3D Acquisition

Maarten Vergauwen & Luc Van Gool

November 18, 2005

Contents

1 Passive 3D Reconstruction: Basics **7**
- 1.1 Introduction . . . 7
- 1.2 Homogeneous Coordinates . . . 7
  - 1.2.1 Points . . . 7
  - 1.2.2 Infinity . . . 8
  - 1.2.3 Lines . . . 8
- 1.3 Camera Models . . . 9
  - 1.3.1 Image Coordinates . . . 9
  - 1.3.2 The Linear Pinhole Model . . . 10
  - 1.3.3 Non-Linear Distortions . . . 14
- 1.4 Deformation of planar objects under projection . . . 15
  - 1.4.1 Case 1 . . . 16
  - 1.4.2 Case 2 . . . 16
  - 1.4.3 Case 3 . . . 17
- 1.5 Deformation of 3D objects under projection . . . 18
  - 1.5.1 Stratification of transformations of 3D objects . . . 18
  - 1.5.2 Transformations and the projection equation . . . 18
- 1.6 Calibration . . . 21
  - 1.6.1 Internal Calibration . . . 21
  - 1.6.2 External Calibration . . . 24
- 1.7 A Simple Stereo Setup . . . 27

2 Relating Image Pairs **33**
- 2.1 Features . . . 33
  - 2.1.1 Invariance . . . 33
  - 2.1.2 Simple Image Features . . . 36
  - 2.1.3 Small- vs. Wide-Baseline Matching . . . 38
  - 2.1.4 Matching Lines . . . 42
- 2.2 The Geometry of Two Images . . . 43
  - 2.2.1 Epipolar Geometry . . . 43
  - 2.2.2 The Fundamental Matrix . . . 44
  - 2.2.3 Computation of the Fundamental Matrix . . . 46
  - 2.2.4 Dealing with Outliers: the RANSAC Algorithm . . . 48
  - 2.2.5 F-Guided Matching . . . 50
  - 2.2.6 Computing Homographies with RANSAC . . . 51

3 Relating Multiple Views **53**
- 3.1 Multiple View Geometry . . . 53
  - 3.1.1 Trifocal Tensor . . . 53
- 3.2 Structure and Motion . . . 55
  - 3.2.1 Initialization Step . . . 56
  - 3.2.2 Projective Pose Estimation . . . 58

3.2.3 Updating Structure . . . 60
3.2.4 Global Minimization . . . 61

4 Self-Calibration **63**
4.1 The Projective Ambiguity . . . 63
4.2 Scene Constraints . . . 65
4.3 Self-Calibration . . . 68
4.3.1 Conics and Quadrics . . . 68
4.3.2 The Absolute Conic . . . 70
4.3.3 Practical Constraints . . . 71
4.3.4 Coupled Self-Calibration . . . 72

5 Model Selection **75**
5.1 Introduction . . . 75
5.2 Problems with Planar Scenes . . . 75
5.3 Detecting Planar Scenes . . . 76
5.3.1 Occam's Razor . . . 76
5.3.2 GRIC . . . 76
5.4 More GRICs . . . 78
5.5 Long Video Sequences . . . 80
5.5.1 Video Frames . . . 82
5.5.2 Long Sequences and Subsequences . . . 84
5.5.3 Intermediate Frames . . . 87
5.5.4 Blurry Frames . . . 88

6 Essential Matrices **93**
6.1 Essential Matrices . . . 93
6.1.1 Normalized Coordinates . . . 93
6.1.2 The Essential Matrix . . . 93
6.1.3 Properties of the Essential Matrix . . . 94
6.1.4 Cameras from the Essential Matrix . . . 95
6.2 Computation of the Essential Matrix . . . 96
6.2.1 8 and 7 Point Algorithm . . . 96
6.2.2 6 Point Algorithm . . . 97
6.2.3 5 Point Algorithm . . . 98
6.2.4 Planar Degeneracy . . . 99

7 Bundle Adjustment **101**

8 Dense Matching **103**
8.1 Introduction . . . 103
8.2 Rectification . . . 103
8.2.1 Planar Rectification . . . 103
8.2.2 Polar Rectification . . . 106
8.3 Stereo Matching . . . 106
8.3.1 Constraints . . . 110
8.3.2 Matching Matrix . . . 110
8.3.3 Cost Function . . . 110
8.3.4 Hierarchical Stereo . . . 112
8.3.5 Post Processing . . . 113
8.4 Linking Stereo Pairs . . . 114
8.5 Fast Matching on the GPU . . . 116
8.5.1 Why GPUs . . . 116
8.5.2 Basic Setup and Conventions . . . 117

8.5.3 Hierarchical Plane-Sweep Algorithm . . . 117
8.5.4 The Connectivity Constraint . . . 118
8.5.5 Retrieval Of Fine Structures . . . 119
8.5.6 Smoothness Penalty . . . 120
8.6 Bayesian Multi-View Matching . . . 121

9 3D Webservice 125
9.1 Introduction . . . 125
9.1.1 3D Technologies in the Cultural Heritage Field . . . 125
9.1.2 Low-cost Pipeline for 3D Reconstruction . . . 125
9.1.3 Paper Overview . . . 125
9.2 System Overview . . . 126
9.2.1 Upload Tool . . . 128
9.2.2 Modelviewer Tool . . . 128
9.3 Automatic Reconstruction Pipeline . . . 129
9.3.1 Pipeline Overview . . . 129
9.3.2 Opportunistic Pipeline . . . 129
9.3.3 Hierarchical Pipeline . . . 132
9.3.4 Parallel Pipeline . . . 132
9.4 Global Image Comparison . . . 132
9.5 Self-calibration . . . 134
9.5.1 Classical SaM techniques . . . 134
9.5.2 Triplet Matching . . . 134
9.5.3 Coupled Self-calibration . . . 135
9.5.4 Statistical Coupled Self-calibration . . . 136
9.6 Reconstruction . . . 136
9.6.1 Upscaling the Result . . . 138
9.6.2 Robust Euclidean Bundle Adjustment . . . 139
9.7 Dense Matching . . . 141
9.7.1 Linked Pairwise Stereo . . . 141
9.7.2 Multi-View Stereo . . . 142
9.8 Results . . . 143
9.9 Conclusion . . . 146

Chapter 1

Passive 3D Reconstruction: Basics

1.1 Introduction

In this chapter we will discuss some important basic principles of computer vision that need to be mastered to understand the techniques of 3D reconstruction. More specifically we will start by explaining the image-formation process by means of camera models. Both linear and non-linear effects are discussed. A stratification of 2D and 3D geometry is given. Then we will explain how cameras can be calibrated with standard calibration procedures using known calibration objects. At the end of this chapter we will discuss the simplest passive 3D reconstruction setup: a stereo-pair.

1.2 Homogeneous Coordinates

Computer scientists often use homogeneous coordinates in their work. While not strictly necessary in the Euclidean world, they make the notation of the equations simpler. In the projective world, homogeneous coordinates are inevitable.

1.2.1 Points

In order to express the coordinates of a point in the plane or in the world, one needs 2 or 3 coordinates respectively. In $\Re^2$ and $\Re^3$ we can write

$$\mathtt{m_e} = \begin{pmatrix} x \\ y \end{pmatrix}, \mathtt{M_e} = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}.$$

Expressing the same points in homogeneous coordinates means that we add a third or fourth coordinate to the points respectively, so in $\Re^2$ and $\Re^3$ the points become

$$\mathtt{m_p} = \begin{pmatrix} x_h \\ y_h \\ w_h \end{pmatrix}, \mathtt{M_p} = \begin{pmatrix} X_h \\ Y_h \\ Z_h \\ W_h \end{pmatrix}.$$

We now of course have one degree of freedom too much. This is solved by enforcing that a point in homogeneous coordinates, multiplied with any non-zero scale-factor remains the same point.

$$\mathtt{m_p} \quad = \quad \begin{pmatrix} x \\ y \\ w \end{pmatrix} = \begin{pmatrix} ax \\ ay \\ aw \end{pmatrix}$$

$$\mathtt{M_p} = \begin{pmatrix} X \\ Y \\ Z \\ W \end{pmatrix} = \begin{pmatrix} aX \\ aY \\ aZ \\ aW \end{pmatrix} (a \neq 0)$$

To write down the homogeneous coordinates of a Euclidean point, we simply add a 1 as the last coordinate. To retrieve the Euclidean coordinates of a point in homogeneous coordinates, we divide by the last homogeneous coordinate to make it 1. This of course only yields meaningful results if this coordinate is not zero.

$$\begin{aligned} \mathtt{m_p} &= \begin{pmatrix} x_h \\ y_h \\ w_h \end{pmatrix} = \begin{pmatrix} x_h/w_h \\ y_h/w_h \\ w_h/w_h \end{pmatrix} = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} (w_h \neq 0) \\ \mathtt{M_p} &= \begin{pmatrix} X_h \\ Y_h \\ Z_h \\ W_h \end{pmatrix} = \begin{pmatrix} X_h/W_h \\ Y_h/W_h \\ Z_h/W_h \\ W_h/W_h \end{pmatrix} = \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} (W_h \neq 0) \end{aligned}$$

1.2.2 Infinity

What happens if the last coordinate of a homogeneous point is 0? It's clear that this point can not be represented in Euclidean space. In projective space, these points are said to be located *at infinity*. In $\Re^2$ all points with a zero third homogeneous coordinate form the line at infinity. In $\Re^3$, the points with zero fourth coordinate form the plane at infinity. The use of points at infinity will become clear in the next paragraph which deals with lines.

1.2.3 Lines

Lines can of course also be expressed in homogeneous coordinates. A line can be described by two points $\mathtt{m}_1$ and $\mathtt{m}_2$ on it. Any point $\mathtt{m}$ on the line can be expressed by starting at point $\mathtt{m}_1$ and traveling a certain distance λ in the direction $\mathtt{m}_2 - \mathtt{m}_1$. In 2D we write this direction as:

$$\begin{aligned} \mathtt{m}_2 - \mathtt{m}_1 &= \begin{pmatrix} w_1x_2 - w_2x_1 \\ w_1y_2 - w_2y_1 \\ w_1w_2 - w_2w_1 \end{pmatrix} \\ &= \begin{pmatrix} w_1x_2 - w_2x_1 \\ w_1y_2 - w_2y_1 \\ 0 \end{pmatrix} \end{aligned}$$

(note that before you can add or subtract two homogeneous points, you have to makes sure their last coordinate is the same)

In 3D, the equivalent equation can of course be written. It is now clear that the direction of the line is a point at infinity. In other words, all points at infinity describe directions of lines. Also, parallel lines have the same direction, so all parallel lines coincide in the same point at infinity.

For 2D lines, an elegant representation of a line can be used which expresses the line as a 3×1 vector. We start from the Euclidean expression of a line

$$ax + by + c = 0$$

This homogeneous equation is unaffected by scaling with any scale-factor w

$$\begin{aligned} awx + bwy + cw &= 0 \\ ax_h + by_h + cw_h &= 0 \end{aligned}$$

The line l can thus be expressed by the vector $l = (a\ b\ c)^T$. The fact that a point $m = (x\ y\ 1)^T = (x_h\ y_h\ w_h)^T$ lies on the line l is easily expressed by enforcing the scalar product between the point and the line vectors to be zero:

$$\begin{aligned}(a\ b\ c)\begin{pmatrix} x_h \\ y_h \\ w_h \end{pmatrix} &= 0 \\ \mathtt{l}^\mathtt{T}\mathtt{m} &= 0\end{aligned}$$

Thus we see that points and lines have the same representation in the projective plane.

We know that a line can be defined as going through 2 points. If the coordinates of these two point $\mathtt{m}_1$ and $\mathtt{m}_2$ are given, what is then the vector $\mathtt{l}$ describing the line? We know that the line must go through both points, hence

$$\begin{aligned}\mathtt{l}^\mathtt{T}\mathtt{m}_1 &= 0 \\ \mathtt{l}^\mathtt{T}\mathtt{m}_2 &= 0\end{aligned}$$

In other words we search for the vector $\mathtt{l}$ which is orthogonal to both vectors $\mathtt{m}_1$ and $\mathtt{m}_2$. We know that the cross product of these two vectors is indeed a vector orthogonal to the vectors, hence

$$\mathtt{l} = \mathtt{m}_1 \times \mathtt{m}_2$$

or

$$\mathtt{l} = [\mathtt{m}_1]_\times \mathtt{m}_2 \tag{1.1}$$

with $[a]_\times$ the skew-symmetric matrix defined as

$$[a]_\times = \begin{bmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{bmatrix} \tag{1.2}$$

where the multiplication of the skew-symmetric matrix created from vector $\mathtt{m}_1$ with vector $\mathtt{m}_2$ indeed delivers the cross-product vector $\mathtt{m}_1 \times \mathtt{m}_2$.

1.3 Camera Models

If we want to understand how to reconstruct 3D objects from images, we first need to know how the reverse process works: how is the world transformed by a camera into images. It is clear that this transformation is one from the 3D world ($\Re^3$) to the 2D world ($\Re^2$). For ease of notation and computation, computer scientists employ homogeneous coordinates to denote these transformations.

1.3.1 Image Coordinates

Before we start with the derivation of the pinhole model, a quick note about image coordinates. In Computer Graphics, image coordinates are typically given in a coordinate system with the origin in the bottom left, an horizontal X-axis pointing right and a vertical Y-axis pointing up. Computer Vision scientists prefer the coordinate system of Fig. 1.1. The origin lies at the top left, the Y-axis points down and the X-axis to the right. This setup has several advantages:

- The way X- and Y-coordinates are given coincides with the way an image is read out by a digital camera with a CCD: starting at the top left and reading line by line.
- The Z-coordinate is pointing away from the image into the world. This means it coincides with the depth of a point w.r.t. the camera. Conceptually this is nice because the depth is the typical unknown in 3D reconstruction problems.
- The coordinate system is right-handed.

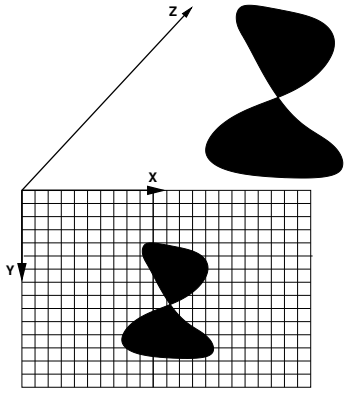


Figure 1.1: Image coordinates

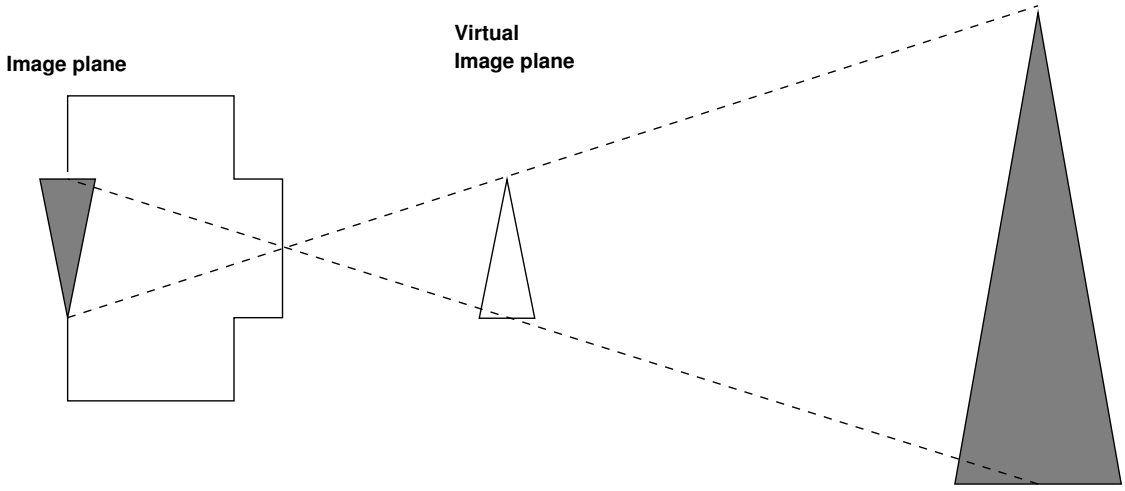


Figure 1.2: The *camera obscura* inspired the pinhole model

1.3.2 The Linear Pinhole Model

The camera model that will be used throughout this text is the linear pinhole model. This camera model is inspired by the *camera obscura*. In it's simplest form the camera obscura is no more than a black box of which side one side is punctured to yield a small hole. On the opposite side, photosensitive paper is attached. Since light travels in straight lines, all light rays going through the hole and falling on the paper will come directly from the 3D world outside the box. On this paper it forms a *negative* image of the world as can be seen in Fig. 1.2. . The amount of light that falls into the box through the small hole is very limited. One can increase this by making the hole bigger but then rays coming from different 3D points can fall onto the same point on the image. One way of getting around this problem is by making use of lenses which focus the light. Complicated formulas exist that model the effect of lenses but in essence the behaviour can still be linearized to yield the same pinhole model we will discuss.

As mentioned, the camera obscura creates a negative image of the world behind the camera-opening. This complicates the mathematical description of the camera. We call the camera-opening the camera center. Let us imagine a *virtual image plane* in front of the camera center at the same distance as the real image plane behind it. The light rays coming from the world and falling onto the camera center intersect this virtual image plane and form a positive image which is clearly shown in Fig. 1.2.

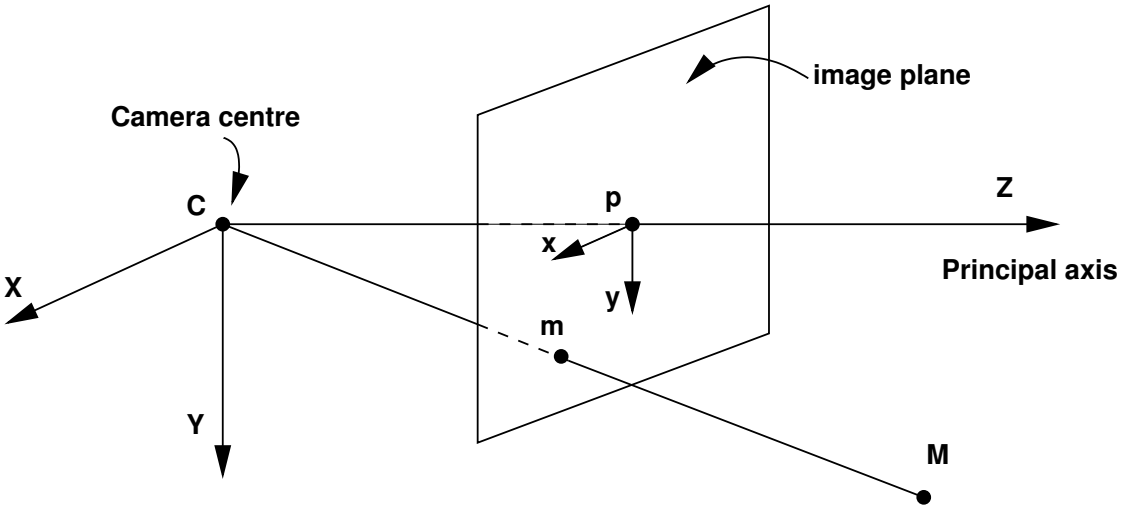


Figure 1.3: The pinhole camera model

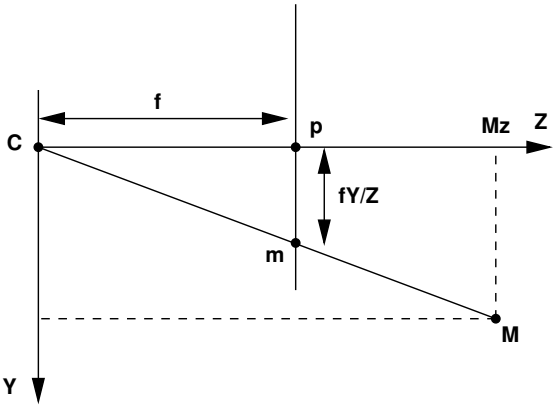


Figure 1.4: The pinhole setup, viewed from the direction of the X-axis

Simple Setup

The mathematical set-up of the pinhole model is shown in Fig. 1.3. It shows a 3D point M. To compute the projection of this point in the image, we connect the camera center C with M and intersect the image plane with this ray, yielding the point m. For the time being we assume that the camera center coincides with the origin of the world. The X- and Y-axes are parallel to the image plane and the Z-axis is the principal axis (the line orthogonal to the image plane and through the camera center) and denotes the depth of a point w.r.t. the camera center. The coordinates in the image are for the moment still expressed w.r.t. the point `p` which is the intersection of the image plane with the principal axis.

If we view the pinhole setup of Fig. 1.3 from the direction of the X-axis, we obtain Fig. 1.4. The image plane is reduced to a line and the distance between this plane and the camera center is f, the *focal length*. The 3D point `M` has coordinates $(X, Y, Z)^T$ and we define the 3D point $\mathtt{M_z}$ as the point on the Z-axis with coordinates $(0, 0, Z)^T$. From congruent triangles $\mathtt{M\text{-}M_z\text{-}C}$ and `m-p-C` we see that the y-coordinate of `m` in the image plane is $\frac{fY}{Z}$, with Y and Z Euclidean coordinates of `M`. The x coordinate of `m` is found analogously from the Y-axis view as $\frac{fX}{Z}$. This means that in this setup, the 3D point $(X, Y, Z)^T$ is projected onto the 2D point $(fX/Z, fY/Z)^T$ or in homogeneous coordinates $(fX, fY, Z)^T$. The projection can hence be written in the form of a matrix multiplication as

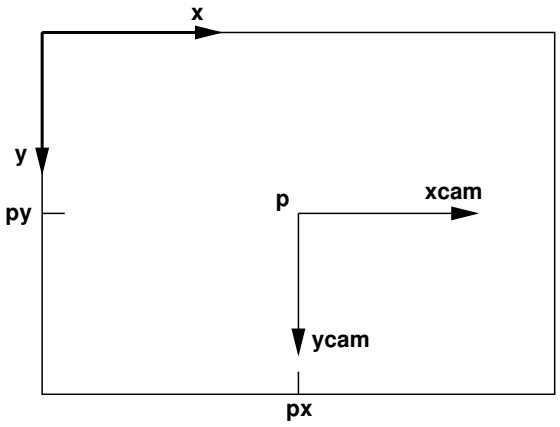


Figure 1.5: The coordinates of the principal point must be taken into account

$$\mathtt{m} \simeq \begin{pmatrix} fX \\ fY \\ Z \end{pmatrix} = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{1.3}$$

Eq. 1.3 needs to be read from right to left. The 3D point `M` is projected through a projection matrix with 3x4 elements onto the 2D point `m`. All projection equations (of which Eq. 1.3 is a simplified version) have this form.

Principal Point Offset

As explained in 1.3.1 we prefer the origin of the image coordinate system to be at the top left of the image. This means we have to take into account the offset given by the *principal point*. This is the intersection point of the principal axis (Z) and the image plane. If the coordinates of this point are given by (p_x and p_y) as shown in Fig. 1.5, then the projection of the point `M` is $(fX/Z + p_x, fY/Z + p_y)^T$ and the projection equation becomes

$$\mathtt{m} \simeq \begin{pmatrix} fX + Zp_x \\ fY + Zp_y \\ Z \end{pmatrix} = \begin{pmatrix} f & 0 & p_x & 0 \\ 0 & f & p_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

Pixel Coordinates

The coordinates of the point `m` are still expressed in the units of the world coordinate system (meter, millimeter, ...). Since images are nothing more than arrays of pixels, measurements in images are done in pixel coordinates. We therefore want to transform the coordinates to pixels. This done by multiplication with a matrix which holds the conversion from world-units to pixels:

$$\mathbf{T}_{world}^{pixel} = \begin{pmatrix} m_1 & 0 & 0 \\ 0 & m_2 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

where m_1 and m_2 are the scale factors that hold how many pixels go into one world-unit. The projection matrix now becomes

$$\begin{pmatrix} m_1 & 0 & 0 \\ 0 & m_2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} f & 0 & p_x & 0 \\ 0 & f & p_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} \alpha_x & 0 & u_x & 0 \\ 0 & \alpha_y & u_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

with α_x and α_y the focal length in pixels for the x and y direction and u_x and u_y the principal point in pixels. In fact there is one more linear term that can be introduced here. Analogue cameras are known to sometimes suffer from *skew*, i.e. the fact that pixels are parallelograms and not rectangles. We can model this by adding a skew factor in the equation.

$$\mathtt{m} \simeq \begin{pmatrix} \alpha_x & s & u_x & 0 \\ 0 & \alpha_y & u_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

This skew-factor s is inversely proportional to the tangent of the angle between the X and Y axis. Digital cameras have rectangular pixels, hence their skew is always zero. The parameters α_x, α_y, s, u_x and u_y define the internal behaviour of the camera and are called the *intrinsic parameters* or *intrinsics* of the camera.

Camera Rotation and Translation

Until now we assumed that the camera-coordinates coincided with the world coordinates. We want to be able to model cameras anywhere in the world, i.e. with a known translation and rotation w.r.t. the world coordinate system. If the rotation and translation are given by a 3×3 rotation matrix $\mathbf{R}$ and a 3×1 translation vector $\mathbf{t}$ respectively, then the transformation of the coordinate system of the camera in the world is given by

$$\mathbf{T}_{cam}^{world} = \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{pmatrix}$$

Since any rotation matrix is orthonormal, the inverse transformation is given by

$$(\mathbf{T}_{cam}^{world})^{-1} = \begin{pmatrix} \mathbf{R}^T & -\mathbf{R}^T\mathbf{t} \\ \mathbf{0} & 1 \end{pmatrix}$$

This means that a point M with coordinates $(X, Y, Z, 1)^T$ in the world coordinate frame, can be expressed in the camera coordinate frame by multiplying it with this inverse matrix.

$$\mathtt{M}_{\mathtt{cam}} = (\mathbf{T}_{cam}^{world})^{-1}\mathtt{M} = \begin{pmatrix} \mathbf{R}^T & -\mathbf{R}^T\mathbf{t} \\ \mathbf{0} & 1 \end{pmatrix} \mathtt{M}$$

Hence, the projection equation becomes

$$\mathtt{m} \simeq \begin{pmatrix} \alpha_x & s & u_x & 0 \\ 0 & \alpha_y & u_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} \mathbf{R}^T & -\mathbf{R}^T\mathbf{t} \\ \mathbf{0} & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

Since the fourth column of the first 3×4 matrix is 0, the fourth row of the matrix with the geometric transformation is of no importance. Bundling the intrinsics in a 3×3 matrix $\mathbf{K}$,

$$\mathbf{K} = \begin{pmatrix} \alpha_x & s & u_x \\ 0 & \alpha_y & u_y \\ 0 & 0 & 1 \end{pmatrix} \tag{1.4}$$

the projection equation becomes

$$\begin{aligned} \mathtt{m} &\simeq \mathbf{K}\left(\mathbf{R}^T | -\mathbf{R}^T\mathbf{t}\right)\mathtt{M} \\ &\simeq \mathbf{P}\mathtt{M} \end{aligned} \tag{1.5}$$



Figure 1.6: An example of a radially distorted image

Decomposition of P

If only the 3×4 projection matrix $\mathbf{P}$ is known, it is possible to retrieve the intrinsic and extrinsic parameters from it. This process is called *decomposing* $\mathbf{P}$ into $\mathbf{K}$, $\mathbf{R}$ and $\mathbf{t}$ and is done as follows. Inspect the upper left 3×3 sub-matrix of $\mathbf{P}$. This matrix is formed by multiplying $\mathbf{K}$ and $\mathbf{R}^T$. The inverse of this matrix is $\mathbf{RK}^{-1}$. The QR-decomposition of a 3×3 matrix of rank 3 decomposes the matrix into two matrices: an orthonormal matrix on the left and an upper triangular matrix on the right. This is exactly the decomposition we are looking for since $\mathbf{R}$ is a rotation matrix and thus orthonormal and $\mathbf{K}$ is by definition an upper triangular matrix, so $\mathbf{K}^{-1}$ is as well. When $\mathbf{K}$ and $\mathbf{R}$ are known, $\mathbf{t}$ is easily computed by pre-multiplying the fourth column of $\mathbf{P}$ with $-\mathbf{RK}^{-1}$.

1.3.3 Non-Linear Distortions

The projection model we described in 1.3.2 is linear which means it can only model linear effects. Images taken by real cameras experience non-linear deformations or distortions which make the simple linear pinhole model inaccurate. The most important and best known non-linear distortion is *radial distortion*. Figure 1.6 shows an example of a radially distorted image. Radial distortion is an alteration in magnification from a center to any point in the image, measured in a radial direction from the center of distortion. In other words, the larger the distance between an image point and the center of distortion, the larger the effect of the distortion. Thus, the effect of the distortion is mostly visible near the edges of the image. This can clearly be seen in Fig. 1.6. Straight lines near the edges of the image are no longer straight but are bent. For practical use the center of radial distortion can be assumed to coincide with the principal point, which usually also coincides with the center of the image.

Radial distortion is a non-linear effect and is typically modeled using a Taylor expansion. It can be shown that only the even order terms play a role in this expansion, i.e. the effect is symmetric around the center. The effect takes place in the lens, hence mathematically the radial distortion should be between the external and internal parameters of the pinhole model. The model we will propose here follows this strategy. Let us define

$$\mathtt{m}_\mathtt{u} = \begin{pmatrix} m_{ux} \\ m_{uy} \\ m_{uw} \end{pmatrix} \simeq \left(-\mathbf{R}^T | -\mathbf{R}^T\mathbf{t}\right) \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

The distance r from the optical axis is then

$$r^2 = \left(\frac{m_{ux}}{m_{uw}}\right)^2 + \left(\frac{m_{uy}}{m_{uw}}\right)^2$$



Figure 1.7: Distorted image (left) and undistorted image (right)

We now define $\mathtt{m}_\mathtt{d}$ as

$$\mathtt{m}_\mathtt{d} = \begin{pmatrix} m_{dx} \\ m_{dy} \\ m_{dw} \end{pmatrix} \simeq \begin{pmatrix} (1 + \kappa_1 r^2 + \kappa_2 r^4 + \kappa_3 r^6 + \ldots)\, m_{ux} \\ (1 + \kappa_1 r^2 + \kappa_2 r^4 + \kappa_3 r^6 + \ldots)\, m_{uy} \\ m_{uw} \end{pmatrix} \tag{1.6}$$

The lower order terms of this expansion are the most important ones and typically one does not compute more than three parameters (κ_1, κ_2, κ_3). Finally the projection $\mathtt{m}$ of the 3D point $\mathtt{M}$ is

$$\mathtt{m} \simeq \mathbf{K}\mathtt{m}_\mathtt{d} \tag{1.7}$$

When the distortion parameters are known, the image can be undistorted in order to make all lines straight again and thus make the linear pinhole model valid. The original and undistorted version of Fig. 1.6 is shown in Fig. 1.7.

The model described above puts the radial distortion parameters between the extrinsic and linear intrinsic parameters of the camera. In literature one often finds that the distortion is put on the left of the intrinsics, i.e. a 3D points is first projected into the image via the linear model and then undistorted. Conceptually the latter model is less suited than the one used here because putting the distortion at the end makes it dependent on the internal parameters, especially on the focal length. This means one has to re-estimate the radial distortion parameters every time the focal length changes. This is not necessary in our model. This however does not mean our model is perfect. In reality the center of radial distortion (now assumed to be the principal point) sometimes changes when the focal length is altered. This effect can not be modeled with our model.

1.4 Deformation of planar objects under projection

In this section we discuss the different types of deformations to be expected under projection of planar objects. We study 3 cases:

1. The curves are viewed from a perpendicular direction and can only rotate about an axis parallel to this direction, any 3D translation is allowed.

2. The curves can be viewed from any direction but should be at a sufficient distance from the camera so as to make the differences in depth (Z) values small relative to the distance to the camera.

3. The curves can be viewed from any direction and can be placed at arbitrary distances from the camera.

It is clear that the camera-object configurations become increasingly arbitrary going down this list. As we will see, the corresponding geometries are of increasing complexity.

1.4.1 Case 1

We suppose the object is described by specifying the three-dimensional coordinates (X, Y, Z). As we know, in this case we only allow the object to rotate around an axis parallel to the Z axis. Furthermore any three-dimensional translation can be applied. If we denote the coordinates of the object after such transformation as (X', Y', Z'), we find a relationship of the form

$$\begin{array}{rclcrcl} X' & = & \cos\theta X & - \sin\theta Y & & + t_1, \\ Y' & = & \sin\theta X & + \cos\theta Y & & + t_2, \\ Z' & = & & & Z & + t_3, \end{array}$$

Now, suppose that the points (X, Y, Z) project to image points (x, y) and the points (X', Y', Z') to (x', y'). We are interested in the transformation bringing the coordinates (x, y) into correspondence with (x', y'), i.e. in the group of transformations describing the deformations between two camera views.

This relationship is particularly easy to derive in this case. Since the planes containing the two objects, i.e. before and after the transformation, are both parallel to the image plane, points of each of the objects will have a constant Z value. This brings us in the situation where the projection equations simplify to a scaling. We have that

$$\begin{array}{rclcrcl} x & = & kX, & & x' & = & k'X', \\ y & = & kY, & & y' & = & k'Y', \end{array}$$

with $k = f/Z$ and $k' = f'/Z'$ two constant scale factors. Thus, using the relationships between X' and X we e.g. find that

$$\frac{x'}{k'} = \cos\theta \, \frac{x}{k} - \sin\theta \, \frac{y}{k} + t_1.$$

Also using the relationship between Y' and Y we obtain

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \frac{k'}{k} \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_1 \\ t_2 \end{pmatrix}.$$

We conclude that the image projection of the object undergoes a similarity transformation. In particular, the object is transformed via combined scaling, rotation, and translation in the image plane. As a further simplification we could consider the case where the object doesn't come closer to or move away from the camera, i.e. $t_3 = 0$. Then $Z' = Z$, hence $k' = k$ if also the camera focal length remains unchanged then the transformation becomes a Euclidean motion, i.e. a combination of rotation and translation (in the image plane). This case is highly relevant in industrial applications where cameras often are placed at a fixed distance from e.g. a conveyor, and looking perpendicularly upon it.

1.4.2 Case 2

In this case the object can undergo arbitrary three-dimensional rotations and translations but the projection model is simplified to a scaling:

$$\begin{pmatrix} X' \\ Y' \\ Z' \end{pmatrix} = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} + \begin{pmatrix} t_1 \\ t_2 \\ t_3 \end{pmatrix}$$

with (r_{ij}) a rotation matrix and (t_i) a translation vector, and

$$\begin{aligned} x &= kX, & x' &= k'X', \\ y &= kY, & y' &= k'Y'. \end{aligned}$$

Using much of a procedure as in case 1, we easily derive

$$x' = \frac{k'}{k} r_{11} x + \frac{k'}{k} r_{12} y + k'\ r_{13} Z + k'\ t_1.$$

In order to eliminate Z we use the planarity of the object (X, Y, Z) which implies that real numbers a, b, c and d can be found such that

$$aX + bY + cZ + d = \frac{a}{k} x + \frac{b}{k} y + cZ + d = 0.$$

Hence

$$Z = -\frac{a}{kc} x - \frac{b}{kc} y - \frac{d}{c}.$$

Using this equation to eliminate Z and carrying out similar calculations for y' we arrive at a relationship of the type

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}.$$

This is a two-dimensional *affine transformation* or *affinity*.

1.4.3 Case 3

We again assume that the object undergoes an arbitrary three-dimensional motion (rotation + translation), but we take the perspective projection model this time:

$$\begin{aligned} x &= f\tfrac{X}{Z}, & x' &= f'\tfrac{X'}{Z'}, \\ y &= f\tfrac{Y}{Z}, & y' &= f'\tfrac{Y'}{Z'}. \end{aligned}$$

Expanding the expression for x' yields:

$$\begin{aligned} x' &= f'\tfrac{X'}{Z'} \\ &= f'\frac{r_{11}X + r_{12}Y + r_{13}Z + t_1}{r_{31}X + r_{32}Y + r_{33}Z + t_3} \\ &= f'\frac{r_{11}\frac{fX}{Z} + r_{12}\frac{fY}{Z} + r_{13}\frac{fZ}{Z} + t_1\frac{f}{Z}}{r_{31}\frac{fX}{Z} + r_{32}\frac{fY}{Z} + r_{33}\frac{fZ}{Z} + t_3\frac{f}{Z}} \end{aligned}$$

Thus,

$$\begin{aligned} x' &= f'\frac{r_{11}x + r_{12}y + r_{13}f + \frac{ft_1}{Z}}{r_{31}x + r_{32}y + r_{33}f + \frac{ft_3}{Z}} \\ y' &= f'\frac{r_{21}x + r_{22}y + r_{23}f + \frac{ft_2}{Z}}{r_{31}x + r_{32}y + r_{33}f + \frac{ft_3}{Z}} \end{aligned}.$$

Because we know the point (X, Y, Z) lies in our contour plane, we can express Z as a function of (x, y) as follows

$$Z = \frac{-df}{ax + by + cf}.$$

Thus

$$x' = f' \frac{(r_{11} - \frac{t_1}{d}a)x + (r_{12} - \frac{t_1}{d}b)y + (r_{13} - \frac{t_1}{d}c)f}{(r_{31} - \frac{t_3}{d}a)x + (r_{32} - \frac{t_3}{d}b)y + (r_{33} - \frac{t_3}{d}c)f},$$
$$y' = f' \frac{(r_{21} - \frac{t_2}{d}a)x + (r_{22} - \frac{t_2}{d}b)y + (r_{23} - \frac{t_2}{d}c)f}{(r_{31} - \frac{t_3}{d}a)x + (r_{32} - \frac{t_3}{d}b)y + (r_{33} - \frac{t_3}{d}c)f}.$$

We find a two-dimensional *projective transformation* or *projectivity*, i.e. a transformation of the type

$$x' = \frac{p_{11}x + p_{12}y + p_{13}}{p_{31}x + p_{32}y + p_{33}}$$
$$y' = \frac{p_{21}x + p_{22}y + p_{23}}{p_{31}x + p_{32}y + p_{33}}$$

In homogeneous coordinates one can write this as

$$\begin{pmatrix} x' \\ y' \\ w' \end{pmatrix} \simeq \mathbf{H} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{1.8}$$

where $\mathbf{H}$ is often called a *planar homography*.

1.5 Deformation of 3D objects under projection

In the previous section we've seen how planar objects can be transformed with Euclidean, similarity, affine and projective transformations when imaged. Table. 1.1 gives an overview of the transformation matrices and the effect on the planar objects. The third column of the table shows the effect every transformation can have on a square.

1.5.1 Stratification of transformations of 3D objects

A logical next step is to extend this discussion to general 3D objects. We will not go through the derivation of the formulas. The different transformations for 3D objects are given by table 1.2. A few notes:

- A similarity transformation is sometimes called a *metric* transformation.
- The 3×3 matrix on the upper left in the Euclidean and similarity transformation is a rotation matrix. These matrices are orthonormal and can be described by three parameters, e.g. three angles of rotation around the three axes. Hence the amount of degrees of freedom (DOG) for the Euclidean and metric transformations are 6 (3 rotation + 3 translation) and 7 (3 rotation + 3 translation + 1 scale) respectively.
- The 3×3 matrix on the upper left in the affine transformation is no longer orthonormal.
- A projective transformation is the most general transformation and has 15 degrees of freedom: 16-1 (homogeneous coordinates describe points up to a scale factor).

1.5.2 Transformations and the projection equation

What is the influence of the above mentioned transformations on the camera model we have derived? Let's revisit the projection equation of Eq. 1.5. What happens to it if we transform the parts of this equation, e.g. with a projective transformation $\mathbf{T}$.

The 3D point $\mathtt{M}$ is transformed by multiplying it on the left with the matrix $\mathbf{T}$:

$$\mathtt{M}' = \mathbf{T}\mathtt{M}$$

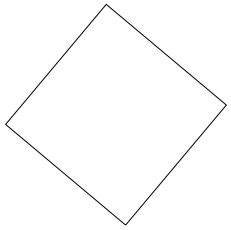
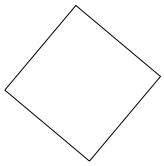
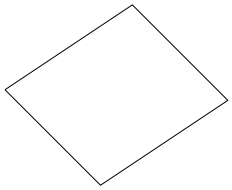
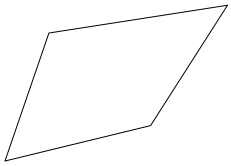
Name	**DOF**	**Transformation**	**Effect**
Euclidean	3	$\begin{pmatrix} \cos\theta & -\sin\theta & t_x \\ \sin\theta & \cos\theta & t_y \\ 0 & 0 & 1 \end{pmatrix}$	
Similarity	4	$\begin{pmatrix} k\cos\theta & -k\sin\theta & t_x \\ k\sin\theta & k\cos\theta & t_y \\ 0 & 0 & 1 \end{pmatrix}$ $= \begin{pmatrix} \cos\theta & -\sin\theta & t_x \\ \sin\theta & \cos\theta & t_y \\ 0 & 0 & \frac{1}{k} \end{pmatrix}$	
Affine	6	$\begin{pmatrix} a_{11} & a_{12} & b_1 \\ a_{21} & b_{22} & b_2 \\ 0 & 0 & 1 \end{pmatrix}$	
Projective	8	$\begin{pmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{pmatrix}$	

Table 1.1: Effect of different transformations on planar objects, illustrated by the effect on a square

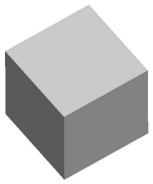
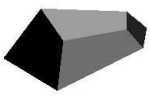
Name	**DOF**	**Transformation**	**Effect**
Euclidean	6	$\begin{pmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$	
Similarity	7	$\begin{pmatrix} kr_{11} & kr_{12} & kr_{13} & t_x \\ kr_{21} & kr_{22} & kr_{23} & t_y \\ kr_{31} & kr_{32} & kr_{33} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & \frac{1}{k} \end{pmatrix}$	
Affine	12	$\begin{pmatrix} a_{11} & a_{12} & a_{13} & b_1 \\ a_{21} & a_{22} & a_{23} & b_2 \\ a_{31} & a_{32} & a_{33} & b_3 \\ 0 & 0 & 0 & 1 \end{pmatrix}$	
Projective	15	$\begin{pmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \\ p_{41} & p_{42} & p_{43} & p_{44} \end{pmatrix}$	

Table 1.2: Effect of different transformations on 3D objects

The camera's extrinsics are transformed as well. The frame of the camera was given by the matrix

$$\mathbf{T}_{cam}^{world} = \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{pmatrix}$$

which needs to be transformed with the transformation $\mathbf{T}$ by multiplying it on the left.

$$\mathbf{T}_{cam}^{world'} = \mathbf{T} \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{pmatrix}$$

However, the camera equation needs the inverse of $\mathbf{T}_{cam}^{world}$. The transformation of this matrix is

$$(\mathbf{T}_{cam}^{world'})^{-1} = \begin{pmatrix} \mathbf{R}^T & -\mathbf{R}^T\mathbf{t} \\ \mathbf{0} & 1 \end{pmatrix} \mathbf{T}^{-1}$$

Hence, the transformed projection matrix becomes

$$\mathbf{P}' = \mathbf{K} \left(\mathbf{R}^T | -\mathbf{R}^T\mathbf{t} \right) \mathbf{T}^{-1}$$

This means that the projection equation in the transformed world reads:

$$\begin{aligned} \mathtt{m}' &\simeq \mathbf{P}'\mathtt{M}' \\ &\simeq \mathbf{K} \left(\mathbf{R}^T | -\mathbf{R}^T\mathbf{t} \right) \mathbf{T}^{-1}\mathbf{T}\mathtt{M} \\ &\simeq \mathbf{K} \left(\mathbf{R}^T | -\mathbf{R}^T\mathbf{t} \right) \mathtt{M} \\ &\simeq \mathtt{m} \end{aligned}$$

Thus, the projection of the transformed 3D point through the transformed camera yields the same 2D point as before. In the projective world (the real world transformed with an arbitrary projective transformation), any 3×4 matrix can be seen as a projection equation

$$\mathtt{m} \simeq \mathbf{P}\mathtt{M}$$

where $\mathbf{P}$ can no longer be decomposed meaningfully in a multiplication of intrinsic and extrinsic matrices.

1.6 Calibration

As explained before a perspective camera can be described by its intrinsic and extrinsic parameters. The process of determining these parameters is known as *camera calibration*. We make a distinction between intrinsic or internal calibration and extrinsic or external calibration, also known as pose-estimation.

1.6.1 Internal Calibration

Several techniques have been developed over the years to recover the intrinsic parameters of a camera. Typical calibration procedures use the paradigm that it is possible to extract the camera parameters from a set of known 3D-2D correspondences, i.e. a set of 3D coordinates with corresponding 2D coordinates in the image for their projections. In order to easily obtain such 3D-2D correspondences, these calibration techniques employ calibration objects, like the ones displayed in Fig 1.8. These objects all have in common that easily recognizable markers are applied to them. A typical calibration algorithm therefore follows the following steps

- Construct a calibration object (either 3D or 2D) like the ones in Fig 1.8. Measure the 3D positions of the markers on this object.
- Take one or more (depending on the algorithm) pictures of the object with the camera to be calibrated.



Figure 1.8: Calibration objects in 3D (left) and 2D (right)

- Extract and identify the markers in the image.
- Fit a linearized calibration model to the 3D-2D correspondences found in the previous step.
- Improve the calibration with a non-linear optimization step.

Tsai's and Zhang's methods

As mentioned, both 2D and 3D calibration objects can be used to perform the calibration. In his famous article Tsai [64] describes an algorithm to calibrate a camera with either 2D or 3D calibration objects. He follows the procedure explained above. Using his technique it is possible to retrieve not only the intrinsic but also the extrinsic parameters of the camera. Using the 2D calibration object, some degenerate setups can appear however, the most important one consisting of the calibration plane being parallel to the image plane.

A flexible technique has been described by Zhang [68]. A 2D calibration object is moved several times and an image is taken by the camera every time. The internal camera calibration is easily extracted from these images. Jean-Yves Bouguet has made a matlab implementation of this algorithm available on the Internet [8]. For the interested reader, the algorithm is explained below.

Notation

The projection equation 1.5 and the definition of $\mathbf{K}$ (Eq. 1.4) are used with a set of 3D points $\mathtt{M}$ and corresponding 2D points $\mathtt{m}$. The calibration object is planar and is moved in front of the camera. Such images can also be interpreted as recorded from a fixed object and a moving camera. Because the 3D points on the object are coplanar, we can set their Z-coordinate to 0 without loss of generality.

$$\mathtt{m} \simeq \mathbf{K}\left(\mathbf{R}^T| - \mathbf{R}^T\mathbf{t}\right)\mathtt{M} = \mathbf{K}\left(\mathbf{r_1}\,\mathbf{r_2}\,\mathbf{r_3}\,\mathbf{t}'\right)\begin{pmatrix} X \\ Y \\ 0 \\ 1 \end{pmatrix} = \mathbf{K}\left(\mathbf{r_1}\,\mathbf{r_2}\,\mathbf{t}'\right)\begin{pmatrix} X \\ Y \\ 1 \end{pmatrix}$$

with $\mathbf{r_i}$ the columns of the rotation matrix $\mathbf{R}^T$ and $\mathbf{t}'$ the vector $-\mathbf{R}^T\mathbf{t}$.

Step 1

Inspecting equation 1.8 this can be seen as a homography between $\mathtt{m}$ and $(X, Y, 1)^T$ with the homography matrix $\mathbf{H}$ being

$$\mathbf{H} = (\mathbf{h_1}\,\mathbf{h_2}\,\mathbf{h_3}) = \lambda\mathbf{K}\left(\mathbf{r_1}\,\mathbf{r_2}\,\mathbf{t}'\right) \tag{1.9}$$

with λ a non-zero scale factor. Thus, if we have identified the correspondences between the 3D points `M` and the 2D points `m`, we can easily write down Eq. 1.8 for all these correspondences and solve for the homography $\mathbf{H}$. Paragraph 2.2.6 will explain how this can be done robustly.

Step 2
Any rotation matrix is an orthonormal matrix. This means that the norm of any column is 1 and that different columns are orthogonal, i.e. their scalar product is zero. From Eq. 1.9 we can write for the first and second column:

$$\begin{aligned} \mathbf{h_1}^T\mathbf{K}^{-T}\mathbf{K}^{-1}\mathbf{h_2} &= 0 \\ \mathbf{h_1}^T\mathbf{K}^{-T}\mathbf{K}^{-1}\mathbf{h_1} &= \mathbf{h_2}^T\mathbf{K}^{-T}\mathbf{K}^{-1}\mathbf{h_2} \end{aligned} \tag{1.10}$$

These are two constraints we can write down for every computed homography (thus for every image of the calibration pattern). If we now define the symmetric matrix $\mathbf{B}$ as

$$\mathbf{B} = \mathbf{K}^{-T}\mathbf{K}^{-1} = \begin{pmatrix} B_{11} & B_{12} & B_{13} \\ B_{12} & B_{22} & B_{23} \\ B_{13} & B_{23} & B_{33} \end{pmatrix}$$

it is clear that every recovered homography offers two linear equations in elements of $\mathbf{B}$, obtained from Eq. 1.10. If 3 or more images of the calibration are taken, $\mathbf{B}$ can be solved for linearly, up to scale factor λ. Since $\mathbf{B}$ is defined as $\mathbf{K}^{-T}\mathbf{K}^{-1}$ and $\mathbf{K}$ is defined by Eq. 1.4, the 5 intrinsic parameters plus λ can easily be derived from knowledge of the 6 elements of $\mathbf{B}$. We leave it as an exercise to the reader to derive these equations for the internal parameters of $\mathbf{K}$

$$\begin{aligned} u_y &= \frac{B_{12}B_{13} - B_{11}B_{23}}{B_{11}B_{22} - B_{12}^2} \\ \lambda &= B_{33} - \frac{B_{13}^2 + u_y(B_{12}B_{13} - B_{11}B_{23})}{B_{11}} \\ \alpha_x &= \sqrt{\frac{\lambda}{B_{11}}} \\ \alpha_y &= \sqrt{\frac{\lambda B_{11}}{B_{11}B_{22} - B_{12}^2}} \\ s &= -\frac{B_{12}\alpha_x^2\alpha_y}{\lambda} \\ u_x &= \frac{su_y}{\alpha} - B_{13}\frac{\alpha^2}{\lambda} \end{aligned}$$

When the linear intrinsic parameters have been computed, the extrinsics readily follow from Eq.. 1.9.

$$\begin{aligned} \mathbf{r_1} &= \mu\mathbf{K}^{-1}\mathbf{h_1}, \quad \mathbf{r_2} = \mu\mathbf{K}^{-1}\mathbf{h_2}, \quad \mathbf{r_3} = \mathbf{r_1} \times \mathbf{r_2}, \quad \mathbf{t} = \mu\mathbf{K}^{-1}\mathbf{h_3} \\ \mu &= \frac{1}{\|\mathbf{K}^{-1}\mathbf{h_1}\|} \end{aligned}$$

Step 3
The intrinsic parameters of the linear pinhole camera model have now been computed. We know however that most lenses suffer from non-linear distortions of which radial distortion is the most important effect. Eq. 1.6 explained how to model this radial distortion. Since we have recovered the extrinsic parameters, we can compute $(m_{ux}, m_{uy}, m_{uz})^T$. If we define

$$n_x = \frac{m_{ux}}{m_{uw}}, \qquad n_y = \frac{m_{uy}}{m_{uw}}$$

as the ideal radially undistorted point, then the radial distorted point $\mathtt{m}$ is

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} \alpha_x n_x(1+\kappa_1 r^2+\kappa_2 r^4) + s n_y(1+\kappa_1 r^2+\kappa_2 r^4) + u_x \\ \alpha_y n_y(1+\kappa_1 r^2+\kappa_2 r^4) + u_y \\ 1 \end{pmatrix}. \tag{1.11}$$

If we want to compute the non-linear radial distortion coefficients, we can do so alternating between two linear methods.

- Compute the linear parameters, ignoring radial distortion (steps 1 and 2)
- If we project the 3D points into the image, using the linear model, deviations will still be present between the projected and extracted points, due to radial distortion. From the linear parameters and the 3D points, we compute (n_x, n_y). Equations 1.11 give us linear constraints on the radial distortion parameters κ_1 and κ_2. Stacking all these equations into one system, we can solve for these parameters.
- Use κ_1 and κ_2 to radially undistort the 2D points and feed this again to the linear parameter estimation. The result of this is again used to project the 3D points into the image to re-estimate κ_1 and κ_2 etc...

By iterating these two linear sets of equations until convergence the full camera model, including radial distortion, can be estimated.

Calibrating Radial Distortion Only

For some computer vision tasks like object recognition, complete calibration of the camera is not necessary. However, radial distortion in the images makes these tasks more difficult. That is why one sometimes simply wants to remove the radial distortion in the images. In order to remove it, the parameters of the distortion need to be known. A simple but efficient way of retrieving the parameters without a full calibration is explained below.

Inspect Fig. 1.9. The picture on the left shows a clearly distorted image and we want to undistort it. The first step in the computation of the distortion parameters consists in the detection of lines. In this example we used the well-known Canny edge detector [9]. The user then identifies a set of curves which he recognizes as lines in the world but which are bent as a result of the radial distortion. These lines are most apparent near the borders of the image because there the distance between the points and the center is large and hence the radial distortion has more influence. For each line, points are sampled along it and a straight line is fitted through this data. If we want the distortion to vanish, the error consisting of the sum of the distances of each point to their line should be zero. Hence a non-linear minimization algorithm like Levenberg-Marquardt is applied to this data. The algorithm is initialized with the distortion parameters set to zero. At every iteration, new values are computed for these parameters, the points are warped accordingly and new lines are fitted. The algorithm stops when it converges to a solution where all selected lines are straight (i.e. the resulting error is close to zero). The resulting unwarped image can be seen on the right of Fig. 1.9.

1.6.2 External Calibration

The calibration techniques of section 1.6.1 are quite intricate because they recover the internal and sometimes the external calibration of a camera. Once a camera has been internally calibrated however, computing its external calibration is a much simpler problem. The computation of the external parameters (rotation and translation) of a camera based on a set of 3D-2D correspondences is also referred to as *Pose Estimation*. The computation of the camera-pose can be done if as few as 3 3D-2D correspondences are available. Already in 1841 Grunert [20] proposed a solution

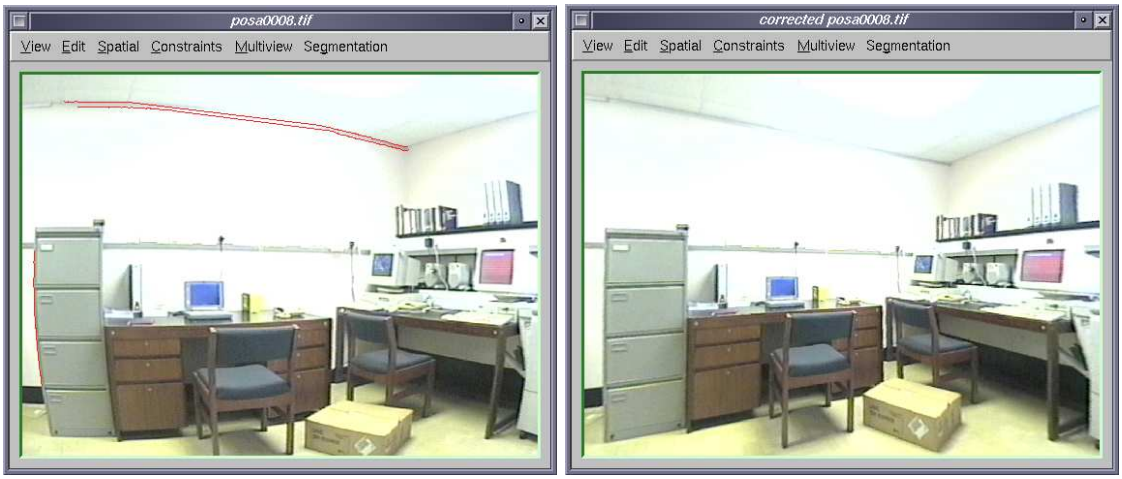


Figure 1.9: Left: distorted image with indicated curves that should be straight lines Right: undistorted image. The lines are now straight

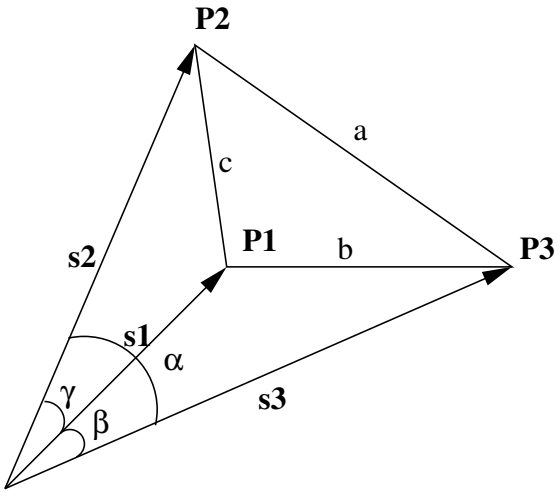


Figure 1.10: The three point perspective pose estimation problem

to this *Three Point Perspective Pose Estimation* problem, illustrated in Fig. 1.10. A review of Grunert's and other solutions can be found in [22]. The problem is stated as follows. If the triangle side lengths a, b and c and the direction of the vectors $\mathbf{s_1}$, $\mathbf{s_2}$ and $\mathbf{s_3}$ (e.g. via the angles α, β and γ) are known, then the pose estimation problem consists in computing the length of these vectors. Grunert proved that the problem boils down to solving for the roots of a fourth-degree polynomial. In order to use Grunert's algorithm for pose estimation, the following procedure is followed.

- Take 3 3D points in the world $\mathtt{M}_\mathtt{i}^\mathtt{w}$ with corresponding 2D points $\mathtt{m}_\mathtt{i}$ $(i = (1, 2, 3))$. Compute the distances a, b and c from $\mathtt{M}_\mathtt{i}^\mathtt{w}$
- Put the camera in the origin. Compute the direction of the backprojected rays $\mathbf{s_i}$ from the known intrinsic parameters.
- Compute the roots of Grunert's fourth-degree polynomial.
- One of the up to four real solutions delivers the lengths of $\mathbf{s_i}$. From $\mathbf{s_i}$, the position of the 3D points in the camera-centered frame $\mathtt{M}_\mathtt{i}^\mathtt{c}$ are readily computed.
- The Euclidean transformation (rotation and translation) between the 3D points in the camera frame and in the world frame, holds the rotation and translation of the camera in the world, i.e. the extrinsic calibration of the camera.
- If more than 1 real solution exist, check the solutions with one extra 3D-2D correspondence to retain the correct one.

Computing the transformation between the world-centered and camera-centered coordinates of the 3D points can be done as follows. We assign a local coordinate system to the 3 points in the world frame

- The Z-axis is chosen as the direction between $\mathtt{M}_1^\mathtt{w}$ and $\mathtt{M}_2^\mathtt{w}$:

$$\mathbf{Z_w} = \frac{\mathtt{M}_2^\mathtt{w} - \mathtt{M}_1^\mathtt{w}}{\|\mathtt{M}_2^\mathtt{w} - \mathtt{M}_1^\mathtt{w}\|}$$

- The X-axis is chosen as the projection of $\mathtt{M}_1^\mathtt{w} - \mathtt{M}_3^\mathtt{w}$ on the plane perpendicular to the chosen Z-axis.

$$\begin{aligned} \mathbf{X}' &= \frac{\mathtt{M}_3^\mathtt{w} - \mathtt{M}_1^\mathtt{w}}{\|\mathtt{M}_3^\mathtt{w} - \mathtt{M}_1^\mathtt{w}\|} \\ \mathbf{X}'' &= \mathbf{X}' - (\mathbf{X}'.\mathbf{Z})\mathbf{Z} \\ \mathbf{X_w} &= \frac{\mathbf{X}''}{\|\mathbf{X}''\|} \end{aligned}$$

- The Y-axis is chosen as the vector product between Z and X

$$\mathbf{Y_w} = \mathbf{Z_w} \times \mathbf{X_w}$$

- Define a rotation matrix $\mathbf{R_w}$

$$\mathbf{R_w} = (\mathbf{X_w} \;\; \mathbf{Y_w} \;\; \mathbf{Z_w})$$

In the same way $\mathbf{R_c}$ is computed for the camera centered coordinates $\mathtt{M}_\mathtt{i}^\mathtt{c}$. The rotation matrix and the translation vector of the camera in the world are then found as

$$\begin{aligned} \mathbf{R} &= \mathbf{R_w}{\mathbf{R_c}}^T \\ \mathbf{t} &= \mathtt{M}_1^\mathtt{w} - \mathbf{R_w}{\mathbf{R_c}}^T \mathtt{M}_1^\mathtt{c} \end{aligned}$$

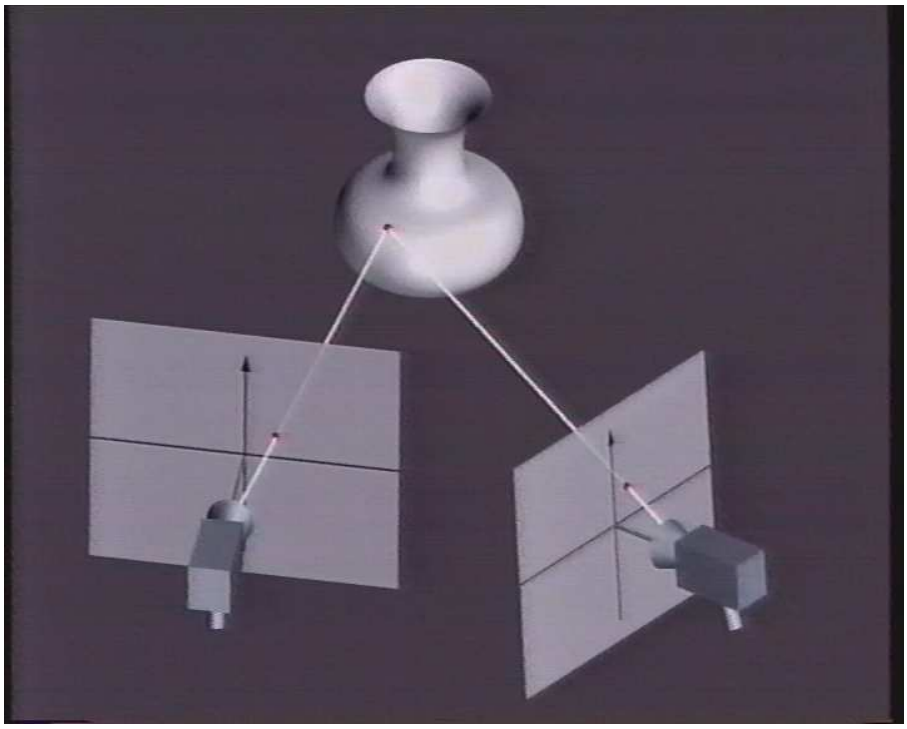


Figure 1.11: The principle behind stereo-based 3D reconstruction is very simple: given two images of a point, the point's position in space is found as the intersection of the two projection rays.

1.7 A Simple Stereo Setup

Now that we know about camera parameters and the calibration thereof, let us discuss what we can do if we put two cameras next to each other, looking at the same scene. This setup is called *stereo*. It is the simplest setup one can use to reconstruct scenes in 3D in a passive way.

Suppose we have two images, taken at the same time and from different viewpoints. The situation is illustrated in Fig. 1.11. The principle behind stereo-based 3D reconstruction is simple: given two projections of the same point, its 3D position is found as the intersection of the two projection rays through its two images. Repeating such a procedure for several points yields the 3D shape and configuration of the objects in the scene. Note that this construction – sometimes referred to as *triangulation* – requires complete knowledge of both the internal and the external parameters of the cameras.

To simplify the mathematical analysis, a very simple but often used configuration is discussed. Suppose that the two cameras have identical internal parameters and that $s = u_x = u_y = 0$. Moreover, the optical axes of the cameras are parallel and their image planes are coplanar, with coincident x-axes. This special configuration is shown in Fig. 1.12. The *baseline* b is the distance between the two camera centers. The image planes are placed in front of the optical centers of the cameras. Also the situation for the external parameters is simplified. We let the world coordinate frame coincide with the camera frame of the left camera. The images of a point with coordinates $(X, Y, Z)^t$ are then

$$\rho \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = K \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}$$

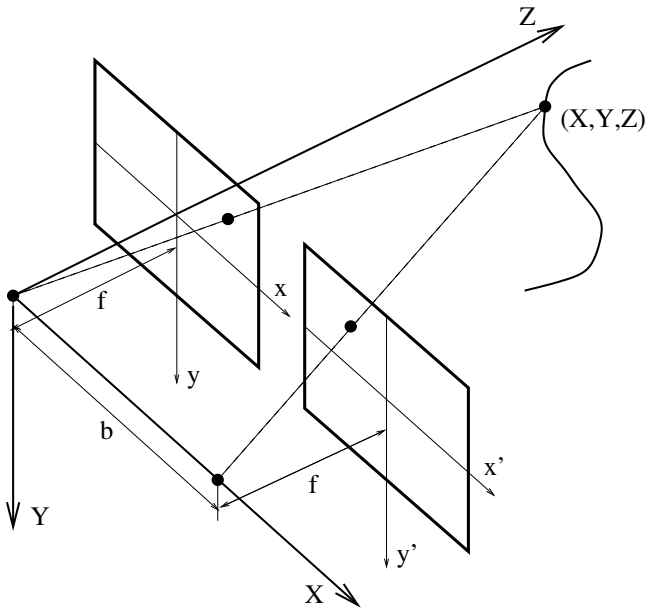


Figure 1.12: Simple stereo setup with cameras having coplanar image planes and parallel axes. Moreover, they have aligned x axes.

for the first camera (left image) and

$$\rho' \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = K \begin{pmatrix} X - b \\ Y \\ Z \end{pmatrix}$$

for the second camera (right image), where

$$K = \begin{pmatrix} fk_x & 0 & 0 \\ 0 & fk_y & 0 \\ 0 & 0 & 1 \end{pmatrix} .$$

Hence,

$$\begin{cases} x = \dfrac{fk_x X}{Z} , \\ y = \dfrac{fk_y Y}{Z} , \end{cases} \qquad \text{and} \qquad \begin{cases} x' = \dfrac{fk_x (X-b)}{Z} , \\ y' = \dfrac{fk_y Y}{Z} , \end{cases}$$

Note that $y = y'$ for all point projections. Solving these equations for X, Y, and Z yields

$$\begin{aligned} llrX &= b\frac{x}{(x - x')} , \\ Y &= b\frac{k_x}{k_y}\frac{y}{(x - x')} , \\ Z &= bk_x\frac{f}{(x - x')} . \end{aligned} \tag{1.12}$$

The difference $(x - x')$ is the so-called *disparity*. The distance Z, also referred to as the *range*, is inversely proportional to disparity. Beyond a certain distance, depth measurement will therefore become very coarse. Human stereo depth perception is limited to distances of about 10 meters.

Beyond, depth impressions arise from other cues. Note also that the disparity is directly proportional to b and f. Depth resolution can be increased by increasing one or both of these variables. Upon such a change, the same distance will correspond to a larger disparity and therefore distance sampling gets finer. It should be noted, however, that one should strike a balance between increasing resolution and keeping visible to both cameras as much of the scene as possible. Indeed, when disparities get larger, chances of finding both projections within the images diminish. Fig 1.13 shows planes of equal disparity for two focal lengths and two baseline distances. The same distance is seen to be sampled finer after the focal length or the baseline have been increased. A smaller part of the scene is visible to both cameras upon such a change, certainly for those parts which are close.

The whole method is seen to be based on finding the projection of points in both images. More specifically, if a point in the left image is selected, the problem is to find the corresponding point in the right image, assuming one exists (the point could be invisible to the right camera). Solving this *correspondence problem* is the hard part of stereo. In order to reconstruct scenes in 3D using a passive stereo setup, one has to solve "the two C's": the *calibration* of the cameras needs to be known, as well as *correspondences* between the cameras.

An example stereo image pair, taken with a simple camera configuration as discussed in this section, is shown in Fig. 1.14. Two views of the three-dimensional reconstruction, obtained by the calculation of stereo disparities, is shown in Fig. 1.15. The results are quite convincing for the torsos, but the homogeneous wall in the background could not be reconstructed. The difference lies in the presence or absence of texture, resp., and the related ease or difficulty of finding corresponding points. In the untextured background all points look the same and the search for correspondences fails.

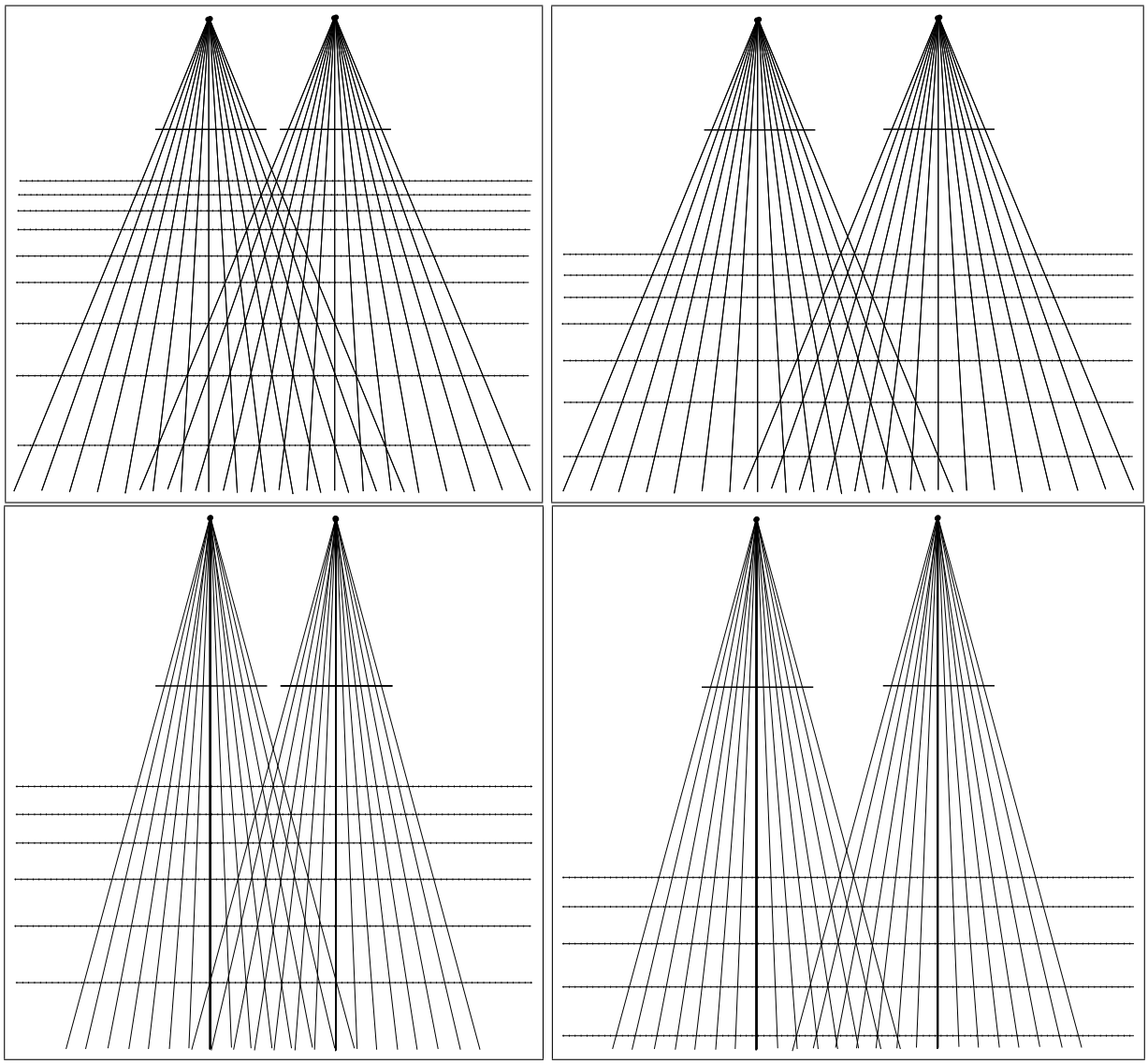


Figure 1.13: Equal disparities correspond to points of equal distance: rays that project onto points with a fixed disparity intersect in planes of constant range. At a given distance these planes get closer (i.e. sample distance more densely) if the baseline is increased (top-left to top-right), the focal length is increased (top-left to bottom-left), or both (top-left to bottom-right).

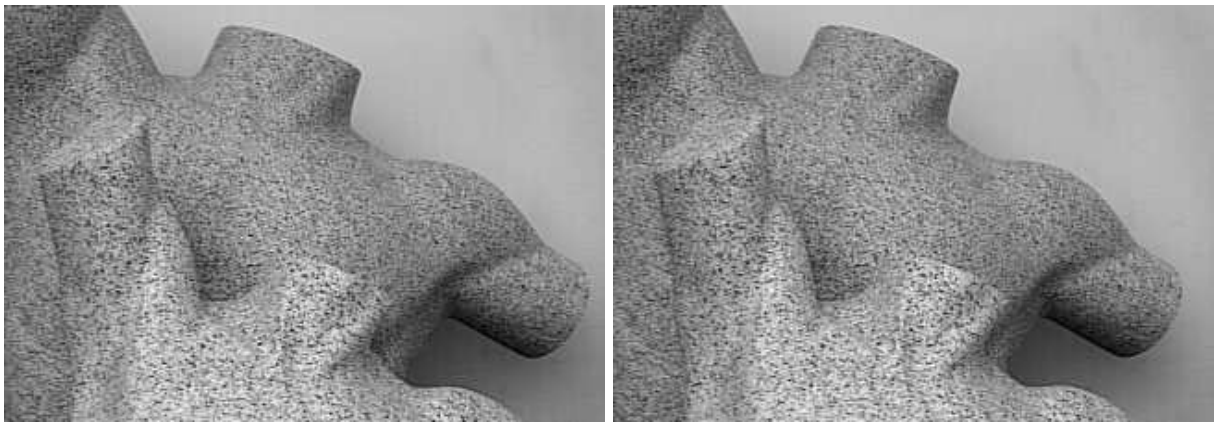


Figure 1.14: Pair of stereo images for a scene with two torsos.

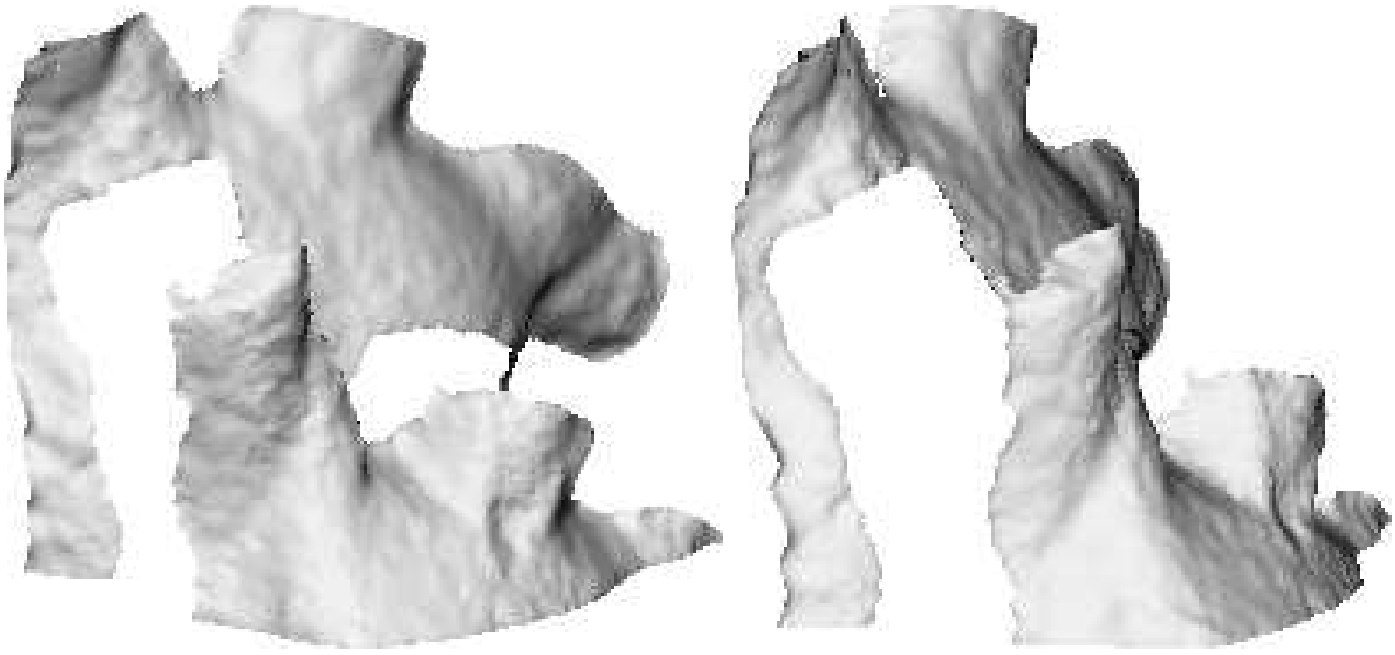


Figure 1.15: Two views of the three-dimensional reconstruction obtained for the stereo image pair of Fig. 1.14.

Chapter 2

Relating Image Pairs

2.1 Features

As explained in chapter 1, one of the important prerequisites for 3D reconstruction is the detection of correspondences between images. For a human operator, this may seem an easy task. Humans are notoriously good at employing high-level information to recognize and identify objects in images and finding these back in other images. Unfortunately, this task is much harder to perform for a computer who does not have access to this high-level information. A naive approach could be to pick a pixel in one image and search for the pixel in the other image with the same or closest color value.

Typically not all parts of an image are equally well suited to match between images. It is clear that points in homogeneous areas are very hard to distinguish and are therefore no good candidates for matching. In this section we will explain the detection and matching of *features*. These are items in an image that can be extracted and identified easily and stably in different images.

2.1.1 Invariance

A key concept in the search for good features is that of *invariance*. Invariance is defined as the quality of being resistant to variation. There are, of course, several kinds of invariance, depending what kind of variation one is resistant to. Two important types of invariance are *geometric* and *photometric* invariance.

Geometric Invariance

Let us investigate the following setup. An object is viewed by two cameras located in different positions. Can we find combinations of points, lines, curves, ... that, when computed in the different images, do not change? These combinations are geometric invariants. A wide gamut of invariants exist for the different kinds of transformations that were described in tables 1.1 and 1.2 for 2D and 3D respectively. The generality of the transformations in these tables increases as they're written down. It is easier to find a feature that is invariant against a similarity than a projective transformation for instance. Hence, if it is known beforehand that the expected transformation between different views of an object is only a similarity, then these similarity invariants suffice. In order to be invariant against all possible changes in viewpoint the feature should be projectively invariant.

Ik wil het hier niet hebben over perspectiviteiten (zie examenvraag). Of is dit wel nodig voor de algemeenheid?

A well known example of a projective invariant is the cross-ratio of four collinear points. If

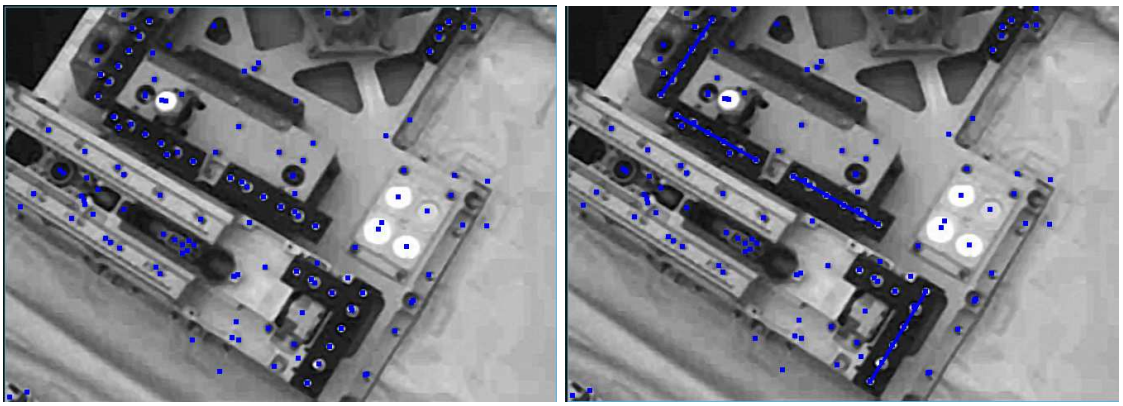


Figure 2.1: Extracted points on an image of the ETS-VII satellite (left). Identified markers as collinear white circles on a black background (right).

four points $\mathtt{m}_1$,$\mathtt{m}_2$,$\mathtt{m}_3$,$\mathtt{m}_4$ are collinear, then their cross-ratio defined as

$$Cr(\mathtt{m}_1, \mathtt{m}_2, \mathtt{m}_3, \mathtt{m}_4) = \frac{\Delta_{13}\Delta_{24}}{\Delta_{14}\Delta_{23}} \tag{2.1}$$

is projectively invariant. Where Δ_{ij} is the Euclidean distance between points i and j

$$\Delta_{ij} = \sqrt{\left(\frac{x_i}{w_i} - \frac{x_j}{w_j}\right)^2 + \left(\frac{y_i}{w_i} - \frac{y_j}{w_j}\right)^2}$$

The fact that this ratio is projectively invariant can be employed by searching for collinear points in an image and computing the cross ratio between sets of 4 points. If one finds back the same cross ratio in two images, this means that we have found back the same constellation of points. Fig. 2.1 shows a practical example of this invariant, used on images of the robot on-board the Japanese engineering test satellite ETS-VII. The first image shows a view of the task-board of this satellite. An edge-detector has been applied to the image, ellipses were fitted to the edges and the centers of these ellipses were kept. These points are shown in blue. On the task-board different markers were available. These markers consisted of collinear white circles, regularly spaced on a black background. The cross-ratio of sets of 4 points was computed on the data extracted from the image. The unique cross-ratio of the marker setup allowed us to correctly identify the visible markers in the image, as can be seen in the second image of Fig. 2.1.

In the application of Fig. 2.1, the collinear points that were identified could be employed for pose-estimation. As explained in paragraph 1.6.2, if enough 3D-2D correspondences are known, the pose of a camera can be computed from these correspondences using Grunert's algorithm if the internal parameters of the camera are known. The position of the different markers was known beforehand, so the 3D coordinates were known. Several images were taken, looking at the task-board from a variety of poses. For each of the images (e.g. the one in Fig. 2.1) the marker identification based on the cross-ratio of collinear points was applied. Once the markers were extracted, they needed to be distinguished from each other. This could be done by computing the intersection point of the different lines and using this point and the marker points in another cross-ratio computation and comparison step. The result of this step is shown in Fig. 2.2 where all marker points have been identified, effectively yielding 36 3D-2D correspondences.

The cross-ratio is invariant against projective transformations. It is one of the infinite possible invariants one can extract and use. The major drawback of these kind of invariants is the amount of points, lines, ... one has to extract and use to compute these invariants. For instance the cross-ratio demands the simultaneous extraction of 4 collinear or 5 non-collinear but coplanar points. If one has prior knowledge of the scene and this scene consists of well-defined, easily extractable

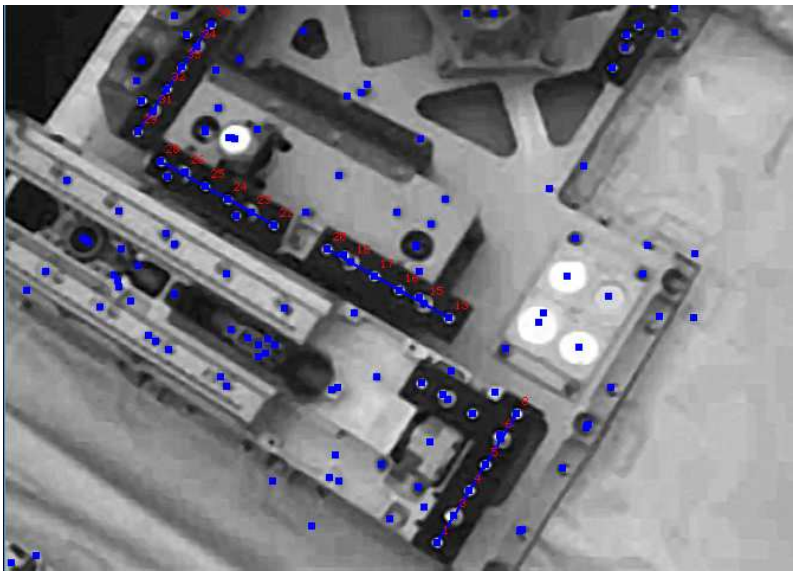


Figure 2.2: All points on the markers are now identified, yielding 36 3D-2D correspondences.

items, these invariants are applicable. One then defines beforehand the constellation of points, lines, ... to be found, computes an invariant and tries to find it back in the images. If one does not have access to this a-priori information, using these invariants becomes very hard. Indeed, in order to match features between different images using the cross-ratio, one would have to compute the cross-ratio from all sets of 4 collinear points in both images and search for matches between these. This process quickly becomes unfeasible if the amount of points increases.

Features that are highly invariant against geometric deformations can play an important role in Computer Vision applications like object recognition and also 3D reconstruction. However, it is important to understand that higher invariance comes with its price, namely less discriminating power. Let us take the example of the square in table 1.1. If we want to recognize this form under e.g. any affine transformation then we know we will have to take into account that the square can deform and end up looking like a parallelogram. However, an object that has a parallelogram shape to begin with also transforms into a parallelogram under an affine transformation. Hence, we can no longer distinguish between squares and parallelograms.

Photometric Invariance

As explained in the previous paragraph, features can be invariant against geometric transformations. There can however be other changes in the environment which make it difficult to recognize and match features between different images, the most important of which is the change in illumination conditions and viewing directions. If one looks at an object or scene from different directions, its appearance might change dramatically because of the material properties of the object which might reflect light in different ways depending on the position of the object w.r.t. the light and the camera. Imagine a freshly washed car in the afternoon sun. Highlights, reflections, even the global color of the car look different when observed from different angles. The lighting conditions themselves can have an impact too of course. The light position can change, clouds might occlude the sun, shadows might or might not appear. All these effects are combined into the color value of the pixels in the image. The invariance of features against (some of) these effects is called *photometric* invariance. Photometric invariant color spaces can be derived in which the various photometric events are separated. From these color spaces photometric invariant features can be computed which are robust with respect to shadows, shading, specularities‘ and changes of the color of the illuminant.

Features and Feature Vectors

Every use of features in the concept of finding correspondences between images employs the same strategy.

- Extract features in image 1. These features have a (2D) location in the image and are described by a feature vector or feature descriptor. This feature vector holds the invariant information of the feature. For instance for the cross-ratio, the location of a feature is the coordinates of the four points. The feature vector consists of one value, the cross-ratio.
- Extract features in image 2 independently in the same way.
- Compare features between image 1 and image 2, using the feature vector as the comparison. Features are possible matches if their feature vectors are the same or close together.

2.1.2 Simple Image Features

Although invariant features can be very stable against viewpoint- and illumination changes, in many cases we can get away with using simple image comparison techniques. The idea is to extract features that are simple points in the image with a corresponding feature vector consisting of the pixels around this point. In order to check if two features resemble each other, the two feature-vectors are compared with and image region comparison function. Several of these functions exist but the most important ones are sum-of-square-differences (SSD) and normalized cross-correlation(NCC).

Consider a window W_I in image I and the corresponding region in image J which is found by transforming the window W_I with transformation $\mathbf{T}$. The dissimilarity between two image regions based on SSD is given by

$$D = \iint_W [J(\mathbf{T}(x,y)) - I(x,y)]^2 v(x,y) dxdy \tag{2.2}$$

where $v(x,y)$ is a weighting function. Typically $v(x,y) = 1$ or it is a Gaussian. In practice, the integral is computed as a summation of the squared difference of pixels in a window in the images with width w and height h.

$$D = \sum_{x=-w}^{x=w} \sum_{y=-h}^{y=h} [J(\mathbf{T}(x,y)) - I(x,y)]^2 v(x,y) \tag{2.3}$$

Note that this value gives the dissimilarity, i.e. a figure which states how much the windows in the two images differ.

While having the advantage of being very easy to compute, the SSD has the major drawback of being influenced by the global intensity of an image. Indeed, if one image is taken in brightness and the second in darker conditions, then two matching windows will still yield a dissimilarity which can be rather large due to the DC intensity shift between the images. The NCC comparison function does not have this problem. It gives a similarity value defined as

$$S = \frac{\iint_W \left(J(\mathbf{T}(x,y)) - \overline{J}\right) . \left(I(x,y) - \overline{I}\right) v(x,y) dxdy}{\sqrt{\iint_W \left(J(\mathbf{T}(x,y)) - \overline{J}\right)^2 v(x,y) dxdy . \sqrt{\iint_W \left(I(x,y) - \overline{I}\right)^2 v(x,y) dxdy}}} \tag{2.4}$$

where $\overline{I} = \iint_W (I(x,y)) dxdy$ and $\overline{J} = \iint_W (J(\mathbf{T}(x,y))) dxdy$ are the mean intensities of the windows W and $\mathbf{T}(W)$ in the images. This time a similarity measure is computed which indicates how well the image windows correspond. The mean intensity of the windows is taken into account which makes this measure stable against global intensity changes.

Equations 2.2 and 2.4 show us the way to check if feature points in different images are possible matches by computing the SSD or NCC measure and checking if they are small or large

respectively. We are of course mostly interested in features that can be matched over multiple images but that are also as unique as possible. It does not help us to extract features that look like many other features in the image because these will have many possible matches in other images. So what we are looking for are features that can be differentiated from other features in the image. Kanade [51] e.a. proposed their *KLT* features and made the following mathematical derivation.

If we extract a feature in image I and match it to a feature in image J, we want the SSD value to be as low as possible, i.e. minimize Eq. 2.2. Let us assume that the transformation of window W between I and J can be modeled by a pure translation in x and y, then our SSD becomes

$$D(x,y) = \iint_W [J(x+\Delta_x, y+\Delta_y) - I(x,y)]^2 v(x,y) dxdy.$$

If we approximate $J(x+\Delta_x, y+\Delta_y)$ by its Taylor series up to the first order term

$$J(x+\Delta_x, y+\Delta_y) = J(x,y) + \begin{bmatrix} \Delta_x & \Delta_y \end{bmatrix} \begin{bmatrix} \frac{\partial J}{\partial x} \\ \frac{\partial J}{\partial y} \end{bmatrix}$$

then we obtain

$$D(x,y) = \iint_W \left[J(x,y) + \begin{bmatrix} \Delta_x & \Delta_y \end{bmatrix} \begin{bmatrix} \frac{\partial J}{\partial x} \\ \frac{\partial J}{\partial y} \end{bmatrix} - I(x,y) \right]^2 v(x,y) dxdy. \tag{2.5}$$

If we want to minimize this value, we have to differentiate equation 2.5 w.r.t. $\Delta = \begin{bmatrix} \Delta_x & \Delta_y \end{bmatrix}$ and equal to 0. This becomes

$$\begin{aligned} 0 &= \frac{\partial D(x,y)}{\partial \Delta} \\ &= \iint_W \left[\begin{bmatrix} \Delta_x & \Delta_y \end{bmatrix} \begin{bmatrix} \frac{\partial J}{\partial x} \\ \frac{\partial J}{\partial y} \end{bmatrix} \begin{bmatrix} \frac{\partial J}{\partial x} & \frac{\partial J}{\partial y} \end{bmatrix} \begin{bmatrix} \Delta_x \\ \Delta_y \end{bmatrix} + (J(x,y) - I(x,y)) \begin{bmatrix} \frac{\partial J}{\partial x} \\ \frac{\partial J}{\partial y} \end{bmatrix} \right] v(x,y) dxdy. \end{aligned}$$

In other words, to find Δ, we have to solve the linear system

$$\mathbf{Z}\Delta = e$$

with

$$\begin{aligned} \mathbf{Z} &= \iint_W \begin{bmatrix} \frac{\partial J}{\partial x} \\ \frac{\partial J}{\partial y} \end{bmatrix} \begin{bmatrix} \frac{\partial J}{\partial x} & \frac{\partial J}{\partial y} \end{bmatrix} v(x,y) dxdy \qquad (2.6) \\ &= \iint_W \begin{bmatrix} \left(\frac{\partial J}{\partial x}\right)^2 & (\frac{\partial J}{\partial x})(\frac{\partial J}{\partial y}) \\ (\frac{\partial J}{\partial x})(\frac{\partial J}{\partial y}) & \left(\frac{\partial J}{\partial y}\right)^2 \end{bmatrix} v(x,y) dxdy \\ e &= \iint_W \left[(I(x,y) - J(x,y)) \begin{bmatrix} \frac{\partial J}{\partial x} \\ \frac{\partial J}{\partial y} \end{bmatrix} \right] v(x,y) dxdy \end{aligned}$$

If we want to determine when a feature is well suited for tracking, we investigate equation 2.6. If we want to match the feature in another image, the matrix $\mathbf{Z}$ should be well conditioned to yield stable matching results. This is the case if certain criterions are fulfilled. For KLT features, Kanada et al. [51] impose that the smallest eigenvalue of $\mathbf{Z}$ should be larger than a threshold. Harris [23] looks at the trace of the matrix and defines the a *corner response* function

$$R = det(\mathbf{Z}) - k(trace(\mathbf{Z}))^2 \tag{2.7}$$

where Harris suggest the value $k = 0.04$. To extract only well conditioned features, local maxima of the corner response function are selected. Sub-pixel precision can be achieved by quadratic



Figure 2.3: Extracted Harris points on an image without forcing distribution over the image (left) and with distribution enabled (right)

approximation of the neighborhood of a local maximum. A typical choice for $v(x, y)$ in this case is a Gaussian with $\sigma = 0.7$.

Harris corners are features that are typically very well localized in the image. In order to match features between different images, a local neighborhood of 7x7 pixels or more (depending on the image size) is extracted around the features in both images and the SSD or NCC is computed between these windows. Fig. 2.3(left) shows an image with 1000 extracted Harris corners. One should be very careful with the selection of the corners. Typically the operator selects an upper limit N on the amount of features to extract. If only the N strongest corners are selected (based on the corner response function), one stands the risk of extracting corners in one part of the image only where the gradients are very strong. This can clearly be seen on the left in Fig. 2.3 where most of the corners are found in the trees. A better option is to select the M strongest corners with $M = 0.7N$. For the other 30%, the image is divided in different regions and the strongest features in each area are added to the list. The result of this algorithm is shown on the right in Fig. 2.3. The better distribution of the features is clearly visible.

2.1.3 Small- vs. Wide-Baseline Matching

The features extracted by KLT or Harris detectors and compared by computing NCC on windows around them are only invariant against a shift in x or y direction and global intensity changes. For images with viewpoints that are close together, this invariance will typically suffice to find many stable features and feature-matches along several views. This is the so-called *small-baseline* situation. For images that are taken further apart or during changing illumination conditions, these features will not be stable enough. In these *wide-baseline* conditions, other techniques are necessary for both feature detection and feature matching. It is beyond the scope of this text to give a complete overview of wide-baseline matching techniques. We will only highlight two well-known and well-used features: affine invariant features based on moment invariants and SIFT features.

Affine Invariant Features

Lately, there has been a lot of interest in affine invariant regions. If we assume that the 3D surface of the scene we are looking at is locally planar and that er is no perspective distortion, then local transformations of pixels from image to another are described by 2D affine transformations. Hence affine invariants offer very interesting features for matching in wide-baseline situations. The invariant feature extractor and descriptor we will explain here exploits the edges present in the image. The rationale behind this is that edges are typically rather stable features, that can be

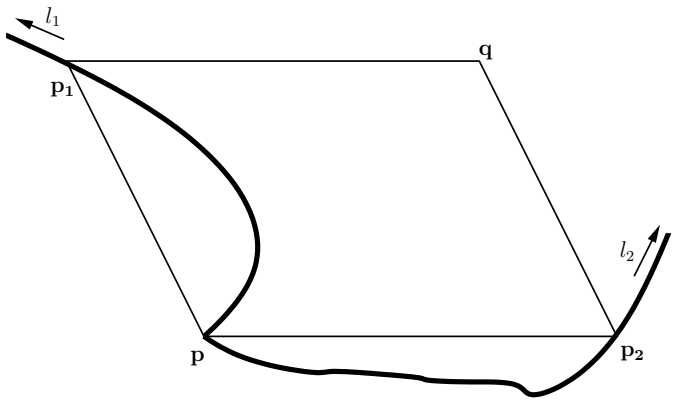


Figure 2.4: Invariant features are determined based on the edges in the neighborhood of a corner point.

detected over a range of viewpoints, scales and/or illumination changes.

In order to compute the affine transformation between two patches, three points are necessary. In practice, we start from a Harris corner point $\mathbf{p}$ and a nearby edge, extracted with the Canny edge detector. To increase the robustness to scale changes, these basic features are extracted at multiple scales. Two points $\mathbf{p_1}$ and $\mathbf{p_2}$ move away from the corner in both directions along the edge, as shown in Fig. 2.4. For curved edges it is possible to uniquely relate a point on one edge with a point on the other edge using an affine invariant parametrization which links l_1 to l_2. This leaves us with one degree of freedom: the size of the parallelogram-shaped region spanned by $\mathbf{p}, \mathbf{p_1}, \mathbf{p_2}$. For all possible parallelogram sizes, we evaluate a function based on photometric quantities of the texture. For instance one of the two functions Inv_1 or Inv_2 can be be used.

$$Inv_1 = abs\left(\frac{|\mathbf{p_1}-\mathbf{p_g}\ \mathbf{p_2}-\mathbf{p_g}|}{|\mathbf{p}-\mathbf{p_1}\ \mathbf{p}-\mathbf{p_2}|}\right)\frac{M^1_{00}}{\sqrt{M^2_{00}M^0_{00}-(M^1_{00})^2}} \tag{2.8}$$

$$Inv_2 = abs\left(\frac{|\mathbf{p}-\mathbf{p_g}\ \mathbf{q}-\mathbf{p_g}|}{|\mathbf{p}-\mathbf{p_1}\ \mathbf{p}-\mathbf{p_2}|}\right)\frac{M^1_{00}}{\sqrt{M^2_{00}M^0_{00}-(M^1_{00})^2}} \tag{2.9}$$

$$\begin{aligned} &with \\ M^n_{pq} &= \int_\Omega I^n(x,y)x^p y^q dxdy \\ \mathbf{p_g} &= \left(\frac{M^1_{10}}{M^1_{00}}, \frac{M^1_{01}}{M^1_{00}}\right) \end{aligned}$$

The parallelogram for which the function reaches an extremum is chosen as the affine invariant feature for the location $\mathbf{p}$. The second factor in the formula for Inv_1 or Inv_2 is computed from image moments. This factor is added to the formula to ensure invariance under an intensity offset. In case of straight edges, the method described above cannot be applied since we cannot uniquely relate $\mathbf{p_1}$ to $\mathbf{p_2}$. Since intersections of two straight edges occur quite often we cannot simply neglect this case. To circumvent this problem, the two functions of Eq. 2.8 and 2.9 are combined and locations where both functions reach an extremum are selected.

Locating the affinely invariant feature is only one part. We also have to assign a feature vector to it. As in the region extraction step, we consider invariance both under affine geometric changes and linear photometric changes with different offsets and different scale factors for each of the three color bands. Each region is characterized by a feature vector of moment invariants, e.g. the *generalized color moments* [39]. These are based on powers of image coordinates and intensities of different color channels:

$$M^{abc}_{pq} = \int\!\!\int_\Omega x^p y^q (R(x,y))^a (G(x,y))^b (B(x,y)^c) dxdy$$

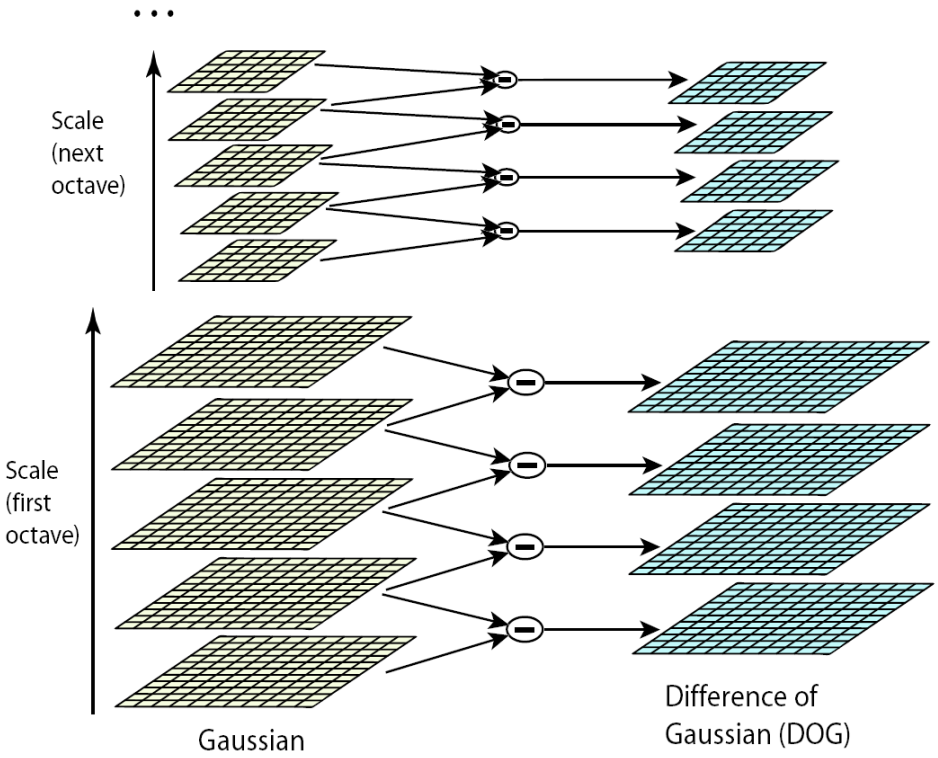


Figure 2.5: Features are extracted at extrema of a Difference of Gaussian function. Gaussians with constantly increasing σ are computed and subtracted for the first octave. The procedure is repeated for the second octave on subsampled images to improve efficiency.

For every feature a feature vector is constructed, consisting of a set of combinations of these generalized color moments. Matching features between two images boils down to comparison of the feature vectors, a process which can be performed quite efficiently by use of hashing techniques.

SIFT Features

Recently Lowe [36]. proposed SIFT features to be used mainly in object recognition tasks but also for other computer vision techniques that are in need of simple and stable features. The SIFT acronym stands for *Scale Invariant Feature Transform*. SIFT features are scale and rotation invariant and are up to some degree robust (though not mathematically invariant) against affine transformations, change in 3D viewpoint and illumination. Since their introduction SIFT features have become very popular mainly because of the existence of fast algorithms to detect and match them.

The detection and description of SIFT features follows the following steps:

- Localization in the image as the extrema of a difference-of-Gaussian function
- Orientation assessment based on image gradient directions in a local image window
- Keypoint descriptor computation based on image gradients

Localization Feature points are located in the image by searching for extrema of a difference-of-Gaussian function of the original image. The image is convolved with a set of gaussians, each of them smoothing the image with a certain standard deviation σ. The σ of two neighboring gaussians is separated by a constant multiplicative factor. The difference of all neighboring gaussians is

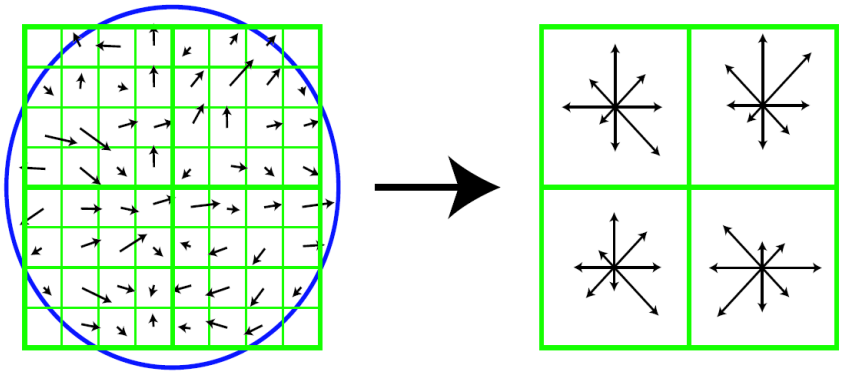


Figure 2.6: Computation of the SIFT descriptor is done by looking at the orientation of the gradients in the neighborhood of the feature location (left). Histograms of these orientations with 8 bins each are created. The right figure shows an 2x2 array of such histograms where the length of the orientation vector denotes the magnitude of that bin. In practice a 4x4 array of 8 bins is used, yielding a feature vector of 128 elements.

computed (the so called difference of gaussian function) and extrema are detected. These extrema typically correspond to stable feature points. As shown in Fig. 2.5 in Lowe's implementation, the DoG is computed for the first octave, then the image is subsampled by a factor 2 and the same gaussians are computed on this subsampled image, etc.

Orientation By assigning a consistent orientation to each keypoint based on local image properties, the keypoint descriptor can be represented relative to this orientation and therefore achieve invariance to image rotation. The scale of the keypoint is used to select the Gaussian smoothed image with the closest scale, so that all computations are performed in a scale-invariant manner. For each image sample at this scale, the gradient magnitude and orientation, is pre-computed using pixel differences. An orientation histogram is formed from the gradient orientations of sample points within a region around the keypoint. The orientation histogram has 36 bins covering the 360 degree range of orientations. Each sample added to the histogram is weighted by its gradient magnitude and by a Gaussian-weighted circular window with a σ that is 1.5 times that of the scale of the keypoint. Peaks in the orientation histogram correspond to dominant directions of local gradients. The highest peak in the histogram is detected. Finally, a parabola is fit to the 3 histogram values closest to the peak to interpolate the peak position for better accuracy.

Descriptor The SIFT descriptor is a 128 dimensional vector computed from the spatial distribution of image gradients over a circular region. It is computed as follows. First the image gradient magnitudes and orientations are sampled around the keypoint location, using the scale of the keypoint to select the level of Gaussian blur for the image. In order to achieve orientation invariance, the coordinates of the descriptor and the gradient orientations are rotated relative to the keypoint orientation. These are illustrated with small arrows at each sample location on the left side of Fig. 2.6. To make the descriptor stable against small localization errors of the feature, the magnitudes are weighted with a Gaussian with σ equal to one half the width of the descriptor window. In this way gradients that are far from the center have a smaller impact on the resulting gradient vector. The keypoint descriptor is shown on the right side of Fig. 2.6. The figure shows eight directions for each orientation histogram, with the length of each arrow corresponding to the magnitude of that histogram entry.

The descriptor is formed from a vector containing the values of all the orientation histogram entries, corresponding to the lengths of the arrows on the right side of Fig. 2.6. The figure shows a 2x2 array of orientation histograms. Typically a 4x4 array of histograms with 8 orientation bins in each is used. Therefor the feature vector has a length of 4x4x8 = 128. Finally, the feature

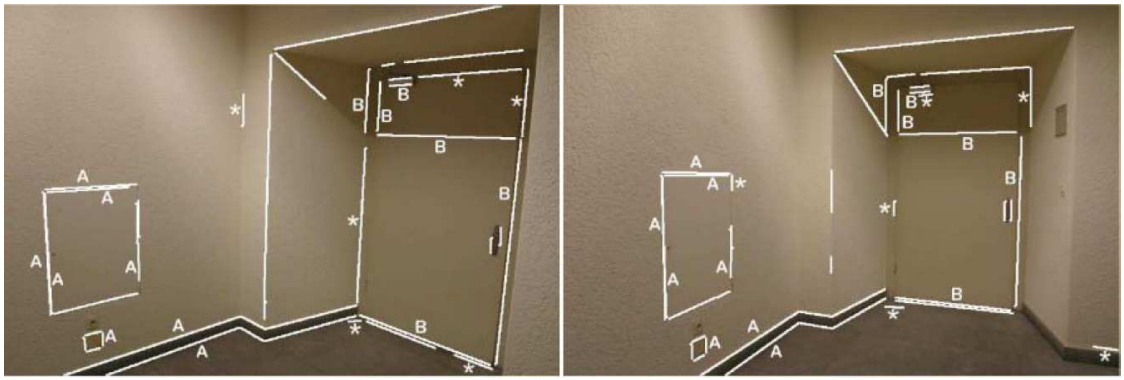


Figure 2.7: Two images of a man-made environment. Finding stable feature-points is very difficult because of the large homogeneous regions. Lines can be extracted and, even though their endpoints are typically not well located, intersections of coplanar can act as stable feature points.

vector is modified to reduce the effects of illumination change. First, the vector is normalized to unit length. A change in image contrast in which each pixel value is multiplied by a constant will multiply gradients by the same constant, so this contrast change will be canceled by vector normalization. A brightness change in which a constant is added to each image pixel will not affect the gradient values, as they are computed from pixel differences. Therefore, the descriptor is invariant to affine changes in illumination. However, non-linear illumination changes can also occur due to camera saturation or due to illumination changes that affect 3D surfaces with differing orientations by different amounts. These effects can cause a large change in relative magnitudes for some gradients, but are less likely to affect the gradient orientations. Therefore, we reduce the influence of large gradient magnitudes by thresholding the values in the unit feature vector to each be no larger than 0.2, and then re-normalizing to unit length. This means that matching the magnitudes for large gradients is no longer as important, and that the distribution of orientations has greater emphasis.

2.1.4 Matching Lines

Although the feature-vectors typically take into account the information of a neighborhood, so far we have only described features that are located at a specified 2D position, i.e. feature-*points*. These are the most popular and well-known features, mostly because geometric relations between points are easily described as we will see in paragraphs and chapters to follow and hence matches between points can immediately be used to extract multiple-view geometry.

Points, however, are not the only features that can be extracted from images and matched between them. One could, for instance, think about detecting and matching lines. Lines have the important advantage that they appear often in structured, man-made environments. Consider Fig. 2.7. This is a typical example of a man-made scene. One would certainly have a hard time detecting and matching feature points here because of the very homogeneous regions which make it very hard to distinguish points. In this scene, however, it is certainly possible to detect line features. Bay e.a. [6] describe a procedure to find the lines, match them between images and use them to obtain feature points.

First an edge detector is applied and a line-fitting algorithm is executed on these edges to yield lines. For every line, a feature vector is extracted consisting of the bins of a color histogram of the pixels in a neighborhood around the line. Using these feature vectors, lines can be matched between different images. As one can expect from the images in Fig. 2.7, the feature vectors of different lines can be quite similar, so this matching step is far from trivial. Topology constraints are used to rule out inconsistent matches.

The problem of using lines for multiple view geometry computations is that the end-points of

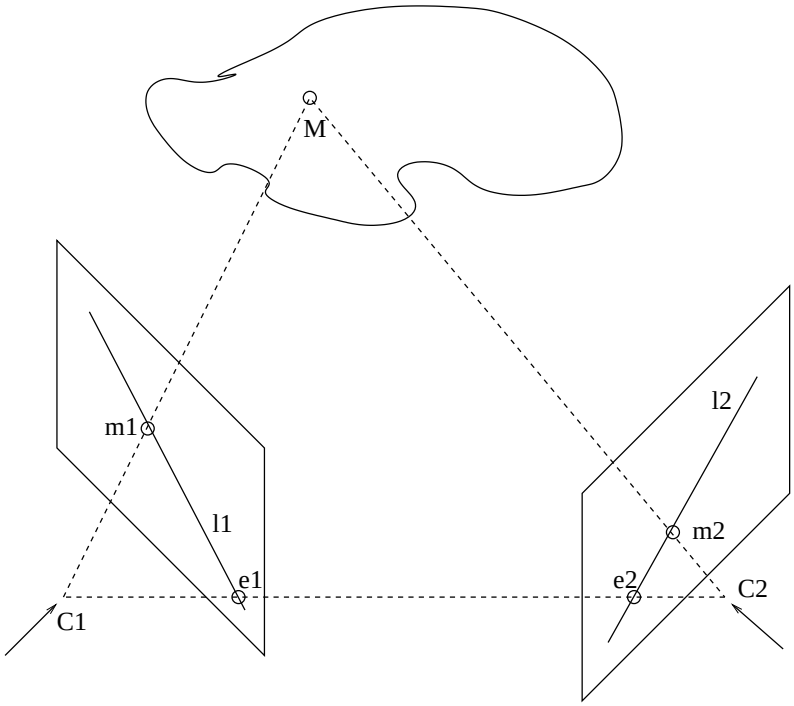


Figure 2.8:

lines are typically not well defined, i.e. if the same line is extracted in two images, the end-points of both lines are typically not the same. However, if we can find out which lines are coplanar, then virtual feature points can be defined at the intersection of lines. Even if the end-points of the line segments are not in correspondence, the intersection points are stable and very well located. These intersection points can act as virtual matches feature points between the two images.

2.2 The Geometry of Two Images

If we want to search for correspondences between two images, there is an underlying geometric structure that can be employed, called **epipolar geometry**.

2.2.1 Epipolar Geometry

Consider again the simple stereo setup we described in 1.7. We explained there that in order to reconstruct points in 3D we need the calibration of the two cameras and correspondencing image points. If a point is given in one image and we want to find the corresponding point in the other image, where do we have to look for this point? It turns out that one does not have to search the entire second image but only on a specific line in this image. Fig. 2.8 explains this. It shows two images, taken by cameras with centers at $\mathtt{C}_1$ and $\mathtt{C}_2$. The 3D point $\mathtt{M}$ is projected onto the image points $\mathtt{m}_1$ and $\mathtt{m}_2$. Let us assume we have found a feature at $\mathtt{m}_1$ and we want to search for the corresponding feature in the other image. We know that the point $\mathtt{m}_1$ must come from a 3D point that is located on the line in space defined by $\mathtt{C}_1$ and $\mathtt{m}_1$, the so-called *backprojected* ray because all points on this line are projected onto the image point $\mathtt{m}_1$. This means that the corresponding point in the second image must lie on the projection of this ray in the second image, called $\mathtt{l}_2$. We call this line the *epipolar* line of $\mathtt{m}_1$. This means that, if the calibration of the cameras is given, we can compute the epipolar line for any point $\mathtt{m}_1$ and restrict the search for the corresponding point to a search along this line, a very strong constraint indeed because it eliminates an entire degree of freedom!

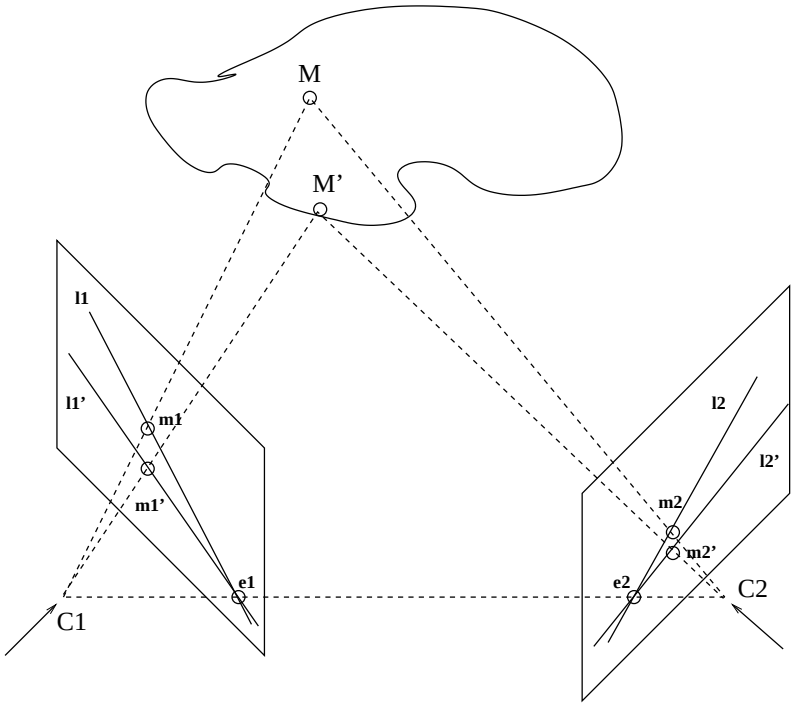


Figure 2.9:

There is more to the concept of epipolar geometry than what has been described before. Inspect Fig. 2.8 again. In fact every 3D point $\mathtt{M}$ defines a plane together with the two camera centra $\mathtt{C}_1$ and $\mathtt{C}_2$. The two epipolar lines $\mathtt{l}_1$ and $\mathtt{l}_2$ are found as the intersection of this plane with the images. Any other 3D point in this plane is hence projected onto these two epipolar lines. In other words, all points on $\mathtt{l}_1$ have a corresponding point on $\mathtt{l}_2$ (provided this correspondences exists, which it does not in the case of occlusion). Any 3D point $\mathtt{M}'$ outside this plane, together with $\mathtt{C}_1$ and $\mathtt{C}_2$ defines another plane, yielding another pair of epipolar lines as shown in Fig. 2.9. The set of all these planes forms a pencil around the line between $\mathtt{C}_1$ and $\mathtt{C}_2$ (the baseline). We call the intersection of this line with the images ($\mathtt{e}_1$ and $\mathtt{e}_2$ respectively) the epipoles. An epipole is in fact the projection of the other camera center in an image. Since the baseline line is part of all the planes, the intersection of this line with the images (the epipole) lies on all epipolar lines. This means that all epipolar lines in the first image coincide in $\mathtt{e}_1$ and all epipolar lines in the second image coincide in $\mathtt{e}_2$ as shown in Fig. 2.9.

2.2.2 The Fundamental Matrix

The fact that all points on epipolar line $\mathtt{l}_1$ have corresponding points in the other image that have to lie on the corresponding epipolar line $\mathtt{l}_2$ can be nicely expressed using the *fundamental matrix*. This matrix is derived as follows.

A point $\mathtt{m}_1$ in image 1 lies on the epipolar line $\mathtt{l}_1$. Since all epipolar lines go through the epipole $\mathtt{e}_1$, the line $\mathtt{l}_1$ can be defined by $\mathtt{m}_1$ and $\mathtt{e}_1$, using equation 1.1:

$$\mathtt{l}_1 = [\mathtt{e}_1]_\times \mathtt{m}_1 \tag{2.10}$$

Let us define the projection matrices of the 2 images as $\mathbf{P_1}$ and $\mathbf{P_2}$, and $\mathtt{M}$ as the 3D point that is projected using these matrices onto $\mathtt{m}_1$ and $\mathtt{m}_2$ respectively

The fact that $\mathtt{m}_1$ lies on $\mathtt{l}_1$ is expressed as

$$\mathtt{l}_1{}^T \mathtt{m}_1 = 0$$

Since $\mathtt{m}_1$ is the projection of $\mathtt{M}$ in image 1,

$$\mathtt{m}_1 \simeq \mathbf{P_1}\mathtt{M}$$

And thus

$$\begin{aligned} {\mathtt{l}_1}^T\mathbf{P_1}\mathtt{M} &= 0 \\ \Pi^T\mathtt{M} &= 0 \end{aligned} \tag{2.11}$$

with

$$\Pi \simeq {\mathbf{P_1}}^T\mathtt{l}_1$$

then we see that Eq. 2.11 actually defines the plane Π which is the backprojected plane from $\mathbf{P_1}$ through $\mathtt{l}_1$. The same can be written for the second image, thus

$$\begin{aligned} \Pi &\simeq {\mathbf{P_1}}^T\mathtt{l}_1 \\ \Pi &\simeq {\mathbf{P_2}}^T\mathtt{l}_2 \end{aligned}$$

So, using the pseudo-inverse of ${\mathbf{P_2}}^T$ denoted with $\dagger$, we can write

$$\mathtt{l}_2 \simeq ({\mathbf{P_2}}^T)^\dagger{\mathbf{P_1}}^T\mathtt{l}_1$$

or, using Eq. 2.10

$$\begin{aligned} \mathtt{l}_2 &\simeq ({\mathbf{P_2}}^T)^\dagger{\mathbf{P_1}}^T[\mathtt{e}_1]_\times\mathtt{m}_1 \\ \mathtt{l}_2 &\simeq \mathbf{F}\mathtt{m}_1 \end{aligned} \tag{2.12}$$

with

$$\mathbf{F} \simeq ({\mathbf{P_2}}^T)^\dagger{\mathbf{P_1}}^T[\mathtt{e}_1]_\times \tag{2.13}$$

The fact that the point $\mathtt{m}_2$ should lie on $\mathtt{l}_2$ is expressed as

$${\mathtt{l}_2}^T\mathtt{m}_2 = 0$$

or transposed

$${\mathtt{m}_2}^T\mathtt{l}_2 = 0$$

With Eq. 2.12 this becomes

$${\mathtt{m}_2}^T\mathbf{F}\mathtt{m}_1 = 0 \tag{2.14}$$

Equation 2.14 is the well-known expression of the relationship between two corresponding points $\mathtt{m}_1$ and $\mathtt{m}_2$ via the *fundamental matrix* $\mathbf{F}$. Given this 3x3 matrix and either $\mathtt{m}_1$ or $\mathtt{m}_2$ the corresponding epipolar line in the other image is easily computed as

$$\begin{aligned} \mathtt{l}_2 &\simeq \mathbf{F}\mathtt{m}_1 \\ \mathtt{l}_1 &\simeq \mathbf{F}^T\mathtt{m}_2 \end{aligned}$$

Since $\mathbf{F}$ was defined as $({\mathbf{P_2}}^T)^\dagger{\mathbf{P_1}}^T[\mathtt{e}_1]_\times$ and $[\mathbf{a}]_\times\mathbf{a} = 0$ for all vectors $\mathbf{a}$, this means that

$$\mathbf{F}\mathtt{e}_1 = 0 \tag{2.15}$$

This can only hold if the fundamental matrix is rank-deficient, i.e. has a maximal rank of 2. Equation 2.15 defines the epipole $\mathtt{e}_1$ as the right nullvector of $\mathbf{F}$ and we see that all epipolar lines `l1` must go through the epipole since

$$\mathtt{m}_1\mathbf{F}\mathtt{e}_1 = 0$$

for all points $\mathtt{m}_1$. Analogously, the epipole $\mathtt{e}_2$ in image 2 is the left nullvector of $\mathbf{F}$.

2.2.3 Computation of the Fundamental Matrix

The fundamental matrix is a key concept in 3D reconstruction algorithms. Knowledge of the fundamental matrix enables one for instance to limit the search for correspondences to a 1D search along the epipolar line. In this paragraph we will explain how the fundamental matrix can be computed.

Linear Computation

It is clear from its definition in Eq. 2.13 that $\mathbf{F}$ can be computed if the projection matrices of the two cameras are known. If this calibration is not known, one has to compute the fundamental matrix from point correspondences. Indeed, Eq. 2.14 shows that every corresponding point pair $(\mathtt{m}_1, \mathtt{m}_2)$ delivers one constraint on the 9 elements of the 3x3 matrix $\mathbf{F}$. Since Eq. 2.14 a homogeneous equation, $\mathbf{F}$ only needs to be computed up to scale, so only 8 constraints are necessary.

Let us assume we have 8 corresponding point pairs to our disposal. We define the vector $\mathbf{f}$ as

$$\mathbf{f} = [F_{11}F_{12}F_{13}F_{21}F_{22}F_{23}F_{31}F_{32}F_{33}]^T$$

which holds the elements of $\mathbf{F}$. If $\mathtt{m}_1 = [x_1 y_1 w_1]^T$ and $\mathtt{m}_2 = [x_2 y_2 w_2]^T$ then Eq. 2.14 boils down to

$$[x_2x_1\ x_2y_1\ x_2w_1\ y_2x_1\ y_2y_1\ y_2w_1\ w_2x_1\ w_2y_1\ w_2w_1]\,\mathbf{f} = 0 \tag{2.16}$$

Every correspondence gives us one equation like Eq. 2.16. We can now stack all 8 equations and thus form the matrix $\mathbf{A}$ where

$$\mathbf{A}\,\mathbf{f} = 0 \tag{2.17}$$

This system of equations is easily solved by Singular Value Decomposition (SVD) [19]. Applying SVD to $\mathbf{A}$ yields the decomposition $\mathbf{USV^T}$ with $\mathbf{U}$ and $\mathbf{V}$ orthonormal matrices and $\mathbf{S}$ a diagonal matrix containing the singular values. These singular values are positive and in decreasing order. Therefore in our case $\sigma_9$is guaranteed to be identically zero (8 equations for 9 unknowns) and thus the last column of $\mathbf{V}$ is the correct solution for $\mathbf{f}$ (at least as long as the eight equations are linearly independent, which is equivalent to all other singular values being non-zero). The matrix we obtain from $\mathbf{f}$ in this way however will, in the presence of noise, not satisfy the rank-2 constraint. This means that there will not be real epipoles through which all epipolar lines pass. A solution to this problem is to obtain as the closest rank-2 approximation of the solution coming out of the linear equations.

Non-Linear Computation

The 8-point algorithm described before is linear and hence very popular. We know however that $\mathbf{F}$ only holds 7 degrees of freedom, so 7 correspondences should suffice to compute it. Since the extra constraint consists of enforcing that the rank of $\mathbf{F}$ should be 2, which is a non-linear constraint, the computation ultimately boils down to solving a non-linear equation.

A similar approach as in the previous section can be followed to characterize the right null-space of the system of linear equations originating from the seven point correspondences. Since we are one constraint short, the solution $\mathbf{f}$ must be a linear combination of the last two eigenvectors of $\mathbf{V}$.

$$\mathbf{f} = \mathbf{v_8} + \lambda\mathbf{v_9}$$

This means the fundamental matrix can be written as

$$\mathbf{F} = \mathbf{F_8} + \lambda\mathbf{F_9}$$

with $\mathbf{F_8}$ and $\mathbf{F_9}$ the matrices corresponding to $\mathbf{f_8}$ and $\mathbf{f_9}$ respectively. We now have to search for the unknown λ which makes the rank of the matrix 2. The rank of $\mathbf{F}$ is only 2 if its determinant is 0. Hence

$$\det(\mathbf{F_8} + \lambda\mathbf{F_9}) = a_3\lambda^3 + a_2\lambda^2 + a_1\lambda + a_0 = 0$$

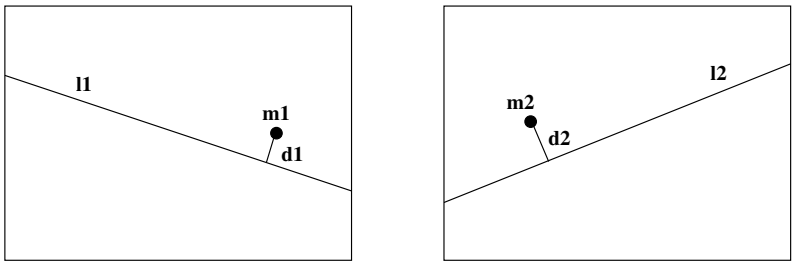


Figure 2.10: The geometric error we want to minimize is the sum of the distances d_1 and d_2 of the points to the respective epipolar line.

which is a polynomial of degree 3 in λ . This can simply be solved analytically. There are always 1 or 3 real solutions. The special case where $\mathbf{F} = \mathbf{F_9}$ (which is not covered by this parametrization) is easily checked separately, i.e. $\mathbf{F_9}$ should have rank-2. If more than one solution is obtained then one extra correspondence is needed to obtain the true fundamental matrix.

More Matches

We can of course extract more than the minimum needed amount of matches. We can make use of all these matches to compute the fundamental matrix. The easiest way to do so is to write down the equation for each match and stack all these equations in the matrix $\mathbf{A}$ of Eq. 2.17. The fundamental matrix, obtained by solving the SVD of this $\mathbf{A}$ is the best solution in a least-squares way. If many matches are available, the amount of rows of $\mathbf{A}$ can become quite large which makes computation of the SVD an expensive task. However, if $\mathbf{Af} = 0$ then $\mathbf{A}^T\mathbf{Af} = 0$ as well. Hence, by taking the SVD of the matrix $\mathbf{A}^T\mathbf{A}$ which always has 9x9 elements, we can also solve for $\mathbf{F}$.

The scheme described above gives equal weight to every match. Also, we actually solve for $\mathbf{F}$ by searching for the solution which minimizes an algebraic error (the least squares error of the algebraic equation 2.14). The error we actually want to minimize is a geometric error, expressing the distance of the points to the corresponding epipolar lines, d_1 and d_2 in Fig. 2.10. In homogeneous coordinates the distance between a point `m` and a line `l` is given by

$$d = \frac{(m_x l_x + m_y l_y + m_w l_w)^2}{l_x^2 + l_y^2 + l_w^2} \tag{2.18}$$

A possible way of minimizing this error is by employing Reweighed least-squares. This technique solves for $\mathbf{F}$ in an iterative way. The SVD of the matrix $\mathbf{A}$ gives us a first solution for $\mathbf{F}$. Then, we compute the residual distance for each match as shown in Fig. 2.10. Every row of $\mathbf{A}$ is then divided by the residual of its respective point match. We then solve again for $\mathbf{F}$ by computing the SVD of the reweighed $\mathbf{A}$. After a couple of iterations this algorithm converges onto a better estimate of $\mathbf{F}$, minimizing the geometric instead of the algebraic error.

Another possibility to take into account multiple matches for the computation of $\mathbf{F}$ is a non-linear optimization. The linear SVD solution can be used as an initialization to a Levenberg-Marquardt algorithm or another non-linear optimization scheme which explicitly minimizes the geometric error.

Conditioning

If we compute the fundamental matrix by solving an SVD containing rows like given in Eq. 2.16, we will quickly run into problems. The result of an SVD is the best solution to the system of equations in a least squares aspect. This only works stably if the different columns of the matrix $\mathbf{A}$ stacking the equations have approximately the same magnitude. Typically we deal with images of 1000x1000 pixels or more. This means the coordinates of the extracted features are in the order

of (1000, 1000, 1). The third, homogeneous, coordinate w is clearly much smaller than the other two. This means the different columns of the matrix $\mathbf{A}$ will have the following order of magnitude.

$$\begin{bmatrix} x_2x_1 & x_2y_1 & x_2w_1 & y_2x_1 & y_2y_1 & y_2w_1 & w_2x_1 & w_2y_1 & w_2w_1 \\ 1e6 & 1e6 & 1e3 & 1e6 & 1e6 & 1e3 & 1e3 & 1e3 & 1 \end{bmatrix}$$

It is clear that an error on the ninth element in the solution vector will have a much smaller impact on the total least squares residual of the system. This means the solution will be biased and hence unusable. A way around this problem is make sure the data is *conditioned* before use. Optimal conditioning algorithms can be found in literature but in practice a simple scaling of the 2D coordinates of the image points to the range [-1,1] yields satisfactory results. The input data is transformed by a matrix $\mathbf{T}$, according to the image width (w) and height (h):

$$\begin{aligned} c_x &= \frac{w}{2} \\ c_y &= \frac{h}{2} \\ s &= \frac{1}{c_x + c_y} \\ \mathbf{T} &= \begin{bmatrix} s & 0 & -sc_x \\ 0 & s & -sc_y \\ 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{2.19}$$

The transformed points $\mathtt{m}' \simeq \mathbf{T}\mathtt{m}$ are filled into Eq. 2.17 where the matrix $\mathbf{A}$ is now well conditioned (all columns have the same order of magnitude). The resulting fundamental matrix $\mathbf{F}'$ is then transformed into real image space

$$\mathbf{F} \simeq \mathbf{T_2}^{-T}\mathbf{F}'\mathbf{T_1}^{-1}$$

2.2.4 Dealing with Outliers: the RANSAC Algorithm

Linear and non-linear estimation of the fundamental matrix is only possible if all input data is reliable. If only one incorrect match is present in the set of equations building $\mathbf{A}$ of Eq. 2.17, the computed fundamental matrix will be biased by this incorrect match and unusable. These wrong matches are called *outliers*. Since correspondences are computed by the computer based on NCC, SSD or comparison of feature vectors, the probability of mistakes is high and the presence of outliers can not be ignored. Hence, we need a robust way to deal with these outliers. There exists a well-known algorithm to do just that: given a data-set polluted by outliers, this algorithm is capable of computing the underlying structure and at the same time identifies the inliers and outliers in the data-set. The algorithm is called *RANSAC* which stands for RAndom SAmpling Consensus and was proposed by Fischler and Bolles [17] in 1981.

The RANSAC algorithm is in essence an iterative scheme. It is based on the assumption that a certain percentage of the input data-set consists of consistent inliers and that the outliers are just random mistakes in the data-set. In every iteration a set of samples is taken out of the data-set randomly. The amount of samples that is selected is equal to the minimum number we need to compute the underlying structure. In the case of the fundamental matrix, this is 8 if we use the linear algorithm or 7 if we employ the non-linear algorithm. Every time the structure is computed using this random set of samples and the support amongst the rest of the data-set is computed. In other words, in every iteration we assume that all 7 or 8 selected samples are inliers, and hence the computed structure is correct. We determine which of the remaining matches in the data-set are inliers for this structure and which are outliers. This is done by checking if matching points are on each other's epipolar line. A few pixels error can be tolerated because of influence of noise.

In reality there will be many iterations in which the set of samples contains at least one outlier and thus for which the computed structure is incorrect. However, the support for such an incorrect structure will be smaller than for the true solution. The iterative procedure is repeated until a

Step 1.	Extract features in the images
Step 2.	Compute a set of possible matches (with NCC, SSD or matching feature-vectors)
Step 3.	While (stop-criterion has not been reached)
	Step 3.1 Select 7 random matches out of the set of possible matches
	Step 3.2 Compute **F** non-linearly using these 7 matches
	Step 3.3 For each of the 1 or 3 solutions, compute the in- and outliers (residual distance $<$ threshold)
Step 4.	Refine **F** on all inliers

Table 2.1: F-RANSAC algorithm

ι	95%	90%	80%	70%	60%	50%	40%	30%	20%
n	3	5	13	35	106	382	1827	13692	233963

Table 2.2: Number of tries n we need to select 7 matches from the data-set to be 95% certain that we have selected at least 1 set of 7 inliers, depending on the percentage of inliers in the data-set ι.

stop-criterion has been met (see later). The solution with the largest support (i.e. the largest number of inliers) is selected as the true solution.

Let us make this more concrete by applying it to the computation of the fundamental matrix. Table 2.2.4 explains the algorithm.

Stop-Criterion

An important question when using RANSAC is when to stop the iterative process. We want to be reasonable sure that we have explored all our options without having to compute **F** from all possible sets of 7 points which is clearly combinatorially impossible. As explained in the previous paragraph, we only need to select one set of matches in which not a single outlier is present. If this is guaranteed to occur, we also know for sure the **F** matrix will be computed and identified correctly from its support amongst the other matches. The amount of tries is thus clearly dependent on the percentage of inliers in the data-set (ι). If we select p matches from the data-set, the chance that we have selected p correct matches is ι^p. If we perform this process n times, the chance that we have selected only good matches becomes

$$P = 1 - (1 - \iota^p)^n$$

Suppose we want to be 95% percent certain that we have at least one set of p inliers, then the number of tries n (depending on ι) is given by Eq. 2.20 where P stands for the certainty we want to achieve (e.g. 95%). Some values are shown in table 2.2.4.

$$n = \frac{1 - P}{1 - \iota^p} \tag{2.20}$$



Figure 2.11: For small-baseline cases, the original search for matches is limited to a neighborhood in the second image around the coordinates of the point in the first image.

Preemptive RANSAC

The RANSAC algorithm is very widely known and the most commonly used tool to deal with data-sets with a significant amount of outliers. Since its introduction in 1981, several improvements have been suggested to make it more stable and faster. A possible extension is *preemptive* RANSAC. The most time-consuming part per iteration of the RANSAC algorithm is typically not the computation of the structure from the minimal sample set but Step 3.3: the determination of the in- and outliers. For F-RANSAC, for instance, the distance between a point and an epipolar line has to be computed twice for every correspondence in the data set (once for image 1 to image 2 and once vice versa). This is a very expensive operation. However, if we can quickly determine that a certain solution is computed from a contaminated sample set and hence is bogus, it doesn't make sense to compute all in- and outliers because we will not use this solution anyway. The probability distribution of the matches in the data-set is described by a binomial distribution with the inlying fraction ι as the probability. Suppose that we have already checked y matches from the data-set. The expected amount of inliers is of course $y\iota$. The standard deviation of the distribution is given by

$$\sigma^2 = y\iota(1-\iota)$$

We define m as the expected amount of inliers minus a 'safety region' of 2σ.

$$\begin{aligned} m &= y\iota - 2\sigma \\ &= y\iota - 2\sqrt{y\iota(1-\iota)} \end{aligned} \tag{2.21}$$

If we have fewer than m inliers, the probability that this solution is not good is about 97% and hence it can be discarded. For example, if we assume a (low) inlying fraction of 20% and after having checked 16 matches for support we still haven't found a single inlier, the probability that this solution is wrong is 97.18% so we can stop checking. This saves a lot of processing time which would otherwise have been spent on checking the rest of the data-set. In order to apply this preemptive approach, checking for support needs to be done on random matches in the data-set because otherwise the solution will be biased towards the matches that appear first.

Selecting a good a-priori value for ι is not always an easy task. However, it is possible to adapt this value during computation. Therefore, we typically start with a low inlying fraction (e.g. 20%). If, during RANSAC processing, we detect a solution with a support of $\kappa > \iota$, we immediately know for certain that the inlying fraction is at least κ. We can thus recompute the number of tries we need and also the minimum amount of inliers given by Eq. 2.21 will be higher, thus wrong solutions are detected sooner.

2.2.5 F-Guided Matching

If we use simple SSD or NCC to compare feature points between (small-baseline) images, a point in one image might have several resembling points in the other image. In order to limit the amount

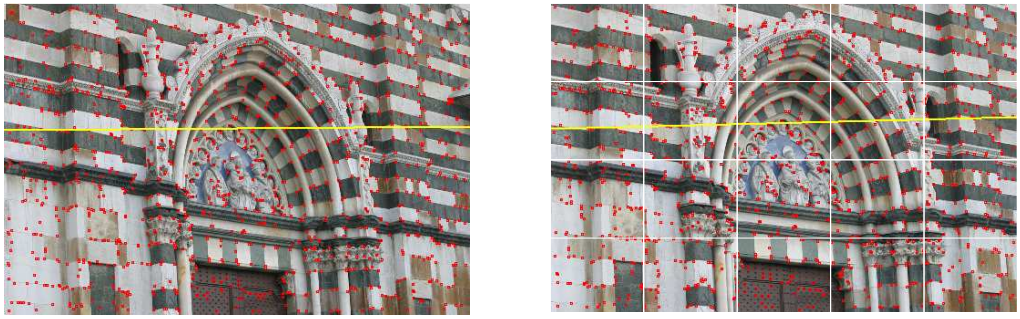


Figure 2.12: F-Guided corner matching. In order to find the corresponding point in the second image, we only have to search along the epipolar line. If we divide this image in tiles, only the features inside the tiles that are crossed by the line have to be checked.

of mismatches for the initial F-RANSAC step, we limit our search to a window in the other image centered around the same coordinates as the point in the first image as shown in figure 2.11. This limitation of course only works if small-baseline conditions apply since then feature points don't move too much between images. However, points in the foreground of the scene can still experience quite large coordinate changes, so that these will not be picked up by the original matching step. Also, because NCC or SSD are not so discriminative, features inside this region might match (slightly) better than the real match. These outliers are removed by the F-RANSAC step.

Now we have computed the epipolar geometry, we can search for more matches. Indeed, the localization limitation can be lifted since we have an even stronger constraint: matching features points must lie on each other's epipolar line! Thus, for each feature in the first image, we search along the epipolar line in the second image for matching points, evaluated by NCC or SSD. By lowering the NCC or SSD threshold for matching and allowing for a epipolar distance of about 3 pixels (depending on the image size of course), this *F-guided matching* step delivers many extra matches.

A naive approach to F-guided matching would be to select a feature in the first image, go over all features in the second image, check if this feature is on the epipolar line and then compute the NCC value. For efficiency reasons, it is better to divide the second into tiles of say 50x50 pixels. Features in the second image are assigned to the tile they are located in. If we then want to search for a match for a feature in the first image, we compute the epipolar line in the second image and determine the tiles this line crosses. Only features in these tiles are possible candidates for matching. Fig. 2.12 illustrates the procedure.

2.2.6 Computing Homographies with RANSAC

RANSAC is a generic tool, not only to be used for computing F-matrices. Another typical example is the computation of an homography matrix H between two images. Suppose we are dealing with a planar scene, then Eq. 1.8 gives the mathematical formulation of the relation between two matching points described by the homography or projective transformation matrix $\mathbf{H}$. We want to compute the matrix $\mathbf{H}$ such that Eq. 1.8 is valid for all matching points. Note knowledge of $\mathbf{H}$ gives a much stronger constraint on matches than knowledge of $\mathbf{F}$. While $\mathbf{F}$ only predicts the epipolar line on which the corresponding point must lie, $\mathbf{H}$ completely defines the position of the corresponding point, i.e. there is no degree of freedom left.

The procedure to do so is very similar to the computation of epipolar geometry. We start by extracting a set of possible matches between the two images. This is the data-set from which we will select random samples to compute $\mathbf{H}$. Next comes the linear computation of $\mathbf{H}$ from a minimal set of matches. There are 9 elements in the 3x3 matrix $\mathbf{H}$ which is determined up to scale so $\mathbf{H}$ has 8 degrees of freedom. Every point match ($\mathtt{m}_1$,$\mathtt{m}_2$) delivers 2 constraints on $\mathbf{H}$ since both

the x and y coordinate of $\mathtt{m}_2$ can be computed from $\mathbf{H}$ and $\mathtt{m}_1$. If we define the vector $\mathbf{h}$ similarly to $\mathbf{f}$ as

$$\mathbf{h} = [H_{11}H_{12}H_{13}H_{21}H_{22}H_{23}H_{31}H_{32}H_{33}]^T$$

and every point match ($\mathtt{m}_1$,$\mathtt{m}_2$) delivers these two equations:

$$\begin{aligned} \frac{x_2}{w_2} &= \frac{\mathbf{H}_1\mathtt{m}_1}{\mathbf{H}_3\mathtt{m}_1} \\ \frac{y_2}{w_2} &= \frac{\mathbf{H}_1\mathtt{m}_1}{\mathbf{H}_3\mathtt{m}_1} \end{aligned}$$

then every point match gives us two row vectors

$$\begin{bmatrix} x_1w_2 & y_1w_2 & w_1w_2 & 0 & 0 & 0 & -x_1x_2 & -y_1x_2 & -w_1x_2 \\ 0 & 0 & 0 & x_1w_2 & y_1w_2 & w_1w_2 & -x_1y_2 & -y_1y_2 & -w_1y_2 \end{bmatrix}$$

that can be stacked to yield

$$\mathbf{A}\,\mathbf{h} = 0$$

Since every point delivers two equations, we only need 4 points to build a linear system with 8 equations. In other words the minimum sample in RANSAC consists of 4 points as opposed to 7 for $\mathbf{F}$. The criterion that decides whether a match is an in- or outliers computes the distance between $\mathtt{m}_2$ and $\mathbf{H}\mathtt{m}_1$. If this distance is smaller than a threshold, the match is an inlier for $\mathbf{H}$. Analogously to the computation of $\mathbf{F}$, the matrix $\mathbf{A}$ needs to have well-conditioned columns. We will therefore condition the points with $\mathbf{T}$ given by Eq. 2.19. The resulting $\mathbf{H}'$ matrix must then be de-conditioned

$$\mathbf{H} \simeq \mathbf{T_2}\mathbf{H}'{\mathbf{T_1}}^{-1}$$

Chapter 3

Relating Multiple Views

3.1 Multiple View Geometry

The previous chapter dealt with finding features and correspondences between them in two views. The concept of epipolar geometry was derived and it was shown that the fundamental matrix can be computed from a set of correspondences that might be contaminated with outliers. Epipolar geometry describes the geometric relationship that exists between two images. It states a relationship between pairs of lines. This concept can be extended to 3 or more views. A naive approach to describe the relationship between 3 images would be to handle the 3-view setup as the combination of two image pairs. Consider Fig. 3.1 showing three images 1,2 and 3. We can compute the fundamental matrices $\mathbf{F}_{13}$ and $\mathbf{F}_{23}$. Let's assume that the point $\mathtt{m}_1$ in the first image and $\mathtt{m}_2$ in the second image are in correspondence (i.e. they are the projection of the same 3D point $\mathtt{M}$). Because we have computed $\mathbf{F}_{13}$ and $\mathbf{F}_{23}$, $\mathtt{m}_1$ yields the epipolar line $\mathtt{l}_1$ and $\mathtt{m}_2$ yields the epipolar line $\mathtt{l}_2$ in the third image. The corresponding point $\mathtt{m}_3$ of $\mathtt{m}_1$ and $\mathtt{m}_2$ can then be found as the intersection between $\mathtt{l}_1$ and $\mathtt{l}_2$. Immediately we can spot some problems with this approach which can be very ill-conditioned. What happens if $\mathtt{l}_1$ and $\mathtt{l}_2$ coincide? This is the case for any 3D point $\mathtt{M}$ located in the plane defined by the three camera centers $\mathtt{C}_1$, $\mathtt{C}_2$ and $\mathtt{C}_3$, called the *trifocal plane*.

The strategy to describe the geometric relationship between 3 views using epipolar geometry between pairs of images is hence not a good choice. If we inspect Fig. 3.1 more closely, however, we can see a way out of the problem of ill-conditioned or even unknown points in the third image. If $\mathtt{m}_1$ and $\mathtt{m}_2$ are known, we can always reconstruct the 3D point $\mathtt{M}$ in the frame of the first two cameras. This point projects to a unique point in the third image, even if $\mathtt{M}$ is located in the trifocal plane.

3.1.1 Trifocal Tensor

Just like the epipolar geometry describes constraints between features in a setup of two views looking at the same scene, other constraints can be written down for a configuration of three views. A very elegant way to describe this setup is by means of the *trifocal tensor*. It is beyond the scope of this document to derive all equations, lemmas and theorems that go hand in hand with this tensor. For an intuitive yet complete overview we refer the reader to [40]. Some important constraints and formulas will be given though.

Inspect Fig. 3.2. It shows a 3D point $\mathtt{M}$ projected into 3 images yielding the 2D points $\mathtt{m}_1$, $\mathtt{m}_2$ and $\mathtt{m}_3$ respectively. Imagine an arbitrary line $\mathtt{l}_3$ in the third image, going through $\mathtt{m}_3$. This line backprojects in the world as a plane. Take an arbitrary line $\mathtt{l}_2$ in the second image, going through $\mathtt{m}_2$. This line projects back as a second plane. The intersection of these two planes (the 3D line $\mathtt{L}$) contains the 3D point $\mathtt{M}$. The projection of this line in the first image $\mathtt{l}_1$ has to contain the point $\mathtt{m}_1$. This property can mathematically be expressed as follows. For every constellation of 3 views,

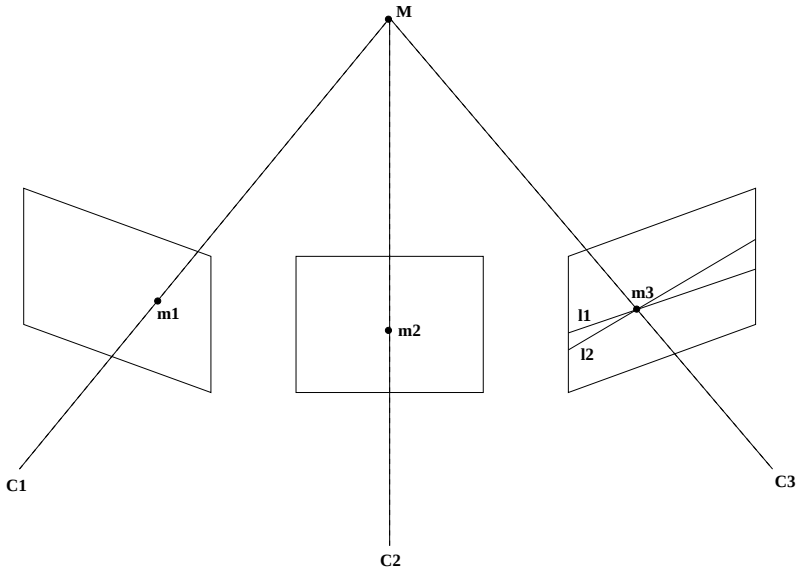


Figure 3.1: The corresponding point $\mathtt{m}_3$ of $\mathtt{m}_1$ and $\mathtt{m}_2$ can be found at the intersection of the two epipolar lines $\mathtt{l}_1$ and $\mathtt{l}_2$ in the third image.

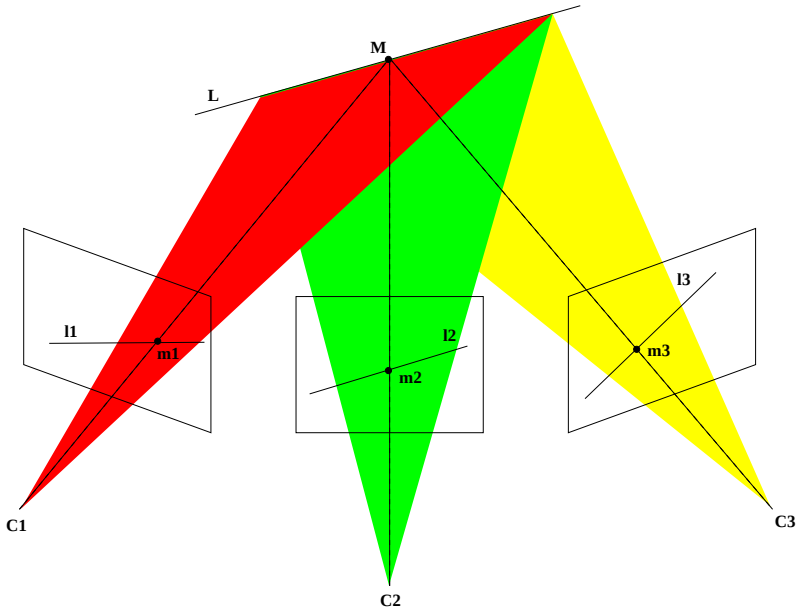


Figure 3.2: The trifocal tensor describes the following property: A 3D point is projected into three images. Lines $\mathtt{l}_2$ and $\mathtt{l}_3$ go through these projections (but don't have to be corresponding lines). The intersection of the two backprojected planes through these lines is a 3D line `L` which contains the 3D point. Hence the projection of this line in the first image has to contain the projection of the 3D point in this image.

there exists a trifocal tensor $\mathbf{T}$ such that

$$\mathtt{l}_2^{\mathtt{T}}[\mathbf{T}\mathtt{m}_1]\mathtt{l}_3 = 0 \tag{3.1}$$

The trifocal tensor $\mathbf{T}$ is a $3 \times 3 \times 3$ array with 27 elements. Only 18 of these 27 parameters are linearly independent, due to additional non-linear constraints.

$$\mathbf{T} = \left(T_k^{ij}\right)_{1 \le i,j,k \le 3}$$

The multiplication with the point $\mathtt{m}_1$ yields a 3×3 matrix

$$[\mathbf{T}\mathtt{m}_1] = \mathbf{T}_1^{ij} x_1 + \mathbf{T}_2^{ij} y_1 + \mathbf{T}_3^{ij} w_1$$

Equation 3.1 expresses the trifocal relation between two lines and one point. It is important to realize that the lines $\mathtt{l}_2$ and $\mathtt{l}_3$ don't have to be corresponding lines for Eq. 3.1 to hold. However, if they are correponding lines (i.e. coming from the same 3D line in space), then it should be possible to compute the corresponding line in the first image $\mathtt{l}_1$ directly since it is the projection of the 3D line in the first image. We have defined that

$$[\mathbf{T}\mathtt{m}] = \mathbf{T}_1 x + \mathbf{T}_2 y + \mathbf{T}_3 w$$

and hence we can write Eq. 3.1 as

$$\mathtt{l}_2^{\mathtt{T}}[\mathbf{T}_1 x + \mathbf{T}_2 y + \mathbf{T}_3 w]\mathtt{l}_3 = 0$$

or

$$(\mathtt{l}_2^{\mathtt{T}}\mathbf{T}_1\mathtt{l}_3)x + (\mathtt{l}_2^{\mathtt{T}}\mathbf{T}_2\mathtt{l}_3)y + (\mathtt{l}_2^{\mathtt{T}}\mathbf{T}_3\mathtt{l}_3)w = 0 \tag{3.2}$$

Because $\mathtt{l}_2$ and $\mathtt{l}_3$ correspond, we can write Eq. 3.2 for all points on them and hence Eq. 3.2 must be valid for all points $(x, y, w)^T$. This means that the corresponding line in image 1 is given by

$$\mathtt{l}_1 \simeq \begin{pmatrix} \mathtt{l}_2^{\mathtt{T}}\mathbf{T}_1\mathtt{l}_3 \\ \mathtt{l}_2^{\mathtt{T}}\mathbf{T}_2\mathtt{l}_3 \\ \mathtt{l}_2^{\mathtt{T}}\mathbf{T}_3\mathtt{l}_3 \end{pmatrix} \tag{3.3}$$

In other words, if the trifocal tensor of a constellation of 3 images is known and corresponding lines in two images are known as well, then the corresponding line in the third image is readily computed from Eq. 3.3. Whereas the epipolar constraint is the "simplest" constraint(in the sense that it involves the smallest number of views) that must hold between corresponding points in different images, so is Eq 3.3 the "simplest" constraint that must hold between corresponding lines in different images. Indeed if only two views of the scene are given, then any pair of lines in the two views might be corresponding lines because they can always be interpreted as being the images of the intersection line $\mathtt{L}$ of their backprojecting planes. To verify whether this is indeed the case, a third view is needed. In fact formula 3.3 actually predicts where the corresponding line is to be found in the image. In other words, Eq. 3.3 allows to transfer lines from two images to a third one.

Relations between lines and points for constellations of more than 3 views exist too. The relation between 4 views for instance can be described by means of a quadrifocal tensor. We will not go into this here. The reader is directed to [40] and [14] for more information on these topics.

3.2 Structure and Motion

Many researchers have studied trifocal, quadrifocal or further tensors for 3D reconstruction. We will follow a more pragmatic approach, using computations in a projective frame. The output of this algorithm will consist of projection matrices for the cameras and 3D coordinates of points in a projective frame. Hence we solve the problem of finding both *Structure and Motion* (SaM) from image information only. The structure of the algorithm we employ is given in table 3.2. The different steps in the algorithm are explained in the following paragraphs.

Step 1.	Compute the epipolar geometry for the first image pair
Step 2.	Initialize structure and camera pair
Step 3.	For every consecutive image, do
	Step 3.1 Compute matches between this and the previous image
	Step 3.2 Estimate the projection matrix
	Step 3.3 Update structure and initialize new points
Step 4.	Global minimization of all results

Table 3.1: SaM algorithm

3.2.1 Initialization Step

Table 3.2 shows that the first two steps we employ consist of the computation of the epipolar geometry between the first two images and the initialization of the structure and motion. The former is done using the algorithms explained in the previous chapter. The combination of F-RANSAC, non-linear optimization and F-guided matching delivers a good estimation of the fundamental matrix $\mathbf{F}_{12}$ and a set of matching image points.

Initializing the Projective Frame

If we want to initialize the structure, we need first to define initial values for the projection matrices $\mathbf{P}_1$ and $\mathbf{P}_2$. If these projection matrices were given, the fundamental matrix would be readily computed from these using Eq. 2.13. Vice versa, if the fundamental matrix is known, some constraints on the projection matrices can be written down. If we choose the first projection matrix to be located in the origin of the world, and an identity matrix as the left 3x3 part, then the two projection matrices can be initialized as

$$\mathbf{P}_1 = [\mathbf{I}_{3\times 3}|\mathbf{0}_3] \tag{3.4}$$

$$\mathbf{P}_2 = [[\mathsf{e}_{12}]_\times \mathbf{F}_{12} - \mathsf{e}_{12}\pi^T | a\mathsf{e}_{12}] \tag{3.5}$$

with e_{12} the epipole in the second image (the projection of the first image center in the second image) and π an arbitrary 3D plane. The exact derivation is beyond the scope of this text; details can be found in [25]. Note that there are 4 degrees of freedom left in Eq. 3.5: the 3 coefficients of the plane π and the scale factor a. This can be logically explained since the fundamental matrix has 7 degrees of freedom (9 elements - 1 for scale - 1 for rank 2 constraint) and a projection matrix has 11 degrees of freedom (12 elements - 1 for scale). This means that Eqs. 3.4, 3.5 can be used to initialize a projective frame for the reconstruction but can not be employed to compute the projection matrices of consecutive images because they would only be defined up to 4 degrees of freedom and hence not in the same projective frame as the first two cameras.

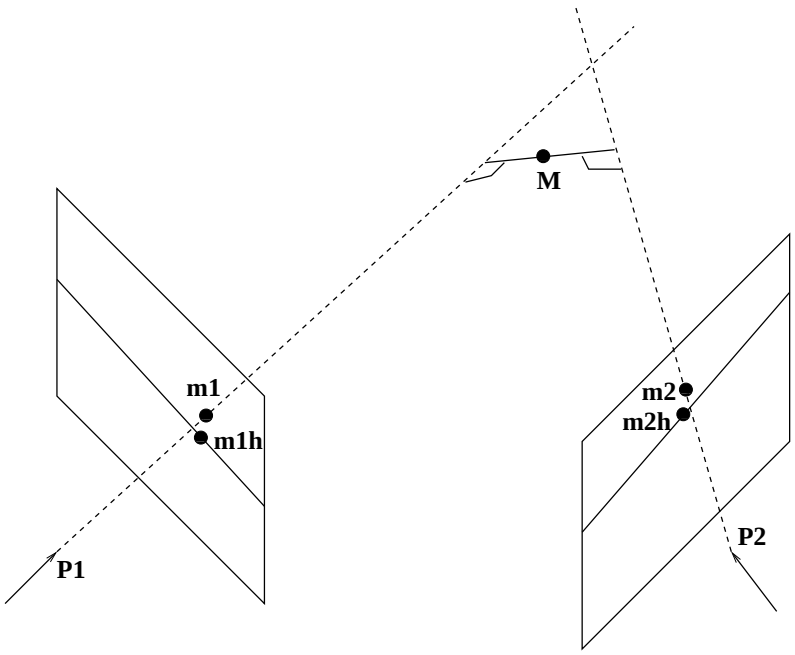


Figure 3.3: Triangulating 2D correspondence $\mathtt{m}_1$, $\mathtt{m}_2$ to yield the 3D point $\mathtt{M}$. In a metric frame we can minimize the 3D distance to the backprojected rays. In a projective frame we have to minimize the distance of the projection of $\mathtt{M}$ to $\mathtt{m}_1$ and $\mathtt{m}_2$

Selecting the Best Initialization Pair

In practice, it is not always a good idea to choose the first two images in the sequence for initialization. Other image pairs might be better suited. Which criterion could one use to determine if an image pair is better suited for initialization of the projective frame than others? Such a pair must fulfill two requirements. First, many correspondences must exist between the two images, in order to start with plenty 3D reconstructed points. Secondly, the two images should not be too close together for well-conditioned initialization. The first requirement is easy to check. The second is harder because in a projective frame we cannot use metric qualities like the distance between cameras. We use the following algorithm.

- Compute the best fitting homography between every consecutive image pair.
- Compute the median residual error of every match w.r.t. this homography.

The homography computation finds the homography that fits the correspondences between two images best in a least squares, as explained in paragraph 2.2.6. No RANSAC procedure is used, just one computation of $\mathbf{H}$. The median residual error is found as the median of all distances between points in the second image and the transformation via $\mathbf{H}$ of the matching point in the first image.

$$median\{D(\mathbf{H}\mathtt{m}_\mathtt{i}, \mathtt{m}_\mathtt{j})\}$$

For every consecutive image pair a combination of the number of matches and the median residual error is computed and the image pair which maximizes this value is selected for initialization. In our implementation we opted for a simple multiplication of the number of matches and the residual.

Triangulation

When the initializing cameras are chosen and the first two cameras are computed with Eq. 3.4, 3.5, all correspondences we found in the F-guided matching step can be reconstructed in the initialized

frame. Fig. 3.3 illustrates the process. Due to noise and the margin we allowed around the epipolar line in the F-guided matching step, it is not guaranteed that the two backprojected rays lie in the same plane. Which 3D point must we choose? In a metric frame this can be done by computing the closest 3D point to the two rays (M in Fig. 3.3). In a projective frame we cannot use a metric quality like distances in 3D. The only measure we can safely minimize is the distance in the images. Two options are available to us. Either we first compute the two 2D points $\mathtt{m}_{1\mathtt{h}}$ and $\mathtt{m}_{2\mathtt{h}}$ which are the points closest to the original points $\mathtt{m}_1$ and $\mathtt{m}_2$ but this time lying on each others epipolar line. This *Hartley-Sturm* adaptation of the points guarantees that the backprojected lines coincide. Another approach is simpler but effective. We search for the 3D point which, projected into the images with $\mathbf{P}_1$ and $\mathbf{P}_2$, lies on $\mathtt{m}_1$ and $\mathtt{m}_2$ respectively. This means

$$\begin{aligned} \mathtt{m}_1 &= \mathbf{P}_1\mathtt{M} \\ \mathtt{m}_2 &= \mathbf{P}_2\mathtt{M} \end{aligned}$$

This can be written lineary for both points $\mathtt{m} = (x, y, w)^T$ and both projection matrices $\mathbf{P} = (P_1, P_2, P_3)^T$ as

$$\begin{aligned} P_3\mathtt{M}x &= P_1\mathtt{M} \\ P_3\mathtt{M}y &= P_2\mathtt{M} \end{aligned} \tag{3.6}$$

If we follow the same approach we used to compute $\mathbf{F}$ and $\mathbf{H}$, we can compute M linearly by computing the SVD of the matrix $\mathbf{A}$

$$\mathbf{A} = \begin{bmatrix} wp_{11} - xp_{31} & wp_{12} - xp_{32} & wp_{13} - xp_{33} & wp_{14} - xp_{34} \\ wp_{21} - yp_{31} & wp_{22} - yp_{32} & wp_{23} - yp_{33} & wp_{24} - yp_{34} \\ w'p'_{11} - x'p'_{31} & w'p'_{12} - x'p'_{32} & w'p'_{13} - x'p'_{33} & w'p'_{14} - x'p'_{34} \\ w'p'_{21} - y'p'_{31} & w'p'_{22} - y'p'_{32} & w'p'_{23} - y'p'_{33} & w'p'_{24} - y'p'_{34} \end{bmatrix} \tag{3.7}$$

where $\mathtt{m}_1 = (x, y, w)^T$, $\mathtt{m}_2 = (x', y', w')^T$, $\mathbf{P}_1 = [p_{ij}]$ and $\mathbf{P}_2 = [p'_{ij}]$.

Actually we don't want to minimize the algebraic error on Eq. 3.6 but are more interested in minimizing the true reprojection error $\sum D(\mathbf{P}_i\mathtt{M}, \mathtt{m}_\mathtt{i})$. This boils down to finding M which minimizes

$$\begin{aligned} x &= \frac{P_1\mathtt{M}}{P_3\mathtt{M}} \\ y &= \frac{P_2\mathtt{M}}{P_3\mathtt{M}} \end{aligned} \tag{3.8}$$

in a least squares sense. This non-linear computation can be performed by computing the solution of Eq. 3.6, which we call $\tilde{\mathtt{M}}$. Then we solve another system of equations

$$\begin{aligned} \frac{P_3\mathtt{M}x}{P_3\tilde{\mathtt{M}}} &= \frac{P_1\mathtt{M}}{P_3\tilde{\mathtt{M}}} \\ \frac{P_3\mathtt{M}y}{P_3\tilde{\mathtt{M}}} &= \frac{P_2\mathtt{M}}{P_3\tilde{\mathtt{M}}} \end{aligned} \tag{3.9}$$

Iterating this approach a number of times will finally yield the solution which minimizes the true reprojection error.

3.2.2 Projective Pose Estimation

After initialization we can start adding other cameras to the projective frame. For this we use the matches we have computed between consecutive images using the F-RANSAC and F-guided matching algorithms. The situation we are in now is depicted in Fig. 3.4. In this figure we have three images with corresponding projective projection matrices and a set of 3D points in the same projective frame with corresponding 2D points. We also have matches between the last image and the new image which we want to add to the projective frame.

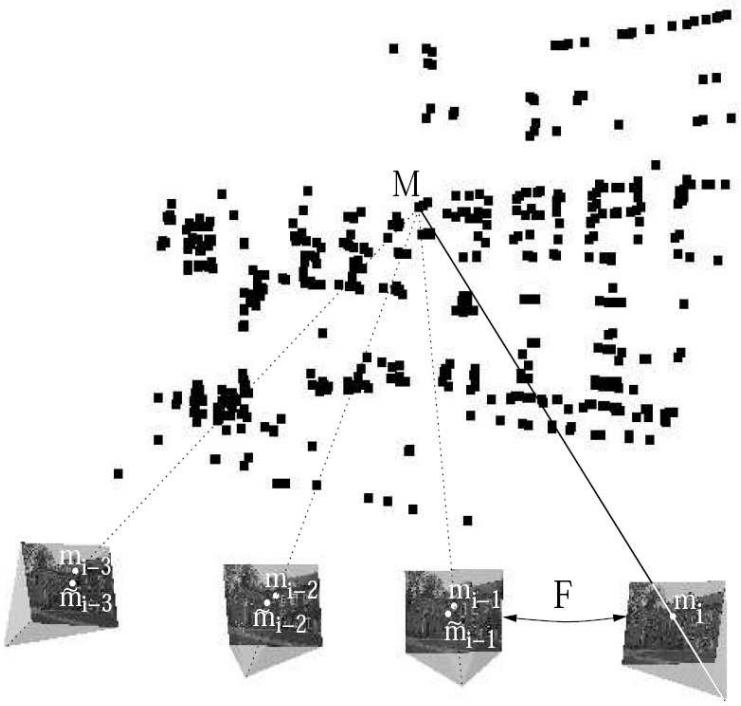


Figure 3.4: The 3D point $\mathtt{M}$ has been seen in and reconstructed from images $i-3$ to $i-1$ where it projects into $\mathtt{m_{i-3}}$ to $\mathtt{m_{i-1}}$. An F-RANSAC and F-Guided matching algorithm between images $i-1$ and i delivers a set of matches between these images. The match $< \mathtt{m_{i-1}}, \mathtt{m_i} >$ provides a possible 3D-2D correspondence $< \mathtt{M}, \mathtt{m_i} >$. If enough of these correspondences are available, the projection matrix $\mathbf{P}_i$ can be computed in the existing frame using RANSAC.

For every match between points $\mathtt{m_{i-1}}$ and $\mathtt{m_i}$ in the last image $i-1$ and the new image i we check if the point $\mathtt{m_{i-1}}$ has a corresponding 3D point $\mathtt{M}$. If it does, the 3D-2D correspondence $< \mathtt{M}, \mathtt{m_i} >$ is a possibly correct piece of information which we can employ to compute the projection matrix $\mathbf{P}_i$ in the existing projective frame. If we have enough of these 3D-2D correspondences, the projection matrix can be computed linearly. Every 3D-2D correspondence yields 2 equations: the x and y coordinate of the projection of the 3D point in the image ($\mathbf{P}_i\mathtt{M}$) must coincide with the x and y coordinate of the point $\mathtt{m_i}$. Since a projection matrix has 11 degrees of freedom (12 elements - 1 for scale) and every correspondence yields 2 equations, we need 6 correspondences to compute $\mathbf{P}_i$ linearly. Of course, we will have to deal with possible outliers again. Since the fundamental matrix only checks if corresponding points are on each other's epipolar line, possible mismatches along this line can still appear. Thus wrong 2D-2D matches between $i-1$ and $i-2$ might have been reconstructed or wrong 2D-2D matches between i and $i-1$ appear. Both give rise to wrong 3D-2D correspondences. This means we will have to use the RANSAC paradigm once again to distinguish the inliers from the outliers and select the projection matrix with the largest support.

The linear algorithm in the inner RANSAC loop again solves for the SVD of a matrix $\mathbf{A}$. This time the matrix is stacked with 2 rows per 3D-2D correspondence $\mathtt{M} = (X, Y, Z, W)^T$, $\mathtt{m} = (x, y, w)^T$.

$$\begin{bmatrix} wX & wY & wZ & wW & 0 & 0 & 0 & 0 & -xX & -xY & -xZ & -xW \\ 0 & 0 & 0 & 0 & wX & wY & wZ & wW & -yX & -yY & -yZ & -yW \end{bmatrix}$$

RANSAC also needs a criterion to check if, given a computed P-matrix, a correspondence is an inlier or an outlier. A naive approach would be to simply project the 3D point $\mathtt{M}$ into the new image i and check if the distance between the projection and the 2D point, $D(\mathbf{P}_i\mathtt{M}, \mathtt{m_i})$, is below a threshold. This is not a good idea. All incorrect 3D points lying on or close to the backprojected ray through $\mathtt{m_i}$ would then always be judged to be inliers.. An alternative could be to triangulate the 3D point from $\mathtt{m_i}$ and $\mathtt{m_{i-1}}$ using $\mathbf{P}_i$ and $\mathbf{P}_{i-1}$ and then check the distance between the reconstructed point and the original point $\mathtt{M}$. In order to do so, however, we would again need to measure the distance between points, which we cannot use because our reconstruction is only valid up to any projective transformation. Hence, we will need to employ information in all other images. We backproject the 3D point again but this time using the 2D points in all images ($\mathtt{m_i}$, $\mathtt{m_{i-1}}$, $\mathtt{m_{i-2}}$, ...) and all projection matrices ($\mathbf{P}_i$, $\mathbf{P}_{i-1}$, $\mathbf{P}_{i-2}$, ...). We use Eq. 3.7 but this time with a matrix $\mathbf{A}$ with more than 4 rows, namely 2 times the number of images, including the new image i. This clearly is an overdetermined system and solving it will yield the 3D point $\tilde{\mathtt{M}}$ that minimizes the projection-residual in the images in a least-squares way. We then check the residual errors $D(\mathbf{P}\tilde{\mathtt{M}}, \mathtt{m})$ in all images. Only if this residual is below a threshold in all images will we accept the 3D-2D correspondence as an inlier for the computed $\mathbf{P}_i$.

3.2.3 Updating Structure

Every time a new projection matrix has been added, the structure of 3D points must be updated. Every new projection matrix has been computed making use of 3D-2D correspondences. Once $\mathbf{P}_i$ is known, we can update the position of the existing 3D points if a new observation of them is available in the form of the correspondence $< \mathtt{M}, \mathtt{m_i} >$. Eq. 3.7 with the extended matrix $\mathbf{A}$, containing rows from all images the point is visible in, is used again to compute the new position of $\mathtt{M}$. It is best to use the iterative scheme of Eq. 3.9 to minimize the true reprojection error.

Feature matches will exist between image i and $i-1$ that have no 3D-2D correspondence, i.e. for which no consistent match existed between $i-1$ and $i-2$. These points can now be instantiated in the projective frame using triangulation with projection matrices $\mathbf{P}_{i-1}$ and $\mathbf{P}_i$. This instantiation is vital because otherwise only points visible in the first images will be used for pose-estimation. Once they would fall out of view, new cameras could not be added anymore.

3.2.4 Global Minimization

Once the structure and motion has been obtained for the whole sequence, it is recommended to refine it through a global minimization step. A maximum likelihood estimation can be obtained through bundle adjustment [63, 54]. The extracted image points are the only piece of information that can be assumed to be fixed. The parameters of the different cameras and the reconstructed 3D points have been computed from them and the goal is to find the projection matrices and 3D points for which the mean squared distances between the observed image points and the projected image points is minimized. For m views and n points the following criterion should be minimized:

$$\sum_{k=1}^{m}\sum_{i=1}^{n} D(\mathtt{m}_{ki}, \mathbf{P}_k \mathtt{M}_i)^2$$

If the image error is zero-mean Gaussian then bundle adjustment is the Maximum Likelihood Estimator. Although it can be expressed very simply, this minimization problem is huge. For a typical sequence of 20 views and 2000 points, a minimization problem in more than 6000 variables has to be solved. A straight-forward computation is obviously not feasible. However, the special structure of the problem can be exploited to solve the problem much more efficiently. The observed points $\mathtt{m}_{ki}$ being fixed, a specific residual $D(\mathtt{m}_{ki}, \mathbf{P}_k \mathtt{M}_i)^2$ is only dependent on the i-th point and the k-th camera view. This results in a sparse structure for the normal equations. Using this structure the points $\mathtt{M}_i$ can be eliminated from the equations, yielding a much smaller but denser problem. Views that have features in common are now related. For a long sequence where features tend to only be seen in a few consecutive views, the matrix that has to be solved is still sparse (typically band diagonal). In this case it can be very interesting to make use of sparse linear algebra algorithms, e.g. [5].

As we have seen in paragraph 1.3.3, images can suffer from radial distortion. If possible, the camera projection model should also take this distortion into account. Unfortunately, as explained in paragraph 1.3.3 the camera model proposed in Eq. 1.6, 1.7 puts the radial distortion parameters between the extrinsic and linear intrinsic parameters of the camera. At this moment we have only computed a projective reconstruction of the scene and cameras and we have no clue what the intrinsics of the cameras are. Therefore, it is impossible to estimate the radial distortion parameters with the proposed model. A way around this is to use a crude estimation of the internal parameters matrix $\mathbf{K}_e$ for the time being. This matrix can be defined as

$$\mathbf{K}_e = \begin{bmatrix} \frac{w+h}{2} & 0 & \frac{w}{2} \\ 0 & \frac{w+h}{2} & \frac{h}{2} \\ 0 & 0 & 1 \end{bmatrix}$$

with w and h the width and height of the image in pixels. The radial distortion is then applied as follows. The matrix $\mathbf{P}'$ is defined as

$$\mathbf{P}' \simeq \mathbf{K}_e^{-1}\mathbf{P}$$

The 3D point is projected via $\mathbf{P}'$. There the radial distortion formula of Eq. 1.6 is applied and the resulting point is multiplied on the left with $\mathbf{K}_e$ again.

Chapter 4

Self-Calibration

4.1 The Projective Ambiguity

The images have been related to each other in a projective frame. After the projective bundle adjustment step, we have recovered the projective projection matrices $\mathbf{P}_k$ for every image and a set of 3D points $\mathtt{M}_i$ that, projected via those projection matrices, result in the smallest residual error w.r.t. the extracted feature points $\mathtt{m}_{ki}$.

Unfortunately, since the only true information we have consists of these image points, the resulting reconstruction is only valid up to any projective transformation as we have derived in paragraph 1.5.2: for every image point that is a projection of a 3D point, the equation

$$\mathtt{m}_{ki} \simeq \mathbf{P}_k \mathtt{M}_i$$

must hold. Inserting an arbitrary 4×4 projective transformation yields the same image point:

$$\begin{aligned} \mathtt{m}_{ki} &\simeq \mathbf{P}_k \mathbf{T} \mathbf{T}^{-1} \mathtt{M}_i \\ \mathtt{m}_{ki} &\simeq (\mathbf{P}_k \mathbf{T})(\mathbf{T}^{-1} \mathtt{M}_i) \\ \mathtt{m}_{ki} &\simeq \mathbf{P}'_k \mathtt{M}'_i \end{aligned}$$

and thus the solution consisting of projection matrices $\mathbf{P}'_k$ and 3D points $\mathtt{M}'_i$ is equally valid and cannot be discerned from other reconstructions based on image feature measurements only.

Because the reconstruction can be transformed by any projective transformation and still be valid, this means that many properties (like orthogonality, relative distances, parallelism) are not preserved. For the reconstruction result to be useful, we need to upgrade it from projective to metric, a process called *self-calibration*. Fig. 4.1 shows one of the images of a sequence that was recorded in the town of Mechelen in Belgium. It shows part of the remaining wall of a medieval building (on the right), called a "Spijker" which was originally a warehouse but performed many other functions later. The resulting projective reconstruction is shown in the top row of Fig. 4.2. It is clear that, although projective properties like planarity are preserved, this reconstruction is unusable. We want to upgrade it to metric, the result of which is shown in the bottom row. The projective deformation is actually quite mild in this example because the plane at infinity is already quite far from the reconstruction compared to its size. Some views of the reconstruction, deformed by another projective transformation where the plane at infinity is placed elsewhere, are shown in Fig. 4.3. This shows the devastating effects a projective transformation can have.

How can the upgrade from a projective to a metric reconstruction be done? Different techniques can be employed, depending on the information one disposes of. In paragraph 4.2 we will explain how self-calibration can be performed if certain constraints on the scene are known. Paragraph 1.6.1 and 1.6.2 explained how to calibrate intrinsics and extrinsics of cameras. More information on the use of such techniques will be given in chapter 6. If no information on the scene or camera motion is known, a self-calibration technique which imposes constraints on the intrinsics of the cameras can be used. This is explained in paragraph 4.3.



Figure 4.1: One of the images of a sequence taken in Mechelen. The grey wall on the right is part of a medieval building.

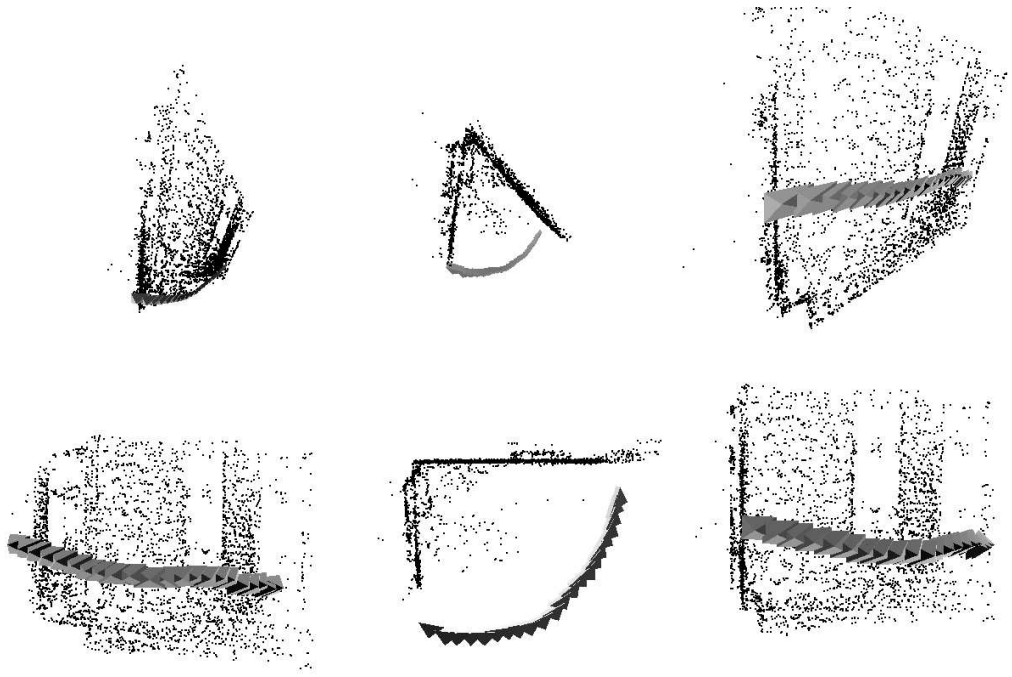


Figure 4.2: Top row: Projectively deformed reconstruction, output of the projective structure and motion algorithm. Planar regions are still planar but metric quantities like angles and (relative) distances are clearly not preserved. Bottom row: Metric reconstruction, obtained via self-calibration.

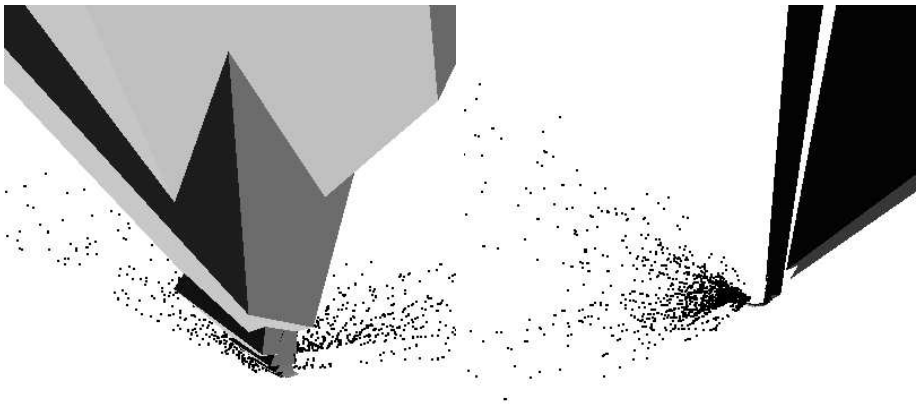


Figure 4.3: Two views of the reconstruction of the sequence of the wall in Mechelen, deformed by another projective transformation. Although the 3D looks nothing like reality, still all projections of 3D points into the images via the projection matrices coincide neatly with the extracted feature points.

4.2 Scene Constraints

We will first discuss how knowledge of the scene can be exploited to perform the upgrade from a projective to a metric reconstruction. The goal is to find new projection matrices $\hat{\mathbf{P}}_k$ and 3D points $\hat{\mathtt{M}}_i$ for which certain constraints imposed on these hold while minimizing the mean squared distances between the observed image points $\mathtt{m}_{ik}$ and the reprojected image points $\hat{\mathtt{m}}_{ik}$. The constraints we can think of here are based on properties that are only preserved in reconstructions, valid up to a metric transformation, such as orthogonality of lines or planes, parallelism of lines or planes (actually an affine property) and relative distances between two points. Imposing these constraints means that one has to have knowledge of where they apply. One could think of techniques that search for these properties automatically but here we will assume that a human operator indicates the constraints via a GUI.

When the different constraints have been indicated, a minimization algorithm is run which imposes these constraints. Formally, given a collection of image points $\mathtt{m}_{ki}$, n 3D points, m images, some collections of orthogonal lines $\mathtt{C}_{\mathtt{ol}}$, parallel lines $\mathtt{C}_{\mathtt{pl}}$, orthogonal planes $\mathtt{C}_{\mathtt{op}}$, parallel planes $\mathtt{C}_{\mathtt{pp}}$ and a collection of distances $\mathtt{C}_{\mathtt{d}}$, a global optimization will be run minimizing the following error function :

$$\mathcal{E}(\hat{\mathbf{P}}_k)$$

where $\mathcal{E}$ depends on the constraint(s) used:

- The error on the projection as described above :

$$\frac{\sum_{i=1}^{n}\sum_{j=1}^{m}(D(\mathtt{m}_{ij},\hat{\mathbf{P}}_j\hat{\mathtt{M}}_i))}{\sigma_p^2}$$

  where $D(\hat{\mathtt{m}},\mathtt{m})$ is the Euclidean image distance.

- Orthogonality of lines :

$$+\frac{\sum_{(k,l)\in\mathtt{C}_{ol}}\left(\frac{\mathtt{L}_k\cdot\mathtt{L}_l}{\|\mathtt{L}_k\|\|\mathtt{L}_l\|}\right)^2}{\sigma_\perp^2}$$

  with $\cdot$ denoting the dot product of two vectors.



Figure 4.4: Two images of an indoor scene, filmed from two surveillance cameras.

- Orthogonality of planes :

$$+\frac{\sum_{(m,n)\in \mathtt{C}_{op}} (\hat{\mathtt{N}}_m \cdot \hat{\mathtt{N}}_n)^2}{\sigma_\perp^2}$$

  $\mathtt{N}_m$ being the normalized normal vector of the plane.

- Parallelism of lines :

$$+\frac{\sum_{(k,l)\in \mathtt{C}_{pl}} \left(\frac{\|\hat{\mathtt{L}}_k \times \hat{\mathtt{L}}_l\|}{\|\hat{\mathtt{L}}_k\| \|\hat{\mathtt{L}}_l\|} \right)^2}{\sigma_\parallel^2}$$

  with $\times$ denoting the cross product of two vectors.

- Parallelism of planes :

$$+\frac{\sum_{(m,n)\in \mathtt{C}_{pp}} (\hat{\mathtt{N}}_m \times \hat{\mathtt{N}}_n)^2}{\sigma_\parallel^2}$$

- Distances between points :

$$+\frac{\sum_{(i,j)\in \mathtt{C}_d} (d_{i,j} - \|\hat{\mathtt{M}}_i - \hat{\mathtt{M}}_j\|)^2}{\sigma_d^2}$$

  with $d_{i,j}$ the constrained distance between points i and $_j$.

The σ values (σ_p, $\sigma_\perp$, $\sigma_\parallel$, σ_d) are inverse weights on every constraint. Uncertain constraints like have a larger σ than highly constrained properties.

Calibration based on scene constraints is a time-consuming process because all constraints have to be identified by an operator. Moreover, there is a risk of constraints being incompatible. However, in some cases we have no other choice. For instance, if only two images of a scene are given, taken by two cameras with different intrinsic parameters, the techniques that will be explained in paragraph 4.3 cannot be applied. An example of this is shown in Fig. 4.4, depicting an indoor scene, filmed by two surveillance cameras. The two images are quite far apart, so for matching them wide-baseline matching techniques have to be used or manual interaction is necessary. Fig. 4.5 shows the information the operator has provided. Parallel and orthogonal planes and lines have been indicated.

A projective reconstruction of indicated regions, based on the recovered epipolar geometry can be seen in Fig. 4.6 on the left. The constraints, indicated in Fig. 4.5 have been used to upgrade this projective reconstruction to metric, the result of which is shown on the right in Fig. 4.6.

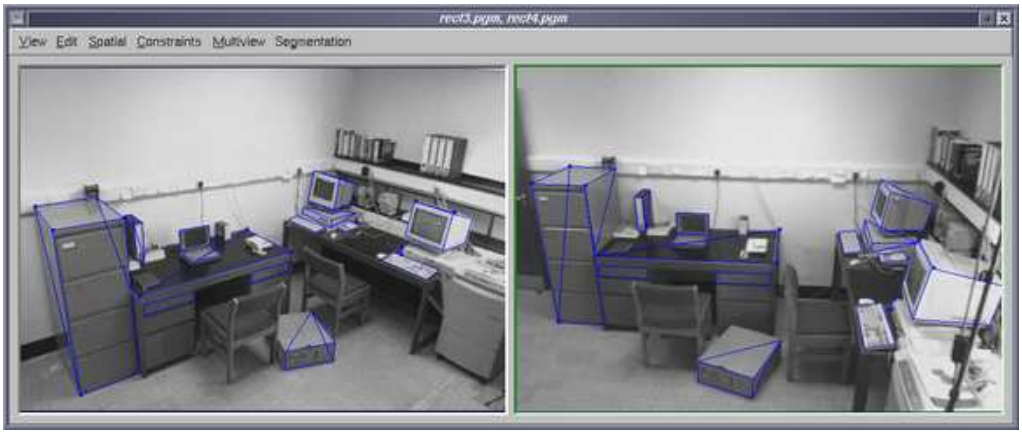


Figure 4.5: Scene-constraints are indicated on the images via a GUI.

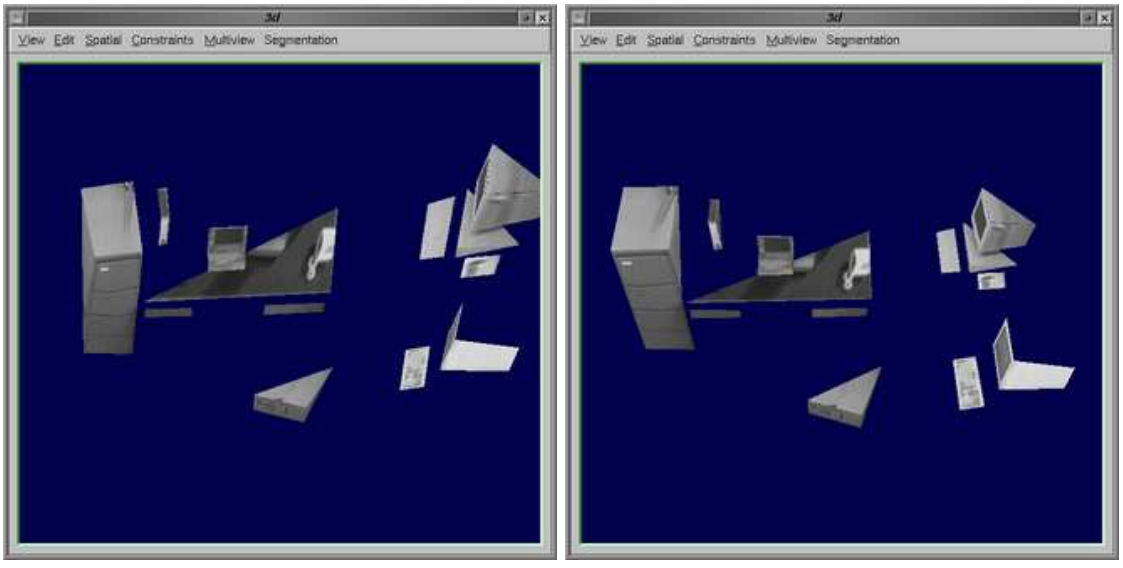


Figure 4.6: Projective (left) and metric reconstruction of the indoor scene. The metric reconstruction is found by upgrading the projective one via a global minimization which takes into account the indicated constraints.

4.3 Self-Calibration

In the previous paragraph we explained how to upgrade the reconstruction from projective to metric using scene constraints. Manual interaction is necessary for this approach. In this section an approach will be given which allows for completely automatic self-calibration. If we cannot employ constraints on the scene (because this is a manual process) or on the camera path (because it was not measured), then the only way to upgrade the reconstruction to metric is by imposing constraints on the intrinsic parameters of the camera. Indeed, in the projective reconstructions all cameras are described by projection matrices. This description does not take into account the fact that a real-life camera took those images and that we have some extra information: typically it is the same camera that was used to record the sequence; the camera might be digital (with zero skew); etc.

We will first explain a few basic concepts on conics and quadrics in projective geometry, before we go into more details on how a specific conic (the *absolute conic*) can be used for self-calibration. Afterwards we will explain a practical algorithm and give more information on how to self calibrate different sequences, recorded with the same camera.

4.3.1 Conics and Quadrics

The use of conics and quadrics in projective geometry has been studied for many years. The concept of the *absolute conic*, which we will define in paragraph 4.3.2 for instance has been known for several decades. However, only in the last 10 years has this quadric shown its use in the area of self-calibration.

Conics

In $\Re^2$ quadratic expressions of homogeneous points amount to the concept of *conics*. One can think of conics as the intersection of a cone with a plane. This intersection can take the form of a parabola, a hyperbola or an ellipse, including all their degenerate forms. In homogeneous coordinates, a conic $\mathbf{C}$ can be defined as the set of points $\mathtt{m}$ for which the homogeneous equation

$$\mathtt{m}^T\mathbf{C}\mathtt{m} = 0 \tag{4.1}$$

holds. The matric $\mathbf{C}$ defining the conic is a 3×3 symmetric matrix which is (of course) only defined up to scale. This means a conic has 5 independent parameters describing it.

We know that in 2D the projective world, almost every concept, defined for points, has its dual concept, defined for lines. The *dual conic* of the conic $\mathbf{C}$ can be defined as the set of lines that are tangential to the points on the conic $\mathbf{C}$. The dual conic is described by the matric $\mathbf{C}^*$ and all lines that are in the set have to satisfy

$$\mathtt{l}^T\mathbf{C}^*\mathtt{l} = 0 \tag{4.2}$$

In order to derive the relation between the conic and its dual we have to compute the intersection points between a line and the conic. These points both lie on the line and the conic. Let us define the line $\mathtt{l}$ as going through two points $\mathtt{m}_1$, $\mathtt{m}_2$. Then a point on this line can be written as $\mathtt{m}_1 + \lambda\mathtt{m}_2$

This point must lie on the conic $\mathbf{C}$ as well, hence

$$\begin{aligned} (\mathtt{m}_1 + \lambda\mathtt{m}_2)^T\mathbf{C}(\mathtt{m}_1 + \lambda\mathtt{m}_2) &= 0 \\ \mathtt{m}_1^T\mathbf{C}\mathtt{m}_1 + \lambda\mathtt{m}_1^T\mathbf{C}\mathtt{m}_2 + \lambda\mathtt{m}_2^T\mathbf{C}\mathtt{m}_1 + \lambda^2\mathtt{m}_2^T\mathbf{C}\mathtt{m}_2 &= 0 \\ \mathtt{m}_1^T\mathbf{C}\mathtt{m}_1 + 2\lambda\mathtt{m}_1^T\mathbf{C}\mathtt{m}_2 + \lambda^2\mathtt{m}_2^T\mathbf{C}\mathtt{m}_2 &= 0 \end{aligned} \tag{4.3}$$

since $\mathtt{m}_2^T\mathbf{C}\mathtt{m}_1 = \mathtt{m}_1^T\mathbf{C}^T\mathtt{m}_2$ because $\mathbf{C}$ is a symmetric matrix.

If the line intersects the conic in two coinciding points, it is a tangent to the conic. This only happens if the discriminant of the quadratic equation 4.3 is 0, in other words if

$$(\mathtt{m}_1^T\mathbf{C}\mathtt{m}_2)^2 = \mathtt{m}_2^T\mathbf{C}\mathtt{m}_2\mathtt{m}_1^T\mathbf{C}\mathtt{m}_1 \tag{4.4}$$

Let us assume that the point $\mathtt{m}_1$ is fixed. Then equation 4.4 is a quadratic equation in the elements of $\mathtt{m}_2$. This equation then represents the two tangent lines to the conic going through the point $\mathtt{m}_1$. If the point $\mathtt{m}_1$ lies on the conic, then $\mathtt{m}_1^T\mathbf{C}\mathtt{m}_1 = 0$. Filling this in in equation 4.4 yields

$$\mathtt{m}_1^T\mathbf{C}\mathtt{m}_2 = 0$$

This is a linear equation in the elements of $\mathtt{m}_2$, hence describing a single tangential line to the conic. The vector describing the line is given by

$$\mathtt{l} \simeq \left(\mathtt{m}_1^T\mathbf{C}\right)^T = \mathbf{C}^T\mathtt{m}_1 = \mathbf{C}\mathtt{m}_1 \tag{4.5}$$

This result allows us to write down the relation between the conic $\mathbf{C}$ and its dual $\mathbf{C}^*$. This dual holds all tangent lines to the conic, i.e. all lines described by Eq. 4.5. Thus, Eq. 4.2 becomes

$$\mathtt{l}^T\mathbf{C}^*\mathtt{l} = \mathtt{m}_1^T\mathbf{C}\mathbf{C}^*\mathbf{C}\mathtt{m}_1 = 0$$

for points $\mathtt{m}_1$ on the conic. Hence

$$\mathbf{C}^* \simeq \mathbf{C}^{-1}$$

The transformation of a conic under a general projective transformation $\mathbf{H}$ can simply be computed by transforming the points in Eq. 4.1 with $\mathtt{m}' = \mathbf{H}\mathtt{m}$. This yields:

$$\begin{aligned} \mathtt{m}'^T\mathbf{C}'\mathtt{m}' &= 0 \\ \mathtt{m}^T\mathbf{H}^T\mathbf{C}'\mathbf{H}\mathtt{m} &= 0 \end{aligned}$$

and thus

$$\begin{aligned} \mathbf{C}' &\simeq \mathbf{H}^{-T}\mathbf{C}\mathbf{H}^{-1} \\ \mathbf{C}^{*\prime} &\simeq \mathbf{H}\mathbf{C}^*\mathbf{H}^T \end{aligned}$$

Quadrics

Where conics represent quadratic expressions of points in $\Re^2$, their extensions to $\Re^3$ are called *quadrics*. Analogue definitions can be given for them, this time using 4×4 symmetric matrices with 9 independent parameters:

$$\mathtt{M}^T\mathbf{Q}\mathtt{M} = 0 \tag{4.6}$$

The dual of a quadric is now defined as the set of planes tangent to the quadric

$$\Pi^T\mathbf{Q}^*\Pi = 0 \tag{4.7}$$

The same derivation can be made here to show that a quadric and its dual are each other's inverse

$$\mathbf{Q}^* = \mathbf{Q}^{-1}$$

Transformations on quadrics with a general projective transformation $\mathbf{T}$ are derived in the same way as for conics which yields:

$$\begin{aligned} \mathbf{Q}' &\simeq \mathbf{T}^{-T}\mathbf{Q}\mathbf{T}^{-1} \\ \mathbf{Q}^{*\prime} &\simeq \mathbf{T}\mathbf{Q}^*\mathbf{T}^T \end{aligned}$$

The projection of a dual quadric into an image, described by projection matrix $\mathbf{P}$ delivers a dual conic. We derive the form of this dual conic as follows. Suppose the dual conic is called $\mathbf{C}^*$, thus

$$\mathtt{l}^T\mathbf{C}^*\mathtt{l} = 0$$

All 3D points $\mathtt{M}$ are projected onto image points $\mathtt{m}$ via the equation

$$\mathtt{m} \simeq \mathbf{P}\mathtt{M}$$

The equation of the points on a line $\mathtt{l}$ in the image is given by

$$\mathtt{l}^T\mathtt{m} = 0$$

With the projection equation, this becomes

$$\mathtt{l}^T\mathbf{P}\mathtt{M} = 0$$

We can interpret this equation differently: it says that all points $\mathtt{M}$ in the plane Π are projected on the line $\mathtt{l}$ if this plane is given by

$$\Pi \simeq \mathbf{P}^T\mathtt{l}$$

If $\mathbf{C}^*$ is to be the projection of the dual quadric $\mathbf{Q}^*$, then the planes Π must satisfy Eq. 4.7 and thus it follows that

$$\mathtt{l}^T\mathbf{P}\mathbf{Q}^*\mathbf{P}^T\mathtt{l}$$

In other words the projection of the dual quadric $\mathbf{Q}^*$ yields the dual conic $\mathbf{C}^*$ if

$$\mathbf{C}^* \simeq \mathbf{P}\mathbf{Q}^*\mathbf{P}^T \tag{4.8}$$

4.3.2 The Absolute Conic

Now we know how conics and quadrics are defined, we can employ them in the self-calibration process. As explained before we want to impose constraints on the intrinsic parameters of the cameras. Unfortunately, the projection matrices we have recovered are only valid up to an arbitrary projection matrix, making it impossible to separate the intrinsics from the extrinsics with the QR-technique described in paragraph 1.3.2.

In literature, the *absolute conic* was introduced because of its specific form. The *absolute conic* is a conic which resides in the plane at infinity. For the sake of simple notation, we typically employ its dual, the *dual absolute quadric* (DAQ) which we denote with the symbol $\mathbf{\Omega}^*$. In Euclidean space, this DAQ is defined as

$$\mathbf{\Omega}^* = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \tag{4.9}$$

Projecting this dual quadric into an image with a Euclidean pinhole camera, as defined in Eq. 1.5, yields the *image of the dual absolute quadric* ω^* which is computed with Eq. 4.8 and the fact that a rotation matrix is orthonormal

$$\begin{aligned} \omega^* &= \mathbf{P}\mathbf{\Omega}^*\mathbf{P}^T & (4.10)\\ &\simeq \mathbf{K}\left(\mathbf{R}^T|-\mathbf{R}^T\mathbf{t}\right)\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}\left(\mathbf{R}^T|-\mathbf{R}^T\mathbf{t}\right)^T\mathbf{K}^T \\ &\simeq \mathbf{K}\mathbf{R}\mathbf{R}^T\mathbf{K}^T \\ &\simeq \mathbf{K}\mathbf{K}^T & (4.11) \end{aligned}$$

In other words, the projection of the dual absolute quadric is described by parameters of the internal calibration matrix of the projection matrices only.

In reality, the projection matrices we have recovered with projective structure and motion are only valid up to an arbitrary projective transformation $\mathbf{T}$ and so is the recovered 3D structure of this space. Hence, the DAQ is not in its canonical position given by Eq. 4.9 but in a transformed position

$$\mathbf{\Omega}^{*\prime} \simeq \mathbf{T}\mathbf{\Omega}^*\mathbf{T}^T \tag{4.12}$$

However, since the images were taken in the real world, the projection of the dual absolute quadric $\mathbf{P}\mathbf{\Omega}^{*\prime}\mathbf{P}^T$ is still in the same position given by Eq. 4.11 because $\mathbf{P}' = \mathbf{P}\mathbf{T}^{-1}$

$$\begin{aligned}
\omega^{*\prime} &\simeq \mathbf{P}'\mathbf{\Omega}^{*\prime}\mathbf{P}'^T \\
&\simeq (\mathbf{P}\mathbf{T}^{-1})(\mathbf{T}\mathbf{\Omega}^{*\prime}\mathbf{T}^T)(\mathbf{T}^{-T}\mathbf{P}^T) \\
&\simeq \mathbf{P}(\mathbf{T}^{-1}\mathbf{T})\mathbf{\Omega}^{*\prime}(\mathbf{T}^T\mathbf{T}^{-T})\mathbf{P}^T \\
&\simeq \mathbf{P}\mathbf{\Omega}^*\mathbf{P}^T \\
&\simeq \omega^*
\end{aligned}$$

In other words, we will employ the following steps in our self-calibration approach.

- Select a set of constraints on the internal parameters $\mathbf{K}$ of the camera.
- Translate these constraints toto constraints on $\mathbf{\Omega}^*$ itself via Eq. 4.10.
- If enough constraints are given, $\mathbf{\Omega}^*$ (9 linearly independent elements) can be computed.
- When $\mathbf{\Omega}^*$ is known, the transformation$\mathbf{T}$ between it and the canonical position of Eq. 4.9 is readily computed from Eq. 4.12.
- The inverse of this transformation is exactly the transformation that brings our projective reconstruction into metric space!

4.3.3 Practical Constraints

The algorithm described above allows us to translate constraints on the intrinsic parameters to constraints on the DAQ and in this way to compute the transformation that brings us to metric space. The question now is which constraints are suitable and how formulate them, so the DAQ is easy to compute. Because of the stability and efficiency of SVD-based methods, we would like to impose constraints that lead to linear equations in the elements of $\mathbf{\Omega}^*$. How this can be done is explained in this paragraph.

We begin with normalizing the projection matrices such that the focal length is in the order of 1 and the principal point is close to the origin. We obviously don't know the focal length yet, so the following normalization is proposed:

$$\mathbf{P}_N = \mathbf{K}_N^{-1}\mathbf{P}$$

with

$$\mathbf{K}_N = \begin{bmatrix} w+h & 0 & \frac{w}{2} \\ 0 & w+h & \frac{h}{2} \\ 0 & 0 & 1 \end{bmatrix}$$

with w and h the width and height of the image. The principal point $(u,\ v)$ typically lies near the center of the image, so the normalization is OK in that respect. The focal length (f) deserves some more explanation. If we approximate it with $w+h$, then this corresponds to a camera with a (35mm equivalent) focal length of 63mm, or with an opening angle of about 30 degrees. If we allow for a deviation of a factor 3, then cameras with focal lengths from 20 to 180mm (82 to 11 degrees) are OK. This covers the range of most used lenses. The aspect ratio (r) is taken to be around 1 and the skew (s) is 0, which is true for all digital cameras. Making these a priori knowledge more explicit and estimating reasonable standard deviations one could for example get $f \approx rf \approx 1 \pm 3$, $u \approx v \approx 0 \pm 0.1$, $r \approx 1 \pm 0.1$ and $s = 0$. It is now interesting to investigate the impact of this knowledge on ω^*:

$$\omega^* \simeq \mathbf{K}\mathbf{K}^T = \begin{bmatrix} f^2+s^2+u^2 & srf+uv & u \\ srf+uv & r^2f^2+v^2 & v \\ u & v & 1 \end{bmatrix} \approx \begin{bmatrix} 1\pm 9 & \pm 0.01 & \pm 0.1 \\ \pm 0.01 & 1\pm 9 & \pm 0.1 \\ \pm 0.1 & \pm 0.1 & 1 \end{bmatrix}$$

Comparison of elements of $\mathbf{K}\mathbf{K}^T$ gives us several possible constraints: elements (1,1) and (2,2) should be equal to element (3,3) (focal length = 1) and should be equal to each other (aspect ratio = 1). Elements (1,2) and (1,3) should be zero (principle point in origin), as well as element (2,3) (skew = 0). Since these are constraints on ω^*, we can translate them to constraints on $\mathbf{\Omega}^*$ via Eq. 4.10. The equations should be weighted according to the uncertainty on the result. This gives the following weighted equations:

$$\begin{aligned}
\frac{1}{9\nu}\left(P_1\mathbf{\Omega}^*P_1^T - P_3\mathbf{\Omega}^*P_3^T\right) &= 0 \\
\frac{1}{9\nu}\left(P_2\mathbf{\Omega}^*P_2^T - P_3\mathbf{\Omega}^*P_3^T\right) &= 0 \\
\frac{1}{0.2\nu}\left(P_1\mathbf{\Omega}^*P_1^T - P_2\mathbf{\Omega}^*P_2^T\right) &= 0 \\
\frac{1}{0.1\nu}\left(P_1\mathbf{\Omega}^*P_2^T\right) &= 0 \\
\frac{1}{0.1\nu}\left(P_1\mathbf{\Omega}^*P_3^T\right) &= 0 \\
\frac{1}{0.01\nu}\left(P_2\mathbf{\Omega}^*P_3^T\right) &= 0
\end{aligned} \tag{4.13}$$

with P_i the ith row of $\mathbf{P}$ and ν a scale factor initially set to 1 and in later iterations to $P_3\mathbf{\Omega}^*P_3$.

An estimate of $\mathbf{\Omega}^*$ can be obtained by solving the above set of equations for all views through linear least-squares. The rank-3 constraint should be imposed by forcing the smallest singular value to zero. This scheme can be iterated until the factors converge.

4.3.4 Coupled Self-Calibration

We will see in later chapters that sometimes different (sub)sequences have been recorded. One could of course self-calibrate the different subsequences separately but if they were recorded with the same camera and the same settings, it is possible to combine the equations derived above into a larger system of equations. Each subsequence has been reconstructed into each own projective frame, thus the $\mathbf{\Omega}^*$ for each subsequence is different. However the fact that the intrinsic parameters of the cameras were the same allows us to solve for the different $\mathbf{\Omega}^*$ while retaining a coupling between the subsequences.

Every subsequence can first be transformed thus that the first camera $\mathbf{P} = [\mathbf{I}\|\mathbf{0}]$. From Eq. 4.10, 4.11 we see that we can write

$$\mathbf{\Omega}^* = \begin{bmatrix} \mathbf{K}\mathbf{K}^T & \mathbf{a} \\ \mathbf{a} & b \end{bmatrix}$$

and thus the equations of Eq. 4.13 can be written in matrix form as

$$\begin{bmatrix} \mathbf{C} & \mathbf{D} \end{bmatrix} \begin{bmatrix} \mathbf{k} \\ \begin{bmatrix} \mathbf{a} \\ b \end{bmatrix} \end{bmatrix}$$

with $\mathbf{k}$ a vector containing 6 coefficients representing the matrix $\mathbf{K}\mathbf{K}^T$, a is a 3-vector and b a scalar and $\mathbf{C}$ and $\mathbf{D}$ are matrices containing the coefficients of the equations. Note that this can be done independently for every 3D subsequence.

If the sequence is recorded with constant intrinsics, the vector $\mathbf{k}$ will be common to all subse-

quences and one obtains the following coupled self-calibration equations:

$$\begin{bmatrix} \mathbf{C_1} & \mathbf{D_1} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{C_2} & \mathbf{0} & \mathbf{D_2} & \dots & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \mathbf{C_n} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{C_n} \end{bmatrix} \begin{bmatrix} \mathbf{k} \\ \begin{bmatrix} \mathbf{a}_1 \\ b_1 \end{bmatrix} \\ \begin{bmatrix} \mathbf{a}_2 \\ b_2 \end{bmatrix} \\ \vdots \\ \begin{bmatrix} \mathbf{a}_n \\ b_n \end{bmatrix} \end{bmatrix}$$

The most important feature is that through the coupling this approach solves for the intrinsics using all information together which allows to get good results even for the shorter subsequences. When $\mathbf{\Omega}^*$ has been computed for every subsequence, they can each be upgraded to metric by computation of the transformation that brings it to its canonical position.

Chapter 5

Model Selection

5.1 Introduction

At this point we are capable of reconstructing 3D points and cameras from images only. The approach we follow starts with establishing pairwise relations between images by means of the fundamental matrix, then moves on to the computation of a projective reconstruction, and finally an upgrades the result to metric. Unfortunately, it happens every so often that so-called critical motions of the camera or critical surfaces in the scene are encountered during recording. Often occuring examples are linear motions or planar surfaces, respectively. These situations will cause our algorithms to fail and we need to be able to deal with them. In [27] the general problem of critical surfaces is described. Different classes of critical motion sequences that cause problems for self-calibration are described in [58, 59, 60, 28].

First we will describe the problems we encounter when dealing with planar scenes. Then we will introduce the concept of the essential matrix and we will explain how to compute it.

5.2 Problems with Planar Scenes

The most common critical surface we encounter in SaM is a planar scene. It might come as a surprise that such a simple scene (a plane) can cause problems for our algorithms, but mathematically it can easily be understood. In paragraph 1.4 we described the transformations a planar object can undergo. Table 1.1 showed an overview of them. It was explained that a general projective transformation can be explained by a so-called planar homography. This means that the transformation from one image to another can be modeled by a 3×3 matrix for which the mathematical expression was given by equation 1.8. Since this matrix is only determined up to scale, an homography has 8 degrees of freedom (DOF).

Let us now inspect the case of the computation of a fundamental matrix $\mathbf{F}$. If only a plane is visible in both images, all point correspondences can be modeled via an homography $\mathbf{H}$. It was explained in paragraph 2.2.6 that, since every point correspondence yields 2 equations (one in x and one in y), only 4 points are needed to compute $\mathbf{H}$. All other correspondences can be predicted from these 4. This would suggest that only 4 independent point matches can be used to compute $\mathbf{F}$ but it was seen in paragraph 2.2.3 that we need at least 7 correspondences. This means that there are 3 DOF in $\mathbf{F}$ we cannot determine without information of points outside the plane. If no such points are available, these remaining DOF are filled in by noise on the extracted features. Fig. 5.1 shows an example of the Spijker in Mechelen we already encountered in paragraph 4.1. Two images of the wall are shown in which clearly only a planar part is visible. The top and bottom image-pair depict the same images but with a different fundamental matrix. Both matrices were computed with the same algorithm, described in table 2.2.4. The only difference is a small perturbation on one of the parameters (i.e. the outlier distance which was changed from 2 to 2.2 pixels). The resulting F-matrices are quite different which is apparent from the epipolar lines. If one compares

the epipolar lines in any of the image pairs however, points on corresponding lines are always in correspondence. This means that both solutions are equally valid and no claim can be made as to the quality of the result without information outside the plane.

Also projective pose estimation is problematic for precisely the same reasons. In paragraph 3.2.2 we saw that, since a projection matrix $\mathbf{P}$ has 11 degrees of freedom (3×4 minus one for scale), and every 3D-2D correspondence yields two equations, we need at least 6 points to compute $\mathbf{P}$ (actually we only need 5 plus one equation in either x or y of a sixth). If all points are on a plane, however, the correspondences can be predicted if the homography is known, thus if 4 correspondences are given. So again, there are 11-8=3 DOF left in $\mathbf{P}$ which we cannot determine if we only see a plane.

5.3 Detecting Planar Scenes

A first step in solving the problem with planar scenes is the detection of this situation. Since images of such a scene can be transformed into eachother via a homography, one could compute such an $\mathbf{H}$ matrix and inspect the residual error. If this error is below a threshold, one could consider the scene to be planar. However, it is unclear which value one should choose for this threshold. A better and more stable approach is to take into account not only the residual error but also the degrees of freedom. This can be done by computing and comparing GRIC values for different models. This will be explained in the following paragraphs.

5.3.1 Occam's Razor

The problem of determining whether a scene is planar or not, based on correspondences between images is a specific example of a more general problem which can be stated as follows. If one has a data-set and different models that can explain the data, then which model should one choose? Already in the middle ages a Franciscan friar called William of Ockham [4] shed some light on this. His theory, known as *Occam's Razor* states

> One should not increase, beyond what is necessary, the number of entities required to explain anything.

In essence this theory means that if two or more theories explain the observations equally well, one should always choose the simplest one.

5.3.2 GRIC

For the case of multiple view geometry, Torr proposed a mathematical description of this principle that is completely general [61]. It calculates a score function for each model taking into account the number of inliers, outliers, the residuals, the standard deviation of the error, the dimensionality of the data, the number of the parameters and dimensionality of the model. The general formula is given by

$$GRIC = \sum \rho(e_i^2) + (nd\ln(r) + k\ln(rn)) \tag{5.1}$$

where

$$\rho(e_i^2) = min\left(\frac{e^2}{\sigma^2}, 2(r-d)\right) \tag{5.2}$$

The different elements in the formula are explained in the following table.

- n is the number of data (inliers + outliers)
- e_i is the residual for every correspondence i. For an homography $\mathbf{H}$ for instance this is the distance between point $\mathtt{m}_2$ and the transformed corresponding point $\mathbf{H}\mathtt{m}_1$.

$$e_i = D(\mathbf{H}\mathtt{m}_1, \mathtt{m}_2)$$



Figure 5.1: An image pair depicting a planar scene. The top and bottom pair show epipolar lines of the resulting F-matrices computed with the same algorithm but with slightly different parameters. The influence of noise on the result is apparent. Although the results are clearly different, both are equally valid which can be verified by comparing points on correspondending epipolar lines.

For a fundamental matrix $\mathbf{F}$ e_i is given by the sum of the distances to the epipolar lines

$$e_i = D(\mathtt{l}_2 \simeq \mathbf{F}\mathtt{m}_1, \mathtt{m}_2) + D(\mathtt{m}_1, \mathtt{l}_1 \simeq \mathbf{F}^T\mathtt{m}_2)$$

- σ is the standard deviation on the measurement error.
- r is the dimensionality of the data or the amount of observations. In the case of 2 views, r is 4 (two times two coordinates).
- k is the number of model parameters. For a homography k is 8, for a fundamental matrix it is 7.
- d is the dimensionality of the structure. For a homography d is 2, for a fundamental matrix it is 3.

Inspecting Eq. 5.1 we clearly identify two parts. The first part is the goodness of the fit. How well does a model describe the data? This is given by the residual error of the corresponding matches. Of course, this residual alone does not suffice since a complicated model that contains a simpler model always explains the data better since it has more degrees of freedom. In the case of a planar surface for instance, the 3 remaining degrees of freedom of a computed fundamental matrix are estimated from noise in the data, allowing for a lower residual score.

That is why the GRIC model has a second part $(nd\ln(r) + k\ln(rn))$ which takes into account the so called *parsimony* of the model. This means that the more complex a model is, the higher this penalty term will become. If we want to choose a more complex model over a simpler one, it should explain the data a lot better, to the extent that its residual error should be so much lower than the residual error of the simpler model that this reduction nullifies the extra cost associated with the higher complexity.

Let us apply this algorithm to the problem of a planar surface visible in a pair of images. We execute the following algorithm.

- Compute the fundamental matrix between the two images, as well as the inliers and outliers using F-RANSAC 2.2.4.
- Compute a non-linear optimization of $\mathbf{F}$ using the inliers only, e.g. with a MLE (Maximum Likelihood Estimator) [1, 24].
- Compute a planar homography matrix $\mathbf{H}$ using all the inliers of $\mathbf{F}$. The latter is absolutely necessary because if only inliers for $\mathbf{H}$ are used to compute it, the H-GRIC value will always be lower than the F-GRIC because of the lower complexity.
- Compute H-GRIC and F-GRIC with Eq. 5.1.
- Compare both GRIC values. If H-GRIC is lower than F-GRIC, then the preferred model to explain the data should be $\mathbf{H}$. This can mean that either the observed scene is planar or that the camera underwent a pure rotation around its own camera center. If the F-GRIC value is lower, then enough 3D information was visible in the scene to compute the F-matrix correctly.

5.4 More GRICs

As explained before, not only the computation of epipolar geometry but also projective pose estimation suffers from dominant planar surfaces. One might hope that the comparison of F-GRICs and H-GRICs tells us when enough 3D information is visible in the scene and when this is not the case. However, observe the situation depicted in Fig. 5.2. Three cameras are looking at a scene with plenty of 3D information. The surface between points 1 and 4 is visible in cameras 1 and 2. The surface between points 3 and 6 is visible in cameras 2 and 3. This means that both

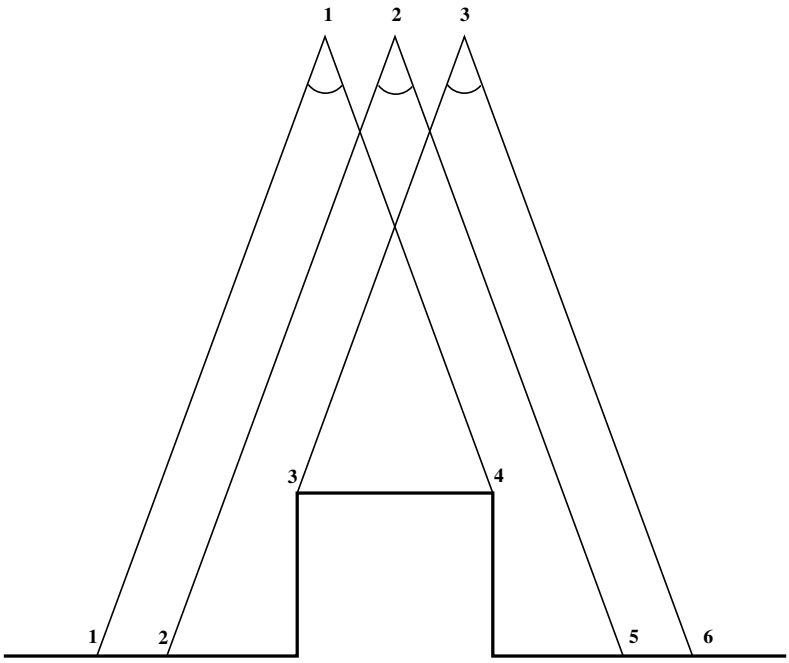


Figure 5.2: The F-GRIC of camera pairs 1-2 and 2-3 is lower than the H-GRIC due to enough 3D information visible in the cameras. However only the planar part 3,4 is seen in common in the three cameras which makes unique projective reconstruction of the cameras impossible.

F-GRICS will yield a lower penalty than their corresponding H-GRIC. However, when looking at the camera triplet, only the surface part between points 3 and 4 is seen in all views. This part is planar and hence the projective reconstruction of the cameras cannot be uniquely determined.

If we want to find out whether there is enough 3D information in the scene for projective reconstruction, we have to take into account the common information in 3 views. This can be done by comparing the HH-GRIC and the PPP-GRIC of such a triplet. This is done as follows.

- Compute a projective reconstruction of the triplet, e.g. by computing epipolar geometry between two images; instantiating the projective frame from these and then performing a projective pose-estimation of the third camera. This also yields a set of triplet-correspondences: features that are matched throughout all three images.

- Compute the best-fitting planar homography between image 1 and 2 and between image 2 and 3, using the same triplet correspondences.

- Compute the HH-GRIC and PPP-GRIC from these results. Table 5.1 gives an overview of the value of the different parameters in the GRIC formula 5.1. The number of model parameters for PPP-GRIC is $3 \times 11 - 15$ which comes from 3 projection matrices with 11 parameters each but up to a general projective transformation with 15 parameters.

- If the resulting HH-GRIC is lower than the PPP-GRIC then only a planar surface is seen in common in the three images and no projective reconstruction can be computed.

A concrete example of the situation of Fig. 5.2 is shown in Fig. 5.3. A triplet of images shot at ESAT depicts a wall with an extrusion. When one observes the images, it is clear that this planar extrusion is the only part common to the three images. The two consecutive image pairs 1-2 and 2-3 have a larger part in common which yields enough 3D scene to compute the epipolar geometry robustly as can be seen in Fig. 5.4 for the first pair. The resulting GRICs are gathered in table 5.2. Since the value in the first column, denoting the F-GRIC is lower than the H-GRIC in the second

	HH-GRIC	**PPP-GRIC**
r	6	6
k	$2 \times 8 = 16$	$3 \times 11 - 15 = 18$
d	2	3

Table 5.1: Parameters in the GRIC formula 5.1 for HH-GRIC and PPP-GRIC

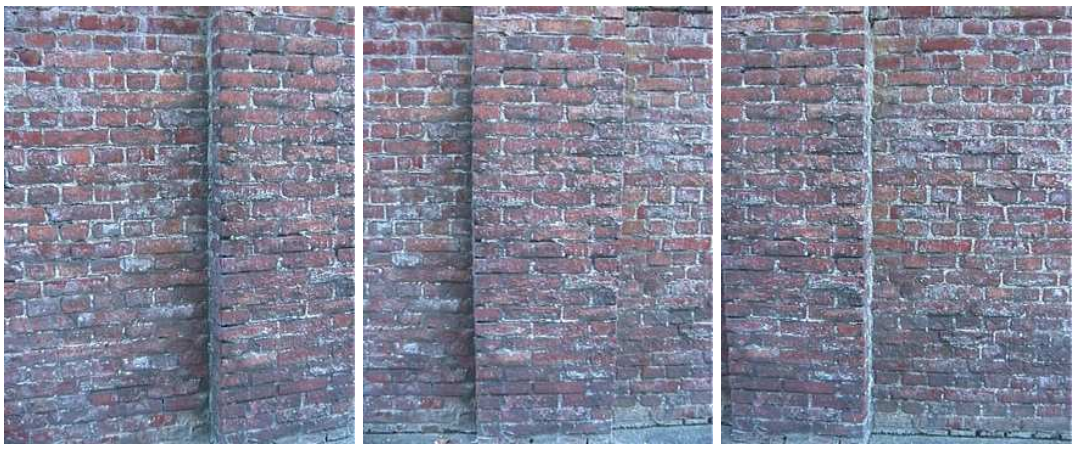


Figure 5.3: Triplet of images with a concrete example of the situation of Fig. 5.2. The common part between the three images is the planar extrusion.

column for both image pairs, the epipolar geometry can be estimated correctly. When we compute the projective projection matrices, however, we see that the inliers are only located on the planar extrusion as can be seen in Fig. 5.5. Due to the planar common part, the PPP-GRIC is higher than the HH-GRIC which means that robust estimation of the projective projection matrices of these images is not possible.

5.5 Long Video Sequences

The previous discussion about model selection and the usage of GRICs can play its role in the computation of the structure and motion of long video sequences as will be explained below. Video sequences of 1000 frames or more require specific algorithms and techniques if one wants to retrieve the camera calibration and 3D scene reconstruction from them. Two important differences can be seen which discern video sequences from sequences of images taken with a still camera. First, video frames are typically much closer together and second, videos of a thousand frames or more yield many more images to process than a normal image sequence. These two observations have their impact on the algorithms.

	F-GRIC	**H-GRIC**	**PPP-GRIC**	**HH-GRIC**
pair 1-2	2332.8	3711.83		
pair 2-3	1424.03	2247.66		
triplet			1435.908	843.01

Table 5.2: The F-GRIC and H-GRIC for the two image pairs show us that fundamental matrix computation is possible. The planar common part however leads to a lower HH-GRIC than PPP-GRIC value which tells us that projective reconstruction of this triplet is not possible.

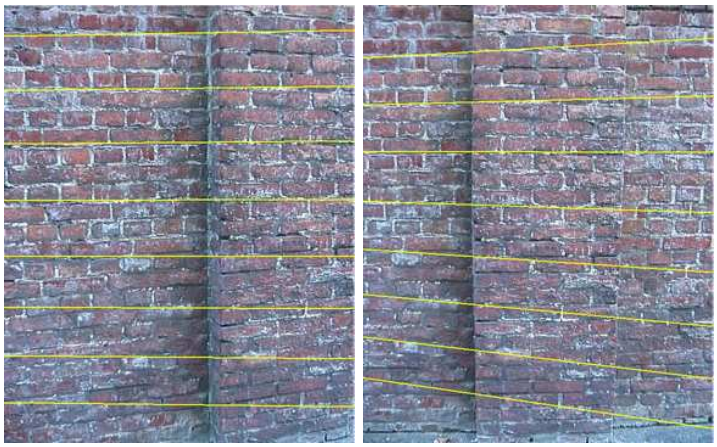


Figure 5.4: Epipolar geometry for the first image pair. Enough 3D information is seen in common to compute the fundamental matrix robustly.



Figure 5.5: Inliers for projective projection matrix estimation. These inliers are clearly only located on the planar extrusion.

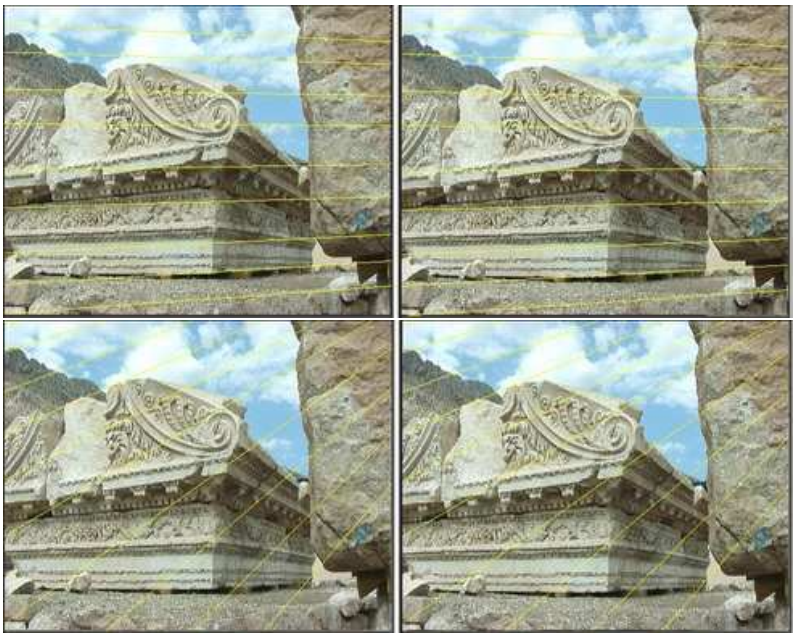


Figure 5.6: Two consecutive frames of a video sequences are so close together that the fundamental matrix cannot be uniquely determined.

5.5.1 Video Frames

Video recordings yield image sequences with frame rates of 25 (PAL) or 30 (NTSC) frames per second. If the video camera does not move at very high speed (which would cause blurring effects anyway, more on which in paragraph 5.5.4) this means that consecutive video frames are typically very close together and thus very much alike. This give us a huge advantage for the computation of matching points between images because such points are typically located close to the coordinates in the previous image and very few distortion has to be dealt with. This makes it possible to *track* features rather than match them. In section 2.1.2 we described how to find good features to track with the formula of KLT-features [51].

Unfortunately, the fact that consecutive images are so similar also has its drawbacks. Inspect Fig. 5.6. This figure shows two consecutive frames of a video sequence of the roof Anthonine Nymphaeum in Sagalassos, Turkey. If we attempt to compute the epipolar geometry between these images we notice that, just like in the case with planar scenes, this process is very sensitive to noise. Two possible solutions of the epipolar geometry are shown in Fig. 5.6 and both are equally valid.

The images shown in Fig. 5.6 depict a scene with plenty of 3D information. If that is the case, then why can't we compute the epipolar geometry stably? This answer lies in the closeness of the two camera centra. Because the two consecutive frames have been recorded very closely together, the translation between the cameras is very small. This means that the camera setup resembles that of a purely rotating camera. If a camera makes such a pure rotational movement, the relation between corresponding pixels can also be explained by an homography as shown in Fig. 5.7.

Mathematically the fact that a rotation gives rise to an homography is easily understood as follows. The 3D point $\mathtt{M}$ is projected in the two images to yield 2D points $\mathtt{m}_1$ and $\mathtt{m}_2$. The translation vector of both cameras $\mathbf{t}$ is the same.

$$\begin{aligned} \mathtt{m}_1 &\simeq \mathbf{K}_1[\mathbf{R}_1^T|-\mathbf{R}_1^T\mathbf{t}]\mathtt{M} \\ \mathtt{m}_2 &\simeq \mathbf{K}_2[\mathbf{R}_2^T|-\mathbf{R}_2^T\mathbf{t}]\mathtt{M} \end{aligned}$$

If we translate the origin of the world coordinate system to this position $\mathbf{t}$, the point $\mathtt{M}$ moves to

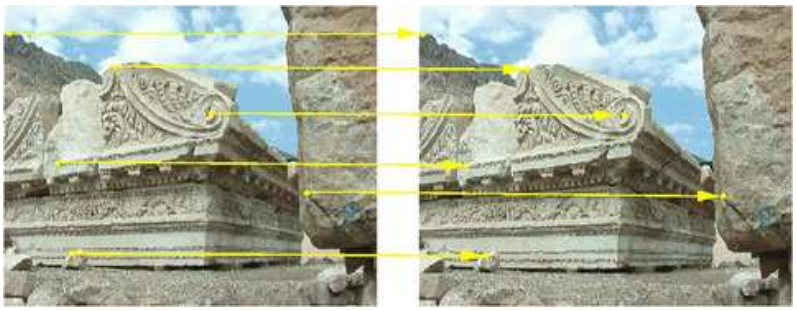


Figure 5.7: Two consecutive frames of a video sequences are so close together that the motion resembles that of a pure rotation. This means that corresponding points can be related by an homography.

$\mathtt{M}'$ and the projection equations become

$$\begin{aligned} \mathtt{m}_1 &\simeq \mathbf{K}_1[\mathbf{R}_1^T|0]\mathtt{M}' \\ \mathtt{m}_2 &\simeq \mathbf{K}_2[\mathbf{R}_2^T|0]\mathtt{M}' \end{aligned}$$

Defining $\mathtt{M}''$ as the 3-vector holding the first three homogeneous coordinates of $\mathtt{M}'$, we get

$$\begin{aligned} \mathtt{m}_1 &\simeq \mathbf{K}_1\mathbf{R}_1^T\mathtt{M}'' \\ \mathtt{m}_2 &\simeq \mathbf{K}_2\mathbf{R}_2^T\mathtt{M}'' \end{aligned}$$

and thus

$$\begin{aligned} \mathtt{m}_2 &\simeq \mathbf{K}_2\mathbf{R}_2^T\left(\mathbf{K}_1\mathbf{R}_1^T\right)^{-1}\mathtt{m}_1 \\ \mathtt{m}_2 &\simeq \mathbf{K}_2\mathbf{R}_2^T\mathbf{R}_1^{-T}\mathbf{K}_1^{-1}\mathtt{m}_1 \end{aligned} \tag{5.3}$$

which is clearly the formula of an homography with the homography matrix $\mathbf{H}$ defined as

$$\mathbf{H} \simeq \mathbf{K}_2\mathbf{R}_2^T\mathbf{R}_1^{-T}\mathbf{K}_1^{-1} \tag{5.4}$$

The computation of epipolar geometry breaks down if images are too close together. The same is true for projective camera estimation where newly computed 3D points and cameras suffer from large uncertainties due to the small baseline between cameras. This means we have to find a technique to increase the baseline between images that are used for structure-and-motion recovery. A possible solution is not to deal with consecutive images but to select every n-th frame of the video sequence. This is not a good idea because it is impossible to choose a value of for n that will work for all video sequences. It would depend on the velocity of the camera, the frame rate and the distance to the scene. These properties could even change during the recording of the video. Imagine a camera filming a scene, standing still every once in a while and then continuing. The value of n would have to be adapted constantly.

We can however make use of the GRIC-concept, explained above. In order to do so we introduce the concept of *keyframes*. We select the first frame in the sequence as the first keyframe and track features to the second frame. We estimate the fundamental matrix and an homography between these frames and compute the corresponding F-GRIC and H-GRIC values. If the H-GRIC is lower than F-GRIC, the homography describes the data better than F and we track the features to the next frame in the sequence where we perform the same process. As long as the H-GRIC is lower, we continue to advance in the sequence. Once the F-GRIC yields the lowest penalty, the epipolar geometry can be reliably estimated and we could select the frame where this happens as the new keyframe. However, in order to reduce the processing time on the structure and motion estimation, we progress a little further in the sequence. For every frame we track the features and estimate epipolar geometry w.r.t. the previous keyframe. While more than 90% of the inliers at the crossing-point are still matched, we continue. When we drop under this 90%, we select this frame as the new keyframe. Figure 5.8 explains the process.

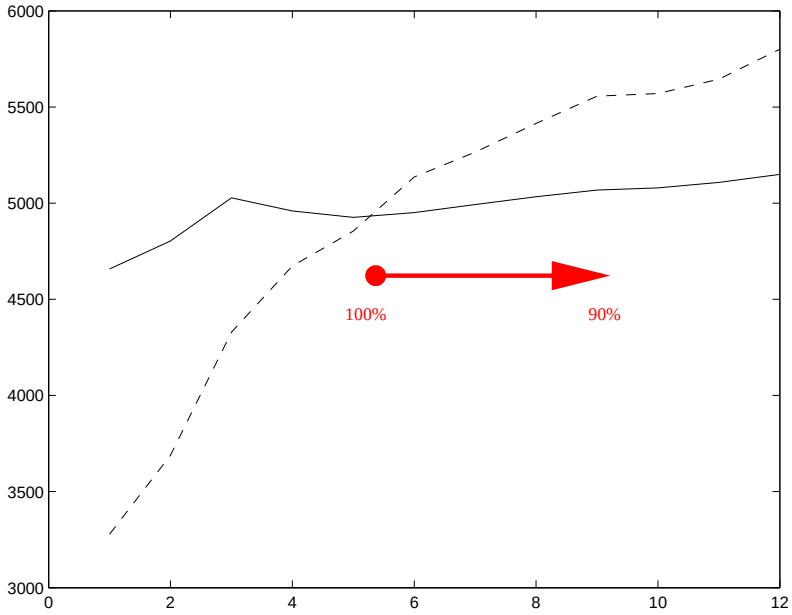


Figure 5.8: While the F-GRIC (dashed line) is higher than the H-GRIC (full line) continue traversing in the sequence. At the cross-over, store the amount of inliers for F. Continue traversing in the sequence while the amount of inliers is higher than 90% of the stored value. When it drops below 90%, select that image as the new keyframe.

5.5.2 Long Sequences and Subsequences

The technique of selecting keyframes in a video sequence combines the best of two worlds. It allows tracking features instead of matching them by exploiting closeness of consecutive video frames. Due to degeneracy of close frames for structure and motion estimation, only a subset of frames is selected. Typically every 5th frame is selected for computation which also makes this problem 5 times less computation- and memory-intensive. Even so, due to accumulation of measurement errors, long video sequences have problems with drift [10]. The drift on the projective reconstruction can become quite substantial. This causes problems for the self-calibration tehchniques explained in paragraph 4.3. If the projective projection matrices at the input of the algorithm are not in the same projective coordinate system, the self-calibration techniques will fail because the pose of the DAQ cannot be found consistently in all images. To alleviate this problem we will split the long video sequence into subsequences.

Subsequences

In order to alleviate drift on the projective reconstruction, we split the original long video sequence into shorter subsequences. Typically these subsequences have a length of about 100 frames. With a mean value of one keyframe per 5 frames this yields 20 images per sequence. Every subsequence is treated as a separate image sequence in that respect that the computation of keyframes, fundamental matrices and projective projection matrices is done independently for every subsequence. The tracking of features however is done before on the original long sequence to ensure that the same features are used in all subsequences.

Subsequences with shorter length suffer less from drift and typically self-calibration is possible for all of them. However, since all subsequences are part of a global sequence, recorded with a single camera, the intrinsic parameters of the camera do not change. This means the coupled self-calibration approach of section 4.3.4 can be used to retrieve these intrinsics. The result of this process is the pose of the DAQ and hence a metric reconstruction of every subsequence

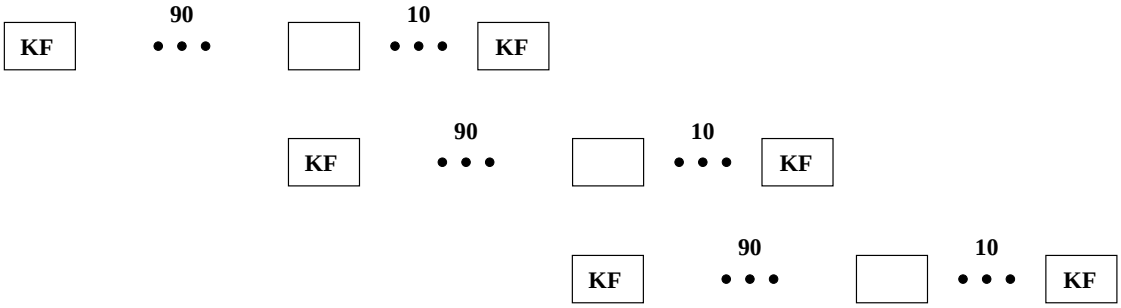


Figure 5.9: The original long video sequence is split in subsequences of 100 frames each. The overlap between subsequences is 10 frames. The first and last frame of every subsequence is always picked as keyframe.

separately. This means that every subsequence is reconstructed in its own metric frame. If we want to put the subsequences together into the same coordinate system, we have to compute the metric transformation (or similarity transformation, see 1.5) between all these subsequences, a process we call *gluing* them together.

Gluing Subsequences

The coupled self-calibration finds the intrinsic parameters of the camera by enforcing the fact that the same camera is used in the different subsequences. It cannot enforce a similar constraint on the extrinsic parameters which means that all subsequences are reconstructed in their own metric coordinate system. In order to glue the subsequences together, we make sure that a certain overlap exists between consecutive subsequences. We typically take an overlap of 10 frames. Fig 5.9 shows this schematically. As explained before, the first frame of a sequence is always picked as a keyframe. Additionally, we always include the last frame of a subsequence as a keyframe as well.

The setup of Fig 5.9 ensures that we have overlap between subsequences which will allow gluing them together. The algorithm we employ is illustrated in Fig. 5.10 and works as follows. We consider two subsequences with an overlap of 10 images. For every subsequence, the keyframes have been reconstructed into its own metric coordinate system. This means metric cameras, 3D points and their corresponding 2D points in the images. Because of the overlap of 10 images and the fact that the feature tracking was done on the entire video sequence, these 2D feature points are the same in the overlapping images. Let us regard a 3D point in the first subsequence which projects onto a 2D point in a keyframe which is part of the overlapping 10 images. The same image is also part of the second subsequence but is not necessarily a keyframe. The 2D feature is found on the same position and might have been tracked further in this second subsequence. We follow the track until we reach the next keyframe in this subsequence. If the tracked 2D feature is part of the 3D reconstruction of this subsequence, then the original 3D point and the newly found 3D point is a possible 3D-3D correspondence between the two reconstructed subsequences.

The approach explained above gives us a set of possible 3D-3D correspondences which we will use to compute the metric transformation between two consecutive subsequences. Not all correspondences are inliers however, which means we have to resort to the robust RANSAC technique again to retrieve this transformation. The metric transformation consists of 7 parameters, 3 for rotation, 3 for translation and 1 for scale. Every 3D-3D correspondence yields 3 equations (the X, Y and Z coordinates should be identical after transformation), which means the minimum sample size is 3. Unfortunately, retrieving the 3 rotation parameters of the transformation requires a non-linear solution method. We propose the following approach to solve for the transformation linearly.

A metric transformation is described by 7 non-linear parameters, yielding a 4×4 matrix with 13 non-zero elements. We know that 3 3D-3D correspondences are enough to compute this matrix but since every correspondence gives rise to 3 equations, we can only write down 9 linear equations.

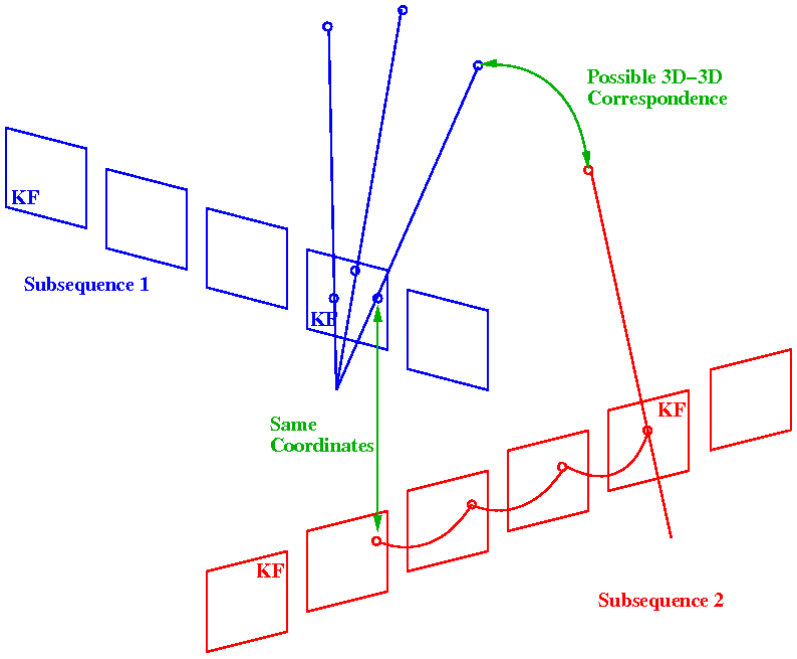


Figure 5.10: The two subsequences (blue and red) have been reconstructed in 3D. A 3D point in the blue subsequence has been reconstructed from 2D features in keyframes. We select a keyframe that is also present in the red subsequence. We take the feature with the same 2D coordinates and follow it until we end up in a keyframe of the red subsequence. If the tracked feature is reconstructed, this 3D point is a possible correspondence of the original blue 3D point.

A way to construct extra non-linearly dependend equations is shown in Fig. 5.11. It shows three points $\mathtt{M}_1$, $\mathtt{M}_2$, $\mathtt{M}_3$ for which we have the corresponding points in the other frame. We then construct 3 new points $\mathtt{M}_4$, $\mathtt{M}_5$, $\mathtt{M}_6$ as follows. First we compute the cross-product of the vectors $\mathtt{M}_2 - \mathtt{M}_1$ and $\mathtt{M}_3 - \mathtt{M}_1$. This vector is the normal of the plane spanned by $\mathtt{M}_1$, $\mathtt{M}_2$, $\mathtt{M}_3$. We then compute the mean distance d between the 3 points and construct the new points as shown in Fig. 5.11.

We now have 6 3D-3D correspondences. If we put the 13 unknowns of the metric transformation in a vector $\mathbf{e}$

$$\mathbf{e} = \begin{bmatrix} R_{11} & R_{12} & R_{13} & R_{21} & R_{22} & R_{23} & R_{31} & R_{32} & R_{33} & t_1 & t_2 & t_3 & s \end{bmatrix}^T$$

then each of the correspondences between a 3D point $\mathtt{M} = (XYZW)^T$ and a point $\mathtt{M}' = (X'Y'Z'W')^T$ gives rise to these three row vectors describing the equality of the 3D coordinates after transformation.

$$\begin{bmatrix} XW' & YW' & ZW' & WW' & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -WX' \\ 0 & 0 & 0 & 0 & XW' & YW' & ZW' & WW' & 0 & 0 & 0 & 0 & -WY' \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & XW' & YW' & ZW' & WW' & -WZ' \end{bmatrix}$$

Stacking all 18 equations gives us an overdetermined 18×13 matrix for which we compute the SVD. The nullvector corresponds to the linear solution for the metric transformation. Since we did not impose any non-linear constraint on the elements of the rotation matrix, the computed values for R_{ij} are not guaranteed to form a true rotation matrix which should be orthonormal with determinant 1. A typical solution to obtain a real rotation matrix is by using QR decomposition [19]. The drawback of this technique is that the extracted orthogonal matrix has no real physical significance. We therefore opt for *Polar decomposition* [52] which gives us the closest possible orthogonal matrix, where closeness is measured with the Frobenius norm.

Every sample of 3 3D-3D correspondences allows us to compute a metric transformation. RANSAC then checks for support by determining inliers and outliers for this solution. A straight-

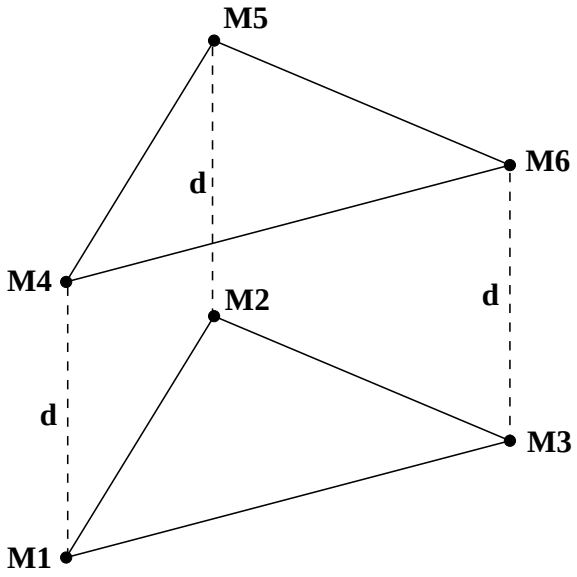


Figure 5.11: Given 3 points $\mathtt{M}_1$, $\mathtt{M}_2$, $\mathtt{M}_3$ we can construct 3 new points $\mathtt{M}_4$, $\mathtt{M}_5$, $\mathtt{M}_6$ located the mean distance d between the original points away from the original points in the direction of the normal on the plane spanned by these points.

forward criterion to do so would be to transform all points of one subsequence to the other using the computed transformation and compute the distance between the transformed point and its corresponding point. If this distance is lower than a threshold, the point is an inlier. However, it is unclear which value one should take for the 3D distance threshold. That is why we go back to the images. We still transform the 3D points of one subsequence with the computed solution and then project these transformed points into the image using the camera matrices of the other subsequence. This projection is then compared to the 2D feature points that were used to reconstruct the original 3D point in the first place. If the distance between the projection and the original point is lower than a threshold (typically one or two pixels), then the point is an inlier.

Once the inliers and outliers are determined and the solution with the largest support is chosen, a non-linear optimization process is executed using this solution as initialization. This process refines the solution and a final transformation is found which minimizes the total error.

Once all subsequences are put in the same coordinate system, a full non-linear optimization on the results can be executed. This bundle-adjustment updates the poses of the cameras, the intrinsics and the position of all 3D points, minimizing the global reprojection error. Care should be taken that common keyframes between subsequences only get one camera matrix and that all inliers of the 3D-3D correspondences relate to the same 3D point. All these constraints lead to index-hell but once implemented properly, the keyframes of the entire video sequence are reconstructed in 3D. Fig. 5.12 shows the result for the *Nymphaeum* sequence. Subsequences (cameras and points) are alternatingly shown in yellow and white. The detailed view on the right shows 22 cameras in the white subsequence. This means that in this specific case approximately every fifth frame was chosen as a keyframe.

5.5.3 Intermediate Frames

Certain applications are not satisfied with the calibration of only a subset of all original frames. The creation of a 3D movie from a video sequence for instance is in need of a depth map for all original frames. This means the camera pose has to be computed for all frames, not only the keyframes. Retrieving the camera calibration for intermediate frames (frames between keyframes) can be done as follows.

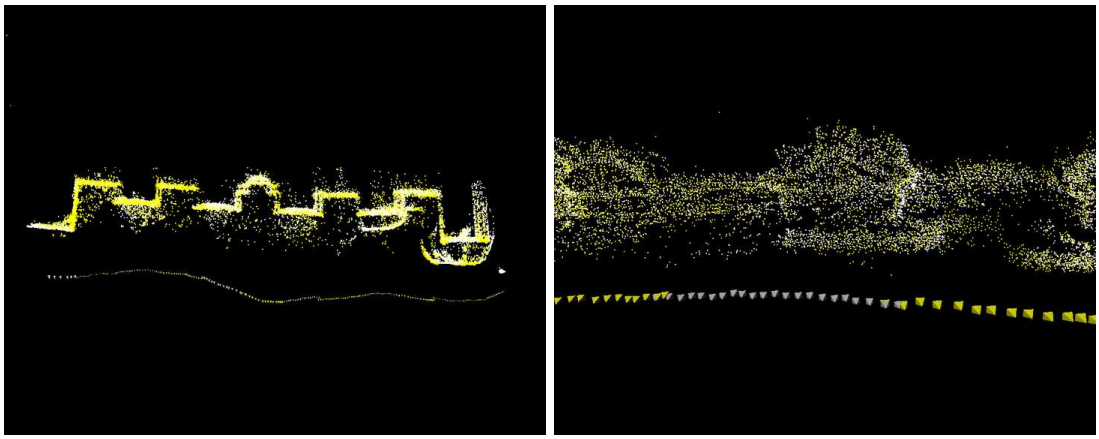


Figure 5.12: Reconstruction of keyframes in subsequences for the *Nymphaeum* sequence. The left view shows an entire overview, the right is a detail.

Remember that a feature tracker was executed on the entire original video sequence. This means that we have correspondences between all frames, and hence between keyframes and intermediate frames. From the set of 3D-2D correspondences in the keyframes, used to reconstruct the scene, we obtain 2D points in the keyframes for which we have 3D information. The feature tracks yield correspondences between these points and 2D points in the intermediate frames, which gives rise to 3D-2D correspondences for the intermediate frames. This means we have enough information to execute a Euclidean pose-estimation process (see paragraph 1.6.2).

In our experience an easier method exists which immediately computes the poses with a non-linear method. Because keyframes are typically not too far apart, we can initialize the pose of intermediate cameras by interpolation between the closest keyframes. The translation is easily found and for the rotation one can use quaternions. Another, very simple, method is the one described in [2] where the transformation of camera i, located between keyframes 1 and 2 is computed as

$$\mathbf{T}_i = \exp(a \log(\mathbf{T}_2) + (1-a) \log(\mathbf{T}_1))$$

where a is the ratio describing how far i is located between 1 and 2 (e.g. for the third frame of seven frames between the two keyframes, a is 3/7). The non-linear pose-estimation can then simply consist of a Euclidean bundle-adjustment process where the 3D points and intrinsic camera parameters remain fixed and only the extrinsics are updated.

Some results for the *globe* sequence are shown in Fig. 5.13. A frame of the sequence is shown in the top left and two views of the reconstruction and calibration of all cameras is shown as well. Fig. 5.14 shows a top view and a detailed view of the recovered cameras for the *Nymphaeum* sequence. A frame and the full calibration of the *Zaragossa* sequence is shown in Fig. 5.15.

5.5.4 Blurry Frames

When video sequences are recorded in a freehand stylem, one can never totally exclude the possibility of blurred frames. This bluriness is typically motion blur which is the result of sudden motions of the camera. Blurred frames cause problems for structure and motion techniques mostly because the amount of tracked features drops dramatically when such a frame is encountered. It is therefore best to try to detect these frames beforehand and deal with them. The simplest solution is of course to simply discard them.

A possible technique to spot blurred frames automatically is based on comparing the image *sharpness* of nearby frames. The value for the sharpness of an image could e.g. be computed as

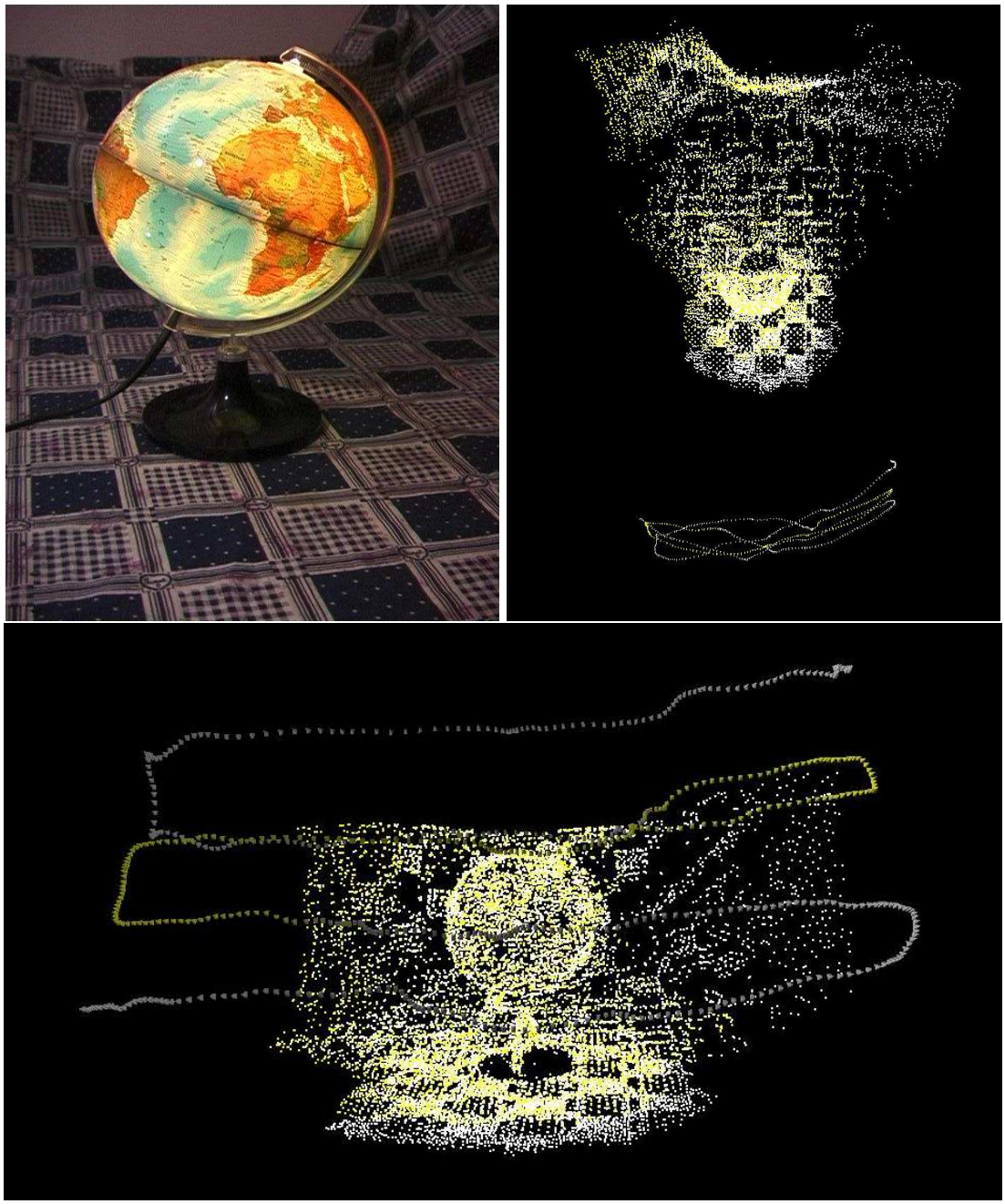


Figure 5.13: One frame of the *globe* sequence (top left). Reconstructed points and cameras shown in alternating yellow and white for alternating subsequences (top right and bottom)

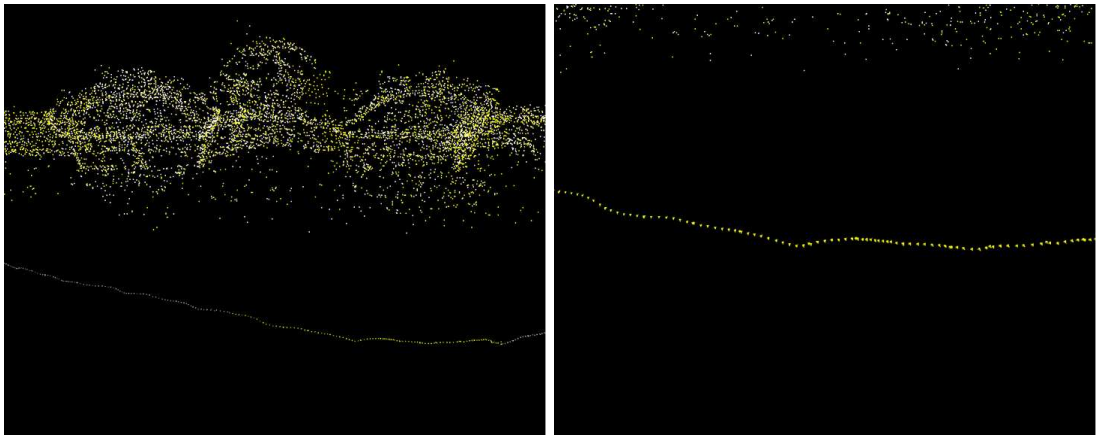


Figure 5.14: Reconstruction of all frames in subsequences for the *Nymphaeum* sequence.



Figure 5.15: One frame of the *Zaragossa* sequence (left). Reconstructed points and cameras shown in alternating yellow and white for alternating subsequences (right)

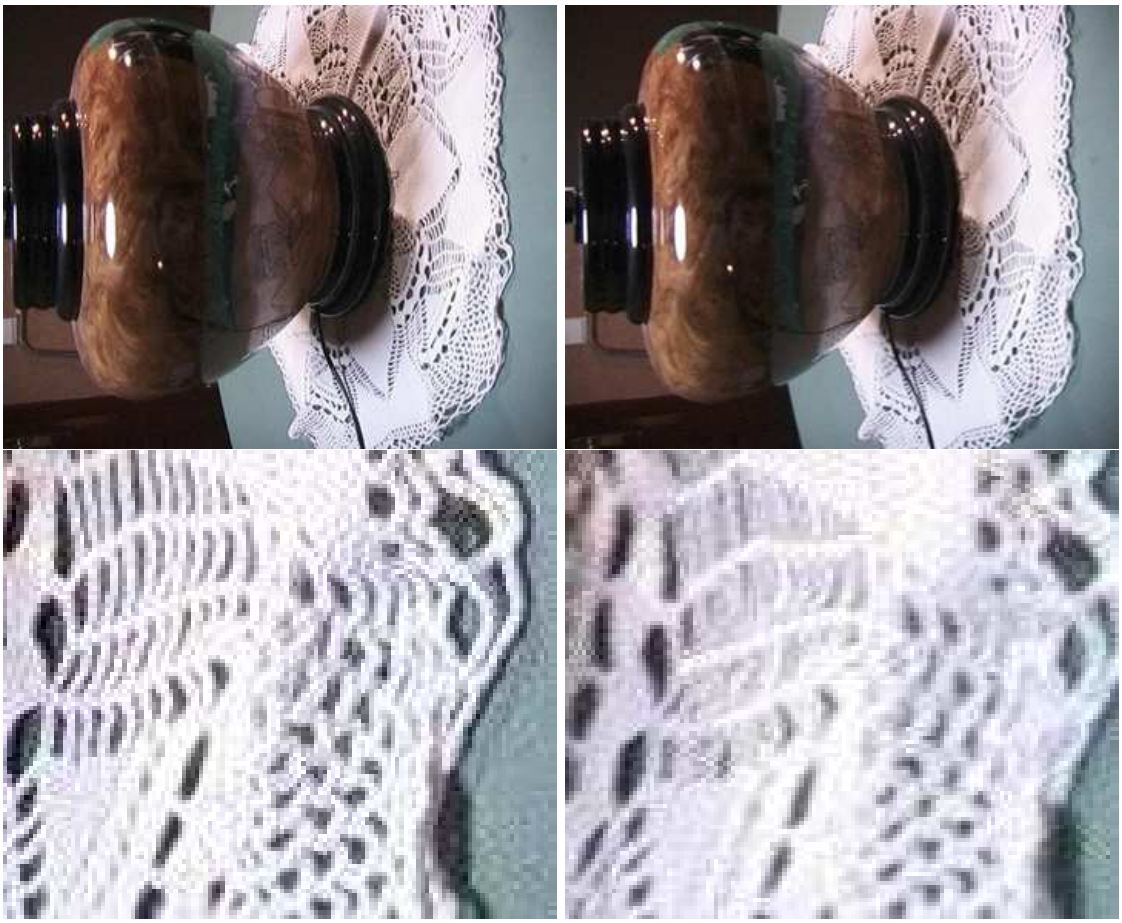


Figure 5.16: Frames 38 and 39 of the *vase* sequence (top). A zoom-in on a high-frequence part of the image shows that frame 39 suffers from motion-blur (bottom).

the mean square of horizontal and vertical intensity gradients, evaluated as finite differences:

$$S = \frac{\sum_{i=0}^{W-1}\sum_{j=0}^{H-1}(I(i+1,j)-I(i,j))^2 + (I(i,j+1)-I(i,j))^2}{WH}$$

The absolute value of such a sharpness measure depends on the image content and thus no absolute threshold can be used to detect blurry frames. However, if one compares the sharpness of frames in each other's neighborhood, then conclusions can be drawn. Due to motion blur, the gradient information in blurred images is significantly lower than in the other images. Fig. 5.16 shows two frames of the *vase* sequence. The second is image suffers from motion blur which is apparent in the zoom-in on a high-frequence part of the image. Fig. 5.17 shows a plot of the sharpness values for the first 150 frames of the sequence. A significant drop is noticable around frames 32, 39 and 100. The images in Fig. 5.16 are franes 38 and 39 respectively. The blurred frames are detected and can be removed from the sequence before further processing.

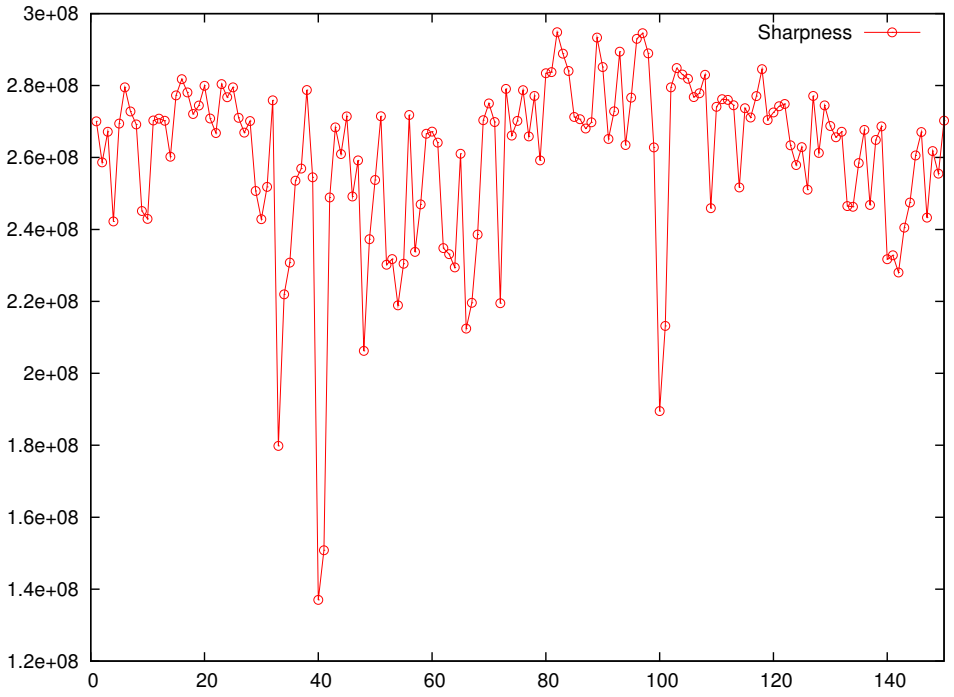


Figure 5.17: The sharpness plot of the first 150 frames of the *vase* sequence shows a noticable drop around frames 32, 39 and 100.

Chapter 6

Essential Matrices

Until now we have worked with cameras for which we did not know the internal calibration in advance. This meant we had to use the intermediate steps of projective reconstructions and self-calibration to recover the entire calibration of these cameras. If, however, we know the intrinsic parameters of the cameras, either through a calibration or self-calibration algorithm, more possibilities arise.

6.1 Essential Matrices

In chapter 2 the concept of the fundamental matrix was derived for an image pair with cameras with unknown internal calibration. This 3×3 matrix imposes a constraint on matching features in that they should lie on each-other's epipolar line. When the internal calibration of the cameras is known, we can *upgrade* this concept to that of an *essential* matrix. Historically these essential matrices were derived first [34] but since the fundamental matrix is more general than an essential matrix we have introduced it first.

6.1.1 Normalized Coordinates

If the intrinsic parameters of a camera are given, we can compute the so-called *normalized* coordinates of image points. Suppose that an image point $\mathtt{m}$ is the projection of a 3D point $\mathtt{M}$. With the projection equation 1.5 we have

$$\mathtt{m} \simeq \mathbf{K}\left[\mathbf{R}^T | -\mathbf{R}^T\mathbf{t}\right]\mathtt{M}$$

If the intrinsic parameters of the camera are known, we can define the normalized coordinates $\mathtt{q}$ as

$$\begin{aligned} \mathtt{q} &\simeq \mathbf{K}^{-1}\mathtt{m} \\ &\simeq \left[\mathbf{R}^T | -\mathbf{R}^T\mathbf{t}\right]\mathtt{M} \end{aligned} \tag{6.1}$$

6.1.2 The Essential Matrix

Let us now investigate a pair of images. Without loss of generality we let the world coordinate system coincide with the first camera. The equations of the two cameras are given by:

$$\begin{aligned} \mathbf{P}_1 &= \mathbf{K}_1\left[\mathbf{I}_{3\times 3}|\mathbf{O}_3\right] \\ \mathbf{P}_2 &= \mathbf{K}_2\left[\mathbf{R}^T | -\mathbf{R}^T\mathbf{t}\right] \\ &= \mathbf{K}_2\left[\mathbf{R}'|\mathbf{t}'\right] \end{aligned}$$

with $\mathbf{R}' = \mathbf{R}^T$ and $\mathbf{t}' = -\mathbf{R}^T\mathbf{t}$. A 3D point $\mathtt{M}$ in the world has the same coordinates in the frame of the first camera. In the second camera, the 3D point has (non-homogeneous) coordinates $\mathtt{M}'$

$$\mathtt{M}' = \mathbf{R}'\mathtt{M} + \mathbf{t}' \tag{6.2}$$

In order to derive the definition of the essential matrix, a couple of non-intuitive steps are taken. First the cross-product of $\mathtt{M}'$ with $\mathbf{t}'$ is computed.

$$\begin{aligned} \mathbf{t}' \times \mathtt{M}' &= \mathbf{t}' \times (\mathbf{R}'\mathtt{M} + \mathbf{t}') && (6.3)\\ &- \mathbf{t}' \times \mathbf{R}'\mathtt{M} + \mathbf{t}' \times \mathbf{t}' \\ &= \mathbf{t}' \times \mathbf{R}'\mathtt{M} && (6.4) \end{aligned}$$

because the cross product of a vector with itself is zero. For points at infinity (with zero fourth homogeneous coordinate), the last term ($\mathbf{t}'$) of Eq.6.2 is removed. This means that the cross-term of translations in Eq.6.3 falls away immediately and Eq.6.4 still holds. Now the scalar product of Eq. 6.4 and $\mathtt{M}'$ is computed. Remember that the scalar product of vectors $\mathbf{a}$ and $\mathbf{b}$ is $\mathbf{a}^T\mathbf{b}$.

$$\mathtt{M}'^T\mathbf{t}' \times \mathtt{M}' = \mathtt{M}'^T(\mathbf{t}' \times \mathbf{R}'\mathtt{M}) \tag{6.5}$$

Since the vector $\mathbf{t}' \times \mathtt{M}'$ is perpendicular to $\mathtt{M}'$ and we compute the scalar product with $\mathtt{M}'$, Eq 6.5 is zero. Using the skew-symmetric matrix, defined in Eq. 1.2 we can write

$$\begin{aligned} \mathtt{M}'^T(\mathbf{t}' \times \mathbf{R}'\mathtt{M}) &= 0 \\ \mathtt{M}'^T[\mathbf{t}']_\times\mathbf{R}'\mathtt{M} &= 0 \\ \mathtt{M}'^T\mathbf{E}\mathtt{M} &= 0 \end{aligned}$$

Because the normalized coordinates $\mathtt{q}_1$ and $\mathtt{q}_2$ are equal up to a scale factor to $\mathtt{M}$ and $\mathtt{M}'$ respectively, we can write

$$\mathtt{q}_2{}^T\mathbf{E}\mathtt{q}_1 = 0 \tag{6.6}$$

with

$$\mathbf{E} \simeq [\mathbf{t}']_\times\mathbf{R}' \tag{6.7}$$

Comparing Eq. 6.6 with Eq. 2.14 shows that the relationship between $\mathbf{E}$ and $\mathbf{F}$ is

$$\mathbf{E} \simeq \mathbf{K}_2^T\mathbf{F}\mathbf{K}_1 \tag{6.8}$$

6.1.3 Properties of the Essential Matrix

Since the essential matrix can be computed from the relative rotation and translation between the two cameras but is only determined up to scale, it has five degrees of freedom. This means there are two extra constraints that have to be enforced w.r.t. the fundamental matrix. These constraints are that the two of its singular values have to be equal and that the third is zero.

This can be proven from the eigenvalue decomposition of a skew-symmetric matrix. In [19] it is derived that a real and skew-symmetric matrix $\mathbf{S}$ can be decomposed into $\mathbf{S} = \mathbf{U}\mathbf{B}\mathbf{U}^\mathbf{T}$ where $\mathbf{B}$ is an orthonormal matrix and $\mathbf{B}$ is a block-diagonal matrix of the form

$\mathbf{B} = diag(a_1\mathbf{Z_1}, a_2\mathbf{Z_2}, \ldots, a_m\mathbf{Z_m}, 0, \ldots, 0)$ where $\mathbf{Z} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$

Let us define two matrices $\mathbf{W}$ and $\mathbf{Z}$

$$\mathbf{W} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \mathbf{Z} = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{6.9}$$

then we see that $\mathbf{W}$ is orthonormal and $\mathbf{Z}$ is skew-symmetric. From Eq. 6.7 an essential matrix is defined as the multiplication of a skew-symmetric matrix $[\mathbf{t}]'_\times$ and an orthonormal matrix $\mathbf{R}'$. We decompose the skew-symmetric part as

$$[\mathbf{t}]'_\times \simeq \mathbf{U}\mathbf{Z}\mathbf{U}^\mathbf{T}$$

From their definition, $\mathbf{Z} = diag(1,1,0)\mathbf{W}$ up to scale, hence

$$[\mathbf{t}]'_\times \simeq \mathbf{U}\mathbf{diag}(\mathbf{1},\mathbf{1},\mathbf{0})\mathbf{W}\mathbf{U}^\mathbf{T}$$

and thus

$$\begin{aligned} E &= [\mathbf{t}]'_\times \mathbf{R}' \simeq \mathbf{U}\mathbf{diag}(\mathbf{1},\mathbf{1},\mathbf{0})\mathbf{W}\mathbf{U}^\mathbf{T}\mathbf{R}' && (6.10)\\ &\simeq \mathbf{U}diag(1,1,0)\mathbf{V}^T && (6.11)\end{aligned}$$

This is indeed a singular value decomposition with two equal non-zero and one zero singular value.

From Eq.6.10 it may seem that we have 6 degrees of freedom in $\mathbf{E}$: 3 to describe the orthonormal matrix $\mathbf{U}$ and 3 for the orthonormal matrix $\mathbf{V}$. However, that fact that two singular values of the SVD are equal, makes the SVD not unique. Equation 6.11 for instance also decomposes $\mathbf{E}$ in an SVD for any 2×2 rotation matrix $\mathbf{R}_{\mathbf{2}\times\mathbf{2}}$.

$$\mathbf{E} \simeq \mathbf{U}\begin{bmatrix} \mathbf{R}_{\mathbf{2}\times\mathbf{2}} & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix} diag(1,1,0) \begin{bmatrix} \mathbf{R}_{\mathbf{2}\times\mathbf{2}}{}^T & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix} \mathbf{V}^T \quad (6.12)$$

For a derivation of the fact that the first two singular values are equal boils down to two constraints instead of one, we refer the reader to [15] where the equivalence of Eq. 6.12 with the Kruppa equations [31] is shown.

6.1.4 Cameras from the Essential Matrix

From Eq. 6.7 we see that the essential matrix is constructed only from the relative extrinsic parameters of the cameras of the two images. It stands to reason that is possible to extract them again if the essential matrix is known. For this we make use of the matrices $\mathbf{Z}$ and $\mathbf{W}$ defined in Eq. 6.9 and the singular decomposition of $\mathbf{E}$ into $\mathbf{U}diag(1,1,0)\mathbf{V}^T$. There are two possible factorizations of $\mathbf{R}'$ and $\mathbf{t}'$:

$$\begin{aligned} [\mathbf{t}']_\times &= \mathbf{U}\mathbf{Z}\mathbf{U}^T && (6.13)\\ \mathbf{R}'_1 &= \mathbf{U}\mathbf{W}\mathbf{V}^T && (6.14)\\ \mathbf{R}'_2 &= \mathbf{U}\mathbf{W}^T\mathbf{V}^T && (6.15)\end{aligned}$$

That this is true can easily be verified by inserting these factorizations into Eq. 6.7 and comparing this (up to scale) to Eq. 6.11. That there are no other factorizations can be seen as follows. Since $E = [\mathbf{t}]'_\times \mathbf{R}'$, the left null-space of $[\mathbf{t}]'_\times$ must be identical to that of $\mathbf{E}$. Thus, Eq. 6.13 must be the decomposition of $[\mathbf{t}]'_\times$. The rotation $\mathbf{R}'$ may be written as $\mathbf{U}\mathbf{X}\mathbf{V}^T$ where $\mathbf{X}$ is an arbitrary rotation matrix. Thus

$$\mathbf{U}diag(1,1,0)\mathbf{V}^T = \mathbf{E} = [\mathbf{t}]'_\times \mathbf{R}' = \mathbf{U}\mathbf{Z}\mathbf{U}^\mathbf{T}\mathbf{U}\mathbf{X}\mathbf{V}^\mathbf{T} = \mathbf{U}(\mathbf{Z}\mathbf{X})\mathbf{V}^\mathbf{T}$$

which means that $\mathbf{Z}\mathbf{X} = diag(1,1,0)$. Because $\mathbf{X}$ must be a rotation matrix, this equation only holds for $\mathbf{X} = \mathbf{W}$ or $\mathbf{X} = \mathbf{W}^T$, from which follow Eq. 6.14 and Eq. 6.15.

Using the factorizations of Eq. 6.13, 6.14 and 6.15 we can now extract the relative rotation and translation from the essential matrix. For the two rotations this immediately follows from the equations. For the translation we follow this reasoning: Because $[\mathbf{t}]'_\times \mathbf{t}' = \mathbf{0}$, it follows that $\mathbf{U}\mathbf{Z}\mathbf{U}^T\mathbf{t}' = 0$, thus $\mathbf{t}'$ must be the third column of $\mathbf{U}$, $\mathbf{u}_3$. the singular vector correspondening to the third, zero singular value. However, the sign of $\mathbf{t}'$ can not be determined which gives rise to two possible solutions for $\mathbf{t}'$: $\mathbf{t}'_1 = \mathbf{u}_3$ and $\mathbf{t}'_1 = -\mathbf{u}_3$. Remark that, since the essential matrix can only be recovered up to scale, the scale of the translation is unknown as well. Only the direction can be recovered.

This means that there are four possible solutions to the relative rotation and translation between the two cameras: $(\mathbf{t}'_1, \mathbf{R}'_1)$, $(\mathbf{t}'_2, \mathbf{R}'_1)$, $(\mathbf{t}'_1, \mathbf{R}'_2)$ and $(\mathbf{t}'_2, \mathbf{R}'_2)$. Beware that the extracted rotation and translation are actually the inverse values we have to fill in in the projection equation

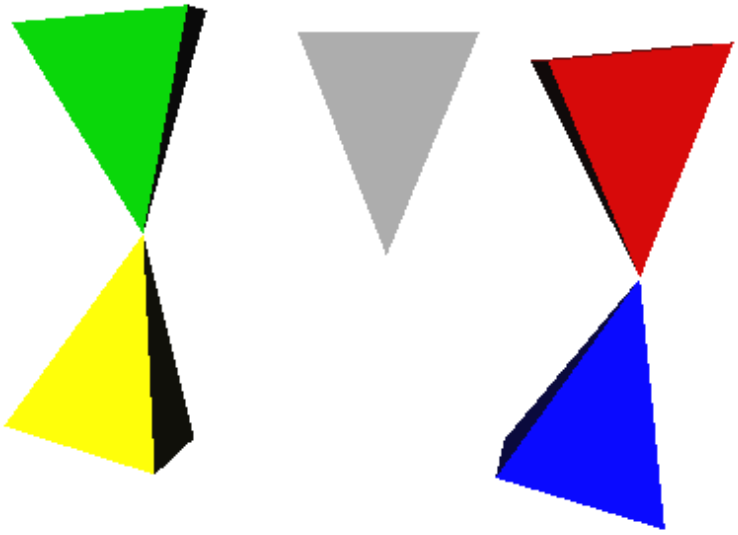


Figure 6.1: From the SVD of the essential matix $\mathbf{E}$, four solutions for the relative translation and rotations can be computed. In this figure, the grey camera is always the first camera, the inverse rotation and translation of the second camera are given by $\mathbf{R}'$ and $\mathbf{t}'$. The four possible solutions $(\mathbf{t}'_1, \mathbf{R}'_1)$, $(\mathbf{t}'_2, \mathbf{R}'_1)$, $(\mathbf{t}'_1, \mathbf{R}'_2)$ and $(\mathbf{t}'_2, \mathbf{R}'_2)$ correspond to the blue, yellow, red and green camera respectively.

since we defined $\mathbf{R}' = \mathbf{R}^T$ and $\mathbf{t}' = -\mathbf{R}^T\mathbf{t}$. The four solutions are depicted in Fig. 6.1. The grey camera in the middle is the first camera, put in the origin. The four solution in the order given before correspond to the pairs consisting of the grey camera with the blue, yellow, red and green camera respectively.

6.2 Computation of the Essential Matrix

The relative rotation and direction of translation between the cameras of a camera pair can directly be extracted from the essential matrix of this pair. This has tempted computer vision researchers to make use of it since many years. There are many techniques to compute the essential matrix from point matches and three of them will be explained here.

6.2.1 8 and 7 Point Algorithm

The most straightforward method to compute the essential matrix is the simple, linear 8-point algorithm we introduced in paragraph 2.2.3 to compute the fundamental matrix. This is the oldest algorithm and suffers from the fact that no constraints on the matrix are enforced. The seven point algorithm enforces the rank-2 constraint. It is non-linear and boils down to finding the roots of a third degree polynomial.

We can employ exactly the same algorithm to compute the essential matrix. Either we follow the same procedure as in paragraph 2.2.3 to compute the fundamental matrix with the 8- or 7 point algorithm. Then we can extract the essential matrix from $\mathbf{F}$ using Eq. 6.8. An alternative algorithm immediately employs the intrinsic parameters to compute the normalized coordinates with Eq. 6.1. Then we solve for $\mathbf{E}$ directly with the 8- or 7 point algorithm.

Both methods described here suffer from the same problem. They don't enforce the extra constraints an essential matrix should fulfill. The 7 point algorithm only requires the rank of the

matrix to be 2. The two remaining singular values of the matrix are not necessarily equal. The following algorithms enforce this constraint as well.

6.2.2 6 Point Algorithm

Rewriting the essential constraint

In 1990 Faugeras and Maybank showed [44] that the constraint on equal singular values of the essential matrix can be written as the following equation on $\mathbf{E}$

$$\mathbf{E}\mathbf{E}^T\mathbf{E} - \frac{1}{2}trace(\mathbf{E}\mathbf{E}^T)\mathbf{E} = 0 \tag{6.16}$$

The proof is actually quite simple. The singular value decomposition of $\mathbf{E}$ is written as

$$\mathbf{E} = \mathbf{U}\mathbf{S}\mathbf{V}^T$$

where

$$\mathbf{S} = \begin{bmatrix} s_1 & & \\ & s_2 & \\ & & s_3 \end{bmatrix}$$

From the properties of the SVD, we know that $\mathbf{U}$ and $\mathbf{V}$ are orthonormal matrices and that $s_1 \geq s_2 \geq s_3 \geq 0$. The trace of a matrix is defined as the sum of its diagonal elements. For instance, the trace of $\mathbf{S}$ is $(s_1 + s_2 + s_3)$. Because of orthogonality, and the fact that $\mathbf{S}^T = \mathbf{S}$ it follows that

$$\begin{aligned} \mathbf{E}\mathbf{E}^T\mathbf{E} &= \mathbf{U}\mathbf{S}\mathbf{V}^T\mathbf{V}\mathbf{S}^T\mathbf{U}^T\mathbf{U}\mathbf{S}\mathbf{V}^T \\ &= \mathbf{U}\mathbf{S}^3\mathbf{V}^T \\ &= \mathbf{U}\begin{bmatrix} s_1^3 & & \\ & s_2^3 & \\ & & s_3^3 \end{bmatrix}\mathbf{V}^T \end{aligned} \tag{6.17}$$

The second term can be written as

$$-\frac{1}{2}trace(\mathbf{E}\mathbf{E}^T)\mathbf{E} = -\frac{1}{2}trace(\mathbf{U}\mathbf{S}^2\mathbf{U}^T)\mathbf{U}\mathbf{S}\mathbf{V}^T$$

Because $\mathbf{U}$ and $\mathbf{V}$ are orthonormal, their influence on the trace of the multiplication with $\mathbf{S}$ is nihil. Thus this term is equal to

$$-\frac{1}{2}trace(\mathbf{E}\mathbf{E}^T)\mathbf{E} = -\frac{1}{2}\left(s_1^2 + s_2^2 + s_3^2\right)\mathbf{U}\mathbf{S}\mathbf{V}^T \tag{6.18}$$

Substituting Eq.6.17 and Eq.6.18 into Eq.6.16 shows us that for $k = 1, 2, 3$

$$s_k^3 = \frac{1}{2}\left(s_1^2 + s_2^2 + s_3^2\right)s_k \tag{6.19}$$

which can only hold if $s_1 = s_2$ and $s_3 = 0$ which is exactly the constraint we want to enforce in the essential matrix.

Computing E from 6 point matches

How can we employ the constraint of Eq. 6.16 to compute $\mathbf{E}$ from 6 matches? A possible strategy is explained in [45]. As a first step we write down the 6 homogeneous equations 6.6 in the 9 elements of $\mathbf{E}$. The svd of this 6x9 matrix yields the right nullspace of this matrix, consisting of 3 vectors. The real solution for $\mathbf{E}$ cannot be computed just from this SVD and is of course a linear combination of these three vectors. The scale factors for this linear combination are called λ_1, λ_2 and λ_3. If we insert this solution (with the three unknowns λ_i into Eq. 6.16, this yields 9

homogeneous polynomial equations of degree 3 in λ_1, λ_2 and λ_3. Since $\mathbf{E}$ can only be computed up to scale, we can only recover ratios of λ_i:

$$x = \frac{\lambda_1}{\lambda_3} \quad y = \frac{\lambda_2}{\lambda_3} \tag{6.20}$$

Since the 9 equations are of degree 3 in λ_i, we can write them down using the substitution in Eq. 6.20, yielding terms in $x^3, x^2y, x^2, xy^2, xy, x, y^3, y^2, y, 1$. Thus, we have 9 equations in 9 unknowns, which can be solved for linearly. The solution for x and y is used to compute λ_i from Eq. 6.20, setting λ_3 arbitrarily to 1.

The advantage of this approach is that it is a linear method to compute the essential matrix, taking into account the constraints on $\mathbf{E}$ explicitely. Only 6 point pairs are needed instead of 7 or 8 which reduces the amount of necessary tries in the RANSAC algorithm as can be seen from formula 2.20.

6.2.3 5 Point Algorithm

We can now compute the essential matrix from 8, 7 or 6 point matches. However, we know that $\mathbf{E}$ has only 5 degrees of freedom, so we search for a method that can compute it from 5 point matches only. Unfortunately, no such algorithm exists that is linear. In [45] a non-linear 5 point algorithm is explained. Nister employs the same technique in [43] to solve for the essential matrix.

The algorithm follows the same steps as the 6-point approach. Five equations on the 9 elements of $\mathbf{E}$ now yield a nullspace of 4 vectors, from which the solution is described as a linear combination in four parameters λ_1, λ_2, λ_3,λ_4. A similar substitution is performed as

$$x = \frac{\lambda_1}{\lambda_4} \quad y = \frac{\lambda_2}{\lambda_4} \quad z = \frac{\lambda_3}{\lambda_4} \tag{6.21}$$

From this a linear system is obtained in variables x^3, x^2y, x^2z, x^2 etc. We can apply a Gauss-Jordan elimination on these 9 equations to yield the following form:

	x^3	x^2y	x^2z	x^2	xy^2	xyz	xy	xz^2	xz	x	y^3	y^2z	y^2	yz^2	yz	y	z^3	z^2	z	1
(a)	1									.	.	.	.	.	.	.	.	.	.	.
(b)		1								.	.	.	.	.	.	.	.	.	.	.
(c)			1							.	.	.	.	.	.	.	.	.	.	.
(d)				1						.	.	.	.	.	.	.	.	.	.	.
(e)					1					.	.	.	.	.	.	.	.	.	.	.
(f)						1				.	.	.	.	.	.	.	.	.	.	.
(g)							1			.	.	.	.	.	.	.	.	.	.	.
(h)								1		.	.	.	.	.	.	.	.	.	.	.
(i)									1	.	.	.	.	.	.	.	.	.	.	.

(6.22)

In the rest of the discussion $[n]$ is used to denote an n-th degree polynomial in z. Let us start by eliminating xz^2 from equations (h) and (i) in Eq. 6.22. which is done by subtracting z times (i) from (h), yielding a new equation

$$x[1] + y^4[1] + y^2[2] + y[3] + [4] = 0 \tag{6.23}$$

In the same way we can eliminate other factors by computing

$$z(g) - (f)$$
$$z(d) - (c)$$

which yields the same form as Eq. 6.23. If we compute $(e) - y(g) - [0](g)$, then we obtain an expression like

$$x[0] + y^4[0] + y^3[1] + y^2[2] + y[3] + [3] = 0 \tag{6.24}$$

The same form is obtained from $(b) - x(g) - [0](g)$. A linear combination of the two equations of form 6.24 can eliminate the term in y^4 which gives us an equation of the form

$$x[0] + y^3[1] + y^2[2] + y[3] + [3] = 0 \tag{6.25}$$

Finally, one more equation is obtained from

$$(a) - x(d) - [0](d) - y[0](e) - [1](e) - [2](g) - [0](6.24) \tag{6.26}$$

Combining the three equations of the form (6.23) and the two equations (6.25) and (6.26) gives us five equations in the variables $x, y^3, y^2, y, 1$ where every coefficient is a polynomial in z. The stacked equations look like Eq. 6.27

$$\begin{array}{rcl} x[1] + y^3[1] + y^2[2] + y[3] + [4] & = & 0 \\ x[1] + y^3[1] + y^2[2] + y[3] + [4] & = & 0 \\ x[1] + y^3[1] + y^2[2] + y[3] + [4] & = & 0 \\ x[0] + y^3[1] + y^2[2] + y[3] + [3] & = & 0 \\ x[3] + y^3[2] + y^2[3] + y[4] + [5] & = & 0 \end{array} \tag{6.27}$$

This homogoneous system of equations only has a non-trivial solution if the determinant of the 5x5 matrix is zero. This constraint leads to a 13-th degree polynomial in z, denoted as $p(z) = 0$. Thus, if we compute the roots of the polynomial $p(z)$, e.g. by means of a Sturm Chain [18] or by computing the eigenvalues of p's companion matrix [13], we compute possible solutions for z. For every solution we solve for x and y using system 6.27, from which we compute λ_i by means of Eq. 6.21 and setting $\lambda_4 = 1$. This then gives us the linear combination of the vectors of the nullspace and yields a possible solution for $\mathbf{E}$. In [43], the technique described above is slightly adapted to yield two 11th degree polynomials of which the roots give possible solutions for $\mathbf{E}$.

6.2.4 Planar Degeneracy

Bearing in mind the discussion we had in chapter 5 about degenerate cases, we wonder how the computation of the essential matrix suffers from such scenes. We are particularly interested in planar scenes because they are the most common form of degenerate surfaces. If we have knowledge of the intrinsic parameters of the camera, we can employ the techniques described above the compute the essential matrix from 5, 6, 7 or 8 point correspondences. Because the 7 and 8 point algorithm are basically the same as the F-matrix algorithm, they don't work either on planar scenes. The 6-point algorithm, while interesting due to the reduction of the needed correspondences, will also fail on purely planar scenes. The 5 point algorithm however is in general unaffected by the planar degeneracy and will yield good results.

It has been shown [21, 37, 42] that two possible solutions exist for the essential matrix of two cameras looking at a plane. This is not a problem because for only one of the two solutions do all reconstructed points lie in front of the cameras. This so called cheirality principle [26] allows us to discern the correct solution from the false which always works provided that not all the points are closer to one of the cameras than to the other.

Sometimes, the specifics of the camera setup allow for an optimized algorithm. Take for example the case of a stereo camera setup, mounted on a pan and tilt axis. This system allows to record stereo pairs in all directions from one position. In order to calibrate the relative position and orientation between the two cameras, we can employ images of an unknown environment. If this environment is planar, the 6, 7 and 8 point algorithms don't work. However, we can think of a different strategy. If we take multiple image pairs with different pan and tilt angles the relative pose of the cameras does not change. This means we can regard this setup as if the cameras don't move but the scene does, as illustrated in Fig. 6.2. For every tilt angle we compute a set of possible matches between feature points. If we now combine these different sets into one and feed this to the standard F-RANSAC algorithm, using the simple 7-point algorithm, then we can

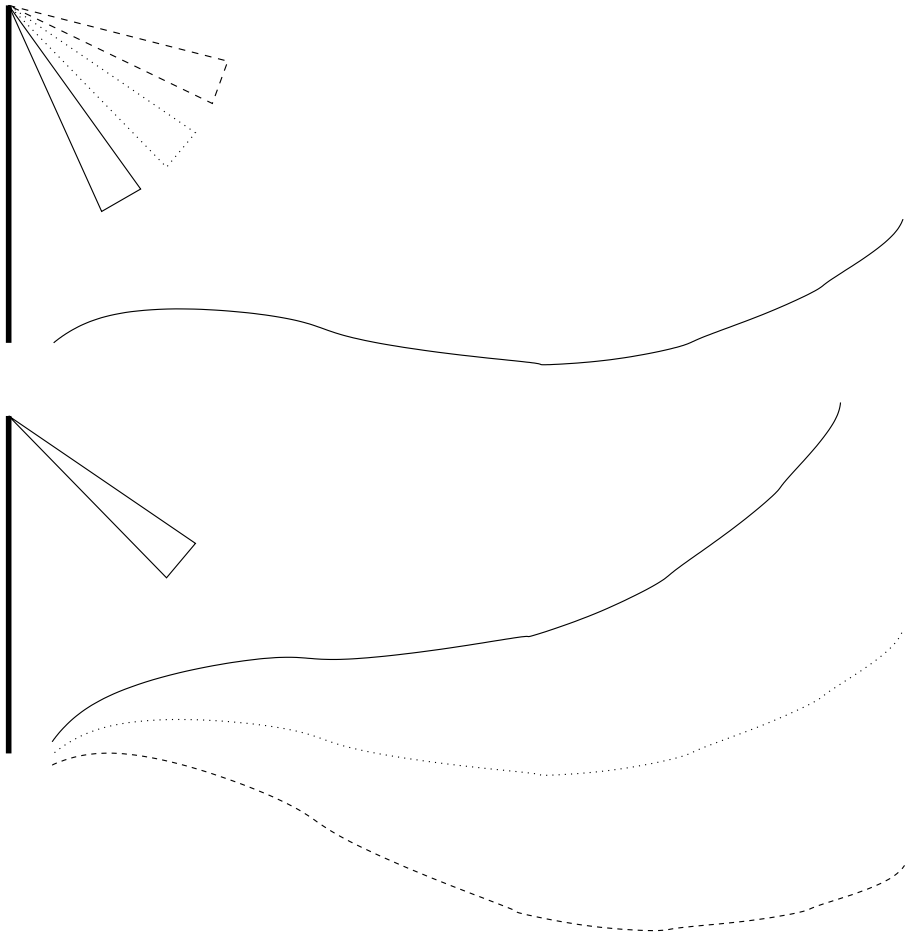


Figure 6.2: Combining different segments. The top view shows the setup of cameras looking at a (possibly planar) scene from different tilt angles. This is equivalent to looking from the same position at a scene consisting of repetitions of the original scene at different positions.

compute the fundamental matrix, even if the terrain is planar because the combination made an artificial scene, consisting of different planes. The essential matrix is easily derived from Eq. 6.8 and the extrinsics of the cameras are computed from Eq. 6.13, 6.14, 6.15.

Chapter 7

Bundle Adjustment

Chapter 8

Dense Matching

8.1 Introduction

The result of the previous chapters consists of camera calibration and sparse 3D point reconstruction. This is only part of the story. In order to obtain convincing 3D models, we must reconstruct much more 3D points, i.e. obtain a *dense* reconstruction of the scene. To do so, we must search for dense matches between the images. In paragraph 8.3 we will examine a possible algorithm for the standard technique to perform dense matching between a pair of images: *stereo matching*. Because an image sequence consists of multiple images and thus multiple image pairs, the depth information, computed from the image pairs must be integrated. A technique to do so is explained in paragraph 8.4. Paragraphs 8.5 and 8.6 explain how dense depth information can be extracted directly from multiple images, using programmables GPUs or a Bayesian approach respectively. First however, we will explain a standard technique, called *rectification* to simplify dense matching by taking into account knowledge on the epipolar geometry.

8.2 Rectification

In section 2.2.1 we introduced the concept of epipolar geometry for an image pair. The epipolar geometry of an image pair delivers pairs of corresponding epipolar lines. This concept is important because and will reduce our dense matching efforts tremendously because a pixel in the first image only needs to be compared to pixels on the corresponding epipolar line in the other image. In general epipolar lines can lie in all directions. This hampers the implementation of search algorithms that want to take advantage of the knowledge of the epipolar geometry. A pre-processing step, called *rectification* will make our lives much easier. The idea is that, since pairs of epipolar lines can be found, it is a good idea to rewarp the images such that these corresponding lines now coincide with (horizontal) image scanlines. This makes implementation of the correspondence search easier because then correspondences must only be searched along scanlines with the same y coordinate in the two images and hence only differ in horizontal displacement, called *disparity*.

8.2.1 Planar Rectification

Different algorithms exist to rectify pairs of images. The most popular kind of rectification is *planar* rectification in which an optimal plane is searched for. The two images I_1 and I_2 are projected onto this plane with a homography. Image I_1 is warped to image I_1' using homography $\mathbf{H_1}$ and I_2 is warped to I_2' with $\mathbf{H_2}$. The resulting image pair should have correspondencing epipolar lines that are horizontal. This boils down to a fundamental matrix of the following form

$$\mathbf{F}' \simeq \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}$$

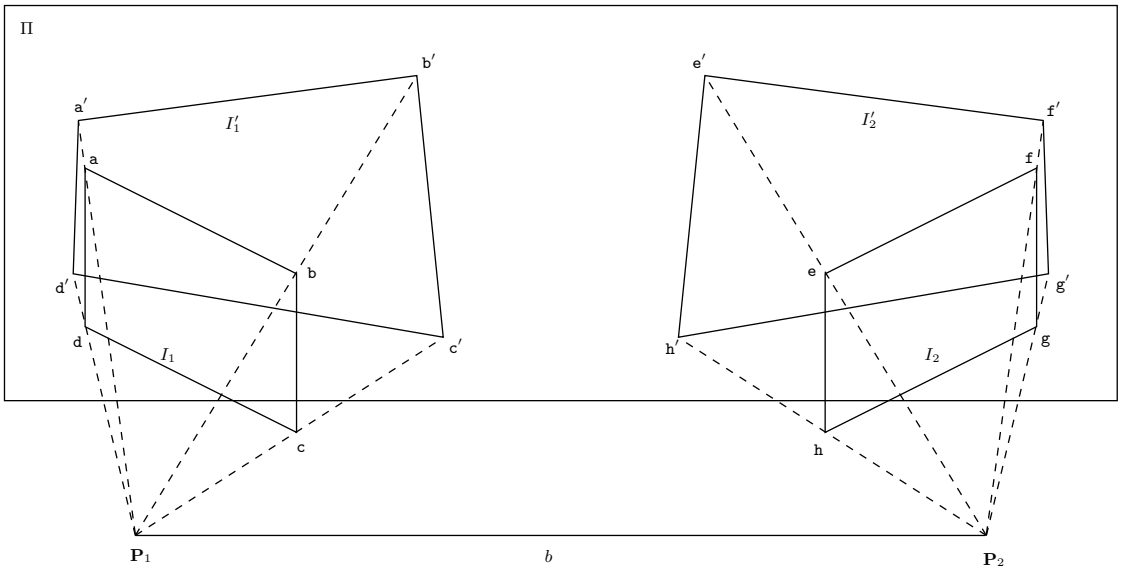


Figure 8.1: Planar Rectification. The plane Π is parallel to the baseline b. The corner points of the images are backprojected through from the respective cameras. The correspondences between the intersection of these rays with the plane and the original points allow to compute the homographies between the original and the rectified images.

Let $\mathbf{F}$ be the (known) fundamental matrix of the original pair I_1, I_2. Then for all corresponding points $\mathtt{m}_1, \mathtt{m2}$ we can write

$$\begin{aligned} {\mathtt{m}_2}^T \mathbf{F} \mathtt{m}_1 &= 0 \\ {\mathtt{m}_2'}^T \mathbf{F}' \mathtt{m}_1' &= 0 \\ \mathtt{m}_2 {\mathbf{H_2}}^T \mathbf{F}' \mathbf{H_1} \mathtt{m}_1 &= 0 \end{aligned}$$

and thus

$$\mathbf{F} \simeq {\mathbf{H_2}}^T \mathbf{F}' \mathbf{H_1} \tag{8.1}$$

Equation 8.1 gives the relation between the old and the new fundamental matrix, using the two homographies. Eq. 8.1 does not uniquely define the two homographies $\mathbf{H_1}$ and $\mathbf{H_2}$ and existing algorithms differ in the choice of the remaining degrees of freedom. The standard approach is to consider the baseline between the two images of the stereo setup and selecting a plane parallel to this baseline. The remaining degrees of freedom correspond to the orientation of the plane and the distance to the baseline. Different techniques exist to find the *optimal* plane w.r.t. a certain criterion, for instance from information on the two optical axes: construct the two epipolar planes, containing the baseline and the optical axes. Select the orientation of the plane such that the normal bisects the angle between these two planes. The distance can be defined such that no compression of pixels is performed. A different approach that minimizes the image deformation is given in [35]. Once the plane is found, the homographies are easily computed as is shown in Fig. 8.1. The corner points $\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}$ of image I are back-projected using $\mathbf{P}_1$ and their intersections $\mathtt{a}', \mathtt{b}', \mathtt{c}', \mathtt{d}'$ with the plane Π are found. From the correspondences between the corner points and their projections, the homography $\mathbf{H_1}$ can then easily be computed with the algorithm explained in paragraph 2.2.6. The homography $\mathbf{H_2}$ is computed analogically. The top of Fig. 8.2 shows two images of an image sequence. The epipolar geometry is depicted in the middle images. At the bottom the resulting planar rectified images are shown. One can check that corresponding epipolar lines are now horizontal and at the same y-coordinate.

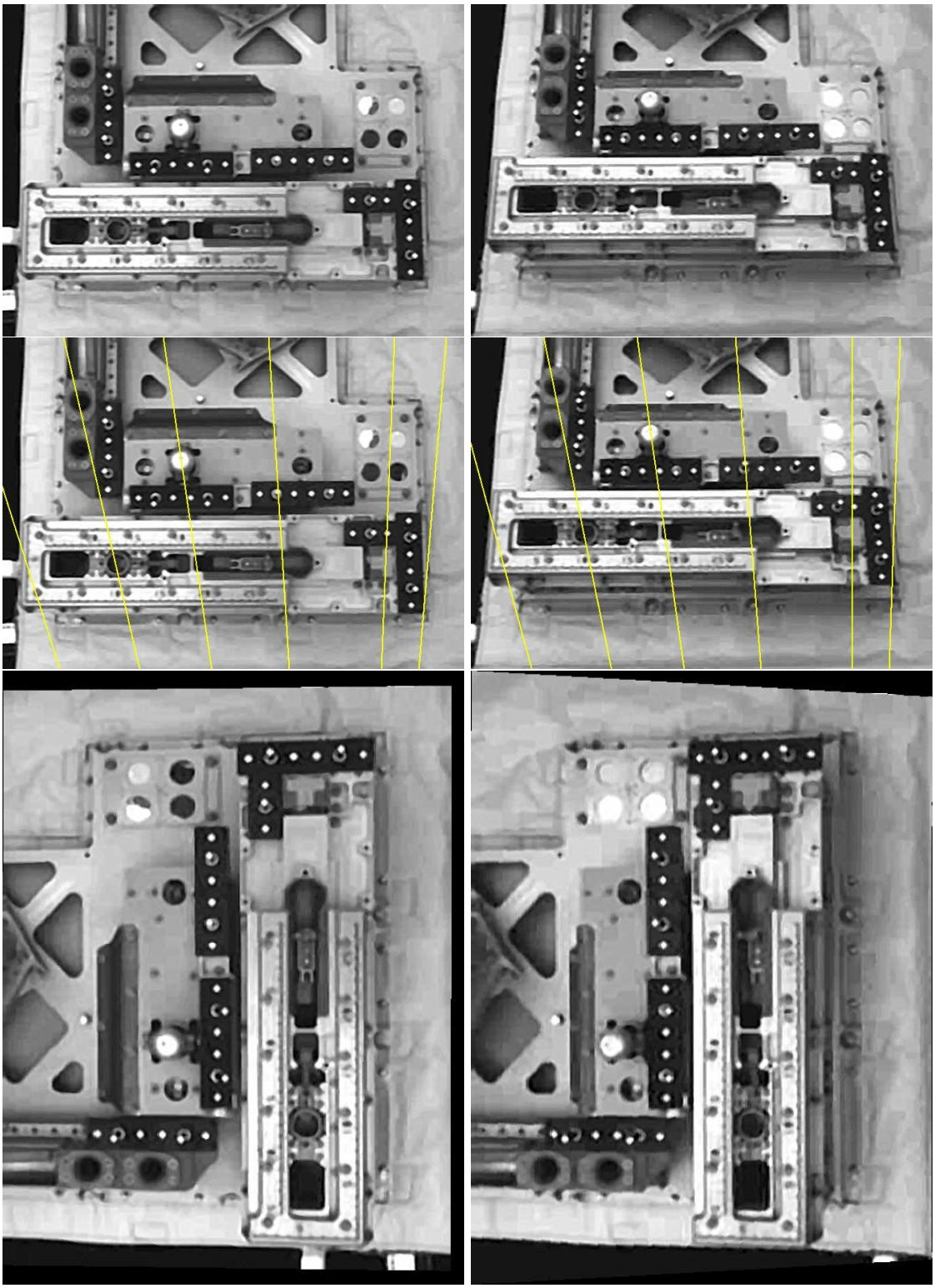


Figure 8.2: Original image pair (top). Epipolar geometry of the original pair (middle). Planar rectified images (bottom)

8.2.2 Polar Rectification

The planar rectification method has several advantages. First, it is relatively easy to compute. The search for the plane is the hardest part. Once it has been located, the rest of the algorithm is straightforward. Second, since the images are only transformed with a planar homography, all projective invariants remain valid. This means e.g. that straight lines in the original images are still straight in the resulting rectifiified images.

Unfortunately, planar rectification has a major weakness: it can not handle all configurations of image pairs. A typical example is the setup in which the camera moves forward. In this case the epipoles are located in the image. If a plane has to be chosen which is parallel to the baseline and we don't want to lose any image information, then the resulting rectified images are doomed to become infinitely large. In [49] an algorithm is described which can handle general camera configurations. This method, called *polar rectification* only makes use of knowledge on the oriented fundamental matrix between images (for more information on oriented epipolar geometry, we refer the reader to [33]). The algorithm essentially considers a pencil of epipolar lines through the epipole of one image and the corresponding lines through the other epipole and is illustrated by Fig 8.3. The pixels that are encountered on the epipolar line with angle θ_1 in image 1 are written in a scanline of the first rectified image. Analogically the corresponding scanline in the rectified second image is formed by using pixels on the epipolar line with angle θ_2. Of course only epipolar lines that intersect both images need to be take in into account. The images are thus traversed in a fashion not unlike *polar* coordinates: with an angle θ describing which epipolar line we take and a distance r between the pixel and the epipole. Fig. 8.4 illustrates this: the rectified version of an image on the left has pixels between θ_{min} and θ_{max} and between r_{min} and r_{max}. In practice, the algorithm is a somewhat more involved than is lead to believe, depending on the location of the epipoles w.r.t. the images. However, the technique can deal with general configurations of cameras.

An example of a polar rectified image pair is shown in Fig 8.5. For well-behaved setups, the result of this approach does not differ much from the planar rectification. The big advantage however is that all camera configurations (including forward motions) can be dealt with. An example of such a setup is shown in Fig. 8.6 which depicts a forward motion in a narrow street in Leuven. The epipolar geometry is shown in the middle with epipolar lines which clearly intersect inside the image, i.e. with an epipole in the image. The resulting polar rectified images are shown underneath and are 90 degrees turned for better visualization, hence corresponding lines are now vertical. Because all epipolar lines coincide in the epipole, and the epipole is inside the image, all angles θs have a pixel which is at $r_{min} = 0$. This can be seen from the rectified image which have a full horizontal line at the top (the θ axis is now horizontal because of the 90 degree rotation).

Compared to the planar rectification method, polar rectification has one drawback. Since it describes the image in polar coordinates, the resulting images are no longer formed by a standard camera and can't be computed from the original images with a projective transformation. This means that projective invariants are not valid anymore. This is already clear in Fig. 8.5 where straight lines (e.g. the edges of the markers) are no longer straight.

8.3 Stereo Matching

Dense stereo matching is the technique which tries to find as many pixel matches between the two imagse of a stereo pair. A straightforward method would be to scrutinize every pixel of the first image independently and simply search for the best matching pixel in the second image. The criterion for matching can then be SSD, NCC or other another window-based correlation measure. This approach has the advantage of simplicity and is easy to parallelize. However this approach will yield many-to-one matches and has unpredictable results if different pixels are similar.

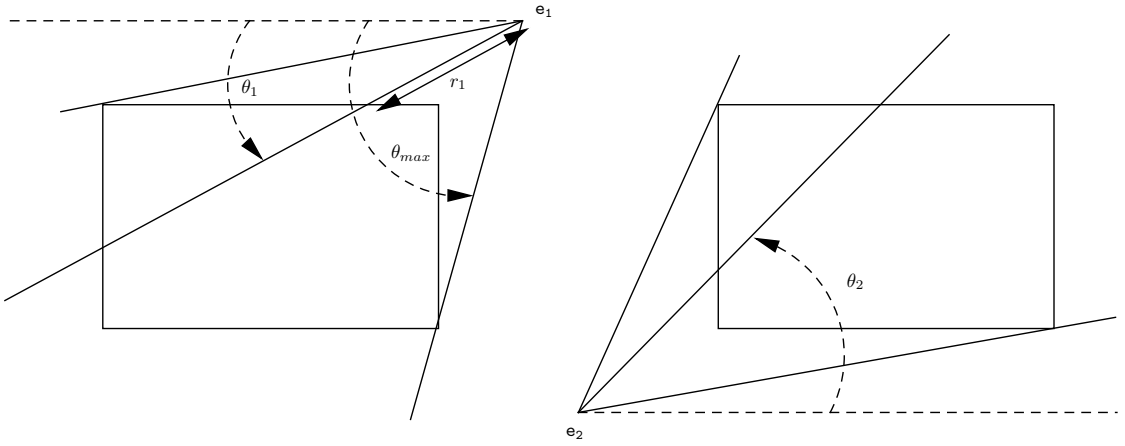


Figure 8.3: Polar Rectification. The epipolar line with angle θ_1 in image 1 corresponds to the epipolar line with angle θ_2 in image 2. The pixels with distance r_1 from the epipole are written on a scanline of the left rectified image and the pixels on the second epipolar line fill the corresponding scanline on the right rectified image.

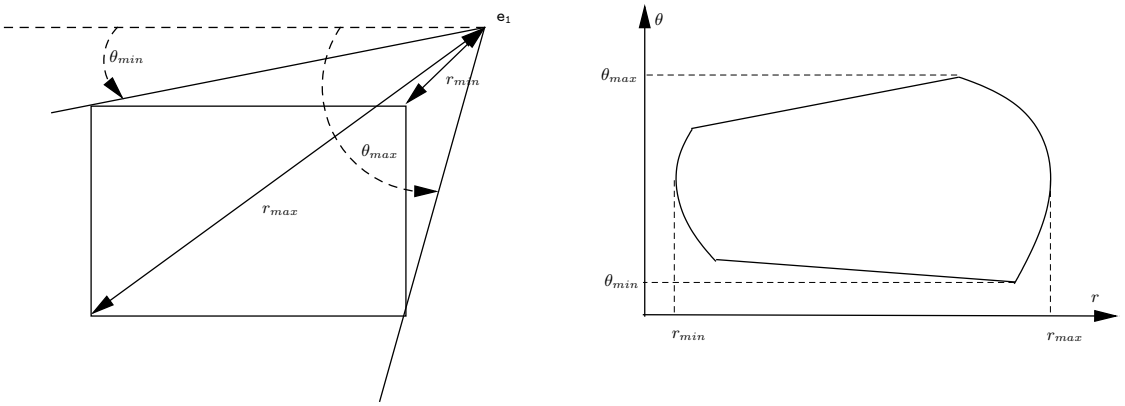


Figure 8.4: Polar Rectification. The size of the rectified image depends on the position of the epipole which fixes θ_{min}, θ_{max}, r_{min} and r_{max}.

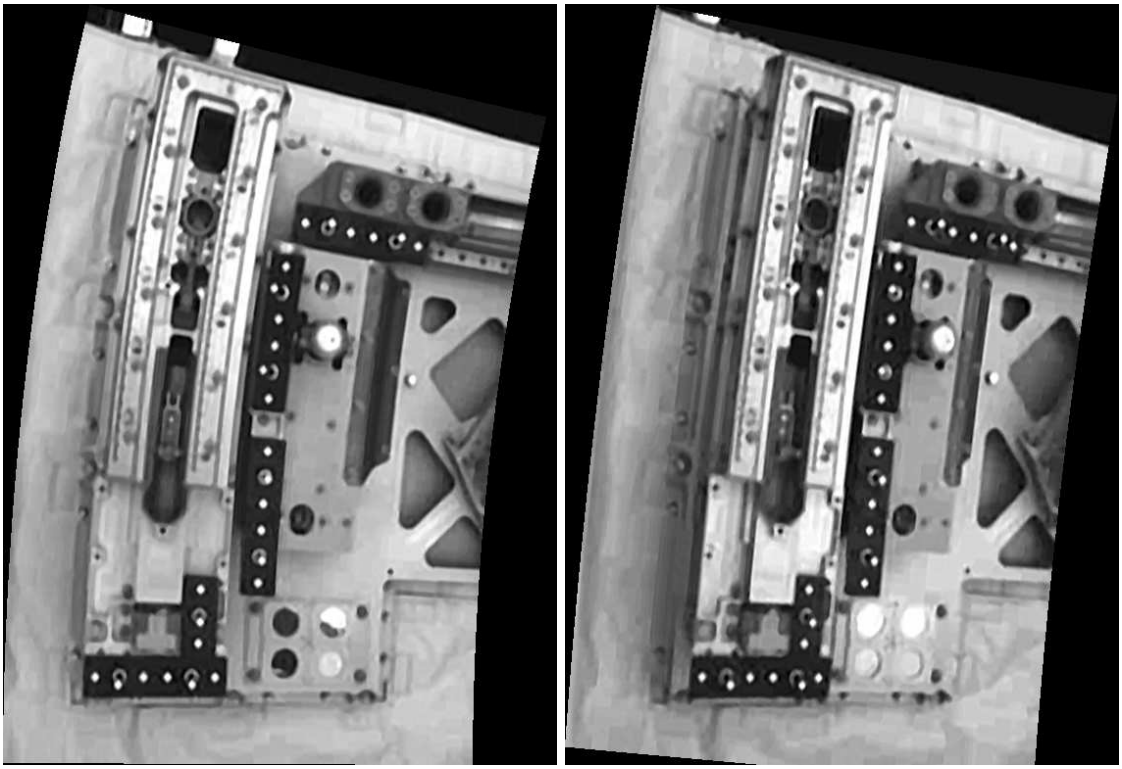


Figure 8.5: Polar rectified images of the *Viable* images. Epipolar lines are horizontal. The polar rectification algorithm clearly breaks projective invariants, since straight lines are no longer straight



Figure 8.6: Original images of the *Street* sequence (top). Epipolar geometry of the pair (middle). The epipoles are clearly located in the image, which poses problems for planar rectification algorithms. Polar rectified images (bottom)

8.3.1 Constraints

This is why stereo algorithms typically take into account several constraints to make the matching more robust. Typical constraints are

- *Epipolar constraint*: if the epipolar geometry is known, matching pixels must only be searched on corresponding epipolar lines. This is why rectification (paragraph 8.2) is so important because it rewarps the images such that epipolar lines become corresponding scanlines and allows the stereo algorithm to only compare pairs of scanlines.

- *Uniqueness constraint*: a 3D point cannot be projected to two 2D points in one image. The stereo algorithm should therefore enforce that a pixel in the second image can only be matched to one pixel in the first image and vice versa.

- *Ordering constraint*: if the 3D scene consists of piecewise planar surfaces, which is commonly the case, the ordering of pixels along epipolar lines is preserved. Because of this constraint, dynamic programming searching techniques [7] can be employed to perform the stereo search.

- *Disparity range constraint*: from the set of reconstructed 3D points which result from the Structure and Motion algorithm, a minimum and maximum depth value can be deduced. These values can be translated to a minimum and maximum disparity range outside which the stereo algorithm does not have to search.

- *Smoothness constraint*: probably the most important constraint because it alleviates the problem of relatively untextured areas in the image. It is also related to the assumption of piecewise planar surfaces. This constraint enforces the disparity to evolve smoothly along the epipolar line except at abrupt changes like depth discontinuities or edges.

8.3.2 Matching Matrix

In order to abide to all the constraints listed above, a global search has to be performed which searches for the optimal global solution to the matching problem. An example of such a global search, implemented as a dynamic programming algorithm is explained in [38]. The most important concept of the matching is of course still the idea that matching pixels should resemble eachother. Therefore, all pixels of one scanline are compared to the pixels of the pixels on the corresponding scanline in the other image, using NCC. Taking into account the disparity range constraint, only pixels with a disparity between d_{min} and d_{max} have to be matched. Fig. 8.7 shows a matching matrix of two scanlines of a stereo pair of the Prato sequence. Only the pixels with a disparity between -72 and 47 are compared, values we deduced from the depth range of the reconstructed 3D points. The goal of the stereo algorithm is now to search for the optimal path inside this matching matrix, taking into account the other constraints.

The algorithm explained in [38] searches for the optimal path in the matching matrix of Fig. 8.7. To traverse the matrix, per step a path can only move one pixel to the right, diagonally to the top-right or to the top. Going back to the left or to the bottom is ruled out because of the ordering constraint. If the path makes a step to the right, this represents an occlusion in the first image, which means this pixel is visible in the second image but not in the first. A step to the top represents an occlusion in the second image while a step to the top-right means a continuous surface. Figure 8.8 illustrates this.

8.3.3 Cost Function

To determine the optimal path, a cost function is assigned to every path which is defined as follows:

$$C(path) = \sum_{\text{matching } x_1,x_2} D(x_1, x_2) + \sum_{\text{left occl}} \beta_{occl} + \sum_{\text{right occl}} \beta_{occl} \tag{8.2}$$

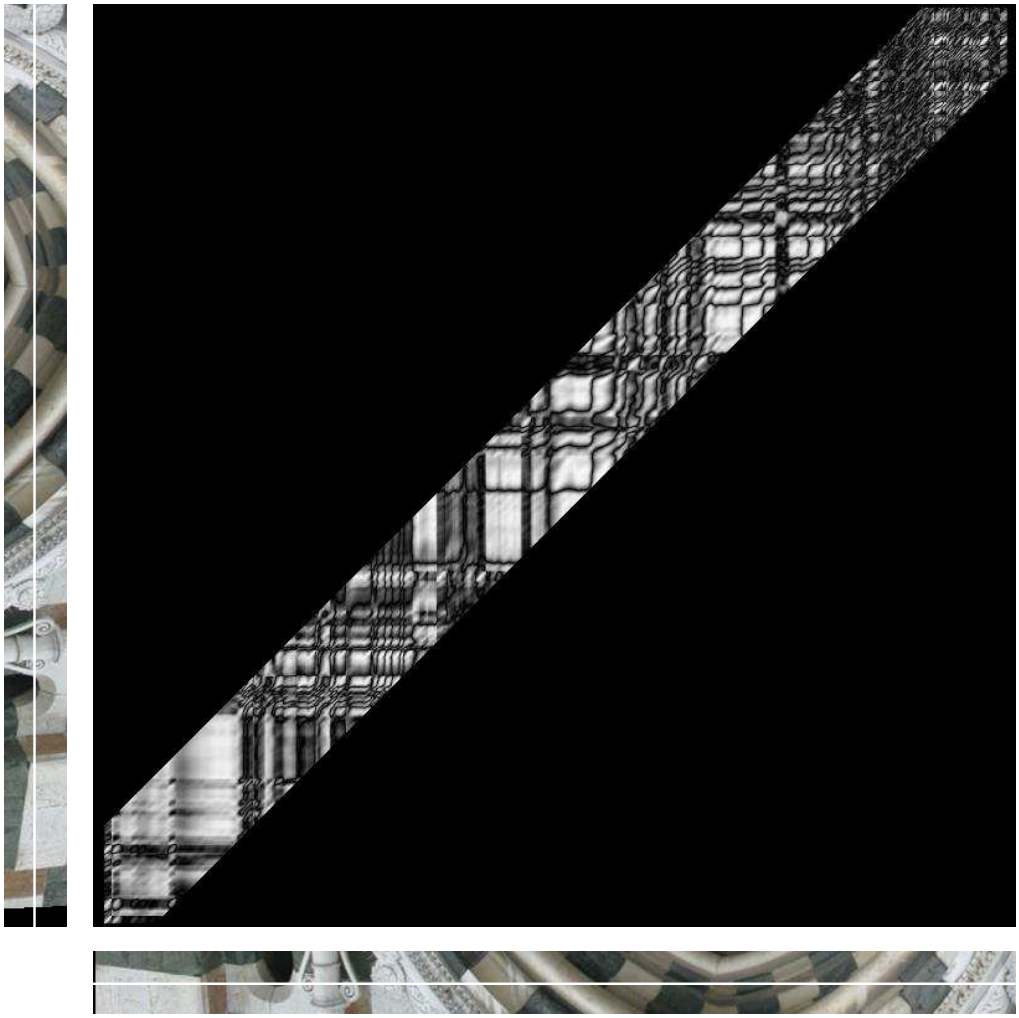


Figure 8.7: Matching matrix of two scanlines of the Prato sequence. For every pixel on the left scanline, all pixels on the right scanline within the disparity range are compared using NCC. This value is put in the correct place in the matrix.

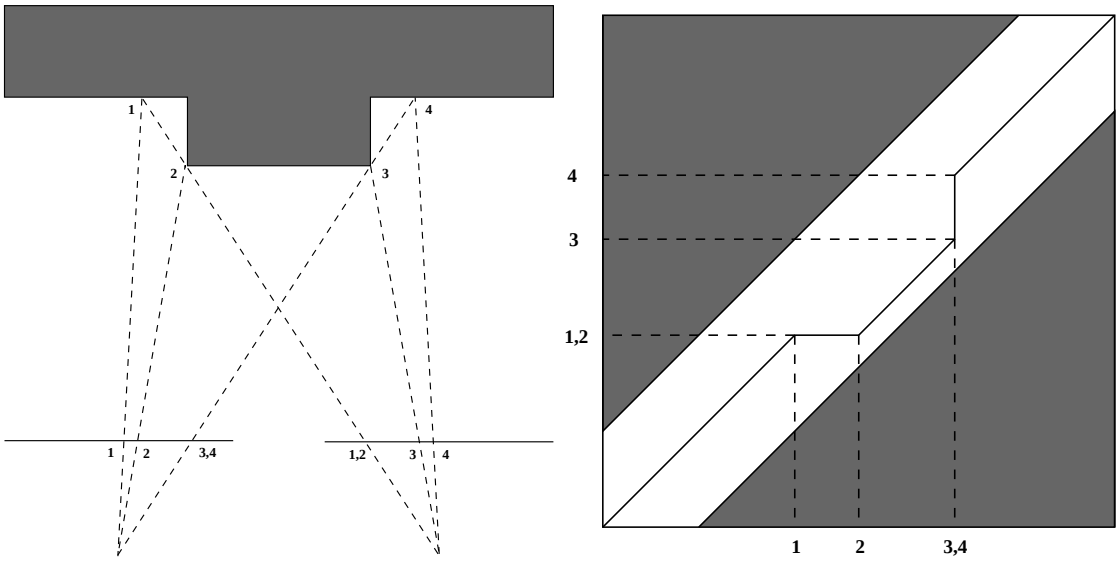


Figure 8.8: The impact of occlusions on the path in the matching matrix. The left figure shows a top view of a scene recorded in two images of a stereo pair. The right rigure shows the path in the matching matrix. Point 3 and 4 fall coincide in the left image, which means that all pixels between 3 and 4 in the second image are not visible in image 1. Hence, a vertical line is found in the path between these points. Analogically point 1 and 2 are the same point in image 2 which means the area between them is occluded in this image and the horizontal piece of the path is explained.

First we go look at all pixels in the first scanline. Some of these pixels have matching pixels in the other scanline and others are occluded. The first term is the sum of all *dissimilarities* for the matching pixels x_1 and x_2. The dissimilarity can be defined as $1 - NCC(x_1, x_2)$ with NCC the normalized cross correlation of two windows centered around the two pixels with a typical size of 5×5 or 7×7. The second term gives a penalty β_{occl} to all occluded pixels. Finally we look at all the pixels in the second scanline and add a penalty β_{occl} for all occluded pixels. The value of the parameter β_{occl} determines how much weight we give to occlusions. If this value is high, we give a high penalty to an occlusion, forcing the path to be smooth. Lower values give more weight to the matching term, accepting only well matched pixels but increasing the amount of occlusions and thus unmatched pixels. The stereo algorithm will then calculate the cost for the different paths and select the cheapest one as the optimal solution. Dynamic programming and backtracking techniques are used to increase the performance. A typical improvement consists in making the disparity range symmetric by shifting one of the epipolar lines left or right. This improves the memory-allocation efficiency.

8.3.4 Hierarchical Stereo

The complexity of the search algorithm described above is $O(WH\delta^2)$ with W and H the width and height of the image and δ the disparity range. Large disparity ranges therefore have a funest influence on the execution time. The chance on mismatches within large disparity ranges increases as well. This is the reason why the stereo algorithm is executed in a hierarchical way. The rectified images are subsampled a number of times, depending on the disparity range. At the highest (i.e. smallest) level, disparity range is limited to e.g. $[-5, 5]$. The optimal path is searched for at this level and the solution is used as an initialization for the the next level. At this next level we limit the disparity range again to a fixed width of e.g. $[-5, 5]$ around the initialization. By executing this algorithm for all levels, using the solution of the previous level as the initialization for the next, we can compute the optimal path at the last level with a complexity of $O(WH)$. As an

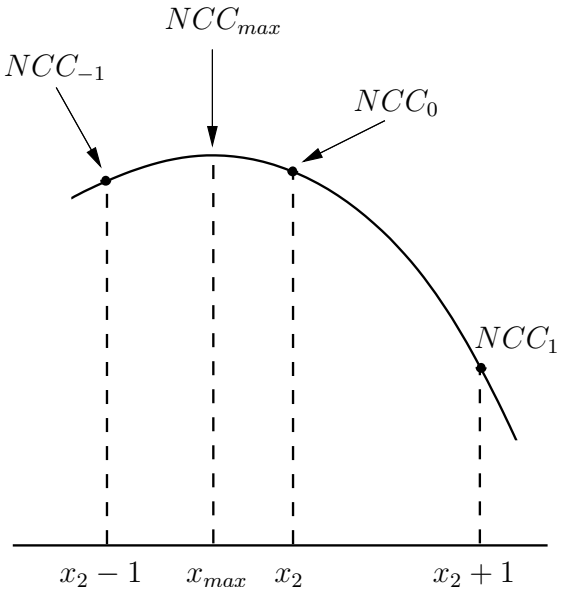


Figure 8.9: The optimal path of the dynamic programming solution says pixel x_1 in the left scanline matches to x_2. We compute the NCC with $x_2 - 1$ and $x_2 + 1$ as well and fit a parabola through the 3 NCC values. The position on the scanline x_{max} where the parabola reaches its maximum is taken as the subpixel match of x_1.

extra advantage we ensure that, by starting the matching with the smallest image, large disparity values can be dealt with too. A downside of this technique is of course that a possible mismatch at the highest level can never be undone in the next levels if this mismatch is more than 5 pixels (for a fixed range of $[-5, 5]$) wrong in the next level.

8.3.5 Post Processing

Even though the dynamic programming algorithm searches for the optimal solution of the path, some post-processing on this result is still possible. The step size of the matching in the algorithm is one pixel. We can improve this to sub-pixel accuracy by combining the information of neighboring matched pixels. A possible technique to refine the match between pixels x_1 and x_2 consists in matching x_1 with $x_2 - 1$ and $x_2 + 1$ using NCC. A parabole can then be fitted through the three values NCC_0, NCC_{-1} and NCC_1 and the x value where this parabola reaches its maximum can be selected as the floating point, sub-pixel match of x_1 on the second scanline. This technique is illustrated in Fig. 8.9.

The output of the stereo algorithm is a disparity map between the rectified images2. This means that in order to find the matching pixels between the original images, the transformations depicted in Fig. 8.10 have to be transformed. The pixel (x_1, y_1) is transformed to its rectified coordinates (x_{1r}, y_{1r}). A lookup in the disparity map tells us how many pixels we have to move to the left or to the right on the same scanline and yields the matching pixel in the second rectified image with coordinates $(x_{2r}, y_{2r} = y_{1r})$. Inverse rectification of these coordinates then yield the coordinates (x_2, y_2) of the matching pixel in the second image. Applying these transformations time after time can be time-consuming task. If this lookup has to be done regularly, it is better to construct two displacement maps from the disparity map. These maps hold the x and y displacement of the pixels in the first image that have to be added to the original coordinates to end up with the coordinates of the matching pixel in the second image.

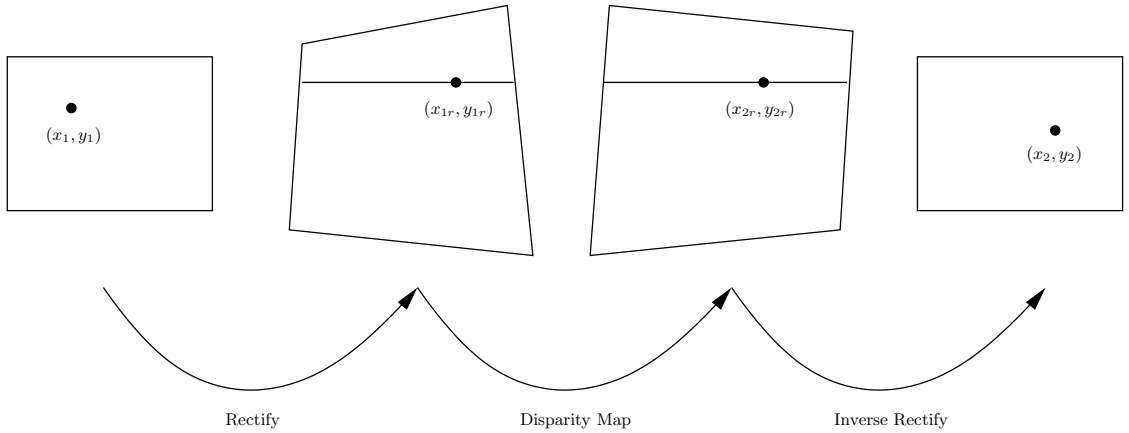


Figure 8.10: The transformations one has to apply to find the matching pixel in the second image comprise of a rectification step, a lookup in the disparity map and an inverse rectification.

8.4 Linking Stereo Pairs

Section 8.3 explained how to compute dense matches between pairs of images. The input of Structure and Motion algorithms however typically consists of multiple images for which SaM computes the cameras and a 3D point cloud reconstruction. This means that there are multiple camera pairs to be considered. In this section we will explain an algorithm, proposed in [30], that allows to link the information of several stereo pairs together in order to create dense depth maps.

Consider a set of images, taken in sequential order and processed by the Structure and Motion algorithms described before. For every image pair, consisting of two consecutive images in the sequence, a dense stereo matching process can compute dense disparity maps or displacement maps. Because the consecutive images are relatively close together, they are similar which helps the matching process to yield accurate correspondences. Unfortunately, a 3D reconstruction based on the dense matches of such an image pair will be fairly inaccurate, due to the relative small baseline between the views, as we can understand from Eq. 1.12 or from Fig. 8.11. The uncertainty ellipsoid on the 3D point reconstructed from $\mathtt{m}_1$ and $\mathtt{m}_2$ with cameras $\mathbf{P}_1$ and $\mathbf{P}_2$ can be computed as follows. We assume we have matched $\mathtt{m}_1$ and $\mathtt{m}_2$ with an uncertainty in the images of 1 pixel. We move these points one pixel in both directions on the epipolar line and backproject these points as well. The 3D points with the maximum deviation from the reconstructed point are chosen and form the uncertainty ellipsoid. Image n, with a larger baseline is considered as well. Because of the larger baseline, the angle between the backprojected rays is larger which decreases the size of the uncertainty ellipsoid, indicating that the error on the reconstruction is smaller.

We see that large baselines yield smaller errors on the 3D reconstruction. However, matching wide baseline views poses many difficult problems. Because the images are taken further apart, they are less similar which makes them harder to match. A matching algorithm using simple comparison techniques like NCC will have a hard time finding correspondences because the affine deformation of the surfaces is not taken into account. Also, because of the wider baseline, the chance on occlusions increases and thus more image points will not be matched.

The algorithm that is proposed in [30] tries to retain the best of both worlds by doing small-baseline matching but combining several of these matches to increase the reconstruction baseline. The concept is clearly illustrated in Fig. 8.12. Each image k in the sequence for we want to compute a dense depth map is selected once as central view. The displacement maps between views k and $k+1$ are used to instantiate the 3D points between matching pixels in these views. Because of the small baseline we expect many corresponding points but also a large error distance e_k. After this initialization step, we move one view to the right. We use the displacement maps between views $k+1$ and $k+2$ to check whether reconstructed 3D points have a match in image $k+2$. For those who do, we reconstructed the 3D point from images k and $k+2$. If this 3D

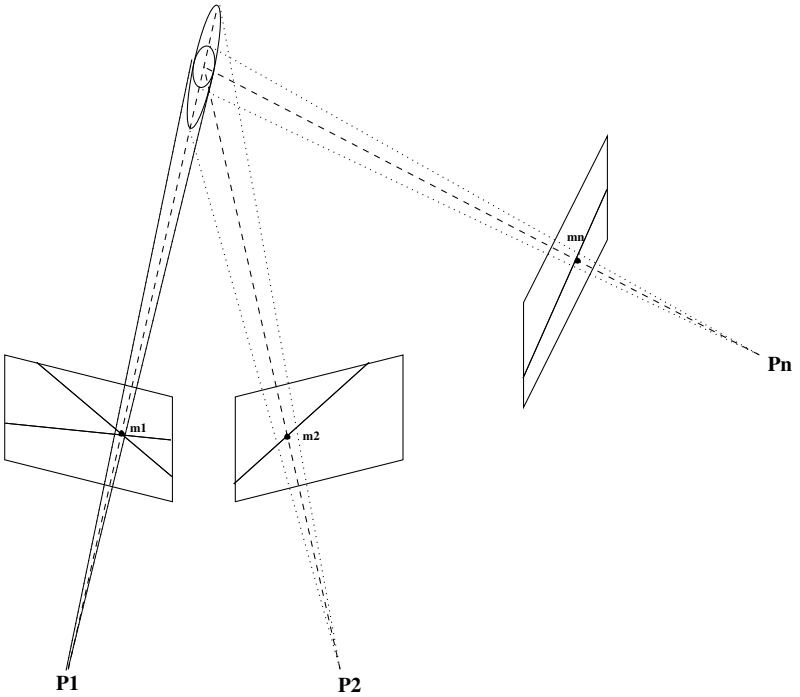


Figure 8.11: The uncertainty ellipsoid on the reconstruction from a small-baseline image pair (1,2) is large in the direction of the backprojected rays. For an image (n) with a larger baseline w.r.t. image 1, the angle between the backprojected rays is larger which leads to a smaller uncertainty.

point lies within the error bound e_k we accept it, update the position of the point accordingly and also update the error to e_{k+1}. If the reconstructed 3D point falls outside e_k, we no longer have a consistent match and we break the link. This process is repeated until we hit the rightmost image or until no more 3D points are linked. The process is then repeated for the images on the left side of the central view, updating the already reconstructed points but also instantiating new points, that were not reconstructed between images k and $k+1$. The update equations of the 3D point and the error are derived from the well known Kalman filtering equations [29]. A discrete system with state $\mathbf{x}$, input $\mathbf{u}$ and output $\mathbf{z}$ is governed by the state equations

$$\mathbf{x}_n = \mathbf{A}\mathbf{x}_{n-1} + \mathbf{B}\mathbf{u}_{n-1} + \mathbf{w}_{n-1} \quad (8.3)$$

$$\mathbf{z}_n = \mathbf{H}\mathbf{x}_{n-1} + \mathbf{v}_{n-1} \quad (8.4)$$

where $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{H}$ define the system and $\mathbf{w}$ and $\mathbf{v}$ are vectors with white noise. Kalman filtering allows to estimate the state $\mathbf{x}$ and the error covariance $\mathbf{V}$ of the output by observing the output, i.e. taking measurements. The general equations are

$$\mathbf{K}_n = \mathbf{V}_{n-1}\mathbf{H}^T(\mathbf{H}\mathbf{V}_{n-1}\mathbf{H}^T + \mathbf{R})^{-1} \quad (8.5)$$

$$\hat{\mathbf{x}}_n = \hat{\mathbf{x}}_{n-1} + \mathbf{K}_n(\mathbf{z_n} - \mathbf{H}\hat{\mathbf{x}}_{n-1}) \quad (8.6)$$

$$\mathbf{V}_n = (\mathbf{I} - \mathbf{K}_n\mathbf{H})\mathbf{V}_{n-1} \quad (8.7)$$

where $\mathbf{K}_n$ is called the Kalman gain, $\hat{\mathbf{x}}_n$ is the estimate of the state at time n and $\mathbf{R}$ is the noise covariance of the measurement. In our case the state equations are very simple. The state is the depth of the 3D point D. There is no input in the system which means $\mathbf{A} = 1$ and $\mathbf{B} = 0$. The observed output is the depth of the point, reconstructed with the last camera, thus $\mathbf{H} = 1$. The Kalman equations therefore become

$$K_n = \frac{V_{n-1}}{V_{n-1} + R} \quad (8.8)$$

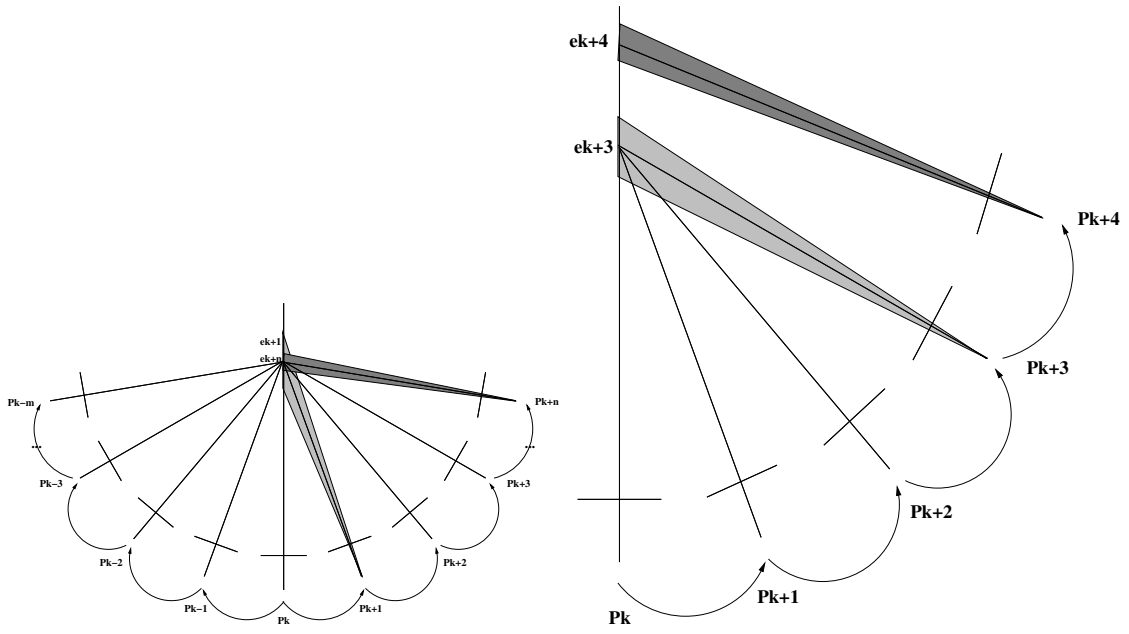


Figure 8.12: The linking algorithm illustrated. The left figure shows a central view k. The 3D point is initiated from the match with view $k+1$ and the error e_{k+1} is computed. From the matches between views $k+1$ and $k+2$, the matching point in view $k+2$ is found and the 3D point and error band are updated. This process is repeated until no more match is found or until the reconstructed point from the new view falls outside the error margin as shown in the right figure. The linking is executed for both views to the right and the left of the central view.

$$\hat{x}_n = \hat{x}_{n-1} + K_n(D_n - \hat{x}_{n-1}) \tag{8.9}$$
$$V_n = (1 - K_n)V_{n-1} \tag{8.10}$$
$$= K_n R \tag{8.11}$$

The linking approach of the dense stereo matches combines the strong points of small- and wide-baseline matching. However, it is only applicable if the set of images is recorded and processed sequentially. If this is not the case, the prerequisites are not fulfilled and the algorithm gives bad results. This can for example be the case if the images don't form a sequence but just a random set in which it is not certain that consecutive images are also close. Another problem arises when two consecutive images are very close together so that 3D points are badly instantiated, which the linking algorithm is not able to recover from. Other approaches are able to deal with multiple images at the same time and combine the information in the different images in one go to yield dense matches. Two approaches are explained in sections 8.5 and 8.6.

8.5 Fast Matching on the GPU

8.5.1 Why GPUs

Nowadays, cutting edge CPUs are capable of five gigaflops. In 1989 the worlds fastest million dollar supercomputer,The Cray 3, achieved this processing power. 15 Years later you can buy that performance for less than $500 at your local computer store. This opens the possibility of richer user environments and more complex real-time digital processing. On the one hand, for the CPU industry, Moore's Law [41] still defines the current and expected increases in growth rate until 2020. On the other hand, due to the stream processor architecture of GPUs and advances in graphics hardware, mainly driven by the game industry, the growth curve for GPUs corresponds

roughly to Moore's Law cubed, doubling processing speed approximately every six months.

Graphics boards have grown to become fully featured processing units with dedicated high performance memory and up to 16 parallel pipelines. The introduction of vertex and fragment programs caused a major breakthrough, giving the programmer almost complete control over the GPU. They provide a large instruction set, comparable to the instruction sets of CPUs, and can be programmed in both assembly and high level languages [50]. Also, the expanding list of new features allow more and more software algorithms, most of them which rely on SIMD (single instruction, multiple data) instructions, to be implemented on a graphics card with a substantial performance gain w.r.t. their software based counterparts. All these features, combined with the increased computational performance gain over time w.r.t. CPUs, make graphics hardware an attractive platform for a vast number of algorithms.

In this section we will explain an algorithm, proposed in [12], which computes depth maps on GPUs. This algorithm computes the depth maps in real-time using a plane-sweep method and can avoid artifacts exhibited by other implementations.

8.5.2 Basic Setup and Conventions

The input to the algorithm consists of two or more images for which the respective internal and external camera parameters are known in a common coordinate frame. These images can originate from pictures from a static scene, possibly taken at different times, or from synchronized cameras, taking pictures of a possibly dynamic scene. For one of these images, referred to as the reference image, we wish to compute the depth map in a fast, robust way. In the remainder of this text, we will refer to the remaining images as sensor images.

The internal camera parameters include radial distortion parameters. Because radial distortion is a non-linear process and can possibly harm our real-time goal, we want to eliminate it. Therefore we preprocess the images such that they are converted to the equivalent images of ideal pinhole cameras. Provided the radial distortion parameters do not vary with time, this preprocessing step can be implemented efficiently by precomputing a lookup texture containing the new undistorted texture coordinates for each pixel.

In order to find corresponding points between image pairs, formed by the reference image and each of the sensor images, we are in need of a dissimilarity measure. In case only two images are used, the sum of squared differences(SSD) is a suitable measure. If more than two images are used, we propose the error measure, computed according to Eq. 8.12.

$$e = \sum_{i=0, i\neq r}^{nr} \min(m, (r_i - r_r)^2 + (g_i - g_r)^2 + (g_i - g_r)^2) \tag{8.12}$$

Where r denotes the index of the reference image, nr equals the number of images and m is an arbitrary threshold, indicating the maximum radius of influence in RGB-space. The saturation of the SSD scores is needed to account for occlusions by constraining the influence of outliers on the total SSD score. Using color information images results in a better dissimilarity measure compared to using grayscale images. Keeping the color channels separated results in more discriminant power for the dissimilarity measure.

8.5.3 Hierarchical Plane-Sweep Algorithm

In [66] a plane-sweep algorithm is explained that can be used to compute dense matches efficiently on a GPU. The idea is simple and is illustrated in Fig. 8.13. One reference image and an arbitary number of sensor images are looking at the same scene. A plane is swept along the optical axis at a number of discrete steps. For every pixel in the reference image the backprojected ray is intersected with the plane. This point is projected into the sensor images and these projections are compared to the original images. For each pixel in the reference image the depth value, corresponding to the depth that provides the lowest dissimilarity measure, is retained. Additionally, we use

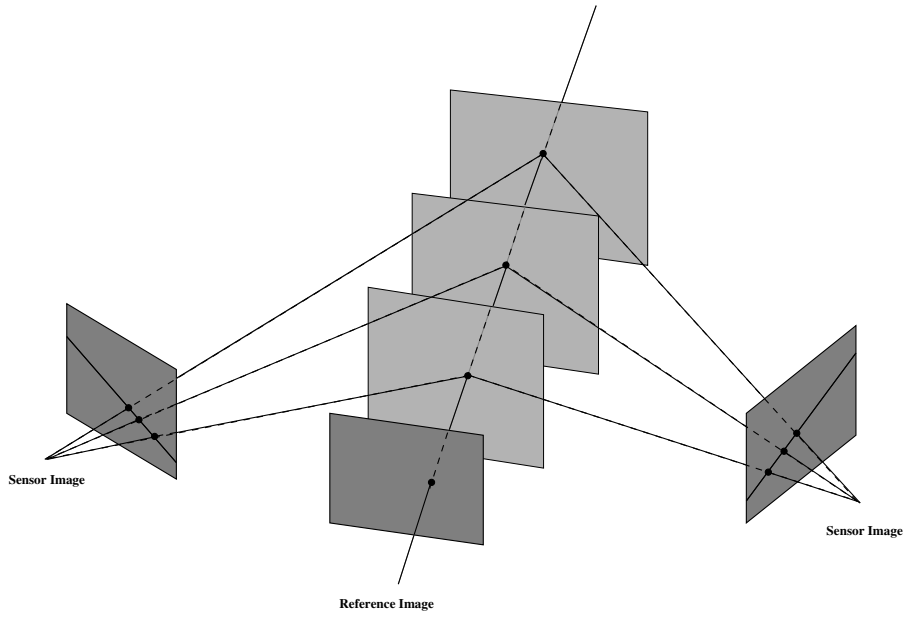


Figure 8.13: The Plane-Sweep algorithm. A plane is swept along the optical axis of the reference image. For every plane position the points in the reference image are transformed to the sensor views. For every pixel the depth that provides the lowest dissimilarity measure is retained.

a hierarchical approach to speed up the computations and typically start at a hierarchical level which spans $1/64th$ the area of the original reference image.

The algorithm does not really compute the intersections of the backprojected rays with the plane but actually transforms the reference image via the plane to the sensor images. This homography-mapping can be performed extremely efficiently on the GPU. Mipmapping is disabled because in the presence of high frequency textures, the information provided by the lower mipmap levels is little to none compared to the performance overhead. If for instance an object in the scene has a uniformly distributed high frequency color pattern, mipmapping will quickly reduce the pattern to an overall constant color.

We can make the algorithm hierarchical in depth resolution by adding small offsets to the depth map. These offsets depend on the iteration count, the hierarchical level and the number of planes in the initialization step. For each iteration we construct a list of depth offsets to be tested. By moving up a level after a fixed number of iterations, the algorithm becomes hierarchical in screen space resolution as well.

8.5.4 The Connectivity Constraint

In most cases, the initial depth map will contain false depth estimates in the presence of homogeneous regions, occlusions and ambiguities. Because of the hierarchical nature of the algorithm, false depths at an early stage of the algorithm will have a significant influence on the final depth estimation at full resolution. Therefore, we add connectivity information at each level to reduce the amount of false depths. We do this by incorporating a heuristic: if the pixel under inspection belongs to the same object as one of its neighbors, the most probable depth for the pixel is either the same depth as its neighbor with optionally a small offset.

The connectivity algorithm works as follows. For each pixel, the algorithm fetches the depths of neighboring pixels in a 3×3 neighborhood according to one of the sampling patterns, shown in Fig. 8.14. The heuristic is implemented by computing the error value for each of these depths and their offsets. The error and depth value corresponding to the lowest error value are stored at the position corresponding to the pixel under inspection. Finally the resulting values for all pixels are

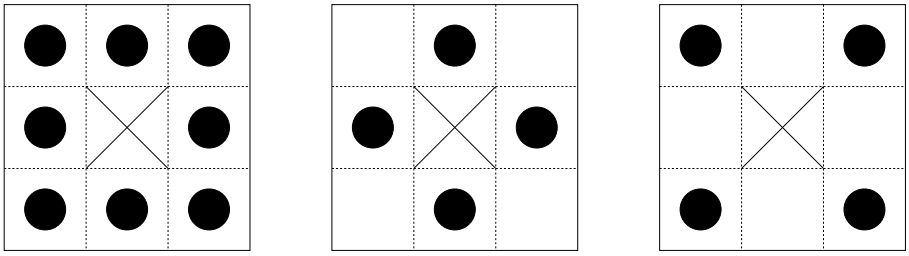


Figure 8.14: Three neighbor-sampling patterns.



Figure 8.15: Left to right: Resulting depth maps after 0,1 and 2 connectivity passes.

copied back into the depth map. This process can be repeated several times at the same pyramid level.

In order to remove isolated, floating points the depth corresponding to the pixel itself is now sampled such that the sampling pattern acts as a median filter. At each iteration, a depth approximating the depth of a neighboring pixel is assigned to the pixel under inspection. Because the neighboring pixels undergo the same routine, it is possible that the depth, corresponding to the current pixel, is assigned to that neighboring pixel. Assuming we use the left pattern in Fig. 8.14 the algorithm would allow flip-flopping of a depth value between two pixels which have an isolated depth w.r.t. their surroundings. To avoid this undesired effect, we alternate between the middle and right sampling pattern at each iteration. By doing this, it requires at least three iterations for a depth value of a pixel to return to that pixel once again, allowing its neighbors to treat its depth as an outlier by filtering it out. In contrast to a median filter, this approach can also provide the pixel with a new, better depth at the same time and it is computationally less expensive. At the end of this step, the depth map will mainly consist of large connected patches. The connectivity algorithm is illustrated by Fig. 8.15 which shows the resulting depth maps after 0,1 and 2 connectivity passes. The amount of erroneous depth values clearly diminishes.

8.5.5 Retrieval Of Fine Structures

As described in paragraph 8.5.4, the depth map was filtered much like a median filter in order to remove isolated, floating points. It is possible that in this step, some pixels with correct depth values were filtered because their surrounding pixels had incorrect depth values and not the pixels themselves, causing them to be filtered out. In subsequent stages of the algorithm, we wish to retrieve these correct depth values, providing they are connected either directly to a larger surface or indirectly, through previously retrieved fine structure points. We do this by sampling the depth values of all the pixels in a 3×3 neighborhood, rather than applying the sampling patterns. Because at the end of the filtering step the depth map consists of large connected patches, the sampled depths for a pixel will belong to one of these patches. This list of depth samples is once

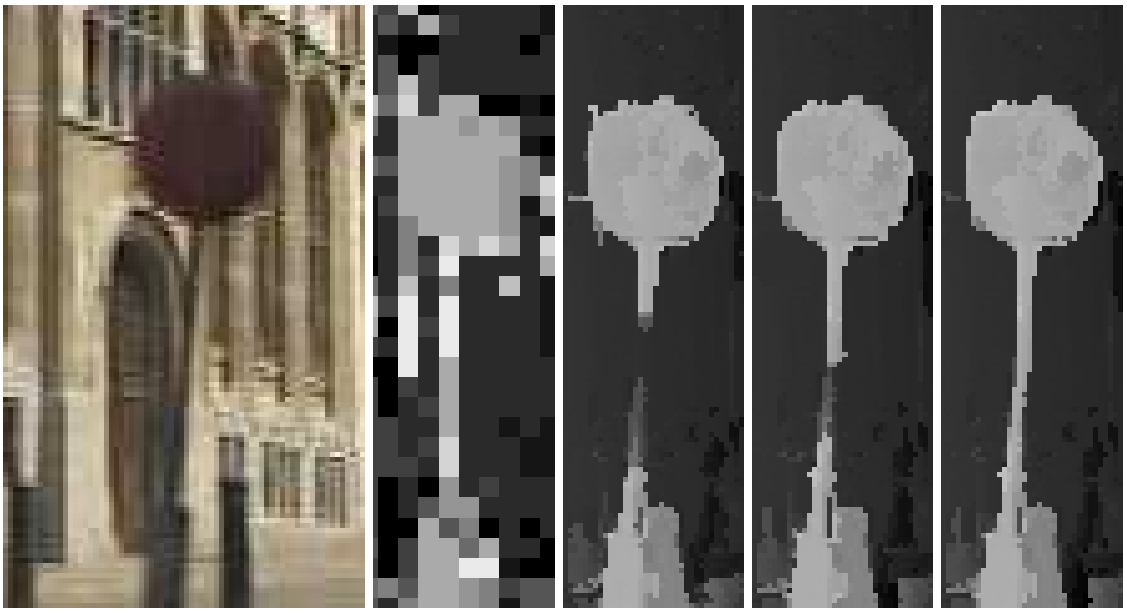


Figure 8.16: Left to right: Final depth map after 0,4,8 and 12 passes per hierarchical level respectively.

more expanded and tested in the same way as described in paragraph 8.5.4.

Figure 8.16 illustrates the retrieval of fine structures for a set of images of a traffic sign. It is clearly noticeable in the third image that most of the outliers are filtered out, but also a number of inliers. The third and fourth image show how the complete traffic sign is retrieved afterwards while the algorithm searches for new points, originating from the base and the top of the traffic sign.

8.5.6 Smoothness Penalty

The proposed algorithm behaves satisfactory in the presence of high frequency textures and occlusions, but the estimated depths for homogeneous regions remain noisy. To remedy this, we add a smoothness penalty to the dissimilarity value, computed as follows. After each iteration, the resulting depth map is passed through a 3×3 gaussian filter and stored in a separate depth map. For each depth tested by a pixel, the distance d_s to the smoothed depth map is calculated. Based on this distance, the following penalty is added to the dissimilarity value:

$$e_{new} = e + \max(|d - d_s|, d_{max}) \tag{8.13}$$

where d_{max} represents the maximum distance of influence.

When all pixels in the 3×3 neighborhood have approximately the same depths, which is the case for median filtered homogeneous surfaces, all the depths to be tested are small variations of d_s, causing the penalty to give preference to the depths closer to d_s and therefore smoothing the surface. If the 3×3 neighborhood contains outliers, which could originate from fine structures, the tested depths will differ from d_s by a large amount. In case this amount is larger than d_{max}, the smoothness penalty has no effect, avoiding the filtering of fine structures based on the smoothness constraint. The possibility remains that an erroneous tested depth approximates d_s by coincidence, influencing the similarity value. However, due to the constant change of the d_s value at each iteration, this erroneous influence has a high chance of being corrected in the following iterations. Figure 8.17 illustrates this smoothing effect.

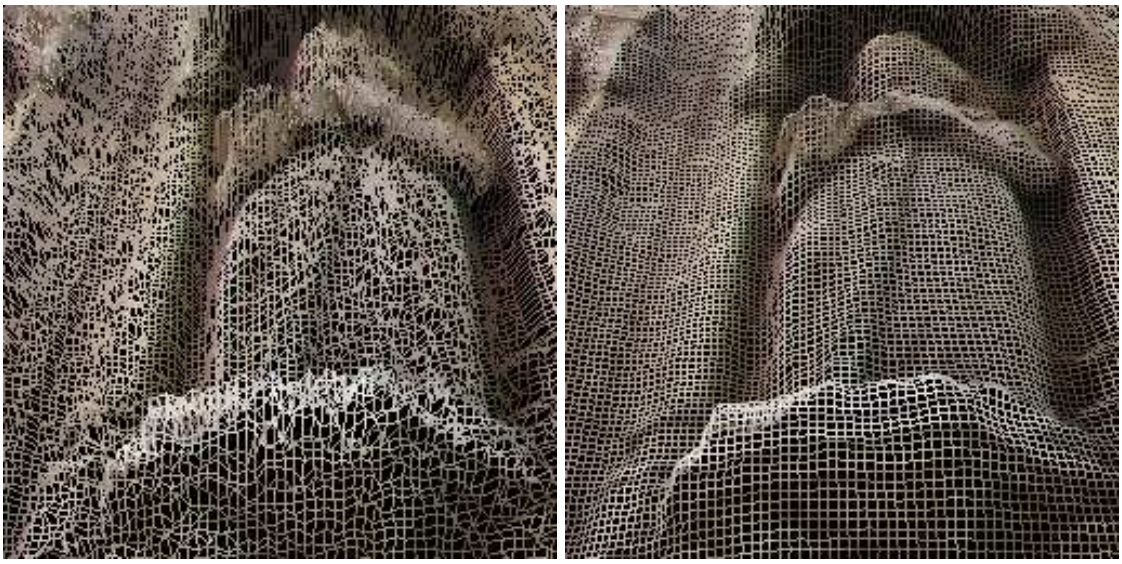


Figure 8.17: Left: Wireframe model without smoothness penalty. Right: With smoothness penalty. The spikes in the homogeneous regions noticable on the left are no longer present in the smoothed version.

8.6 Bayesian Multi-View Matching

A true multiview dense matching approach, like the one of section 8.5 implemented on GPUs, typically considers one reference view and takes into account matches with all other images in the set for which we have recovered the calibration. Although GPUs are very good at performing operations on entire images, their programming capability is still limited when it comes to conditional code, i.e. making decisions and performing different operations on pixels depending on certain conditions. For well behaved image-sets the GPU algorithms can very rapidly compute qualitative depth maps. However, when the images span larger baselines or when more intricate scenes are recorded, the simple algorithms on the GPU will not suffice.

Algorithms exist that combine multiple views, often taken from all around the object. Examples are voxel carving [32], photo hulls [53] and level sets [16]. Several of these approaches use a discretized volume and restrict possible depth values to a predefined accuracy. This is not the case for pixel-based PDE approaches [3, 56], which do not need 3D discretization and compute a continuous depth for every pixel. For large images with fine details, it is questionable if volume based algorithms can combine reasonable speed and memory requirements with high accuracy.

Inherent to the wide-baseline setting is the problem of occlusion, which means that not all parts of the scene, which are visible in a particular image, are also visible in the other images. Also, because of the large differences in viewpoints, we have to consider the possibility that image pixels in different images, which are projections of the same point in the scene, will have different color values due to non-diffuse reflections. For these situations, a Bayesian multi-view matching approach was developed in [55]. Here, the wide-baseline stereo problem is addressed from a probabilistic point of view. We primarily focus on the occlusion part of the problem, that is to say, we assume that we are dealing with mainly diffuse objects, and all deviations from this model are caught by a noise term. In the proposed algorithm, each input image is regarded as a noisy measurement of an unknown image irradiance or *true* image, which is estimated as part of the optimization problem. This image combines the information from all other views and can be used as a texture map for the final reconstruction.

When we want to estimate the depth map of the reference image, we again use the same paradigm as in section 8.5 where the depth of an image-point is chosen such that the color information of correspondences with other views is similar. However, we have to take into account

possible occlusions in other views. In order to do so, we introduce a visability map $\mathcal{V}_i(\mathbf{x}_1)$ for every sensor image which has value 1 at pixels of the reference view that are visible in the sensor image and value 0 elsewhere. Because we regard every image as a noisy measurement, we end up trying to estimate the depth map for the reference view $\mathcal{D}_1$ and all visibility maps which make the image correspondences for pixels with visibility 1 most probable. The problem is thus stated as the computation of the depth map , the *true* images and the visibility maps which are introduced as hidden variables. Solving this problem is far from trivial and boils down to optimizing a Bayesian framekwork. In [55] this is done with Estimation-Maximization techniques which fall out of the scope of this text. Fig. 8.18 shows the input images and results of the Bayesian multi-view matching method.

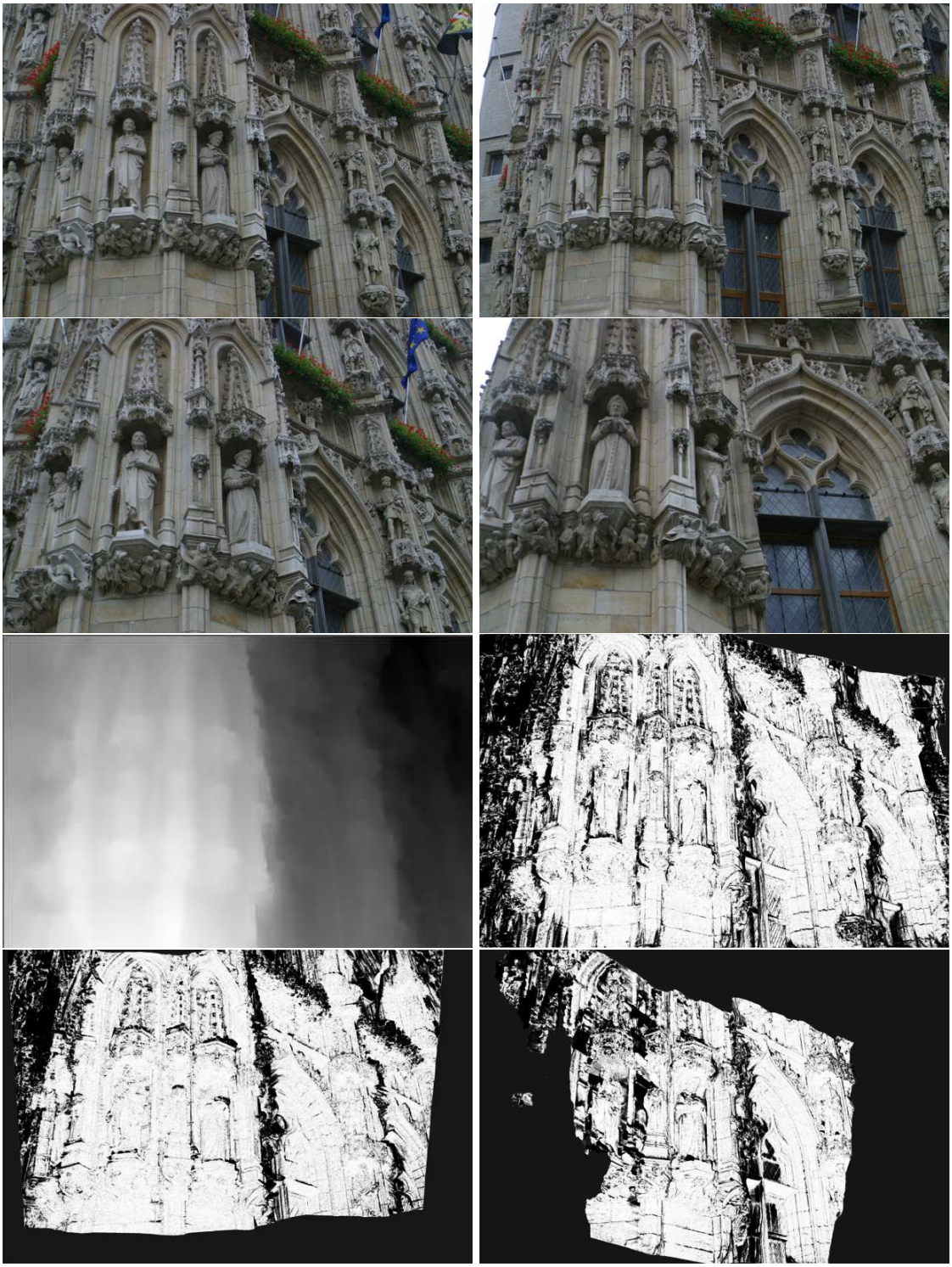


Figure 8.18: **Cityhall scene**: *The top two row show the 4 original input images. The bottom 2 display some of the output of the algorithm, where the* 1st *image of the sequence* ($\mathcal{I}_1$) *was taken as the reference view. The fifth image is the depth map* $\mathcal{D}_1$ *of* $\mathcal{I}_1$*. The last* 3 *images are the visibility maps* $\mathcal{V}_2$*,* $\mathcal{V}_3$ *and* $\mathcal{V}_4$*, which signal which pixels in* $\mathcal{I}_1$ *are visible in the other images.*

Chapter 9

3D Webservice

9.1 Introduction

9.1.1 3D Technologies in the Cultural Heritage Field

The rise of new information technologies has led to the development of many applications in the cultural heritage area. By now, it is commonly accepted that ICT applications in general and 3D reconstruction and rendering technologies in particular carry large potential for this field. Especially the power to acquire accurate and realistic 3D representations of objects and scenes is highly valued, for it offers the possibility to preserve cultural heritage for generations to come. The European Research Network of Excellence EPOCH is a network of about a hundred European cultural institutions joining their efforts to improve the quality and effectiveness of the use of Information and Communication Technology for Cultural Heritage. Based on inquiries of the cultural heritage field and technical partners, a research agenda has been composed which clearly indicates needs that are present in this area and how they relate to existing technologies. One of the elements in this agenda is versatile 3D scanning. Scanning technologies should become cheaper and more flexible.

9.1.2 Low-cost Pipeline for 3D Reconstruction

Two EPOCH partners, the ESAT-PSI lab of K.U.Leuven (Belgium) and the Visual Computing Lab of CNR-ISTI (Italy) have been active for a long time in the field of 3D reconstruction, albeit each in their own specific sub-area, passive reconstruction from images and active reconstruction with laser-scanning respectively. Both labs decided to combine strengths and set-up a low-cost 3D reconstruction pipeline to be used in the cultural heritage field. The idea is that only a digital photo-camera and an Internet connection are necessary prerequisites for a user to reconstruct scenes in 3D. Fig. 9.1 shows a schematic overview of the complete pipeline. It consists of three services that are performed on servers, accessible via the Internet and fed by data from the user and two local applications. Images of the object of scene to be reconstructed are first checked for quality, then uploaded to the first web service which computes dense range maps. A second web service is capable of merging these range maps. The user can also align range maps from different image sequences locally and merge them. The third web service is in charge of mesh creation and simplification.

9.1.3 Paper Overview

This paper describes one aspect of the web-based 3D reconstruction service that is being developed to relieve the needs of the Cultural-Heritage field for low-cost and easy to use 3D reconstruction of objects and scenes. It only deals with the first web-service of Fig. 9.1: the computation of dense range (or depth-) maps from images uploaded by the user. The outline of the paper is

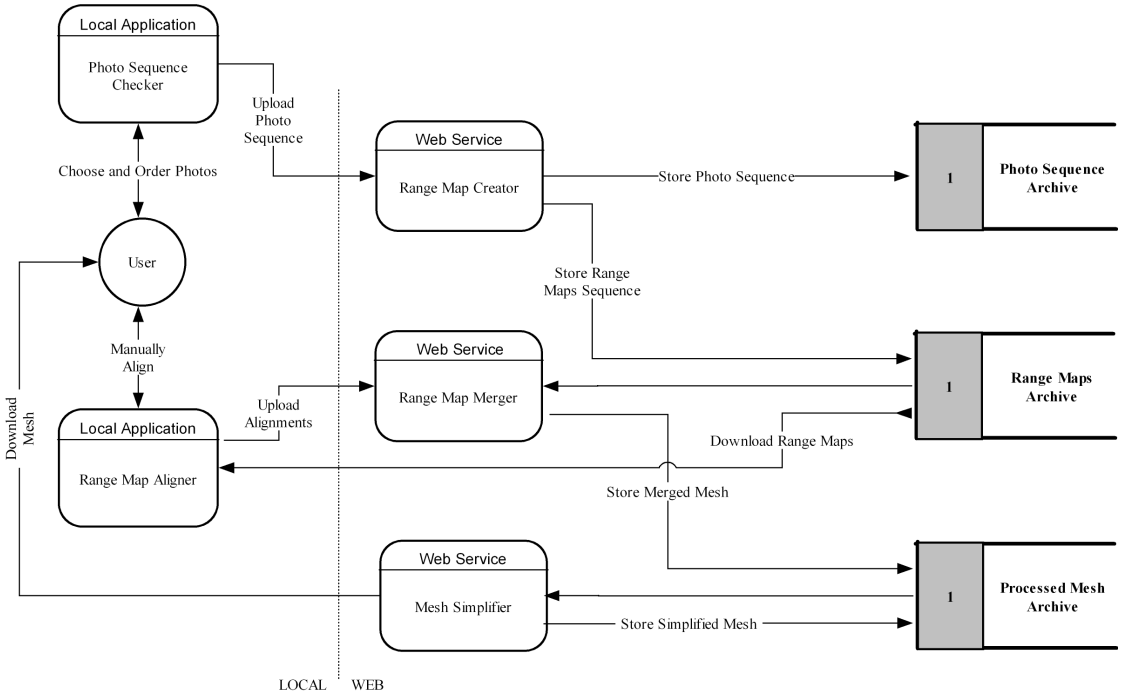


Figure 9.1: The main architecture of the combined 3D system: a local application prepares and checks the photos collected by a user and uploads them to a web service; the web service creates and stores range maps; these range maps are then aligned by the user and finally merged, simplified and exported to various formats

as follows. Section 9.2 gives an overview of the client-server system that allows users to upload images to a server which computes 3D information from these images. Section 9.3 describes the automatic reconstruction pipeline running on the server. Sections 9.4, 9.5, 9.6, 9.7 give more detailed information on several core aspects of this pipeline. Section 9.8 shows some results we obtained from uploaded images and in section 9.9 we come to a conclusion.

9.2 System Overview

Fig. 9.2 shows a schematic overview of the client-server setup of the 3D webservice. The client- (or user-) part is located at the top. The server side is at the bottom. On his PC, the user can run two programs, the *upload tool* and the *modelviewer tool*, indicated **A** and **B** respectively. These tools are explained in more detail in paragraphs 9.2.1, 9.2.2. In the upload tool, images can be imported that were taken with a digital camera. Once authenticated, the user can transfer these images (indicated **C**) over the Internet to the EPOCH server at ESAT. There a fully automatic parallel process is launched which computes dense 3D information from the uploaded images. The parallel processes are run on a cluster of Linux PC's (letter **D**). When the server has finished processing, the user is notified by email and the results can be downloaded from the ESAT server by FTP. These results consist of dense depth maps for every image and the camera calibration (letter **E**). The modelviewer tool allows the user to inspect the results. Every reconstructed depth map in the image set can be shown in 3D, unwanted areas can be masked out and the meshes can be saved in a variety of formats.

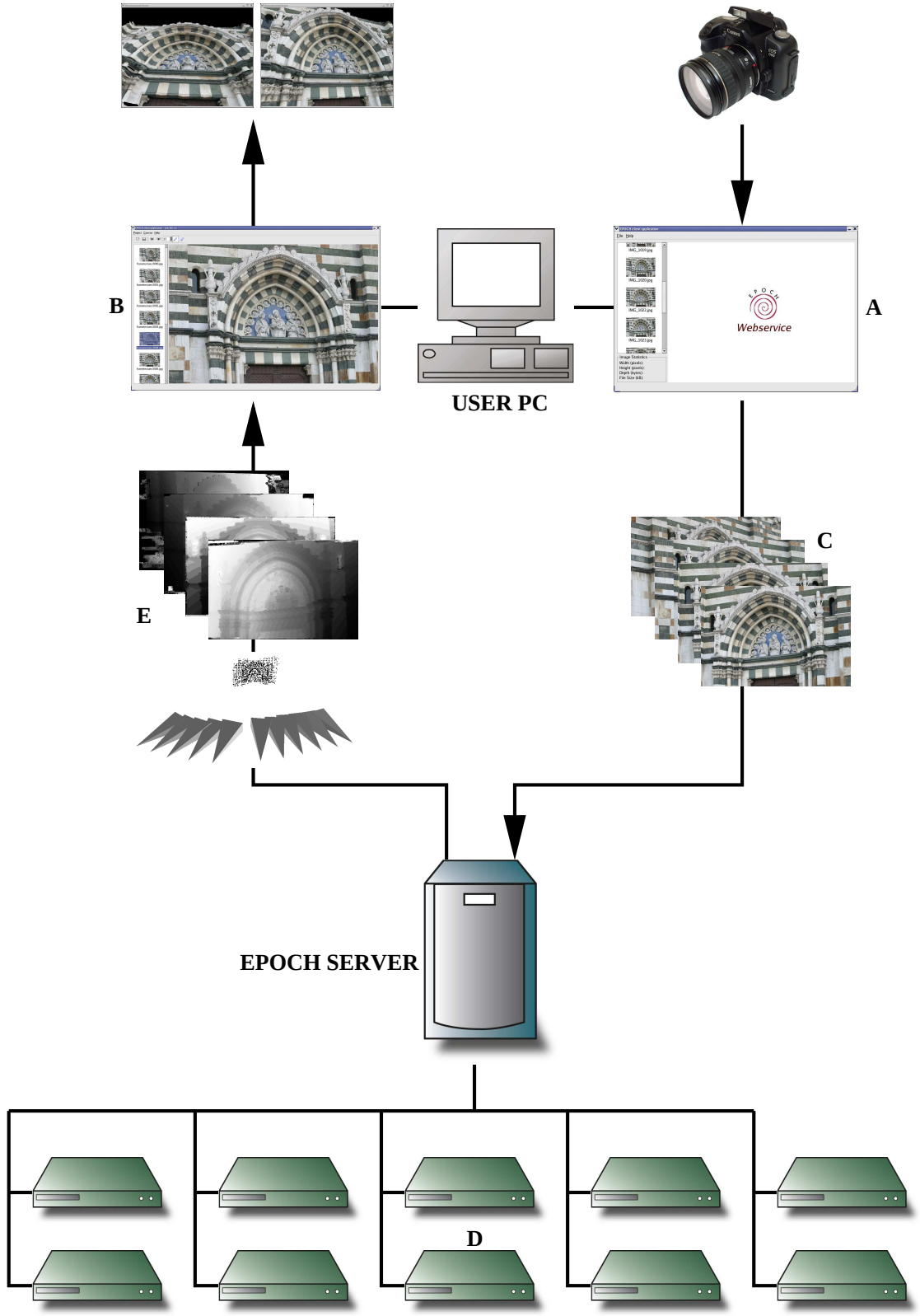


Figure 9.2: Schematic overview of the client-server setup. Images (C) are uploaded from the upload tool (A) on the user side to the server. There they are processed on a PC cluster (D). The results (E) can be downloaded via ftp and visualized on the user PC with the modelviewer tool (B).



Figure 9.3: Upload tool. Thumbnails of the images to be uploaded are shown as well as some statistics. When satisfied, the user uploads the images to the reconstruction server.

9.2.1 Upload Tool

The first tool a user of the 3D webservice encounters is the upload tool. This is the program that uploads images to the server via the Internet. Fig. 9.3 shows a screenshot of the GUI of this tool. First, the user selects the images which he wants to upload. Thumbnails of these images are shown on the left and a large version of the image is shown in the main panel when a thumbnail is clicked. Images can be removed from the list or extra images can be added. On the bottom left, statistics on the image such as the number of pixels, the size on disk, and the format are shown.

When the user is satisfied with the selected images, he can upload them to the reconstruction server. In order to do so, the user first needs to authenticate himself with his username and password. The upload is started and a progress dialog shows the speed of the upload. In order to limit the upload- and processing time, the user can decide to downscale the images by a certain percentage.

9.2.2 Modelviewer Tool

Upon completion of the 3D reconstruction, the user is notified that the results are ready. They are stored in a zip file on the ESAT ftp server in a directory with a random name whose parent is not listable. This makes sure the user's reconstructions are save from prying eyes. The zip file contains the original images, the calibration of the cameras, the dense depth maps and quality maps for every image. The results can be inspected by means of the modelviewer tool, a screenshot of which is shown at the top of Fig. 9.4. The layout of the modelviewer tool purposely resembles that of the upload tool.

A thumbnail is shown for every image in the set. If clicked, a large version of the image is shown in the main window. A triangulated 3D model can be created for the current image, using the depth map and camera of this image. Every pixel of the depth map can be put in 3D and a triangulation in the image creates a 3D mesh, using the texture of the current image.

Every depth map has its corresponding quality map, indicating the quality or certainty of the 3D coordinates of every pixel. The user can select a quality threshold. High thresholds yield models with fewer but more certain points. Lower thresholds yield more complete models including areas with a lower reconstruction quality. If required, certain areas of the image can be masked or grayed out by drawing with a black brush in the image. Areas that are masked will not be part of the 3D model. The resulting 3D model is displayed in an interactive 3D widget. Two viewpoints are shown in the bottom part of Fig. 9.4. The model can be exported in different 3D file formats like VRML2 or OpenSG's native file format.

9.3 Automatic Reconstruction Pipeline

The 3D webservice is meant to create 3D reconstructions from images from a wide variety of sources. Because no user interaction is possible once the images have been uploaded, an important prerequisite is the need for robustness and autonomy on the server part. In order to avoid frustration and multiple uploads at the client side, the server should maximize the chance of obtaining a good 3D result, even from sequences of images that are not perfectly suited for the purpose. This requirement has lead us to the development of a hierarchical, massively-parallelized and opportunistic 3D reconstruction scheme.

9.3.1 Pipeline Overview

A schematic flowchart of the reconstruction pipeline is shown in Fig. 9.5. In this figure rectangles represent procedures or actions. Parallelograms represent data structures. The input of the pipeline is found on the top-left and consists of the set of images the user has uploaded to the server using the upload client. At the bottom right the result can be seen consisting of dense 3D depth (or range) maps. The pipeline can be seen to consist of roughly four steps:

1. A step that computes a set of image pairs that can be used for matching, including the Subsampling and Global Image Comparison steps. In this step, the images are first subsampled (hence the hierarchical nature of the pipeline). Since images can be uploaded in non-sequential order, we have to figure out which images can be matched. This is the task of the Global Image Comparison algorithm which yields a set of image pairs that are candidates for pairwise matching.

2. A step that performs the Pairwise and Projective Triplet Matching and the Self Calibration. In this step, feature points are extracted on the subsampled images. All possible matching candidates of step 1 are now tried. Based on the resulting pairwise matches, all image triplets are selected that stand a chance for projective triplet reconstruction. This process is performed and the results are fed to the self-calibration routine which finds the intrinsic parameters of the camera.

3. A step that computes the Euclidean reconstruction and upscales the result to full resolution. In this step all image triplets and matches are combined into one 3D Euclidean reconstruction.

4. A step that is responsible for the dense matching, yielding dense depth maps for every image.

Sections 9.4, 9.5, 9.6 and 9.7 describe our approach for each of these steps respectively.

9.3.2 Opportunistic Pipeline

Classic Structure And Motion pipelines make use of the fact that the set of input images is taken in a sequential manner. This helps the reconstruction process tremendously because only consecutive pairs of images must be matched for 3D reconstruction. Unfortunately the 3D Webservice described in this paper can not rely on this assumption. Users can upload images in non-sequential

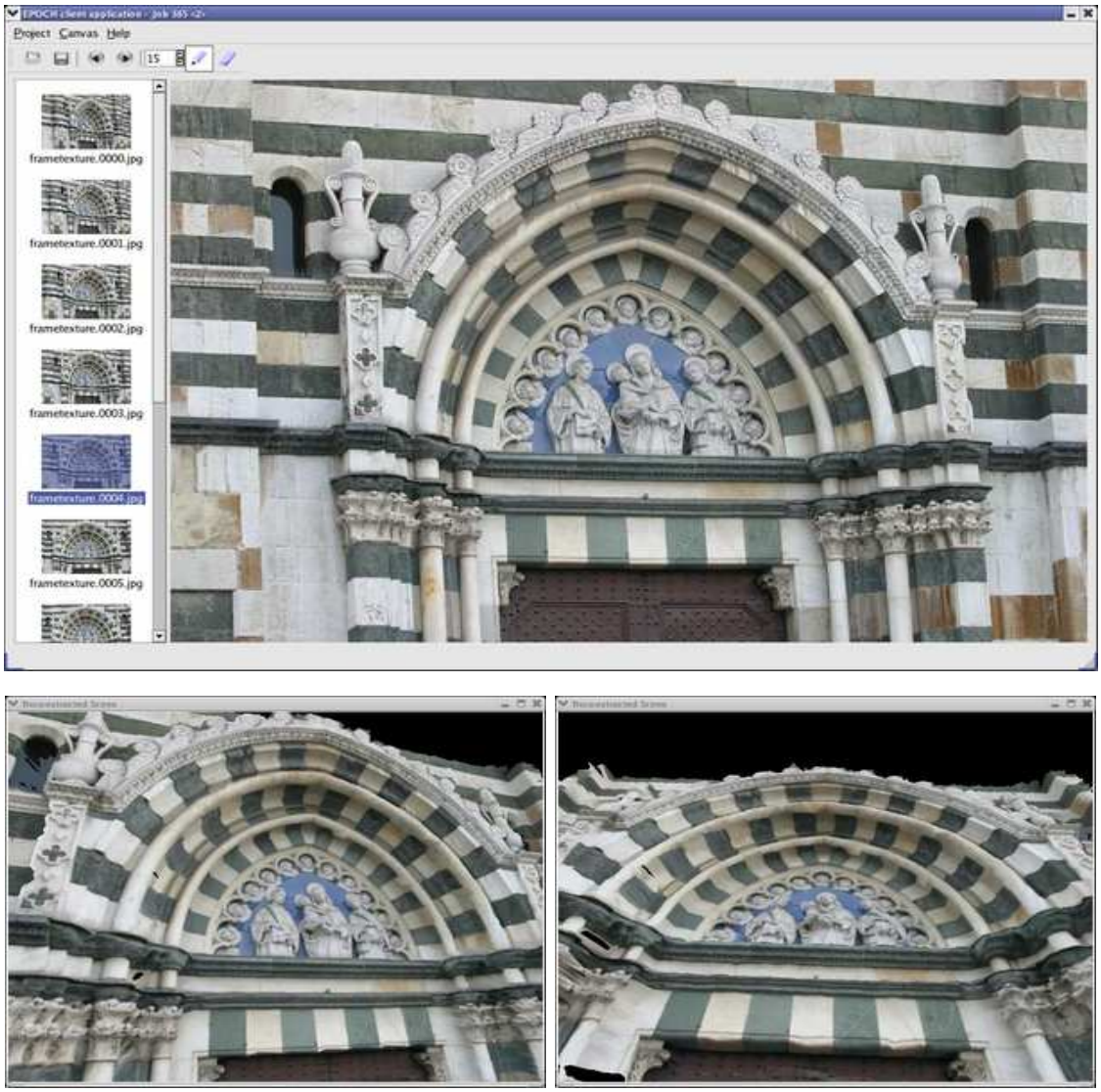


Figure 9.4: Modelviewer tool. The layout of the tool (top) deliberately resembles that of the upload tool, with thumbnails on the left and a large image in the main panel. The tool allows the user to create textured wireframe representations of the depth maps. The bottom two pictures show different views of such a 3D representation.

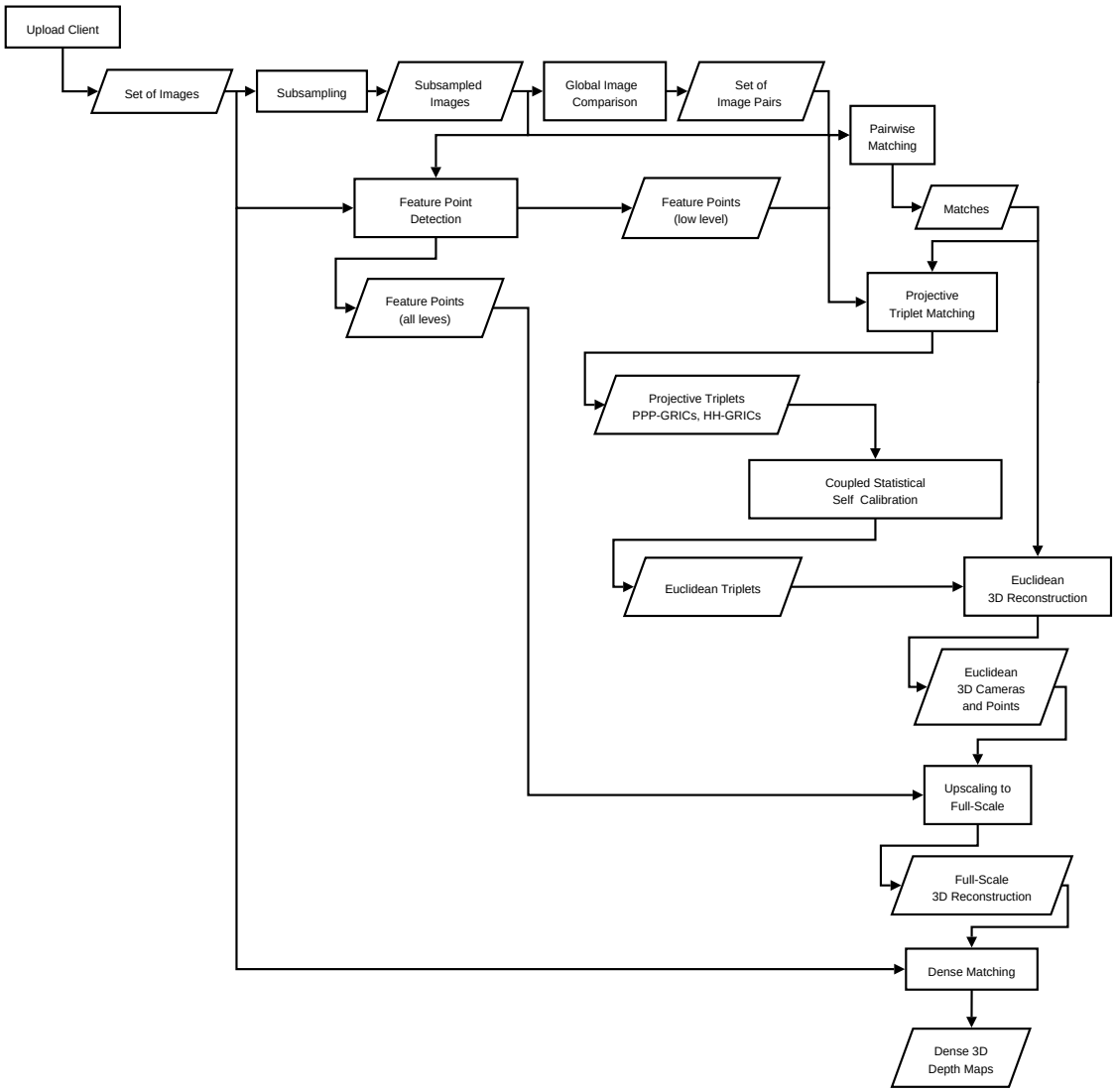


Figure 9.5: Global flowchart of the Reconstruction Pipeline

order or even use images that were taken in a random fashion. This has an impact on the matching step, the reconstruction step and the dense matching step.

9.3.3 Hierarchical Pipeline

In general the quality and accuracy of the resulting depth maps is proportional to the size of the input images. However, computing feature-points and matches on large-scale images is very time-consuming and not so stable a process. That is why all incoming images are first subsampled a number of times until they reach a typical size in the order of 1000x1000. As can be seen in Fig. 9.5 most of the Structure And Motion processing is performed on these subsampled images. It is only in the upscaling step (at the bottom right) that the result is upgraded from the low-resolution to the high input resolution. This hierarchical approach combines a number of advantages. First it is more stable and has a higher chance of success than direct processing of the high resolution images. Indeed, it is easier to find matching features between small images because the search range is smaller and therefore fewer false matches are extracted. On lower-level images feature-points are typically more dominant than on higher level where features also appear in noisy and cluttered areas that are very hard to match. Furthermore processing times decrease when dealing with smaller images. The upscaling step receives a very good initialization from the lower levels. This means that only a small search and optimization needs to be performed.

9.3.4 Parallel Pipeline

Several operations in the reconstruction pipeline have to be performed many times and independently of each other. Image comparison, feature extraction, pairwise or triplet matching, dense matching, etc are all examples of such operations. The pipeline is implemented as a python script which is automatically triggered by the SQL database when a new job arrives. The script has to go through several steps and every step can only be started when the previous one has finished. Inside one step, however, the processing is parallelized. The server on which the script runs has access to a queuing system of our own making, as shown in Fig. 9.6. When an operation must be performed, the server sends the job to the queuing system which returns a job-id. The queuing system has access to a PC cluster and continuously checks the memory and CPU load of each of the nodes. When a node is free, the job on top of the queue is sent to that node. When the job has finished, the server is automatically notified by the creation of a file with the job-id.

9.4 Global Image Comparison

The images that are uploaded to the reconstruction server can be taken in any random order and don't necessarily have to be in sequential order. This means of course that it no longer suffices to process consecutive images. Unfortunately, matching every possible pair of images is a very time-consuming step. That's why we want to find out which images can be matched first.

In [56, 57] techniques have been proposed that can deal with non-sequential, wide-baseline image sets. Admittedly, these wide-baseline matching techniques are very powerful but for our purpose we deem them not very well suited because the typical scenarios where our technique would fail and wide-baseline techniques would succeed are image-sets with such large disparities or changes between the images that it becomes very difficult for the dense matching step to yield good results. Since the goal of our pipeline is the creation of dense depth maps, these image sets would be useless anyway because the dense matching process would yield reconstructions with large holes in those areas where not enough information is visible in different images.

For our pipeline we have implemented an alternative for these techniques which is simple and fast and still delivers good results. All possible pairs of images are fed to the *Global Image Comparison* algorithm. This algorithm is based on comparing the two images under scrutiny with a comparison function like Normalized Cross Correlation (NCC). The goal of the this step is to

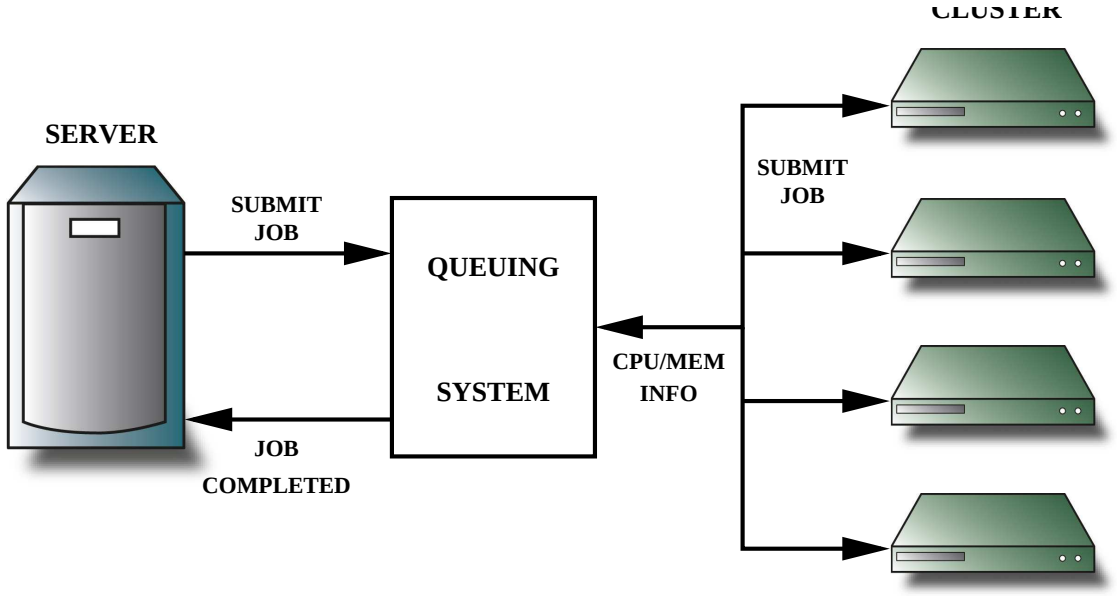


Figure 9.6: Queuing system. The server sends a job to the system. The queue has access to a PC cluster and sends the job to any of the nodes that becomes free.

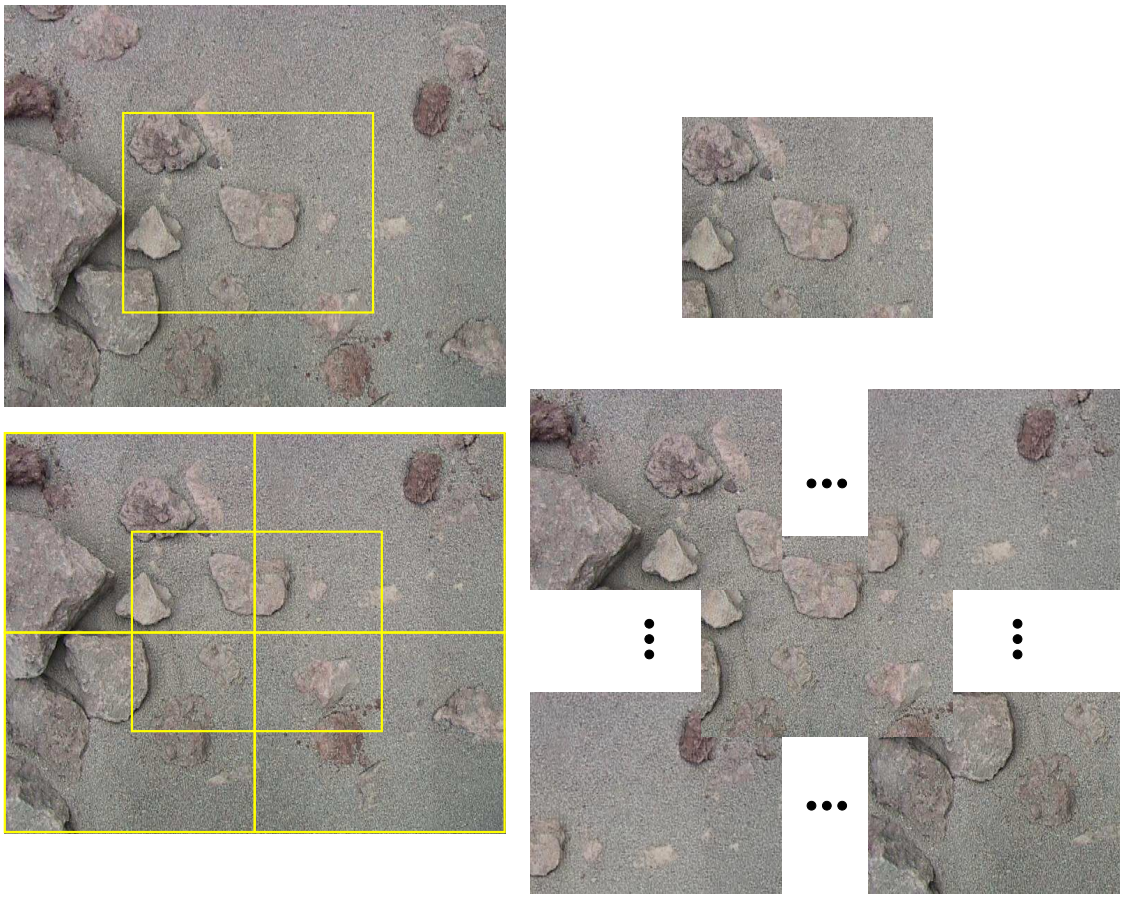


Figure 9.7: Comparing two images. The global overlap between the two images is computed as well as the resulting correlation value

quickly come up with pairs of images for which matching and computation of epipolar geometry is likely to succeed.

Fig. 9.7 shows a schematic overview of the algorithm. First the center part of the first image is selected. A window with the same size slides over the second image and the NCC value of both windows is evaluated. From the window position for the largest NCC value, it can be computed how many pixels the image content is shifted from the first to the second image, and thus the overlap value can be calculated. The value of the correlation is a good measure of how well these two images correspond. For efficiency reasons the images are subsampled a couple of times. A beneficial side-effect of the subsampling is that the chance of local maxima of the NCC values is greatly reduced. All possible image pairs are processed in parallel (as explained in section 9.3.4) and only those pairs with a NCC value that is sufficiently high are kept for real pairwise matching. The shift between the windows that yielded this highest NCC value is stored and used later to facilitate the search for matches between the images.

9.5 Self-calibration

9.5.1 Classical SaM techniques

Classic Structure And Motion systems either use a-priori information on internal camera calibration parameters or go through the process of first reconstructing the scene in a projective framework and then upgrading this reconstruction to metric using self-calibration and bundle-adjustment techniques [46, 25].

First feature points are extracted and consecutive images are matched. Epipolar geometry is retrieved and a projective frame is initialized. All consecutive images are matched to their predecessor and every new projective camera is computed in the projective frame of the first pair. As explained by Cornelis et al in [11] all projective reconstructions obtained in a sequential way suffer from drift. In a worst-case scenario, the drift of the recovered solution w.r.t. the true solution can become so substantial that self-calibration is no longer possible because the recovered projective reconstruction is not consistent anymore.

The simple sequential reconstruction approach poses other problems as well. Very regularly images are taken of a scene or object which is planar or close to it. Obtaining a unique projective reconstruction of such a planar scene is mathematically impossible because multiple projective solutions exist. The appearance of different projective frames in the solution causes self-calibration algorithms to fail as well. Also, contrary to the classical techniques, our input images are not always taken in a purely sequential way. It is very disadvantageous if these non-sequential images are only processed sequentially. Images should not just be matched to their direct predecessor to reconstruct their cameras.

9.5.2 Triplet Matching

In order to alleviate the problems mentioned in the previous paragraph, a new reconstruction and self-calibration algorithm has been developed which has proved to be stable and rewarding on image sequences that could not be processed with the simple sequential approach. As explained in paragraph 9.4 all images in the uploaded set can efficiently be compared to each other. This comparison yields a list of image pairs that are candidates for feature matching. The extracted feature points are matched between each of these image pairs and epipolar geometry is computed. If the matching yields a sufficiently good result, the result is stored. When this matching step has finished possible image-triplets are selected based on the matching results. Only triplets with enough matches between both images 1 and 2 and images 2 and 3 are withheld. For these triplets a projective reconstruction is computed, in the form of 3 projective cameras and a set of 3D points. All these calculations (image-pair matching and image-triplet reconstruction) can be performed independently and thus all processing is done in parallel using the queuing system described in paragraph 9.3.4.

Our self-calibration will later make use of the reconstructed triplets. Not only do we want to reconstruct as many triplets as possible, we also need to investigate if they are useful for self-calibration purposes. In order to do so we make use of the Geometric Robust Information Criterion (GRIC) proposed in [61], a mathematical description of Occam's razor which states that "one should not increase, beyond what is necessary, the number of entities required to explain anything". If different models exist to explain certain data, the GRIC paradigm selects the model with the lowest penalty. The penalty of a model is obtained by summing two contributions. The first one is related to the goodness of the fit and the second one is related to the parsimony of the model. GRIC Takes into account the number n of inliers, the residuals e_i, the standard deviation of the measurement error σ, the dimension of the data r, the number k of model parameters and the dimension d of the structure:

$$GRIC = \sum \rho(e_i^2) + (nd\ln(r) + k\ln(rn)) \tag{9.1}$$

For a triplet, we try to explain matches between images in two different ways, either as a structure with 3 projection matrices or as two times two images with one common image in the middle, related by planar homographies. The GRIC values for these two models are called **PPP-GRIC** and **HH-GRIC** respectively. If a scene is near to planar or if only a planar part is in common between the 3 views of a triplet, the corresponding HH-GRIC will have a lower penalty than the PPP-GRIC, indicating that a model consisting of two homographies better explains the data than the model using three projection matrices. Such a triplet is therefore not suited for self-calibration and can only be correctly reconstructed if the internal camera parameters have already been recovered, e.g. from other parts of the data.

9.5.3 Coupled Self-calibration

In[48] the concept of coupled self-calibration was introduced for a limited set of projectively reconstructed sequences which share the same intrinsic parameters. The approach is based on the projection equation for the absolute quadric [62]:

$$\mathbf{K}\mathbf{K}^\top \sim \mathbf{P}\Omega^*\mathbf{P}^\top \tag{9.2}$$

where Ω^* represents the absolute quadric. In metric space $\Omega^* = \mathrm{diag}(1,1,1,0)$, in projective space Ω^* is a 4×4 symmetric rank 3 matrix representing an imaginary disc-quadric. When choosing $\mathbf{P} = [\mathbf{I}|\mathbf{0}]$ for one of the projection matrices it can be seen from Equation (9.2) that Ω^* can be written as:

$$\Omega^* = \begin{bmatrix} \mathbf{K}\mathbf{K}^\top & \mathbf{a} \\ \mathbf{a}^\top & b \end{bmatrix} \tag{9.3}$$

Now any set of constraints on the internal parameters are translated to constraints on the absolute quadric and can be written as:

$$\begin{bmatrix} \mathbf{C} & \mathbf{D} \end{bmatrix} \begin{bmatrix} \mathbf{k} \\ \begin{bmatrix} \mathbf{a} \\ b \end{bmatrix} \end{bmatrix} \tag{9.4}$$

where $\mathbf{k}$ is a vector containing 6 coefficients representing the matrix $\mathbf{K}\mathbf{K}^\top$, $\mathbf{a}$ is a 3-vector and b a scalar and $\mathbf{C}$ and $\mathbf{D}$ are matrices containing the coefficients of the equations. Their size is nx6 and nx4 respectively with n the number of constraints. Note that this can be done independently for every 3D subsequence. If the sequence is recorded with constant intrinsics, the vector $\mathbf{k}$ will

be common to all subsequences and one obtains the following coupled self-calibration equations:

$$\begin{bmatrix} \mathbf{C}_1 & \mathbf{D}_1 & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{C}_2 & \mathbf{0} & \mathbf{D}_2 & \cdots & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \mathbf{C}_n & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{D}_n \end{bmatrix} \begin{bmatrix} \mathbf{k} \\ \begin{bmatrix} \mathbf{a}_1 \\ b_1 \end{bmatrix} \\ \begin{bmatrix} \mathbf{a}_2 \\ b_2 \end{bmatrix} \\ \vdots \\ \begin{bmatrix} \mathbf{a}_n \\ b_n \end{bmatrix} \end{bmatrix} \tag{9.5}$$

9.5.4 Statistical Coupled Self-calibration

Coupled self-calibration is a very powerful extension of self-calibration of a single projective sequence as described in [47]. It allows to calibrate reconstructions in different projective frames in one step. This is of high interest if only short projective sequences are available. Such sequences do not entail enough information to recover all parameters of the calibration model in a stable way if used by themselves. Coupled in one computation, they will allow for the internal parameters of the camera to be recovered. The most extreme example of a short projective reconstruction is a projective triplet like the ones we recovered as described in 9.5.2. The calibration algorithm only yields good results if all reconstructions at the input are consistent in their projective frame. With only three views present in each triplet there is a chance that a combination of several triplets contains at least one weakly reconstructed triplet. That is the reason why we opted for a statistical approach.

All reconstructed triplets of 9.5.2 are possible candidates for coupled self-calibration. First the PPP- and HH-GRIC of each triplet is compared. Only triplets for which the PPP-GRIC value is lower than the HH-GRIC are retained because we are certain that there is enough 3D information in these triplets for self-calibration. From this set of triplets we randomly pick a certain number (7 in the current implementation). The matrix on the left of equation 9.5 is built and we solve for the internal parameters. If the smallest singular value of the SVD which solves this system is small enough the solution is accepted and stored. This step is repeated many ($\sim$1000) times or until all triplet-combinations have been tried, each time picking another random set of triplets. Each solution for the absolute quadric corresponds to a matrix with intrinsic parameters K. We retrieve the top left element of each K, representing the focal length in pixels in x-directions and create a histogram of these values. A typical histogram is shown in Fig. 9.8. This figure shows the typical Gaussian-like distribution of the self-calibration results we obtained from a multitude of experiments. The maximum of the Gaussian is selected and the K-matrix corresponding to this value is chosen. Of course the self-calibration procedure is not meant to yield perfect calibration results but is rather used as a good initialization for bundle-adjustment steps that are executed later.

9.6 Reconstruction

Paragraphs 9.5.2 and 9.5.4 explained how we retrieved projective reconstructions of many image triplets and how we used these triplets to compute the intrinsic parameters of the camera that took the images. A straightforward next step is to upgrade every triplet from its projective to a metric frame, using the intrinsic parameters K. However, this procedure brings us only part-way towards our true goal: to retrieve the camera calibration and 3D points for all images in the same metric frame because each triplet is still reconstructed in its own metric frame.

In order to bring all information together, we start by searching for the triplet which is best suited for initializing our final 3D reconstruction. This triplet should have a large number of 3D points and have a large $\frac{HH-GRIC}{PPP-GRIC}$ value. When this triplet is found, our reconstruction is initialized. Other images can now be added to the reconstruction one at the time. This is not

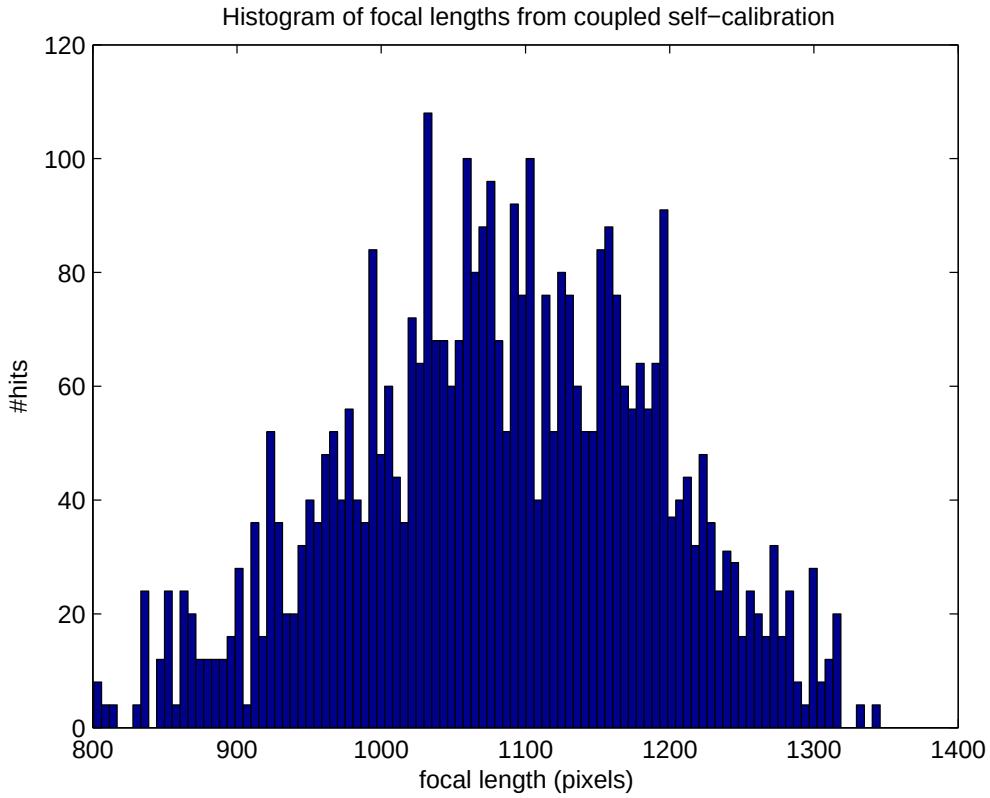


Figure 9.8: Histogram of focal lengths for all solutions of coupled self-calibration from a set of triplets.

done in a purely sequential way where only matches to the previously added image are employed but rather using all matches with all already added images. Using these matches a robust pose-estimation algorithm is executed which computes the camera of the newly added image in the metric frame of the reconstruction. The fact that the pose-estimation can be done in metric space is an advantage because now also images which only observe a planar part of the scene can be reconstructed.

9.6.1 Upscaling the Result

All previous computations were performed on the subsampled images, both for efficiency and stability reasons. It is now time to upgrade the result to the full-scale images that were uploaded, a process we have called *upscaling* the result. An iterative algorithm was implemented to upscale the resulting reconstruction. It is executed as many times as needed to go from the small low-level images to the highest level of the original size images. The simplest way to upscale the result to the original image size is of course to upscale the resulting calibration. Every upscaling with a factor of two is simply done by scaling the intrinsic parameters of the lower level (K_0) to the higher level (K_1) for every image i. The extrinsic parameters (R and t) of every camera i and the position of every 3D point j (M) can stay the same. This yields the new cameras P_i.

$$
\begin{aligned}
K_{0i} &= \begin{bmatrix} f_{x0i} & s_{0i} & u_{x0i} \\ 0 & f_{y0i} & u_{y0i} \\ 0 & 0 & 1 \end{bmatrix} \\
K_{1i} &= \begin{bmatrix} f_{x1i} & s_{1i} & u_{x1i} \\ 0 & f_{y1i} & u_{y1i} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 2f_{xoi} & s_{0i} & 2u_{xoi} \\ 0 & 2f_{yoi} & 2u_{yoi} \\ 0 & 0 & 1 \end{bmatrix} \\
R_{1i} &= R_{0i} \\
t_{1i} &= t_{0i} \\
M_{1j} &= M_{0j} \\
P_{1i} &= K_{1i} \left[R_{1i}^T | -R_{1i}^T t_{1i} \right]
\end{aligned}
\tag{9.6}
$$

This solution can of course only serve as an initialization for an optimization process because all errors of the low-level reconstruction are upscaled as well. As can be seen in Fig. 9.5 we computed feature points in the images on all levels, i.e. on both subsampled and original images (and possibly intermediate levels too, depending on the size of the originals). The number of extracted features is increased for every consecutive level. These features will help us to optimize the result of 9.6. The algorithm that implemented to upscale the resulting reconstruction is iterative and works as follows.

1. Compute a first initialization for the cameras P_{1i} and points M_{1j} of the new level based on the level below as explained in equation (9.6).
2. The 3D-2D correspondences that indicate in which images every 3D point is visible remain identical.
3. For every 3D point M_{1j} perform the following operation:
   - Project M_{1j} in the images where it is visible using the new cameras P_{1i}.
   - Search the extracted features of the image at level 1 for the feature that is closest to the projected point.
   - Change the 3D-2D correspondence to this closest feature.
4. Perform a Robust Euclidean Bundle Adjustment on all 3D-2D correspondences. This step computes updated values for the 3D points M_{1j} and cameras P_{1i}. This step is explained in paragraph 9.6.2.

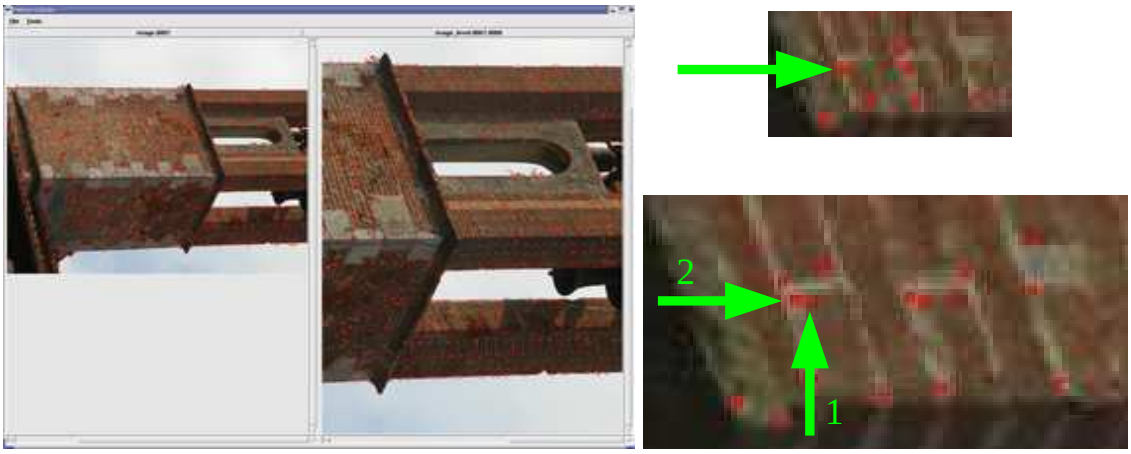


Figure 9.9: Subsampled and original image with extracted features are shown on the left. On the right one part of the images has been cut out. The arrow at the top points to a feature with a 3D point in the low-level image. At the bottom arrow 1 points to the feature in the high-resolution image which is closest to the reprojection of the 3D point using the initialization (eq 9.6). Arrow 2 points to the feature that is selected after 1 iteration of the 3D points and camera parameters.

5. Iterate steps 3 and 4. Because of the updated correspondences, the updated values of the points and cameras will yield new projections. Some of these will now be closer to another feature in the feature set. Fig. 9.9 shows an example. This implies a new change to the 3D-2D correspondences and hence again an updated solution. The iteration is stopped when no more updates to the 3D-2D correspondences are made. This is typically reached after 5-6 iterations.

The upscale step described above succeeds in computing the Euclidean projection matrices and 3D points for the full-scale images. Non-linear radial distortion parameters can be estimated as well. The technique has one drawback however. Only originally matched points will be upscaled and no extra points are added to the reconstruction. The reconstructed points are those that are inliers to the epipolar geometry of image pairs. Since at the time of computation of this geometry no information on the radial distortion is available, typically few points near the borders of the image are matched if the images suffer from this kind of distortion. This is because radial distortion has the largest effect on points far away from the center of the image. After the upscaling step, however, we have the parameters describing the distortion to our disposal. This means that now we can search for more matching points, taking into account this distortion which yields many extra points that are also located near the border of the images. Fig. 9.10 shows an example. The top two images show an image pair matched at the smallest pyramid-level without taking radial distortion into account. The bottom image pair shows the matches on the full size images accounting for radial distortion. Especially the foreground region of the images shows much more matched features.

9.6.2 Robust Euclidean Bundle Adjustment

The Robust Euclidean Bundle Adjustment of step 4 is an extension of the traditional bundle adjustment techniques [63]. In these least-squares estimators the global reprojection error is minimized. It has been shown in numerous studies that least-squares estimators are vulnerable to the violation of the assumption that all data at the input consists of inliers. Sometimes even when the data contains only one bad sample, least-squares estimates may be completely perturbed. In the upscaling-step outliers are inevitable. That is why a robust version of the classical bundle adjustment technique is called for. A popular technique to deal with possible outliers is the so-called *M-estimators*. Their use and applicability is well described in [67]. In short, M-estimators

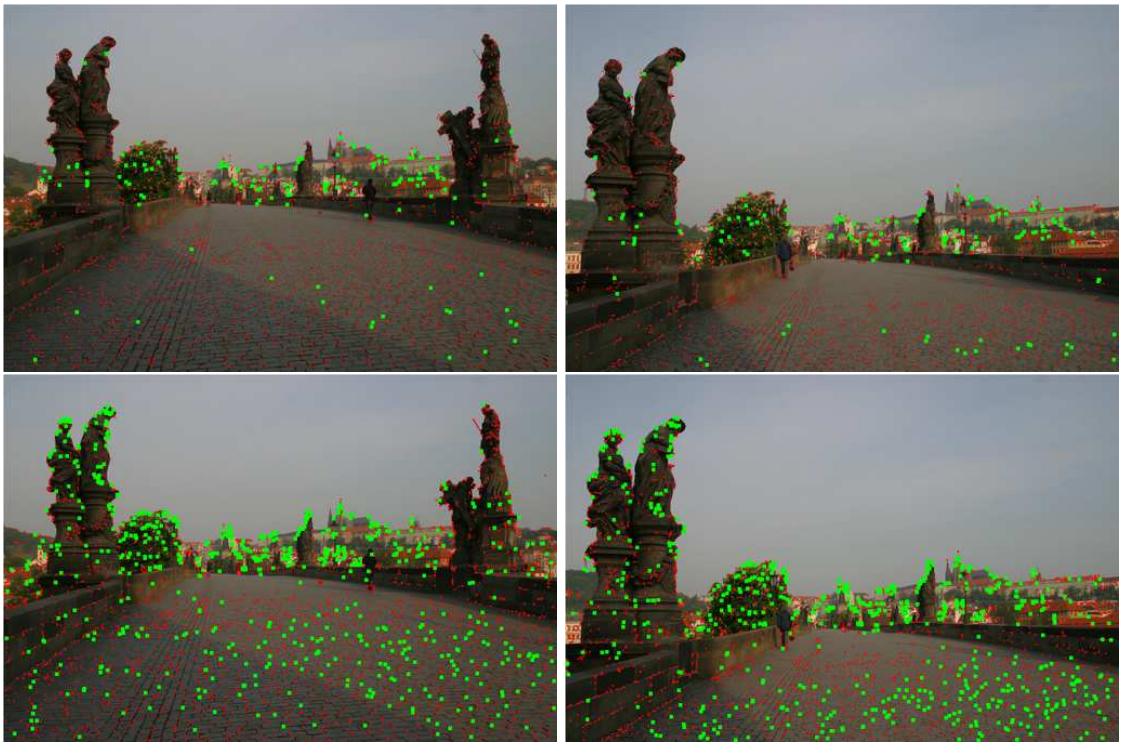


Figure 9.10: The top two images show an image pair matched at the smallest pyramid-level without taking radial distortion into account. The bottom image pair shows the matches on the full size images accounting for radial distortion. Especially the foreground region of the images shows much more matched features.

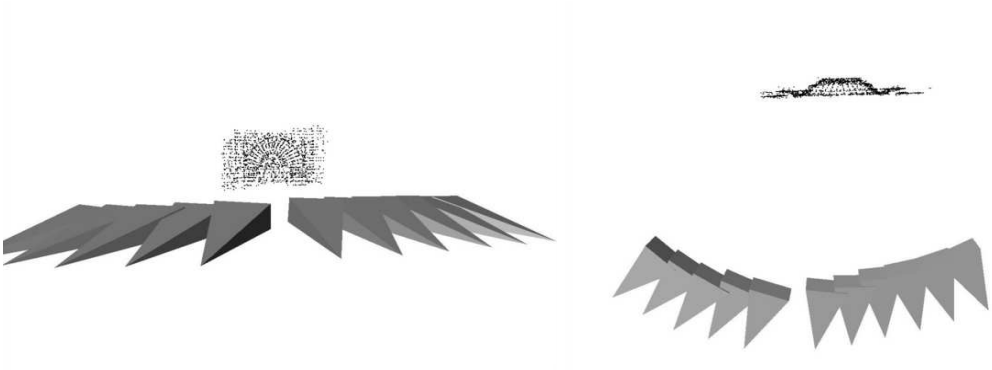


Figure 9.11: Reconstruction of points and cameras for the "Prato" example.

try to reduce the effect of outliers by replacing the squared residuals by another function of the residuals. In our implementation we chose Cauchy's function, also known as the Lorentzian function given in equation 9.7. In stead of minimizing the sum of the residuals $\sum r_i^2$ we now minimize $\sum \rho(r_i^2)$. The parameter c of equation 9.7 is set to 1 which means that the M-estimator starts limiting the influence of outliers when they are more than 1 pixel away from their estimate.

$$\rho(x) = \frac{c^2}{2} \log\left(1 + \left(\frac{x}{c}\right)^2\right) \tag{9.7}$$

Although the resulting 3D points and calibration from the upscaling algorithm are accurate, the amount of 3D points can be rather low because the algorithm only removes outliers and never adds 3D points or 3D-2D correspondences. As will be explained in paragraph 9.7 some dense-matching approaches can use reconstructed feature points as seeds for their dense reconstruction. It is therefore important to increase the amount of 3D points. The recovered calibration is used to search for new correspondences in very localized areas (along the epipolar line for the first two images and in a neighborhood around the projection of the 3D point in consecutive images). A typical result, consisting of calibrated cameras and a set of reconstructed 3D points is shown in Fig. 9.11 and Fig. 9.12, showing a sequential and non-sequential image set respectively. The calibration of the non-sequential image set of Fig. 9.12 makes use of non-sequential links between images. After the initialization with the best triplet, new cameras are added using all matches with all already added images. The links that were used in this specific example are shown as lines connecting the camera centers.

9.7 Dense Matching

The result of the Euclidean reconstruction explained in the previous sections is a set of points and cameras, as shown in Fig. 9.11. This sparse set of points can not be the end-point of the reconstruction process. The cultural heritage partners expect dense, textured 3D models. In order to deliver these, a dense matching process between the input images is necessary.

9.7.1 Linked Pairwise Stereo

In previous publications [47, 46] a technique for dense matching using linked pairwise stereo has already been thoroughly explained. This strategy entails matching all consecutive pairs of an image sequence with a stereo algorithm [38]. The resulting disparity maps between the image pairs are then combined into depth maps for every image using a statistical algorithm that checks for consistency of matched pixels over multiple pairs. This approach has proven its merit on sequential image sets, like the Prato example of Fig. 9.11. The quality maps (see section 9.2.2) indicating the confidence we have in the 3D reconstruction of every pixel are easily computed from

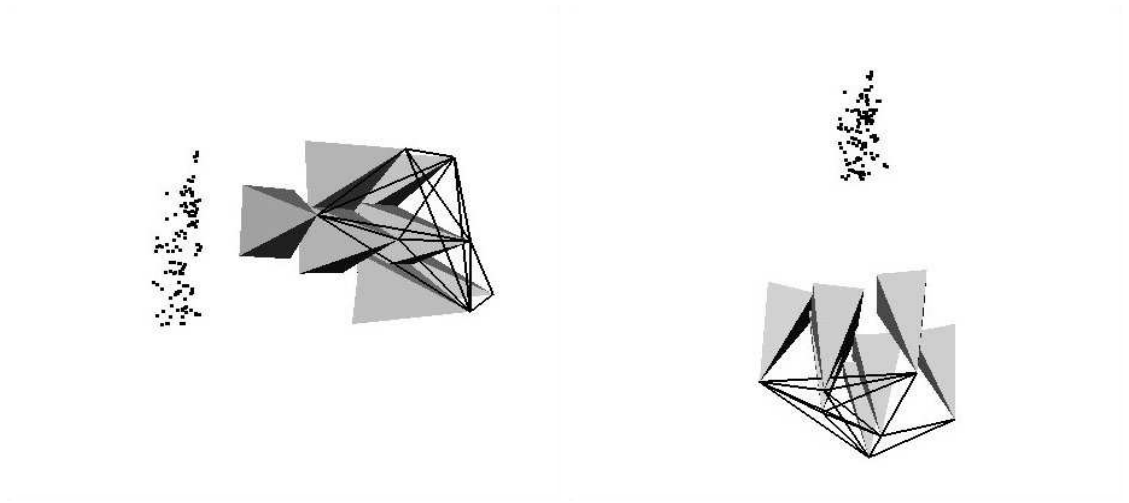


Figure 9.12: Reconstruction of points and cameras for the "Indian statue" example.

the linking approach. The more consecutive views in which a pixel is matched consistently, the more confident we are of its 3D position, hence the higher the quality measure for this pixel.

As already mentioned, however, we can not assume that the uploaded image sets have indeed been recorded and uploaded sequentially. Using the linked pairwise stereo method can easily lead to failure with such non-sequential image sets. If we just process the images pairwise as before the chances of encountering an image-pair with extreme disparities or even hardly any overlap are very real. This would immediately break the link between the pairs, resulting in much worse depth maps. A possible strategy would be to search for an optimal sequence connecting the images in the image-set in the best order. This is not an easy task. We have to decide which criterion will be used to assess whether two images should be next to each other in the sequence or not. For this we can use the number of pairwise matches (a result of the algorithm described in section 9.5.2). Unfortunately, while these matches give a good indication of the matching quality between every pair, finding the optimal path in the image set boils down to solving a Traveling Salesman problem, which is n-p hard and scales very badly. That is why we want to change the linked pairwise stereo method for a true multi-view stereo approach.

9.7.2 Multi-View Stereo

Multi-view stereo algorithms seek to reconstruct the 3D shape of a scene from a collection of images taken from different viewpoints. The most difficult step is to find the correspondence between points in different viewpoints.

One can make a distinction between two techniques. The first are so-called volumetric algorithms. This approach starts with an approximation of the volume of the 3D scene. This can be computed from reconstructed feature points or, more common, with visual hull techniques. The initialization of the volume evolves towards the true shape by enforcing photo-consistency between points that are seen in different images [65].

Other techniques do not make use of a volumetric description of the scene but try to find multi-view correspondences from the images directly [55], possibly initialized from reconstructed feature points.

Nico's approach is one of the latter but of course his idea is all about speed on the GPU. Should we describe his system here? Maybe we don't want to steal his thunder and wait for a publication of his system? Anyway, we need results before we can publish this.



Figure 9.13: Reconstruction of the "Prato" example. The top two views show textured models, the bottom view shows a non-textured version.

9.8 Results

At the time of submission of this paper, the 3D web-based reconstruction service was in beta-testing. Several image sets have been uploaded to the service by various users in the cultural heritage field. In this section we show some results.

Fig. 9.13 shows some resulting views of the façade of a church in Prato, Italy. This example was actually used to prove the feasibility of the web-based approach at EPOCH's first term review. The image set was recorded and uploaded sequentially as can be seen in Fig. 9.11. This was not the case for the image set of an Indian statue, for which the structure and motion results are depicted in Fig. 9.12. The dense depth map results are shown in Fig. 9.14. Both textured and untextured 3D models, created from the depth maps are shown.

Fig. 9.15 shows resulting depth maps of a stratigraphy example. This is of particular interest to archaeologists in the field. Many archaeological sites are uncovered with stratigraphic techniques in which one layer after another is dug away. Instead of using the classical technique of describing each layer by words and measuring places of artefacts with a ruler, our webservice technique offers a much more efficient alternative. Each layer is recorded with a photo camera and these pictures are uploaded to the service. The results describe the layer in 3D, including textures. The second screenshot in Fig. 9.15 shows a top view. The image from which the depth map was computed that is depicted here, was taken from the side. It is clear that occluded areas behind rocks and other obstacles have not been reconstructed and show up as holes. These should be filled by merging depth maps from this image set or from other image sets of the same scene. This is the goal of the steps that follow the reconstruction pipeline described in this paper in Fig. 9.1.



Figure 9.14: Reconstruction of the "Indian statue" example. The top two views show textured models, the bottom view shows a non-textured version.

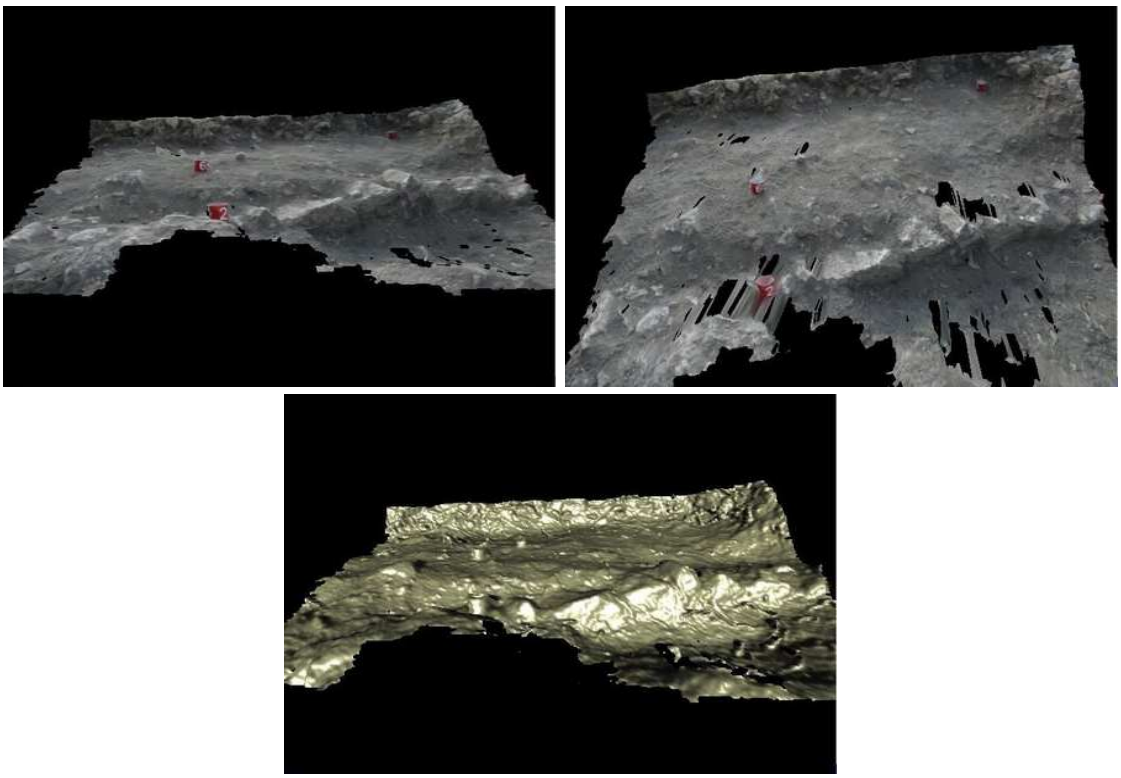


Figure 9.15: Reconstruction of a stratigraphic layer in Sagalassos. The top two views show textured models, the bottom view shows a non-textured version.

9.9 Conclusion

In this paper we described the first part of a complete 3D reconstruction system for the cultural heritage field. The system under scrutiny is in charge of computing the camera calibration and dense depth maps from images of an object or a scene. These images are uploaded to a processing server which automatically computes the results, making use of a cluster of PCs, and makes these results available on an ftp server.

Several specifics of the upload-server paradigm have to be taken into account, such as the fact that no user interaction is possible and that input images might not be taken or uploaded in a sequential order. These constraints drove the design of the pipeline which is automatic, opportunistic, hierarchical and tries to execute as many processes in parallel as possible.

Future Work:

- Further integration with Epoch reconstruction system
  - More automatic segmentation good/bad
  - Better export of models
- Feedback to user when reconstruction is not possible
- Use of better features (SIFT?) to be more stable to rotations and stuff
- More intelligence on client side. How feasible is this? We now have a parallel system on the server side. This is not possible at the user side. We can only go back to simple sequential processing. Maybe some statistics? How many well located features are there and how do they match sequentially?

Bibliography

[1] John Aldrich. R.a. fisher and the making of maximum likelihood 1912-1922. *Statistical Science*, 12(3):162–176, 1997.

[2] Marc Alexa. Linear combination of transformations. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 380–387, New York, NY, USA, 2002. ACM Press.

[3] L. Alvarez, R. Deriche, J. Sanchez, and J. Weickert. Dense disparity map estimation respecting image discontinuities: a pde and scalespace based approach, 2000.

[4] R. Ariew. *Ockham's Razor: A Historical and Philosophical Analysis of Ockham's Principle of Parsimony*. PhD thesis, University of Illinois, 1976.

[5] Cleve Ashcraft and Roger Grimes. SPOOLES: An object-oriented sparse matrix library. In *Proceedings of the Ninth SIAM Conference on Parallel Processing for Scientific Computing*, March 22–27 1999. `http://www.netlib.org/linalg/spooles/`.

[6] Herbert Bay, Vittorio Ferrari, and Luc J. Van Gool. Wide-baseline stereo matching with line segments. In *CVPR (1)*, pages 329–336, 2005.

[7] R E Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[8] J.Y. Bouguet. Camera calibration toolbox for matlab. `http://www.vision.caltech.edu/bouguetj/calib_doc/`.

[9] J. F. Canny. A computational approach to edge detection. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 8(6):679–698, 1986.

[10] K. Cornelis, F. Verbiest, and L. Van Gool. Drift detection and removal for sequential structure from motion algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(10):1249–1259, October 2004.

[11] K. Cornelis, F. Verbiest, and L. Van Gool. Drift detection and removal for sequential structure from motion algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(10):1249–1259, October 2004.

[12] Nico Cornelis and Luc J. Van Gool. Real-time connectivity constrained depth map computation using programmable graphics hardware. In *CVPR (1)*, pages 1099–1104, 2005.

[13] Alan Edelman and H. Murakami. Polynomial roots from companion matrix eigenvalues. *Math. Comput.*, 64(210):763–776, 1995.

[14] O. Faugeras and B. Mourrain. On the geometry and algebra of the point and line correspondences between n images. In *ICCV '95: Proceedings of the Fifth International Conference on Computer Vision*, page 951, Washington, DC, USA, 1995. IEEE Computer Society.

[15] Olivier Faugeras, Quang-Tuan Luong, and T. Papadopoulou. *The Geometry of Multiple Images: The Laws That Govern The Formation of Images of A Scene and Some of Their Applications*. MIT Press, Cambridge, MA, USA, 2001.

[16] Olivier D. Faugeras and Renaud Keriven. Complete dense stereovision using level set methods. In *ECCV '98: Proceedings of the 5th European Conference on Computer Vision-Volume I*, pages 379–393, London, UK, 1998. Springer-Verlag.

[17] Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, 1981.

[18] Andrew S. Glassner. *Graphics Gems*. Academic Press, Inc., Orlando, FL, USA, 1990.

[19] G. H. Golub and C. F. Van Loan. *Matrix Computations*. John Hopkins University Press, Baltimore, 1983.

[20] J.A. Grunert. Das pothenotische problem in erweiterter gestalt nebst ueber seine anwendungen in der geodäsie. *Grunerts Archiv für Mathematik und Physik, Band 1*, pages 238–248, 1841.

[21] H H. Longuet-Higgins. The reconstruction of a plane surface from two perspective projections. In *Proceedings of Royal Society of London*, volume 227 of B, pages 399–410, 1986.

[22] Robert M. Haralick, Chung-Nan Lee, Karsten Ottenberg, and Michael Noelle. Review and analysis of solutions of the three point perspective pose estimation problem. *Int. J. Comput. Vision*, 13(3):331–356, 1994.

[23] Chris Harris and Mike Stephens. A combined corner and edge detector. In M. M. Matthews, editor, *Proceedings of the 4th ALVEY vision conference*, pages 147–151, University of Manchester, England, September 1988.

[24] John W. Harris and Horst Stocker. *The Handbook of Mathematics and Computational Science*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1997.

[25] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[26] Richard Hartley. Cheirality. *International Journal of Computer Vision*, 26(1):41–61, 1998.

[27] W. Hofmann. *Das problem der Gefährlichen Flächen in Theorie und Praxis -Ein Beitrag zur Hauptaufgabe der Photogrammetrie*. PhD thesis, Fakultät für Bauwesen, Technische Universität München, Germany, 1953.

[28] Fredrik Kahl, Bill Triggs, and Kale Aastrom. Critical motions for auto-calibration when some intrinsic parameters can vary. *Journal of Mathematical Imaging and Vision*, 13(2):131–146, October 2000.

[29] Emil Kalman, Rudolph. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[30] Reinhard Koch, Marc Pollefeys, and Luc J. Van Gool. Multi viewpoint stereo from uncalibrated video sequences. In *ECCV '98: Proceedings of the 5th European Conference on Computer Vision-Volume I*, pages 55–71, London, UK, 1998. Springer-Verlag.

[31] E. Kruppa. Zur ermittlung eines objektes aus zwei perspektiven mit innerer orientierung. *Kaiser- licher Akademie der Wissenschaften (Wien) Kl. Abt*, IIa(122):1939–1948, 1913.

[32] Kiriakos N. Kutulakos and Steven M. Seitz. A theory of shape by space carving. *Int. J. Comput. Vision*, 38(3):199–218, 2000.

[33] Stéphane Laveau and Olivier D. Faugeras. Oriented projective geometry for computer vision. In *ECCV '96: Proceedings of the 4th European Conference on Computer Vision-Volume I*, pages 147–156, London, UK, 1996. Springer-Verlag.

[34] H. C. Longuet-Higgins. A computer algorithm for reconstructing a scene from two projections. *Nature*, 293:133–135, 1981.

[35] C. Loop and Z. Zhang. Computing rectifying homographies for stereo vision. Technical Report MSR-TR-99-21, Microsoft Research, 1999.

[36] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[37] Stephen Maybank. *Theory of Reconstruction from Image Motion*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1992.

[38] G. Van Meerbergen, M. Vergauwen, M. Pollefeys, and L. Van Gool. A hierarchical symmetric stereo algorithm using dynamic programming. *International Journal on Computer Vision*, 47(1):275–285, April 2002.

[39] Florica Mindru, Theo Moons, and Luc J. Van Gool. Recognizing color patterns irrespective of viewpoint and illumination. In *CVPR*, pages 1368–1373, 1999.

[40] Theo Moons. A guided tour through multiview relations. In *SMILE'98: Proceedings of the European Workshop on 3D Structure from Multiple Images of Large-Scale Environments*, pages 304–346, London, UK, 1998. Springer-Verlag.

[41] Gordon E. Moore. Cramming more components onto integrated circuits. *Readings in computer architecture*, pages 56–59, 2000.

[42] S. Negahdaripour. Closed-form relationship between the two interpretations of a moving plane. *Journal of the Optical Society of America*, 7(2):279–285, 1990.

[43] David Nister. An efficient solution to the five-point relative pose problem. In *Proc. Conf. Computer Vision and Pattern Recognition*, volume 2, pages 195–202. IEEE Computer Society Press, 2003.

[44] Steven Maybank Olivier Faugeras. Motion from point matches: multiplicity of solutions. *International Journal of Computer Vision*, 4(3):225–246, 1990.

[45] Johan Philip. A non-iterative algorithm for determining all essential matrices corresponding to five point pairs. *The Photogrammetric Record*, 15(88):589–599, 1996.

[46] M. Pollefeys, L. Van Gool, M. Vergauwen, F. Verbiest, K. Cornelis, J. Tops, and R. Koch. Visual modeling with a hand-held camera. *International Journal of Computer Vision*, 59(3):207–232, 2004.

[47] M. Pollefeys, R. Koch, and L. Van Gool. Self-calibration and metric reconstruction in spite of varying and unknown internal camera parameters. *International Journal of Computer Vision*, 32(1):7–25, 1999.

[48] M. Pollefeys, F. Verbiest, and L. Van Gool. Surviving dominant planes in uncalibrated structure and motion recovery. In *Proceedings of the 7th European Conference on Computer Vision-Part II*, pages 837–851, 2002.

[49] Marc Pollefeys, Reinhard Koch, and Luc J. Van Gool. A simple and efficient rectification method for general motion. In *Proc. of Intl. Conference on Computer Vision 1999*, pages 496–501, 1999.

[50] Randi J. Rost, John M. Kessenich, Barthold Lichtenbelt, and Marc Olano. *Open Gl Shading Language*. Addison-Wesley Professional, 2004.

[51] Jianbo Shi and Carlo Tomasi. Good features to track. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'94)*, Seattle, June 1994.

[52] Ken Shoemake and Tom Duff. Matrix animation and polar decomposition. In *Proceedings of the conference on Graphics interface '92*, pages 258–264, San Francisco, CA, USA, 1992. Morgan Kaufmann Publishers Inc.

[53] Gregory G. Slabaugh, W. Bruce Culbertson, Thomas Malzbender, Mark R. Stevens, and Ronald W. Schafer. Methods for volumetric reconstruction of visual scenes. *Int. J. Comput. Vision*, 57(3):179–199, 2004.

[54] C. Slama. *Manual of Photogrammetry*. American Society of Photogrammetry, Falls Church, VA, USA, 4th edition, 1980.

[55] C. Strecha, R. Fransen, and L. Van Gool. Wide-baseline stereo from multiple views: A probabilistic account. In *Proc. CVPR 2004*, pages 552–559, 2004.

[56] C. Strecha, T. Tuytelaars, and L. Van Gool. Dense matching of multiple wide-baseline views. In *Proc. 2003 International Conference on Computer Vision (2)*, pages 1194–1201, 2003.

[57] C. Strecha, F. Verbiest, M. Vergauwen, and L. Van Gool. Shape from video v.s. still images. In *Proc. Optical 3D measurement techniques (2)*, pages 168–175, 2003.

[58] Peter Sturm. Critical motion sequences for monocular self-calibration and uncalibrated euclidean reconstruction, 1996.

[59] Peter Sturm. Critical motion sequences and conjugacy of ambiguous euclidean reconstructions. In M. Frydrych, J. Parkkinen, and A. Visa, editors, *Proceedings of the 10th Scandinavian Conference on Image Analysis, Lappeenranta, Finland*, volume I, pages 439–446, June 1997.

[60] Peter Sturm. Critical motion sequences for the self-calibration of cameras and stereo systems with variable focal length. In *British Machine Vision Conference, Nottingham, England*, pages 63–72, Sep 1999.

[61] Phil Torr. An assessment of information criteria for motion model selection. In *Proc. CVPR97*, pages 47–53, 1997.

[62] B. Triggs. The absolute quadric. In *Proc. 1997 Conference on Computer Vision and Pattern Recognition*, pages 609–614, 1997.

[63] B. Triggs, P. McLauchlan, R. Hartley, and A. Fitzgibbon. Bundle adjustment: A modern synthesis. *Vision Algorithms: Theory and Practice, LNCS*, 1883:298–372, 2000.

[64] Roger Y. Tsai. A versatile camera calibration technique for high-accuracy 3d machine vision metrology using off-the-shelf tv cameras and lenses. *Radiometry*, pages 221–244, 1992.

[65] G. Vogiatzis, P. Torr, and R. Cipolla. Multi-view stereo via volumetric graph-cuts. In *Proc. CVPR 2005*, pages 391–398, 2005.

[66] R. Yang and M. Pollefeys. Multi-resolution real-time stereo on commodity graphics hardware, 2003.

[67] Z. Zhang. Parameter estimation techniques: A tutorial with application to conic fitting. *Image and Vision Computing Journal*, 15(1):59–76, 1997.

[68] Zhengyou Zhang. Flexible camera calibration by viewing a plane from unknown orientations. In *ICCV*, pages 666–673, 1999.