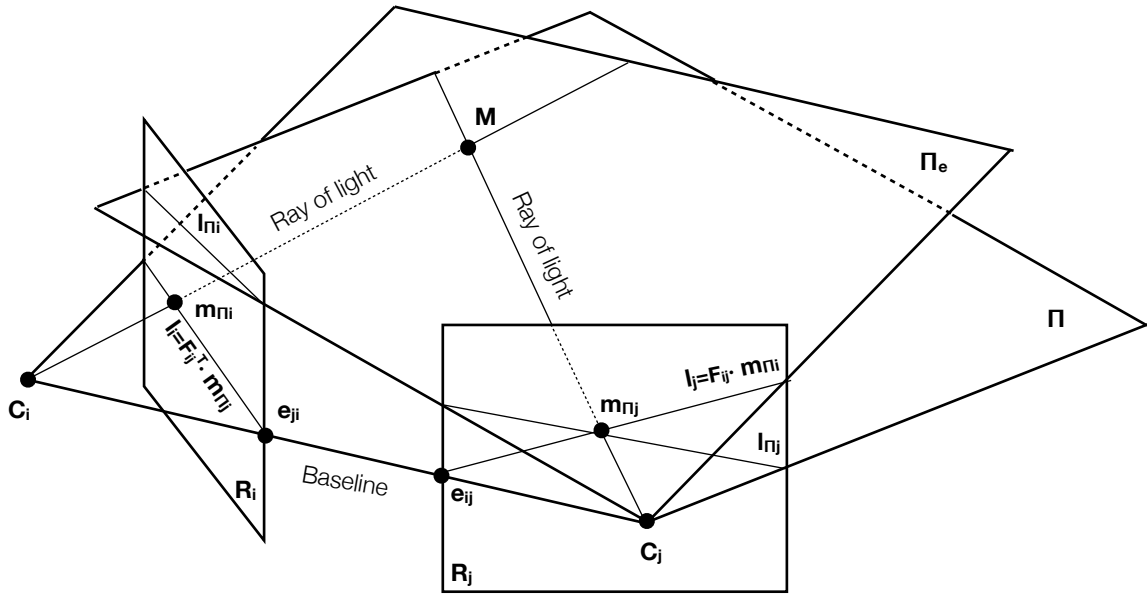


Fig. 3.4. Homography between two views. C_i : i^{th} camera center. C_j : j^{th} camera center. e_{ji} : i^{th} epipole of j^{th} camera. e_{ij} : j^{th} epipole of i^{th} camera. l_i : i^{th} epipolar line. l_j : j^{th} epipolar line. l_{ni} : intersection of plane Π with i^{th} view. l_{nj} : intersection of plane Π with j^{th} view. Π_e : Epipolar plane. Π : arbitrary plane. R_i : i^{th} image plane. R_j : j^{th} image plane. M : 3D world point laying on the intersection of planes Π and Π_e . m_{ni} : i^{th} image point laying on l_i . m_{nj} : j^{th} image point laying on the intersection of l_j and l_{nj} .

Then, by combining (3.66), (3.67) and (3.68) we obtain:

$$\left(\mathbf{H}_{ij}^{\Pi} \cdot \mathbf{m}_i\right)^T \cdot \mathbf{F}_{ij} \cdot \mathbf{m}_i = 0 \Rightarrow \left(\mathbf{H}_{ij}^{\Pi} \cdot \mathbf{m}_i\right)^T \cdot \mathbf{l}_j = 0 \Rightarrow \left(\mathbf{H}_{ij}^{\Pi} \cdot \mathbf{m}_i\right)^T \cdot \left[\mathbf{e}_{ij}\right]_{\times} \cdot \mathbf{m}_j = 0 \quad (3.69)$$

And finally, because the dot product can be commuted, we obtain:

$$\left(\left[\mathbf{e}_{ij}\right]_{\times} \cdot \mathbf{m}_j\right)^T \cdot \mathbf{H}_{ij}^{\Pi} \cdot \mathbf{m}_i = 0 \Rightarrow \mathbf{m}_j^T \cdot \left[\mathbf{e}_{ij}\right]_{\times}^T \cdot \mathbf{H}_{ij}^{\Pi} \cdot \mathbf{m}_i = 0 \quad (3.70)$$

Notice that because $[\mathbf{e}_{ij}]_{\times}$ is skew-symmetric (see expression (1.9), then it satisfies $[\mathbf{e}_{ij}]_{\times}^T = -[\mathbf{e}_{ij}]_{\times}$, therefore, (3.70) becomes:

$$-\mathbf{m}_j^T \cdot \left[\mathbf{e}_{ij}\right]_{\times} \cdot \mathbf{H}_{ij}^{\Pi} \cdot \mathbf{m}_i = 0 \Rightarrow \mathbf{m}_j^T \cdot \left[\mathbf{e}_{ij}\right]_{\times} \cdot \mathbf{H}_{ij}^{\Pi} \cdot \mathbf{m}_i = 0 \quad (3.71)$$

Finally, comparing (3.71) with (3.66),

$$\mathbf{m}_j^T \cdot \mathbf{F}_{ij} \cdot \mathbf{m}_i = \mathbf{m}_j^T \cdot \left[\mathbf{e}_{ij}\right]_{\times} \cdot \mathbf{H}_{ij}^{\Pi} \cdot \mathbf{m}_i = 0 \quad (3.72)$$

Therefore, the fundamental matrix relating i^{th} and j^{th} views can be expressed as:

$$\mathbf{F}_{ij} \sim \left[\mathbf{e}_{ij}\right]_{\times} \cdot \mathbf{H}_{ij}^{\Pi} \quad (3.73)$$

Hence, the fundamental matrix $\mathbf{F}_{ij}$ can be expressed in terms of the homography $\mathbf{H}_{ij}^{\Pi}$ relating i^{th} and j^{th} views through reference plane Π .

On the other hand, by looking at figure 3.4, we can observe that given a line $\mathbf{l}_{nj}$, we can obtain the plane Π as its back-projection. Then if $\mathbf{m}_{ni}$ is an image point obtained as the projection of world point $\mathbf{M}$ that belongs to this plane Π on i^{th} view, then its corresponding epipolar line $\mathbf{l}_j$ on the j^{th} view can be obtained as:

$$\mathbf{l}_j = \mathbf{F}_{ij} \cdot \mathbf{m}_{ni} \quad (3.74)$$

Because the projection on j^{th} view of world point $\mathbf{M}$ belonging to both planes Π and $\Pi\mathbf{e}$ will lie on both lines $\mathbf{l}_j$ and $\mathbf{l}_{nj}$. Then such lines will intersect in such projected image point $\mathbf{m}_{nj}$. Therefore, such image point on the j^{th} view obtained from the intersection of two lines can be found as:

$$\mathbf{m}_{nj} = \mathbf{l}_{nj} \times \mathbf{l}_j = \mathbf{l}_{nj} \times (\mathbf{F}_{ij} \cdot \mathbf{m}_{ni}) = [\mathbf{l}_{nj}]_{\times} \cdot \mathbf{F}_{ij} \cdot \mathbf{m}_{ni} \quad (3.75)$$

where this expression will be true as long as the lines are not coincident and intersect on a single point. To always avoid lines $\mathbf{l}_j$ and $\mathbf{l}_{nj}$ being coincident, we could for instance ensure that is $\mathbf{l}_{nj}$ not an epipolar line. For that reason, we need to ensure that epipole $\mathbf{e}_{ij}$ does not belong to line $\mathbf{l}_{nj}$ by means of expression (1.7). Therefore,

$$\mathbf{l}_{nj}^T \cdot \mathbf{e}_{ij} \neq 0 \quad (3.76)$$

Consequently an obvious choice for line $\mathbf{l}_{nj}$ is the epipole $\mathbf{e}_{ij}$ itself, just as follows:

$$\mathbf{l}_{nj}^T \sim \mathbf{e}_{ij} \quad (3.77)$$

Therefore from (3.75) and (3.77), and analogously to (3.65), the homography $\mathbf{H}_{ij}^{\Pi}$ relating image points $\mathbf{m}_{ni}$, $\mathbf{m}_{nj}$ in i^{th} and j^{th} views through the reference plane Π can be written as:

$$\mathbf{H}_{ij}^{\Pi} = [\mathbf{l}_{nj}]_{\times} \cdot \mathbf{F}_{ij} = [\mathbf{e}_{ij}]_{\times} \cdot \mathbf{F}_{ij} \quad (3.78)$$

Once the expression of the homography relating points on i^{th} and j^{th} views through plane Π is known, we can then use expressions (2.170) and (2.171) to finally obtain the first pair of projective cameras $\mathbf{P}_1$ and $\mathbf{P}_2$, by means of the fundamental matrix relating both views $\mathbf{F}_{12}$ as:

$$\mathbf{P}_1 = [\mathbf{I}_{3 \times 3} | \mathbf{0}_3] \quad (3.79)$$

where $\mathbf{P}_1$ is the canonical camera set at the origin of coordinates and pointing towards the z-axis of coordinates, and $\mathbf{P}_2$ can be written as:

$$\mathbf{P}_2 = [\mathbf{H}_{12}^{REF} | \mathbf{e}_{12}] = [\mathbf{H}_{12}^{\Pi} + \mathbf{e}_{12} \cdot \boldsymbol{\pi}^T | \mathbf{e}_{12}] = [[\mathbf{e}_{12}]_{\times} \cdot \mathbf{F}_{12} + \mathbf{e}_{12} \cdot \boldsymbol{\pi}^T | \mathbf{e}_{12}] \quad (3.80)$$

where (3.79) is defined up to scale, $\boldsymbol{\pi}$ are the first three coordinates of any arbitrary plane Π , $\mathbf{e}_{12}$ is the epipole on the second view corresponding to the first camera projected on that view, and $\mathbf{F}_{12}$ is the fundamental matrix relating both views.

As we saw in previous sections, for the two views case (i.e. left and right views), usually $\mathbf{F}_{12}$ is known as $\mathbf{F}$, and $\mathbf{e}_{12}$ is known as $\mathbf{e}'$. Therefore, in summary the first pair of projective cameras can be written as:

$$\begin{aligned} \mathbf{P}_1 &= [\mathbf{I}_{3 \times 3} | \mathbf{0}_3] \\ \mathbf{P}_2 &= [[\mathbf{e}']_{\times} \cdot \mathbf{F} + \mathbf{e}' \cdot \boldsymbol{\pi}^T | \lambda \cdot \mathbf{e}'] \end{aligned} \quad (3.81)$$

where $\mathbf{\pi}$ defines the director vector of the arbitrary plane Π , and λ defines the overall scale factor.

Notice that the expression (3.81) can only be used to initiate a reference projective framework, other consecutive cameras (i.e. $\mathbf{P}_3$, $\mathbf{P}_4$, etc.) cannot be obtained using this procedure. Also notice that as we will show next, cameras $\mathbf{P}_1$ and $\mathbf{P}_2$ are defined in a projective stratum only.

3.2.4.1.- Proving that the initial pair of projective cameras is not unique

Given the pair of cameras $P1$ and $P2$ that we found in expression (3.81), we will prove that such pair is not unique because it is defined in a projective stratum. Therefore, any projective transformation T will produce the same epipolar geometry.

Indeed, in (2.153), we saw that regardless of the projective transformation, we always obtain the same projected points on the image plane, assuming then that the world points $\mathbf{M}$ get transformed into $\mathbf{M}'$, and the camera matrix $\mathbf{P}$ transforms into $\mathbf{P}'$, as shown below:

$$\begin{aligned}\mathbf{m} &= \mathbf{P} \cdot \mathbf{M} = (\mathbf{P} \cdot \mathbf{T}^{-1}) \cdot (\mathbf{T} \cdot \mathbf{M}) = \mathbf{P}' \cdot \mathbf{M}' \Rightarrow \\ \mathbf{M}' &= \mathbf{T} \cdot \mathbf{M} \\ \mathbf{P}' &= \mathbf{P} \cdot \mathbf{T}^{-1}\end{aligned}\tag{3.82}$$

Consequently, because the fundamental matrix $\mathbf{F}$ is only estimated from corresponding image points by means of expression (3.26), which we have proved that remain invariant to projective transformations, then so will the fundamental matrix $\mathbf{F}$.

Indeed, by using expression (3.11), we can also prove that the fundamental matrix remains invariant, as well as the epipoles. Therefore, taking into account transformed camera matrices $\mathbf{P}_1' = \mathbf{P}_1 \cdot \mathbf{T}^{-1}$, $\mathbf{P}_2' = \mathbf{P}_2 \cdot \mathbf{T}^{-1}$, then the fundamental matrix can be computed as:

$$\begin{aligned}\mathbf{F} &= [\mathbf{e}'_{12}]_{\times} \cdot (\mathbf{P}_1' \cdot \mathbf{P}_2'^{\dagger}) = [\mathbf{e}'_{12}]_{\times} \cdot (\mathbf{P}_1 \cdot \mathbf{T}^{-1} \cdot (\mathbf{P}_2 \cdot \mathbf{T}^{-1})^{\dagger}) = [\mathbf{e}'_{12}]_{\times} \cdot (\mathbf{P}_1 \cdot \mathbf{T}^{-1} \cdot (\mathbf{T}^{-1})^{-1} \cdot \mathbf{P}_2^{\dagger}) = \\ &[\mathbf{e}'_{12}]_{\times} \cdot (\mathbf{P}_1 \cdot \mathbf{T}^{-1} \cdot \mathbf{T} \cdot \mathbf{P}_2^{\dagger}) = [\mathbf{e}'_{12}]_{\times} \cdot (\mathbf{P}_1 \cdot \mathbf{P}_2^{\dagger})\end{aligned}\tag{3.83}$$

Therefore, the fundamental matrix will remain invariant to any projective transformation as long as the epipole remains invariant. However, the epipole $\mathbf{e}_{12}$ can be seen as the projection of the center $\mathbf{C}_1$ of the 1st camera $\mathbf{P}_1$ into the second view by means of $\mathbf{P}_2$. Then by means of (3.82) we obtain:

$$\mathbf{e}'_{12} = \mathbf{P}_2' \cdot \mathbf{C}_1' = \mathbf{P}_2 \cdot \mathbf{T}^{-1} \cdot \mathbf{T} \cdot \mathbf{C}_1 = \mathbf{P}_2 \cdot \mathbf{C}_1 = \mathbf{e}_{12}\tag{3.84}$$

Consequently, since the epipole remains invariant, so does the fundamental matrix.

$$\mathbf{F} = [\mathbf{e}'_{12}]_{\times} \cdot (\mathbf{P}_1' \cdot \mathbf{P}_2'^{\dagger}) = [\mathbf{e}_{12}]_{\times} \cdot (\mathbf{P}_1 \cdot \mathbf{P}_2^{\dagger})\tag{3.85}$$

3.2.5.- Improving point correspondences

In chapter 2 we described the camera model and methods to triangulate corresponding image points $\mathbf{m}$, $\mathbf{m}'$ into 3D world points by means of the DLT algorithm if a pair of cameras $\mathbf{P}$ and $\mathbf{P}'$ are provided.

Because our goal was to obtain 3D world points and camera setup just from image correspondences (i.e. cameras are not known beforehand), then with the methods shown in chapter 2 we were unable to obtain a pair of cameras needed for point triangulation just from correspondences of image points.

However, in section 3.2.4, we have proved that for any given fundamental matrix $\mathbf{F}$, we can then obtain a pair of projective cameras $\mathbf{P}$ and $\mathbf{P}'$ that generates the epipolar geometry shown in $\mathbf{F}$ (see expression 3.81). And because we showed in section 3.2.2 that the fundamental matrix $\mathbf{F}$ can be computed just from image point correspondences $\mathbf{m}$, $\mathbf{m}'$ (i.e. by means

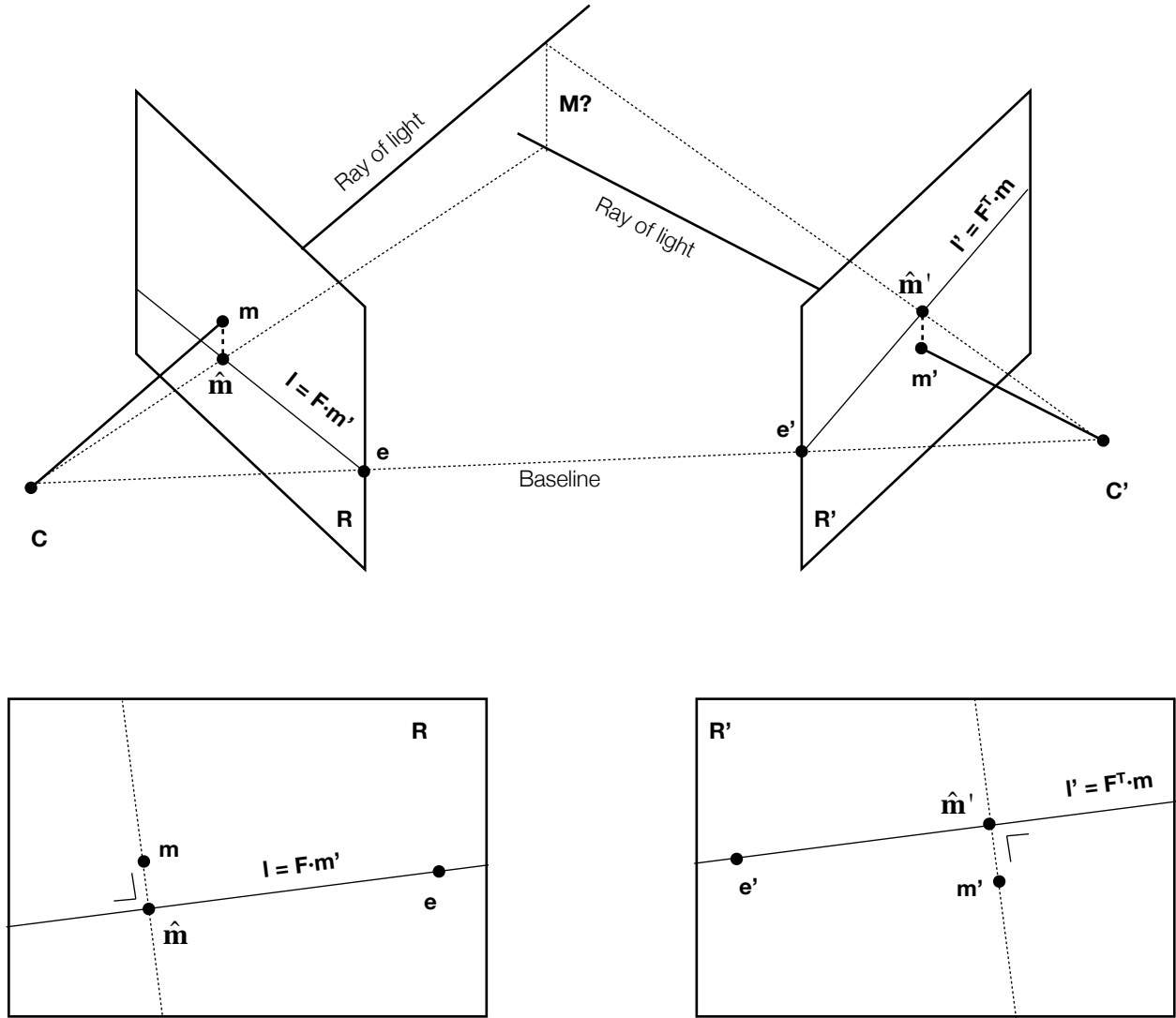


Figure 3.5. **Top:** Skewed back-projected rays of light because matched image points m and m' do not lie exactly on epipolar lines, therefore world point M cannot be exactly found. **Bottom-left:** image point m and the epipolar line l generated by image point m' on the other view. Image point m must be corrected to lie on epipolar line l , this can be done by choosing the point $\hat{m}$ located at shortest perpendicular distance from line l that also lies in such line. **Bottom-right:** image point m' and the epipolar line l' generated by image point m on the first view. Image point m' must be corrected to lie on epipolar line l' , this can be done by choosing the point $\hat{m}'$ located at shortest perpendicular distance from line l' that also lies in such line.

of the 7-point or 8-point algorithms or using RANSAC), then we are now able to obtain triangulated 3D world points just from image points correspondences m, m' in a **projective** stratum.

It is important to notice that triangulated world points will be in an arbitrary projective stratum because expression (3.81) generates pairs of cameras in such stratum.

In section 2.6 we saw how to triangulate points by means of the DLT algorithm, however, it was always assumed that matched image points m, m' always generated a perfect epipolar geometry (i.e. exactly laid in epipolar lines), which resulted in back-projected rays that intersected in a given 3D world point. However, due to small errors in match estimations this might not be exactly true. Indeed, the epipolar geometry (i.e. fundamental matrix F) could also be estimated with a certain error, although we will always assumed that the error is located in image point matches. In such case were small amounts of

error are present, then the back-projection of rays of light might be skewed (i.e. rays of light won't lie on the same epipolar plane Π_e), and therefore such rays won't intersect on a given 3D world point, as depicted in figure 3.5.

We saw in section 2.6, that when more image points corresponding to the same world point $\mathbf{M}$ are provided in more views, then the DLT algorithm can find a LMSE solution for such world point $\mathbf{M}$, however, initially the fundamental matrix $\mathbf{F}$ can only generate a single pair of cameras $\mathbf{P}$ and $\mathbf{P}'$, and with such number of views and in the presence of noise in point correspondence, the DLT algorithm cannot find an exact solution, only an LMSE one that due to the low number of views will be very poor in terms of error.

It is then a good strategy to try to force corresponding image points to perfectly lie on the estimated epipolar geometry. That way the DLT algorithm can find an accurate solution when triangulating points.

Notice that the correction must be done on the image planes (i.e. the only known data which is invariant to projective transformations) because generated points and cameras will be on an **arbitrary projective stratum**. In other words, attempting to find world points as the midpoint of the common perpendicular to two back-projected rays in space, as in figure 3.5, won't be valid in any general projective stratum (although it would be valid in a metric one).

Therefore, it is desired to find corrections for image point matches $\mathbf{m}$, $\mathbf{m}'$ for a known epipolar geometry or fundamental matrix $\mathbf{F}$ (which can be robustly computed from such point matches) so that we obtain a projective invariant method for improving accuracy of world point triangulations by means of DLT. For that reason, we will define the following cost function to be minimized:

$$C(\mathbf{m}, \mathbf{m}') = d(\mathbf{m}, \hat{\mathbf{m}})^2 + d(\mathbf{m}', \hat{\mathbf{m}}')^2, \quad \hat{\mathbf{m}}' \cdot \mathbf{F} \cdot \hat{\mathbf{m}} = 0 \quad (3.86)$$

where $\mathbf{m}$ and $\mathbf{m}'$ are the corresponding image points that can be found with any disparity estimation method shown in chapter 9, and $\hat{\mathbf{m}}$ and $\hat{\mathbf{m}}'$ are the corrected corresponding image points, which as can be seen on (3.86) are in complete accordance with the epipolar geometry generated by the fundamental matrix $\mathbf{F}$, and $d(\cdot, \cdot)$ represents the euclidean distance between two image points using their inhomogeneous representation.

In order to minimize the cost function in (3.86) we will provide two methods that will be shown next. Both of these methods are based on finding the closest points to $\mathbf{m}$ and $\mathbf{m}'$ that exactly lie on their respective epipolar lines $\mathbf{l}$ and $\mathbf{l}'$, as shown in figure 3.5.

3.2.5.1.- The Sampson approximation

The first method to obtain appropriate correction of corresponding image points $\mathbf{m}$ and $\mathbf{m}'$ is based on the Sampson error measure. This measure is a first-order approximation to the geometric correction obtained from minimizing the cost function in (3.86), therefore, as any first-order approximation, it will only be valid for small amounts of error in point correspondences (i.e. typically 1 or 2 pixels).

In order to obtain the Sampson error measure, we will first define vector $\mathbf{X}$ containing the inhomogeneous coordinates of a pair of corresponding points $\mathbf{m}$ and $\mathbf{m}'$. Therefore,

$$\mathbf{X} = (\tilde{\mathbf{m}}, \tilde{\mathbf{m}}')^T = (x, y, x', y')^T \quad (3.87)$$

Hence, the cost function in (3.86) can be written as:

$$C(\mathbf{X}) = C(\mathbf{m}, \mathbf{m}') \quad (3.88)$$

Therefore, by taking a first-order approximation of the cost function in terms of $\mathbf{X}$ using Taylor series, we obtain that for small displacements (i.e. corrections) $\delta \mathbf{x}$ the cost function becomes:

$$C(\mathbf{X} + \delta_{\mathbf{x}}) = C(\mathbf{X}) + \frac{\partial C(\mathbf{X})}{\partial \mathbf{X}} \cdot \delta_{\mathbf{x}} \quad (3.89)$$

where the displacement or correction $\delta_{\mathbf{x}}$ can be defined as:

$$\delta_{\mathbf{x}} = \hat{\mathbf{X}} - \mathbf{X} \quad (3.90)$$

where $\hat{\mathbf{X}}$ is the corrected version of $\mathbf{X}$ containing the corrected inhomogeneous coordinates of matched points, as follows:

$$\hat{\mathbf{X}} = (\hat{\mathbf{m}}, \hat{\mathbf{m}}')^T = (\hat{x}, \hat{y}, \hat{x}', \hat{y}')^T \quad (3.91)$$

From (3.89) we could further define ε as the cost for the points without correction and $\mathbf{J}$ as the Jacobian of the cost function, therefore:

$$\begin{aligned} \varepsilon &= C(\mathbf{X}) \\ \mathbf{J} &= \frac{\partial C(\mathbf{X})}{\partial \mathbf{X}} \end{aligned} \quad (3.92)$$

It must be noticed that taking into account (3.89) and (3.90) where $\delta_{\mathbf{x}}$ is the correction to be done to obtain the optimal corrected points, then the cost function evaluated at the corrected points is zero as shown below,

$$\begin{aligned} C(\mathbf{X} + \delta_{\mathbf{x}}) &= C(\mathbf{X} + \hat{\mathbf{X}} - \mathbf{X}) = C(\hat{\mathbf{X}}) = C(\hat{\mathbf{m}}, \hat{\mathbf{m}}') = \\ d(\hat{\mathbf{m}}, \hat{\mathbf{m}})^2 + d(\hat{\mathbf{m}}', \hat{\mathbf{m}}')^2 &= 0 + 0 = 0 \end{aligned} \quad (3.93)$$

Consequently, from (3.89), (3.92) and (3.93) we obtain:

$$C(\mathbf{X} + \delta_{\mathbf{x}}) = C(\mathbf{X}) + \frac{\partial C(\mathbf{X})}{\partial \mathbf{X}} \cdot \delta_{\mathbf{x}} = 0 \quad \Rightarrow \quad \varepsilon + \mathbf{J} \cdot \delta_{\mathbf{x}} = 0 \quad (3.94)$$

Therefore, we want to find the smallest correction $\delta_{\mathbf{x}}$ that we can do, subject to the constraint in (3.94). In other words, we want to find $\delta_{\mathbf{x}}$ that minimizes $\|\delta_{\mathbf{x}}\|^2$ subject to $\varepsilon + \mathbf{J} \cdot \delta_{\mathbf{x}} = 0$. This can be done by means of Lagrange multipliers.

Notice that $\|\delta_{\mathbf{x}}\|^2$ the square of the norm of $\delta_{\mathbf{x}}$ will be minimized because its derivative can be found more easily.

3.2.5.1.1.- An overview of Lagrange multipliers

Lagrange multipliers is a technique to minimize or maximize a given function $f(\mathbf{x})$ subject to the constraint $g(\mathbf{x}) = c$. Then, considering the function $f(\mathbf{x})$ as a surface and the constraint $g(\mathbf{x}) = c$ as a given curve, as depicted in figure 3.6. Then, by looking at the contour curves of $f(\mathbf{x})$ we can observe that as we traverse the curve $g(\mathbf{x}) = c$, the function $f(\mathbf{x})$ increases or decreases, and therefore the curve $g(\mathbf{x}) = c$ intersects the contour curves of $f(\mathbf{x})$, whereas when $g(\mathbf{x}) = c$ is tangent to the contour curve of $f(\mathbf{x})$, then $f(\mathbf{x})$ does not increase neither decrease (i.e. is a local maximum or minimum). It can then be observed graphically that when the contour curve of $f(\mathbf{x})$ and $g(\mathbf{x}) = c$ are tangent, then their respective gradients are parallel. Therefore,

$$\nabla_{\mathbf{x}} f = -\lambda \nabla_{\mathbf{x}} g \quad (3.95)$$

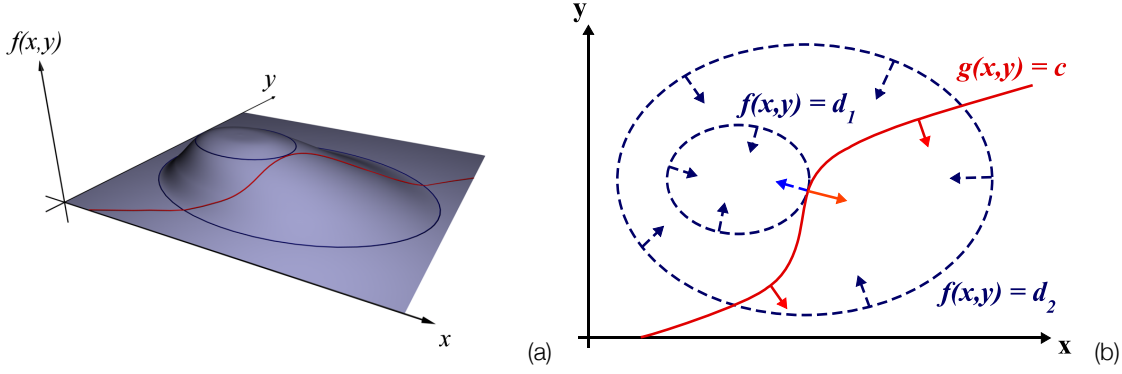


Fig. 3.6. Finding maximum of function $f(\mathbf{x})$ subject to constraint $g(\mathbf{x}) = c$ using Lagrange multipliers. (a) 3D plot of the surface $f(\mathbf{x})$ and constraint $g(\mathbf{x}) = c$. (b) Contour curves of $f(\mathbf{x})$ and $g(\mathbf{x}) = c$. The desired maximum is obtained when gradients of both curves are parallel.

where λ is the Lagrange multiplier that accounts for differences in the magnitudes of both gradients, including sign, although both have the same direction (i.e. are parallel).

Consequently, expression (3.95) as well as the function to minimize/maximize and the provided constraint can be written in a single equation by means of the following auxiliary function:

$$\Lambda(\mathbf{x}, \lambda) = f(\mathbf{x}) + \lambda(g(\mathbf{x}) - c) \quad (3.96)$$

and then solving the gradient of (3.96) equated to zero:

$$\nabla_{\mathbf{x}, \lambda} \Lambda(\mathbf{x}, \lambda) = 0 \quad (3.97)$$

Indeed, we can prove that (3.97) indeed contains the provided constraint when deriving by λ , and forces gradients to be parallel when deriving by vector $\mathbf{x}$, as follows:

$$\nabla_{\mathbf{x}, \lambda} \Lambda(\mathbf{x}, \lambda) = 0 \Rightarrow \begin{cases} \nabla_{\mathbf{x}} \Lambda(\mathbf{x}, \lambda) = 0 \Rightarrow \nabla_{\mathbf{x}} f + \lambda \nabla_{\mathbf{x}} g = 0 \\ \nabla_{\lambda} \Lambda(\mathbf{x}, \lambda) = 0 \Rightarrow g(\mathbf{x}) - c = 0 \end{cases} \quad (3.98)$$

where it is observed in (3.98) that the first equation is equivalent to (3.95), and the second equation is the constraint that we provided.

Therefore, to find $\delta_{\mathbf{x}}$ that minimizes $f(\delta_{\mathbf{x}}) = \|\delta_{\mathbf{x}}\|^2$ subject to the constraint $g(\delta_{\mathbf{x}}) = \boldsymbol{\varepsilon} + \mathbf{J} \cdot \delta_{\mathbf{x}} = 0$ using Lagrange multipliers, we set the following equation:

$$\Lambda(\delta_{\mathbf{x}}, \lambda) = \|\delta_{\mathbf{x}}\|^2 + \lambda^T (\mathbf{J} \cdot \delta_{\mathbf{x}} + \boldsymbol{\varepsilon}) = \delta_{\mathbf{x}}^T \cdot \delta_{\mathbf{x}} + \lambda^T (\mathbf{J} \cdot \delta_{\mathbf{x}} + \boldsymbol{\varepsilon}) \quad (3.99)$$

where now λ is a vector of Lagrange multipliers.

And by equating the gradient of (3.99) to zero, we obtain:

$$\nabla_{\delta_{\mathbf{x}}, \lambda} \Lambda(\delta_{\mathbf{x}}, \lambda) = 0 \Rightarrow \begin{cases} \frac{\partial \Lambda(\delta_{\mathbf{x}}, \lambda)}{\partial \delta_{\mathbf{x}}} = 0 \Rightarrow 2\delta_{\mathbf{x}}^T + \lambda^T \cdot \mathbf{J} = 0 \\ \frac{\partial \Lambda(\delta_{\mathbf{x}}, \lambda)}{\partial \lambda} = 0 \Rightarrow \mathbf{J} \cdot \delta_{\mathbf{x}} + \boldsymbol{\varepsilon} = 0 \end{cases} \quad (3.100)$$

where the first equation ensures that gradients are parallel, and the second equation is the provided constraint $g(\delta_x) = \epsilon + \mathbf{J} \cdot \delta_x = 0$.

By using the first equation of (3.100) we can then obtain the amount of correction δ_x to be used over $\mathbf{X}$ to optimize the cost function in (3.89) as follows,

$$2\delta_x^T + \lambda^T \cdot \mathbf{J} = 0 \Rightarrow \delta_x^T = -\frac{\lambda^T}{2} \cdot \mathbf{J} \Rightarrow \delta_x = -\frac{1}{2} \mathbf{J}^T \cdot \lambda \quad (3.101)$$

And substituting the amount of correction δ_x of (3.101) in expression (3.94), then the constraint $g(\delta_x) = \epsilon + \mathbf{J} \cdot \delta_x = 0$ becomes:

$$\epsilon + \mathbf{J} \cdot \delta_x = 0 \Rightarrow \epsilon = -\mathbf{J} \cdot \delta_x = \frac{1}{2} \mathbf{J} \cdot \mathbf{J}^T \cdot \lambda \quad (3.102)$$

where we recall that ϵ is the cost for the points without correction.

Next, by means of expression (3.102), we can then solve the vector of Lagrange multipliers λ as:

$$\epsilon = \frac{1}{2} \mathbf{J} \cdot \mathbf{J}^T \cdot \lambda \Rightarrow \lambda = 2 \left(\mathbf{J} \cdot \mathbf{J}^T \right)^{-1} \cdot \epsilon \quad (3.103)$$

And finally, substituting (3.103) into the amount of correction shown in expression (3.101), we obtain that the correction to be applied to points is:

$$\begin{aligned} \delta_x &= -\frac{1}{2} \mathbf{J}^T \cdot \lambda = -\frac{1}{2} \mathbf{J}^T \cdot 2 \left(\mathbf{J} \cdot \mathbf{J}^T \right)^{-1} \cdot \epsilon \Rightarrow \\ \delta_x &= -\mathbf{J}^T \left(\mathbf{J} \cdot \mathbf{J}^T \right)^{-1} \cdot \epsilon \end{aligned} \quad (3.104)$$

Therefore, by evaluating function $f(\delta_x) = \|\delta_x\|^2$ being minimized, we obtain,

$$\|\delta_x\|^2 = \delta_x^T \cdot \delta_x = \epsilon^T \left(\mathbf{J} \cdot \mathbf{J}^T \right)^{-T} \cdot \mathbf{J} \cdot \mathbf{J}^T \left(\mathbf{J} \cdot \mathbf{J}^T \right)^{-1} \cdot \epsilon = \epsilon^T \left(\mathbf{J} \cdot \mathbf{J}^T \right)^{-T} \cdot \epsilon \quad (3.105)$$

And because $\mathbf{J} \cdot \mathbf{J}^T$ is a symmetric matrix, then its inverse is also symmetric, and consequently $(\mathbf{J} \cdot \mathbf{J}^T)^{-T} = (\mathbf{J} \cdot \mathbf{J}^T)^{-1}$. Therefore,

$$\|\delta_x\|^2 = \epsilon^T \left(\mathbf{J} \cdot \mathbf{J}^T \right)^{-1} \cdot \epsilon = \epsilon^T \left(\mathbf{J} \cdot \mathbf{J}^T \right)^{-1} \cdot \epsilon = \frac{\epsilon^T \cdot \epsilon}{\mathbf{J} \cdot \mathbf{J}^T} \quad (3.106)$$

which is the expression of the **Sampson Error**, that can be seen as a similar measure to Mahalanobis distance.

Consequently, by using expressions (3.90) and (3.104), we obtain that the corrected points are:

$$\hat{\mathbf{X}} = \mathbf{X} + \delta_x = \mathbf{X} - \mathbf{J}^T \left(\mathbf{J} \cdot \mathbf{J}^T \right)^{-1} \cdot \epsilon \quad (3.107)$$

Then, it only remains to define the Jacobian of the cost function $\mathbf{J}$ and ϵ the cost function for a pair of matched points $\mathbf{m}$ and $\mathbf{m}'$ to finally obtain the corrected points in terms of the fundamental matrix $\mathbf{F}$.

As we saw in expression (3.92), the cost function for a pair of matched points ϵ can be defined as:

$$\varepsilon = C(\mathbf{X}) = C(\mathbf{m}, \mathbf{m}') = d(\mathbf{m}, \hat{\mathbf{m}})^2 + d(\mathbf{m}', \hat{\mathbf{m}}')^2 \quad (3.108)$$

Then, by looking at figure 3.5, we can observe that the corrected points are the points laying on epipolar lines that have shortest distance to points $\mathbf{m}$ and $\mathbf{m}'$ respectively, therefore, as we saw in section 1.2.4, these distances measures can be seen as the shortest distance of $\mathbf{m}$ respect to epipolar line $\mathbf{l}$, and the shortest distance of $\mathbf{m}'$ respect to epipolar line $\mathbf{l}'$. Because we are minimizing the cost function $C(\mathbf{X})$, then for simplification, it will be enough to define such distances up to scale by means of expression (1.10), consequently (3.108) can be rewritten as:

$$\varepsilon = C(\mathbf{X}) = C(\mathbf{m}, \mathbf{m}') = d(\mathbf{m}, \mathbf{l}) + d(\mathbf{m}', \mathbf{l}') = \mathbf{m}^T \cdot \mathbf{l} + \mathbf{m}'^T \cdot \mathbf{l}' \quad (3.109)$$

and the solution found for (3.108) by means of Lagrange multipliers will also minimize (3.109) because both are equivalent.

Then by using the expressions (3.3) and (3.5) of generated epipolar lines $\mathbf{l}$, $\mathbf{l}'$ for a given image point $\mathbf{m}$ or $\mathbf{m}'$ and a given fundamental matrix $\mathbf{F}$, (3.109) becomes:

$$\varepsilon = C(\mathbf{X}) = C(\mathbf{m}, \mathbf{m}') = d(\mathbf{m}, \mathbf{l}) + d(\mathbf{m}', \mathbf{l}') = \mathbf{m}^T \cdot \mathbf{l} + \mathbf{m}'^T \cdot \mathbf{l}' = \mathbf{m}^T \cdot \mathbf{F}^T \cdot \mathbf{m}' + \mathbf{m}'^T \cdot \mathbf{F} \cdot \mathbf{m} \quad (3.110)$$

And because the cost $\mathbf{m}'^T \cdot \mathbf{F} \cdot \mathbf{m}$ is a **scalar value**, then the scalar product can be commuted obtaining:

$$\varepsilon = C(\mathbf{X}) = \mathbf{m}^T \cdot \mathbf{F}^T \cdot \mathbf{m}' + \mathbf{m}'^T \cdot \mathbf{F} \cdot \mathbf{m} = 2 \cdot \mathbf{m}'^T \cdot \mathbf{F} \cdot \mathbf{m} \quad (3.111)$$

And again, because is enough to define the cost function to minimize up to scale in order to find the optimal solution, we obtain:

$$\varepsilon = C(\mathbf{X}) \sim \mathbf{m}'^T \cdot \mathbf{F} \cdot \mathbf{m} \quad (3.112)$$

On the other hand, again by expression (3.92), the Jacobian of the cost function is defined as:

$$\begin{aligned} \mathbf{J} &= \frac{\partial C(\mathbf{X})}{\partial \mathbf{X}} = \frac{\partial \varepsilon}{\partial (\mathbf{m}, \mathbf{m}')} = \frac{\partial (\mathbf{m}'^T \cdot \mathbf{F} \cdot \mathbf{m})}{\partial (\mathbf{m}, \mathbf{m}')} = \\ &= \frac{\partial}{\partial (x, y, x', y')} \left(\begin{bmatrix} x' & y' & 1 \end{bmatrix} \cdot \mathbf{F} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \right) = \\ &= \left[(\mathbf{F}^T \cdot \mathbf{m}')_1, (\mathbf{F}^T \cdot \mathbf{m}')_2, (\mathbf{F} \cdot \mathbf{m})_1, (\mathbf{F} \cdot \mathbf{m})_2 \right] \end{aligned} \quad (3.113)$$

where $(\mathbf{F} \cdot \mathbf{m})_i = \mathbf{l}'_i$ indicates i^{th} element of epipolar line $\mathbf{l}' = \mathbf{F} \cdot \mathbf{m}$, and conversely $(\mathbf{F}^T \cdot \mathbf{m}')_i = \mathbf{l}_i$ indicates i^{th} element of epipolar line $\mathbf{l} = \mathbf{F}^T \cdot \mathbf{m}'$.

Therefore, the Jacobian of the cost function $\mathbf{J}$ is a 1x4 vector, and consequently the product $\mathbf{J} \cdot \mathbf{J}^T$ is the following scalar value:

$$\mathbf{J} \cdot \mathbf{J}^T = (\mathbf{F} \cdot \mathbf{m})_1^2 + (\mathbf{F} \cdot \mathbf{m})_2^2 + (\mathbf{F}^T \cdot \mathbf{m}')_1^2 + (\mathbf{F}^T \cdot \mathbf{m}')_2^2 \quad (3.114)$$

And consequently, using (3.112), (3.113) and (3.114) the corrected points in (3.107) become:

$$\hat{\mathbf{X}} = \mathbf{X} + \delta_{\mathbf{x}} = \mathbf{X} - \mathbf{J}^T (\mathbf{J} \cdot \mathbf{J}^T)^{-1} \cdot \boldsymbol{\varepsilon} = \mathbf{X} - \frac{\boldsymbol{\varepsilon} \cdot \mathbf{J}^T}{\mathbf{J} \cdot \mathbf{J}^T} \Rightarrow$$

$$\begin{bmatrix} \hat{x} \\ \hat{y} \\ \hat{x}' \\ \hat{y}' \end{bmatrix} = \begin{bmatrix} x \\ y \\ x' \\ y' \end{bmatrix} - \frac{\mathbf{m}'^T \cdot \mathbf{F} \cdot \mathbf{m}}{(\mathbf{F} \cdot \mathbf{m})_1^2 + (\mathbf{F} \cdot \mathbf{m})_2^2 + (\mathbf{F}^T \cdot \mathbf{m}')_1^2 + (\mathbf{F}^T \cdot \mathbf{m}')_2^2} \cdot \begin{bmatrix} (\mathbf{F}^T \cdot \mathbf{m}')_1 \\ (\mathbf{F}^T \cdot \mathbf{m}')_2 \\ (\mathbf{F} \cdot \mathbf{m})_1 \\ (\mathbf{F} \cdot \mathbf{m})_2 \end{bmatrix} \quad (3.115)$$

which defines the **Sampson correction** for any pair of matched homogeneous image points $\mathbf{m}$ and $\mathbf{m}'$ whose inhomogeneous coordinates are (x, y, x', y') .

3.2.5.2.- The Gold Standard method

The Gold Standard method is an optimal minimization algorithm for the cost function defined in expression (3.86) under the assumption of gaussianity of errors in matched points $\mathbf{m}$ and $\mathbf{m}'$.

Recalling expression (3.86) where we defined the cost function to minimize subject to the constraint of full epipolar compliance, as shown below,

$$C(\mathbf{m}, \mathbf{m}') = d(\mathbf{m}, \hat{\mathbf{m}})^2 + d(\mathbf{m}', \hat{\mathbf{m}}')^2, \quad \hat{\mathbf{m}}' \cdot \mathbf{F} \cdot \hat{\mathbf{m}} = 0 \quad (3.116)$$

Then we seek the pair of corrected matched image points $\hat{\mathbf{m}}, \hat{\mathbf{m}}'$ that minimizes such expression.

In figure 3.5, we saw that such corrected image points must lie on the generated epipolar lines $\mathbf{l}$ and $\mathbf{l}'$, and consequently the distance between the erroneously matched image points $\mathbf{m}$ and $\mathbf{m}'$ respect to their corrected versions $\hat{\mathbf{m}}, \hat{\mathbf{m}}'$, is exactly the shortest distance of image points $\mathbf{m}, \mathbf{m}'$ to their respective epipolar lines $\mathbf{l}$ and $\mathbf{l}'$. Therefore, the cost function in (3.116) now becomes:

$$C(\mathbf{m}, \mathbf{m}') = d(\mathbf{m}, \hat{\mathbf{m}})^2 + d(\mathbf{m}', \hat{\mathbf{m}}')^2 = d(\mathbf{m}, \mathbf{l})^2 + d(\mathbf{m}', \mathbf{l}')^2 \quad (3.117)$$

where distances $d(\mathbf{m}, \mathbf{l})$ and $d(\mathbf{m}', \mathbf{l}')$ where defined in expression (1.11).

Because we look for unknown corrected points $\mathbf{m}$ and $\mathbf{m}'$, that lie in epipolar lines $\mathbf{l}$ and $\mathbf{l}'$ in order to reduce as much as possible the cost function in (3.117), then in principle such epipolar lines $\mathbf{l}$ and $\mathbf{l}'$ are unknown because the corrected image points are also unknown. Therefore, we will parameterize both lines as $\mathbf{l}(t)$ and $\mathbf{l}'(t)$. Consequently, we will finally minimize the cost function shown below:

$$C(\mathbf{m}, \mathbf{m}') = d(\mathbf{m}, \mathbf{l}(t))^2 + d(\mathbf{m}', \mathbf{l}'(t))^2 \quad (3.118)$$

As will be seen, this minimization problem can be computed by solving the real roots of a polynomial of degree 6, as long as the points $\mathbf{m}$ and $\mathbf{m}'$ do not lie on one or both epipoles (i.e. because then points cannot be triangulated).

Then in order to simplify the analysis, we will assume that both non-corrected image points $\mathbf{m}$ and $\mathbf{m}'$ are moved to the origin $(0,0,1)^T$ using homogeneous coordinates. This can be achieved by using the following transformation matrices $\mathbf{T}$ and $\mathbf{T}'$,

$$\mathbf{T} = \begin{bmatrix} 1 & -x \\ & 1 & -y \\ & & 1 \end{bmatrix}, \quad \mathbf{T}' = \begin{bmatrix} 1 & -x' \\ & 1 & -y' \\ & & 1 \end{bmatrix}, \quad \bar{\mathbf{m}} = \mathbf{T} \cdot \mathbf{m}, \quad \bar{\mathbf{m}}' = \mathbf{T}' \cdot \mathbf{m}' \quad (3.119)$$

Therefore, on transforming image points into $\mathbf{T} \cdot \mathbf{m}$ and $\mathbf{T}' \cdot \mathbf{m}'$, then as in section 3.2.2.1.1 the fundamental matrix $\mathbf{F}$ must be replaced by:

$$\hat{\mathbf{F}} = \mathbf{T}'^T \cdot \mathbf{F} \cdot \mathbf{T}^{-1} \quad (3.120)$$

Also, similarly to the Hartley's rectification method shown in section 9.2.1, we could assume that the epipoles can be rotated to the x-axis at points $\mathbf{e} = (1, 0, f)^T$ and $\mathbf{e}' = (1, 0, f')^T$ respectively, by means of a rigid transformations $\mathbf{R}$ and $\mathbf{R}'$. Notice that when $f = 0$ then epipole $\mathbf{e}$ is at infinity, and analogously when $f' = 0$ then epipole $\mathbf{e}'$ is also at infinity.

Hence, we compute the epipoles $\mathbf{e} = (e_1, e_2, e_3)^T$ and $\mathbf{e}' = (e'_1, e'_2, e'_3)^T$ from $\mathbf{F}$ by means of $\mathbf{e}^T \cdot \mathbf{F} = 0$ and $\mathbf{F} \cdot \mathbf{e} = 0$. And then because epipoles are defined up to scale in homogeneous coordinates, normalize so that $e_1^2 + e_2^2 = 1$ and $e'_1{}^2 + e'_2{}^2 = 1$. In that case the rigid transformations $\mathbf{R}$ and $\mathbf{R}'$ can be written as:

$$\mathbf{R} = \begin{bmatrix} e_1 & e_2 & \\ -e_2 & e_1 & \\ & & 1 \end{bmatrix}, \quad \mathbf{R}' = \begin{bmatrix} e'_1 & e'_2 & \\ -e'_2 & e'_1 & \\ & & 1 \end{bmatrix} \quad (3.121)$$

where both transformations $\mathbf{R}$ and $\mathbf{R}'$ are rotation matrices that fulfill $\mathbf{R}^{-1} = \mathbf{R}^T$ and $\det(\mathbf{R}) = 1$.

Then, in this case the rotated epipoles become:

$$\begin{aligned} \bar{\mathbf{e}} = \mathbf{R} \cdot \mathbf{e} &= (e_1^2 + e_2^2, 0, e_3)^T \sim (1, 0, f)^T \Rightarrow f = \frac{e_3}{e_1^2 + e_2^2} \\ \bar{\mathbf{e}}' = \mathbf{R}' \cdot \mathbf{e}' &= (e_1'^2 + e_2'^2, 0, e_3')^T \sim (1, 0, f')^T \Rightarrow f' = \frac{e_3'}{e_1'^2 + e_2'^2} \end{aligned} \quad (3.122)$$

And after applying transformations $\mathbf{R}$ and $\mathbf{R}'$, then the fundamental matrix in (3.120) must be replaced again, as follows:

$$\bar{\mathbf{F}} = \mathbf{R}'^T \cdot \hat{\mathbf{F}} \cdot \mathbf{R}^{-1} = \mathbf{R}' \cdot \hat{\mathbf{F}} \cdot \mathbf{R}^T = \mathbf{R}' \cdot \mathbf{T}'^T \cdot \mathbf{F} \cdot \mathbf{T}^{-1} \cdot \mathbf{R}^T \quad (3.123)$$

Therefore, after transformations $\mathbf{T}$, $\mathbf{T}'$ and rotations $\mathbf{R}$, $\mathbf{R}'$ we have:

$$\begin{aligned} \bar{\mathbf{e}} &= (1, 0, f)^T \\ \bar{\mathbf{e}}' &= (1, 0, f')^T \\ \bar{\mathbf{m}} = \bar{\mathbf{m}}' &= (0, 0, 1)^T \end{aligned} \quad (3.124)$$

where for the purpose of minimizing the cost in (3.118), the problem still remains the same.

Because of expressions (3.12) and (3.18) where the epipoles $\mathbf{e}$ and $\mathbf{e}'$ represent the right and left null-spaces of fundamental matrix $\mathbf{F}$ respectively, then we can assert that for the transformed epipoles and fundamental matrix, it is true that

$$\begin{aligned}\bar{\mathbf{F}} \cdot \bar{\mathbf{e}} = \mathbf{0}_3 &\Rightarrow \bar{\mathbf{F}} \cdot (1, 0, f)^T = \mathbf{0}_3 \\ \bar{\mathbf{e}}^T \cdot \bar{\mathbf{F}} = \mathbf{0}_3^T &\Rightarrow (1, 0, f') \cdot \bar{\mathbf{F}} = \mathbf{0}_3\end{aligned}\quad (3.125)$$

Then, by assuming that the tranformed fundamental matrix is a 3x3 matrix like the following one:

$$\bar{\mathbf{F}} = \begin{bmatrix} e & g & h \\ i & a & b \\ j & c & d \end{bmatrix} \quad (3.126)$$

we obtain the following equations from (3.125) and (3.126),

$$\begin{aligned}\bar{\mathbf{F}} \cdot \bar{\mathbf{e}} = \mathbf{0}_3 &\Rightarrow \begin{bmatrix} e & g & h \\ i & a & b \\ j & c & d \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \\ f \end{bmatrix} = \begin{bmatrix} e + f \cdot h \\ i + f \cdot b \\ j + f \cdot d \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \\ \bar{\mathbf{e}}^T \cdot \bar{\mathbf{F}} = \mathbf{0}_3^T &\Rightarrow \begin{bmatrix} 1 & 0 & f' \end{bmatrix} \cdot \begin{bmatrix} e & g & h \\ i & a & b \\ j & c & d \end{bmatrix} = (e + f' \cdot j, g + f' \cdot c, h + f' \cdot d) = (0, 0, 0)\end{aligned}\quad (3.127)$$

Therefore from (3.127) we obtain,

$$e = -f \cdot h \quad (3.128)$$

$$i = -f \cdot b \quad (3.129)$$

$$j = -f \cdot d \quad (3.130)$$

$$e = -f' \cdot j \quad (3.131)$$

$$g = -f' \cdot c \quad (3.132)$$

$$h = -f' \cdot d \quad (3.133)$$

And finally, by combining (3.133) into (3.128) we obtain that e value is:

$$e = f \cdot f' \cdot d \quad (3.134)$$

Then, in such case, after substituting (3.129-3.134) into (3.126), we obtain that the transformed fundamental matrix must have the following special form:

$$\bar{\mathbf{F}} = \begin{bmatrix} f \cdot f' \cdot d & -f' \cdot c & -f' \cdot d \\ -f \cdot b & a & b \\ -f \cdot d & c & d \end{bmatrix} \quad (3.135)$$

Therefore, an epipolar line $\mathbf{l}(t)$ on the first image passing through the corrected point that we want to find $\hat{\mathbf{m}} = (0, t, 1)^T$ and the rotated epipole $(1, 0, f)^T$ can be written by means of (1.8) as:

$$\mathbf{l}(t) = \hat{\mathbf{m}} \times \bar{\mathbf{e}} = (0, t, 1) \times (1, 0, f) = (t \cdot f, 1, -t) \quad (3.136)$$

And so, using expressions (1.11), (3.124) and (3.136), the shortest distance from the transformed measured image point $\bar{\mathbf{m}}$ to epipolar line $\mathbf{l}(t)$ that is a term of the cost function in (3.118) can be expressed as:

$$d(\bar{\mathbf{m}}, \mathbf{l}(t))^2 = \frac{(\bar{\mathbf{m}}^T \cdot \mathbf{l}(t))^2}{\mathbf{l}(t)_1^2 + \mathbf{l}(t)_2^2} = \frac{((0, 0, 1)(t \cdot f, 1, -t)^T)^2}{(t \cdot f)^2 + 1^2} = \frac{t^2}{1 + (t \cdot f)^2} \quad (3.137)$$

Then, by means of expression (3.3), we obtain that the epipolar line $\mathbf{l}'(t)$ on the second image generated by the point we want to find $\hat{\mathbf{m}}$ can be expressed as:

$$\mathbf{l}'(t) = \bar{\mathbf{F}} \cdot \hat{\mathbf{m}} = \begin{bmatrix} f \cdot f' \cdot d & -f' \cdot c & -f' \cdot d \\ -f \cdot b & a & b \\ -f \cdot d & c & d \end{bmatrix} \begin{bmatrix} 0 \\ t \\ 1 \end{bmatrix} = \begin{bmatrix} -f'(ct + d) \\ at + b \\ ct + d \end{bmatrix} \quad (3.138)$$

Therefore, using again expressions (1.11) and (3.11), then the shortest distance from transformed measured point $\bar{\mathbf{m}}'$ to epipolar line $\mathbf{l}'(t)$ that is also the second term of the cost function in (3.118) can be analogously expressed as:

$$d(\bar{\mathbf{m}}', \mathbf{l}'(t))^2 = \frac{(\bar{\mathbf{m}}'^T \cdot \mathbf{l}'(t))^2}{\mathbf{l}'(t)_1^2 + \mathbf{l}'(t)_2^2} = \frac{((0, 0, 1)(-f'(ct + d), at + b, ct + d)^T)^2}{(-f'(ct + d))^2 + (at + b)^2} \Rightarrow \quad (3.139)$$

$$d(\bar{\mathbf{m}}', \mathbf{l}'(t))^2 = \frac{(ct + d)^2}{(at + b)^2 + f'^2 \cdot (ct + d)^2}$$

Consequently, by combining expressions (3.137) and (3.139) into the cost function in (3.118), and taking into account that after transformations $\mathbf{T}$, $\mathbf{T}'$ and rotations $\mathbf{R}$, $\mathbf{R}'$ the minimization problem remains the same; then the cost function in (3.118) can be written as the following polynomial in terms of t ,

$$s(t) = C(\mathbf{m}, \mathbf{m}') = d(\mathbf{m}, \mathbf{l}(t))^2 + d(\mathbf{m}', \mathbf{l}'(t))^2 = d(\bar{\mathbf{m}}, \mathbf{l}(t))^2 + d(\bar{\mathbf{m}}', \mathbf{l}'(t))^2 \Rightarrow \quad (3.140)$$

$$s(t) = \frac{t^2}{1 + (t \cdot f)^2} + \frac{(ct + d)^2}{(at + b)^2 + f'^2 \cdot (ct + d)^2}$$

In order to minimize the cost function (ie. polynomial) in expression (3.140), we then only need to equate its derivative to zero, to find local minima/maxima, and then evaluate the polynomial $s(t)$ on each possible local minima/maxima to find the global minima. Therefore, the derivative of $s(t)$ is:

$$s'(t) = \frac{\partial s(t)}{\partial t} = \frac{2t}{(1 + f^2 \cdot t^2)^2} - \frac{2(a \cdot d - b \cdot c)(at + b)(ct + d)}{((at + b)^2 + f'^2 \cdot (ct + d)^2)^2} = 0 \quad (3.141)$$

And then, by setting a common denominator, then the condition in (3.141) becomes:

$$g(t) = t \left((at + b)^2 + f^2 (ct + d)^2 \right)^2 - (ad - bc)(at + b)(ct + d)(1 + f^2 t^2)^2 = 0 \quad (3.142)$$

And by expanding $g(t)$, the polynomial in (3.142) becomes:

$$g(t) = t \left((at + b)^2 + f^2 (ct + d)^2 \right)^2 - (ad - bc)(at + b)(ct + d)(1 + f^2 t^2)^2 = 0 \quad (3.143)$$

$$g(t) = t \left((a^2 + c^2 f^2) t^2 + 2(ab + cd f^2) t + b^2 + d^2 f^2 \right)^2 - (ad - bc)(at + b)(ct + d)(1 + f^2 t^2)^2 = 0 \quad (3.144)$$

$$g(t) = (a^2 + c^2 f^2)^2 t^5 + 4(a^2 + c^2 f^2)(ab + cd f^2) t^4 + 2((a^2 + c^2 f^2)(b^2 + d^2 f^2) + 2(ab + cd f^2)^2) t^3 + 4(ab + cd f^2)(b^2 + d^2 f^2) t^2 + (b^2 + d^2 f^2)^2 t - (ad - bc)(at + b)(ct + d)(1 + f^2 t^2)^2 = 0 \quad (3.145)$$

$$g(t) = (a^2 + c^2 f^2)^2 t^5 + 4(a^2 + c^2 f^2)(ab + cd f^2) t^4 + 2((a^2 + c^2 f^2)(b^2 + d^2 f^2) + 2(ab + cd f^2)^2) t^3 + 4(ab + cd f^2)(b^2 + d^2 f^2) t^2 + (b^2 + d^2 f^2)^2 t - ((a^2 d - abc) t + ab d - b^2 c)(ct + d)(1 + f^2 t^2)^2 = 0 \quad (3.146)$$

$$g(t) = (a^2 + c^2 f^2)^2 t^5 + 4(a^2 + c^2 f^2)(ab + cd f^2) t^4 + 2((a^2 + c^2 f^2)(b^2 + d^2 f^2) + 2(ab + cd f^2)^2) t^3 + 4(ab + cd f^2)(b^2 + d^2 f^2) t^2 + (b^2 + d^2 f^2)^2 t - ((a^2 d - abc) c t^2 + ((a^2 d - abc) d + (ab d - b^2 c) c) t + (ab d - b^2 c) d)(1 + 2 f^2 t^2 + f^4 t^4) = 0 \quad (3.147)$$

$$\begin{aligned}
g(t) = & (a^2 + c^2 \cdot f'^2)^2 \cdot t^5 + 4(a^2 + c^2 \cdot f'^2)(a \cdot b + c \cdot d \cdot f'^2) \cdot t^4 + \\
& 2\left((a^2 + c^2 \cdot f'^2)(b^2 + d^2 \cdot f'^2) + 2(a \cdot b + c \cdot d \cdot f'^2)^2\right) \cdot t^3 + \\
& 4(a \cdot b + c \cdot d \cdot f'^2)(b^2 + d^2 \cdot f'^2) \cdot t^2 + (b^2 + d^2 \cdot f'^2)^2 \cdot t \\
& - (a^2 \cdot d - a \cdot b \cdot c) \cdot c \cdot f^4 \cdot t^6 - \left((a^2 \cdot d - a \cdot b \cdot c) \cdot d + (a \cdot b \cdot d - b^2 \cdot c) \cdot c\right) \cdot f^4 \cdot t^5 \\
& - \left(2(a^2 \cdot d - a \cdot b \cdot c) \cdot c \cdot f^2 + (a \cdot b \cdot d - b^2 \cdot c) \cdot d \cdot f^4\right) \cdot t^4 \\
& - 2\left((a^2 \cdot d - a \cdot b \cdot c) \cdot d + (a \cdot b \cdot d - b^2 \cdot c) \cdot c\right) \cdot f^2 \cdot t^3 \\
& - \left((a^2 \cdot d - a \cdot b \cdot c) \cdot c + 2(a \cdot b \cdot d - b^2 \cdot c) \cdot d \cdot f^2\right) \cdot t^2 \\
& - \left((a^2 \cdot d - a \cdot b \cdot c) \cdot d + (a \cdot b \cdot d - b^2 \cdot c) \cdot c\right) \cdot t - (a \cdot b \cdot d - b^2 \cdot c) \cdot d = 0
\end{aligned} \tag{3.148}$$

$$\begin{aligned}
g(t) = & - (a^2 \cdot d - a \cdot b \cdot c) \cdot c \cdot f^4 \cdot t^6 + \\
& \left((a^2 + c^2 \cdot f'^2)^2 - \left((a^2 \cdot d - a \cdot b \cdot c) \cdot d + (a \cdot b \cdot d - b^2 \cdot c) \cdot c\right) \cdot f^4\right) \cdot t^5 + \\
& \left(4(a^2 + c^2 \cdot f'^2)(a \cdot b + c \cdot d \cdot f'^2) - \left(2(a^2 \cdot d - a \cdot b \cdot c) \cdot c \cdot f^2 + (a \cdot b \cdot d - b^2 \cdot c) \cdot d \cdot f^4\right)\right) \cdot t^4 + \\
& 2\left((a^2 + c^2 \cdot f'^2)(b^2 + d^2 \cdot f'^2) + 2(a \cdot b + c \cdot d \cdot f'^2)^2 - \left((a^2 \cdot d - a \cdot b \cdot c) \cdot d + (a \cdot b \cdot d - b^2 \cdot c) \cdot c\right) \cdot f^2\right) \cdot t^3 + \\
& \left(4(a \cdot b + c \cdot d \cdot f'^2)(b^2 + d^2 \cdot f'^2) - \left((a^2 \cdot d - a \cdot b \cdot c) \cdot c + 2(a \cdot b \cdot d - b^2 \cdot c) \cdot d \cdot f^2\right)\right) \cdot t^2 + \\
& \left((b^2 + d^2 \cdot f'^2)^2 - \left((a^2 \cdot d - a \cdot b \cdot c) \cdot d + (a \cdot b \cdot d - b^2 \cdot c) \cdot c\right) \cdot t\right) \cdot t \\
& - (a \cdot b \cdot d - b^2 \cdot c) \cdot d = 0
\end{aligned}$$

(3.149)

And finally $g(t)$ can be expressed as the following polynomial of degree 6,

$$g(t) = A \cdot t^6 + B \cdot t^5 + C \cdot t^4 + D \cdot t^3 + E \cdot t^2 + F \cdot t + G \tag{3.150}$$

where the parameters A, B, C, D, E, F and G of the polynomial are:

$$A = - (a^2 \cdot d - a \cdot b \cdot c) \cdot c \cdot f^4 \tag{3.151}$$

$$B = \left((a^2 + c^2 \cdot f'^2)^2 - \left((a^2 \cdot d - a \cdot b \cdot c) \cdot d + (a \cdot b \cdot d - b^2 \cdot c) \cdot c\right) \cdot f^4\right) \tag{3.152}$$

$$C = \left(4(a^2 + c^2 \cdot f'^2)(a \cdot b + c \cdot d \cdot f'^2) - \left(2(a^2 \cdot d - a \cdot b \cdot c) \cdot c \cdot f^2 + (a \cdot b \cdot d - b^2 \cdot c) \cdot d \cdot f^4\right)\right) \tag{3.153}$$

$$D = 2 \left((a^2 + c^2 \cdot f'^2) (b^2 + d^2 f'^2) + 2(a \cdot b + c \cdot d \cdot f'^2)^2 - ((a^2 \cdot d - a \cdot b \cdot c) \cdot d + (a \cdot b \cdot d - b^2 \cdot c) \cdot c) \cdot f'^2 \right) \quad (3.154)$$

$$E = \left(4(a \cdot b + c \cdot d \cdot f'^2) (b^2 + d^2 f'^2) - ((a^2 \cdot d - a \cdot b \cdot c) \cdot c + 2(a \cdot b \cdot d - b^2 \cdot c) \cdot d \cdot f'^2) \right) \quad (3.155)$$

$$F = \left((b^2 + d^2 f'^2)^2 - ((a^2 \cdot d - a \cdot b \cdot c) \cdot d + (a \cdot b \cdot d - b^2 \cdot c) \cdot c) \right) \quad (3.156)$$

$$G = -(a \cdot b \cdot d - b^2 \cdot c) \cdot d \quad (3.157)$$

Then, the polynomial $g(t)$ can be solved by finding the eigenvalues of its companion matrix, as we saw in section 3.2.2.2.1. Hence, by means of (3.55), the companion matrix of polynomial $g(t)$ in terms of its parameters A, B, C, D, E, F and G, is:

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & -G/A \\ 1 & 0 & 0 & 0 & 0 & -F/A \\ 0 & 1 & 0 & 0 & 0 & -E/A \\ 0 & 0 & 1 & 0 & 0 & -D/A \\ 0 & 0 & 0 & 1 & 0 & -C/A \\ 0 & 0 & 0 & 0 & 1 & -B/A \end{bmatrix} \quad (3.158)$$

Once the roots t_i , $i = 1 \dots 6$ of the polynomial are found, then the cost function (i.e. polynomial $s(t)$) in (3.140) is evaluated only on each real root (i.e imaginary ones are ignored) t_i , $i = 1 \dots 6$ to find the root $t_{\min}$ that produces a global minimum on $s(t_{\min})$. Notice that to avoid determining whether roots are completely real or not, only their real part is taken into account when evaluating $s(t)$.

Then, for such root $t_{\min}$ that produces a global minimum of cost function $s(t)$, epipolar lines $\mathbf{l}(t_{\min})$ and $\mathbf{l}'(t_{\min})$ can be evaluated using expressions (3.136) and (3.138) to obtain the optimal epipolar lines where the rectified image points must lie.

Such rectified points can be found as the closest points on these lines to the origin, because points $\mathbf{m}$ and $\mathbf{m}'$ have been transformed to lie at $(0,0,1)^T$. Indeed, from (3.137) and (3.139) we can obtain such distances, although we won't need them.

Hence, as in figure 3.5, the coordinates of the corrected and transformed point $\bar{\mathbf{m}}$ laying on epipolar line $\mathbf{l}(t_{\min})$ can be found by setting another line $\mathbf{l}_p(t_{\min})$ perpendicular to epipolar line $\mathbf{l}(t_{\min})$ that passes through transformed and measured point $\bar{\mathbf{m}}$.

Therefore, because $\mathbf{l}(t_{\min})$ and $\mathbf{l}_p(t_{\min})$ are perpendicular, it must be true that the product of their slopes is -1. Hence, using expression (3.136), we can arbitrarily set $\mathbf{l}_p(t_{\min})$ as:

$$\mathbf{l}(t_{\min}) = (t_{\min} \cdot f, 1, -t_{\min}) \Rightarrow \mathbf{l}_p(t_{\min}) = (-1, t_{\min} \cdot f, c_p) \quad (3.159)$$

where it can be proved that the product of the slopes of $\mathbf{l}(t_{\min})$ and $\mathbf{l}_p(t_{\min})$ is -1 (i.e. $(-t_{\min} \cdot f/1) \cdot (1/t_{\min} \cdot f) = -1$), and c_p is a free parameter that will be set to force line $\mathbf{l}_p(t_{\min})$ to pass through transformed measured point $\bar{\mathbf{m}}$. Therefore, by means of expression (1.7) and taking into account that measured points have been transformed to lie in the origin as in (3.124) we have:

$$\bar{\mathbf{m}}^T \cdot \mathbf{l}_p(t_{\min}) = 0 \Rightarrow (0, 0, 1) \cdot (-1, t_{\min} \cdot f, c_p) = 0 \Rightarrow c_p = 0 \quad (3.160)$$

And consequently line $\mathbf{l}_p(t_{\min})$ perpendicular to epipolar line is already defined.

Therefore, the point intersecting both lines $\mathbf{l}(t_{\min})$ and $\mathbf{l}_p(t_{\min})$ is the corrected and transformed point $\bar{\hat{\mathbf{m}}}$ we are looking for.

Such point could be solved setting a linear system of equations using expression (1.7) and indicating that the point intersects both lines, but it can also be more simply solved as the cross product of the vectors containing lines $\mathbf{l}(t_{\min})$ and $\mathbf{l}_p(t_{\min})$, just as follows:

$$\begin{aligned} \bar{\hat{\mathbf{m}}} &= \mathbf{l}(t_{\min}) \times \mathbf{l}_p(t_{\min}) = [\mathbf{l}(t_{\min})]_{\times} \cdot \mathbf{l}_p(t_{\min}) = \\ &= (t_{\min} \cdot f, 1, -t_{\min}) \times (-1, t_{\min} \cdot f, c_p) = \\ &= (t_{\min} \cdot f, 1, -t_{\min}) \times (-1, t_{\min} \cdot f, 0) = \\ &= \begin{bmatrix} t_{\min} \cdot f \\ 1 \\ -t_{\min} \end{bmatrix}_{\times} \begin{bmatrix} -1 \\ t_{\min} \cdot f \\ 0 \end{bmatrix} = \begin{bmatrix} 0 & t_{\min} & 1 \\ -t_{\min} & 0 & -t_{\min} \cdot f \\ -1 & t_{\min} \cdot f & 0 \end{bmatrix} \cdot \begin{bmatrix} -1 \\ t_{\min} \cdot f \\ 0 \end{bmatrix} = \begin{bmatrix} -t_{\min}^2 \cdot f \\ t_{\min} \\ 1 + (t_{\min} \cdot f)^2 \end{bmatrix} \end{aligned} \quad (3.161)$$

Analogously, the corrected and transformed point for the second view can be found by setting a line $\mathbf{l}'_p(t_{\min})$ passing through measured and transformed image point $\bar{\mathbf{m}}'$ which is perpendicular line to epipolar line $\mathbf{l}'(t_{\min})$.

Therefore, using expression (3.138) that determines the epipolar line $\mathbf{l}'(t_{\min})$, then its perpendicular line can arbitrarily be set to (as long as their product of slopes is -1):

$$\mathbf{l}'(t_{\min}) = \begin{bmatrix} -f'(ct + d) \\ at + b \\ ct + d \end{bmatrix} \Rightarrow \mathbf{l}'_p(t_{\min}) = \begin{bmatrix} -at - b \\ -f'(ct + d) \\ c'_p \end{bmatrix} \quad (3.162)$$

where as we saw in (3.135), a, b, c and d are elements of the transformed fundamental matrix, f' is the last homogeneous component of the transformed epipole on the second view as in (3.122), and as in expression (3.159), c'_p is a parameter that can be set to force $\mathbf{l}'_p(t_{\min})$ to pass through measured and transformed image point $\bar{\mathbf{m}}'$ by using again expressions (1.7) and (3.124). Therefore,

$$\bar{\mathbf{m}}'^T \cdot \mathbf{l}'_p(t_{\min}) = 0 \Rightarrow \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} -at - b \\ -f'(ct + d) \\ c'_p \end{bmatrix} = c'_p = 0 \Rightarrow c'_p = 0 \quad (3.163)$$

Consequently, as in expression (3.161), both perpendicular lines $\mathbf{l}'(t_{\min})$ and $\mathbf{l}_p'(t_{\min})$ are defined, and the corrected and transformed image point $\bar{\mathbf{m}}'$ on the second view lies on the intersection of both lines. Such intersection can be found as the cross product of the vectors of both lines and using expressions (3.162) and (3.163), just as follows:

$$\begin{aligned}\bar{\mathbf{m}}' &= \mathbf{l}'(t_{\min}) \times \mathbf{l}_p'(t_{\min}) = [\mathbf{l}'(t_{\min})]_{\times} \cdot \mathbf{l}_p'(t_{\min}) = \\ & \begin{bmatrix} -f'(ct+d) \\ at+b \\ ct+d \end{bmatrix}_{\times} \begin{bmatrix} -at-b \\ -f'(ct+d) \\ 0 \end{bmatrix} = \\ & \begin{bmatrix} 0 & -(ct+d) & at+b \\ ct+d & 0 & f'(ct+d) \\ -(at+b) & -f'(ct+d) & 0 \end{bmatrix} \begin{bmatrix} -(at+b) \\ -f'(ct+d) \\ 0 \end{bmatrix} = \begin{bmatrix} f'(ct+d)^2 \\ -(at+b)(ct+d) \\ (at+b)^2 + f'^2(ct+d)^2 \end{bmatrix}\end{aligned}\tag{3.164}$$

where again a , b , c and d are elements of the transformed fundamental matrix, and f' is the last homogeneous component of the transformed epipole on the second view as in (3.122).

Hence, once we have the corrected and transformed points $\bar{\mathbf{m}}$, and $\bar{\mathbf{m}}'$, we need to undo the transformations $\mathbf{T}$, $\mathbf{T}'$ and $\mathbf{R}$, $\mathbf{R}'$ that we saw in (3.119) and (3.121) as follows:

$$\begin{aligned}\hat{\mathbf{m}} &= \mathbf{T}^{-1} \cdot \mathbf{R}^T \cdot \bar{\mathbf{m}} \\ \hat{\mathbf{m}}' &= \mathbf{T}'^{-1} \cdot \mathbf{R}'^T \cdot \bar{\mathbf{m}}'\end{aligned}\tag{3.165}$$

where $\hat{\mathbf{m}}$ and $\hat{\mathbf{m}}'$ are the optimally corrected and corresponding image points that can be used for point triangulation using the methods that we saw in section 2.6.

3.2.6.- Comparison between Fundamental matrices

In section 3.2.1.1 we saw that for a given pair of cameras $\mathbf{P}$ and $\mathbf{P}'$ it is possible to compute the fundamental matrix $\mathbf{F}$ that spans their epipolar geometry by means of expression (3.11).

Therefore, by manually calibrating a pair of cameras, as in section 2.5, using calibration grids to retrieve their respective camera matrices $\mathbf{P}$ and $\mathbf{P}'$, we can then compute their corresponding fundamental matrix $\mathbf{F}_{\text{calib}}$ as well, which can be considered a ground-truth fundamental matrices.

As we have previously mentioned, our goal however, is to estimate the fundamental matrices $\mathbf{F}$ just by image point correspondences (see chapter 9 to see how to estimate these correspondences), as we will attempt to retrieve 3D structure without any prior knowledge on camera locations (i.e camera calibrations). Therefore, provided a pair of calibrated cameras with their corresponding ground-truth fundamental matrix $\mathbf{F}_{\text{calib}}$, it can be useful to validate whether the methods shown in section 3.2.2 actually provide good estimates of the fundamental matrix $\mathbf{F}$ just by means of image point correspondences $\mathbf{m}$ and $\mathbf{m}'$.

However, to validate whether the estimated fundamental matrices $\mathbf{F}$ are good or not, we will need first to define measures of comparison between pairs of fundamental matrices, as will be shown next.

3.2.6.1.- Frobenius norm of difference

The simplest method to obtain a metric of comparison between a pair of matrices summarized in a single scalar value, can consist on just the retrieving the Frobenius norm of their difference. However, as we saw in section 3.2.1, that the fundamental matrix relates image points $\mathbf{m}$ and $\mathbf{m}'$ using homogeneous coordinates, and consequently **is only defined up to scale** (i.e it has 8 DOF plus the rank 2 constraint, which results on 7 DOF).

Therefore, to compare a pair of fundamental matrices that can differ in their scale using the Frobenius norm of their difference, it seems logical to first normalize them. Normalization of each fundamental matrix to be compared can be done by means of their corresponding Frobenius norms to ensure that both have the same scale up to sign (i.e. norm is always positive, but fundamental matrices can differ in sign as well because they are only defined up to scale). Therefore, the normalized fundamental matrices are:

$$\hat{\mathbf{F}} = \frac{\mathbf{F}}{\|\mathbf{F}\|}, \quad \hat{\mathbf{F}}_{\text{calib}} = \frac{\hat{\mathbf{F}}_{\text{calib}}}{\|\hat{\mathbf{F}}_{\text{calib}}\|} \quad (3.166)$$

And consequently, the similarity distance measure can be defined as:

$$d_F(\mathbf{F}, \mathbf{F}_{\text{calib}}) = \|\hat{\mathbf{F}} - \hat{\mathbf{F}}_{\text{calib}}\| = \left\| \frac{\mathbf{F}}{\|\mathbf{F}\|} - \frac{\mathbf{F}_{\text{calib}}}{\|\mathbf{F}_{\text{calib}}\|} \right\| \quad (3.167)$$

where $d_F(\mathbf{F}, \mathbf{F}_{\text{calib}})$ produces a scalar value between 0 and 1, where 0 indicates very similar matrices and 1 indicates very different matrices.

However, it must be noticed that this comparison metric does not have any geometric meaning, an in general does not perform very well (i.e it might yield very low distances for poorly estimated fundamental matrices, and oppositely it can result in large distances for accurately estimated fundamental matrices). Besides, if fundamental matrices $\mathbf{F}$ and $\mathbf{F}_{\text{calib}}$ differ in sign, then the metric reverses its behaviour and similar matrices produce values close to 1, and different matrices produce values close to zero.

Therefore, the Frobenius norm of difference of matrices is a poor metric that also has an ambiguity on the sign of the matrices being compared. However, the ambiguity of sign can be removed by checking both the difference and the sum of matrices.

Hence, a slightly improved version of this metric that removes this sign ambiguity can also be defined as,

$$d_F(\mathbf{F}, \mathbf{F}_{\text{calib}}) = \min\left(\|\hat{\mathbf{F}} - \hat{\mathbf{F}}_{\text{calib}}\|, \|\hat{\mathbf{F}} + \hat{\mathbf{F}}_{\text{calib}}\|\right) = \min\left(\left\| \frac{\mathbf{F}}{\|\mathbf{F}\|} - \frac{\mathbf{F}_{\text{calib}}}{\|\mathbf{F}_{\text{calib}}\|} \right\|, \left\| \frac{\mathbf{F}}{\|\mathbf{F}\|} + \frac{\mathbf{F}_{\text{calib}}}{\|\mathbf{F}_{\text{calib}}\|} \right\|\right) \quad (3.167)$$

where we pick the smallest value between the difference and the sum of the normalized fundamental matrices being compared. Such value will range between 0 for equal matrices, and 1 for different ones.

However, it must still be noticed that this metric has no geometric meaning and performs poorly, although it is very fast to compute.

3.2.6.2.- Mean Epipolar distance

In previous section we saw a method for comparing pairs of fundamental matrices, however, because of its lack of geometric meaning, it usually resulted in non reliable metric values.

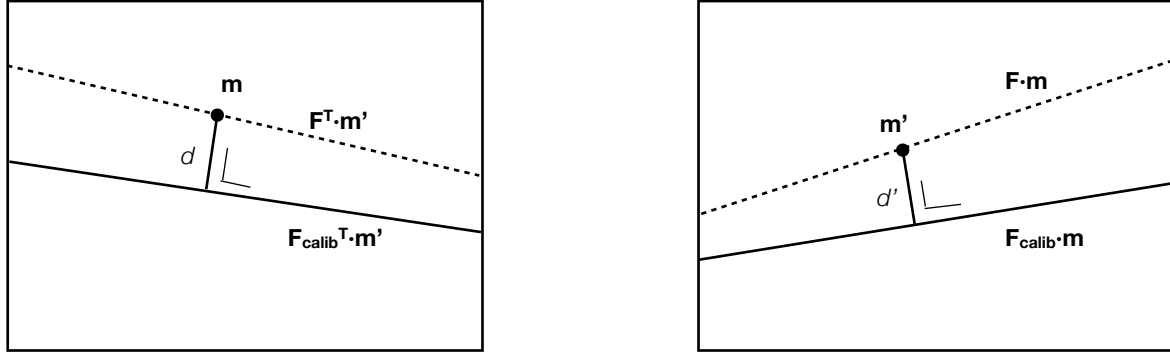


Fig. 3.7. Fundamental matrices comparison in terms of epipolar distances

In this chapter we have described what epipoles ($\mathbf{e}$, $\mathbf{e}'$) and epipolar lines ($\mathbf{l}$, $\mathbf{l}'$) are on the retinal planes $\mathbf{R}$ and $\mathbf{R}'$ of a pair of views. Therefore, we will use these structures in order to define a robust metric with a geometric meaning.

Therefore, we know that because we have a pair of slightly different fundamental matrices $\mathbf{F}$ (i.e. estimated fundamental matrix) and $\mathbf{F}_{\text{calib}}$ (i.e. ground-truth fundamental matrix), then such matrices generate skewed epipolar lines, as we saw in figure 3.5, and as depicted on figure 3.7.

Hence, the metric to compare fundamental matrices can be defined by means of the mean epipolar distances of matched points respect to the different epipolar lines generated by both matrices, just as defined in the following steps, which is also graphically summarized on algorithm 3.2.

First a random point $\mathbf{m}$ on the first view is selected, and then the epipolar line $\mathbf{l}' = \mathbf{F} \cdot \mathbf{m}$ generated on the second view is computed, as shown in figure 3.7. If the epipolar line $\mathbf{l}'$ does not intersect the second image, then another random point $\mathbf{m}$ is selected, otherwise then a second random point $\mathbf{m}'$ on the second view is selected. Such point $\mathbf{m}'$ must lie on generated epipolar line $\mathbf{l}' = \mathbf{F} \cdot \mathbf{m}$ within a typical disparity range (i.e. usually around 10% of image size).

Then for point $\mathbf{m}$ and using the ground-truth fundamental matrix $\mathbf{F}_{\text{calib}}$, we generate the epipolar line $\mathbf{l}_{\text{calib}}' = \mathbf{F}_{\text{calib}} \cdot \mathbf{m}$ on the second view, and compute distance d_{calib}' between point $\mathbf{m}'$ and epipolar line $\mathbf{l}_{\text{calib}}'$, as shown on figure 3.7.

Analogously, using point $\mathbf{m}'$, we can obtain epipolar line $\mathbf{l}_{\text{calib}} = \mathbf{F}_{\text{calib}}^T \cdot \mathbf{m}'$ on the first view, and compute the distance d_{calib} between point $\mathbf{m}$ and epipolar line $\mathbf{l}_{\text{calib}}$.

The same procedure can be repeated but reversing the roles of $\mathbf{F}$ and $\mathbf{F}_{\text{calib}}$ to obtain d and d' , and finally, the process is repeated N times (i.e. typically about 10000 times is enough) in order to reliably compute the average distances d and d' and obtain the mean of both values as a measure of similarity between fundamental matrices with a geometric meaning.

3.2.7.- The Essential matrix

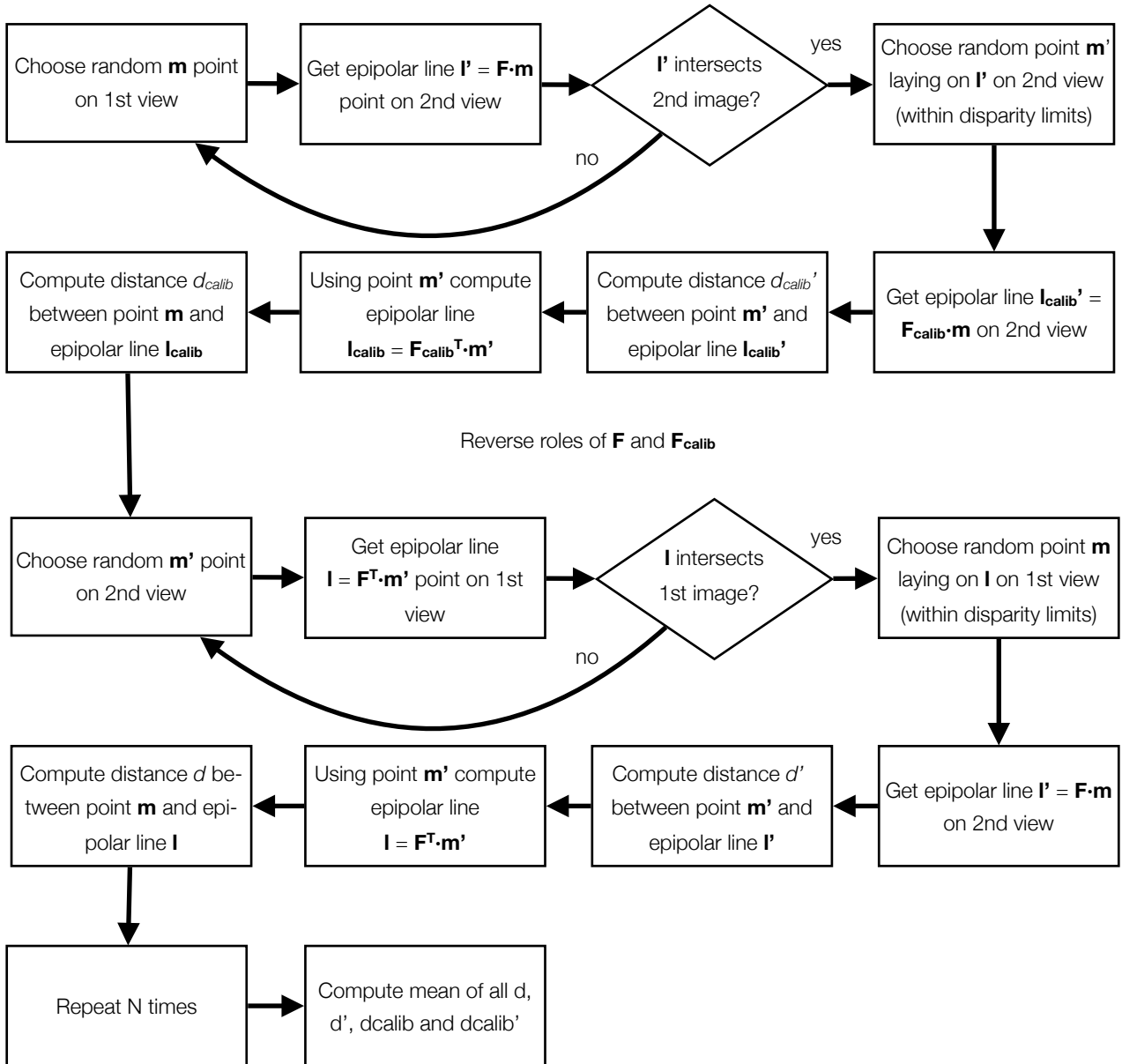
In section 3.2.1.1, we saw how to obtain the fundamental matrix from a pair of known camera matrices $\mathbf{P}$ and $\mathbf{P}'$ as shown in expression (3.11), which by using the camera model given in (2.24),

$$\begin{aligned} \mathbf{P} &= \mathbf{K} [\mathbf{I} | \mathbf{0}_3] \\ \mathbf{P}' &= \mathbf{K}' [\mathbf{R} | \mathbf{t}], \quad \mathbf{t} = -\mathbf{R} \tilde{\mathbf{C}}' \end{aligned} \quad (3.168)$$

where the camera center of the first camera $\mathbf{P}$ is located at world origin:

$$\mathbf{C} = (0, 0, 0, 1)^T \quad (3.169)$$

Algorithm 3.2. Comparing fundamental matrices using mean epipolar distance



the center of the second camera $\mathbf{P}'$ is located at:

$$\mathbf{C}' = \begin{bmatrix} -\mathbf{R}^T \cdot \mathbf{t} \\ 1 \end{bmatrix} \quad (3.170)$$

And the pseudo-inverse of $\mathbf{P}$ can be written as:

$$\mathbf{P}^\dagger = \begin{bmatrix} \mathbf{K}^{-1} \\ \mathbf{0}_3^T \end{bmatrix} \quad (3.171)$$

then (3.11) can be written as:

$$\begin{aligned}
\mathbf{F} &= [\mathbf{e}']_{\times} (\mathbf{P}' \cdot \mathbf{P}^\dagger) = [\mathbf{P}' \cdot \mathbf{C}]_{\times} (\mathbf{P}' \cdot \mathbf{P}^\dagger) \\
&= \left[\mathbf{K}' [\mathbf{R} | \mathbf{t}] \cdot \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \right]_{\times} \cdot \left(\mathbf{K}' [\mathbf{R} | \mathbf{t}] \cdot \begin{bmatrix} \mathbf{K}^{-1} \\ \mathbf{0}_3^T \end{bmatrix} \right) = \\
&= [\mathbf{K}' \mathbf{t}]_{\times} (\mathbf{K}' \mathbf{R} \mathbf{K}^{-1} + \mathbf{K}' \mathbf{t} \mathbf{0}_3^T) = [\mathbf{K}' \mathbf{t}]_{\times} \cdot \mathbf{K}' \mathbf{R} \mathbf{K}^{-1} = \\
&= \mathbf{K}'^T [\mathbf{t}]_{\times} \cdot \mathbf{R} \mathbf{K}^{-1} = \mathbf{K}'^T \mathbf{R} [\mathbf{R}^T \mathbf{t}]_{\times} \cdot \mathbf{K}^{-1}
\end{aligned} \tag{3.172}$$

And the epipoles $\mathbf{e}$ and $\mathbf{e}'$, defined as the projection of the other camera center are:

$$\begin{aligned}
\mathbf{e} &= \mathbf{P} \cdot \mathbf{C}' = \mathbf{K} [\mathbf{I} | \mathbf{0}_3] \cdot \begin{bmatrix} -\mathbf{R}^T \mathbf{t} \\ 1 \end{bmatrix} = \mathbf{K} (-\mathbf{R}^T \mathbf{t} + \mathbf{0}_3) \sim \mathbf{K} \mathbf{R}^T \mathbf{t} \\
\mathbf{e}' &= \mathbf{P}' \cdot \mathbf{C} = \mathbf{K}' [\mathbf{R} | \mathbf{t}] \cdot \begin{bmatrix} \mathbf{0}_3 \\ 1 \end{bmatrix} = \mathbf{K}' (\mathbf{R} \mathbf{0}_3 + \mathbf{t}) = \mathbf{K}' \mathbf{t}
\end{aligned} \tag{3.173}$$

Therefore, if the calibration matrices $\mathbf{K}$ and $\mathbf{K}'$ are known in advance, then points $\mathbf{m}$ and $\mathbf{m}'$ can be normalized as:

$$\begin{aligned}
\hat{\mathbf{m}} &= \mathbf{K}^{-1} \cdot \mathbf{m} \\
\hat{\mathbf{m}}' &= \mathbf{K}'^{-1} \cdot \mathbf{m}'
\end{aligned} \tag{3.174}$$

Consequently, expression (3.4) can be written in terms of normalized image points as:

$$\hat{\mathbf{m}}'^T \cdot \mathbf{E} \cdot \hat{\mathbf{m}} = 0 \tag{3.175}$$

where $\mathbf{E}$ is a 3x3 rank-2 matrix that contains the fundamental matrix for normalized points, which is known as the **Essential matrix**.

By using expression (3.174), then (3.175) becomes:

$$\hat{\mathbf{m}}'^T \cdot \mathbf{E} \cdot \hat{\mathbf{m}} = 0 \Rightarrow \mathbf{m}'^T \cdot \mathbf{K}'^T \cdot \mathbf{E} \cdot \mathbf{K}^{-1} \cdot \mathbf{m} = 0 \tag{3.176}$$

And therefore, by means of (3.4) we can see that the Essential matrix $\mathbf{E}$ and the Fundamental matrix $\mathbf{F}$ are related by the following expression:

$$\mathbf{F} = \mathbf{K}'^T \cdot \mathbf{E} \cdot \mathbf{K}^{-1} \Rightarrow \mathbf{E} = \mathbf{K}'^T \cdot \mathbf{F} \cdot \mathbf{K} \tag{3.177}$$

Hence, by means of (3.172), then the Essential matrix becomes:

$$\begin{aligned}
\mathbf{E} &= \mathbf{K}'^T \cdot \mathbf{F} \cdot \mathbf{K} = \mathbf{K}'^T \cdot (\mathbf{K}'^T [\mathbf{t}]_{\times} \mathbf{R} \mathbf{K}^{-1}) \cdot \mathbf{K} = (\mathbf{K}'^T \mathbf{R} [\mathbf{R}^T \mathbf{t}]_{\times} \mathbf{K}^{-1}) \Rightarrow \\
\mathbf{E} &= [\mathbf{t}]_{\times} \mathbf{R} = \mathbf{R} [\mathbf{R}^T \mathbf{t}]_{\times}
\end{aligned} \tag{3.178}$$

where we can see that the Essential matrix $\mathbf{E}$ only depends on rotation and translation terms.

3.2.7.1.- Properties of the Essential matrix

As we have already seen, the Essential matrix is related by (3.177) to the fundamental matrix. Therefore, the essential matrix is also a 3x3 rank-2 matrix, however, because of its specific structure additional properties can be extracted.

On first place, we saw in (3.178) that the essential matrix only depends on rotation and translation terms between camera pairs, and we saw in chapter 2 that a rotation can be expressed by 3 DOF (the 3 angles of rotation), and the translation is also expressed by 3 parameters in the 3D world space. Therefore, the Essential matrix would have 6 DOF. However, because the Essential matrix is defined up to scale as the fundamental matrix, then the Essential matrix **E** has only 5 degrees of freedom.

As an additional property we will see that any essential matrix **E** has 2 equal singular values and a third singular value equal to zero. This is proved next.

Proof 3.4

Given expression (3.178) where the essential matrix can be seen as a product of a skew-symmetric matrix and a rotation matrix **E** = **[t]_x·R** = **S·R**. And using matrices,

$$\mathbf{W} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{Z} = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad (3.179)$$

where it can be easily verified that **W** is orthogonal and **Z** skew-symmetric, then the skew-symmetric matrix **S** can be decomposed (up to scale k) as:

$$\mathbf{S} = k \cdot \mathbf{U} \cdot \mathbf{Z} \cdot \mathbf{U}^T \quad (3.180)$$

where **U** is a 3x3 orthogonal matrix, and **Z** is defined as:

$$\mathbf{Z} = \text{diag}(1,1,0) \cdot \mathbf{W} \quad (3.181)$$

Therefore, the skew-symmetric matrix **S** can be written as:

$$\mathbf{S} = \mathbf{U} \cdot \text{diag}(1,1,0) \cdot \mathbf{W} \cdot \mathbf{U}^T \quad (3.182)$$

And finally, the Essential matrix can then be written as:

$$\mathbf{E} = \mathbf{S} \cdot \mathbf{R} = \mathbf{U} \cdot \text{diag}(1,1,0) \cdot (\mathbf{W} \cdot \mathbf{U}^T \cdot \mathbf{R}) = \mathbf{U} \cdot \text{diag}(1,1,0) \cdot \mathbf{V}^T \quad (3.183)$$

where **U** is a 3x3 orthogonal matrix and **V** = **W·U^T·R** is also a 3x3 orthogonal matrix as it is the product of 3 orthogonal matrices as well. Therefore, expression (3.183) can be seen as a singular value decomposition of the essential matrix **E**, where it is clearly seen that the two non-null singular values are both equal.

3.2.7.2.- Initial metric structure

The Essential matrix can be computed using the same methods for fundamental matrix estimation but using normalized image point coordinates, or otherwise we can estimate the fundamental matrix **F**, and then use expression (3.177) to obtain the essential matrix **E**.

Once the essential matrix is known, we can retrieve a pair of normalized matrices **[l_{3x3} | 0₃]** and **[R | t]**. And because **K** and **K'** are also known, then we can definitely find the pair of cameras **P** = **K·[l_{3x3} | 0₃]** and **P'** = **K'·[R | t]** in a metric stratum.

With such pair of camera matrices we will be able to compute metric 3D reconstructions or initialize a metric stratum where more cameras can be added.

Then, given the SVD decomposition of the essential matrix $\mathbf{E}$ shown in (3.183) there are two possible factorizations to obtain $\mathbf{E} = \mathbf{S} \cdot \mathbf{R}$, where $\mathbf{S}$ follows expression (3.180) and $\mathbf{R}$ can be written as

$$\mathbf{R} = \mathbf{U} \cdot \mathbf{W} \cdot \mathbf{V}^T \quad \text{or} \quad \mathbf{U} \cdot \mathbf{W}^T \cdot \mathbf{V}^T \quad (3.184)$$

It must be noticed that $\mathbf{S}$ matrix in $\mathbf{E} = \mathbf{S} \cdot \mathbf{R}$ decomposition contains the translation term $\mathbf{t}$ of the camera matrix $\mathbf{P}'$ since $\mathbf{S}$ is the skew-symmetric term $\mathbf{S} = [\mathbf{t}]_{\times}$. However, the Frobenius norm of $\mathbf{S} = \mathbf{U} \cdot \mathbf{Z} \cdot \mathbf{U}^T$ is the squared root of 2, which means that if $\mathbf{S} = [\mathbf{t}]_{\times}$ including scale then $\|\mathbf{t}\| = 1$, which is a convenient normalization of the baseline of the pair of camera matrices $\mathbf{P}$ and $\mathbf{P}'$. Because $\mathbf{S}$ is related to the cross product of vector $\mathbf{t}$, then it is true that $\mathbf{S} \cdot \mathbf{t} = \mathbf{0}_3$, and hence taking into account (3.182), we can observe by using the last column of $\mathbf{U}$, which has a corresponding zero value on the diagonal matrix term $\text{diag}(1, 1, 0)$, the result $\mathbf{S} \cdot \mathbf{t} = \mathbf{0}_3$ is fulfilled. Consequently, the translation term can be written as:

$$\mathbf{t} = \mathbf{u}_3 \quad (3.185)$$

where $\mathbf{t}$ is determined up to sign.

Therefore, there are two possible decompositions of $\mathbf{E}$ due to $\mathbf{t}$ sign, and further two possible decompositions due to expression (3.184). Consequently, there are 4 possible choices for second camera matrix $\mathbf{P}' = \mathbf{K}' \cdot [\mathbf{R} \mid \mathbf{t}]$ due to the ambiguity in rotation and translation.

Hence, for first camera matrix $\mathbf{P} = \mathbf{K} \cdot [\mathbf{I}_{3 \times 3} \mid \mathbf{0}_3]$ we can obtain the following 4 choices for camera matrix $\mathbf{P}'$:

$$\mathbf{P}' = \mathbf{K}' \cdot \left[\mathbf{U} \cdot \mathbf{W} \cdot \mathbf{V}^T \mid +\mathbf{u}_3 \right] \quad \text{or} \quad \mathbf{K}' \cdot \left[\mathbf{U} \cdot \mathbf{W} \cdot \mathbf{V}^T \mid -\mathbf{u}_3 \right] \quad \text{or} \quad \mathbf{K}' \cdot \left[\mathbf{U} \cdot \mathbf{W}^T \cdot \mathbf{V}^T \mid +\mathbf{u}_3 \right] \quad \text{or} \quad \mathbf{K}' \cdot \left[\mathbf{U} \cdot \mathbf{W}^T \cdot \mathbf{V}^T \mid -\mathbf{u}_3 \right] \quad (3.186)$$

From the 4 choices above, we can observe that the sign in the translation term reverses the order of cameras, and the 3x3 top-left term may indicate a rotation of 180° about the line joining both cameras' centers $\mathbf{C}$ and $\mathbf{C}'$. Two solutions related this way are known as a "twisted pair", as depicted in figure 3.8.

Therefore, in order to determine which of the 4 solutions in (3.186) is the real solution we will need to determine whether all world points $\mathbf{M}$ triangulated from corresponding matched points $\mathbf{m}$ and $\mathbf{m}'$ lie in front of both cameras, as will happen in normal situations. In order to perform this check, we can either compute the signed depth of world points respect to cameras, or their cheirality, as was shown in section 2.4. It must be noticed however, that cheirality is simpler to compute than depth of world points respect to cameras.

It may happen however, that due to small errors in matched image points $\mathbf{m}$ and $\mathbf{m}'$, then triangulated world points $\mathbf{M}$ lie behind the cameras even for the correct pair of cameras $\mathbf{P}$ and $\mathbf{P}'$. Therefore, the most robust way to select the correct pair of cameras $\mathbf{P}$ and $\mathbf{P}'$ in expression (3.186), is to check for the pair of cameras that produces the largest amount of correct cheiralities in a set of world points $\mathbf{M}$.

3.2.8.- Degenerate cases

When estimating the fundamental matrix through the 7-point algorithm in section 3.2.2.2, we already saw that degeneracies could be detected if epipolar geometry was not uniquely defined (i.e. more than a single real solution existed for (3.44)), which is equivalent to saying that the null-space of $\mathbf{A}$ in $\mathbf{A} \cdot \mathbf{f}$ seen on (3.30) has dimension greater than 1.

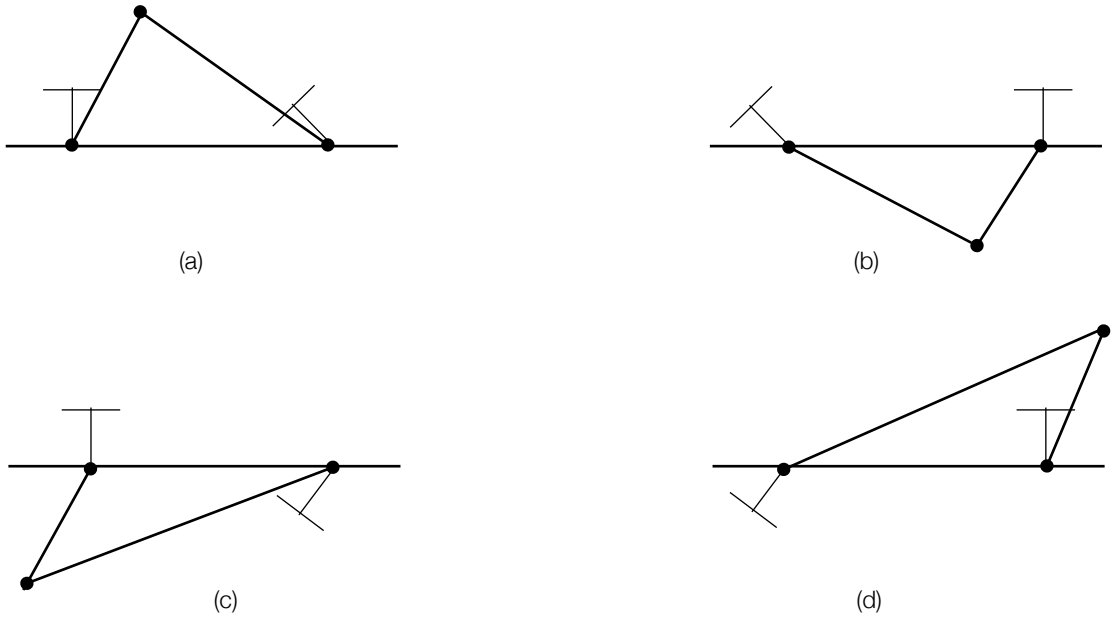


Fig. 3.8. The four possible solutions for the metric reconstruction from essential matrix $\mathbf{E}$. Baseline reversal between left and right sides, and 180° rotation about the baseline between top and bottom rows. Only option (a) is valid since world point lies in front of both cameras

Such degeneracies arise on some specific situations. For instance, from expression (3.11), we can observe that cameras rotated around the same center, then the fundamental matrix $\mathbf{F}$ becomes zero. Therefore, recalling expression (3.11),

$$\mathbf{F} = [\mathbf{e}']_{\times} \cdot (\mathbf{P}' \cdot \mathbf{P}^{\dagger}) \quad (3.187)$$

where the epipole $\mathbf{e}'$ is $\mathbf{e}' = \mathbf{P} \cdot \mathbf{C}'$. However, because the camera centers $\mathbf{C}$ and $\mathbf{C}'$ are now equal, we obtain:

$$\mathbf{e}' = \mathbf{P} \cdot \mathbf{C}' = \mathbf{P} \cdot \mathbf{C} \quad (3.188)$$

And from expression (2.33) we know that $\mathbf{P} \cdot \mathbf{C} = \mathbf{0}_3$, hence the epipole $\mathbf{e}'$ in (3.188) becomes:

$$\mathbf{e}' = \mathbf{P} \cdot \mathbf{C} = \mathbf{0}_3 \quad (3.189)$$

And consequently, from the definition of $[\cdot]_{\times}$ in (1.9), we obtain that expression (3.187) becomes a 3×3 zero-matrix,

$$\mathbf{F} = \mathbf{0}_{3 \times 3} \quad (3.190)$$

which is obviously completely degenerate (i.e. the null-space of $\mathbf{A}$ in $\mathbf{A} \cdot \mathbf{f} = 0$ has dimension 3, which generates a two-parameter family of solutions).

This situation is very common on surveillance cameras, where cameras can be rotated but their center is not moved.

However, there are other situations that might produce degeneracy of $\mathbf{F}$ due to a null-space of $\mathbf{A}$ with dimension greater than 1. For instance when points on the 3D world space and camera centers lie on what's called a ruled quadric. A typical example is when all 3D world points are co-planar (i.e. lie on the same plane). In this case, all world points and the two

camera centers lie on a ruled quadric surface, which consists of two planes, the plane through the world points, plus a plane passing through the two camera centers.

Other degeneracies might appear for other specific ruled quadrics, and other structures, which are nearly degenerate can also produce errors due to numerical precision. However, those situations are less common, and all of them can also be detected by means of the 7-point algorithm.

In any case, to avoid degeneracies or nearly degenerate scenarios, we should ensure that camera centers are as separated as possible (which also increases resolution of 3D reconstruction as we saw in section 2.6.2), and we should avoid scenes where world points are located in degenerate or nearly degenerate configurations.

As we saw in section 3.2.2.2, when working with video sequences, if an epipolar degeneracy is found, then such pair of frames is discarded and proceed to analyze the next frame until no degeneracy is found.

3.2.9.- Uncertainty of the Epipolar geometry

We already saw in section 3.2.5 that due to errors in image point correspondences, the epipolar geometry might result skewed, and then matches need to be corrected. This can also affect to image rectification processes or to the process of search for dense disparity estimation as will be seen in chapter 9. Besides, as will be seen on chapter 5 having some knowledge on the uncertainty of the fundamental matrix can also be used to improve the self-calibration process when using Kruppa equations [30].

Therefore, it is convenient to have an overall knowledge on how uncertainty affects to epipolar geometry and 3D reconstructions in general terms.

3.2.9.1.- Computing the covariance matrix of the fundamental matrix

Because the fundamental matrix can be represented as a vector $\mathbf{f}$, just as we saw in (3.28), then, given N measures of this vector, we can compute its mean and its covariance statistically. Indeed, the mean value of $\mathbf{f}$ will be the fundamental matrix that we desire to estimate, and its covariance will represent the error associated to its estimation.

3.2.9.1.1.- Statistical approach

Hence, the mean of the fundamental matrix vector $\mathbf{f}$ can be simply computed as:

$$\bar{\mathbf{f}} = E[\mathbf{f}] = \frac{1}{N} \sum_{i=1}^N \mathbf{f}_i \quad (3.191)$$

And its covariance can be approximated as:

$$\begin{aligned} \Sigma_{\mathbf{f}} &= E\left[(\mathbf{f} - E[\mathbf{f}])(\mathbf{f} - E[\mathbf{f}])^T\right] = \\ &E\left[\mathbf{f}\mathbf{f}^T - 2\mathbf{f} \cdot E[\mathbf{f}]^T + E[\mathbf{f}] \cdot E[\mathbf{f}]^T\right] = \\ &E\left[\mathbf{f}\mathbf{f}^T\right] - E[\mathbf{f}] \cdot E[\mathbf{f}]^T = \\ &\frac{1}{N} \sum_{i=1}^N \mathbf{f}_i \mathbf{f}_i^T - \bar{\mathbf{f}} \bar{\mathbf{f}}^T \end{aligned} \quad (3.192)$$

However, this statistically approach, although it is valid if enough $\mathbf{f}$ samples are available (i.e. typically $N = 1000$), it is also slow to compute, because it converges very slowly to the true solution as samples are added.

3.2.9.1.2.- Analytical approach

Another approach can be taken into account if some analytical considerations are taken into account.

We can start by the vector $\mathbf{X}$ containing matched pairs of points $\mathbf{m}$ and $\mathbf{m}'$, as we saw in (3.87), therefore,

$$\mathbf{X} = (\tilde{\mathbf{m}}, \tilde{\mathbf{m}}')^T = (x, y, x', y')^T \quad (3.193)$$

Therefore, we can define a function $\mathbf{f} = \boldsymbol{\varphi}(\mathbf{X})$ that transforms vector of matched points $\mathbf{X}$ into the fundamental matrix vector $\mathbf{f}$, by means of any of the methods in section 3.2.2.

Therefore, assuming that the mean $E[\mathbf{X}]$ and covariance of matched points $\mathbf{X}$ is known, then by means of Taylor series, then $\mathbf{f} = \boldsymbol{\varphi}(\mathbf{X})$ can be expanded as:

$$\mathbf{f} = \boldsymbol{\varphi}(\mathbf{X}) = \boldsymbol{\varphi}(E[\mathbf{X}]) + \frac{\partial \boldsymbol{\varphi}(E[\mathbf{X}])}{\partial \mathbf{X}} \cdot (\mathbf{X} - E[\mathbf{X}]) + o(\mathbf{X}) \quad (3.194)$$

where for simplification of terminology,

$$\mathbf{J}_\varphi = \frac{\partial \boldsymbol{\varphi}(E[\mathbf{X}])}{\partial \mathbf{X}} \quad (3.195)$$

is the Jacobian of function $\boldsymbol{\varphi}(\mathbf{X})$ at $E[\mathbf{X}]$, and $o(\mathbf{X})$ is the approximation error inherent to Taylor series when $\mathbf{X}$ differs from $E[\mathbf{X}]$. Therefore, if we assume that $\mathbf{X}$ is sufficiently close to $E[\mathbf{X}]$ then $o(\mathbf{X})$ can be neglected, and $\boldsymbol{\varphi}(\mathbf{X})$ can be approximated as:

$$\begin{aligned} \mathbf{f} &\approx \boldsymbol{\varphi}(E[\mathbf{X}]) \\ \boldsymbol{\varphi}(\mathbf{X}) - \boldsymbol{\varphi}(E[\mathbf{X}]) &\approx \mathbf{J}_\varphi \cdot (\mathbf{X} - E[\mathbf{X}]) \end{aligned} \quad (3.196)$$

And consequently the covariance of fundamental matrix vector $\mathbf{f}$ can be computed as:

$$\begin{aligned} \Sigma_{\mathbf{f}} &= E[(\mathbf{f} - E[\mathbf{f}])(\mathbf{f} - E[\mathbf{f}])^T] = \\ &= E[(\boldsymbol{\varphi}(\mathbf{X}) - E[\boldsymbol{\varphi}(\mathbf{X})])(\boldsymbol{\varphi}(\mathbf{X}) - E[\boldsymbol{\varphi}(\mathbf{X})])^T] = \\ &= E[(\boldsymbol{\varphi}(\mathbf{X}) - \boldsymbol{\varphi}(E[\mathbf{X}])(\boldsymbol{\varphi}(\mathbf{X}) - \boldsymbol{\varphi}(E[\mathbf{X}]))^T] \end{aligned} \quad (3.197)$$

which by means of (3.196) becomes:

$$\begin{aligned} \Sigma_{\mathbf{f}} &= E[(\boldsymbol{\varphi}(\mathbf{X}) - \boldsymbol{\varphi}(E[\mathbf{X}])(\boldsymbol{\varphi}(\mathbf{X}) - \boldsymbol{\varphi}(E[\mathbf{X}]))^T] = \\ &= E[(\mathbf{J}_\varphi \cdot (\mathbf{X} - E[\mathbf{X}])(\mathbf{J}_\varphi \cdot (\mathbf{X} - E[\mathbf{X}]))^T] = \\ &= \mathbf{J}_\varphi \cdot E[(\mathbf{X} - E[\mathbf{X}])(\mathbf{X} - E[\mathbf{X}])^T] \cdot \mathbf{J}_\varphi^T = \\ &= \mathbf{J}_\varphi \cdot \Sigma_{\mathbf{X}} \cdot \mathbf{J}_\varphi^T \end{aligned} \quad (3.198)$$

And consequently the covariance matrix of fundamental matrix vector $\mathbf{f}$ can be directly found from the covariance matrix of matched points $\mathbf{X} = (x, y, x', y')^T$.

$$\Sigma_f = \mathbf{J}_\varphi \cdot \Sigma_X \cdot \mathbf{J}_\varphi^T \quad (3.199)$$

3.2.9.1.3.- The case of estimation of fundamental matrix as minimization of criterion function

In section 3.2.2, we saw how to obtain a solution for (3.30) in LMSE terms by means of SVD, however, we didn't showed why this was a solution to the fundamental matrix $\mathbf{F}$ in terms of minimization of a criterion function. We will prove in this section that the result obtained in section 3.2.2 is also the consequence of a criterion minimization.

Indeed, we saw that in order to obtain the fundamental matrix we wanted to solve (3.30) in LMSE terms. Therefore, for each pair of corresponding points $\mathbf{m}_i$ and $\mathbf{m}'_i$, we can obtain an equation like (3.29), which can be expressed as $\mathbf{A}_i \cdot \mathbf{f} = 0$. However, because we will find an LMSE solution for $\mathbf{f}$, then expression (3.29) won't be exactly fulfilled. For that reason, in order to obtain $\mathbf{f}$ as an LMSE solution for all matched points $\mathbf{m}_i$ and $\mathbf{m}'_i$, we will attempt to minimize:

$$C = \sum_{i=1}^N C_i(\mathbf{m}_i, \mathbf{m}'_i)^2 \quad (3.200)$$

where N is the number of matched pairs and $C_i(\mathbf{m}_i, \mathbf{m}'_i)$ is the criterion for each pair of points to be minimized, which is directly obtained from (3.29) as $\mathbf{A}_i \cdot \mathbf{f}$. Therefore,

$$C_i(\mathbf{m}_i, \mathbf{m}'_i) = \mathbf{A}_i \cdot \mathbf{f} = (\tilde{x} \cdot \tilde{x}', \tilde{y} \cdot \tilde{x}', \tilde{x} \cdot \tilde{y}', \tilde{y} \cdot \tilde{y}', \tilde{y}', \tilde{x}, \tilde{y}, 1) \cdot \mathbf{f} \quad (3.201)$$

Therefore (3.200) can be rewritten as:

$$\begin{aligned} C &= \sum_{i=1}^N C_i(\mathbf{m}_i, \mathbf{m}'_i)^2 = \sum_{i=1}^N (\mathbf{A}_i \cdot \mathbf{f})^T \cdot (\mathbf{A}_i \cdot \mathbf{f}) \\ &= \sum_{i=1}^N \mathbf{f}^T \cdot \mathbf{A}_i^T \cdot \mathbf{A}_i \cdot \mathbf{f} = \mathbf{f}^T \cdot \left(\sum_{i=1}^N \mathbf{A}_i^T \cdot \mathbf{A}_i \right) \cdot \mathbf{f} = \mathbf{f}^T \cdot \mathbf{A}^T \cdot \mathbf{A} \cdot \mathbf{f} \end{aligned} \quad (3.202)$$

where we have taken into account that vectors $\mathbf{A}_i$ are usually stacked into matrix $\mathbf{A}$ as in expression (3.30).

Then, in order to minimize (3.202) we assume a typical constraint to avoid numerical degeneracy due to machine precision so that $\|\mathbf{f}\| = 1$. Hence, we want to find $\mathbf{f}$ that minimizes (3.202) subject to $\|\mathbf{f}\| = 1$. This can be solved by means of Lagrange multipliers as we saw in section 3.2.5.1.1, by setting the following auxiliary function:

$$\Lambda(\mathbf{f}, \lambda) = C + \lambda(\mathbf{f}^T \cdot \mathbf{f} - 1) = \mathbf{f}^T \cdot \mathbf{A}^T \cdot \mathbf{A} \cdot \mathbf{f} + \lambda(\mathbf{f}^T \cdot \mathbf{f} - 1) \quad (3.203)$$

And by setting the gradient of (3.203) to zero, we obtain:

$$\begin{aligned} \nabla_{\mathbf{f}, \lambda} \Lambda(\mathbf{f}, \lambda) &= \left(\frac{\partial \Lambda(\mathbf{f}, \lambda)}{\partial \mathbf{f}}, \frac{\partial \Lambda(\mathbf{f}, \lambda)}{\partial \lambda} \right) = (0, 0) \Rightarrow \\ \frac{\partial \Lambda(\mathbf{f}, \lambda)}{\partial \mathbf{f}} &= 2 \cdot \mathbf{A}^T \cdot \mathbf{A} \cdot \mathbf{f} + 2 \cdot \mathbf{f}^T \cdot \mathbf{f} = 0 \\ \frac{\partial \Lambda(\mathbf{f}, \lambda)}{\partial \lambda} &= \mathbf{f}^T \cdot \mathbf{f} - 1 = 0 \end{aligned} \quad (3.204)$$

where the derivative respect to λ is the constraint that we have imposed and the derivative respect to $\mathbf{f}$ can also be expressed as:

$$\frac{\partial \Lambda(\mathbf{f}, \lambda)}{\partial \mathbf{f}} = 2 \cdot \mathbf{A}^T \cdot \mathbf{A} \cdot \mathbf{f} + 2 \cdot \lambda \cdot \mathbf{f} = 0 \Rightarrow \mathbf{A}^T \cdot \mathbf{A} \cdot \mathbf{f} \sim \lambda \cdot \mathbf{f} \quad (3.205)$$

where the sign of lambda can be set arbitrarily, because $\mathbf{f}$ vector is expressed through homogeneous coordinates of points.

It can be seen that (3.205) is the equation that defines the eigen-decomposition of $\mathbf{A}^T \cdot \mathbf{A}$, and hence, $\mathbf{f}$ is an eigen-vector of $\mathbf{A}^T \cdot \mathbf{A}$. Indeed, we already saw that $\mathbf{f}$ is the singular vector corresponding to the smallest singular value of $\mathbf{A}$. However, taking into account that the SVD of the $N \times 9$ matrix $\mathbf{A}$ can be expressed as:

$$\mathbf{A} = \mathbf{U} \cdot \mathbf{S} \cdot \mathbf{V}^T \quad (3.206)$$

where $\mathbf{U}$ is an $N \times 9$ orthonormal matrix, $\mathbf{S}$ is a 9×9 diagonal matrix containing the singular values and $\mathbf{V}$ is a 9×9 orthonormal matrix.

Therefore, the product $\mathbf{A}^T \cdot \mathbf{A}$ can be expressed in terms of the SVD of $\mathbf{A}$ as shown below:

$$\mathbf{A}^T \cdot \mathbf{A} = (\mathbf{U} \cdot \mathbf{S} \cdot \mathbf{V}^T)^T \cdot \mathbf{U} \cdot \mathbf{S} \cdot \mathbf{V}^T = \mathbf{V} \cdot \mathbf{S}^T \cdot \mathbf{U}^T \cdot \mathbf{U} \cdot \mathbf{S} \cdot \mathbf{V}^T \quad (3.207)$$

However, because $\mathbf{U}$ is orthonormal, then the product $\mathbf{U}^T \cdot \mathbf{U}$ is equal to $\mathbf{U}^T \cdot \mathbf{U} = \mathbf{I}_{9 \times 9}$, and since $\mathbf{S}$ is diagonal then $\mathbf{S}^T = \mathbf{S}$. Hence, (3.207) becomes:

$$\mathbf{A}^T \cdot \mathbf{A} = \mathbf{V} \cdot \mathbf{S} \cdot \mathbf{S} \cdot \mathbf{V}^T = \mathbf{V} \cdot \mathbf{\Lambda} \cdot \mathbf{V}^T \quad (3.208)$$

where $\mathbf{V}$ is the 9×9 orthonormal matrix from the SVD of $\mathbf{A}$, and $\mathbf{\Lambda}$ is a diagonal matrix containing the squared of the singular values of $\mathbf{A}$. We can also observe that expression (3.208) is the eigen-decomposition of the product $\mathbf{A}^T \cdot \mathbf{A}$.

Hence, we can observe that the product $\mathbf{A}^T \cdot \mathbf{A}$ is related to the SVD of $\mathbf{A}$ by the squares of the singular values, and also we can observe that the singular vector $\mathbf{v}_9$ (associated to the smallest singular value), is also the eigen-vector of $\mathbf{A}^T \cdot \mathbf{A}$ associated to the smallest eigen-value, and hence, the solution $\mathbf{f}$ found by means of SVD of $\mathbf{A}$, is also the solution that minimizes the criterion in (3.202).

Once the minimization criterion to obtain $\mathbf{f}$ is known, then, as given in [30] the covariance matrix of $\mathbf{f}$ can be obtained as:

$$\Sigma_{\mathbf{f}} = \frac{2 \cdot C_{\min} \cdot \mathbf{H}^T}{N - p} \quad (3.209)$$

where N is the number of provided matched pairs of points, and p for the case of fundamental matrix estimation is $p = 7$, which is the number of DOF. Also, $C_{\min}$ is the cost obtained with the estimated fundamental matrix vector $\mathbf{f}$, therefore:

$$C_{\min} = \sum_{i=1}^N C_i(\mathbf{m}_i, \mathbf{m}'_i) \quad (3.210)$$

and finally, taking into account that $\mathbf{m}_i$ and $\mathbf{m}'_i$ can be expressed in terms of their inhomogeneous coordinates as:

$$\mathbf{m}_i = (\tilde{x}_i, \tilde{y}_i, 1)^T, \quad \mathbf{m}'_i = (\tilde{x}'_i, \tilde{y}'_i, 1)^T \quad (3.211)$$

then $\mathbf{H}$ in expression (3.209) is defined as follows [30]:

$$\mathbf{H} \approx 2 \sum_{i=1}^N \frac{\partial C_i(\mathbf{m}_i, \mathbf{m}'_i)^T}{\partial \mathbf{f}} \cdot \frac{\partial C_i(\mathbf{m}_i, \mathbf{m}'_i)}{\partial \mathbf{f}} \quad (3.212)$$

where the derivative of the cost term C_i respect to fundamental matrix vector $\mathbf{f}$, can be expressed as:

$$\begin{aligned} \frac{\partial C_i(\mathbf{m}_i, \mathbf{m}'_i)}{\partial \mathbf{f}} &= \frac{\partial C_i(\mathbf{m}_i, \mathbf{m}'_i)}{\partial (f_{11}, f_{12}, \dots, f_{33})} \\ &= (\tilde{x}_i \cdot \tilde{x}'_i, \tilde{y}_i \cdot \tilde{x}'_i, \tilde{x}_i \cdot \tilde{y}'_i, \tilde{x}_i \cdot \tilde{y}'_i, \tilde{y}_i \cdot \tilde{y}'_i, \tilde{y}'_i, \tilde{x}_i, \tilde{y}_i, 1) \end{aligned} \quad (3.213)$$

and consequently (3.212) becomes:

$$\begin{aligned} \mathbf{H} &\approx 2 \sum_{i=1}^N \frac{\partial C_i(\mathbf{m}_i, \mathbf{m}'_i)^T}{\partial \mathbf{f}} \cdot \frac{\partial C_i(\mathbf{m}_i, \mathbf{m}'_i)}{\partial \mathbf{f}} \\ &= 2 \sum_{i=1}^N \begin{bmatrix} \tilde{x}_i \cdot \tilde{x}'_i \\ \tilde{y}_i \cdot \tilde{x}'_i \\ \tilde{x}_i \\ \tilde{x}_i \cdot \tilde{y}'_i \\ \tilde{y}_i \cdot \tilde{y}'_i \\ \tilde{y}'_i \\ \tilde{x}_i \\ \tilde{y}_i \\ 1 \end{bmatrix} \cdot \begin{bmatrix} \tilde{x}_i \cdot \tilde{x}'_i & \tilde{y}_i \cdot \tilde{x}'_i & \tilde{x}_i & \tilde{x}_i \cdot \tilde{y}'_i & \tilde{y}_i \cdot \tilde{y}'_i & \tilde{y}'_i & \tilde{x}_i & \tilde{y}_i & 1 \end{bmatrix} = \\ &= 2 \sum_{i=1}^N \begin{bmatrix} \tilde{x}_i^2 \cdot \tilde{x}'_i{}^2 & \tilde{x}_i \cdot \tilde{y}_i \cdot \tilde{x}'_i{}^2 & \tilde{x}_i \cdot \tilde{x}'_i{}^2 & \tilde{x}_i^2 \cdot \tilde{x}'_i \cdot \tilde{y}'_i & \tilde{x}_i \cdot \tilde{y}_i \cdot \tilde{x}'_i \cdot \tilde{y}'_i & \tilde{x}_i \cdot \tilde{x}'_i \cdot \tilde{y}'_i & \tilde{x}_i^2 \cdot \tilde{x}'_i & \tilde{x}_i \cdot \tilde{y}_i \cdot \tilde{x}'_i & \tilde{x}_i \cdot \tilde{x}'_i \\ \tilde{x}_i \cdot \tilde{y}_i \cdot \tilde{x}'_i{}^2 & \tilde{y}_i^2 \cdot \tilde{x}'_i{}^2 & \tilde{y}_i \cdot \tilde{x}'_i{}^2 & \tilde{x}_i \cdot \tilde{y}_i \cdot \tilde{x}'_i \cdot \tilde{y}'_i & \tilde{y}_i^2 \cdot \tilde{x}'_i \cdot \tilde{y}'_i & \tilde{y}_i \cdot \tilde{x}'_i \cdot \tilde{y}'_i & \tilde{x}_i \cdot \tilde{y}_i \cdot \tilde{x}'_i & \tilde{y}_i^2 \cdot \tilde{x}'_i & \tilde{y}_i \cdot \tilde{x}'_i \\ \tilde{x}_i \cdot \tilde{x}'_i{}^2 & \tilde{y}_i \cdot \tilde{x}'_i{}^2 & \tilde{x}'_i{}^2 & \tilde{x}_i \cdot \tilde{x}'_i \cdot \tilde{y}'_i & \tilde{y}_i \cdot \tilde{x}'_i \cdot \tilde{y}'_i & \tilde{x}'_i \cdot \tilde{y}'_i & \tilde{x}_i \cdot \tilde{x}'_i & \tilde{x}'_i \cdot \tilde{y}_i & \tilde{x}'_i \\ \tilde{x}_i^2 \cdot \tilde{x}'_i \cdot \tilde{y}'_i & \tilde{x}_i \cdot \tilde{y}_i \cdot \tilde{x}'_i \cdot \tilde{y}'_i & \tilde{x}_i \cdot \tilde{x}'_i \cdot \tilde{y}'_i & \tilde{x}_i^2 \cdot \tilde{y}'_i{}^2 & \tilde{x}_i \cdot \tilde{y}_i \cdot \tilde{y}'_i{}^2 & \tilde{x}_i \cdot \tilde{y}'_i{}^2 & \tilde{x}_i^2 \cdot \tilde{y}'_i & \tilde{x}_i \cdot \tilde{y}_i \cdot \tilde{y}'_i & \tilde{x}_i \cdot \tilde{y}'_i \\ \tilde{x}_i \cdot \tilde{y}_i \cdot \tilde{x}'_i \cdot \tilde{y}'_i & \tilde{y}_i^2 \cdot \tilde{x}'_i \cdot \tilde{y}'_i & \tilde{y}_i \cdot \tilde{x}'_i \cdot \tilde{y}'_i & \tilde{x}_i \cdot \tilde{y}_i \cdot \tilde{y}'_i{}^2 & \tilde{y}_i^2 \cdot \tilde{y}'_i{}^2 & \tilde{y}_i \cdot \tilde{y}'_i{}^2 & \tilde{x}_i \cdot \tilde{y}_i \cdot \tilde{y}'_i & \tilde{y}_i^2 \cdot \tilde{y}'_i & \tilde{y}_i \cdot \tilde{y}'_i \\ \tilde{x}_i \cdot \tilde{x}'_i \cdot \tilde{y}'_i & \tilde{y}_i \cdot \tilde{x}'_i \cdot \tilde{y}'_i & \tilde{x}'_i \cdot \tilde{y}'_i & \tilde{x}_i \cdot \tilde{y}'_i{}^2 & \tilde{y}_i \cdot \tilde{y}'_i{}^2 & \tilde{y}'_i{}^2 & \tilde{x}_i \cdot \tilde{y}'_i & \tilde{y}_i \cdot \tilde{y}'_i & \tilde{y}'_i \\ \tilde{x}_i^2 \cdot \tilde{x}'_i & \tilde{x}_i \cdot \tilde{y}_i \cdot \tilde{x}'_i & \tilde{x}_i \cdot \tilde{x}'_i & \tilde{x}_i^2 \cdot \tilde{y}'_i & \tilde{x}_i \cdot \tilde{y}_i \cdot \tilde{y}'_i & \tilde{x}_i \cdot \tilde{y}'_i & \tilde{x}_i^2 & \tilde{x}_i \cdot \tilde{y}_i & \tilde{x}_i \\ \tilde{x}_i \cdot \tilde{y}_i \cdot \tilde{x}'_i & \tilde{y}_i^2 \cdot \tilde{x}'_i & \tilde{x}'_i \cdot \tilde{y}_i & \tilde{x}_i \cdot \tilde{y}_i \cdot \tilde{y}'_i & \tilde{y}_i^2 \cdot \tilde{y}'_i & \tilde{y}_i \cdot \tilde{y}'_i & \tilde{x}_i \cdot \tilde{y}_i & \tilde{y}_i^2 & \tilde{y}_i \\ \tilde{x}_i \cdot \tilde{x}'_i & \tilde{y}_i \cdot \tilde{x}'_i & \tilde{x}'_i & \tilde{x}_i \cdot \tilde{y}'_i & \tilde{y}_i \cdot \tilde{y}'_i & \tilde{y}'_i & \tilde{x}_i & \tilde{y}_i & 1 \end{bmatrix} \end{aligned} \quad (3.214)$$

Therefore, using expressions (3.209), (3.210) and (3.214) the covariance matrix $\Sigma_{\mathbf{f}}$ of fundamental matrix $\mathbf{F}$ (or fundamental matrix vector $\mathbf{f}$) can already be computed.

In the following sections we will see that using the covariance of $\mathbf{F}$ we will be able to also find the uncertainty of epipolar lines and epipoles, and ultimately the uncertainty on 3D reconstruction.

3.2.9.2.- The envelope of epipolar lines

As we mentioned in section 3.2.5, because of errors in estimated matched points, epipolar geometry might result skewed, which makes corresponding point not to lie exactly on their corresponding epipolar lines. However, if the covariance of the fundamental matrix is known Σ_f , then an area surrounding the epipolar line (i.e. **the epipolar band**) can be found with a certain probability that matched points lie within such area and assuming gaussianity of errors in the estimated fundamental matrix.

Therefore, assuming that the covariance of the fundamental matrix Σ_f is known (see section 3.2.9.1), taking into account that epipolar lines can be expressed as in (3.3) $\mathbf{l}' = \mathbf{F} \cdot \mathbf{m}$,

$$\mathbf{l}' = \mathbf{F} \cdot \mathbf{m} = \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} f_{11} \cdot x + f_{12} \cdot y + f_{13} \cdot w \\ f_{21} \cdot x + f_{22} \cdot y + f_{23} \cdot w \\ f_{31} \cdot x + f_{32} \cdot y + f_{33} \cdot w \end{bmatrix} \quad (3.215)$$

therefore, taking into account that the Jacobian of $\mathbf{l}'$ respect to the terms of the fundamental matrix vector $\mathbf{f}$ is,

$$\mathbf{J}_{l'} = \frac{\partial \mathbf{l}'(\mathbf{f}, \mathbf{m})}{\partial \mathbf{f}} \quad (3.216)$$

where $\mathbf{J}_{l'}$ is a 3x9 matrix, as can be seen below after expansion of (3.216),

$$\mathbf{J}_{l'} = \frac{\partial \mathbf{l}'(\mathbf{f}, \mathbf{m})}{\partial \mathbf{f}} = \frac{\partial \mathbf{l}'(\mathbf{f}, \mathbf{m})}{\partial (f_{11}, f_{12}, \dots, f_{33})} = \begin{bmatrix} x & y & w & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & x & y & w & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & x & y & w \end{bmatrix} \quad (3.217)$$

Then by analogy with (3.199) the covariance of the estimated epipolar line $\mathbf{l}'$ generated by a known point $\mathbf{m}$ is:

$$\Sigma_{l'} = \mathbf{J}_{l'} \cdot \Sigma_f \cdot \mathbf{J}_{l'}^T \quad (3.218)$$

where we know that the 3x3 covariance $\Sigma_{l'}$ of $\mathbf{l}'$ is singular because $\mathbf{l}'$ is a line constrained in a $\mathbf{P}^2$ surface (i.e. the image plane) within a $\mathbf{P}^3$ space world.

Then, we can find all lines $\mathbf{l}'$ with equal likelihood by means of the Mahalanobis distance as:

$$D_M(\mathbf{l}') = (\mathbf{l}' - E[\mathbf{l}'])^T \cdot \Sigma_{l'}^+ \cdot (\mathbf{l}' - E[\mathbf{l}']) = k^2 \quad (3.219)$$

where k is a constant defining the square of Mahalanobis distance, and we have taken the pseudo-inverse of the covariance matrix because it is singular.

Next, using the eigen-decomposition of the covariance of epipolar line $\mathbf{l}'$, we obtain:

$$\Sigma_{l'} = \mathbf{U} \cdot \hat{\Sigma}_{l'} \cdot \mathbf{U}^T \quad (3.220)$$

where $\mathbf{U}$ is an orthonormal matrix (i.e. $\mathbf{U}^{-1} = \mathbf{U}^T$), and $\hat{\Sigma}_{l'}$ is a diagonal matrix having the following form,

$$\hat{\Sigma}_{\mathbf{I}'} = \text{diag}(\sigma_{\mathbf{I}'_1}^2, \sigma_{\mathbf{I}'_2}^2, 0) \quad (3.221)$$

which contains the variance of the first and second components of $\mathbf{I}'$, and the last component is zero, because epipolar lines lie on the same plane (i.e the image plane), and because as we said before, $\Sigma_{\mathbf{I}'}$ the covariance of $\mathbf{I}'$ is singular. Therefore, (3.221) is also singular and its inverse does not exist, but its pseudo-inverse can be expressed as:

$$\hat{\Sigma}_{\mathbf{I}'}^+ = \text{diag}\left(\frac{1}{\sigma_{\mathbf{I}'_1}^2}, \frac{1}{\sigma_{\mathbf{I}'_2}^2}, 0\right) = \begin{bmatrix} \tilde{\Lambda}_{\mathbf{I}'}^{-1} & \mathbf{0}_2 \\ \mathbf{0}_2^T & 0 \end{bmatrix} \quad (3.222)$$

where the 2x2 top-left minor contains the reciprocals of the variance of the first two components of epipolar line $\mathbf{I}'$.

Besides, because any covariance is symmetric (and in this case is real, which means that is definite positive, or with all eigenvalues real and positive or equal to zero), then (3.220) can also be written as its transpose,

$$\Sigma_{\mathbf{I}'} = \Sigma_{\mathbf{I}'}^T = \mathbf{U}^T \cdot \hat{\Sigma}_{\mathbf{I}'}^T \cdot \mathbf{U} = \mathbf{U}^T \cdot \hat{\Sigma}_{\mathbf{I}'} \cdot \mathbf{U} \quad (3.223)$$

where we have taken into account that $\hat{\Sigma}_{\mathbf{I}'}$ is a diagonal matrix, and therefore $\hat{\Sigma}_{\mathbf{I}'} = \hat{\Sigma}_{\mathbf{I}'}^T$.

Hence, expression (3.219) can be rewritten as:

$$\begin{aligned} D_M(\mathbf{I}') &= (\mathbf{I}' - E[\mathbf{I}'])^T \cdot \Sigma_{\mathbf{I}'}^+ \cdot (\mathbf{I}' - E[\mathbf{I}']) \\ &= (\mathbf{I}' - E[\mathbf{I}'])^T \cdot (\mathbf{U}^T \cdot \hat{\Sigma}_{\mathbf{I}'} \cdot \mathbf{U})^+ \cdot (\mathbf{I}' - E[\mathbf{I}']) \\ &= (\mathbf{I}' - E[\mathbf{I}'])^T \cdot \mathbf{U}^{-1} \cdot \hat{\Sigma}_{\mathbf{I}'}^+ \cdot \mathbf{U}^T \cdot (\mathbf{I}' - E[\mathbf{I}']) \\ &= (\mathbf{I}' - E[\mathbf{I}'])^T \cdot \mathbf{U}^T \cdot \hat{\Sigma}_{\mathbf{I}'}^+ \cdot \mathbf{U} \cdot (\mathbf{I}' - E[\mathbf{I}']) \\ &= (\mathbf{U} \cdot (\mathbf{I}' - E[\mathbf{I}']))^T \cdot \hat{\Sigma}_{\mathbf{I}'}^+ \cdot (\mathbf{U} \cdot (\mathbf{I}' - E[\mathbf{I}'])) \\ &= (\mathbf{U} \cdot \mathbf{I}' - \mathbf{U} \cdot E[\mathbf{I}'])^T \cdot \hat{\Sigma}_{\mathbf{I}'}^+ \cdot (\mathbf{U} \cdot \mathbf{I}' - \mathbf{U} \cdot E[\mathbf{I}']) \\ &= (\mathbf{U} \cdot \mathbf{I}' - E[\mathbf{U} \cdot \mathbf{I}'])^T \cdot \hat{\Sigma}_{\mathbf{I}'}^+ \cdot (\mathbf{U} \cdot \mathbf{I}' - E[\mathbf{U} \cdot \mathbf{I}']) = k^2 \end{aligned} \quad (3.224)$$

And assuming that epipolar line $\mathbf{I}'$ can be normalized using orthonormal matrix $\mathbf{U}$,

$$\hat{\mathbf{I}}' = \mathbf{U} \cdot \mathbf{I}' \quad (3.225)$$

then (3.224) becomes,

$$(\hat{\mathbf{I}}' - E[\hat{\mathbf{I}}'])^T \cdot \hat{\Sigma}_{\mathbf{I}'}^+ \cdot (\hat{\mathbf{I}}' - E[\hat{\mathbf{I}}']) = k^2 \quad (3.226)$$

Taking into account that transformed line $\mathbf{I}'$ can be expressed in terms of its inhomogeneous representation as:

$$\hat{\mathbf{I}}' = \begin{pmatrix} \hat{\mathbf{I}}' \\ 1 \end{pmatrix}^T \quad (3.227)$$

Then, by using (3.222) and (3.227), we obtain that (3.226) becomes:

$$\begin{aligned}
 & \left(\hat{\mathbf{I}}' - E \left[\hat{\mathbf{I}}' \right] \right)^T \cdot \hat{\Sigma}_r^+ \cdot \left(\hat{\mathbf{I}}' - E \left[\hat{\mathbf{I}}' \right] \right) = \\
 & \left(\hat{\mathbf{I}}' - E \left[\hat{\mathbf{I}}' \right] \right)^T \cdot \begin{bmatrix} \tilde{\Lambda}_r^{-1} & \mathbf{0}_2 \\ \mathbf{0}_2^T & 0 \end{bmatrix} \cdot \left(\hat{\mathbf{I}}' - E \left[\hat{\mathbf{I}}' \right] \right) = \\
 & \left(\hat{\mathbf{I}}' - E \left[\hat{\mathbf{I}}' \right] \right)^T \cdot \tilde{\Lambda}_r^{-1} \cdot \left(\hat{\mathbf{I}}' - E \left[\hat{\mathbf{I}}' \right] \right) = k^2
 \end{aligned} \tag{3.228}$$

And finally, by expanding previous expression, we obtain,

$$\hat{\mathbf{I}}'^T \cdot \tilde{\Lambda}_r^{-1} \cdot \hat{\mathbf{I}}' - \hat{\mathbf{I}}'^T \cdot \tilde{\Lambda}_r^{-1} \cdot E \left[\hat{\mathbf{I}}' \right] - E \left[\hat{\mathbf{I}}'^T \right] \cdot \tilde{\Lambda}_r^{-1} \cdot \hat{\mathbf{I}}' + E \left[\hat{\mathbf{I}}'^T \right] \cdot \tilde{\Lambda}_r^{-1} \cdot E \left[\hat{\mathbf{I}}' \right] - k^2 = 0 \tag{3.229}$$

This expression can be compacted by factorizing in terms of the transformed epipolar line as in (3.227), just as follows:

$$\begin{aligned}
 & \begin{bmatrix} \hat{\mathbf{I}}'^T & 1 \end{bmatrix} \cdot \begin{bmatrix} \tilde{\Lambda}_r^{-1} & -\tilde{\Lambda}_r^{-1} \cdot E \left[\hat{\mathbf{I}}' \right] \\ -E \left[\hat{\mathbf{I}}'^T \right] \cdot \tilde{\Lambda}_r^{-1} & E \left[\hat{\mathbf{I}}'^T \right] \cdot \tilde{\Lambda}_r^{-1} \cdot E \left[\hat{\mathbf{I}}' \right] - k^2 \end{bmatrix} \cdot \begin{bmatrix} \hat{\mathbf{I}}' \\ 1 \end{bmatrix} = 0 \Rightarrow \\
 & \begin{bmatrix} \hat{\mathbf{I}}'^T & 1 \end{bmatrix} \cdot \hat{\mathbf{C}}'^{-1} \cdot \begin{bmatrix} \hat{\mathbf{I}}' \\ 1 \end{bmatrix} = 0 \Rightarrow \hat{\mathbf{I}}'^T \cdot \hat{\mathbf{C}}'^{-1} \cdot \hat{\mathbf{I}}' = 0
 \end{aligned} \tag{3.230}$$

where expression (3.230) is analogous to (1.33), because the middle term is a 3x3 symmetric matrix, and therefore such matrix $\hat{\mathbf{C}}'^{-1}$ can be seen as a dual conic $\hat{\mathbf{C}}'^* = \hat{\mathbf{C}}'^{-1}$, taking into account that it is defined in the transformed coordinates obtained after using $\mathbf{U}$ as a transformation.

$$\hat{\mathbf{C}}'^{-1} = \begin{bmatrix} \tilde{\Lambda}_r^{-1} & -\tilde{\Lambda}_r^{-1} \cdot E \left[\hat{\mathbf{I}}' \right] \\ -E \left[\hat{\mathbf{I}}'^T \right] \cdot \tilde{\Lambda}_r^{-1} & E \left[\hat{\mathbf{I}}'^T \right] \cdot \tilde{\Lambda}_r^{-1} \cdot E \left[\hat{\mathbf{I}}' \right] - k^2 \end{bmatrix} \tag{3.231}$$

Hence by using the blockwise inversion shown in expression (1.189), we can obtain the expression of such conic $\mathbf{C}'$ as follows:

$$\begin{aligned}
\hat{\mathbf{C}}' &= \begin{bmatrix} \tilde{\Lambda}_r^{-1} & -\tilde{\Lambda}_r^{-1} \cdot E[\hat{\mathbf{I}}] \\ -E[\hat{\mathbf{I}}^T] \cdot \tilde{\Lambda}_r^{-1} & E[\hat{\mathbf{I}}^T] \cdot \tilde{\Lambda}_r^{-1} \cdot E[\hat{\mathbf{I}}] - k^2 \end{bmatrix}^{-1} = \\
&= \begin{bmatrix} \tilde{\Lambda}_r + \tilde{\Lambda}_r \cdot \tilde{\Lambda}_r^{-1} \cdot E[\hat{\mathbf{I}}] \left(E[\hat{\mathbf{I}}^T] \cdot \tilde{\Lambda}_r^{-1} \cdot E[\hat{\mathbf{I}}] - k^2 - E[\hat{\mathbf{I}}^T] \cdot \tilde{\Lambda}_r^{-1} \cdot \tilde{\Lambda}_r \cdot \tilde{\Lambda}_r^{-1} \cdot E[\hat{\mathbf{I}}] \right)^{-1} \cdot E[\hat{\mathbf{I}}^T] \cdot \tilde{\Lambda}_r^{-1} \cdot \tilde{\Lambda}_r & \tilde{\Lambda}_r \cdot \tilde{\Lambda}_r^{-1} \cdot E[\hat{\mathbf{I}}] \left(E[\hat{\mathbf{I}}^T] \cdot \tilde{\Lambda}_r^{-1} \cdot E[\hat{\mathbf{I}}] - k^2 - E[\hat{\mathbf{I}}^T] \cdot \tilde{\Lambda}_r^{-1} \cdot \tilde{\Lambda}_r \cdot \tilde{\Lambda}_r^{-1} \cdot E[\hat{\mathbf{I}}] \right)^{-1} \\ \left(E[\hat{\mathbf{I}}^T] \cdot \tilde{\Lambda}_r^{-1} \cdot E[\hat{\mathbf{I}}] - k^2 - E[\hat{\mathbf{I}}^T] \cdot \tilde{\Lambda}_r^{-1} \cdot \tilde{\Lambda}_r \cdot \tilde{\Lambda}_r^{-1} \cdot E[\hat{\mathbf{I}}] \right)^{-1} \cdot E[\hat{\mathbf{I}}^T] \cdot \tilde{\Lambda}_r^{-1} \cdot \tilde{\Lambda}_r & \left(E[\hat{\mathbf{I}}^T] \cdot \tilde{\Lambda}_r^{-1} \cdot E[\hat{\mathbf{I}}] - k^2 - E[\hat{\mathbf{I}}^T] \cdot \tilde{\Lambda}_r^{-1} \cdot \tilde{\Lambda}_r \cdot \tilde{\Lambda}_r^{-1} \cdot E[\hat{\mathbf{I}}] \right)^{-1} \end{bmatrix} = \\
&= \begin{bmatrix} \tilde{\Lambda}_r + E[\hat{\mathbf{I}}] \left(E[\hat{\mathbf{I}}^T] \cdot \tilde{\Lambda}_r^{-1} \cdot E[\hat{\mathbf{I}}] - k^2 - E[\hat{\mathbf{I}}^T] \cdot \tilde{\Lambda}_r^{-1} \cdot \tilde{\Lambda}_r \cdot \tilde{\Lambda}_r^{-1} \cdot E[\hat{\mathbf{I}}] \right)^{-1} \cdot E[\hat{\mathbf{I}}^T] & E[\hat{\mathbf{I}}] \left(E[\hat{\mathbf{I}}^T] \cdot \tilde{\Lambda}_r^{-1} \cdot E[\hat{\mathbf{I}}] - k^2 - E[\hat{\mathbf{I}}^T] \cdot \tilde{\Lambda}_r^{-1} \cdot \tilde{\Lambda}_r \cdot \tilde{\Lambda}_r^{-1} \cdot E[\hat{\mathbf{I}}] \right)^{-1} \\ \left(E[\hat{\mathbf{I}}^T] \cdot \tilde{\Lambda}_r^{-1} \cdot E[\hat{\mathbf{I}}] - k^2 - E[\hat{\mathbf{I}}^T] \cdot \tilde{\Lambda}_r^{-1} \cdot \tilde{\Lambda}_r \cdot \tilde{\Lambda}_r^{-1} \cdot E[\hat{\mathbf{I}}] \right)^{-1} \cdot E[\hat{\mathbf{I}}^T] & \left(E[\hat{\mathbf{I}}^T] \cdot \tilde{\Lambda}_r^{-1} \cdot E[\hat{\mathbf{I}}] - k^2 - E[\hat{\mathbf{I}}^T] \cdot \tilde{\Lambda}_r^{-1} \cdot \tilde{\Lambda}_r \cdot \tilde{\Lambda}_r^{-1} \cdot E[\hat{\mathbf{I}}] \right)^{-1} \end{bmatrix} = \\
&= \begin{bmatrix} \tilde{\Lambda}_r + E[\hat{\mathbf{I}}] \cdot (-k^2)^{-1} \cdot E[\hat{\mathbf{I}}^T] & E[\hat{\mathbf{I}}] \cdot (-k^2)^{-1} \\ (-k^2)^{-1} \cdot E[\hat{\mathbf{I}}^T] & (-k^2)^{-1} \end{bmatrix} = \begin{bmatrix} \tilde{\Lambda}_r - \frac{1}{k^2} \cdot E[\hat{\mathbf{I}}] \cdot E[\hat{\mathbf{I}}^T] & -\frac{1}{k^2} \cdot E[\hat{\mathbf{I}}] \\ -\frac{1}{k^2} \cdot E[\hat{\mathbf{I}}^T] & -\frac{1}{k^2} \end{bmatrix}
\end{aligned} \tag{3.232}$$

And because conics $\mathbf{C}'$ are only defined up to scale, then we can normalize the bottom-right element of (3.232) as follows,

$$\hat{\mathbf{C}}' = \begin{bmatrix} \tilde{\Lambda}_r - \frac{1}{k^2} \cdot E[\hat{\mathbf{I}}] \cdot E[\hat{\mathbf{I}}^T] & -\frac{1}{k^2} \cdot E[\hat{\mathbf{I}}] \\ -\frac{1}{k^2} \cdot E[\hat{\mathbf{I}}^T] & -\frac{1}{k^2} \end{bmatrix} \sim \begin{bmatrix} E[\hat{\mathbf{I}}] \cdot E[\hat{\mathbf{I}}^T] - k^2 \cdot \tilde{\Lambda}_r & E[\hat{\mathbf{I}}] \\ E[\hat{\mathbf{I}}^T] & 1 \end{bmatrix} \tag{3.233}$$

And finally, (3.233) can be written in terms of the homogeneous representation of epipolar lines taking into account that the last component of (3.227) is 1 and using (3.222). Therefore,

$$\hat{\mathbf{C}}' \sim \left[E[\hat{\mathbf{I}}] \cdot E[\hat{\mathbf{I}}^T] - k^2 \cdot \hat{\Sigma}_r^+ \right] \tag{3.234}$$

where we can prove that (3.234) is equal to (3.233) as shown next.

Proof 3.5

$$\begin{aligned}
\hat{\mathbf{C}}' &\sim \left[E[\hat{\mathbf{I}}] \cdot E[\hat{\mathbf{I}}^T] - k^2 \cdot \hat{\Sigma}_r^+ \right] = \\
&= \begin{bmatrix} E[\hat{\mathbf{I}}] \\ 1 \end{bmatrix} \cdot \begin{bmatrix} E[\hat{\mathbf{I}}^T] & 1 \end{bmatrix} - k^2 \cdot \begin{bmatrix} \tilde{\Lambda}_r^{-1} & \mathbf{0}_2 \\ \mathbf{0}_2^T & 0 \end{bmatrix} = \\
&= \begin{bmatrix} E[\hat{\mathbf{I}}] \cdot E[\hat{\mathbf{I}}^T] & E[\hat{\mathbf{I}}] \\ E[\hat{\mathbf{I}}^T] & 1 \end{bmatrix} - k^2 \cdot \begin{bmatrix} \tilde{\Lambda}_r^{-1} & \mathbf{0}_2 \\ \mathbf{0}_2^T & 0 \end{bmatrix}
\end{aligned} \tag{3.235}$$

which can be finally be written as in (2.233) as shown below

$$\hat{\mathbf{C}}' \sim \begin{bmatrix} E[\hat{\mathbf{I}}'] \cdot E[\hat{\mathbf{I}}'^T] & E[\hat{\mathbf{I}}'] \\ E[\hat{\mathbf{I}}'^T] & 1 \end{bmatrix} - k^2 \begin{bmatrix} \tilde{\Lambda}_1^{-1} & \mathbf{0}_2 \\ \mathbf{0}_2^T & 0 \end{bmatrix} = \begin{bmatrix} E[\hat{\mathbf{I}}'] \cdot E[\hat{\mathbf{I}}'^T] - k^2 \cdot \tilde{\Lambda}_1 & E[\hat{\mathbf{I}}'] \\ E[\hat{\mathbf{I}}'^T] & 1 \end{bmatrix} \quad (3.236)$$

Hence, by means of expression (1.59), we obtain that given the line transformation in (3.225), then the transformed conic is:

$$\hat{\mathbf{C}}' \sim \mathbf{U} \cdot \mathbf{C} \cdot \mathbf{U}^T \Rightarrow \mathbf{C}' \sim \mathbf{U}^{-1} \cdot \hat{\mathbf{C}}' \cdot \mathbf{U}^{-T} \quad (3.237)$$

Therefore, by using expression (3.234) into (3.237), we obtain that the conic in the non transformed coordinates is:

$$\begin{aligned} \mathbf{C}' &\sim \mathbf{U}^{-1} \cdot \hat{\mathbf{C}}' \cdot \mathbf{U}^{-T} = \mathbf{U}^{-1} \cdot \left[E[\hat{\mathbf{I}}'] \cdot E[\hat{\mathbf{I}}'^T] - k^2 \cdot \hat{\Sigma}_1^+ \right] \cdot \mathbf{U}^{-T} \\ &= \mathbf{U}^{-1} \cdot E[\hat{\mathbf{I}}'] \cdot E[\hat{\mathbf{I}}'^T] \cdot \mathbf{U}^{-T} - k^2 \cdot \mathbf{U}^{-1} \cdot \hat{\Sigma}_1^+ \cdot \mathbf{U}^{-T} \\ &= E[\mathbf{U}^{-1} \cdot \hat{\mathbf{I}}'] \cdot E[\hat{\mathbf{I}}'^T \cdot \mathbf{U}^{-T}] - k^2 \cdot \mathbf{U}^{-1} \cdot \hat{\Sigma}_1^+ \cdot \mathbf{U}^{-T} \\ &= E[\mathbf{U}^{-1} \cdot \hat{\mathbf{I}}'] \cdot E[(\mathbf{U}^{-1} \cdot \hat{\mathbf{I}}')^T] - k^2 \cdot (\mathbf{U}^T \cdot \hat{\Sigma}_1^+ \cdot \mathbf{U})^+ \end{aligned} \quad (3.238)$$

And by using (3.223) and (3.225) then (3.238) becomes:

$$\mathbf{C}' \sim E[\mathbf{I}'] \cdot E[\mathbf{I}'^T] - k^2 \cdot \Sigma_1^+ \quad (3.239)$$

And hence, epipolar bands define an area surrounding epipolar lines defined by the conic $\mathbf{C}'$ shown above, where we have

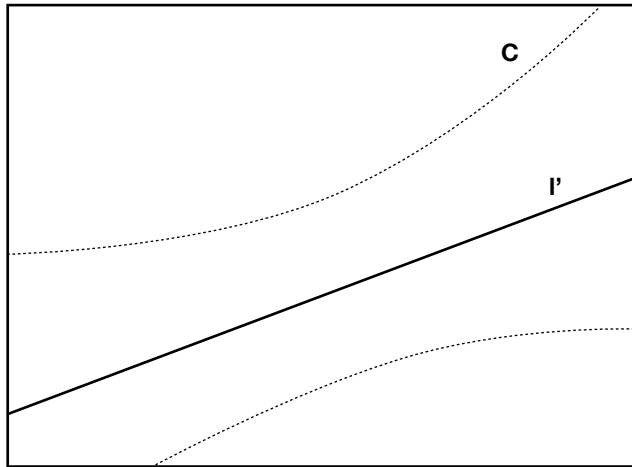


Fig. 3.9. Epipolar band $\mathbf{C}$ surrounding epipolar line $\mathbf{P}$ for a given confidence α

assumed that epipolar lines $\mathbf{I}'$ follow a gaussian model with mean $E[\mathbf{I}']$ and covariance Σ_1 , and where we made a first-order approximation. Hence expression (3.238) will only be valid in situations with small variance and for epipolar lines close to the mean. Notice as well that as parameter k^2 (i.e. Mahalanobis distance) increases, then the surrounding epipolar band area increases as well because the likelihood of finding all possible epipolar lines inside the band increases.

In other words, when $k = 0$, the conic degenerates to $E[\mathbf{I}'] \cdot E[\mathbf{I}'^T]$ which is a single line, and as k increases the conic becomes an hyperbola whose branches lie on both sides on epipolar line $E[\mathbf{I}']$, as depicted in figure 3.9.

Notice as well, that because of expression (3.218), then k^2 follow a chi-square distribution χ_n^2 (where $n = 2$ because the covariance matrix Σ_f has rank 2). Hence by means of the cumulative chi-squared distribution (which is generally available in table form) we can find the value of k to ensure that with certain probability α all epipolar lines lie within the region defined by conic $\mathbf{C}'$ (i.e. an hyperbola). For instance, to have a confidence value of 95% ($\alpha = 0.95$), then k^2 must be $k^2 = 5.9915$, however, because of the first-order approximation we made, such high confidence values usually do not fit very well on the assumption of Gaussian distribution of errors.

Finally, it must be noticed that a conic $\mathbf{C}$ for epipolar lines $\mathbf{l}$ on the first view can also be obtained by means of expression (3.238), just as follows:

$$\mathbf{C} \sim E[\mathbf{l}] \cdot E[\mathbf{l}^T] - k^2 \cdot \Sigma_l^+ \quad (3.240)$$

where now the covariance of epipolar line $\mathbf{l}$ is defined analogously to (3.218) as:

$$\Sigma_l = \mathbf{J}_l \cdot \Sigma_f \cdot \mathbf{J}_l^T \quad (3.241)$$

where the covariance of fundamental matrix Σ_f can be obtained as shown in section 3.2.9.1, and the Jacobian of $\mathbf{l}$ respect to the terms of the fundamental matrix vector $\mathbf{f}$ is defined as:

$$\mathbf{J}_l = \frac{\partial \mathbf{l}(\mathbf{f}, \mathbf{m}')}{\partial \mathbf{f}} \quad (3.242)$$

where epipolar line $\mathbf{l}$ can now be obtained from fundamental matrix $\mathbf{F}$ using expression (3.5), just as follows:

$$\mathbf{l} = \mathbf{F}^T \cdot \mathbf{m}' = \begin{bmatrix} f_{11} & f_{21} & f_{31} \\ f_{12} & f_{22} & f_{32} \\ f_{13} & f_{23} & f_{33} \end{bmatrix} \cdot \begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} f_{11} \cdot x' + f_{21} \cdot y' + f_{31} \cdot w' \\ f_{12} \cdot x' + f_{22} \cdot y' + f_{32} \cdot w' \\ f_{13} \cdot x' + f_{23} \cdot y' + f_{33} \cdot w' \end{bmatrix} \quad (3.243)$$

And consequently, expression (3.242) can be expanded as:

$$\mathbf{J}_l = \frac{\partial \mathbf{l}(\mathbf{f}, \mathbf{m}')}{\partial \mathbf{f}} = \frac{\partial \mathbf{l}(\mathbf{f}, \mathbf{m}')}{\partial (f_{11}, f_{12}, \dots, f_{33})} = \begin{bmatrix} x' & 0 & 0 & y' & 0 & 0 & w' & 0 & 0 \\ 0 & x' & 0 & 0 & y' & 0 & 0 & w' & 0 \\ 0 & 0 & x' & 0 & 0 & y' & 0 & 0 & w' \end{bmatrix} \quad (3.244)$$

3.2.9.3.- The uncertainty of epipoles and matched image points

In (3.239) we saw the conic $\mathbf{C}'$ that embeds with a given likelihood all possible epipolar lines $\mathbf{l}'$ generated by a given image point $\mathbf{m}$. As a consequence, any point $\mathbf{m}'$ corresponding to that image point $\mathbf{m}$ must be located inside of that conic $\mathbf{C}'$, and in the extreme case for a given likelihood α , matched points $\mathbf{m}'$ laying on epipolar line $\mathbf{l}'$ can lie on the conic $\mathbf{C}'$. Therefore:

$$\mathbf{m}'^T \cdot \mathbf{C}' \cdot \mathbf{m}' = 0 \quad (3.245)$$

which can be expanded by means of (3.239) as:

$$\mathbf{m}'^T \cdot (E[\mathbf{l}'] \cdot E[\mathbf{l}'^T] - k^2 \cdot \Sigma_l^+) \cdot \mathbf{m}' = 0 \quad (3.246)$$

where we assume that the matched point $\mathbf{m}'$ is known and fixed, and therefore the remaining unknown on epipolar line $\mathbf{l}'$ is the epipole location $\mathbf{e}'$. Hence taking into account that epipolar line $\mathbf{l}'$ passes through epipole $\mathbf{e}'$ and point $\mathbf{m}'$, it can be expressed by means of expression (1.9) as:

$$\mathbf{l}' = [\mathbf{e}']_{\times} \cdot \mathbf{m}' = [\mathbf{m}']_{\times}^T \cdot \mathbf{e}' \quad (3.247)$$

And consequently, assuming that $\mathbf{m}'$ is fixed, (3.246) can be rewritten as follows:

$$\begin{aligned} \mathbf{m}'^T \cdot \left(E[\mathbf{l}'] \cdot E[\mathbf{l}'^T] - k^2 \cdot \Sigma_{\mathbf{l}'}^+ \right) \cdot \mathbf{m}' &= 0 \Rightarrow \\ \mathbf{m}'^T \cdot \left(E\left[[\mathbf{m}']_{\times}^T \cdot \mathbf{e}' \right] \cdot E\left[\left([\mathbf{m}']_{\times}^T \cdot \mathbf{e}' \right)^T \right] - k^2 \cdot \Sigma_{\mathbf{l}'}^+ \right) \cdot \mathbf{m}' &= 0 \Rightarrow \\ \mathbf{m}'^T \cdot \left([\mathbf{m}']_{\times}^T \cdot E[\mathbf{e}'] \cdot E[\mathbf{e}'^T] [\mathbf{m}']_{\times} - k^2 \cdot \Sigma_{\mathbf{l}'}^+ \right) \cdot \mathbf{m}' &= 0 \Rightarrow \\ \mathbf{m}'^T \cdot [\mathbf{m}']_{\times}^T \cdot E[\mathbf{e}'] \cdot E[\mathbf{e}'^T] [\mathbf{m}']_{\times} \cdot \mathbf{m}' - k^2 \cdot \mathbf{m}'^T \cdot \Sigma_{\mathbf{l}'}^+ \cdot \mathbf{m}' &= 0 \Rightarrow \\ \mathbf{0}_3^T \cdot E[\mathbf{e}'] \cdot E[\mathbf{e}'^T] [\mathbf{m}']_{\times} \cdot \mathbf{m}' - k^2 \cdot \mathbf{m}'^T \cdot \Sigma_{\mathbf{l}'}^+ \cdot \mathbf{m}' &= 0 \Rightarrow \\ -k^2 \cdot \mathbf{m}'^T \cdot \Sigma_{\mathbf{l}'}^+ \cdot \mathbf{m}' &= 0 \Rightarrow \\ \mathbf{m}'^T \cdot \left(k^2 \cdot \Sigma_{\mathbf{l}'}^+ \right) \cdot \mathbf{m}' &= 0 \Rightarrow \\ \mathbf{m}'^T \cdot \mathbf{C}'_{\mathbf{m}} \cdot \mathbf{m}' &= 0 \end{aligned} \quad (3.248)$$

Therefore, all points $\mathbf{m}'$ on the second view corresponding to $\mathbf{m}$ on the first view, must lie in a conic indicating the uncertainty of such correspondence:

$$\mathbf{C}'_{\mathbf{m}} = k^2 \cdot \Sigma_{\mathbf{l}'}^+ \quad (3.249)$$

with a given likelihood α .

Finally, using expressions (3.222) and (3.223) we obtain that (3.249) can be expressed as:

$$\begin{aligned} \mathbf{C}'_{\mathbf{m}} &= k^2 \cdot \Sigma_{\mathbf{l}'}^+ = k^2 \cdot \left(\mathbf{U}^T \cdot \hat{\Sigma}_{\mathbf{l}'} \cdot \mathbf{U} \right)^+ \\ &= k^2 \cdot \mathbf{U}^T \cdot \hat{\Sigma}_{\mathbf{l}'}^+ \cdot \mathbf{U} = k^2 \cdot \mathbf{U}^T \cdot \begin{bmatrix} \tilde{\Lambda}_{\mathbf{l}'}^{-1} & \mathbf{0}_2 \\ \mathbf{0}_2^T & 0 \end{bmatrix} \cdot \mathbf{U} = \\ &= \mathbf{U}^T \cdot \begin{bmatrix} k^2 \cdot \tilde{\Lambda}_{\mathbf{l}'}^{-1} & \mathbf{0}_2 \\ \mathbf{0}_2^T & 0 \end{bmatrix} \cdot \mathbf{U} \end{aligned} \quad (3.250)$$

which can be seen as the eigen-decomposition of $\mathbf{C}'_{\mathbf{m}}$

Therefore, $\mathbf{C}'_{\mathbf{m}}$ is a rank 2 conic, with two positive eigenvalues equal to:

$$\lambda'_1 = \frac{k^2}{\sigma_{\mathbf{l}'1}^2}, \quad \lambda'_2 = \frac{k^2}{\sigma_{\mathbf{l}'2}^2} \quad (3.251)$$

where $\sigma_{l'1}^2$ and $\sigma_{l'2}^2$ signal2 are the variances of the first and second components of the epipolar line vector $\mathbf{l}'$ where image point $\mathbf{m}'$ is laying. As can be seen such components indicate the uncertainty on the direction of the epipolar line $\mathbf{l}'$.

Therefore because $\mathbf{C}_m$ has only 2 positive eigenvalues, it can be observed that the conic where image point $\mathbf{m}'$ lies corresponds to an ellipse whose minor and major axes have sizes λ'_1 , λ'_2 defined by the non-null eigenvalues, and the direction of such axes is defined by the eigenvectors $\mathbf{u}_1$, $\mathbf{u}_2$ corresponding to such eigenvalues λ'_1 and λ'_2 .

As in the case of epipolar lines, analogous expressions can be obtained for the first view, so that:

$$\mathbf{m}^T \cdot \mathbf{C}_m \cdot \mathbf{m} = 0$$

$$\mathbf{C}_m = k^2 \cdot \Sigma_1^+ = k^2 \cdot \mathbf{U}^T \cdot \hat{\Sigma}_1^+ \cdot \mathbf{U} = \mathbf{U}^T \cdot \begin{bmatrix} k^2 \cdot \tilde{\Lambda}_1^{-1} & \mathbf{0}_2 \\ \mathbf{0}_2^T & 0 \end{bmatrix} \cdot \mathbf{U} \quad (3.252)$$

And in this case $\mathbf{C}_m$ is the uncertainty ellipse for matched points $\mathbf{m}$ on the first view.

3.2.9.4.- The uncertainty of projective 3D reconstruction

We know that projection of 3D world points into image points is represented by expression (2.3) $\mathbf{m} = \mathbf{P} \cdot \mathbf{M}$, therefore expression (3.252) can also be written as:

$$\mathbf{m}'^T \cdot \mathbf{C}'_m \cdot \mathbf{m}' = \mathbf{M}^T \cdot (\mathbf{P}^T \cdot \mathbf{C}_m \cdot \mathbf{P}) \cdot \mathbf{M} = \mathbf{M}^T \cdot \mathbf{Q}_M \cdot \mathbf{M} = 0 \quad (3.253)$$

where now the term between $\mathbf{Q}_M$ is a 4x4 symmetric matrix than can be seen as the quadric of uncertainty where reconstructed world points $\mathbf{M}$ lie.

$$\mathbf{Q}_M = \mathbf{P}^T \cdot \mathbf{C}_m \cdot \mathbf{P} \quad (3.254)$$

And assuming the camera model given in (2.24) $\mathbf{P} = \mathbf{K} \cdot [\mathbf{R} \mid \mathbf{t}]$, and using expression (3.252), then (3.254) can be expanded as follows:

$$\begin{aligned} \mathbf{Q}_M &= \mathbf{P}^T \cdot \mathbf{C}_m \cdot \mathbf{P} = k^2 \cdot \begin{bmatrix} \mathbf{R}^T \\ \mathbf{t}^T \end{bmatrix} \cdot \mathbf{K}^T \cdot \mathbf{U}^T \cdot \hat{\Sigma}_1^+ \cdot \mathbf{U} \cdot \mathbf{K} \cdot \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} \\ &= k^2 \cdot \begin{bmatrix} (\mathbf{U} \cdot \mathbf{K} \cdot \mathbf{R})^T \\ (\mathbf{U} \cdot \mathbf{K} \cdot \mathbf{t})^T \end{bmatrix} \cdot \hat{\Sigma}_1^+ \cdot \begin{bmatrix} \mathbf{U} \cdot \mathbf{K} \cdot \mathbf{R} & \mathbf{U} \cdot \mathbf{K} \cdot \mathbf{t} \end{bmatrix} \\ &= \begin{bmatrix} (\mathbf{U} \cdot \mathbf{K} \cdot \mathbf{R})^T \cdot k^2 \cdot \hat{\Sigma}_1^+ \cdot (\mathbf{U} \cdot \mathbf{K} \cdot \mathbf{R}) & (\mathbf{U} \cdot \mathbf{K} \cdot \mathbf{R})^T \cdot k^2 \cdot \hat{\Sigma}_1^+ \cdot (\mathbf{U} \cdot \mathbf{K} \cdot \mathbf{t}) \\ (\mathbf{U} \cdot \mathbf{K} \cdot \mathbf{t})^T \cdot k^2 \cdot \hat{\Sigma}_1^+ \cdot (\mathbf{U} \cdot \mathbf{K} \cdot \mathbf{R}) & (\mathbf{U} \cdot \mathbf{K} \cdot \mathbf{t})^T \cdot k^2 \cdot \hat{\Sigma}_1^+ \cdot (\mathbf{U} \cdot \mathbf{K} \cdot \mathbf{t}) \end{bmatrix} \end{aligned} \quad (3.255)$$

where it can be clearly seen that $\mathbf{Q}_M$ is a 4x4 symmetric matrix.

We'd like now to determine the shape that the quadric $\mathbf{Q}_M$ represents, for that reason we will attempt to find its eigen-decomposition, in the following form:

$$\mathbf{Q}_M = \mathbf{U}_M \cdot \Lambda_M \cdot \mathbf{U}_M^T \quad (3.256)$$

where $\mathbf{U}_M$ is a 4x4 orthogonal matrix and Λ_M is the following 4x4 diagonal matrix,

$$\Lambda_M = \text{diag}(A, B, C, D) \quad (3.257)$$

Therefore, because $\mathbf{U}_M$ must be orthogonal, it must be true that $\mathbf{U}_M \cdot \mathbf{U}_M^T$ is a diagonal matrix up to scale.

Hence, we can assume that the 4x4 orthogonal matrix $\mathbf{U}_M$ can have the following form:

$$\mathbf{U}_M = \begin{bmatrix} (\mathbf{U} \cdot \mathbf{K} \cdot \mathbf{R})^T & \mathbf{a} \\ (\mathbf{U} \cdot \mathbf{K} \cdot \mathbf{t})^T & b \end{bmatrix} \quad (3.258)$$

where we know that the former 3 columns are orthonormal because the eigenvectors $\mathbf{U}$ of the pseudo-inverse of the covariance of epipolar line Σ^+ are orthonormal, and because matrix $\mathbf{R}$ is orthonormal and $\mathbf{K}$ is upper triangular (i.e its determinant is the product of its diagonal elements, and therefore is not singular, and consequently neither is the product $\mathbf{U} \cdot \mathbf{K} \cdot \mathbf{R}$). However, we need to find the last column of $\mathbf{U}_M$ defined as $(\mathbf{a}^T, b)^T$ because $\mathbf{U}_M$ needs to be a 4x4 orthogonal matrix. For the moment we will assume that this column is found by some orthogonalization method, although, as it will be seen next, we won't need to explicitly find the values of such column $(\mathbf{a}^T, b)^T$.

Therefore, by combining (3.256) and (3.257) we obtain:

$$\begin{aligned} \mathbf{Q}_M &= \mathbf{U}_M \cdot \Lambda_M \cdot \mathbf{U}_M^T = \\ &= \begin{bmatrix} (\mathbf{U} \cdot \mathbf{K} \cdot \mathbf{R})^T & \mathbf{a} \\ (\mathbf{U} \cdot \mathbf{K} \cdot \mathbf{t})^T & b \end{bmatrix} \cdot \text{diag}(A, B, C, D) \cdot \begin{bmatrix} \mathbf{U} \cdot \mathbf{K} \cdot \mathbf{R} & \mathbf{U} \cdot \mathbf{K} \cdot \mathbf{t} \\ \mathbf{a}^T & b \end{bmatrix} = \\ &= \begin{bmatrix} (\mathbf{U} \cdot \mathbf{K} \cdot \mathbf{R})^T \cdot \text{diag}(A, B, C) & D \cdot \mathbf{a} \\ (\mathbf{U} \cdot \mathbf{K} \cdot \mathbf{t})^T \cdot \text{diag}(A, B, C) & b \cdot D \end{bmatrix} \cdot \begin{bmatrix} \mathbf{U} \cdot \mathbf{K} \cdot \mathbf{R} & \mathbf{U} \cdot \mathbf{K} \cdot \mathbf{t} \\ \mathbf{a}^T & b \end{bmatrix} = \\ &= \begin{bmatrix} (\mathbf{U} \cdot \mathbf{K} \cdot \mathbf{R})^T \cdot \text{diag}(A, B, C) \cdot (\mathbf{U} \cdot \mathbf{K} \cdot \mathbf{R}) + D \cdot \mathbf{a} \cdot \mathbf{a}^T & (\mathbf{U} \cdot \mathbf{K} \cdot \mathbf{R})^T \cdot \text{diag}(A, B, C) \cdot (\mathbf{U} \cdot \mathbf{K} \cdot \mathbf{t}) + D \cdot b \cdot \mathbf{a}^T \\ (\mathbf{U} \cdot \mathbf{K} \cdot \mathbf{t})^T \cdot \text{diag}(A, B, C) \cdot (\mathbf{U} \cdot \mathbf{K} \cdot \mathbf{R}) + D \cdot b \cdot \mathbf{a}^T & (\mathbf{U} \cdot \mathbf{K} \cdot \mathbf{t})^T \cdot \text{diag}(A, B, C) \cdot (\mathbf{U} \cdot \mathbf{K} \cdot \mathbf{t}) + D \cdot b^2 \end{bmatrix} \end{aligned} \quad (3.259)$$

and because we want (3.259) to be equal to (3.255), then it must be true that,

$$\text{diag}(A, B, C) = k^2 \cdot \hat{\Sigma}_1^+ = \begin{bmatrix} k^2 \cdot \tilde{\Lambda}_1^{-1} & \mathbf{0}_2 \\ \mathbf{0}_2^T & 0 \end{bmatrix} = \begin{bmatrix} k^2 / \sigma_{11}^2 & 0 & 0 \\ 0 & k^2 / \sigma_{12}^2 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad (3.260)$$

$$D = 0$$

And consequently the diagonal matrix in (3.257) containing the eigenvalues of quadric QM becomes:

$$\Lambda_{\mathbf{M}} = \text{diag}(A, B, C, D) = \begin{bmatrix} k^2 \cdot \hat{\Sigma}_1^+ & \mathbf{0}_3 \\ \mathbf{0}_3^T & 0 \end{bmatrix} = \begin{bmatrix} k^2 / \sigma_{11}^2 & 0 & 0 & 0 \\ 0 & k^2 / \sigma_{12}^2 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad (3.261)$$

Therefore, by simple inspection of the eigenvalues of quadric $\mathbf{Q}_{\mathbf{M}}$, it is obvious that such quadric has rank-2 and it is degenerate. Besides, because the non-null eigenvalues are both positive, we can then also state that such quadric $\mathbf{Q}_{\mathbf{M}}$ corresponds to an elliptic cylinder indicating the uncertainty of back-projection of a ray of light that projects into a point $\mathbf{m}$ on the first view.

The same analysis can be done for the second view, obtaining a second cylinder of uncertainty after back-projecting the corresponding ray of light for image point $\mathbf{m}'$, as depicted in figure 3.10.

It can then be observed, that for a given desired level of confidence alpha, then the volume containing the intersection of the cylinders of uncertainty of back-projection of image points, is the region where the reconstructed 3D world point will lie. This intersection is shown in figure 3.10 along with the uncertainty cylinders of both views.

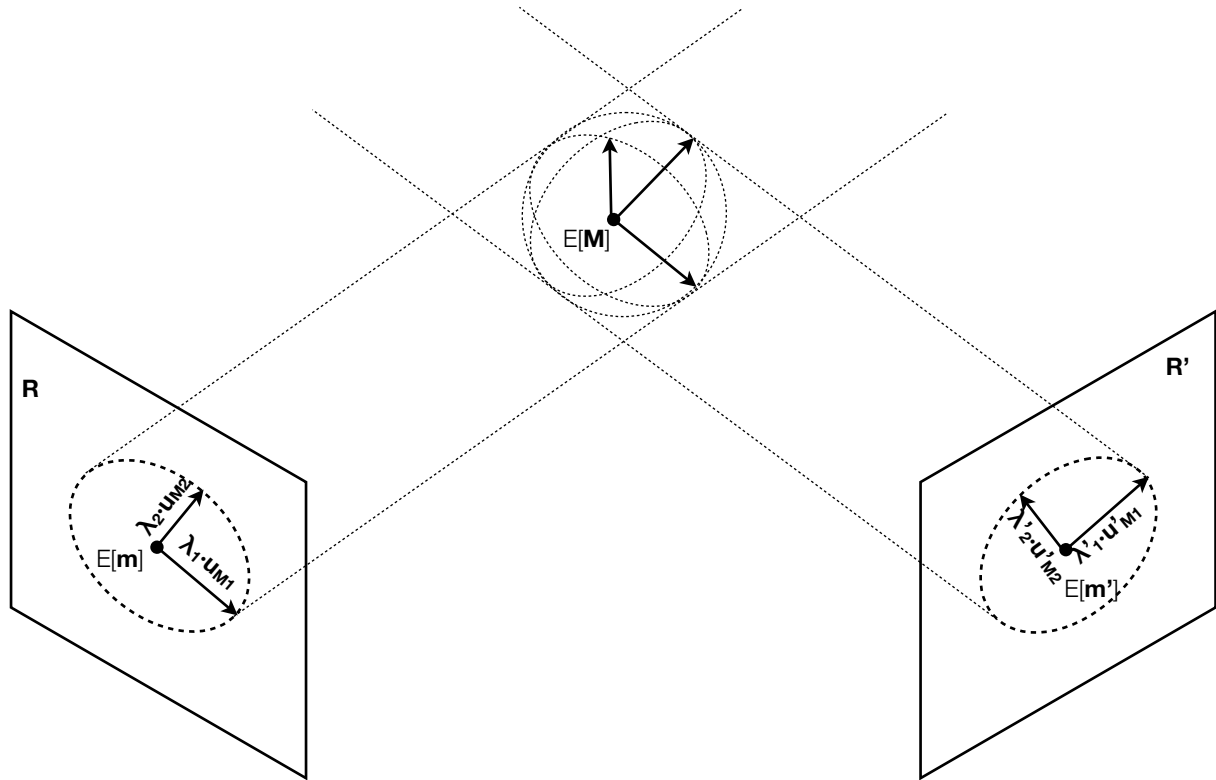


Fig. 3.10. Uncertainty of back-projection and triangulation. Cylinders of uncertainty after back-projecting from left ($\mathbf{R}$) and right ($\mathbf{R}'$) views are shown. The intersection of both cylinders create a rhomboid shape of uncertainty that can be approximated by an ellipsoid of uncertainty. The remaining symbols have the following meaning: $E[\mathbf{m}]$ mean value of left image point. $E[\mathbf{m}']$ mean value of right image point. $E[\mathbf{M}]$ mean value of triangulated 3D world point. $\lambda_1 \cdot \mathbf{u}_{M1}$ major semi-axis of ellipse of uncertainty on left view. $\lambda_2 \cdot \mathbf{u}_{M2}$ minor semi-axis of ellipse of uncertainty on left view. $\lambda'_1 \cdot \mathbf{u}'_{M1}$ major semi-axis of ellipse of uncertainty on right view and $\lambda'_2 \cdot \mathbf{u}'_{M2}$ minor semi-axis of ellipse of uncertainty on right view.

The resulting shape of the region of uncertainty for triangulated 3D world points will be a 3D rhomboid-like volume, however, for simplification, specially when more views are added, we will simplify such shape into an ellipsoid by means of eigen-decomposition.

It can be observed that the rhomboid shapes will change depending on the baseline configuration of the pair of cameras as well as the depth of the reconstructed world point. Indeed, these rhomboids correspond to the ones shown in figures 2.13 or 9.41, and hence are related to the precision of the reconstruction as was shown in section 2.6.2.

In order to simplify the rhomboid shapes of uncertainty, we will stack together all the eigenvectors and their corresponding non-null eigenvalues of all the cylinders of uncertainty in both views. This means that we will use 2 eigen-vectors with their corresponding eigen-values for each view, because we have seen that each cylinder of uncertainty has rank-2 (i.e. two non-null eigenvalues). Hence,

$$\mathbf{Q}_{3D} = \begin{bmatrix} \mathbf{u}_{M1} & \mathbf{u}_{M2} & \mathbf{u}'_{M1} & \mathbf{u}'_{M2} \end{bmatrix} \cdot \begin{bmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \lambda'_1 & \\ & & & \lambda_2 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{u}_{M1}^T \\ \mathbf{u}_{M2}^T \\ \mathbf{u}'_{M1}^T \\ \mathbf{u}'_{M2}^T \end{bmatrix} \quad (3.263)$$

And finally, by means of eigen-decomposition of $\mathbf{Q}_{3D}$ we will be able to choose the 3 eigenvectors corresponding to the largest 3 eigen-values that will define the 3 axis of the ellipsoid of uncertainty of the reconstructed 3D world point.

$$\begin{aligned} \mathbf{Q}_{3D} &= \mathbf{U}_{3D} \cdot \mathbf{\Lambda}_{3D} \cdot \mathbf{U}_{3D}^T = \mathbf{U}_{3D} \cdot \text{diag}(\lambda_1, \lambda_2, \lambda_3, \lambda_4) \cdot \mathbf{U}_{3D}^T \Rightarrow \\ \tilde{\mathbf{Q}}_{3D} &= \mathbf{U}_{3D} \cdot \text{diag}(\lambda_1, \lambda_2, \lambda_3, 0) \cdot \mathbf{U}_{3D}^T \end{aligned} \quad (3.264)$$

As will be seen in the following chapter, in order to reconstruct 3D points, more than 2 views can be used if available. In those cases $\mathbf{Q}_{3D}$ will be formed by stacking all the eigenvectors with their corresponding non-null eigenvalues for all available views, and again by means of eigen-decomposition of $\mathbf{Q}_{3D}$ we can obtain an approximation to the quadric by choosing the eigen-vectors corresponding to the 3 largest eigen-values, just as was shown in (3.264).

3.3.- Three-view geometry

In this chapter we have seen with great detail the epipolar geometry arisen from 2 views of the same fixed scene. However, when more views are available, additional geometric properties can be inferred.

We will briefly indicate some of these properties for the sake of completeness of multiple view theory. However, for the purpose of the work shown in this document, only 2-view geometry will be used due to its lower computational complexity.

Therefore when 3 views of the same scene are available, they can be analyzed as 2 pairs of views using the epipolar geometry seen in section 3.2. This produces 2 epipolar planes for each world point, with their corresponding epipolar lines on each view produced by the intersection of such epipolar planes with the retinal planes, as depicted in figure 3.11.

It can be seen in figure 3.11 that when the fundamental matrices relating views $\mathbf{R}$ and $\mathbf{R}''$ or $\mathbf{R}'$ and $\mathbf{R}''$ are known, then the epipolar lines on view $\mathbf{R}''$ generated by points $\mathbf{m}$ and $\mathbf{m}'$ can be obtained. It can then be seen that the matched point $\mathbf{m}''$ can be found as the intersection of the pair of epipolar lines on view $\mathbf{R}''$ generated by points $\mathbf{m}$ and $\mathbf{m}'$ on the first and second views. This analysis can be done as long as the world point $\mathbf{M}$ does not lie on the trifocal plane (i.e. the plane going through the three camera centers of projection), analogously to the case when point $\mathbf{M}$ was located in the baseline between two camera centers for the two view case.

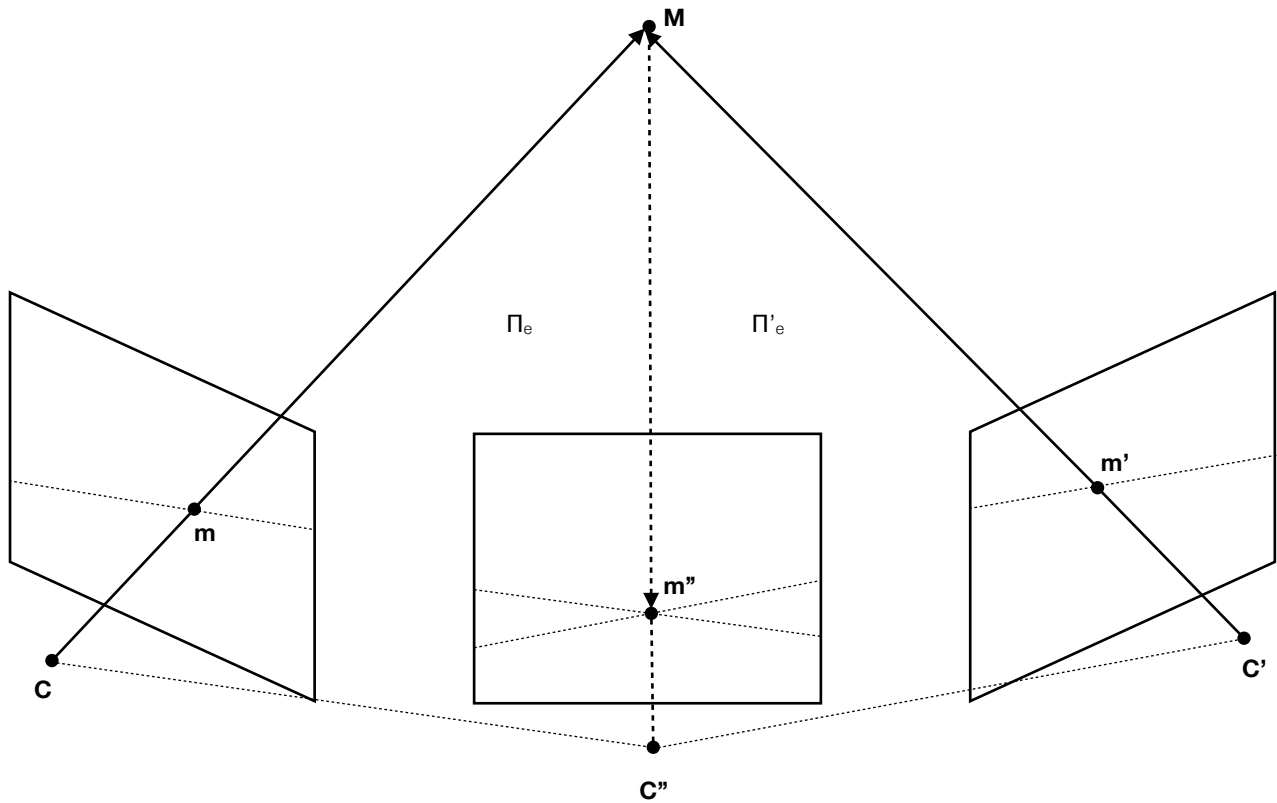


Fig. 3.11. Three view geometry. Relation between the image of a point in three views is shown. Epipolar lines of points m and m' can be used to obtain m'' on the third view.

Analogously to the two-view geometry case, where the fundamental matrix defined such geometry, in the three view case, all geometry relations are defined by the trifocal tensor T , which is a $3 \times 3 \times 3$ tensor containing 27 elements, where only 18 of them are independent due to additional nonlinear constraints. Further information about three view geometry, the trifocal tensor and possible extensions of the trifocal tensor to further views (i.e. the quadrifocal tensor) can be found on [8].

4.- Structure and motion

4.1.- Introduction

In chapter 3 we saw how pairs of views are related by means of epipolar geometry and the fundamental matrix, and we provided techniques to estimate such relations, which allowed us to retrieve an initial pair of cameras in an arbitrary projective stratum if the calibration matrices were unknown, or in a metric stratum by means of the essential matrix if the camera calibration matrices were known.

Then, using the pair of retrieved cameras and the correspondence between image points (which can be found by means of dense disparity estimation as will be shown in chapter 9), we were able to triangulate points into an arbitrary projective stratum (or metric stratum if calibration matrices are known). Besides, in chapter 3 we provided means to correct errors in image point correspondences in arbitrary projective stratums so that triangulations were as accurate as possible.

In this chapter we will give an overview of the techniques needed to obtain the initial structure of the scene and deal with the problem of adding more views into the 3D reconstruction so that point triangulation uncertainty improves for points already present on the first two views, and so that more points can be added into the reconstruction in order to fill holes and occlusions.

We will also briefly describe some techniques to obtain point meshes to be represented by current 3D hardware, as well as some texturing techniques for the estimated reconstruction.

4.2.- Initial structure and motion

The first and most important step for 3D reconstruction consists on accurately selecting the first pair of views to initialize the 3D structure in an arbitrary projective stratum.

We already saw in section 3.2.8 that depending on the configuration of the pairs of cameras or the world points, the epipolar geometry might become degenerate. Hence, we need to ensure that no epipolar degeneracy happens on the first pair of views, otherwise, the resulting reconstruction would completely fail.

For that reason, in order to avoid epipolar degeneracy, we need to ensure that cameras' centers are separate enough, and world points do not lie on a ruled quadric (i.e. a plane, etc). If points are located in a ruled quadric, a small trick can be used in order to avoid degeneracy, which consists on adding some random noise to the matched image points, although at the expense of loosing resolution. However, this trick in general won't be used, and the pair of images will simply be discarded as discussed next.

Hence, in general, the first restriction that must be taken into account is to ensure that cameras have moved enough to have separated centers. This is specially crucial in video sequences, where cameras might not have moved enough in consecutive frames. Consequently, this restriction allows us to skip some frames if epipolar degeneracy is found.

On the other hand in section 2.6.2, we saw that resolution of triangulated world points increased as the baseline (i.e. separation between centers of cameras) increased as well for a given fixed resolution on image point correspondence (i.e. disparity). However, this effect can be alleviated when using sub-pixel accuracy on point correspondences, as will be seen on section 9.5.1.

Hence, if no sub-pixel accuracy is available, we will need to ensure that cameras are as much separated as possible in order to increase 3D resolution and avoid epipolar degeneracy, taking into account that usually disparity estimation becomes less reliable as points of view become more separated. Then, we have a trade-off between resolution/epipolar degeneracy vs accuracy of estimated image point correspondences. This is dealt with more detail in [7].

On the other hand, when using sub-pixel accuracy, it will be usually enough just to ensure that cameras have enough separation to avoid epipolar degeneracy. This can be simply checked when using the 7-point estimation for the fundamental matrix as seen on section 3.2.2.2, along with a robust estimation method such as RANSAC, as was shown in section 3.2.3.

Of course, if we can also find the best solution for the tradeoff between resolution vs accuracy of matches, then the 3D reconstruction will improve. However, due to the robustness of the sub-pixel disparity method based on multiresolution that is shown in chapter 9, usually, it will be enough just to avoid epipolar degeneracy, without requiring further constraints. The whole process to obtain the best initial pair of images is summarized on algorithm 4.1.

Hence, assuming that the best pair of views has been chosen to estimate the epipolar geometry (i.e. fundamental matrix), we will obtain the first pair of cameras in order to initialize the 3D structure.

In our experiments we found that attempting to find a quasi-euclidean initial frame for the first pair of cameras is desirable as it provides better results when adding more views into the reconstruction.

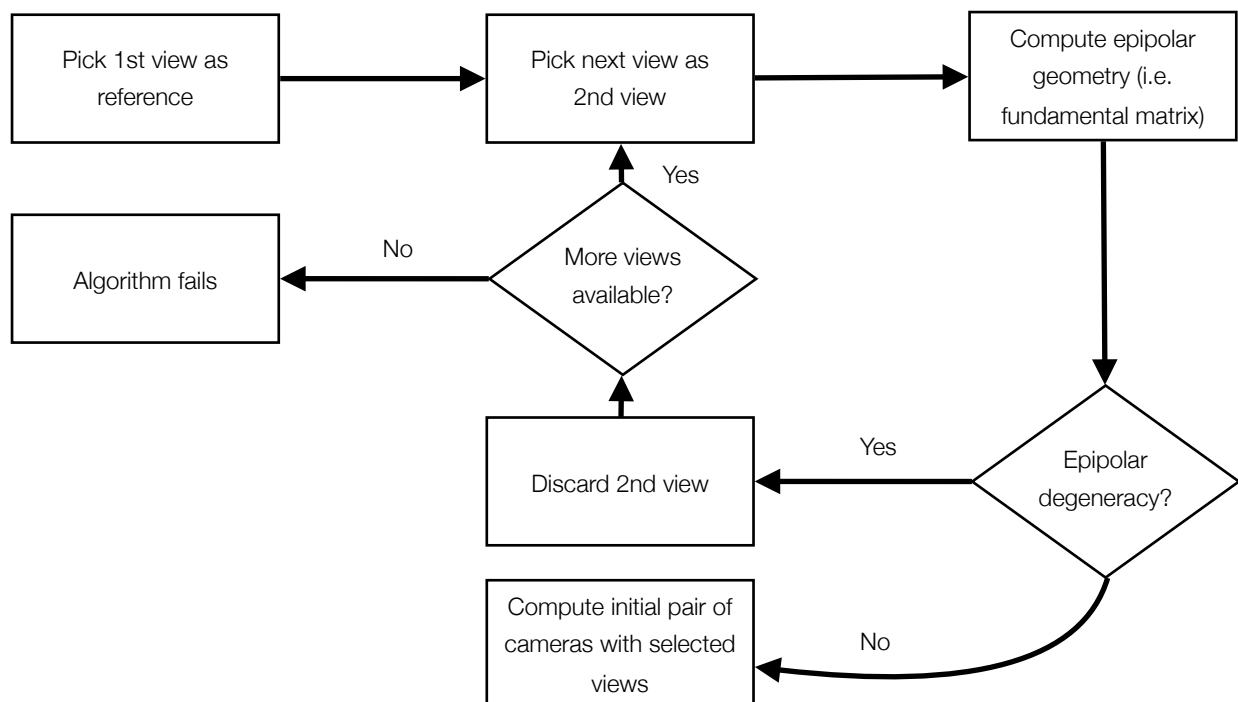
Hence, in order to find such initial quasi-euclidean frame, different techniques exist, as explained in [7]. However, we found experimentally that using the essential matrix with a typical calibration matrix proved to be the most reliable technique. This will be shown next.

4.2.1.- Obtaining an initial quasi-euclidean reconstruction

When attempting to find an initial reconstruction framework, the first step will always consist on finding the epipolar geometry by means of the fundamental matrix.

In section 3.2.2 we already saw the 8-point algorithm and the 7-point algorithm that allow us to find the fundamental matrix, typically providing better results after point normalization as seen in section 3.2.2.1.1. However, when using dense disparity

Algorithm 4.1. Choosing the best initial pair of images



to estimate correspondences of image points, then it is possible to use robust techniques such as RANSAC along with other typical ones such as the 7-point algorithm with a reasonable convergence of the RANSAC algorithm due to the large number of available correspondences and the large rate of inliers if the disparity estimation has enough quality (which is true for the multiresolution technique shown in chapter 9).

After the fundamental matrix $\mathbf{F}$ has been estimated, we can then assume a typical calibration matrix for the initial pair of cameras such as:

$$\mathbf{K} = \mathbf{K}' = \begin{bmatrix} w+h & 0 & w/2 \\ 0 & w+h & h/2 \\ 0 & 0 & 1 \end{bmatrix} \quad (4.1)$$

where w is image width in pixels, and h is image height in pixels. These calibration matrices $\mathbf{K}$ and $\mathbf{K}'$ will be reasonable for any camera with a focal length of about 60 mm, which is valid for most conventional cameras.

Therefore, by means of expression (3.176), we can obtain the essential matrix $\mathbf{E}$ as follows:

$$\mathbf{E} = \mathbf{K}'^T \cdot \mathbf{F} \cdot \mathbf{K} = \mathbf{K}^T \cdot \mathbf{F} \cdot \mathbf{K} \quad (4.2)$$

Then, before attempting to find the initial pair of cameras as described in section 3.2.7, in order to accurately triangulate image points for the possible 4 solutions of the pair of cameras, we will first correct point correspondences using either the Sampson approximation or the Gold Standard algorithms seen in section 3.2.5 when the epipolar geometry is known.

Then, once the points have been corrected, they can be normalized using the inverse of the calibration matrix $\mathbf{K}$ as shown in expression (3.174), and then, using the corrected and normalized point correspondences along with the estimated essential matrix $\mathbf{E}$, we can proceed to compute the initial normalized pair of cameras,

$$\hat{\mathbf{P}} = \mathbf{K}^{-1} \cdot \mathbf{P}, \quad \hat{\mathbf{P}}' = \mathbf{K}'^{-1} \cdot \mathbf{P}' \quad (4.3)$$

using the technique shown in section 3.2.7.2 .

As we explained in such section, pairs of points will need to be triangulated for each of the 4 possible solutions, and then pick the solution that produces the largest amount of correct cheirality on both views for the triangulated points, to ensure that world points lie in front of both cameras $\mathbf{P}$ and $\mathbf{P}'$, as in the real world. As explained in chapter 2, to simplify cheirality computation, we will typically enforce a positive determinant on the top-left 3x3 minor of the camera matrix, to ensure that the principal axis vector points towards the front of the cameras.

Then, once the best initial pair of cameras has been chosen, we will have the normalized initial cameras as well as the triangulated points in the normalized framework, and those points that have valid cheirality.

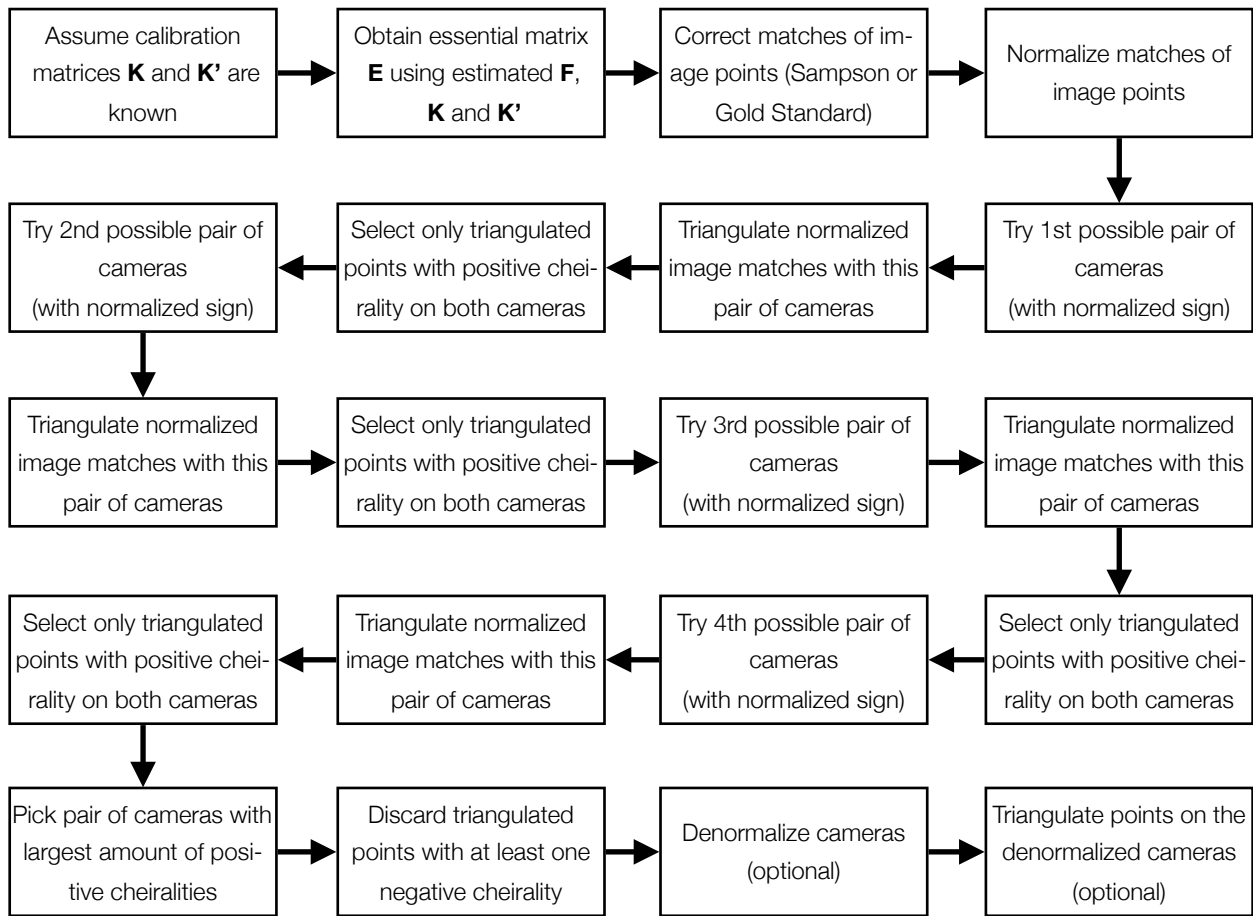
Hence, first of all cameras will have to be de-normalized, which can be easily done by means of (4.3) as:

$$\hat{\mathbf{P}} = \mathbf{K}^{-1} \cdot \mathbf{P}, \quad \hat{\mathbf{P}}' = \mathbf{K}'^{-1} \cdot \mathbf{P}' \quad (4.4)$$

Secondly, triangulated world points will be valid only on the normalized cameras framework. We can then choose to work either on the normalized framework and then attempt to estimate the true calibration matrices as will be explained in chapter 5, or otherwise triangulate points on the de-normalized cameras framework.

Finally, because we have checked cheirality of triangulated points on the normalized framework, when we chose the best solution among the 4 possible pairs of cameras, we then know beforehand which points can be discarded for triangulation on the 3D reconstruction. The whole process is summarized on algorithm 4.2.

Algorithm 4.2. Obtaining a quasi-euclidean initial pair of cameras



4.3.- Adding more views

When attempting to add new views into a reconstruction, the simplest step can consist on relating image points in a new view by means of disparity estimation, to other points in another view where a camera has already been found.

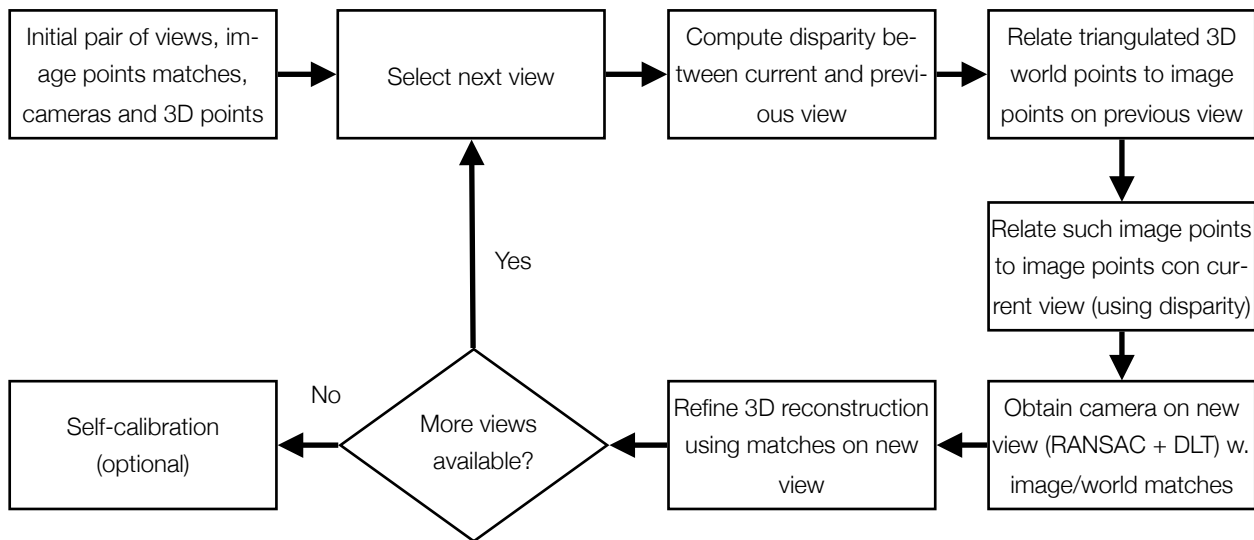
In Layman terms, initially we will have camera P1 and P2 for the first pair of cameras using the method described in section 4.2. Therefore, to add a 3rd view, we would estimate disparity between the second and third view, and because points on the 1st and 2nd view have already been triangulated, then we know that those triangulated world points are related to image points on the third view by means of the estimated disparity. Finally, because using this approach we will have a large set of correspondences between image points and 3D world points, we will be able to estimate the projection camera matrix on the third view by means of DLT algorithm, or using RANSAC along with DLT for a robust solution, just as we saw in section 2.5.

As a reminder, take into account that we will generally normalize camera matrices, after being estimated, to enforce that the top-left 3x3 minor has positive determinant. This way it is ensured that the principal axis vector points towards the front of the estimated cameras and cheirality computation gets simplified as we saw in section 2.4.2.

This process can be repeated indefinitely for new views, as long as enough correspondences are found between image points on a previous view where world points are already available.

Other techniques such as the ones shown in [7] can also be used to determine image point and world point matches not only with the last view on a sequence (which is already valid for video sequences), but with any other close view. This way usually the number of triangulated world points get reduced and the accuracy is further improved when scanning with the

Algorithm 4.3. Adding new views to the 3D reconstruction



camera the same area more than one time, because repeated points are avoided. The whole process of adding more views is summarized on algorithm 4.3.

4.3.1.- Refining and extending structure

When more views are added, as in algorithm 4.3, it happens that some 3D world points appear in more than two views, and as we saw in section 2.6, points can be triangulated if cameras and correspondences of image points are available for at least 2 views. Therefore, it seems reasonable to use the additional views to update world points with LMSE solutions along all the views so that solutions become more robust in case that some errors in image point matches are present.

Indeed, we showed in section 3.2.9.4 how to retrieve the ellipsoid of uncertainty for triangulated points. It can also be shown that when more views are added, then such ellipsoid can be obtained as the intersection of all the cylinders of uncertainty for all views. We saw that such intersection can be obtained by stacking together all the eigenvectors corresponding to non-null eigenvalues of each cylinder of uncertainty on each view, and then by means of eigen-decomposition, choosing only the 3 most significant eigenvalues and eigenvectors which would determine the ellipsoid of uncertainty of the reconstructed 3D world point.

It can be seen that the resulting eigen-decomposition of all the stacked eigenvectors and eigenvalues of all the cylinders of uncertainty can be seen as a process similar to the orthogonalization of all such vectors. Indeed, by using a simple orthogonalization method such as Gram-Schmidt, we can observe that the resulting orthogonal vectors are obtained by subtracting the parallel component to any previous vector on the basis, and therefore, the resulting orthogonal vectors (before normalization), tend to be shorter in modulus than the original non-orthogonal vectors. For that reason, we can then infer that the resulting ellipsoid of uncertainty will become smaller as the number of views increases, as long as the resolution in new views increases as well. This can also be seen from a statistical point of view, because if new measures are added with similar amount of noise, then the resulting variance on the estimated mean measure will reduce as the amount of measures increases. And finally, geometrically it also makes sense that as we increase the number of views, then the solution of the triangulation becomes more accurate.

In section 2.6.1 we showed how points can be triangulated using homogeneous coordinates with LMSE solutions by means of SVD. However, we also showed that due to the homogeneous nature of the LMSE solution, small errors in the last homogeneous coordinate can transform into large errors in the inhomogeneous representation of such points. For that reason in 2.6.1.1 we showed that an LMSE solution in inhomogeneous terms can also be found by means of pseudo-inverse if world points are not located at or near infinity, which is true for the quasi-euclidean framework that we have obtained in section 4.2.1 because the plane at infinity Π_∞ is correctly located (i.e. the image plane does not cross the plane

at infinity producing large distortions on the reconstruction and world points located at infinity) and then typically only some affine distortions are present, which can be later corrected by means of self-calibration as will be seen on chapter 5.

Besides, in [7] we also obtained another method to approximately update the 3D world reconstructions with a lower computational cost as new views are added into the reconstruction.

The method is based on the DLT equations of expression (2.140), however, in this case, previous triangulated point $\mathbf{M}_{k-1}$ is also taken into account to weight the equations as follows:

$$\begin{bmatrix} \frac{1}{\mathbf{p}_{k-1}^{3T} \cdot \mathbf{M}_{k-1}} (\tilde{x}_{k-1} \cdot \mathbf{p}_{k-1}^{3T} - \mathbf{p}_{k-1}^{1T}) \\ \frac{1}{\mathbf{p}_{k-1}^{3T} \cdot \mathbf{M}_{k-1}} (\tilde{y}_{k-1} \cdot \mathbf{p}_{k-1}^{3T} - \mathbf{p}_{k-1}^{2T}) \\ \frac{1}{\mathbf{p}_k^{3T} \cdot \mathbf{M}_{k-1}} (\tilde{x}_k \cdot \mathbf{p}_k^{3T} - \mathbf{p}_k^{1T}) \\ \frac{1}{\mathbf{p}_k^{3T} \cdot \mathbf{M}_{k-1}} (\tilde{y}_k \cdot \mathbf{p}_k^{3T} - \mathbf{p}_k^{2T}) \end{bmatrix} \cdot \mathbf{M}_k = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad (4.5)$$

where, the updated point $\mathbf{M}_k$ on new view k , is obtained by solving the homogeneous system of equations in (4.5), where each equation has been weighted by the last homogeneous image point component obtained by projecting the previously triangulated world point $\mathbf{M}_{k-1}$ on previous $(k-1)$ and new views (k) .

4.4.- Building a mesh of 3D points

When attempting to represent a 3D reconstruction into any computer program, is not enough to have the set of triangulated 3D points, instead it is necessary to also obtain the relation of each 3D point with their neighbor points and create a mesh of points that can be easily represented.

Several techniques exist to create a 3D mesh of triangles from sets of 3D points such as [47-49]. However, most of these techniques assume that orientation of the surface of 3D points is known (or that it has been reliably estimated, which is not trivial), or they assume that 3D points can be sparsely sampled (i.e. as if only points of interest or corners -see section 8.4- are triangulated), which later translates in larger computational effort because volumetric carvings need to be computed in order to obtain an optimal solution that optimizes a given cost function or fusion techniques are employed to remove redundant 3D points on the resulting mesh.

In our case, because we will assume that matched image points are obtained by means of dense disparity estimation techniques as the ones shown in chapter 9, then we will know that each image will have areas that will be triangulated into 3D points. Those areas are usually zones in each view with consistent dense disparity and epipolar geometry.

In such areas, each pixel will be triangulated as a 3D world point because correspondences with another view will be available, and because pixels are consecutively located in a grid of points over a plane, we will know that for most common objects neighboring pixels on an image will correspond to neighbor world points that can be related in a 3D mesh, as depicted in figure 4.1.

This idea alone will generally work very fast because pixels are already ordered on an image, and will produce very good results for pairs of images as long as disparity and triangulated points are accurately computed, as is the case with the multiresolution technique shown in chapter 9.

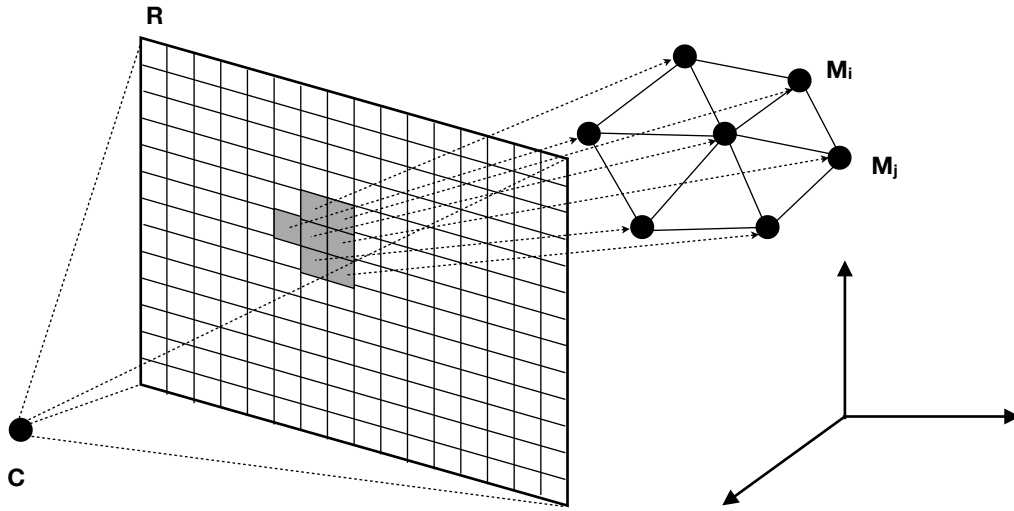


Fig. 4.1. Building a 3D mesh from densely sampled 3D points. **C**: Camera center. **M_i** and **M_j** triangulated points belonging to the same triangle in the mesh because they are neighbor pixels in the retinal plane **R**.

However, when adding more views, the estimated mesh usually contains artifacts corresponding to spurious triangles in the mesh that occur mostly at object boundaries when joining foreground and background surfaces. However, most of this spurious triangles can be removed taking into account that the distance between a foreground and a background point is typically larger than for most correct neighbor points, and also taking into account that this triangles occur at object boundaries and are almost parallel to the principal axis vector of the camera.

For that reason, we have found experimentally that most of the spurious artifacts can be removed by imposing restrictions to the largest euclidean distance between a pair of triangulated points **M_i**, **M_j** that correspond to neighbor pixels (i.e. image points) **m_i**, **m_j**, on a given view *k*, and hence can be related by the same triangle in the 3D mesh as shown in figure 4.1. Therefore,

$$\|\mathbf{M}_i - \mathbf{M}_j\| < D \quad (4.6)$$

where *D* is the largest allowed euclidean distance between points on the same triangle in the mesh. If a given pair of points surpasses this distance, then the triangle in the mesh is discarded.

Conversely, given two world points **M_i**, **M_j**, if the angle of the vector joining both points **v_M** = **M_i** - **M_j** respect to the principal axis vector **v_k** of a given view *k*, where such points project as neighbour image points (i.e. can be related into the same triangle in the mesh), is smaller than a given angle threshold,

$$\theta_M = \arccos \left(\frac{(\mathbf{M}_i - \mathbf{M}_j) \cdot \mathbf{v}_k}{\|\mathbf{M}_i - \mathbf{M}_j\| \cdot \|\mathbf{v}_k\|} \right) \geq \alpha \quad (4.7)$$

where **v_k** is the principal axis vector of *k*-th view, and **α** is the threshold angle respect to the principal axis vector. And any triangle formed by the pair of points **M_i**, **M_j** that produces an angle **θ_M** respect to **v_k** smaller than **α** will be discarded as well.

Experimentally we found that setting *D* to about 3% of the size of the whole 3D reconstruction and **α** about 35° worked fine for most situations. For ease of understanding both restrictions are depicted in figure 4.2.

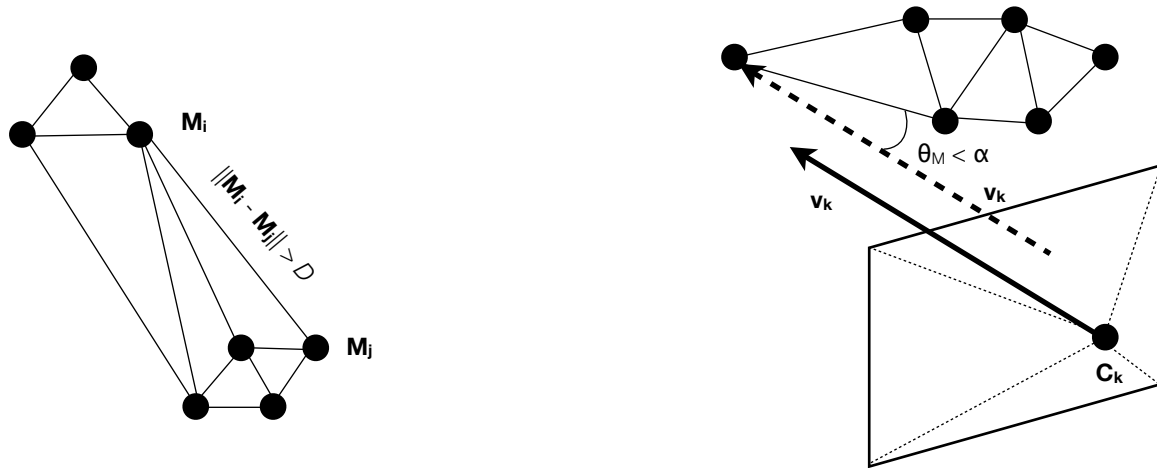


Fig. 4.2. Constraints to build a 3D mesh of points. Left: Distance constraint is not fulfilled and triangle must be discarded. Right: Angle constraint is not fulfilled and triangle must be discarded because angle θ_M respect to principal axis vector v_k is too small.

4.5.- Texture computation

Finally, in order to obtain 3D reconstructions with realistic appearance, it is necessary to provide a certain texture to the computed mesh of 3D points.

In our case, because the 3D reconstruction has been obtained directly from a set of images of a given fixed scene depicted from different points of view, then we can use such images to compute the texture on the reconstruction.

However, because we know that some 3D world points appear in more than one image (i.e. otherwise the 3D reconstruction wouldn't be computed), then we can use such redundancy to improve the texture in the 3D reconstruction. Among other things we are able to reduce artifacts such as glares and shadows; and if the reconstruction has been obtained with sub-pixel accuracy (i.e. using sub-pixel accuracy on disparity estimation as in the multiresolution technique shown in chapter 9), then we can compute super-resolution textures, which means that resolution can be improved respect to the original images, which allows to obtain further detail and can be useful in applications such as surveillance (i.e. reading a car's plate located too far away to be read on the original images).

Next we will briefly show how textures can be computed.

4.5.1.- Reducing glares and shadows

Because some areas of each image appear in more than one view, then in order to compute the corresponding texture for such areas we can just simply perform simple operations such as average, median, minimum values or maximum values. This way glares or shadows can be reduced.

Hence, given a triangle in the 3D mesh where its vertices correspond to 3 consecutive pixels in two views, then the resulting triangle in the texture can be formed by vertices having colors equal to typical operations such as the ones mentioned above, like average color, or median.

Therefore, in our case, because the triangles are formed by consecutive pixels since we have used dense disparity estimation, we can then pack the whole texture as we show in figure 4.3 where all images are placed side by side into a single image.

Notice that in order to avoid some artifacts when representing the final texture in a display, it might be interesting to add some padding between packed triangles by repeating the color of the border pixels in the texture image. This is specially

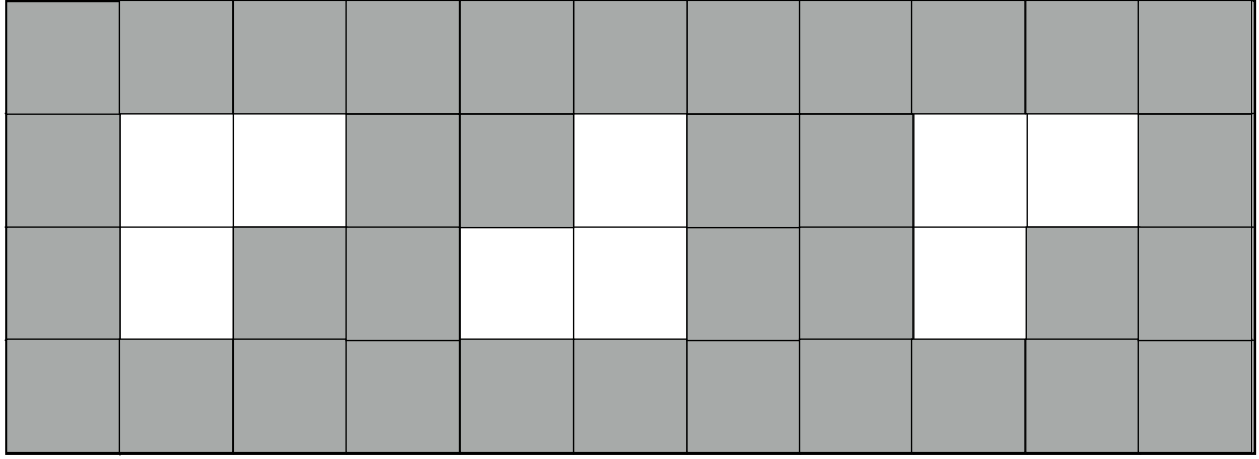


Fig. 4.3. Packing all triangles (shown in white) into a single image with proper padding (in grey)

important when neighbor triangles in the texture correspond to completely different areas in the images of the reconstructed scene.

4.5.2.- Computing super-resolution textures

Analogously to the situation shown in section 4.5.1, we might want to pack the textures of all triangles in the mesh into a single image. However, oppositely to the previous case, if disparity was estimated with sub-pixel accuracy, it is then possible to retrieve textures with super-resolution by means of interpolations, and hence improve image quality respect to the original images that were used for 3D reconstruction.

In this case we will assume that triangles in the mesh do not necessarily need to be formed by 3 pixels (i.e. one for each triangle vertex) on each view where the triangle appears. Although we will assume that triangles are small enough to neglect any possible projective distortion so that super-resolution can still be a viable assumption.

Hence, as shown in figure 4.4, we might have 3D triangles that correspond to image areas on each view that need to be packed into a single triangle in a similar way to the situation shown in figure 4.3, although in this case packed triangles might have more than 3 pixels to achieve super-resolution. Therefore, for each triangle on each view, we will need to find a change of basis to represent such triangle into the packed image of triangles.

Hence, given vectors $\mathbf{v}_1$ and $\mathbf{v}_2$ forming any arbitrary triangle on a given view, we want to find the change of basis so that such vectors become the normalized vectors $\mathbf{u}_1$, $\mathbf{u}_2$, or $\mathbf{u}'_1$, $\mathbf{u}'_2$ as depicted in figure 4.4 depending whether the packed triangle looks upwards or downwards.

Therefore, for the case where the packed triangle looks upwards, we want to transform $\mathbf{v}_1$ and $\mathbf{v}_2$ so that they become $\mathbf{u}_1$ and $\mathbf{u}_2$. Hence, assuming that $\mathbf{u}_1$ and $\mathbf{u}_2$ are the following normalized vectors forming the upwards triangle,

$$\begin{aligned}\mathbf{u}_1 &= \sqrt{\frac{w^2 + h^2}{2}} \cdot (1, 1)^T \\ \mathbf{u}_2 &= h \cdot (0, 1)^T\end{aligned}\tag{4.8}$$

where w and h represent respectively the width and height in pixels of the packed triangles.

Then we want to find a linear application (i.e. the change of basis) $\mathbf{A}$, so that:

$$\mathbf{A} \cdot \begin{bmatrix} \mathbf{v}_1 & \mathbf{v}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{u}_1 & \mathbf{u}_2 \end{bmatrix}\tag{4.9}$$

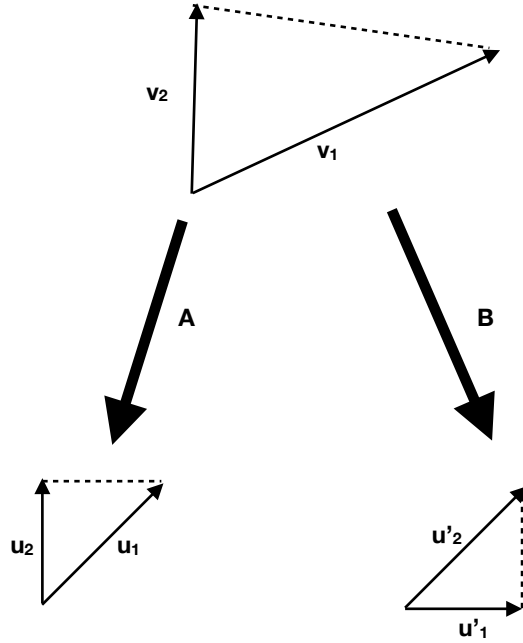


Fig. 4.4. Transforming arbitrary triangles into packed triangles on a texture image

And consequently, $\mathbf{A}$ can be computed as:

$$\mathbf{A} = \begin{bmatrix} \mathbf{u}_1 & \mathbf{u}_2 \end{bmatrix} \begin{bmatrix} \mathbf{v}_1 & \mathbf{v}_2 \end{bmatrix}^{-1} \quad (4.10)$$

On the other hand, given vectors $\mathbf{v}_1$ and $\mathbf{v}_2$ forming any arbitrary triangle on a given view, we can transform it into a packed triangle that looks downwards, by transforming $\mathbf{v}_1$ and $\mathbf{v}_2$ so that they become $\mathbf{u}'_1$ and $\mathbf{u}'_2$. And therefore, assuming now that $\mathbf{u}'_1$ and $\mathbf{u}'_2$ are the following normalized vectors forming the downwards triangle,

$$\begin{aligned} \mathbf{u}'_1 &= w(1,0)^T \\ \mathbf{u}'_2 &= \sqrt{\frac{w^2 + h^2}{2}}(1,1)^T \end{aligned} \quad (4.11)$$

where again w and h represent respectively the width and height in pixels of the packed triangles.

Hence, now we want to find the linear application (i.e. the change of basis) $\mathbf{B}$, so that:

$$\mathbf{B} \cdot \begin{bmatrix} \mathbf{v}_1 & \mathbf{v}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{u}'_1 & \mathbf{u}'_2 \end{bmatrix} \quad (4.12)$$

And consequently, $\mathbf{B}$ can be computed as:

$$\mathbf{B} = \begin{bmatrix} \mathbf{u}'_1 & \mathbf{u}'_2 \end{bmatrix} \begin{bmatrix} \mathbf{v}_1 & \mathbf{v}_2 \end{bmatrix}^{-1} \quad (4.13)$$

Once the we know how to transform arbitrary triangles in a given view into packed triangles in the resulting texture image, we can then proceed to compute such texture image. However, because the transformation might not produce integer pixel locations, it is therefore more useful to use backwards transformation from the coordinates of the packed triangles towards the coordinates in the arbitrary triangles in a given view. Hence, given change of basis matrices $\mathbf{A}$ and $\mathbf{B}$, we can compute such backwards transformation as:

$$\begin{aligned} \begin{bmatrix} x_v \\ y_v \end{bmatrix} &= \mathbf{A}^{-1} \cdot \begin{bmatrix} x_u \\ y_u \end{bmatrix} \\ \begin{bmatrix} x_v \\ y_v \end{bmatrix} &= \mathbf{B}^{-1} \cdot \begin{bmatrix} x_u \\ y_u \end{bmatrix} \end{aligned} \quad (4.14)$$

where x_u and y_u are the coordinates of the packed triangle in the texture image assuming that the triangle is at the origin of coordinates, and x_v , y_v are the coordinates on a given input view. This is shown with more detail on figure 4.5.

Backwards transformation is performed for all the views where the same triangle in the 3D mesh appears, and all the transformed triangles from all the views are combined into a single packed triangle by means of simple operations such as mean, median, minimum or maximum, as we saw in section 4.5.1.

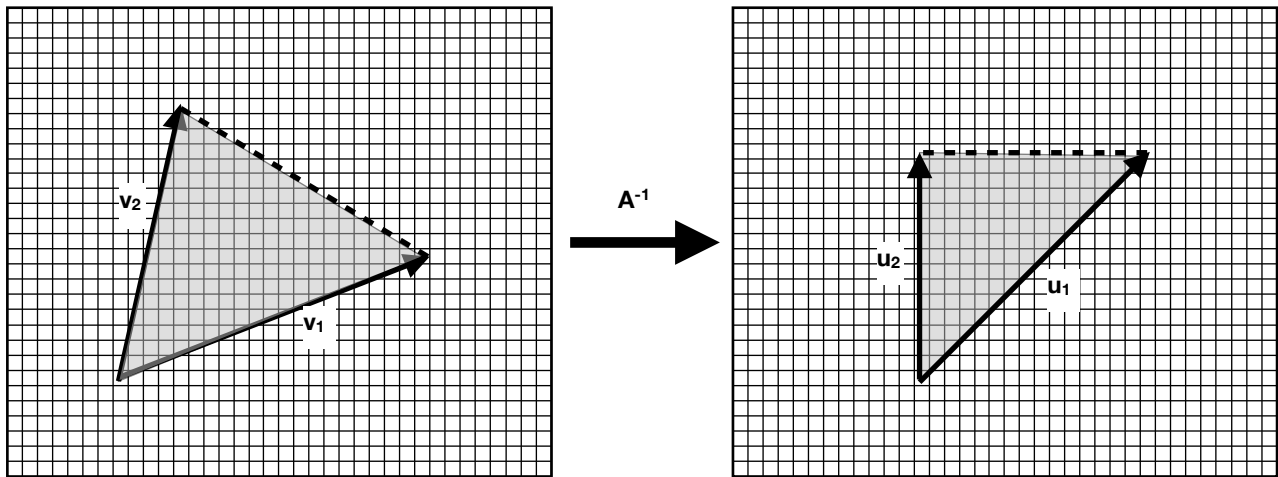


Fig. 4.5. Backwards transformation from arbitrary triangles in a view (left) to a packed triangle (right)

Finally, as in section 4.5.1, packed triangles are put side by side on a single image, and some padding is used by duplicating border pixels on packed triangles (as in figure 4.3) to avoid artifacts on the displayed reconstruction.

4.6.- Closure

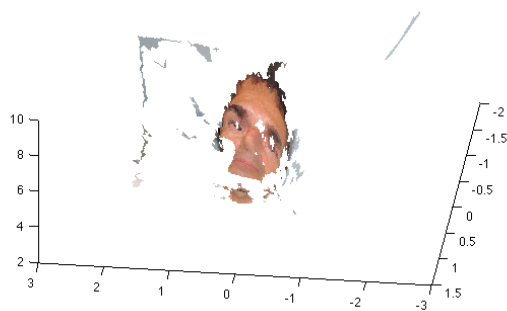
On this chapter we have mainly seen how to add more views into an initial 3D reconstruction obtained from two views, as we had learned from chapters 2 and 3; and then we learned how the triangulated points could be related in a 3D mesh and then how texture could be computed in order to obtain a realistic appearance.



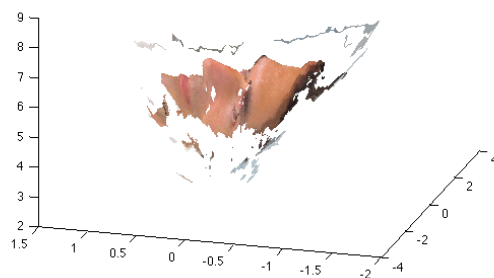
(a)



(b)



(c)



(d)

Fig. 4.6. Example of 3D reconstruction using two views by means of disparity estimated with a multiresolution technique and approximately known calibration matrices.

Up to this point we have assumed that either the obtained reconstruction was in an arbitrary projective stratum or that the calibration matrices $\mathbf{K}_i$ of the cameras were known, and then the 3D reconstruction could be obtained in a metric stratum by means of the essential matrix.

In order to appreciate the results obtained using the techniques described in chapters 1 to 4, we will provide next some examples of 3D reconstructions obtained after taking some pictures on different scenes. In order to obtain such reconstructions we estimated the disparity between pairs of images using the multiresolution technique shown in chapter 9, and we assumed that the calibration matrices $\mathbf{K}_i$ were known using the approximation given in expression (4.1).

In figure 4.7 we can observe that due to the approximation that was done on the calibration matrices, some affine distortion is present on the reconstruction. In order to remove this distortion, we will show in chapter 5 some self-calibration techniques that can be used so that the reconstruction is upgraded to the metric stratum when more views are added. Also, as will be shown in the examples provided in chapter 5, when more views are added, we will also be able to fill holes and occlusions in the obtained 3D reconstruction.

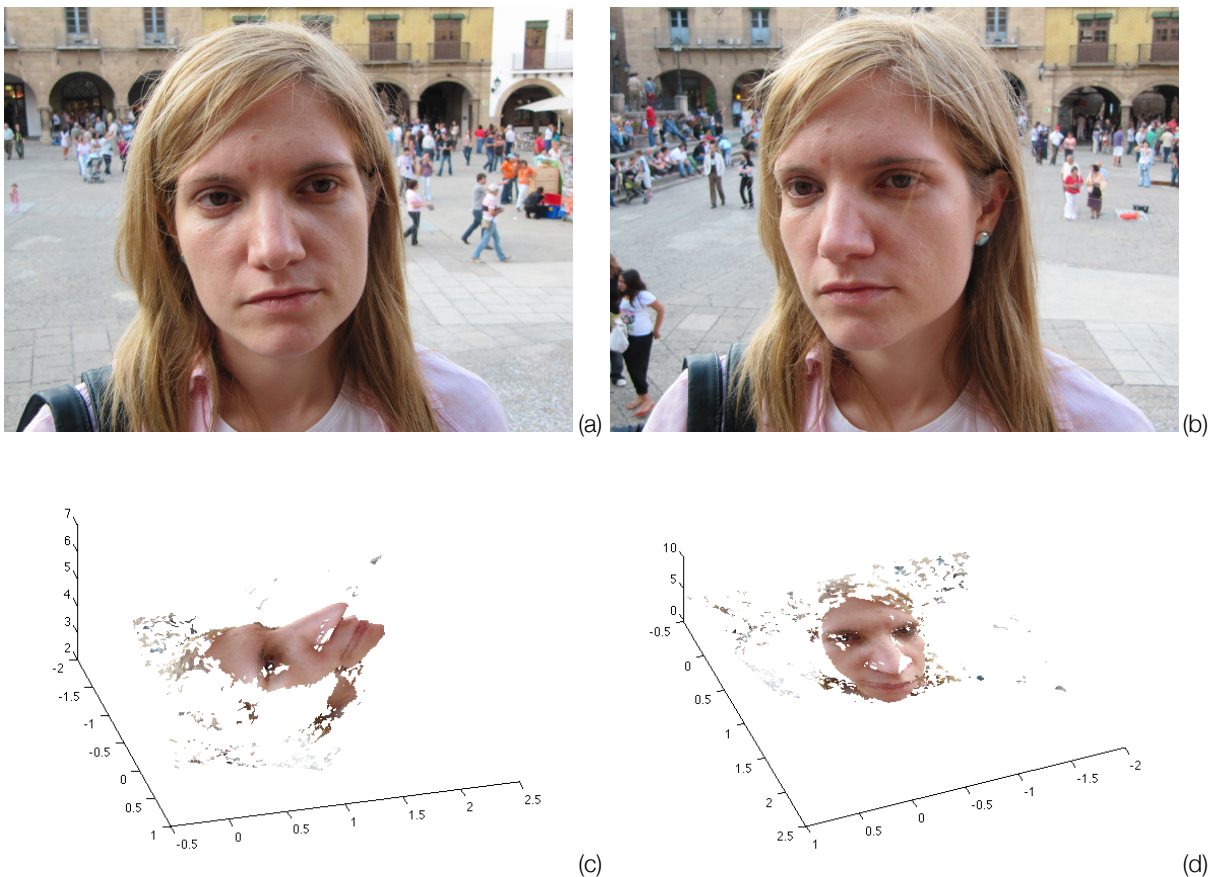


Fig. 4.7. Example of 3D reconstruction using two views by means of disparity estimated with a multiresolution technique and approximately known calibration matrices. Affine distortion can be clearly observed especially in the area close to the nose.

5.- Self-calibration

5.1.- Introduction

In chapter 1 to 4 we revisited all the basics in geometry, as well as the relations appearing on multiple views by means of epipolar geometry when using projective cameras. We described how from those relations a 3D reconstruction of the scene could be retrieved from an initial pair of views, and then how we could add further views in order to increase the quality of the obtained reconstruction. Finally, we also described how to convert the triangulated 3D points in a 3D mesh and how to add a realistic texture so that the 3D reconstruction looked natural.

However, during this process, in chapter 3 we assumed that the obtained reconstruction was either laying on an arbitrary projective stratum, or that the calibration matrices (i.e. the intrinsic parameters) of the cameras were known beforehand.

Although some restrictions are generally applied to the calibration matrices and their structure is more or less known, some parameters such as the focal length of the camera might not be precisely known in advance for any unknown camera. For that reason, although the approximation given in expression (4.1) is more or less valid for most situations, still some affine distortion is usually present in the reconstructions.

In this chapter we will provide different techniques that can be used to upgrade the reconstruction to a metric stratum, which in other words can be seen as estimating the calibration matrices $\mathbf{K}$ of all the cameras under certain assumptions.

5.2.- The absolute dual quadric and the dual image of the absolute conic

In chapter 1 we already reviewed concepts like the absolute dual quadric, and the transformation needed to upgrade from any projective stratum to a metric one. Indeed we saw that the absolute dual quadric $\mathbf{Q}^*_{\infty}$ had expression (1.179) in the **metric** stratum,

$$\mathbf{Q}^*_{\infty} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad (5.1)$$

And we also saw that the transformation needed to upgrade 3D world points in any projective stratum to a metric one was as shown in (1.186),

$$\mathbf{T}_{\text{PM}} = \begin{bmatrix} \mathbf{A}^{-1} & \mathbf{0}_3 \\ \boldsymbol{\pi}_{\infty}^T & 1 \end{bmatrix} \quad (5.2)$$

And taking into account its inverse transformation (i.e. from metric to projective strata), as in (1.192),

$$\mathbf{T}_{MP} = \mathbf{T}_{PM}^{-1} = \begin{bmatrix} \mathbf{A} & \mathbf{0}_3 \\ -\pi_\infty^T \cdot \mathbf{A} & 1 \end{bmatrix} \quad (5.3)$$

we observe that the absolute dual quadric could be expressed in any arbitrary projective stratum as in (1.193),

$$\mathbf{Q}_\infty^* \sim \mathbf{T}_{PM}^{-1} \cdot \mathbf{Q}_\infty^* \cdot \mathbf{T}_{PM}^{-T} = \begin{bmatrix} \mathbf{A} \cdot \mathbf{A}^T & -\mathbf{A} \cdot \mathbf{A}^T \cdot \pi_\infty \\ -\pi_\infty^T \cdot \mathbf{A} \cdot \mathbf{A}^T & \pi_\infty^T \cdot \mathbf{A} \cdot \mathbf{A}^T \cdot \pi_\infty \end{bmatrix} \quad (5.4)$$

On the other hand, on chapter 2, we observed that the dual image of the absolute conic (DIAC), could be obtained by projecting the absolute dual quadric in any arbitrary projective stratum into the image plane, just as shown in expression (2.158),

$$\omega_\infty^* \sim \mathbf{P} \cdot \mathbf{Q}_\infty^* \cdot \mathbf{P}^T \quad (5.5)$$

Where we can observe in the **metric** stratum that by using the camera projection matrix $\mathbf{P}$ in the form shown in (2.24), then (5.5) becomes,

$$\begin{aligned} \omega_\infty^* &= \mathbf{K} \cdot \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{I}_{3 \times 3} & \mathbf{0}_3 \\ \mathbf{0}_3^T & 0 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{R}^T \\ \mathbf{t}^T \end{bmatrix} \cdot \mathbf{K}^T = \\ &= \mathbf{K} \cdot \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{R}^T + \mathbf{0}_3 \cdot \mathbf{t}^T \\ \mathbf{0}_3^T \cdot \mathbf{R}^T + 0 \cdot \mathbf{t}^T \end{bmatrix} \cdot \mathbf{K}^T = \\ &= \mathbf{K} \cdot \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{R}^T \\ \mathbf{0}_3^T \end{bmatrix} \cdot \mathbf{K}^T = \mathbf{K} \cdot [\mathbf{R} \cdot \mathbf{R}^T + \mathbf{t} \cdot \mathbf{0}_3^T] \cdot \mathbf{K}^T = \\ &= \mathbf{K} \cdot \mathbf{I}_{3 \times 3} \cdot \mathbf{K}^T = \mathbf{K} \cdot \mathbf{K}^T \end{aligned} \quad (5.6)$$

as was already mentioned in chapter 1. We can observe in (5.6) that the DIAC is a 3x3 real symmetric matrix (i.e. hence it is positive definite), and consequently the calibration matrix $\mathbf{K}$ can be obtained by means of Cholesky decomposition of the DIAC.

Finally, it must be noticed that by taking into the first canonical camera expression,

$$\mathbf{P} = [\mathbf{I}_{3 \times 3} | \mathbf{0}_3] \quad (5.7)$$

where no rotation or translation is present, then the same camera in the **metric** stratum can be represented as:

$$\mathbf{P}_M = \mathbf{K} [\mathbf{I}_{3 \times 3} | \mathbf{0}_3] \quad (5.8)$$

and finally taking into account that cameras in any projective strata $\mathbf{P}$ are related by transformation $\mathbf{T}_{PM}$ to the metric stratum, just as follows:

$$\mathbf{P}_M = \mathbf{P} \cdot \mathbf{T}_{PM}^{-1} \quad (5.9)$$

as it was shown in (2.154), where $\mathbf{m} = \mathbf{P}_M \cdot \mathbf{M}_M = \mathbf{P} \cdot \mathbf{T}_{PM}^{-1} \cdot \mathbf{T}_{PM} \cdot \mathbf{M}$.

Therefore, by combining (5.3), (5.7), (5.8) and (5.9) we obtain:

$$\begin{aligned} \mathbf{P}_M &= \mathbf{P} \cdot \mathbf{T}_{PM}^{-1} \Rightarrow \\ \mathbf{K} \cdot [\mathbf{I}_{3 \times 3} | \mathbf{0}_3] &= [\mathbf{I}_{3 \times 3} | \mathbf{0}_3] \cdot \begin{bmatrix} \mathbf{A} & \mathbf{0}_3 \\ -\pi_\infty^T \cdot \mathbf{A} & 1 \end{bmatrix} = \\ \begin{bmatrix} \mathbf{I}_{3 \times 3} \cdot \mathbf{A} - \mathbf{0}_3 \cdot \pi_\infty^T \cdot \mathbf{A} & \mathbf{I}_{3 \times 3} \cdot \mathbf{0}_3 + 1 \cdot \mathbf{0}_3 \end{bmatrix} &= \begin{bmatrix} \mathbf{A} & \mathbf{0}_3 \end{bmatrix} \Rightarrow \\ \mathbf{A} &= \mathbf{K} \end{aligned} \quad (5.10)$$

and hence because $\mathbf{A} = \mathbf{K}$, then expressions (5.2), (5.3) and (5.4) can be rewritten in terms of the calibration matrix $\mathbf{K}$ and the dual image of the absolute conic (DIAC).

Therefore, the transformation from projective to metric must be:

$$\mathbf{T}_{PM} = \begin{bmatrix} \mathbf{K}^{-1} & \mathbf{0}_3 \\ \pi_\infty^T & 1 \end{bmatrix} \quad (5.11)$$

Its inverse transformation,

$$\mathbf{T}_{MP} = \mathbf{T}_{PM}^{-1} = \begin{bmatrix} \mathbf{K} & \mathbf{0}_3 \\ -\pi_\infty^T \cdot \mathbf{K} & 1 \end{bmatrix} \quad (5.12)$$

and the absolute dual quadric:

$$\mathbf{Q}_\infty^* \sim \mathbf{T}_{PM}^{-1} \cdot \mathbf{Q}_\infty^* \cdot \mathbf{T}_{PM}^{-T} = \begin{bmatrix} \mathbf{K} \cdot \mathbf{K}^T & -\mathbf{K} \cdot \mathbf{K}^T \cdot \pi_\infty \\ -\pi_\infty^T \cdot \mathbf{K} \cdot \mathbf{K}^T & \pi_\infty^T \cdot \mathbf{K} \cdot \mathbf{K}^T \cdot \pi_\infty \end{bmatrix} = \begin{bmatrix} \omega_\infty^* & -\omega_\infty^* \cdot \pi_\infty \\ -\pi_\infty^T \cdot \omega_\infty^* & \pi_\infty^T \cdot \omega_\infty^* \cdot \pi_\infty \end{bmatrix} \quad (5.13)$$

Taking all these concepts into account we will provide the methods to find the appropriate transformation $\mathbf{T}_{PM}$ to upgrade a given reconstruction on an arbitrary projective stratum to a metric one.

5.3.- Self-calibration methods

In the following sections a couple of methods for self-calibration will be provided. The first one is a direct method based on the estimation of the absolute dual quadric, which from (5.13) we can observe that contains both the DIAC (and hence the calibration matrix $\mathbf{K}$) and the plane at infinity, which are both needed for the transformation from projective to metric, as shown in (5.11) and (5.12).

The second method is based on the Kruppa equations, which only need pairwise information from views by means of their respective fundamental matrices. This method, although we will see that is simpler to compute than the former one, it also tends to obtain poorer results because some additional degeneracies might appear respect to the method based on the absolute dual quadric.

5.3.1.- Calibration using the absolute dual quadric

Using expression (5.5) that relates the DIAC with the absolute dual quadric we will be able to estimate the absolute dual quadric by imposing certain restrictions on the DIAC, which is related to the calibration matrix $\mathbf{K}$ by means of expression (5.6).

Indeed, because of expression (5.6) that relates the DIAC with the calibration matrix $\mathbf{K}$, and taking into account the expression of the calibration matrix shown in (2.17), then the DIAC can be expressed as:

$$\begin{aligned}\omega_{\infty}^* &= \mathbf{K} \cdot \mathbf{K}^T = \begin{bmatrix} \alpha_x & s & x_0 \\ 0 & \alpha_y & y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \alpha_x & 0 & 0 \\ s & \alpha_y & 0 \\ x_0 & y_0 & 1 \end{bmatrix} = \\ &= \begin{bmatrix} \alpha_x^2 + s^2 + x_0^2 & s \cdot \alpha_y + x_0 \cdot y_0 & x_0 \\ s \cdot \alpha_y + x_0 \cdot y_0 & \alpha_y^2 + y_0^2 & y_0 \\ x_0 & y_0 & 1 \end{bmatrix} \quad (5.14)\end{aligned}$$

Common assumptions can be made on the calibration matrix $\mathbf{K}$, such as skew parameter $s = 0$, or a principal point located at the origin of coordinates of the image ($x_0 = 0$, $y_0 = 0$), assuming that the origin of coordinates is located at the center of the image, otherwise proper image coordinates transformation must be done, as when we normalized image coordinates in (2.116) or as in (4.1), or even it could be assumed that aspect ratio is 1 and hence both horizontal and vertical focal lengths are equal (i.e. $\alpha = \alpha_x = \alpha_y$).

When assuming such simplifications, which are true for most cameras, then only the focal length α is unknown, and then, expression (5.14) becomes:

$$\omega_{\infty}^* = \mathbf{K} \cdot \mathbf{K}^T = \begin{bmatrix} \alpha & 0 & 0 \\ 0 & \alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \alpha & 0 & 0 \\ 0 & \alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \alpha^2 & 0 & 0 \\ 0 & \alpha^2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (5.15)$$

where, taking into account that the DIAC is symmetric, we observe that we have 3 different elements equal to zero, which we will see that by means expression (5.5) those 3 elements will become three equations that will be used for solving the absolute dual quadric $\mathbf{Q}_{\infty}^*$. Besides, because of the constraint in the aspect ratio, an additional equation will be available.

We must also notice that expression (5.5) can be expressed in terms of each i -th view as:

$$\omega_{\infty i}^* \sim \mathbf{P}_i \cdot \mathbf{Q}_{\infty}^* \cdot \mathbf{P}_i^T \quad (5.16)$$

where $\mathbf{Q}_{\infty}^*$ is the absolute dual quadric and $\mathbf{P}_i$ is the i -th camera projection matrix both in an **arbitrary projective stratum**. Notice that the i -th camera projection matrix must have the form,

$$\mathbf{P}_i = \begin{bmatrix} \mathbf{R}_i & \mathbf{t}_i \end{bmatrix} \quad (5.17)$$

where $\mathbf{R}$ is a 3x3 rotation matrix (i.e. orthogonal) and $\mathbf{t}$ is the translation component.

Notice that **if camera matrices don't have the form in (5.17), then they should be decomposed to remove their individual and arbitrary calibration matrices $\mathbf{K}_i$ before attempting to use (5.16).**

Indeed, by means of (5.1), (5.9) and (5.13) then expression (5.16) can also be written as:

$$\omega_{\infty i}^* \sim \mathbf{P}_i \cdot \mathbf{Q}_{\infty}^* \cdot \mathbf{P}_i^T \Rightarrow$$

$$\mathbf{K}_i \cdot \mathbf{K}_i^T = \begin{bmatrix} \mathbf{R}_i & \mathbf{t}_i \end{bmatrix} \cdot \begin{bmatrix} \mathbf{K}_1^{-1} & \mathbf{0}_3 \\ \pi_{\infty}^T & 1 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{I}_{3 \times 3} & \mathbf{0}_3 \\ \mathbf{0}_3^T & 0 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{K}_1^T & \pi_{\infty} \\ \mathbf{0}_3^T & 1 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{R}_i^T \\ \mathbf{t}_i^T \end{bmatrix} \quad (5.18)$$

where $\mathbf{K}_i$ is the camera calibration matrix of the i-th view, and $\mathbf{K}_1$ is the camera calibration matrix of the first view.

By taking into account the assumptions of zero skewness, zero principal point and aspect ratio equal to one, then because of (5.15) and (5.16) we obtain **for each i-th view 4 equations** to solve the absolute dual quadric as seen in (5.15) (i.e. three elements in the DIAC equal to zero, and two equal elements in the DIAC). This will be shown next.

Indeed, by expressing the i-th camera matrix in terms of its 3 rows in an arbitrary projective stratum as follows:

$$\mathbf{P}_i = \begin{bmatrix} \mathbf{p}_i^{1T} \\ \mathbf{p}_i^{2T} \\ \mathbf{p}_i^{3T} \end{bmatrix} \quad (5.19)$$

where as we indicated the i-th camera also has the form in (5.17). Then expression (5.16) can also be written as:

$$\omega_{\infty i}^* \sim \mathbf{P}_i \cdot \mathbf{Q}_{\infty}^* \cdot \mathbf{P}_i^T = \begin{bmatrix} \mathbf{p}_i^{1T} \\ \mathbf{p}_i^{2T} \\ \mathbf{p}_i^{3T} \end{bmatrix} \cdot \mathbf{Q}_{\infty}^* \cdot \begin{bmatrix} \mathbf{p}_i^1 & \mathbf{p}_i^2 & \mathbf{p}_i^3 \end{bmatrix} =$$

$$\begin{bmatrix} \mathbf{p}_i^{1T} \cdot \mathbf{Q}_{\infty}^* \cdot \mathbf{p}_i^1 & \mathbf{p}_i^{1T} \cdot \mathbf{Q}_{\infty}^* \cdot \mathbf{p}_i^2 & \mathbf{p}_i^{1T} \cdot \mathbf{Q}_{\infty}^* \cdot \mathbf{p}_i^3 \\ \mathbf{p}_i^{2T} \cdot \mathbf{Q}_{\infty}^* \cdot \mathbf{p}_i^1 & \mathbf{p}_i^{2T} \cdot \mathbf{Q}_{\infty}^* \cdot \mathbf{p}_i^2 & \mathbf{p}_i^{2T} \cdot \mathbf{Q}_{\infty}^* \cdot \mathbf{p}_i^3 \\ \mathbf{p}_i^{3T} \cdot \mathbf{Q}_{\infty}^* \cdot \mathbf{p}_i^1 & \mathbf{p}_i^{3T} \cdot \mathbf{Q}_{\infty}^* \cdot \mathbf{p}_i^2 & \mathbf{p}_i^{3T} \cdot \mathbf{Q}_{\infty}^* \cdot \mathbf{p}_i^3 \end{bmatrix} \quad (5.20)$$

Therefore, for element (1,2) in the DIAC equal to zero we obtain the first equation:

$$\mathbf{p}_i^{1T} \cdot \mathbf{Q}_{\infty}^* \cdot \mathbf{p}_i^2 = 0 \quad (5.21)$$

where $\mathbf{p}_i^1$ is the i-th row of a given camera matrix.

Also, element (1,3) on the DIAC is equal to zero, producing the second equation:

$$\mathbf{p}_i^{1T} \cdot \mathbf{Q}_{\infty}^* \cdot \mathbf{p}_i^3 = 0 \quad (5.22)$$

By taking into account also that element (2,3) on the DIAC is zero we obtain the third equation:

$$\mathbf{p}_i^{2T} \cdot \mathbf{Q}_{\infty}^* \cdot \mathbf{p}_i^3 = 0 \quad (5.23)$$

And finally, we take into account that elements (1,1) and (2,2) are equal on the DIAC because we assume that the aspect ratio is equal to one. Hence we obtain the last equation:

$$\mathbf{p}_i^{1T} \cdot \mathbf{Q}_\infty^* \cdot \mathbf{p}_i^1 = \mathbf{p}_i^{2T} \cdot \mathbf{Q}_\infty^* \cdot \mathbf{p}_i^2 \quad (5.24)$$

It must be noticed that because the DIAC has the form in (5.15) after the assumptions we have made, then by means of (5.13) the dual absolute quadric in any arbitrary projective stratum has the following form:

$$\mathbf{Q}_\infty^* = \begin{bmatrix} \omega_\infty^* & -\omega_\infty^* \cdot \pi_\infty \\ -\pi_\infty^T \cdot \omega_\infty^* & \pi_\infty^T \cdot \omega_\infty^* \cdot \pi_\infty \end{bmatrix} = \begin{bmatrix} \alpha^2 & 0 & 0 & -\alpha^2 \cdot \beta \\ 0 & \alpha^2 & 0 & -\alpha^2 \cdot \gamma \\ 0 & 0 & 1 & -\delta \\ -\alpha^2 \cdot \beta & -\alpha^2 \cdot \gamma & -\delta & \alpha^2 \cdot \beta^2 + \alpha^2 \cdot \gamma^2 + \delta^2 \end{bmatrix} = \begin{bmatrix} a & 0 & 0 & b \\ 0 & a & 0 & c \\ 0 & 0 & f & d \\ b & c & d & e \end{bmatrix} \quad (5.25)$$

where the dual absolute quadric is defined up to scale, and for that reason f could be different from zero, but for practical reasons we will assume that (5.25) is normalized and $f = 1$; and where the first three elements of the plane at the infinity are:

$$\pi_\infty = (\beta, \gamma, \delta)^T \quad (5.26)$$

And taking into account that the elements of the i -th camera $\mathbf{P}_i$ can be expressed as:

$$\mathbf{P}_i = \begin{bmatrix} p_{11}^i & p_{12}^i & p_{13}^i & p_{14}^i \\ p_{21}^i & p_{22}^i & p_{23}^i & p_{24}^i \\ p_{31}^i & p_{32}^i & p_{33}^i & p_{34}^i \end{bmatrix} \quad (5.27)$$

Then equation (5.21) can be linearized as follows:

$$\mathbf{p}_i^{1T} \cdot \mathbf{Q}_\infty^* \cdot \mathbf{p}_i^2 = 0 \Rightarrow \begin{bmatrix} p_{11}^i & p_{12}^i & p_{13}^i & p_{14}^i \end{bmatrix} \cdot \begin{bmatrix} a & 0 & 0 & b \\ 0 & a & 0 & c \\ 0 & 0 & f & d \\ b & c & d & e \end{bmatrix} \cdot \begin{bmatrix} p_{21}^i \\ p_{22}^i \\ p_{23}^i \\ p_{24}^i \end{bmatrix} = 0 \Rightarrow \begin{bmatrix} p_{11}^i & p_{12}^i & p_{13}^i & p_{14}^i \end{bmatrix} \cdot \begin{bmatrix} a \cdot p_{21}^i + b \cdot p_{24}^i \\ a \cdot p_{22}^i + c \cdot p_{24}^i \\ f \cdot p_{23}^i + d \cdot p_{24}^i \\ b \cdot p_{21}^i + c \cdot p_{22}^i + d \cdot p_{23}^i + e \cdot p_{24}^i \end{bmatrix} = 0 \quad (5.28)$$

And continuing expansion, we obtain:

$$a \cdot p_{21}^i \cdot p_{11}^i + b \cdot p_{24}^i \cdot p_{11}^i + a \cdot p_{22}^i \cdot p_{12}^i + c \cdot p_{24}^i \cdot p_{12}^i + f \cdot p_{23}^i \cdot p_{13}^i + d \cdot p_{24}^i \cdot p_{13}^i + b \cdot p_{21}^i \cdot p_{14}^i + c \cdot p_{22}^i \cdot p_{14}^i + d \cdot p_{23}^i \cdot p_{14}^i + e \cdot p_{24}^i \cdot p_{14}^i = 0 \Rightarrow \quad (5.29)$$

$$a(p_{21}^i \cdot p_{11}^i + p_{22}^i \cdot p_{12}^i) + b(p_{24}^i \cdot p_{11}^i + p_{21}^i \cdot p_{14}^i) + c(p_{24}^i \cdot p_{12}^i + p_{22}^i \cdot p_{14}^i) + d(p_{24}^i \cdot p_{13}^i + p_{23}^i \cdot p_{14}^i) + e(p_{24}^i \cdot p_{14}^i) + f(p_{23}^i \cdot p_{13}^i) = 0$$

The same process can be repeated on expression (5.22) to obtain:

$$\mathbf{p}_i^{1T} \cdot \mathbf{Q}_\infty^* \cdot \mathbf{p}_i^3 = 0 \Rightarrow$$

$$\begin{bmatrix} p_{11}^i & p_{12}^i & p_{13}^i & p_{14}^i \end{bmatrix} \cdot \begin{bmatrix} a & 0 & 0 & b \\ 0 & a & 0 & c \\ 0 & 0 & f & d \\ b & c & d & e \end{bmatrix} \cdot \begin{bmatrix} p_{31}^i \\ p_{32}^i \\ p_{33}^i \\ p_{34}^i \end{bmatrix} = 0 \Rightarrow$$

$$\begin{bmatrix} p_{11}^i & p_{12}^i & p_{13}^i & p_{14}^i \end{bmatrix} \cdot \begin{bmatrix} a \cdot p_{31}^i + b \cdot p_{34}^i \\ a \cdot p_{32}^i + c \cdot p_{34}^i \\ f \cdot p_{33}^i + d \cdot p_{34}^i \\ b \cdot p_{31}^i + c \cdot p_{32}^i + d \cdot p_{33}^i + e \cdot p_{34}^i \end{bmatrix} = 0 \Rightarrow \quad (5.30)$$

$$a \cdot p_{31}^i \cdot p_{11}^i + b \cdot p_{34}^i \cdot p_{11}^i + a \cdot p_{32}^i \cdot p_{12}^i + c \cdot p_{34}^i \cdot p_{12}^i + f \cdot p_{33}^i \cdot p_{13}^i + d \cdot p_{34}^i \cdot p_{13}^i + b \cdot p_{31}^i \cdot p_{14}^i + c \cdot p_{32}^i \cdot p_{14}^i + d \cdot p_{33}^i \cdot p_{14}^i + e \cdot p_{34}^i \cdot p_{14}^i = 0 \Rightarrow$$

$$a(p_{31}^i \cdot p_{11}^i + p_{32}^i \cdot p_{12}^i) + b(p_{34}^i \cdot p_{11}^i + p_{31}^i \cdot p_{14}^i) + c(p_{34}^i \cdot p_{12}^i + p_{32}^i \cdot p_{14}^i) + d(p_{34}^i \cdot p_{13}^i + p_{33}^i \cdot p_{14}^i) + e(p_{34}^i \cdot p_{14}^i) + f(p_{33}^i \cdot p_{13}^i) = 0$$

And also we can expand expression (5.23) as follows:

$$\mathbf{p}_i^{2T} \cdot \mathbf{Q}_\infty^* \cdot \mathbf{p}_i^3 = 0 \Rightarrow$$

$$\begin{bmatrix} p_{21}^i & p_{22}^i & p_{23}^i & p_{24}^i \end{bmatrix} \cdot \begin{bmatrix} a & 0 & 0 & b \\ 0 & a & 0 & c \\ 0 & 0 & f & d \\ b & c & d & e \end{bmatrix} \cdot \begin{bmatrix} p_{31}^i \\ p_{32}^i \\ p_{33}^i \\ p_{34}^i \end{bmatrix} = 0 \quad (5.31)$$

And expansion can be continued as follows:

$$\begin{aligned}
& \begin{bmatrix} p_{21}^i & p_{22}^i & p_{23}^i & p_{24}^i \end{bmatrix} \cdot \begin{bmatrix} a \cdot p_{31}^i + b \cdot p_{34}^i \\ a \cdot p_{32}^i + c \cdot p_{34}^i \\ f \cdot p_{33}^i + d \cdot p_{34}^i \\ b \cdot p_{31}^i + c \cdot p_{32}^i + d \cdot p_{33}^i + e \cdot p_{34}^i \end{bmatrix} = 0 \Rightarrow \\
& a \cdot p_{31}^i \cdot p_{21}^i + b \cdot p_{34}^i \cdot p_{21}^i + a \cdot p_{32}^i \cdot p_{22}^i + c \cdot p_{34}^i \cdot p_{22}^i + f \cdot p_{33}^i \cdot p_{23}^i + d \cdot p_{34}^i \cdot p_{23}^i + \\
& b \cdot p_{31}^i \cdot p_{24}^i + c \cdot p_{32}^i \cdot p_{24}^i + d \cdot p_{33}^i \cdot p_{24}^i + e \cdot p_{34}^i \cdot p_{24}^i = 0 \Rightarrow \\
& a(p_{31}^i \cdot p_{21}^i + p_{32}^i \cdot p_{22}^i) + b(p_{34}^i \cdot p_{21}^i + p_{31}^i \cdot p_{24}^i) + c(p_{34}^i \cdot p_{22}^i + p_{32}^i \cdot p_{24}^i) + \\
& d(p_{34}^i \cdot p_{23}^i + p_{33}^i \cdot p_{24}^i) + e(p_{34}^i \cdot p_{24}^i) + f(p_{33}^i \cdot p_{23}^i) = 0
\end{aligned} \tag{5.32}$$

And finally equation (5.24) can be expanded as follows:

$$\begin{aligned}
& \mathbf{p}_i^{1T} \cdot \mathbf{Q}_\infty^* \cdot \mathbf{p}_i^1 = \mathbf{p}_i^{2T} \cdot \mathbf{Q}_\infty^* \cdot \mathbf{p}_i^2 \Rightarrow \\
& \begin{bmatrix} p_{11}^i & p_{12}^i & p_{13}^i & p_{14}^i \end{bmatrix} \cdot \begin{bmatrix} a & 0 & 0 & b \\ 0 & a & 0 & c \\ 0 & 0 & f & d \\ b & c & d & e \end{bmatrix} \cdot \begin{bmatrix} p_{11}^i \\ p_{12}^i \\ p_{13}^i \\ p_{14}^i \end{bmatrix} = \\
& \begin{bmatrix} p_{21}^i & p_{22}^i & p_{23}^i & p_{24}^i \end{bmatrix} \cdot \begin{bmatrix} a & 0 & 0 & b \\ 0 & a & 0 & c \\ 0 & 0 & f & d \\ b & c & d & e \end{bmatrix} \cdot \begin{bmatrix} p_{21}^i \\ p_{22}^i \\ p_{23}^i \\ p_{24}^i \end{bmatrix} \Rightarrow \\
& \begin{bmatrix} p_{11}^i & p_{12}^i & p_{13}^i & p_{14}^i \end{bmatrix} \cdot \begin{bmatrix} a \cdot p_{11}^i + b \cdot p_{14}^i \\ a \cdot p_{12}^i + c \cdot p_{14}^i \\ f \cdot p_{13}^i + d \cdot p_{14}^i \\ b \cdot p_{11}^i + c \cdot p_{12}^i + d \cdot p_{13}^i + e \cdot p_{14}^i \end{bmatrix} = \\
& \begin{bmatrix} p_{21}^i & p_{22}^i & p_{23}^i & p_{24}^i \end{bmatrix} \cdot \begin{bmatrix} a \cdot p_{21}^i + b \cdot p_{24}^i \\ a \cdot p_{22}^i + c \cdot p_{24}^i \\ f \cdot p_{23}^i + d \cdot p_{24}^i \\ b \cdot p_{21}^i + c \cdot p_{22}^i + d \cdot p_{23}^i + e \cdot p_{24}^i \end{bmatrix} \Rightarrow
\end{aligned} \tag{5.33}$$

And finally (5.33) expands as:

$$\begin{aligned}
& a \cdot p_{11}^{i2} + b \cdot p_{14}^i \cdot p_{11}^i + a \cdot p_{12}^{i2} + c \cdot p_{14}^i \cdot p_{12}^i + f \cdot p_{13}^{i2} + d \cdot p_{14}^i \cdot p_{13}^i + b \cdot p_{11}^i \cdot p_{14}^i + c \cdot p_{12}^i \cdot p_{14}^i + d \cdot p_{13}^i \cdot p_{14}^i + e \cdot p_{14}^{i2} = \\
& a \cdot p_{21}^{i2} + b \cdot p_{24}^i \cdot p_{21}^i + a \cdot p_{22}^{i2} + c \cdot p_{24}^i \cdot p_{22}^i + f \cdot p_{23}^{i2} + d \cdot p_{24}^i \cdot p_{23}^i + b \cdot p_{21}^i \cdot p_{24}^i + c \cdot p_{22}^i \cdot p_{24}^i + d \cdot p_{23}^i \cdot p_{24}^i + e \cdot p_{24}^{i2} \Rightarrow \\
& a(p_{11}^{i2} + p_{12}^{i2} - p_{21}^{i2} - p_{22}^{i2}) + 2 \cdot b(p_{14}^i \cdot p_{11}^i - p_{24}^i \cdot p_{21}^i) + 2 \cdot c(p_{14}^i \cdot p_{12}^i - p_{24}^i \cdot p_{22}^i) + \\
& 2 \cdot d(p_{14}^i \cdot p_{13}^i - p_{24}^i \cdot p_{23}^i) + e(p_{14}^{i2} - p_{24}^{i2}) + f(p_{13}^{i2} - p_{23}^{i2}) = 0
\end{aligned} \tag{5.34}$$

where p_{jk}^{i2} is the squared value of element (j,k) of i -th camera matrix $\mathbf{P}_i$.

Finally, by combining (5.29), (5.30), (5.32) and (5.34), we can obtain a , b , c , d , e , and f values that form the absolute dual quadric $\mathbf{Q}^*_{\infty}$ as shown in (5.25) by solving the following homogeneous linear system of equations,

$$\begin{bmatrix}
p_{21}^i \cdot p_{11}^i + p_{22}^i \cdot p_{12}^i & p_{24}^i \cdot p_{11}^i + p_{21}^i \cdot p_{14}^i & p_{24}^i \cdot p_{12}^i + p_{22}^i \cdot p_{14}^i & p_{24}^i \cdot p_{13}^i + p_{23}^i \cdot p_{14}^i & p_{24}^i \cdot p_{14}^i & p_{23}^i \cdot p_{13}^i \\
p_{31}^i \cdot p_{11}^i + p_{32}^i \cdot p_{12}^i & p_{34}^i \cdot p_{11}^i + p_{31}^i \cdot p_{14}^i & p_{34}^i \cdot p_{12}^i + p_{32}^i \cdot p_{14}^i & p_{34}^i \cdot p_{13}^i + p_{33}^i \cdot p_{14}^i & p_{34}^i \cdot p_{14}^i & p_{33}^i \cdot p_{13}^i \\
p_{31}^i \cdot p_{21}^i + p_{32}^i \cdot p_{22}^i & p_{34}^i \cdot p_{21}^i + p_{31}^i \cdot p_{24}^i & p_{34}^i \cdot p_{22}^i + p_{32}^i \cdot p_{24}^i & p_{34}^i \cdot p_{23}^i + p_{33}^i \cdot p_{24}^i & p_{34}^i \cdot p_{24}^i & p_{33}^i \cdot p_{23}^i \\
p_{11}^{i2} + p_{12}^{i2} - p_{21}^{i2} - p_{22}^{i2} & 2(p_{14}^i \cdot p_{11}^i - p_{24}^i \cdot p_{21}^i) & 2(p_{14}^i \cdot p_{12}^i - p_{24}^i \cdot p_{22}^i) & 2(p_{14}^i \cdot p_{13}^i - p_{24}^i \cdot p_{23}^i) & p_{14}^{i2} - p_{24}^{i2} & p_{13}^{i2} - p_{23}^{i2}
\end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{5.35}$$

where we can observe that by using a single i -th view we only have 4 equations for 6 unknowns. Hence, in order to solve the 6 unknowns (i.e. when assuming zero-skew, zero principal point and unitary aspect ratio) at least a second view is needed, which leads to 8 equations for 6 unknowns, forming an homogeneous linear system of equations like the following one:

$$\mathbf{A} \cdot \mathbf{q} = \mathbf{0} \tag{5.36}$$

where $\mathbf{q} = (a, b, c, d, e, f)$ is the vector that contains the non-null elements of the absolute dual quadric $\mathbf{Q}^*_{\infty}$, and $\mathbf{A}$ is an $N \times 6$ matrix where N is greater than 6. This linear system of equations, can be solved as in previous cases by means of SVD in order to obtain a solution for $\mathbf{q}$ in LMSE terms.

Notice that because the reference canonical matrix has the form $\mathbf{P}_1 = [\mathbf{I}_{3 \times 3} \mid \mathbf{0}_3]$, such camera matrix cannot be used on (5.35) because it results in a degenerate linear system of equations, and for that reason at least three views (where the first one is discarded) are needed.

Hence, given the SVD of matrix $\mathbf{A}$ as:

$$\mathbf{A} = \mathbf{U} \cdot \mathbf{S} \cdot \mathbf{V}^T \tag{5.37}$$

where $\mathbf{A}$ is an $N \times 6$ matrix, $\mathbf{U}$ is an $N \times 6$ orthonormal matrix, $\mathbf{S}$ is a 6×6 diagonal matrix and $\mathbf{V} = (\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \mathbf{v}_4, \mathbf{v}_5, \mathbf{v}_6)$ is a 6×6 orthonormal matrix and $\mathbf{v}_i$ is the i -th column of $\mathbf{V}$.

Therefore, the LMSE solution for vector $\mathbf{q}$ corresponds to the last column of $\mathbf{V}$ after SVD decomposition of matrix $\mathbf{A}$. Hence,

$$\mathbf{q} = (a, b, c, d, e, f)^T = \mathbf{v}_6 \tag{5.38}$$

However, as we mentioned in (5.25), the last element of vector $\mathbf{q}$ (i.e. f) must be equal to 1. Hence, we normalize vector $\mathbf{q}$ as follows:

$$\tilde{\mathbf{q}} = (a, b, c, d, e, 1)^T = \frac{\mathbf{v}_6}{\mathbf{V}_{66}} \quad (5.39)$$

where $\mathbf{V}_{66}$ is element (6,6) of matrix $\mathbf{V}$ in (5.37).

Finally, from (5.25) we can observe that the **focal length on the first view** α (as a consequence of (5.18)) can be directly obtained from parameter a in vector $\mathbf{q}$ as:

$$\alpha = \sqrt{a} \quad (5.40)$$

and the parameters β , γ and δ that form $\boldsymbol{\pi}_\infty$ the director vector of the plane at infinity can be obtained from parameters a , b , c and d from vector $\mathbf{q}$ as:

$$\boldsymbol{\pi}_\infty = (\beta, \gamma, \delta)^T = \left(-\frac{b}{a}, -\frac{c}{a}, -d \right) \quad (5.41)$$

Because from the assumptions we have made the only unknown intrinsic parameter on each view is the focal length, then we can compute the DIAC on each view by means of (5.16). Besides, we know from (5.15) that the squared of the focal length of i -th view α_i^2 will be located at element (1,1) of the DIAC, which from (5.20) we know it can be computed as:

$$\alpha_i^2 = \mathbf{p}_i^{1T} \cdot \mathbf{Q}_\infty^* \cdot \mathbf{p}_i^1 \Rightarrow \alpha_i = \sqrt{\mathbf{p}_i^{1T} \cdot \mathbf{Q}_\infty^* \cdot \mathbf{p}_i^1} \quad (5.42)$$

And hence we know the focal length for each i -th view on the sequence.

It now only remains to find the projective to metric transformation as will be seen next.

5.3.1.1.- Assuming constant intrinsic parameters

If we assume constant intrinsic parameters, then we can assume that all calibration matrices $\mathbf{K}$ for each i -th view are equal. Hence,

$$\mathbf{K} = \mathbf{K}_1 = \mathbf{K}_i = \begin{bmatrix} \alpha & 0 & 0 \\ 0 & \alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (5.43)$$

where we have assumed zero-skew, zero principal point and unitary aspect ratio.

However, we've seen in (5.42) that each view can have a different focal length α . Hence, as an approximation we can assume that $\mathbf{K}$ contains the mean focal length among all views. Therefore:

$$\mathbf{K} = \mathbf{K}_1 = \mathbf{K}_i = \begin{bmatrix} \bar{\alpha} & 0 & 0 \\ 0 & \bar{\alpha} & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \bar{\alpha} = \frac{1}{N} \sum_{i=1}^N \alpha_i \quad (5.44)$$

And finally, because we saw in chapter 1 that the plane at infinity does not change on any affine, metric or euclidean strata, then expression (5.41) remains unchanged among all views. Consequently the projective to metric transformation given in (5.11) can be written as:

$$\mathbf{T}_{\mathbf{PM}} = \begin{bmatrix} \mathbf{K}^{-1} & \mathbf{0}_3 \\ \pi_\infty^T & 1 \end{bmatrix} = \begin{bmatrix} 1/\bar{\alpha} & 0 & 0 & 0 \\ 0 & 1/\bar{\alpha} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \beta & \gamma & \delta & 1 \end{bmatrix} \quad (5.45)$$

where beta, gamma, delta are the parameters shown in (5.41) and we have used the inverse of the mean focal length shown in (5.44).

It must be finally noticed that this transformation $\mathbf{T}_{\mathbf{PM}}$ can be used to upgrade triangulated 3D points from arbitrary projective stratum to a metric one, and by (5.9) the inverse transformation $\mathbf{T}_{\mathbf{PM}}^{-1}$ can be used to upgrade the camera projection matrices from projective to metric stratum as well.

5.3.1.2.- Assuming variable intrinsic parameters

Oppositely to section 5.3.1.1, if we assume variable intrinsic parameters, then the calibration matrix used for the projective to metric transformation indeed corresponds to the first view in the sequence $\mathbf{K}_1$. Hence using (5.42) we can compute $\mathbf{K}_1$ as:

$$\mathbf{K}_1 = \begin{bmatrix} \alpha_1 & 0 & 0 \\ 0 & \alpha_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (5.46)$$

And because in this case the plane at infinity also remains constant, then the projective to metric transformation now becomes:

$$\mathbf{T}_{\mathbf{PM}} = \begin{bmatrix} \mathbf{K}_1^{-1} & \mathbf{0}_3 \\ \pi_\infty^T & 1 \end{bmatrix} = \begin{bmatrix} 1/\alpha_1 & 0 & 0 & 0 \\ 0 & 1/\alpha_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \beta & \gamma & \delta & 1 \end{bmatrix} \quad (5.47)$$

where again this transformation can be used to upgrade triangulated 3D points to metric stratum, and the inverse transformation $\mathbf{T}_{\mathbf{PM}}^{-1}$ to upgrade camera projection matrices to the metric stratum.

5.3.2.- The Kruppa equations

Self-calibration based on the Kruppa equations was the former self-calibration method that was available. The Kruppa equations are obtained from the correspondence of epipolar lines tangent to a conic. As shown in figure 5.1, given the conics $\mathbf{C}$ and $\mathbf{C}'$ as the images of a world conic $\mathbf{C}_w$ laying on a world plane. Then the duals of the image conics $\mathbf{C}^*$ and $\mathbf{C}'^*$ can be used for line tangency.

Therefore, given two epipolar lines $\mathbf{l}_1 = [\mathbf{e}]_x \cdot \mathbf{m}_1$ and $\mathbf{l}_2 = [\mathbf{e}]_x \cdot \mathbf{m}_2$ on the first view tangent to the dual image conic $\mathbf{C}^*$. Then from (1.33) we have:

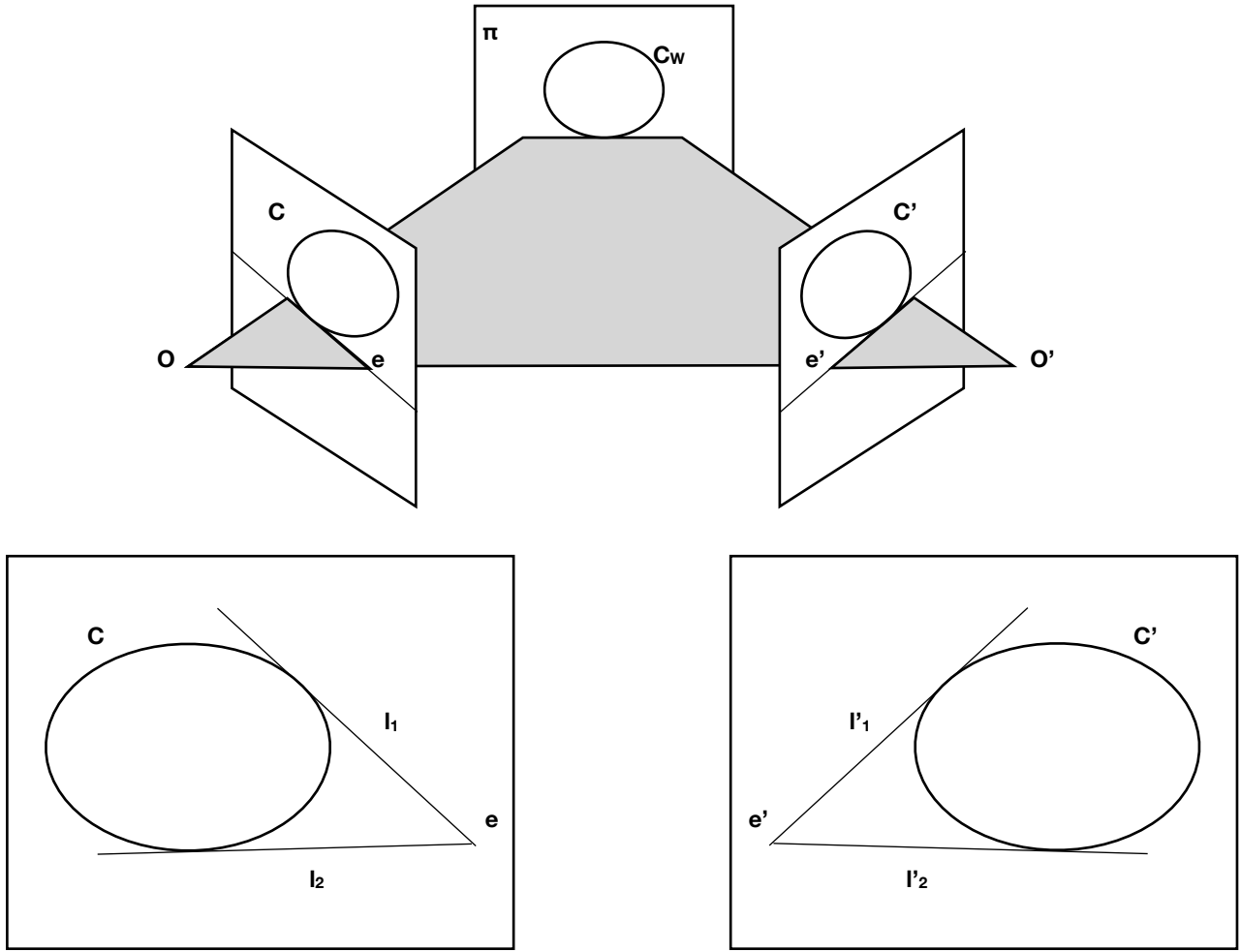


Fig. 5.1. Epipolar tangency for a conic. **e**: left epipole. **e'**: right epipole. **C**: left image conic. **C'**: right image conic. **C_w**: world conic. **π**: world plane. **O**: left camera center. **O'**: right camera center. **l₁** and **l₂**: epipolar lines tangent to conic **C** on the left view. **l'₁** and **l'₂** corresponding epipolar lines tangent to conic **C'** on the right view.

$$\left. \begin{aligned} \mathbf{l}_1^T \cdot \mathbf{C}^* \cdot \mathbf{l}_1 &= 0 \\ \mathbf{l}_2^T \cdot \mathbf{C}^* \cdot \mathbf{l}_2 &= 0 \end{aligned} \right\} \Rightarrow \left. \begin{aligned} -\mathbf{m}_1^T \cdot [\mathbf{e}]_x \cdot \mathbf{C}^* \cdot [\mathbf{e}]_x \cdot \mathbf{m}_1 &= 0 \\ -\mathbf{m}_2^T \cdot [\mathbf{e}]_x \cdot \mathbf{C}^* \cdot [\mathbf{e}]_x \cdot \mathbf{m}_2 &= 0 \end{aligned} \right\} \Rightarrow \quad (5.48)$$

$$\mathbf{C}_t = [\mathbf{e}]_x \cdot \mathbf{C}^* \cdot [\mathbf{e}]_x$$

where **C_t** is a point conic containing all the points **m₁** and **m₂** in the pair of epipolar lines tangent to **C**.

Analogously, using epipolar lines on the second view **l'₁** = **[e']_x · m'₁** and **l'₂** = **[e']_x · m'₂** on the first view tangent to the dual image conic **C'***. Then we obtain the point conic **C'_t** containing all the points **m'₁** and **m'₂** in the pair of epipolar lines tangent to **C'** on the second view as:

$$\mathbf{C}'_t = [\mathbf{e}']_x \cdot \mathbf{C}'^* \cdot [\mathbf{e}']_x \quad (5.49)$$

Because we know from (3.65) that image points can be related by homographies **H** induced by some world plane **π** so that,

$$\mathbf{m}'_i = \mathbf{H} \cdot \mathbf{m}_i \quad (5.50)$$

then according to (1.59) the conic $\mathbf{C}'_t$ on the second view is related to conic $\mathbf{C}_t$ on the first view as follows:

$$\mathbf{m}'_i{}^T \cdot \mathbf{C}'_t \cdot \mathbf{m}'_i = \mathbf{m}_i^T \cdot \mathbf{H}^T \cdot (\mathbf{H}^T \cdot \mathbf{C}_t \cdot \mathbf{H}^{-1}) \cdot \mathbf{H} \cdot \mathbf{m}_i \Rightarrow \mathbf{C}'_t = \mathbf{H}^{-T} \cdot \mathbf{C}_t \cdot \mathbf{H}^{-1} \quad (5.51)$$

Then, combining (5.48), (5.49) and (5.50) we obtain,

$$\begin{aligned} \mathbf{C}'_t &= \mathbf{H}^{-T} \cdot \mathbf{C}_t \cdot \mathbf{H}^{-1} \Rightarrow \\ [\mathbf{e}']_x \cdot \mathbf{C}'^* \cdot [\mathbf{e}']_x &= \mathbf{H}^{-T} \cdot [\mathbf{e}]_x \cdot \mathbf{C}^* \cdot [\mathbf{e}]_x \cdot \mathbf{H}^{-1} \Rightarrow \\ \mathbf{H}^T [\mathbf{e}']_x \cdot \mathbf{C}'^* \cdot [\mathbf{e}']_x \cdot \mathbf{H} &= [\mathbf{e}]_x \cdot \mathbf{C}^* \cdot [\mathbf{e}]_x \Rightarrow \\ -\mathbf{H}^T [\mathbf{e}']_x^T \cdot \mathbf{C}'^* \cdot [\mathbf{e}']_x \cdot \mathbf{H} &= [\mathbf{e}]_x \cdot \mathbf{C}^* \cdot [\mathbf{e}]_x \end{aligned} \quad (5.52)$$

where we have noticed that $[\mathbf{e}']^T_x = -[\mathbf{e}']_x$.

And taking into account expression (3.73), we have that the fundamental matrix $\mathbf{F}$ can be expressed in terms of the homography $\mathbf{H}$ as:

$$\mathbf{F} = [\mathbf{e}']_x \cdot \mathbf{H} \quad (5.53)$$

And taking into account that because $\mathbf{F}$ is defined only up to scale, its transpose can be written as:

$$\mathbf{F}^T = \mathbf{H}^T [\mathbf{e}']_x^T \sim -\mathbf{H}^T [\mathbf{e}']_x^T \quad (5.54)$$

Then (5.52) can be rewritten as:

$$\begin{aligned} -\mathbf{H}^T [\mathbf{e}']_x^T \cdot \mathbf{C}'^* \cdot [\mathbf{e}']_x \cdot \mathbf{H} &= [\mathbf{e}]_x \cdot \mathbf{C}^* \cdot [\mathbf{e}]_x \Rightarrow \\ \mathbf{F}^T \cdot \mathbf{C}'^* \cdot \mathbf{F} &= [\mathbf{e}]_x \cdot \mathbf{C}^* \cdot [\mathbf{e}]_x \end{aligned} \quad (5.55)$$

Finally, if instead of any arbitrary world conic $\mathbf{C}_w$, we use the absolute conic Ω_∞ laying on the plane at infinity Π_∞ , then image conics $\mathbf{C}$ and $\mathbf{C}'$ correspond to the images of the absolute conic in their respective views. Therefore $\mathbf{C} = \omega_\infty$ and $\mathbf{C}' = \omega'_\infty$. And consequently their duals are defined as $\mathbf{C}^* = \omega_\infty^*$ and $\mathbf{C}'^* = \omega_\infty^{*'}.$ Consequently (5.55) becomes:

$$\mathbf{F}^T \cdot \omega_\infty^{*'} \cdot \mathbf{F} = [\mathbf{e}]_x \cdot \omega_\infty^* \cdot [\mathbf{e}]_x \quad (5.56)$$

which is the original form of the Kruppa equations.

By means of the SVD decomposition of the fundamental matrix $\mathbf{F} = \mathbf{U} \cdot \mathbf{S} \cdot \mathbf{V}^T$ where $\mathbf{U} = (\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3)$, $\mathbf{S} = \text{diag}(\sigma_1, \sigma_2, 0)$ and $\mathbf{V} = (\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3)$, and taking into account that epipoles can be written in terms of the columns of $\mathbf{U}$ and $\mathbf{V}$ as $\mathbf{e} = \mathbf{v}_3$ and $\mathbf{e}' = \mathbf{u}_3$ as we saw in (3.15) and (3.23) respectively, then expression (5.56) can also be expressed as the following equalities [8]:

$$\frac{\mathbf{u}_2^T \cdot \omega_\infty^{*'} \cdot \mathbf{u}_2}{\sigma_1^2 \cdot \mathbf{v}_1^T \cdot \omega_\infty^* \cdot \mathbf{v}_1} = -\frac{\mathbf{u}_1^T \cdot \omega_\infty^{*'} \cdot \mathbf{u}_2}{\sigma_1 \cdot \sigma_2 \cdot \mathbf{v}_1^T \cdot \omega_\infty^* \cdot \mathbf{v}_2} = \frac{\mathbf{u}_1^T \cdot \omega_\infty^{*'} \cdot \mathbf{u}_1}{\sigma_2^2 \cdot \mathbf{v}_2^T \cdot \omega_\infty^* \cdot \mathbf{v}_2} \quad (5.57)$$

To find more easily a solution for the DIAC assuming some constraints such as constant intrinsic parameters, zero-skewness, zero principal point and unitary aspect ratio so that the only unknown parameter is the focal length.

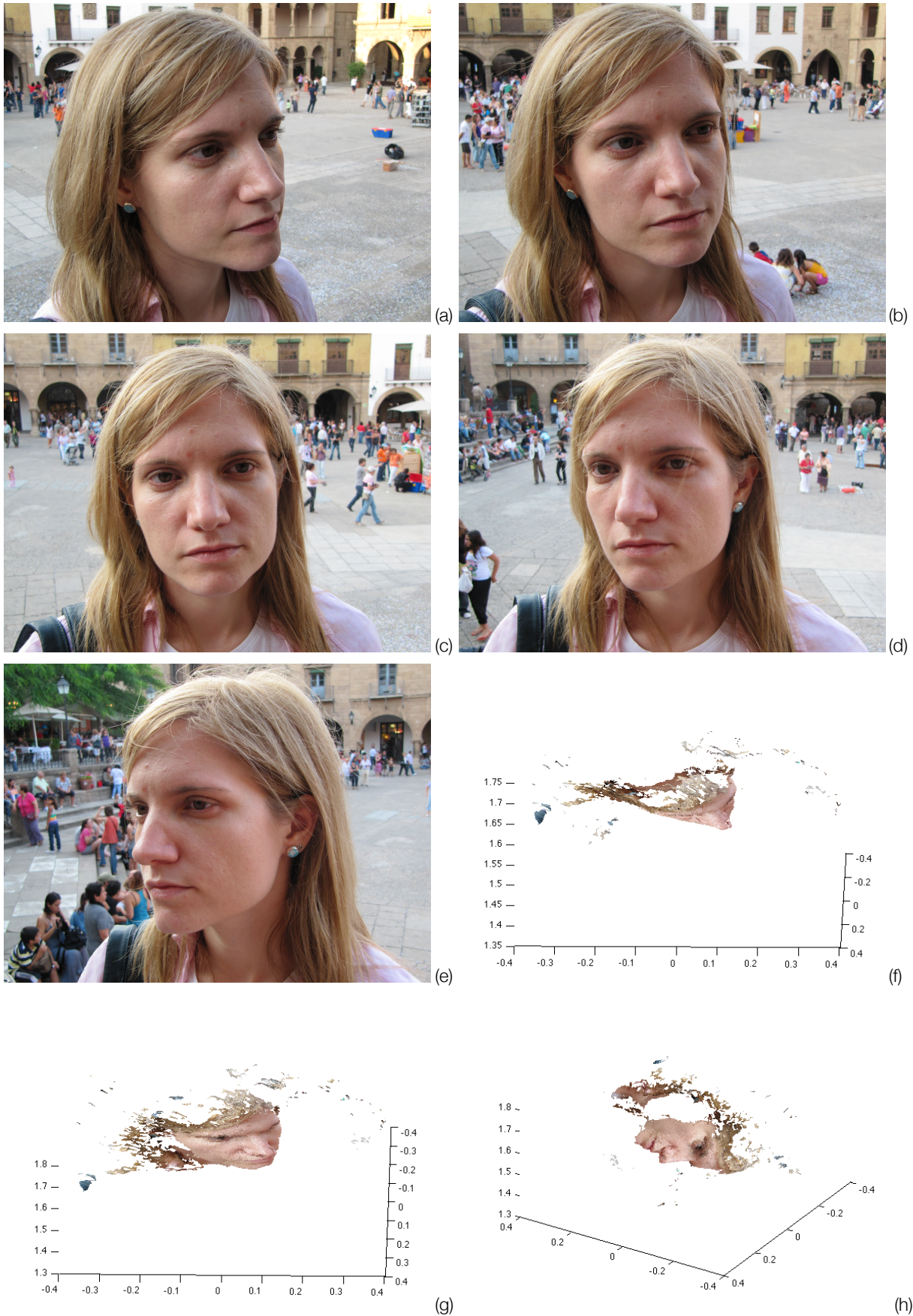


Fig. 5.2. Metric 3D reconstruction from 5 views using a multiresolution technique for disparity estimation and after self-calibrating by means of the absolute dual quadric estimation. (a - e) Original 5 views of the sequence. (f - h) 3D metric reconstruction from different points of view

It can be seen that this method allows the retrieval of the focal length for pairs of images, while the method based on the estimation of the absolute dual quadric needs at least 3 views because the first one (i.e the reference view) cannot be used on the linear system of equations in (5.35). However, this method only estimates the DIAC, and not the plane at infinity, this will result on added degeneracies and for that reason this method is discarded in favor of the estimation of the absolute dual quadric.

5.4.- Critical motion sequences

Similarly to the estimation of epipolar geometry, some degeneracies might also appear when computing self-calibration [8], which makes the process fail resulting in a non-valid 3D reconstruction. For that reason, one cannot put complete trust on self-calibration techniques and some precautions must be taken into account.

For the purpose of this document where general movement of a handheld camera is assumed to obtain a 3D reconstruction of a scene, care should be taken to avoid ambiguous motions sequences, or too restricted motions, such as moving about a single axis, making small movements throughout the sequence of views or covering a small field of view. Motion should be as wide and general as possible.

On the other hand, in this chapter we have made several assumptions on the calibration matrix **K**. Although less assumptions could be made resulting in less equations per view on the estimation of the absolute dual quadric, it is best to **use as much information** known beforehand **on the calibration matrix K** as possible to leave less free intrinsic parameters that could later account for degeneracies on self calibration.

5.5.- Closure

In this chapter we have provided a couple of techniques for self-calibration so that 3D reconstructions can be obtained in a metric stratum.

In chapter 4 we saw how to obtain 3D reconstructions of a scene from multiple views in arbitrary or affine stratum. Indeed in figure 4.7 we saw a 3D reconstruction obtained with 2 views in an affine stratum where distortions could be evidently observed.

Now that we know how to self-calibrate, and since we also know how to obtain a 3D reconstruction from multiple views as we saw in chapter 4. Then, we can compare the results after self calibration using the self-calibration technique based on the estimation of the absolute dual quadric, which we have seen that is more robust than the one based on the Kruppa equations. The results are shown in figure 5.2 where we have used 5 images to obtain the 3D reconstruction of a face. Comparing figure 5.2 with figure 4.7 we can observe that projective distortions have been removed.

PART 2: Photogrammetry

Introduction

In this part of the document we will provide a deep insight into point correspondence estimation either using dense or sparse disparity estimation methods.

The techniques described in this document will allow us to retrieve point correspondences that might later be used to compute camera self-calibration by estimating the fundamental matrices between pairs of images (using the 7 or 8 points algorithms or robust solutions such as RANSAC or LMedS), and when more than 2 views are available to also estimate the intrinsic camera parameters in order to be able to upgrade projective 3D reconstructions to metric ones, which are equal to the real scenario up to scale or rotation.

The dense disparity estimation techniques that will be presented, will also allow to retrieve enough point correspondences, so that once the cameras have been self-calibrated, they can be used to build a cloud of 3D triangulated points by means of the DLT algorithm and proper correspondence corrections using either the Sampson approximation or the Gold Standard algorithm.

Although in this part of the document we will focus just on point correspondence, once the cameras are self-calibrated and the dense point correspondences triangulated, we can use all this information to display a high quality 3D representation of the scene, by joining the 3D triangulated points in order to build a mesh of triangles, and using the input image to apply proper texture to the 3D shape of the scene.

This part of the document will be structured in the following chapters:

- **Image pre-processing**, where we will indicate different techniques that can be used to later improve point correspondence between pairs of images. Among them, we will explain techniques such as blurring, Birchfield and Tomasi dissimilarity measures and non-parametric image transforms such as Census or Rank. We will see that the techniques presented here will only be useful for the most simple techniques for disparity estimation, and that when using robust methods such as the multiresolution technique for dense disparity estimation, no pre-processing will be needed at all.
- **Match measures**, where we will provide different matching metrics to locally compare pixels between pairs of images. We will indicate the most common metrics employed for that purpose, taking special care on metrics such as CV (Covariance-Variance) and NCC (Normalized Cross-Correlation) which will be specially useful as they are invariant to changes in illumination. We will also indicate means of improving these metrics by using truncation, Birchfield/Tomasi dissimilarity measure, proper windowing (i.e. typically Gaussian windowing) or adaptive windowing methods, as well as other robust methods that take into account projective distortions, such as the log-polar transform, which can take into account local rotations and scalings (i.e. any projective distortion can be locally approximated by an affine transformation containing those local rotations and scalings). Finally, we will also indicate an efficient method for fast and dense point correspondence, which will also be useful for the dense disparity estimation method based on the multiresolution technique seen on last chapter.
- **Feature extraction and matching**, where we will indicate means of retrieving image areas that are easier to be compared by using the structural tensor over image gradient data. We will also indicate how to accurately estimate such gradient data by using Simoncelli or Savitzky-Golay filters, and how all that information is related to corner detectors, such as the Harris corner detector among others, which are typically used to locate points for easy sparse point correspondence using simple

match metrics such as CV or NCC, noticing that these sparse correspondences will allow to compute the self-calibration of cameras and recover the sparse 3D structure of the scene.

- **Dense depth estimation.** This is the most extensive chapter as it will contain a collection of different algorithms for dense disparity estimation. We will introduce the reader into the image rectification concept and will provide two algorithms (Hartley's rectification and Pollefe's polar rectification methods) to compare their advantages and disadvantages. We will then indicate the reader why and how to compute a cost functional to be minimized in order to estimate dense disparity and the different constraints that can be applied to obtain piece-wise disparities that follow a correct epipolar geometry. After that, we will show the user cost functional minimization algorithms to estimate dense disparity, which are grouped into local and global methods. Among the local ones we will show a WTA approach and Evolutionary techniques. On the other hand, among the global ones we will show Dynamic Programming algorithms and a multiresolution technique. We will take special care on the latter one as it proved to be the most robust method while at the same time being among the most computational efficient methods. Finally, we will provide post-processing techniques to attain sub-pixel accuracy on disparity estimation by means of polynomial fitting, detect disparity incoherences by using left/right consistency tests and a combined method that uses at the same time left/right consistency tests, image segmentation and curve fitting in order to obtain piece-wise disparity that preserves object boundaries with sub-pixel accuracy.

6.- Image pre-processing

6.1.- Introduction

Before attempting to retrieve matches between pairs of points on two different images, sometimes can be useful to perform some kind of pre-processing in order to obtain better matching results.

This is specially true in cases where pairs of images are obtained with different cameras, which can result in different color tones or different amounts of illumination due to differences in the sensors of both cameras.

Aside from that, since it is usually necessary to obtain sub-pixel accuracy between matched points, other pre-processing techniques that can alleviate aliasing problems between samples arisen by the camera sensor can also be used.

In this chapter we will explain some of these techniques and their appropriate use.

6.2.- Blurring

Some researches [2] have observed that some sensors still retain some aliasing between samples. Depending on the camera lenses, this effect can be neglected as the lens already behaves as a low-pass band filter, thus preventing spatial aliasing between samples. However, this is not always ensured. For this reason it has been proposed to slightly blur input images by using a rectangular, triangular or gaussian averaging filter with a small kernel size.

For instance, by using a simple 2x2 square averaging filter, we can reduce bandwidth to half its value, fulfilling the Nyquist criterion to avoid spatial aliasing.

As indicated, other filter kernels might be used as long as the Nyquist criterion is fulfilled, which is shown below on (6.1).

$$f_s < B / 2 \tag{6.1}$$

For a typical image, in the worst case we can assume that the bandwidth (B) of its data is 1 sample/pixel. Hence, we need to set f_s (sampling frequency) to as much as 1/2 samples/pixel, which can easily be fulfilled with an averaging square filter of at least 2x2 in size.

As mentioned in [2], it has been proved that by using some blurring, matching can slightly improve, however, other techniques such as the Birchfield and Tomasi dissimilarity measure [3], using Gaussian windowing [2][4] or multi-resolution pyramids [5] can usually obtain either better or faster results for the same amount of match quality.

6.3.- Birchfield and Tomasi dissimilarity measure

Birchfield and Tomasi [3] offer a different view with their dissimilarity measure. It is based on using neighbor pixels to obtain minimum and maximum interpolated values. The minimum and maximum values are grouped into pairs of images for each input image, that are then used later for comparison using a metric similar to SAD, which will be explained in chapter 7.

The advantage of Birchfield/Tomasi dissimilarity measure is that it is insensitive to sampling in the sense that two corresponding pixels always have a dissimilarity value near zero. Similar results with traditional stereo algorithms can be obtained by interpolating input images as a pre-processing step, and hence working on sub-pixel resolution levels, (i.e. by using search steps on the DSI smaller than one pixel - see chapter 9 -). However, Birchfield/Tomasi dissimilarity measure obtains similar results with only a slight increase of around 10% in computational time, whereas working with interpolated images on traditional stereo algorithms might result in increases of around 1,100% as stated in [3].

Birchfield/Tomasi dissimilarity is computed as follows:

$$I_R^- = \hat{I}_R\left(x_R - \frac{1}{2}\right) = \frac{1}{2}\left(I_R(x_R) + I_R(x_R - 1)\right) \quad (6.2)$$

where I_R is the right view on the stereo algorithm, x_R is the integer pixel location on right view, $\hat{I}_R$ is an abstract representation of right view image with infinite precision, and I_R^- is the interpolated value of right view located half pixel to the left from x_R assuming that images are rectified and only horizontal disparity is allowed (see chapter 9).

Analogously, one can also define

$$I_R^+ = \hat{I}_R\left(x_R + \frac{1}{2}\right) = \frac{1}{2}\left(I_R(x_R) + I_R(x_R + 1)\right) \quad (6.3)$$

where again I_R is the right view on the stereo algorithm, x_R is the integer pixel location on right view, $\hat{I}_R$ is an abstract representation of right view image with infinite precision, and I_R^+ is the interpolated value of right view located half pixel to the right from x_R assuming that images are rectified and only horizontal disparity is allowed.

Once we know interpolated right view image intensity at locations x_R , $x_R - 1$, and $x_R + 1$, we define

$$I_{R\min} = \min\{I_R^-, I_R^+, I_R(x_R)\} \quad (6.4)$$

$$I_{R\max} = \max\{I_R^-, I_R^+, I_R(x_R)\} \quad (6.5)$$

which are respectively, the minimum and maximum intensity values for each image pixel and its interpolated neighbor images on the right view. The same measures can be defined for the left view as shown on fig. 6.1.

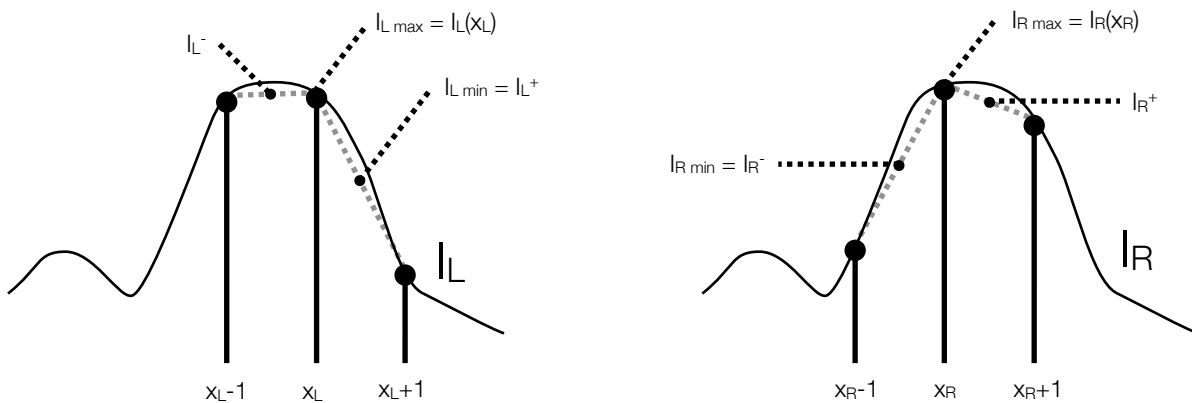


Fig. 6.1. Effect of sampling on image intensities for a pair of corresponding regions, where the measures for Birchfield/Tomasi dissimilarity measures are shown. Left side shows region on left view at locations $x_L - 1$, x_L and $x_L + 1$, and right side shows its corresponding region at locations $x_R - 1$, x_R and $x_R + 1$.

Using expressions (6.4) and (6.5) Birchfield/Tomasi dissimilarity is defined as

$$\bar{d}(x_L, x_R, I_L, I_R) = \max \{0, I_L(x_L) - I_{R_{\max}}, I_{R_{\min}} - I_L(x_L)\} \quad (6.6)$$

which is a slightly different way of defining SAD distance (see chapter 7) between corresponding points x_L and x_R for provided left and right views.

Analogously its symmetric counterpart can be defined as

$$\bar{d}(x_R, x_L, I_R, I_L) = \max \{0, I_R(x_R) - I_{L_{\max}}, I_{L_{\min}} - I_R(x_R)\} \quad (6.7)$$

As it can be seen, Birchfield/Tomasi dissimilarity can be considered as a pre-processing step in the sense that $I_{R_{\min}}$, $I_{R_{\max}}$, $I_{L_{\min}}$ and $I_{L_{\max}}$ need to be computed before comparison, but then a simple metric comparison very similar to SAD is used, which represents a very small overhead over the whole disparity estimation process.

6.4.- Nonparametric Image Transformations

Most metric dissimilarity measures, as will be seen on next chapter, are based on image data, or statistics of such data, within a given windowed region. Hence, if for some reason different camera sensors are used to capture images, or other illumination changes appear between pairs of images, such changes will affect dissimilarity measures.

In order to make dissimilarity measures as invariant as possible to these illumination changes, a series of image transforms exist that modify pixel values into numerical or binary values that only depend on neighbor pixel values while at the same time making them completely independent of their original values of pixel or illumination data.

As explained in [2], [5] and [6] we will give some deeper insight into the RANK and Census transforms in sections below.

6.4.1.- RANK transform

The RANK transform is based on neighbor pixel values in order to set the transformed value of current pixel. To do so without taking into account actual illumination or pixel values, each transformed value is obtained as a measure of how many pixels around current pixel have intensity values higher or lower than current pixel. This can be seen on Fig. 6.2 and is explained below.

27	26	18
19	23	29
17	21	25

Fig. 6.2. The 4 pixels in bold have lower values than pixel in the center (P_c) of neighborhood, and hence the RANK transform of such pixel is 4.

First of all pixels are compared respect to a centered pixel (P_c) in a given region of size $n \times m$ containing pixels P using expression (6.8).

$$\xi(I, P_c, P) = \begin{cases} 0 & \text{if } I(P) \geq I(P_c) \\ 1 & \text{otherwise} \end{cases} \quad (6.8)$$

Then, using (6.8) the RANK transform is defined as the summation of all pixels set to one as follows

$$T[I, P_c] = \sum_{P \in W(P_c, m, n)} \xi(I, P_c, P) \quad (6.9)$$

where W represents the neighborhood region of P_c of size $m \times n$ where all pixels having lower values than centered pixel P_c are counted on each pixel in input image I

Hence, the RANK transform indicates how many pixels are below the intensity of the reference pixel in a given window of size $n \times m$ for each pixel in the input image. An example of the results obtained on a RANK transformed image is shown on fig. 6.3.

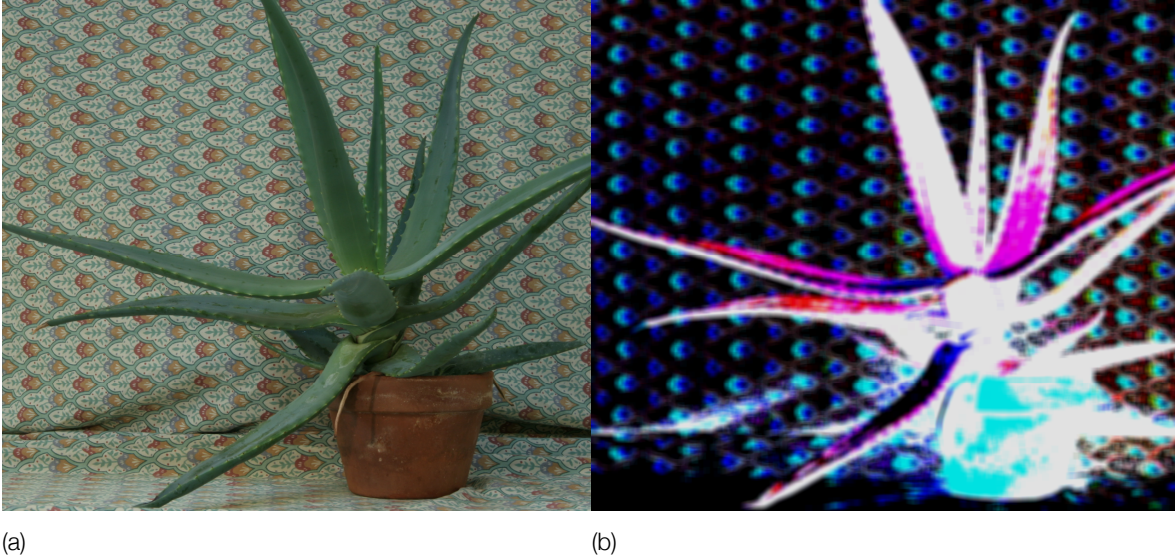


Fig. 6.3. (a) Original Image [40]. (b) RANK transformed image. Notice that colors have been added for representation only and have no meaning.

6.4.2.- Census transform

The Census transform follows a similar approach to the RANK transform, however, in this case transformed pixels are instead a string of binary values indicating whether neighbor pixels in a given window have greater or lower intensity as obtained from (6.8).

In this case, since we obtain a binary array of data per transformed pixel, it is important to define the order in which this array is obtained and keep such order consistent throughout all the implemented dissimilarity measures that use such binary arrays of data. As a general rule, the order can be defined as a clockwise spiral throughout the window, starting at the outer top-left corner and finishing at the center of the window. However, other orders can also be defined arbitrarily.

Once the order of comparison is defined on both left and right views, any binary comparison measure (see section 7.3) can be used, although typically a simple Hamming distance is used as follows:

$$H(\xi_L[I_L, P_{c_L}, P], \xi_R[I_R, P_{c_R}, P]) = \sum_{P \in W} \xi_L[I_L, P_{c_L}, P] \otimes \xi_R[I_R, P_{c_R}, P] \quad (6.10)$$

where $\xi_L[I_L, P_{c_L}, P]$ is the binary representation of a region on left view I_L centered at pixel P_{c_L} using a region W of P points around it (i.e. is the CENSUS transform of P_{c_L}). Analogously the same is defined for the right view centered at a different pixel P_{c_R} , and finally an exclusive or $\otimes$ is used to compare both CENSUS transformed pixels, and the result is summed up to obtain a measure of comparison, which in this case it is called a Hamming distance.

Notice that since each pixel is converted into an array of bits, when windows around transformed pixel grow up in size, arrays might grow to a point where each pixel takes too much memory to fit in a single machine data type (such as a long integer). For instance, for a typical window of 7x7 on a grayscale image, each pixel would take 49 bits, which is twice as much than the memory needed by a typical 24 bits/pixel RGB color bitmap. Besides, on multiple color banded images, this value would be multiplied by the number of color bands making this problem even worse. This is depicted on fig. 6.4, where the CENSUS transform of an RGB image using 7x7 windows needs 2 images (or 6 color channels) to be represented.

When summing up all the memory needed by all transformed pixels in a given image, the required amount of memory can easily grow to a point where it becomes unfeasible to be implemented, specially if we're talking of small embedded devices like cell phones. To avoid this latter problem, Census transform can be simplified to what is called the Sparse Census

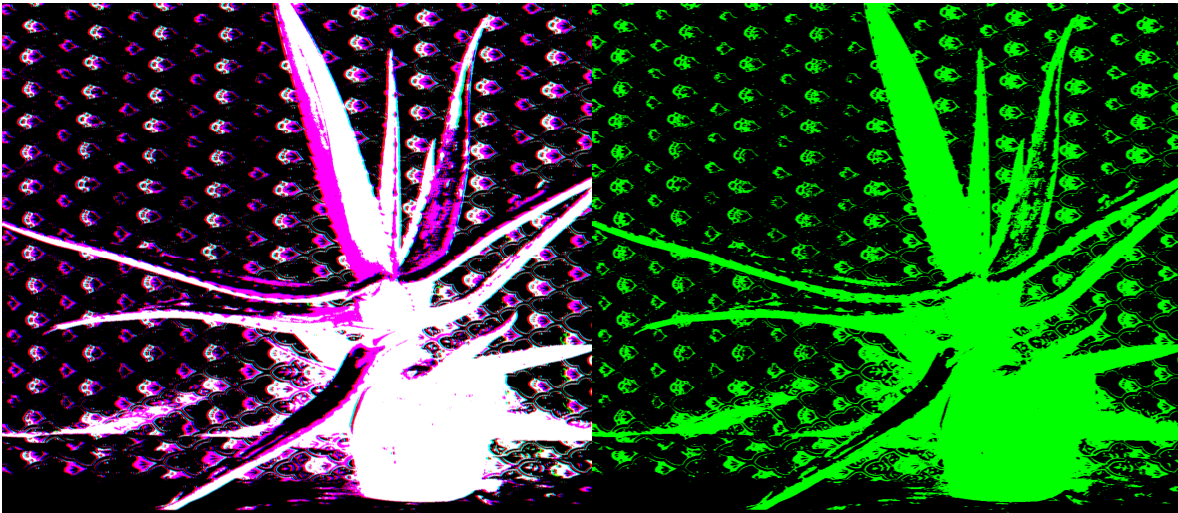


Fig. 6.4. Census transform of image shown on Fig. 6.3.a. A window of size 7x7 has been used for CENSUS transformation, and hence the original RGB color image must be displayed as a 6 color channel image, or two images shown side by side as depicted here. Notice that colors on CENSUS transformed images are arbitrary and only depend on the order chosen for binary representation of transformed data.

Transform, which is discussed below, also for matching purposes the Reduced Census Coding can be used to obtain faster results [5]. The explanation of how Reduced Census Coding works is out of the scope of this text, however, it works in a similar way as the optimized method explained in section 7.8.

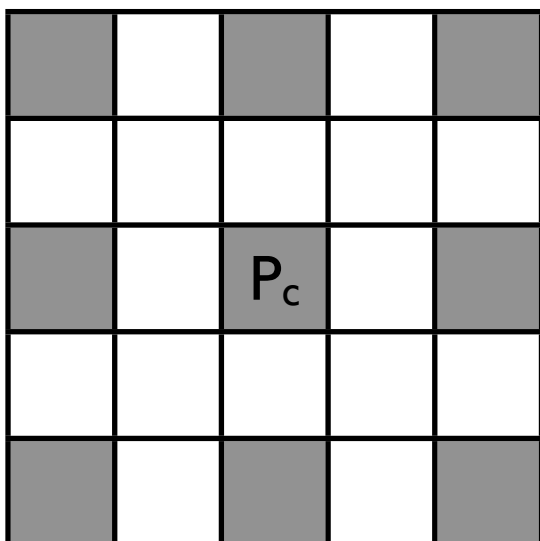


Fig. 6.5. Grey squares represent sparsely selected pixels on a 5x5 window centered around pixel P_c .

6.4.2.1.- Sparse Census Transform

Sparse Census Transform [5] can be seen as a sparse sampling of the binary strings obtained from (6.8) within a given window around a given pixel. Instead of using all pixels in the window following any given order (i.e. a clockwise spiral), we could instead just pick a few pixels at given locations inside the window and determine whether their intensity values are lower or greater than the reference pixel to build the binary strings. This will work as long as the sparsely picked pixels inside the window remain in a consistent order as well. This procedure is depicted on Fig. 6.5.

The advantage of this technique is that similar results to Census Transform matching can be obtained with significantly less memory needed to store each CENSUS transformed image.

6.5.- Closure

In this chapter we have presented some of the most common pre-processing methods that can be applied to images for stereo

correspondence purposes; however as will be seen on the remainder of this text, many of this pre-processing techniques are not needed depending on the matching measures and correspondence techniques being employed.

More specifically, NCC or CV matching measures (chapter 7) are invariant to changes in illumination, and hence non-parametric pre-processing become unnecessary. On the other hand, dense stereo algorithms such as the multiresolution technique shown on chapter 9, estimate disparity with sub-pixel accuracy. In that case, blurring or Birchfield-Tomasi measures become unnecessary as well.

In summary, it can be observed by the reader, that the techniques presented on this chapter will only slightly improve results when using the simplest techniques shown in chapter 9. If more quality is desired on estimated disparities, more robust

techniques can be employed such as the multiresolution one, and consequently many of the pre-processing techniques shown here will not be needed at all.

7.- Match measures

7.1.- Introduction

Matching in terms of photogrammetry is usually referred to finding correspondences between points on pairs of images. For the purpose of 3D reconstruction, this is specially useful on images containing the same static scene as seen from different points of view. Finding point correspondences in such cases allows to recover camera structure and triangulate those matches into 3D points, thus obtaining the 3D reconstruction of the whole scene.

However, in order to find matches between points, several signal processing techniques can be employed by assuming different sets of restrictions, and also different measures can be used to determine whether two pairs of points match better than a different pair.

The former will be discussed more deeply on chapter 9, since once the camera or some sparse structure is known, this can be employed to add restrictions in order to find more point matches in a more reliable way. More information regarding on methods to obtain camera and sparse structure can be found at [5, 7 - 10].

On the other hand, the latter will be discussed on this chapter, as many measures and techniques exist for any given pair of points that need to be matched without any prior knowledge on camera structure or calibration. All these techniques alone suffice to dedicate a whole chapter to explain their implementation, strengths and weaknesses.

7.2.- Distances of image regions

Different metrics to match pairs of points on different images exist, either by using direct image data, or by using transformed image data as obtained by RANK or Census transforms (section 6.4). In this section we will focus on metrics which work directly on grayscale image data, and in subsequent subsections we will explain how to extend these metrics to multichannel color images.

7.2.1.- Matching distances for intensity images

Many matching metrics exist to compare points between pairs of images [5], [6]; most of those metrics take into account local neighborhood around points to be matched, either using a square or any other kind of windowing (i.e. Gaussian).

Hence, for any window $\mathbf{w}$ of size $\mathbf{w}_w \times \mathbf{w}_h$ pixels defining the neighborhood $\mathbf{U}$ around a given pixel:

$$w(i, j) \quad \forall \quad i \in [1, w_h], j \in [1, w_w] \quad (7.1)$$

where $\mathbf{w}_w$ is window width and $\mathbf{w}_h$ is window height in pixels.

We can compute the following metrics on a pair of intensity images $\mathbf{I}_L$ and $\mathbf{I}_R$. (Notice that L and R subscripts stand for left and right views respectively) in order to find best horizontal and vertical disparity values ($\mathbf{d}_x$, $\mathbf{d}_y$) as follows:

Table 7.1. The most common matching distances for intensity images

D_{SAD} - Sum of Absolute Differences

(7.2)

$$D_{SAD} = \sum_{(i,j) \in U} \left| w(i,j) \cdot \left(I_L(x+i, y+j) - I_R(x+d_x+i, y+d_y+j) \right) \right|$$

D_{ZSAD} - Zero mean Sum of Absolute Differences

(7.3)

$$D_{ZSAD} = \sum_{(i,j) \in U} \left| w(i,j) \cdot \left(\left(I_L(x+i, y+j) - \overline{I_L(x,y)} \right) - \left(I_R(x+d_x+i, y+d_y+j) - \overline{I_R(x+d_x, y+d_y)} \right) \right) \right|$$

D_{SSD} - Sum of Squared Differences

(7.4)

$$D_{SSD} = \sum_{(i,j) \in U} \left(w(i,j) \cdot \left(I_L(x+i, y+j) - I_R(x+d_x+i, y+d_y+j) \right) \right)^2$$

D_{ZSSD} - Zero mean Sum of Squared Differences

(7.5)

$$D_{ZSSD} = \sum_{(i,j) \in U} \left(w(i,j) \cdot \left(\left(I_L(x+i, y+j) - \overline{I_L(x,y)} \right) - \left(I_R(x+d_x+i, y+d_y+j) - \overline{I_R(x+d_x, y+d_y)} \right) \right) \right)^2$$

D_{SSD-N} - Normalized Sum of Squared Differences

(7.6)

$$D_{SSD-N} = \frac{\sum_{(i,j) \in U} \left(w(i,j) \cdot \left(I_L(x+i, y+j) - I_R(x+d_x+i, y+d_y+j) \right) \right)^2}{\sqrt{\sum_{(i,j) \in U} \left(w(i,j) \cdot I_L(x+i, y+j) \right)^2 \cdot \sum_{(i,j) \in U} \left(w(i,j) \cdot I_R(x+d_x+i, y+d_y+j) \right)^2}}$$

D_{ZSSD-N} - Zero mean Normalized Sum of Squared Differences

(7.7)

$$D_{ZSSD-N} = \frac{\sum_{(i,j) \in U} \left(w(i,j) \cdot \left(\left(I_L(x+i, y+j) - \overline{I_L(x,y)} \right) - \left(I_R(x+d_x+i, y+d_y+j) - \overline{I_R(x+d_x, y+d_y)} \right) \right) \right)^2}{\sqrt{\sum_{(i,j) \in U} \left(w(i,j) \cdot \left(I_L(x+i, y+j) - \overline{I_L(x,y)} \right) \right)^2 \cdot \sum_{(i,j) \in U} \left(w(i,j) \cdot \left(I_R(x+d_x+i, y+d_y+j) - \overline{I_R(x+d_x, y+d_y)} \right) \right)^2}}$$

D_{CV} - Covariance-Variance

(7.8)

$$D_{CV} = \frac{\sum_{(i,j) \in U} \left(w^2(i,j) \cdot \left(I_L(x+i, y+j) - \overline{I_L(x,y)} \right) \cdot \left(I_R(x+d_x+i, y+d_y+j) - \overline{I_R(x+d_x, y+d_y)} \right) \right)}{\sqrt{\sum_{(i,j) \in U} \left(w(i,j) \cdot \left(I_L(x+i, y+j) - \overline{I_L(x,y)} \right) \right)^2 \cdot \sum_{(i,j) \in U} \left(w(i,j) \cdot \left(I_R(x+d_x+i, y+d_y+j) - \overline{I_R(x+d_x, y+d_y)} \right) \right)^2}}$$

D_{CC} - Cross Correlation

(7.9)

$$D_{CC} = \sum_{(i,j) \in U} \left(w^2(i,j) \cdot I_L(x+i, y+j) \cdot I_R(x+d_x+i, y+d_y+j) \right)$$

D_{NCC} - Normalized Cross Correlation

(7.10)

$$D_{NCC} = \frac{\sum_{(i,j) \in U} \left(w^2(i,j) \cdot I_L(x+i, y+j) \cdot I_R(x+d_x+i, y+d_y+j) \right)}{\sqrt{\sum_{(i,j) \in U} \left(w(i,j) \cdot I_L(x+i, y+j) \right)^2 \cdot \sum_{(i,j) \in U} \left(w(i,j) \cdot I_R(x+d_x+i, y+d_y+j) \right)^2}}$$

where $I_k(x, y)$ stands for the intensity value of k-view (could be left or right view) at coordinates (x,y). And on the other hand $\overline{I_k(x, y)}$ represents the average intensity value of k-view on neighborhood **U** as shown below:

$$\overline{I_k(x, y)} = \frac{1}{N} \sum_{(i,j) \in U} I_k(x+i, y+j) \quad (7.11)$$

where N is the number of elements within region **U** defined by window **w(i,j)**, or in other words: $N = \#U$. A typical size for region U is usually 7x7, however other restrictions can be taken into account to produce better results as will be shown in section 7.7. It should also be noticed that when using uniform windowing, the term **w(i,j)** can be neglected in the expressions above.

We should then choose any of the metrics in table 7.1 in order to find the best disparity values (**d_x**, **d_y**) that produce the best matching values along with different restrictions and minimization algorithms as will be shown on chapter 9. It must also be noticed that metrics in table 7.1 can be used for local and dense disparity estimation.

Notice as well that as we have already mentioned, in order to find the best disparity values, we need to find the best metric value, which for SAD, ZSAD, SSD, ZSSD, SSD-N, ZSSD-N is zero (the lower the match value the better). On the other hand for the remaining metrics (CV, CC and NCC) the match is better as the metric produces a higher value (for CV and NCC the maximum value is 1).

Among all metrics proposed in table 7.1, usually SAD and SSD produce the fastest results, as they can also be easily implemented taking advantage of hardware acceleration, however they also produce the least reliable results. On the other hand CV usually gives the best results at the expense of also being the most computationally complex metric. The nicest feature of CV metric is that it is invariant to linear transformations of the two images being matched [5] as it holds that:

$$D_{CV}(I_L, I_R) = D_{CV}(a_L \cdot I_L + b_L, a_R \cdot I_R + b_R) \quad (7.12)$$

where $a_{L,R}$ and $b_{L,R}$ are constants ($a_{L,R}$ needs to have the same sign). Hence D_{CV} remains robust against changes in illumination.

However, even if D_{CV} is robust against changes in illumination between pairs of images, none of the metrics shown in table 7.1 are robust against projective transformations, which are common in a projective 3D space as long as the motion between pairs of images and the scene is not completely planar. For that reason, robust methods are usually needed as well as further restrictions in order to obtain reliable matches.

7.2.2.- Matching distances for multichannel images

Most of the metrics explained up to this point are defined for grayscale images. However, when working with color or multichannel images, the additional information can be used to improve matching scores [5].

A typical approach might consist on a simple average, or a weighted average of the metrics on each color channel, as shown below:

$$D_w = \sum_{i=1}^B w_i \cdot D_k(I_L, I_R) \quad (7.13)$$

where D_w is the weighted distance metric over all color bands (B) in left and right views (I_L, I_R) using weights w_i for each color band.

However, when referring to multichannel images, it doesn't include just RGB color images. First of all, other color schemes such as YUV exist, and secondly, color channels on images can be used to store not only image data, but other information such as gradient, coherence or orientation data, as it is the case for the structural tensor (section 8.2).

The structural tensor obtains local coherence and orientation information from gradient data, and stores such information in additional color bands along image data. As expected, when using the structural tensor, also the weighted approach can still be used by appropriately setting each weight.

Other approaches in [5] use Frobenius or Minkowsky distances, but in summary they can also be seen as SAD or SSD metrics extended using uniform weighting as shown in (7.13).

7.3.- Distances for bit strings

Some matching algorithms work directly with bit strings. This is specially useful when working with CENSUS transformed images. (section 6.4.2).

When comparing bit strings, many measures can be used, being the most popular among them the Hamming distance [5], [6], as shown in (7.14). However, other metrics for bit string comparison exist as well:

Table 7.2. The most common matching measures for bit streams

D_H - Hamming Distance (7.14)

$$D_H(\mathbf{a}, \mathbf{b}) = \frac{1}{N} \sum_{i=1}^N a_i \otimes b_i$$

D_T - Tanimoto Distance

(7.15)

$$D_T(\mathbf{a}, \mathbf{b}) = \begin{cases} 1 & \text{if } \mathbf{a} = \mathbf{b} = \mathbf{0} \\ 1 - \frac{\mathbf{a}^T \mathbf{b}}{\mathbf{a}^T \mathbf{a} + \mathbf{b}^T \mathbf{b} - \mathbf{a}^T \mathbf{b}} & \text{otherwise} \end{cases}$$

 D_{DK} - Dixon-Koehler Distance

(7.16)

$$D_{DK}(\mathbf{a}, \mathbf{b}) = D_H(\mathbf{a}, \mathbf{b}) \cdot D_T(\mathbf{a}, \mathbf{b})$$

 D_{WT} - Weighted Tanimoto Distance

(7.17)

$$D_{WT}(\mathbf{a}, \mathbf{b}) = \eta \cdot D_T(\mathbf{a}, \mathbf{b}) + (1 - \eta) \cdot D_T(\neg \mathbf{a}, \neg \mathbf{b})$$

where $\mathbf{a}$ and $\mathbf{b}$ represent binary vectors being compared, $\neg$ represents bit negation and $\otimes$ represents exclusive-or (XOR).

Among all metrics in Table 7.2, the Hamming Distance does not give greater importance to 1's respect to 0's present in vectors. The remaining measures usually stress more matches on 1's, although the latest metric (WT) uses parameter η to indicate how important are 1's respect to 0's. For all these reasons, for its simplicity and because Hamming distance does not stress more 1's than 0's, it is the most used binary metric.

7.4.- Statistical Distances

Some distance measures are based on statistics of data. Some of the most typical ones are the Mahalanobis or the Kullback-Liebler distance, which model data as vectors following a given specific PDF (usually Gaussian assumption is taken), and compares the distance between vectors by comparing them against the model using a certain normalization method. A few statistical distances can be seen on Table 7.3.

Table 7.3. The most common statistical distances

 D_G - Gaussian Distance

(7.18)

$$D_G(\mathbf{a}, \mathbf{b}) = e^{-\left(\frac{D_x(\mathbf{a}, \mathbf{b})}{\sigma}\right)^2}$$

where σ is a parameter that controls the width of the distribution. D_x denotes the euclidean norm between vectors $\mathbf{a}$ and $\mathbf{b}$, and hence D_x^2 can be seen as D_{SSD} . Also Mahalanobis distance can be used instead if σ is set to 1.

 D_{MAH} - Mahalanobis Distance

(7.19)

$$D_{MAH}(\mathbf{a}, \mathbf{b}) = (\mathbf{a} - \mathbf{b})^T \mathbf{A}^{-1} (\mathbf{a} - \mathbf{b})$$

where $\mathbf{A}$ is the covariance matrix which is computed for a given population of data points, from which $\mathbf{a}$ and $\mathbf{b}$ are drawn.

$$D_{CHY}(\mathbf{a}, \mathbf{b}) = \log \left[1 + \left(\frac{D_x(\mathbf{a}, \mathbf{b})}{\tau} \right)^2 \right]$$

where τ is a parameter that controls height and tails of the Cauchy distribution. and D_x is a norm (usually an Euclidean distance) between $\mathbf{a}$ and $\mathbf{b}$.

7.5.- Distances based on information theory

Other methods of matching are based on histogram computation for each image on their respective regions. Once histograms (which can also be seen as rough estimations of PDF's) are computed, different measures of comparison can be used, most of them based on entropy concepts [5].

Table 7.4. The most common information theory distances

$$D_{KL}(p, q) = \sum_{x \in A} p(x) \log_a \frac{p(x)}{q(x)}$$

where p and q are two PDF's. We assume that $0 \cdot \log(0/0) = 0$, $0 \cdot \log(0/q) = 0$, and $p \cdot \log(p/0) = \text{inf}$. DKL is a measure of discrepancy between two distributions p and q , besides Kullback-Leibler distance it is also called relative entropy.

$$D_J(p, q) = \frac{1}{2} (D_{KL}(p, q) + D_{KL}(q, p))$$

where D_{KL} is the Kullback-Leibler distance. The nice feature of D_J is that its measure is symmetric, opposite to D_{KL} , which means that $D_J(p, q) = D_J(q, p)$, or in other words, distance from p to q is equal to distance from q to p , which is the common behavior for most distances we are used to.

7.6.- Improving matching values

Some matching techniques that have previously been explained (especially the ones based on direct image data comparison, as seen on section 7.2) can be divided in two stages [2]:

- A pixel matching stage: which is usually a simple difference or product between pixels inside a given window.
- An aggregation stage: which consists on summing up all pixel matches. Usually some kind of weighting can also be used instead of just using a square window (uniform weights). The most typical weighting is the one that yields Gaussian windowing.

As a result, when using the matching metrics shown in previous sections, some techniques can be used to further improve matching results. Among those techniques, we can use truncation [2], [11], [12], [13] or Birchfield-Tomasi dissimilarity measure [3] at the pixel matching level. Other techniques can also be used at aggregation level, which will be explained later.

7.6.1.- Truncation

Truncation is based on putting limits to the maximum differences between pixels in regions being compared in a pair of images just before the aggregation stage. This is specially useful in methods such as SAD or SSD (or their zero-mean counterparts). When a given pixel dissimilarity surpasses the maximum dissimilarity threshold, it simply gets truncated to such threshold value.

The idea of truncation is to provide better match values when new objects appear in regions being matched, which is specially true at objects borders, where most occlusions are located. In such cases, new objects usually yield high pixel differences over small parts of windows being compared, however after the aggregation stage, these differences get summed up as well and diluted among all pixels inside the windows which in the end produces slightly different aggregated match values which can also cause matching algorithms to detect slightly wrong matches.

By using truncation when mixed populations of pixels exist (foreground and background points, especially at object borders), we can limit the influence of wrong matches and slightly improve the estimated disparity [2]. Good truncation values range from 5 to 50, typically around 20.

It must be noticed that if truncation values are chosen too low for pixel dissimilarity, just below the noise level (e.g. 1 or 2 levels of intensity), errors might become very large; it is then advised not to choose a very strict truncation value. It also should be noticed that the larger the chosen truncation value is, the less effect truncation will have on overall disparity estimation, being the extreme case of a truncation threshold set at infinity which has no effect at all.

7.6.2.- Birchfield-Tomasi dissimilarity measure

Birchfield/Tomasi match [3] approach attempts to reduce the effects of spatial aliasing on samples of images by computing a pair of images for each image containing the minimum and maximum intensity values between each pixel intensity and the interpolated intensities of neighbor pixels, just as it was shown on section 6.3.

Once the pair of new images are computed, they are used for matching against another image using any of the match measures explained on this chapter, although as it was shown on section 6.3, usually a measure similar to SAD is used.

7.7.- Aggregation of match measures

As has been explained on section 7.6, many matching measures rely on an aggregation step, which consists on a weighted summation of all pixel matches within a given window.

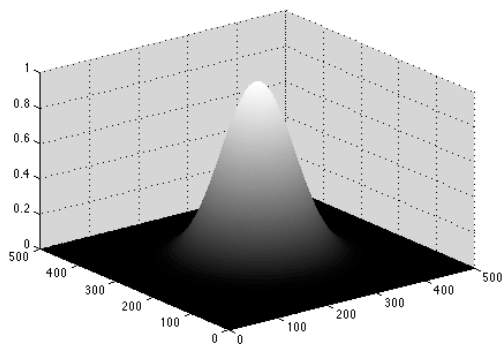


Fig. 7.1. A depiction of a Gaussian window

As will be seen, in the aggregation stage different windows can be used with different properties, and also windows can be modified adaptively in order to always obtain the best possible matches.

7.7.1.- Gaussian windowing

Among all possible windows, aside from uniform windowing, Gaussian windows are probably the most common ones [5].

The main advantage of Gaussian windowing respect to the uniform one is that it becomes more robust when new objects appear in the region of comparison, which is especially important at object boundaries where most occlusions are located.

Gaussian windowing can be seen as a uniform windowing at its center that gets extended in such a way that the further a pixel is from window center, the less important it is considered (see fig. 7.1). This way, when moving the window around to search for the best match, match values change smoothly as new objects appear in the window, whereas uniform windowing yields a more trembling variation of match scores.

Since match values change smoothly, many algorithms can take advantage of that to iteratively search for the best pair of matched points, even with sub-pixel accuracy, whereas these algorithms would probably get stuck on suboptimal matches

when using uniform windowing. Some of the most typical algorithms used for such optimization are minimization algorithms such as Gradient Descent, Levenberg-Marquardt, or evolutive algorithms such as Simulated Annealing. A deeper insight on such algorithms and others, as well as the different restriction criteria that can be used will be seen on chapter 9.

It must be noticed however that Gaussian windowing has one big drawback. Since weights decrease as pixels are further away from window center, effective window size (the area near the center that can be considered to be nearly uniform) can become too small if Gaussian weights decrease too fast. To avoid such problem, usually when using Gaussian windowing, bigger window sizes respect to uniform windowing are needed (usually about 4 times larger on each direction).

Since bigger windows are needed, this translates into a much higher computational effort to obtain matches, especially when many matches need to be computed, as is usually the case for dense disparity estimation. In order to address this issue and many others, other techniques such as adaptive windowing can be used instead.

7.7.2.- Adaptive windowing

Adaptive windowing is based on the concept of modifying the window region for comparison between pairs of images depending on factors such as whether a given area is highly textured or not, or if it's assumed that matching can be easily obtained or not.

Many adaptive windowing techniques exist as can be seen on [2, 5, 12 - 16], however we will focus on techniques which are based on modifying window size depending on the factors we mentioned above.

First of all, it must be noticed that as the window of comparison increases in size, usually matches become more robust. However, at the same time window increases, matches usually become very smoothed, usually up to the point where object contours are completely lost. Hence, it is necessary to find the optimum window size so that matches are robust enough but object boundaries are preserved as well. However, since common images usually contain some highly textured areas and some other areas where matching is highly inconsistent due to lack of texture, it can be easily seen that the same window size is not the optimum one in all areas, as we need the smallest window in each area that yields robust matching while preserving boundaries. As a consequence, methods to estimate appropriate window size are needed. Two methods will be explained based on pixel similarity and coherence on structural tensor.

7.7.2.1.- Window size based on pixel similarity

This method is based on the idea of [12]. Basically, when high similarity exist between pixels located at the center of the window, it will be assumed that matching is being done on a low textured region, and hence a bigger window size is needed. Oppositely, if high dissimilarity is present between center points in windows being matched, then window size will be reduced assuming that we are in a region with high texture that can be matched more easily. This concept is summarized on the expression below, where we define the window that can be used on any of the metrics (7.2 - 7.10) that we have already seen.

$$w(i, j) = e^{-\left(\left(\frac{2i}{w_w}\right)^2 \sigma_i + \left(\frac{2j}{w_h}\right)^2 \sigma_j\right) \frac{\Delta_c}{\gamma_c}} \quad (7.23)$$

where $w(i, j)$ is the window defined for horizontal i and vertical j coordinates, w_w is window width in pixels, w_h window height in pixels (as defined in (7.1)), σ_i is the horizontal exponential girth factor, σ_j is the vertical exponential girth factor, γ_c is center pixel dissimilarity factor and Δ_c is center pixel dissimilarity.

It can be seen from expression (7.23) that the larger the exponential girth factors (σ_i , σ_j) are, the faster the exponential window decreases, and hence the smaller its effective size becomes. Typical values for exponential girth factors are around 4, which translates on window values of approximately 0.01 at window borders.

On the other hand Δ_c indicates how similar image intensities located at window centers are, and can be computed as follows:

$$\Delta_c = \left| I_L(x, y) - I_R(x + d_x, y + d_y) \right| \quad (7.24)$$

where I_L stands for left view image, I_R for right view image, x for horizontal coordinate (which in this case corresponds to the horizontal coordinate of window center on left view), y for vertical coordinate (which analogously corresponds to vertical coordinate of window center on left view), d_x for horizontal disparity and d_y for vertical disparity, where $(x + d_x, y + d_y)$ are the coordinates of window center on right view.

It can be seen that expression (7.24) is very similar to a SAD metric like the one shown on (7.2) but on a region of a single pixel, and hence a small Δ_c value indicates that centers on left and right windows are very similar, and so we can assume that we are located on a low textured area that needs bigger window sizes for matching. Oppositely, the larger Δ_c becomes, the smaller the effective size of window on expression (7.23) becomes.

Finally the center dissimilarity factor (γ_c) indicates how much effective window size (i.e. exponential girth) should be increased or decreased depending on similarity of centers on left and right windows. The higher this value is, the less effective window size will change. Our experiments have shown that a typical value of 40 is good enough for most situations (when using image pixels with values ranging from 0 to 255).

It must be noticed, that the main drawback of this adaptive windowing method is that estimation of amount of texture might not be very robust as only one pixel is being matched without any support region. This lack of robustness will result in effective window sizes with a slightly random variation. To avoid such problem, diffusion techniques can also be employed as in [4] and [17].

7.7.2.2.- Window size based on coherence of structural tensor

As we will see on chapter 8, the structural tensor allows retrieval of coherence on each pixel inside an image. Coherence is expressed numerically as values ranging from zero (no coherence) to one (maximum coherence). The details on how these values are computed will be explained on section 8.2, however, it must be noticed that the more coherent an area is, usually smaller windows are needed. This can be expressed as follows:

$$w(i, j) = e^{-\left(\left(\frac{2i}{w_w} \right)^2 \sigma_i + \left(\frac{2j}{w_h} \right)^2 \sigma_j \right) \frac{c_c}{\gamma_c}} \quad (7.25)$$

We can see that (7.25) is very similar to (7.23) except for the term that controls exponential girth (c_c), which now depends on coherence of window centers instead of center similarity and can be computed as a mean coherence between left and right window centers, just as shown below:

$$c_c = \frac{c_L + c_R}{2} \quad (7.26)$$

where c_L stands for coherence of window center on left view and c_R for coherence of window center on right view. Both values range from zero (no coherence) to one (maximum coherence), as well as their mean value (c_c).

Oppositely to section 7.7.2.1, it must also be noticed that when using expressions (7.25) and (7.26), in this case a typical value for γ_c is around 0.2.

Even though this method might seem more reliable than the adaptive method on section 7.7.2.1, as will be shown on next chapter, it usually happens that most coherent areas are located just at object boundaries. This fact can lead to a big handicap to be solved, namely, occlusions. Occlusions are usually located at object boundaries and are due to areas of images that only appear in one view of a pair of views because an object is covering them in one of the views. This will cause that matches found on those areas will usually be incoherent (i.e. do not pass a left/right consistency check) and will usually be wrong because a region in one image does not appear in the other view, and hence cannot be correctly matched.

In any case, most matching algorithms suffer from the same mismatching problem on occluded areas. The good news is that usually if matches cannot be correctly estimated or corrected in occluded areas, at least such areas can usually be detected and avoided with a simple consistency check [18]. That way, even if occluded areas are present in an image, mismatches won't affect at structure recovery or calibration steps since those matches will be simply ignored.

7.8.- Efficient Computation of Distances

Up to this point, most of the matching distances that have been explained can be applied locally to detect matches of sparse points in pairs of images. However, sometimes sparse correspondence is not enough, and dense disparity estimation is needed. This is especially true for 3D reconstruction.

As will be seen on chapter 9, there are different methods for dense disparity estimation, however two main groups exist: methods based on local disparity estimation and methods based on global disparity estimation [5]. The former is based on the distance measures that have been already explained, but the latter can take advantage of some optimizations since it needs computation of match values for all possible disparity values between each pair of points in a pair of images. It must be noticed however, that global disparity methods usually require more resources in terms of computational time and memory. However, some exceptions to this rule exist, as will be seen next.

In cases where computation of match values for all possible disparity values (within a given limit or search range) is needed, in order to compute a dense disparity estimation, instead of computing those matches separately using any of the expressions in sections 7.2, 7.3 or 7.4 we could take advantage of the fact that all possible disparity values are checked during WTA search (see chapter 9) in order to compute those matches more efficiently and avoid duplicate re-computations. This can be better seen on figures 7.2 and 7.3.

Figure 7.2 shows how disparity is computed for each point separately by checking all possible disparity values on right view until the best match for each point on left view is found, whereas fig. 7.3 shows that we can make computations in a different order to obtain faster results. The idea in this case is based on "shifting images" and compute all matches on all points for a given disparity value that lay on the overlapped region between shifted views, repeat the process for each disparity value, and pick on each pixel the disparity value that produces the best match.

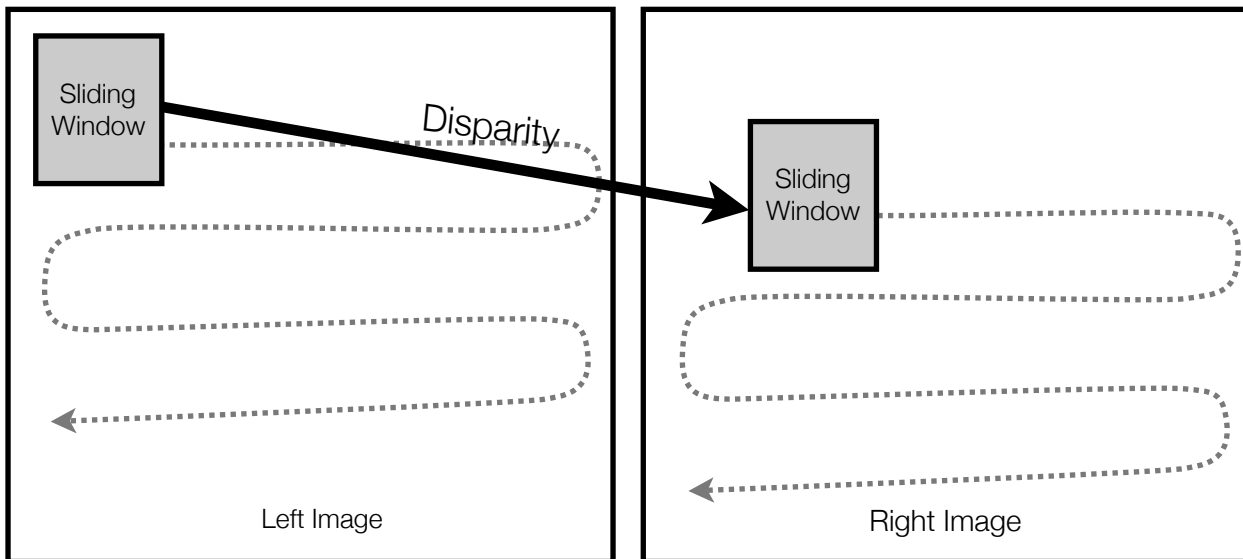


Fig. 7.2. Local WTA disparity minimization.

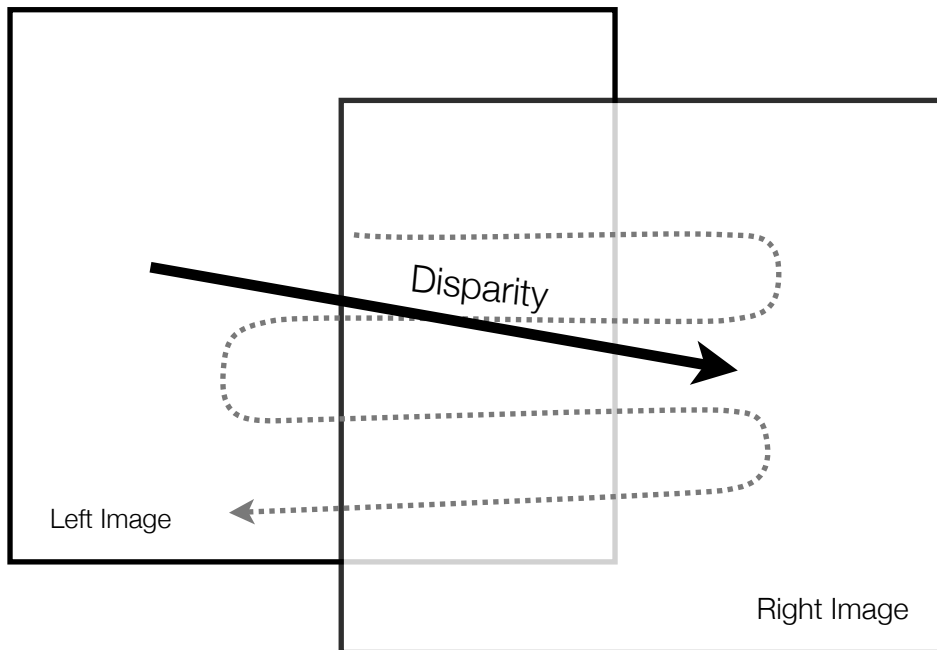
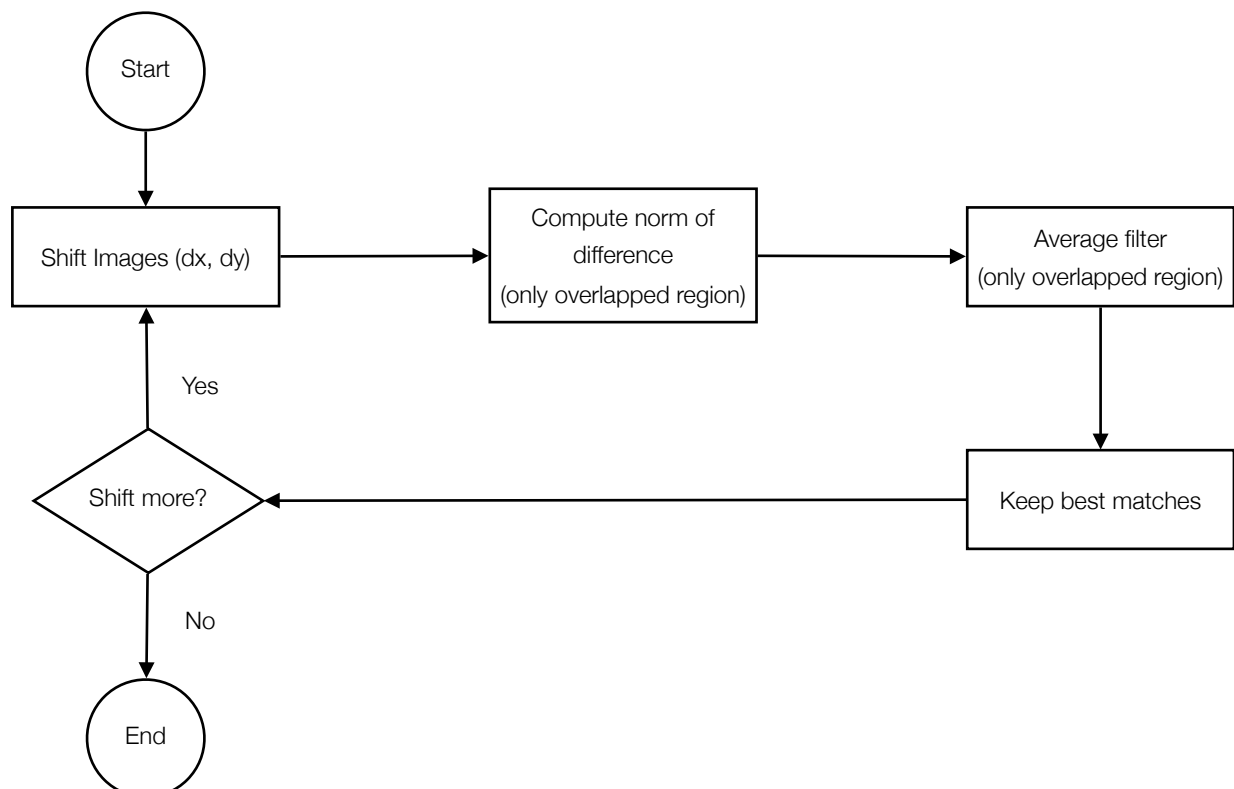


Fig. 7.3. Global WTA disparity minimization

An example of this process is given in algorithm 7.1 for a simple SAD distance measure. As we already know SAD on a uniform window consists on computing the aggregation of intensity differences of images in a given region. Algorithm 7.1 shows that this can be obtained first by shifting the images to the desired amount of disparity, then computing the differences of the shifted images on the overlapped region, and finally computing the aggregation by means of a simple averaging filter with a kernel of the same size as the desired window size. The best disparity matches are kept for all pixels and finally this process is repeated for each disparity value.

Algorithm 7.1. Efficient disparity estimation using Global minimization and SAD



It can be easily seen that this algorithm can simply be extended to non-uniform windowing by using any other type of filtering (i.e. an average filter with gaussian kernel to obtain gaussian windowing), and also can be seen that other measures can be computed. For instance a cross-correlation (CC) or normalized cross-correlation (NCC) can be computed by substituting the differences of the shifted images by their products. Finally, for the case of zero-mean measures such as a covariance (CV) measure, before starting the algorithm, images can be pre-processed using a high-pass filter of the same size as the region of interest in order to remove the DC component.

It must be noticed however, that this method is not valid for all distance measures (for instance, adaptive windowing cannot be implemented this way), and as will be seen on chapter 9, this is just the implementation of a simple Winner Takes All (WTA) minimization approach without taking into account any restrictions. It will be seen that other minimization techniques exist (although many cannot be implemented this way), and that some restrictions (such as epipolar restrictions) can also be used as well.

7.9.- A robust match measure

Even though many matching measures have been explained up to this point, most of them suffer from the same inconvenience when attempting to match pairs of images having a wide baseline (very different points of view), namely, matches fail.

Although no algorithm is perfect, and no matches can be effectively computed if points of view are too different because usually points to be matched disappear from the scene or get occluded; matching reliability can be improved, which will allow us to use slightly wider baselines by using the robust match techniques as follows.

In this section we will mainly focus on robust match measures based on the log-polar transformation [5], but other methods can be used instead [7], and all of them are usually based on the same idea: correcting projective transformations.

It can be easily observed that for small regions, projective transformations (due to different points of view in the cameras) can be approximated around such regions by an affine transformation (i.e. rotation and scale variation). In other words, for small regions to be compared on images having a wide baseline, it can be seen that objects might appear in the scene slightly rotated or scaled respect to the other image. This fact makes matching more difficult, since all the measures seen up to this point assume that regions to be matched have simply been translated. Hence, it is necessary to estimate by some means the amount of rotation (or slanting) and scaling) that appears between a pair of regions in order to correct such distortions and be able to correctly match them.

7.9.1.- The Log-polar transformation

The Log-polar transformation is based on warping images so that they are expressed in terms of angle (φ) and scale (r) (i.e. polar coordinates) instead of horizontal and vertical coordinates. It must be noticed however, that the scales are expressed in logarithmic scale instead of a linear one as it happens with common polar transformations.

The expressions to warp images are the following:

$$r = \log_B \left(\sqrt{(x - x_0)^2 + (y - y_0)^2} \right) \quad (7.27)$$

$$\varphi = \arctan \frac{y - y_0}{x - x_0}, \quad \text{for } x \neq x_0 \quad (7.28)$$

where (x_0, y_0) are the coordinates of the center of transformation, which will usually be located at the center of regions $w(i,j)$ being matched.

However, as in many warping algorithms, usually the inverse of the warping transformation is used, so that coordinates are obtained over the original image to interpolate pixels, and gaps or jagged results are avoided, just as follows:

$$x = B^r \cdot \cos(\varphi) + x_0, \quad y = B^r \cdot \sin(\varphi) + y_0 \quad (7.29)$$

assuming that $B > 1$, and $0 \leq r \leq r_{\max}$, $0 \leq \varphi \leq 2\pi$. Where r is expressed as vertical transformed coordinates and φ is expressed as horizontal transformed coordinates; then even though B can be set arbitrarily, in order to fit all possible r values between 0 and $r_{\max}$, we usually set B as:

$$B = \frac{r_{\max}}{\sqrt{d_{\max}}}, \quad d_{\max} > 1, \quad r_{\max} > 1 \quad (7.30)$$

where $d_{\max}$ is the maximum diagonal distance on input image between center (x_0, y_0) of region being transformed and any of its corners, and $r_{\max}$ can usually be set to $r_{\max} = 1$. Finally notice that output log-polar transformed image can be computed with the desired size by defining the appropriate incremental steps for coordinates r and φ .

The main advantage on using a log-polar transformation is that differences in scale and rotation are transformed into shifts of the images along the scale or rotation axes, which will help us to find robust matches even when scaling and rotation is present. This can be seen on fig. 7.4.

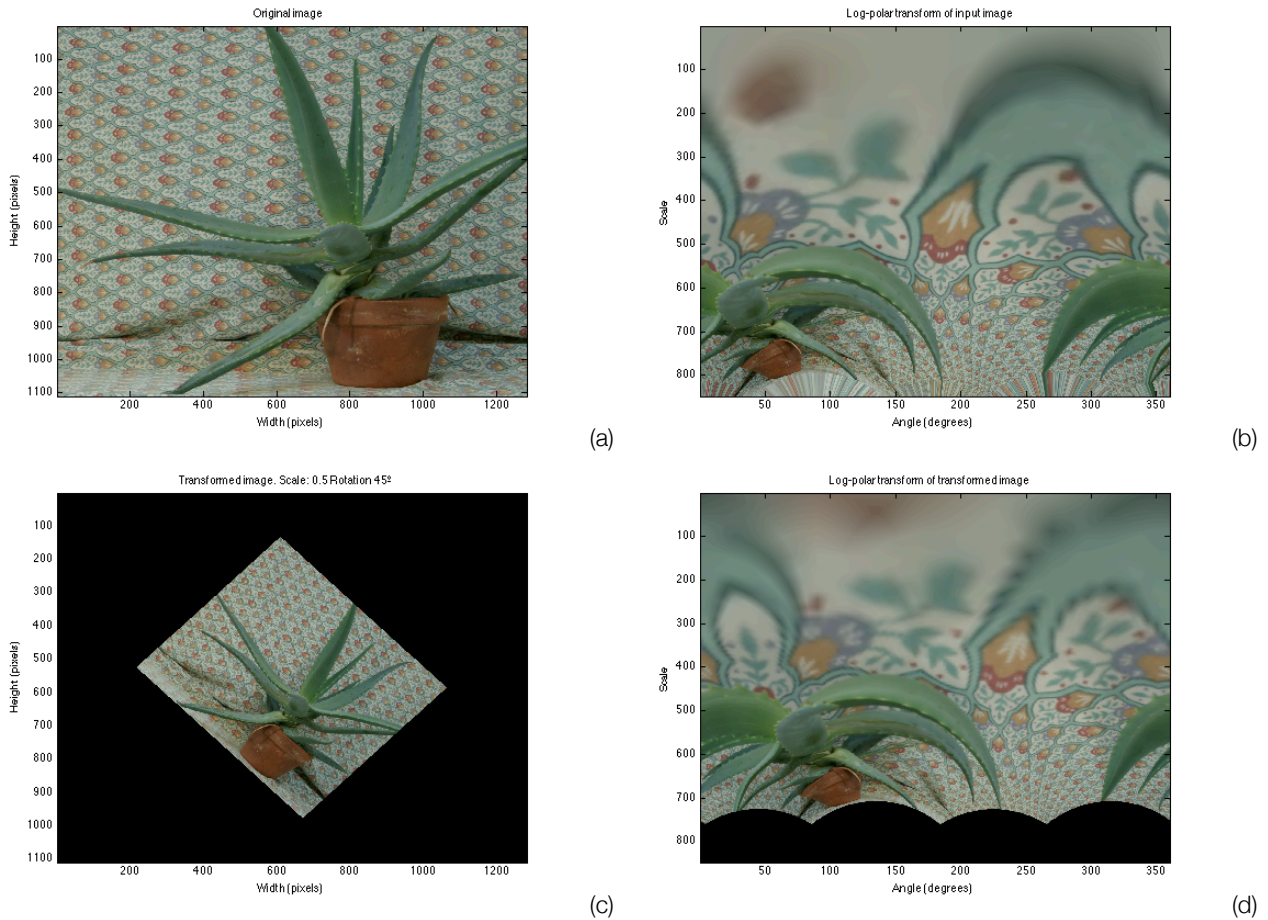


Fig. 7.4. (a) Original image. (b) Log-polar transformation of original image. (c) Rotated and scaled image. (d) Log-polar transformation of rotated and scaled image.

7.9.2.- Matching of the log-polar transformation

As we have already explained on previous section, differences on scale and rotation are seen as shifts along their respective axes, hence, in order to retrieve the corresponding scale and rotation variations between a pair of images, we just need to match their log-polar transformations using any of the methods that have already been explained. Our practical results however show that a covariance (CV) match measure using Gaussian windowing yields the best results.

In order to obtain the matches, we can just follow a WTA (Winner Takes All) approach by shifting the log-polar transformation of one image along the log-polar transformation of the other image, and pick the scale and rotation that yields the best

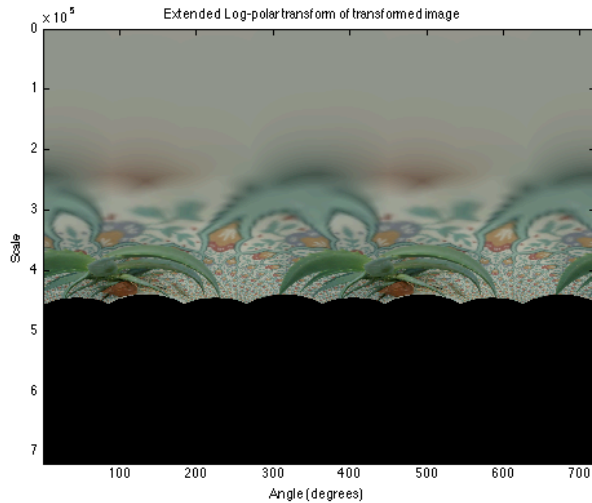


Fig. 7.5. Extended log-polar transform. In order to retrieve scale and rotation respect to original image, the extended log-polar transform should be matched against transform on fig. 7.4.b using a WTA approach.

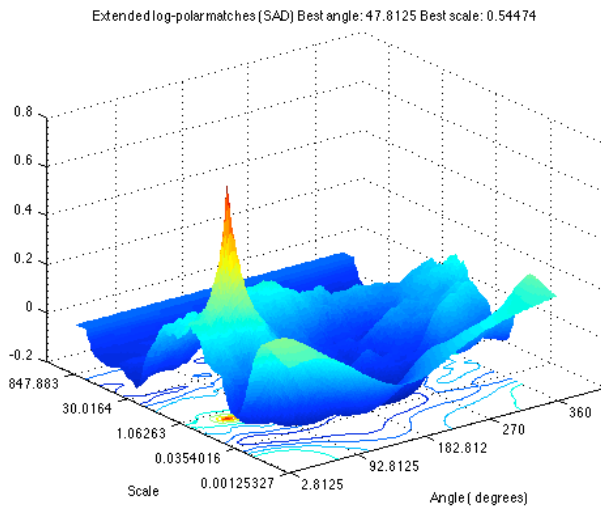


Fig. 7.6. Matching of log-polar transforms on fig. 7.5 and 7.4.b using Covariance-Variance (CV) distance metric. Highest peak indicates best scale and angle.

Even though this techniques is very expensive in computational terms, it can also be seen that it might not be used for dense disparity estimation, but instead it can be used to sparsely find correspondences between points that can later be used in techniques such as self-calibration or epipolar

match (see fig. 7.6). It must be noticed however, that in order to use this technique, one of the log-polar transforms needs to be extended so that the match search can be done along all different scales and rotations, otherwise when shifting the transformed windows we would end up outside of image limits and matching could not be done. This is all depicted on fig. 7.5. As it will logically be observed, the log-polar transform is periodical along the angle axis, which implies that extension over the angles is only done to twice the original size, whereas for scales, it is extended both for smaller and larger scales up to three times the original size. In other words, in order to extend the log-polar transform we need to define $\varphi_{\min} = 0$, $\varphi_{\max} = 4\pi$, $r_{\min} = -1$ and $r_{\max} = 2$.

Up to this point we have been able to recover scale and rotation variation, however, the purpose of this techniques is to robustly obtain matches between points. This can be obtained by using for instance a WTA search along 4 dimensions (horizontal disparity, vertical disparity, rotation and scale). Or in other words, for each disparity value to be checked we need to search for best rotation and scale matches so that their effect gets corrected, and there's no need to actually rotate or scale the images around the regions being matched, instead their log-polar transformed versions are matched.

It can be seen that this technique is among the most robust ones to compute point correspondence, however it is also among the most computationally intensive ones, since a 4D search needs to be done. This problem can be reduced to 3 dimensions as will be seen on the following chapters if we use epipolar restrictions or rectified images. However, it's still more computationally intensive than other solutions.

geometry estimation, where only a handful of corresponding points are needed (at least 7). Hence, this restriction on the number of correspondences makes this technique suitable for such purposes, although other techniques such as CV matching might obtain similar results with less computational effort if baseline separation between pairs of images is not too big.

7.10.- Closure

In this chapter we have seen different measures that can be used to match pairs of pixels in two different images. Several measures have been provided, however, our experimental results have proven that the most convenient ones are CV or NCC as they provide more robust respect to other measures and normalized values that can easily be compared within a given range even though they might seem more computationally intensive than a simple SAD measure.

More importantly, we have shown that CV measure is invariant to changes in illumination, hence by using this measure we can avoid pre-processing on input images (see chapter 6). Besides, if images are high-pass filtered (i.e. by computing a Laplacian of Gaussian pyramid as seen on chapter 9, or their gradient as seen on chapter 8), we can then equivalently work with NCC measures which are faster to compute than CV to obtain the same results in this case.

We have also mentioned methods to make measures more robust, such as adaptive windows, truncation, or log-polar transforms. Our results have also proven that by using a robust technique such as multiresolution, we can avoid many of this improvements and simply use a Gaussian window to ensure smoothness of matching scores.

Finally, we must make a final comment regarding the efficient method to compute matching distances for dense disparity estimation on section 7.8. Although we have presented this method as a faster way of computing dense disparity using a WTA approach (see chapter 9), the same idea can also be used to increase the speed of computation of matching scores on each level of resolution for the multiresolution approach that will be seen on chapter 9; although this time we will only compute and keep the scores for top, bottom, left, right and center disparity variations to later estimate the appropriate parabolas.

8.- Feature extraction and matching

8.1.- Introduction

Whenever sparse structure of a scene is needed, or camera position, orientation and internal parameters are needed (i.e. calibration), usually only a handful of point correspondences is needed, and consequently there is no need to compute dense disparity without any prior knowledge. This fact will allow simplification of the problem of dense disparity estimation as will be seen on chapter 9.

We have also seen on chapter 7 that most matching measures only can provide reliable results at certain areas where there is enough texture.

This chapter will introduce the reader into techniques to obtain most coherent areas so that enough reliable point correspondences can be obtained in order to compute a sparse 3D texture or self-calibrate all cameras for the different views.

Most of the techniques to obtain such coherent areas are based either on robust image gradient estimation or the usage of the structural tensor, both of which will be explained on this chapter as well.

8.2.- Gradient estimation of an image

As we will see on section 8.3 and 8.4 most corner detectors and the structural tensor itself rely on the accurate estimation of gradient information from an image. This section will focus on different techniques to estimate as reliably as possible gradient data from images.

Starting from the classic 1D definition of differentiation for a continuous function:

$$\frac{\partial}{\partial x} I(x) \equiv \lim_{\delta \rightarrow 0} \frac{I(x + \delta) - I(x)}{\delta} \quad (8.1)$$

Assuming that an image $I[n]$ is the discrete sampling of a differentiable function $I(x)$, we could drop the limit by assuming a small step Δx ,

$$\frac{\partial}{\partial x} I(x) \approx \frac{I(x + \Delta x) - I(x)}{\Delta x} \quad (8.2)$$

which for the case of an image is equal to 1 pixel ($\Delta x = 1$). Then, expression (8.2) transforms into

$$\frac{\partial}{\partial x} I(x) \approx I[n + 1] - I[n] \quad (8.3)$$

which leads to a not symmetrical differentiation approximation.

One could then naively interpret that gradient information can be retrieved from an image just by computing the difference between consecutive pixels, which would be equivalent to using a filter $[-1, +1]$.

To slightly improve the results, one could also define a symmetric differentiation definition starting from:

$$\frac{\partial}{\partial x} I(x) \equiv \lim_{\delta \rightarrow 0} \frac{I(x + \delta) - I(x - \delta)}{2\delta} \quad (8.4)$$

which could be approximated as

$$\frac{\partial}{\partial x} I(x) \approx \frac{I[n+1] - I[n-1]}{2} \quad (8.5)$$

and is equivalent to using a filter $1/2[-1, 0, +1]$

However, even though the expressions above might give apparently good results, there are more robust techniques to compute discrete differentiation as will be shown in the following sub-sections.

8.2.1.- Simoncelli filters

The idea proposed by Simoncelli starts by differentiating continuous signals that have been interpolated from their initial discrete versions. Hence, assuming that no aliasing is present on input signals according to the sampling theorem [1] we can define any continuous signal as:

$$I(x) = \sum_{n=-\infty}^{+\infty} I[n] \cdot A(x - nT) \quad (8.6)$$

where n stands for the sample number, T for the sampling period, x as the continuous representation of n which is nT and finally $A(x)$ which is a continuous reconstructing function that allows the retrieval of an interpolated continuous signal from its discrete version.

We can now differentiate expression (8.6) as follows:

$$\begin{aligned} \frac{\partial}{\partial x} I(x) &= \frac{\partial}{\partial x} \left[\sum_{n=-\infty}^{+\infty} I[n] \cdot A(x - nT) \right] = \sum_{n=-\infty}^{+\infty} I[n] \cdot \frac{\partial}{\partial x} [A(x - nT)] \\ &= \sum_{n=-\infty}^{+\infty} I[n] \cdot D_A(x - nT) \end{aligned} \quad (8.7)$$

Finally, sampling again the continuous differentiation on (8.7) using the same sampling period T as in the original sampled signal $I[n]$ we obtain:

$$\begin{aligned} \frac{\partial}{\partial x} I[k] &= \sum_{n=-\infty}^{+\infty} I[n] \cdot D_A(x - nT) \Big|_{x=kT} = \sum_{n=-\infty}^{+\infty} I[n] \cdot D_A[(k - n)T] \\ &= \sum_{n=-\infty}^{+\infty} I[n] \cdot d_A[k - n] = I[k] * d_A[k] \end{aligned} \quad (8.8)$$

where $d_A[k]$ is a sampled derivative D_A of the approximation function $A(x)$. In other words, to compute the derivative of I , we just need to convolve it with a derivative mask.

Expression (8.7) and (8.8) can also be extended for the 2D case, which is of particular interest for image analysis. In such case, we obtain the following expressions for the horizontal and vertical derivatives of an image:

$$\begin{aligned}
 \frac{\partial}{\partial x} I(x, y) &= \frac{\partial}{\partial x} \left[\sum_{n=-\infty}^{+\infty} \sum_{m=-\infty}^{+\infty} I[n, m] \cdot A(x - nT, y - mT) \right] \\
 &= \sum_{n=-\infty}^{+\infty} \sum_{m=-\infty}^{+\infty} I[n, m] \cdot \frac{\partial}{\partial x} [A(x - nT, y - mT)] \\
 &= \sum_{n=-\infty}^{+\infty} \sum_{m=-\infty}^{+\infty} I[n, m] \cdot D_{Ax}(x - nT, y - mT)
 \end{aligned} \tag{8.9}$$

Again, by sampling we obtain:

$$\begin{aligned}
 \frac{\partial}{\partial x} I[i, j] &= \sum_{n=-\infty}^{+\infty} \sum_{m=-\infty}^{+\infty} I[n, m] \cdot D_{Ax}(x - nT, y - mT) \Big|_{x=iT, y=jT} \\
 &= \sum_{n=-\infty}^{+\infty} \sum_{m=-\infty}^{+\infty} I[n, m] \cdot d_{Ax}[i - n, j - m] = I[i, j] * d_{Ax}[i, j]
 \end{aligned} \tag{8.10}$$

As we can see, for the 2D case, we can also compute the derivative of a sampled image $I[i, j]$ by convolving it with a derivative mask $d_{Ax}[i, j]$.

Analogously, the vertical derivative of a sampled image would be:

$$\frac{\partial}{\partial y} I[i, j] = I[i, j] * d_{Ay}[i, j] \tag{8.11}$$

where d_{Ax} and d_{Ay} correspond to the horizontal and vertical sampled derivatives of the approximation function $A(x, y)$, just as follows:

$$d_{Ax}[i, j] = D_{Ax}(x, y) \Big|_{x=iT, y=jT} = \frac{\partial}{\partial x} [A(x, y)] \Big|_{x=iT, y=jT} \tag{8.12}$$

$$d_{Ay}[i, j] = D_{Ay}(x, y) \Big|_{x=iT, y=jT} = \frac{\partial}{\partial y} [A(x, y)] \Big|_{x=iT, y=jT} \tag{8.13}$$

However, it still remains to define the approximation function $A(x, y)$ to obtain the sampled derivative of an image. Farid and Simoncelli [5] proposed that the interpolation function A followed the restrictions shown below:

- A should be separable, which allows faster convolution with the derivative mask.
- A should be symmetric at the origin, which avoids spatial shifts on the estimated derivatives.

These two restrictions implies for the 2D case that A must be expressed as follows:

$$A(x, y) = p(x) \cdot p(y) \tag{8.14}$$

where $p(x)$ and $p(y)$ are the same continuous signal expressed horizontally and vertically.

Hence the horizontal and vertical derivative of $A(x,y)$ are:

$$\frac{\partial}{\partial x} A(x,y) = \frac{\partial}{\partial x} p(x) \cdot p(y) = d_1(x) \cdot p(y) \quad (8.15)$$

$$\frac{\partial}{\partial y} A(x,y) = \frac{\partial}{\partial y} p(x) \cdot p(y) = p(x) \cdot d_1(y) \quad (8.16)$$

Hence the horizontal and vertical **sampled** derivatives in (8.12) and (8.13) become:

$$d_{Ax}[i,j] = d_1(x) \cdot p(y) \Big|_{x=iT, y=jT} = d_1[i] \cdot p[j] = \mathbf{d}_1^T \cdot \mathbf{p} \quad (8.17)$$

$$d_{Ay}[i,j] = p(x) \cdot d_1(y) \Big|_{x=iT, y=jT} = p[i] \cdot d_1[j] = \mathbf{p}^T \cdot \mathbf{d}_1 \quad (8.18)$$

We can clearly observe that $\mathbf{d}_{Ax}$ is the transpose of $\mathbf{d}_{Ay}$, and that both are expressed by the separable column vectors $\mathbf{p}$ and $\mathbf{d}_1$, where $\mathbf{d}_1$ is the sampled derivative of $\mathbf{p}$. Hence, the whole convolution process can be done in two steps by convoluting first with $\mathbf{d}_1$ and then with $\mathbf{p}$ to obtain the horizontal derivative, or conversely first with $\mathbf{p}$ and then with $\mathbf{d}_1$ to obtain the vertical one.

It only remains to determine which vectors $\mathbf{p}$ and $\mathbf{d}_1$ to use.

Simoncelli proposed a series of numerically computed vectors $\mathbf{p}$ and $\mathbf{d}_1$ chosen so that they fulfill a common property of gradients:

$$\frac{\partial}{\partial \mathbf{v}} I(x,y) = v_x \frac{\partial}{\partial x} I(x,y) + v_y \frac{\partial}{\partial y} I(x,y) \quad (8.19)$$

Hence, by using (8.19), Simoncelli defined an error functional $E\{p, d_1\}$ in order to search for best solutions, which are summarized on table 8.1.

Simoncelli-Farid filters of order 1,2					
p	0.229879	0.540242	0.229879		
d1	0.425287	0	-0.425287		
Simoncelli-Farid filters of order 1,3					
p	0.037659	0.249153	0.426375	0.249153	0.037659
d1	0.109604	0.276691	0	-0.276691	-0.109604

Table 8.1. Symmetrical differentiating Simoncelli-Farid filters

8.2.2.- Savitzky-Golay filters

The idea behind Savitzky-Golay filters is to fit a polynomial of a certain degree around each sample point using a certain window W around it in order to fit a polynomial. This polynomial is expressed as follows

$$\tilde{x}(i) = \sum_{k=0}^N f_k \cdot i^k, \quad \text{for } -r \leq i \leq r \quad (8.20)$$

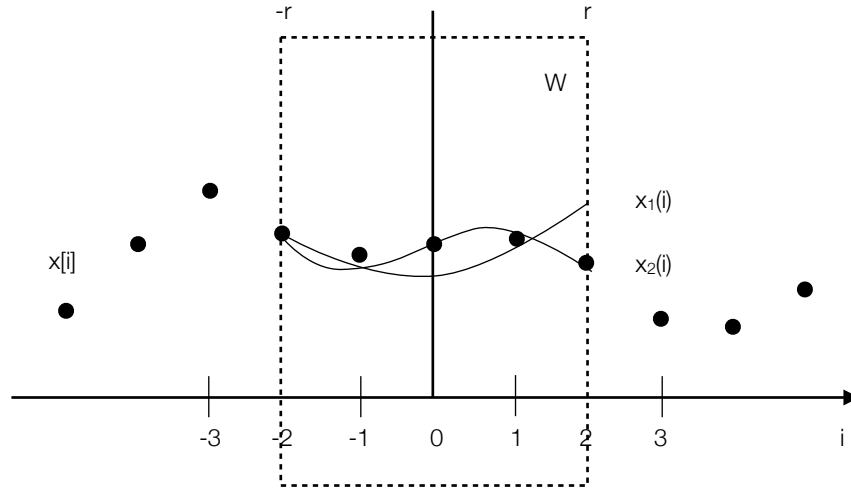


Fig. 8.1. 2nd and 4th degree polynomial fitting for the same set of data

where $\tilde{x}(i)$ is the estimated data sample after fitting the polynomial, f_k are the polynomial coefficients up to degree N , and i are the locations where the polynomials are evaluated within a range $[-r, r]$. Notice that locations i will be relative to data samples (i.e. for $i = 0$ we are evaluating over the current data sample). Hence, since we will take range $[-r, r]$ within window W , we will pick data samples within such range to estimate the polynomial. This is depicted on figure 8.1 for the case of polynomials of degree 2 and 4, which are the most typically used.

As it can be seen on figure 8.1, estimated polynomial will depend on chosen degree as well as on window size for the same set of data. Depending on the chosen parameters, different estimations might be obtained.

From expression (8.20), we can also see that a linear system of equations can be obtained if enough data points $x[i]$ are available. However, availability of those points will be determined by window size. It can be then inferred that a minimum window size exists in order to estimate a polynomial of degree N , which will determine values $-r$ and r as follows:

$$r \geq \left\lceil \frac{N+1}{2} \right\rceil \quad (8.21)$$

In summary, in order to fit a polynomial of degree $N = 4$, r would be 2, and expression (8.20) would transform into:

$$\begin{cases} x[-2] \approx \tilde{x}(-2) = f_0 + f_1 \cdot (-2) + f_2 \cdot (-2)^2 + f_3 \cdot (-2)^3 + f_4 \cdot (-2)^4 \\ \dots \\ x[2] \approx \tilde{x}(2) = f_0 + f_1 \cdot (2) + f_2 \cdot (2)^2 + f_3 \cdot (2)^3 + f_4 \cdot (2)^4 \end{cases} \quad (8.22)$$

It can be observed from (8.22) that the linear system of equations is linear on the unknowns f_k , which are the polynomial coefficients, and can also be expressed in matrix form as:

$$\mathbf{P} \cdot \mathbf{f} = \mathbf{X} \quad (8.23)$$

where $\mathbf{f}$ is a column vector of the coefficients f_k , $\mathbf{X}$ is a column matrix containing data samples to be fitted in a polynomial $x[k]$, $k \in W$, and $\mathbf{P}$ is a matrix of powers of indices, as follows:

$$P_{ij} = i^j, \text{ for } 0 \leq j \leq N \quad (8.24)$$

It must be noticed that P is completely independent of data (i.e. only depends on the polynomial degree, or conversely on the $[-r, r]$ range of i samples. This fact will be useful when attempting to fit several polynomials for different points in an image as P will only need to be inverted once in order to solve the multiple linear systems of equations.

Hence, if exactly $N + 1$ equations are available, then P is an $N \times N$ square matrix and the system of equations can be solved. However, if more equations are provided, we can also find a least squares error solution by means of the Moore-Penrose pseudo-inverse, just as follows:

$$\mathbf{f} = (\mathbf{P}^T \mathbf{P})^{-1} \mathbf{P}^T \mathbf{X} \quad (8.25)$$

where $\mathbf{P}^+ = (\mathbf{P}^T \mathbf{P})^{-1} \mathbf{P}^T$ is the pseudo-inverse and the f vector contains the fitted polynomial coefficients for vector data X .

Notice that expression (8.25) is also valid for exactly $N + 1$ equations, as the pseudo-inverse becomes the inverse of P for this case.

Once we have fitted a polynomial for each point, we can now easily use the estimated coefficients to either find a smoothed version of data as:

$$\tilde{x}(0) = f_0 \quad (8.26)$$

or find the polynomial derivative evaluated at such point as:

$$\tilde{x}'(0) = \left. \frac{\partial}{\partial i} x(i) \right|_{i=0} = f_1 \quad (8.27)$$

Notice that further derivatives can be estimated as well as:

$$\left. \frac{\partial^k}{\partial i^k} x(i) \right|_{i=0} = k! \cdot f_k, \quad \text{for } 0 \leq k \leq N \quad (8.28)$$

where k indicates the order of derivative. As it can be observed from (8.28), since we only have a vector $\mathbf{f}$ for k ranging between 0 and N , we will only be able to estimate the first N derivatives for a fitted polynomial of degree N .

It must also be noticed that since $\mathbf{P}$ is common for all data samples, it must only be inverted once in order to solve vector $\mathbf{f}$ several times for different points. Besides, if only the first derivative is needed as in (8.27), we can obtain such result as the inner product of the corresponding row of the inverse of $\mathbf{P}$ (in this case it would be the second row which corresponds to f_1) with data samples $\mathbf{X}$.

Up to this point we have seen how to obtain fitted polynomials for 1D data, however we are interested in processing images to obtain their gradient information, so we're going to extend the procedure mentioned above to the 2D case for an intensity image.

For the 2D case, the polynomial of order N in (8.20) becomes:

$$\tilde{x}(i, j) = \sum_{k_1, k_2=0}^N f_{k_1 k_2} \cdot i^{k_1} \cdot j^{k_2}, \quad \text{for } k_1 + k_2 \leq N \quad \text{and} \quad -r \leq i, j \leq r \quad (8.29)$$

As before, the parameters $f_{k_1 k_2}$ can be obtained by solving a similar linear system of equations where P has now a different structure. Hence, for the the case of a polynomial of order 4, expression (8.29) would be:

$$\begin{aligned}
\tilde{x}(i, j) = & f_{00} + f_{01}j + f_{02}j^2 + f_{03}j^3 + f_{04}j^4 + \\
& f_{10}i + f_{11}ij + f_{12}ij^2 + f_{13}ij^3 + \\
& f_{20}i^2 + f_{21}i^2j + f_{22}i^2j^2 + \\
& f_{30}i^3 + f_{31}i^3j + \\
& f_{40}i^4
\end{aligned} \tag{8.30}$$

and each k-th row P_k of matrix $\mathbf{P}$ would become now:

$$P_k = \begin{bmatrix} 1 & j & j^2 & j^3 & j^4 & i & ij & ij^2 & ij^3 & i^2 & i^2j & i^2j^2 & i^3 & i^3j & i^4 \end{bmatrix} \tag{8.31}$$

In order to solve the polynomial coefficients, it only remains to obtain the value of r that delimits window size.

For the 2D case, the minimum number of data samples (i.e. equations) to fit a polynomial of order N will be:

$$S = \frac{1}{2}(N+1)(N+2) \tag{8.32}$$

Assuming we use a square window W of size $2r \times 2r$ and using (8.32), then r becomes:

$$r \geq \left\lceil \frac{\sqrt{S}}{2} \right\rceil \tag{8.33}$$

Once P and X are known using respectively (8.31) and at least the minimum image samples within W , the coefficients f can be obtained using expression (8.25) as well. Again if more points than the required minimum are provided, the pseudo-inverse will provide a least squared error solution.

Once the polynomial coefficients are found, a smoothed 2D signal can be obtained as:

$$\tilde{x}(i, j)|_{(0,0)} = f_{00} \tag{8.34}$$

And the directional (horizontal and vertical) derivatives can be obtained as:

$$\frac{\partial}{\partial i} \tilde{x}(i, j) \Big|_{(0,0)} = f_{10} \text{ and } \frac{\partial}{\partial j} \tilde{x}(i, j) \Big|_{(0,0)} = f_{01} \tag{8.35}$$

which correspond to $\mathbf{f}(6)$ and $\mathbf{f}(2)$ respectively if using $\mathbf{P}$ in the order shown in (8.31). In other words, once the inverse of $\mathbf{P}$ is computed, we can obtain the directional derivatives of each pixel of an intensity image just by computing the inner product of the 6th and 2nd rows of the inverse of $\mathbf{P}$ by the input data $\mathbf{X}$.

8.3.- The structural tensor

It has been proved that human vision is not only sensitive to changes in luminance, but also to changes in scale of texture and its orientation as well [5]. The structural tensor is a tool aimed at detecting those areas within an image which can be more easily processed by human vision.

The structural tensor is a structure containing data related to coherence and orientation of features appearing inside an image on each pixel. It can be seen as an image having different color channels, but instead of containing intensity of each color, contains other data such as gradient information, orientation or coherence.

The structural tensor is computed from the gradient information on a grayscale or intensity image. Gradient data can be obtained by means of integer differential estimation methods seen before, however usually a simple technique such as Simoncelli filters (section 8.2.1) is enough. Gradient of an intensity image is represented as follows:

$$\nabla I = \nabla I(x, y) = \left(\frac{\partial}{\partial x} I(x, y), \frac{\partial}{\partial y} I(x, y) \right) \quad (8.36)$$

Once Gradient information of an intensity image has been computed, it can be used for orientation retrieval as will be shown next. Besides orientation, also a measure of coherence of the results will be given as well.

8.3.1.- Computation of Orientation

If we look closely at the gradient measures (which contain horizontal and vertical differentiations) we can see that their mean value within a region around any given pixel is close to zero. This is true because differentiation methods behave as high-pass filters that remove the DC component.

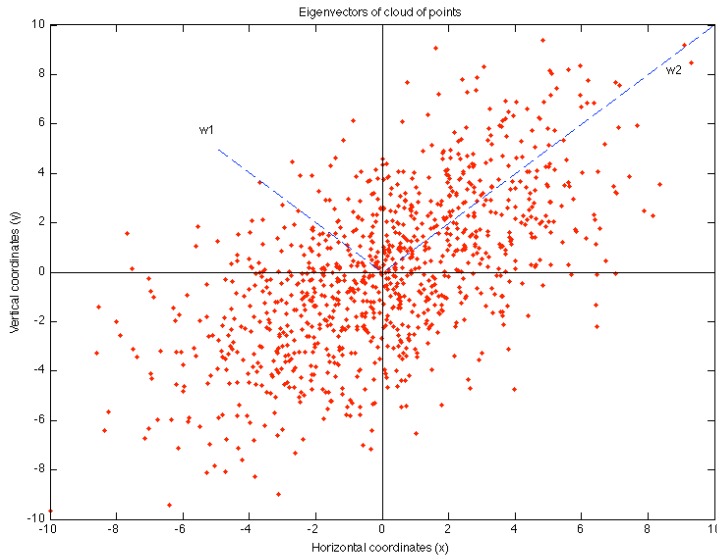


Fig. 8.2. Eigenvectors of scattered cloud of points. The eigenvector $\mathbf{w}_2$ corresponding to the largest eigenvalue lays in the most significative orientation.

We know that gradient vectors give a measure of orientation at any given point as the vectors point towards any local maxima in an image, and hence they also point perpendicular to any intensity variation.

Our intention is to robustly retrieve orientation at any pixel location, specially in areas where orientation is quite random due to lack of texture. We could naively try to compute gradient mean around a neighborhood of any pixel, however, as we have seen, the mean would be close to zero, producing an invalid estimation of orientation. Instead, to robustly estimate orientation, we will retrieve the most significant orientation.

From a statistical point of view, we can retrieve orientation as follows. Assuming Gaussianity of gradient measures, we could plot gradient vectors within a neighborhood around each pixel forming a scattered cloud of points. Because we have assumed Gaussianity, we also know the cloud of points will have an ellipsoid shape. It can easily be observed that the most significative orientation will be the one where the major axis of the ellipsoid is contained. This is depicted in fig. 8.2.

Starting from the covariance matrix of gradient measures within a region U around any given pixel (assuming zero-mean):

$$\Sigma_{\nabla I} = \frac{1}{N} \sum_{\mathbf{x} \in U} (\nabla I - \overline{\nabla I}) \cdot (\nabla I - \overline{\nabla I})^T \quad (8.37)$$

where N is the number of pixels $\mathbf{x}$ on image I within region U , ∇I stands for gradient image at pixel $\mathbf{x}$, and $\overline{\nabla I}$ stands for the mean gradient of image I at pixel $\mathbf{x}$ within region U , which should be zero as we have assumed.

$$\overline{\nabla I} = \frac{1}{N} \sum_{\mathbf{x} \in U} \nabla I = 0 \quad (8.38)$$

Hence, expression (8.37) becomes (up to scale) the following expression, which is known as the structural tensor for a single pixel $\mathbf{x}$.

$$\mathbf{T} = \sum_{\mathbf{x} \in U} \nabla I \cdot \nabla I^T = \begin{pmatrix} \sum_{\mathbf{x} \in U} \left(\frac{\partial}{\partial x} I \right)^2 & \sum_{\mathbf{x} \in U} \left(\frac{\partial}{\partial x} I \cdot \frac{\partial}{\partial y} I \right) \\ \sum_{\mathbf{x} \in U} \left(\frac{\partial}{\partial x} I \cdot \frac{\partial}{\partial y} I \right) & \sum_{\mathbf{x} \in U} \left(\frac{\partial}{\partial y} I \right)^2 \end{pmatrix} \quad (8.39)$$

where we have removed the scale factor on $\mathbf{T}$ as it will be of no interest for major axis computation. The structural tensor is the extension of matrix $\mathbf{T}$ for all pixels within an image.

We can observe that by computing the eigen-decomposition of matrix $\mathbf{T}$ for a given pixel $\mathbf{x}$ on image I we can obtain the following expression:

$$\mathbf{T} = \mathbf{Q} \cdot \mathbf{D} \cdot \mathbf{Q}^T \quad (8.40)$$

where $\mathbf{Q}$ is an orthonormal matrix containing the eigenvectors of $\mathbf{T}$ on each column and $\mathbf{D}$ is a diagonal matrix containing the eigenvalues corresponding to each eigenvector.

For the 2D case, which is of particular interest for intensity image analysis, $\mathbf{Q}$ and $\mathbf{D}$ are 2x2 matrices, where the eigenvectors of $\mathbf{Q}$ contain the minor and major axis of the scattered cloud of points seen on fig. 8.2, and the eigenvalues of $\mathbf{D}$ indicate whether each eigenvector corresponds to the minor or major axis. In particular, we will always be interested in the eigenvector corresponding to the largest eigenvalue, as it will correspond to the major axis of the cloud of points.

The reasoning behind the explanation shown above for retrieval of **most significant** orientation can also be demonstrated analytically as follows.

Proof 8.1

Being U the local neighborhood around point $\mathbf{x}_0$, local directional image data (i.e. image gradient data) located within neighborhood $\mathbf{x} \in U$ can be associated to vector $\mathbf{q}(\mathbf{x})$. We are looking for vector $\mathbf{w}(\mathbf{x}_0)$ that is as close as possible to all gradient values $\mathbf{q}(\mathbf{x})$ within neighborhood U . For that purpose, first we define the metric that will compare how similar two vectors are by using their inner product.

$$\rho(\mathbf{x}) = \mathbf{q}(\mathbf{x})^T \cdot \mathbf{w}(\mathbf{x}_0) \quad (8.41)$$

Since we are interested in orientation measures which are invariant up to 180°, $\mathbf{q}$ and $-\mathbf{q}$ become equivalent, and we can use the square of expression (8.41) to find the best vector $\mathbf{w}(\mathbf{x}_0)$ that contains the most significant orientation by defining the following functional to be maximized.

$$Q = \int_{U(x_0)} \rho(\mathbf{x})^2 \cdot d\mathbf{x} \quad (8.42)$$

Expression (8.42) can be expanded as follows:

$$\begin{aligned} Q &= \int_{U(x_0)} \rho(\mathbf{x})^2 \cdot d\mathbf{x} = \int_{U(x_0)} \left(\mathbf{q}(\mathbf{x})^T \cdot \mathbf{w}(\mathbf{x}_0) \right)^2 \cdot d\mathbf{x} \\ &= \int_{U(x_0)} \left(\mathbf{q}(\mathbf{x})^T \cdot \mathbf{w}(\mathbf{x}_0) \right) \left(\mathbf{q}(\mathbf{x})^T \cdot \mathbf{w}(\mathbf{x}_0) \right) \cdot d\mathbf{x} \\ &= \int_{U(x_0)} \left(\mathbf{w}(\mathbf{x}_0)^T \cdot \mathbf{q}(\mathbf{x}) \right) \left(\mathbf{q}(\mathbf{x})^T \cdot \mathbf{w}(\mathbf{x}_0) \right) \cdot d\mathbf{x} \\ &= \mathbf{w}(\mathbf{x}_0)^T \cdot \left[\int_{U(x_0)} \mathbf{q}(\mathbf{x}) \cdot \mathbf{q}(\mathbf{x})^T \cdot d\mathbf{x} \right] \cdot \mathbf{w}(\mathbf{x}_0) \\ &= \mathbf{w}(\mathbf{x}_0)^T \cdot \mathbf{T}(\mathbf{x}_0) \cdot \mathbf{w}(\mathbf{x}_0) \end{aligned} \quad (8.43)$$

where it can be seen that $\mathbf{T}(\mathbf{x}_0)$ is equivalent to the same $\mathbf{T}$ at point $\mathbf{x}_0$ seen on expression (8.39).

Our purpose is to find the most significative orientation $\mathbf{w}$ that maximizes Q:

$$\max_{\mathbf{w}}(Q) = \max_{\mathbf{w}}(\mathbf{w}^T \mathbf{T} \mathbf{w}) \quad (8.44)$$

Expression (8.44) can be maximized by using constrained optimization (using the theorem of Lagrange multipliers). This theorem states that a function $a(\mathbf{w})$ can reach its maximum given a certain constrain $b(\mathbf{w}) = 0$ and using the following functional:

$$L(\mathbf{w}) = a(\mathbf{w}) - \lambda \cdot b(\mathbf{w}) \quad (8.45)$$

where λ is a Lagrange multiplier and $a(\mathbf{w})$ is the function to maximize, hence: $a(\mathbf{w}) = \mathbf{w}^T \mathbf{T} \mathbf{w}$

In order to find a maximum or minimum of a function, a sufficient condition is that its partial derivatives become zero:

$$\frac{\partial L(\mathbf{w}, \lambda)}{\partial \mathbf{w}} = 0 \quad \text{and} \quad \frac{\partial L(\mathbf{w}, \lambda)}{\partial \lambda} = 0 \quad (8.46)$$

It only remains to set the initial condition $b(\mathbf{w})$, which could be set initially to:

$$\mathbf{w}^T \mathbf{w} = c \quad (8.47)$$

where $c = \|\mathbf{w}\|^2$ is a constant, as it is the initial condition. Because we also need $b(\mathbf{w}) = 0$, it follows that:

$$b(\mathbf{w}) = \mathbf{w}^T \mathbf{w} - c = 0 \quad (8.48)$$

Then, expression (8.45) becomes:

$$L(\mathbf{w}) = \mathbf{w}^T \mathbf{T} \mathbf{w} - \lambda \cdot (\mathbf{w}^T \mathbf{w} - c) \quad (8.49)$$

And using the partial derivatives, we obtain:

$$\frac{\partial L(\mathbf{w}, \lambda)}{\partial \lambda} = \mathbf{w}^T \mathbf{w} - c = b(\mathbf{w}) = 0 \quad (8.50)$$

which we know is zero from (8.48), and

$$\begin{aligned} \frac{\partial L(\mathbf{w}, \lambda)}{\partial \mathbf{w}} &= \frac{\partial}{\partial \mathbf{w}} (\mathbf{w}^T \mathbf{T} \mathbf{w}) - \lambda \frac{\partial}{\partial \mathbf{w}} (\mathbf{w}^T \mathbf{w}) \\ &= [\mathbf{T} + \mathbf{T}^T] \mathbf{w} - 2\lambda \mathbf{w} = 2\mathbf{T} \mathbf{w} - 2\lambda \mathbf{w} = 0 \end{aligned} \quad (8.51)$$

which can also be expressed as:

$$\mathbf{T} \mathbf{w} = \lambda \mathbf{w} \quad (8.52)$$

we know that this last expression (8.52) holds if $\mathbf{w}$ is an eigenvector and λ is its corresponding eigenvalue.

Up to this point, partial derivatives only have ensured that we have found the maximum or minimum of Q . We need to ensure that Q is maximized by substituting expression (8.52) into (8.44) and using result from (8.47).

$$Q_{\max} = \mathbf{w}^T \mathbf{T} \mathbf{w} = \mathbf{w}^T \lambda \mathbf{w} = \lambda \mathbf{w}^T \mathbf{w} = \lambda c \quad (8.53)$$

Since we know that c is constant, then the only way to maximize Q is to choose the largest eigenvalue λ , and consequently the most significant orientation (i.e. gradient vector) $\mathbf{w}$ that maximizes Q , is the eigenvector of $\mathbf{T}$ corresponding to the largest eigenvalue λ .

Up to this point we have seen a general overview of how the structural tensor works, and although it can also be computed over color or multichannel images, its application over intensity images is equally good, and as we will see next, direct expressions for eigen-decomposition and other features can be obtained in a very simple way since the $\mathbf{T}$ structure will always be a 2×2 matrix for the purpose of image processing.

8.3.2.- The structural tensor on intensity images

Intensity images can be seen as 2D matrices of data, and hence $\mathbf{T}$ can be expressed as in (8.39)

$$\mathbf{T} = \begin{pmatrix} \sum_{\mathbf{x} \in U} \left(\frac{\partial}{\partial x} I \right)^2 & \sum_{\mathbf{x} \in U} \left(\frac{\partial}{\partial x} I \cdot \frac{\partial}{\partial y} I \right) \\ \sum_{\mathbf{x} \in U} \left(\frac{\partial}{\partial x} I \cdot \frac{\partial}{\partial y} I \right) & \sum_{\mathbf{x} \in U} \left(\frac{\partial}{\partial y} I \right)^2 \end{pmatrix} = \begin{pmatrix} T_{11} & T_{12} \\ T_{21} & T_{22} \end{pmatrix} \quad (8.54)$$

where T_{12} and T_{21} are equal. Hence, matrix $\mathbf{T}$ is symmetric and real, which ensures that it is positive-definite (i.e. all its eigenvalues are greater or equal than zero).

For a 2x2 case as shown in (8.54), we can directly solve (8.52) as follows:

$$\mathbf{T}\mathbf{w} = \lambda\mathbf{w} \Rightarrow (\mathbf{T} - \lambda\mathbf{I})\mathbf{w} = \mathbf{0} \quad (8.55)$$

Because we want a solution $\mathbf{w}$ different from the obvious solution $\mathbf{w} = \mathbf{0}$ in order to solve the homogeneous linear system of equations, we need to force

$$\det(\mathbf{T} - \lambda\mathbf{I}) = 0 \quad (8.56)$$

Expanding (8.56) we obtain

$$\begin{aligned} \varphi(\lambda) = \det(\mathbf{T} - \lambda\mathbf{I}) &= \begin{vmatrix} T_{11} - \lambda & T_{12} \\ T_{12} & T_{22} - \lambda \end{vmatrix} \\ &= \lambda^2 - \lambda(T_{11} + T_{22}) + (T_{11}T_{22} - T_{12}^2) = 0 \end{aligned} \quad (8.57)$$

And because for the 2D case expression (8.57) is just a quadratic expression over λ , we can analytically solve it as follows:

$$\begin{aligned} \lambda_1 &= \frac{1}{2} \left[(T_{11} + T_{22}) + \sqrt{(T_{11} - T_{22})^2 + 4T_{12}^2} \right], \\ \lambda_2 &= \frac{1}{2} \left[(T_{11} + T_{22}) - \sqrt{(T_{11} - T_{22})^2 + 4T_{12}^2} \right], \end{aligned} \quad (8.58)$$

From (8.56) it can be easily observed that:

$$\lambda_1 + \lambda_2 = T_{11} + T_{22} = \text{Tr}(\mathbf{T}) \quad (8.59)$$

$$\lambda_1 - \lambda_2 = \sqrt{(T_{11} - T_{22})^2 + 4T_{12}^2} \quad (8.60)$$

$$\lambda_1 \cdot \lambda_2 = \det(\mathbf{T}) = T_{11}T_{22} - T_{12}^2 \quad (8.61)$$

where $\text{Tr}(\mathbf{T})$ stands for the trace of $\mathbf{T}$, and $\det(\mathbf{T})$ for its determinant. From (8.60) we can also see that $\lambda_1 \geq \lambda_2$. Taking this fact into account, let's now review the following special cases:

1. Equal eigenvalues:

$$\lambda_1 = \lambda_2 \Leftrightarrow (T_{11} = T_{22} \wedge T_{12} = T_{21} = 0) \quad (8.62)$$

If in addition the two are zero, then

$$\lambda_1 = \lambda_2 = 0 \Leftrightarrow T_{11} = T_{22} = 0 \quad (8.63)$$

2. Different eigenvalues with the smallest one equal to zero:

$$\lambda_2 = 0 \Leftrightarrow \{\det(\mathbf{T}) = 0 \wedge \lambda_1 = \text{Tr}(\mathbf{T})\} \quad (8.64)$$

Besides, different relative values of eigenvalues also have a meaning on image texture, as it is summarized on table 8.2

Rank of $\mathbf{T}$	Eigenvalues	Type of local structure
0	$\lambda_1 = \lambda_2 = 0$	Constant intensity in an image - no signal change.
1	$\lambda_1 > \lambda_2 = 0$	Intensity signal does not change in the direction associated with the smallest eigenvalue λ_2 . However, there is a change in the direction associated with λ_1 . In this case the eigenvector associated to λ_1 points towards the direction of maximal signal change in the neighborhood of pixels U. If $\lambda_1 \gg \lambda_2 \approx 0$ then it can indicate lines in images.
2	$\lambda_1 > \lambda_2 > 0$	The two eigenvalues are greater than zero, this means that there are changes in all directions. If one of the eigenvalues is dominating, then there is a dominating direction of signal changes. If $\lambda_1 \geq \lambda_2 \gg 0$ then it can indicate corners in images.
2	$\lambda_1 = \lambda_2 > 0$	This is a special case of the above. Signal changes are equal in all directions.

Table 8.2. Meaning of the eigenvalues of $\mathbf{T}$.

Up to this point we have seen how to compute the eigenvalues of $\mathbf{T}$ for the 2D case (i.e. intensity images). It now remains to obtain their respective eigenvectors.

The eigenvectors can be obtained by substituting the expressions of the eigenvalues (8.58) into homogeneous equation (8.55), which for the largest eigenvalue λ_1 is:

$$(\mathbf{T} - \lambda_1 \mathbf{I}) \mathbf{w} = 0 \quad (8.65)$$

$$\begin{pmatrix} T_{11} - \lambda_1 & T_{12} \\ T_{21} & T_{22} - \lambda_1 \end{pmatrix} \mathbf{w} = 0 \quad (8.66)$$

$$\begin{pmatrix} \frac{1}{2} \left[(T_{11} - T_{22}) - \sqrt{(T_{11} - T_{22})^2 + 4T_{12}^2} \right] & T_{12} \\ T_{21} & \frac{1}{2} \left[(-T_{11} + T_{22}) - \sqrt{(T_{11} - T_{22})^2 + 4T_{12}^2} \right] \end{pmatrix} \mathbf{w} = 0 \quad (8.67)$$

Since we know that both equations are linearly dependent, we only need to arbitrarily choose one of them to retrieve $\mathbf{w}$. In our case we chose the first equation, obtaining

$$\frac{1}{2} \left[(T_{11} - T_{22}) - \sqrt{(T_{11} - T_{22})^2 + 4T_{12}^2} \right] \cdot w_1 + T_{12} \cdot w_2 = 0 \quad (8.68)$$

And hence, the eigenvector for the eigenvalue λ_1 can be any linear combination of:

$$\mathbf{w} = \begin{bmatrix} -T_{12} \\ \frac{1}{2} \left[(T_{11} - T_{22}) - \sqrt{(T_{11} - T_{22})^2 + 4T_{12}^2} \right] \end{bmatrix} \quad (8.69)$$

Analogously, for the smallest eigenvalue λ_2 , its corresponding eigenvector can be found by substituting again (8.58) into (8.55) as follows:

$$(\mathbf{T} - \lambda_2 \mathbf{I}) \mathbf{w} = 0 \quad (8.70)$$

$$\begin{pmatrix} T_{11} - \lambda_2 & T_{12} \\ T_{21} & T_{22} - \lambda_2 \end{pmatrix} \mathbf{w} = 0 \quad (8.71)$$

$$\begin{pmatrix} \frac{1}{2} \left[(T_{11} - T_{22}) + \sqrt{(T_{11} - T_{22})^2 + 4T_{12}^2} \right] & T_{12} \\ T_{21} & \frac{1}{2} \left[(-T_{11} + T_{22}) + \sqrt{(T_{11} - T_{22})^2 + 4T_{12}^2} \right] \end{pmatrix} \mathbf{w} = 0 \quad (8.72)$$

Again, we know that both equations are linearly dependent, and we chose the first equation (the second one could also be chosen)

$$\frac{1}{2} \left[(T_{11} - T_{22}) + \sqrt{(T_{11} - T_{22})^2 + 4T_{12}^2} \right] \cdot w_1 + T_{12} \cdot w_2 = 0 \quad (8.73)$$

And hence, the eigenvector for the eigenvalue λ_2 can be any linear combination of:

$$\mathbf{w} = \begin{bmatrix} -T_{12} \\ \frac{1}{2} \left[(T_{11} - T_{22}) + \sqrt{(T_{11} - T_{22})^2 + 4T_{12}^2} \right] \end{bmatrix} \quad (8.74)$$

From now on, we will only be interested in the eigenvector from the largest eigenvalue in (8.69) as it contains the most significative orientation. However, in (8.42) we assumed 180° rotation invariance, which needs to be enforced, as any linear combination (i.e. sign change, etc.) of $\mathbf{w}$ would also be a valid eigenvector.

Assuming that ξ is the phase of vector $\mathbf{w}$, we would be interested on a vector whose phase rotates twice as fast. Using the formula of double tangent we obtain,

$$\tan(2\xi) = \frac{2 \tan(\xi)}{1 - \tan^2(\xi)}, \quad \tan(\xi) \neq 1 \quad (8.75)$$

And taking into account the definition of tangent,

$$\tan(\xi) = \frac{w_2}{w_1}, \quad w_1 \neq 0 \quad (8.76)$$

and after substituting (8.76) into (8.75)

$$\tan(2\xi) = \frac{2w_1w_2}{w_1^2 - w_2^2}, \quad w_1 \neq w_2 \quad (8.77)$$

And finally substituting (8.69) into (8.77)

$$\begin{aligned}
\tan(2\xi) &= \frac{-4T_{12} \left[(T_{11} - T_{22}) - \sqrt{(T_{11} - T_{22})^2 + 4T_{12}^2} \right]}{4T_{12}^2 - (T_{11} - T_{22})^2 + 2(T_{11} - T_{22})\sqrt{(T_{11} - T_{22})^2 + 4T_{12}^2} - (T_{11} - T_{22})^2 - 4T_{12}^2} \\
&= \frac{-4T_{12} \left[(T_{11} - T_{22}) - \sqrt{(T_{11} - T_{22})^2 + 4T_{12}^2} \right]}{-2(T_{11} - T_{22}) \left[(T_{11} - T_{22}) - \sqrt{(T_{11} - T_{22})^2 + 4T_{12}^2} \right]} = \frac{2T_{12}}{T_{11} - T_{22}}, \quad T_{11} \neq T_{22}
\end{aligned} \tag{8.78}$$

Since (8.78) contains the phase of the desired vector $\mathbf{w}$ containing the most significant orientation with 180° invariance, by using again the definition of tangent as in (8.76), we can obtain the components of $\mathbf{w}$ as:

$$\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} T_{11} - T_{22} \\ 2T_{12} \end{bmatrix} \tag{8.79}$$

8.3.2.1.- Augmented Structural Tensor and Coherence

The augmented structural tensor is a structure similar to (8.79) that allows us to distinguish between two important cases: $\lambda_1 = \lambda_2 = 0$ and $\lambda_1 = \lambda_2 > 0$. If we look at the trace of $\mathbf{T}$ in (8.59) we know that it is equal to the sum of eigenvalues. Hence, on the first case the trace will be zero, and on the second case it will be twice the value of the eigenvalues.

By using the trace of $\mathbf{T}$, the augmented structural tensor becomes:

$$\mathbf{w}^* = \begin{bmatrix} Tr(\mathbf{T}) \\ \mathbf{w} \end{bmatrix} = \begin{bmatrix} T_{11} + T_{22} \\ T_{11} - T_{22} \\ 2T_{12} \end{bmatrix} \tag{8.80}$$

From expression (8.80) we can easily recover T_{11} , T_{22} , and T_{12} which completely define the whole structure $\mathbf{T}$. Consequently, by using expressions (8.58), the eigenvalues, and also the eigenvectors in (8.69) and (8.74) become uniquely defined.

A measure of coherence can also be defined by using the eigenvalues of structure $\mathbf{T}$ taking into account their meaning seen on table 8.2. Coherence measure indicates whether a given orientation is mainly isotropic (i.e. orientation not well defined mainly because of lack of texture) or on the other hand it has a single linear direction (i.e. orientation is well defined in areas with enough texture). Coherence is defined as follows:

$$c = \begin{cases} \left(\frac{\lambda_1 - \lambda_2}{\lambda_1 + \lambda_2} \right)^2 = \frac{\|\mathbf{w}\|^2}{Tr(\mathbf{T})}, & Tr(\mathbf{T}) \neq 0 \\ 0, & Tr(\mathbf{T}) = 0 \end{cases} \tag{8.81}$$

Where a value of 0 indicates no coherence (i.e. orientation not well defined) and 1 indicates a mainly linear direction (i.e. orientation is well defined).

Sometimes, an equivalent representation of $\mathbf{w}^*$ is given using more meaningful parameters, which for some applications is more useful,

$$\mathbf{s} = \begin{bmatrix} Tr(\mathbf{T}) \\ \angle \mathbf{w} \\ c \end{bmatrix} \quad (8.82)$$

where $Tr(\mathbf{T})$ is the trace of $\mathbf{T}$, which from (8.19) can be seen as an averaged square modulus of local gradient as seen below.

$$\begin{aligned} \|\nabla I\|^2 &= \nabla_x^2 I + \nabla_y^2 I, \\ \int_U \|\nabla I\|^2 d\mathbf{x} &= \int_U \nabla_x^2 I d\mathbf{x} + \int_U \nabla_y^2 I d\mathbf{x} = T_{11} + T_{22} = Tr(\mathbf{T}) \end{aligned} \quad (8.83)$$

$\angle \mathbf{w}$ indicates the phase of the most significative orientation (i.e. the phase of the eigenvector corresponding to the largest eigenvalue) and c indicates the coherence measure seen on (8.81)

8.3.3.- Extending the structural tensor to multichannel images

The structural tensor cannot be directly computed over multichannel images as explained in section 8.3.2, instead different approaches can be taken into account before attempting the computation of the structural tensor.

The first and most logical approach consist on converting multichannel images (i.e. RGB, YUV, etc) into intensity images. Once images have a single channel of information, the structural tensor can be directly computed as in section 8.3.2.

The second approach consists on computing gradient information $\mathbf{q}(\mathbf{x})$ on each channel separately, and then compute the summation of such gradients on all M channels before computing $\mathbf{T}$, as shown below:

$$\begin{aligned} \mathbf{T}(\mathbf{x}_0) &= \int_{U(\mathbf{x}_0)} \sum_{k=1}^M (\mathbf{q}_k(\mathbf{x}) \mathbf{q}_k^T(\mathbf{x})) d\mathbf{x} \\ &= \sum_{k=1}^M \int_{U(\mathbf{x}_0)} \sum_{k=1}^M (\mathbf{q}_k(\mathbf{x}) \mathbf{q}_k^T(\mathbf{x})) d\mathbf{x} = \sum_{k=1}^M \mathbf{T}_k(\mathbf{x}_0) \end{aligned} \quad (8.84)$$

Also, an extension to (8.84) can be used by using a weighted summation of each component tensor

$$\mathbf{T}(\mathbf{x}_0) = \sum_{k=1}^M c_k \mathbf{T}_k(\mathbf{x}_0) \quad (8.85)$$

8.3.4.- Converting the structural tensor for visualization

Up to this point we have seen how to compute the structural tensor itself, however, the values obtained might be difficult to be used for direct visualization. In this section we explain a simple conversion that can be useful for visualization purposes.

On (8.82) we saw that the structural tensor can be expressed in a more meaningful way for many applications, one of the is for visualization purposes. The three components of vector $\mathbf{s}$ can be easily visualized in the HSI color space by using the following mapping:

$$\begin{aligned} \angle \mathbf{w} &\rightarrow H \\ c &\rightarrow S \\ Tr(\mathbf{T}) &\rightarrow I \end{aligned} \quad (8.86)$$

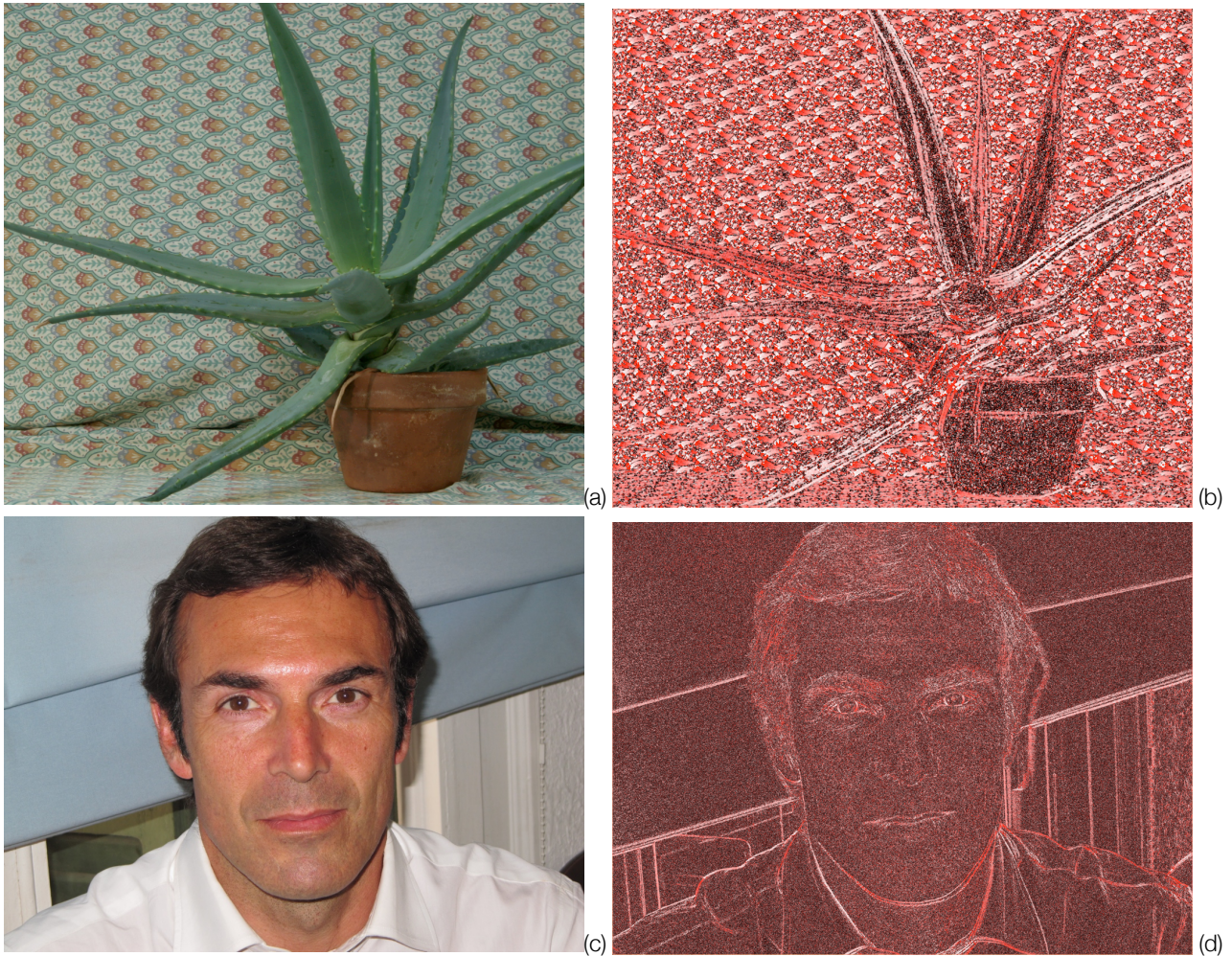


Fig. 8.3. Visualization of the structural tensor. Dark areas represent areas with low coherence (and hence low texture), oppositely, bright areas represent areas with high coherence. Changes in hue indicate changes in local orientation. (a) Original studio image [40]. (b) Structural tensor of (a). It can be seen that the background of (a) has a great degree of texture, and for that reason the overall image on (b) has a high degree of coherence. (c) Original image of a common scene. (d) Structural tensor of (c). It can be seen that only object boundaries contain a high degree of coherence while the remainder of the image lacks enough texture for dense or sparse correspondence purposes.

By using mapping on (8.86) we can obtain visualization results of the structural tensor as shown in fig. 8.3. We give two samples, the first one containing an image from the Middlebury dataset which has a great degree of coherence. It must be noticed that images from the Middlebury dataset are specially crafted so that dense disparity can be computed over them. For that reason a highly textured blanked has ben set on the background. However in common scenarios as shown in the second example, usually coherence is much lower. For that reason we must detect which regions can be more easily and robustly matched, and that's where the structural tensor plays its main role along with corner detectors as will be seen next.

8.4.- Corner detectors

We have seen how the structural tensor allows retrieval of coherence and orientation information in an image by means of gradient information. In this section we'll see how we can take advantage of such data in order to pick points of interest in a given image, which in this case are considered as corners.

It must be noticed that the meaning of "corner" in this section does not refer to actual corner structures in an image, but instead to areas having two main gradient components. For that reason, many corner detectors are based on gradient information of an image.

It can be seen that corners are usually located in areas with high coherence in the structural tensor, and as such are usually easy to match, although it might happen that corners appear in areas which are prone to be occluded, as high coherence areas are usually located at object boundaries.

The task of a corner detector will be to pick the most salient corners which are likely to appear in multiple views as robustly and exactly as possible so that precise matches can be obtained and discard the remaining ones. It will also be desirable to pick corners as uniformly distributed (over an image or a region of interest inside an image) as possible, since this will allow more computational accuracy when estimating epipolar geometry and new camera poses during the calibration and structure recovery process. If matches are all located too close to each other on the same region it is likely that degeneracy might appear due to numerical machine precision, leading to wrong 3D sparse structure and wrong epipolar geometry estimation. To avoid this, usually a region of interest is picked on each image, and tiling is used to pick the most salient corners on each tile as will be shown in the following sections.

8.4.1.- Harris corner detector

Harris corner detector [19] is based on the structure $\mathbf{T}$ (8.39) containing aggregated gradient data in order to obtain the most salient corners on an image. Corners are estimated directly from gradient data without needing the eigen-decomposition of the structural tensor by means of the following corner response function for each pixel of an image:

$$R = \det(\mathbf{T}) - k \cdot \text{Tr}^2(\mathbf{T}) \quad (8.87)$$

where k is a parameter that can be adjusted for different results, although Harris obtained empirically that the best value to use was $k = 0.04$. It can be seen that, since no actual eigen-decomposition is needed, Harris corner detector becomes a simple way for corner detection. We can also see that from (8.59) and (8.61), expression (8.87) is equivalent to:

$$R = \lambda_1 \cdot \lambda_2 - k \cdot (\lambda_1 + \lambda_2)^2 \quad (8.88)$$



Fig. 8.4. Example of Harris corners detector on the Ballet sequence [41].

this directly over the score values, we'd see that all the highest scores are located on a single small area, and as we have seen before, this is undesirable as it will later create degeneracy and numerical precision errors when epipolar geometry is estimated.

And if we look to table 8.2 to observe how the eigenvalues affect the response function, we obtain that the higher R becomes, the more likely it is that a real corner is present (i.e. when $\lambda_1 \geq \lambda_2 \gg 0$).

Indeed it can be proved empirically that the larger the $\text{Tr}(\mathbf{T})$ becomes, the more likely an edge is present, and conversely, the larger $\det(\mathbf{T})$ becomes, the more likely an edge or a corner is present. For that reason expression (8.87) subtracts $\text{Tr}(\mathbf{T})$ from $\det(\mathbf{T})$, so that only corners get selected.

Since expression (8.87) assigns corner scores to each pixel in an image. Now we only need to pick the pixels with the highest score values. However, if we did

In order to avoid this and obtain corners in a more uniform way, it can be useful to uniformly divide images into smaller regions or tiles and select the most salient corner scores on each tile. If we also require that corners do not get too close to each other on each tile, selected corners on each tile must correspond to local maxima on the response function R .

This method has proven to be good enough, and our experiments have shown that using a 10 x 10 tile grid over the whole image and selecting between 1 and 10 corners on each tile is more than enough. An example of Harris corner detector in action is shown in fig. 8.4.

8.4.2.- Corner detector using the structural tensor

Other corner detectors can also be estimated from gradient information, such as Beaudet, Förstner, SUSAN, or as will be explained below from the structural tensor [5]. In table 8.2 we showed the meaning of the eigenvalues of $\mathbf{T}$, and hence we know that corners are located in areas where:

$$\lambda_1(x_i, y_i) \geq \lambda_2(x_i, y_i) \gg k \quad (8.89)$$

where k is a threshold parameter, that can be set to 0 if tiling is used to prioritize corners in a uniform way.

Expression (8.89) can be rewritten in a more practical way as:

$$\lambda_1(x_i, y_i) - \lambda_2(x_i, y_i) > k \quad (8.90)$$

where k is usually set to 0. And by using expression (8.60), then (8.90) becomes:

$$\left[T_{11}(x_i, y_i) - T_{22}(x_i, y_i) \right]^2 + 4T_{12}^2(x_i, y_i) > k^2 \quad (8.91)$$

where eigen-decomposition of $\mathbf{T}$ is no longer needed.

As in the case of the Harris corner detector, expression (8.91) provides scores for each pixel where the highest the score is the most likely a pixel is a salient corner. Again, before picking the highest scores directly, it is desirable to first apply some tiling to pick most salient corners on each tile separately to avoid degeneracy and precision errors, and if further separation between corners on each tile is needed, it can be enforced that corners lie on local maxima of response function (8.91).



Fig. 8.5. Example of Structural Tensor corner detector on the Ballet sequence [41].

Also, as in the Harris corner detector, a 10 x 10 tile grid with 1 to 10 corners on each tile has proven to be good enough for reliable matching between pairs of images. Again, an example of this corner detector in action is shown in fig. 8.5.

8.4.3.- SIFT

Among the corner detectors explained in this section, SIFT [20] is probably the most robust among them, but it is also by far the most computationally intensive one.

It is based on the usage of a pyramid of DoG (Difference of Gaussians) in order to reliably estimate salient points. On each pyramid level, salient points are checked in order to discard those that are not likely to provide reliable matching. At the same

time, SIFT also computes descriptors based on coherence and orientation in a similar way as the structural tensor does by using eigen-decomposition of aggregated gradient data over a given region.

It must be noticed that even though SIFT is very invariant upon affine transformations, it is not so robust against projective ones, which is the main case of interest for 3D reconstruction, especially for wide baselines between pairs of images, where projective distortions are more visible.

In any case, we have experimentally observed that for wide baselines, SIFT usually obtains the most robust corner points to be reliably matched, however, for smaller baselines, SIFT provides results equally good or even worse than a conventional Harris corner detector, while at the same time being a lot more computationally intensive.

That said, it can be easily inferred that when working with video sequences for 3D scene reconstruction, usually simple match measures such as CV (Covariance-Variance) and a simple corner detector such as Harris or the one based on the structural tensor are enough to obtain reliable matches since baselines are small. However, when working with still images, if baselines are not too big for reliable matching, usually more robust solutions like log-polar transformed matches and SIFT corner detectors are needed since baselines are usually larger than in video sequences.

8.4.4.- Sub-pixel accuracy on corner detection

For some applications (i.e. self-calibration), it might be of particular interest to be able to detect corner locations with a high degree of accuracy, usually below 1 pixel. This goal can be achieved on some corner detection algorithms by means of their respective corner response functions and interpolation.

8.4.4.1.- One dimensional polynomial interpolation

Indeed, corners are usually located at maxima of a given corner response function; and by means of a second degree polynomial curve fitting we can retrieve a more exact location. So looking at fig. 8.6, which we can image that corresponds to a particular response function at a corner location in a given direction (in fig. 8.6 horizontal direction is depicted, but the

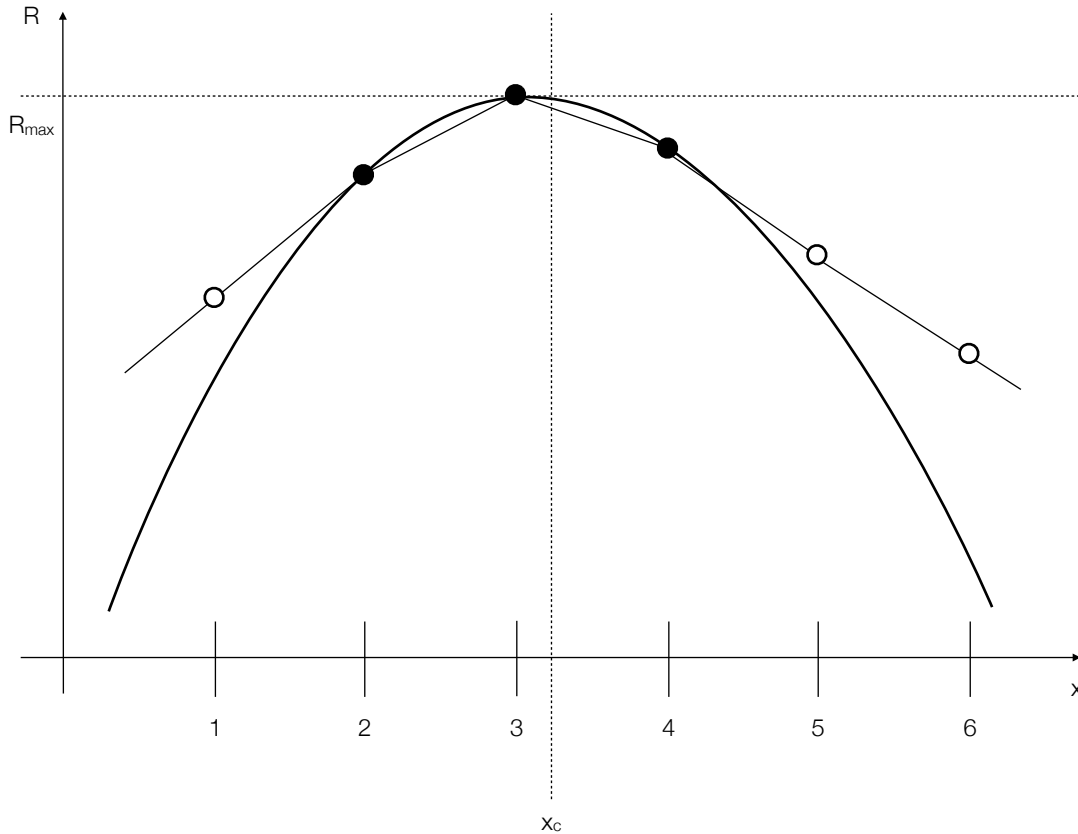


Fig. 8.6. Corner response function at a corner location in the x-direction

vertical case can be analyzed analogously), we can see that by using the 2 neighbor points surrounding a given maximum response, we can establish the following set of data:

$$\{x_{i-1}, R_{i-1}\}, \{x_i, R_i\}, \{x_{i+1}, R_{i+1}\} \quad (8.92)$$

where $R_i = R(x_i)$ stands for the corner response function at location x_i , and indexes $i-1$ and $i+1$ stand for locations at both sides of corner response maxima, which is located at x_i .

Hence, we want to determine the following 2nd degree polynomial:

$$f(x_i) = ax_i^2 + bx_i + c = R_i, \quad (a \neq 0) \quad (8.93)$$

Using expression (8.93) and the set of data on (8.92) we can establish the following linear system of equations:

$$\begin{cases} ax_{i-1}^2 + bx_{i-1} + c = R_{i-1} \\ ax_i^2 + bx_i + c = R_i \\ ax_{i+1}^2 + bx_{i+1} + c = R_{i+1} \end{cases} \quad (8.94)$$

which can be expressed in matrix form as:

$$\mathbf{X}\boldsymbol{\beta} = \mathbf{R} \quad (8.95)$$

where $\mathbf{X}$ is the matrix of locations:

$$\mathbf{X} = \begin{bmatrix} x_{i-1}^2 & x_{i-1} & 1 \\ x_i^2 & x_i & 1 \\ x_{i+1}^2 & x_{i+1} & 1 \end{bmatrix} \quad (8.96)$$

$\boldsymbol{\beta}$ is the vector of polynomial parameters

$$\boldsymbol{\beta} = (a, b, c)^T \quad (8.97)$$

and $\mathbf{R}$ is the vector containing corner response values

$$\mathbf{R} = \begin{bmatrix} R_{i-1} \\ R_i \\ R_{i+1} \end{bmatrix} \quad (8.98)$$

The system can be easily be solved when 3 points of data are provided, however, if more points are provided, we need to solve an overdetermined linear system of equations. In this case, using a linear least squares minimization criterium, then the system of equations can be solved by means of the Moore-Penrose pseudo-inverse as follows:

$$\boldsymbol{\beta} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{R} \quad (8.99)$$

where $\mathbf{X}^\dagger = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T$ is the Moore-Penrose pseudo-inverse.

Once the polynomial parameters have been estimated using the procedure shown above, we can double differentiate the polynomial in order to precisely find where the maximum is located. Indeed, the maximum will be located at a point where the first derivative is zero, and the second derivative is negative. In other words:

$$\frac{\partial f(x_c)}{\partial x_c} = 2ax_c + b = 0 \quad \text{and} \quad \frac{\partial^2 f(x_c)}{\partial x_c^2} = 2a < 0 \quad (8.100)$$

Hence,

$$x_c = -\frac{b}{2a} \quad \text{and} \quad a < 0 \quad (8.101)$$

Notice that if multiple corners need to be adjusted with sub-pixel accuracy, it is interesting to always shift the corner coordinates to the origin, so that $\mathbf{X}$ remains constant and its pseudo-inverse needs only to be computed once. In that case shifted $\mathbf{X}$ becomes:

$$\mathbf{X}' = \begin{bmatrix} x_{i-1}^2 & x_{i-1}' & 1 \\ x_i^2 & x_i' & 1 \\ x_{i+1}^2 & x_{i+1}' & 1 \end{bmatrix} = \begin{bmatrix} (-1)^2 & -1 & 1 \\ 0 & 0 & 1 \\ 1^2 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & -1 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad (8.102)$$

and we would solve for the shifted polynomial parameters $\beta' = (a', b', c')^T$ which would allow us to find the exact shifted maxima x'_c by using expression (8.101) over the shifted polynomial parameters.

$$x'_c = -\frac{b'}{2a'} \quad \text{and} \quad a' < 0 \quad (8.103)$$

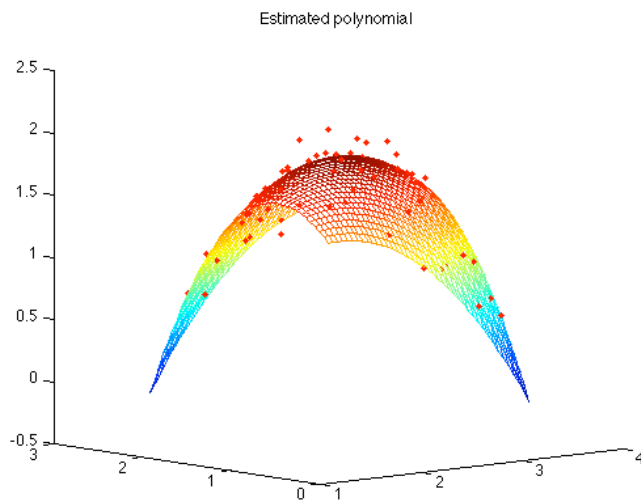


Fig. 8.7. Linear least squares two dimensional polynomial estimation from data samples.

That way, the exact shifted maxima x'_c would become a value ranging between -1 and 1.

And finally, once the shifted maxima is computed, we can recover the actual maxima location by reversing the shift as follows:

$$x_c = x_i + x'_c \quad (8.104)$$

where x_c is the exact horizontal corner location, x_i is the integer horizontal location of the corner response maxima, and x'_c is the shifted horizontal corner response maxima.

Up to this point we have seen how to accurately obtain the horizontal location of a corner, however, it must be noticed that the vertical location can be obtained analogously by solving (8.95) over the vertical coordinates.

8.4.4.2.- Two dimensional polynomial interpolation

The method shown on previous sub-section already gives good accuracy for corner locations, however if even more precision is desired, we can achieve even better results at the expense of a slight computational burden by means of a two dimensional polynomial as the one shown in fig. 8.7:

$$f(\mathbf{x}) = f(x_i, y_i) = ax_i^2 + by_i^2 + cx_iy_i + dx_i + ey_i + f = R_i \quad (8.105)$$

Which can be fitted to provided corner response data by means of linear least squares by defining the following linear system of equations:

$$\mathbf{X}\beta = \mathbf{R} \quad (8.106)$$

where the matrix $\mathbf{X}$ of locations is now:

$$\mathbf{X} = \begin{bmatrix} x_{i-1}^2 & y_{j-1}^2 & x_{i-1}y_{j-1} & x_{i-1} & y_{j-1} & 1 \\ x_i^2 & y_{j-1}^2 & x_iy_{j-1} & x_i & y_{j-1} & 1 \\ x_{i+1}^2 & y_{j-1}^2 & x_{i+1}y_{j-1} & x_{i+1} & y_{j-1} & 1 \\ x_{i-1}^2 & y_j^2 & x_{i-1}y_j & x_{i-1} & y_j & 1 \\ x_i^2 & y_j^2 & x_iy_j & x_i & y_j & 1 \\ x_{i+1}^2 & y_j^2 & x_{i+1}y_j & x_{i+1} & y_j & 1 \\ x_{i-1}^2 & y_{j+1}^2 & x_{i-1}y_{j+1} & x_{i-1} & y_{j+1} & 1 \\ x_i^2 & y_{j+1}^2 & x_iy_{j+1} & x_i & y_{j+1} & 1 \\ x_{i+1}^2 & y_{j+1}^2 & x_{i+1}y_{j+1} & x_{i+1} & y_{j+1} & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} \quad (8.107)$$

which contains the location of all values directly surrounding a maximum in the corner response function.

On the other hand, the vector of parameters beta is now:

$$\beta = (a, b, c, d, e, f)^T \quad (8.108)$$

And the vector $\mathbf{R}$ containing the corner response values is:

$$\mathbf{R} = (R_{(i-1,j-1)}, R_{(i,j-1)}, R_{(i+1,j-1)}, R_{(i-1,j)}, R_{(i,j)}, R_{(i+1,j)}, R_{(i-1,j+1)}, R_{(i,j+1)}, R_{(i+1,j+1)}, \dots)^T \quad (8.109)$$

It can be observed that expression (8.106) can be solved with at least six points. If more points are provided (as will usually be the case), then a least squares solution is found by means of the Moore-Penrose pseudo-inverse, as in (8.99).

Analogously to previous section, once the two dimensional polynomial parameters have been found, we can search for the exact maximum by means of double differentiation. More precisely, we will set the first derivatives to zero, and ensure that the second derivatives are negative.

$$\frac{\partial f(x_c, y_c)}{\partial x_c} = 2ax_c + cy_c + d = 0 \quad \text{and} \quad \frac{\partial^2 f(x_c, y_c)}{\partial x_c^2} = 2a < 0 \quad (8.110)$$

$$\frac{\partial f(x_c, y_c)}{\partial y_c} = 2by_c + cx_c + e = 0 \quad \text{and} \quad \frac{\partial^2 f(x_c, y_c)}{\partial y_c^2} = 2b < 0 \quad (8.111)$$

By analyzing expression (8.110) and (8.111) we see that we have another linear system of equations in order to solve the exact corner location (x_c, y_c) .

$$\begin{cases} 2ax_c + cy_c = -d \\ cx_c + 2by_c = -e \end{cases} \quad (8.112)$$

Which can be rewritten in matrix form as:

$$\begin{bmatrix} 2a & c \\ c & 2b \end{bmatrix} \begin{bmatrix} x_c \\ y_c \end{bmatrix} = \begin{bmatrix} -d \\ -e \end{bmatrix} \quad (8.113)$$

And consequently, the exact corner location can be solved as:

$$\mathbf{x}_c = \begin{bmatrix} x_c \\ y_c \end{bmatrix} = \begin{bmatrix} 2a & c \\ c & 2b \end{bmatrix}^{-1} \begin{bmatrix} -d \\ -e \end{bmatrix} \quad (8.114)$$

As in previous sub-section, when many exact corners need to be computed, it is desirable to shift their coordinates to the origin so that the pseudo-inverse of (8.107) only needs to be computed once. In that case we can set $(x'_i, y'_i) = (0, 0)$ and then expression (8.107) becomes after shifting:

$$\mathbf{X}' = \begin{bmatrix} x_{i-1}^{i2} & y_{j-1}^{i2} & x'_{i-1} y'_{j-1} & x'_{i-1} & y'_{j-1} & 1 \\ x_i^{i2} & y_{j-1}^{i2} & x'_i y'_{j-1} & x'_i & y'_{j-1} & 1 \\ x_{i+1}^{i2} & y_{j-1}^{i2} & x'_{i+1} y'_{j-1} & x'_{i+1} & y'_{j-1} & 1 \\ x_{i-1}^{i2} & y_j^{i2} & x'_{i-1} y'_j & x'_{i-1} & y'_j & 1 \\ x_i^{i2} & y_j^{i2} & x'_i y'_j & x'_i & y'_j & 1 \\ x_{i+1}^{i2} & y_j^{i2} & x'_{i+1} y'_j & x'_{i+1} & y'_j & 1 \\ x_{i-1}^{i2} & y_{j+1}^{i2} & x'_{i-1} y'_{j+1} & x'_{i-1} & y'_{j+1} & 1 \\ x_i^{i2} & y_{j+1}^{i2} & x'_i y'_{j+1} & x'_i & y'_{j+1} & 1 \\ x_{i+1}^{i2} & y_{j+1}^{i2} & x'_{i+1} y'_{j+1} & x'_{i+1} & y'_{j+1} & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} = \begin{bmatrix} (-1)^2 & (-1)^2 & (-1) \cdot (-1) & -1 & -1 & 1 \\ 0^2 & (-1)^2 & 0 \cdot (-1) & 0 & -1 & 1 \\ 1^2 & (-1)^2 & 1 \cdot (-1) & 1 & -1 & 1 \\ (-1)^2 & 0^2 & (-1) \cdot 0 & -1 & 0 & 1 \\ 0^2 & 0^2 & 0 \cdot 0 & 0 & 0 & 1 \\ 1^2 & 0^2 & 1 \cdot 0 & 1 & 0 & 1 \\ (-1)^2 & 1^2 & (-1) \cdot 1 & -1 & 1 & 1 \\ 0^2 & 1^2 & 0 \cdot 1 & 0 & 1 & 1 \\ 1^2 & 1^2 & 1 \cdot 1 & 1 & 1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & -1 & -1 & 1 \\ 0 & 1 & 0 & 0 & -1 & 1 \\ 1 & 1 & -1 & 1 & -1 & 1 \\ 1 & 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & -1 & -1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} \quad (8.115)$$

Then for each corner we would solve the shifted polynomial parameters $\beta' = (a', b', c', d', e', f')^T$. And then, once they are computed, we can retrieve the shifted maxima $\mathbf{x}'_c$ by using expression (8.114) over the shifted polynomial parameters:

$$\mathbf{x}'_c = \begin{bmatrix} x'_c \\ y'_c \end{bmatrix} = \begin{bmatrix} 2a' & c' \\ c' & 2b' \end{bmatrix}^{-1} \begin{bmatrix} -d' \\ -e' \end{bmatrix} \quad (8.116)$$

We can observe that the shifted maxima in (8.116) will be located within the square region defined by points $(-1,-1)$, $(1,-1)$, $(-1,1)$ and $(1,1)$. Again as in section 8.4.4.1 we can find the exact corner location by reversing the shift as follows:

$$\begin{aligned} \mathbf{x}_c &= \mathbf{x}_i + \mathbf{x}'_c \\ (x_c, y_c)^T &= (x_i, y_i)^T + (x'_c, y'_c)^T \end{aligned} \quad (8.117)$$

where $\mathbf{x}_c$ vector is the exact corner location, $\mathbf{x}_i$ vector is the integer location of the corner response maxima, and $\mathbf{x}'_c$ vector is the shifted corner response maxima.

8.5.- Closure

In this chapter we have seen how to accurately estimate gradient information from intensity images, and why it is important to make such accurate estimation.

We then saw how the gradient information is related to image coherence and orientation by means of the structural tensor, and how such structural tensor is related to corner detectors. Aside from that, the concepts explained to retrieve the structural tensor will also be employed in other chapters, such as for adaptive windowing in chapter 7, for control point computation on dynamic programming techniques or for anisotropic filtering during the multiresolution process for dense disparity estimation, which both are shown in chapter 9.

We have also mentioned how to uniformly detect corners of high coherence on images, which can be useful for sparse structure retrieval and cameras' self-calibration. We also showed, that from the different corners detectors that we presented, we proved experimentally that the Harris corner detector will usually be the most convenient because of its simplicity and computational speed, especially when working with video sequences recorded using a hand-held camera where baseline length between consecutive frames is small. In such cases, once the corners are computed this way, a simple NCC or CV match measure might be used to match them between pairs of images.

As we also mentioned, we showed that if we intend to work with still images taken from different points of view, then more robust techniques will be needed, such as SIFT for corner estimation, and log-polar transforms (see chapter 7) to correct distortions when matching estimated corners. This is due to the fact that for still images usually the baseline is wider between pairs of images and hence projective distortions become more evident, making point correspondence less reliable.

Finally, because epipolar geometry estimation should be done as robustly as possible, we have also provided means to detect corners with sub-pixel accuracy by means of polynomial interpolation. Although robust techniques such as RANSAC or LMedS exist for the Fundamental matrix estimation (which contains the epipolar geometry), that take into account possible outliers. If sub-pixel accuracy is provided on corner measures, then the accuracy of estimation of epipoles and epipolar lines will be further increased (i.e. reaching sub-pixel accuracy).

As a conclusion, it must be observed that corner detection and matching can be used to estimate epipolar geometry, and as will be seen on the following chapter, this will be used for image rectification (i.e. to impose epipolar constraints), which can help to increase quality of dense disparity estimation. And because from epipolar geometry also camera self-calibration can be computed, this will allow us to compute a 3D reconstruction of the scene.

On the other hand, this can also be done the other way around by noticing that if by some means a correct dense disparity can be obtained without any knowledge on any prior epipolar geometry, then, epipolar geometry and self calibration of cameras can also be obtained faster and more robustly from such dense disparity information if it is accurate enough. In that sense, we will see that the multiresolution technique for dense disparity estimation explained in chapter 9 can be used this way, allowing us to precisely obtain both cameras and dense disparity estimation quite efficiently.

9.- Dense depth estimation

9.1.- Introduction

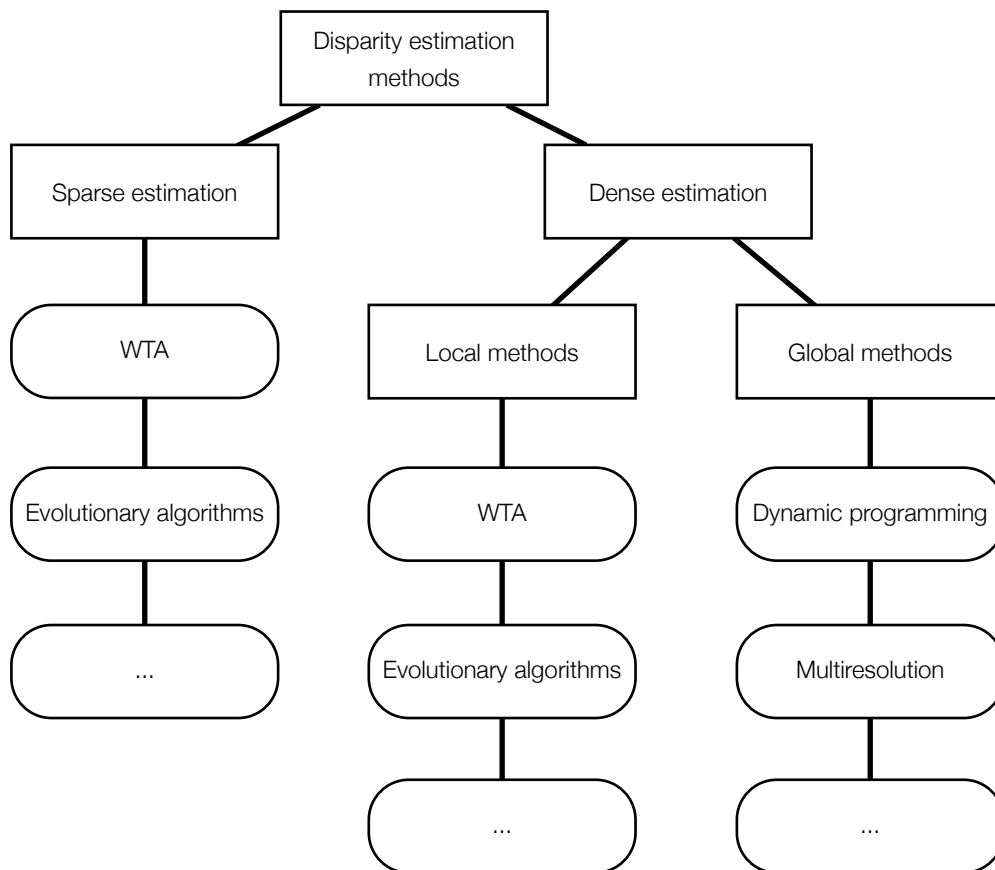


Fig. 9.1. Diversity of disparity estimation methods

On chapter 6 and 7 we saw methods to process images and obtain matched pairs of points, and chapter 8 has shown how to sparsely detect the most significant points to obtain reliable matches that will allow us to compute self-calibration of cameras and a sparse 3D structure of the scene to be reconstructed.

However, for the purposes of accurate 3D reconstruction and accurate texturing (especially if super-resolution [7] texturing is needed), it might be needed to compute dense disparity (i.e. point correspondences) so that many more triangulated 3D points can be estimated.

This chapter introduces the reader into different techniques for dense disparity estimation, explaining the different restrictions that can be applied to increase the quality of matches as well as different minimization algorithms that can be used to reliably estimate dense disparity between a pair of images. Finally it will be explained different post-processing techniques that can

be used either to increase matching quality up to sub-pixel accuracy, or to detect incoherent or occluded areas to be avoided during the reconstruction stage.

For the sake of clarity, it can be useful to understand that disparity estimation can be classified in two main groups: sparse and dense disparity estimation. The former is referred to the estimation of isolated point correspondences as it was seen on chapters 7 and 8, the latter refers to finding point correspondences for all the points on an image (or at least as many as possible if occlusions or incoherences appear or a region of interest is defined). Among the latter group, another typical subdivision is done between local and global dense disparity algorithms. The former are usually faster to compute although quality is slightly worse, on the other hand the latter group usually offers the best quality at the expense of a much higher computational cost (although a few exceptions exist as it was seen on section 7.8). This classification is summarized on figure 9.1.

9.2.- Image rectification

Image rectification is the process of transforming epipolar lines so that they become parallel (i.e. the epipoles are moved to the infinity) and coincident (a transformed epipolar line at a given horizontal location on one image matches another transformed epipolar line on another image at the same vertical location). This way, there's no need to search for matches at different horizontal and vertical disparity values, and the search problem is reduced from 2D to 1D usually by searching only along the transformed epipolar lines, which are coincident and usually horizontal.

Hence, after rectification of a pair of images, only horizontal disparity values need to be taken into account during match search for each pixel. As it can simply be observed, this hugely simplifies the search problem for many algorithms that either obtain more robust or faster (or both) matches.

It must be observed also, that by means of rectification, search can be done over the rectified epipolar lines, and this can also be equivalently implemented by directly searching on generated epipolar lines over the images without the need of rectification, and thus avoiding possible disadvantages of rectification that will be seen next (i.e. distortion and epipoles inside image limits). It must be noticed, though, that the main advantage of rectification is that usually matching algorithms work faster on rectified images by simply matching horizontal disparity values instead of trying to search for matches over each generated epipolar lines, however rectification also has its disadvantages.

In the following sections we will discuss different image rectification techniques, as well as their advantages and disadvantages and when should they be used or not.

9.2.1.- Hartley's rectification

The method proposed by Hartley [8] is based on the fundamental matrix, which must be robustly estimated before attempting to rectify a pair of images. By using the fundamental matrix a pair of 2D projective transformations is obtained (one for each image) so that epipolar lines become parallel and mostly coincident (i.e. only horizontal disparity is present and any vertical disparity is removed).

9.2.1.1.- Mapping the epipole to infinity

In order to obtain parallel epipolar lines on a pair of images, we need to compute a transformation $\mathbf{H}$ that maps the epipole to the infinity, only in such case epipolar lines become parallel. Moreover, we want the epipolar lines to be parallel with the x-axis of the pair of images. For that reason we need that $\mathbf{H}$ maps the epipole to the particular point $(1,0,0)^T$. However, because $\mathbf{H}$ will be a projective transform, if $\mathbf{H}$ is not chosen carefully, severe distortions might appear on the transformed image.

For that reason, usually a point x_0 is chosen (usually at image center) so that distortion is minimal on its neighborhood, and projective distortion becomes more evident as we get further from that point. And by setting the point in the middle, overall distortion becomes less evident.

Hence, supposing x_0 at the origin of coordinates and an epipole $\mathbf{e} = (f, 0, 1)^T$ laying on the x-axis, we can consider the following transformation:

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1/f & 0 & 1 \end{bmatrix} \quad (9.1)$$

we can clearly see that transformation (9.1) takes the epipole $\mathbf{e}$ to infinity, as required,

$$\mathbf{G} \cdot \mathbf{e} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1/f & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} f \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} f \\ 0 \\ 0 \end{bmatrix} \quad (9.2)$$

In fact, any point $(x, y, 1)^T$ gets transformed by $\mathbf{G}$ into:

$$(\hat{x}, \hat{y}, 1)^T = \mathbf{G} \cdot (x, y, 1)^T = (x, y, 1 - x/f)^T \quad (9.3)$$

which can be expressed in inhomogeneous coordinates as:

$$(\hat{x}, \hat{y}, 1)^T = \left(\frac{x}{1 - x/f}, \frac{y}{1 - x/f}, 1 \right)^T \quad (9.4)$$

to approximate, we can take the Taylor series of the denominator at origin $x=0$, so that given the function

$$f(x) = \frac{1}{1 - x/f} \quad (9.5)$$

Its Taylor series of first order is:

$$\begin{aligned} f(a) &= f(x)|_{x=a} \approx f(a) + \frac{1}{1!} \frac{\partial}{\partial x} f(x) \Big|_{x=a} \cdot (x - a) + \dots \\ &= \frac{1}{1 - a/f} + \frac{1}{f(1 - a/f)^2} (x - a) + \dots, \quad |x/f| < 1 \end{aligned} \quad (9.6)$$

And evaluated at the origin it becomes:

$$f(0) \approx 1 + x/f + \dots, \quad |x/f| < 1 \quad (9.7)$$

but since we're evaluating at the origin, and expression (9.5) has a singularity at $x=f$, then expression (9.7) is only valid in the range $|x/f| < 1$, where expression (9.5) is differentiable and has no singularities.

Hence, expression (9.4) becomes,

$$(\hat{x}, \hat{y}, 1)^T = (x(1 + x/f + \dots), y(1 + x/f + \dots), 1)^T \quad (9.8)$$

And the Jacobian of (9.8) can be computed as:

$$\frac{\partial(\hat{x}, \hat{y})}{\partial(x, y)} = \begin{bmatrix} \frac{\partial \hat{x}}{\partial x} & \frac{\partial \hat{x}}{\partial y} \\ \frac{\partial \hat{y}}{\partial x} & \frac{\partial \hat{y}}{\partial y} \end{bmatrix} = \begin{bmatrix} 1 + 2x/f & 0 \\ y/f & 1 + x/f \end{bmatrix} \quad (9.9)$$

which we can see that at the origin ($x = 0, y = 0$) it becomes close to the identity matrix, which indicates that **G** produces no distortion close to the origin.

Up to this point we have seen how **G** maps an epipole located on the x-axis to the infinity assuming that the origin is at zero. We'd like now to be able to map any epipole to the infinity assuming the origin of coordinates (or point of interest) at any point x_0 , which is usually the center of the image. Hence we define the following **H** transform as a product:

$$\mathbf{H} = \mathbf{G} \cdot \mathbf{R} \cdot \mathbf{T} \quad (9.10)$$

where **T** is a translation taking the point x_0 to the origin, **R** is a rotation about the origin taking the epipole e' to a point $(f, 0, 1)^T$ on the x-axis and **G**, as we have already seen, is the mapping seen on (9.1) that takes $(f, 0, 1)^T$ to infinity.

Hence, **T** can be expressed as:

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & -x_0 \\ 0 & 1 & -x_0 \\ 0 & 0 & 1 \end{bmatrix} \quad (9.11)$$

And **R** can be expressed as:

$$\mathbf{R} = \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix} \quad (9.12)$$

where **R** rotates the resulting epipole **T·e** clockwise by the angle θ . Hence, angle θ must be equal to the angle of **T·e** respect to the x-axis. In other words:

$$\mathbf{v} = \mathbf{T}\mathbf{e} = (v_1, v_2, 1)^T$$

$$\theta = \arctan\left(\frac{v_2}{v_1}\right) \quad (9.13)$$

9.2.1.2.- Making epipolar lines coincident

Up to this point we have seen how to move the epipole to the infinity so that epipolar lines becomes parallel, our purpose now is to ensure that transformed epipolar lines match to each other in a pair of images by obtaining a *matched pair* of transformations **H**, and **H'**.

More specifically making epipolar lines coincident can be expressed as:

$$\mathbf{H}^{-T}\mathbf{l} = \mathbf{H}'^{-T}\mathbf{l}' \quad (9.14)$$

where $\mathbf{H}$ is the transformation in (9.10) for the left view, $\mathbf{H}'$ is the transformation in (9.10) for the right view, $\mathbf{l}$ is an epipolar line on the left view and $\mathbf{l}'$ is an epipolar line on the right view (recall that $\mathbf{H}^{-T}$ is the line map corresponding to the point map $\mathbf{H}$).

In order to find a matched pair of transformations, we will first compute the Fundamental matrix $\mathbf{F}$ of a pair of images, compute its epipole $\mathbf{e}'$ on the right image, and use the procedure on section 9.2.1.1 to obtain the transformation $\mathbf{H}'$ that moves the epipole $\mathbf{e}'$ to infinity. Once $\mathbf{H}'$ is known we will find the transformation $\mathbf{H}$ by minimizing the following sum-of-squared distances criterium:

$$\sum_i d(\mathbf{H}\mathbf{x}_i, \mathbf{H}'\mathbf{x}_i')^2 \quad (9.15)$$

For a pair of images $\mathbf{l}$ and $\mathbf{l}'$ with fundamental matrix $\mathbf{F} = [\mathbf{e}']_{\times} \mathbf{M}$ and being $\mathbf{H}'$ the projective transformation of image $\mathbf{l}'$.

We obtain that a projective transformation $\mathbf{H}$ of image $\mathbf{l}$ matches $\mathbf{H}'$ if and only if $\mathbf{H}$ is of the form

$$\mathbf{H} = (\mathbf{I} + \mathbf{H}'\mathbf{e}'\mathbf{a}^T) \mathbf{H}'\mathbf{M} \quad (9.16)$$

for some vector $\mathbf{a}$.

Proof 9.1

If $\mathbf{x}$ is a point in $\mathbf{l}$, then $\mathbf{e} \times \mathbf{x}$ is a line that goes from epipole $\mathbf{e}$ to point $\mathbf{x}$ on the first image (i.e. an epipolar line), and $\mathbf{F}\mathbf{x}$ is an epipolar line on the second image. Then expression (9.14) becomes:

$$\mathbf{H}^{-T}(\mathbf{e} \times \mathbf{x}) = \mathbf{H}'^{-T}\mathbf{F}\mathbf{x} \quad (9.17)$$

Since expression (9.17) must hold for any point $\mathbf{x}$, (epipolar lines always go through the epipole and are generated by the Fundamental matrix as explained above), then we might equivalently obtain:

$$\mathbf{H}^{-T}[\mathbf{e}]_{\times} = \mathbf{H}'^{-T}\mathbf{F} = \mathbf{H}'^{-T}[\mathbf{e}']_{\times} \mathbf{M} \quad (9.18)$$

taking into account that cross product between epipole $\mathbf{e}$ and point $\mathbf{x}$ can be also expressed as:

$$(\mathbf{e} \times \mathbf{x}) = [\mathbf{e}]_{\times} \cdot \mathbf{x} \quad (9.19)$$

where $\mathbf{e} = (e_1, e_2, e_3)^T$ is the epipole and $[\mathbf{e}]_{\times}$ is the skew-symmetric matrix representation:

$$[\mathbf{e}]_{\times} = \begin{bmatrix} 0 & -e_3 & e_2 \\ e_3 & 0 & -e_1 \\ -e_2 & e_1 & 0 \end{bmatrix} \quad (9.20)$$

that allows computation of a cross product between vectors as the product of the skew-symmetric matrix by the other vector.

By taking into account the following lemma:

Lemma 9.1

Supposing that the rank 2 matrix $\mathbf{F}$ can be decomposed in two different ways a $\mathbf{F} = [\mathbf{e}]_{\times} \mathbf{M}$ and $\mathbf{F} = [\tilde{\mathbf{e}}]_{\times} \tilde{\mathbf{M}}$, where the epipole can change up to scale $\tilde{\mathbf{e}} = k\mathbf{e}$ and $\mathbf{M}$ can change as follows $\tilde{\mathbf{M}} = k^{-1}(\mathbf{M} + \mathbf{e}\mathbf{v}^T)$ for some non-zero constant k and 3-vector $\mathbf{v}$.

Then expression (9.18) can be rewritten as:

$$\mathbf{H}^T [\mathbf{e}]_{\times} = \mathbf{H}'^T [\mathbf{e}']_{\times} \mathbf{M} = \mathbf{H}'^T [k\mathbf{e}']_{\times} k^{-1} (\mathbf{M} + \mathbf{e}'\mathbf{v}^T) \quad (9.21)$$

as the fundamental matrix is:

$$\mathbf{F} = [\mathbf{e}']_{\times} \mathbf{M} = [k\mathbf{e}']_{\times} k^{-1} (\mathbf{M} + \mathbf{e}'\mathbf{v}^T) \quad (9.22)$$

In expression (9.21) constant k gets cancelled resulting in:

$$\mathbf{H}^T [\mathbf{e}]_{\times} = \mathbf{H}'^T [\mathbf{e}']_{\times} (\mathbf{M} + \mathbf{e}'\mathbf{v}^T) \quad (9.23)$$

Again by using the following lemma:

Lemma 9.2

For any vector $\mathbf{t}$ and non-singular matrix $\mathbf{M}$ one has:

$$[\mathbf{t}]_{\times} \mathbf{M} = \mathbf{M}^* [\mathbf{M}^{-1}\mathbf{t}]_{\times} = \mathbf{M}^T [\mathbf{M}^{-1}\mathbf{t}]_{\times} \quad (\text{up to scale}) \quad (9.24)$$

And hence, by considering $\mathbf{H} = \mathbf{M}$ and $\mathbf{e} = \mathbf{M}^{-1}\mathbf{t}$, we observe that in expression (9.23)

$$\mathbf{H}^T [\mathbf{e}]_{\times} = \mathbf{H}^T [\mathbf{H}^{-1}\mathbf{t}]_{\times} = [\mathbf{t}]_{\times} \mathbf{H} \quad (9.25)$$

and since $\mathbf{t} = \mathbf{M}\mathbf{e}$:

$$\mathbf{H}^T [\mathbf{e}]_{\times} = [\mathbf{H}\mathbf{e}]_{\times} \mathbf{H} \quad (9.26)$$

And analogously:

$$\mathbf{H}'^T [\mathbf{e}']_{\times} = [\mathbf{H}'\mathbf{e}']_{\times} \mathbf{H}' \quad (9.27)$$

And then expression (9.23) becomes:

$$[\mathbf{H}\mathbf{e}]_{\times} \mathbf{H} = [\mathbf{H}'\mathbf{e}']_{\times} \mathbf{H}' (\mathbf{M} + \mathbf{e}'\mathbf{v}^T) \quad (9.28)$$

Because we know that both $\mathbf{H}\mathbf{e}$ and $\mathbf{H}'\mathbf{e}'$ map the epipoles to a point in infinity $(1,0,0)^T$, then both expressions are equal:

$$\mathbf{H}\mathbf{e} = \mathbf{H}'\mathbf{e}' = (1,0,0)^T \quad (9.29)$$

And so, expression (9.28) becomes:

$$\mathbf{H} = \mathbf{H}'(\mathbf{M} + \mathbf{e}'\mathbf{v}^T) = \mathbf{H}'\mathbf{M} + \mathbf{H}'\mathbf{e}'\mathbf{v}^T = \left(\mathbf{I} + \mathbf{H}'\mathbf{e}'\mathbf{v}^T(\mathbf{H}'\mathbf{M})^{-1}\right)\mathbf{H}'\mathbf{M} \quad (9.30)$$

Because $\mathbf{v}$ can be any 3-vector, we can safely assume that:

$$\mathbf{a}^T = \mathbf{v}^T(\mathbf{H}'\mathbf{M})^{-1} \quad (9.31)$$

where $\mathbf{a}$ is also any 3-vector. This way expression (9.30) becomes equal to expression (9.16) and the proof is finished.

Once we know the expressions of the matched transformations $\mathbf{H}$ and $\mathbf{H}'$ we can take into account (9.30) and so $\mathbf{H}$ becomes:

$$\mathbf{H} = \left(\mathbf{I} + (1,0,0)^T \mathbf{a}^T\right)\mathbf{H}'\mathbf{M} = \mathbf{H}_A \mathbf{H}'\mathbf{M} = \mathbf{H}_A \mathbf{H}_0 \quad (9.32)$$

where $\mathbf{H}_A$ has the following form:

$$\mathbf{H}_A = \begin{bmatrix} a & b & c \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (9.33)$$

where it can be observed that (9.33) is an affine transformation, and $\mathbf{H}_0$ is,

$$\mathbf{H}_0 = \mathbf{H}'\mathbf{M} \quad (9.34)$$

We can now rewrite the minimization expression (9.15) as:

$$\sum_i d(\mathbf{H}_A \hat{\mathbf{x}}_i, \hat{\mathbf{x}}'_i)^2 \quad (9.35)$$

where $\hat{\mathbf{x}}_i = \mathbf{H}_0 \mathbf{x}_i$ and $\hat{\mathbf{x}}'_i = \mathbf{H}' \mathbf{x}'_i$. This way the minimization problem in (9.35) becomes a problem where $\mathbf{H}_A$ needs to be estimated.

In particular, we can assume that $\hat{\mathbf{x}}_i = (\hat{x}_i, \hat{y}_i, 1)^T$ and $\hat{\mathbf{x}}'_i = (\hat{x}'_i, \hat{y}'_i, 1)^T$, where points $\hat{\mathbf{x}}_i$ and $\hat{\mathbf{x}}'_i$ can be directly obtained from the matched points $\mathbf{x}_i \leftrightarrow \mathbf{x}'_i$ because $\mathbf{H}'$ and $\mathbf{M}$ are already known. Indeed, $\mathbf{H}'$ is obtained following the procedure in section 9.2.1.1 and $\mathbf{M}$ can be obtained from the decomposition of the Fundamental Matrix $\mathbf{F}$ shown in (9.22) once the epipole has been computed. Recall that the epipole $\mathbf{e}'$ can directly be obtained from the Fundamental Matrix as the nullspace from the transpose of $\mathbf{F}$, just as follows:

$$\mathbf{F}^T = \mathbf{U}\mathbf{S}\mathbf{V}^T \quad (9.36)$$

where (9.36) shows the singular value decomposition of the transpose of $\mathbf{F}$

$$\mathbf{e}' = \mathbf{V}_3 \quad (9.37)$$

and hence, the epipole on the right image is the last (i.e third) column of $\mathbf{V}$.

Notice that the epipole on the left image can be obtained in a similar way but without transposing $\mathbf{F}$ on (9.36)

Then $\mathbf{M}$ can be obtained as:

$$\mathbf{M} = \left([\mathbf{e}']_{\times} \right)^{-1} \mathbf{F} \quad (9.38)$$

Taking all this into account, then expression (9.35) can be rewritten as:

$$\sum_i (a\hat{x}_i + b\hat{y}_i + c - \hat{x}'_i)^2 + (\hat{y}_i - \hat{y}'_i)^2 \quad (9.39)$$

Since $(\hat{y}_i - \hat{y}'_i)^2$ is constant, then (9.39) can be further simplified to minimize the following expression:

$$S = \sum_i (a\hat{x}_i + b\hat{y}_i + c - \hat{x}'_i)^2 \quad (9.40)$$

Expression (9.40) can be solved by means of a simple linear least squares. Indeed, we can observe that expression (9.40) is minimized when its gradient with respect to each parameter (a , b , c) is equal to zero (i.e. the partial derivatives respect the parameters of interest are zero). Hence,

$$\begin{aligned} \frac{\partial S}{\partial a} &= 2 \sum_i (a\hat{x}_i + b\hat{y}_i + c - \hat{x}'_i) \cdot \hat{x}_i = 0 \\ \frac{\partial S}{\partial b} &= 2 \sum_i (a\hat{x}_i + b\hat{y}_i + c - \hat{x}'_i) \cdot \hat{y}_i = 0 \\ \frac{\partial S}{\partial c} &= 2 \sum_i (a\hat{x}_i + b\hat{y}_i + c - \hat{x}'_i) = 0 \end{aligned} \quad (9.41)$$

By rearranging the normal equations (9.41) we obtain:

$$\begin{aligned} \sum_i (a\hat{x}_i^2 + b\hat{x}_i\hat{y}_i + c\hat{x}_i) &= \sum_i \hat{x}'_i \hat{x}_i \\ \sum_i (a\hat{x}_i + b\hat{y}_i^2 + c\hat{y}_i) &= \sum_i \hat{x}'_i \hat{y}_i \\ \sum_i (a\hat{x}_i + b\hat{y}_i + c) &= \sum_i \hat{x}'_i \end{aligned} \quad (9.42)$$

If the parameters are arranged in a vector as shown below:

$$\boldsymbol{\beta} = (a, b, c)^T \quad (9.43)$$

The samples are expressed in a matrix as follows:

$$\mathbf{X} = \begin{bmatrix} \hat{x}_1 & \hat{y}_1 & 1 \\ \hat{x}_2 & \hat{y}_2 & 1 \\ \vdots & \vdots & \vdots \\ \hat{x}_i & \hat{y}_i & 1 \\ \vdots & \vdots & \vdots \end{bmatrix} \quad (9.44)$$

And the coordinates to compare to in a vector:

$$\mathbf{y} = \begin{bmatrix} \hat{x}'_1 \\ \hat{x}'_2 \\ \vdots \\ \hat{x}'_i \\ \vdots \end{bmatrix} \quad (9.45)$$

Then it can be observed that equations in (9.42) can be expressed as:

$$(\mathbf{X}^T \mathbf{X}) \boldsymbol{\beta} = \mathbf{X}^T \mathbf{y} \quad (9.46)$$

And consequently the parameters in (9.43) can be solved as:

$$\boldsymbol{\beta} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y} \quad (9.47)$$

where $\mathbf{X}^\dagger = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T$ is the Moore-Penrose pseudo-inverse.

9.2.1.3.- Constraints

Up to this point we have seen how Hartley's image rectification works. However there is still one issue to mention. In order to compute the Jacobian in (9.9) we had to use a Taylor approximation in (9.7) which is only valid when $|x / f| < 1$. In particular this constraint refers to an epipole that has been translated to the infinity respect to the origin (usually located at the center of the image by means of a transformation) and that also has been rotated to lay on the x-axis.

For any particular epipole, constraint in (9.7) means that if the epipole lays within image limits (or very close to them), then for rectified points close to the image boundaries, we lay on or very close to the singularity in (9.5), which invalidates Taylor approximation in (9.7) and hence the Jacobian in (9.9). This indeed translates into high projective distortions on rectified images, which invalidates the whole rectification process because no matching algorithm produces satisfactory results with such degree of distortion.

In Layman terms, the main issue with Hartley's rectification is that it only produces satisfactory results when the epipoles are located far from image boundaries, otherwise a high degree of distortion appears. Hence, if we are working on a controlled environment where camera calibration is known beforehand, then Hartley's rectification can usually be used safely if cameras are properly located. However, when working with uncalibrated sets of cameras where motion is unknown (i.e. a video sequence recorded with a hand-held camera), this fact might suppose a big problem to be solved, as the epipoles locations are unknown a priori and depending on the motion of the sequence the whole rectification process might fail.

On fig. 9.2.c and 9.2.d we show examples of Hartley's rectification of pair of images shown on fig. 9.2.a and 9.2.b. Pair of images on this figure follow a near parallel camera setup, and hence epipoles lay outside of image boundaries (they are not



Fig. 9.2. (a) and (b) show original pair of images where their epipoles lay outside of image boundaries. (c) and (d) show their corresponding Hartley's rectification. (e) and (f) show their polar rectification.



Fig. 9.3. (a) and (b) show a pair of images following a frontal translation motion, where as usually, epipoles lay inside of images (marked in red on the bushes near the leftmost bicycle).

depicted). Because of that Hartley's rectification can be successfully computed without large amounts of distortion.

On the other hand fig. 9.3 shows a nearly frontal motion camera setup, which makes the epipoles lay within image boundaries (marked with red dots). Because of this fact, Hartley's rectification cannot be computed without large amounts of distortion. Indeed, on fig. 9.4.a and 9.4.b we can observe that Hartley's rectification has failed as the rectified images appear split in two, because of the plane at infinity, with large amounts of distortion, so that image contents cannot be recognized any longer, and hence making them unsuitable for stereo correspondence.

As it can be observed, on hand-held recorded sequences where motion is not controlled, it can be very hard to ensure that Hartley's rectification will always be valid. For that reason, other rectification methods can be used as shown next.

9.2.2.- Pollefe's polar rectification

Because of the limitations on the rectification method shown on previous section whenever epipoles lay close or within image limits, research has focused on finding methods that do not impose this kind of restriction, which is especially useful when working on non-controlled environments.

One interesting method is the one proposed in [7, 42] which is based on polar rectification of a pair of images for any possible camera motion and obtaining minimal image sizes on rectified images without any pixel loss. This method, that will be shown next, only requires knowledge of the oriented fundamental matrix, which can be estimated using any robust method such as the ones explained in [5, 7, 8], and consists on using a polar parameterization of the image around the epipole.

9.2.2.1.- Oriented geometry

Recalling that any line in one view can be transferred to another view by the following expression:

$$\mathbf{l}' \sim \mathbf{H}^T \mathbf{l} \quad (9.48)$$

which can also be rewritten as

$$\mathbf{l} \sim \mathbf{H}^T \mathbf{l}' \quad (9.49)$$

where $\mathbf{l}$ is a line in the left view, $\mathbf{l}'$ is the corresponding line in the right view and $\mathbf{H}$ is the transfer matrix defined up to scale. We must observe that by comparing (9.14) and (9.48), both are equivalent by assuming that $\mathbf{H}'$ is the identity.

Since we also know that $\mathbf{F} = [\mathbf{e}']_{\times} \mathbf{M}$, which is equivalent to saying that $\mathbf{M} = [\mathbf{e}']_{\times} \mathbf{F}$ and by using expression (9.30)

where $\mathbf{H}'$ is in this case the identity, we obtain that expression (9.30) can be expressed now as:

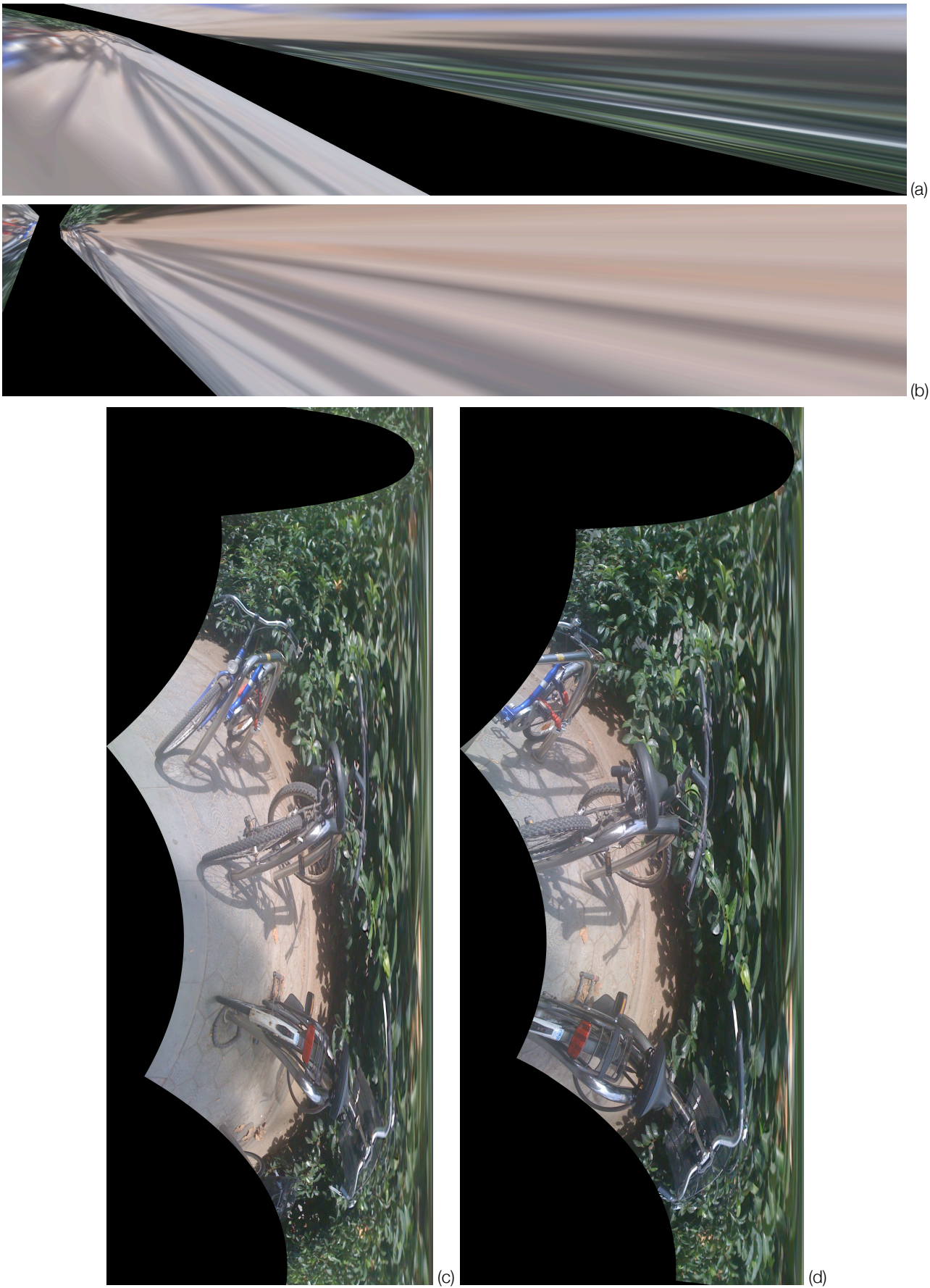


Fig. 9.4. (a) and (b) show Hartley's rectification of images on fig. 9.3. (c) and (d) show their polar rectification.

$$\mathbf{H} = [\mathbf{e}']_{\times} \mathbf{F} + \mathbf{e}' \mathbf{v}^T \quad (9.50)$$

where $\mathbf{v}$ is any random 3-vector for which $\det(\mathbf{H}) \neq 0$ so that $\mathbf{H}$ can be inverted.

Taking into account oriented projective geometry [43, 44] we can retrieve the oriented epipole $\mathbf{e}'$ thanks to its cheirality and consequently the oriented Fundamental Matrix $\mathbf{F}$ as shown above. Once the epipolar geometry is oriented, an epipolar line is obtained as:

$$\mathbf{l}' = \mathbf{F} \mathbf{m} = \mathbf{e} \times \mathbf{m} = [\mathbf{e}]_{\times} \mathbf{m} \quad (9.51)$$

which for the other view becomes,

$$\mathbf{l} = \mathbf{F}^T \mathbf{m}' = \mathbf{e}' \times \mathbf{m}' = [\mathbf{e}']_{\times} \mathbf{m}' \quad (9.52)$$

where $\mathbf{l}$ and $\mathbf{l}'$ are the epipolar lines for left and right view, $\mathbf{F}$ is the Fundamental Matrix and $\mathbf{m}$ or $\mathbf{m}'$ are points in the left or right view respectively. It must be noticed that since $\mathbf{F}$ is now oriented, epipolar lines become oriented as well, which allows us to define the following expression in order to separate the image plane into a positive and a negative region:

$$f_1(\mathbf{m}) = \mathbf{l}^T \mathbf{m} \quad \text{with} \quad \mathbf{m} = [x \ y \ 1]^T \quad (9.52)$$

And analogously for the other view:

$$f_{1'}(\mathbf{m}') = \mathbf{l}'^T \mathbf{m}' \quad \text{with} \quad \mathbf{m}' = [x' \ y' \ 1]^T \quad (9.53)$$

where expressions (9.52) and (9.53) should be zero for all $\mathbf{m}$ and $\mathbf{m}'$ points laying on their corresponding epipolar lines $\mathbf{l}$ and $\mathbf{l}'$, positive for all points located at one side of their corresponding epipolar lines, and negative for the other side, as depicted on fig. 9.5.

It must be noticed that, in epipolar geometry, for a pair of matching points $(\mathbf{m}, \mathbf{m}')$ both $f_1(\mathbf{m})$ and $f_{1'}(\mathbf{m}')$ should have the same sign. Consequently, since $\mathbf{l}'$ is obtained from $\mathbf{l}$ by (9.48), then the sign of the scale factor of $\mathbf{H}$ is determined. Once the sign is determined, the epipolar line transfer $\mathbf{H}$ becomes oriented as well, and both $\mathbf{l}$ and $\mathbf{l}'$ become oriented (as well as the epipoles).

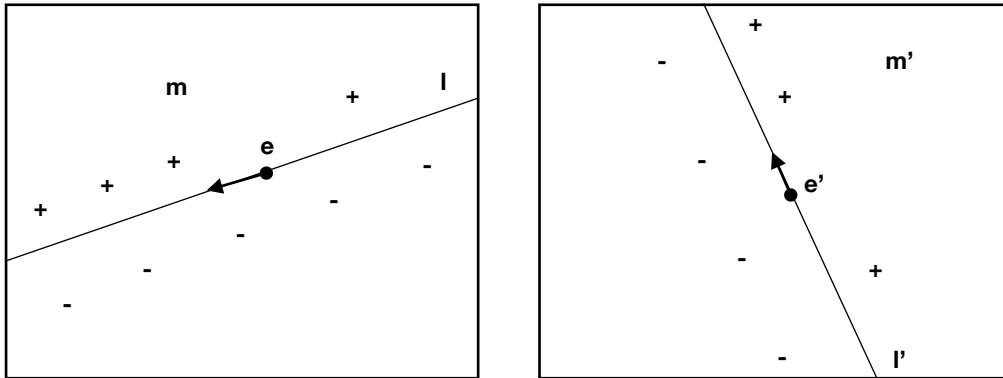


Fig. 9.5. Orientation of epipolar lines.

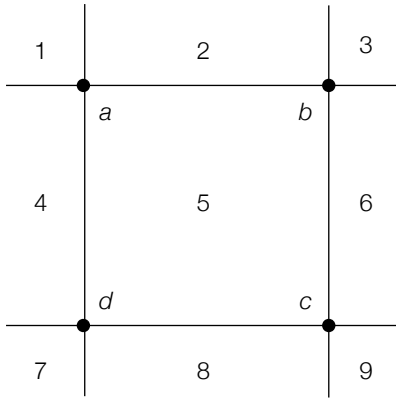


Fig. 9.6. Regions where epipoles can be located

Hence, in practical terms, $\mathbf{H}$ sign can be determined by choosing any pair of matched points $\mathbf{m}$ and $\mathbf{m}'$ and using expressions (9.52) and (9.53) on a pair of corresponding epipolar lines $\mathbf{l}$ and $\mathbf{l}'$, where $\mathbf{l}'$ has been obtained using (9.48) from any arbitrary epipolar line $\mathbf{l}$. If the signs of (9.52) and (9.53) are different, then $\mathbf{H}$ must be negated in order to obtain an oriented epipolar geometry, otherwise epipolar geometry is already oriented and $\mathbf{H}$ does not need to change its sign.

9.2.2.2.- Determining the common region

Because image rectification is used for stereo matching purposes, and only the common region between a pair of images can be matched; then only this common region is rectified in order to achieve the minimum size on output rectified images, the remaining parts will just be ignored as they won't be used for stereo correspondence.

In order to find the common region, first the image area must be divided into 9 regions as shown in fig. 9.6, where a, b, c, d are the coordinates of the corners of an image. Then the epipole, which is known from the fundamental matrix, may lie within the image limits if it is located on region 5, or outside of image boundaries if it lays in any of the other regions. Whenever the epipole lays outside of image boundaries, the extremal epipolar lines can be found as the epipolar lines that touch the outer image corners, as shown in figure 9.7, otherwise when epipole lies within region 5, any arbitrary epipolar line can be chosen as starting and ending angles. In this latter case, shown on figure 9.8, it is always good idea to choose the ending point on the neighbor epipolar line respect to the starting one so that a little more than 360 degrees is used during polar rectification, that way the inverse of the rectification can be obtained without losing information on the original image.

Whenever the epipole lies outside of image boundaries, once the extremal epipolar lines are found for an image in a pair of images, then they are transferred to the other view by using (9.48). Once the epipolar lines and the transferred epipolar lines are available, then the common region is found as the smallest region between the extremal epipolar lines and the transferred extremal epipolar lines that intersect image boundaries, as shown on fig. 9.7.

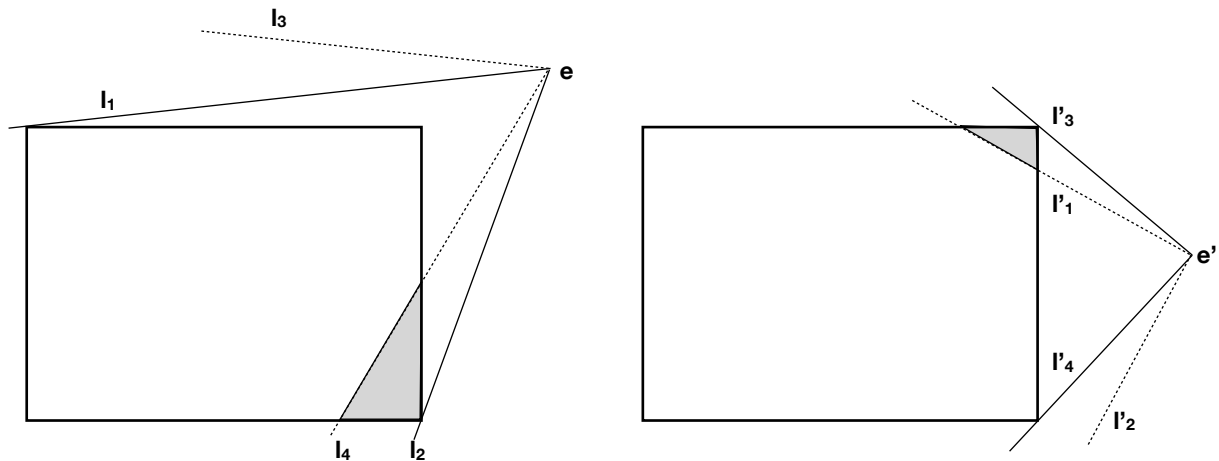


Fig. 9.7. Determination of the common region when the epipole lies outside image boundaries. The extreme epipolar lines $\mathbf{l}_1$, $\mathbf{l}_4$ and $\mathbf{l}'_1$, $\mathbf{l}'_4$ determine start and end angle on polar rectified images. $\mathbf{l}'_1$ and $\mathbf{l}'_4$ are obtained from $\mathbf{l}_1$ and $\mathbf{l}_4$ respectively by using (9.48). Notice that only common regions (in white) between left and right will be processed, the remaining parts (in gray) will be ignored.

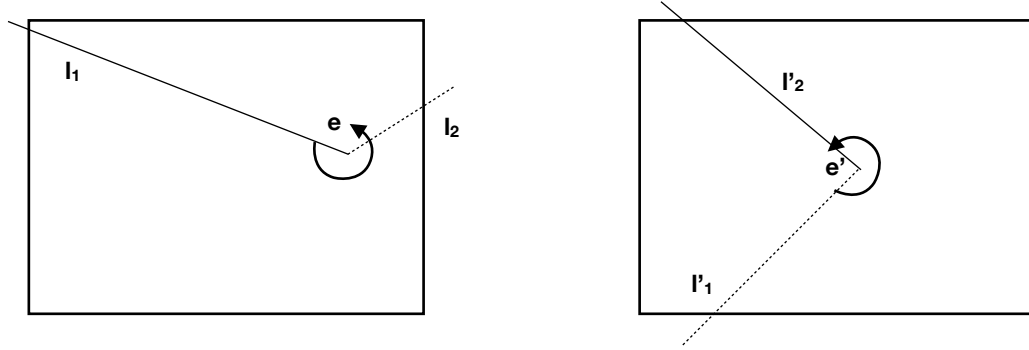


Fig. 9.8. Determination of the common region when the epipole lies within image boundaries. An arbitrary epipolar line l_1 is chosen as a start and end angle on polar rectified images so that 360 degrees are traversed on the polar rectified images. Then the epipolar line is transferred to the other view by using (9.48) and also used as the start and end angle. Notice that the whole image is rectified and no common region needs to be found.

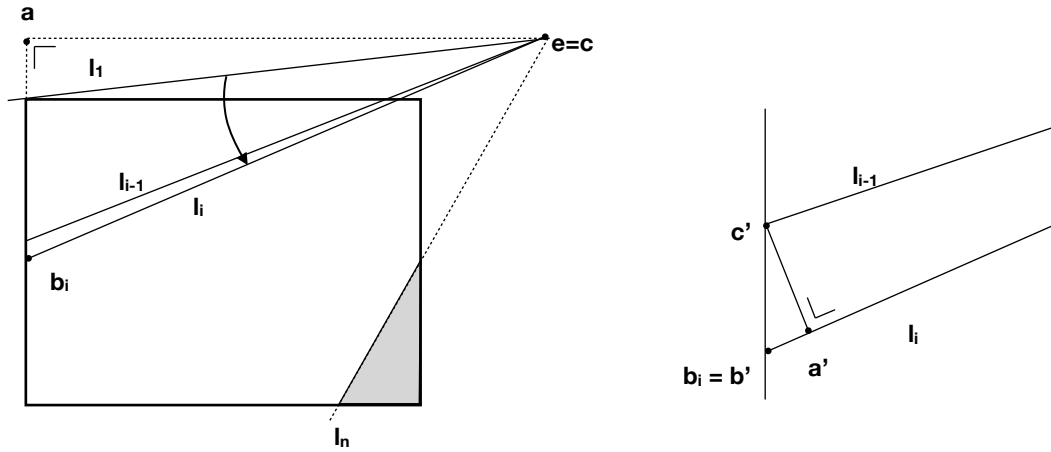


Fig. 9.9. Determination of maximum distance between consecutive epipolar lines. By using similarity between triangles abc and $a'b'c'$ we can demonstrate that in order to enforce a maximum separation of 1 pixel between consecutive epipolar lines, then segment $b'c'$ must be equal to bc/ac .

On the other hand, when the epipole lies within image boundaries, there is no need to search for a common region and the whole image is rectified, as shown on fig. 9.8.

9.2.2.3.- Determining the distance between epipolar lines

Once the common region defined by the extremal epipolar lines is known, the rectified images can be built line by line as a polar transformation from the epipole by traversing the epipolar lines longitudinally and choosing each new consecutive epipolar line to traverse by defining a maximum angular distance between consecutive traversed epipolar lines so that no image compression occurs. Notice that since epipolar lines are oriented, we follow their orientation direction in order to traverse them longitudinally.

To avoid image compression, we need to ensure that consecutive epipolar lines have an angular separation so that each pixel in an image is sampled at least once. In other words we need to ensure that the maximum allowed distance between consecutive epipolar lines within common region is of one pixel in the worst case, that way all pixels will be sampled at least once. This is depicted on figure 9.9, and it can be shown that the worst case is located in the opposite extrema of the image respect to the epipole, or in other words, where longitudinal distance on the epipolar line respect to the epipole is maximal.

On figure 9.9 and 9.10 we can observe the similarity between triangles abc and $a'b'c'$. More specifically by summing up the angles on the larger triangle abc , which we know that equals to π , we obtain:

$$\frac{\pi}{2} + \alpha + \beta = \pi \Rightarrow \beta = \frac{\pi}{2} - \alpha \quad (9.54)$$

On the other hand, the summation of the angles on the smaller triangle $a'b'c'$ is equal to:

$$\alpha' + \beta + \frac{\pi}{2} = \pi \Rightarrow \alpha' = \frac{\pi}{2} - \beta \quad (9.55)$$

And by combining (9.54) and (9.55) we obtain that both angles α and α' are equal:

$$\alpha' = \frac{\pi}{2} - \beta = \frac{\pi}{2} - \left(\frac{\pi}{2} - \alpha \right) \Rightarrow \alpha' = \alpha \quad (9.56)$$

Hence, we have proved that both triangles abc and $a'b'c'$ have equal angles α , β and $\pi/2$, which indicates that both triangles are similar (i.e. equal up to scale or rotation).

Hence, from the similarity of triangles abc and $a'b'c'$ we have the following property:

$$k = \frac{ac}{a'c'} = \frac{bc}{b'c'} = \frac{ab}{a'b'} \quad (9.57)$$

where k is the scale factor between both triangles, and ac , $a'c'$, bc , $b'c'$, ab and $a'b'$ are the lengths of the sides of both triangles.

From (9.57) we want to recover $b'c'$ as the maximum allowed separation between consecutive epipolar lines by imposing that at least each pixel is sampled once in the worst case ($a'c' = 1$). Hence, this can be computed as:

$$b'c' = \frac{bc}{ac} \quad (9.58)$$

Notice that by choosing the maximum allowed distance between consecutive epipolar lines ($a'c'$) as 1 pixel, we ensure minimum output size on rectified images. However, imposing this limit on each consecutive epipolar line makes that angular separation on output rectified images doesn't stay constant; instead angular separation will change between consecutive

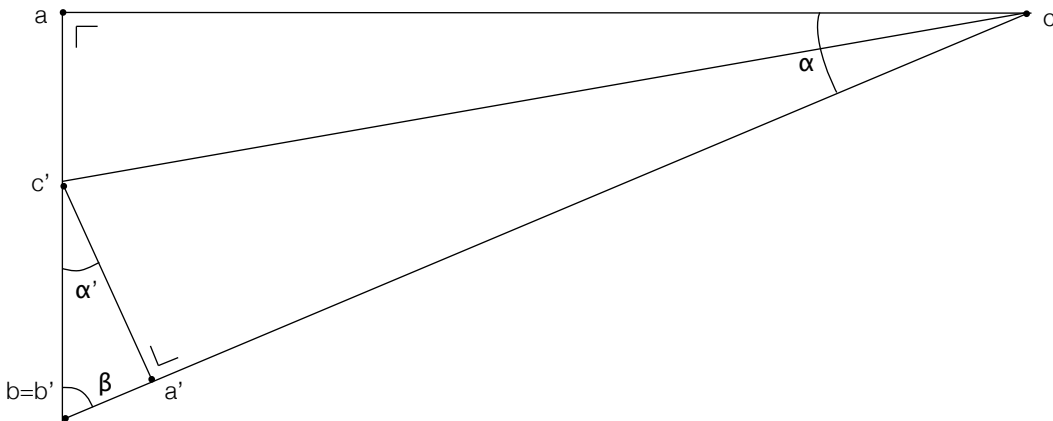


Fig. 9.10. Similarity of triangles abc and $a'b'c'$. Note: depiction is exaggerated so that similarity can easily be observed.

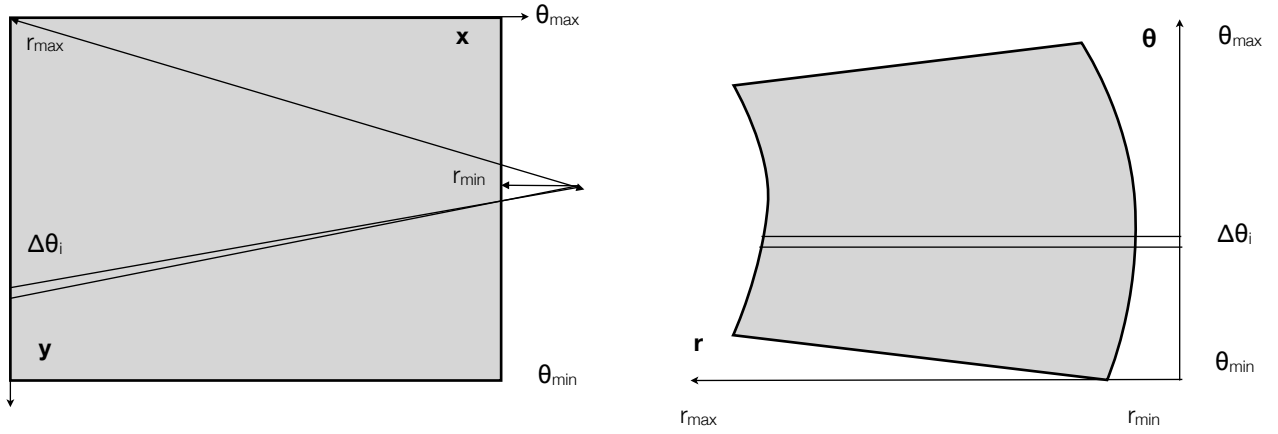


Fig. 9.11. Image transformation from (x,y) space to polar (r,θ) space.

epipolar lines in order to achieve minimum output size. This fact requires to save each angular separation in a lookup table so that the original image can be recovered when undoing the rectification process.

If on the other hand constant angular separation is desired to avoid the burden of storing a lookup table, then this separation can be computed for the worst case in the image (i.e. where longitudinal distance to epipole is maximal) at the expense of obtaining slightly bigger rectified images.

9.2.2.4.- Building the rectified images

The process shown above allows us to rectify an image in one view by building a polar transformation row by row, as shown on figure 9.11. However, in order to also rectify the other view and make epipolar lines coincident, then each traversed consecutive epipolar line on the first view $\mathbf{l}_i$ must be transferred to the other view as $\mathbf{l}'_i$ so that it can also be traversed longitudinally to obtain another polar rectification.

Notice as well that to ensure that output size is minimal on the longitudinal direction, each epipolar line is traversed longitudinally without compressing the images by sampling on each pixel and following the same orientation on both images so that epipolar lines remain coincident on rectified output.

Because we already know the separation between consecutive epipolar lines (which has been stored in a look-up table). It only remains to traverse epipolar lines longitudinally in the correct direction on both images.

First of all it will be assumed that epipolar lines have the correct orientation as shown on fig. 9.5 and that they are properly transferred by $\mathbf{H}$ on (9.50) preserving their orientation. Once this is done, epipolar lines will be traversed longitudinally by using a signed distance function as the one defined below:

$$d(x,y) = \frac{ax + by + c}{\sqrt{a^2 + b^2}} \quad (9.59)$$

where expression (9.59) defines the smallest signed perpendicular distance from a given point $\mathbf{p} = (x,y)^T$ to a given line $\mathbf{l} = (a,b,c)^T$. Sign on this expression only indicates whether point $\mathbf{p}$ is located at one side or the other of line $\mathbf{l}$. Indeed, we can observe that this expression is proportional to expression (9.52) where geometry orientation is defined. It must also be observed that rectified images will be expressed in terms of this signed distance, as shown on the horizontal axis of rectified image on fig. 9.11.

On the other hand, because epipolar lines are oriented, then their orthogonal counterparts will preserve their orientation as well. Then, given any epipolar line $\mathbf{l} = (a,b,c)^T$ passing through epipole $\mathbf{e} = (x_0, y_0)^T$, we can define its orthogonal version $\mathbf{l}'$ passing through the same epipole as:

$$\mathbf{l}' = (a', b', c')^T \quad (9.60)$$

where a' and b' can be defined as:

$$\begin{aligned} a' &= -b \\ b' &= a \end{aligned} \quad (9.61)$$

to ensure that the slope is perpendicular to line $\mathbf{l}$ (this can be proved by ensuring that the product of the slopes of both lines is -1). Then, to also ensure that line $\mathbf{l}'$ passes through epipole $\mathbf{e}$ the following must hold:

$$\mathbf{l}'^T \cdot \mathbf{e} = a'x_0 + b'y_0 + c' = 0 \quad (9.62)$$

and consequently, c' can be expressed as:

$$c' = -a'x_0 - b'y_0 = bx_0 - ay_0 \quad (9.63)$$

Then, once the orthogonal epipolar line $\mathbf{l}'$ passing through epipole $\mathbf{e}$ is defined as in (9.61) and (9.63), we can find any point $\mathbf{p}_d = (x_d, y_d)$ laying on epipolar line $\mathbf{l}$ at a given signed distance d from epipole $\mathbf{e}$ by using expression (9.59) over line $\mathbf{l}'$:

$$a'x_d + b'y_d = d\sqrt{a'^2 + b'^2} - c' \quad (9.64)$$

and taking into account that point $\mathbf{p}_d$ also lies on epipolar line $\mathbf{l}$ the following expression also holds:

$$ax_d + by_d = -c \quad (9.65)$$

Then, by combining (9.64) and (9.65) point $\mathbf{p}_d$ can be obtained by solving the following linear system of equations:

$$\begin{aligned} \mathbf{A} \cdot \mathbf{p}_d &= \mathbf{b} \\ \begin{bmatrix} a' & b' \\ a & b \end{bmatrix} \begin{bmatrix} x_d \\ y_d \end{bmatrix} &= \begin{bmatrix} d\sqrt{a'^2 + b'^2} - c' \\ -c \end{bmatrix} \end{aligned} \quad (9.66)$$

This process is repeated for each desired signed distance in order to find the coordinates of point $\mathbf{p}_d$ so that the whole epipolar line is traversed, as shown on fig. 9.12. Also, the same process can be repeated to traverse epipolar lines transferred to the second view.

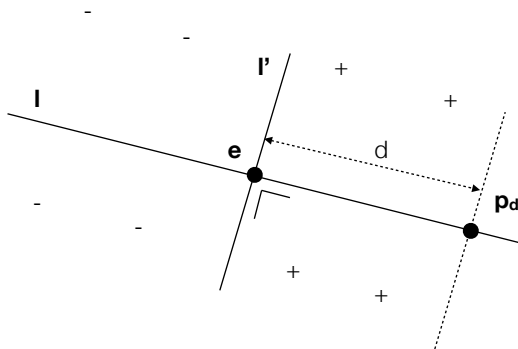


Fig. 9.12. Finding point $\mathbf{p}_d$ laying on epipolar line $\mathbf{l}$ at a given signed distance d

9.2.2.5.- Hartley's vs polar rectification

On figures 9.2.e, 9.2.f, 9.4.c and 9.4.d we can observe the results of polar rectification.

On the first case on fig. 9.2, we can see that since epipoles lay outside of image boundaries, both Hartley's and polar rectification work as expected. However, even in this situation, because epipoles probably lay too close to image boundaries, it can be seen that Hartley's result has a larger amount of distortion than the result provided by Pollefe's polar method. Hence, polar rectification will probably yield better results for stereo matching in this case.

Besides, on the second case shown on fig. 9.3, Hartley's method simply fails and results in an extremely large image with large amounts of distortion. Indeed we observe that the rectified image simply gets split in two and all image details get lost, which makes stereo correspondence an impossible task in this situation. On the other hand, polar rectification can successfully cope with this situation by providing a rectified pair of images, that although they have evident distortion (because of polar transformation), is completely valid for stereo correspondence. Indeed, objects can correctly be identified, specially if they lay far from the epipole. Consequently it can be deduced that in most situations Pollefe's polar rectification will provide better results.

As a last comment, it must be noticed that polar rectification only creates a large amount of distortion on areas surrounding the epipole (due to oversampling). Indeed on fig. 9.4.c and 9.4.d we can observe that the rightmost areas, which are the closest to the epipoles, become largely smoothed, and hence will be difficult to be matched by any stereo matching algorithm. However, this fact is not really important either, because due to numerical precision, points located close to the epipole cannot be properly triangulated because they lay too close to the baseline even if the correct stereo correspondence was available.

9.3.- Cost function

When attempting to find the best match between any given set of points on a pair of images, usually match scores are used (see chapter 7), and the best match is picked among each pair of points as the correct match, however, when other factors are taken into account, such as the piece-wise smoothness of disparity, or epipolar restrictions, this is not enough. On those cases a cost function is defined that takes into account all restrictions and matching scores. On such cases, the objective is to define a cost function that can easily be minimized (or maximized, depending on the cost function meaning) over the set of horizontal and vertical disparity values in order to perfectly match each pixel between a given pair of images.

9.3.1.- Grey value constancy assumption

Indeed, a cost function is defined because it is desired to match a pair of images as follows:

$$I_1(x, y) \approx I_2(x + u(x, y), y + v(x, y)), \quad \forall (x, y) \in \mathbb{R}^2 \quad (9.67)$$

where I_1 refers to the first view for any pixel located at (x, y) , I_2 refers to the second view for any pixel located at provided coordinates (x, y) , u refers to the horizontal disparity at coordinates (x, y) and analogously v refers to the vertical disparity at coordinates (x, y) . Indeed, expression (9.67) attempts to match the second view to the first one by using appropriate displacements $u(x, y)$ and $v(x, y)$ between any pair of pixels. Hence, it can be observed that in order to find the correspondence between a pair of images defined as,

$$h(x, y) = (u(x, y), v(x, y)) \quad (9.68)$$

we will later define a cost function E to be minimized. It must also be observed that expression (9.67) holds as long as the *grey value constancy assumption* is true [23, 38, 45]. In other words, it is assumed that the intensity value of a pixel is not changed by the displacement. We can observe that this assumption will hold as long as the motion between pairs of images is mainly formed by translations. However, when other kind of motions exist (i.e. rotations), then this assumption usually fails as projective transformations due to changes in perspective appear between pairs of corresponding pixels.

Hence, as a general rule whenever we work with consecutive frames on video sequences, changes in perspective are so small that the grey value constancy assumption holds and consequently (9.67) remains valid. On the other hand, when working with still images, if the baseline between the pair of images to be matched is too big, then the grey value constancy assumption is no longer valid and (9.67) fails, making stereo correspondence a hard problem to be solved.

We can rewrite expression (9.67) so that I_1 and I_2 are expressed in terms of time as consecutive frames on a video sequence, so that:

$$\begin{aligned}
I_1(x, y) &= I(x, y, t) \\
I_2(x, y) &= I(x, y, t + \delta t)
\end{aligned} \tag{9.69}$$

And then expression (9.67) becomes:

$$I(x, y, t) = I(x + \delta x, y + \delta y, t + \delta t) \tag{9.70}$$

where δt , which is usually $\delta t = 1$, is an integer value indicating the separation between frames on the first and second views, and δx , δy are horizontal and vertical displacements which are related to the the optical flow to be found as expressed in (9.68) .

By expanding the right side of expression (9.70) using Taylor series, which implies that image data is differentiable (i.e. does not have abrupt changes), we obtain:

$$I(x, y, t) = I(x, y, t) + \delta x \frac{\partial I(x, y, t)}{\partial x} + \delta y \frac{\partial I(x, y, t)}{\partial y} + \delta t \frac{\partial I(x, y, t)}{\partial t} + \epsilon \tag{9.71}$$

where ϵ contains second and higher order terms in Taylor series, which are considered as errors along with other errors due to image sampling and quantization. Hence, after subtracting $I(x, y, t)$ from both sides and dividing by δt we obtain:

$$\frac{\delta x}{\delta t} \frac{\partial I(x, y, t)}{\partial x} + \frac{\delta y}{\delta t} \frac{\partial I(x, y, t)}{\partial y} + \frac{\partial I(x, y, t)}{\partial t} + o(\delta t) = 0 \tag{9.72}$$

Assuming that time variation is small enough, indeed on the limit $\delta t \rightarrow 0$, then the term error term $o(\delta t)$ can be neglected. Hence expression (9.72) becomes:

$$\frac{\partial x}{\partial t} \frac{\partial I(x, y, t)}{\partial x} + \frac{\partial y}{\partial t} \frac{\partial I(x, y, t)}{\partial y} + \frac{\partial I(x, y, t)}{\partial t} = 0 \tag{9.73}$$

where the horizontal and vertical terms of the optical flow on (9.68) are:

$$\begin{aligned}
u(x, y) &= \frac{\partial x}{\partial t} \\
v(x, y) &= \frac{\partial y}{\partial t}
\end{aligned} \tag{9.74}$$

By using the *chain rule for differentiation*, It can then be seen that expression (9.73) is equivalent to saying:

$$\frac{\partial I}{\partial t} = 0 \tag{9.75}$$

Indeed expression (9.75) is the *grey value constancy assumption*, which assumes that the brightness or intensity of a particular point in an image is constant, which as we have proved, is only valid for small time-space displacements between image pairs.

Then, using together (9.73) and (9.74) we can obtain a linearized version of the *grey value constancy assumption* which is called the *optical flow constraint*:

$$(I_x, I_y) \cdot (u, v) = -I_t \quad (9.76)$$

where subindices indicate partial derivatives along a given direction (horizontal, vertical or time-space).

By approximating partial derivative along consecutive frames as their difference, then expression (9.76) can also be written as:

$$\nabla I_1(x, y) \cdot h(x, y) = I_1(x, y) - I_2(x, y) \quad (9.77)$$

Hence the *optical flow constraint* can be used to numerically solve the linear system of equations shown in (9.77) in order to obtain the optical flow between a pair of images as long as the *grey value constancy assumption* holds.

However, it must be noticed that the *optical flow constraint* seen so far is quite susceptible to bright variations, and hence usually fails whenever displacements between pairs of images are too big. For that reason, usually the non-linearized expression in (9.67) can be used as well.

9.3.2.- Gradient constancy assumption

As we have seen on previous sub-section, images can be directly matched by using their intensity values on each pixel. However, this measure can be sensitive to small bright variations. For that reason, gradient data, which has the DC component removed, can also be matched in a similar way by means of the *gradient constancy assumption*:

$$\nabla I(x, y, t) = \nabla I(x + \delta x, y + \delta y, t + \delta t) \quad (9.78)$$

Hence, expression (9.78) gives us three more equations (one for each partial derivative over x, y and t).

Again, following the same methodology as in previous subsection we can linearize the *gradient constancy assumption* by expanding the right side of (9.78) using Taylor series. This is shown for the partial derivative over x:

$$\begin{aligned} \frac{\partial}{\partial x} I(x, y, t) &= \frac{\partial}{\partial x} I(x + \delta x, y + \delta y, t + \delta t) = \\ \frac{\partial}{\partial x} I(x, y, t) &+ \delta x \frac{\partial^2}{\partial x^2} I(x, y, t) + \delta y \frac{\partial^2}{\partial x \partial y} I(x, y, t) + \delta t \frac{\partial^2}{\partial x \partial t} I(x, y, t) + \varepsilon \end{aligned} \quad (9.79)$$

And after subtracting the partial derivative over x of image $I(x, y, t)$ on both sides and dividing by δt we obtain:

$$\frac{\delta x}{\delta t} \frac{\partial^2}{\partial x^2} I(x, y, t) + \frac{\delta y}{\delta t} \frac{\partial^2}{\partial x \partial y} I(x, y, t) + \frac{\partial^2}{\partial x \partial t} I(x, y, t) + o(\delta t) = 0 \quad (9.80)$$

And because we are assuming that time-space displacements are small enough ($\delta t \rightarrow 0$), then expression (9.80) finally becomes:

$$\frac{\partial x}{\partial t} \frac{\partial^2}{\partial x^2} I(x, y, t) + \frac{\partial y}{\partial t} \frac{\partial^2}{\partial x \partial y} I(x, y, t) + \frac{\partial^2}{\partial x \partial t} I(x, y, t) = 0 \quad (9.81)$$

And again we can use (9.74) and (9.81) to obtain a more compact representation as:

$$(I_{xx}, I_{xy}) \cdot (u, v) = -I_{xt} \quad (9.82)$$

where subindices indicate second order partial derivatives on directions xx, xy and xt.

This process can also be repeated for the remaining partial derivatives on (9.78) and we finally obtain the following three compact equations:

$$\begin{aligned} (I_{xx}, I_{xy}) \cdot (u, v) &= -I_{xt} \\ (I_{yx}, I_{yy}) \cdot (u, v) &= -I_{yt} \\ (I_{tx}, I_{ty}) \cdot (u, v) &= -I_{tt} \end{aligned} \quad (9.83)$$

Again, it must be noticed that the *gradient constancy assumption* is only valid for small displacements, otherwise expression (9.78) should be used.

However, if both grey value constancy assumption and gradient constancy assumptions are assumed to be valid, it can be easily seen that the desired **disparity can be linearly obtained by solving the overdetermined linear system of equations** over (u,v) for a given pixel (x,y) using expressions (9.76) and (9.83). Because this method can be significantly unstable if gradient and grey value constancy assumptions are not valid, then the disparity values obtained using this method can simply be used as an initialization value to minimize a given cost function E that will be shown next.

9.3.3.- Defining an energy functional

As we have seen, when the grey value and gradient constancy assumptions fail, we need to use expressions (9.67) and (9.78). This expressions can be rewritten in a straightforward way to obtain an energy functional that penalizes deviations from these model assumptions. Consequently this can be seen as a cost function that needs to be minimized.

$$E(u, v) = E_{Data}(u, v) = \int_{\Omega} \left(|I_2(\mathbf{x} + \mathbf{h}) - I_1(\mathbf{x})|^2 + \gamma |\nabla I_2(\mathbf{x} + \mathbf{h}) - \nabla I_1(\mathbf{x})|^2 \right) d\mathbf{x} \quad (9.84)$$

where Ω represents a neighborhood around each pixel $\mathbf{x} = (x, y)^T$ and $\mathbf{h} = (u, v)^T$ represents the optical flow to be found, and γ is a weight value between both the grey value and the gradient constancy assumptions. It can be observed that expression (9.84) can be seen as the weighted sum of two SAD matching distances between image intensities and their gradients as seen on expression (7.2):

$$E(u, v) = E_{Data}(u, v) = D_{SAD}(I_1, I_2) + \gamma D_{SAD}(\nabla I_1, \nabla I_2) \quad (9.85)$$

Consequently, the energy functional is directly obtained from image data, and hence the subindex data. Also, it can be seen that aside from a SAD matching distances, any other distance metrics shown in chapter 7 can be used as well, as long as the desired optical flow values can be obtained by minimizing the provided functional, Consequently expression (9.85) could also be written as:

$$E_{Data}(u, v) = D_{xx}(I_1, I_2) + \gamma D_{xx}(\nabla I_1, \nabla I_2) \quad (9.86)$$

where xx subindex stands for any distance metric. Notice also that the energy functional shown so far only takes into account image data. However, other constraints can be taken into account in order to simplify the minimization process. Consequently the energy functional can also be expressed as a weighted sum of functionals to be minimized:

$$E(u, v) = E_{Data}(u, v) + \alpha E_1(u, v) + \beta E_2(u, v) + \dots \quad (9.87)$$

where subindices stand for any of the terms of the cost function to be minimized, where at least one is due to image data, and the remaining ones are due to other restrictions such as smoothness or epipolar restrictions.

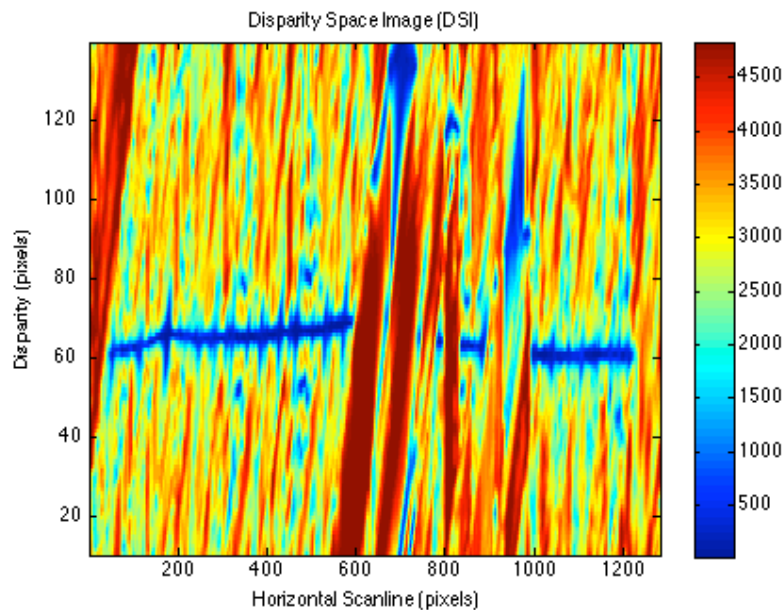


Fig. 9.13.- DSI over a scanline using SAD distance metric on the aloe Middlebury set [40]. The lower the DSI value the better the match.

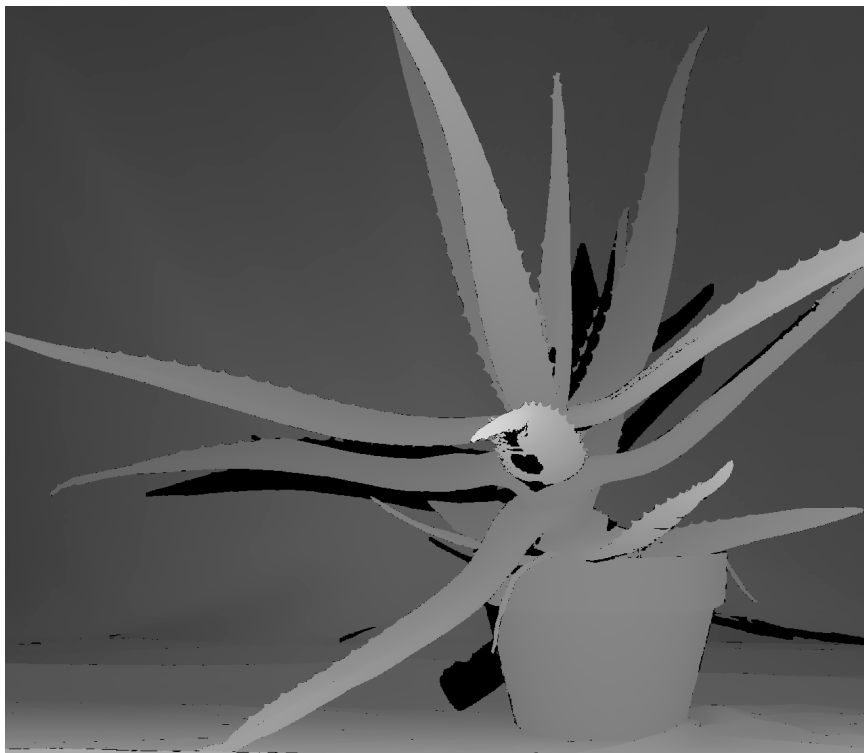


Fig. 9.14.- Disparity ground-truth on the aloe Middlebury set [40]

It can be seen that the chosen cost function E spans what is called the Disparity Space Image (DSI) over a given scanline (y coordinate constant) to be matched on a pair of rectified images as seen on figure 9.13. This space is an image because it's a 2D space that spans over the horizontal location at a given scanline and its associated disparity, in other words, if images are rectified only horizontal disparity is present, and the DSI depicts cost values over a scanline (constant y) for different x coordinates and horizontal disparities $u(x,y)$.

The DSI can also be interpreted over non-rectified images, but in this case the generated space is no longer 2D, instead it's a 3D space that spans over horizontal and vertical disparities ($u(x,y)$, $v(x,y)$) and the horizontal location x at a given scanline (constant y). Besides, for many global algorithms, the DSI is not minimized over a single scanline at a time, and instead the DS (Disparity Space) is considered over the whole image for any set of disparities at each pixel, in such cases the DS is now a 4D space that spans over horizontal and vertical disparities as well as the horizontal and vertical locations on the dense disparity map that must be minimized for the whole image (i.e.. global minimization). Finally, if robust estimation is also desired (i.e. by using a log-polar transform as seen on chapter section 7.9), the DS could then span a 6D space, where the 2 new dimensions would be related to rotation and scale of the log-polar transform [5].

As it can easily be seen, the DS or DSI can be very simple or very complex depending on the restrictions taken into account, however as a rule of thumb, the more dimensions the DS spans, the less likely and more computationally intensive will be to obtain reliable matches. Also it must be noticed that the obtained results will depend on the approach taken for minimization that yields the optimum dense disparity map. It will be seen that usually some properties are desired for the DS so that its minimization can be more easily and reliably computed, as will be explained on section 9.4.

On the following sections we will focus on typical restrictions that can be applied to the cost function, and the different expressions that such function can take.

9.3.4.- Smoothness restriction

By looking at a typical DSI like the one on figure 9.13, we can observe that disparity values usually have smooth variations inside object limits indicating small changes in depth, whereas abrupt variations appear at object borders indicating big changes on depth. As a result, usually objects can directly be identified on a dense disparity map by simple inspection as can be seen on figure 9.14. This is formally called piece-wise smoothness.

It is then desirable to impose certain restrictions so that piece-wise smoothness can be easily obtained from DS cost function minimization. The typical restrictions imposed for such purpose [2, 15, 21-28] imposes that optical flow $h(x,y) = (u(x,y), v(x,y))$ should remain as constant as possible, which implies that its gradient should be as small as possible as shown below:

$$E_{Smooth}(u, v) = \int_{\Omega} (|\nabla_3 u|^2 + |\nabla_3 v|^2) d\mathbf{x} \quad (9.88)$$

where $\nabla_3 = (\partial_x, \partial_y, \partial_t)$, and hence the smoothness term takes into account not only the horizontal and vertical optical flow variation, but also its variation along consecutive frames. This will lead, as will be seen, to iterative methods such as HRM that start from an initial coarse optical flow estimation, and iterate on consecutive frames until convergence to the true optical flow is found. This allows to estimate optical flow very fast on video sequences where displacements between consecutive frames are small.

However, expression (9.88) only imposes an smoothness restriction, but it does not ensure that disparity becomes piece-wise smooth. It can usually be assumed that the lower the gradient is, usually the lower the disparity variation is present; on the other side, object boundaries usually have large gradient components, and so in those areas greater disparity variation is present, making disparity appear as a piece-wise smooth representation of displacements. This criterion is usually expressed as follows:

$$E_{Piece-Smooth}(u, v) = \int_{\Omega} (|\nabla_3 u|^2 + |\nabla_3 v|^2) d\mathbf{x} - \delta \int_{\Omega} |\nabla I_1|^2 d\mathbf{x} \quad (9.89)$$

where the second term is weighted by a factor $\delta > 0$ accounting for the smoothness restriction relaxation whenever a high gradient on first (reference) view I_1 at pixel (x,y) is present. This way greater disparity variation is allowed at object boundaries, where image gradients are usually larger.

9.3.5.- Epipolar restriction

Besides smoothness restriction, another typical restriction that is usually taken into account is the epipolar restriction. The epipolar restriction comes from the equation that defines the Fundamental matrix [8]:

$$\mathbf{x}_R^T \cdot \mathbf{F} \cdot \mathbf{x}_L = 0 \quad (9.90)$$

where $\mathbf{x}_L$ accounts for matched points on left view, $\mathbf{x}_R$ accounts for matched points on right view (both using homogeneous coordinates) and $\mathbf{F}$ for the 3x3 rank-2 Fundamental matrix.

It can directly be observed from this equation that the product of a point on the left view by the fundamental matrix between a pair of views generates an epipolar line on the right view expressed as follows:

$$\mathbf{l}' = \mathbf{F} \cdot \mathbf{x}_L = (a', b', c')^T \quad (9.91)$$

Also a point on the right view generates a corresponding epipolar line on the left view as follows:

$$\mathbf{l} = \mathbf{F}^T \cdot \mathbf{x}_R = (a, b, c)^T \quad (9.92)$$

Hence, the fundamental matrix equation indeed tells that matched points on the right view lay on the epipolar line on the right view generated by its corresponding point on the left view. Oppositely, a point on the left view lays on the epipolar line on the same view generated by its corresponding point on the right view. Both of these assertions can be expressed as follows by joining together equations 9.90, 9.91 and 9.92:

$$\begin{aligned} \mathbf{l}'^T \cdot \mathbf{x}_R &= 0 \\ \mathbf{l}^T \cdot \mathbf{x}_L &= 0 \end{aligned} \quad (9.93)$$

Hence, once the fundamental matrix is known, it is trivial to compute epipolar lines for any given point. It must be noticed however, that estimation of the fundamental matrix is not always perfect (even when robust estimation is used), and some imprecision is present due to matching errors during estimation of the epipolar geometry. Due to this fact, matched points doesn't necessarily need to lay along epipolar lines, and instead due to errors, they will lay around epipolar lines generating what is known as **epipolar bands** [30], [31] and [32]. It must be noticed that epipolar bands follow an hyperbolic expression for a given amount of confidence of points laying inside, however, for practical purposes, they can be linearly approached simply as areas surrounding epipolar lines up to a given distance limit. Our practical experiments have shown that if fundamental matrix accuracy is around one pixel of epipolar distance, then epipolar bands 4 pixels wide (2 pixels on each direction respect to the epipolar line), will suffice to find most reliable matches with enough confidence.

We now know that matched pairs of points should occur throughout epipolar bands, more precisely they should lie as close as possible to epipolar lines. The requirement now is to compute the generated epipolar line from the reference view (usually left view) on the right view, and compute the distance of each possible match respect to its corresponding epipolar line as follows:

$$d_{\mathbf{x}_R} = \left| \frac{a' \cdot x' + b' \cdot y' + c'}{\sqrt{a'^2 + b'^2}} \right| \quad (9.94)$$

where epipolar line on the right view is $\mathbf{l}' = (a', b', c')^T$, and point to be matched on the right view is

$$\mathbf{x}_R = (x', y')^T = (x + u(x, y), y + v(x, y))^T \quad (9.95)$$

Once the epipolar distance is known it can be used to compute the epipolar restriction cost as:

$$E_{Epipolar}(u, v) = d_{x_R} \quad (9.96)$$

Notice that even though cost is defined for any given epipolar distance, in order to accelerate search over the DS, hard limits can be imposed on the maximum allowed epipolar distance to reduce the search area to the epipolar bands, since as we mentioned before we know that with a given confidence, matches will occur within those bands of a given width. It is then not necessary to check for all disparity values that are located too far away from the corresponding epipolar lines.

Hence, up to this point we have seen that the most complete cost function could be written as:

$$E(u, v) = E_{Data}(u, v) + \alpha E_{Piece-Smooth}(u, v) + \beta E_{Epipolar}(u, v) + \dots \quad (9.97)$$

But notice that is up to the reader to add as many restrictions on expression (9.97) as she feels is necessary to reliably estimate disparity values. Indeed, even further restrictions could be used as long as the DS becomes well-behaved for cost minimization using any of the algorithms that will be seen later.

9.3.5.1.- Epipolar restriction vs. image rectification

It can be observed now that image rectification is an equivalent way of imposing an epipolar restriction. The advantage is that since rectification is computed before DS minimization, the DS minimization does not need to take into account any epipolar information and only horizontal disparity values need to be used for DS minimization. Whereas on the other hand, algorithms that take into account epipolar restrictions, need to compute epipolar lines and epipolar distances for each pair of points to be checked for matching. This makes computations slightly slower, and hence **algorithms that work over rectified images are usually faster**.

However, it must be noticed that some rectification techniques (see section 9.2) can introduce large amounts of distortion, that translate into lots of wrong matches; and also it might happen that some images cannot be rectified at all if the epipole lies inside image boundaries. This is a big inconvenient, since when working with unknown camera calibration, this condition cannot be always satisfied. This is especially true when working on sequences where the camera is moved freely or translated frontally (e.g.. along a corridor), making epipoles accidentally lay inside image boundaries.

9.4.- DS minimization

We have seen now how the DS is generated for any given cost function taking into account a given set of restrictions. We need to know now how to obtain reliable matches once the cost function is known.

Usually this is done by means of minimizing the cost function using some minimization algorithm, as will be explained next. The main difference between choosing one algorithm or another relies basically on the quality of the results and the computational effort to compute dense matches. It will be seen that usually there is a trade off between quality and speed.

As will be discussed next and was shown on fig. 9.1, DS minimization can be classified in two main groups: local minimization and global minimization.

9.4.1.- Local DS minimization

In local DS minimization the whole DS search space is not computed in advance for all pixels, and so, each pixel's DS is usually computed separately. For that reason the best match for each pixel can actually be computed independently from neighbor pixels (except for some restrictions like the smoothness one that might take into account neighbor disparity values). Oppositely, as will be seen later on section 9.4.2, global DS minimization does compute the whole DS space in advance to later find a global solution.

Hence, it can be easily seen that since the whole DS space does not need to be computed (recall that it can easily be a 4D or 6D space), computation time is usually much faster than on many global algorithms. Besides, the whole DS space doesn't need to be stored either, which helps to also reduce memory requirements for DS minimization.

Since local DS minimization is usually based on the independent computation of the DS scores for each pixel on the reference image by using any desired restriction and any of the matches distances on chapter 7 for the smallest possible amount of disparity hypotheses, it will be seen that local DS minimization algorithms will mainly differ on the amount of times DS cost function needs to be evaluated for each pixel when searching for the best match. For that reason, a trade-off between quality and speed of matches computation will usually exist.

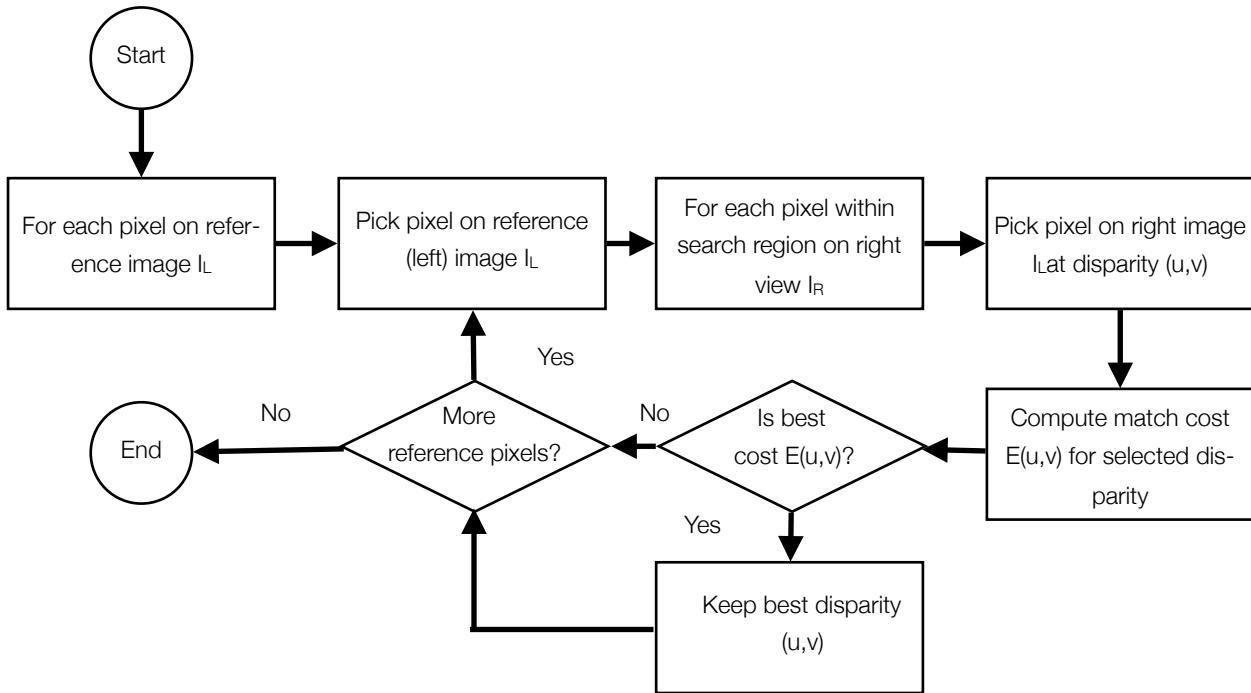
9.4.1.1- Winner Takes All

This is the simplest DS local minimization algorithm. It's based on the brute force, and hence if restrictions are correctly applied, usually yields the best results at the expense of being the most computational intensive local algorithm [5].

The algorithm is very easy to implement, and simply consists on computing the cost function score on each point on the reference image for any possible combination of horizontal and vertical disparity values (if images are rectified only horizontal disparities need to be evaluated). Usually a restriction on the maximum and minimum disparity values is imposed around the 10% or 20% of image size. From all the cost values obtained on a given point on the reference image, the disparity values that produce the best cost value are kept. This process is then repeated for all pixels in the reference image to obtain the dense disparity map. The algorithm is summarized below.

Because the algorithm uses brute force, it must be noticed that the whole DSI as shown in fig. 9.13 will be computed on each scanline for a pair of rectified images such as the ones in Middlebury's dataset [40]; however, because pixels are processed in an independent way, the whole DSI does not need to be kept in memory as any global minimization algorithm would do.

Algorithm 9.1. WTA DS cost function minimization



It must be noticed that efficient ways of implementing a local WTA algorithm exist, as it was shown in section 7.8. Indeed, this way of efficiently implementing a WTA algorithm can be seen as a mixture between a global and a local DS minimization algorithm, since the DS is computed globally, but only the best matches are kept, and hence the whole DS space does not need to be stored in memory. It can experimentally be seen that this kind of implementation of a local WTA algorithm, although similar to a global algorithm, which are usually much computationally intensive than local ones, is much faster to compute than a local WTA implementation as seen on this section.

An example of the results obtained using a WTA minimization technique is shown in fig. 9.15. Notice that the spurious spots present on estimated disparities correspond to incoherences that can be corrected using a simple left/right consistency check as will be seen at the end of this chapter.

9.4.1.2.- Evolutionary algorithms

Evolutionary algorithms are common techniques that are used in artificial intelligence as a way to minimize complex functions [34] when many variables need to be taken into account. Many evolutionary algorithms are inspired by aspects of nature such as biological evolution, as is the case for genetic algorithms, or man made techniques like annealing in metallurgy which is the main idea on the simulated annealing algorithm.

Even though each evolutionary algorithm might seem to be inspired by completely unrelated ideas, all of them have in common that the minimization of a complex function is obtained iteratively, and hence they are known as evolutionary.

In the following subsections a few of the most common evolutionary algorithms will be explained for the specific case of local DS minimization. It must be noticed however that any other evolutionary algorithm can also be adapted for the purposes of local DS minimization.



Fig. 9.15. Dense disparity estimation using WTA. (a) Left view. (b) Right view. (c) Dense disparity using left view as reference. (d) Dense disparity using right view as reference.

9.4.1.2.1.- Gradient Descent

Gradient Descent is among the simplest evolutionary algorithms. It's usually easy to compute, however, if the function to be minimized is not well behaved, the algorithm can easily get stuck on local minima, and hence provide unreliable minimization results.

However, if the algorithm is initialized at a point close to the desired solution, and if the function to be minimized is well behaved around such solution, usually the algorithm converges successfully to the right global minima.

Since Gradient Descent algorithm is based on the computation of gradients of the function to be minimized, we understand as a well behaved function to be minimized a function where gradient can be computed and changes smoothly. In layman terms this means that the function to be minimized should not have any abrupt changes and it should have a more or less concave shape (i.e. a parabola shape) at least around the minimum where the solution is found. Notice that concavity, for the case of local DS minimization over non-rectified images (where horizontal and vertical disparities need to be found), is referred to an actual 3D parabola of costs generated by each horizontal and vertical disparities, whereas if images are rectified, then the DS minimization is a one dimension problem for each pixel and the cost function should be a 2D parabola of costs respect to each horizontal disparity.

Hence, taking into account that optical flow at a given pixel (x,y) can be expressed as:

$$h(x,y) = (u(x,y), v(x,y)) \quad (9.98)$$

or in a shorter form:

$$\mathbf{h} = (u, v)^T \quad (9.99)$$

Then, given a cost function to minimize $E(u,v) = E(\mathbf{h})$, the Gradient Descent iteratively obtains the solution by using the following expression:

$$\mathbf{h}_{n+1} = \mathbf{h}_n - \mu \cdot \nabla E(\mathbf{h}_n), \quad n \geq 0 \quad (9.100)$$

where $\mathbf{h}_n$ is the disparity value at pixel (x,y) for iteration n , which can also be expressed as $\mathbf{h}_n(x,y) = (u_n(x,y), v_n(x,y))^T$, μ is the damping parameter to adjust convergence of the algorithm. Usually μ is a positive small number (i.e. typically $\mu = 0.9$) that ensures proper convergence by making displacements close to the gradient values. Notice that if μ is too big, the algorithm would constantly iterate without achieving convergence, and on the other hand, if μ is too small, then the algorithm would converge very slowly or get stuck in local minima. Finally $\nabla E(\mathbf{h})$ is the gradient of the cost function as defined in section 9.3 and evaluated at disparity $\mathbf{h}_n(x,y)$.

As it can be seen from expression (9.100), on each iteration we obtain a new disparity value depending on the gradient value of cost function at previous iteration. Indeed it can be observed in fig. 9.16 that gradient of a function points towards its local maxima, and because we are moving towards $-\mu$, we are then moving towards local minima (and hence the negative sign before the μ parameter). Indeed fig. 9.16 depicts each step $-\mu \cdot \nabla E(\mathbf{h}_n)$ that makes disparity to converge to a local minima.

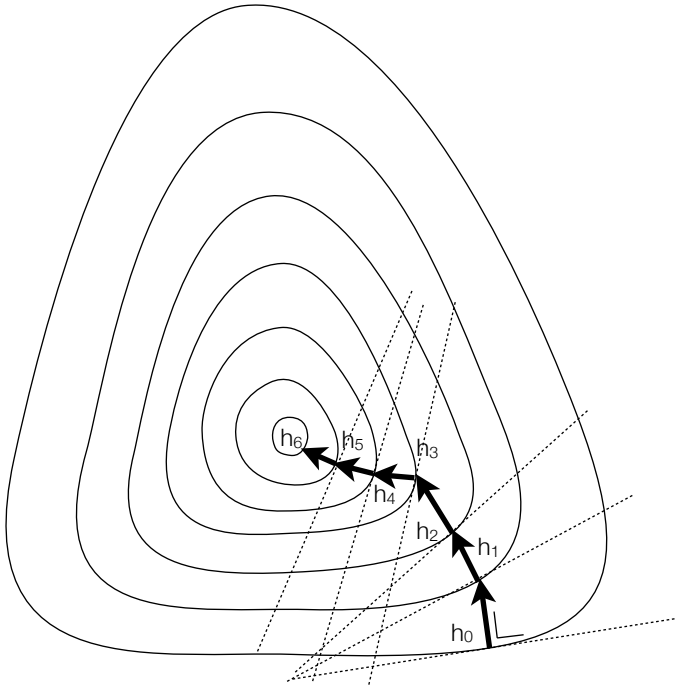


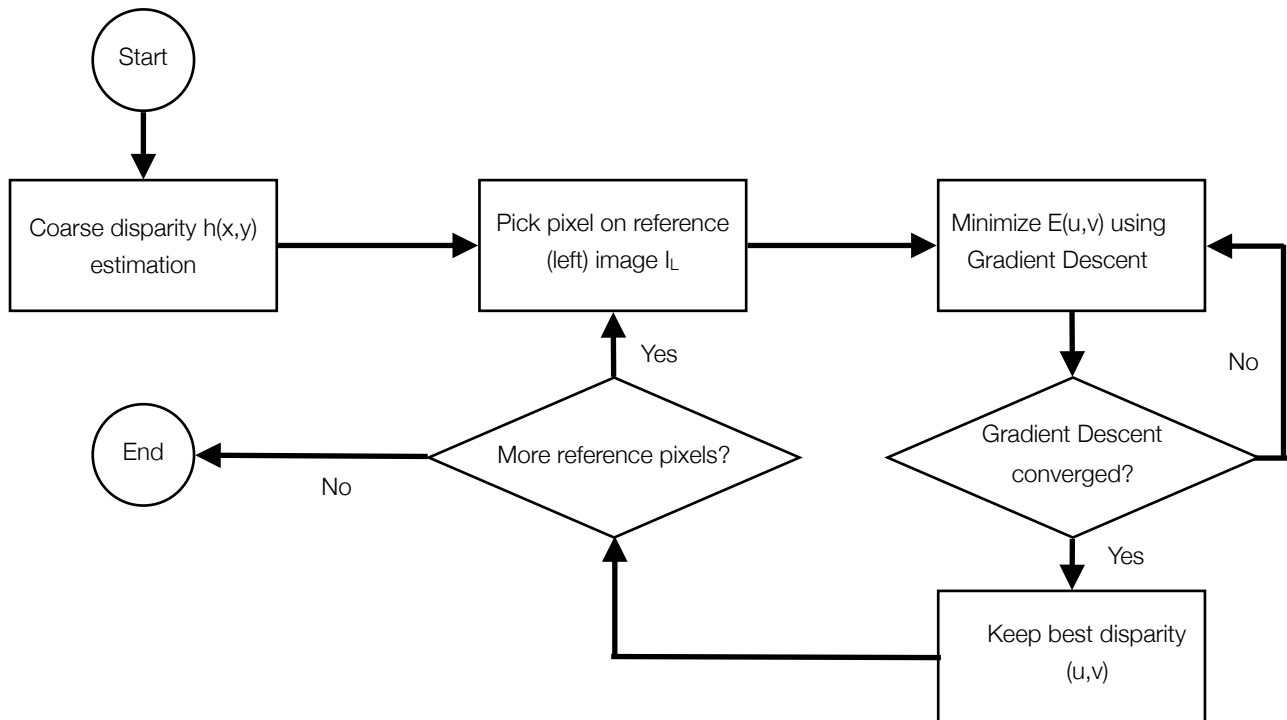
Fig. 9.16. Illustration of gradient descent. Solid curves are the contour lines of cost function $E(u,v)$ with a minimum located at its center. Algorithm converges to center on 6 iterations at $\mathbf{h}_6$

It can be observed then, that Gradient Descent only ensures that a **local minimum** is found over a provided cost function, which might not be located at the actual disparity values (which should be located at a **global minimum** if the cost function is well behaved). For that reason the Gradient Descent algorithm should be initialized at a horizontal and vertical disparity values close enough to the solution, otherwise the algorithm might not properly converge and get stuck in a local minimum. Also notice that the cost function to be minimized must depend on disparity parameters to be found, hence if rectified images are being used, then only horizontal disparity is used and vertical disparity is assumed to be zero.

An example of a method to obtain an initial hypothesis close to the real solution will be seen on section 9.4.1.2.4. Also, if the *grey value and gradient constancy assumptions* are valid, expressions (9.76) and (9.83) can be used to create a linear system of equations and find initial guesses for disparity values.

In summary, to obtain a dense disparity map using a Gradient Descent algorithm, first an initial coarse solution should be computed, then starting from that solution, for each pixel on the reference image, the

Algorithm 9.2. Gradient Descent DS cost function minimization



(a)



(b)

Fig. 9.16. Dense disparity estimation using Gradient Descent over images in fig. 9.15.a and 9.15.b. (a) Dense disparity using left view as reference. (b) Dense disparity using right view as reference.

best solution is found using Gradient Descent until convergence. This is summarized on algorithm 9.2.

An example of the estimated disparity obtained using this minimization algorithm is shown in fig. 9.16.

9.4.1.2.2.- Levenberg-Marquardt.

The Levenberg-Marquardt algorithm is an improvement over the Gradient Descent algorithm in order to obtain more robust solutions by avoiding getting stuck on local minima. The algorithm can be seen as a mixture between a Gauss-Newton algorithm and a Gradient Descent, and the main idea to achieve more robustness respect Gradient Descent is to adaptively

change the step or damping parameter on the Gradient Descent algorithm depending on the results of the optimization on previous iteration, and hence becoming adaptive.

In short, the more the minimization improves respect to previous iteration, the more the step parameter will increase so that the part of the minimization algorithm makes bigger steps and achieves faster convergence. On the other hand, if the cost function worsens respect to previous iteration, then the step parameter is reduced so that the minimization algorithm makes smaller steps to search for the right solution in case that the right solution might have been overlooked in previous iteration due to a step that was too big. This way more robustness is obtained.

Hence, because the convergence parameter can be adaptively changed, expression (9.100) becomes:

$$\mathbf{h}_{n+1} = \mathbf{h}_n - \mu_n \cdot \nabla E(\mathbf{h}_n), \quad n \geq 0 \quad (9.101)$$

and then the damping parameter on iteration $n+1$ can be obtained as:

$$\mu_{n+1} = \begin{cases} \mu_n \cdot \nu, & E(\mathbf{h}_{n+1}) < E(\mathbf{h}_n) \\ \mu_n / \nu, & E(\mathbf{h}_{n+1}) \geq E(\mathbf{h}_n) \end{cases} \quad (9.102)$$

where $\nu > 0$ is the factor to increase or reduce the damping parameter μ_n on each iteration.

Notice that on the first iteration, the damping parameter can be initialized to the same typical value used on Gradient Descent, and hence $\mu_0 = 0.9$.

By looking at the behavior of this method in algorithm 9.3, it can be easily seen why the Levenberg-Marquardt algorithm usually obtains faster convergence and more accurate results than a typical Gradient Descent algorithm. This translates into less iterations, and hence less DS cost function evaluations, which at the same time translates into faster computational time for the same match quality respect to a simple Gradient Descent used for local DS minimization.

Algorithm 9.3. Levenberg-Marquardt DS cost function minimization

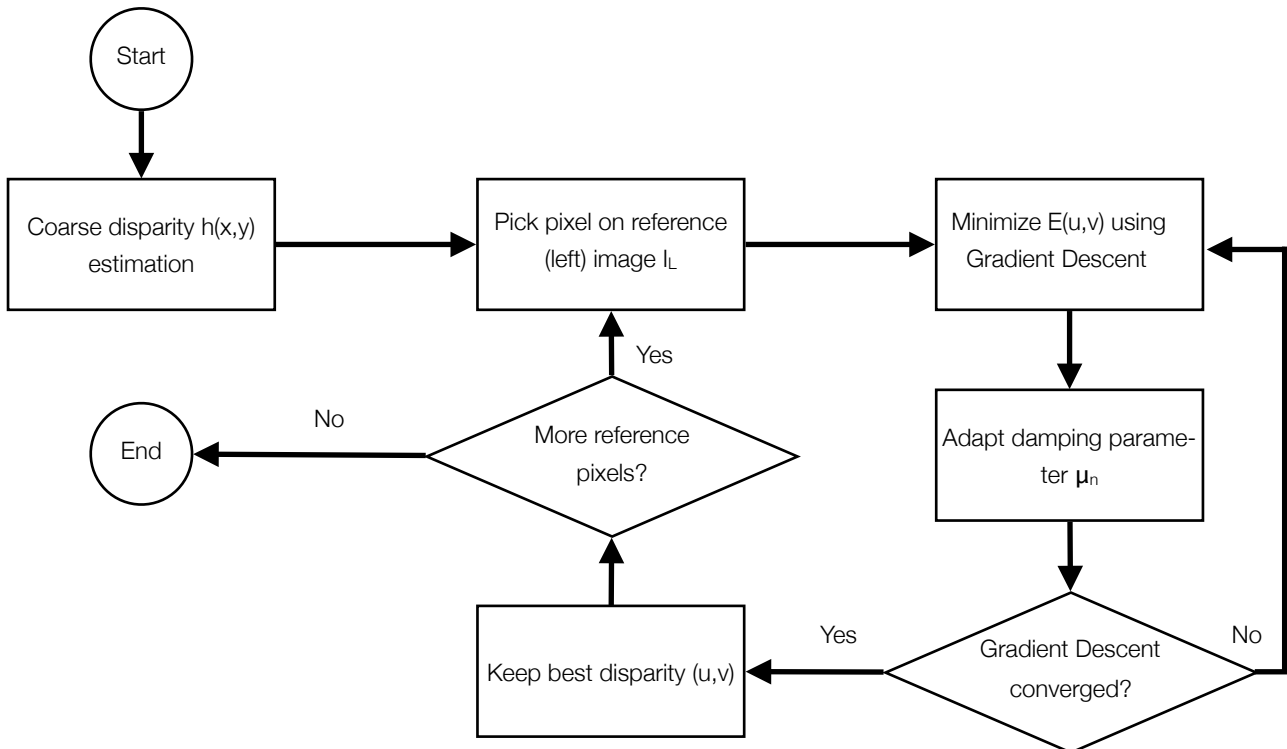




Fig. 9.17. Dense disparity estimation using Levenberg-Marquardt over images in fig. 9.15.a and 9.15.b. (a) Dense disparity using left view as reference. (b) Dense disparity using right view as reference.

Once the inner workings of the Levenberg-Marquardt are known, its adaptation to local DS minimization is very similar to the case of Gradient Descent. First an initial coarse solution should be computed, and then from that solution, for each pixel on the reference image, the best solution is iteratively found using Levenberg-Marquardt algorithm (where the step size is adapted at each iteration) until convergence. This is summarized on algorithm 9.3.

An example of the results obtained using a Levenberg-Marquardt minimization technique is shown in fig. 9.17.

9.4.1.2.3.- Simulated Annealing

Another well known evolutionary algorithm for function minimization is Simulated Annealing. This algorithm was inspired by the annealing process in metallurgy. It comes from the idea that when iron or steel is initially melted, the heat causes their atoms to move rapidly to other locations. As the mixture of the alloy cools down, their atoms start to move slower to regions where they achieve a stable equilibrium until the mixture has been completely chilled and atoms remain still creating a solid crystalline structure. The funny part of the story is that if by some reason the speed or the way temperature decreases is modified, then the obtained crystalline structure (i.e. the most stable minimum energy solution) on the alloy becomes different as well, which translates into different properties in such alloy.

Simulated Annealing works with a very similar idea, but in this case, instead of the alloy we have a cost function to be minimized, and the goal of the algorithm is to find the most stable solution that produces a minimum cost.

The algorithm works first by defining the annealing temperature schedule, which is the way temperature is decreased. Typical schedules decrease temperature either linearly or exponentially throughout a predefined number of iterations, although usually the latter produces better results. Hence, the linear annealing schedule can be expressed as follows:

$$T(x) = T_{Start} + (T_{End} - T_{Start}) \cdot x, \quad 0 \leq x \leq 1, \quad T_{Start} \leq T_{End} \quad (9.103)$$

where x denotes the progress (expressed as a value between 0 and 1) on the number of iterations the Simulated Annealing is allowed to make. T_{Start} denotes the initial temperature (when $x = 0$), T_{End} is the end temperature (when $x = 1$) and $T(x)$ is the temperature for provided progress x . Notice that because temperature is supposed to decrease, then $T_{End} < T_{Start}$.

On the other hand, if annealing schedule is desired to decrease temperature more smoothly, it can be defined using its exponential definition as:

$$T(x) = (T_{Start} - T_{End}) \cdot (e^{-\lambda \cdot x} - x \cdot e^{-\lambda}) + T_{End}, \quad 0 \leq x \leq 1, T_{Start} \leq T_{End}, \lambda \geq 0 \quad (9.104)$$

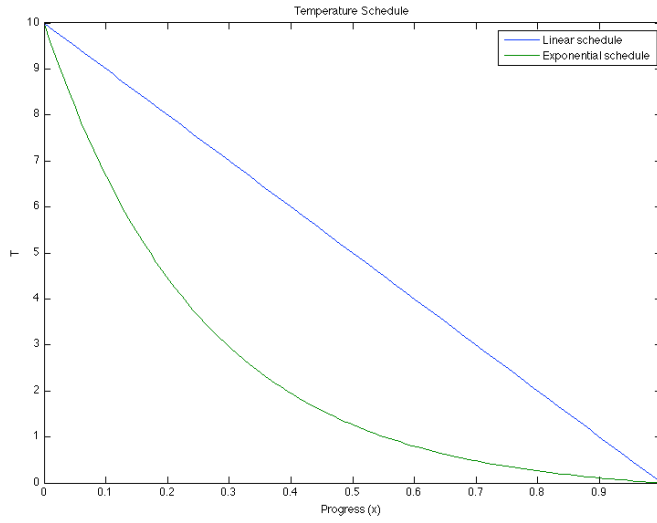


Fig. 9.18. Temperature schedules for Simulated Annealing minimization.

decreases, random distance decreases as well. This is not always used but our experiments have proven to yield better results if random distance is decreased.

If the obtained cost at this current random hypothesis is the best one, we keep the disparity parameters that produced such cost. Otherwise if the new cost is worse than the previous one, with a given *probability that will decrease as the temperature decreases*, we will decide whether we will move to the current solution to pick another random hypothesis around it on next iteration or not, just as shown below:

$$P(E_{n-1}, E_n, T_n) = \begin{cases} 1, & E_n < E_{n-1} \\ e^{-\frac{E_n - E_{n-1}}{T_n}}, & E_n \geq E_{n-1} \end{cases} \quad (9.105)$$

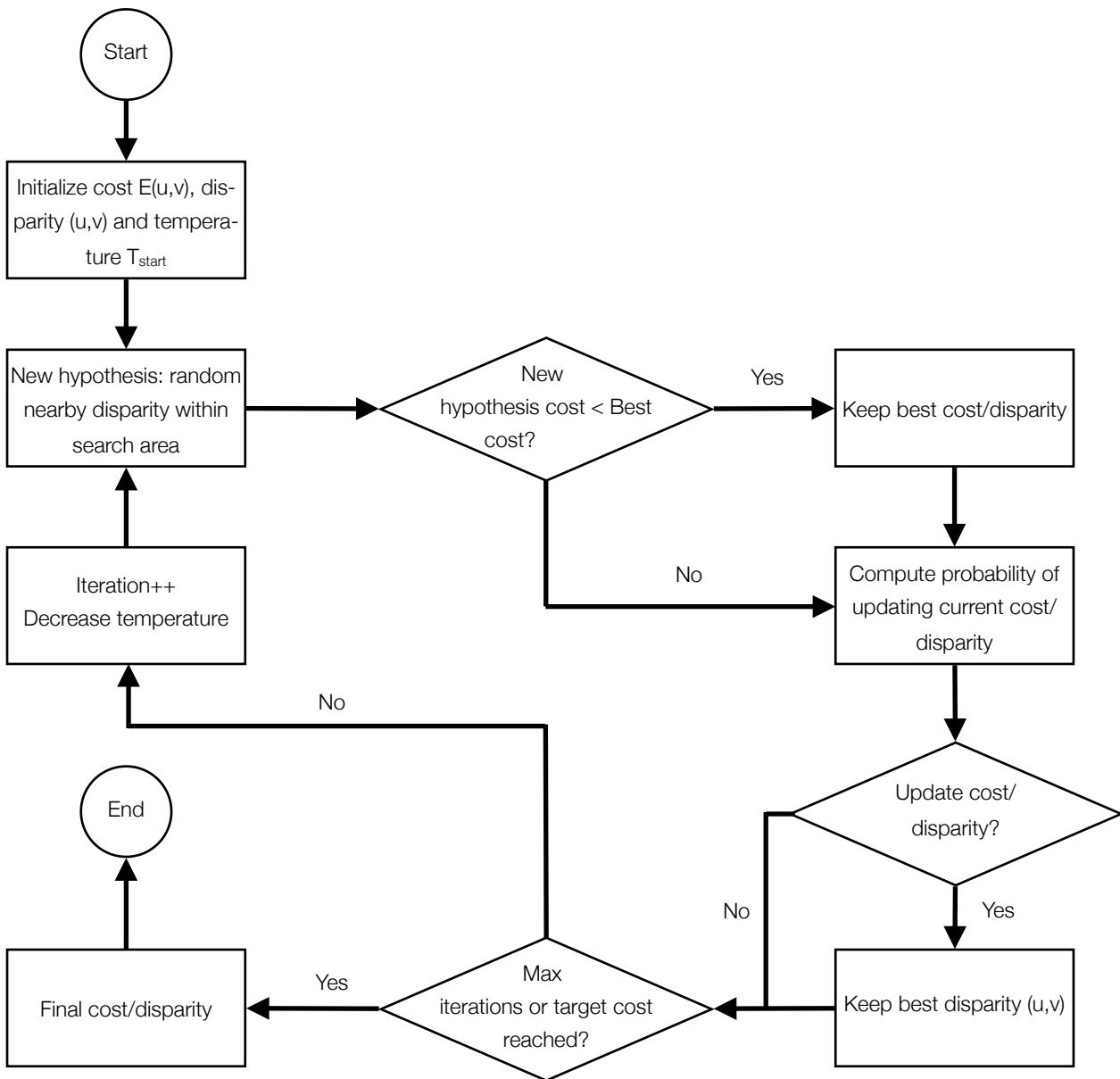
where E_n stands for the cost functional at iteration n , T_n stands for temperature at iteration n and $P(E_{n-1}, E_n, T_n)$ stands for the probability of accepting a new disparity hypothesis, which can be seen that is always accepted for better new costs (i.e. probability equal to 1), and sometimes it is accepted with a decreasing probability as the temperature decreases when the new costs are worse.

Finally, if we don't move to the latest disparity hypothesis location, then on next iteration another random hypothesis is generated around current location, otherwise the new location is used to generate a new nearby random hypothesis.

In any case, because disparities producing the best cost are always kept even if we don't move to their location; after the Simulated Annealing finishes, the disparities that produced the lowest cost are chosen. Hence, this behavior allows finding the global minimum on the cost function $E(u, v)$ without getting stuck in local minima because we let the algorithm move towards worse locations with a decreasing probability. This behavior is summarized on algorithm 9.4.

From the previous paragraph it can be deduced that simulated annealing does not always accept the best found solution during an iteration as other algorithms like Gradient Descent do. Instead, on initial iterations it is more prone to accept worse solutions whereas as the temperature decreases, new worse solutions are usually discarded. This difference in behavior respect to Gradient Descent helps to avoid getting stuck in suboptimal solutions and makes Simulated Annealing a very robust algorithm for function minimization even if the function is not very well behaved. On the other hand, there's a price to

Algorithm 9.4. Inner Workings of Simulated Annealing



pay for such robustness, this price comes in terms of computational time, as this algorithm converges very slowly (although we have experimentally seen that our implementation minimizes the DS 2.5 faster than WTA minimization), and also in complexity of configuration, since annealing schedule is complex to be set so that the best results are always obtained.

However, convergence of simulated annealing can be improved if proper initialization is provided (i.e using any of the methods for coarse disparity estimation indicated on Gradient Descent or Levenberg-Marquardt algorithms), leaving then only the complexity of the best annealing schedule as a problem to be solved. In that sense, our experiments have determined that an exponential schedule with start temperature of $T_{Start} = 10$ and end temperature of $T_{End} = 1e-6$ with decay equal to $\lambda = 4$ worked the best.

Again, once we know how the Simulated Annealing algorithm inner workings, its adaptation to local DS minimization is very similar to other evolutionary algorithms seen before. We start by first computing an initial approximate solution, and then for each pixel on the reference image, the local DS is minimized independently using Simulated Annealing until convergence. This process is summarized on algorithm 9.5.

Indeed, by using algorithms 9.4 and 9.5 for DS minimization, we can iteratively obtain disparities as depicted on fig. 9.19.

As it can be observed on fig. 9.19, convergence is quite random, mostly due to the random nature of the whole simulated annealing process, but we can see that in general terms, convergence is correct in most situations (error tends towards zero), except in some areas where some occlusions are present (i.e errors do not converge to zero, and instead converge to other values).

An example of the results obtained using a Simulated Annealing minimization technique is shown in fig. 9.20 for comparison purposes against other minimization algorithms.

Algorithm 9.5. Simulated Annealing DS cost function minimization

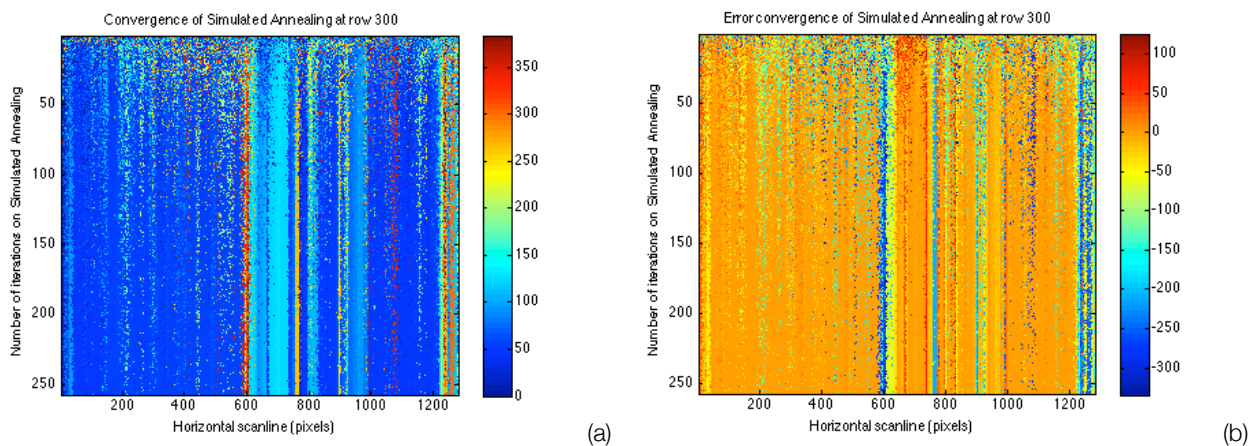
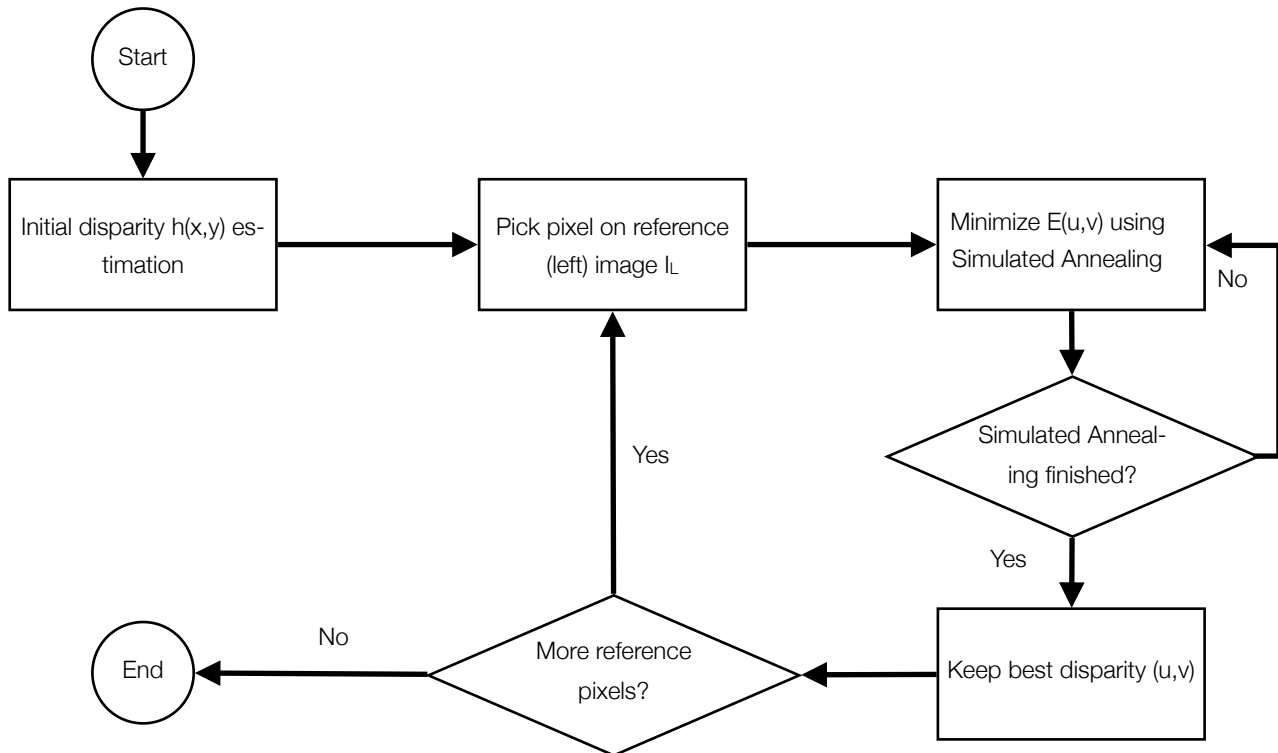


Fig. 9.19. Convergence of disparities using Simulated Annealing over a single scanline on Middlebury's Aloe dataset [40]. (a) Convergence of disparity values. (b) Convergence of disparity errors respect to ground-truth.



Fig. 9.20. Dense disparity estimation using Simulated Annealing over images in fig. 9.15.a and 9.15.b. (a) Dense disparity using left view as reference. (b) Dense disparity using right view as reference.

9.4.1.2.4.- An example of evolutionary algorithms: HRM

Hybrid Recursive Matching (HRM) [33], is an algorithm that was intended primarily for motion estimation for video coding. Since disparity (or optical flow) is inherently related to the movement of pixels on a scene between pairs of frames, we thought that HRM could be used for dense disparity estimation with some minor modifications. The main difference is basically that motion estimation on video coding is usually a decimated version of dense disparity since motion is only estimated at block level whereas dense disparity estimates motion at pixel level. This fact will mean that HRM will need to be modified so that more detail in motion estimation is obtained, which will mean that more computational effort will be needed.

On the other hand, since HRM is intended to be used on video sequences, it was designed to work as fast as possible (almost real time) at the expense of slow convergence. In other words, the first frames on the sequence always have large amounts of error on motion estimation, and as more frames are added, since usually movements on video sequences are small enough, the algorithm slowly converges to the right solution (in about 20 frames), and from that point on, usually convergence is good enough. Besides, because HRM was designed for video coding, matches between blocks were found so that error was minimized and could be more easily encoded.

This behavior might be good enough for video coding, however for the sake of 3D reconstruction and dense disparity estimation, it might be completely unacceptable. This is especially true if we need precise 3D reconstruction or super-resolution texturing. First of all, epipolar constraints must also be taken into account so that the correct reconstruction can be computed from a fixed scene that has been recorded from different points of view. On the other hand, in order to be able to accurately reconstruct 3D locations we need precise stereo matching, and hence, accuracy becomes more important than speed, for that reason we chose to use any of the robust evolutionary algorithms that we have previously shown until convergence on each pixel was obtained. That way we can obtain enough accuracy within acceptable execution times.

It is now time to get a deeper insight into the implementation of HRM, which is basically divided in two stages: coarse and fine disparity estimation.

First a coarse disparity estimation is computed by dividing the image into big blocks of size 4x4. Each block is initialized with a random disparity value on all of its pixels (within the allowed limits, or within their respective epipolar lines if epipolar restrictions are imposed).

Once all blocks have been initialized, coarse disparity estimation is computed as follows: for each coarse block, a given number of spatial and temporal neighbor coarse blocks are picked (usually 1 temporal and 3 spatial blocks, but more can be used for faster convergence at the expense of a higher computational effort) and the cost function $E(u,v)$ is evaluated at the

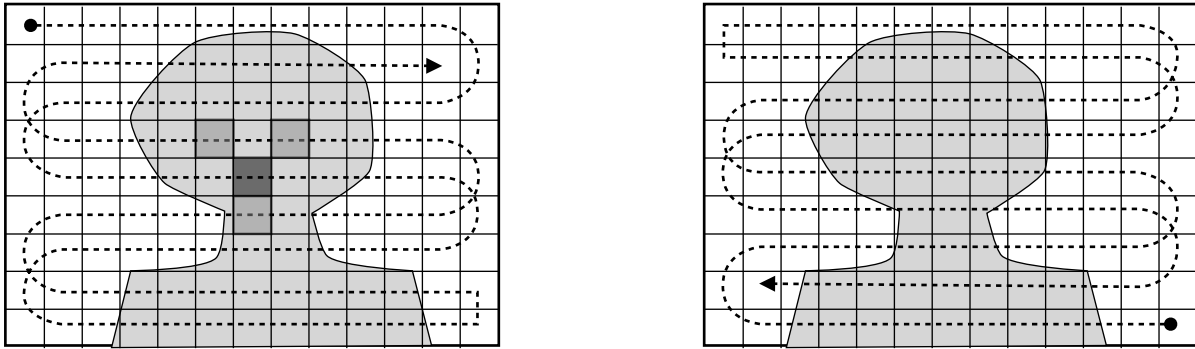


Fig. 9.21. Coarse block matching stage. Disparity on each block is updated using spatial and temporal neighbor blocks following a zig-zag path. Darkest block represents current block, and the mid-dark ones contains disparity of neighbor blocks to use during search for best block disparity.

center of the current coarse block with each of the initial random disparities on each selected neighbor coarse block. The disparity that produces the best cost is assigned to all the pixels in the current coarse block. This process is repeated for all coarse blocks following a zig-zag between scanlines. Since coarse blocks are consecutively evaluated following a zig-zag, the disparity obtained on each coarse block will converge to an approximation of the real dense disparity because as consecutive coarse blocks are evaluated a certain smoothness restriction is imposed. Also because we use a temporal coarse block placed at the same location on the disparity obtained on previous frame (if available), smoothness is also imposed in the temporal domain, thus allowing the algorithm to slowly converge to the right solution in a video sequence. The whole coarse block matching is depicted in fig. 9.21.

It must be observed, that if not enough frames are available to obtain proper temporal convergence of the algorithm, then more neighbor coarse blocks should be used to obtain faster convergence.

It will also be noticed that this approximation of dense disparity will have a coarse square nature due to the size of coarse blocks. Hence, the smaller the coarse blocks are, the better the coarse dense disparity will be at the expense of being more computationally intensive. Our experiments have proved that coarse blocks of size 4x4 produce good results. Besides, if we increase the number of neighbor coarse blocks to be evaluated, also the coarse dense disparity will be more reliable and will converge faster (again at the expense of computational time). Also, in our experiments we evaluated all of the 8 coarse blocks directly surrounding each block (for coarse blocks located at image borders, we simply evaluated all direct neighbor coarse blocks) whenever no previous temporal frame was available so that faster convergence of the coarse block matching was obtained. As the reader might already have noticed, this process of coarse dense disparity estimation, as mentioned in previous sub-sections can be used to initialize any evolutionary algorithm for local DS minimization.

Once the coarse disparity estimation stage is completed, the fine disparity estimation starts by refining the disparity on each coarse block at a pixel level. This is done by locally minimizing the DS using an evolutionary algorithm (a Levenberg-Marquardt will be enough) at each pixel on the reference image using the disparity of the coarse block where the pixel is located as an initialization for the evolutionary algorithm. Usually a few iterations (i.e. less than 20) will be enough to achieve good results for single pairs of images, on the other hand, when working over video sequences even less iterations can be used (i.e. 1 iteration). Again it must be noticed that if a more robust evolutionary algorithm is used and it is left until full convergence, then usually better results will be obtained again at the expense of more computational time.

Once all the pixels within a block have been processed, the pixel with the lowest achieved cost is used to update the coarse block disparity of its corresponding block as long as the match is better than the previously available coarse block match. This last condition ensures that the algorithm does not diverge to wrong solutions over consecutive frames on a video sequence. The fine disparity estimation is depicted on fig. 9.22.

An example of the results obtained using this method can be seen on fig. 9.23.

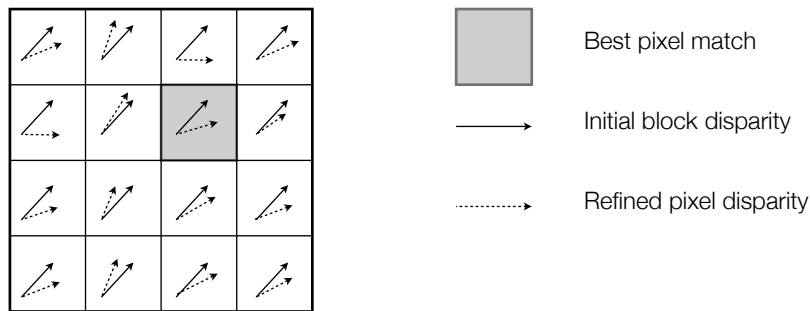


Fig. 9.22. Fine pixel matching stage. 4x4 block containing initial block disparity (solid arrows) and refined pixel disparities (dashed arrows). Best pixel match (marked in dark) is used to update block disparity for next frame.

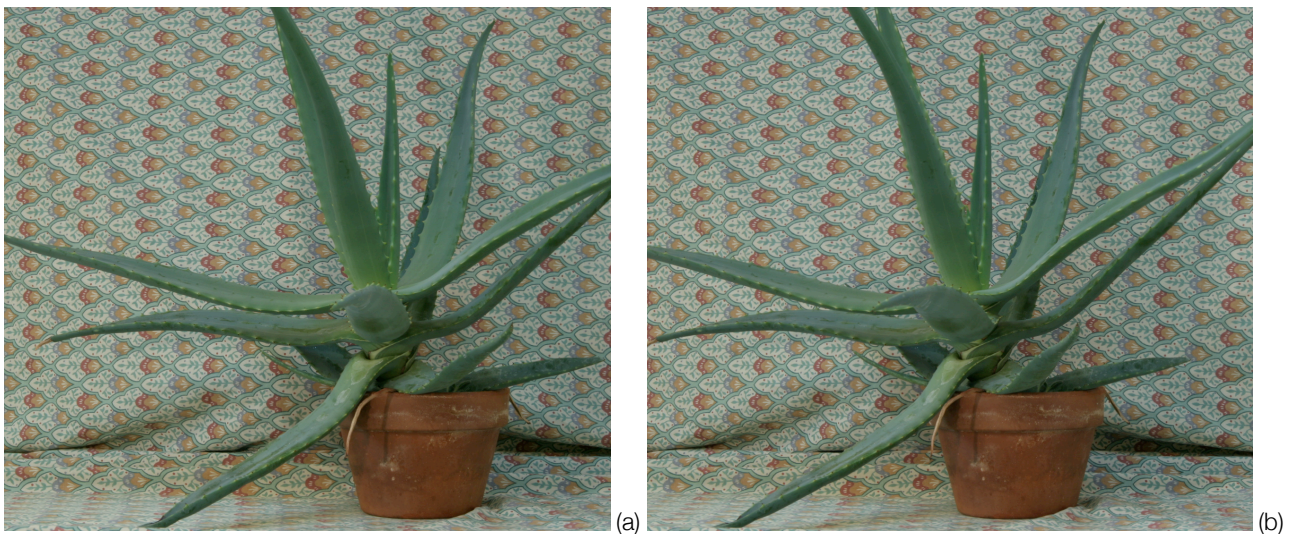


Fig. 9.23. Dense disparity estimation using HRM over images in fig. 9.15.a and 9.15.b. (a) Dense disparity using left view as reference. (b) Dense disparity using right view as reference.

9.4.2.- Global DS minimization

Global DS minimization is based on the whole DS computation in advance for all pixels in the reference image and all possible disparity values before attempting minimization. This imposes a hard problem, as the whole DS computation itself is very computationally intensive, and storage of the whole DS might be unachievable for many devices, especially in small embedded devices like gaming devices or smartphones.

Many global DS minimization algorithms exist, and aside from their extremely high computational needs, they usually yield the best results, because DS is no longer minimized independently for each pixel on the reference image. However due their specific needs in terms of memory and computational time, we didn't implement any of the most computational intensive ones. The reader is suggested to check the following reference for further information on global minimization algorithms such as graph cuts or believe propagation [5].

Instead we have implemented some of the least computational intensive global DS minimization algorithms, that usually do not need to store the whole DS in advance (just a portion of it), and hence produce high quality results within reasonable amounts of computational time and memory.

We show next some of these typical global DS minimization techniques.

9.4.2.1.- Dynamic Programming

Dynamic Programming is a technique to solve complex problems (usually minimization problems), by means of breaking the problem into smaller ones that can usually be more easily solved.

The Dynamic Programming techniques that we have implemented attempt to minimize the DS on each scanline in a similar way to [29], so if the images are rectified, minimization is done over a 2D space, whereas if they're not, then minimization is

done over a 3D space. Notice that if robust matching is used further dimensions might be needed (i.e. scale and rotation for log-polar transforms). In any case, it must be taken into account that the larger the number of dimensions on the DS, the less robust and more computationally intensive disparity estimation will be.

The goal of Dynamic Programming algorithms for the purpose of disparity estimation, will be to find the minimum cost path on the DS throughout each scanline. More specifically, by looking at fig. 9.24, where the DS for a single scanline using SAD is depicted, we can observe that throughout the scanline there are contiguous blue disparity areas that traverse the whole scanline forming a path of disparities (except for some discontinuities that are due to changes in depth at object boundaries). This is the minimum cost path, which must be found by means of Dynamic Programming algorithms.

As it can easily be seen, since minimization is done over a whole scanline, then the minimization algorithms are no longer completely local, as pixels are not minimized independently. However, minimization is neither completely global, as the whole DS throughout the whole image is not minimized at once (just one scanline each time), and the whole DS space does not need to be computed and stored in advance either. Also, since each scanline is minimized at once by finding the minimum cost path, it can be ob-

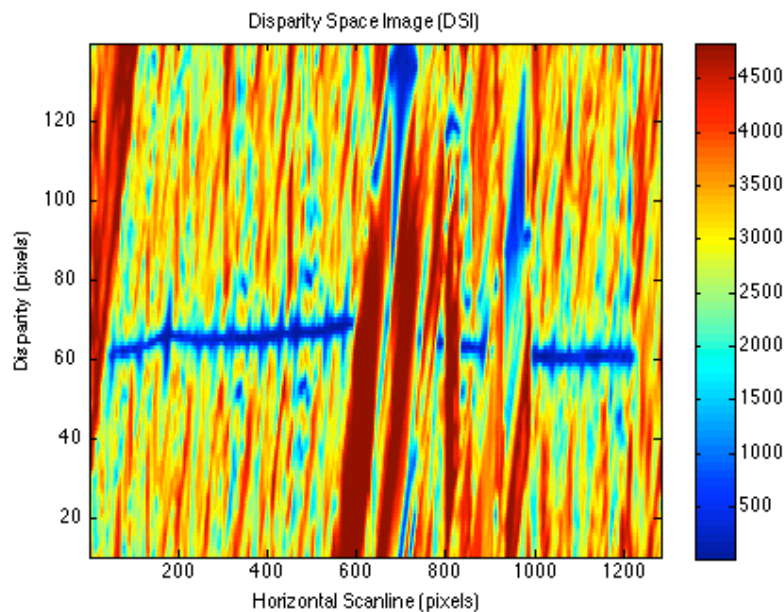


Fig. 9.24.- DSI over a scanline using SAD distance metric on the aloe Middlebury set [40]. Contiguous blue areas contain the minimum cost path that traverses the scanline.

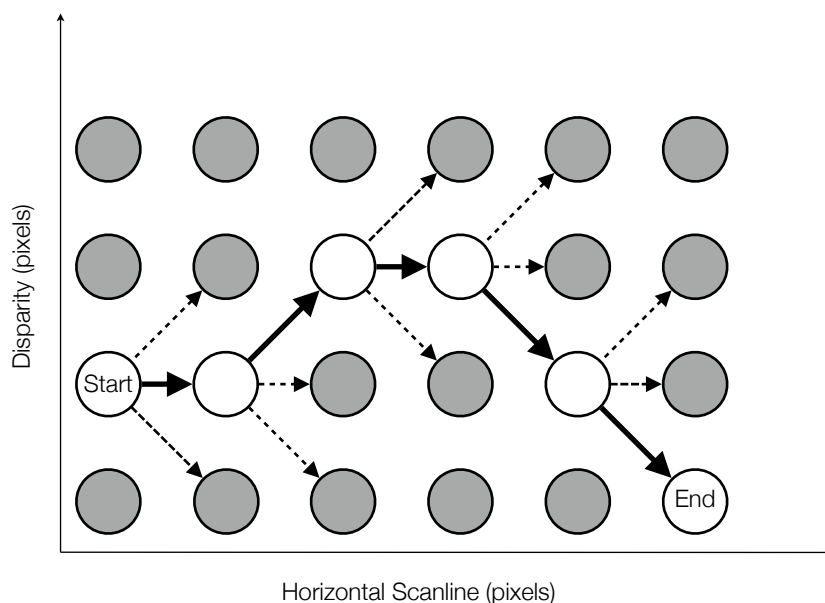


Fig. 9.25. The DS seen as a network of nodes having their associated costs $E(u,v)$. White nodes indicate minimum cost path. Dashed arrows indicate nodes that have been tested during search for minimum cost path. Notice that only nearby and forward nodes respect to previously chosen nodes are tested.

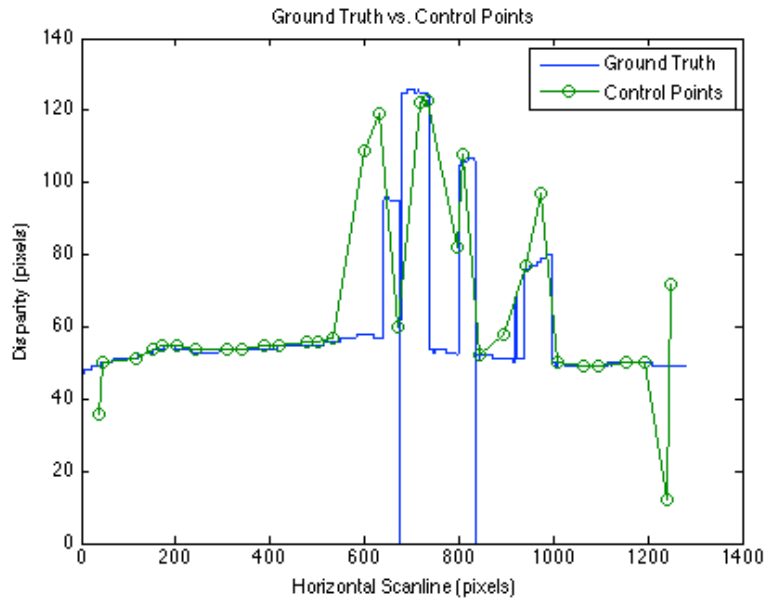


Fig. 9.26. Ground truth of disparity vs. Control Points over a scanline on Middlebury's Aloe set [40].

served that a smoothness constraint on DS minimization will always be implicitly imposed as well.

In order to find the minimum cost path, it will be assumed that the DS created by the cost function $E(u,v)$ is a network, which for the case of rectified images is as shown in fig.9.25 and has one node on each horizontal scanline location and each disparity value, and where each node has its associated cost defined as $E(u(x,y),v(y,y))$.

Also, because the scanline must be traversed from side to side, at least one start and end nodes must be selected to find the minimum cost path that joins them. Besides, because some Dynamic Programming algorithms might not produce reliable or optimal solutions, control points are

defined throughout the scanline which force the minimum cost path to pass throughout them. They can be seen as stops in a journey that goes from the start node on the leftmost column of the DS to the end node located at the rightmost column of the DS. Finally, because each pixel can only have a single disparity assigned to it, the minimum cost path won't be allowed to go backwards to previous pixels in the scanline.

There are different Dynamic Programming algorithms, each attempting to solve the same minimization problem with different amounts of quality on the obtained correspondences, as well as different computational times. As usual, it will be seen next that there is a trade-off between quality and speed.

We will show next how to compute the control points to traverse the scanline, and then a couple of the most typical dynamic programming algorithms.

9.4.2.1.1.- Finding the control points

As we have shown before, in order to find the minimum cost path, the start and end nodes must at least be defined. Also, depending on the chosen Dynamic Programming algorithm, it might be desirable to define control points throughout the scanline.

These control points (including start and end points) can be estimated by any of the local metrics explained in chapter 7 at points with high coherence (as well as at leftmost and rightmost locations for start and end points), as explained in chapter 8, to obtain more or less reliable matches at such control points.

As we saw on chapter 8, it is possible to detect the most coherent areas by means of the structural tensor. Hence, we could use that information in a similar way to determine the points that are easier to be matched. Once those points are uniformly determined (i.e. by using uniform tiling over the scanline), we can robustly find their corresponding disparities by means of **local** DS minimization over a single reference pixel (i.e. using WTA, Evolutionary algorithms, etc.) within a given search range of disparities. Notice that log-polar transforms can be used to further increase robustness.

In our experiments, we also observed that usually areas of low gradient tend to have less occlusions, and for that reason we proposed dividing the scanline into uniform tiles or areas, and search on each area for the smallest gradient point where the control point match is estimated by means of robust local DS minimization using a large enough window if the area has low

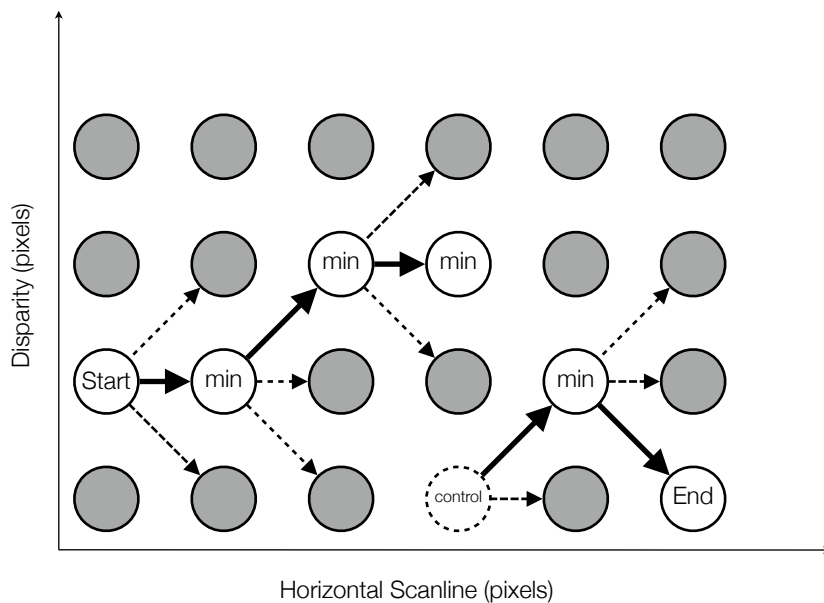


Fig. 9.27. Greedy algorithm traversing the scanline. A control point (dashed node) is enforced during traversal.

searches for the best cost value at each location on the scanline. In other words, since we know that disparity is piece-wise smooth, for each location in the scanline, a small area around the previous best disparity value is checked (which is usually formed by only the direct neighbours on the network) to search for the new best disparity location until the whole scanline is traversed. This is depicted on fig. 9.27.

Since we also know that there is a correlation between disparity changes and image gradient because disparity is piece-wise smooth, as it was shown in section 9.3.4; usually the small area to be searched can change in size depending on reference

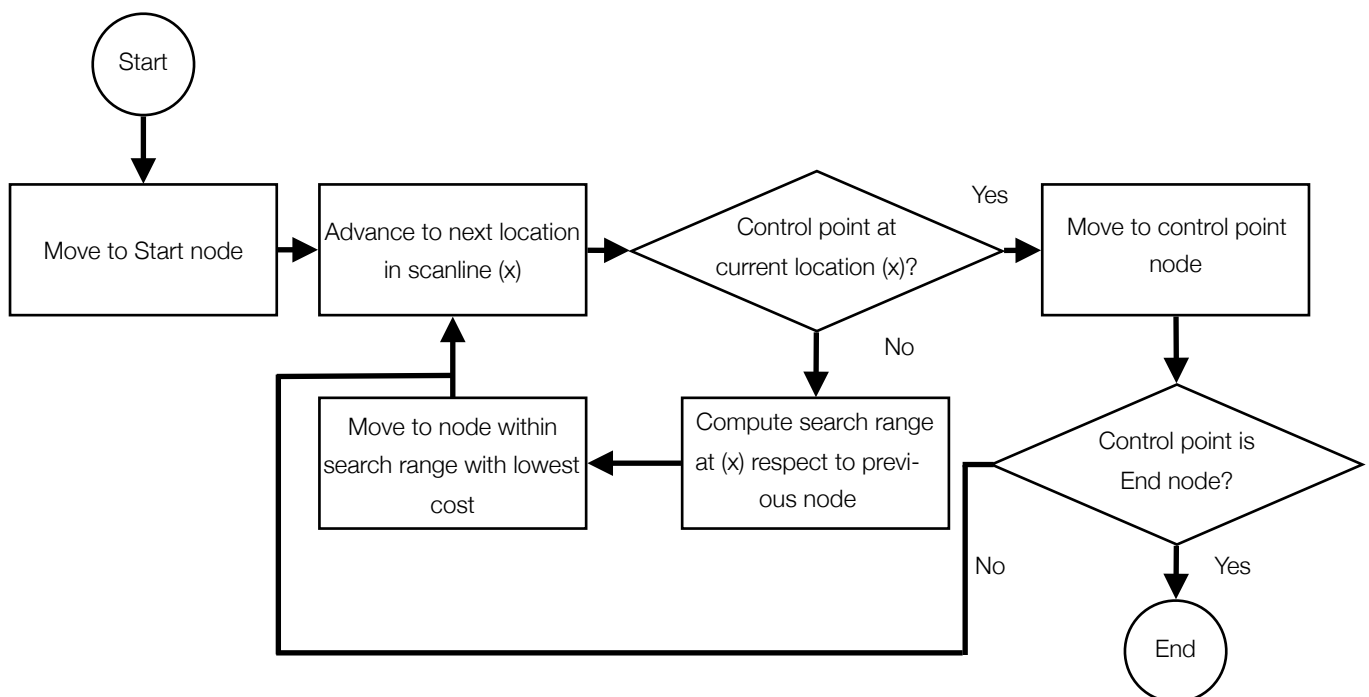
texturing (as is usually the case on low gradient areas). However this is not the only neither the most robust way to obtain reliable matches on control points. Usually it will be best to use matches obtained by re-projecting triangulated points on each view if a sparse 3D structure of the scene is available.

Once the control points get defined throughout the scanline, a sparse representation of the minimum cost path is obtained as shown on fig. 9.26.

9.4.2.1.2.- Greedy's Dynamic Programming algorithm

It is probably the simplest dynamic programming algorithm [35] as for dense disparity estimation it simply

Algorithm 9.6. Greedy's Dynamic Programming DS cost function minimization for a single scanline



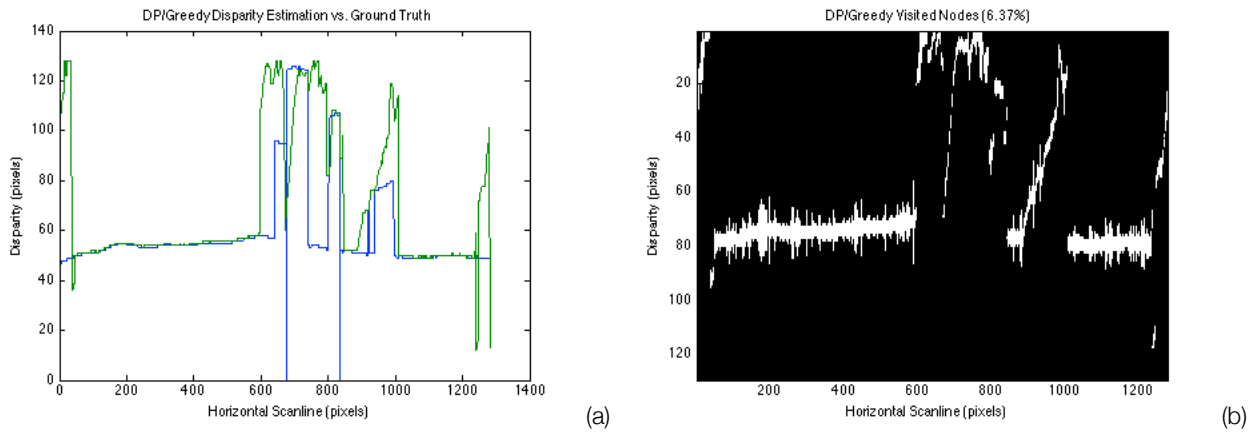


Fig. 9.28. (a) Greedy's result over a scanline on Middlebury's Aloe set [40] vs. disparity ground-truth. (b) DS Visited nodes where cost $E(u,v)$ was computed during scanline traversal (in white). Only 6.37% of nodes on the DS were visited.

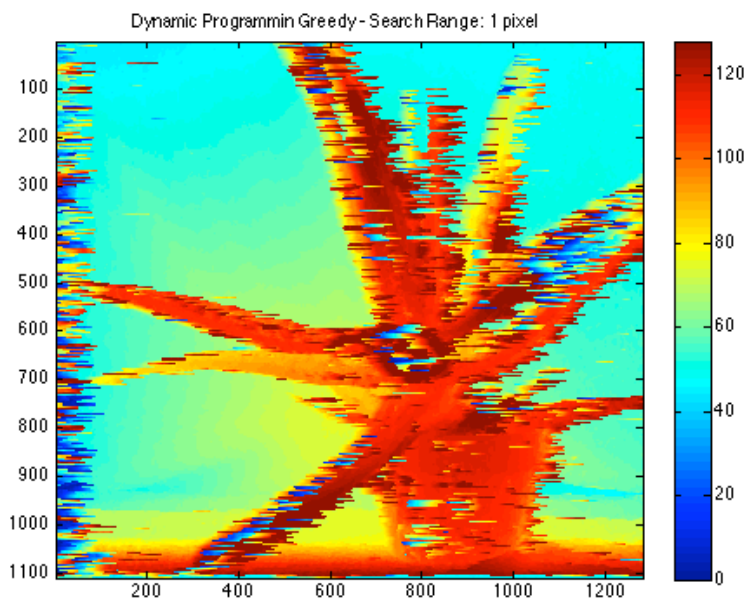


Fig. 9.29. Disparity estimation using Greedy's Dynamic Programming minimization algorithm. It can clearly be observed the "streaking" effect due to independent scanline minimization of the DS.

the estimated disparity from the true value, we impose a series of control points throughout the scanline where the dynamic programming algorithm is restarted to a known disparity value. This can also be seen on fig. 9.27.

Although Greedy's disparity estimation might be poor in terms of quality, if enough control points are provided (at the expense of a higher computational effort), then it's quality will be good enough for 3D reconstruction purposes. Besides, due to its simplicity, Greedy's Dynamic Programming is an extremely fast way of estimating dense disparity between pairs of images.

Even if enough control points are provided, Greedy's algorithm tends to suffer from a "streaking" effect that is due to independent minimization of each scanline, as can be seen on fig. 9.29. By providing even more control points, this problem can be alleviated, however, computation of each control point can be very expensive. Another more simple and usually faster solution is to interpolate the results between consecutive scanlines and impose smoothness restrictions on estimated disparities between such consecutive scanlines.

image gradient at that point, so that the larger the gradient is the larger the search area becomes. This is shown on fig. 9.28 where visited nodes for each horizontal location on the scanline increase or decrease depending on reference image gradient. This way we allow greater changes in disparity at points located at object boundaries. This process is repeated until the whole scanline is analyzed as it is summarized in algorithm 9.6.

It must be noticed that this algorithm does not ensure that the minimum cost path throughout the whole scanline is obtained, as only a simple and local analysis is done. For that reason, usually errors tend to accumulate during search for the best path and obtained results are usually poor. In order to prevent this sort of error accumulation that drifts

9.4.2.1.3.- Dijkstra's Dynamic Programming algorithm

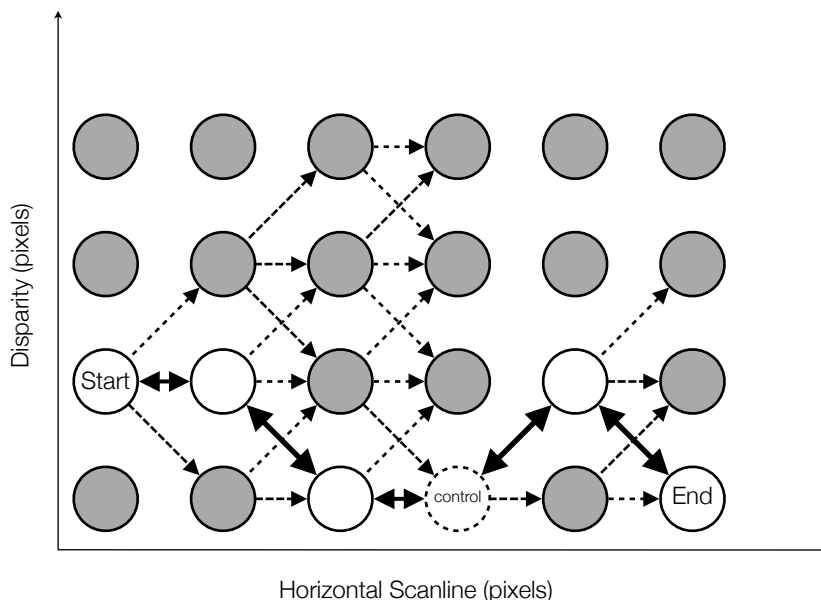
Dijkstra's algorithm is a well known algorithm that has been widely used to search for minimum cost paths for routing purposes among other things [35].

In order to estimate disparity between a pair of images, again, we proposed the minimization of the DS through each scanline separately, but instead of using Greedy's approach, we used Dijkstra's one with some slight modifications.

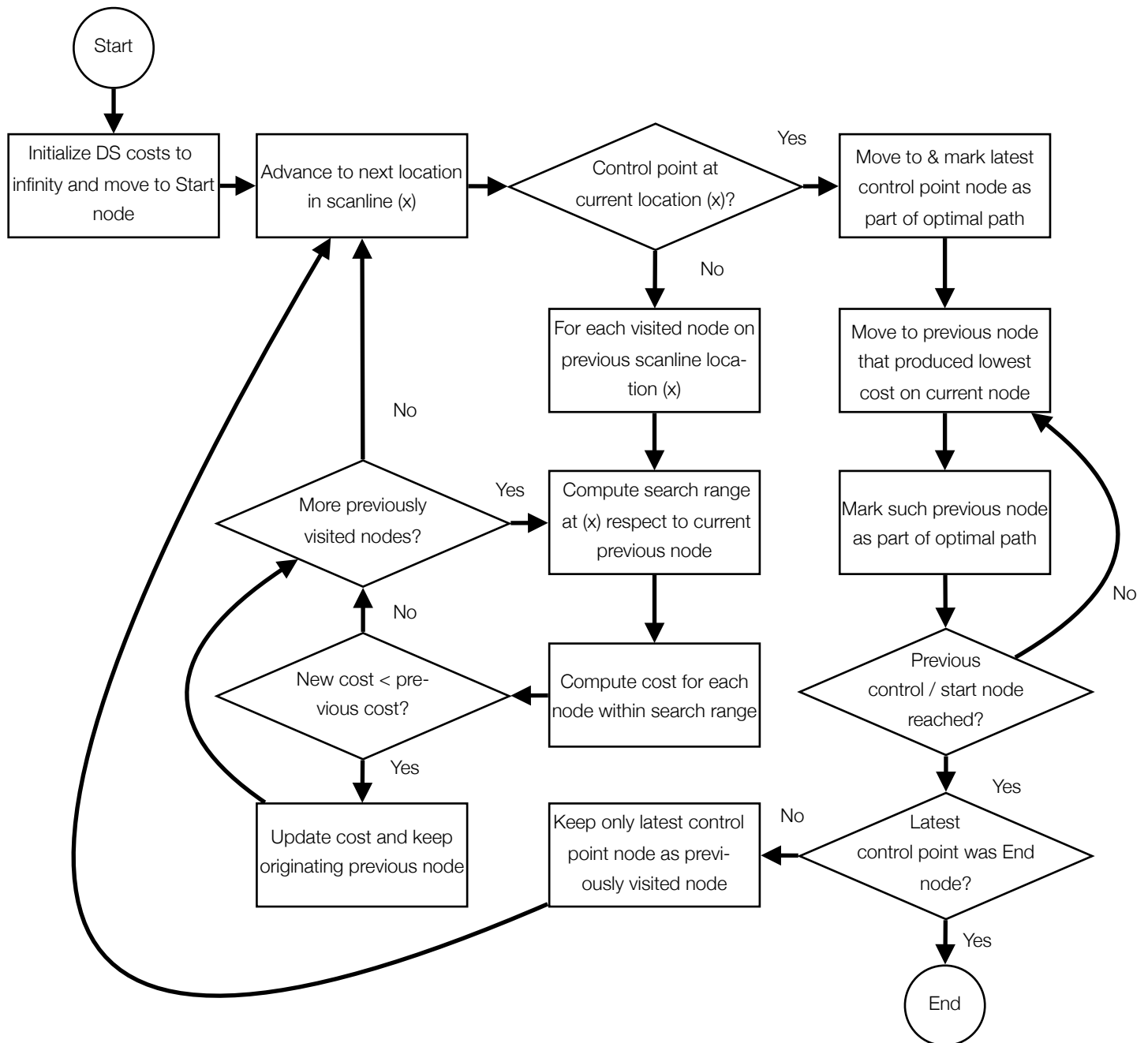
Oppositely to the Greedy's case, Dijkstra's does ensure that the obtained results will indeed be the minimum cost path on each scanline, however, Dijkstra's computation can be a lot more complex than Greedy's one. On the other side, since optimal minimum cost path is ensured, usually less number of control points are needed to attain the same quality on disparity estimation, which helps to reduce the computational load due to their computation.

In our implementation we proposed an implementation of Dijkstra's algorithm by assuming that each integer horizontal and vertical disparity value combination in the DS was a node in a network, as shown on fig. 9.30. The purpose was set to retrieve the minimum cost path without letting the algorithm go back in the network once a node had been fully traversed. In layman terms, we didn't allow the algorithm to create backwards paths. Besides, we also modified Dijkstra's algorithm by adaptively increasing or reducing the search area at each location depending on the reference image gradient at such location, just as we did on Greedy's implementation. This simplification allowed us to reduce the computational time of Dijkstra's algorithm, although at the expense of not ensuring always the optimal path. In any case, our experimental results proved that the results where very similar if the search area was correctly adapted, and the computational load was this way approximately 1.4 times smaller respect to a whole DS search.

In order to compute the minimum cost path we can see in fig. 9.30 that many more nodes need to be visited on consecutive scanline locations respect to Greedy's approach. Indeed besides the network of nodes also costs associated to reach each node from a node on a previous scanline location and references to such originating previous nodes need to be stored in memory behind the scenes to allow the algorithm to compute the optimal path. These reach costs are initialized to infinity before starting the algorithm and the references to originating nodes remain unset (or set to an invalid node reference).



Algorithm 9.7. Dijkstra's Dynamic Programming DS cost function minimization for a single scanline



node until current node, and again if the cost is lower than any previously available cost, it is updated and the previous node that produced such cost is kept. This whole process is repeated until a control or end node is reached. Then the scanline is traversed backwards marking the previous nodes that produced the lowest costs as the optimal minimum cost path. This process is depicted in algorithm 9.7, which shows how the DS for a single scanline is minimized. Finally, this process is repeated for each scanline on the reference image in order to obtain a dense disparity estimation.

In our experiments we found that the smoothness constraints and the robustness of Dijkstra's algorithm were so high, that sometimes it was better not to force the algorithm to pass through each control point, and instead just let it find the best cost path that was as close as possible to them. This is a desirable behavior since control points might also contain errors on their estimation, which we found that could usually be avoided this way.

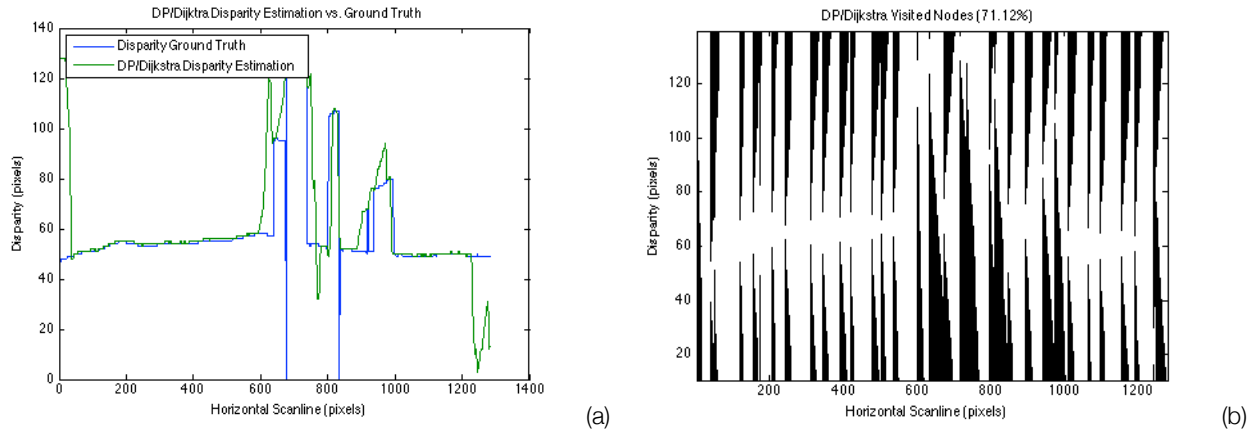


Fig. 9.31. (a) Dijkstra's result over a scanline on Middlebury's Aloe set [40] vs. disparity ground-truth. (b) DS Visited nodes where cost $E(u,v)$ was computed during scanline traversal (in white). Only 71.12% of nodes on the DS were visited.

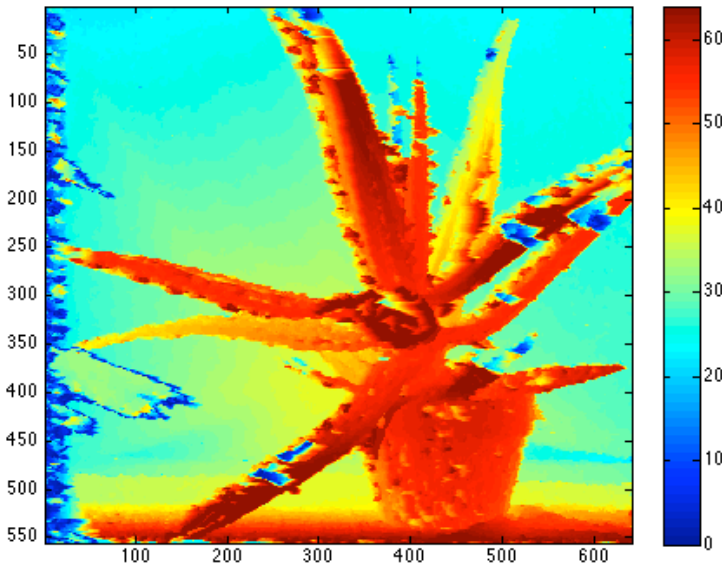


Fig. 9.32. Disparity estimation using Dijkstra's Dynamic Programming minimization algorithm. It can be observed that the "streaking" effect due to independent scanline minimization of the DS is less evident than in Greedy's approach.

Again, as in the case of Greedy's algorithm, some "streaking" might also appear because scanlines are optimized independently. However, due to the more robust nature of Dijkstra's results, usually this "streaking" can be almost neglected in most situations, as shown in fig. 9.32. In any case, this effect can be further reduced by also interpolating results and imposing smoothness restrictions between scanlines as we did on Greedy's case.

We also show in fig. 9.31 that although results are closer to the ground-truth respect to Greedy's results on fig. 9.28, also the number of visited nodes, and hence the computational time, is much larger than in Greedy's case. Even so, computational time is still smaller than in a typical WTA approach.

9.4.2.2.- Multiresolution

Multiresolution techniques are based on the analysis of pairs of images at different resolution levels starting from the coarsest resolution till the finest one, so that dense disparity is refined at each level, as seen on [5], [23], [36-39].

The main aspects to take into account on multiresolution techniques are to accurately create a pyramid of multiple resolution images, as well as to estimate optical flow on each resolution level with sub-pixel accuracy. Both of this issues will be explained on following sub-sections.

It will be seen that computing a multiresolution pyramid of images is not a trivial task, and usually involves efficient Gaussian filtering (i.e. using separable filter kernels) using precise parameters depending on the desired number of levels, which also



Fig. 9.33. Gaussian filtered images at different resolution levels. Top-left contains original image. Images become blurrier as they are repeatedly filtered.

need to be properly selected for the purpose of dense disparity estimation so that transition between scales becomes as smooth as possible for robust disparity estimation on each level.

Besides, also proper matching measures and sub-pixel accuracy on the matches will be required at each resolution level, so that when moving to a finer resolution level, correct matches can be obtained from matches on previous resolution level. Failing to obtain correct matches with sub-pixel accuracy will lead to wrong matches on finer resolution levels.

It must be noticed that if proper resolution pyramids, and sub-pixel accuracy are used, this method usually yields very good results. Indeed because of its multiresolution nature, it allows to correctly estimate optical flow even when a wide dynamic range of pixel displacement search is present between images, it is immune to false targets such as when texture repetition is present, it can cope with perspective projection differences (i.e. does not need robust matching by means of log-polar transforms) and doesn't require preresctification of input images (although if images are rectified then this technique can be computed slightly faster, or instead the algorithm can be restricted to search only within epipolar bands).

On the other hand, because of the way optical flow will be estimated on each resolution level, we can obtain a dense disparity field of the area of interest on input image with sub-pixel accuracy and at the same time, because of the match distance metric that will be employed (i.e. CV or NCC with some assumptions – see chapter 7), it is ensured that differences in gain and black-level on input images don't affect the output result while at the same time retrieving a data confidence map that indicates how good disparity estimations will be on each pixel location.

Finally, it must be noticed, that control points or an initial disparity estimation could also be provided to the algorithm if matches for some points are accurately known in advance. This will help to further increase quality of results due to the smoothness constraint implicitly imposed by the multiple resolution levels, although in general this is never needed and initial disparity for the coarsest resolution level is assumed to be zero. However, this idea can also be used when analyzing video sequences in order to obtain a faster estimation of disparity for consecutive video frames by starting on a somewhat fine resolution level using the estimated disparity on a previous pair of frames

9.4.2.2.1.- Gaussian filtering and illumination invariant matching

The multidimensional Gaussian expression is a typical filter kernel that is employed to smooth images. It has several advantages respect to other filters such as a simple square filter, which are summarized below:

- pixels located away from the location designed as the center of the Gaussian are considered less important. This will be specially useful when using weighted NCC or CV matching metrics as it will ensure that the scores we obtain are as smooth as possible when searching for best matches and new objects appear within matching windows.

- Gaussian filtering is separable, which means that it can be computed first horizontally and then vertically (or the other way around), with a significantly lower computational cost respect to other non-separable 2D filters.
- It can be used to build a resolution pyramid of images because Gaussian filtering behaves as a low-pass filter, and hence because of the Nyquist sampling theorem we can safely decimate any low-pass filtered signal.

The expression of a typical Gaussian kernel centered at the origin (0,0) is:

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (9.106)$$

where x and y are the horizontal and vertical coordinates respectively and σ is the parameter that controls Gaussian width (i.e. the lower σ is the narrower the Gaussian function becomes).

It must be noticed that expression (9.106) can be decomposed as:

$$G(x, y, \sigma) = g(x, \sigma) \cdot g(y, \sigma) \quad (9.107)$$

taking into account that

$$g(t, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{t^2}{2\sigma^2}} \quad (9.108)$$

Hence, it is easily observed that Gaussian kernels are separable, which translates into faster filtering as a 2D filtering can be computed as filtering twice using two 1D filters (one horizontal filter and one vertical filter).

However, when building a Gaussian multiresolution pyramid, it can be observed that information on different levels of resolution becomes redundant. In other words, finer resolution levels contains the same information within coarser levels plus further details, as depicted on fig. 9.33.

Besides, for easier matching, removal of DC components when images have different levels of illumination can also improve the resulting match. In that sense CV match seen on chapter 7 already removes the DC component for each matching computation. However, we can observe that by removing the DC component of images before matching, then NCC (see chapter 7) matches become equivalent to CV, which are illumination invariant, and also NCC matches are faster to compute than CV ones. Hence, after this dissertation, it seems logical to remove DC from Gaussian filtered images before matching, which will result on a pass-band filtering on each resolution level, which is similar to wavelets decomposition. This is shown next.

Taking a Gaussian filtered image from one of the resolution levels of the multiple resolution pyramid (we will see later how to build this pyramid):

$$I_G(x, y) = I(x, y) * G(x, y, \sigma_s) \quad (9.109)$$

where σ_s is the width of the Gaussian on such resolution level.

If we retrieve the DC component of expression (9.109) by using again a second Gaussian filtering with σ_w , which we know that behaves as a low-pass band filtering, then we obtain:

$$I_{G_{DC}}(x, y) = I_G(x, y) * G(x, y, \sigma_w) \quad (9.110)$$

where now σ_w is a parameter that allows retrieval of the DC component and is usually related to σ_s by the following expression:

$$\sigma_w = \lambda \cdot \sigma_s \quad (9.111)$$

where λ is typically a value around $\lambda = 4.08$ as shown experimentally in [5].

Then, by using (9.109), (9.110) and (9.111) the image at a given resolution level without DC becomes:

$$I_{NO-DC}(x,y) = I_G(x,y) - I_{GDC}(x,y) = I(x,y) * G(x,y,\sigma_s) * (1 - G(x,y,\lambda \cdot \sigma_s)) \quad (9.112)$$

Then, it can be observed that this is equivalent to using the following filter:

$$h(x,y) = G(x,y,\sigma_s) * (1 - G(x,y,\lambda \cdot \sigma_s)) = h_1(x,y) * h_2(x,y) \quad (9.113)$$

We now want to obtain the frequency responses of $h_1(x,y)$ and $h_2(x,y)$. Taking into account that the Fourier transform of (9.108) is

$$\hat{G}(\omega,\sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{\sigma^2 \omega^2}{2}} \quad (9.114)$$

Then the frequency response of $h_1(x,y)$, taking into account that Gaussian filtering is separable, immediately becomes:

$$\hat{H}_1(\omega_x,\omega_y) = \hat{G}(\omega_x,\sigma_s) \cdot \hat{G}(\omega_y,\sigma_s) = \frac{1}{2\pi\sigma_s^2} e^{-\frac{\sigma_s^2(\omega_x^2 + \omega_y^2)}{2}} \quad (9.115)$$

and the frequency response of $h_2(x,y)$ is:

$$\hat{H}_2(\omega_x,\omega_y) = 1 - \hat{G}(\omega_x,\lambda \cdot \sigma_s) \cdot \hat{G}(\omega_y,\lambda \cdot \sigma_s) = 1 - \frac{1}{2\pi\lambda^2\sigma_s^2} e^{-\frac{\lambda^2\sigma_s^2(\omega_x^2 + \omega_y^2)}{2}} \quad (9.116)$$

Then, by using the convolution property of the Fourier transform, the frequency response of (9.113) is:

$$\hat{H}(\omega_x,\omega_y) = \hat{H}_1(\omega_x,\omega_y) \cdot \hat{H}_2(\omega_x,\omega_y) = \frac{1}{2\pi\sigma_s^2} e^{-\frac{\sigma_s^2(\omega_x^2 + \omega_y^2)}{2}} \cdot \left(1 - \frac{1}{2\pi\lambda^2\sigma_s^2} e^{-\frac{\lambda^2\sigma_s^2(\omega_x^2 + \omega_y^2)}{2}} \right) \quad (9.117)$$

Then, by looking at the previous expression, we can easily observe that the resulting resolution level image without DC component behaves as a pass-band filtered version of the input image. This will be shown analytically for the 1D case, as expression (9.117) has revolution geometry.

For simplification purposes we can take the 1D version of the frequency response of $h(x,y)$ in order to analytically obtain the exact pass-band frequency. Hence, the filter in this case becomes:

$$h(t) = h_1(t) * h_2(t) = g(t,\sigma_s) * (1 - g(t,\lambda \cdot \sigma_s)) \quad (9.118)$$

and its frequency response is:

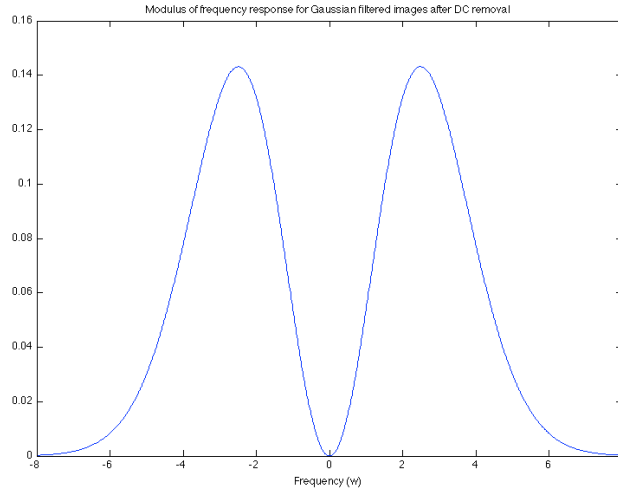


Fig. 9.35. Frequency response of the resulting filter of Gaussian filtered images after DC removal in 1D. Bands are centered at $\pm 2\sqrt{\pi}$ because $\sigma_w = 1/\sqrt{2\pi}$. Also we have used the following values: $\lambda = \sqrt{2/\pi}$ and $\sigma_s = 0.5$

$$\hat{H}(\omega) = H_1(\omega) \cdot H_2(\omega) = \frac{1}{\sqrt{2\pi}\sigma_s} e^{-\frac{\sigma_s^2 \omega^2}{2}} \cdot \left(1 - \frac{1}{\sqrt{2\pi}\lambda\sigma_s} e^{-\frac{\lambda^2 \sigma_s^2 \omega^2}{2}} \right) \quad (9.119)$$

A depiction of expression (9.119) is shown in figure 9.35. Then, in order to find where the maximum response is located, we take the derivatives of $h_1(t)$ and $h_2(t)$ over ω , obtaining:

$$\frac{\partial \hat{H}_1(\omega)}{\partial \omega} = \frac{-\sigma_s \omega}{\sqrt{2\pi}} e^{-\frac{\sigma_s^2 \omega^2}{2}} \quad (9.120)$$

$$\frac{\partial \hat{H}_2(\omega)}{\partial \omega} = \frac{\lambda \sigma_s \omega}{\sqrt{2\pi}} e^{-\frac{\lambda^2 \sigma_s^2 \omega^2}{2}} \quad (9.121)$$

Hence, the derivative of (9.119) over ω becomes:

$$\begin{aligned} \frac{\partial \hat{H}(\omega)}{\partial \omega} &= \frac{\partial \hat{H}_1(\omega)}{\partial \omega} \hat{H}_2(\omega) + \hat{H}_1(\omega) \frac{\partial \hat{H}_2(\omega)}{\partial \omega} = \\ &= \frac{-\sigma_s \omega}{\sqrt{2\pi}} e^{-\frac{\sigma_s^2 \omega^2}{2}} \cdot \left(1 - \frac{1}{\sqrt{2\pi}\lambda\sigma_s} e^{-\frac{\lambda^2 \sigma_s^2 \omega^2}{2}} \right) + \frac{1}{\sqrt{2\pi}\sigma_s} e^{-\frac{\sigma_s^2 \omega^2}{2}} \cdot \frac{\lambda \sigma_s \omega}{\sqrt{2\pi}} e^{-\frac{\lambda^2 \sigma_s^2 \omega^2}{2}} = \\ &= \frac{\sigma_s \omega}{\sqrt{2\pi}} e^{-\frac{\sigma_s^2 \omega^2}{2}} \left(\frac{1}{\sqrt{2\pi}\sigma_s} \left(\frac{1}{\lambda} + \lambda \right) e^{-\frac{\lambda^2 \sigma_s^2 \omega^2}{2}} - 1 \right) \end{aligned} \quad (9.122)$$

And by equating (9.122) to zero $\frac{\partial \hat{H}(\omega)}{\partial \omega} = 0$ we obtain the minima and maxima frequency responses. We can directly

observe that (9.122) becomes zero for $\omega = 0$ and $\omega = \infty$ just by looking at the first couple of factors. From fig. 9.35 we can

observe that these frequencies belong to the minima of the frequency response. Hence, the remaining factor will correspond to the maxima. Consequently:

$$\begin{aligned}
 \frac{1}{\sqrt{2\pi}\sigma_s} \left(\frac{1}{\lambda} + \lambda \right) e^{-\frac{\lambda^2 \sigma_s^2 \omega^2}{2}} - 1 &= 0 \Rightarrow \\
 e^{-\frac{\lambda^2 \sigma_s^2 \omega^2}{2}} &= \frac{1}{\frac{1}{\sqrt{2\pi}\sigma_s} \left(\frac{1}{\lambda} + \lambda \right)} = \frac{\sqrt{2\pi}\lambda\sigma_s}{1 + \lambda^2} \Rightarrow \\
 -\frac{\lambda^2 \sigma_s^2 \omega^2}{2} &= \ln \left(\frac{\sqrt{2\pi}\lambda\sigma_s}{1 + \lambda^2} \right) \Rightarrow \\
 \omega^2 &= -\frac{2}{\lambda^2 \sigma_s^2} \ln \left(\frac{\sqrt{2\pi}\lambda\sigma_s}{1 + \lambda^2} \right)
 \end{aligned} \tag{9.123}$$

Then, the maxima of the frequency response of Gaussian filtered images on the multiple resolution pyramid after DC removal is:

$$\omega = \pm \frac{1}{\lambda\sigma_s} \sqrt{-2 \ln \left(\frac{\sqrt{2\pi}\lambda\sigma_s}{1 + \lambda^2} \right)} \tag{9.124}$$

For simplification, we can take the Taylor series of the logarithm term, taking into account that:

$$\ln(1+x) \approx x, \quad |x| \leq 1, \quad x \neq 1 \tag{9.125}$$

Hence, we obtain:

$$\ln \left(\frac{\sqrt{2\pi}\lambda}{1 + \lambda^2} \sigma_s \right) = \ln \left(1 + \frac{\sqrt{2\pi}\lambda}{1 + \lambda^2} \sigma_s - 1 \right) \approx \frac{\sqrt{2\pi}\lambda}{1 + \lambda^2} \sigma_s - 1 \tag{9.126}$$

where the x term is $x = \frac{\sqrt{2\pi}\lambda}{1 + \lambda^2} \sigma_s - 1$. And consequently we obtain the following restriction on σ_s :

$$|x| \leq 1 \Rightarrow \frac{\sqrt{2\pi}\lambda}{1 + \lambda^2} \sigma_s - 1 \leq 1 \Rightarrow \sigma_s \leq \frac{2(1 + \lambda^2)}{\sqrt{2\pi}\lambda} \tag{9.127}$$

For typical values of $\lambda = 4.08$ [5] as we have mentioned before, this results on a limit on σ_s of $\sigma_s \leq 3.45$. Hence, after using Taylor series on the logarithm term, expression (9.124) becomes:

$$\omega = \pm \frac{1}{\lambda\sigma_s} \sqrt{-2 \ln \left(\frac{\sqrt{2\pi}\lambda\sigma_s}{1 + \lambda^2} \right)} \approx \pm \frac{1}{\lambda\sigma_s} \sqrt{2} \sqrt{1 - \frac{\sqrt{2\pi}\lambda}{1 + \lambda^2} \sigma_s} \tag{9.128}$$

By using again Taylor series, but this time on the square root using the following approximation:

$$\sqrt{1+x} \approx 1 + \frac{1}{2}x, \quad |x| < 1 \quad (9.129)$$

Then, by taking $x = -\frac{\sqrt{2\pi}\lambda}{1+\lambda^2}\sigma_s$, we obtain from expression (9.128):

$$\omega \approx \pm \frac{1}{\lambda\sigma_s} \sqrt{2} \left(1 - \frac{1}{2} \frac{\sqrt{2\pi}\lambda}{1+\lambda^2} \sigma_s \right) \quad (9.130)$$

where now under the constraint $|x| < 1$ we must ensure that:

$$\left| -\frac{\sqrt{2\pi}\lambda}{1+\lambda^2} \sigma_s \right| < 1 \Rightarrow \sigma_s < \frac{1+\lambda^2}{\sqrt{2\pi}\lambda} \quad (9.131)$$

which for a typical value of $\lambda = 4.08$ [5] we obtain that $\sigma_s < 1.73$. Which results in a stricter constraint than the previous one that we obtained on (9.127).

Finally we can observe that,

$$\omega \approx \pm \frac{1}{\lambda\sigma_s} \sqrt{2} \left(1 - \frac{1}{2} \frac{\sqrt{2\pi}\lambda}{1+\lambda^2} \sigma_s \right) = \pm \left(\frac{\sqrt{2}}{\lambda\sigma_s} - \frac{\sqrt{\pi}}{1+\lambda^2} \right) \quad (9.132)$$

where typically the last term is substantially smaller than the first one. Hence we can further simplify expression (9.132) as follows:

$$\omega \approx \pm \left(\frac{\sqrt{2}}{\lambda\sigma_s} - \frac{\sqrt{\pi}}{1+\lambda^2} \right) \approx \pm \frac{\sqrt{2}}{\lambda\sigma_s} = \pm \frac{\sqrt{2}}{\sigma_w} \quad (9.133)$$

As will also be seen later, when using resolution pyramids, decimation will also be involved. This will translate into fixed σ_s and σ_w on each resolution level, however, the effective sigmas will depend on the amount of decimation as well. This will ultimately translate into the fact that the multiple resolution pyramid of images without DC component behave as a pass-band filter bank, which is a similar concept that is applied in the field of wavelets.

As a final note, notice that Gaussian expressions will be sampled into vectors of finite size for filtering purposes on a machine. In order to contain the whole Gaussian transitory shape, it is usual to take filter kernels having a size of at least $3\sigma \times 3\sigma$. Smaller matrices won't contain the whole transitory shape, whereas longer ones will result in unnecessary higher computational cost when filtering images.

9.4.2.2.2.- Building multiple resolution pyramids

As we have already seen on section 9.4.2.2.1, we know that we can create multiple resolution pyramids by means of Gaussian filtering of any input image.

It remains, though, to define how the scale factor should be chosen to create the multiple resolution pyramid as well as the amount of Gaussian filtering defined by its σ parameter.

Because we know that Gaussian filtering behaves as a low pass filter, by the sampling theorem we know that we can decimate images by a given scale factor between resolution levels without losing any signal information.

Besides we also know that when consecutively applying Gaussian filtering, as shown below:

$$I(x,y) * G(x,y,\sigma_0) * G(x,y,\sigma_c) = I(x,y) * G(x,y,\sigma_1) \quad (9.134)$$

then it is equivalent to filter using a Gaussian kernel with σ equal to:

$$\sigma_1^2 = \sigma_0^2 + \sigma_s^2 \quad (9.135)$$

where σ_s is the amount of Gaussian filtering applied on a single scale level.

Hence, in order to efficiently construct a Gaussian multiple resolution pyramid, we will repeatedly filter using a kernel with σ_s on each resolution level, and then we will decimate by a scale factor ρ , as shown on fig. 9.36. Hence, the resulting blur σ_i for level i will be:

$$\sigma_i = \frac{1}{\rho} \sqrt{\sigma_{i-1}^2 + \sigma_s^2} \quad (9.136)$$

For matching purposes it would be desirable that each resolution level preserves the same apparent blur. Hence,

$$\sigma_0 = \sigma_1 = \sigma_2 = \dots = \sigma_{i-1} = \sigma_i \quad (9.137)$$

Consequently, combining expressions (9.136) and (9.137) we obtain that the amount of blur that must be repeatedly applied between consecutive resolution levels is:

$$\sigma_s = \sigma_0 \sqrt{\rho^2 - 1} \quad (9.138)$$

It must be noticed that because we decimate resulting images between consecutive resolution levels, their apparent blur remains constant. However, if we didn't decimate them, the effective blur on each resolution level would be:

$$\sigma_{effective} = \sigma_0 \cdot \rho^i \quad (9.139)$$

where i is the scale number, being zero the finest resolution level as depicted on fig. 9.36. It can be observed then that as we compute further resolution levels, we are indeed filtering with a smaller low-pass band for Gaussian filtering or with a pass-band located closer to zero (see expression 9.133) if we remove the DC component of Gaussian filtered images. In both cases the maximum frequency present of images after filtering gets reduced on each resolution level, and for that reason we can decimate using ρ scale factor.

It must be observed that on expressions (9.134) to (9.139) we have assumed that σ_0 is known. This is not exactly true as will be shown next. Indeed, any picture taken by any given camera has some inherent blur due to the camera lens or acquisition sensor. This blur could be directly considered as σ_0 , however it usually happens that this inherent blur cannot be known in advance if the camera specifications are unknown. For that reason we usually apply some initial blur σ_{init} that can almost be neglected if applied to the original image, but that makes expressions (9.134) to (9.139) to remain valid, and hence the blur on level zero σ_0 is assumed to be:

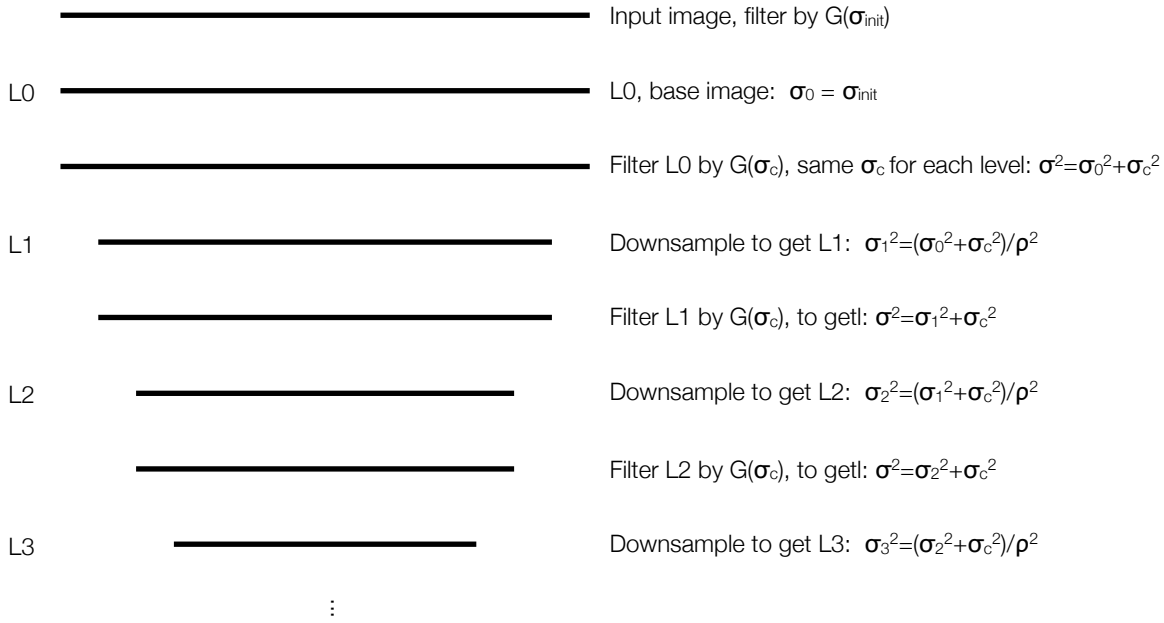


Fig. 9.36. Building a multiple resolution pyramid

$$\sigma_0 = \sqrt{\sigma_{init}^2 + \sigma_{image}^2} \quad (9.140)$$

where typical values for σ_{init} are $\sigma_{init} = 0.5$ and σ_{image} is unknown, and hence usually considered to be $\sigma_{image} = 0$.

Then, up to this point the only parameters left free to choose are the number of desired levels and the scale factor, being the latter the most important one. Typically the scale factor is 2 (one octave) or $\sqrt{2}$ (half octave), and the number of levels usually depends on the chosen scale factor so that for typical images the number of levels is such that image size at the coarsest resolution level becomes about the same size of the window being used for matching at a single level (using either NCC or CV). Consequently, when using one octave scale factor the number of levels is about 7 and for half octave levels is about 15 for typical images of size 1024x768.

However, even if the scale factor is properly chosen so that transition between consecutive resolution levels is as smooth as possible, some people complain that this is not enough (especially if the scale factor is too large, as is the case for octave or half octave). In those cases a semi-pyramid of multiple resolutions can be build instead, as will be shown next.

9.4.2.2.3.- Building semi-pyramids of multiple resolution

The concept behind semi-pyramids of multiple resolution is very similar to any pyramid of multiple resolution, but having additional levels of resolution which are called inter-levels.

These inter-levels are located between each level of resolution, as shown on fig. 9.37, on any typical pyramid of resolutions and contain images with the same size as its previous finer resolution level (hence, they have not been decimated), but with some additional blur that slowly increases as shown in the following expression:

$$\sigma_n = \sigma_0 \sqrt[n]{\rho^n}, \quad n \in 1, \dots, N-1 \quad (9.141)$$

where N is the number of inter-levels between each pair of levels, and n is the current inter-level taking into account that n=0 corresponds to the previous finer resolution level and n=N corresponds to the next coarser resolution level.

In order to further reduce the computational complexity when computing consecutive inter-levels we can take into account that:

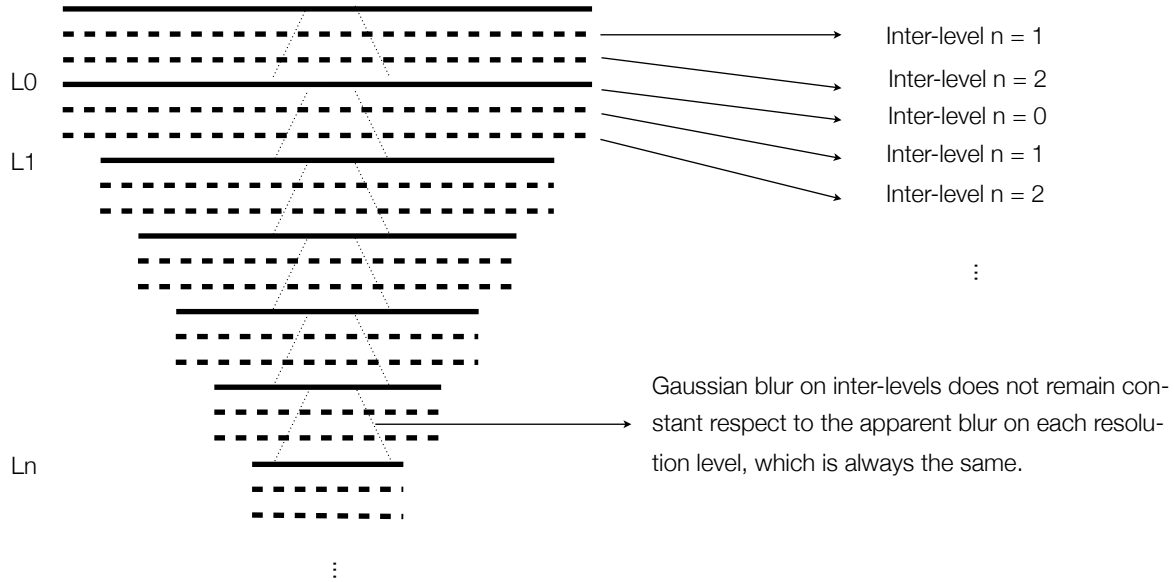


Fig. 9.37. A multiple resolution semi-pyramid

$$\sigma_n = \sigma_0 \cdot \rho^{\frac{n}{N}}, \quad \sigma_{n+1} = \sigma_0 \cdot \rho^{\frac{n+1}{N}} \quad (9.142)$$

And by adapting expression (9.142) to (9.141) we obtain that:

$$\sigma_{n+1}^2 = \sigma_n^2 + \sigma_{sub}^2 \quad (9.143)$$

And consequently the amount of blurring that must be applied between consecutive inter-levels is:

$$\sigma_{sub} = \sigma_0 \cdot \sqrt{\rho^{\frac{2n+2}{N}} - \rho^{\frac{2n}{N}}} \quad (9.144)$$

9.4.2.2.4.- Matching at a single scale.

Once the multiple resolution pyramids are defined for a given pair of images to be matched, we can proceed to match them at each resolution level. In order to match pairs of images we will use a CV matching metric using Gaussian weighted windows with. Hence, by using expression of D_{CV} on chapter 7 we obtain:

$$D_{CV} = \frac{\sum_{(i,j) \in U} \left(w^2(i,j) \cdot \left(I_L(x+i, y+j) - \overline{I_L(x,y)} \right) \cdot \left(I_R(x+d_x+i, y+d_y+j) - \overline{I_R(x+d_x, y+d_y)} \right) \right)}{\sqrt{\sum_{(i,j) \in U} \left(w(i,j) \cdot \left(I_L(x+i, y+j) - \overline{I_L(x,y)} \right) \right)^2 \cdot \sum_{(i,j) \in U} \left(w(i,j) \cdot \left(I_R(x+d_x+i, y+d_y+j) - \overline{I_R(x+d_x, y+d_y)} \right) \right)^2}} \quad (9.145)$$

where $w(i,j)$ stands for the Gaussian weighted window, which is expressed as shown below:

$$w(i,j) = \frac{1}{2\pi\sigma_w^2} e^{-\frac{i^2+j^2}{2\sigma_w^2}} \quad (9.146)$$

where it can be observed that $w(i,j)$ is a Gaussian expression centered at the origin that uses the σ_w seen on (9.111).

It must be noticed however that when working with images where the DC component has been removed, then expression (9.145) can be further simplified until we get a typical NCC matching metric, which is faster to compute than CV. This is shown below:

$$D_{NCC} = \frac{\sum_{(i,j) \in U} \left(w^2(i,j) \cdot I_L(x+i, y+j) \cdot I_R(x+d_x+i, y+d_y+j) \right)}{\sqrt{\sum_{(i,j) \in U} \left(w(i,j) \cdot I_L(x+i, y+j) \right)^2 \cdot \sum_{(i,j) \in U} \left(w(i,j) \cdot I_R(x+d_x+i, y+d_y+j) \right)^2}} \quad (9.147)$$

As in CV matching metric, NCC matching scores will range from -1 to 1, being 1 the best possible matching. For that reason, if we build a cost function of the form $E(u(x,y), v(x,y)) = D_{NCC}(u(x,y), v(x,y))$, then we will need to **maximize** such cost function, otherwise, if we wanted to minimize the cost function as in section 9.3, $E(u,v)$ should be negated.

Notice that other constrains such as the smoothness one can be added as it was explained on section 9.3, however, due to the multiresolution nature of this technique, the smoothness constraint will be mostly unnecessary as it is implicitly imposed.

Other restrictions such as the epipolar one can also be imposed if epipolar geometry has been robustly estimated beforehand, however, because results provided by this algorithm are very robust, it is usually more convenient to do it the other way around and estimate first the optical flow without pre-rectifying images, and then use the estimated dense optical flow to robustly find the epipolar geometry. This way results are usually more precise and less computational intensive because epipolar geometry becomes easier to compute due to the consistent nature of the estimated optical flow.

Once the metric to be used for matching at a given resolution level is defined, it also needs to be defined the search area to use in order to obtain the best pixel matching. In that sense, it has been empirically observed that when matching two similar signals using correlation, the obtained result has a shape similar to the one depicted on fig. 9.38. It can be observed that within a given limit $\pm D_{\max}$ the correlation behaves monotonically and can be approximated to the shape of a parabola. This fact will help us to search for the correct disparity value as long as we lay within the $\pm D_{\max}$ limit by means of polynomial curve fitting. Experimentally [5] it has been obtained that a typical value for $D_{\max}$ is:

$$D_{\max} = 1 \quad (9.148)$$

Because a search range of $D_{\max}$ is done on each resolution level, it can be observed that the total search range throughout the whole multiple resolution pyramid will be of:

$$D_{Total} = D_{\max} \cdot \rho^k \quad (9.149)$$

where k indicate the number of levels from coarsest to finest, and ρ stands for the scale factor between resolution levels.

Then in order to search for best match on each reference pixel, we start from an initial known optical flow, which initially on the coarsest resolution level can be set to zero if it is not known, and on subsequent finer resolution levels, it can be the expanded optical flow obtained on a previous resolution level. This way optical flow gets refined as we traverse each resolution level.

Hence, starting from a known displacement for a given matched pair of pixels, we will check for horizontal and vertical displacements located at $\pm D_{\max}$ as shown on fig. 9.39. Once the NCC scores for each location have been computed, we can fit two parabolas (one for the horizontal disparities and the other for the vertical ones). Notice that curve fitting for the parabolas is explained for the horizontal case, but the vertical one is analogous.

Then, starting from the following parabola equation:

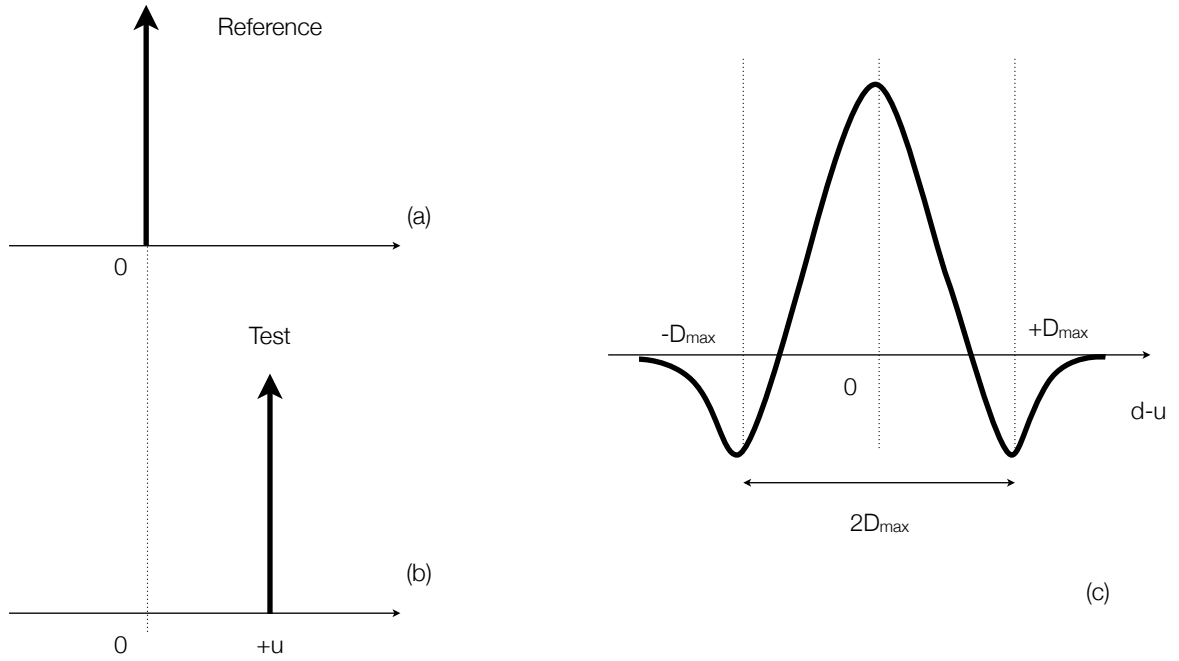


Fig. 9.38. (a) Delta. (b) Time-shifted delta. (c) Correlation of both deltas. At point close to the displacement there exists a maximum on correlation output. Besides, close to that area correlation decreases approximately monotonically until $\pm D_{\max}$.

$$c(x) = ax^2 + bx + c \quad (9.150)$$

Since we know the NCC matching scores at $x = -D_{\max}$, 0 and $+D_{\max}$ (we move x to the origin to simplify computations, this can later be undone), we can then place the following linear system of equations:

$$\begin{aligned} c(0) &= c \\ c(D_{\max}) &= a \cdot D_{\max}^2 + b \cdot D_{\max} + c \\ c(-D_{\max}) &= a \cdot D_{\max}^2 - b \cdot D_{\max} + c \end{aligned} \quad (9.151)$$

Then, by defining the following intermediate variables:

$$\begin{aligned} \alpha &= c(D_{\max}) - c(0) = a \cdot D_{\max}^2 + b \cdot D_{\max} \\ \beta &= c(-D_{\max}) - c(0) = a \cdot D_{\max}^2 - b \cdot D_{\max} \end{aligned} \quad (9.152)$$

We obtain that the parabola parameters (a, b, c) are:

$$a = \frac{\alpha + \beta}{2D_{\max}^2}, \quad b = \frac{\alpha - \beta}{2D_{\max}}, \quad c = c(0) \quad (9.153)$$

Finally, we know that if parameter a is negative, then the parabola will have a global maximum, which will be located at:

$$d_x = -\frac{b}{2a}, \quad a < 0 \quad (9.154)$$

It must be observed that expression (9.154) is obtained from equating to zero the derivative of (9.150). Hence, expression (9.154) will find both local maxima and minima. However, we know that maxima will be present only when a is negative.

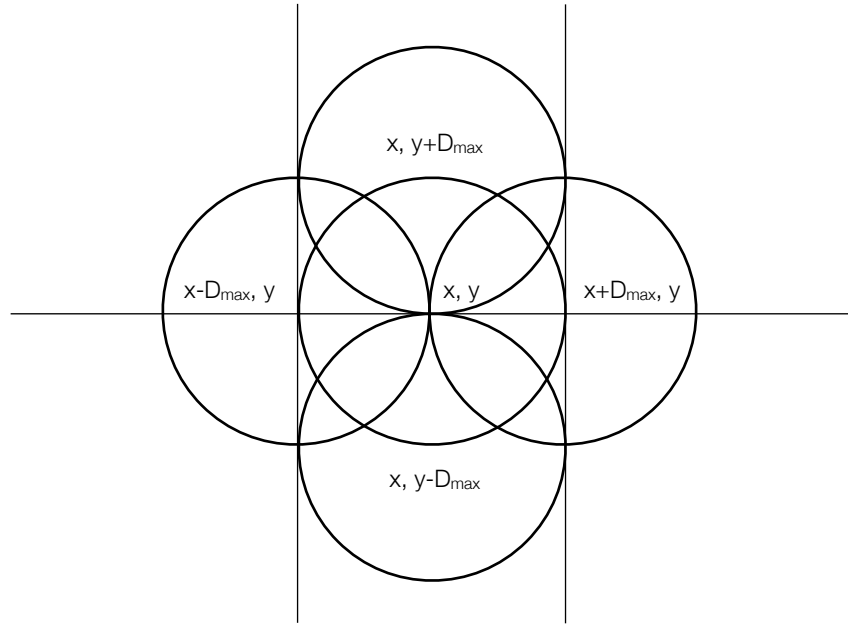


Fig. 9.39. Search locations within a given resolution level

Because dense disparity will be found iteratively on each resolution level, we need to impose certain constraints in order to avoid getting stuck at minima due to expression (9.154). In that sense we will define the following rules:

1. If $a < 0$ and d_x is in range $\pm 1.5 \cdot D_{\max}$ then d_x is the correct disparity estimate for the current location.
2. If $a < 0$ and d_x is outside of range, $\pm 1.5 \cdot D_{\max}$ then we are probably located on one side of the parabola and we need to move towards its maximum to obtain a more accurate measure. Hence we shift current location by $\pm 1.5 \cdot D_{\max}$ (sign is chosen so that we move towards maximum) and continue to refine as in step 1 in subsequent iterations.
3. If $a > 0$ then we have found a local minima. We need to move out of this minimum in order to avoid getting stuck on it. Hence, we check for maxima at $\pm D_{\max}$, shift current location by $\pm 1.5 \cdot D_{\max}$ and continue to refine as in step 1 in subsequent iterations.
4. If $a = 0$, then no refinement is possible as we have no clues where search should be done. In such case we set $d_x = 0$.

Finally notice that because we shifted the coordinates to the origin before placing the linear system of equations, then d_x only contains the sub-pixel displacement part. In order to retrieve the real pixel location on the right view then:

$$\tilde{x}_R = x_R + d_x \quad (9.155)$$

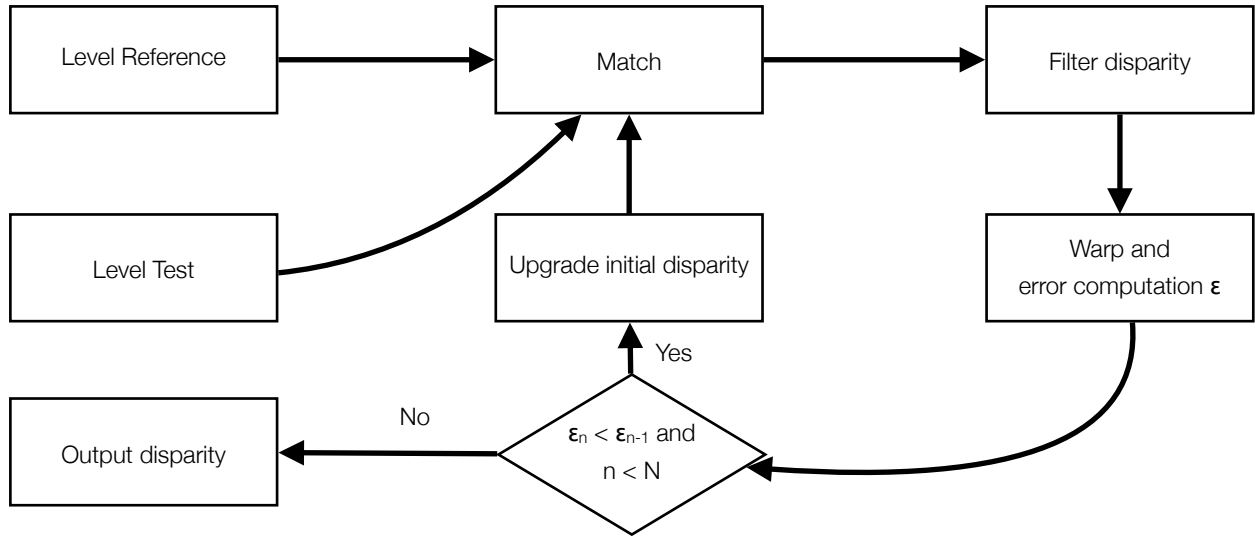
And consequently the horizontal optical flow for a given pixel location (x, y) on the reference image becomes:

$$u(x, y) = \tilde{x}_R(x, y) - x = x_R(x, y) + d_x(x, y) - x \quad (9.156)$$

On the other hand, as we mentioned before, NCC or CV matching scores range between -1 and 1, hence their values can be used as a measure of confidence indicating how good are the matches we have found (i.e. the closer to one the better). Because we have fitted a parabola over the matching scores and we know that the optimal match is located at d_x , then the confidence value can be obtained as the fitted matching cost at such location:

$$C = c(d_x) \quad (9.157)$$

Algorithm 9.8. Dense disparity refinement on a single scale level.



However, expression (9.157) will only be valid for rectified images, where no vertical search is done. Otherwise, we would have both horizontal and vertical confidences, one for each fitted parabola. In such cases, a typical approach consists on averaging both measures so that:

$$C = \frac{1}{2} \left(c(d_x) + c(d_y) \right) \quad (9.158)$$

Finally, this method of fitting parabolas to matching scores when searching for best match will be repeated for each pixel on the reference image in order to obtain a dense disparity estimation and confidence map.

9.4.2.2.5.- Refining optical flow at a single scale

Up to this point we have seen how to obtain a dense disparity estimation on a given resolution level and how to avoid getting stuck on wrong disparity estimations located at local DS minima. As we saw in previous section, disparities are estimated to be placed at local DS maxima by fitting parabolas. However, because displacements $D_{\max}$ are small, it might be necessary to repeat the whole process to ensure that optimal disparity results are obtained by iteratively curve fitting until convergence to the most plausible solution. For that reason we propose to repeat the whole dense disparity estimation on a single resolution level following the diagram shown on algorithm 9.8.

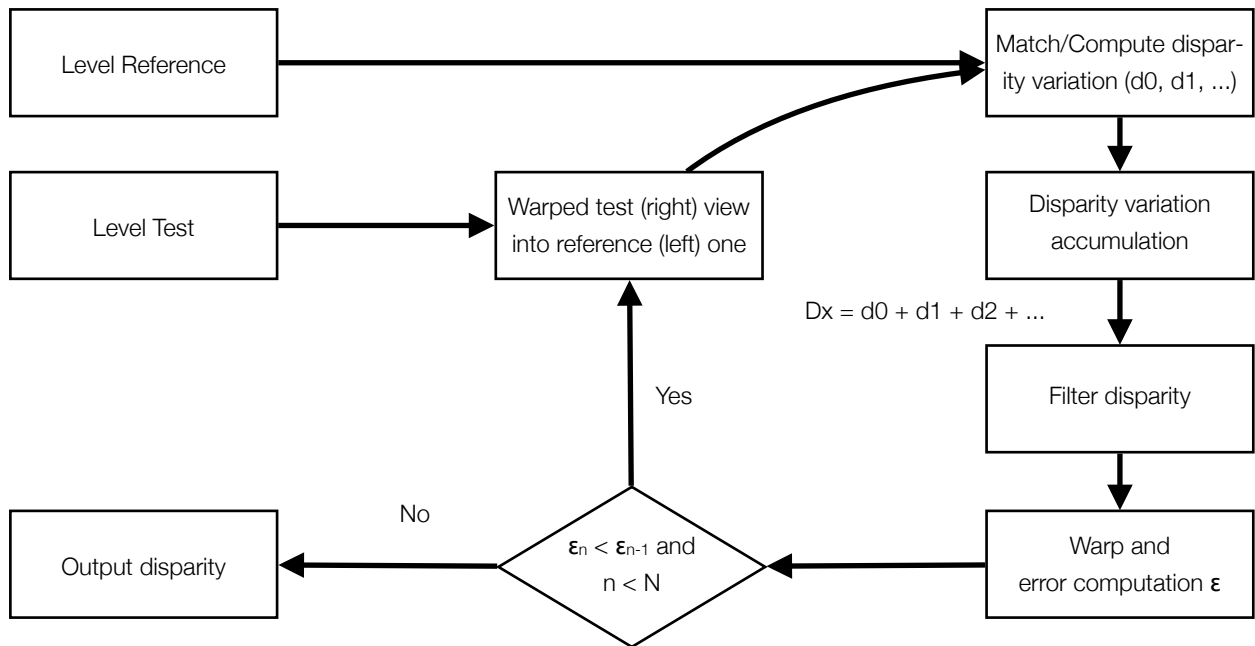
As shown on the diagram the purpose will be to iteratively upgrade the disparity by using the different (dx, dy) estimations on each iteration as long as the total error ϵ as defined below is minimized.

$$\epsilon = \sqrt{\frac{\sum_{x=1}^X \sum_{y=1}^Y \left(I_L(x, y) - I_R(x + D_x(x, y), y + D_y(x, y)) \right)^2}{X \cdot Y}} \quad (9.159)$$

where ϵ accounts for the total error after warping right view into left view, I_L for the left view image, I_R for the right view image, D_x for the horizontal disparity field, and D_y for the vertical disparity field. Notice that $x + D_x(x, y)$ and $y + D_y(x, y)$ might not be integer values. In such cases, in order to obtain a more precise measure of ϵ , linear pixel interpolation can also be used.

For practical reasons, a maximum number of iterations N is defined on each resolution level to avoid the error to be minimized indefinitely in some situations. This maximum number of iterations is typically 5.

Algorithm 9.9. Dense disparity refinement on a single scale level using disparity feedback.



To avoid spurious results it is also proposed to smooth dense disparity results on each resolution level. However, to ensure that object boundaries are preserved and do not become too much blurry, it is also proposed to use an anisotropic diffusion filter on the obtained disparities on each iteration as shown on algorithm 9.8. Our experiments proved that a typical median filtering over the estimated disparity will work fine in most situations, however, other anisotropic solutions such as [46] can be used instead.

After iterating the proper number of times, the obtained dense disparity on current resolution level is passed as an initialization for the next finer resolution level as will be seen on section 9.4.2.2.8.

9.4.2.2.6.- Refining optical flow at a single scale with disparity feedback

Section 9.4.2.2.5 described a method to refine disparity estimation on each level of resolution. However, the method described on that section was only meant as an introduction to the reader to the concept of disparity refinement. If implemented as described there, results will probably be very poor.

In order to obtain more robust results, feedback from the estimated disparity must be obtained somewhat. In that sense, we will use the warped right image into the left one as shown in (9.159) in order to create a loop so that disparity estimation by means of parabola fitting (see section 9.4.2.2.4) is applied over the left view and the right view warped into the left one, instead of the right view without warping. That way the estimated disparity by means of parabola fitting becomes instead an estimation of disparity variation that needs to be accumulated. In other words, it can be seen as if the right view coordinates are already located on the origin, and hence the estimated parabola maxima d_x and d_y seen on (9.154) already contain the disparity variation values that need to be accumulated. This is shown more detailed on algorithm 9.9.

From algorithm 9.9 it can easily be observed that initially on the first iteration, the right (test) view must be warped using the expanded estimated disparity on previous resolution level, which for the coarsest level is usually assumed to be zero. That way, as we mentioned above, matches obtained from parabola fitting as explained in section 9.4.2.2.4 will actually estimate disparity variations on each iteration of the disparity feedback loop.

Because we want robust results, it can also be observed that warping of right view onto the left one by means of expression (9.159) needs to be done with sub-pixel accuracy by means of linear interpolation.

By implementing this behavior, we create a feedback loop for disparity estimations, which along with the low-pass anisotropic filter behaves similarly to PLL's used in radio communications.

As in previous section, this loop is repeated until the error in (9.159) does not decrease any longer, or a maximum number of iterations (i.e. typically 5) is reached.

9.4.2.2.7.- Efficient computation of optical flow at a single scale

When using disparity feedback, besides a more robust disparity estimation, we can also benefit from several optimizations that can be applied to NCC matching computations when estimating parabola maxima in order to retrieve disparity variations.

In that sense, because the test (right) image is warped into the reference (left) one on a feedback loop, instead of estimating disparities to match both views, we need to estimate disparity variations. That way, by looking at picture 9.39, we see that we only need to compute NCC match scores for 5 points (center, top, bottom, left and right) in order to estimate horizontal and vertical parabolas for each point.

Because those 5 points are always referred to the origin since the right view has been warped; we can use a technique similar to the one shown in section 7.8. By using the concept of "image shifting", we will shift both the reference and the warped test image by the amounts $D_{\max}$ shown in figure 9.39 so that we can obtain the 5 NCC match scores that we need.

So for each shift of the pair of images, we can compute NCC scores of all pixels for a given disparity variation of $D_{\max}$, but instead of making a simple difference as in section 7.8, we now need to compute the product of both images in order to obtain NCC scores. Once the product is computed, we then proceed to filter the result using a separable *squared* gaussian filter with the σ_w that we saw in (9.111). Up to this point we have already computed cross-correlation scores, but we now need to normalize them.

In order to normalize cross-correlation scores, we will compute the energy of both images being matched (i.e. the reference and the warped test views). Again this can be done just once for all NCC scores in an efficient manner. In other words, in order to compute the energy of an image we just need to compute the squared values of such image and then apply a separable *squared* gaussian filter with the same σ_w that we used before for cross-correlation scores. This is repeated for the pair of images being matched and it is used for cross-correlation normalization by computing the squared root of the product of the shifted images' energies, as seen on expression (9.147)

It must be noticed that because images are assumed to have been high-pass filtered (i.e. no DC component remains), then NCC scores are equivalent to CV, and then the match scores that we have obtained by using this technique are invariant to changes in illumination.

Once the 5 NCC match scores have been computed, then the process is the same as in 9.4.2.2.6, and we just need to find the maximum of each parabola by means of expression (9.154).

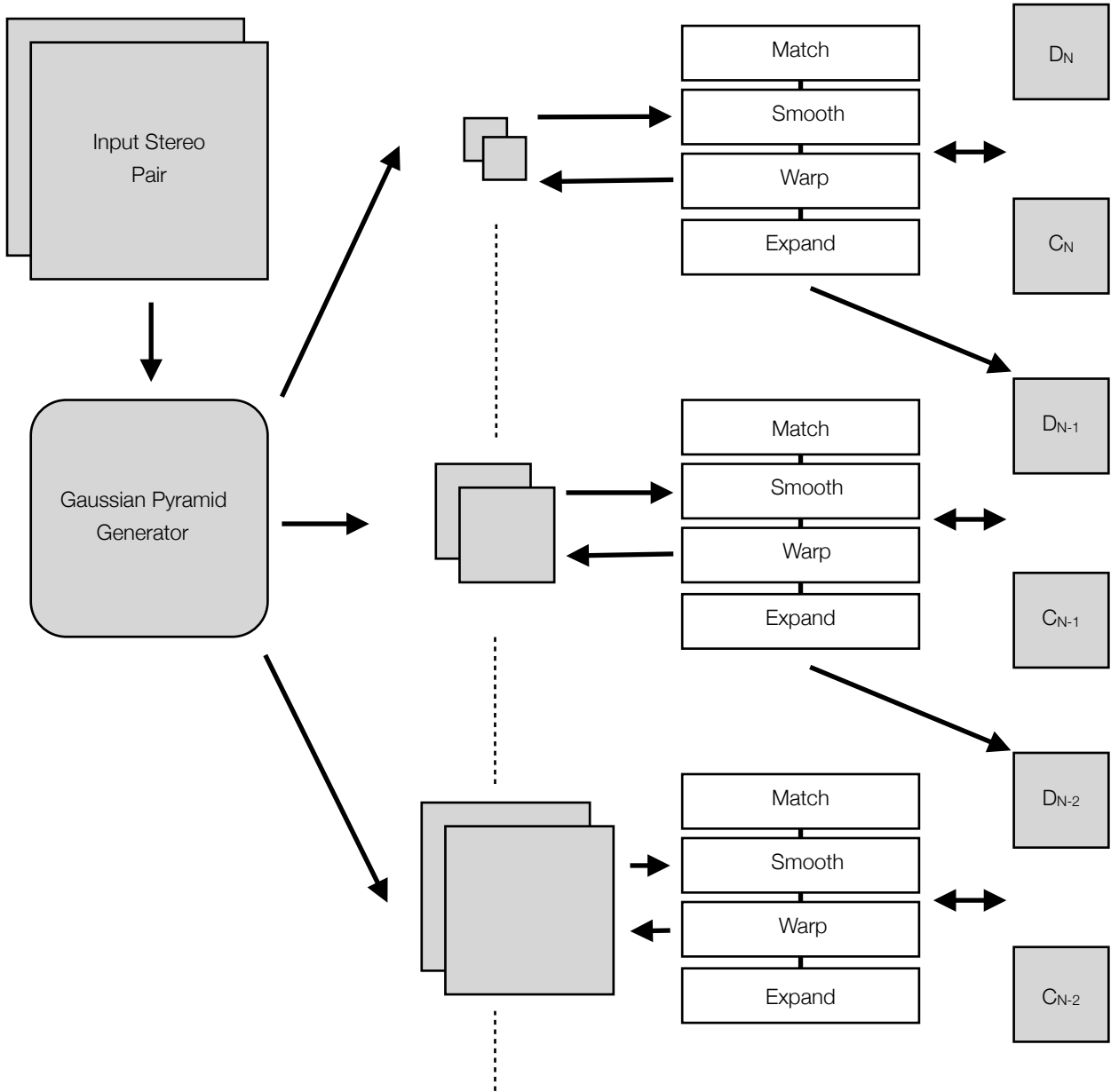
9.4.2.2.8.- Coarse-to-fine matching over scale

Once we know how to correctly build multiple resolution pyramids and how to match images on each resolution level, then it only remains to traverse the whole pyramid of resolutions starting from the coarsest one to the finest one, and use the obtained disparity on each resolution level as an initialization for the next finer level (on start, if no initialization is provided, disparity is assumed to be zero).

However, in order to use an estimated disparity for a finer resolution level, it must be properly interpolated to fit the size of the next resolution level. However, because disparity contains vectors pointing to pixel locations, a simple interpolation is not enough, and its values must also be rescaled using the scale factor as shown below:

$$\tilde{D}(x,y) = \rho \cdot \left(\left(\uparrow \rho \right) D(x,y) \right) \quad (9.160)$$

Algorithm 9.10. Coarse-to-fine matching over scale



where $(\uparrow p)$ indicates interpolation by p factor.

Notice that if next finer level is an inter-level image with the same size than current level, then interpolation and scaling can be skipped.

Finally, in order to obtain a measure of confidence throughout all scales, we need to take into account the obtained confidences on each resolution level. Again, because each resolution level might have different image sizes, the confidence map must be expanded when moving to a finer resolution level using interpolation (if next finer level has the same size, again this can be skipped). In this case, because confidence values must keep their range of values (from -1 to 1), we won't scale them. However, in order to take into account confidences in all levels, we will update the confidence map when moving to the next level using a given learning rate as shown next:

$$C_n(x, y) = \alpha \cdot C_{current}(x, y) + (1 - \alpha) \cdot C_{n-1}(x, y) \quad (9.161)$$

where a typical value for the learning rate α is 0.7;

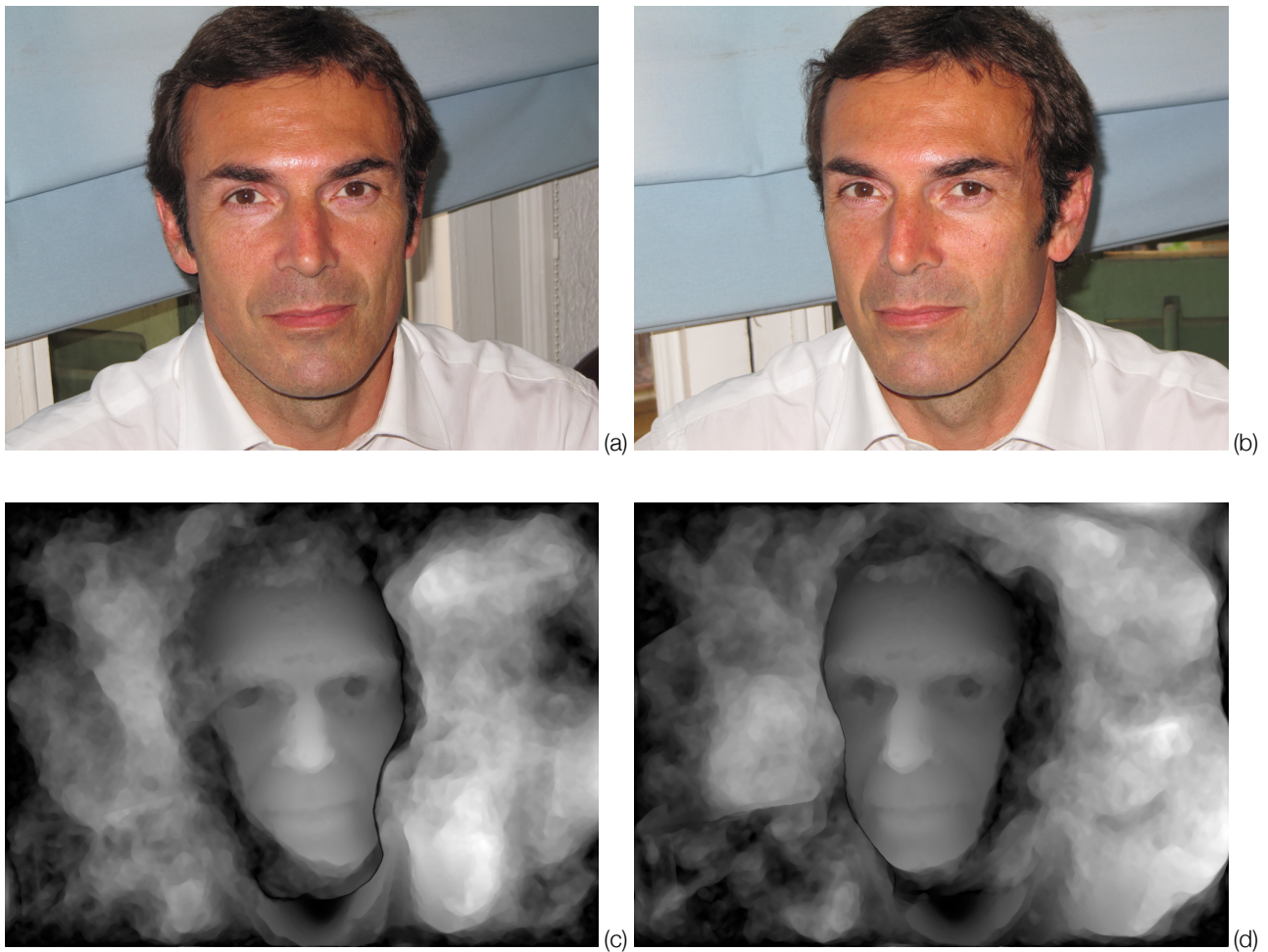


Fig. 9.40. Disparity estimation using Multiresolution algorithm. (a) Input left view. (b) Input right view. (c) Disparity modulus with reference set on left view. (d) Disparity modulus with reference set on right view

The whole multiresolution algorithm is summarized on algorithm 9.10, and some results obtained using this algorithm can be seen on fig. 9.40.

9.5.- Dense disparity map post-processing

After dense disparity has been estimated, it might still need some post-processing because, either some algorithms do not obtain sub-pixel accuracy, or because some inconsistencies exist that need to be detected and corrected. In the following sections we'll explain different post processing techniques.

9.5.1.- Obtaining sub-pixel accuracy

Usually sub-pixel accuracy is desired between matches for different reasons. However, the main reason is that when reconstructing matches into 3D points, if sub-pixel accuracy is not available, then 3D resolution decreases with depth for any fixed baseline. Also, the smaller baselines between pairs of images are, the worse 3D resolution becomes. This effect is depicted in figure 9.41, where rays of light are back-projected for each integer pixel location on the image plane. Hence, intersections of rays of light indicate possible 3D locations, which get sparser (i.e. with lower resolution) as points are located further from the image plane.

It usually happens that when we work with video sequences, baselines tend to be small. This translates into bad 3D triangularizations that need to be refined among several views to be accurately determined (i.e. more than the minimum of 2 views in order to be able to triangulate two matched image points into 3D). However, if we are able to provide sub-pixel accuracy, then triangularization improves because back-projected rays of light on figure 9.41 no longer need to pass through integer pixel locations on the image plane.

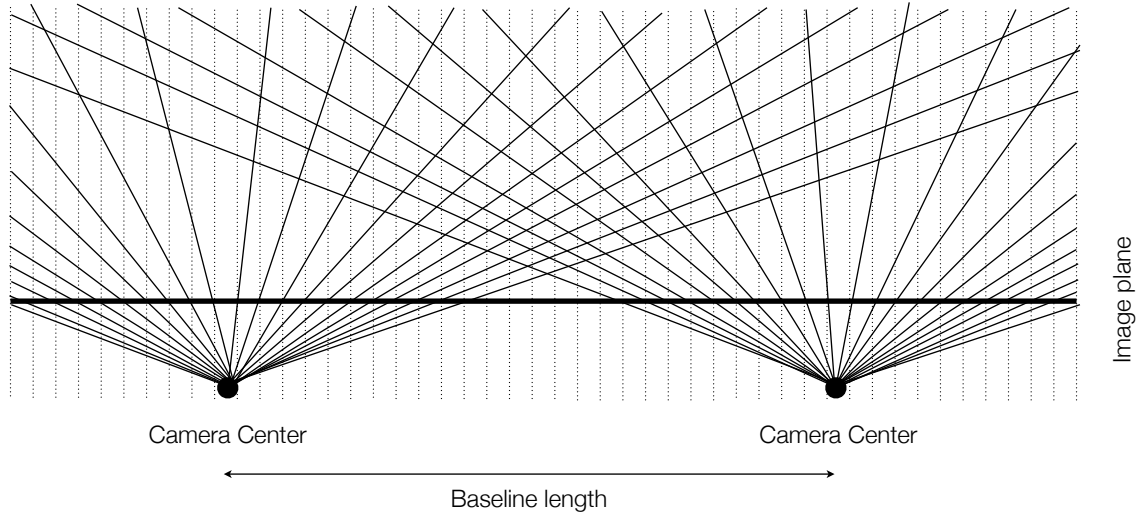


Fig. 9.41. 3D resolution vs depth and baseline length. 3D resolution gets worse as depth increases, and as the baseline decreases overall 3D resolution gets worse as well.

On the other hand, when super-resolution is needed, bad triangularizations and inaccurate matches can lead to texture that does not achieve the desired resolution. In those cases, sub-pixel accuracy also helps to increase texturing resolution.

Many algorithms can obtain sub-pixel accuracy by means of polynomial interpolation around the best matches found on the DS cost function. Similarly to figure 9.39, by computing the DS cost values for neighbor disparity values (in the horizontal and vertical directions) around the best disparity that has been found for a given pixel on the reference image, we obtain 3 points on each direction that can be used to fit a parabola (2nd degree polynomial) of costs around the best cost that we have found on both horizontal and vertical disparity directions [5]. Indeed, The process of polynomial fitting is indeed identical to the one explained on section 8.4.4 except for the fact that we usually want to find local minima on the DS, instead of local maxima on the corner response function. Hence, the only difference is that we will search for parabolas ax^2+bx+c where the first parameter a is positive. Hence by adapting expression (8.95) to (8.104) we can estimate the parabola parameter by solving the following linear system of equations,

$$\mathbf{X}'\beta = \mathbf{E} \quad (9.162)$$

where $\mathbf{X}'$ is a matrix that contains coordinates locations. In order to reduce the amount of computations and compute the inverse of $\mathbf{X}'$ only once for the whole image, we will assume that coordinates are always shifted to the origin (i.e. each x, y coordinates lies in the origin, and hence its horizontal neighbor coordinates are at $-1, y$ and $1, y$, and analogously for the vertical coordinates. Hence, $\mathbf{X}'$ can be expressed as in (8.102):

$$\mathbf{X}' = \begin{bmatrix} x_{i-1}^2 & x_{i-1}' & 1 \\ x_i^2 & x_i' & 1 \\ x_{i+1}^2 & x_{i+1}' & 1 \end{bmatrix} = \begin{bmatrix} (-1)^2 & -1 & 1 \\ 0 & 0 & 1 \\ 1^2 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & -1 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad (9.163)$$

On the other hand, beta is a vector containing the parabola parameters, and is expressed as in (8.97)

$$\beta = (a, b, c)^T \quad (9.164)$$

And finally, $\mathbf{E}$ is a vector containing the computed costs $E(u(x, y), v(x, y))$ for each neighbor points respect to (x, y) that have been used to fit the parabola. Hence:

$$\mathbf{E} = (E_{x-1}, E_x, E_{x+1})^T \quad (9.165)$$

Notice that if at least three points are provided, the linear system of equations in (9.162) can already be solved. However, if more points are provided, we can also solve it in least squares terms by means of the Moore-Penrose pseudo-inverse:

$$\beta = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{E} \quad (9.166)$$

Once the parabola for the horizontal disparities is found, by equating its derivative to zero, we can find where its minimum is located, which results in:

$$d'_x = -\frac{b}{2a}, \quad a > 0 \quad (9.167)$$

where a must be positive if we are looking for a local minimum on the DS. It must also be observed on (9.167) that the obtained horizontal disparity d'_x is referred to the origin because as we mentioned before, in order to solve the linear system of equations in (9.162) coordinates were shifted to the origin. Hence the true sub-pixel disparity will be:

$$\tilde{d}_x = d_x + d'_x \quad (9.168)$$

where d_x was the previous non sub-pixel disparity and d'_x contains the fractional sub-pixel part.

The same process can be repeated vertically in order to retrieve the vertical disparity component with sub-pixel accuracy.

Again, analogously to the process shown in section 8.4.4.2, two-dimensional polynomial curve fitting can be used as well, obtaining slightly more precise results at the expense of a higher computational cost.

As a final note, it must be taken into account that fitting parabolas around best costs will only be an acceptable approach if we assume that cost functions are well behaved and can be approximated by parabolas around correct matches, as it was mentioned in section 9.4.1.2.

9.5.2.- Uniqueness constraint

The uniqueness constraint refers to letting each pixel on an image be matched by only a single pixel onto another image. In other words, there must be a 1 to 1 correspondence. Besides, attempting to do the matching between a pair of points by reversing their order (i.e. by exchanging the reference image), should produce consistent results, or in layman terms, attempting to match on point on the reference image to another point should produce the same result that by picking the matched point on the other image and attempting to match back against the reference image, which ideally should yield the original point as the best match.

As the reader might have incorrectly thought, matches between points on pairs of images are not always reciprocal, although ideally they should be. This incoherence is usually produced by occluded areas, by objects that are visible in one view but covered in another. This fact makes that matching can be obtained in one direction, but it cannot be correctly recovered back in the opposite direction if the reference image is exchanged.

This behavior makes that occluded areas can be easily detected by making a simple left/right consistency check, which at the same time will ensure the uniqueness constraint for all consistent points. The left/right consistency check is computed as follows:

$$\begin{aligned} \varepsilon_x &= u_L(x, y) + u_R(x + u_L(x, y), y + v_L(x, y)) \\ \varepsilon_y &= v_L(x, y) + v_R(x + u_L(x, y), y + v_L(x, y)) \end{aligned} \quad (9.169)$$

where ϵ_x and ϵ_y indicate the horizontal and vertical inconsistencies respectively. u_L indicates the horizontal component of disparity taking the left image as reference, u_R is the horizontal component of disparity taking the right image as reference, v_L is the vertical component of disparity taking the left image as reference, and finally v_R is the vertical component of disparity taking the right image as reference at point (x,y) . Proof of expression (9.169) is shown below.

Proof 9.2

We know that a point in the right image (x_R, y_R) respect to a point in the left image (x_L, y_L) can be expressed as:

$$(x_R, y_R)^T = (x_L, y_L)^T + (u_L(x_L, y_L), v_L(x_L, y_L))^T \quad (9.170)$$

where u_L and v_L stand for the disparity components taking the left image as the reference, (x_L, y_L) are the coordinates of a point on the left image, and (x_R, y_R) are the coordinates of a point on the right image.

Conversely, a point in the left image respect to another in the right image can be expressed as:

$$(x_L, y_L)^T = (x_R, y_R)^T + (u_R(x_R, y_R), v_R(x_R, y_R))^T \quad (9.171)$$

Hence, by joining expressions (9.170) and (9.171) we obtain:

$$\begin{aligned} (x_L, y_L)^T &= (x_L, y_L)^T + (u_L(x_L, y_L), v_L(x_L, y_L))^T + (u_R(x_R, y_R), v_R(x_R, y_R))^T \Rightarrow \\ (u_L(x_L, y_L), v_L(x_L, y_L))^T &+ (u_R(x_R, y_R), v_R(x_R, y_R))^T = 0 \Rightarrow \\ (u_L(x_L, y_L), v_L(x_L, y_L))^T &+ (u_R(x_L + u_L(x_L, y_L), y_L + v_L(x_L, y_L)), v_R(x_L + u_L(x_L, y_L), y_L + v_L(x_L, y_L)))^T = 0 \end{aligned} \quad (9.172)$$

Because some inconsistencies might be present between disparities (u_L, v_L) and (u_R, v_R) , then the expression on (9.172) won't be exactly zero, so we account for a certain margin of error (ϵ_x, ϵ_y) . On the other hand we can rename points (x_L, y_L) simply as (x,y) in order to simplify expression (9.172) and we obtain:

$$\begin{aligned} \epsilon_x &= u_L(x, y) + u_R(x + u_L(x, y), y + v_L(x, y)) \\ \epsilon_y &= v_L(x, y) + v_R(x + u_L(x, y), y + v_L(x, y)) \end{aligned} \quad (9.169)$$

which completes the proof.

It must be observed that by setting the appropriate thresholds (around a couple of pixels), we can detect all matches that do not pass this test because they have inconsistencies larger than the thresholds. Hence, those pixels will also be discarded for any future computations during 3D triangulation or texturing.

9.5.3.- Dense disparity correction

We have already seen how we can obtain sub-pixel accuracy and how inconsistent matches are detected, we now propose a solution that attains both goals, and at the same time corrects many inconsistencies while preserving piecewise smoothness on estimated disparity.

The process that we propose is mainly based on [22], In order to obtain piecewise smooth disparity we initially over-segment the reference image using the assumption that disparity will change smoothly inside each segment as gradients tend to be small inside those segments, while on the other hand, abrupt changes on estimated disparity might appear at segment borders, as they usually are coincident with real object boundaries. This way we can obtain piecewise smooth disparity that

at the same time preserves objects boundaries. The whole process is summarized in algorithm 9.11 and explained in more detail below.

The process starts first of all by detecting all the inconsistencies with a left/right consistency test. In order to obtain better accuracy in matches, all left and right matches marked as consistent are averaged, and the inconsistent matches are corrected initially by setting their left and right disparity values to the disparity value having a smallest norm between them, as many occlusions or wrong matches tend to appear as spots having much larger disparity values than the real disparity value.

Once this initial correction is done, all inconsistencies are recomputed again with a left/right consistency test. Now, all inconsistent points are detected and are segmented away from all consistent points. The small inconsistent segments, as we mentioned before usually correspond to spots having much different disparity values than the pixels surrounding them, which have correct consistent disparity values. In order to correct the points inside each inconsistent region, we use all disparity values in the points surrounding the inconsistent region in order to fit a curve of second degree or a plane inside the inconsistent region. This should correct many inconsistencies, however, since this does not take into account the actual segmentation on the image, object shapes might not be preserved.

Hence, to finally preserve object shape, and at the same time obtain sub-pixel accuracy, we take all consistent matches inside each over-segmented region on the reference image, and fit a second degree curve or a plane for all those matches.

This will allow us to obtain a dense disparity map with sub-pixel accuracy that preserves objects boundaries. However, there is still one problem to be solved. It might happen that due to wrong but consistent matches, curves are not correctly estimated. In those cases it might be interesting to repeat the whole process by joining all over-segmented regions with similar disparities by assuming some smoothness criterion.

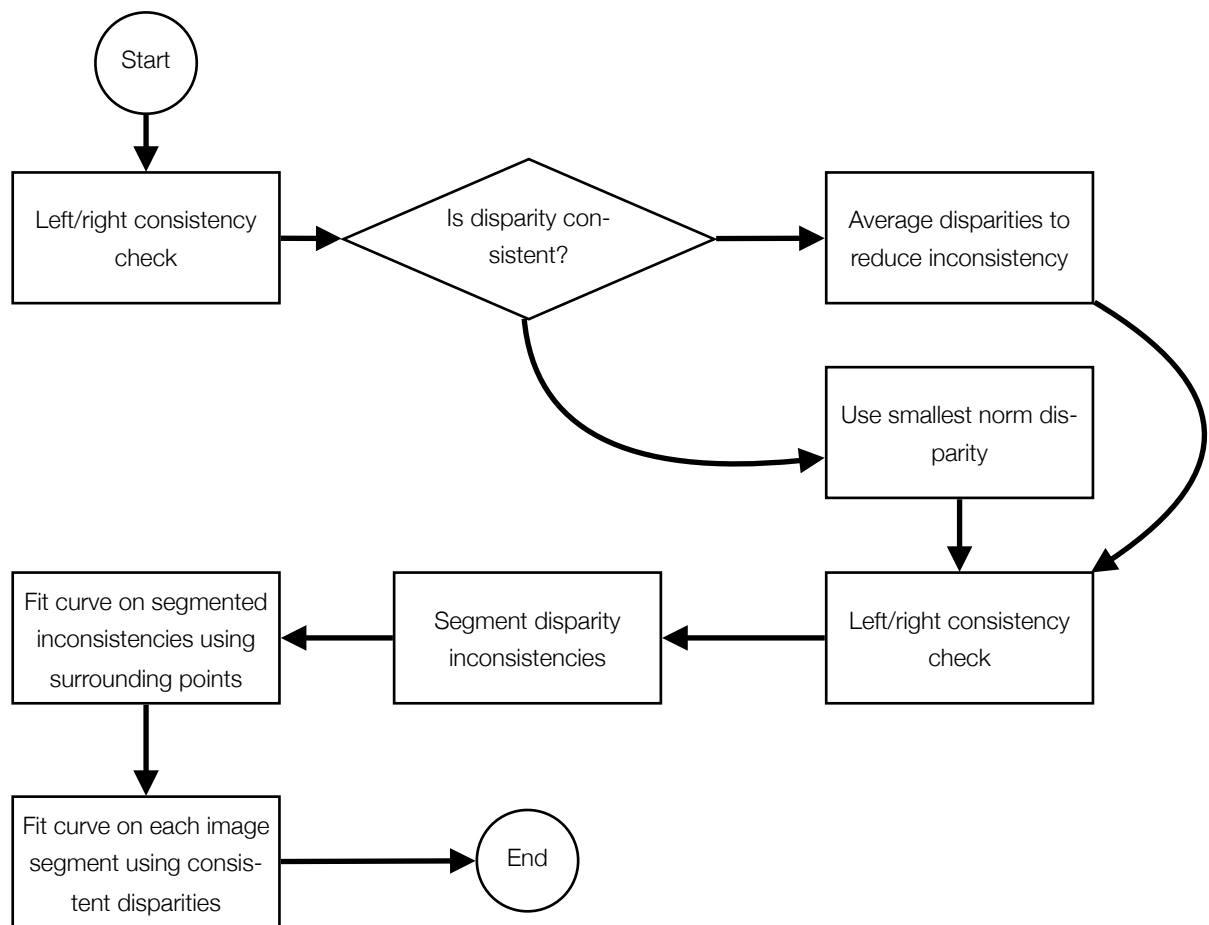
In figure 9.42 we show the initial disparity estimations (9.42a and 9.42b) and inconsistencies (9.42e and 9.42f), and the final corrected disparities (9.42c and 9.42d) and inconsistencies (9.42g and 9.42h), when using left image as reference (left column) or right image as reference (right column). As it can be observed, inconsistencies get substantially reduced, which will help to improve the 3D reconstruction, which is shown in figure 9.43 along with left and right images used as input for the reconstruction.

9.6.- Closure

In this chapter we have provided an extensive introduction to image rectification assuming that epipolar geometry (i.e. fundamental matrix) between a pair of images is known. We indicated what rectification is used for (i.e. mainly to accelerate dense disparity estimation and provide epipolar constraints on dense disparity estimation) and how to compute it by indicating two different algorithms.

In that sense, we indicated how the Hartley's rectification works, and its main limitation when epipoles lie within image boundaries. For that reason, we provided a second method based on polar transformations around the epipoles which seems to produce better results (i.e. less distortion) in most situations, and does not fail if epipoles lie within image boundaries. Indeed, our experimental results, as we explained, corroborated that Pollefe's polar rectification method produces less distortion than Hartley's one, except around epipoles (if those lie within image boundaries). However, in this latter case we also indicated that points laying close to the epipoles can't be triangulated either, due to machine precision as they are too close to the baseline. This makes back-projected rays of light coincident throughout the whole ray instead of a single point and for that reason triangulation fails in that area.

Once the rectification process was understood, we proceeded to explain how dense disparity can be estimated by taking into account the grey value and gradient constancy assumptions. We saw that these assumptions can help us to linearly compute disparity values by solving linear systems of equations taking into account only image grey levels and gradient data. However, we proved that these assumptions are only valid for small pixel displacements and completely fail when projective distortion is present (which is the most common situation). For that reason, instead, we defined a cost functional to minimize the DS (Disparity Space) in order to densely estimate disparity, and we indicated that the linearized grey value and gradient



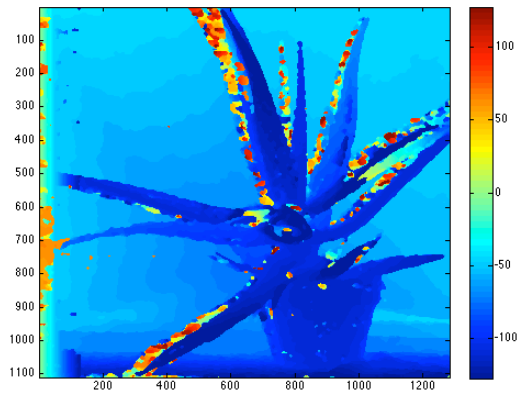
constancy assumptions could be used as a coarse disparity estimation to initialize other minimization algorithms over the DS cost functional.

Once we knew what the cost functional was, we indicated several improvements that could be done, by indicating which distance measures were the most robust (i.e. CV or NCC as they are invariant to changes in illumination), and other constraints that could be applied to ensure that the estimated disparity was piece-wise smooth and followed the estimated epipolar geometry.

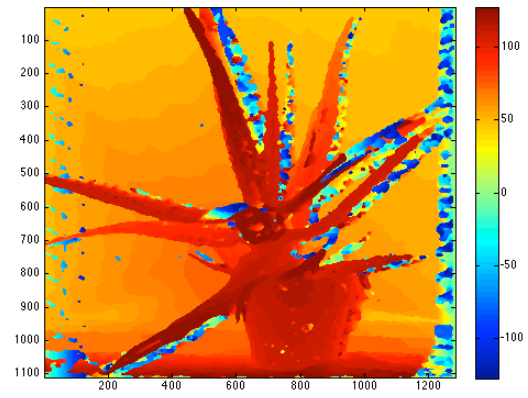
Taking into account all those constraints on the cost functional, we proceeded to provide different minimization algorithms in order to densely estimate disparity between pairs of images. In that sense, we indicated that there are two main approaches for dense disparity estimation: local and global ones.

Among the local ones, which attempt to estimate disparity on each pixel on the reference image independently, we explained WTA and Evolutionary algorithms (i.e. Gradient Descent, Levenberg-Marquardt and Simulated Annealing), and we proposed a practical implementation based on HRM, which uses a coarse block structure to initialize disparity which is later refined at pixel level.

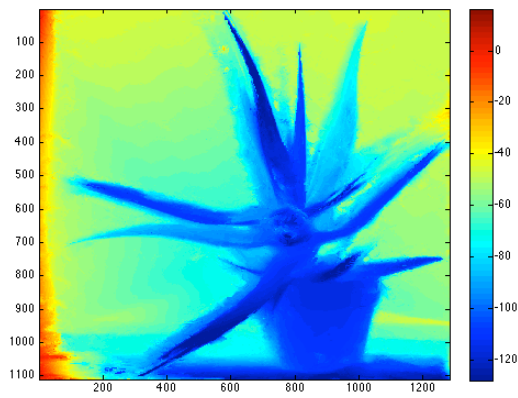
Among the global ones, which tend to provide better results than the local ones, we indicated how to implement the least computational intensive methods. In that sense, we indicated two Dynamic Programming search methods based on a greedy approach and on a modified Dijkstra's algorithm. And finally, we provided a **multiresolution** approach which from among all the local and global methods **is the one that provides the most robust results**.



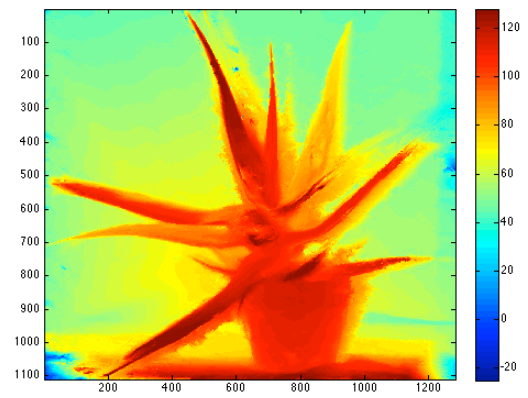
(a)



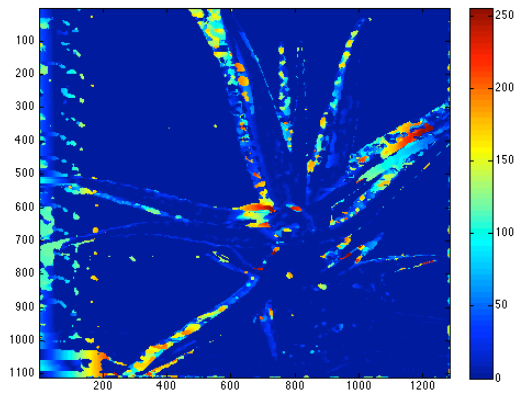
(b)



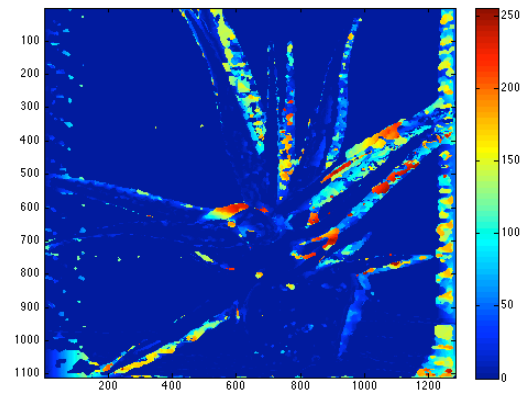
(c)



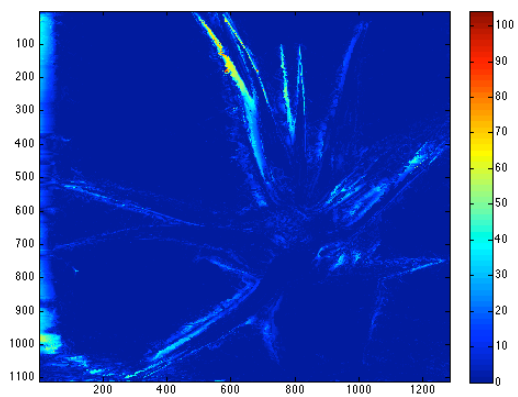
(d)



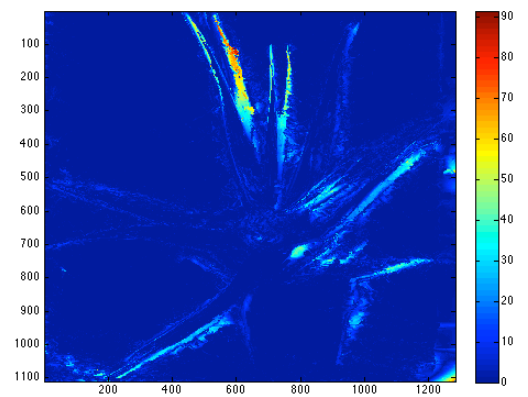
(e)



(f)



(g)



(h)

Fig. 9.42. Corrected dense disparity using segmentation and left/right consistency.
3D Face and Object Reconstruction for Mobile Applications

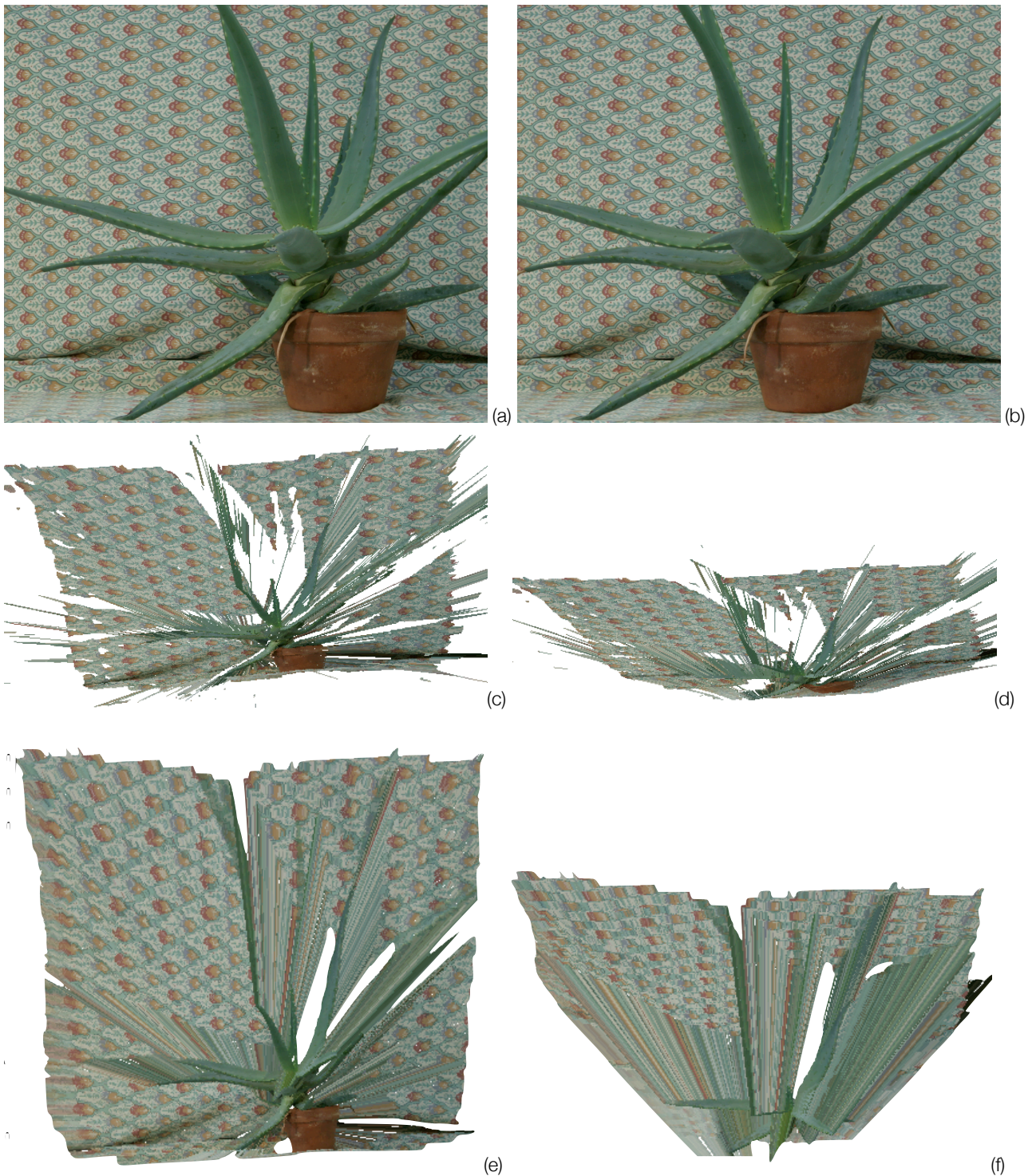


Fig. 9.43. (a) Left view. (b) Right view. (c) and (d) Different points of view of the 3D reconstruction using estimated disparity. (e) and (f) Different points of view of the 3D reconstruction using ground-truth disparity. Notice that holes in the 3D reconstruction are due to occlusions or disparity inconsistencies.

Indeed, we proved that the multiresolution approach that we implemented is so robust that no image pre-processing or rectification is needed. Also, our implementation can take advantage of separable filters and efficient dense disparity estimation (because of right image warping into the left view) to obtain results as fast as possible.

Finally, in order to increase the quality of the results obtained on any of the dense disparity estimation methods that we presented, we indicated a series of post-processing techniques that can be applied, which are based on polynomial fitting to obtain sub-pixel accuracy on disparity estimation, left/right consistency tests to discard disparities that produced bad

matches, and a combined technique that takes into account left/right consistency and segmentation of reference image in order to produce piece-wise disparities that preserve the shape of objects by taking into account their segmentation on the reference view.

We noticed however, that because the multiresolution approach already estimated disparity with sub-pixel accuracy, some of the post-processing, such as the polynomial fitting techniques weren't needed at all. Also, the anisotropic filtering incorporated on each resolution level already ensured that object boundaries got preserved, making the segmentation based technique less relevant, although experimentally we saw that it could improve object border preservation in some situations. Hence, the only post-processing technique that we saw of interest for the multiresolution approach was a simple left/right consistency test to detect occlusions and avoid their triangulation and texturing.

References

- [1] Nyquist, Harry. "Certain factors affecting telegraph speed". Bell System Technical Journal, 324–346, 1924.
- [2] D. Scharstein and R. Szeliski. "A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms". IJCV 2002.
- [3] S. Birchfield and C. Tomasi. "A Pixel Dissimilarity Measure That Is Insensitive to Image Sampling". IEEE Transactions on Pattern Analysis and Machine Intelligence. Vol. 20. 1998.
- [4] D. Scharstein and R. Szeliski. "Stereo Matching with Nonlinear Diffusion". IJCV 1998.
- [5] B. Cyganek and J. P. Siebert. "An Introduction to 3D Computer Vision Techniques and Algorithms". Wiley & Sons. 2009.
- [6] J. Banks and P. Corke. "Quantitative Evaluation of Matching Methods and Validity Measures for Stereo Vision". The International Journal of Robotics Research. 2001.
- [7] M. Pollefeys. "Tutorial on 3D Modeling from Images". Lecture Notes of ECCV2000, Dublin, 2000.
- [8] R. Hartley and A. Zisserman. "Multiple View Geometry in Computer Vision". Cambridge University Press. 2003.
- [9] O. Faugeras. "Three-Dimensional Computer Vision. A Geometric Viewpoint". The MIT Press. 1993.
- [10] O. Faugeras and Q. Luong. "The Geometry of Multiple Images: The Laws That Govern the Formation of Multiple Images of a Scene and Some of Their Applications". The MIT Press. 2004.
- [11] P. Mordohai and G. Medioni. "Stereo Using Monocular Cues within the Tensor Voting Framework". Transactions on Pattern Analysis and Machine Intelligence, Vol. 28. 2006.
- [12] K. Yoon and I. Kweon. "Adaptive Support-Weight Approach for Correspondence Search". Transactions on Pattern Analysis and Machine Intelligence, Vol. 28. 2006.
- [13] Y. Taguchi et al. "Stereo Reconstruction with Mixed Pixels Using Adaptive Over-Segmentation". CVPR. 2008.
- [14] T. Kanade and M. Okutomi. "A Stereo Matching Algorithm with an Adaptive Window: Theory and Experiment". IEEE Transactions on Pattern Analysis and Machine Intelligence. Vol. 16. 1994.
- [15] L. Wang et al. "High-quality Real-time Stereo using Adaptive Cost Aggregation and Dynamic Programming". Third International Symposium on 3D Data Processing, Visualization, and Transmission. 2006.
- [16] F. Tombari et al. "Segmentation-Based Adaptive Support for Accurate Stereo Correspondence". IEEE Pacific-Rim Symposium on Image and Video Technology. 2007.
- [17] J. Yin and J. R. Cooperstock. "Improving Depth Maps by Nonlinear Diffusion". WSCG. 2004.
- [18] H. Hirschmüller. "Stereo Vision in Structured Environments by Consistent Semi-Global Matching". CVPR 2006.

- [19] C. Harris and M. Stephens. "A Combined Corner and Edge Detector". Proceedings of The Fourth Alvey Vision Conference. 1988.
- [20] D. G. Lowe. "Object Recognition from Local Scale-Invariant Features". Proceeding of the International Conference on Computer Vision. 1999.
- [21] E. Mémin and P. Pérez. "Hierarchical Estimation and Segmentation of Dense Motion Fields". International Journal of Computer Vision. 2002.
- [22] Z. Wang and Z. Zheng. "A Region Based Stereo Matching Algorithm Using Cooperative Optimization". CVPR 2008.
- [23] T. Brox et al. "High Accuracy Optical Flow Estimation Based on a Theory for Warping". Proc. 8th European Conference on Computer Vision. 2004.
- [24] M. Mainberger et al. "Is Dense Optic Flow Useful to Compute the Fundamental Matrix?". Lecture Notes in Computer Science. 2008.
- [25] S. Kang et al. "Handling Occlusions in Dense Multi-view Stereo". CVPR, vol. 1, pp. 103. CVPR 2001.
- [26] M. J. Blakc. "Robust Dynamic Motion Estimation Over Time". Computer Vision and Pattern Recognition. 1991
- [27] A. Bhusnurmath and C. J. Taylor. "Solving Stereo Matching Problems using Interior Point Methods". European Conference on Computer Vision. 2008.
- [28] N. Slesareva et al. "Optic Flow Goes Stereo: A Variational Method for Estimating Discontinuity-Preserving Dense Disparity Maps". LNCS 3663, pp. 33-40. 2005.
- [29] Y. Ohta and T. Kanade. "Stereo by Intra- and Inter-Scanline Search Using Dynamic Programming". IEEE Trans. Pattern Analysis and Machine Intelligence, Vol. PAMI-7, No. 2. 1985.
- [30] G. Csurka, C. Zeller, Z. Zhang and O. Faugeras. "Characterizing the Uncertainty of the Fundamental Matrix". Computer Vision and Image Understanding. 1996.
- [31] Q. T. Luong and O. D. Faugeras. "The Fundamental Matrix: theory, algorithms, and stability analysis". IJCV. 1995.
- [32] Z. Zhang. "Determining the Epipolar Geometry and its Uncertainty: A Review". International Journal of Computer Vision, 27 (3), 161-198. 1998.
- [33] N. Atzpadin, P. Kauff, O. Schreer. "3D Analysis by Hybrid Recursive Matching for Real-Time Immersive Video Conferencing". IEEE Trans. on Circuits and Systems for Video Technology. Special Ussue on Immersive Telecommunications, pp. 321-334. Vol. 14, 2004.
- [34] W. H. Press, S. A. Teukolsky, W. T. Vetterling and B. P. Flannery. "Numerical Recipes: The Art of Scientific Computing". Cambridge University Press. 3rd Edition. 2007.
- [35] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein. "Introduction to Algorithms". The MIT Press, 2nd Edition. 2001.
- [36] M. George and T. Tjahjadi. "Multi-Resolution Optical Flow Estimation Using Adaptive Shifting". International Conference on Image Processing. 1999.
- [37] L. Alvarez, J. Wickert and J. Sánchez. "Reliable Estimation of Dense Optical Flow Fields with Large Displacements". International Journal of Computer Vision. Vol 39. 2000.
- [38] L. Alvarez, J. Esclarin, M. Lefebure and J. Sánchez. "A PDE model for computing the Optical Flow". Proceedings of CEDYA XVI. 1999.

- [39] J. R. Bergen, P. Ananda, K. J. Hanna and R. Hingorani. "Hierarchical Model-Based Motion Estimation". European Conference on Computer Vision. 1992
- [40] Datasets from Middlebury Stereo Vision Page. www.middlebury.edu/stereo.
- [41] C.L. Zitnick, S.B. Kang, M. Uyttendaele, S. Winder, and R. Szeliski. "High-quality video view interpolation using a layered representation". ACM SIGGRAPH and ACM Trans. on Graphics. 2004.
- [42] M. Pollefeys, R. Koch and L.V. Gool. "A simple and efficient rectification method for general motion". ICCV. 1999.
- [43] R. Hartley. "Cheirality Invariants". Proc. D.A.R.P.A. Image Understanding Workshop, pp. 743-753, 1993.
- [44] S. Laveau and O. Faugeras, "Oriented Projective Geometry for Computer Vision". Computer Vision - ECCV'96, Lecture Notes in Computer Science, Vol. 1064, Springer-Verlag, pp. 147-156, 1996.
- [45] B. Horn and B. Schunck. "Determining Optical Flow". Artificial Intelligence, 17:185-203, 1981.
- [46] P. Perona and J. Malik. "Scale-Space and Edge Detection Using Anisotropic Diffusion". IEEE Transactions on Pattern Analysis and Machine Intelligence. Vol, 12. No. 7. July 1990.
- [47] A. Hornung and L. Kobbelt. "Robust Reconstruction of Watertight 3D Models from Non-uniformly Samples Point Clouds Without Normal Information". Eurographics Symposium on Geometry Processing. 2006.
- [48] M. Kazhdan, M. Bolitho and H. Hoppe. "Poisson Surface Reconstruction". Eurographics Symposium on Geometry Processing. 2006.
- [49] P. Merrell, A. Akbarzadeh, L. Wang, P. Mordohai, J. Frahm, R. Yang, D. Nistér, and M. Pollefeys. "Real-Time Visibility-Based Fusion of Depth Maps". ICCV. 2007