Integration of virtual objects into an image from a single view

Geometrical properties of images and panoramas

Alain Pagani
Matr.-Nr. 1013264

Technische Universität Darmstadt
Fachbereich Informatik
Fachgebiet Graphisch-Interaktive Systeme
Fraunhoferstraße 5
D-64283 Darmstadt

Supervisor:
Dipl.-Ing. Didier Stricker
Examiner:
Professor Dr.-Ing. José L. Encarnação

Declaration

I hereby declare that this dissertation and the work described in it is my own work, except where otherwise stated, done only with the indicated sources. All the parts, which were inferred from the sources, are marked as such. It has not been submitted before for any degree or examination, at any other university.

Darmstadt, November 15th 2002

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Diplomarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, 15. November 2002

Abstract

The aim of this work is the study of the integration of virtual objects into single views. The use of single views in augmented reality is much more flexible for the end-user than stereo pairs of images, because they require less intervention and preparation. A fundamental problem of augmented reality is to accurately recover the needed camera parameters. While this problem is well-known when two views are available, specific algorithms must be considered for the case of single views. These algorithms generally make use of the geometric particularities of the scene. The camera model and its geometry are therefore of great importance, and are largely analyzed in this report.
The first task is to calibrate the camera by finding its internal parameters (mainly the focal length and the principal point). One method for it is the use of vanishing points. Techniques for the recovery of vanishing points in an image are presented, as well as known calibration methods based on it. Another way to calibrate the camera is to use planar structures, and an algorithm based on multiple planes is derived. The special case of panorama is also discussed, with details about its construction and its calibration.
The second task for the integration of virtual objects is the estimation of the camera position and orientation relative to a given coordinate system (pose). Again, vanishing points and plane-based methods are discussed. Two novel algorithms for pose estimation from multiple planes are presented. The pose can also be estimated in the case of known 3D points coordinates. Seven recent algorithms based on this assumption are detailed and compared. All the algorithms mentioned in this work have been implemented as a practical goal, and simple objects could be integrated into various images using different techniques. The report discusses the implementation particularities and presents the results obtained by comparative tests.

Zusammenfassung

Das Ziel dieser Arbeit ist die Studie der Integration virtueller Objekte in einzelne Ansichten. Die Benutzung einzelner Ansichten in erweiterter Realität ist viel flexibler für den Benutzer als stereoskopische Bilderpaare, da sie weniger Aufwand und weniger Vorbereitung erfordern. Ein grundlegendes Problem in erweiterter Realität ist die präzise Bewertung der notwendigen Kameraparamter. Wenn zwei Ansichten vorhanden sind, lässt sich dieses Problem leicht lösen. Im Fall nur einer Ansicht müssen spezifische Algorithmen betrachtet werden. Diese Algorithmen nutzen im allgemeinen die geometrischen Eigenschaften der Szene. Das Modell der Kamera und die entsprechende Geometrie sind also außerordentlich wichtig, und werden ausführlich in dieser Arbeit analysiert.

Der erste Schritt ist die Kalibrierung der Kamera. Hier werden die internen Parameter der Kamera (hauptsächlich die Brennweite und der Hauptpunkt) bewertet. Eine Methode benutzt dazu Fluchtpunkte. Die notwendige Techniken zur Detektion von Fluchtpunkten und den darauf basierenden Kalibrierungsmethoden werden in der Arbeit vorgestellt. Eine weitere Möglichkeit der Kalibrierung, die ebenfalls in der Arbeit behandelt wird, ist die Verwendung von Ebenen. Auch Thema der Arbeit ist der besondere Fall des Panoramas, mit einer Beschreibung seiner Konstruktion und seiner Kalibrierung.
Der zweite Schritt für die Integration virtueller Objekte besteht darin, die Position und Orientierung der Kamera relativ zu einem gegebenen Koordinatensystem zu berechnen. Entsprechende Methoden, die auf Fluchtpunkten und auf Ebenen basieren, werden diskutiert. Zwei neue Algorithmen für die Position- und Orientierungsberechnung mit Hilfe von mehreren Ebenen werden vorgestellt. Position und Orientierung der Kamera können auch mittels bekannter 3D Punktkoordinaten berechnet werden. Sieben Algorithmen die auf dieser Vorkenntnis basieren werden detailliert beschrieben und verglichen. Alle erwähnten Algorithmen werden als praktisches Ziel implementiert, und einfache Objekte werden in diverse Bilder mit verschiedenen Techniken integriert. Die Besonderheiten bei der Implementierung und die Ergebnisse von Vergleichtests werden vorgestellt.

Résumé

Le but de ce travail est l'étude de l'intégration d'objets virtuels dans des prises de vues uniques. L'utilisation de vues uniques en réalité augmentée est beaucoup plus flexible pour l'utilisateur que les paires d'images stéréoscopiques, car elles demandent moins d'intervention et de préparation. Un problème fondamental en réalité augmentée est l'évaluation précise des paramètres de la caméra. Si ce problème est bien connu quand deux vues sont disponibles, des algorithmes spécifiques doivent être considérés pour le cas des vues uniques. Ces algorithmes utilisent généralement les particularités géométriques de la scène. Le modèle de la caméra et sa géométrie ont donc une grande importance et sont largement analysés dans ce rapport.
La première tâche est de calibrer la caméra en calculant ses paramètres internes (principalement la distance focale et le point principal). Pour cela, l'une des méthodes est l'utilisation de points de fuite. Les techniques de détection de points de fuite dans une image sont donc abordées, ainsi que les méthodes de calibration s'y rapportant. On peut calibrer la caméra d'une autre façon à l'aide de structures planes; un algorithme fondé sur plusieurs plans est étudié. Le cas particulier du panorama est également discuté, avec une description de sa construction et de sa calibration.
La deuxième tâche nécéssaire à l'intégration d'objets virtuels est le calcul de la position

et de l'orientation de la caméra par rapport à un système de coordonnées fixé (pose). Là encore, des méthodes fondées sur les points de fuites, puis sur les plans, sont discutées. Deux nouveaux algorithmes de calcul de pose dans le cas de plans multiples sont proposés. La pose peut également être calculée dans le cas où les coordonnées 3D de points de référence sont connues; sept algorithmes récents s'y rapportant sont étudiés et comparés. Au niveau pratique, tous les algorithmes mentionnés dans ce travail ont été implémentés, et des objets simples ont pu être intégrés dans diverses images de différentes façons. Le rapport traite des particularités de l'implémentation et présente les résultats obtenus lors de tests comparatifs.

Acknowledgments

I would like to thank my supervisor, Didier Stricker, for his enthusiasm, help and guidance throughout this project.

I also wish to thank:

- All the members of the Department for Virtual and Augmented Reality (A4) of the Fraunhofer IGD for providing an interesting and stimulating working environment, and in particular Mario Becker for his help in the security process and Bernd Lutz for his synthetic images.
- Daniel DeMenthon from the LAMP laboratory at the University of Maryland for helping me in the implementation of his pose estimation method.

Contents

Chapter 1

Introduction

The objective of this work is the study of single views in visual computing. While many algorithms for camera calibration and image augmentation require at least two different views of a scene, the case of single views has become much interest in the computer vision community, as it is much more flexible. It has the advantage to be easier for the end-user, and preexisting photographs can be used for augmentation. However, a single image contains much less information than a set of two or more, and efficient methods must be found to extract the interesting parameters of the images. This report extensively reviews and classifies the known algorithms and suggests novel approaches for dealing with single views. It also describes the implementation of a calibration toolkit and gives the results of comparative tests.

First, the motivations and objectives of my study are outlined. An overview of the remaining of this report is then provided.

1.1 Motivation and objectives

Augmented reality (AR) is a technique in which the user's view is enhanced or augmented with additional information generated from a computer model. Basically, applications of this technology use the virtual objects to aid the user's understanding of his environment. The user can view the augmented scene on a portable screen or with a head-mounted display. For example, the maintenance of a machine is made easier with virtual arrows that show where are the buttons to press. Applications are found in medicine, industrial processes, as well as in advertising and movies production.

In order to make AR systems effective, the computer generated objects and the real scene must be combined seamlessly so that the virtual objects align well with the real ones. It is therefore essential to compute accurately the viewpoint and the optical proper-

ties of the camera. This step is known as *alignment* between real and virtual world.

The alignment process is well understood in the case of stereo image pairs, and much relevant information is available on this topic. The case of single images is more difficult, as no information about the depth of points can be directly recovered. It has however the advantage to be much more flexible, allowing for augmentation of images or videos that have not be taken specially for an augmentation. Even paintings of the Renaissance, which follows the same perspective rules that the eye or a camera, can be augmented, or reconstructed, as seen in [6].

However, when considering only one view of a scene, new algorithms must be found, as the methods for stereo images make use of the points correspondences between the views. In the case of single views, the geometrical properties of the image is the basis of the alignment algorithms. The objective of this work is to easily and correctly recover the camera parameters from these properties. To this aim, a study of the geometry involved in the imaging process is provided, and the known algorithms are detailed, with the idea to classify them into main approaches. With the help of this registration, new found algorithms are also described.

The practical goal of the work is the setup of a calibration toolkit, which enables an user to augment single images in a efficient and fast way. The toolkit was to be implemented in *C++*.

1.2 Overview

Chapter 2 gives an overview about the geometry of a camera, and details useful notions for the algorithms. The hierarchy of geometries is explained there, and is connected to the alignment problem via the properties of each geometry.

Chapter 3 deals with the camera model used in this report and details the two parts of the alignment problem, namely the camera internal calibration and the pose estimation.

Chapter 4 presents the algorithms for camera calibration, divided in plane-based, vanishing points and panorama methods, while chapter 5 deals with the three kinds of pose estimation (plane-based, vanishing points and correspondences methods).

The implementation of the toolkit *MAXCAL* is detailed in chapter 6, where several tests of the algorithms are also described and interpreted. Conclusions are drawn in chapter 7.

Chapter 2

From projective to Euclidean Geometry

2.1 Introduction

Geometry is a good mathematical tool for representing objects in a world. To this aim, Euclidean geometry is most used, insofar as it represents our 3D world very well, and as the known invariants in Euclidean geometry correspond to those of a real world: for example, the sides of objects have known or calculable lengths, intersecting lines describe angles between them, and so on. Euclidean geometry is however not the only possible geometry representation, and is for example limited when it comes to describe the imaging process of a camera.

Geometry can be divided into four groups. The simplest and more general group is projective geometry, which is a basis for all three other groups. Affine geometry is a subgroup of the projective one, and in turn includes metric geometry. The last level is Euclidean geometry.

Each of these groups can be defined by their transformations and invariants. A transformation defined at a geometric level leaves specifics objects or properties unchanged. These objects are called invariants of the geometric level. Each geometry will be explained in term of its transformations and invariants in the next sections of this chapter.

Projective geometry models the imaging process of a camera very well, as it allows for perspective projections. It also provides a mathematical representation appropriate for computations. Knowing the projective geometry of an object and making use of the known invariants of the hierarchy of geometries allows to recover up to the Euclidean geometry of this object.

The following section are based on the introduction to geometry by Faugeras [10],

Hartley and Zisserman [17] and Pollefeys [23].

2.2 Projective geometry

2.2.1 Definitions

A point in projective n-space, $\mathcal{P}^n$, is given by a $(n+1)$-vector of coordinates $\boldsymbol{x} = [x_1, x_2, ..., x_{n+1}]^\top$. At least one of these coordinates must differ from zero. Two points represented by $(n+1)$-vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ are equal if and only if there exists a nonzero scalar λ such that $\boldsymbol{x} = \lambda\boldsymbol{y}$. This will be indicated by $\boldsymbol{x} \sim \boldsymbol{y}$. These coordinates represent the same point and are called *homogeneous coordinates* of the point.

Homogeneous points with coordinate $x_{n+1} = 0$ are called *points at infinity*

A *collineation* is a mapping between projective spaces, which preserves collinearity (i.e. collinear points are mapped to collinear points). A collineation from $\mathcal{P}^n$ to $\mathcal{P}^m$ is mathematically represented by a $((n+1)\times(m+1))$-matrix $\boldsymbol{H}$. Again $\boldsymbol{H}$ and $\lambda\boldsymbol{H}$ represent the same collineation, with λ a nonzero scalar.

A *projective basis* for $\mathcal{P}^n$ is defined as any set of $(n+2)$ points such that no $(n+1)$ of them are linearly dependent. The set $\boldsymbol{e}_i = [0, ..., 1, ..., 0]^\top$ for every $i = 1, ..., n$, where 1 is in the ith position and $\boldsymbol{e}_{n+2} = [1, 1, ..., 1]^\top$ is the standard projective basis. Any point of $\mathcal{P}^n$ can be represented by a linear combination of any $(n+1)$ vectors of the standard basis.

2.2.2 The projective plane

The projective plane is the projective space $\mathcal{P}^2$. A point of $\mathcal{P}^2$ is represented by a 3-vector $\boldsymbol{x} = [x_1, x_2, x_3]$.

Points and lines

In an Euclidean plane, a line is represented by an equation such as $ax + by + c = 0$. In the projective plane, this line is therefore naturally represented by the homogeneous vector $(a, b, c)^\top$ insofar as for any $k \neq 0$, (ka, kb, kc) represents the same line.

A point of representation (x, y) in the Euclidean plane lies on the line if and only if $ax+by+c = 0$ that is if $(x, y, 1)(a, b, c)^\top = 0$. In fact, the equation $(kx, ky, k)(a, b, c)^\top = 0$ is also true for any $k \neq 0$. Therefore is the homogeneous vector $(x, y, 1)$ a natural rep-

resentation of this point in $\mathcal{P}^2$.

There is no formal difference between a point and a line in $\mathcal{P}^2$. The role of line and points can be interchanged in statements concerning the properties of lines and points. This is known as the *principle of duality*. *"To any theorem of 2-dimensional projective geometry there corresponds a dual theorem, which may be derived by interchanging the roles of points and lines in the original theorem"* [17].

A point $\boldsymbol{x}$ lies on a line $\boldsymbol{l}$ if and only if $\boldsymbol{x}^\top \boldsymbol{l} = 0$. The intersection of two lines $\boldsymbol{l}$ and $\boldsymbol{l}'$ is the point $\boldsymbol{x} = \boldsymbol{l} \times \boldsymbol{l}'$. The line through two points $\boldsymbol{x}$ and $\boldsymbol{x}'$ is the line $\boldsymbol{l} = \boldsymbol{x} \times \boldsymbol{x}'$

The line $\boldsymbol{l}_\infty$ with canonical coordinates $(0, 0, 1)$ is the line made of points of intersection of parallel lines.

Conics and dual conics

In Euclidean geometry, second-order curves such as ellipses, parabolas and hyperbolas are easily defined. In projective geometry, these curves are collectively known as *conics*.

The equation of a conic in homogeneous coordinates is:

$$ax_1^2 + bx_2^2 + cx_3^2 + 2dx_1x_2 + 2ex_1x_3 + 2fx_2x_3$$

or in matrix form:

$$\boldsymbol{x}^\top \boldsymbol{C} \boldsymbol{x} = 0$$

where the symmetric matrix $\boldsymbol{C}$ is given by:

$$\boldsymbol{C} = \begin{pmatrix} a & d & e \\ d & b & f \\ e & f & c \end{pmatrix}$$

$\boldsymbol{C}$ is an homogeneous representation of the conic. It has then 5 degrees of freedom (it is represented by 5 independent values) Five points define five equations on the parameters (a, b, c, d, e, f), resulting in an one-dimensional space of solutions. This shows that a conic is entirely defined (up to scale) by a set of five points.

The line tangent to $\boldsymbol{C}$ at a point $\boldsymbol{x}$ on $\boldsymbol{C}$ is given by $\boldsymbol{l} = \boldsymbol{C}\boldsymbol{x}$, as shown by Faugeras in [10].

A line $\boldsymbol{l}$ tangent to the conic $\boldsymbol{C}$ satisfies $\boldsymbol{l}^\top \boldsymbol{C}^* \boldsymbol{l} = 0$. $\boldsymbol{C}^*$ is the dual conic with matrix adjoint to $\boldsymbol{C}$. In case of non-singular symmetric matrix $\boldsymbol{C}^* = \boldsymbol{C}^{-1}$ (up to scale).

Collineations

A 2D-collineation (also known as homography or projectivity) is an invertible mapping from points in $\mathcal{P}^2$ such that preserves collinearity. It is represented by a non singular 3×3 matrix $\boldsymbol{H}$. It is defined by 8 parameters (9 values of the matrix less one for the unimportant scale factor) and has then 8 degrees of freedom. For example, a central projection (such a camera) maps points of a world plane to points of the image plane. Such an homography is written $\boldsymbol{x}' = \boldsymbol{H}\boldsymbol{x}$. Knowing $\boldsymbol{H}$ enables us to remove the projective distortion of a plane in the image. Figure 2.1 shows a metric rectification of a plane with a homography. A well-known result is that the correspondence of 4 pairs of points is sufficient to entirely define $\boldsymbol{H}$ (up to scale). This result is derived in appendix A.1.

(a) (b)

Figure 2.1: **Metric rectification with a planar homography** *Four points suffice to compute the homography which maps a window to a perfect rectangle. (a) Original view of the building facade (b) Metric rectified view*

The point transformation is by definition $\boldsymbol{x}' = \boldsymbol{H}\boldsymbol{x}$. If a point $\boldsymbol{x}$ lies on a line $\boldsymbol{l}$, then the transformed point $\boldsymbol{x}'$ lies on the line $\boldsymbol{l}' = \boldsymbol{H}^{-\top}\boldsymbol{l}$, since the incidence of points on lines is preserved by $\boldsymbol{x}'^{\top}\boldsymbol{l}' = \boldsymbol{x}\boldsymbol{H}^{\top}\boldsymbol{H}^{-\top}\boldsymbol{l} = 0$. This gives the transformation rule for a line:

$$\boldsymbol{l}' = \boldsymbol{H}^{-\top}\boldsymbol{l}$$

In a similar manner, it can be shown that conics transforms as:

$$\boldsymbol{C}' = \boldsymbol{H}^{-\top}\boldsymbol{C}\boldsymbol{H}^{-1}$$

Cross-ratio in $\mathcal{P}^2$

Given 4 colinear points $\boldsymbol{x}_i$, $i = 1, ..., 4$ the *cross ratio* is defined as the ratio of ratios of signed distances. In other words, one can express the points $\boldsymbol{x}_i$ by:

$$\boldsymbol{x}_i = \boldsymbol{y} + \lambda_i \boldsymbol{z}$$

for two points of the line ***y*** and ***z***. Then the cross ratio is by definition:

$$\{\boldsymbol{x}_1, \boldsymbol{x}_2; \boldsymbol{x}_3, \boldsymbol{x}_4\} = \frac{\lambda_1 - \lambda_3}{\lambda_1 - \lambda_4} : \frac{\lambda_2 - \lambda_3}{\lambda_2 - \lambda_4}$$

Like collinearity, the cross-ratios are invariants of a projective transformation. A cross ratio is also defined for 4 lines intersecting at one point. For such a configuration, the cross-ratio is defined as the cross ratio $\{\boldsymbol{x}_1, \boldsymbol{x}_2; \boldsymbol{x}_3, \boldsymbol{x}_4\}$ of their 4 intersection points with any line ***l*** not passing through the common point of intersection (see figure 2.2).

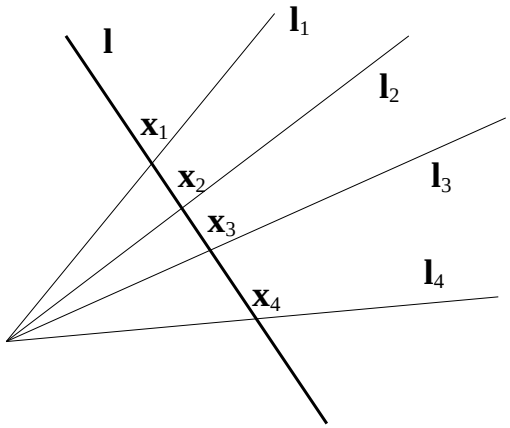


Figure 2.2: **Cross ratio of four concurrent lines** *Four concurrent lines* $\boldsymbol{l}_i$ *intersect the line* ***l*** *in the four points* $\boldsymbol{x}_i$*. The cross ratio of the lines is independent on the line* ***l*** *and is given by the cross ratio of the four points* $\boldsymbol{x}_i$*.*

2.2.3 The projective space

The projective space $\mathcal{P}^3$ is also called simply projective space. A point of $\mathcal{P}^3$ is represented by a 4-vector $\boldsymbol{x} = [x_1, x_2, x_3, x_4]$. The dual entity of a point in $\mathcal{P}^3$ is the plane. It is also described by a 4-vector $\pi = [\pi_1, \pi_2, \pi_3, \pi_4]$. A point ***x*** lies on a plane π if and only if

$$\sum_{i=1}^{4} \pi_i x_i = 0$$

Points with coordinates $x_4 = 0$ form the plane π_∞ called the plane at infinity.

Collineations

A 3D-collineation is an invertible mapping from points in $\mathcal{P}^3$ that preserves collinearity. It is represented by a non singular 4×4 matrix $\boldsymbol{H}$ defined up to a scale factor and has 15 degrees of freedom.

As was the case in $\mathcal{P}^2$ The point transformation is by definition $\boldsymbol{x}' = \boldsymbol{H}\boldsymbol{x}$. If a point $\boldsymbol{x}$ lies on a plane π, then the transformed point $\boldsymbol{x}'$ lies on the plane $\pi' = \boldsymbol{H}^{-\top}\pi$, since the incidence of points on planes is preserved by $\boldsymbol{x}'^{\top}\pi' = \boldsymbol{x}\boldsymbol{H}^{\top}\boldsymbol{H}^{-\top}\pi = 0$. This gives the transformation rule for a plane:

$$\pi' = \boldsymbol{H}^{-\top}\pi$$

Cross-ratio

The cross-ratio in $\mathcal{P}^3$ is defined for four planes $\pi_{\mathbf{1}}, \pi_{\mathbf{2}}, \pi_{\mathbf{3}}$ and $\pi_{\mathbf{4}}$ that intersect at a line $\boldsymbol{l}$. The cross-ratio $\{\pi_{\mathbf{1}}, \pi_{\mathbf{2}}; \pi_{\mathbf{3}}, \pi_{\mathbf{4}}\}$ is by definition the cross-ratio $\{\boldsymbol{l}_1, \boldsymbol{l}_2; \boldsymbol{l}_3, \boldsymbol{l}_4\}$ of their four lines of intersection with any plane π not going through $\boldsymbol{l}$ (see figure 2.3) This cross-ratio is projectively invariant.

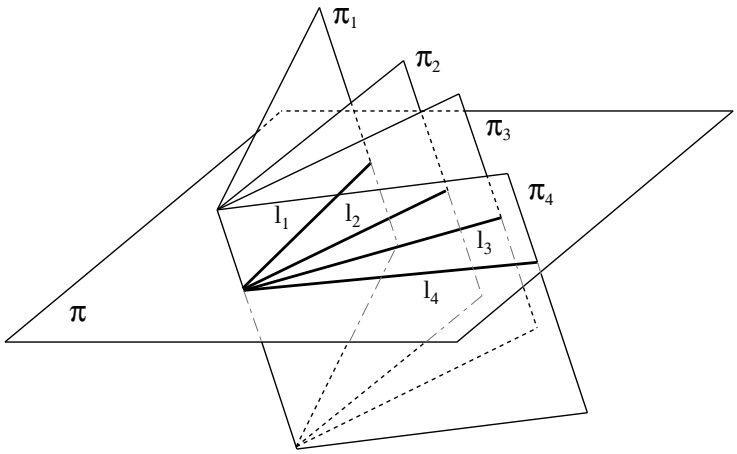


Figure 2.3: **Cross ratio of four planes intersecting at a line** *Four planes intersecting at a line π_i intersect the plane π at the four lines $\mathbf{l}_i$. The cross ratio of the planes is independent of the plane π and is given by the cross ratio of the four lines $\mathbf{l}_i$.*

2.2.4 Discussion

Now that a framework for projective geometry has been created, it is possible to define the 3D Euclidean space as embedded in a projective space $\mathcal{P}^3$. In a similar way, the image

plane of the camera is embedded in a projective space $\mathcal{P}^2$. Then, a collineation exists that maps the 3D space to the image plane $\mathcal{P}^3 \mapsto \mathcal{P}^2$ via a 3×4 matrix. This will be dealt with in detail in the next chapter.

As was outlined, the cross-ratio stays invariant to projective transformations. The relations of *incidence*, *collinearity* and *tangency* are also projectively invariant.

2.3 Affine geometry

The next stratum is the affine one. In the hierarchy of groups it is located between the projective and the metric group. This stratum contains more structure than the projective one, but less than the metric or the Euclidean strata.

2.3.1 The affine plane

The line of the projective plane made of points with $x_3 = 0$ is called the *line at infinity* or $\boldsymbol{l}_\infty$. It is represented by the vector $\boldsymbol{l}_\infty = (0, 0, 1)^\top$.

The affine plane can be considered to be embedded in the projective plane under a correspondence of $\mathcal{A}^2 \mapsto \mathcal{P}^2$: $\boldsymbol{X} = [X_1, X_2]^\top \mapsto [X_1, X_2, 1]^\top$. There is a one-to-one mapping between the affine plane and the projective plane minus the line at infinity (with equation $x_3 = 0$.

Any two distinct affine parallel lines will intersect on the line at infinity. The point of intersection $\boldsymbol{x} = [x_1, x_2, 0]^\top$ has the coordinates of the common *direction* $[x_1, x_2]$ of these lines. Thus, $\boldsymbol{l}_\infty$ can be seen as made of directions of affine lines.

Affine transformations

A point $\boldsymbol{x}$ is transformed in the affine plane as follows:

$$\boldsymbol{x}' = \boldsymbol{B}\boldsymbol{x} + \boldsymbol{b} \tag{2.1}$$

with $\boldsymbol{B}$ being a 2×2 invertible matrix, and $\boldsymbol{b}$ a 2×1 vector. These transformations form a group called the affine group, which is a subgroup of the projective group and which leaves the line at infinity invariant. Note that every point of this line will not be invariant, but the transformed point remains on this line, which is said *globally* invariant.

In projective space $\mathcal{P}^2$ it is then possible to define a collineation that keeps $\boldsymbol{l}_\infty$ invariant. This collineation is defined by a 3×3 matrix $\boldsymbol{A}$ of rank 3:

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{B} & \boldsymbol{b} \\ \boldsymbol{0}_2^\top & 1 \end{bmatrix}$$

2.3.2 The affine space

As in the previous section, the plane at infinity π_∞ has equation $x_4 = 0$ and the affine space can be considered to be embedded in the projective space under a correspondence of $\mathcal{A}^3 \mapsto \mathcal{P}^3$: $\boldsymbol{X} = [X_1, X_2, X_3]^\top \mapsto [X_1, X_2, X_3, 1]^\top$. This is the one-to-one correspondence between the affine space and the projective space minus the plane at infinity (with equation $x_4 = 0$).

As in $\mathcal{P}^2$, it can be seen that any point $\boldsymbol{x} = [x_1, x_2, x_3, 0]^\top$ on π_∞ represents the direction parallel to the vector $[x_1, x_2, x_3]^\top$. This means that two distinct affine parallel planes can be considered as two projective planes intersecting at a line in the plane at infinity π_∞.

Affine transformations

Affine transformations of space can be written exactly as in equation (2.1), but with $\boldsymbol{B}$ being a 3×3 invertible matrix and $\boldsymbol{b}$ a 3×1 vector. Writing the affine transformation using homogeneous coordinates, this can be rewritten as:

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{B} & \boldsymbol{b} \\ \mathbf{0}_3^\top & 1 \end{bmatrix}$$

Once the plane at infinity π_∞ is known, one can upgrade the projective representation to an affine one by applying a transformation which brings the plane at infinity to its canonical position (*i.e.* $\pi_\infty = [0, 0, 0, 1]^\top$).This equation should therefore satisfy:

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \sim \boldsymbol{T}^{-\top} \pi_\infty$$

or:

$$\boldsymbol{T}^\top \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \sim \pi_\infty$$

This equation determines the fourth row of $\boldsymbol{T}$ and all other elements are not constrained:

$$\boldsymbol{T}_{PA} \sim \begin{bmatrix} \boldsymbol{A}_{3\times4} \\ \pi_\infty^\top \end{bmatrix}$$

where the last element of π_∞ is scaled to 1, and $\boldsymbol{A}_{3\times4} = \begin{bmatrix} \boldsymbol{A} & \mathbf{0}_3 \end{bmatrix}$, with $\det(\boldsymbol{A}) \neq 0$.

2.3.3 Discussion

The invariants of the affine stratum are clearly the points, lines and planes at infinity. Affine transformations also preserves parallel lines or planes, ratios of lengths of parallel segments, ratios of areas, known as *affine properties*.

As shown in the previous section, obtaining the plane at infinity (or similary the line at infinity in 2D structures) allows for an upgrade to an affine representation. The plane at infinity can be calculated *e.g.* by finding three vanishing points in the image.

2.4 Metric geometry

The metric stratum corresponds to the group of similarities. These transformations correspond to Euclidean transformations (*i.e.* orthonormal transformation and translation) complemented with a scaling. In this case there are two important new invariant properties: relative lengths and angles. Similar to the affine case, these new invariant properties are related to invariant geometric entities: the *circular points* for 2D and the *absolute conic* for 3D. The metric stratum allows for a complete reconstruction up to an unknown scale.

2.4.1 The metric plane

Affine transformations can be adapted to not only preserve the line at infinity, but also preserve two points on that line called the *absolute points* or *circular points*. The circular points are two complex conjugate points lying on the line at infinity. They are represented by $\boldsymbol{I} = [1, i, 0]^\top$ and $\boldsymbol{J} = [1, -i, 0]^\top$, with $i = \sqrt{-1}$.

Imposing the constraint that $\boldsymbol{I}$ and $\boldsymbol{J}$ be invariant, the following is obtained from equation (2.1):

$$\frac{1}{i} = \frac{b_{11}1 + b_{12}i + b_1 0}{b_{21}1 + b_{22}i + b_2 0}$$
$$\frac{1}{-i} = \frac{b_{11}1 - b_{12}i + b_1 0}{b_{21}1 - b_{22}i + b_2 0}$$

which results in the following:

$$(b_{11} - b_{22})i - (b_{12} + b_{21}) = 0 - (b_{11} - b_{22})i - (b_{12} + b_{21}) = 0$$

Then $(b_{11} - b_{22}) = (b_{12} + b_{21}) = 0$ and the following transformation is obtained:

$$\boldsymbol{X}' = c \begin{bmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{bmatrix} \boldsymbol{X} + \boldsymbol{b} \tag{2.2}$$

where $c > 0$ and $0 \leq \alpha < 2\pi$. This transformation can therefore be interpreted as follows: the affine point $\boldsymbol{X}$ is first rotated by an angle α around the origin, then scaled by c and then translated by $\boldsymbol{b}$. Such a transformation is called a similarity.

Any similarity leaves the circular points invariant.

2.4.2 The metric space

In the metric space, affine transformations are adapted to leave the absolute conic invariant. The absolute conic $\boldsymbol{\Omega}_\infty$ is obtained as the intersection of the quadric of equation $\sum_{i=1}^{4} x_i^2 = 0$ with π_∞:

$$\sum_{i=1}^{4} x_i^2 = x_4 = 0$$

which can be interpreted as an imaginary circle of radius $i = \sqrt{-1}$ in the plane at infinity. All the points on $\boldsymbol{\Omega}_\infty$ have complex coordinates. If $\boldsymbol{x}$ is a point on $\boldsymbol{\Omega}_\infty$, then the complex conjugate point $\bar{\boldsymbol{x}}$ is also on $\boldsymbol{\Omega}_\infty$.

It can be shown that an affine transformation keeps $\boldsymbol{\Omega}_\infty$ invariant if and only if it can be written in the following form:

$$\boldsymbol{X}' = c\boldsymbol{C}\boldsymbol{X} + \boldsymbol{b}$$

where $c > 0$ and $\boldsymbol{C}$ is an orthogonal matrix (*i.e.* $\boldsymbol{C}\boldsymbol{C}^\top = \boldsymbol{I}_{3\times 3}$). The transformation is then defined as a similarity. Writing the affine transformation using homogeneous coordinates, this can be rewritten as in equation (2.2) with:

$$\boldsymbol{T}_M \sim \begin{bmatrix} c\boldsymbol{C} & b \\ \boldsymbol{0}_3^\top & 1 \end{bmatrix} \tag{2.3}$$

While the absolute conic $\boldsymbol{\Omega}_\infty$ is represented by two equations, the dual absolute conic $\boldsymbol{\Omega}_\infty^*$ can be represented as a single quadric:

$$\boldsymbol{\Omega}_\infty^* = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

which is its canonical form. The image of the absolute conic ω (IAC) and the dual image of the absolute conic ω^* (DIAC) are the 2D representations of the conics (after imaging process). Their canonical form are $\omega \sim \boldsymbol{I}_{3\times 3}$ and $\omega^* \sim \boldsymbol{I}_{3\times 3}$

To upgrade the recovered affine representation of the previous section to a metric one, the absolute conic needs to be identified. This is done via the images of the absolute conics as explained in later chapters.

2.4.3 Discussion

The absolute conic is the invariant object of the metric stratum. Two other invariants in this group are relative distances and angles.

As the upgrade from an affine to a metric representation requires the camera calibration matrix, this section is closely related to the topic of camera calibration, which will be described in chapter 4.

2.5 Euclidean geometry

Euclidean geometry is the same as metric geometry, the only difference being that the relative lengths are upgraded to absolute lengths. This means that the Euclidean transformation matrix is the same as in equation (2.3), but without the scale factor:

$$\boldsymbol{T}_M \sim \begin{bmatrix} \boldsymbol{C} & b \\ \mathbf{0}_3^\top & 1 \end{bmatrix}$$

The principal invariant of Euclidean geometry is the Euclidean absolute distance between points.

2.6 Degrees of freedom

A given transformation has a certain degree of freedom (dof) defined as the number of variables needed to defined the transformation. For example a collineation in 2D geometry has 8 dof, whereas an affine transformation only has 6 dof.

2.6.1 Number of invariants

Hartley and Zisserman give a study of the hierarchy of the geometric structure together with an interesting result [17]:
"Given a geometric configuration and a transformation of it, the number of functionally independent invariants is equal to, or greater than, the number of degrees of freedom of the configuration less the number of degrees of freedom of the transformation."

This property can be used for finding the number of invariants of a configuration as well as for finding the geometric transformations needed to upgrade to higher levels in the hierarchy, as explained in the next subsection.

2.6.2 Constraints and degrees of freedom

Given two transformations Tr_1 and Tr_2 with respective degrees of freedom dof_1 and dof_2 (assuming $dof_2 > dof_1$) and one configuration geometric C. One can measure the Tr_1-properties of $Tr_2(C)$ by specifying $dof_2 - dof_1$ functionally independent Tr_1-constraints on the configuration C. With the example of an imaged plane, we have already seen that there is an homography between the image-plane and the real plane. Since an homography has 2 dof more than an affine transformation, one can make affine measurements on the image plane (ratios of areas, ratios of lengths on parallel or colinear segments) once 2 affine-constraints have been given. Such a constraint can be the parallelism of two lines or a known ratio of lengths on colinear segments. In the same way, one can make metric measurements providing 4 metric constraints on that plane, or alternatively 2 affine constraints and 2 metric constraints (using this result twice).

2.7 Summary

In this chapter, we have seen the hierarchy of geometries characterized by their transformations and invariant geometric *properties* that are equivalent to the invariance of a geometric *entity*. This hierarchy has been detailed for the cases of 2D and 3D geometry. These relations are summarized in table 2.1.

Note that the recovery of high-level geometry is necessary for the augmentation of an image. When no absolute yardstick is available, metric geometry is the highest level of geometric structure that can be retrieved from images. Metric and Euclidean geometries differ only by the knowledge of the overall scale factor.

The presented hierarchy of geometries can be helpful for image processing, even without a proper calibration. Indeed, replacing the invariant geometric objects in their canonical position allows for an upgrade to a higher level geometry, where one can make use of the corresponding properties. For example, Criminisi developed a metrology theory from uncalibrated images, where precise affine measurements (*e.g.* ratio of lengths on parallel lines) are deduced (see [6], [5]). A good comprehension of these geometries also leads to simpler algorithms for pose and calibration, as developed in the next chapters.

Geometry	**Matrix**	**Invariant properties**	**Invariant entities**
Projective	$\begin{bmatrix} \boldsymbol{A} & \boldsymbol{t} \\ \boldsymbol{v}^\top & v \end{bmatrix}$	Collinearity, concurrency, cross-ratio	
Affine	$\begin{bmatrix} \boldsymbol{A} & \boldsymbol{t} \\ \boldsymbol{0}^\top & 1 \end{bmatrix}$	Parallelism, ratio of lengths on collinear or parallel lines, ratio of areas and volumes, linear combination of vectors (*e.g.* centroid)	The line at infinity $\boldsymbol{l}_\infty$ (2D), the plane at infinity π_∞ (3D)
Metric	$\begin{bmatrix} s\boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0}^\top & 1 \end{bmatrix}$	Ratio of lengths, angles	The circular points $\boldsymbol{I}$, $\boldsymbol{J}$ (2D), the absolute conic $\boldsymbol{\Omega}_\infty$ (3D)
Euclidean	$\begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0}^\top & 1 \end{bmatrix}$	Length, area, volume	

Table 2.1: **The hierarchy of geometries, corresponding transformation matrices and invariants** *In a n-dimensional space, the matrix **A** is an invertible $n \times n$ matrix, **R** is a n-dimensional rotation matrix, **t** and **v** n-vectors and v a scalar.*

Chapter 3

Camera model and camera calibration

In order to study the properties of images taken by a television, photographic or CCD camera, we need to know an accurate model of the imaging process, which transforms real 3D scenes into a planar image. In this chapter, we will take a closer look to the common used pinhole camera model.

3.1 Camera model

3.1.1 The pinhole camera model

The laws of image formation were already understood by the renaissance painters, who studied geometry in order to reproduce correctly the perspective effects in the images of the world that they were observing. In this section, a model of camera is presented.

We consider the central projection of points in space onto a plane showed in figure 3.1. Let the center of projection be the origin of an Euclidean coordinate system, and consider the plane $z = f$, which is called the *image plane* or *focal plane*.

Under the pinhole camera model, a point in space with coordinates $\boldsymbol{M} = (X, Y, Z)^\top$ is mapped to the point on the image plane where a line joining the point $\boldsymbol{M}$ to the center of projection meets the image plane, as shown in figure 3.1.

By similar triangles, one quickly computes that the point $(X, Y, Z)^\top$ is mapped to the point $(fX/Z, fY/Z, f)^\top$ on the image plane. Ignoring the final image coordinates we see that the transformation

$$(X, Y, Z)^\top \mapsto (fX/Z, fY/Z, f)^\top \tag{3.1}$$

describes the central projection mapping from world to image coordinates.

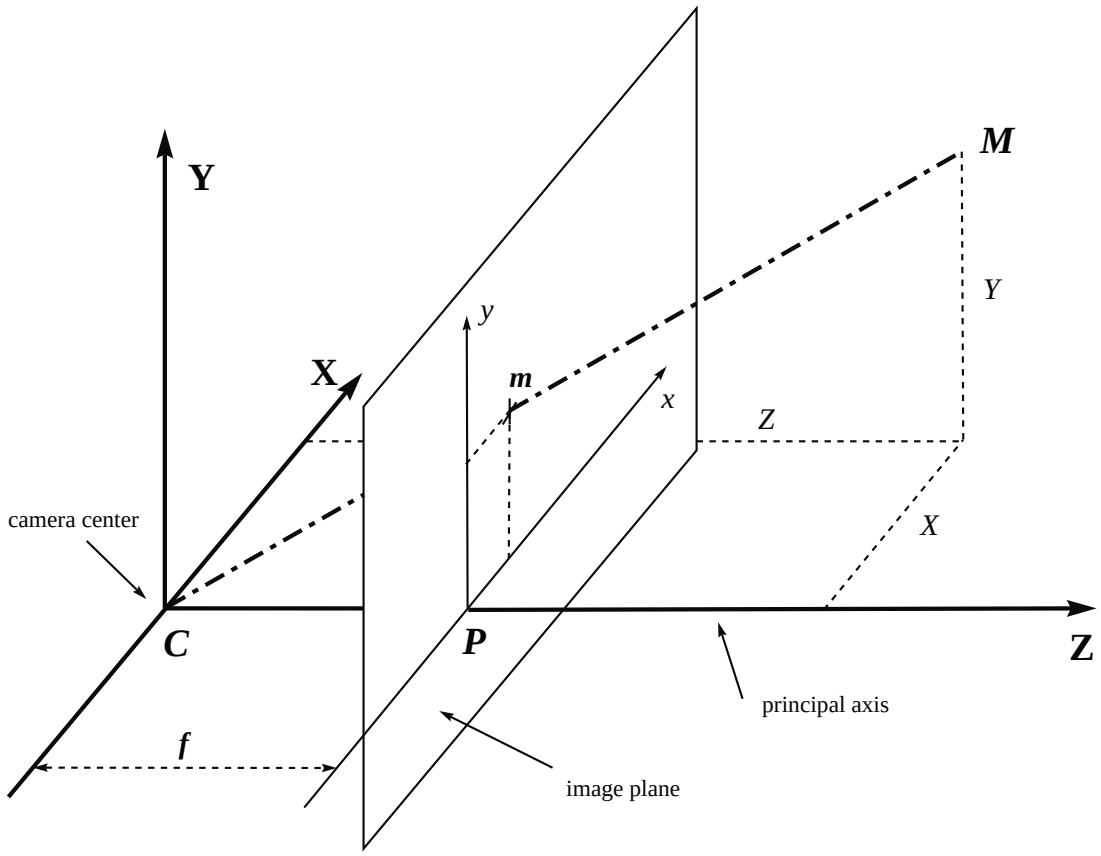


Figure 3.1: **Pinhole camera geometry** ***C*** *is the camera center and* ***P*** *the principal point.*

The center of projection is called the *camera center*. It is also known as the *optical center*. The line from the camera center perpendicular to the image plane is called the *principal axis* of the camera, and the point where the principal axis meets the image plane is called the *principal point*. The plane through the camera center parallel to the image plane is called the *principal plane* of the camera.

3.1.2 The camera projection matrix

If the world and image points are represented by homogeneous vectors, the central projection is simply expressed as a linear mapping between their homogeneous coordinates.

In particular, equation (3.1) may be rewritten in terms of matrix multiplication as:

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \mapsto \begin{pmatrix} fX \\ fY \\ Z \end{pmatrix} = \begin{bmatrix} f & & & 0 \\ & f & & 0 \\ & & 1 & 0 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{3.2}$$

The matrix in this expression may be written as $diag(f, f, 1)[\boldsymbol{I}|\boldsymbol{0}]$. Introducing the notation $\boldsymbol{X}$ for the world point represented by the homogeneous 4-vector $(X, Y, Z, 1)$, $\boldsymbol{x}$ for the image point represented by a homogeneous 3-vector, and $\boldsymbol{P}$ for the 3×4 homogeneous *camera projection matrix*, the equation (3.2) is written compactly as:

$$\boldsymbol{x} = \boldsymbol{P}\boldsymbol{X}$$

which also defines the camera matrix for the pinhole model of central projection as:

$$\boldsymbol{P} = diag(f, f, 1)[\boldsymbol{I}|\boldsymbol{0}]$$

3.1.3 The camera calibration matrix

The expression (3.1) assumes that the origin of coordinates in the image plane is at the principal point. In practice, it may not be, so that in general it exists a mapping

$$(X, Y, Z)^\top \mapsto (fX/Z + p_x, fY/Z + p_y, f)^\top$$

where $(p_x, p_y)^\top$ are the coordinates of the principal point. This equation may be expressed conveniently in homogeneous coordinates as:

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \mapsto \begin{pmatrix} fX + Zp_x \\ fY + Zp_y \\ Z \end{pmatrix} = \begin{bmatrix} f & & p_x & 0 \\ & f & p_y & 0 \\ & & 1 & 0 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{3.3}$$

Now, writing:

$$\boldsymbol{K} = \begin{bmatrix} f & & p_x \\ & f & p_y \\ & & 1 \end{bmatrix} \tag{3.4}$$

equation (3.3) has the concise form:

$$\boldsymbol{x} = \boldsymbol{K}[\boldsymbol{I}|\boldsymbol{0}]\boldsymbol{X} \tag{3.5}$$

The matrix $\boldsymbol{K}$ is called the *camera calibration* matrix.

3.1.4 Camera rotation and translation

Equation (3.5) holds if the camera is located at the origin of the world Euclidean coordinate system with principal axis of the camera pointing straight down the Z-axis, as in figure 3.1. In general, points in space will be expressed in terms of a different Euclidean coordinate frame, known as the *world coordinate frame*. We see in figure 3.2 that the two coordinate frames are related via a rotation and a translation.

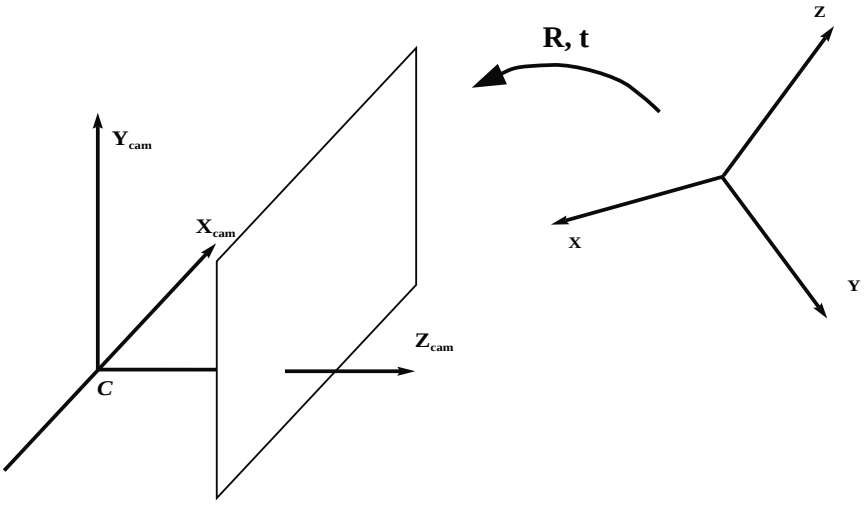


Figure 3.2: **Euclidean transformation between the world and camera frames**

If $\tilde{\boldsymbol{X}}$ is an inhomogeneous 3-vector representing the coordinates of a point in the world coordinate frame, and $\tilde{\boldsymbol{X}}_{cam}$ represents the same point in the camera coordinate frame, then we may write $\tilde{\boldsymbol{X}}_{cam} = \boldsymbol{R}\tilde{\boldsymbol{X}} + \boldsymbol{t}$, where $\boldsymbol{R}$ is a 3×3 rotation matrix representing the orientation of the camera coordinate frame, and $\boldsymbol{t}$ is a 3-vector representing the coordinates of the origin of the world frame in the camera frame. This equation may be written in homogeneous coordinates as:

$$\tilde{\boldsymbol{X}}_{cam} = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0} & 1 \end{bmatrix} \tilde{\boldsymbol{X}} \tag{3.6}$$

Putting this together with (3.5) leads to the formula:

$$\boldsymbol{x} = \boldsymbol{K}[\boldsymbol{R}|\boldsymbol{t}]\boldsymbol{X} \tag{3.7}$$

where $\boldsymbol{X}$ is now in a world coordinate frame. This is the general mapping given by a pinhole camera. The parameters contained in $\boldsymbol{K}$ are called the *internal* camera parameters,

or the *internal orientation* of the camera. The parameters of $\boldsymbol{R}$ and $\boldsymbol{t}$, which relate the camera orientation and position to a world coordinate system, are called the *external* parameters or the *exterior orientation*.

3.1.5 More about $\boldsymbol{K}$

CCD cameras

The pinhole camera model just derived assumes that the image coordinates are Euclidean coordinates having equal scales in both axial directions. In the case of CCD cameras, there is the additional possibility of having non-square pixels. If image coordinates are measured in pixels, rectangular pixels introduce unequal scale factors in each direction. In particular, if the number of pixels per unit distance in image coordinates are m_x and m_y in the x and y directions, then the transformation from world coordinates to pixel coordinates is obtained by multiplying (3.4) on the left by an extra factor $diag(m_x, m_y, 1)$. Thus the general form of the calibration matrix of a CCD camera is:

$$\boldsymbol{K} = \begin{bmatrix} f_x & & x_0 \\ & f_y & y_0 \\ & & 1 \end{bmatrix} \tag{3.8}$$

where $f_x = fm_x$ and $f_y = fm_y$ represent the focal length of the camera in terms of pixel dimensions in the x and y directions respectively. Similarly, $\tilde{\boldsymbol{x}}_0 = (x_0, y_0)$ is the principal point in terms of pixel dimensions, with coordinates $x_0 = m_x p_x$ and $y_0 = m_y p_y$.

Finite projective camera

For added generality, we can consider a calibration matrix of the form:

$$\boldsymbol{K} = \begin{bmatrix} f_x & s & x_0 \\ & f_y & y_0 \\ & & 1 \end{bmatrix} \tag{3.9}$$

The added parameter s is known as the *skew* parameter. The skew parameter will be zero for most normal cameras. However, in certain unusual instances (*e.g.* image of an image), it can take non-zero values.

A camera for which the calibration matrix $\boldsymbol{K}$ is of the form (3.9) will be called a *finite projective camera*.

Note that the left hand 3×3 submatrix of $\boldsymbol{P}$, equal to $\boldsymbol{KR}$, is non-singular. Conversely, any 3×4 matrix $\boldsymbol{P}$ for which the left hand 3×3 submatrix is non-singular is the camera matrix of some finite projective camera, because $\boldsymbol{P}$ can be decomposed as $\boldsymbol{P} = \boldsymbol{K}[\boldsymbol{R}|\boldsymbol{t}]$, using a QR-decomposition of the product $\boldsymbol{KR}$.

3.2 Radial distortion

The pinhole camera model described in the previous section is a good and simple model for linear distortions. However, some video optics — like wide-angle or fish-eye lenses — generates a lot of non-linear distortions. The most important deviation is generally a radial distortion. This section describes the radial distortion problem and the solution applied for this work, inspired by a proposition of Devernay and Faugeras in [9].

3.2.1 Real cameras

A camera follows the pinhole model if and only if the projection of every line in space is viewed in the image as a line. In practice, this may not be the case, and lines will transform into curves, as shown in figure 3.3.

Figure 3.3: **Radial distortion in an image** *The lines at the periphery are the most distorded.*

The mapping between 3D points and 2D image points can be decomposed into a perspective projection and a function that models the deviation from the ideal pinhole camera. The image distortion model is usually given as a mapping from the distorded image coordinates, which are observable in the acquired images, to the undistorded coordinates, which are needed for further calculations.

The image distortion function can be decomposed in two terms: radial and tangential distortion. Radial distortion is a deformation of the image along the direction from a point called the *center of distortion* to the considered image point, and tangential distortion is a deformation perpendicular to this direction. Actually, the tangential distortion need not to be considered, insofar as its effect can be neglected in most of the application. In this

model, the center of distortion can be different from the principal point of the camera.

Let R be the radial distortion function, and $\boldsymbol{c} = (c_x, c_y)$ the center of distortion. $\boldsymbol{x}_d = (x_d, y_d)$ being a point in the distorded image, we define $\hat{\boldsymbol{x}_d} = \boldsymbol{x}_d - \boldsymbol{c}$ as the new coordinates relative to the center of distortion. Similarly, $\boldsymbol{x}_u$ and $\hat{\boldsymbol{x}_u}$ represent the undistorded point (after correction) in the image coordinate system and relative to $\boldsymbol{c}$ respectively. We also define the radius $r_d = \sqrt{\hat{x_d}^2 + \hat{y_d}^2}$ and $r_u = \sqrt{\hat{x_u}^2 + \hat{y_u}^2}$ (see figure 3.4). Then R is the invertible mapping between r_u and r_d:

$$R : r_u \mapsto r_d = R(r_u)$$

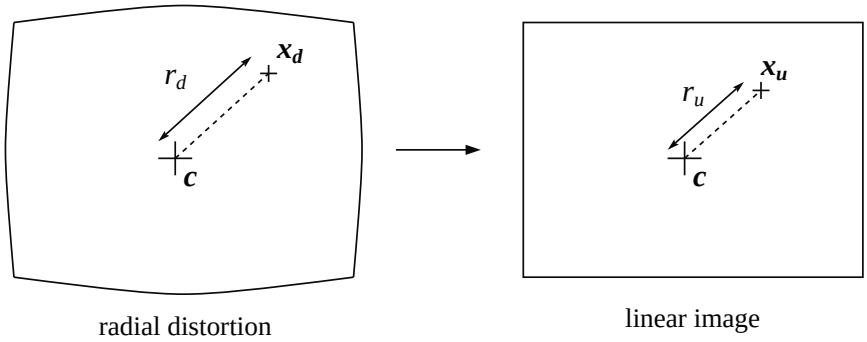


Figure 3.4: **The model of radial distortion**

The distortion model can then be written as:

$$\hat{x_u} = \hat{x_d}\frac{R^{-1}(r_d)}{r_d}, \hat{y_u} = \hat{y_d}\frac{R^{-1}(r_d)}{r_d} \tag{3.10}$$

and similarly the inverse distortion model is given as:

$$\hat{x_d} = \hat{x_u}\frac{R(r_u)}{r_u}, \hat{y_d} = \hat{y_u}\frac{R(r_u)}{r_u}$$

3.2.2 Radial distortion model

The question here is to choose an accurate model for the function R. The lens distortion model (3.10) can be written as a finite series:

$$\hat{\boldsymbol{x}_u} = \hat{\boldsymbol{x}_d}(1 + k_1 r_d^2 + k_2 r_d^4 + ...)$$

It has been showed that using only the first-order parameter k_1 leads to a sufficient accuracy (0.1 pixel is a common value). The undistorded coordinates are then given by the formula:

$$\hat{\boldsymbol{x}_u} = \hat{\boldsymbol{x}_d}(1 + k_1 r_d^2) \tag{3.11}$$

The inverse distortion model is obtained by solving the following equation for r_d, given r_u:

$$r_u = r_d(1 + k_1 r_d^2) \tag{3.12}$$

This is a polynomial of degree three in r_d, and can be solved using the Cardan method, which is a direct method for solving polynomials of degree three (see appendix A.2).

3.2.3 Removing radial distortion

To find the distortion parameters of the camera, the idea is to enforce the lines to be straight [9]. The algorithm is then semi-automatic, the user being asked to give sets of points belonging to the same line (each set represents a line). Then a basic iterative error minimizing algorithm is used, as follows:

- Assign reasonable values to the distortion parameters as initial values (practically $k_1 = 0$ and $\boldsymbol{c}$ in the center of the image).
- Undistord the points using these parameters and equation (3.11).
- Fit each set of undistorded points to a line using an SVD least squares line fitting, as shown in figure 3.5.

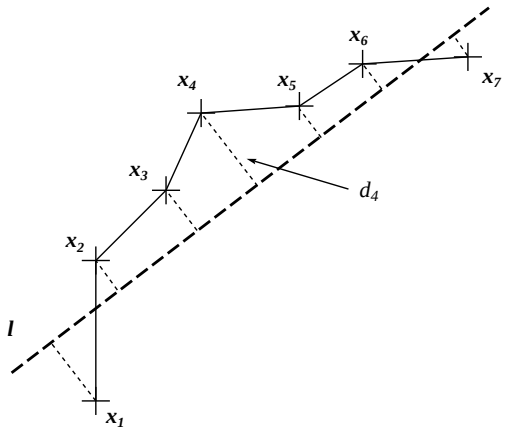
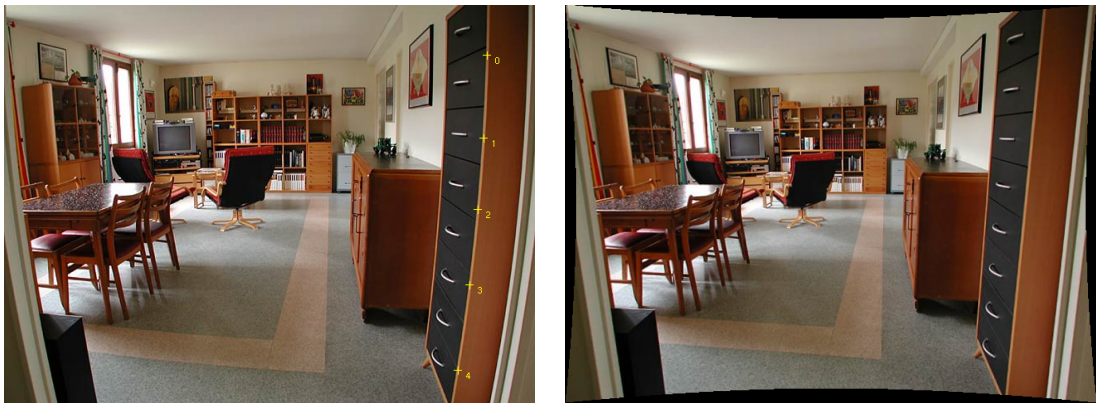


Figure 3.5: **Best fit line for a set of points** *The best fit line for the points $\boldsymbol{x}_i$ is the line $\boldsymbol{l}$ that minimizes the sum of the squared perpendicular distances $\sum_i d_i^2$.*

- Compute the distortion error $\sum \chi^2$ (χ^2 is the sum of the squares of the distances from the points to the corresponding line), the sum being done over all the lines sets.
- Optimize the distortion parameters k_1, c_x, c_y to minimize the total distortion error, by a nonlinear least-squares minimization method (Levenberg-Maquart).
- Compute the new distortion error
- When the relative change of error is less than a threshold, stop here. Else, update the parameters with the optimized values and begin optimization again.

Having found the correct parameters, we usually want to use an image where the distortion effects have been removed. To this aim, a new image is computed. For each pixel of the new image, equation (3.12) is used to find the corresponding coordinates in the distorded original image. In general, these coordinates will not be integers, so that a linear interpolation of the neighbor pixels is computed to find the color of the undistorded pixel. Figure 3.6 shows the results of distortion removal in the image of an indoor scene. In the remaining of this report, it will be assumed that the images have been correctly undistorded using the algorithm presented in this section.



(a) (b)

Figure 3.6: **Distortion removal in an image of interior scene** *(a) Distorded image with selected points (b) Result of distortion removal*

3.3 Camera calibration

When working with images, it is often the case that no information about the camera parameters is available. Camera calibration is therefore a necessary step in 3D computer

vision in order to extract metric information from 2D images. For example, augmenting a real image with virtual objects requires knowledge of the camera projection matrix in the rendering step. In this section camera calibration approaches are outlined in more detail.

3.3.1 The camera projection matrix

As explained in section 3.1, the model used for the camera is the pinhole camera model, where the camera transformation is summarized as a 3×4 matrix $\boldsymbol{P}$:

$$\boldsymbol{x} = \boldsymbol{P}\boldsymbol{X}$$

where $\boldsymbol{X}$ is a real three-dimensional point and $\boldsymbol{x}$ is the corresponding point in the image. $\boldsymbol{P}$ can be decomposed into an upper-triangular 3×3 matrix $\boldsymbol{K}$ and a 3×4 pose matrix $[\boldsymbol{R} \mid \boldsymbol{t}]$:

$$\boldsymbol{P} = \boldsymbol{K}[\boldsymbol{R} \mid \boldsymbol{t}]$$

where $\boldsymbol{R}$ is an orthonormal 3×3 rotation matrix and $\boldsymbol{t}$ a 3-dimensional vector.

The elements of the camera calibration matrix $\boldsymbol{K}$ are the following 5 parameters:

$$\boldsymbol{K} = \begin{pmatrix} f_x & s & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{pmatrix}$$

f_x and f_y are the focal distance in the x and y directions, s is the skew factor, x_0 and y_0 are the coordinates of the cameras principal point.

3.3.2 Calibration approaches

The first method for camera calibration is the direct recovery of the elements of $\boldsymbol{P}$. This computation is known as *resectioning*. It is usually performed from correspondences between 3-space and images entities. The most commonly used resectioning method is perhaps the DLT (direct linear transformation) method originally reported by Abdel-Aziz and Karara [1], which will be discussed in subsection 3.3.3.

When processing more than one image taken by the same camera, or a video sequence, it can be useful to take the intrinsic parameters and the extrinsic one separately into account. To this aim, and following the decomposition of the matrix $\boldsymbol{P}$, camera calibration can be further decomposed into two kinds of calibration:

- the **intrinsic calibration** is the calculation of the intrinsic parameters, *i.e.* of the matrix $\boldsymbol{K}$. When these parameters are fixed for all the images, the calibration has to be done just once.

- the **pose estimation** is the calculation of the matrix $[\boldsymbol{R} \mid \boldsymbol{t}]$. In fact, $\boldsymbol{R}$ is the rotation matrix between the world coordinate and the camera coordinate, and $\boldsymbol{t}$ is the translation between the two centers of these coordinate systems. This calibration can however be performed only if the intrinsic parameters are known, and must be done for each image.

Note that given a matrix $\boldsymbol{P}$, it is always possible to retrieve the decomposition $\boldsymbol{K}[\boldsymbol{R} \mid \boldsymbol{t}]$ by applying a QR-decomposition.

In the literature, the term *camera calibration* is used for both complete calibration (intrinsic and extrinsic orientation) and just intrinsic calibration. In the remaining of this report, we will use this term for intrinsic orientation only, insofar as we will generally focus on the two problems separately.

3.3.3 The DLT algorithm

The direct linear transformation algorithm is a very simple algorithm to compute the matrix $\boldsymbol{P}$ [17]. We assume a number of point correspondences $\boldsymbol{X}_i \leftrightarrow \boldsymbol{x}_i$ between 3D points $\boldsymbol{X}_i$ and 2D image points $\boldsymbol{x}_i$ are given. We are required to find the camera matrix $\boldsymbol{P}$, such that $\boldsymbol{x}_i = \boldsymbol{P}\boldsymbol{X}_i$ for all i. This equation may also be expressed in terms of the vector cross product as $\boldsymbol{x}_i \times \boldsymbol{P}\boldsymbol{X}_i = \boldsymbol{0}$. If the j-th row vector of the matrix $\boldsymbol{P}$ is denoted by $\boldsymbol{P}^j$, then we may write:

$$\boldsymbol{P}\boldsymbol{X}_i = \begin{pmatrix} \boldsymbol{P}^{1\top}\boldsymbol{X}_i \\ \boldsymbol{P}^{2\top}\boldsymbol{X}_i \\ \boldsymbol{P}^{3\top}\boldsymbol{X}_i \end{pmatrix}$$

Writing $\boldsymbol{x}_i = (x_i, y_i, w_i)^\top$, the cross product may then be given explicitly as:

$$\boldsymbol{x}_i \times \boldsymbol{P}\boldsymbol{X}_i = \begin{pmatrix} y_i\boldsymbol{P}^{3\top}\boldsymbol{X}_i - w_i\boldsymbol{P}^{2\top}\boldsymbol{X}_i \\ w_i\boldsymbol{P}^{1\top}\boldsymbol{X}_i - x_i\boldsymbol{P}^{3\top}\boldsymbol{X}_i \\ x_i\boldsymbol{P}^{2\top}\boldsymbol{X}_i - y_i\boldsymbol{P}^{1\top}\boldsymbol{X}_i \end{pmatrix}$$

Since $\boldsymbol{P}^{j\top}\boldsymbol{X}_i = \boldsymbol{X}_i^\top\boldsymbol{P}^j$ for all j, this gives a set of three equations in the entries of $\boldsymbol{P}$, which may be written in the form:

$$\begin{bmatrix} \boldsymbol{0}^\top & -w_i\boldsymbol{X}_i^\top & y_i\boldsymbol{X}_i^\top \\ w_i\boldsymbol{X}_i^\top & \boldsymbol{0}^\top & -x_i\boldsymbol{X}_i^\top \\ -y_i\boldsymbol{X}_i^\top & x_i\boldsymbol{X}_i^\top & \boldsymbol{0}^\top \end{bmatrix} \begin{pmatrix} \boldsymbol{P}^1 \\ \boldsymbol{P}^2 \\ \boldsymbol{P}^3 \end{pmatrix} = \boldsymbol{0} \tag{3.13}$$

The three equations of (3.13) are linearly independent, so that only the first two equations can be used:

$$\begin{bmatrix} \boldsymbol{0}^\top & -w_i\boldsymbol{X}_i^\top & y_i\boldsymbol{X}_i^\top \\ w_i\boldsymbol{X}_i^\top & \boldsymbol{0}^\top & -x_i\boldsymbol{X}_i^\top \end{bmatrix} \begin{pmatrix} \boldsymbol{P}^1 \\ \boldsymbol{P}^2 \\ \boldsymbol{P}^3 \end{pmatrix} = \boldsymbol{0} \tag{3.14}$$

From a set of n points correspondences, we obtain a $2n \times 12$ matrix $\boldsymbol{A}$ by stacking up the equations (3.14) for each correspondence. The projection matrix $\boldsymbol{P}$ is computed by solving the set of equations $\boldsymbol{Ap} = \boldsymbol{0}$, where $\boldsymbol{p}$ is the vector containing the entries of the matrix $\boldsymbol{P}$. This can be done very simply by using a single value decomposition (SVD) of the matrix $\boldsymbol{A}$ [24]. 6 points correspondences are thus needed to retrieve the elements of $\boldsymbol{P}$. If the data is not exact, because of noise in the point coordinates, the SVD method has the advantage to give the best solution in the least-squares sense.

3.3.4 Camera calibration

The camera calibration — or interior orientation problem — has many solution methods which will be discussed in chapter 4. In the following subsection, we will present different approaches of the problem and derive important general results. Basically, the task is to find the parameters of the calibration matrix:

$$\boldsymbol{K} = \begin{pmatrix} f_x & s & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.15}$$

This matrix has in general 5 parameters.

Common assumptions

When calibrating a camera, assumptions are often made to reduce the number of unknowns in the equations. Among these assumptions, we can cite:

Zero-skew assumption The skew effect can be neglected most of the time, insofar as the skew value tends to be small. Setting $s = 0$ gives us a 4-parameter matrix $\boldsymbol{K}$.

Square pixels assumption When the pixels are believed to be square, the values of f_x and f_y are identical (focal length of the camera), resulting in a 4-parameter matrix. This is also called the *unit aspect ratio assumption*, because pixel have an aspect ratio of 1. The case of non-unit, but known aspect ratio α is equivalent to unit aspect ratio in terms of unknowns reduction, providing a simple conditioning step. The zero-skew assumption and the square pixel assumption can be combined together to form the 3-parameter matrix

$$\boldsymbol{K} = \begin{pmatrix} f & 0 & x_0 \\ 0 & f & y_0 \\ 0 & 0 & 1 \end{pmatrix}$$

Known principal point assumption The principal point can be known or arbitrary set to the center of the image. In the latter case, the results seem to be good enough, although the calculated principal point can lie far away from this center. When this assumption is used, the origin of the image coordinate system is often set to the principal point by a pre-translation of the image points, providing the computationally easier case $x_0 = y_0 = 0$. The three mentioned assumptions may be combined together: The resulting matrix $\boldsymbol{K}$ depends only on the parameter f, and is known as the *one-parameter matrix* $\boldsymbol{K}$.

$$\boldsymbol{K} = \begin{pmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

The image of the absolute conic

The image of the absolute conic is an interesting geometrical object for intrinsic calibration. Indeed, there is a simple relation between the IAC ω and the camera calibration matrix $\boldsymbol{K}$, which we will derive in this section.

We refer to the notions of plane at infinity π_∞ and absolute conic $\boldsymbol{\Omega}_\infty$ defined in chapter 2. The points lying on π_∞ are called the *directions* $\boldsymbol{X}_\infty = (X_1, X_2, X_3, 0) = (\boldsymbol{d}^\top, 0)$. $\boldsymbol{\Omega}_\infty$ It is a point conic on π_∞.

With a general camera $\boldsymbol{P} = \boldsymbol{K}[\boldsymbol{R} \mid \boldsymbol{t}]$, points on π_∞ are imaged as:

$$\boldsymbol{x} = \boldsymbol{P}\boldsymbol{X}_\infty = \boldsymbol{K}[\boldsymbol{R} \mid \boldsymbol{t}] \begin{pmatrix} \boldsymbol{d} \\ 0 \end{pmatrix} = \boldsymbol{K}\boldsymbol{R}\boldsymbol{d} \tag{3.16}$$

$\boldsymbol{KR}$ is the planar homography between π_∞ and its image. Now, under a point homography $\boldsymbol{x} \mapsto \boldsymbol{H}\boldsymbol{x}$ a conic $\boldsymbol{C}$ maps as $\boldsymbol{C} \mapsto \boldsymbol{H}^{-\top}\boldsymbol{C}\boldsymbol{H}^{-1}$. Since the absolute conic Ω_∞ is on π_∞, we can compute its image under this camera, and find

$$\begin{aligned} \omega &= \boldsymbol{H}^{-\top}\boldsymbol{I}\boldsymbol{H}^{-1} \\ &= (\boldsymbol{K}\boldsymbol{R})^{-\top}\boldsymbol{I}(\boldsymbol{K}\boldsymbol{R})^{-1} \\ &= \boldsymbol{K}^{-\top}\boldsymbol{R}\boldsymbol{R}^{-1}\boldsymbol{K}^{-1} \\ &= \boldsymbol{K}^{-\top}\boldsymbol{K}^{-1} \end{aligned}$$

Like Ω_∞, ω is an imaginary point conic with no real points. However, it will be very useful for algebraic computations. Its inverse is called the dual image of the absolute conic ω^*.

Once ω or equivalently ω^* is identified in an image, then $\boldsymbol{K}$ is also determined since we have the equations:

$$\omega = \boldsymbol{K}^{-\top}\boldsymbol{K}^{-1} \tag{3.17}$$

$$\omega^* = \boldsymbol{K}\boldsymbol{K}^\top \tag{3.18}$$

We thus use equation (3.18) and the property that a symmetric positive definite matrix may be uniquely decomposed into a product of an upper-triangular matrix and its transpose, by the Choleski factorization [24].

Assumptions and the IAC

As explained above, the image of the absolute conic ω(IAC) and its dual conic ω^* (DIAC) can be used in the calibration algorithms instead of directly $\boldsymbol{K}$. The corresponding matrices are symmetric 3×3 matrices and have 5 degrees of freedom (6 for the upper-triangular part less 1 for the arbitrary scale factor). When $\boldsymbol{K}$ is used, common assumptions are made to reduce this number of parameters. We will now look at the influence of these parameters on ω and ω^*.

The forms of the IAC and the DIAC for a camera with calibration matrix $\boldsymbol{K}$ as in equation (3.15) are:

$$\omega^* = \begin{bmatrix} f_x^2 + s^2 + x_0^2 & sf_y + x_0y_0 & x_0 \\ sf_y + x_0y_0 & f_y^2 + y_0^2 & y_0 \\ x_0 & y_0 & 1 \end{bmatrix}$$

and:

$$\omega = \frac{1}{f_x^2 f_y^2} \begin{bmatrix} f_y^2 & -sf_y & -x_0f_y^2 + y_0sf_y \\ -sf_y & f_x^2 + s^2 & x_0sf_y - y_0f_x^2 - s^2y_0 \\ -x_0f_y^2 + y_0sf_y & x_0sf_y - y_0f_x^2 - s^2y_0 & f_x^2f_y^2 + f_x^2y_0^2 + (f_yx_0 - sy_0)^2 \end{bmatrix}$$

If the zero-skew assumption is made, *i.e.* $s = 0$, then the expressions simplify to:

$$\omega^* = \begin{bmatrix} f_x^2 + x_0^2 & x_0y_0 & x_0 \\ x_0y_0 & f_y^2 + y_0^2 & y_0 \\ x_0 & y_0 & 1 \end{bmatrix}$$

and:

$$\omega = \begin{bmatrix} 1/f_x^2 & 0 & -x_0/f_x^2 \\ 0 & 1/f_y^2 & -y_0/f_y^2 \\ -x_0/f_x^2 & -y_0/f_y^2 & 1 + x_0^2/f_x^2 + y_0^2/f_y^2 \end{bmatrix}$$

Concerning the IAC ω, one can observe following simple computational results.

- if $s = 0$ then $\omega_{12} = \omega_{21} = 0$. This reduces the number of parameters to 4.
- if furthermore $f_x = f_y$, then $\omega_{11} = \omega_{22}$, and ω depends on 3 parameters.

- if the principal point (x_0, y_0) is also known, then $\omega_{13} = \omega_{31} = -x_0\omega_{11}$, and $\omega_{23} = \omega_{32} = -y_0\omega_{22}$, all together leading to a one-parameter matrix ω.

In the case of the DIAC ω^*, the results are slightly different.

- if $s = 0$ then no linear relation between the ω^*_{ij} can be found, and ω^* still has 5 independent unknown parameters.
- if the principal point (x_0, y_0) is known, then $\omega^*_{13} = x_0, \omega^*_{23} = y_0$ and ω^* has 3 independent unknown parameters.
- if $f_x = f_y$ together with the two above mentioned assumptions, then ω^* has 2 independent unknown parameters.

Table 3.1 gives the behavior of the IAC and the DIAC under different assumptions. The number of remaining independent unknowns is given in the last two columns. Under the assumption of known principal point, the IAC and DIAC have the same number of independent parameters as $\boldsymbol{K}$, except when the aspect ratio is known without a zero-skew assumption. However, a known principal point is a far less tenable assumption than the two others, and the principal point is usually an unknown to be recovered. Table 3.1 shows that the IAC should be preferred in this case, insofar as its form more clearly reflects the role of each calibration parameter than does the form of the DIAC.

This results will be useful for determining the minimal number of equations needed for solving equations involving ω or ω^*.

3.3.5 Pose estimation

The pose estimation — or exterior orientation — is the problem of determining the position and attitude of a perspective camera from correspondences between three-dimensional reference objects and their two-dimensional images. It can be solved only once the calibration is known. The basic idea is to find the Euclidean transformation between the camera coordinate system and the world coordinate system (see figure 3.2 *p*27), in form of a 3×3 rotation matrix $\boldsymbol{R}$ and a 3×1 translation vector $\boldsymbol{t}$.

Different algorithm for pose estimation are presented in chapter 5.

KPP	**ZS**	**KAR**	**Constraints on ω**	**Constraints on ω^***	**# K**	**# ω**	**# ω^***
					5	5	5
		×			4	5	5
	×		$\omega_{12} = 0$		4	4	5
	×	×	$\omega_{12} = 0$, $\omega_{11} = r^2\omega_{22}$		3	3	5
×			$\omega_{13} = -x_0\omega_{11} - y_0\omega_{12}$, $\omega_{23} = -x_0\omega_{12} - y_0\omega_{22}$	$\omega^*_{13} = x_0$, $\omega^*_{23} = y_0$	3	3	3
×		×	$\omega_{13} = -x_0\omega_{11} - y_0\omega_{12}$, $\omega_{23} = -x_0\omega_{12} - y_0\omega_{22}$	$\omega^*_{13} = x_0$, $\omega^*_{23} = y_0$	2	3	3
×	×		$\omega_{12} = 0$, $\omega_{13} = -x_0\omega_{11}$, $\omega_{23} = -y_0\omega_{22}$	$\omega^*_{12} = x_0y_0$, $\omega^*_{13} = x_0$, $\omega^*_{23} = y_0$	2	2	2
×	×	×	$\omega_{12} = 0$, $\omega_{13} = -x_0\omega_{11}$, $\omega_{23} = -y_0\omega_{22}$, $\omega_{11} = r^2\omega_{22}$	$\omega^*_{12} = x_0y_0$, $\omega^*_{13} = x_0$, $\omega^*_{23} = y_0$, $\omega^*_{22} - y_0^2 = r^2(\omega^*_{11} - x_0^2)$	1	1	1

Table 3.1: **Effects of common assumptions on the IAC and the DIAC** *Linear constraints obtained under the assumptions of zero skew (ZS), known principal point (x_0, y_0) (KPP) and known aspect ratio $r = f_y/f_x$ (KAR). The three last columns give the number of remaining linearly independent parameters in **K**, ω and ω^* respectively.*

Chapter 4

Single view camera calibration

Camera calibration is the determination of the calibration matrix $\boldsymbol{K}$. When only one view is used, the calibration may be performed with a calibration grid, or from the properties of the scene, such as vanishing points. In this chapter, the main camera calibration methods for a single view are described.

4.1 Calibration from planar structures

4.1.1 One plane

This algorithm performs a camera calibration from 4 coplanar imaged points with known world coordinates.

We make the **zero-skew** assumption and the **known principal point** assumption. Hence, $\boldsymbol{K}$ depends on the two parameters f_x and f_y.

The idea here is to calculate the planar homography $\boldsymbol{H}$ which maps the points on the world plane to the image. Insofar as we consider only one plane, we can always choose the coordinate system so that the points have a null z-coordinate. If the equation of the plan is known and not $z = 0$, it will be no matter to change the world coordinate system after the calibration process.

Now, choosing the plane equation as $z = 0$, a 3D point on the plane has the form $\boldsymbol{X} = (X, Y, 0, 1)$, and is defined by just two coordinates x and y. It is projected into the image via the matrix $\boldsymbol{P}$, yielding the measured 2D point $\boldsymbol{x}$:

$$\begin{aligned} \boldsymbol{x} &= \boldsymbol{P}\boldsymbol{X} \\ &= \boldsymbol{K}\left[\boldsymbol{r}_1\boldsymbol{r}_2\boldsymbol{r}_3\boldsymbol{t}\right]\begin{pmatrix} X \\ Y \\ 0 \\ 1 \end{pmatrix} \\ &= \boldsymbol{K}\left[\boldsymbol{r}_1\boldsymbol{r}_2\boldsymbol{t}\right]\begin{pmatrix} X \\ Y \\ 1 \end{pmatrix} \\ &= \boldsymbol{H}\begin{pmatrix} X \\ Y \\ 1 \end{pmatrix} \end{aligned} \tag{4.1}$$

It is a well known problem to compute a planar homography from points correspondences. Such a homography is a 3×3 matrix defined up to a scale factor and is therefore entirely defined by 8 values. It has been shown that the correspondence of two points gives two independent linear equation on these values, so that 4 sets of two points in general position (no three of them are colinear) are sufficient to define the homography (see also appendix A.1). We assume that this homography has been found. Then the calibration problem is simplified to retrieving $\boldsymbol{P}$ given the homogeneous equation:

$$\boldsymbol{K}\left[\boldsymbol{r}_1\boldsymbol{r}_2\boldsymbol{t}\right] \sim \boldsymbol{H} \tag{4.2}$$

In this section we show how to perform the intrinsic calibration of the camera from the equation (4.2). We first turn this homogeneous equation into an inhomogeneous equation, introducing a homogeneous factor λ:

$$\boldsymbol{H} = \lambda \boldsymbol{K}\left[\boldsymbol{r}_1\boldsymbol{r}_2\boldsymbol{t}\right] \tag{4.3}$$

which we transform:

$$\boldsymbol{K}^{-1}\boldsymbol{H} = \lambda\left[\boldsymbol{r}_1\boldsymbol{r}_2\boldsymbol{t}\right] \tag{4.4}$$

Using the fact that $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ are orthonormal, we can multiply (4.4) by its transpose to obtain:

$$\boldsymbol{H}^{\top}\boldsymbol{K}^{-\top}\boldsymbol{K}^{-1}\boldsymbol{H} = \begin{pmatrix} \lambda^2 & 0 & \times \\ 0 & \lambda^2 & \times \\ \times & \times & \times \end{pmatrix} \tag{4.5}$$

$\boldsymbol{K}^{-\top}\boldsymbol{K}^{-1}$ is known as the *image of the absolute conic* ω and is described in chapter 3. Its matrix can be simplified, when $(x_0, y_0) = (0, 0)$ (provided an easy conditioning step before the computation of $\boldsymbol{H}$) and $s = 0$:

$$\omega = \begin{bmatrix} 1/f_x^2 & 0 & 0 \\ 0 & 1/f_y^2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Then, the equality of the two first terms of the diagonal in the equation (4.5) on the one hand, and the nullity of the upper middle term on the other hand give two linear equations in $1/f_x^2$ and $1/f_y^2$:

$$h_{11}^2 1/f_x^2 + h_{21}^2 1/f_y^2 + h_{31}^2 = h_{12}^2 1/f_x^2 + h_{22}^2 1/f_y^2 + h_{32}^2 \tag{4.6}$$

$$h_{11}h_{12}1/f_x^2 + h_{21}h_{22}1/f_y^2 + h_{31}h_{32} = 0 \tag{4.7}$$

These equations can be solved with *e.g.* a Singular Value Decomposition (SVD) [24], to obtain the values of f_x and f_y.

In a single view, the coordinates of world points are not always known. A good way to use this planar calibration is to manually or automatically detect an imaged square, and then assign the world coordinates $(0,0),(0,1),(1,0)$ and $(1,1)$ to the corners for the calibration.

4.1.2 Multiple planes

The calibration method described above can easily be extended to multiple planes [28], [33]. Having p planes, the idea is to compute the p homographies $\boldsymbol{H}_i$ corresponding to each plane-to-image transformation, and make use of the equations $\boldsymbol{H}_i \sim \boldsymbol{K}[\boldsymbol{r}_1^i \boldsymbol{r}_2^i \boldsymbol{t}^i]$ and of the relations between the $\boldsymbol{r}^i$'s.

We first compute for the p planes the homography between the plane and its image. We need for this the correspondence of 4 points of each plane. This can be simplified by square markers with unit coordinates. We then have for each plane i:

$$\boldsymbol{K}\left[\boldsymbol{r}_1^i \boldsymbol{r}_2^i \boldsymbol{t}\right] \sim \boldsymbol{H}_i \tag{4.8}$$

where $[\boldsymbol{r}_1^i \boldsymbol{r}_2^i \boldsymbol{t}]$ is the transformation matrix from the camera coordinate system to the local coordinate of the plane i.

Like in section 4.1.1, we then use the fact that $\boldsymbol{r}_1^i$ and $\boldsymbol{r}_2^i$ are orthonormal to find:

$$\boldsymbol{H}_i^\top \omega \boldsymbol{H}_i = \begin{pmatrix} \lambda^2 & 0 & \times \\ 0 & \lambda^2 & \times \\ \times & \times & \times \end{pmatrix} \tag{4.9}$$

The equality of the two first terms in the diagonal of equation (4.9) gives one equation in the 6 independent terms of ω:

$$\boldsymbol{h}_1^\top \omega \boldsymbol{h}_1 - \boldsymbol{h}_2^\top \omega \boldsymbol{h}_2 = 0 \tag{4.10}$$

which is linear in the terms of ω:

$$\begin{array}{c} h_{12}h_{11}\omega_{11} + h_{22}h_{21}\omega_{22} + h_{32}h_{31}\omega_{33} \\ +(h_{22}h_{11} + h_{12}h_{21})\omega_{12} + (h_{32}h_{11} + h_{12}h_{31})\omega_{13} \\ +(h_{32}h_{21} + h_{22}h_{31})\omega_{23} = 0 \end{array} \tag{4.11}$$

The nullity of the upper-middle term in equation (4.9) gives another equation in the ω_{ij}'s:

$$\boldsymbol{h}_1^\top \omega \boldsymbol{h}_2 = 0 \tag{4.12}$$

also linear in the terms of ω:

$$\begin{array}{c} (h_{11}^2 - h_{12}^2)\omega_{11} + (h_{21}^2 - h_{22}^2)\omega_{22} + (h_{31}^2 - h_{32}^2)\omega_{33} \\ +2(h_{21}h_{11} - h_{22}h_{12})\omega_{12} + 2(h_{31}h_{11} - h_{32}h_{12})\omega_{13} \\ +2(h_{31}h_{21} + h_{32}h_{22})\omega_{23} = 0 \end{array} \tag{4.13}$$

Applying this for all p planes leads to a set of $2p$ equations in the ω_{ij}'s. When 3 planes are used, the entire matrix ω can be retrieved, but one can also make assumptions to reduce the number of unknowns and improve precision. For example, when zero-skew and square pixels are assumed, only two planes are necessary. If in addition the principal point is known, one plane is enough. In either case, an SVD decomposition [24] of the matrix of equations helps to find the best result. When ω is known, an easy Cholesky decomposition of ω gives the matrix $\boldsymbol{K}$. It is worth noticing here that a too important noise may result in a non positive definite matrix ω, which can not be decomposed.

This algorithm can be used either with p planes in one image, or with one plane seen from different views of a camera with constant intrinsic parameters. When finding the homographies, the patterns (squares, rectangles) do not need to be equally scaled, because only the orthogonality of the local rotation is considered.

A geometric interpretation

Hartley and Zisserman give in [17] a *simple calibration device* consisting of three squares on planes which are not parallel. The homography between each square and its image is computed. Their algorithm then differs from the one presented above: for each plane-to-image homography, the image of the circular points $\boldsymbol{I}, \boldsymbol{J} = (1, \pm i, 0)$ are computed. A conic is then fitted to the six obtained imaged circular points. Since the circular points belong to the absolute conic $\boldsymbol{\Omega}_\infty$, their images belong to the image of the absolute conic (IAC) ω, and the fitted conic is ω.

The derived equations are actually the same as those explained above. Indeed, given a homography $\boldsymbol{H}$, the fact that the image of $\boldsymbol{I}$ and $\boldsymbol{J}$ are points of ω is expressed:

$$\begin{pmatrix} 1 & \pm i & 0 \end{pmatrix} \boldsymbol{H}^\top \omega \boldsymbol{H} \begin{pmatrix} 1 \\ \pm i \\ 0 \end{pmatrix} = 0$$

$$(\boldsymbol{h}_1 \pm i\boldsymbol{h}_2)^\top \omega (\boldsymbol{h}_1 \pm i\boldsymbol{h}_2) = 0$$

Considering separately the real and imaginary parts of this equation gives:

$$\boldsymbol{h}_1^\top \omega \boldsymbol{h}_1 - \boldsymbol{h}_1^\top \omega \boldsymbol{h}_1 = 0$$

and:

$$\boldsymbol{h}_1^\top \omega \boldsymbol{h}_2 = 0$$

which are the equations (4.10) and (4.12). This can be explained geometrically as follows: as seen in chapter 2, the absolute conic $\boldsymbol{\Omega}_\infty$ is an invariant of the *metric geometry*. By enforcing the images of the circular points to be on the IAC, we leave the absolute conic invariant under the transformation, which therefore must be a similarity. The first derivation also assumes that points undergo a similarity (orthogonality and same length of the two first rotation vectors). The fact that the transformation is a similarity is thus expressed in two different ways through these two derivations.

4.2 Calibration from vanishing points

Vanishing points are very important in the calibration process. In this section the contribution of vanishing points is outlined in more details, and example of possible use are given.

4.2.1 Calibration and rays

In the projection process, an image point $\boldsymbol{x}$ back-projects to a ray defined by the camera center and $\boldsymbol{x}$. Calibration relates the image point to the ray's *direction*. Suppose points on the ray are written as $\tilde{\boldsymbol{X}} = \lambda \boldsymbol{d}$ in the camera Euclidean coordinate system, then these points map all to the point $\boldsymbol{x} = \boldsymbol{K}[\boldsymbol{I}|\boldsymbol{0}](\lambda \boldsymbol{d}^\top, 1)^\top = \boldsymbol{K}\boldsymbol{d}$. Thus, the camera calibration matrix $\boldsymbol{K}$ is the transformation between $\boldsymbol{x}$ and the ray's direction $\boldsymbol{d} = \boldsymbol{K}^{-1}\boldsymbol{x}$ measured in the camera's Euclidean coordinate system.

The angle between two rays corresponding to image points $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ can be obtained from the cosine formula for the angle between two vectors:

$$\begin{aligned}\cos\theta &= \frac{(\boldsymbol{K^{-1}x}_1)^\top(\boldsymbol{K^{-1}x}_2)}{\sqrt{(\boldsymbol{K^{-1}x}_1)^\top(\boldsymbol{K^{-1}x}_1)}\sqrt{(\boldsymbol{K^{-1}x}_2)^\top(\boldsymbol{K^{-1}x}_2)}} \\ &= \frac{\boldsymbol{x}_1^\top(\boldsymbol{K^{-\top}K^{-1}})\boldsymbol{x}_2}{\sqrt{\boldsymbol{x}_1^\top(\boldsymbol{K^{-\top}K^{-1}})\boldsymbol{x}_1}\sqrt{\boldsymbol{x}_2^\top(\boldsymbol{K^{-\top}K^{-1}})\boldsymbol{x}_2}} \\ &= \frac{\boldsymbol{x}_1^\top\omega\boldsymbol{x}_2}{\sqrt{\boldsymbol{x}_1^\top\omega\boldsymbol{x}_1}\sqrt{\boldsymbol{x}_2^\top\omega\boldsymbol{x}_2}}\end{aligned} \tag{4.14}$$

where $\omega = \boldsymbol{K}^{-\top}\boldsymbol{K}^{-1}$ is the image of the absolute conic.

In general, this equation can not easily be used to find ω, insofar as it leads to a quadratic equation in the entries of ω. However, when the two points correspond to two orthogonal rays, this equation simply becomes:

$$\boldsymbol{x}_1^\top\omega\boldsymbol{x}_2 = 0 \tag{4.15}$$

Such relations between two points lead to linear relations in the entries of ω. Once ω has been found, it is no matter to retrieve $\boldsymbol{K}$ by a Cholesky decomposition [24]. The question is how to find points whose corresponding rays are orthogonal.

4.2.2 Vanishing points

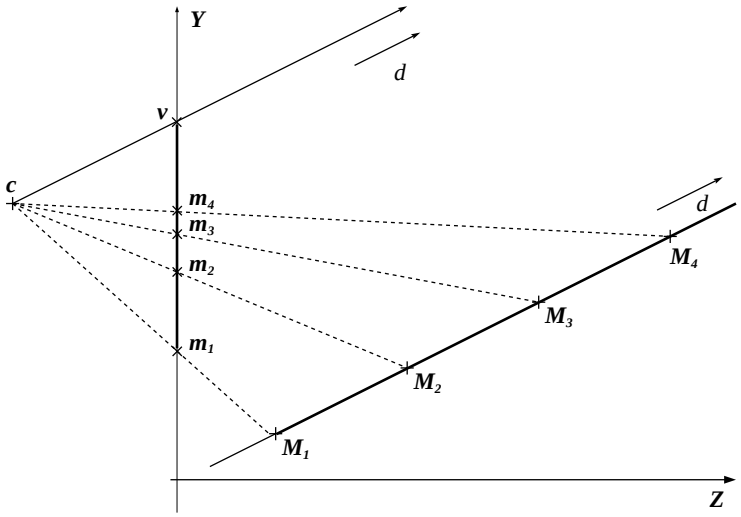


Figure 4.1: **Vanishing point formation** *The vanishing point of the world line is obtained by intersecting the image plane with a ray parallel to the world line through the camera center* $\boldsymbol{c}$.

The perspective geometry that gives rise to vanishing points is illustrated in figure 4.1. It is evident that geometrically the vanishing point of a line is obtained by intersecting the

image plane with a ray parallel to the world line and passing through the camera center. Thus, if two (sets of parallel) lines are orthogonal, their vanishing points will transform into two orthogonal rays, and equation (4.15) can be used to find ω. More generally, if $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ are the vanishing points of two lines in an image, the angle θ between the two lines direction is given by:

$$\cos\theta = \frac{\boldsymbol{v}_1^\top \omega \boldsymbol{v}_2}{\sqrt{\boldsymbol{v}_1^\top \omega \boldsymbol{v}_1}\sqrt{\boldsymbol{v}_2^\top \omega \boldsymbol{v}_2}} \tag{4.16}$$

4.2.3 Finding vanishing points in an image

Any perspective transformation leaves intersections invariant. Two parallel lines intersect on the plane at infinity. Their images will then also intersect at the vanishing point. Finding the vanishing point for the common direction returns to finding the intersection of two or more parallel lines. Alternatively, when only one line is seen in the image, one can used the cross-ratio invariance property.

More than two parallel lines

Determining a vanishing point by the intersection of just two parallel lines leads to high sensibility, especially when the lines lie near from each other. A good improvement is to allow the user to draw all the parallel lines he wants in one direction to find the vanishing point of that direction. However, three or more imaged parallel lines will in general not intersect at the same point in the image and a method must be found to efficiently recover the coordinates of the vanishing point.

Centroid

A first way to solve the multiple line intersection problem is to find the intersection of each pair of lines. This will give $\frac{n(n-1)}{2}$ points of intersection, of which we take the centroid as the correct global intersection point. This does not give the optimal vanishing point, but is an acceptable approximation when the pairwise intersection points are near to each other, at low computational cost.

Linear method

Another method is to find the closest point to all the lines. This is basically an optimization method with quadratic cost function. In this case the partial differential functions are linear and one exact solution can be found.

We consider a line $\boldsymbol{l}_i$ defined by two points $\boldsymbol{a}^i = (a_x^i, a_y^i)^\top$ and $\boldsymbol{b}^i = (b_x^i, b_y^i)^\top$. We note $d_{ix} = (b_x^i - a_x^i)/\|\boldsymbol{b} - \boldsymbol{a}\|$ and $d_{iy} = (b_y^i - a_y^i)/\|\boldsymbol{b} - \boldsymbol{a}\|$. As shown in figure 4.2, if

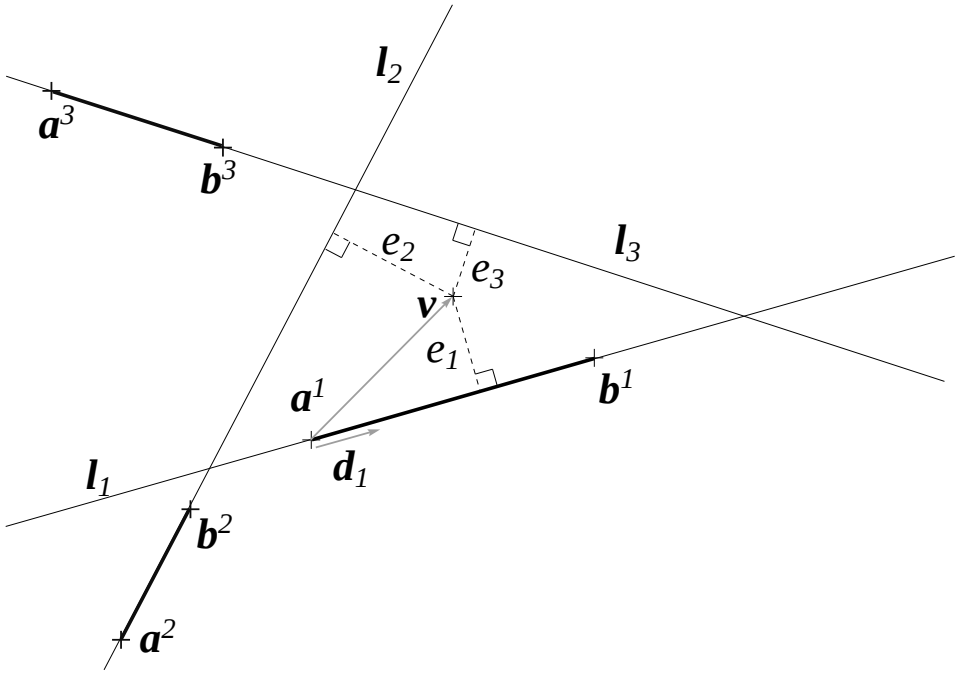


Figure 4.2: **Best-fit intersection point** *The best fit intersection point* $\boldsymbol{v}$ *minimizes the sum* $\sum_i e_i^2$ *of squared perpendicular distances to the lines* $\boldsymbol{l}_i$.

$\boldsymbol{v} = (x, y)^\top$ is the vanishing point to be found, the squared distance between $\boldsymbol{l}_i$ and $\boldsymbol{v}$ is:

$$e_i^2 = (\boldsymbol{v} - \boldsymbol{a}^i)^2 - ((\boldsymbol{v} - \boldsymbol{a}^i).(\frac{\boldsymbol{b}^i - \boldsymbol{a}^i}{\|\boldsymbol{b}^i - \boldsymbol{a}^i\|}))^2$$

which can be rewritten as:

$$e_i^2 = (x - a_x^i)^2 + (y - a_y^i)^2 - ((x - a_x^i)d_{ix} + (y - a_y^i)d_{iy})^2$$

The sum of these distances is:

$$\boldsymbol{E} = \sum_{i=1}^{n} e_i^2$$

The partial derivatives are:

$$\frac{\partial \boldsymbol{E}}{\partial x} = \sum_{i=1}^{n}(2(x - a_x^i) - 2d_{ix}((x - a_x^i)d_{ix} + (y - a_y^i)d_{iy})$$

and:

$$\frac{\partial \boldsymbol{E}}{\partial y} = \sum_{i=1}^{n}(2(y - a_y^i) - 2d_{iy}((x - a_x^i)d_{ix} + (y - a_y^i)d_{iy})$$

which we can write as two linear equations in x and y:

$$\frac{1}{2}\frac{\partial \boldsymbol{E}}{\partial x} = x(n - \sum_{i=1}^{n} d_{ix}^2) - y(\sum_{i=1}^{n} d_{ix}d_{iy}) + \sum_{i=1}^{n}(a_x^i d_{ix}^2 + a_y^i d_{ix}d_{iy} - a_x^i)$$

$$\frac{1}{2}\frac{\partial \boldsymbol{E}}{\partial y} = y(n - \sum_{i=1}^{n} d_{iy}^2) - x(\sum_{i=1}^{n} d_{ix}d_{iy}) + \sum_{i=1}^{n}(a_y^i d_{iy}^2 + a_x^i d_{ix}d_{iy} - a_x^i)$$

We are seeking an extremum of $\boldsymbol{E}$, so that the partial derivatives must be zero. This gives a linear system of two equations in two variables easily solved by computing the inverse of a matrix.

Weighted linear method

The above described method give the same weight to all the lines. A long line is however generally more precise than a small one, so that the same method can be applied with weighted squared errors, the weight being proportional to the square of the line length.

Maximum likelihood estimate

The method of finding the closest point to all the line is however not optimal. The maximum likelihood estimate (MLE) method, derived in [20], tends to not only optimize the vanishing point position but the vanishing point and the n corresponding lines as a set, as shown in figure 4.3.

With the notations of above, the maximum likelihood estimate of the vanishing point is the point $\hat{\boldsymbol{v}}$ such that there exist lines $\hat{\boldsymbol{l}_1}, ..., \hat{\boldsymbol{l}_n})$ that minimize the function

$$(\hat{\boldsymbol{v}}, \hat{\boldsymbol{l}_1}, ..., \hat{\boldsymbol{l}_n}) \mapsto \sum_{i=1}^{n}(d(\hat{\boldsymbol{l}_i}, \boldsymbol{a}^i) + d(\hat{\boldsymbol{l}_i}, \boldsymbol{b}^i))$$

where $d(\boldsymbol{l}, \boldsymbol{x})$ is the perpendicular distance between the line $\boldsymbol{l}$ and the point $\boldsymbol{x}$. Writing $g_j(\hat{\boldsymbol{l}_i}) = d(\hat{\boldsymbol{l}_i}, \boldsymbol{a}^i) + d(\hat{\boldsymbol{l}_i}, \boldsymbol{b}^i))$, the search for an optimal $\hat{\boldsymbol{v}}$ is greatly simplified by the fact that for all $\hat{\boldsymbol{v}}$ and all i, the line $\hat{\boldsymbol{l}_i}$ passing through $\hat{\boldsymbol{v}}$ that minimizes $g_j(\hat{\boldsymbol{l}_i})$ is either the line $\hat{\boldsymbol{l}_i}'$ that passes through $\hat{\boldsymbol{v}}$ and $\boldsymbol{c}^i = (\boldsymbol{a}^i + \boldsymbol{b}^i)/2$, or the line $\hat{\boldsymbol{l}_i}''$ orthogonal to $\hat{\boldsymbol{l}_i}'$ and passing through $\hat{\boldsymbol{v}}$ [14]. One has then to minimize a function of $\hat{\boldsymbol{v}}$ alone:

$$\hat{\boldsymbol{v}} \mapsto \sum_{i=1}^{n} \min\{g_j(\hat{\boldsymbol{l}_i}'), g_j(\hat{\boldsymbol{l}_i}'')\}$$

The minimum of this function is found using a Levenberg-Macquart optimization. The vanishing point given by the weighted linear method is found to be a good initial value for the optimization to succeed. An extension to the case of lines defined by more than 2 points is straightforward, as explained in [14].

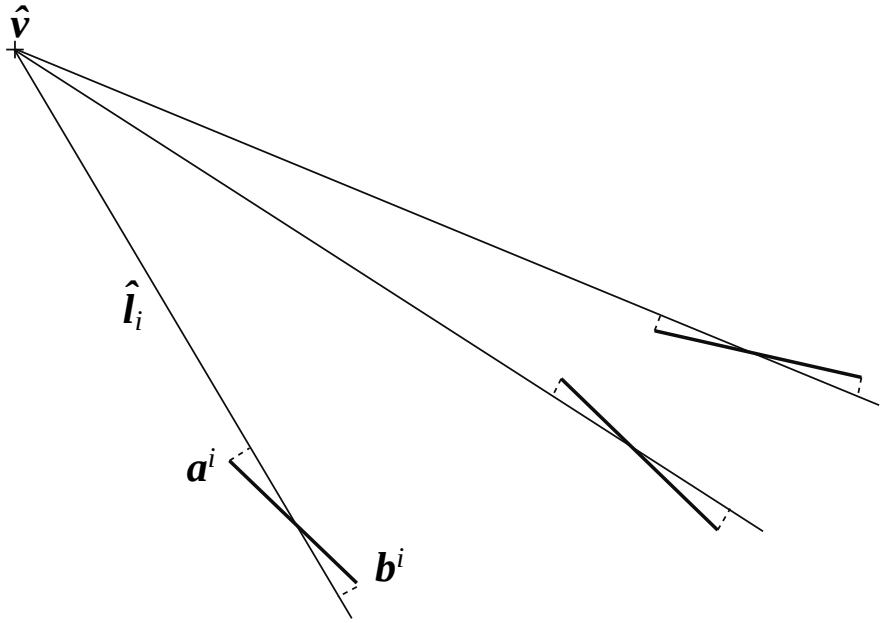


Figure 4.3: **Maximum likelihood estimate vanishing point** *The vanishing point and the lines are optimized as a set.*

Cross-ratio invariance

Given two intervals on a line with known length ratio, the vanishing point on the line may be determined [17]. A typical case is where three points $\boldsymbol{a}'$, $\boldsymbol{b}'$ and $\boldsymbol{c}'$ are identified on a line in an image. Suppose $\boldsymbol{a}$, $\boldsymbol{b}$ and $\boldsymbol{c}$ are the corresponding points on the world line, and the length ratio $|\boldsymbol{ab}| : |\boldsymbol{ac}|$ is known (see figure 4.4).

If $\boldsymbol{v}$ is the point at infinity of that line and $\boldsymbol{v}'$ the vanishing point on the image, the cross-ratio invariance gives:

$$\frac{|\boldsymbol{a}'\boldsymbol{b}'|.|\boldsymbol{v}'\boldsymbol{c}'|}{|\boldsymbol{a}'\boldsymbol{c}'|.|\boldsymbol{v}'\boldsymbol{b}'|} = \frac{|\boldsymbol{ab}|.|\boldsymbol{vc}|}{|\boldsymbol{ac}|.|\boldsymbol{vb}|} = \frac{|\boldsymbol{ab}|}{|\boldsymbol{ac}|}$$

$$1 + \frac{|\boldsymbol{b}'\boldsymbol{c}'|}{|\boldsymbol{v}'\boldsymbol{b}'|} = \frac{|\boldsymbol{ab}|}{|\boldsymbol{ac}|}.\frac{|\boldsymbol{a}'\boldsymbol{c}'|}{|\boldsymbol{a}'\boldsymbol{b}'|}$$

$$|\boldsymbol{v}'\boldsymbol{b}'| = \frac{|\boldsymbol{b}'\boldsymbol{c}'|}{\frac{|\boldsymbol{ab}||\boldsymbol{a}'\boldsymbol{c}'|}{|\boldsymbol{ac}||\boldsymbol{a}'\boldsymbol{b}'|} - 1}$$

which uniquely defines $\boldsymbol{v}'$ (the distances are algebraic). A common usage is the case of equal distance $|\boldsymbol{ab}| = |\boldsymbol{bc}|$, as shown in figure 4.4.

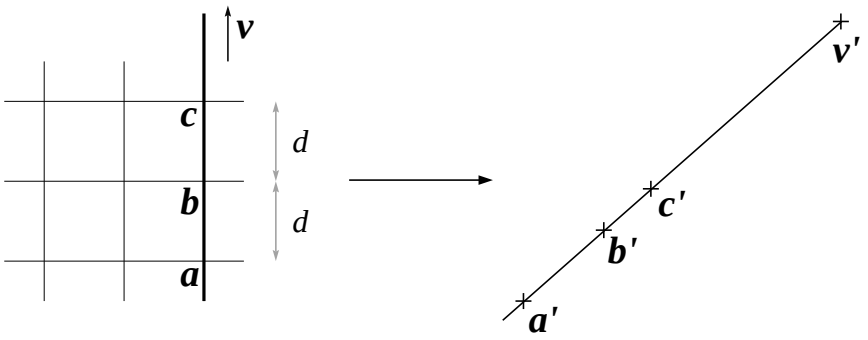


Figure 4.4: **Vanishing point and cross-ratio invariance**

4.2.4 Determining the calibration matrix *K* from vanishing points

ω is a 3×3 symmetric matrix defined up to a scale factor. It is therefore defined by 5 linearly independent parameters (6 for the upper triangular part less 1 for the scale factor). An equation of the type (4.15) give one homogeneous equation linear in these parameters. A minimum of 5 pairs of orthogonal lines is also required to entirely solve the equations and find ω. Once ω has been found, a Choleski decomposition of $\omega = \boldsymbol{K}^{-\top}\boldsymbol{K}^{-1}$ directly gives the calibration matrix $\boldsymbol{K}$, which needs to be scaled so that $k_{33} = 1$. An useful way to solve these equations — especially when more than 5 pairs of lines are used — is the Singular Value Decomposition [24] of the equations matrix, which provides the best solutions in the least-squares sense.

Five pairs of orthogonal lines in general position is, however, not always as easy to find as expected. That is why one may make use of common assumptions on the intrinsic parameters to reduce the number of unknowns (see table 3.1 *p*39).

- if $s = 0$ then the number of parameters is reduced to 4, and so the number of orthogonal pairs of lines to find.
- if furthermore $f_x = f_y$, then 3 pairs are needed.
- if the principal point (x_0, y_0) is also known, only one pair of lines is required.

4.2.5 Two orthogonal vanishing points

When only one pair $(\boldsymbol{u}, \boldsymbol{v})$ of orthogonal vanishing points are available, the matrix $\boldsymbol{K}$ must be reduced to one parameter, by fixing skew to 0, aspect ratio to 1 and principal point

to the center of the image. As the directions $\boldsymbol{K}^{-1}\boldsymbol{u}$ and $\boldsymbol{K}^{-1}\boldsymbol{v}$ are orthogonal, we have $\boldsymbol{u}^\top \boldsymbol{K}^{-\top}\boldsymbol{K}^{-1}\boldsymbol{v} = 0$, that is:

$$\begin{pmatrix} x_u & y_u & 1 \end{pmatrix} \begin{pmatrix} \frac{1}{f^2} & 0 & 0 \\ 0 & \frac{1}{f^2} & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_v \\ y_v \\ 1 \end{pmatrix} = 0$$

which provides:

$$x_u x_v + y_u y_v + f^2 = 0$$

and hence f^2.
Simon used this algorithm for an initialization step of a markerless tracking protocol [26], where the camera is calibrated from vanishing points. The idea here is to recognize a rectangle in a plane, giving two orthogonal sets of parallel lines, and thus two orthogonal vanishing points.

4.2.6 The case of 3 orthogonal vanishing points

A set of three vanishing points, each one being orthogonal to the two others, is often visible in images of human-made environments. In this special case, each subset of 2 vanishing points is a pair of orthogonal vanishing points that can be used in equation (4.15). These 3 pairs of orthogonal vanishing points are sufficient to find the focal length and the principal point of the camera (three-parameter matrix $\boldsymbol{K}$).

Caprile and Torre [3] and Cipolla [4] give the following geometrical interpretation of this result. The image principal point is the orthocenter of a triangle formed by the three vanishing points of orthogonal directions. To show this, let us write equation (4.15) for two orthogonal vanishing points $\boldsymbol{u}$ and $\boldsymbol{v}$ and a 3-parameter matrix $\boldsymbol{K}$:

$$\begin{pmatrix} x_u & y_u & 1 \end{pmatrix} \begin{pmatrix} 1/f^2 & 0 & -x_0/f^2 \\ 0 & 1/f^2 & -y_0/f^2 \\ -x_0/f^2 & -y_0/f^2 & 1 + x_0^2/f^2 + y_0^2/f^2 \end{pmatrix} \begin{pmatrix} x_v \\ y_v \\ 1 \end{pmatrix} = 0$$

which can be rewritten:

$$(\boldsymbol{u} - \boldsymbol{x}_0)(\boldsymbol{v} - \boldsymbol{x}_0)^\top + f^2 = 0 \tag{4.17}$$

where $\boldsymbol{x}_0 = (x_0, y_0, 1)^\top$

Considering the other vanishing point $\boldsymbol{w}$ similarly gives:

$$(\boldsymbol{v} - \boldsymbol{x}_0)(\boldsymbol{w} - \boldsymbol{x}_0)^\top + f^2 = 0 \tag{4.18}$$

$$(\boldsymbol{u} - \boldsymbol{x}_0)(\boldsymbol{w} - \boldsymbol{x}_0)^\top + f^2 = 0 \tag{4.19}$$

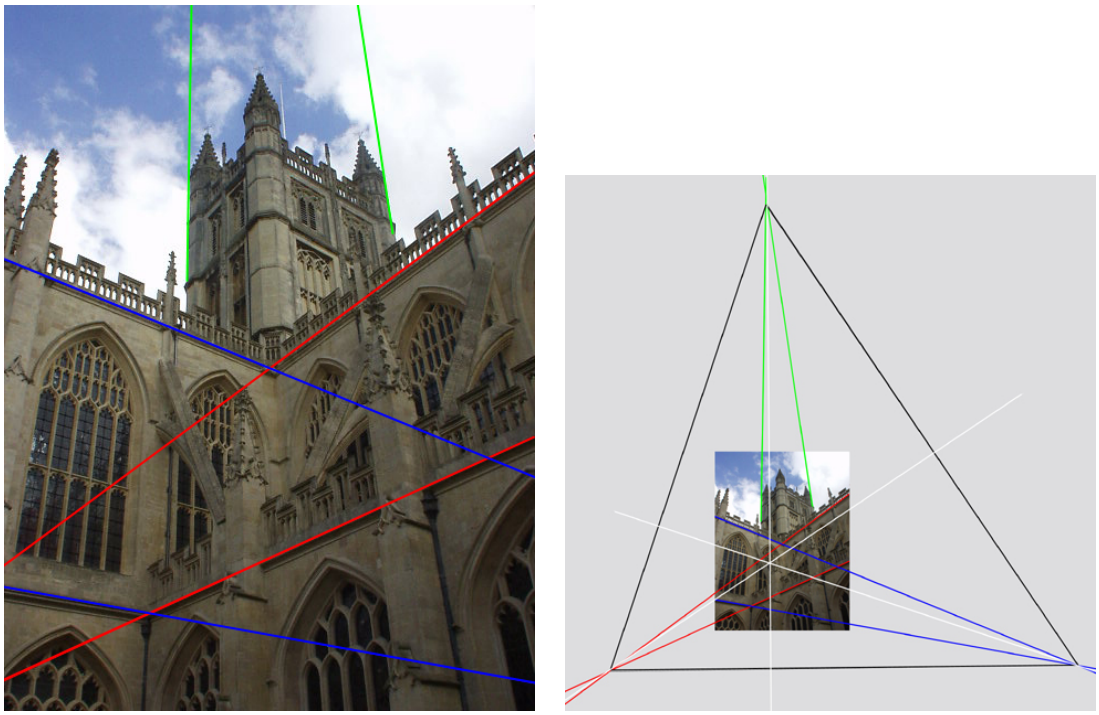
Subtracting (4.19) from (4.19) gives:

$$(\boldsymbol{u} - \boldsymbol{x}_0)(\boldsymbol{v} - \boldsymbol{x}_0)^\top - (\boldsymbol{u} - \boldsymbol{x}_0)(\boldsymbol{w} - \boldsymbol{x}_0)^\top = 0$$

hence:

$$(\boldsymbol{u} - \boldsymbol{x}_0)(\boldsymbol{v} - \boldsymbol{w})^\top = 0$$

This is the condition that $(\boldsymbol{v} - \boldsymbol{w})$ is orthogonal to $(\boldsymbol{u} - \boldsymbol{x}_0)$ (see figure 4.5). The other two orthogonality conditions can be similarly obtained and these imply that $\boldsymbol{x}_0$ is the orthocenter of $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}$. It is thus possible to compute the matrix $\boldsymbol{K}$ in a geometrical way. Equation (4.17) can then be used to compute f.



(a) (b)

Figure 4.5: **Geometric construction of the principal point** *(a) Vanishing points detection with three orthogonal sets of two parallel lines. (b) The principal point is the orthocenter of the triangle formed by three orthogonal vanishing points. Heights are in white.*

4.3 Calibration in panoramas

When the viewing angle of a camera is small, one can construct panoramic views by using properties of the perspective transformation. Such a panorama not only gives a larger view

of the scene, but also includes more information than a single view. As we will see in this section, this information can be helpful for camera calibration.

4.3.1 Planar panoramic mosaicing

We consider a set of images taken with a camera that is rotating around its center. Algebraically, if $\boldsymbol{x}$ and $\boldsymbol{x}'$ are the images of a point $\boldsymbol{X}$ before and after the rotation, we have:

$$\boldsymbol{x} \sim \boldsymbol{K}\left[\boldsymbol{I} \;\middle|\; \boldsymbol{0}\right]\boldsymbol{X}$$

$$\begin{aligned}\boldsymbol{x}' &\sim \boldsymbol{K}\left[\boldsymbol{R} \;\middle|\; \boldsymbol{0}\right]\boldsymbol{X} \\ &\sim \boldsymbol{K}\boldsymbol{R}\boldsymbol{K}^{-1}\boldsymbol{K}\left[\boldsymbol{I} \;\middle|\; \boldsymbol{0}\right]\boldsymbol{X} \\ &\sim \boldsymbol{K}\boldsymbol{R}\boldsymbol{K}^{-1}\boldsymbol{x} \\ &\sim \boldsymbol{H}\boldsymbol{x}\end{aligned}$$

with $\boldsymbol{H} = \boldsymbol{K}\boldsymbol{R}\boldsymbol{K}^{-1}$. There is then a homography $\boldsymbol{H}$ between the two images. Even if the rotation is not known, this homography can be computed from 4 or more point correspondences $\boldsymbol{x} \leftrightarrow \boldsymbol{x}'$, as seen in appendix A.1. In fact, the rotation can also be recovered from $\boldsymbol{H}$ without knowledge of $\boldsymbol{K}$ (see [17] and [30]).

The first application is the planar panoramic mosaicing: images acquired by a rotating camera about its center are related to each other by a planar homography. One can choose one image as reference image and register the others with the reference plane by projectively warping them with the homographies which relate them to the reference image. Then the initial reference image is augmented with the non overlapping part of the warped images. Such a resulting panorama is shown in figure 4.6.

4.3.2 Autocalibration from panoramas

The idea of autocalibration, or self-calibration, is that a camera can be calibrated directly from multiple images, without calibrations grids or knowledge of the 3D structure of the scene. It was first mentioned by Faugeras et. al. [11], and an application for the case of panoramas was developed by Hartley [18].

Linear equations involving ω^* and ω

In the construction of a panorama, an image I_0 is the reference image, and there is an homography $\boldsymbol{H}_i$ between each other image I_i and I_0, the relative motion being a pure rotation. This homography can be computed from correspondences of images points, and

Figure 4.6: **Example of panorama** *Three images are merged into one after a correcting warping.*

requires no information about the 3D scene. Now assuming that the intrinsic parameters are constant for all images, we have for image I_i:

$$\boldsymbol{H}_i \sim \boldsymbol{K}\boldsymbol{R}_i\boldsymbol{K}^{-1}$$

or:

$$\boldsymbol{H}_i\boldsymbol{K} \sim \boldsymbol{K}\boldsymbol{R}_i \tag{4.20}$$

The transpose of this equation is:

$$\boldsymbol{K}^\top\boldsymbol{H}_i^\top \sim \boldsymbol{R}_i^\top\boldsymbol{K}^\top \tag{4.21}$$

and multiplying each sides of equations (4.20) and (4.21) gives:

$$\boldsymbol{H}_i\boldsymbol{K}\boldsymbol{K}^\top\boldsymbol{H}_i^\top \sim \boldsymbol{K}\boldsymbol{R}_i\boldsymbol{R}_i^\top\boldsymbol{K}^\top$$

$$\boldsymbol{H}_i\omega^*\boldsymbol{H}_i^\top \sim \omega^* \tag{4.22}$$

Although equation (4.22) is a homogeneous equation, $\boldsymbol{H}_i$ can be normalized so that $\det \boldsymbol{H}_i = 1$, this fixing the scale factor to be 1. We then have:

$$\boldsymbol{H}_i\omega^*\boldsymbol{H}_i^\top = \omega^* \tag{4.23}$$

which can also be written as:

$$\boldsymbol{H}_i^{-\top}\omega\boldsymbol{H}_i^{-1} = \omega \tag{4.24}$$

Equations (4.23) and (4.24) results in 6 linear equations for the independent elements of the symmetric matrices ω^* and ω respectively. They can then be rewritten in an homogeneous linear form:

$$\boldsymbol{A}_i\boldsymbol{c} = \boldsymbol{0} \tag{4.25}$$

where $\boldsymbol{A}_i$ is a 6×6 matrix and $\boldsymbol{c}$ is the 6×1 vector of the elements of ω^* or ω respectively. If the panorama has been constructed with $m + 1$ views, the matrices $\boldsymbol{A}_i$ can be stacked together to build a $6m \times 6$ matrix $\boldsymbol{A}$, and in general $\boldsymbol{c}$ is determined uniquely from the SVD of $\boldsymbol{A}$ [24].

Ambiguities in autocalibration

Although in equation (4.25) $\boldsymbol{A}_i$ is a 6×6 matrix, two views are not sufficient to solve for $\boldsymbol{c}$. It can be shown (see [18]) that in this case, a one-parameter family of matrices ω^* satisfies the equation (4.23), so that $\boldsymbol{A}_i$ has rank 4. Zisserman et al. reviewed possible ambiguities in [34]. In the case of panoramas, the principal one is the case of rotations with common axis. This ambiguity can be removed by providing more than 2 views with different axis, or by making assumptions on the internal parameters of the camera (see chapter 3). Note that with the zero-skew assumptions, ambiguities still remains when the rotation is about the x-, y- or z-axis of the camera. In this case, the added square pixels assumption resolves the ambiguity.

ω versus ω^*

We have seen that equivalent equations are obtained for ω and ω^*. In fact using the equations involving ω is attractive, because the zero-skew assumption directly gives linear equations in the elements of ω. On the other hand, it is necessary to assume that the principal point is known to obtain such linear equations in the case of ω^*. This second assumption is less tenable than the first one, and ω will therefore be usually preferred.

Extensions

Once ω, or equivalently ω^* has been found, $\boldsymbol{K}$ is deduce by a Cholesky decomposition. Extensions of this algorithm have been described. Hartley gives a method of calibration in the case of varying intrinsic parmeters in [16], and Triggs extended the algorithm to other motions, by using planar scenes [30].

4.4 Summary

Three main approaches for camera calibration in single views have been described. The first one is based on planar structures, and requires the computation of plane-to-image

homographies. Another make use of the vanishing points, which can easily be extracted from images with parallel lines. Different methods exist for estimating the vanishing points, the best one involving a non linear optimization. Panoramas are a special case of single views, insofar as they are single images constructed from multiple views. The homographies computed for this reconstruction are helpful for a dedicated calibration technique. Details of the implementation of the algorithms derived in this chapter are given in chapter 6, along with the results of tests.

Chapter 5

Single view pose estimation

Estimating the pose of the camera is the problem of finding the orientation and position of the camera in a given world coordinate frame. This is a necessary step in augmented reality, because the positions of objects are usually known relative to each other but not to the camera. In this chapter, algorithms for pose estimation in a single view are presented. Where not explicitly stated, we assume that the camera is calibrated, *i.e.* the calibration matrix $\boldsymbol{K}$ is known.

5.1 Pose estimation from planar structures

5.1.1 One plane

As in section 4.1.1, we assume that the planar homography $\boldsymbol{H}$ between a plane and its image is known. $\boldsymbol{H}$ can be found from 4 or more points correspondences. As explained in chapter 4, without loss of generality, we assume that the equation of the plane is $z = 0$. We make use of the equation (4.4) already derived in chapter 4:

$$\boldsymbol{H} = \lambda \boldsymbol{K} [\boldsymbol{r}_1 \boldsymbol{r}_2 \boldsymbol{t}]$$

With the knowledge of $\boldsymbol{K}$ we can easily extract $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ from the normalized two first columns of $\boldsymbol{K}^{-1}\boldsymbol{H}$. This leads to the scale factors λ, and thus to the correct value for $\boldsymbol{t}$. Then, because $\boldsymbol{R}$ is a rotation matrix and therefore an orthonormal matrix, $\boldsymbol{r}_3$ is given by the vector product $\boldsymbol{r}_1 \times \boldsymbol{r}_2$. We can then write:

$$\boldsymbol{P} = \boldsymbol{K} \begin{bmatrix} \boldsymbol{r}_1 & \boldsymbol{r}_2 & (\boldsymbol{r}_1 \times \boldsymbol{r}_2) & \boldsymbol{t} \end{bmatrix}$$

Since we only know $\|\lambda\|$ and not the sign of λ, there subsists an ambiguity about the sign of the orientation matrix. Assuming that the plane is seen in front of the camera, the z-value of $\boldsymbol{t}$ must be positive. This constraint is known as the *visibility constraint*.

Note that this algorithm not always leads to an orthonormal matrix $\boldsymbol{R}$, due mostly to the noise when computing $\boldsymbol{H}$. However, the orthogonality of $\boldsymbol{R}$ can be enforced by applying properties of the Singular Value Decomposition of a matrix (see [24]).

5.1.2 Using a rectangle

The algorithm described in the last section can be easily applied if a square is detected on the reference plane. Then, one has just to compute the homography which maps the four imaged corners to the world coordinates $(0,0)$, $(0,1)$, $(1,1)$ and $(1,0)$ respectively. Simon suggested an original way to use a rectangle in the reference plane in a similar manner [26].

$\boldsymbol{H}$ is the homography which maps the world coordinates of a reference plane to the image. We assume a rectangle has been detected in the reference plane. The four corners are then assigned to the world coordinates of the rectangle $(0,0),(1,0),(1,\alpha),(0,\alpha)$, where α is the (unknown) aspect ratio of the world rectangle. This aspect ratio need not actually be known, since the orthogonality of the lines are another valuable information. We first make the assumption that $\alpha = 1$. The four sets of two points define the homography $\boldsymbol{H}_s$ (s stands for square), which is computed like in the previous section (plane $z = 0$). Now, because we know $\boldsymbol{K}$, and observing that the effect of non-unit α is to premultiply the world coordinates by the diagonal matrix $\boldsymbol{D}$=diag(1,α, 1), we have:

$$\boldsymbol{H}_s = \boldsymbol{H}\boldsymbol{D} \tag{5.1}$$

The arguments of equation (4.1) still hold, and we have in our case:

$$\boldsymbol{H} \sim \boldsymbol{K}[\ \boldsymbol{r}_1 \quad \boldsymbol{r}_2 \quad \boldsymbol{t}\ ] \tag{5.2}$$

which leads to the equation:

$$\boldsymbol{H}_s \sim \boldsymbol{K}[\ \boldsymbol{r}_1 \quad \alpha\boldsymbol{r}_2 \quad \boldsymbol{t}\ ] \tag{5.3}$$

α is thus given by the ratio of the first two columns of the orthogonal matrix $\boldsymbol{K}^{-1}\boldsymbol{H}_s$. We can then retrieve the original homography by $\boldsymbol{H} = \boldsymbol{H}_s\boldsymbol{D}^{-1} = \boldsymbol{H}_s$diag(1, 1/$\alpha$, 1).

5.1.3 Multiple planes pose algorithm

In the last section, we assumed that the plane had equation $z = 0$. However, if the plane has not the equation $z = 0$, the use of a transformation matrix will be helpful. A general transformation matrix $\boldsymbol{M}_p$ from a frame where the plane equation is $z = 0$ to a frame where the equation is $ax + by + cz + d = 0$ can be obtained as follows:

if $a = 0$ and $b = 0$
then

$$\boldsymbol{M}_p = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -d \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

else

$$\boldsymbol{M}_p = \begin{bmatrix} -b & -ac & a & t_x \\ a & -bc & b & t_y \\ 0 & a^2+b^2 & c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where $t_x = -d/a$ and $t_y = 0$ if $a \neq 0$, or $t_x = 0$ and $t_y = -d/b$ if $a = 0$.

Indeed, one can verify that $\boldsymbol{M}_p$ is always invertible, and considering the case where $a \neq 0$, a point $\boldsymbol{X} = (x, y, 0, 1)^\top$ on the plane of equation $z = 0$ will transform to:

$$\boldsymbol{X}' = \boldsymbol{M}_p\boldsymbol{X} = \begin{pmatrix} -bx - acy - d/a \\ ax - bcy \\ (a^2+b^2)y \\ 1 \end{pmatrix}$$

which verifies:

$$ax' + by' + cz' + d = -abx - a^2cy - d + bax - b^2cy + c(a^2+b^2)y + d = 0.$$

Now, let derive a generalized version of equation (4.1). For more readability, we denote by $\langle \boldsymbol{A} \rangle$ a matrix $\boldsymbol{A}$ without its third column. If $\boldsymbol{M}$ is the transformation matrix reflecting the equation of the considered plane, then we have:

$$\begin{aligned} \boldsymbol{H} &\sim \lambda\langle \boldsymbol{PM} \rangle \\ &\sim \boldsymbol{P}\langle \boldsymbol{M} \rangle \\ &\sim \boldsymbol{K}[\boldsymbol{R} \mid \boldsymbol{t}]\langle \boldsymbol{M} \rangle \end{aligned}$$

and thus:

$$\boldsymbol{K}^{-1}\boldsymbol{H} \sim [\boldsymbol{R} \mid \boldsymbol{t}]\langle \boldsymbol{M} \rangle \tag{5.4}$$

Multiplan with known planes equations

Equation (5.4) can then be rewritten as:

$$[\boldsymbol{R} \mid \boldsymbol{t}]\langle \boldsymbol{M} \rangle \boldsymbol{H}^{-1}\boldsymbol{K} \sim \boldsymbol{I}_{3\times 3}$$

where $\boldsymbol{I}_{3\times3}$ is the 3×3 identity matrix.
Calling $\boldsymbol{d}_i, i = 1...3$ the (known) columns of the matrix $\langle \boldsymbol{M} \rangle \boldsymbol{H}^{-1}\boldsymbol{K}$ and $\boldsymbol{q}_i^\top$ the (unknown) rows of the matrix$[\boldsymbol{R} \mid \boldsymbol{t}]$, we have:

$$\begin{pmatrix} \boldsymbol{q}_1^\top \\ \boldsymbol{q}_2^\top \\ \boldsymbol{q}_3^\top \end{pmatrix} \begin{pmatrix} \boldsymbol{d}_1 & \boldsymbol{d}_2 & \boldsymbol{d}_3 \end{pmatrix} \sim \boldsymbol{I}_{3\times3}$$

This last 3×3 matrix homogeneous equation leads to 8 equations in the elements of the $\boldsymbol{q}_i$'s, namely:

$$\boldsymbol{d}_1^\top \boldsymbol{q}_1 - \boldsymbol{d}_2^\top \boldsymbol{q}_2 = 0$$

$$\boldsymbol{d}_1^\top \boldsymbol{q}_1 - \boldsymbol{d}_3^\top \boldsymbol{q}_3 = 0$$

and for $i \neq j$:

$$\boldsymbol{d}_i^\top \boldsymbol{q}_j = 0$$

These equations can be written in the following 8×12 matrix A_p:

$$\begin{bmatrix} \boldsymbol{d}_1^\top & -\boldsymbol{d}_2^\top & \boldsymbol{0} \\ \boldsymbol{d}_1^\top & \boldsymbol{0} & -\boldsymbol{d}_3^\top \\ \boldsymbol{d}_2^\top & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{d}_3^\top & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{d}_1^\top & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{d}_3^\top & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{d}_1^\top \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{d}_2^\top \end{bmatrix} \begin{pmatrix} \boldsymbol{q}_1 \\ \boldsymbol{q}_2 \\ \boldsymbol{q}_3 \end{pmatrix} = 0$$

Applying this for p planes, we obtain a linear equation system of the form $\boldsymbol{A}\boldsymbol{q} = 0$, where $\boldsymbol{A}$ is a $8p \times 12$ matrix, and $\boldsymbol{q} = \begin{pmatrix} \boldsymbol{q}_1^\top & \boldsymbol{q}_2^\top & \boldsymbol{q}_3^\top \end{pmatrix}^\top$. This system can be solved from 2 planes on, using a SVD method [24].

Multiplane without planes equations

The above mentioned multiple planes algorithm has the drawback that the relative positions of the planes have to be known. Thus, one could also compute the coordinates of the planar points in the global world coordinate frame and then use another method, like DLT.

In many applications, the situation where the homographies between the planes and their images are known, but not the planes equations, may arise. For example, if squares are detected on different unknown planes, one can compute for each plane the homography which transforms an unit square to its image. The idea here is to use only these

homographies to compute the global pose. This problem is very similar to the computation of relative orientation between to cameras from planar scenes. Wunderlich showed in [32] the following result: knowing the homography $\boldsymbol{H}$ between two views of a plane allows to find the relative orientation between the two poses, that is, the position of the second camera in the first camera's coordinates frame (see also figure 5.1) . Triggs give in an extension of [30] how to compute this pose from $\boldsymbol{H}$ using a SVD-based method.

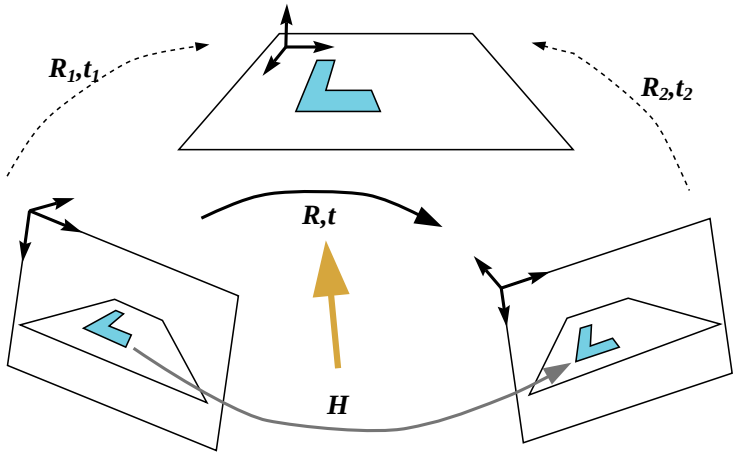


Figure 5.1: **Relative pose from a planar homography** *Given two views of a planar scene, the homography between the images of the plane leads to the transformation between the coordinates systems of the two cameras*

The analogy between one plane seen by two cameras and two planes seen by one camera is however not straightforward. Suppose one plane is seen from two cameras. $\boldsymbol{P}_i$ denotes the pose matrix of the camera in the plane coordinate system for image i, with

$$\boldsymbol{P}_i = \begin{bmatrix} \boldsymbol{R}_i & \boldsymbol{t}_i \\ \boldsymbol{0} & 1 \end{bmatrix}$$

Then from the homography $\boldsymbol{H}_{12}$ between the images we can compute the relative pose $\boldsymbol{P}_r = \boldsymbol{P}_2^{-1}\boldsymbol{P}_1$. If now two planes are seen from one camera, then the decomposition of $\boldsymbol{H}_{12}$ still gives $\boldsymbol{P}_r = \boldsymbol{P}_2^{-1}\boldsymbol{P}_1$, but the transformation between the two planes is $\boldsymbol{P}'_r = \boldsymbol{P}_2\boldsymbol{P}_1^{-1}$. Unfortunately, there is no way to find $\boldsymbol{P}'_r$ from $\boldsymbol{P}_r$.

Instead, we can retrieve the matrices $\boldsymbol{P}_i$ for each plane using the planar pose estimation for one plane explained in section 5.1.1. Then, one plane is taken as a reference plane, say plane 1, and we can compute the relative position of the other planes with the formula $\boldsymbol{P}'_{ri} = \boldsymbol{P}_1\boldsymbol{P}_i^{-1}, i = 2...n$. These $\boldsymbol{P}'_{ri}$ are in turn used in a similar way as the M_p to apply a

multiple plane algorithm with equations.

This algorithm has the drawback that we use the less accurate one-plane algorithm to find the equations, but the advantage that no equations have to be known in advance.

Extension to multiple views

In single view, one or multiple planes can be considered for pose estimation. If multiple views are available, the case of one plane seen in multiple views may be interesting, because the motions between the views can be recovered from an inter-image homography, and the result for one plane can be used for each view, leading to more accuracy. The special case of multiple planes in multiple views is discussed in [27].

5.2 Pose estimation from vanishing points

One property of the vanishing points in camera calibration has been shown in section 4.2 *p*44, where the relations between orthogonal vanishing points and the image of the absolute conic has been outlined. The details about the recovery of vanishing points in an image is also detailed in section 4.2.3 *p*46. In this section, we assume that the three vanishing points corresponding to the three directions of the world coordinate frame have been recognized in the image. We denote by $\boldsymbol{u}, \boldsymbol{v}$, and $\boldsymbol{w}$ the vanishing points for the x-, y- and z-direction respectively.

Now, $\boldsymbol{u}$ is the image of the vector $(1,0,0,0)^\top$, so that we can write:

$$\boldsymbol{u} \sim \boldsymbol{P} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \boldsymbol{K}\boldsymbol{r}_1$$

where $\boldsymbol{r}_1$ is the first column of the rotation matrix $\boldsymbol{R}$. Thus,

$$\boldsymbol{r}_1 \sim \boldsymbol{K}^{-1}\boldsymbol{u}$$

Since $\boldsymbol{R}$ is an orthonormal matrix, $\boldsymbol{r}_1$ must have an unit length. $\boldsymbol{r}_1$ is then defined by:

$$\boldsymbol{r}_1 = \frac{\boldsymbol{K}^{-1}\boldsymbol{u}}{\|\boldsymbol{K}^{-1}\boldsymbol{u}\|}$$

$\boldsymbol{r}_2$ and $\boldsymbol{r}_3$ are similarly found from $\boldsymbol{v}$ and $\boldsymbol{w}$, and the complete matrix $\boldsymbol{R}$ is:

$$\boldsymbol{R} = \begin{pmatrix} \frac{\boldsymbol{K}^{-1}\boldsymbol{u}}{\|\boldsymbol{K}^{-1}\boldsymbol{u}\|} & \frac{\boldsymbol{K}^{-1}\boldsymbol{v}}{\|\boldsymbol{K}^{-1}\boldsymbol{v}\|} & \frac{\boldsymbol{K}^{-1}\boldsymbol{w}}{\|\boldsymbol{K}^{-1}\boldsymbol{w}\|} \end{pmatrix}$$

Note that the sign of the vectors $\boldsymbol{r}_i$ cannot be recovered. Since the determinant of $\boldsymbol{R}$ is positive, the three signs are known once two have been set, so that there is generally 4 possible orientations among which we have to choose. The differences between these orientations is only a switch of sense along the principal directions.

The fourth column of the projection matrix depends on the position of the world coordinate system relative to the camera coordinate system. An arbitrary reference point $\boldsymbol{O}$ can be chosen as the origin. Its image coordinates $\boldsymbol{o} = (\boldsymbol{o}_x, \boldsymbol{o}_y, 1)$ fix the translation $\boldsymbol{t}$ up to an arbitrary scale factor λ:

$$\lambda \boldsymbol{o} = \boldsymbol{P} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \boldsymbol{K}\boldsymbol{t}$$

hence:

$$\boldsymbol{t} = \lambda \boldsymbol{K}^{-1} \boldsymbol{o} \tag{5.5}$$

In a single view and with no metric information this scale is indeterminate and can be arbitrary set, *e.g.* $\lambda = 1$. However, a fifth point with known world coordinates will set the metric with no ambiguity. Suppose we know the coordinates between a point $\boldsymbol{M}$ of the world and of its image $\boldsymbol{m}$. Then we have:

$$\boldsymbol{m} \sim \boldsymbol{K}(\boldsymbol{R}\boldsymbol{M} + \boldsymbol{t}) \tag{5.6}$$

Putting (5.5) in (5.6) gives:

$$\boldsymbol{m} \sim \boldsymbol{K}\boldsymbol{R}\boldsymbol{M} + \lambda \boldsymbol{o}$$

The vector product of the two sides of this equation is zero, which uniquely defines λ.

5.3 Pose estimation from 2D/3D correspondences

The information available for solving the pose estimation problem can be given in the form of a set of point correspondences, each composed of a 3D reference point expressed in object coordinates and its 2D projection expressed in image coordinates. Fishler and Bolles [13] first mentioned the "Perspective-n-point problem" as the problem to find the pose of an object from n such point correspondences. Three correspondences are the minimal set which provides a finite number of solutions. Solutions for the perspective-3-point problem can be found in [7] and [15]. For three points, the algorithms are generally slow and they do not provide unique solutions, as they amount to solve fourth degree polynomial equations. Four points generally suffice for uniqueness [25] and there exists linear methods in this case.

The most widely used and most accurate methods for solving pose estimation use iterative optimization methods. In these approaches, a cost function describing a pose error

is computed and iteratively optimized.

In this section, we described algorithms for pose estimation. Their efficiency and robustness has been tested and the results are given in chapter 6.

5.3.1 Direct Linear Transform

The Direct Linear Transform algorithm described in section 3.3.3 *p*34 for the recovery of the entire camera matrix $\boldsymbol{P}$ can also be used for pose estimation only (that is, not simultaneously with camera calibration). To this aim, the image points are first multiplied by $\boldsymbol{K}^{-1}$ and the correspondences $\boldsymbol{K}^{-1}m \leftrightarrow M$ between corrected image points and world points are used as input data for the algorithm. This pose estimation requires to know at least 6 correspondences. As the method solve for each of the entries of the matrix $[\boldsymbol{R}|\boldsymbol{t}]$, the matrix $\boldsymbol{R}$ may not be orthogonal. It can however be enforced using the properties of the Singular Value Decomposition [24].

5.3.2 The absolute orientation problem

Figure 5.2 shows the imaging process of a camera for a point $\boldsymbol{M}_i$. This point is transformed into the point $\boldsymbol{m}_i$, whose coordinates in the camera coordinates system are $(m_{ix}-x_0, m_{iy}-y_0, f)$.

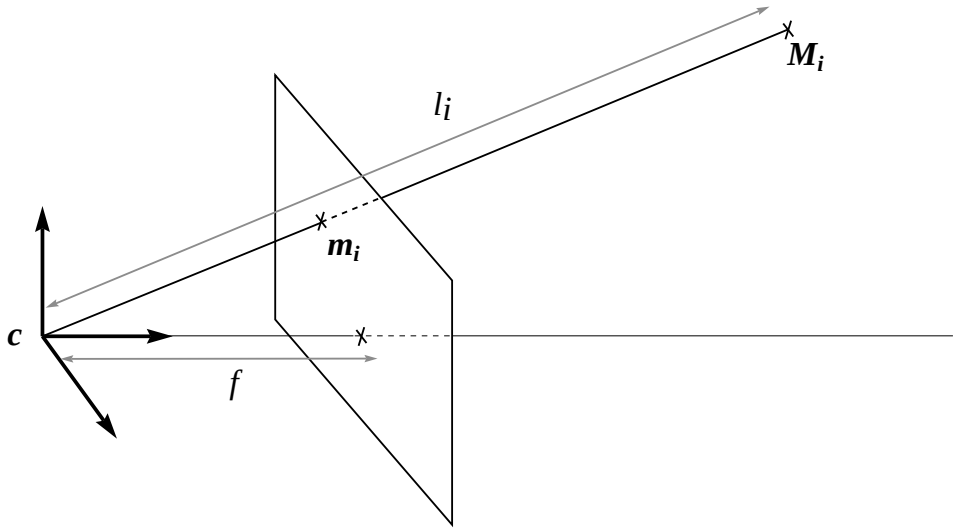


Figure 5.2: **Imaging process of a camera for one point**

In fact, every point of the ray $l_i \boldsymbol{m}_i / \|\boldsymbol{m}_i\|, l_i = 0...\infty$ will transform to $\boldsymbol{m}_i$. Once l_i is known, the coordinates of $\boldsymbol{M}_i$ are defined in the camera coordinate frame. When the normalized vector $\boldsymbol{m}_i / \|\boldsymbol{m}_i\|$ is used, l_i is the camera-point distance. Now suppose the correspondences $\boldsymbol{M}_i \leftrightarrow \boldsymbol{m}_i$ are known for n points and that for each correspondence, the distance l_i is known. We then have the coordinates of the points $\boldsymbol{M}_i$ in the world coordinates frame, and from $l_i \boldsymbol{m}_i / \|\boldsymbol{m}_i\|$ we recover the coordinates of the points $\boldsymbol{M}_i$ in the camera coordinate frame. The pose estimation is then reduced to the problem of finding the relationship between two coordinate systems using pairs of measurements of the coordinates of points in both systems, referred to as *absolute orientation problem*.

A closed-form solution for this problem can be found in [19] and [12]. In fact there is a solution even if the points are scaled from one coordinate system to the other, that is, if a similitude can be retrieved between the two sets of points. For better readability, we denote by $\boldsymbol{a}_i$ the points in one coordinate system and $\boldsymbol{b}_i$ the points in the other. Then we have:

$$\boldsymbol{b}_i = s\boldsymbol{R}(\boldsymbol{a}_i + \boldsymbol{t}), i = 1...n$$

Summing these equations for all i and dividing by n shows that the optimal translation is found via

$$\boldsymbol{t} = s^{-1}\boldsymbol{R}^\top \boldsymbol{b}_0 - \boldsymbol{a}_0$$

where $\boldsymbol{b}_0$ and $\boldsymbol{a}_0$ are the centroids of the two data sets. Combining this with our original equation gives $\tilde{\boldsymbol{b}_i} = s\boldsymbol{R}\tilde{\boldsymbol{a}_i}$, where $\tilde{\boldsymbol{b}_i} = \boldsymbol{b}_i - \boldsymbol{b}_0$ and $\tilde{\boldsymbol{a}_i} = \boldsymbol{a}_i - \boldsymbol{a}_0$.

The rotation matrix does not change the lengths of the vectors, thus a scale s that minimizes the symmetric error is given by:

$$s = \sqrt{\frac{\sum_i \|\tilde{\boldsymbol{a}_i}\|^2}{\sum_i \|\tilde{\boldsymbol{b}_i}\|^2}}$$

Let $\boldsymbol{B}$ be the $3 \times n$ matrix formed by stacking the points $\tilde{\boldsymbol{b}_i}$ side by side and $\boldsymbol{A}$ the matrix formed by stacking the points $\tilde{\boldsymbol{a}_i}$ similarly. The rotation matrix that minimizes the sum of the square of the errors $\sum_i \|\tilde{\boldsymbol{b}_i} - s\boldsymbol{R}\tilde{\boldsymbol{a}_i}\|^2$ is given by:

$$\boldsymbol{R} = \boldsymbol{V}\boldsymbol{U}^\top$$

where $\boldsymbol{U}$ and $\boldsymbol{V}$ are the left and right singular vectors of the SVD $\boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^\top$ of the matrix $\boldsymbol{A}\boldsymbol{B}^\top$.

This solves the absolute orientation problem. Following pose estimation focus then only on retrieving the lengths l_i

5.3.3 Object rigidity

With the notations of the last section, the rigidity of the world object can be expressed in terms of the invariance of the inter-points distance. Indeed, the known distance $d_{ij} =$

$\|\boldsymbol{M}_i - \boldsymbol{M}_j\|$ is the same in both coordinates frames, and thus gives a constraint on the unknown camera-point distances l_i and l_j:

$$d_{ij}^2 = l_i^2 + l_j^2 - 2l_i l_j \cos\theta_{ij} \tag{5.7}$$

where θ_{ij} is the angle between the two rays (see figure 5.3).

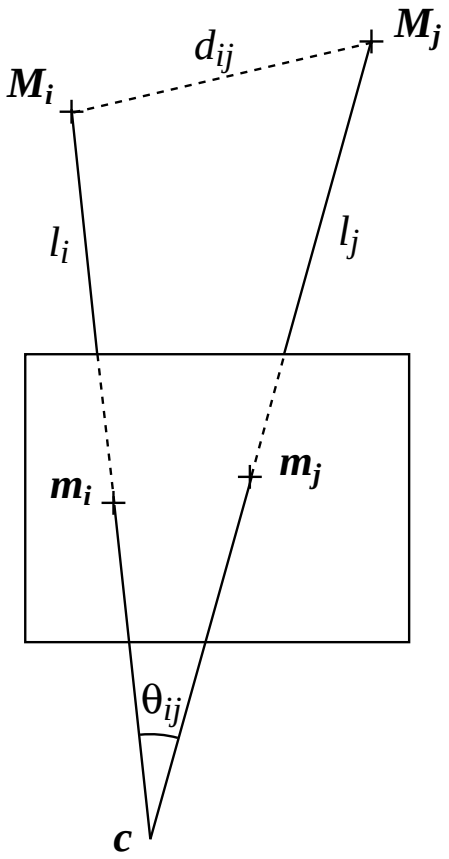


Figure 5.3: **Imaging process for two points** *The lengths are related via the formula* $d_{ij}^2 = l_i^2 + l_j^2 - 2l_i l_j \cos\theta_{ij}$

The cosine of this angle is directly computed from the camera calibration matrix and the points $\boldsymbol{m}_i$ and $\boldsymbol{m}_j$ as:

$$\cos\theta_{ij} = \frac{\boldsymbol{m}_i^\top \omega \boldsymbol{m}_j}{\sqrt{\boldsymbol{m}_i^\top \omega \boldsymbol{m}_i}\sqrt{\boldsymbol{m}_j^\top \omega \boldsymbol{m}_j}}$$

where $\omega = \boldsymbol{K}^{-\top}\boldsymbol{K}^{-1}$.

For n points, we have $\frac{n(n-1)}{2}$ quadric constraints type (5.7). Following algorithms are different ways to solve these equations linearly by changing the variables.

Using resultants

One way to solve the equations of type (5.7) is to use resultants [25]. This classical result allows to eliminate variables in polynomials. Equation (5.7) can be summarized $f_{ij}(l_i, l_j) = 0$. We now show how to recover the length l_1. Considering the equations involving l_1, l_2 and l_i, we can use classical Sylvester resultant to eliminate l_i between $f_{1i}(l_1, l_i)$ and $f_{2i}(l_1, l_i)$ and get a polynomial $h(l_1, l_2)$. Then, further elimination of l_2 between $f_{12}(l_1, l_2)$ and $h(l_1, l_2)$ gives an 8th degree polynomial in l_1 with only even terms, *i.e.* a 4th degree polynomial in $l = l_1^2$.

$$g(l) = a_5 l^4 + a_4 l^3 + a_3 l^2 + a_2 l + a_1 = 0$$

what we can also write $\boldsymbol{a}^\top \boldsymbol{x} = 0$, with $\boldsymbol{a} = (a_1, a_2, a_3, a_4, a_5)^\top$ and $\boldsymbol{x} = (x_0, x_1, x_2, x_3, x_4)^\top = (1, l, l^2, l^3, l^4)^\top$.

For n points, we have $\frac{n(n-1)}{2}$ quadric constraint of type $f_{ij}(l_i, l_j) = 0$ and $\frac{(n-1)(n-2)}{2}$ 4th degree polynomials of type $\boldsymbol{a}^\top \boldsymbol{x} = 0$ in one variable $l = l_1^2$. Stacking together all these equations results in the following matrix equation:

$$\boldsymbol{A}\boldsymbol{x} = 0$$

where each row of $\boldsymbol{A}$ is one of the vectors $\boldsymbol{a}$ defined above.

From $n = 5$ on, there are sufficiently many 4th degree polynomials to linearly solve this equation up to scale, generally with the SVD of $\boldsymbol{A}$ [24].

In the special case of 4 points, the matrix $\boldsymbol{A}$ has dimension 3×5 so that Ker($\boldsymbol{A}$) has dimension 2. The SVD of $\boldsymbol{A}$ leads to a base $(\boldsymbol{v}_1, \boldsymbol{v}_2)$ of Ker($\boldsymbol{A}$), and there exists two real factors λ_1 and λ_2, so that

$$\boldsymbol{x} = \lambda_1 \boldsymbol{v}_1 + \lambda_2 \boldsymbol{v}_2$$

We now consider the nonlinear constraints among the components of $\boldsymbol{x}$. It is clear that for $i + j = k + l, 0 \leq i, j, k, l \leq 4$, we have $x_i x_j = x_k x_l$. Up to 7 such relations can be built, resulting in 7 homogeneous quadratic equations in λ_1 and λ_2 of type

$$b_1 \lambda_1^2 + b_2 \lambda_1 \lambda_2 + b_3 \lambda_2^2 = 0$$

which can be written in a matrix form as:

$$\boldsymbol{B} \begin{pmatrix} \lambda_1^2 \\ \lambda_1 \lambda_2 \\ \lambda_2^2 \end{pmatrix} = \boldsymbol{B}\boldsymbol{y} = 0$$

Again, this is solved linearly for the variables λ_1^2, $\lambda_1\lambda_2$ and λ_2^2 by SVD. This gives λ_1^2 and λ_1^2 up to scale, the scale being recovered from the equation involving $x_0 = 1$.

The final l is taken to be $l = x_1/x_0$ or $l = x_2/x_1$ or $l = x_3/x_2$ or $l = x_4/x_3$ or the average of all these values. Since $l = l_1^2$, the final length is $l_1 = \sqrt{l}$.

Two remarks are important to point out here. First, solving the equation $\boldsymbol{Ax} = 0$ may be problematic insofar as the variables $x_0, ..., x_4$ have generally extremely different weights. Indeed, they represents a length at orders of magnitude ranging from 0 for x_0 to 4 for x_4. This can cause the SVD of $\boldsymbol{A}$ to have important errors and can be solved by normalizing the matrix $\boldsymbol{A}$ before the SVD.

Second, once l_1 has been recovered, one can recover the other l_i's by using again the properties of the resultants and back substituting the found value. This makes however the whole solution extremely dependent on only the two first points. Solving for each l_i in the same manner as for l_1 will on the contrary split the errors over the points. If the computation time is not critical, this solution should therefore be considered. Note that this method involves matrices with complicated coefficients extracted from 4th degree polynomials, and its implementation is somewhat cumbersome [31].

Linear system from quadratic constraints

Ansar and Daniilidis [2] solved these equations for the variables $l_{ij} = l_i l_j$. With $\rho = 1$, equation (5.7) is rewritten as:

$$\rho d_{ij}^2 = l_{ii} + l_{jj} - 2l_{ij}\cos\theta_{ij}$$

Since $l_{ij} = l_{ji}$, this is an homogeneous linear system in the $\frac{n(n+1)}{2} + 1$ variables $\{\rho, l_{ij}, 1 \leq i \leq j \leq n\}$, which can be written as:

$$\boldsymbol{Al} = \boldsymbol{0}$$

with $\boldsymbol{l} = (l_{11}, l_{12}, ..., l_{nn}, \rho)^\top$. In this case, Ker($\boldsymbol{A}$) has dimension $n + 1$, and the solution is a linear combination of the $n + 1$ vectors $\boldsymbol{v}_i$ of the basis of Ker($\boldsymbol{A}$), easily obtained from the SVD of $\boldsymbol{A}$ [24]:

$$\begin{pmatrix} l_{11} \\ l_{12} \\ \vdots \\ l_{nn} \\ \rho \end{pmatrix} = \sum_{i=1}^{n+1} \lambda_i \boldsymbol{v}_i \tag{5.8}$$

The idea is then to solve for the λ_i by reimposing the quadratic nature of the original problem: observing that for any integers a, b, c, d and any permutation a', b', c', d' we have

$l_{ab}l_{cd} = l_{a'b'}l_{c'd'}$, we can substitute individual rows of equation (5.8) into such relations, which results in quadratic homogeneous constraints on the λ_i.

$$\sum_{p=1}^{n+1} \lambda_p^2(\boldsymbol{v}_p^{ab}\boldsymbol{v}_p^{cd} - \boldsymbol{v}_p^{a'b'}\boldsymbol{v}_p^{c'd'}) + \sum_{p=1}^{n+1}\sum_{q=p+1}^{n+1} \lambda_p\lambda_q(\boldsymbol{v}_p^{ab}\boldsymbol{v}_q^{cd} - \boldsymbol{v}_p^{a'b'}\boldsymbol{v}_q^{c'd'} + \boldsymbol{v}_q^{ab}\boldsymbol{v}_p^{cd} - \boldsymbol{v}_q^{a'b'}\boldsymbol{v}_p^{c'd'}) = 0$$

where $\boldsymbol{v}_p^{ab}$ refers to the row of $\boldsymbol{v}_p$ corresponding to the variable l_{ab} in $\boldsymbol{l}$.

Again, this system can be expressed linearly for the $\frac{(n+1)(n+2)}{2}$ new variables $\lambda_{ij} = \lambda_i\lambda_j$ as:

$$\boldsymbol{B}\lambda = \boldsymbol{0}$$

with $\lambda = (\lambda_{11}, ..., \lambda_{n+1,n+1})^\top$. In this case however, Ker($\boldsymbol{B}$) is one dimensional, and λ is recovered up to scale. The scale factor is found by imposing $\rho = 1$. Once the λ_i are known, we can compute $\boldsymbol{l}$ with (5.8) and obtain l_i as $\sqrt{l_{ii}}$.

As discussed in later sections, this algorithm gives very good results, and a good robustness to noise. However, it is computationally intensive, as it requires the building and SVD of the $\frac{n(n-1)}{2} \times (\frac{n(n+1)}{2} + 1)$ matrix $\boldsymbol{A}$ and $(\frac{n^2(n-1)}{2}) \times (\frac{(n+1)(n+2)}{2})$ matrix $\boldsymbol{B}$. It can be used even with 4 points.

5.3.4 Using weight matrices

Fiore [12] gives an interesting way to solve for the lengths l_i linearly. We first define the data matrix as:

$$\boldsymbol{D} = \begin{bmatrix} \boldsymbol{M}_1 & ... & \boldsymbol{M}_n \\ 1 & ... & 1 \end{bmatrix}$$

Assuming that $\boldsymbol{D}$ has rank 4, and $\boldsymbol{D} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^\top$ being the SVD of $\boldsymbol{D}$, we call weight matrix $\boldsymbol{W}$ the $n \times (n-4)$ matrix of the singular vectors (columns of $\boldsymbol{V}$) corresponding to the null space of $\boldsymbol{D}$. As each column of $\boldsymbol{W}$ is in the null space of $\boldsymbol{D}$ we have for all j:

$$\sum_{i=1}^{n} w_{ij}\boldsymbol{M}_i = \boldsymbol{0}$$

and:

$$\sum_{i=1}^{n} w_{ij} = 0$$

As the columns of $\boldsymbol{W}$ are orthonormal, we have also:

$$\sum_{i=1}^{n} w_{ij}^2 = 0$$

We then use linear combinations of the correspondence equations

$$l_i\frac{\boldsymbol{m}_i}{\|\boldsymbol{m}_i\|} = \boldsymbol{R}\boldsymbol{M}_i + \boldsymbol{t}$$

For the jth combination, we use the jth column of $\boldsymbol{W}$. Thus we have:

$$\begin{aligned}\sum_{i=1}^{n} w_{ij} l_i \frac{\boldsymbol{m}_i}{\|\boldsymbol{m}_i\|} &= \sum_{i=1}^{n} w_{ij}(\boldsymbol{R}\boldsymbol{M}_i + \boldsymbol{t}) \\ &= \boldsymbol{R}\sum_{i=1}^{n} w_{ij}\boldsymbol{M}_i + \boldsymbol{t}\sum_{i=1}^{n} w_{ij} \\ &= \boldsymbol{R}\boldsymbol{0} + \boldsymbol{t}.0 \\ &= \boldsymbol{0}\end{aligned}$$

This can be written in a matrix form as:

$$\boldsymbol{A}\boldsymbol{l} = \boldsymbol{0}$$

Since $\boldsymbol{A}$ have dimensions $3(n-4) \times n$, this is sufficient to solve with a classical SVD, for $n \geq 6$ [24]. However, noting that every third row of $\boldsymbol{A}$ is in fact a scaled column of $\boldsymbol{W}$ forces $\boldsymbol{l}$ to be in the left null space of $\boldsymbol{W}$, which is spanned by $\boldsymbol{D}^\top$. We therefore have that

$$\boldsymbol{l} = \boldsymbol{D}^\top \alpha$$

for an unknown 4×1 vector α and we can rewrite the equation:

$$\boldsymbol{B}\alpha = \boldsymbol{A}\boldsymbol{D}^\top \alpha = \boldsymbol{0}$$

The matrix $\boldsymbol{B}$ has $2(n-4)$ nonzero rows and 4 columns, so that a minimum of 6 points is also required with this method. The advantage is that the SVD computation is faster with the smaller matrix $\boldsymbol{B}$, especially for a large n.

If the points are coplanar (and only in this case), the rank of $\boldsymbol{D}$ will be less than 4. In this case, a similar method to solve for $n \geq 4$ points can be found in [12].

5.3.5 Iterative algorithms

Iterative algorithm for pose estimation usually define error functions reflecting the distance from the correct solution, and optimize these functions by an iterative process. We will here briefly describe the algorithms that have been tested.

Image space error

This method tends to minimize the *reprojection error* over the parameters $\boldsymbol{R}$ and $\boldsymbol{t}$, defined as the sum of the squares of the distances between an image point and its corresponding reprojected world point using $\boldsymbol{R}$ and $\boldsymbol{t}$. As shown in figure 5.4, if $\boldsymbol{m}'_i$ is the reprojected point $\boldsymbol{K}(\boldsymbol{R}\boldsymbol{M}_i + \boldsymbol{t})$ scaled so that its z-element is 1, the error is:

$$e(\boldsymbol{R}, \boldsymbol{t}) = \sum_{i=1}^{n} \|\boldsymbol{m}_i - \boldsymbol{m}'_i\|^2$$

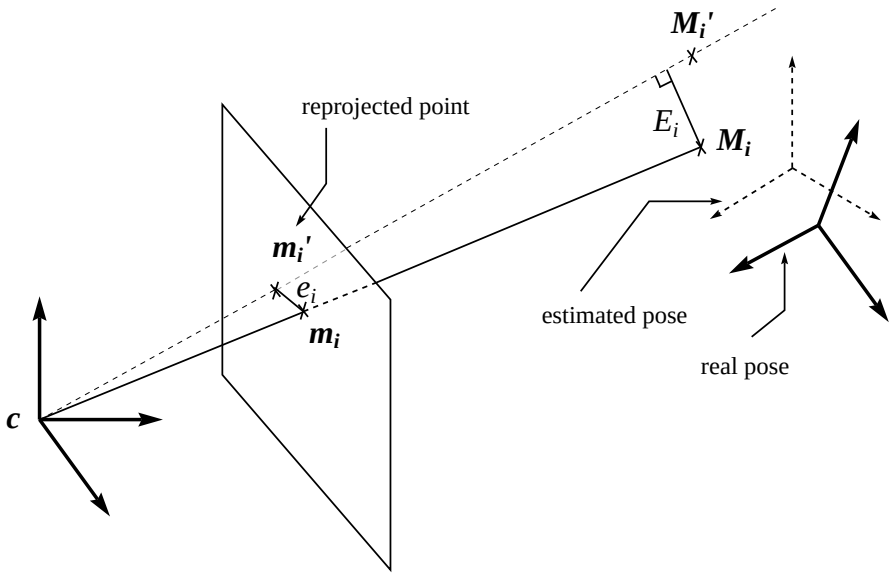


Figure 5.4: **Image error and world error** *When an estimation of the pose is known, the points may be reprojected in the image. The image distance between the original point and the reprojected point is the image error e_i. The orthogonal distance from the obtained ray and the original world point is the world error E_i.*

The rotation matrix is usually parameterized using Euler angles. Then an optimization algorithm such as the Levenberg-Macquardt method is used.

The major drawback of this method is that a good initial approximate must be given to ensure convergence. It is therefore often used for optimizing after having applied a first linear method.

Object space error

Lu, Hager and Mjolsness give in [21] an iterative algorithm which minimizes the object space error (see figure 5.4). When the image points are normalized by pre-multiplying by $\boldsymbol{K}^{-1}$, this error is:

$$E(\boldsymbol{R}, \boldsymbol{t}) = \sum_{i=1}^{n} \|(\boldsymbol{I} - \boldsymbol{V}_i)(\boldsymbol{R}\boldsymbol{M}_i + \boldsymbol{t})\|^2$$

where $\boldsymbol{V}_i$ is the observed line-of-sight projection matrix defined as:

$$\boldsymbol{V}_i = \frac{\boldsymbol{m}_i \boldsymbol{m}_i^\top}{\boldsymbol{m}_i^\top \boldsymbol{m}_i}$$

This problem is analytically similar to an absolute orientation problem, where the image points would depend on $\boldsymbol{R}$. The idea is then to solve it as the absolute orientation problem, but iteratively: at step k, $\boldsymbol{R}^{(k)}$ is used to compute the fictive image points, and the absolute orientation algorithm is applied to find $\boldsymbol{R}^{(k+1)}$. More precisions can be found in [21].

Lu et al have shown that this iterative process is globally convergent. However, convergence does not implies finding the solution but a fixed point. There is therefore again a need of good initial values for $\boldsymbol{R}$ and $\boldsymbol{t}$. These initial values can be obtained from the weak perspective model explained in next section.

POSIT

The algorithm POSIT, developed by DeMenthon and Davis use the scaled orthographic projection model [8], or *weak-perspective model*. In this model, the world points undergo a first orthographic projection along the camera axis on a plane with depth s, called *principle depth*, and then a perspective projection on the image plane. In the figure 5.5, the principle depth is chosen to be the depth Z_0 of the first point $\boldsymbol{M}_0$.

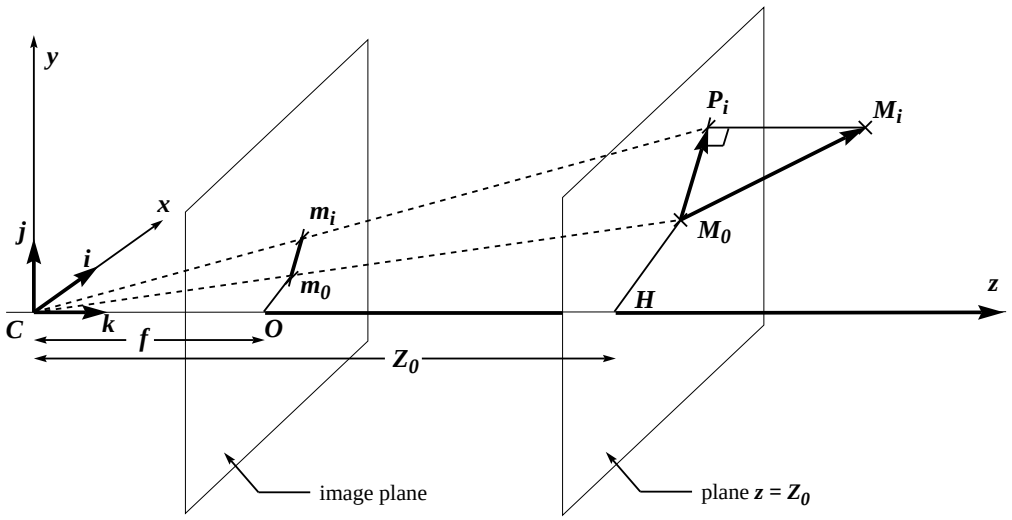


Figure 5.5: **Scaled orthographic projection model** *With a principle depth Z_0, a point $\boldsymbol{M}_i$ is first orthographically projected on the plane of equation $z = Z_0$ and then perspectively projected on the image plane.*

With the notations of the figure, one can geometrically show the following results:

$$\boldsymbol{M}_0\boldsymbol{M}_i.\boldsymbol{I} = x_i(1 + \epsilon_i) - x_0 \tag{5.9}$$

$$\boldsymbol{M}_0\boldsymbol{M}_i.\boldsymbol{J} = y_i(1+\epsilon_i) - y_0 \tag{5.10}$$

with (x_i, y_i, f) the coordinates of $\boldsymbol{m}_i$ relative to the camera center, and:

$$\boldsymbol{I} = \frac{f}{Z_0}\boldsymbol{i}, \boldsymbol{J} = \frac{f}{Z_0}\boldsymbol{j} \tag{5.11}$$

$$\epsilon_i = \frac{1}{Z_0}\boldsymbol{M}_0\boldsymbol{M}_i.\boldsymbol{k} \tag{5.12}$$

In a first algorithm, Pose from Orthography and Scaling (POS), values are given to the ϵ_i's in equations (5.9) and (5.10), resulting in linear equations in the elements of $\boldsymbol{I}$ and $\boldsymbol{J}$. Solving these equations and normalizing $\boldsymbol{I}$ and $\boldsymbol{J}$ leads to $\boldsymbol{i}$ and $\boldsymbol{j}$, which are the two first rows of the rotation matrix $\boldsymbol{R}$, and to Z_0, which in turn gives $\boldsymbol{t}$. The third row of $\boldsymbol{R}$ is obtained from $\boldsymbol{k} = \boldsymbol{i} \times \boldsymbol{j}$. When all ϵ_i's are zero, the classical weak perspective approximation is made.

The iterative algorithm POSIT (POS with ITerations) takes $\epsilon_i = 0$ as initial values and POS is applied. At each step, a pose is found, which permits to compute new ϵ_i's with equation (5.12). Iterations stop when the ϵ_i's stay unchanged.

POSIT has the advantages that it does not require an initial pose estimate, its code is very compact, and its computation time is significantly reduced in relation to other classical algorithms.

5.4 Summary

The pose of the camera can be estimated in different ways in a single view. Planar methods use the homographies between a planes and their images. The known planar method for one plane has been extended to two multiple planes algorithms, considering respectively the cases of known and unknown equations of the planes. A vanishing point pose estimation has also been detailed. The special case of known 2D/3D points correspondences, has been discussed through the derivation of 7 dedicated methods. The implementation and tests of these different algorithms are detailed in the next chapter.

Chapter 6

Implementation and results

After the theoretical approach discussed in chapters 4 and 5, this chapter covers the practical aspects of the work. The implementation and its particularities are first outlined, then results of comparative tests are discussed.

6.1 A calibration toolkit

6.1.1 Presentation

In order to test and compare the different algorithms, a software was necessary. The preexisting software *MAXCAL* allowed for viewing images, drawing points, selecting viewers and points. This application was written in *C++*, using *Trolltech's Qt* and *OpenGL* for the Graphical User Interface. In this work, this software has been upgraded to eventually become a complete calibration toolkit mainly dedicated to single views.

6.1.2 Features

An original version of *MAXCAL* was already available. This earlier application was mainly a multiple image viewer, with the possibilities to load images into separate viewers. Once a viewer has been selected, the user could augment the image with points by clicking on the image. The coordinates of the points in the image are stored in a file, along with their 3D coordinates given as an input (for correspondence algorithms). Here is an overview of what have been added to this first version. Figure 6.1 shows a window of the software.

Figure 6.1: **The calibration toolkit *MAXCAL***

Draw toolbar

The draw toolbar now allows for drawing points, lines and polygons. These objects can be easily deleted and replaced with the selection tool. Another feature is the possibility to join points or lines in a specific group. For example, parallel lines are grouped together to find the corresponding vanishing point. Figure 6.2 shows the draw toolbar.

Algorithm toolbar

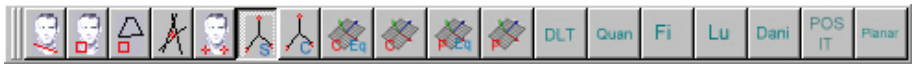
The different algorithms are represented as buttons in a toolbar. Only one algorithm can be selected at a time, and whenever an object is moved, the current algorithm is applied, enabling a first estimation of the noise robustness. Figure 6.2 shows the algorithm toolbar.

Other features

MAXCAL also includes other features like a radial distortion remover, a toolbar for the vanishing point detection method, and a mosaicing tool, which allows to merge two images with corresponding points. These features are grouped in a main toolbar, as seen in figure 6.2.

(a)

(b)



(c)

Figure 6.2: ***MAXCAL* toolbars** *(a) Main toolbar (b) Draw toolbar (c) Algorithm toolbar.*

Image augmentation

As explained in the introduction, one of the goals of the study is to augment the image with virtual objects. The work however focused on the calibration algorithms, and image augmentation was only an aside to confirm the correctness of the algorithms. That is why the augmentation mainly consisted of a wired box drawn with *OpenGL*. This box is placed on the plane of equation $z = 0$, and can be moved along the x- and y-axis with the arrow keys. Such a box is represented in figure 6.3. The coordinate system was also represented by a red, a green and a blue line, for the x-, y-, and z-axis respectively.

6.2 Implementation particularities

This section discusses the implementation particularities of the algorithms described in chapters 4 and 5. Some of the recurrent problems are outlined as well as issues specific to each algorithm.

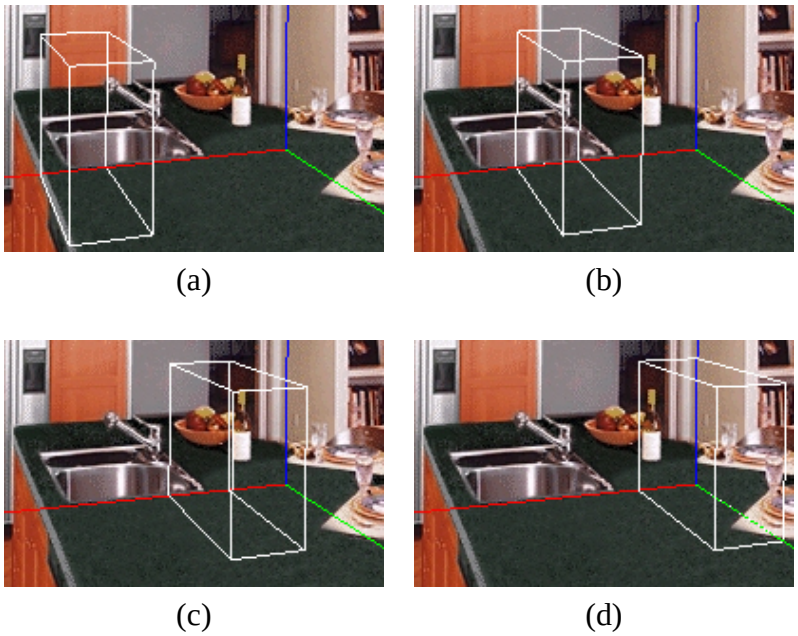


Figure 6.3: **Wired box image augmentation** *The box can be moved in the image*

6.2.1 Radial distortion

The algorithm presented in section 3.2 *p*29 has been implemented so that the user just has to draw sets of points which should be aligned in the image. Then the points corresponding to the same line are grouped together via the group button. The algorithm uses a non linear optimization technique to find the parameter k_1 and the center of radial distortion $\boldsymbol{c}$. As initial parameters, we take $k_1 = 0$ and $\boldsymbol{c}$ in the middle of the image.

While k_1 correctly converges towards its true value, the center $\boldsymbol{c}$ seems not to move. This has no effect in the most common case where the center of radial distortion actually is the center of the image. That is why this algorithm usually gives good results. However, in some cases, the center of radial distortion can lie at another place in the image. This is mostly the case when the image has been cut and recentered. The iterative optimization algorithm then stops in a local minimum with $\boldsymbol{c}$ in the middle of the image. Although this algorithm has been improved by the possibility to specify the initial position of the center of distortion, there is still no convergence of the center of distortion.

Once the parameters are recovered, the image is corrected. For each pixel of the corrected image, the distorded coordinates are computed, and the color values are taken as an interpolation of the neighbor pixels in the original distorded image. This correction requires to solve a third degree polynomial for each pixel and is quite slow.

6.2.2 Autocalibration in panoramas

To construct the panorama, the user can open several images in *MAXCAL* simultaneously, and then click on the images to give the position of the features points to be matched. Corresponding points must have the same index number on each image representing it.

The algorithms presented in section 4.3.2 *p*53 are efficient but demand a good look at the constraints imposed for a correct use. Indeed, the correct construction of a panorama, and thus the recovery of the correct homographies, depend on the relative position of the camera for each view. Theoretically, the camera must be rotating around its center. This is a hard constraint, because even if the camera is fixed to a tripod, the rotation is not necessary around the center of projection, which is not a physical point.

Another difficulty is the fact that the corresponding points on the image are drawn by the user, which can lead to incorrectness. When the images have a reduced overlapping area, the number of points is limited, and the homography can be imprecise. Figure 6.4 shows the resulting panorama of two views with reduced overlapping area.

Figure 6.4: **Imprecise panorama building** *When the overlapping area is small, the computed homography may be incorrect, and so the resulting panorama.*

Table 3.1 *p*39 shows that the image of the absolute conic ω and its dual ω^* do not play the same role in calibration when assumptions are made. However, even when the resulting number of parameters is the same for ω and ω^*, both equations (4.24 *p*54) and (4.23 *p*54) can be used in this algorithm, one being the inverse of the other. The results of tests have shown that the found matrices $\boldsymbol{K}$ are not the same when using one or the

other equation. The reason is probably the fact that the SVD solution (that is the least squared error solution) is not the same in either case. An error analysis could help to chose between the two options.

6.2.3 Numerical conditioning for homographies

The method for computing an homography from 4 or more point correspondences is detailed in appendix A.1. However, this method includes a Single Value Decomposition, and the result of the algorithm depends on the coordinate frame in which points are expressed. The usual coordinate frame is not the best for computing an accurate homography. That is why a data normalization prior to the algorithm is often made. This normalization is known as *numerical conditioning*, and has the benefits to improve the accuracy of the result and to make the algorithm invariant to arbitrary choice of scale and coordinate origin.

As a first step, the coordinates in the first image are translated so as to bring the centroid of the set of points to the origin. The coordinates are then scaled so that the average point distance to the origin is $\sqrt{2}$. This transformation for the first image is applied using a transformation matrix $\boldsymbol{T}$ (transformation plus scaling), and we note $\tilde{\boldsymbol{x}}_i = \boldsymbol{T}\boldsymbol{x}_i$ the transformed point. Similarly, the points $\boldsymbol{x}'_i$ of the second image are also transformed using the matrix $\boldsymbol{T}'$ such that the centroid of the transformed points $\tilde{\boldsymbol{x}}'_i$ is the coordinate origin and their average distance from the origin is $\sqrt{2}$.

The usual algorithm of appendix A.1 is then applied to the correspondences $\tilde{\boldsymbol{x}}_i \leftrightarrow \tilde{\boldsymbol{x}}'_i$ to find an homography $\tilde{\boldsymbol{H}}$.

The denormalization step sets then $\boldsymbol{H} = \boldsymbol{T}'^{-1}\tilde{\boldsymbol{H}}\boldsymbol{T}$. Hartley and Zisserman show in [17] that data normalization gives much better results, so that *it should not be considered optional*. A special button in *MAXCAL* enables the user to switch the numerical conditioning on or off.

6.2.4 SVD normalizing

The Singular Value Decomposition (SVD) of a matrix is well known and an explanation can be found in [24]. It is particularly useful for solving equations of the form:

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{0}$$

where $\boldsymbol{A}$ is a $m \times n$ matrix of rank $(n-1)$ and $\boldsymbol{x}$ the n-dimensional vector of the unknowns.

In this case the solution is one-dimensional, and with the SVD $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^\top$, one solution is the column of $\boldsymbol{V}$ corresponding to the smallest singular value in $\boldsymbol{S}$. In fact,

each row of $\boldsymbol{A}$ leads to one linear equation in the unknowns of $\boldsymbol{x}$, and the SVD solution is proven to be the solution that minimizes the sum of the squared errors:

$$E = \sum_{i=1}^{m} (\boldsymbol{a}^{i\top} \boldsymbol{x})^2$$

where $\boldsymbol{a}^{i\top}$ is the ith row of $\boldsymbol{A}$. According to this, one can normalize the matrix $\boldsymbol{A}$ prior to the SVD computation to improve the accuracy of the results. The two possible normalizations are

- **Row normalization** The equations of type $\boldsymbol{a}^{i\top}\boldsymbol{x} = 0$ have the same solutions if the vector $\boldsymbol{a}^i$ is multiplied by a scale factor. If the rows $\boldsymbol{a}^{i\top}$ of $\boldsymbol{A}$ are normalized so that $\|\boldsymbol{a}^i\| = 1$, the system of linear equations will remain algebraically the same, but the errors values $\boldsymbol{a}^{i\top}\boldsymbol{x}$ will be equally weighted. In this case, the SVD solution will consider each *equation* with the same importance.

- **Column normalization** If the variables in $\boldsymbol{x}$ have different orders of magnitude, the equations involving essentially small variables will generally be neglected in relation to the equation with large ones. This is generally seen in the matrix $\boldsymbol{A}$, where columns corresponding to small variables usually contains large coefficients (making the equation somewhat homogeneous). To prevent this, one can normalize the columns $\boldsymbol{a}_i$ of $\boldsymbol{A}$, so that the coefficients are homogenized. The equations system does not remain the same in this case, since we then work with new variables. The new variables actually are the old ones scaled with the scale factor used for the corresponding column normalization. The SVD solution will then consider each *unknown* with the same importance. When a solution has been found, the variables are rescaled to find the solution to the original problem.

The two normalization techniques for SVD presented here can be used separately or together. In some cases, this is an essential step to obtain a realistic solution, as seen in section 6.2.6.

6.2.5 Algorithms for calibration and pose

The implementation of the calibration and pose estimation algorithms mainly involves user interaction. The user is asked to draw points, lines and possibly to group them to indicate *e.g.* parallelism. Then a specific button launches the algorithm and the result is immediately seen as a wired box image augmentation.

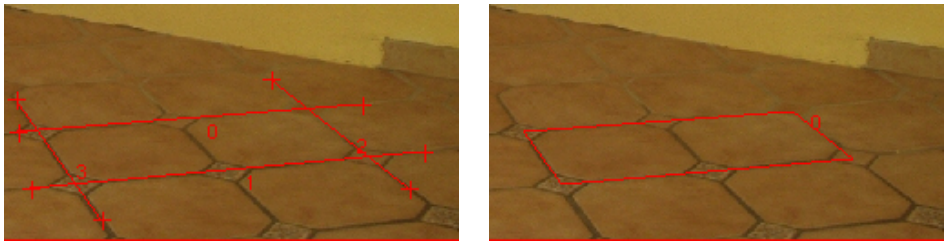
Vanishing points

To find a vanishing point, the main method is the intersection of parallel lines. The user has the to draw all the parallel lines he/she sees in the image (even if the lines are not

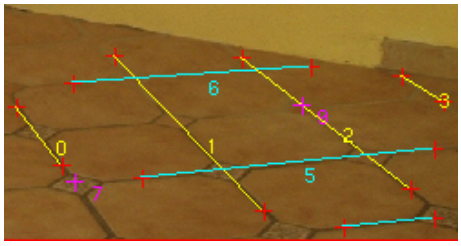
coplanar). The more lines, the better the accuracy. The parallel lines are then grouped together using the group tool. This is applied for the calibration with two or three vanishing points, as well as for the pose estimation from three orthogonal vanishing points. A specific toolbar has also been created to test the different line intersection algorithms: given a set of lines, the estimated intersection point is drawn once a button is pressed.

Drawing a rectangle

For the one-plane calibration and pose estimation algorithms, a rectangle can be drawn with lines or with a polygon. Alternatively, the user can give two perpendicular sets of parallel lines to specify the directions of the rectangle and then only draw two opposite corners of the rectangle, as seen in figure 6.5.



(a) (b)



(c)

Figure 6.5: **Drawing a rectangle** *Different methods to draw the rectangle (a) lines (b) polygon (c) groups of parallel lines and two opposite corners*

Additional information

In some cases, additional information is needed by the algorithms. For example, the methods involving a world-to-image plane homography need the points coordinates in some local coordinate frame (this is the case for the one- and multiple-plane-based algorithms). All the algorithms based on 2D/3D correspondences also need extra information. To this

aim, once the points are drawn and indexed, the coordinates of the points in a world coordinate system can be specified. This information is stored in a file and loaded with the image if already existing. This file also stores the camera calibration data, for the cases where only the pose is estimated.

For the multiple planes algorithms, the equation of the planes can be specified or not. In the former case, the coordinates of the points in the 3D world are given in a global coordinate system, and a pre-computation determines for each point its belonging plane. In the latter case, points are considered in a local coordinate frame with $z = 0$. Then only the x- and y-values are entered. To specify the belonging plane, we simply use the z-value as an integer index: points with coordinates (x, y, i) belong to the plane i, but have real local coordinates $(x, y, 0)$.

6.2.6 2D/3D pose estimation

The usage of the correspondence pose estimation algorithms is very simple and follows the process for additional information explained above. However, the implementation of some of them raised specific problems which are discussed in this section.

Resultants

The algorithm presented by Quan and Lan detailed in section 5.3.3 *p*67 builds up a linear system of equations in the variables $(1, l, l^2, l^3, l^4)$, where l is the squared distance from the camera center to the first point. These unknowns have obviously different order of magnitude, and the problem discussed in section 6.2.4 was particulary important here. Without an SVD normalization, the value for the unknown would generally not respect the quadric constraint (the found l^2 was not the square of l, and so on). Computing l in the different ways $l = l^4/l^3$, $l = l^3/l^2$, $l = l^2/l$, and $l = l/1$ gave different values with 50 to 100% error between them. With an SVD normalization of the columns, this error reduced to less than 0.1%.

In the original paper, Quan and Lan only gives the method to find the first camera-to-point distance l_1, suggesting that the same could be done for the other points. Triggs gives however an implementation where the next distances are deduced from the first one, using the already made resultants computation. In this case, the whole solution is extremely dependant on the first two points, leading to bad results. That is the reason why the tests have been made with the same method for each point.

Linear system from quadratic constraints

Ansar and Daniilidis gave another solution for the same problem, by solving linear equation in quadratic combinations of the length (see section 5.3.3 *p*68). The results showed that this solution was very robust and accurate. However, it is important to note that this algorithm has a long computation time, since it is certainly linear, but in n^2 variables, and involves the computation and SVD of $n^3 \times n^2$ matrices. It must therefore be known that this algorithm can only be used when time is not critical.

Iterative object-space-error optimization

Lu, Hager and Mjolsness presented an iterative algorithm that minimizes the object space error, as described in section 5.3.5 *p*71. They also gave a proof of global convergence for the algorithm. The major drawback is however that the suggested initial value found using the weak perspective model does not always converge towards to right solution. Instead, the pose parameters seems to converge in another local minimum. For the tests, the initial parameters where chosen as the result of less precise linear algorithms like DLT.

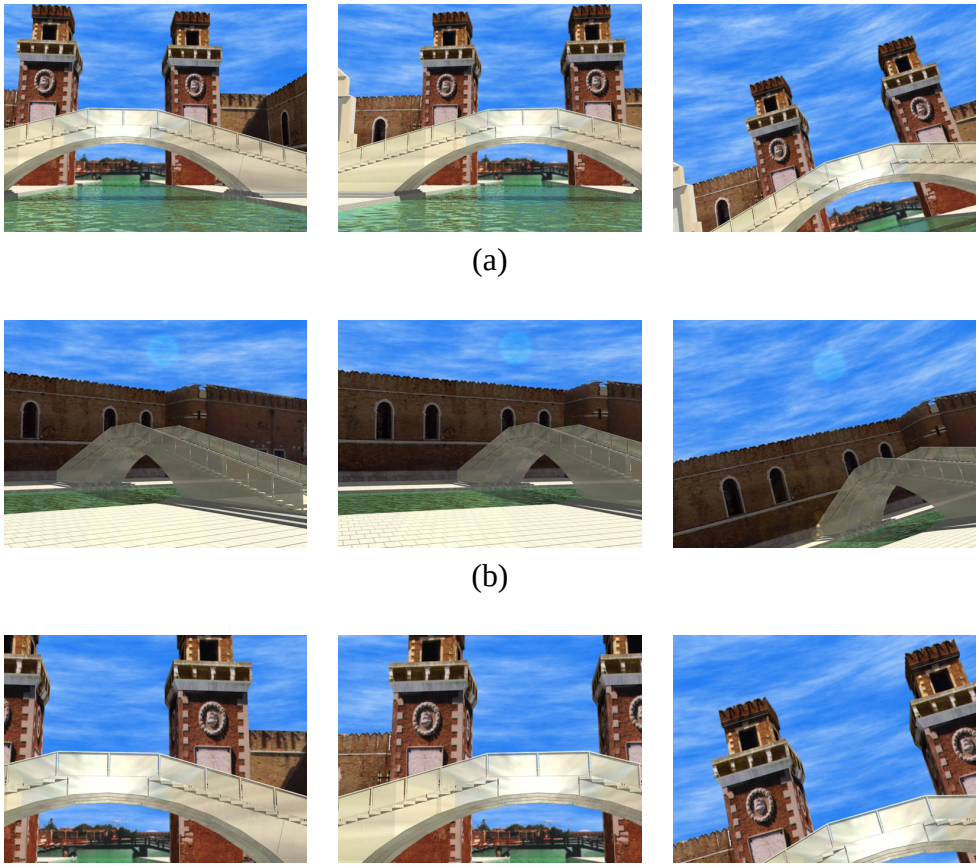
6.3 Tests and results

Once the algorithms had been implemented, several tests have been realized. For some algorithms, the immediate visible results was sufficient, but others demanded a deepest look and elaborated tests. In this section, these tests are described and their results are shown and commented.

6.3.1 Calibration from panoramas

The panorama-based calibration method was tested by constructing a panorama from synthetic images. Thus, the calibration matrix $\boldsymbol{K}$ is known in advance and can be compared to the results of the algorithms. The movements of the camera consist in pure rotation with different axis. Three sets of images have been tested: the Venice view (**VEN**), the Bridge (**BRI**) and the Towers (**TOW**), each with a different focal length. The images have the size 1024×768, so that the real principal point has coordinates $(512, 384)$. The test images are shown in figure 6.6.

In the case of panoramas, we have a set of 3 synthetic images where the view differs only by a rotation. From image-to-image points correspondences we can first reconstruct the panorama, and then use the computed homographies to apply the algorithm. As explained in section 4.3.2 *p*53, there are many ways to recover the matrix $\boldsymbol{K}$. Here four of them are considered. The two first compute the complete matrices ω, and ω^*. Another make use of the zero-skew and square pixels assumptions for the IAC, leading to a 3-parameter matrix $\tilde{\omega}$. The last one only calculates the focal length f, as it furthermore



(a) (b) (c)

Figure 6.6: **Synthetic images for the panorama-based calibration** *(a) The Venice view (b) the Bridge (c) the Towers. In each set, the different poses of the camera only differ by a rotation.*

Set	Ground Truth $\boldsymbol{K}$	$\boldsymbol{K}$ with ω	$\boldsymbol{K}$ with ω^*	$\boldsymbol{K}$ with $\tilde{\omega}$	f
VEN	$f_x = 1128.69$ $f_y = 1128.69$ $x_0 = 512$ $y_0 = 384$ $s = 0$	$f_x = 1121.65$ $f_y = 1129.96$ $x_0 = 513.39$ $y_0 = 373.41$ $s = -9.94$	$f_x = 1062.20$ $f_y = 1015.63$ $x_0 = 389.51$ $y_0 = 321.38$ $s = 79.01$	$f = 1124.20$ $x_0 = 525.84$ $y_0 = 380.57$	$f = 1110.52$
BRI	$f_x = 1228.79$ $f_y = 1228.79$ $x_0 = 512$ $y_0 = 384$ $s = 0$	$f_x = 1258.74$ $f_y = 1258.85$ $x_0 = 524.19$ $y_0 = 398.61$ $s = 47.16$	$f_x = 1403.91$ $f_y = 1491.37$ $x_0 = 735.67$ $y_0 = 422.91$ $s = 79.01$	$f = 1255.40$ $x_0 = 491.39$ $y_0 = 361.18$	$f = 1256.12$
TOW	$f_x = 1638.40$ $f_y = 1638.40$ $x_0 = 512$ $y_0 = 384$ $s = 0$	$f_x = 1637.20$ $f_y = 1623.86$ $x_0 = 512.49$ $y_0 = 348.71$ $s = -11.65$	$f_x = 1423.22$ $f_y = 1365.71$ $x_0 = 288.81$ $y_0 = 279.50$ $s = 6.2013$	$f = 1637.23$ $x_0 = 516.47$ $y_0 = 358.08$	$f = 1632.06$

Table 6.1: **Results of the panorama-calibration** *Each row represents a set of images (VEN: the Venice view, BRI: the Bridge, TOW: the Towers). The 3nd and 4rd columns presents the results for the entire matrices ω and ω^*. In column 5, $\tilde{\omega}$ assumes zero skew and squares pixels, while in column 6, the principal point is also known, and only f is computed. The 2nd column shows the correct values for* $\boldsymbol{K}$.

assumes the principal point at the center of the image.

About 30 points were manually marked on each image for the computation of the homographies. The result depends on the accuracy with which these points are marked, but gives an idea of what can be achieved by the algorithm. Results are summarized in table 6.1.

For the three sets of images, the results are better for ω than ω^*. The first method solves for a matrix equation, and the second solves for its exact inverse. This shows that in the second case, the numerical values are unstable (in particular in the computation of the SVD), and the first one should be chosen for this algorithm. The values found for ω

have very little errors. In particular the focal length is retrieved with less than 3% error. The found principal point is also very accurate, with at most 9% error. However, the values found for the skew are unrealistic. An interesting point is that when assumptions are made, the values do not change much. A good compromise seems to compute the matrix $\boldsymbol{K}$ for the entire matrix ω, and to set then $s = 0$.

Two remarks are worth noting here. First, the chosen views have a large overlapping surface, so that correspondence points are very easy to find. In an usual set of images taken for a panorama, images may have much less overlapping. Second, a lot of points correspondences are necessary to obtain correct results. When testing with only 10 points, the homography could not be correctly found and the values were very unrealistic. As a conclusion, this algorithm gives very good results if the images are taken with this aim in mind. An automatic extraction of the points of interest and a robust matching algorithm (*e.g.* RANSAC, see [13]) could be added to reduce the user interaction.

6.3.2 Calibration from vanishing points

The test of vanishing points calibration does not differ much from the plane-based calibration explained above. In this case, vanishing points are detected by selecting parallel lines in the same images. Once the vanishing points are known, one can apply the algorithm to find the focal length (2 vanishing points) and the principal point (3 orthogonal vanishing points), and compare them to the real values. Results are shown in table 6.2.

The vanishing point technique also give reasonable results with a maximal error of 5% for the focal length. Again, the error for the principal point is less than 9%. This methods proves to give good results when the principal points are easy to find. In particular, when parallel lines converge rapidly to their intersection point, the noise effect is reduced (as in the test images). In other images, this is not applicable, as no straight lines are visible.

6.3.3 Plane-based calibration

To compute the homographies, we need known planar pattern in the image. Figure 6.7 shows a real image with a known planar pattern taped up on three planes. The focal length is not known in advance, but we can compare the values obtained from the vanishing point and the planar method. 12 points were used for each plane (see table 6.3).

The comparison shows that the focal length is correct with the multiple plane method, in particular when the square pixels assumption is made. However, the principal point coordinate differs from one method to the other. Note that the image was a real photograph and may have a principal point that is not at the center of the image. Other tests showed that the focal length has much more importance than the principal point in image augmentation, so that the center of the image can be taken instead of the found values.

Set	Ground Truth $\boldsymbol{K}$	$\boldsymbol{K}$ with 3 VP	$\boldsymbol{K}$ with 2 VP
VEN	$f_x = 1128.69$ $f_y = 1128.69$ $x_0 = 512$ $y_0 = 384$ $s = 0$	$f = 1189.12$ $x_0 = 524.05$ $y_0 = 352.09$	$f = 1113.87$
BRI	$f_x = 1228.79$ $f_y = 1228.79$ $x_0 = 512$ $y_0 = 384$ $s = 0$	$f = 1237.10$ $x_0 = 498.55$ $y_0 = 381.75$	$f = 1243.75$
TOW	$f_x = 1638.40$ $f_y = 1638.40$ $x_0 = 512$ $y_0 = 384$ $s = 0$	$f = 1585.30$ $x_0 = 501.82$ $y_0 = 373.34$	$f = 1616.10$

Table 6.2: **Results of the vanishing points calibration** *The 3rd and 4th columns present the results for the cases of 3 and 2 orthogonal vanishing points respectively.*

$\boldsymbol{K}$ with 3 VP	$\boldsymbol{K}_1$ with planes	$\boldsymbol{K}_2$ with planes
$f = 848.57$ $x_0 = 326.54$ $y_0 = 227.30$	$f_x = 829.75$ $f_y = 877.50$ $x_0 = 249.94$ $y_0 = 348.77$ $s = -20.68$	$f = 841.46$ $x_0 = 254.62$ $y_0 = 329.77$

Table 6.3: **Results of the multiple plane calibration** *The multiple plane calibration without assumptions (column 2) and with zero skew and squares pixels (column 3) is compared to the vanishing points method.*

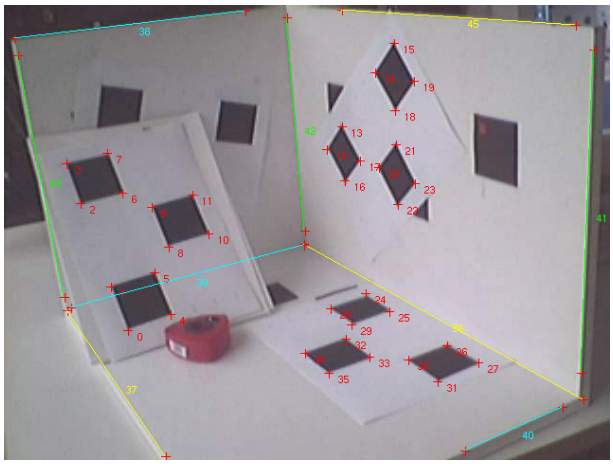


Figure 6.7: **Multiple plane calibration** *12 points define a planar homography. Parallel lines used for vanishing points detection are marked in color.*

6.3.4 One plane pose estimation

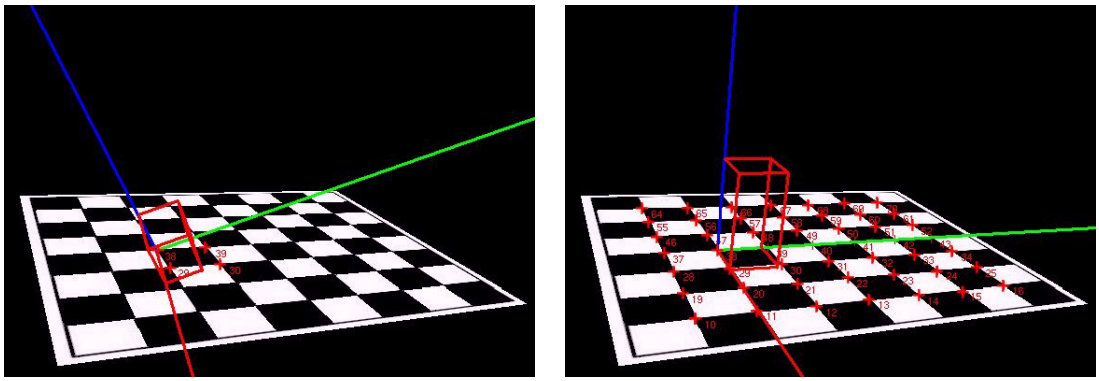
Pose estimation algorithms for one plane imply computing a plane-to-image homography. When the number of points is reduced (4 is a minimum), the homography will be very dependent on the noise of that points, and the pose will be false. Furthermore, if the reprojection error is acceptable on the considered plane, it will generally increase with the distance of the 3D point to that plane. We can fortunately go around this problem in two different ways.

Homography from more points

The homography is generally more accurate if more 2D/2D correspondences are taken into account. Tho this aim, the user can provide more points with known coordinates on the plane (see figure 6.8), or an efficient automatic feature extraction followed by a robust matching algorithm can be used. Such a robust estimator is the random sample consensus (RANSAC) from Fisher and Bolles [13].

Manual pose adaptation

In the case of pose estimation with a rectangle, when the rectangle is drawn manually, the first pose estimation is generally incorrect. The main problem is that the z-axis does not seem perpendicular to the rectangle's plane. A solution can be to manually move the corners while applying the algorithm for each new position. When the z-axis is vertical, a correct pose is found. This is useful for applications where the exact pose must not be known, and where a visually correct augmentation is needed.



(a) (b)

Figure 6.8: **Homography computation accuracy** *(a) When using only 4 points, the homography is not accurate and the resulting pose is false. (b) More points (here 49) provide a better accuracy in both homography and pose estimation.*

6.3.5 Multiple planes pose estimation

In the case of multiple planes, a minimum of 8 2D/3D points correspondences are needed to compute the homographies (a minimum of 4 for each of the homographies and at least 2 homographies). If the equations of the planes are known, the multiple plane algorithm with equations allows to find a good compromise between the solutions for each plane separately. However, this algorithm also has the problem of homography computation, which needs more than the minimum of 4 points to be accurate. Taking 3 planes and 6 points for each plane implies knowing 18 points correspondences. In this case, a simple DLT algorithm proves to give better results.

When the equations of the planes are not known, however, one can use the same plane pattern of markers for each plane. This can easily be done with a printed pattern which is attached to the planes of the scene . Then the local world coordinates of the markers are the same for each plane, and minimum information is required. Figure 6.9 shows the result of a calibration obtained in this manner.

6.3.6 Pose estimation from vanishing points

The algorithm presented in section 5.2 *p*62 has good visual results when the vanishing points are correctly computed. The major problem which can arise is the fact that 3 orthogonal vanishing points are needed. In most of the images, there is no problem to find 2 orthogonal vanishing points (using a regular tiled floor, for example), but the third vanishing point is more difficult to find. In particular, when the camera is hold parallel to

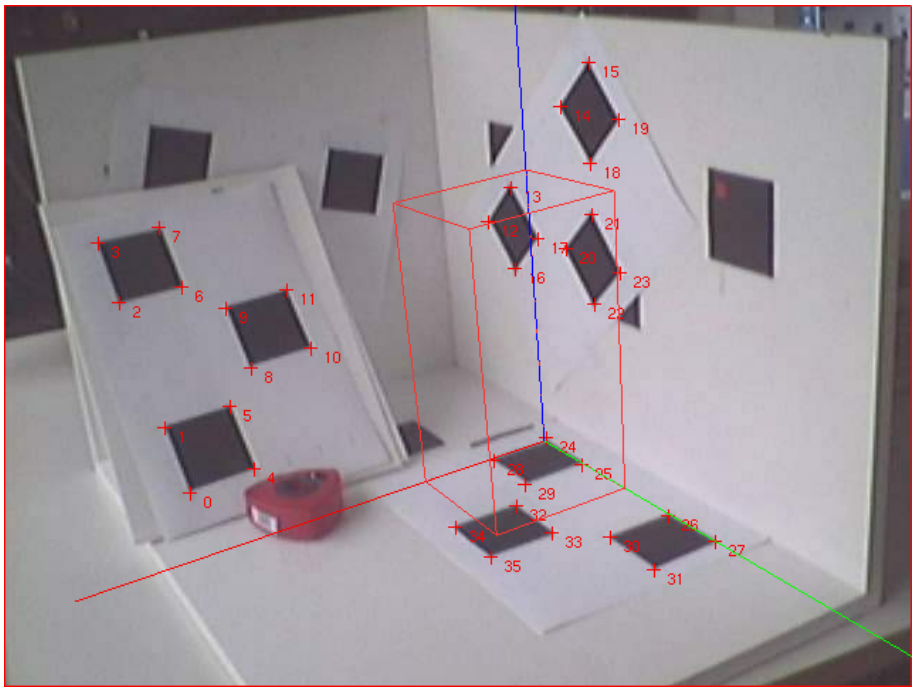


Figure 6.9: **Multiple planes pose estimation without equations of the planes**

the ground, vertical planes will be perpendicular to the camera principal axis, and vertical lines will remain parallel. That is why the user should be aware of the fact that the picture has to be taken in such a way, that no set of parallel lines is perpendicular to the principal axis. This is mostly the case when pointing the camera toward a corner of a room or a building.

The advantage of this algorithm mainly consist of the minimal information needed about the 3D world. Parallel lines just have to be recognized, and their exact position has not to be known.

6.3.7 Pose estimation from 2D/3D correspondences

In the chapter 5, seven algorithms for pose estimation based on correspondences between world points and imaged points have been derived. Insofar as they all take the same input (a number of correspondences), it is interesting to see how accurate is the result and in which cases some algorithms are better than others. To this aim, Monte-Carlo simulations have been conducted as following.

The calibration and pose of a real image are used. In this case, the values were:

$$\boldsymbol{R} = \begin{bmatrix} -0.5931 & 0.8049 & -0.0167 \\ 0.2190 & 0.1413 & -0.9654 \\ -0.7747 & -0.5763 & -0.2601 \end{bmatrix}$$

$$\boldsymbol{t} = \begin{pmatrix} -14.1343 & 10.1104 & 114.8236 \end{pmatrix}^\top$$

and:

$$\boldsymbol{K} = \begin{bmatrix} 796.099 & 0 & 421.584 \\ 0 & 796.099 & 318.655 \\ 0 & 0 & 1 \end{bmatrix}$$

Then, the positions of n points $\boldsymbol{M}_i$ in the 3D world are given arbitrarily, so that they are seen in the image. From these positions, the imaged points are computed using $\boldsymbol{K}$, $\boldsymbol{R}$ and $\boldsymbol{t}$.

$$\boldsymbol{m}_i \sim \boldsymbol{K}(\boldsymbol{R}\boldsymbol{M}_i + \boldsymbol{t})$$

We then have the perfect correspondences $\boldsymbol{m}_i \leftrightarrow \boldsymbol{M}_i$. We then make 100 trials with each imaged point subject to a constant Gaussian noise with known standard deviation. Each time we consider the found pose for each algorithm $\tilde{\boldsymbol{R}}$ and $\tilde{\boldsymbol{t}}$, and measure the error as $Fnorm(\tilde{\boldsymbol{R}}^\top \boldsymbol{R} - \boldsymbol{I})$ for the rotation, where *Fnorm* is the Frobenius norm, and $\|\tilde{\boldsymbol{t}} - \boldsymbol{t}\|$ for the translation. The points are also reprojected using the found pose and the reprojection error is measured. After 100 trials we take the mean error for each algorithm and a constant Gaussian noise. We repeat this process for Gaussian noises ranging from 0 to 10 pixels, with a 0.5 pixel step. We can then plot the function giving the mean error against noise level for each algorithm, and thus efficiently compare the different algorithms. The points are placed in a general configuration. Note that since the configuration of the points is fixed, this only allows for a comparison between algorithms and not for an overall evaluation of a particular algorithm, which would involve tests for many random positions of points.

The results are of particular interest for a limited number of points, since a large number of points can be handled by an usual DLT method. The results for $n = 4, 5$ and 6 points are shown in figures 6.10 to 6.13.

Figure 6.10 shows the errors for an increasing noise level in the case of a first set of 4 non coplanar points. The first remark is the inferiority of the algorithm of Quan and Lan (**Quan-Lan**) relative to the others. In fact, this algorithm presented serious numerical problems discussed in section 6.2.6. Even if the method with an entire computation for each length has been implemented, the numerical unstability was not completely removed.

The two iterative algorithms (Lu, Hager and Mjolsness (**Lu-Hager**) and Non Linear Least Squares (**NLLS**)) give exactly the same results, assumed to be the best reachable

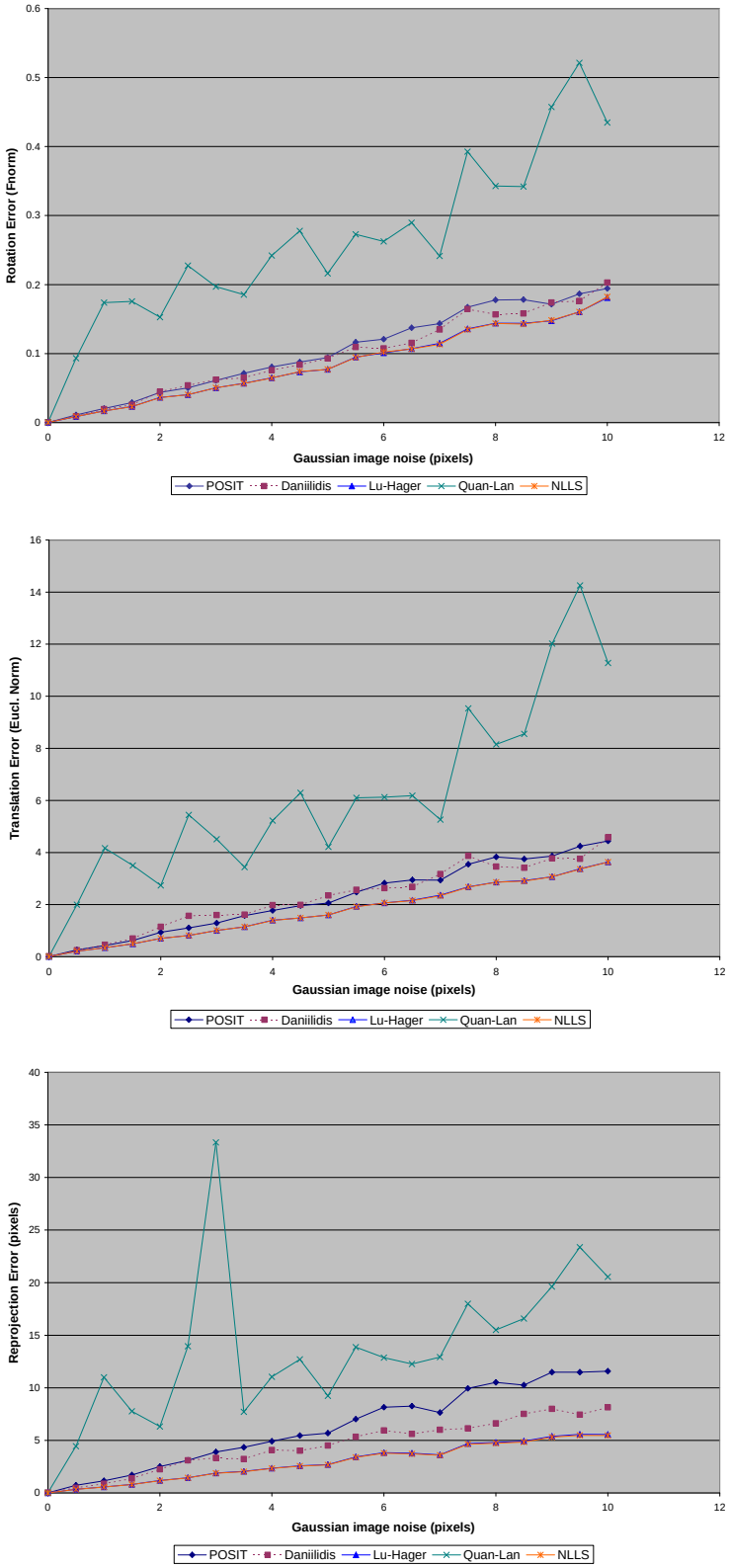


Figure 6.10: **Pose estimation errors for 4 points correspondences (a)** *Rotation error (top), translation error (middle) and reprojection error (bottom)*

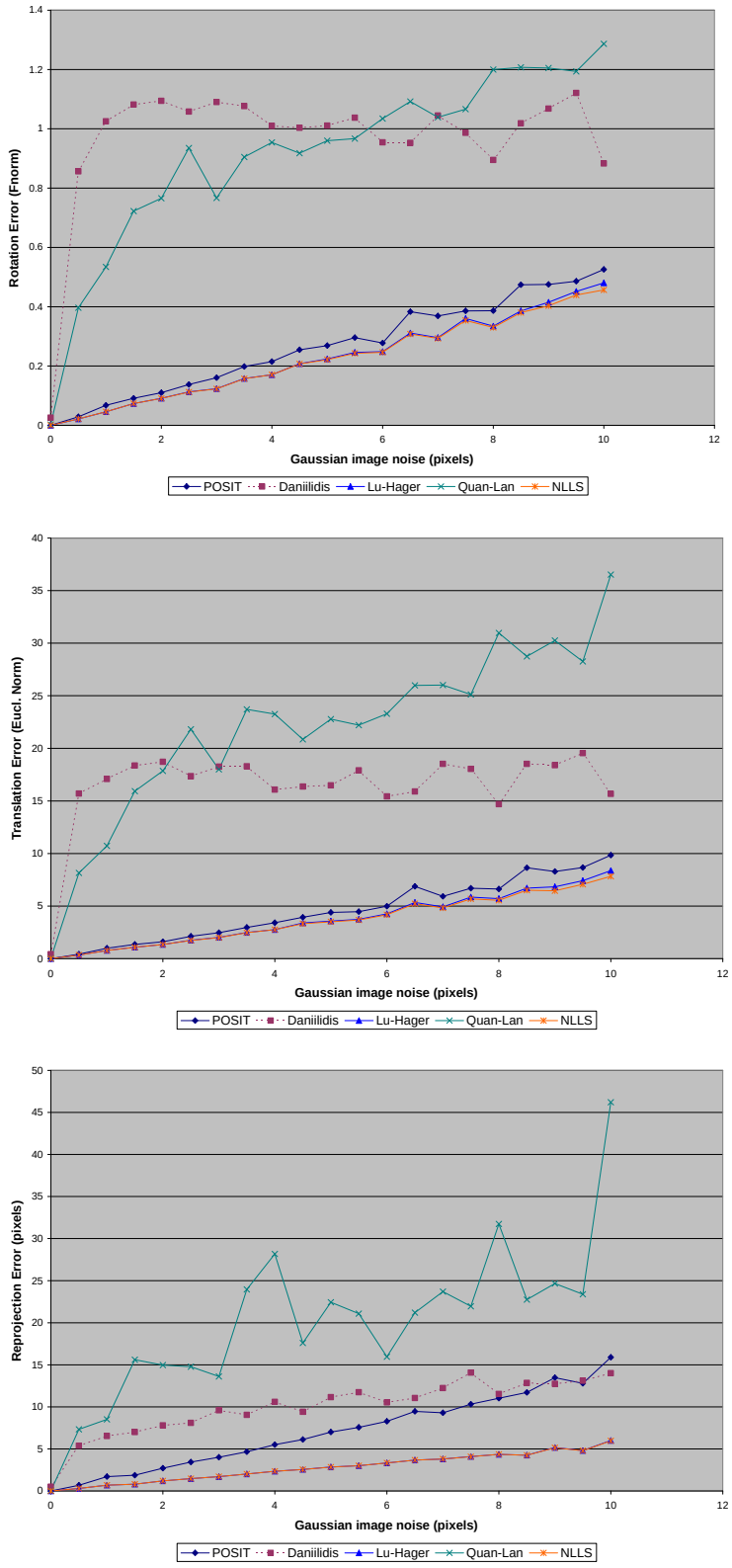


Figure 6.11: **Pose estimation errors for 4 points correspondences (b)** *Rotation error (top), translation error (middle) and reprojection error (bottom)*

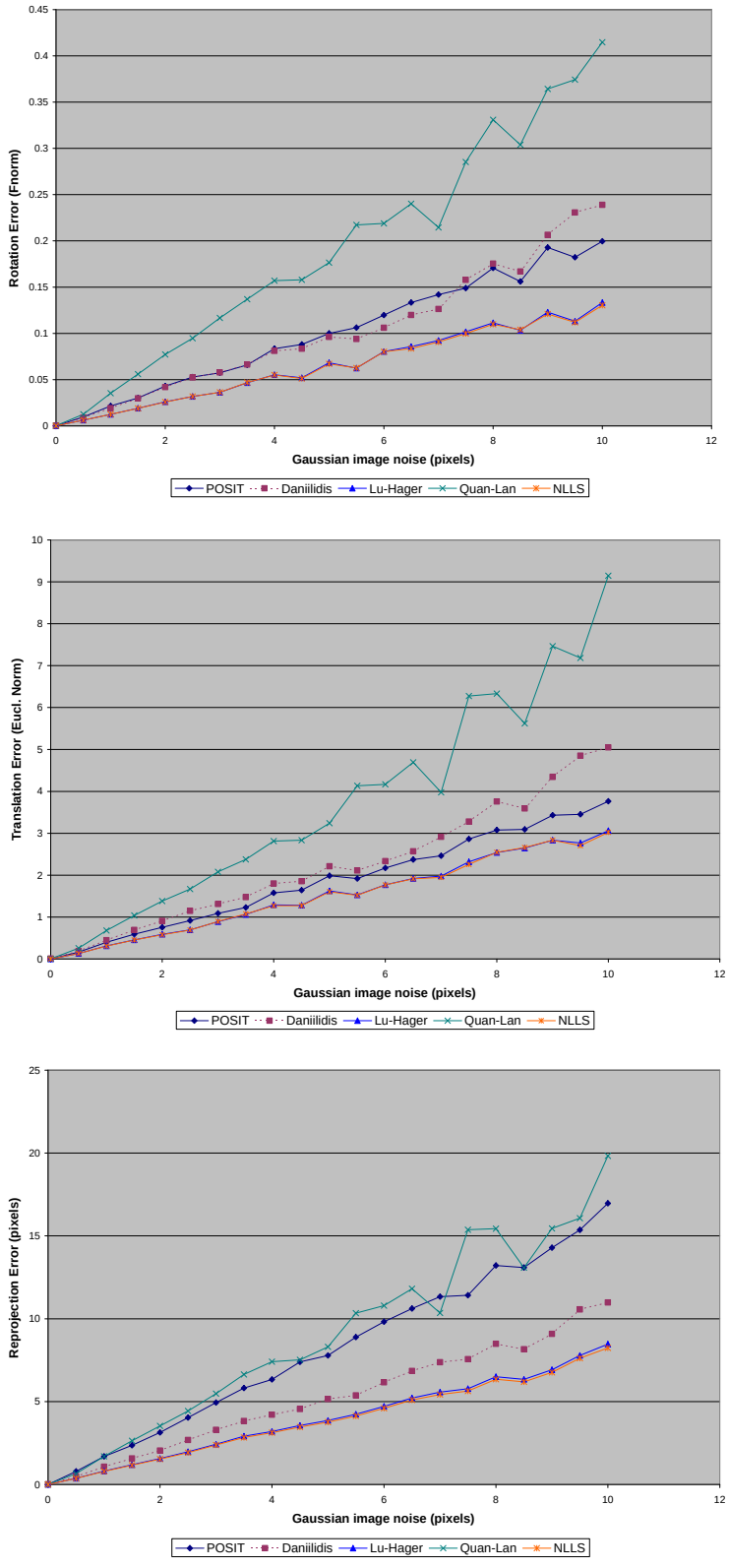


Figure 6.12: **Pose estimation errors for 5 points correspondences** *Rotation error (top), translation error (middle) and reprojection error (bottom)*

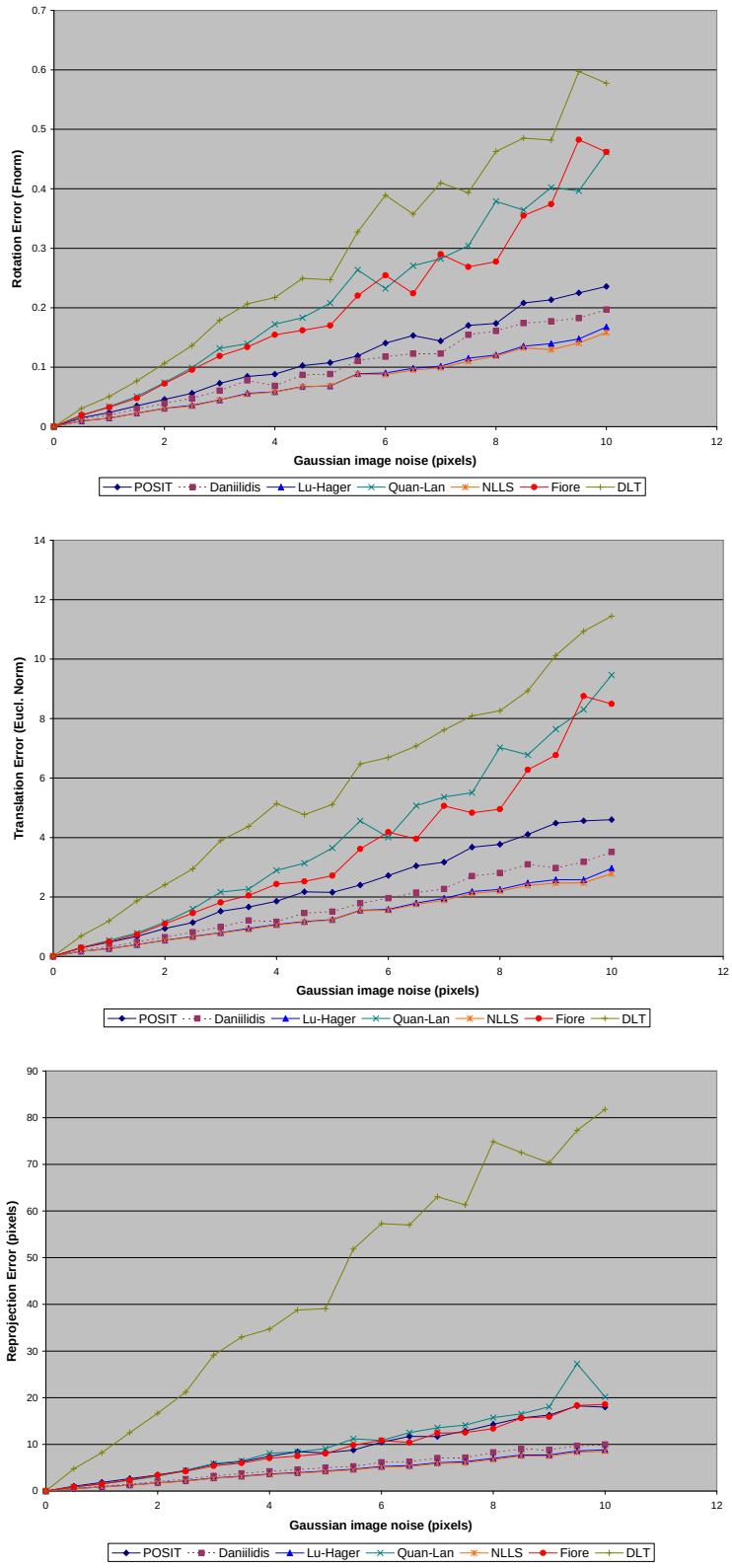


Figure 6.13: **Pose estimation errors for 6 points correspondences** *Rotation error (top), translation error (middle and reprojection error (bottom)*

solution, and given as a reference for the linear algorithms. It is interesting to see that the algorithm of DeMenthon and Davis (**POSIT**) and the one from Ansar and Daniilidis (**Daniilidis**) have similar results, and are very close to the best iterative solution. (Note that POSIT is also an iterative algorithm, but its convergence is very fast and it can be compared to the linear algorithms). In terms of reprojection error, Daniilidis give better results.

In figure 6.11, another set of 4 points is considered. While Quan-Lan remains unnstable, POSIT still gives good results. Daniilidis however rapidly diverges. This example shows the dependence of Daniilidis to the configuration of points, especially in the minimal case of 4 points.

The case of 5 correspondences is studied in figure 6.12. Again Daniilidis and POSIT have similar results for the rotation and translation errors. For the reprojection error, Daniilidis has better results, similar to the interative algorithms.

From 6 points on, more algorithms can be applied. in addition to the precedent ones, the algorithm of Fiore (**Fiore**) and the classical direct linear transform (**DLT**) are plotted. We first see that DLT has particular bad results. In this algorithm, the orthogonality of the rotation matrix is not enforced, and in the tests, a further step replaced the found matrix $\boldsymbol{R}$ by the nearest orthogonal one in the sense of the Frobenius norm, which explains the large reprojection error. We can also see that Quan-Lan has now more stability, with results comparable to Fiore. In this case again, Daniilidis have excellent results, especially in the reprojection error.

All the algorithms tested here have not the same computation time. Although Daniilidis is said linear, it has the longest computation time, so that it can not be used for real time applications. Quan-Lan, Fiore and POSIT are on the contrary fast enough for real time. In an augmented reality application, the reprojection error has the more importance, as virtual objects are reprojected in the real scene. These three algorithms have comparable results in terms of reprojection. As a conclusion, when the time is crucial, POSIT is recommended as a fast and exact algorithm, in terms of reprojection error as well as pose accuracy. When time is no matter, Daniilidis is indicated as it gives the best results, except when only 4 points are detected, in which case POSIT provides more stability. Note however that POSIT is not globally convergent, as it is based on a weak perspective model. When the points are placed on the periphery of the image, POSIT can fail, and Fiore or Quan-Lan should be used.

6.4 Summary

All the algorithms presented in this report have been implemented into the calibration toolkit *MAXCAL* . The principal implementation specificities have been discussed, and the algorithms evaluation with comparative tests has been detailed. In particular, the correspondences-based pose estimation is handled by algorithms with different time cost and efficiency and the choice of the indicated algorithm has been thoroughly discussed. In general, the different presented methods have their own advantages, and a good comprehension of the way they operate is helpful for choosing the right algorithm for the right case.

Chapter 7

Conclusion

In this report, techniques for integration of virtual objects into a single view have been presented. The case of single views is flexible for augmented reality, insofar as it does not require several cameras nor to move an unique camera. To correctly insert objects into a single image, it is important to first study the geometrical properties of the image, and then to use them correctly in a number of tasks, namely the camera internal calibration, the pose estimation and the image augmentation itself.

Calibration and pose algorithms for single views are essentially based on geometric particularities of the image. Thus, the first part of this work consists in an extensive analysis of the single view geometry. Four geometrical levels and corresponding transformations have been studied in detail: Projective geometry, affine geometry, metric geometry and Euclidean geometry. Each level is also defined as a set of invariant properties and invariant geometrical objects. These properties and objects are explicitly used in many algorithms presented throughout this report.

A brief overview of the camera model has defined the two main tasks of objects integration. First, the camera has to be calibrated. The problem is the recovery of the internal parameters of the camera (mainly the focal length and the principal point of the camera), which build the camera calibration matrix $\boldsymbol{K}$. Second, the pose of the camera relative to a world coordinate system must be estimated. This pose decomposes into a rotation matrix $\boldsymbol{R}$ and a translation vector $\boldsymbol{t}$. Real images can however be subject to radial distortion, in which case they will not fit the model. A possible radial distortion can be first removed using a correction algorithm, also provided in the report.

Single view camera calibration presented in chapter 4 mainly consists in three approaches: a plane-based calibration, calibration from vanishing points and the special case of panoramas. An algorithm for camera calibration based on planes has been detailed. This algorithm requires the computation of an homography between each plane

and its image. Any number of planes can be used, and the accuracy of the results generally increases with the number of planes. A geometrical interpretation of this algorithm has also been described. Calibration using vanishing points has been studied, with two algorithms for the calibration itself and five methods for the recovery of a vanishing point in an image. The geometry of panoramic images and their construction have then been explained, together with an algorithm for calibration of panoramas.

All the algorithms mentioned in this report have been implemented and evaluated. The plane-based calibration gives good results if the homographies can be correctly computed. To this aim, calibrations objects are inevitable in most of the cases (planar patterns with known configuration for example). This approach is thus less flexible than the other ones. In this sense, camera calibration with a panorama is easier. It also has the advantage to provide a larger image, that can be practical for a further image processing. Nevertheless, the images have to be taken with camera rotations only and no translations. Panorama calibration can then not be used for pre-existing images or for only one strict single view. Vanishing points are in the contrary easy to find in single images with parallel lines (like interiors or buildings), and can be retrieved with a good accuracy, even (and maybe especially) on paintings. In the case of an interior scene augmentation with virtual pieces of furniture, the vanishing point calibration method would be recommended as the easiest calibration technique.

The estimation of the pose of the camera in single views has been treated in chapter 5. This estimation can be performed directly with correspondences between known 3D points and their 2D image, with planes or with vanishing points. In the case of planes, an existing algorithm using only one plane has been briefly explained. It has then been extended to a novel approach for the case where several planes are visible in the image. This approach handles the situations of known and unknown equations of the planes with two separate methods. Pose estimation with vanishing points has been discussed with the presentation of one algorithm. The problem of pose estimation from the 3D coordinates of a number of points is well known, and several solution have been suggested in the computer vision community. Among them, seven recent methods have been presented in this report. Their implementation has been discussed in detail. Their robustness to noise have been compared using Monte Carlo simulations and has been commented.

If the exact position of a number of points is known in the scene, the direct correspondences algorithms are the best method in terms of accuracy. Among them, the linear algorithms seem to give results that are good enough for a acceptable visual augmentation, and have a low computation time. They are thus indicated for real-time application like video scenes augmentation. When the time is not crucial, an additional non-linear optimization algorithm can be performed to refine the results. Such iterative algorithms improve the quality of the result. When the user can prepare the scene, however, a practical method

is the plane-based estimation. In this case, regular planar patterns can be simply taped up on planar structures, and the multiple plane pose estimation without equations of planes gives immediate results, without any knowledge of the scene other than planar structures. When dealing with images of unknown scenes, the only available visible information is often parallel sets of lines. The vanishing point method is then indicated, and can be easily combined with the calibration of the camera.

Image augmentation has been made mainly with a wired box in the tests for calibration and pose algorithms. An improvement of this work could include the choice of the object to be added in the seen, as well as a 3D reconstruction of the image to allow occlusion handling and changes in the viewing position and orientation.

Most of the algorithms presented here require the intervention of the user. Some of them could however be automatized. It is for example possible to automatically detect the vanishing points using a Hough transform approach [22]. For panoramas, Torr and Zisserman showed that homographies between images can be automatically estimated [29]. This can be done using a robust correspondence estimation between interest points like RANSAC [13].

As explained above, there is no best algorithm either for calibration and pose estimation. Consequently, one has to choose the best compromise between available information, speed and efficiency when augmenting a single view. An improvement of this work could thus be to write a more general algorithm which would analyze the information contained in the image (parallel lines, planes, other objects) and automatically choose the algorithm that is adapted to the situation. Such an algorithm could reduce the user interaction and thus make the integration of virtual objects into a single view even easier.

Appendix A

Numerical methods

A.1 Planar homography from four points

The fact that a planar homography can be computed from a set of four 2D to 2D point correspondences $\boldsymbol{x}_i \leftrightarrow \boldsymbol{x}'_i$ is well known. Here follows a derivation of this result, inspired from [17].

The transformation is given by the equation

$$\boldsymbol{x}' \sim \boldsymbol{H}\boldsymbol{x}$$

where $\boldsymbol{H}$ is a 3×3 matrix. This homogeneous equation may be expressed in terms of the vector cross product as $\boldsymbol{x}' \times \boldsymbol{H}\boldsymbol{x} = \boldsymbol{0}$. This form will enable a simple linear solution for $\boldsymbol{H}$ to be derived. Writing $\boldsymbol{h}_i^\top$ the ith row of $\boldsymbol{H}$ and $\boldsymbol{x}' = (x', y', w')$, the cross product may be given explicitly as

$$\boldsymbol{x}' \times \boldsymbol{H}\boldsymbol{x} = \begin{pmatrix} y'\boldsymbol{h}_3^\top \boldsymbol{x} - w'\boldsymbol{h}_2^\top \boldsymbol{x} \\ w'\boldsymbol{h}_1^\top \boldsymbol{x} - x'\boldsymbol{h}_3^\top \boldsymbol{x} \\ x'\boldsymbol{h}_2^\top \boldsymbol{x} - y'\boldsymbol{h}_1^\top \boldsymbol{x} \end{pmatrix}$$

Since $\boldsymbol{h}_i^\top \boldsymbol{x} = \boldsymbol{x}^\top \boldsymbol{h}_i$, this gives a set of three equations in the entries of $\boldsymbol{H}$, which may be written in the form

$$\begin{bmatrix} \boldsymbol{0}^\top & -w'\boldsymbol{x}^\top & y'\boldsymbol{x}^\top \\ w'\boldsymbol{x}^\top & \boldsymbol{0}^\top & -x'\boldsymbol{x}^\top \\ -y'\boldsymbol{x}^\top & x'\boldsymbol{x}^\top & \boldsymbol{0}^\top \end{bmatrix} \begin{pmatrix} \boldsymbol{h}_1 \\ \boldsymbol{h}_2 \\ \boldsymbol{h}_3 \end{pmatrix} = \boldsymbol{0} \tag{A.1}$$

The three equations of (A.1) are linearly independent, so that only the first two equations can be used:

$$\begin{bmatrix} \boldsymbol{0}^\top & -w'\boldsymbol{x}^\top & y'\boldsymbol{x}^\top \\ w'\boldsymbol{x}^\top & \boldsymbol{0}^\top & -x'\boldsymbol{x}^\top \end{bmatrix} \boldsymbol{h} = \boldsymbol{0} \tag{A.2}$$

where $\boldsymbol{h} = (\boldsymbol{h}_1^\top, \boldsymbol{h}_2^\top, \boldsymbol{h}_3^\top)^\top$ is the vector made of the entries of $\boldsymbol{H}$. When n points correspondences are considered, the n equations of type (A.2) are stacked in one matrix equation

$$\boldsymbol{A}\boldsymbol{h} = \boldsymbol{0} \tag{A.3}$$

where $\boldsymbol{A}$ is a $2n \times 9$ matrix.

This shows that with a minimum of 4 points correspondences, the matrix $\boldsymbol{A}$ has the size 8×9 and a rank 8 in general case. Equation (A.3) has then a one-dimensional solution space, which is sufficient to find the matrix $\boldsymbol{H}$, defined up to a scale factor. The solving method is usually a single value decomposition (SVD) of $\boldsymbol{A}$.

A.2 Cardan's Method

Polynomial equations of degree 3 can be directly solved using a method first derived by Cardan as early as 1545.
We consider a polynomial equation

$$ax^3 + bx^2 + cx + d = 0$$

where $a \neq 0$. Dividing the equation by a will not change the solution, so that we can assume that $a = 1$ without loss of generality. We first note that this equation can be reduced to a particular form using the simple variable change $y = x + \frac{b}{3}$ to get

$$y^3 + \alpha y + \beta = 0 \tag{A.4}$$

where $\alpha = c - \frac{b^2}{3}$, $\beta = d - \frac{bc}{3} + \frac{2b^3}{27}$. We will then consider that the equation to solve has the form of (A.4) (note that the equation 3.12 *p*31 obtained for the problem of radial distortion already has this from).

Now, let $y = u + v$. Equation (A.4) becomes

$$(u+v)^3 + \alpha(u+v) + \beta = 0$$

that is

$$u^3 + 3u^2v + 3uv^2 + v^3 + \alpha(u+v) + \beta = 0$$

which can be rewritten as

$$(u^3 + v^3 + q) + (u+v)(3uv + \alpha) = 0.$$

Thus provided we can find u and v that satisfy the equations:

$$\begin{aligned} u^3 + v^3 + \beta &= 0 \\ 3uv + \alpha &= 0 \end{aligned} \tag{A.5}$$

then $y = u + v$ satisfies equation (A.4).

We thus search for two variables u^3 and v^3, whose sum is $-\beta$ and whose product is $-\frac{\alpha^3}{27}$. These are the solution of the quadric equation in z:

$$z^2 + \beta z - \frac{\alpha^3}{27} = 0 \tag{A.6}$$

whose discriminant is $\delta = \beta^2 + 4\frac{\alpha^3}{27}$. Since we will consider the sign of this determinant, we will note $Q = \frac{\alpha}{3}$ and $R = -\frac{\beta}{2}$ and use the discriminant $\Delta = Q^3 + R^2$ which has the same sign. The solutions then depends on the sign of Δ:

- **Case 1:** $\Delta \geq 0$**.** Equation (A.6) has two real solutions $z_2^1 = \frac{\beta}{2} \pm \sqrt{\Delta}$. The only real solution of equation (A.4) is $\sqrt[3]{z_1} + \sqrt[3]{z_2}$, that is

$$\sqrt[3]{R + \sqrt{\Delta}} - \frac{Q}{\sqrt[3]{R + \sqrt{\Delta}}}$$

- **Case 2:** $\Delta < 0$**.** Equation (A.6) has two complex solutions $z_2^1 = \frac{\beta}{2} \pm i\sqrt{-\Delta}$. z_1 and z_2 have each three complex cubic roots, but their product must be real, so that there is only three real solutions to equation (A.4)

$$\begin{gathered} -S\cos T + S\sqrt{3}\sin T \\ -S\cos T - S\sqrt{3}\sin T \\ S\cos T \end{gathered}$$

where $S = \sqrt[3]{\sqrt{R^2 - \Delta}}$ and $T = \frac{1}{3}\arctan\frac{\sqrt{-\Delta}}{R}$.

Note that in the case of the radial distortion problem, the solution must be positive and the continuity at $k_1 = 0$ gives the only solution:

$$-S\cos T - S\sqrt{3}\sin T$$

List of Figures

List of Tables

Bibliography

[1] Y. I. Abdel-Aziz and H. M. Karara. Direct linear transformation from comparator coordinates into object space coordinates in close-range photogrammetry. In *Proc. Symposium on Close-Range Photogrammetry*, pages 1–18, 1971.

[2] A. Ansar and K. Daniilidis. Linear pose estimation from points or lines. In *Proc. ECCV*, volume 4, pages 282–296, 2002.

[3] B. Caprile and V. Torre. Using vanishing points for camera calibration. *Int. Journal of Computer Vision*, 4:127–139, 1990.

[4] R. Cipolla, T.Drummond, and D.Robertson. Camera calibration from vanishing points in images of architectural scenes. In *Proc. British Machine Vision Conference*, 1999.

[5] A. Criminisi. *Accurate Visual Metrology from Single and Multiple Uncalibrated Images*. Springer-Verlag, 2001.

[6] A. Criminisi, I. Reid, and A. Zisserman. Single view metrology. *Int. Journal of Computer Vision*, 40(2):123–148, 2000.

[7] D. F. DeMenthon and L. S. Davis. Exact and approximate solutions of the perspective-three-point problem. *IEEE Trans. PAMI*, 14(11):1100–1105, 1992.

[8] D. F. Dementhon and L. S. Davis. Model-based objects pose in 25 lines of code. *Int. Journal of Computer Vision*, 15(1):123–141, 1995.

[9] F. Devernay and O. D. Faugeras. Straight lines have to be straight. *Machine Vision and Applications*, 13:14–24, 2001.

[10] O. D. Faugeras. *Three-Dimensional Computer Vision: a Geometric Viewpoint*. MIT Press, 1993.

[11] O. D. Faugeras, Q. Long, and S. J. Maybank. Camera self-calibration: Theory and experiments. In Springer-Verlag, editor, *Proc. ECCV*, volume LNCS 588, pages 563–578, 1992.

[12] P. D. Fiore. Efficient linear solution of exterior orientation. *IEEE Trans. PAMI*, 23(2):140–148, February 2001.

[13] M. A. Fishler and R. C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Comm. Assoc. Comp. Mach.*, 24(6):381–395, 1981.

[14] E. Grossmann, D. Ortin, and J. Santos-Victor. Single and multi-view reconstruction of structured scenes. In *Proc. Asian Conference on Computer Vision*, 2002.

[15] R. M. Haralick, C. Lee, K. Ottenberg, and M. Nölle. Analysis and solutions of the three point perspective pose estimation problem. In *Proc. Conf. on CVPR*, pages 592–598, 1991.

[16] R. Hartley, L. de Agapito, and E. Hayman. Linear calibration of a rotating and zooming camera. In *Proc. Conf. CVPR*, pages 15–21, 1999.

[17] R. Hartley and A. Zisserman. *Multiple View Geometry*. Cambridge University Press, 2000.

[18] R.I. Hartley. Self-calibration from multiple views with a rotating camera. In Springer-Verlag, editor, *Proc. ECCV*, volume LNCS 800/801, pages 471–478, 1994.

[19] B. K. P. Horn, H. M. Hilden, and S. Negahdaripour. Closed form solution of absolute orientation using orthonormal matrices. *Journal of the Optical Society*, 5(7):1127–1135, July 1988.

[20] D. Liebowitz and A. Zisserman. Metric rectification for perspective images of planes. In *Proc. Conf. CVPR*, pages 482–488, jun 1998.

[21] C. P. Lu, G. D. Hager, and E. Mjolsness. Fast and globally convergent pose estimation from video images. *IEEE Trans. PAMI*, 22(6):610–622, 2000.

[22] E. Lutton, H. Maitre, and J. Lopez-Krahe. Contribution to the determination of vanishing points using hough transform. *IEEE Trans. PAMI*, 16:430–438, 1994.

[23] M. Pollefeys. Tutorial on 3d modeling from images. In *Proc. ECCV*, 2000.

[24] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical recipes in C: the art of scientifc computing*. Cambridge University Press, second ed., 1992.

[25] L. Quan and Z. Lan. Linear n-point camera pose determination. *IEEE Trans. PAMI*, 21:774–780, 1999.

[26] G. Simon, A. W. Fitzgibbon, and A. Zisserman. Markerless tracking using planar structures in the scene. In *Proc. ISAR*, 2000.

[27] P. Sturm. Algorithms for plane-based pose estimation. In *Proc. Conf. CVPR*, pages 706–711, 2000.

[28] P. Sturm and S. Maybank. On plane-based camera calibration: A general algorithm, singularities, applications. In *Proc. Conf. CVPR*, pages 432–437, 1999.

[29] P. H. S. Torr and A. Zisserman. Robust computation and parametrization of multiple view relations. In *Proc. ICCV*, pages 727–732, 1998.

[30] B. Triggs. Autocalibration from planar scenes. In *Proc. ECCV*, 1998.

[31] B. Triggs, M.-A. Ameller, and L. Quan. Camera pose revisited - new linear algorithms. *Internal Report - Equipe MOVI - Inrialpes*, 2000.

[32] W. Wunderlich. Rechnerische rekonstruktion eines ebenen objekts aus zwei photographien. *Mittsilungen Geodät. Inst. TU Gras*, 40:365–377, 1982.

[33] Z. Zhang. A flexible new technique for camera calibration. *IEEE Trans. PAMI*, pages 1330–1334, 2000.

[34] A. Zisserman, D. Liebowitz, and M. Armstrong. Resolving ambiguities in autocalibration. *Phil. Trans. R. Soc. Lond.*, 356:1193–1211, 1998.