Lecture Notes on Computer Vision

Henrik Aanæs
DTU informatics
e-mail: haa@imm.dtu.dk

November 27, 2009

Contents

Preface

In it's present form this text is intended for the DTU course 02501. In this regard many of the section titles have been marked with an "*". This indicates that it is not part of the curriculum. The material in these sections have been included to give the interested reader a more complete picture of the curriculum, and because experience has shown that this is the material that students often ask about later on in their studies. This opportunity should also be used to thank Christian Hollensen, Lasse Farnung Laursen, J. Andreas Bærentzen, Oline Vinter Olesen, Rasmus Ramsbøl Jensen and Francois Anton for invaluable help in completing these notes, by clarifying things for me and via proof reading.

/Henrik

Part I

View Geometry

Chapter 1

Single Camera Geometry - Modelling a Camera

Much of computer vision and image analysis deals with inference about the world from images thereof. Many of these inference tasks require an understanding of the imaging process. Since such computer vision tasks are implemented on a computer this understanding needs to be in the form of a mathematical model. This is the subject here, where the relationship between the physical 3D world and 2D image there of is described, and mathematically modelled. It should be mentioned that this is only a short introduction to the vast field of (single) camera geometry, and that a good general reference for further reading is [14].

1.1 Homogeneous Coordinates

In order to have a more concise and less confusing representation of camera geometry, we use homogenous coordinates, which are introduced here. At the outset homogeneous coordinates are a silly thing, in that all coordinates — be they 2D or 3D — have an extra number or dimension added to them. We e.g. use three real numbers to represent a 2D coordinate and four reals to represent a 3D coordinate. As will be seen later this trick, however, makes sense and gives a lot of advantages. The extra dimension added is a scaling of the coordinate, such that the 2D point x, y is represented by the vector

$$\begin{bmatrix} sx \\ sy \\ s \end{bmatrix} .$$

Thus the coordinate or point $(3, 2)$ can, in homogeneous coordinates, be represented by a whole range of length three vectors, e.g.

$$\begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \cdot 3 \\ 1 \cdot 2 \\ 1 \end{bmatrix}, \quad \begin{bmatrix} 6 \\ 4 \\ 2 \end{bmatrix} = \begin{bmatrix} 2 \cdot 3 \\ 2 \cdot 2 \\ 2 \end{bmatrix}, \quad \begin{bmatrix} -6 \\ -4 \\ -2 \end{bmatrix} = \begin{bmatrix} -2 \cdot 3 \\ -2 \cdot 2 \\ -2 \end{bmatrix}, \quad \begin{bmatrix} 30 \\ 20 \\ 10 \end{bmatrix} = \begin{bmatrix} 10 \cdot 3 \\ 10 \cdot 2 \\ 10 \end{bmatrix} .$$

The same holds for 3D coordinates, where the coordinate x, y, z is represented as

$$\begin{bmatrix} sx \\ sy \\ sz \\ s \end{bmatrix} .$$

An as an example the point $(1, -2, 3)$ can be expressed in homogeneous coordinates as e.g.

$$\begin{bmatrix} 1 \\ -2 \\ 3 \\ 1 \end{bmatrix}, \quad \begin{bmatrix} -1 \\ 2 \\ -3 \\ -1 \end{bmatrix}, \quad \begin{bmatrix} 2 \\ -4 \\ 6 \\ 2 \end{bmatrix}, \quad \begin{bmatrix} 10 \\ -20 \\ 30 \\ 10 \end{bmatrix} .$$

This representation, as mentioned, has certain advantages in achieving a more compact and consist representation. Consider e.g. the points (x, y) located on a line, this can be expressed by the following equation, given a, b, c:

$$0 = ax + by + c \ . \tag{1.1}$$

Multiplying both sides of this equation by a scalar s we achieve

$$0 = sax + sby + sc = \begin{bmatrix} a \\ b \\ c \end{bmatrix}^T \begin{bmatrix} sx \\ sy \\ s \end{bmatrix} = \mathbf{l}^T\mathbf{q} \ , \tag{1.2}$$

where $\mathbf{l}$ and $\mathbf{q}$ are vectors, and specifically $\mathbf{q}$ is the possible 2D points on the line, represented in homogeneous coordinates. So now the equation for a line is reduced to the inner product between two vectors. A similar thing holds in 3D, where points on the plane (x, y, z) are solutions to the equation, given (a, b, c, d)

$$\begin{aligned} 0 &= ax + by + cz + d \Rightarrow \\ 0 &= sax + sby + scz + sd \\ &= \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}^T \begin{bmatrix} sx \\ sy \\ sz \\ s \end{bmatrix} \\ &= \mathbf{p}^T\mathbf{q} \ . \end{aligned} \tag{1.3}$$

Where $\mathbf{p}$ and $\mathbf{q}$ are vectors the latter representing the homogeneous 3D points on the plane. Another operation which can be performed differently with homogeneous coordinates is addition. Assume, e.g. that $(\Delta x, \Delta y)$ should be added to (x, y), then this can be written as

$$\begin{bmatrix} 1 & 0 & \Delta x \\ 0 & 1 & \Delta y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} sx \\ sy \\ s \end{bmatrix} = \begin{bmatrix} sx + s\Delta x \\ sy + s\Delta y \\ s \end{bmatrix} = \begin{bmatrix} s(x + \Delta x) \\ s(y + \Delta y) \\ s \end{bmatrix} \ , \tag{1.4}$$

which is equivalent to the point $(x + \Delta x, y + \Delta y)$, as needed. The same thing can be done in 3D[1]. This implies that we can combine operations into one matrix and e.g. represent the multiplication of a point by a matrix followed by an addition as a multiplication by one matrix. As an example let $\mathbf{A}$ be a 3×3 matrix, $\mathbf{q} = (x, y, z)$ a regular (non-homogeneous) 3D point and $\mathbf{\Delta q}$ another 3 vector, then we can write $\mathbf{Aq} + \mathbf{\Delta q}$ as

$$\begin{bmatrix} \mathbf{A} & \mathbf{\Delta q} \\ 000 & 1 \end{bmatrix} \begin{bmatrix} s\mathbf{q} \\ s \end{bmatrix} = \begin{bmatrix} s\mathbf{Aq} + s\mathbf{\Delta q} \\ s \end{bmatrix} = \begin{bmatrix} s(\mathbf{Aq} + \mathbf{\Delta q}) \\ s \end{bmatrix} \ . \tag{1.5}$$

In dealing with the pinhole camera model this will be a distinct advantage.

1.1.1 Points at Infinity*

There are naturally much more to homogeneous coordinates, especially due to their close link to projective geometry, and the interested reader is referred to [14]. A few subtleties will, however, be touched upon here, firstly points infinitely fare away. These are in homogeneous coordinates represented by the last entry being zero, i.e.

$$\begin{bmatrix} 2 \\ 1 \\ -1.5 \\ 0 \end{bmatrix} \ , \tag{1.6}$$

which if we try to convert it to regular coordinates corresponds to

$$\begin{bmatrix} 2/0 \\ 1/0 \\ -1.5/0 \end{bmatrix} \rightarrow \begin{bmatrix} \infty \\ \infty \\ \infty \end{bmatrix} \ . \tag{1.7}$$

[1] Do the calculations, and see for your self!

The advantage of the homogeneous representation in (1.6), as compared to a the regular in (1.7), is that the homogeneous coordinate represents infinitely far away with a given direction, i.e. a good model of the suns location. This is not captured by the regular coordinates, since $c\infty = \infty$, for any constant. One can naturally represent directions without homogeneous coordinates, but not in a seamless representation. I.e. in homogeneous coordinates we can both estimate the direction to the sun and a nearby object in the same framework. This also implies that in homogeneous coordinates, as in projective geometry, infinity is a place like any other. As a last note on points at infinity, consider the plane

$$\mathbf{p} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \quad , \quad \mathbf{p}^T\mathbf{q} = 0 \ . \tag{1.8}$$

which exactly contains all the homogeneous points, $\mathbf{q}$, which are located at infinity. Thus $\mathbf{p}$ in (1.8) is also known as the plane at infinity.

1.1.2 Intersection of Lines in 2D

As seen in (1.2), a point, $\mathbf{q}$, is located on a line, $\mathbf{l}$, iff[2] $\mathbf{l}^T\mathbf{q} = 0$. Thus the point, $\mathbf{q}$, which is the intersection of two lines, $\mathbf{l_1}$ and $\mathbf{l_2}$, will satisfy

$$\begin{bmatrix} \mathbf{l_1^T} \\ \mathbf{l_2^T} \end{bmatrix} \mathbf{q} = 0 \ . \tag{1.9}$$

Thus $\mathbf{q}$ is the right null space of the matrix

$$\begin{bmatrix} \mathbf{l_1^T} \\ \mathbf{l_2^T} \end{bmatrix} \ ,$$

which also gives an algorithm for finding the $\mathbf{q}$ which is the intersection of two lines. Another way of achieving the same is by taking the cross product between $\mathbf{l_1}$ and $\mathbf{l_2}$. The idea is that the cross product is perpendicular to the two vectors, i.e.

$$\begin{bmatrix} \mathbf{l_1^T} \\ \mathbf{l_2^T} \end{bmatrix} (\mathbf{l_1} \times \mathbf{l_2}) = 0 \ . \tag{1.10}$$

Thus the intersection of two lines, $\mathbf{l_1}$ and $\mathbf{l_2}$, is also given by:

$$\mathbf{q} = \mathbf{l_1} \times \mathbf{l_2} \ . \tag{1.11}$$

As an example consider the intersection of two lines $\mathbf{l_1} = [1, 0, 2]^T$ and $\mathbf{l_2} = [0, 2, 2]^T$. Then the intersection is given by

$$\mathbf{q} = \mathbf{l_1} \times \mathbf{l_2} = \begin{bmatrix} -4 \\ -2 \\ 2 \end{bmatrix} \ ,$$

which corresponds to the regular coordinate $(-2, -1)$, which the active reader can try and plug into the relevant equations and se that it fits. To illustrate the above issue of points at infinity, consider the two parallel lines $\mathbf{l_1} = [1, 0, -1]^T$ and $\mathbf{l_2} = [2, 0, -1]^T$. The intersection of these two lines is given by

$$\mathbf{q} = \mathbf{l_1} \times \mathbf{l_2} = \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix} \ ,$$

which is a point at infinity, as expected since these lines are parallel.

[2] "iff" means if and only if and is equivalent with the logical symbol $\Leftrightarrow$.

1.1.3 Distance to a Line*

A more subtle property of the homogeneous line representation, i.e. (1.2), is that it can easily give the distance to the line, $\mathbf{l}$, *if* the first two coordinates are normalized to one. I.e.

$$a^2 + b^2 = 1 \quad for \quad \mathbf{l} = \begin{bmatrix} a \\ b \\ c \end{bmatrix} .$$

In this case the distance of a point (x, y) is given by

$$dist = \left| \mathbf{l}^T \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \right| . \tag{1.12}$$

Comparing to (1.2), it is seen that points on the line are those that have zero distance to it — which seems natural.

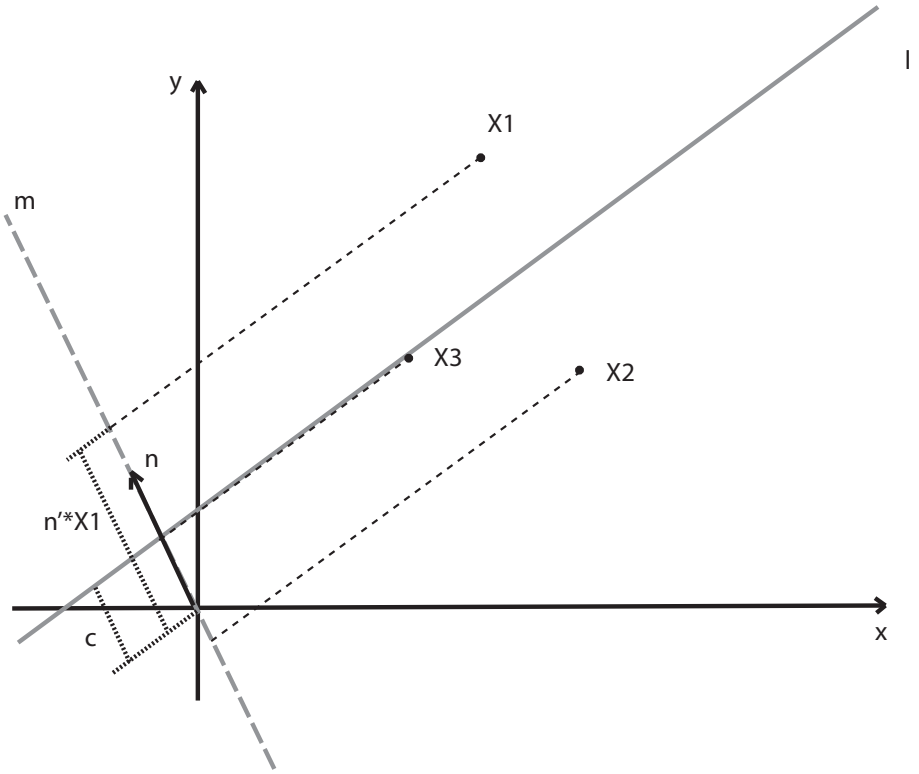

Figure 1.1: The distance from a point X_i to the line $\mathbf{l} = [\mathbf{n}^T, -c]^T$, is preserved by projecting the point onto a line normal to $\mathbf{l}$ (with direction $\mathbf{n}$). The distance is thus the inner product of X_i and $\mathbf{n}$ minus the projection of l onto its perpendicular line, c.

The reasoning is as follows: Firstly, denote the normal $\mathbf{n}$ to the line is given by, see Figure 1.1, by

$$\mathbf{n} = \begin{bmatrix} a \\ b \end{bmatrix} .$$

For any given point, $\mathbf{q} = [x, y, 1]^T$, project it *along* the line $\mathbf{l}$ onto the line $\mathbf{m}$. The line $\mathbf{m}$ passes through the origo[3] with direction $\mathbf{n}$. It is seen, that these two lines, $\mathbf{m}$ and $\mathbf{l}$, are perpendicular. The signed distance of this projection ($\mathbf{q}$ onto $\mathbf{m}$) to the origo is

$$\mathbf{n}^T \begin{bmatrix} x \\ y \end{bmatrix} , \tag{1.13}$$

see Figure 1.1, and is – obviously – located on $\mathbf{m}$. It is further more seen, c.f. Figure 1.1, that the signed distance of the projection, of $\mathbf{q}$ onto $\mathbf{m}$, to $\mathbf{l}$, is the same as the distance between $\mathbf{q}$ and $\mathbf{l}$. This latter fact, is,

[3] origo is the center of the coordinate system with coordinates $(0, 0)$.

among others, implied by projecting $\mathbf{q}$ parallel to $\mathbf{l}$. The problem thus boils down to finding the signed distance from the intersection of $\mathbf{m}$ and $\mathbf{l}$ to the origo, and subtracting that from (1.13). This distance can be derived from any point $\mathbf{q}_3$ on $\mathbf{l}$ (following the notation in Figure 1.1), for which it holds, by (1.2),

$$\mathbf{l}^T \begin{bmatrix} x_3 \\ y_3 \\ 1 \end{bmatrix} = \mathbf{n}^T \begin{bmatrix} x_3 \\ y_3 \end{bmatrix} + c = 0 \Rightarrow \mathbf{n}^T \begin{bmatrix} x_3 \\ y_3 \end{bmatrix} = -c \ .$$

This implies (1.12), since the signed distance is given by

$$\mathbf{n}^T \begin{bmatrix} x \\ y \end{bmatrix} - (-c) = \begin{bmatrix} \mathbf{n} \\ c \end{bmatrix}^T \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \mathbf{l}^T \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \ ,$$

and thus the (unsigned) distance is given by (1.12). This is consistent with the usual formula for the distance from a point to a line — as found in most tables of mathematical formulae — namely

$$dist = \frac{|ax + by + c|}{\sqrt{a^2 + b^2}} \ ,$$

where it is noted that $\sqrt{a^2 + b^2} = a^2 + b^2 = 1$, as assumed in our case.

1.1.4 Planes*

Moving from 2D to 3D, many of the properties of lines transfers to planes, with equivalent arguments. Specifically the distance of a 3D point, $\mathbf{q}$ to a plane, $\mathbf{p}$, is given by

$$dist = \frac{|\mathbf{p^T q}|}{||\mathbf{n}||_2} \quad , \quad \mathbf{p} = \begin{bmatrix} \mathbf{n} \\ -\alpha \end{bmatrix} \ ,$$

where $\mathbf{n}$ is the normal to the plane. This normal can be found as the cross product of two linearly independent vectors in the plane. To see this note that the normal has to be perpendicular to all vectors in the plane. From the equation for a plane

$$\mathbf{p^T q} = 0 \ .$$

It can be deduced that if a point $\mathbf{q}$ is located on two planes $\mathbf{p_1}$ and $\mathbf{p_2}$, then

$$\begin{bmatrix} \mathbf{p_1^T} \\ \mathbf{p_2^T} \end{bmatrix} \mathbf{q} = 0 \ .$$

This describes the points at the intersection of the two planes, which (if the two planes are not coincident) is a line in 3D.

1.2 Modelling a Camera

As mentioned, a mathematical model is needed of a camera, in order to solve most inference problems involving 3D. Specifically this model should relate the 3D model, the camera is viewing, and the generated image, see Figure 1.2. The form of the model is naturally of importance. So before such models are derived, it is good to consider what a *good model* is — which will be done in the following. Following this a few common models are introduced, for further information the interested reader is referred to [14, 24].

1.2.1 What is a Good Model

As an example of modelling consider dropping an object from a given height and predicting it's position, which pretty much boils down to modelling the objects acceleration, see Figure 1.3. A simple high school physics problem, would be most students reaction; the acceleration, a, is equal to $g \approx 9.81 m/s^2$. This answer is indeed a good one, and in many cases this is a good model of the problem. It is, however, not exact. This model does

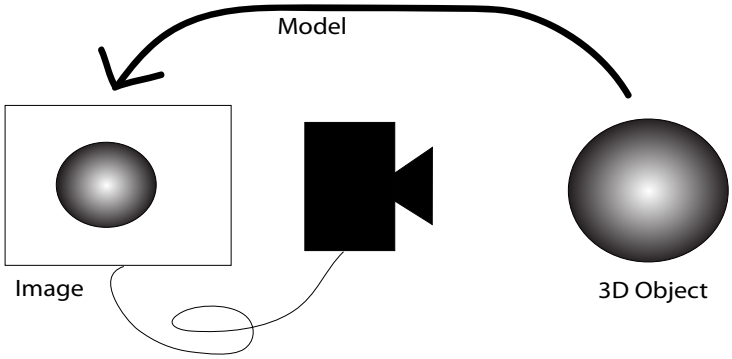

Figure 1.2: The required camera model should relate the 3D object viewed and the image generated by the camera.

not include wind resistance – if the object was e.g. a feather – and more subtle effects like relativity theoretical effects etc.

Two things should be observed from this example. Firstly, with very few exceptions, *perfect models of physical phenomena do not exist!* Where 'perfect' should be understood as exactly describing the physical process. Thus noise is often added to a model to account for unmodelled effects. Secondly, the more exact a model gets, the more complicated it usually gets, which makes calculations more difficult.

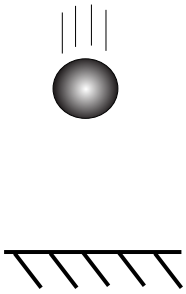

Figure 1.3: How fast will a dropped object accelerate?

So what is a good model? This answer depends on the purpose of the modelling. In Science the aim is to understand phenomena, and thus more exact models are usually the aim. In engineering the aim is solving real world problems via science, and thus a good model is one that enables you to solve the problem in a satisfactory manner. Since camera geometry is most often used for engineering problems, the latter position will be taken here, and we are looking for models with a good trade off between expressibility and simplicity.

1.2.2 Camera and World Coordinate Systems - Frame of Reference

Measurements have to be made in a frame of reference to make sense. With position measurements, e.g. $[1, -3.4, 3]^T$, this frame of reference is a *coordinate system*. A coordinate system is mathematically speaking a set of basis vectors, e.g. the x-axis, y-axis and z-axis, and an origo. The origo, $[0, 0, 0]^T$, is the center of the coordinate system. Here the coordinates, e.g. $[x, y, z]$, denote 'how much of' each basis vector is need to get to the point from the origo of the coordinate system. The typical coordinate system used is a right handed Cartesian system, where the basis vectors are orthogonal to each other and have length one. Right handed implies that the z-axis is equal to the cross product of the x-axis and y-axis. In this text, a right handed Cartesian coordinate system will be assumed, unless otherwise stated.

Often, in camera geometry, we have several coordinate systems, e.g. one for every camera and perhaps a global coordinate system, and a robot coordinate system as well. The reason being that often times it is better and easier to express image measurements in the reference frame of the camera taking the image, see Figure 1.4.

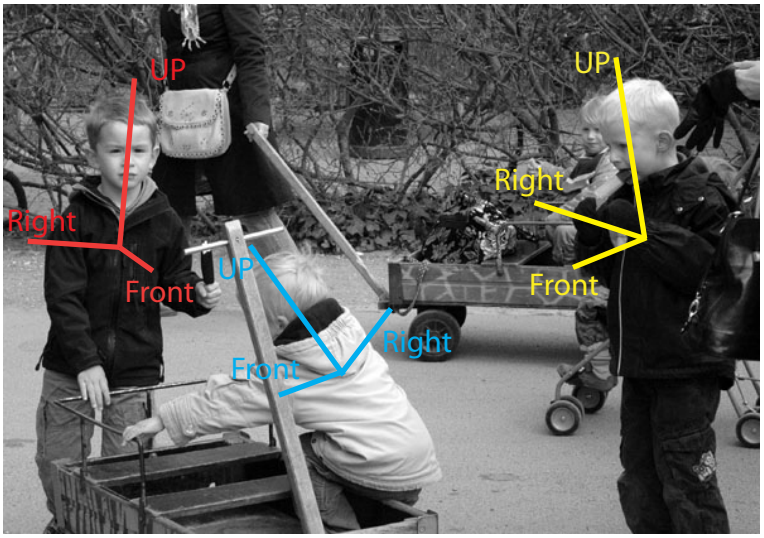

Figure 1.4: It is not only in camera geometry, where a multitude of reference frames exist. What is to the right of the boy to the left is in front of the boy to the right.

Experience has, however, shown that one of the things that makes camera geometry difficult is this abundance of coordinate systems, and especially the transformations between these, see Figure 1.5. Coordinate system transformations[4] will, thus, be shortly covered here for a right handed Cartesian coordinate system, and in a bit more detail in Appendix A.

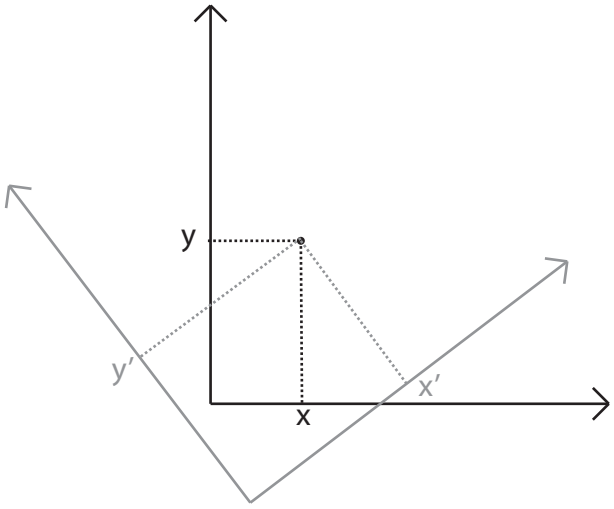

Figure 1.5: An example of a change of coordinate systems. The aim is to find the coordinates of the points in the gray coordinate system, i.e. (x', y'), given the coordinates of the point is the black coordinate system, i.e. (x, y). Note, that the location of the point does not change (in some sort of global coordinate system).

From basic mathematics, it is known that we can transform a point from any right handed Cartesian coordinate system to another via a rotation and a translation, see Appendix A.4. That is, if a point Q is given in one coordinate system, it can be transferred to any other, with coordinates Q' as follows

$$Q' = \mathbf{R}Q + \mathbf{t} \ , \tag{1.14}$$

where $\mathbf{R}$ is a 3 by 3 rotation matrix, and $\mathbf{t}$ is a translation vector of length 3. Rotation matrices are treated briefly in Appendix A. As seen in (1.5) this can in homogeneous coordinates be written as

$$Q' = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ 000 & 1 \end{bmatrix} Q \ . \tag{1.15}$$

[4]This is also called *basis shift* in linear algebra.

The inverse transformation, $\mathbf{R}', \mathbf{t}'$ is given by (note that the inverse of a rotation matrix is given by its transpose, i.e. $\mathbf{R}^{-1} = \mathbf{R}^T$)

$$\begin{aligned} Q' &= \mathbf{R}Q + \mathbf{t} \Rightarrow \\ \mathbf{R^T}Q' &= Q + \mathbf{R^T}\mathbf{t} \Rightarrow \\ \mathbf{R^T}Q' - \mathbf{R^T}\mathbf{t} &= Q \Rightarrow \\ \mathbf{R}' = \mathbf{R^T} \quad &, \quad \mathbf{t}' = -\mathbf{R^T}\mathbf{t} \ . \end{aligned}$$

Finally note, that it does matter if the coordinate is first rotated and the translated, as in (1.14), or first translated and then rotated, i.e. in general

$$\mathbf{R}Q + \mathbf{t} \neq \mathbf{R}(X + \mathbf{t}) = \mathbf{R}Q + \mathbf{R}\mathbf{t} \ .$$

1.3 The Orthographic Projection Model

One of the simplest camera models is the orthographic or parallel projection. This assumes that light hitting the image sensor travels in parallel lines, see Figure 1.6-left. Assuming that the camera is aligned with the world coordinate system, such that it is viewing along the z-axis, then a world point $Q_i = [X_i, Y_i, Z_i]^T$ will project to the image point $\mathbf{q_i} = [x_i, y_i]^T = [X_i, Y_i]^T$. This is equivalent to projecting the world point along the z-axis, and letting the xy-plane being the image plane, see Figure 1.6-right. Mathematically this can be written as (in homogeneous coordinates)

$$\begin{bmatrix} sx_i \\ sy_i \\ s \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} sX_i \\ sY_i \\ sZ_i \\ s \end{bmatrix} \ . \tag{1.16}$$

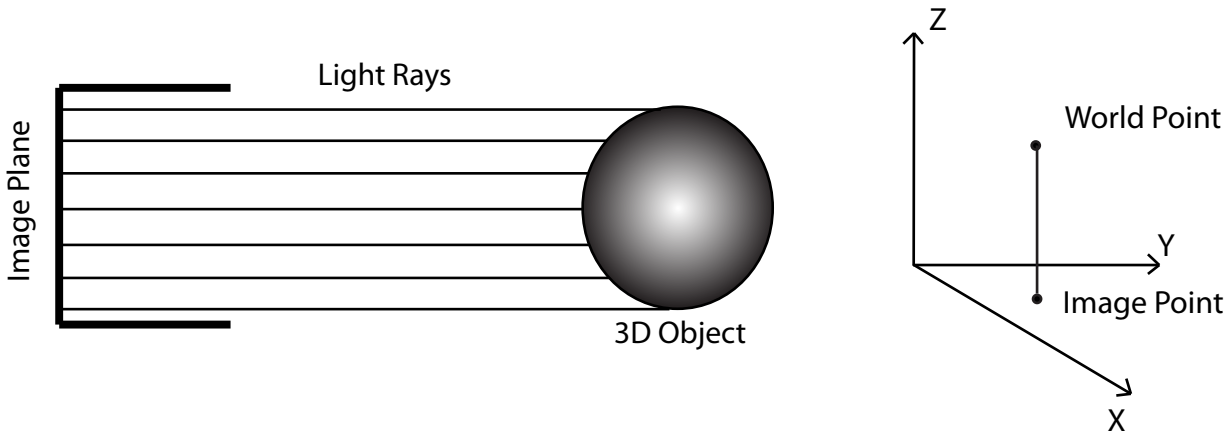

Figure 1.6: **Left:** Illustration of an orthographic camera, where it is assumed that light only travels in parallel lines, thereupon illuminating the photosensitive material. **Right:** This is mathematically equivalent to projecting a world coordinate along an axis, here the z-axis, onto a plane, in this case the xy-plane.

There are two 'errors' with the model in (1.16). Firstly, this model assumes that the camera is viewing along the z-axis, which will seldom be the case. This is equivalent to saying that the world and the camera coordinate system are equivalent. As described in Section 1.2.2, we can transform the coordinates of the points, Q_i — which are expressed in the world coordinate system — into the camera coordinate system, via a rotation and a translation, transforming (1.16) into

$$\begin{bmatrix} sx_i \\ sy_i \\ s \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} \begin{bmatrix} sX_i \\ sY_i \\ sZ_i \\ s \end{bmatrix} \ .$$

The second 'error' (1.16), is that the unit of measurement of the world coordinate system and of the image is seldom the same. So a pixel might correspond to a 10m by 10m area. Thus there is a need to scale the result by

a constant c, giving us the final orthographic projection model

$$\mathbf{q_i} = \mathbf{P_{ortho}} Q_i \quad , \quad \mathbf{P_{ortho}} = \begin{bmatrix} c & 0 & 0 & 0 \\ 0 & c & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} \ . \tag{1.17}$$

1.3.1 Discussion of the Orthographic Projection Model

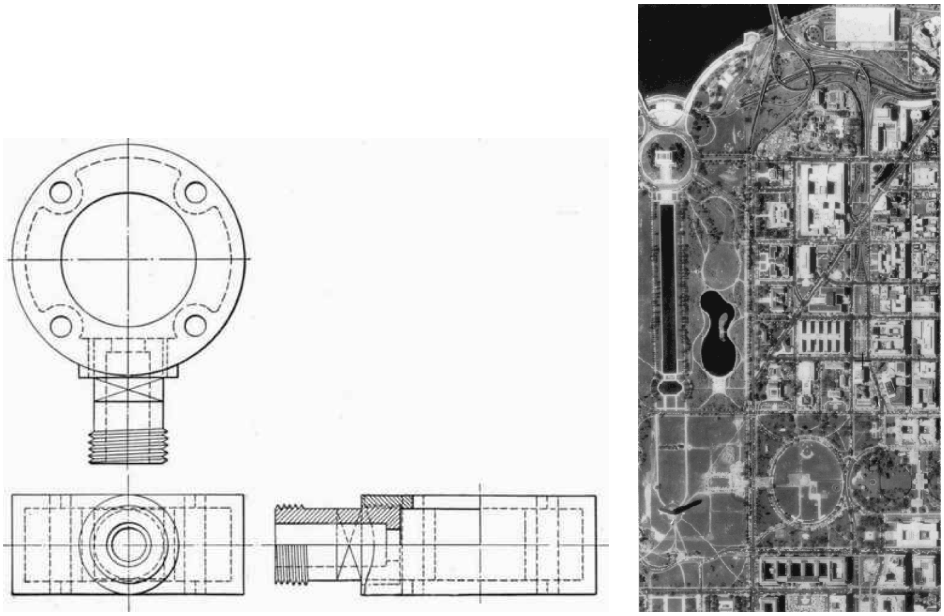

Figure 1.7: Examples of orthographic projections in technical drawings (**Left**) and maps (**Right**). The right image is taken from Wikipedia.

Although the orthographic projection model does not resemble any common camera well, it is still a very used model. Examples of its use are architectural and technical drawings, see Figure 1.7-Left, and maps, see Figure 1.7-Right, which is in the form of the so called orthophotos.

A main issue with the orthographic projection model is that it indifferent to depth. Consider (1.16), where a change in the z-coordinate of the 3D world point Q_i will not change the projected point. Here the z-axis is the depth direction. This implies that there is no visual effect of moving an object further away from, or close to, the camera. There is thus no perspective effect. Hence in applications where depth is of importance the orthographic projection model is seldom suitable.

1.4 The Pinhole Camera Model

The most used camera model in computer vision and photogrammetry is the pinhole or projective camera model. This model is also the one manufactures of 'normal' lenses often aim at having their optics resemble. In this text this model will be assumed, unless otherwise stated. Needless to say that this is an important model to understand and master, so in the following it will be derived in some detail.

1.4.1 Derivation of the Pinhole Camera Model

The idea behind the pinhole model is that the image sensor is in-caged in a box with a small pinhole in it, as illustrated in Figure 1.8. The light is then thought to pass through this hole and illuminate the image sensor.

The coordinate system of the pin hole camera model is situated such that the image plane is coincident with the xy-plane and the z axis is along the viewing axis. To derive the pinhole camera model consider first the xz-plane of this camera model, as seen in Figure 1.9. In this 2D world camera it is seen that

$$x = \frac{x}{1} = \frac{X}{Z} \ . \tag{1.18}$$

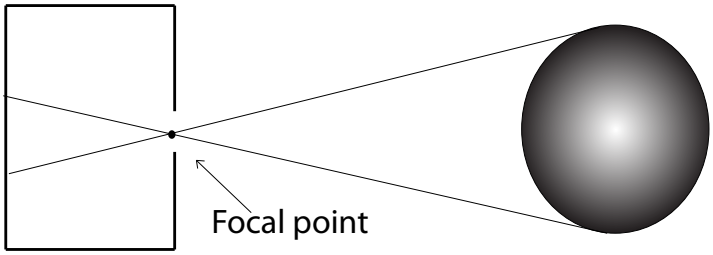

Figure 1.8: Illustration of an ideal pinhole camera, where it is assumed that light only passes through a tiny (i.e. pin) hole in the camera housing, and thus illuminating the photosensitive material.

which constitutes the camera model – somewhat simplified. This is simplified in the sense that the image plane is assumed to be unit distance from the origo, and the camera coordinate system aligned with the global coordinate system. One thing to note is that in Figure 1.8 the image plane is behind the origo of the coordinate system and in Figure 1.9, it is in front. Apart from a flipping of the image plane these are equivalent, as illustrated in Figure 1.10

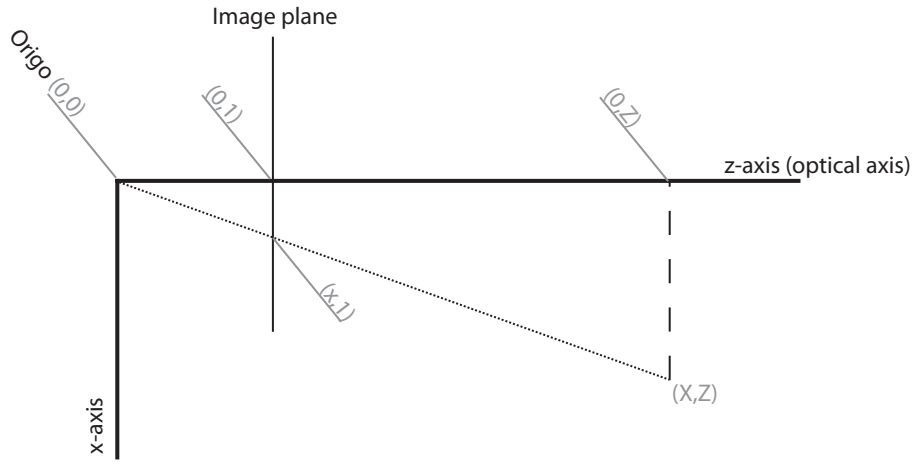

Figure 1.9: The point (X, Z) is projected onto the image plane with coordinates x. It is seen that the triangles $\triangle(0,0),(0,1),(x,1)$ and $\triangle(0,0),(0,Z),(X,Z)$ are scaled versions of each other , and thus $x/1 = X/Z$. Here the image plane is assumed to be unit distance from the origo, and the camera coordinate system aligned with the global coordinate system.

Extending from this 2D camera to 3D, it is seen that, since the x and y axis are orthogonal, the model from (1.18), is still valid for the x image coordinate and that an equivalent model holds for the y image coordinate, i.e.

$$x = \frac{X}{Z} \quad , \quad y = \frac{Y}{Z} \ . \tag{1.19}$$

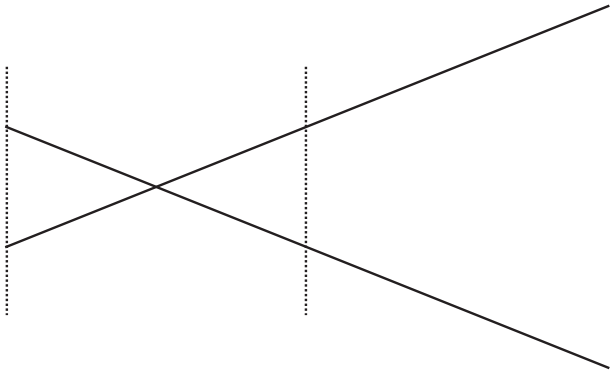

Figure 1.10: Except for a flipping of the image plane, a projection in front of or behind the focal point, or origo, are equivalent, as long as the distance from the focal point is the same.

See Figure 1.11. This can be written with the use of homogeneous coordinates, where the 2D point, $\mathbf{q}$, is homogeneous coordinates and the 3D point, Q is in regular coordinates (assuming that the depth $Z \neq 0$):

$$\begin{aligned} \mathbf{q_i} &= Q_i \\ \begin{bmatrix} s_i x_i \\ s_i y_i \\ s_i \end{bmatrix} &= \begin{bmatrix} X_i \\ Y_i \\ Z_i \end{bmatrix} \Leftrightarrow \\ s_i = Z_i \quad s_i x_i &= X_i \quad s_i y_i = Z_i \Leftrightarrow \\ Z_i x_i = X_i \quad , & \quad Z_i y_i = Y_i \Leftrightarrow \\ x_i = \frac{X_i}{Z_i} \quad , & \quad y_i = \frac{Y_i}{Z_i} \end{aligned} \tag{1.20}$$

This camera model in (1.20), assumes that the camera and the global[5] coordinate systems are the same. This is seldom the case, and as explained in Section 1.2.2, this shortcoming can be addressed with a rotation and a translation, making the camera model

$$\mathbf{q_i} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} Q_i \ .$$

This model, however, has not captured the camera optics, i.e. the internal parameters. This model, as an example, assumes that the distance of the image plane from the origo is one, which it seldom is. The internal parameters will be described in more detailed in the following. With the pinhole model these internal parameters are represented by a linear model, expressible by the 3 by 3 matrix $\mathbf{A}$, making the pinhole camera model

$$\mathbf{q_i} = \mathbf{A} \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} Q_i = \mathbf{P} Q_i \quad , \quad \mathbf{P} = \mathbf{A} \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} \ , \tag{1.21}$$

where Q_i is now in homogeneous coordinates. *Thus* (1.21) *constitutes the final and actual pinhole camera model.* Denoting $\mathbf{P}$ by it's elements

$$\mathbf{P} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \ ,$$

the pinhole camera model in (1.21) can be written out in terms of the world coordinates $[X_i, Y_i, Z_i]^T$ and image coordinates $[x_i, y_i]^T$ as

$$\begin{aligned} x_i &= \frac{p_{11} X_i + p_{12} Y_i + p_{13} Z_i + p_{14}}{p_{31} X_i + p_{32} Y_i + p_{33} Z_i + p_{34}} \ , \\ y_i &= \frac{p_{21} X_i + p_{22} Y_i + p_{23} Z_i + p_{24}}{p_{31} X_i + p_{32} Y_i + p_{33} Z_i + p_{34}} \ . \end{aligned} \tag{1.22}$$

This hopefully illustrates that the use of homogeneous coordinates makes camera geometry more concise.

[5] The coordinate system of the 3D points.

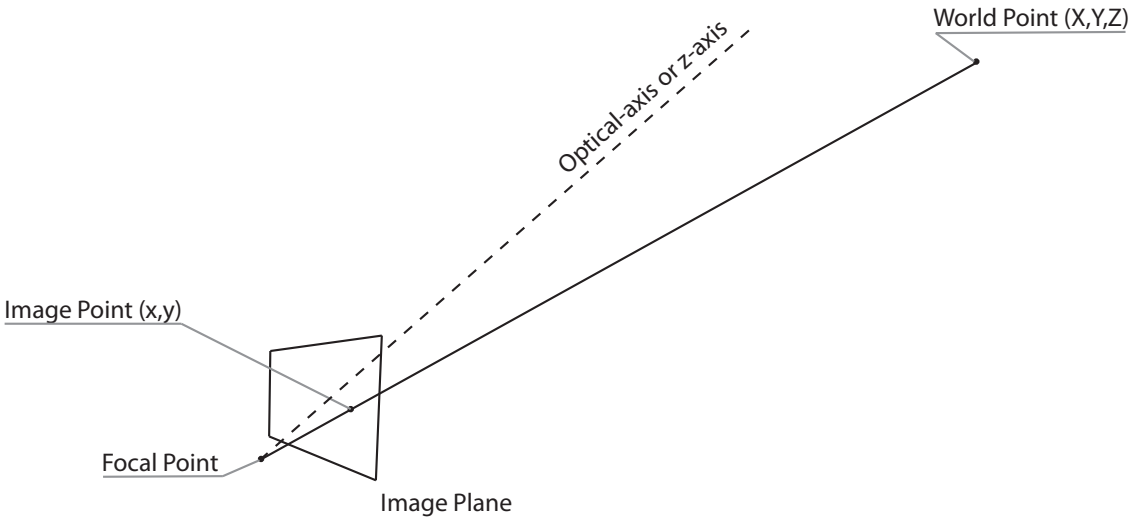

Figure 1.11: An illustration of a projection in 3D where a 3D world point (X, Y, Z), projects to a 2D image point (x, y). This projection can be found by determining where the straight line between the focal point and the 3D point (X, Y, Z) intersects the image plane.

1.4.2 Camera Center*

Sometimes it is necessary to calculate the projection center of a camera, Q_c, given its projection matrix $\mathbf{P}$. This is to be done in global coordinates. Applications include robot navigation where we want to know where the camera center is – and thus the robot – from an estimate of the camera. It is seen that in the camera coordinate system $Q_c = \mathbf{0}$, thus

$$\mathbf{0} = \mathbf{A} \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} Q_c \quad \Rightarrow \quad \mathbf{0} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} Q_c \quad \Rightarrow \quad \mathbf{0} = \mathbf{R}\tilde{Q}_c + \mathbf{t} \quad \Rightarrow \quad \tilde{Q}_c = -\mathbf{R}^{\mathbf{T}}\mathbf{t} \ . \tag{1.23}$$

Where $\tilde{Q}_c$ is the *in*homogeneous coordinates corresponding to Q_c. An alternative way of calculating the camera center is by noting that (1.23) states that Q_c is the right null vector of $\mathbf{P}$. Thus Q_c can e.g. be found via the following MatLab code

```
[u,s,v]=svd(P);
Qc=v(:,end);
Qc=Qc/Qc(4);
```

1.4.3 Internal Parameters

So far we have not dealt with the internal part of the camera, such as lenses, film size and shape etc. This naturally also needs to be dealt with. As with cameras themselves there are naturally also many models for these internal parameters. The model $\mathbf{A}$ presented here, used in (1.21), is by far the most common used linear model, and has the form

$$\mathbf{A} = \begin{bmatrix} f & f\beta & \Delta x \\ 0 & \alpha f & \Delta y \\ 0 & 0 & 1 \end{bmatrix} \ . \tag{1.24}$$

The typical way to extend this model is via including non-linear terms, which is the subject of Section 1.5. In the rest of this subsection the parameters of $\mathbf{A}$ will be covered.

Focal Length — f

The focal length, f, the distance from the focal point to the image plane in the above derivation, which was set to one. That is (1.18) should actually have been

$$\frac{x}{f} = \frac{X}{Z} \Rightarrow x = \frac{fX}{Z} \ ,$$

which is the effect this parameter has when (1.24) is applied in (1.21). If this is not clear please derive this, by setting $\Delta x = 0,\ \Delta y = 0,\ \beta = 0$ and $\alpha = 1$, and assuming that the camera coordinate system and the global coordinate system are identical – i.e. $R = I$ and $t = \mathbf{0}$. It is this focal lens which determines if we have[6] a wide angle lens e.g. $f = 10mm$, a normal angle lens e.g. $50mm$ or a zoom lens e.g. $200mm$. The focal length is also directly linked to the field of view, as treated in Section 1.4.4.

Image Coordinate of Optical Axis — $\Delta x, \Delta y$

As illustrated in Figure 1.11 and Figure 1.9, the optical axis tends to intersect the image plane around the middle. This intersection we sometimes call the optical point. If nothing else is done the image origo, pixel coordinate $(0, 0)$, would thus be at this optical point, due to the nature of the camera coordinate system. This is inconsistent with the usual way we represent images on a computer, where we like the origo to be in one of the image corners — e.g. the upper left. To address this, we want to translate the image coordinate system with a vector $[\Delta x, \Delta y]^T$, such that the chosen image corner translates to $0, 0$. The value of this vector, $[\Delta x, \Delta y]^T$, is equal to the coordinate of the optical point in the image coordinate system. To see this, note that before the translation the optical point has coordinate $(0, 0)$ so after adding $[\Delta x, \Delta y]^T$, it will have coordinate $(\Delta x, \Delta y)$, see Figure 1.12. As seen in (1.4), this translation can be done as in (1.24).

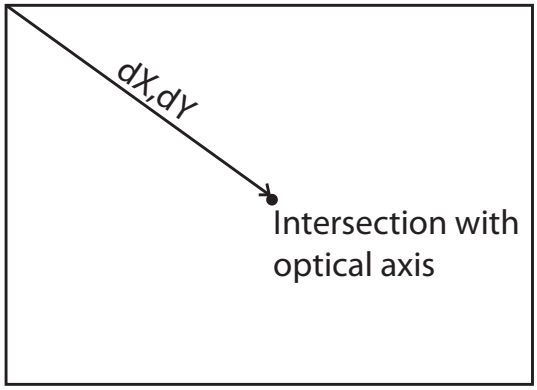

Figure 1.12: The translation vector, $[\Delta x, \Delta y]^T$, needed to get the upper left corner to the origo, is the vector from this image corner to the optical point.

Affine Image Deformation — α & β

The last two internal parameters, α and β, are not as relevant as they have been, and are mainly concerned with what happens when celluloid film was processed and scanned. In this case some deformation of the film could occur, which is here modelled by a scaling, α and a shearing, β, see Figure 1.13. Since we today mainly deal with images recorded directly onto an imaging chip, it is often safe to assume that $\alpha = 1$ and $\beta = 0$.

Another reason that α and β are kept in the model is that this gives $\mathbf{P}$ twelve degrees of freedom corresponding to its twelve elements. This has the advantage in some algorithms, e.g. for estimating $\mathbf{P}$, where a 3 by 4 matrix, with no constraints, can be estimated.

1.4.4 Properties of the Pinhole Camera Model

To restate the pinhole camera model from (1.21), it projects a 3D world point Q_i onto a 2D image point (both in homogeneous representation), via

$$\mathbf{q_i} = \mathbf{P}Q_i \quad , \quad \mathbf{P} = \mathbf{A}\begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} \quad , \quad \mathbf{A} = \begin{bmatrix} f & f\beta & \Delta x \\ 0 & \alpha f & \Delta y \\ 0 & 0 & 1 \end{bmatrix} \quad ,$$

where $\mathbf{P}$ is a 3 by 4 matrix, $\mathbf{A}$ are the internal parameters, $\mathbf{R}$ is a rotation and $\mathbf{t}$ a translation. In this subsection some of the properties of the pinhole camera will be discussed.

[6] The interpretation of the angles is for a standard consumer camera with a $35mm$ celluloid film.

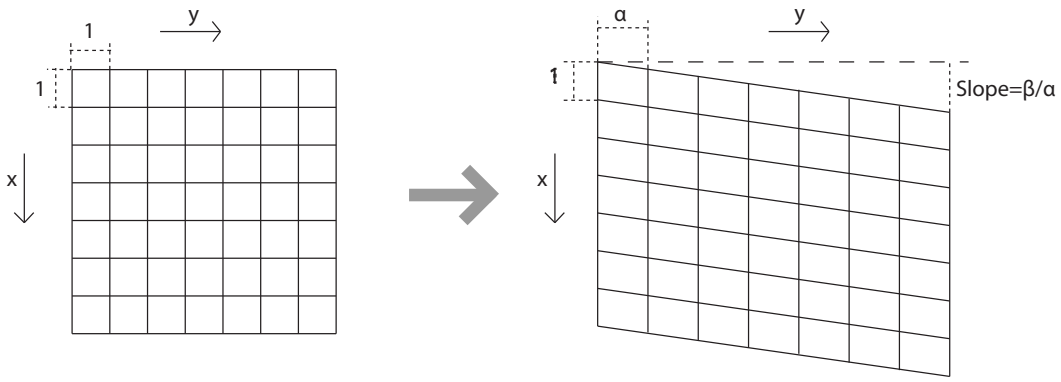

Figure 1.13: The scaling by a factor of α and a shearing of a factor of β is illustrated on a regular grid.

Internal parameters, $\mathbf{A}$	5
Rotation, $\mathbf{R}$	3
Translation, $\mathbf{t}$,	3
Scale	1
Total	12

Table 1.1: The total number of degrees of freedom of the pinhole camera model. As for further insight into parametrization of a rotation see Appendix A.

Degrees of Freedom

The number of parameters in this model is accounted for in Table 1.1, where some or all may be known. The total number of parameters is equal to twelve, the same as the number of elements in the 3 by 4 matrix $\mathbf{P}$. The scale comes from the fact, that in this transformation between homogeneous coordinates, scale does not matter, so it is a degree of freedom, i.e.

$$\mathbf{q_i} \approx s\mathbf{q_i} = s\mathbf{P}Q_i \ ,$$

where s is a scalar and $\approx$ means homogeneous equivalent. The fact that the pinhole model has twelve degrees of freedom indicates that any full rank 3 by 4 matrix can be a projection matrix $\mathbf{P}$, which also holds true. As mentioned above this makes many algorithms easier.

Field of View

One of the properties often enquired about a camera is its field of view, i.e. how wide an angle is the field of view. This tells us how much is viewed by the camera, and is an important fact in setting up surveillance cameras, e.g. for industrial inspection or security reasons. This also gives rise to a taxonomy of lenses based on viewing angle (and accompanying sensor sizes) , c.f. Table 1.2. Since the image sensor is square, the viewing 'volume' is pyramid shaped, and the angle of this pyramid is a matter of definition. If nothing else is stated, the field of view is — usually – thought to mean the diagonal angle, see Figure 1.14. If another angle, e.g. the vertical or horizontal, is sought, the derivation here can be adapted in a straight forward manner, by using a different length l.

The center of the field of view derivation is noting that the distance from the projection center to the image plane is the focal length f, as depicted in Figure 1.14. Consider the triangle composed of the diagonal of the image plane and the projection center, see Figure 1.14-right. This angle of this triangle opposite the image

Narrow angle	$0^\circ < \theta \leq 45^\circ$
Normal Angle	$45^\circ < \theta \leq 75^\circ$
Wide Angle	$75^\circ < \theta \leq 105^\circ$
Super wide angle	$105^\circ < \theta < 360^\circ$

Table 1.2: A taxonomy of lenses based on field of view, θ, c.f. [5].

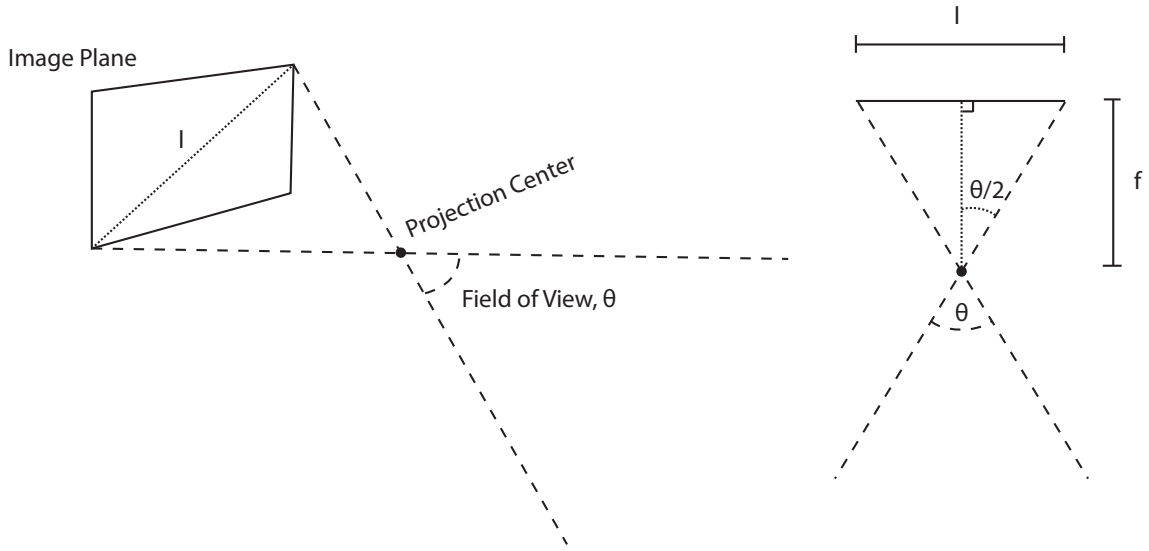

Figure 1.14: A schematic view of the pinhole camera model in relation to the field of view, θ. It is seen that we can form a right angled triangle, with one angle equaling $\theta/2$ and the length of the two sides f and $l/2$.

plane is also equal to the field of view, θ, and its height is f and its base is the diagonal length of the image plane l. Splitting this triangle in half, see Figure 1.14-right, will give us a right triangle, and thus

$$\begin{aligned} \tan\left(\frac{\theta}{2}\right) &= \frac{l/2}{f} \Rightarrow \\ \frac{\theta}{2} &= \arctan\left(\frac{l/2}{f}\right) \Rightarrow \\ \theta &= 2\arctan\left(\frac{l/2}{f}\right) \ . \end{aligned} \tag{1.25}$$

As an example consider an image with dimensions 1200×1600, and thus with the length of the diagonal equaling

$$l = \sqrt{1200^2 + 1600^2} = 2000 pixels \ .$$

This image is taken with a focal length of $2774.5 pixels$, thus

$$\theta = 2\arctan\left(\frac{l/2}{f}\right) = 2\arctan\left(\frac{2000/2}{2774.5}\right) = 39.64^\circ \ ,$$

equaling a narrow angle lens, according to Table 1.2.

Images of Straight Lines are Straight

The pinhole camera model, c.f. (1.21), is not a linear model in itself, e.g. as seen by (1.22). The pinhole camera model, however, has some linear properties, in that it maps straight lines in 3D world coordinates to straight lines in 2D. In other words the image of a straight line will be straight, if we assume the pinhole camera model. To see this, denote a given line by

$$Q + \alpha S$$

where Q and S are homogeneous 3D points, and α is a free scalar parameter. Projecting this via the pinhole camera model gives

$$\mathbf{l^h}(\alpha) = \mathbf{P}(Q + \alpha S) = \mathbf{P}Q + \alpha \mathbf{P}S = \mathbf{q} + \alpha \mathbf{s} \ , \tag{1.26}$$

where $\mathbf{q}$ and $\mathbf{s}$ are the homogeneous image points that are the projections of Q and S. It is seen that $\mathbf{l^h}(\alpha)$ in (1.26) is a line in 2D homogeneous space. To see that this is also a line is inhomogeneous space, i.e. that

$$\mathbf{l}(\alpha) = \begin{bmatrix} \frac{\mathbf{l^h_x}(\alpha)}{\mathbf{l^h_s}(\alpha)} & \frac{\mathbf{l^h_y}(\alpha)}{\mathbf{l^h_s}(\alpha)} \end{bmatrix}^T = \begin{bmatrix} \frac{\mathbf{q_x} + \alpha \mathbf{s_x}}{\mathbf{q_s} + \alpha \mathbf{s_s}} & \frac{\mathbf{q_y} + \alpha \mathbf{s_y}}{\mathbf{q_s} + \alpha \mathbf{s_s}} \end{bmatrix}^T$$

is a line, where the x, y, s subscripts denote the coordinates, take the derivative wrt. α, i.e.

$$\begin{aligned}\frac{\partial}{\partial\alpha}\mathbf{l}(\alpha) &= \left[\frac{\mathbf{s_x}(\mathbf{q_s}+\alpha\mathbf{s_s})-\mathbf{s_s}(\mathbf{q_x}+\alpha\mathbf{s_x})}{(\mathbf{q_s}+\alpha\mathbf{s_s})^2} \quad \frac{\mathbf{s_y}(\mathbf{q_s}+\alpha\mathbf{s_s})-\mathbf{s_s}(\mathbf{q_y}+\alpha\mathbf{s_y})}{(\mathbf{q_s}+\alpha\mathbf{s_s})^2}\right]^T \\ &= \left[\frac{\mathbf{s_x}\mathbf{q_s}-\mathbf{s_s}\mathbf{q_x}+(\mathbf{s_x}\mathbf{s_s}-\mathbf{s_s}\mathbf{s_x})\alpha}{(\mathbf{q_s}+\alpha\mathbf{s_s})^2} \quad \frac{\mathbf{s_y}\mathbf{q_s}-\mathbf{s_s}\mathbf{q_y}+(\mathbf{s_y}\mathbf{s_s}-\mathbf{s_s}\mathbf{s_y})\alpha}{(\mathbf{q_s}+\alpha\mathbf{s_s})^2}\right]^T \\ &= \frac{1}{(\mathbf{q_s}+\alpha\mathbf{s_s})^2}\begin{bmatrix}\mathbf{s_x}\mathbf{q_s}-\mathbf{s_s}\mathbf{q_x} & \mathbf{s_y}\mathbf{q_s}-\mathbf{s_s}\mathbf{q_y}\end{bmatrix}^T ,\end{aligned}$$

which is a constant vector times a scalar function. The direction of the derivative,

$$\frac{\frac{\partial}{\partial\alpha}\mathbf{l}(\alpha)}{||\frac{\partial}{\partial\alpha}\mathbf{l}(\alpha)||} ,$$

is thus constant, and the direction of the line in the image. Furthermore the line will go through $\mathbf{q}$, hereby fully defining the line.

1.4.5 Examples

Figure 1.15: Two images of the same scene, taken with a camera modelled well by the pinhole camera model. The difference between the images is that the left image is taken with a focal length of ca. $70mm$ and the right with a focal length of ca. $18mm$. The position of the camera is also varied such that the position and size of the mannequins torso is approximately the same. This illustrates the effect of perspective, which is a main difference between the pinhole and the orthographic camera model.

As an example of the pinhole camera model, consider Figure 1.15 and Figure 1.16. For the example in Figure 1.16, we are given the 3D point

$$Q = \begin{bmatrix}-1.3540\\0.5631\\8.8734\\1.0000\end{bmatrix} ,$$

Figure 1.16: An image of a model house onto which we have projected a 3D point of a window onto the image via the pinhole camera model, denoted by the red dot.

and are informed that the pinhole camera has the external parameters

$$\mathbf{R} = \begin{bmatrix} 0.9887 & -0.0004 & 0.1500 \\ 0.0008 & 1.0000 & -0.0030 \\ -0.1500 & 0.0031 & 0.9887 \end{bmatrix} \quad , \quad \mathbf{t} = \begin{bmatrix} -2.1811 \\ 0.0399 \\ 0.5072 \end{bmatrix} \quad .$$

The internal parameters are given by (this is a digital camera)

$$\begin{aligned} f &= 2774.5 \\ \Delta x &= 806.8 \\ \Delta y &= 622.6 \\ \alpha &= 1 \\ \beta &= 0 \end{aligned}$$

so

$$\mathbf{A} = \begin{bmatrix} 2774.5 & 0 & 806.8 \\ 0 & 2774.5 & 622.6 \\ 0 & 0 & 1 \end{bmatrix} \quad ,$$

and the combined pinhole camera model is given by

$$\mathbf{P} = \mathbf{A} \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} = \begin{bmatrix} 2622.1 & 1.5 & 1213.9 & -5642.4 \\ -91.1 & 2776.4 & 607.2 & 426.5 \\ -0.2 & 0 & 1.0 & 0.5 \end{bmatrix} \quad .$$

The 3D point Q thus projects to

$$\mathbf{q} = \mathbf{P}Q = \begin{bmatrix} 2622.1 & 1.5 & 1213.9 & -5642.4 \\ -91.1 & 2776.4 & 607.2 & 426.5 \\ -0.2 & 0 & 1.0 & 0.5 \end{bmatrix} \begin{bmatrix} -1.3540 \\ 0.5631 \\ 8.8734 \\ 1.0000 \end{bmatrix} = \begin{bmatrix} 1579.7 \\ 7500.7 \\ 9.5 \end{bmatrix} = 9.5 \begin{bmatrix} 166.5 \\ 790.8 \\ 1 \end{bmatrix} \quad .$$

So the inhomogeneous image point corresponding to Q is $(166.5, 790.8)$, as depicted in Figure 1.16.

1.5 Radial Distortion - Refined Pinhole Model

The standard pinhole model presented above, c.f. Section 1.4, does not model a camera accurate enough for some higher precision measurement task. This is an effect that typically increases with the field of view of the less and with a lowering of the lens cost. Therefore, when higher precision is required the pinhole model is refined. There are several ways of doing this, all associated with making a better internal model of the camera. The most common way of doing this, and what will be covered here, is dealing with radial distortion. The effects of radial distortion are illustrated in Figure 1.17. The problem addressed is that straight 3D lines are not depicted straight, as they should be according to the pinhole camera model, c.f. Section 1.4.4. This error increases with the distance from the center of the image, i.e. near the edges.

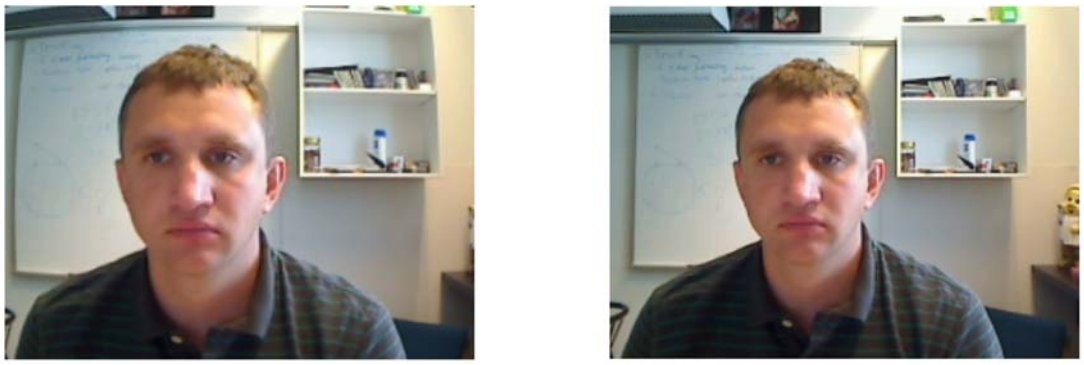

Figure 1.17: The effects of radial distortion, **Left:** An image with (allot) of radial distortion. Notice the rounding of the face, and that the edges are curved, especially close to the edges. **Right:** The same image without radial distortion. This image can be produced from the left image, if the radial distortion parameters are known.

Radial distortion is largely ascribed to a modern photographic lens having several lens elements. As the name indicates, radial distortion is a distortion in the radial 'direction' of the image, i.e. objects should appear closer to, or further from the center of the image then they do. Thus the distance from the center of the image, r, is a good way of parameterizing the effect, see Figure 1.18.

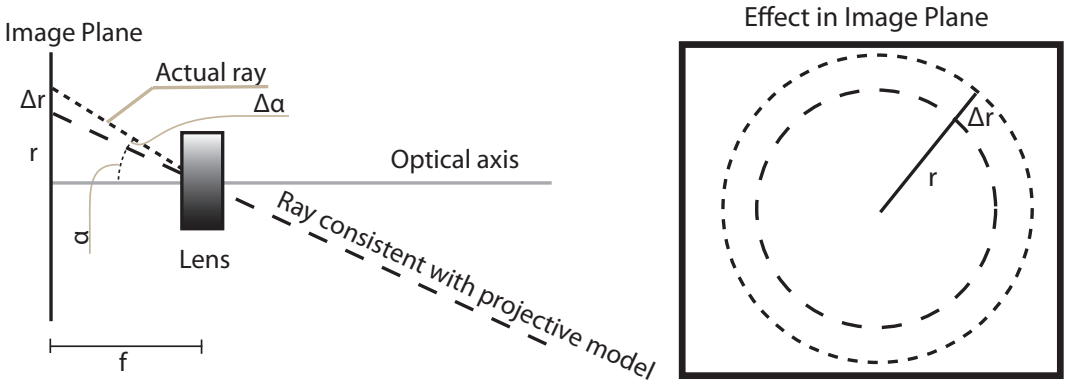

Figure 1.18: **Left:** Radial distortion is a nonlinear modelling of the effect, that a ray of light is 'bent' with the amount of $\Delta\alpha$, where $\Delta\alpha$ is a function of the angle α to the optical axis. Since the radius, r, and the angle α are related by $\alpha = \arctan\left(\frac{r}{f}\right)$, such a change in α results in a change of the radius Δr, as a function of the radius r. **Right:** In the image plane, points of equal radius from the optical axis form a circle. All points on such a circle have the same amount of radial distortion, Δr.

To refine the pinhole camera model (1.21), such that we can do non-linear corrections based on the radial distance, we have to split the linear model of the internal camera mode, i.e. $\mathbf{A}$ in (1.24). This is done in

the following manner, we define the *distorted* projection coordinate[7], $\mathbf{p^d} = [sx^d, sy^d, s]^T$, and the *corrected* projection coordinate, $\mathbf{p^c} = [sx^c, sy^c, s]^T$, such that the transformation from 3D world coordinates Q_i to 2D image coordinates $\mathbf{q_i}$ is given by

$$\begin{aligned} \mathbf{p_i^d} &= \mathbf{A_p} \left[\begin{array}{cc} \mathbf{R} & \mathbf{t} \end{array} \right] Q_i \ , \\ \left[\begin{array}{c} x_i^c \\ y_i^c \end{array} \right] &= \left[\begin{array}{c} x_i^d \\ y_i^d \end{array} \right] (1 + \Delta(r_i)) \ , \\ \mathbf{q_i} &= \mathbf{A_q} \mathbf{p_i^c} \ , \end{aligned} \tag{1.27}$$

Where $\Delta(r_i)$ is the radial distortion, which is a function of the radius

$$r_i = \sqrt{x_i^{d2} + y_i^{d2}} \ . \tag{1.28}$$

As mentioned the $\mathbf{A}$ of (1.24) has been split into $\mathbf{A_p}$ and $\mathbf{A_q}$, where

$$\mathbf{A_p} = \left[\begin{array}{ccc} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{array} \right] \ , \tag{1.29}$$

$$\mathbf{A_q} = \left[\begin{array}{ccc} 1 & \beta & \Delta_x \\ 0 & \alpha & \Delta_y \\ 0 & 0 & 1 \end{array} \right] \ . \tag{1.30}$$

Much of the computations in (1.27) are done in standard *in*homogeneous coordinates, i.e. $[x_i^d y_i^d]$ and $[x_i^c y_i^c]$. The reason is that the distortion is related to actual distances, and as such the formulae would 'break down' if there were an arbitrary scaling parameter present. In is noted that *if* $\Delta(r_i) = 0$ then $\mathbf{p_i^c} = \mathbf{p_i^d}$ and

$$\mathbf{q_i} = \mathbf{A_q} \mathbf{p_i^d} = \mathbf{A_q} \mathbf{A_p} \left[\begin{array}{cc} \mathbf{R} & \mathbf{t} \end{array} \right] Q_i = \mathbf{A} \left[\begin{array}{cc} \mathbf{R} & \mathbf{t} \end{array} \right] Q_i \ ,$$

Equaling (1.21), as expected.

Via the camera model in (1.27), we have extended the pinhole camera model, such that we can incorporate a nonlinear radial distortion $\Delta(r_i)$ as a function of the distance to the optical axis. What needs to be done is to define this radial distortion function $\Delta(r_i)$. The defacto standard way of modelling $\Delta(r_i)$ is as a polynomial in r_i. There is no real physical rationale behind this modelling, and the use of polynomials — i.e. a Taylor expansion — in this manner is a standard 'black box' way of fitting a function. The polynomial used to fit $\Delta(r_i)$ does not include odd terms as well as the zeroth order term, a motivation for this is given in Section 1.5.2.The standard way of expressing the nonlinear radial distortion is thus

$$\Delta(r_i) = k_3 r_i^2 + k_5 r_i^4 + k_7 r_i^6 + \dots \ , \tag{1.31}$$

where $k_3, k_5, k_7, \dots$ are coefficients.

Often times radial distortion is brought into image algorithms, by warping the image such that it appears as it would have *without* radial distortion, see Figure 1.17. For computing this warp, and e.g. for use in some camera optimization algorithms, it is useful to have the inverse radial distortion map (1.27), this is found in [15]. The effects of radial distortion have also had a nomenclature attached to them, namely barrel and pincushion distortion, see Figure 1.19. Lastly, it should be mentioned, that radial distortion is only one non-linear extension of the internal camera parameters — although typically the first non-linear component included — and that others exist e.g. tangential distortion, c.f. e.g. [15].

1.5.1 An Example

Here the example on page 24 is extended by assuming that the image in Figure 1.4.5 had not been warped to correct for radial distortion, but that we are given the parameters for the radial distortion, namely

$$k_3 = -5.1806e-8 \quad , \quad k_5 = 1.4192e-15 \ .$$

[7] I have chosen this nomenclature, because it is technically correct, and because it is not very likely that it will be confused with other terms, c.f. Section 1.7.2.

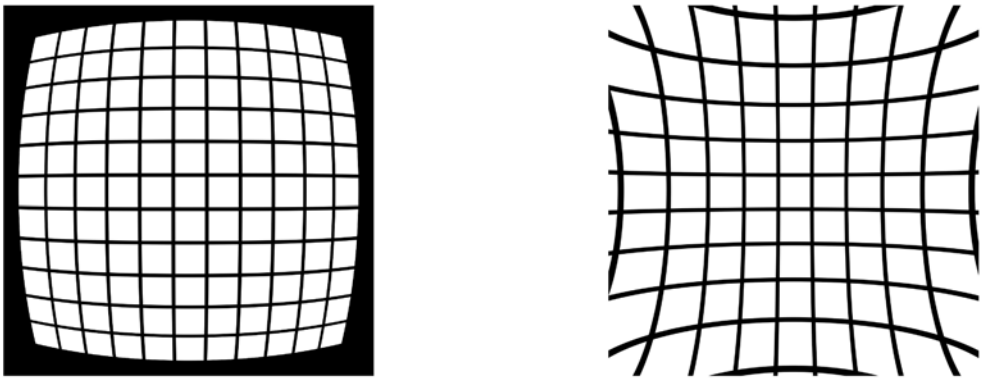

Figure 1.19: The effects of radial distortion on a regular grid, **Left:** Barrel distortion. **Right:** Pincushion distortion.

Then, according to the values supplied on page 24

$$\mathbf{A_p} = \begin{bmatrix} 2774.5 & 0 & 0 \\ 0 & 2774.5 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{A_q} = \begin{bmatrix} 1 & 0 & 806.8 \\ 0 & 1 & 622.6 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{A_p} \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} = \begin{bmatrix} 2743.1 & -1.0 & 416.2 & -6051.6 \\ 2.3 & 2774.5 & -8.4 & 110.7 \\ -0.2 & 0.0 & 1.0 & 0.5 \end{bmatrix}$$

Inserting this into (1.27), we get

$$\begin{aligned} \mathbf{p^d} &= \mathbf{A_p} \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} Q \\ &= \begin{bmatrix} 2743.1 & -1.0 & 416.2 & -6051.6 \\ 2.3 & 2774.5 & -8.4 & 110.7 \\ -0.2 & 0.0 & 1.0 & 0.5 \end{bmatrix} \begin{bmatrix} -1.3540 \\ 0.5631 \\ 8.8734 \\ 1.0000 \end{bmatrix} \\ &= \begin{bmatrix} -6072.9 \\ 1595.3 \\ 9.5 \end{bmatrix} = 9.5 \begin{bmatrix} -640.27 \\ 168.19 \\ 1.0000 \end{bmatrix} \end{aligned}$$

Thus $x^d = -640.27$ and $y^d = 168.19$ and

$$\begin{aligned} r &= \sqrt{x^{d2} + y^{d2}} = \sqrt{-640.27^2 + 168.19^2} = 661.99 \\ \Delta(r) &= k_3 r^2 + k_5 r^4 = (-5.1806e-8) \cdot 4.3823e5 + (1.4192e-15) \cdot 1.9205e11 = -0.0224 \\ \begin{bmatrix} x^c \\ y^c \end{bmatrix} &= \begin{bmatrix} x^d \\ y^d \end{bmatrix} (1 + \Delta(r)) = \begin{bmatrix} -640.27 \\ 168.19 \end{bmatrix} (1 - 0.0224) = \begin{bmatrix} -625.9074 \\ 164.4202 \end{bmatrix} \\ \mathbf{q} &= \mathbf{A_q p^c} = \begin{bmatrix} 1 & 0 & 806.8 \\ 0 & 1 & 622.6 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -625.9074 \\ 164.4202 \\ 1 \end{bmatrix} = \begin{bmatrix} 180.90 \\ 787.03 \\ 1 \end{bmatrix} . \end{aligned}$$

So the inhomogeneous image point corresponding to Q is $(180.90, 787.03)$, with the use of radial distortion. The result is depicted in Figure 1.20. Visually comparing between the results in Figure 1.16 and Figure 1.20 it is hard to see how they differ, hence the difference image is presented in Figure 1.21. To further quantify

the effect of radial distortion, let us consider the numerical deviation of the projection with and without radial distortion (c.f. page 1.4.5)

$$\left\| \begin{bmatrix} 180.90 \\ 787.03 \end{bmatrix} - \begin{bmatrix} 166.5 \\ 790.8 \end{bmatrix} \right\|_2 = \left\| \begin{bmatrix} 14.40 \\ -3.77 \end{bmatrix} \right\|_2 = 14.89 \, pixels \ .$$

Figure 1.20: The same scenario as in Figure 1.16, except that the image has not been warped to account for radial distortion, and that has been incorporated into the camera or projection model.

1.5.2 Motivation for Equation (1.31) *

The polynomial in (1.31) is an expression that (using (1.27))

$$\begin{aligned} \begin{bmatrix} x_i^c \\ y_i^c \end{bmatrix} &= \begin{bmatrix} x_i^d \\ y_i^d \end{bmatrix} (1 + \Delta(r_i)) \\ &= \begin{bmatrix} x_i^d \\ y_i^d \end{bmatrix} (1 + k_3 r_i^2 + k_5 r_i^4 + k_7 r_i^6 + \dots) \\ &= \frac{[x_i^d y_i^d]^T}{r_i} (r + k_3 r_i^3 + k_5 r_i^5 + k_7 r_i^7 + \dots) \ , \end{aligned}$$

Where the reason for dividing $[x_i^d y_i^d]^T$ by r_i is that $\frac{[x_i^d y_i^d]^T}{r_i}$ becomes a direction with unit length. And we see that Δr in Figure 1.18 is given by

$$r + \Delta r = r_i \left(1 + \Delta(r_i)\right) = r_i + k_3 r_i^3 + k_5 r_i^5 + k_7 r_i^7 + \dots \ .$$

This is seen to be the Taylor expansion of an odd fuction[8]. The reason why $r + \Delta r$ should only be an odd function, lies in the fact that the basis for the distortion is a change in the light ray angle α — as illustrated in Figure 1.18. By standard trigonometry it is seen

$$\alpha + \Delta\alpha = \tan\left(\frac{r + \Delta r}{f}\right) \Rightarrow f \arctan\left(\alpha + \Delta\alpha\right) = r + \Delta r \ .$$

[8] An odd function f is one for which $f(x) = -f(-x)$. An even function is one for which $f(x) = f(-x)$. Any function can be expressed completely as a sum of an odd and an even function.

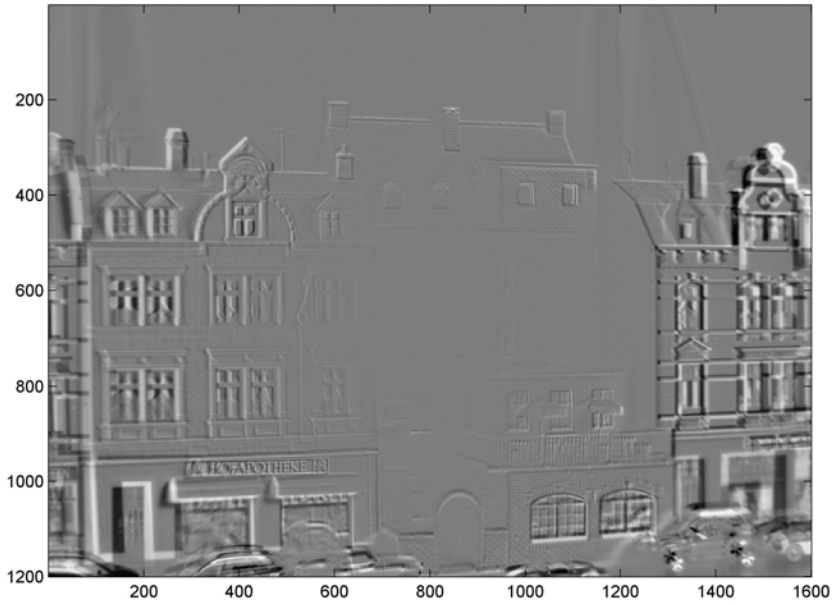

Figure 1.21: The difference image between the images in Figure 1.16 and Figure 1.20. Notice how the deviation of the two images increases away from the center, which is consistent with the effect of radial distortion being largest there.

When the focal length, f, is factored out before the radial distortion is applied, as in (1.27), this becomes

$$\arctan(\alpha + \Delta) = r + \Delta r \ .$$

Since $\arctan$ is an odd function so should $r + \Delta r$ be. The motivation for not including a zeroth term in (1.31) is that this would be equivalent to a constant scaling of the radius r_i. Such a scaling can also be achieved by changing the focal length f. Thus a zeroth order term in (1.31) would be indistinguishable from a different focal length, and would thus be a redundant over-parametrization of the camera model, and thus not done.

1.6 Camera Calibration

After having presented the orthographic and pinhole camera model — the latter with or without non-linear distortion — the question arises how to obtain the parameters of such models, given a camera. This is also known as camera calibration. There are several ways of doing this, some of the most advanced will do it automatically from an image sequence c.f. e.g. [14]. The most standard way of camera calibration is, however, taking images of known 3D objects, typically in the form of known 3D points Q_i. The latter is so common that it is sometimes just referred to as camera calibration.

By taking images of known 3D points, we will get pairs of known 2D-3D points, i.e. $(\mathbf{q_i}, Q_i)$. The task is then finding the parameterized model that best fit these data or 2D-3D point pairs. With the pinhole camera model, this amounts to finding the $\mathbf{P}$ that makes the $\mathbf{P}Q_i$ most equal to $\mathbf{q_i}$. Here "most equal to" typically implies minimizing the 2-norm of the inhomogeneous differences, c.f. Section 2.4.3. Therefore, define the function $\Pi(\mathbf{q_i})$, which takes a homogeneous coordinate and returns a inhomogeneous coordinate, i.e.

$$\begin{bmatrix} x \\ y \end{bmatrix} = \Pi\left(\begin{bmatrix} sx \\ sy \\ s \end{bmatrix}\right) = \begin{bmatrix} \frac{sx}{s} \\ \frac{sy}{s} \end{bmatrix} \ .$$

The camera calibration problem thus becomes

$$\min_{\mathbf{P}} \sum_i ||\Pi(q_i) - \Pi(\mathbf{P}Q_i)||_2^2 \ . \tag{1.32}$$

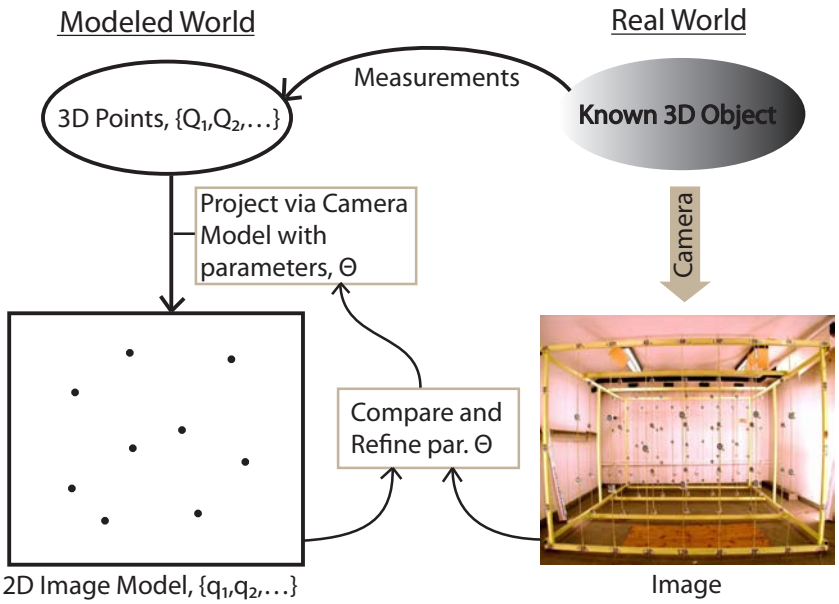

Figure 1.22: The camera calibration procedure works by modelling the 3D world via 3D point measurements. These measured 3D points are the projected into the model image plane via the camera model. These 2D model measurements are compared to the real image of the known 3D object. Based on this comparison the camera model parameters, Θ , are refined iteratively, such that the model and the real image fit as well as possible.

This is a non-linear optimization problem in the parameters of the camera model, here the 12 parameters of $\mathbf{P}$. As with radial distortion, c.f. Section 1.5, we project to inhomogeneous, because we need to work in actual distances. The camera calibration process is illustrated in Figure 1.22. In setting up, or choosing, the 3D camera calibration object, it in necessary that it spans the 3D space — i.e. that all the points do not lie in a plane, else (1.32) becomes ill-posed.

There are several free online software packages for doing camera calibration, e.g. an implementation of the method in [15] is available from the authors homepage. Another software package, available from `http://www.vision.caltech.edu/bouguetj/calib_doc/`, implements a more convenient method of camera calibration, since the calibration object is easier to come by. It consist of taking images of a checker board patter from several angles.

1.7 End Notes

Here a few extra notes will be made on camera modelling, by briefly touching on what is modelled in other contexts, and on what notation other authors use. Furthermore, the pinhole camera model, being the most central, is summarized in Table 1.3.

1.7.1 Other Properties to Model*

As mentioned in Section 1.2, a model in general only captures part of a phenomena, here the imaging process of a camera. The camera models presented here thus only captures a subset of this imaging process, albeit central ones. Here a few other properties that are sometimes modelled are mentioned briefly. A property of the optics, arising from a larger then infinitesimal aperture or pinhole is *depth of field*. The effect of a — limited – depth of field is that only objects at a certain distance interval are in focus or 'sharp', see Figure 1.23-Right. Apart from depth of field limitation being a nuisance, and used as a creative photo option, it has also been used to infer the depth of objects by varying the depth of field and noting when objects were in focus, c.f. e.g. [7]. Along side the geometric camera properties, a lot of effort has also been used on modelling the color or chromatic camera properties, and is a vast field in itself. Such chromatic camera models can e.g. be calibrated via a color card as depicted in Figure 1.23-Left.


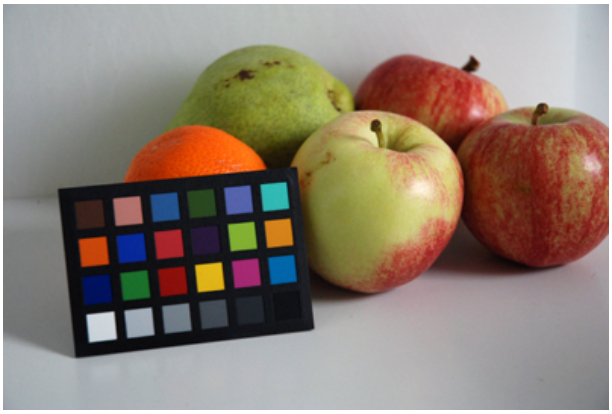

Figure 1.23: **Left:** Example of depth of field of a camera, note that it is only the flower and a couple of straws that are in focus. **Right:** Example of a color calibration card. This colors of the squares in the card a very well know. As such the image can be chromatically calibrated.

1.7.2 Briefly on Notation

Our world is not a perfectly and systemized place. Just as Esperanto[9] has not become the language of all humans, enabling unhindered universal communication, the notation of camera models have not either. In fact, the proliferation of camera geometry use in a vast number of fields, has spawned serval different notations. Apart from the specific names given to the entities, the notation also varies in how many different terms the camera model is split into. A further source of confusion is the definition of the coordinate system. As an example, in computer graphics the origo of the image is in the *lower* left corner. This is in contrast to the upper left corner typically used in computer vision. Another example is that the image x-axis and y axis are sometimes swaped by multiplying $\mathbf{A}$, and thus $\mathbf{P}$ by

$$\begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} .$$

Thus, when stating to use a new frame work using camera geometry, e.g. a software package, it is thus important to check the notation. It is, however, worth noting that it is the same underlying models in play.

[9] Esperanto was introduced in 1887

SUMMARY OF THE PINHOLE CAMERA MODEL

The output and input are 3D world points, Q_i, and 2D image point, $\mathbf{q_i}$.
Std. Pinhole camera model, (1.21) & (1.24)

$$\mathbf{q_i} = \mathbf{P}Q_i$$

with

$$\mathbf{P} = \mathbf{A}\begin{bmatrix}\mathbf{R} & \mathbf{t}\end{bmatrix} \quad , \quad \mathbf{A} = \begin{bmatrix} f & f\beta & \Delta x \\ 0 & \alpha f & \Delta y \\ 0 & 0 & 1 \end{bmatrix}$$

and

$\mathbf{R}$	Rotation	,	$\mathbf{t}$	Translation
f	Focal length	,	$\Delta x, \Delta y$	Coord. of Optical Axis
α, β	Affine image def.			

Pinhole camera model with radial distortion, (1.27)

$$\begin{aligned} \mathbf{p_i^d} &= \mathbf{A_p}\begin{bmatrix}\mathbf{R} & \mathbf{t}\end{bmatrix} Q_i \ , \\ \begin{bmatrix} x_i^c \\ y_i^c \end{bmatrix} &= \begin{bmatrix} x_i^d \\ y_i^d \end{bmatrix} (1 + \Delta(r_i)) \ , \\ \mathbf{q_i} &= \mathbf{A_q}\mathbf{p_i^c} \ , \end{aligned}$$

Where

$$\Delta(r_i) = k_3 r_i^2 + k_5 r_i^4 + k_7 r_i^6 + \ldots$$

is the radial distortion, and a function of the radius

$$r_i = \sqrt{x_i^{d2} + y_i^{d2}} \ ,$$

and

$$\mathbf{A_p} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad , \quad \mathbf{A_q} = \begin{bmatrix} 1 & \beta & \Delta_x \\ 0 & \alpha & \Delta_y \\ 0 & 0 & 1 \end{bmatrix} \ .$$

Where $\mathbf{p^d} = [sx^d, sy^d, s]^T$ are *distorted* projection coordinate, and $\mathbf{p^c} = [sx^c, sy^c, s]^T$ are *corrected* projection coordinate. Note that$[x_i^d y_i^d]$ and $[x_i^c y_i^c]$ are in *in*homogeneous coordinates.

Table 1.3: Summary of the Pinhole Camera Model

Chapter 2

Geometric Inference from Cameras - Multiple View Geometry

In Chapter 1 the focus was on modelling how a given 3D world point would project to a 2D image point via a known camera. Here the inverse problem will be considered, i.e. what does 2D image observations tell about the 3D world and cameras. In fact 3D geometric inference is one of the main uses of camera models or camera geometry. Another related matter – also covered here – is that camera models also provide constraints between images viewing the same 3D object. Although this 3D object might be unknown, both images still have to be consistent with it, thus providing constraint. Lastly, multiple view geometry is a vast field and the amount of results are staggering, as such only a fraction of this material is covered here, and the interested reader is referred to [14] for a more in depth treatment.

2.1 What does an Image Point Tell Us About 3D?

In this chapter we will mainly be working with image features in the form of image points. There are naturally a whole variety of possible image features, e.g. lines, curves, ellipsoids etc. but points are the simplest to work with. The theory developed for points will also generalize to the majority of other features, and will give the basic understanding of 3D estimation.

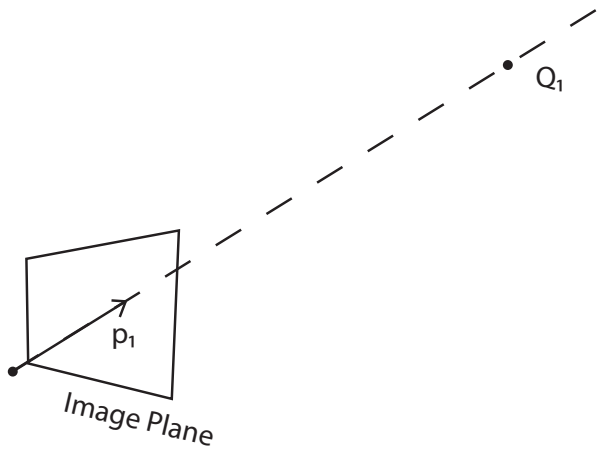

Figure 2.1: The back-projection of the point $\mathbf{Q_1}$ is the dotted line in the direction of the vector $\mathbf{p_1}$. Thus the 3D point Q_1, which projects to $\mathbf{q_1}$, must lie on this back-projected line.

Dealing with 2D image points, the question arises; what does a point tell us about 3D? Assuming that we are given a camera described by the pinhole camera model, (1.21) — and we know this model — a 2D image point tells us that the 3D point, it is a projection of, is located on a given line, see Figure 2.1. To derive this mathematically, denote by $\mathbf{p_i}$ the image coordinate before the internal parameters are applied[1], i.e.

$$\mathbf{p_i} = \mathbf{A}^{-1}\mathbf{q_i} \ .$$

[1] In this chapter the $\mathbf{p_i}$ deviate from the $\mathbf{p_i}$ in Chapter 1 by a factor of f.

Then by taking outset in the pinhole model

$$\begin{aligned} \mathbf{q_i} &= \mathbf{A}\left[\ \mathbf{Rt}\ \right] Q_i \Rightarrow \\ \mathbf{p_i} &= \mathbf{A^{-1}q_i} = \left[\ \mathbf{Rt}\ \right] Q_i \Rightarrow \\ \alpha\mathbf{p_i} &= \mathbf{A^{-1}q_i} = \left[\ \mathbf{Rt}\ \right] Q_i \Rightarrow \\ \alpha\mathbf{R^T p_i} - \mathbf{t} &= \tilde{Q}_i \ , \end{aligned}$$

where $\tilde{Q}_i$ is *in*homogeneous 3D coordinate corresponding to Q_i, and α is a free scalar. The reason we can multiply by α like we do, is that $\mathbf{p_i}$ is a homogeneous coordinates. It is thus seen that

$$\alpha\mathbf{R^T p_i} - \mathbf{t} = \tilde{Q}_i \ , \tag{2.1}$$

which is equal to a line through $\mathbf{t}$ and with direction $\mathbf{R^T p_i}$, as proposed. In other words an image point *back-projects* to a 3D line. So *if* the camera coordinate system was equal to the global coordinate system, i.e. $\mathbf{R} = \mathbf{I}$ and $\mathbf{t} = 0$, then $\mathbf{p_i}$ would be the direction the point Q_i was in - from the origo of the coordinate system.

The physical explanation for this is, that a camera basically records light that hits its sensor after passing through the lens. Prior to that light is assumed to travel in a straight line[2]. The camera thus records that a bundle of light has hit it from a given direction, $\mathbf{p_i}$, but has no information on how far the light has travelled. Thus the distance to the object omitting or reflecting the light is unknown. This is the same as saying that relative to the camera coordinate system, we do not know the depth of an object, as illustrated in Figure 1.9. These line constraints on 3D points, are the basic building blocks of camera based 3D estimation, and used in the remainder of this chapter.

2.1.1 Back-Projection of an Image Line

To illustrate how the analysis of image points, and because the result is needed later, we will now consider what a straight image line back-projects to, i.e. what constraint an image line poses to the 3D geometry 'creating' it. The answer is a plane. This can be seen since each point on the image line constraints the 3D geometry to a line. Each of these constraining 3D lines go trough the optical center of the camera, and the straight line corresponding to the 2d image line, see Figure 2.2. The collection of all these lines thus constitutes a plane, and all 3D points on it.

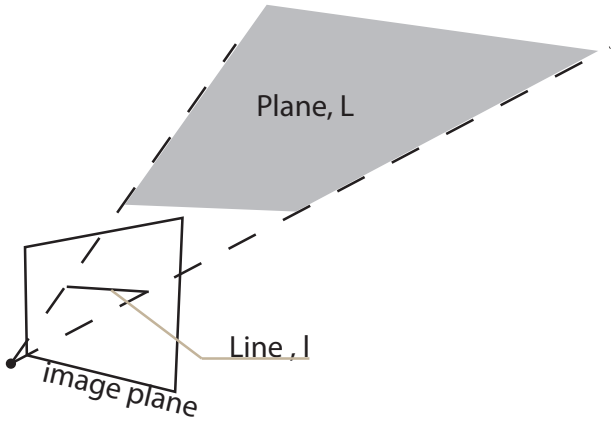

Figure 2.2: The image of a line back-projects to a plane. All 3D points projecting to this 2D image line must thus be on this back-projected plane.

An alternative version of this argument will be given here using the theory of homogeneous coordinates, see Section 1.1. This argument is hopefully more constructive and straight forward. The points, $\mathbf{q_i}$ on the 2D image line are given by

$$\mathbf{l^T q_i} = 0 \ , \tag{2.2}$$

[2]It is herby assumed that the camera and the objects it is photographing are in the same medium, e.g. air, and that the light does not go through any transparent project like water. These phenomena can be modelled but are outside the scope of this text.

where $\mathbf{l}$ denotes the line coefficients. These points are the projection of a set of 3D points Q_i, by use of the pinhole model (1.21), which combined with (2.2) gives

$$\mathbf{l^T q_i} = \mathbf{l^T P} Q_i = L^T Q_i = 0 \quad , \quad L^T = \mathbf{l^T P} \ , \tag{2.3}$$

where L is a 4 vector and the coefficients of the 3D plane that (2.3) is the equation of. In other words the points projecting to the line, $\mathbf{l}$ are constraint to a plane with the equation $L = \mathbf{P^T l}$.

2.2 Epipolar Geometry

When describing how to infer 3D information about the world from images thereof, we will first consider the constraint two images viewing the same scene poses to the images themselves. This is directly related on the cameras *relative orientation* to each other and is also referred to as epipolar geometry.

To derive the epipolar eometry, assume that two cameras are viewing the same 3D point, Q, with unknown coordinates. Assume also that the coordinate system of the first camera is equal to the global coordinate system, i.e.

$$\mathbf{P_1} = \mathbf{A_1} \begin{bmatrix} \mathbf{I} & \mathbf{0} \end{bmatrix} \quad , \quad \mathbf{P_2} = \mathbf{A_2} \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} \ .$$

This is done without loss of generality and is illustrated in Figure 2.3. The coincidence of the first cameras coordinate system and the global coordinate system is made for notational convenience. If this is not the case the coordinate systems can be transformed accordingly, c.f. Appendix A. Denote by $\mathbf{q_1}$ and $\mathbf{q_2}$ the projections of the 3D point Q in the two cameras respectively, and also $\mathbf{p_1} = \mathbf{A_1^{-1}} \mathbf{q_1}$ and $\mathbf{p_2} = \mathbf{A_2^{-1}} \mathbf{q_2}$.

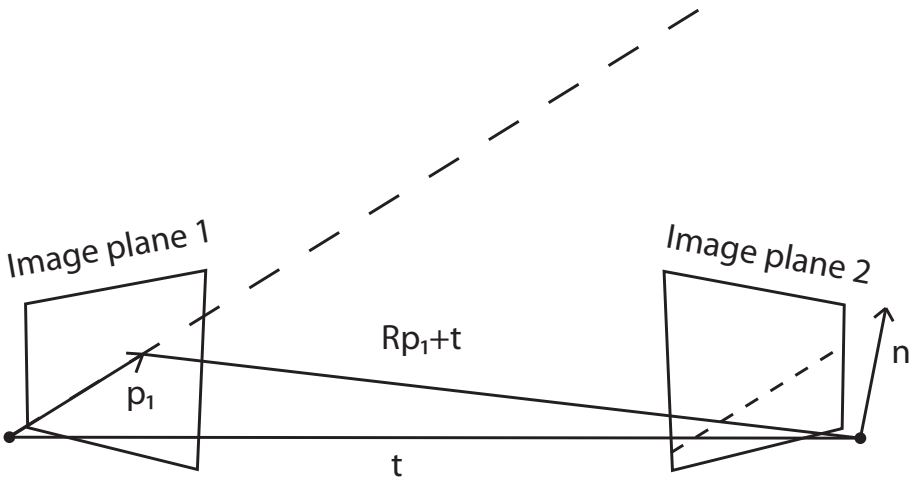

Figure 2.3: The two camera centers and the point $\mathbf{p_1}$ define a plane that also includes the back-projection of point $\mathbf{q_1}$. The normal of this epipolar plane is n, and the intersection of this plane ad camera two's image plane is the projection of the back-projected line of $\mathbf{q_1}$ into camera two.

We wish to investigate what constraints $\mathbf{q_1}$ poses on $\mathbf{q_2}$, and thus the constraints on possible images of the same thing, even though the 3D structure is unknown. The general idea is, that the back-projected 3D line of $\mathbf{q1}$ constrains the position of Q. This virtual back-projected 3D line can then be projected into camera two giving a virtual 2D line in that image.This is illustrated in Figure 2.3. The projection of Q in camera two, equalling $\mathbf{q_2}$, must then be on this virtual 2D line. This is the so called *epipolar constraint*. Also, the line $\mathbf{q_2}$ is constraint to is called an *epipolar line*.

To derive this more formally, referring to Figure 2.3, note that the centers of cameras one and two and $\mathbf{p_1}$ lie on a plane. This is the *epipolar plane*. The intersection of this epipolar plane and the image plane of camera two is the epipolar line in image two. The epipolar plane is spanned by the vector between the two camera centers and the vector between $\mathbf{p_1}$ and the center of camera two. In the coordinate system of camera two (where the center of camera two is $[0,0,0]^T$) these vector have the coordinates, as denoted in Figure 2.3

$$\mathbf{R} \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + \mathbf{t} = \mathbf{t} \quad \textit{and} \quad \mathbf{R p_1} + \mathbf{t} \ .$$

Which is a transformation from the global coordinate system of camera center one, $[0,0,0]^T$, and $\mathbf{p_1}$, to the coordinate system of camera two. The normal of the epipolar plane $\mathbf{n}$ is then given by the cross product of the two vectors, i.e.

$$\begin{aligned}\mathbf{n} &= \mathbf{t} \times (\mathbf{R}\mathbf{p_1} + \mathbf{t}) \\ &= \mathbf{t} \times \mathbf{R}\mathbf{p_1} + \mathbf{t} \times \mathbf{t} \\ &= [\mathbf{t}]_\times \mathbf{R}\mathbf{p_1} + 0 \\ &= [\mathbf{t}]_\times \mathbf{R}\mathbf{p_1} \ . \end{aligned} \tag{2.4}$$

In (2.4) we express the cross product with $\mathbf{t}$ as an operator ($[\mathbf{t}]_\times$) , see Appendix A.5. We also set $\mathbf{t} \times \mathbf{t} = 0$ since the cross product between a vector and itself is zero. This normal $\mathbf{n}$ is expressed in the coordinate system of camera two. In this coordinate system the epipolar plane also goes through the camera center of camera two, namely origo $[0,0,0]^T$. Thus any point, $\mathbf{p}$, on this plane is given by

$$\mathbf{p^T n} = 0 \ . \tag{2.5}$$

This will in particular hold for $\mathbf{p_2}$. Combining (2.4) and (2.5) yeilds

$$\mathbf{p_2^T}[\mathbf{t}]_\times \mathbf{R}\mathbf{p_1} = 0 \ . \tag{2.6}$$

This relationship is so fundamental that we name

$$\mathbf{E} = [\mathbf{t}]_\times \mathbf{R} \ , \tag{2.7}$$

the *essential matrix*, and thus

$$\mathbf{p_2^T E p_1} = 0 \ . \tag{2.8}$$

Relating this to image points $\mathbf{q_1}$ and $\mathbf{q_2}$, we have that $\mathbf{p_1} = \mathbf{A_1^{-1}}\mathbf{q_1}$ and $\mathbf{p_2} = \mathbf{A_2^{-1}}\mathbf{q_2}$, which combined with (2.6) gives

$$\begin{aligned} 0 &= \mathbf{p_2^T}[\mathbf{t}]_\times \mathbf{R}\mathbf{p_1} \\ &= \left(\mathbf{A_2^{-1}}\mathbf{q_2}\right)^\mathbf{T} [\mathbf{t}]_\times \mathbf{R}\mathbf{A_1^{-1}}\mathbf{q_1} \\ &= \mathbf{q_2^T}\mathbf{A_2^{-T}}[\mathbf{t}]_\times \mathbf{R}\mathbf{A_1^{-1}}\mathbf{q_1} \ , \end{aligned} \tag{2.9}$$

where

$$\mathbf{F} = \mathbf{A_2^{-T}}[\mathbf{t}]_\times \mathbf{R}\mathbf{A_1^{-1}} \ , \tag{2.10}$$

is called the *fundamental matrix*, and thus

$$\mathbf{q_2^T F q_1} = 0 \ . \tag{2.11}$$

Here, with some abuse of notation, $A_2^{-T} = \left(A_2^{-1}\right)^T$.

To see the relation of (2.11) to lines, define $\mathbf{l_2} = \mathbf{F}\mathbf{q_1}$, making (2.11) equivalent to

$$\mathbf{q_2^T l_2} = 0 \ .$$

This is seen to be a line in image two, c.f. Section 1.1. Thus if $\mathbf{F}$ and $\mathbf{q_1}$ are known so is $\mathbf{l_2}$ and we have the equation for the epipolar line, as depicted in Figure 2.3. Likewise, we can define $\mathbf{l_1^T} = \mathbf{q_2^T F}$ such that

$$\mathbf{l_1^T q_1} = 0 \ ,$$

defining the epipolar line in image one, for given $\mathbf{F}$ and $\mathbf{q_2}$.

It should be noted that both the essential and fundamental matrices do not include any terms relating to the individual observations, i.e. $\mathbf{p1}$ and $\mathbf{p_2}$. They are thus general for the specific camera setup, and do not depend on a particular observation.

Figure 2.4: An image pair, with image one to the left and image two to the right. A point, $\mathbf{q_1}$, is annotated in image one (blue dot). The corresponding epipolar line, $\mathbf{l_2}$, and 2D point, $\mathbf{p_2}$ is annotated in image two. This is done by the black line and blue dot respectively.

2.2.1 An Example

As an example consider the images in Figure 2.4. The camera matrices of the two cameras are

$$\mathbf{P_1} = \mathbf{A_1}\begin{bmatrix}\mathbf{I} & \mathbf{0}\end{bmatrix} = \begin{bmatrix}3117.5 & 0 & 1501.9\\ 0 & 3117.5 & 984.8\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\end{bmatrix},$$

$$\mathbf{P_2} = \mathbf{A_2}\begin{bmatrix}\mathbf{R} & \mathbf{t}\end{bmatrix} = \begin{bmatrix}3117.5 & 0 & 1501.9\\ 0 & 3117.5 & 984.8\\ 0 & 0 & 1\end{bmatrix}\left[\begin{bmatrix}0.9885 & -0.0388 & -0.1459\\ 0.0514 & 0.9952 & 0.0836\\ 0.1419 & -0.0902 & 0.9858\end{bmatrix}\begin{bmatrix}3.5154\\ -0.2712\\ -1.3704\end{bmatrix}\right].$$

The fundamental matrix is thus given by, (2.10)

$$\begin{aligned}\mathbf{F} &= \mathbf{A_2^{-T}}[\mathbf{t}]_\times\mathbf{R}\mathbf{A_1^{-1}}\\ &= \left(\begin{bmatrix}3117.5 & 0 & 1501.9\\ 0 & 3117.5 & 984.8\\ 0 & 0 & 1\end{bmatrix}^{-1}\right)^T\begin{bmatrix}0 & 1.3704 & -0.2712\\ -1.3704 & 0 & -3.5154\\ 0.2712 & 3.5154 & 0\end{bmatrix}\cdot\\ &\quad\begin{bmatrix}0.9885 & -0.0388 & -0.1459\\ 0.0514 & 0.9952 & 0.0836\\ 0.1419 & -0.0902 & 0.9858\end{bmatrix}\begin{bmatrix}3117.5 & 0 & 1501.9\\ 0 & 3117.5 & 984.8\\ 0 & 0 & 1\end{bmatrix}^{-1}\\ &\approx \begin{bmatrix}0.0000 & 0.0000 & -0.0002\\ -0.0000 & 0.0000 & -0.0008\\ 0.0003 & 0.0009 & 0.0151\end{bmatrix}.\end{aligned}$$

Where $\approx$ is used instead of $=$ due to numerical rounding. The annotated point in image one, corresponding to the left image in Figure 2.4, is given by

$$\mathbf{q_1} = \begin{bmatrix}1260\\ 100\\ 1\end{bmatrix}.$$

The corresponding epipolar line in image two (right image in Figure 2.4) is then given by

$$\mathbf{l_2} = \mathbf{F}\mathbf{q_1} = \begin{bmatrix}0.0000 & 0.0000 & -0.0002\\ -0.0000 & 0.0000 & -0.0008\\ 0.0003 & 0.0009 & 0.0151\end{bmatrix}\begin{bmatrix}1260\\ 100\\ 1\end{bmatrix} = \begin{bmatrix}-0.0002\\ -0.0010\\ 0.5136\end{bmatrix}.$$

This epipolar line is depicted in Figure 2.4 right. The point, $\mathbf{q_2}$, corresponding to $\mathbf{q_1}$ is given by

$$\mathbf{q_2} = \begin{bmatrix}1330\\ 269.8\\ 1\end{bmatrix},$$

which is seen to lie on the epipolar line $\mathbf{l_2}$, since

$$\mathbf{q_2^T l_2} = \begin{bmatrix} 1330 \\ 269.8 \\ 1 \end{bmatrix}^T \begin{bmatrix} -0.0002 \\ -0.0010 \\ 0.5136 \end{bmatrix} \approx 0 \ .$$

2.2.2 Some Additional Properties of Epipolar Geometry *

Here some additional properties of the fundamental and essential matrices will be covered. This is by no means a complete treatment of the subject, and for a more complete presentation the interested reader is referred to [14].

Epipoles*

Considering again the derivation of the fundamental matrix, where it is seen that all epipolar planes go through the camera centers of the two cameras. Thus all epipolar lines in camera two must go through the projection of camera center one, albeit it is often outside the physical boundary of the image. The similar thing holds for image one, see Figure 2.5 and Figure 2.6. The projection of these camera centers are called the *epipoles*.

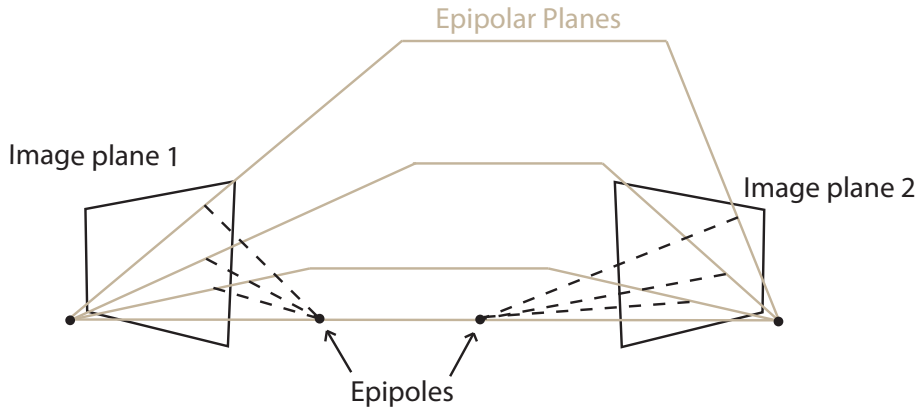

Figure 2.5: Since all the epipolar planes intersect in a common line – i.e. line connecting the two camera centers – all the epipolar lines will intersect in a common point, called the epipole. This epipole is often located outside the physical image.

More formally consider the projection of camera center one into image two, which is given by:

$$\mathbf{e_2} = \mathbf{A_2}\left(\mathbf{R}\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + \mathbf{t}\right) = \mathbf{A_2 t} \ ,$$

Inserting this into (2.9), gives

$$\begin{aligned} \mathbf{e_2^T F p_1} &= \mathbf{t^T A_2^T A^{-T_2} [t]_\times R A_1^{-1} q_1} \\ &= \mathbf{t^T [t]_\times R A_1^{-1} q_1} \\ &= \mathbf{0 R A_1^{-1} q_1} \\ &= \mathbf{0 q_1} \end{aligned} \tag{2.12}$$

Here it is used that

$$\mathbf{t^T}[\mathbf{t}]_\times = \left([\mathbf{t}]_\times^T \mathbf{t}\right)^T = -\left([\mathbf{t}]_\times \mathbf{t}\right)^T - \left(\mathbf{t}_\times \mathbf{t}\right)^T = \mathbf{0} \ ,$$

Since the cross product between a vector and itself is zero. Equation (2.12) states that the epipole of camera two will fulfil the epipolar geometry, i.e. (2.9), for all points in image one. Thus all epipolar lines in image two goes through $\mathbf{e_2}$. The symetric property hold for the epipole in image one, $\mathbf{e_1}$

Figure 2.6: A collection of epipolar lines in the right image one for each point in the left image. It is seen that all the epipolar lines form a bundle intersecting in the same point, i.e. the epipole.

Degrees of Freedom of F*

The degrees of freedom (dof.) of the fundamental and essential matrices, which is equivalent with the number of constraints needed to estimate them, is now considered. The fundamental matrix has $9 = 3 \cdot 3$ elements, so at the outset it has nine dof. The fundamental matrix is, however, indifferent to scaling, as seen from (2.11), where both sides can be multiplied by a scalar still giving the same result. A more subtle property of the fundamental matrix is that it has rank two and that it's determinant must be 0, i.e.

$$\det(\mathbf{F}) = 0 \ .$$

This is seen by $[\mathbf{t}]_\times$ having rank two (it cannot have full rank since $[\mathbf{t}]_\times \mathbf{t} = \mathbf{0}$, implying that it has a nontrivial null space.) This rank constraint eliminates an additional dof. Thus the fundamental matrix has $7 = 9 - 2$ dof.

The Essential Matrix*

Firstly, note that the essential matrix can be viewed as a special case of the fundamental matrix with

$$\mathbf{A_1} = \mathbf{A_2} = \mathbf{I} \ .$$

Secondly, the singular values of the essential matrix are $\{s, s, 0\}$, where s is some scalar. Therefor it is a rank two matrix with the added constraint that the two non-zero singular values are equal. For a motivation of this c.f. [14], where it will also be explained why the essential matrix has 5 dof.

2.3 Homographies for Two View Geometry

The term homography covers a class of geometric transformations, which have a wide use in computer vision, computer graphics and other places where view geometry is used. Mathematically it can be described by a (full rank) 3 by 3 matrix, $\mathbf{H}$, that maps between homogeneous coordinates $\mathbf{q_1}$ and $\mathbf{q_2}$, as follows:

$$\mathbf{q_1} = \mathbf{H}\mathbf{q_2} \ . \tag{2.13}$$

Note that any full rank 3 by 3 matrix describes a homography, and hence the inverse of a homograpy is also a homograpy, i.e.

$$\mathbf{q_2} = \mathbf{H}^{-1}\mathbf{q_1} \ . \tag{2.14}$$

For a more thorough description of the theory of homographies the reader is referred to [14].

In the context of this chapter homographies form a good model for describing two view geometry in the cases where a planar surface is viewed and/or there is no motion between the views (i.e. $\mathbf{t} = 0$ in (1.21)). In the latter case $[\mathbf{t}]_\times = \mathbf{0}$, and the essential and fundamental matrices, c.f. (2.7) and (2.10), are zero, making these models useless. The fundamental matrix cannot be estimated from an image of a pure planar structure either. Thus the homography is in some sense a fall back solution for the special cases where the epipolar geometry fails.

The fact that the homography describes the viewing of a plane also makes it very used for texture mapping, e.g. in computer graphics. The reason being that the triangular mesh is by far the most common 3D surface representation. The individual faces of such a mesh are planes.

2.3.1 Photographing a Plane

A plane can be described by a point in that plane C and two linear independent vectors in that plane A and B, see Figure 2.7. This also serves as a local coordinate system for that plane such that any point Q in that plane can be described as

$$Q = aA + bB + C = \begin{bmatrix} A & B & C \end{bmatrix} \begin{bmatrix} a \\ b \\ 1 \end{bmatrix} \tag{2.15}$$

Where (a, b) is the local coordinate of Q, and $[A, B, C]$ is a 3×3 matrix. Here denoted by $\mathbf{q_p} = [a, b, 1]^T$ the homogeneous coordinate of (a, b). Assuming that the plane in question is viewed by a camera described by the pinhole camera model, c.f. (1.21), the image $\mathbf{q_1}$ of Q is given by

$$\mathbf{q_1} = \mathbf{P}Q = \mathbf{P} \begin{bmatrix} A & B & C \\ 0 & 0 & 1 \end{bmatrix} \mathbf{q_p} \ . \tag{2.16}$$

Since $\mathbf{P}$ is a 3×4 matrix

$$\mathbf{H} = \mathbf{P} \begin{bmatrix} A & B & C \\ 0 & 0 & 1 \end{bmatrix} \ ,$$

Is a 3×3 matrix representing the homography that maps from $\mathbf{q_p}$ to $\mathbf{q_1}$. Combining this with (2.16) gives

$$\mathbf{q_1} = \mathbf{H}\mathbf{q_p} \ , \tag{2.17}$$

Denoting the homograpic map, which we aimed to derive. It can be shown, c.f. [14], that if the camera center is *not* in the plane in question, then $\mathbf{H}$ will have full rank.

2.3.2 Photogrammetry in a Plane

In many 3D inference problem, especially in surveillance, it is known that something or some one is located on the ground, and we want to know where. Frequently this ground is well approximated by a plane, and thus homographies are useful. As an example consider the chess board in Figure 2.8. Here the homography between the image of the chess board (right image) and the chess board it self is given by

$$\mathbf{H} = \begin{bmatrix} 0.0191 & -0.0302 & 5.7963 \\ 0.0203 & 0.0484 & -13.1140 \\ -0.0000 & 0.0026 & 1.0000 \end{bmatrix} \ .$$

The coordinate system used for the chess board is one where the axis are aligned with the board with, $(0, 0)$ is at one corner and one square equals one unit of measurement. This will allow us to determine where a piece is on the chess board from an image coordinate. Consider the the depicted chess piece, which has image coordinates $(404, 255)$, thus the position of this piece on the board is given by:

$$\mathbf{q_1} = \mathbf{H} \begin{bmatrix} 404 \\ 255 \\ 1 \end{bmatrix} = \begin{bmatrix} 5.8157 \\ 7.4609 \\ 1.6575 \end{bmatrix} \approx \begin{bmatrix} 3.5087 \\ 4.5013 \\ 1.0000 \end{bmatrix} \ ,$$

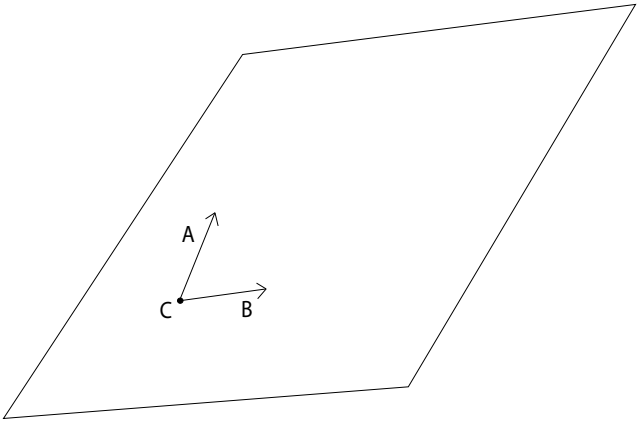

Figure 2.7: All point on a plane in 3D, can be represented by a point in the plane, C, plus a linear combination of two vectors in the plane, A and B. Here the linear combination of A and B consists of a local coordinate system.

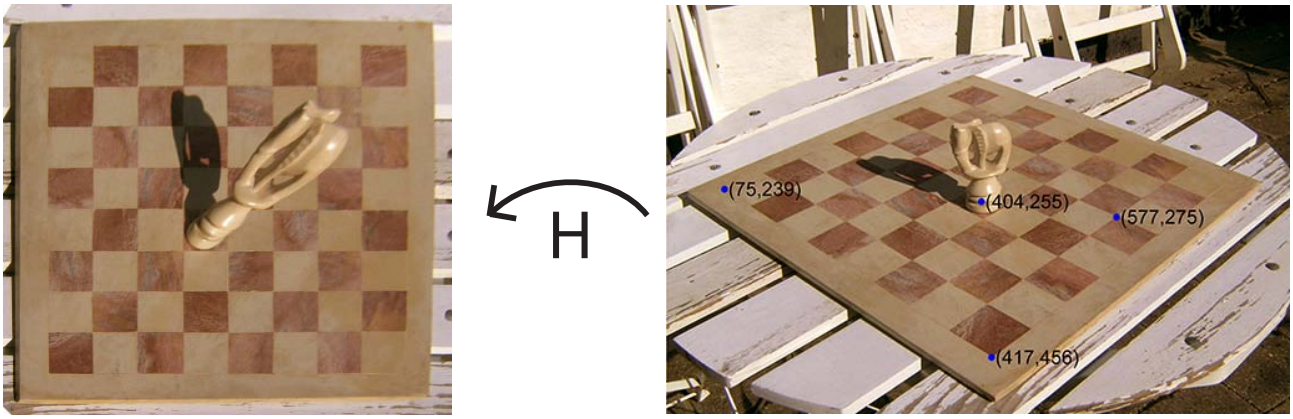

Figure 2.8: An image of a chess board to the right, and a warped version of it, via the homography $\mathbf{H}$, to the left. The left image is a pseudo view of how the chess board would look from straigh above, or a so called fronto parallel view.

Where $\approx$ here indicates homogeneous equivalent. Thus the chess piece is located at ca. $(3.5, 4.5)$, which is what is seen in Figure 2.8-Left, when the origo – $(0, 0)$ – is at the upper left corner. To continue the example consider the three other points annotated in Figure 2.8-Right:

$$
\begin{aligned}
\mathbf{q_2} &= \mathbf{H}\begin{bmatrix} 75 \\ 239 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.0164 \\ -0.0093 \\ 1.6244 \end{bmatrix} \approx \begin{bmatrix} 0.0101 \\ -0.0057 \\ 1.0000 \end{bmatrix}, \\
\mathbf{q_3} &= \mathbf{H}\begin{bmatrix} 417 \\ 456 \\ 1 \end{bmatrix} = \begin{bmatrix} -0.0013 \\ 17.4630 \\ 2.1840 \end{bmatrix} \approx \begin{bmatrix} -0.0006 \\ 7.9958 \\ 1.0000 \end{bmatrix}, \\
\mathbf{q_4} &= \mathbf{H}\begin{bmatrix} 577 \\ 275 \\ 1 \end{bmatrix} = \begin{bmatrix} 8.5156 \\ 11.9504 \\ 1.7053 \end{bmatrix} \approx \begin{bmatrix} 4.9936 \\ 7.0078 \\ 1.0000 \end{bmatrix}.
\end{aligned}
$$

This homography can also be used to map the image to give a pseudo view of how the image would look, if the plane was viewed in a fronto parallel manner. This is actually how the image in Figure 2.8-Left is produced[3]. This is done via an image warp, where the particular homography dictates how the individual pixels should be mapped. In this case it is noted that the chess piece now covers several squares, which is not consistent with how it would look if the real scene were seen from above. This is the reason for the 'pseudo' in 'pseudo view'. The reason is that the chess piece is *not* in the plane and does thus not fit the model used. Another view on the mater is, that the chess piece covers the exact same part of the chess board as in Figure 2.8-Right, which is image that is warped.

2.3.3 Two Cameras Viewing a Plane

If two cameras are viewing a plane, then the relationship between the two images taken is also a homography. To see this denote, as above, by Q a 3D point in the plane. Let also Q have coordinates $\mathbf{q_p}$, in some arbitrary coordinate system in this plane. Assume that we have a pair of corresponding points $\mathbf{q_1}$ and $\mathbf{q_2}$ in the two images respectively and that they are depictions of Q. Then two homographies, $\mathbf{H_1}$ and $\mathbf{H_2}$, exist such that

$$\mathbf{q_1} = \mathbf{H_1 q_p} \quad , \quad \mathbf{q_2} = \mathbf{H_2 q_p} \Rightarrow \mathbf{q_p} = \mathbf{H_2^{-1} q_2} \ .$$

This implies that

$$\mathbf{q_1} = \mathbf{H_1 q_p} = \mathbf{H_1 H_2^{-1} q_2} \ .$$

Since $\mathbf{H_1}$ and $\mathbf{H_2}$ are both assumed full rank 3×3 matrices $\mathbf{H} = \mathbf{H_1 H_2^{-1}}$ is also a full rank 3×3 matrix defining a homography. Therefor, the transfer from $\mathbf{q_1}$ to $\mathbf{q_2}$ is given by

$$\mathbf{q_1} = \mathbf{H q_2} \ , \tag{2.18}$$

where $\mathbf{H}$ is the sought after homography.

2.3.4 Two View Geometry Without a Baseline

The *baseline* between two cameras is the distance between their respective camera centers. If this baseline becomes zero, i.e. the camera centers are co-located, then the $\mathbf{t} = \mathbf{0}$ in (2.7) and (2.10), making these models meaningless. In this special case, the homography can be used instead. Without loss of generality we can assume that the coordinate system of the first camera is equal to the global coordinate system. This implies that the camera center of the second camera is also the origo. Thus the pinhole camera model for the two cameras are given by:

$$\mathbf{P_1} = \mathbf{A_1} \begin{bmatrix} \mathbf{I} & \mathbf{0} \end{bmatrix} \quad , \quad \mathbf{P_2} = \mathbf{A_2} \begin{bmatrix} \mathbf{R} & \mathbf{0} \end{bmatrix} \ .$$

To find the relationship between two corresponding points, $\mathbf{q_1}$ and $\mathbf{q_2}$, in the two images respectively, assume that they are the projection of the 3D point Q. Then by the pinhole model, (1.21), we have that

$$\begin{aligned} \mathbf{q_1} &= \mathbf{A_1} \begin{bmatrix} \mathbf{I} & \mathbf{0} \end{bmatrix} Q \quad \Rightarrow \\ \mathbf{p_1} &= \mathbf{A_1^{-1} q_1} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \end{bmatrix} Q = \tilde{Q} \ , \end{aligned}$$

Where $\tilde{Q}$ is the *in*homogeneous coordinate corresponding to the homogeneous coordinate Q. Continuing

$$\begin{aligned} \mathbf{q_2} &= \mathbf{A_2} \begin{bmatrix} \mathbf{R} & \mathbf{0} \end{bmatrix} Q \\ &= \mathbf{A_2 R} \tilde{Q} \\ &= \mathbf{A_2 R A_1^{-1} q_1} \end{aligned} \tag{2.19}$$

Here $\mathbf{A_1}$, $\mathbf{R}$ and $\mathbf{A_2}$ are all full rank 3×3 matrices, therefor so is

$$\mathbf{H} = \mathbf{A_2 R A_1^{-1}} \ , \tag{2.20}$$

which denotes the homography relating corresponding observations.

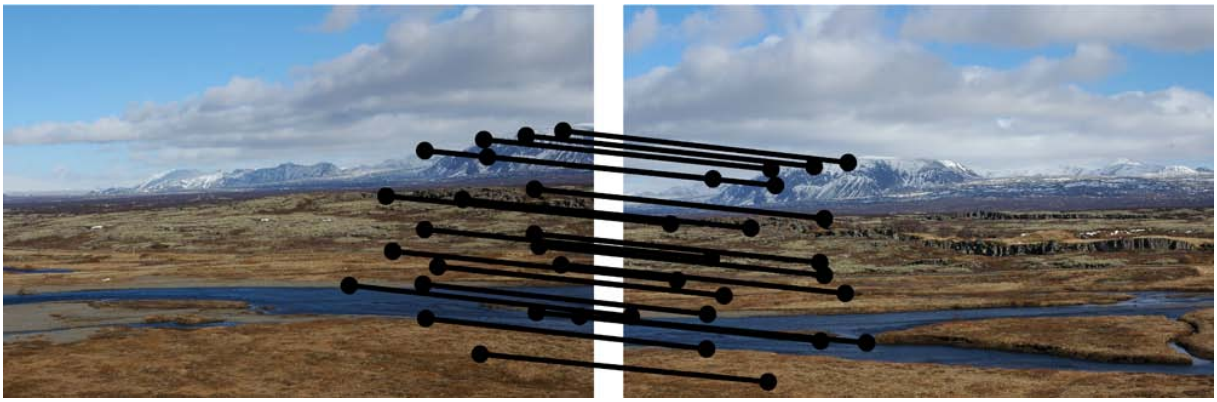

Figure 2.9: Two landscape images taken from the same spot, i.e. zero baseline. The black lines and dots illustrate the 20 points correspondences manually annotated.

Figure 2.10: Based on the 20 annotated point correspondences in Figure 2.9, a homography is estimated and the left image is warped to fit the right, resulting in this figure.

An example: Image Panoramas

A place where a homography is often used to relate two images taken with zero base line is in generating image panoramas. This is done from a series of images taken from the same spot, c.f. Figure 2.10. This result is generated from the two images in Figure 2.9, where 20 point correspondences have been annotaed manually, and the method of Section 2.8 is used to estimate the needed homography. This homography is then used to warp the left image of Figure 2.9, c.f. Section 2.3.2. The two images are averaged together resulting in Figure 2.10. It is noted that much better and automated methods exist for both finding the correspondences between the images, and for blending them together. The aim here is, however, illustrating the theory and not making nice results.

2.4 Point Triangulation

One of the classical task of 3D inference from cameras, is making 3D measurements from two or more known cameras. This is known as triangulation, and in the case of points; point triangulation. Specifically we have a set of cameras, where we know the internal and external calibration of the all the cameras, and we want to find the position of an object that is identified in all cameras. This boils down to finding the coordinates of a 3D point Q from it's known projections, $\mathbf{q_i}$ $i = [1, \ldots, n]$, in n known cameras, $\mathbf{P_i}$. This will be covered here,

[3]The homography has been scaled by a factor of 40 ($diag(40, 40, 1) \cdot \mathbf{H}$), such that each chess board square would be 40×40 pixels, instead of 1×1.

firstly through a linear algorithm, upon which it is discussed how the estimate can become statistically more meaningful.

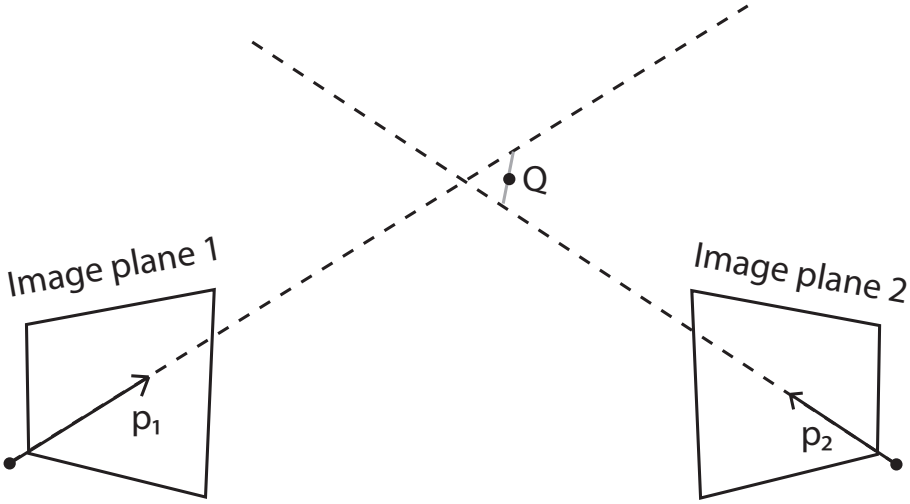

Figure 2.11: The result of point triangulation is the 3D point, Q, closest to the back-projections of the observed 2D points.

The basis of point triangulation, as seen in Figure 2.11, is that the back-projected line of each observed 2D points, $\mathbf{q_i}$, forms a constraint on the position of the 3D point, Q. So in the absence of noise, one would only have to find the intersection of these 3D lines. Two, or more, lines do not in general intersect in 3D, so with noise, we need to find the 3D point that is closest to thee lines. What is meant by closest will be discussed in Section 2.4.3.

2.4.1 A Linear Algorithm

Here a linear algorithm for 3D point triangulation is presented. To ease notation, the rows of the $\mathbf{P_i}$ will *here* be denoted by a superscript, i.e.

$$\mathbf{P_i} = \begin{bmatrix} P_i^1 \\ P_i^2 \\ P_i^3 \end{bmatrix} ,$$

and thus the pinhole camera model, (1.21), can be expanded as follows

$$\begin{aligned} \mathbf{q_i} &= \begin{bmatrix} s_i x_i \\ s_i y_i \\ s_i \end{bmatrix} = \begin{bmatrix} P_i^1 \\ P_i^2 \\ P_i^3 \end{bmatrix} Q \quad \Rightarrow \\ s_i x_i &= P_i^1 Q \quad , \quad s_i y_i = P_i^2 Q \quad , \quad s_i = P_i^3 Q \quad \Rightarrow \\ x_i &= \frac{s_i x_i}{s_i} = \frac{P_i^1 Q}{P_i^3 Q} \quad , \quad y_i = \frac{s_i y_i}{s_i} = \frac{P_i^2 Q}{P_i^3 Q} \ . \end{aligned} \tag{2.21}$$

Doing a bit of arithmetic on (2.21) results in

$$\begin{aligned} x_i = \frac{P_i^1 Q}{P_i^3 Q} \quad &, \quad y_i = \frac{P_i^2 Q}{P_i^3 Q} \quad \Rightarrow \\ P_i^3 Q x_i = P_i^1 Q \quad &, \quad P_i^3 Q y_i = P_i^2 Q \quad \Rightarrow \\ P_i^3 Q x_i - P_i^1 Q = 0 \quad &, \quad P_i^3 Q y_i - P_i^2 Q = 0 \quad \Rightarrow \\ \left(P_i^3 x_i - P_i^1\right) Q = 0 \quad &, \quad \left(P_i^3 y_i - P_i^2\right) Q = 0 \ . \end{aligned} \tag{2.22}$$

Here (2.22) is seen to be linear constraints in Q. Since Q has three degrees of freedom we need at least three such constraint to determine Q. This corresponds to projections in at least two known cameras, since each camera poses two linear constraints in general. Comparing with Section 1.1 it is seen that the x and y parts of (2.22) correspond to planes. E.g. knowing the x-coordinate of Q's image in a given camera, defines a plane

with coefficients $P_i^3 x_i - P_i^1$, which Q lies on. The intersection of the planes the x and y coordinates pose is the 3D line corresponding to the back-projection of the 2D point $\mathbf{q_i}$.

The way Q is calculated from all these linear constraints, is to stack them all in matrix[4]

$$\mathbf{B} = \begin{bmatrix} P_1^3 x_i - P_1^1 \\ P_1^3 y_i - P_1^2 \\ P_2^3 x_i - P_2^1 \\ P_2^3 y_i - P_2^2 \\ \vdots \\ P_n^3 x_i - P_n^1 \\ P_n^3 y_i - P_n^2 \end{bmatrix} ,$$

Then (2.22) is equivalent to

$$\mathbf{B}Q = \mathbf{0} \ .$$

With noisy measurements, however, this will not hold perfectly, and we instead solve

$$\min_Q ||\mathbf{B}Q||_2^2 \ . \tag{2.23}$$

This is seen to be a least squares problem, which is straight forward to solve c.f. Appendix A, as illustrate in the following MatLab code

```
[u,s,v]=svd(B);
Q=v(:,end);
```

2.4.2 An Example

Figure 2.12: Two images with known camera models. Two points corresponding to the same 3D point have been annotated, such that this 3D point can be estimated.

As an example of point triangulation consider the case in Figure 2.12. Here two points corresponding to the same 3D point have been annotated in the two images with coordinates [5]

$$\mathbf{q_1} = \begin{bmatrix} 1800 \\ 730 \\ 1 \end{bmatrix} \quad , \quad \mathbf{q_2} = \begin{bmatrix} 930 \\ 600 \\ 1 \end{bmatrix} \ ,$$

and the corresponding cameras are given by

$$\mathbf{P_1} = \begin{bmatrix} 3274 & -447 & -1027 & 47431 \\ 1120 & 2952 & 848 & 6798 \\ 1 & 0 & 1 & 4 \end{bmatrix} \quad , \quad \mathbf{P_2} = \begin{bmatrix} 3315 & 314 & 941 & 11949 \\ 398 & 3024 & 1177 & -2417 \\ 0 & 0 & 1 & -2 \end{bmatrix} \ .$$

[4] If $n = 2$ there is naturally only four rows of $\mathbf{B}$ corresponding to $\mathbf{P_1}$ and $\mathbf{P_2}$.

[5] This Example is made from real data, as such rounding errors occur.

The linear equations in the form of $\mathbf{B}$ are then given by

$$\mathbf{B} = \begin{bmatrix} P_1^3 x_i - P_1^1 \\ P_1^3 y_i - P_1^2 \\ P_2^3 x_i - P_2^1 \\ P_2^3 y_i - P_2^2 \end{bmatrix} = \begin{bmatrix} -2068 & 212 & 2342 & -39501 \\ -631 & -3047 & -314 & -3582 \\ -3160 & 240 & -27 & -13571 \\ -298 & -3071 & -587 & 1371 \end{bmatrix} ,$$

The solution to (2.23) is then given by

$$Q = \begin{bmatrix} -4.5257 \\ -1.5911 \\ 13.0108 \\ 1 \end{bmatrix} ,$$

This is the linear estimate of the 3D point.

2.4.3 A Statistical Issue*

As mentioned, there is an issue with the linear algorithm presented in Section 2.4.1. This issue is that the pinhole camera mode, as expanded in (2.21), and the localization of the 2D points $\mathbf{q_i}$ are assumed to be perfect and without noise. This is not realistic. Thus a more accurate (2.21) should look as follows

$$x_i = \frac{P_i^1 Q}{P_i^3 Q} + \varepsilon_i^x \quad , \quad y_i = \frac{P_i^2 Q}{P_i^3 Q} + \varepsilon_i^y \ . \tag{2.24}$$

Where $(\varepsilon_i^x, \varepsilon_i^y)$ is the noise of the point 2D location, (x_i, y_i). Redoing the calculations of (2.22) based on (2.24) instead of (2.21) gives

$$\begin{aligned} x_i = \frac{P_i^1 Q}{P_i^3 Q} + \varepsilon_i^x \quad &, \quad y_i = \frac{P_i^2 Q}{P_i^3 Q} + \varepsilon_i^y \quad \Rightarrow \\ P_i^3 Q x_i = P_i^1 Q + P_i^3 Q \varepsilon_i^x \quad &, \quad P_i^3 Q y_i = P_i^2 Q + P_i^3 Q \varepsilon_i^y \quad \Rightarrow \\ \left(P_i^3 x_i - P_i^1\right) Q = P_i^3 Q \varepsilon_i^x \quad &, \quad \left(P_i^3 y_i - P_i^2\right) Q = P_i^3 Q \varepsilon_i^y \ . \end{aligned} \tag{2.25}$$

Where $P_i^3 Q$ is equal to the distance from the camera to the 3D point, as can be seen from the derivation in Section 1.4. So with the error model as in (2.24), the minimization problem in (2.23) is equivalent to

$$\begin{aligned} &\min_Q ||\mathbf{B}Q||_2^2 \quad = \\ &\min_Q \sum_{i=1}^n \left(P_i^3 Q \varepsilon_i^x\right)^2 + \left(P_i^3 Q \varepsilon_i^y\right)^2 \quad = \\ &\min_Q \sum_{i=1}^n \left(P_i^3 Q\right)^2 \left|\left|\begin{bmatrix} \varepsilon_i^x \\ \varepsilon_i^y \end{bmatrix}\right|\right|_2^2 \ . \end{aligned} \tag{2.26}$$

That is observations made by cameras further from the 3D point are weighted more. Given that the error model in (2.24) is the correct one, this is not a meaningful quantity to minimize. It is however the result a a linear algorithm which is computationally feasible and in general gives decent results. Algorithms with this property are said to minimize an *algebraic error measure*.

On The Error Model*

The error model given in (2.24) basically states that the meaningful error is on the image point location, c.f. Figure 2.13-Left. The motivation for this is that the two main sources of error is identifying, where in an image entity is actually seen, c.f. Figure 2.13-Right, and unmodelled optical phenomena, such as radial distortion. Both these entities are captured well by a deviation in the point location, as in (2.24).

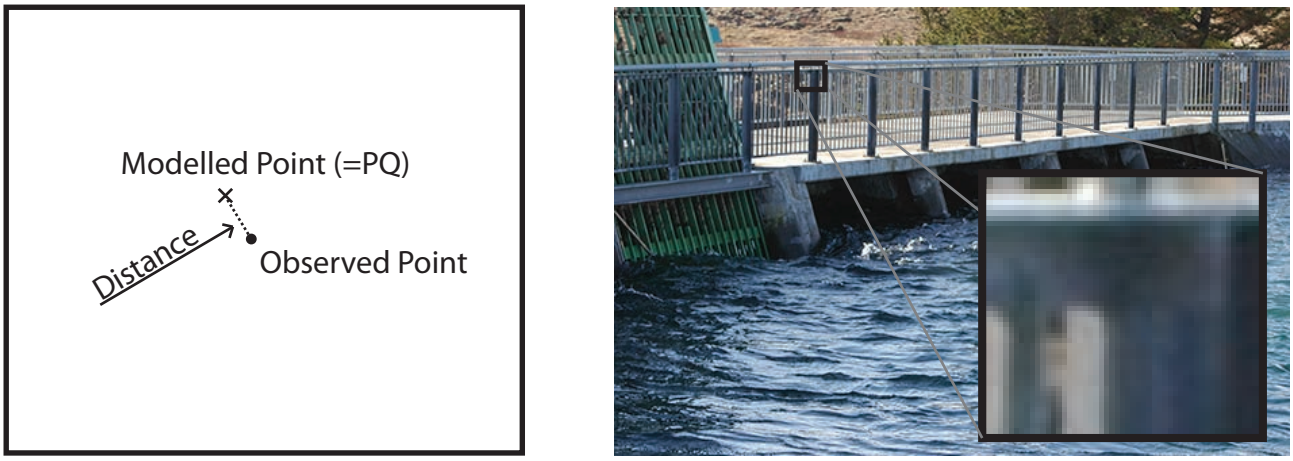

Figure 2.13: **Left:** A Schematic view of the distance to be minimized, namely the distance between the observed point at the one originating from the model. In the point triangulation case the model is $\mathbf{q} = \mathbf{P}Q$. **Right:** An illustration that even though an image apparently has many well defined corners, when we zoom in at a pixel level, the localization is still somewhat uncertain. This will in general hold for most image features.

Non-linear Minimization*

If a statistically more meaningful estimate is needed, compared to the linear algorithm in Section 2.4.1, then the procedure is to solve a nonlinear optimization problem, similar to that of (1.32) in Section 1.6. What we want to minimize, w.r.t. Q, is

$$\min_{Q} \sum_{i=1}^{n} \left|\left|\left[\begin{array}{c} \varepsilon_i^x \\ \varepsilon_i^y \end{array}\right]\right|\right|_2^2 = \min_{Q} \sum_{i=1}^{n} ||\Pi(q_i) - \Pi(\mathbf{P}Q_i)||_2^2 \ . \tag{2.27}$$

Here the function $\Pi(\cdot)$ appears again, which takes homogeneous coordinates and produces the inhomogeneous correspondent. In (2.27) the two norm squared is used, which is consistent with a Gaussian nose model for the ε, and is equal to minimizing the squared Euclidian distance. This is the standard assumption and methodology. There is, however, some debate as to norm to use, especially in relation to outlier suppression. This norm issue is beyond the scope of this text. Lastly it should be mentioned, that the non-linear optimization methods used to solve (2.27), requires an initial guess. This guess is typically supplied by the *linear* algorithm from Section 2.4.1.

2.5 A Light Projector as a 'Camera'

One of the main problems in reconstructing 3Dsurfaces or geometry from images, is that it is hard to determine the correspondence between the relevant 2D features or image points. I.e. it is often hard to get the 2D input of Figure 2.12. To address this issue light projectors are often used to make a known light pattern, making correspondences easier to achieve, c.f. Figure 2.15-Left. Much of the camera geometry derived in this chapter is, however, indifferent to which way the light travels, i.e. if it enters a camera or leaves a projector. In particular, much of the theory builds on the back-projection of image observations, and the light emitted from a projector can be seen as the embodiment of this back-projection. In the following a particular instance of such an active 3D capturing system will be covered, namely the laser scanner.

2.5.1 Laser Scanners

A laser scanner basically consists of a camera and a laser emitting a point or a plane. Here the laser plane version is considered, c.f. Figure 2.14 and Figure 2.15. The idea is that the emitted laser plan will hit and deflect off a surface, and that deflected light will hit the camera, as illustrated in Figure 2.14. Each light point hitting the camera is then known to lie on the back-projection of that point *and* on the laser plane. The laser plane is assumed known.

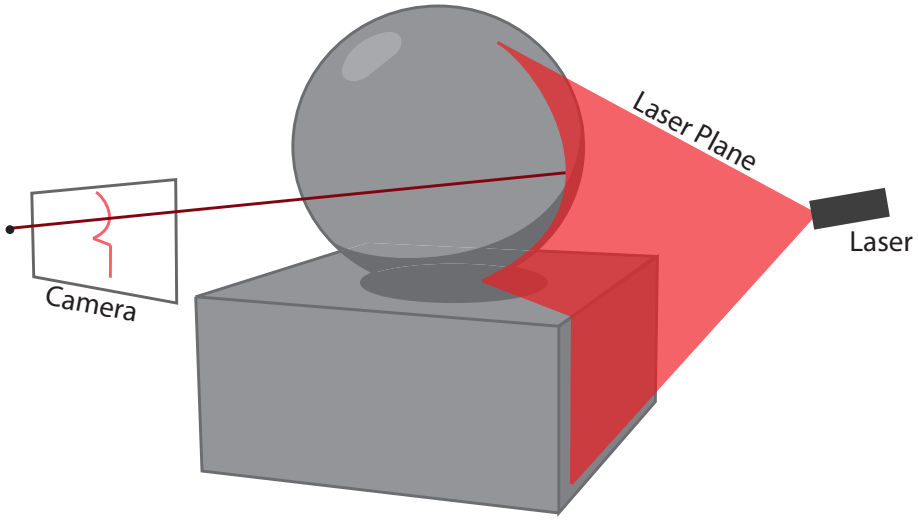

Figure 2.14: A schematic illustration of a laser scanner. The laser emits a laser plane that hits the surface and reflects light into the camera. The intersection of the laser plane and the back-projected line defines a 3D point Q.

Referring to the derivation in Section 2.4.1, the 3D line back-projecting from a point, (x, y) can be seen as the intersection of two planes, namely

$$\left(P^3 x - P^1\right) Q = 0 \quad , \quad \left(P^3 y - P^2\right) Q = 0 \ .$$

Where $\mathbf{P}$ is the known camera matrix. Assuming that the laser plane is given by

$$L^T \cdot Q = 0 \ ,$$

and following the deriviation in Section 2.4.1, we have three linear constraints on the 3D point Q, which lies on the surface we wish to reconstruct. In particular we have

$$\mathbf{B}Q = \begin{bmatrix} P^3 x - P^1 \\ P^3 y - P^2 \\ L^T \end{bmatrix} Q = 0 \ . \tag{2.28}$$

The solution is thus the right null space of $\mathbf{B}$. Since there is three linear constraint and three degrees of freedom, then a Q can be found that perfectly fulfills all three linear constraints. Thus, there is no need to consider noise models in this minimal case. As an example consider some results from the digital michelangelo project c.f. [18], as depicted in Figure 2.15.

2.6 Camera Resection

Just as a 3D point was estimated from the corresponding 2D projections in known cameras, c.f. Section 2.4, a camera can be estimated from known 3D points and the corresponding 2D projections. This is known as camera resectioning or estimating the *camera pose*, and can amongst others be used to determine the position of a camera. This is useful in e.g. mobil robot navigation, where camera resectioning can determine where the robot is from observed known 3D points.

Camera resectioning is highly related to camera calibration, as covered in Section 1.6, in that both tasks are concerned with estimating the camera parameters. Here the material covered in Section 1.6 will be extended by a linear algorithm for determining a pinhole camera. This algorithm is useful in its own right, but is also often used to initialized optimization problems of (1.32). This linear algorithm, as the one in Section 2.4.1 does not minimizes a statistical meaningful error, but minimizes an *algebraic error* in stead. If resectioning is to be done for a camera with known internal parameters, special purpose algorithms exist, c.f. e.g. [12].

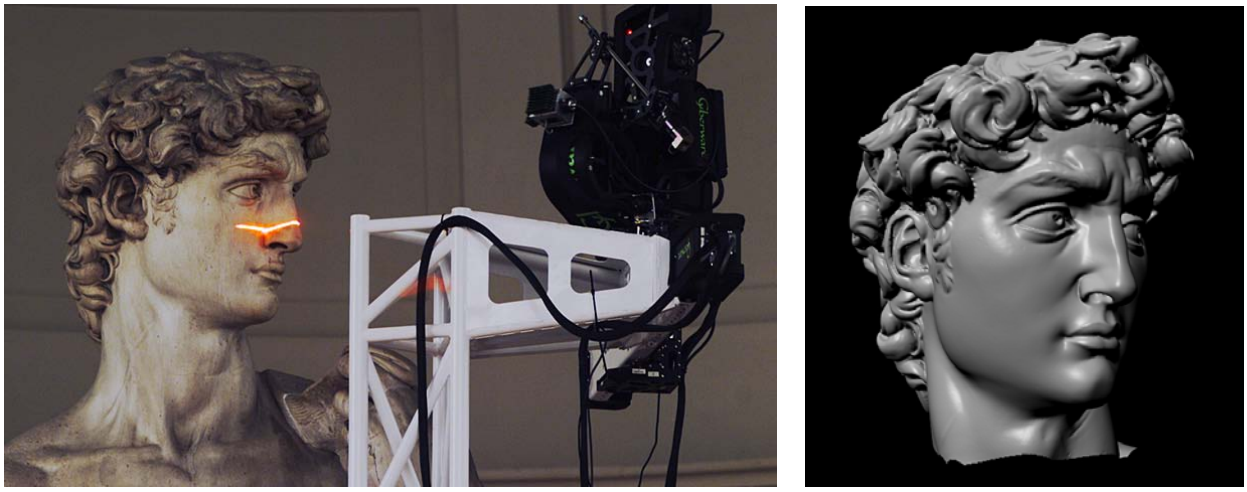

Figure 2.15: results from the digital michelangelo project c.f. [18]. **Left:** The laser plane and camera set up applied to the head of David. **Right:** The scanned result, where allot of 3D points have been scanned and used as vertices of a triangular mesh, which is rendered here.

2.6.1 A Linear Algorithm

First the linear constraints the known 3D point, $Q_i = [X_i, Y_i, Z_i, 1]^T$ and it's corresponding 2D observation $\mathbf{q_i} = [x_i, y_i, 1]^T$ poses on the the 3×4 pinhole camera matrix $\mathbf{P}$. Given the pinhole camera model (1.21)

$$
\begin{aligned}
&\mathbf{q_i} = \mathbf{P}Q_i \qquad \Rightarrow \\
&\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} = [\mathbf{q_i}]_\times \mathbf{P} Q_i \\
&= \begin{bmatrix} 0 & -1 & y_i \\ 1 & 0 & -x_i \\ -y_i & x_i & 0 \end{bmatrix} \begin{bmatrix} P_{11} & P_{12} & P_{13} & P_{14} \\ P_{21} & P_{22} & P_{23} & P_{24} \\ P_{31} & P_{32} & P_{33} & P_{34} \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix} \\
&= \begin{bmatrix} 0 & -1 & y_i \\ 1 & 0 & -x_i \\ -y_i & x_i & 0 \end{bmatrix} \begin{bmatrix} P_{11}X_i + P_{12}Y_i + P_{13}Z_i + P_{14} \\ P_{21}X_i + P_{22}Y_i + P_{23}Z_i + P_{24} \\ P_{31}X_i + P_{32}Y_i + P_{33}Z_i + P_{34} \end{bmatrix} \\
&= \begin{bmatrix} -P_{21}X_i + y_iP_{31}X_i - P_{22}Y_i + y_iP_{32}Y_i - P_{23}Z_i + y_iP_{33}Z_i - P_{24} + y_iP_{34} \\ P_{11}X_i - x_iP_{31}X_i + P_{12}Y_i - x_iP_{32}Y_i + P_{13}Z_i - x_iP_{33}Z_i + P_{14} - x_iP_{34} \\ -y_iP_{11}X_i + x_iP_{21}X_i - y_iP_{12}Y_i + x_iP_{22}Y_i - y_iP_{13}Z_i + x_iP_{23}Z_i - y_iP_{14} + x_iP_{24} \end{bmatrix} \\
&= \begin{bmatrix} 0 & -X_i & y_iX_i & 0 & -Y_i & y_iY_i & 0 & -Z_i & y_iZ_i & 0 & -1 & y_i \\ X_i & 0 & -x_iX_i & Y_i & 0 & -x_iY_i & Z_i & 0 & -x_iZ_i & 1 & 0 & -x_i \\ -y_iX_i & x_iX_i & 0 & -y_iY_i & x_iY_i & 0 & -y_iZ_i & x_iZ_i & 0 & -y_i & x_i & 0 \end{bmatrix} vec(\mathbf{P}) \\
&= \mathbf{b_i^T} \cdot vec(\mathbf{P}) \ . \qquad (2.29)
\end{aligned}
$$

Where

$$
vec(\mathbf{P}) = \begin{bmatrix} P_{11} & P_{21} & P_{31} & P_{12} & P_{22} & P_{32} & P_{13} & P_{23} & P_{33} & P_{14} & P_{24} & P_{34} \end{bmatrix}^T \ .
$$

The first step of the calculations in (2.29) requires an explanation. Since the pinhole camera model is formulated in terms of homogeneous coordinates, $\mathbf{q_i}$ and $\mathbf{P}Q_i$ are thus equal only up to scale. This makes the equation a bit harder to work with. Viewing $\mathbf{q_i}$ and $\mathbf{P}Q_i$ as vectors in 3D, it is seen that the pinhole camera model states that they are vectors with the same direction, and with (possible) different lengths. This implies that the cross product between the two must be zero, which is the argument used to get from the first to the second line of (2.29).

As in Section 2.4.1 $vec(\mathbf{P})$, and thus $\mathbf{P}$, is estimated from (2.29) by stacking the $\mathbf{b_i^T}$ into a matrix

$$\mathbf{B} = \begin{bmatrix} \mathbf{b_1^T} \\ \mathbf{b_2^T} \\ \vdots \\ \mathbf{b_n^T} \end{bmatrix} ,$$

and solving

$$\min_{vec(\mathbf{P})} ||\mathbf{B} \cdot vec(\mathbf{P})||_2^2 \quad . \tag{2.30}$$

Even though the $\mathbf{b_i^T}$ are of dimension 3×12 the in general only have rank two, and thus one 2D-3D point correspondence only pose two linear constraints on $\mathbf{P}$. Since $\mathbf{P}$ has eleven degrees of freedom ($12 = 3 \cdot 4$ minus one for overall scale) six 2D-3D point correspondences are needed as a minimum. Most often more 2D-3D point correspondence more are advantageous, since it serves to reduce noise.

Use of the Kronecker Product*

The amount of terms in (2.29) illustrates the power of using the Kronecker product, c.f. Appendix A.7, in that

$$Q_i^T \otimes [\mathbf{q_i}]_\times = \begin{bmatrix} 0 & -X_i & y_iX_i & 0 & -Y_i & y_iY_i & 0 & -Z_i & y_iZ_i & 0 & -1 & y_i \\ X_i & 0 & -x_iX_i & Y_i & 0 & -x_iY_i & Z_i & 0 & -x_iZ_i & 1 & 0 & -x_i \\ -y_iX_i & x_iX_i & 0 & -y_iY_i & x_iY_i & 0 & -y_iZ_i & x_iZ_i & 0 & -y_i & x_i & 0 \end{bmatrix} ,$$

and that (c.f. Appendix A.7)

$$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} = [\mathbf{q_i}]_\times \mathbf{P} Q_i = \left(Q_i^T \otimes [\mathbf{q_i}]_\times\right) \cdot vec(\mathbf{P}) \ , \tag{2.31}$$

Which is equivalent to (2.29). Actually using the Kronecker product as in (2.31) is often preferable to the formulation in (2.29), simply due to the number of typing mistakes avoided.

2.7 Estimating the Epipolar Geometry*

Just like the fundamental matrix can be used as a constraint on point correspondences, it can also be estimated from such point correspondences. As in Section 2.4.1 linear algorithms – minimizing an algebraic error — will first be considered, followed by a brief mention of non-linear methods.

2.7.1 The 8-Point Algorithm*

Assume that n point correspondences $\mathbf{q_{1i}}, \mathbf{q_{2i}} \quad i \in [1, \ldots, n]$, are given, where

$$\mathbf{q_{1i}} = \begin{bmatrix} x_{1i} \\ y_{1i} \\ 1 \end{bmatrix} \quad , \quad \mathbf{q_{2i}} = \begin{bmatrix} x_{2i} \\ y_{2i} \\ 1 \end{bmatrix} \quad .$$

Inserting this into (2.11), gives for each i

$$\begin{aligned} 0 &= \mathbf{q_{2i}^T F q_{1i}} \\ &= \begin{bmatrix} x_{2i} \\ y_{2i} \\ 1 \end{bmatrix}^T \begin{bmatrix} F_{11} & F_{12} & F_{13} \\ F_{21} & F_{22} & F_{23} \\ F_{31} & F_{32} & F_{33} \end{bmatrix} \begin{bmatrix} x_{1i} \\ y_{1i} \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} x_{2i} \\ y_{2i} \\ 1 \end{bmatrix}^T \begin{bmatrix} F_{11}x_{1i} + F_{12}y_{1i} + F_{13} \\ F_{21}x_{1i} + F_{22}y_{1i} + F_{23} \\ F_{31}x_{1i} + F_{32}y_{1i} + F_{33} \end{bmatrix} \\ &= F_{11}x_{1i}x_{2i} + F_{21}x_{1i}y_{2i} + F_{31}x_{1i} + F_{12}y_{1i}x_{2i} + F_{22}y_{1i}y_{2i} + F_{32}y_{1i} + F_{13}x_{2i} + F_{23}y_{2i} + F_{33} \\ &= \begin{bmatrix} x_{1i}x_{2i} & x_{1i}y_{2i} & x_{1i} & y_{1i}x_{2i} & y_{1i}y_{2i} & y_{1i} & x_{2i} & y_{2i} & 1 \end{bmatrix} vec(\mathbf{F}) \\ &= \mathbf{b_i^T} \cdot vec(\mathbf{F}) \ , \end{aligned} \tag{2.32}$$

Where

$$vec(\mathbf{F}) = \begin{bmatrix} F_{11} & F_{21} & F_{31} & F_{12} & F_{22} & F_{32} & F_{13} & F_{23} & F_{33} \end{bmatrix}^T \ .$$

As in the previous sections, the $\mathbf{b_i^T}$ are arranged in a matrix

$$\mathbf{B} = \begin{bmatrix} \mathbf{b_1^T} \\ \mathbf{b_2^T} \\ \vdots \\ \mathbf{b_n^T} \end{bmatrix} \ ,$$

and a solution is found via

$$\min_{vec(\mathbf{F})} ||\mathbf{B} \cdot vec(\mathbf{F})||_2^2 \ . \tag{2.33}$$

Since this algorithm does not exploit the the fact that the fundamental matrix has a determinant of zero, c.f. Section 2.2.2, eight constraints are needed to determine $\mathbf{F}$. Since each point correspondence yields one linear constraint this algorithm is known as the 8-point algorithm. Like many of the linear algorithm presented in this chapter, the 8-point algorithm minimizes an algebraic error.

7-Point Algorithm*

The constraint that the determinant of $\mathbf{F}$ is zero, not used in the 8-point algorithm, can relatively easily be incorporated. Thus only seven constraints are needed giving rise to the 7-point algorithm. If only seven point correspondences are used the $\mathbf{B}$ in (2.33), will have a null space of dimension two. The solution to (2.33) will therefore be a linear combination

$$\mathbf{F} = \alpha \mathbf{F}' + \mathbf{F}^\dagger \ ,$$

spanning this 2D subspace. The correct solution is then found by inserting this in to the constraint

$$\det(\mathbf{F}) = \det(\alpha \mathbf{F}' + \mathbf{F}^\dagger) = 0 \ . \tag{2.34}$$

This is a third order polynomial in α, which will have one or three real solutions. Once alpha is determined, so will $\mathbf{F}$, since $\mathbf{F}'$ and $\mathbf{F}^\dagger$ are known. For further details, among others how to chose between the possible three solutions in RANSAC, the interested reader is referred to [14].

5-Point Algorithm*

If the epipolar geometry is to be estimated from cameras with known internal parameters, then the essential matrix suffices. There are two ways of estimating the essential matrix, the first is to estimate the fundamental matrix, and then try to deduce the essential matrix from here by multiplying the estimated $\mathbf{F}$ with the internal parameters $\mathbf{A_1}$ and $\mathbf{A_2}$. This approach has the disadvantage, that with noise a valid essential matrix is unlikely to appear. This latter issue can, however, likely be dealt with via a non-linear optimization step. Another disadvantage is that this will require at least seven points to estimate, where 5 in theory are all that is needed.

The other, somewhat more complicated, approach to essential matrix estimation is to compute it directly from the observations. Doing this will result in solving a tenth degree polynomial in several variables, and the derivation of this algorithm is beyond the scope of this text. The interested reader is referred to [21]. This algorithm requires 5 point correspondences, hance the algorithm is called the 5-point algorithm.

Use of the Kronecker Product*

Just as in Section 2.6, the Kronecker product can make the calculations in (2.32) easier and less prone to typing mistakes, c.f. Appendix A.7, in that

$$\mathbf{q_{1i}^T} \otimes \mathbf{q_{2i}^T} = \begin{bmatrix} x_{1i}x_{2i} & x_{1i}y_{2i} & x_{1i} & y_{1i}x_{2i} & y_{1i}y_{2i} & y_{1i} & x_{2i} & y_{2i} & 1 \end{bmatrix} \ ,$$

and that (c.f. Appendix A.7)

$$0 = \mathbf{q_{2i}^T F q_{1i}} = \left(\mathbf{q_{1i}^T} \otimes \mathbf{q_{2i}^T}\right) \cdot vec(\mathbf{F}) \ , \tag{2.35}$$

Which is equivalent to (2.32).

2.7.2 Normalization of Points*

When implementing algorithms on a computer numerical considerations are almost always of importance. When minimizing an algebraic error, especially in the fundamental matrix case, experience has shown that most algorithms will fail without such considerations. In the fundamental matrix case this is recommended done by changing the image coordinates, such that the mean and variance of the points in an image are zero and one respectively. This can be done via a scaling, s, and a translation, $[\Delta x, \Delta y]^T$, which in homogeneous coordinates can be implemented as

$$\mathbf{T} = \begin{bmatrix} s & 0 & \Delta x \\ 0 & s & \Delta y \\ 0 & 0 & 1 \end{bmatrix} .$$

This can be used to make a change of coordinates in each of the images, such that

$$\tilde{\mathbf{q_{1i}}} = \mathbf{T_1 q_{1i}} \quad , \quad \tilde{\mathbf{q_{2i}}} = \mathbf{T_2 q_{2i}} \ ,$$

Where $\mathbf{T_1}$ and $\mathbf{T_2}$ are the transformations needed in each of the two images. The mean and variance of the $\tilde{\mathbf{q_{1i}}}$ are then zero and one respectively. The same should hold for the $\tilde{\mathbf{q_{2i}}}$. The fundamental matrix is then estimated using the $\tilde{\mathbf{q_{1i}}}$ and $\tilde{\mathbf{q_{2i}}}$ giving a $\tilde{\mathbf{F}}$, such that

$$0 = \tilde{\mathbf{q_{2i}}}^{\mathbf{T}} \tilde{\mathbf{F}} \tilde{\mathbf{q_{1i}}} = \mathbf{q_{2i}^T T_2^T} \tilde{\mathbf{F}} \mathbf{T_1 q_{1i}} \ , \tag{2.36}$$

Implying that the fundamental matrix sought is

$$\mathbf{F} = \mathbf{T_2^T} \tilde{\mathbf{F}} \mathbf{T_1} \ . \tag{2.37}$$

2.7.3 Non-Linear Optimization*

To improve upon the estimate of the linear algorithms minimizing an algebraic error presented above, a non-linear optimization is often used. Using the statistical optimal error is often too cumbersome and a first order approximation to it is often minimized. This error measure is called the Sampson error and is given by

$$d_{Samp}(\mathbf{F}, \mathbf{q_{1i}}, \mathbf{q_{2i}}) = \frac{\mathbf{q_{2i}^T F q_{1i}}}{(\mathbf{F q_{1i}})_1^2 + (\mathbf{F q_{1i}})_2^2 + (\mathbf{q_{2i}^T F})_1^2 + (\mathbf{q_{2i}^T F})_2^2}$$

Here $(\mathbf{F q_{1i}})_j^2$ denotes the j^{th} coordinate of $\mathbf{F q_{1i}}$ squared, j being one or two. This is minimized over all point correspondences, i.e. the following minimization problem is solved

$$\min_{\mathbf{F}} \sum_{i=1}^{n} d_{Samp}(\mathbf{F}, \mathbf{q_{1i}}, \mathbf{q_{2i}})$$

In solving this minimization, experience has shown that care has to be taken in parametrization of $\mathbf{F}$. A further discussion of this parameterizations issue and on the error measures for minimizing $\mathbf{F}$ is found in [14].

2.8 Estimating a Homography*

The last entity we want to estimate in this chapter is a homography from 2D point correspondences, $\mathbf{q_{1i}}$ and $\mathbf{q_{2i}}$. This is done by first introducing a linear algorithm – minimizing an algebraic measure – whereupon a non-linear approach incorporating better statistical reasoning will be covered.

2.8.1 A Linear Algorithm*

The linear algorithm for homography estimation resembles the linear algorithm for camera resectioning a lot. Assume that n 2D point correspondences, i.e. $\mathbf{q_{1i}}$ and $\mathbf{q_{2i}}$ $i \in \{1, \cdots\}$ are given. The linear constraint such a

constraint poses on $\mathbf{H}$ is derived as follows, inserting into (2.13)

$$\begin{aligned}
\mathbf{q_{1i}} &= \mathbf{H q_{2i}} \quad \Rightarrow \\
\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} &= [\mathbf{q_{1i}}]_\times \mathbf{H q_{2i}} \\
&= \begin{bmatrix} 0 & -1 & y_{1i} \\ 1 & 0 & -x_{1i} \\ -y_{1i} & x_{1i} & 0 \end{bmatrix} \begin{bmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ H_{31} & H_{32} & H_{33} \end{bmatrix} \begin{bmatrix} x_{2i} \\ y_{2i} \\ 1 \end{bmatrix} \\
&= \begin{bmatrix} 0 & -1 & y_{1i} \\ 1 & 0 & -x_{1i} \\ -y_{1i} & x_{1i} & 0 \end{bmatrix} \begin{bmatrix} H_{11}x_{2i} + H_{12}y_{2i} + H_{13} \\ H_{21}x_{2i} + H_{22}y_{2i} + H_{23} \\ H_{31}x_{2i} + H_{32}y_{2i} + H_{33} \end{bmatrix} \\
&= \begin{bmatrix} -H_{21}x_{2i} + H_{31}x_{2i}y_{1i} - H_{22}y_{2i} + H_{32}y_{2i}y_{1i} - H_{23} + H_{33}y_{1i} \\ H_{11}x_{2i} - H_{31}x_{2i}x_{1i} + H_{12}y_{2i} - H_{32}y_{2i}x_{1i} + H_{13} - H_{33}x_{1i} \\ -H_{11}x_{2i}y_{1i} + H_{21}x_{2i}x_{1i} - H_{12}y_{2i}y_{1i} + H_{22}y_{2i}x_{1i} - H_{13}y_{1i} + H_{23}x_{1i} \end{bmatrix} \\
&= \begin{bmatrix} -x_{2i} + x_{2i}y_{1i} - y_{2i} + y_{2i}y_{1i} - 1 + y_{1i} \\ x_{2i} - x_{2i}x_{1i} + y_{2i} - y_{2i}x_{1i} + 1 - x_{1i} \\ -x_{2i}y_{1i} + x_{2i}x_{1i} - y_{2i}y_{1i} + y_{2i}x_{1i} - y_{1i} + x_{1i} \end{bmatrix} \cdot vec(\mathbf{H}) \\
&= \mathbf{b_i^T} \cdot vec(\mathbf{H}) \ . \qquad (2.38)
\end{aligned}$$

Where

$$vec(\mathbf{H}) = \begin{bmatrix} H_{11} & H_{21} & H_{31} & H_{12} & H_{22} & H_{32} & H_{13} & H_{23} & H_{33} \end{bmatrix}^T \ .$$

As in the previous sections, the $\mathbf{b_i^T}$ are arranged in a matrix

$$\mathbf{B} = \begin{bmatrix} \mathbf{b_1^T} \\ \mathbf{b_2^T} \\ \vdots \\ \mathbf{b_n^T} \end{bmatrix} \ ,$$

and a solution is found via

$$\min_{vec(\mathbf{H})} ||\mathbf{B} \cdot vec(\mathbf{H})||_2^2 \ . \qquad (2.39)$$

Also as in the previous sections it should also be mentioned that there are several advantages of using the Kronecker Product in doing the calculations of (2.38), c.f. Appendix A.7.

2.8.2 Non-Linear Optimization*

To minimize a statistically more meaningful error, assume that the noise free measurements in the two images, $\mathbf{s_{1i}}$ and $\mathbf{s_{2i}}$, are given by

$$\mathbf{q_{1i}} = \mathbf{s_{1i}} + \varepsilon_{1i} \quad , \quad \mathbf{q_{2i}} = \mathbf{s_{2i}} + \varepsilon_{1i} \ ,$$

Where ε_{1i} and ε_{2i} are noise components that we want to minimize. For the true $\mathbf{H}$ we furthermore have the following relationship

$$\mathbf{s_{1i}} = \mathbf{H s_{2i}} \ ,$$

The minimization problem that we wish to solve is thus

$$\min \sum_{i=1}^{n} \left(||\mathbf{s_{1i}} - \mathbf{p_{1i}}||_2^2 + ||\mathbf{s_{2i}} - \mathbf{p_{2i}}||_2^2 \right) = \min \sum_{i=1}^{n} \left(||\mathbf{H s_{2i}} - \mathbf{p_{1i}}||_2^2 + ||\mathbf{s_{2i}} - \mathbf{p_{2i}}||_2^2 \right) \ ,$$

Which is a minimization problem in the elements of $\mathbf{H}$ and the $\mathbf{s_{2i}}$ over all $i \in \{1, \cdots, n\}$. A good solution to minimizing the statistical meaningful error in the images – assuming Gaussian distribution of the ε – is therefor solving

$$\min_{\mathbf{H}, \mathbf{s_{2i}}} \sum_{i=1}^{n} \left(||\mathbf{H s_{2i}} - \mathbf{p_{1i}}||_2^2 + ||\mathbf{s_{2i}} - \mathbf{p_{2i}}||_2^2 \right) \ . \qquad (2.40)$$

2.9 End Notes*

As mentioned in the introduction to this chapter, it is only a small subset of multiple view geometry that is covered here. To the authors best judgment, the part chosen are the most introductory, and the ones most often use in practical 3D estimation with cameras. There are, however, two additional issues that the reader should be aware of. Firstly, the estimation algorithms mentioned here are the easies to understand and the 'standard' ones. They however either minimize an algebraic error or in the case of the non-linear algorithms are iterative and have a risk of converging to a local minimum. Recently algorithms have been developed, that solve such optimization problems in a guarantied optimal way, c.f. e.g. [17]. The second issue not dealt much with here is the geometry of calibrated cameras, i.e. that the $\mathbf{A}$ are known. This case has large practical importance, but unfortunately often gives rise to high order polynomial equations in many variables. As such these algorithms are beyond the scope of this text, two few references are [12, 21, 26].

Part II

Image Features and Matching

Chapter 3

Extracting Features

Computer vision and image analysis is mostly concerned with doing inference from images. In doing so we often resort to extracting features from the images, such as points and lines, which we believe have some intrinsic meaning. One of the most important uses of such features is in helping to solve the correspondence problem. This is the problem of finding the correspondence between the 'same thing' depicted in two different images, and is the subject of Chapter 4. This chapter, thus, serves as a prelude to the next. There are naturally many types of features to be extracted from images, such as cars letters and so on, but here the focus is on point and line features. The motivation behind this is that these are the most commonly used in a general setting. A good reference on feature extraction for matching is [27]. As for the organization, the first two sections, namely Section 3.1 and Section 3.2, present some general considerations in relation to feature extraction, upon which two point detection and a line detection scheme is presented in Sections 3.3, 3.4 and 3.5 respectively.

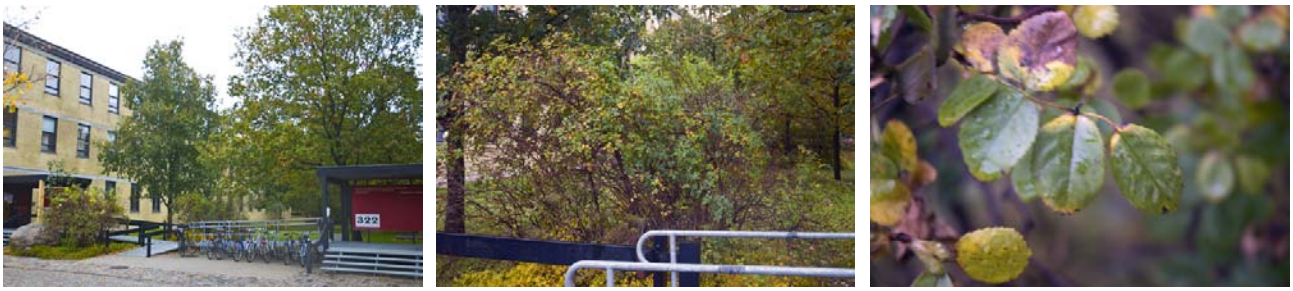

Figure 3.1: Things appear at a scale, as illustrated in the above images. This scale can also vary, as e.g. the leaves of the bush above. When extracting features, we often have to choose which scale we want to work at. This is naturally related to what we want to extract, e.g. the leaves, the branches or the trees.

3.1 Importance of Scale

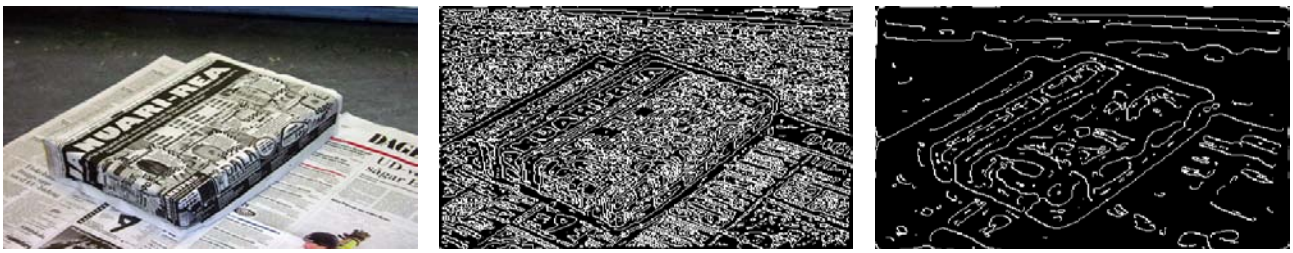

Figure 3.2: The maximum of the gradient magnitude is computed for the leftmost image and depicted in the middle and right images. The difference between the middle and right image is the scale at which the gradient is computed. Notice how different structures appear at the different scales.

The things or features that we want to extract from an image appear at a scale, c.f. Figure 3.1. This scale depends e.g. on the camera position relative to the object, the camera optics, and the actual size of the object

photographed. When extracting features – as well as performing many other image processing tasks – we have to chose what scale we want to work on. This is illustrated in Figure 3.2, where different scales extract different structures.

To explain the choice of scale in more detail consider the one dimensional signal in Figure 3.3. When detecting edges on this signal using a filter, we find one or several edges depending on the extent of this filter. If we use a small filter all the ridges, which might otherwise be viewed as noise, will give an edge responses. If we use a large filter only the one dominant edge will give a response. The dominant edge will, furthermore, not be detected by a small filter, because this edge only becomes dominant on a larger scale. So one view of scale relates to the size of the filters used. These filters are very often used to perform basic and preliminary image processing. Another view is that scale, and the size of the filters used, distinguish between what is noise and what is signal, in a frequency, or fourier setting.

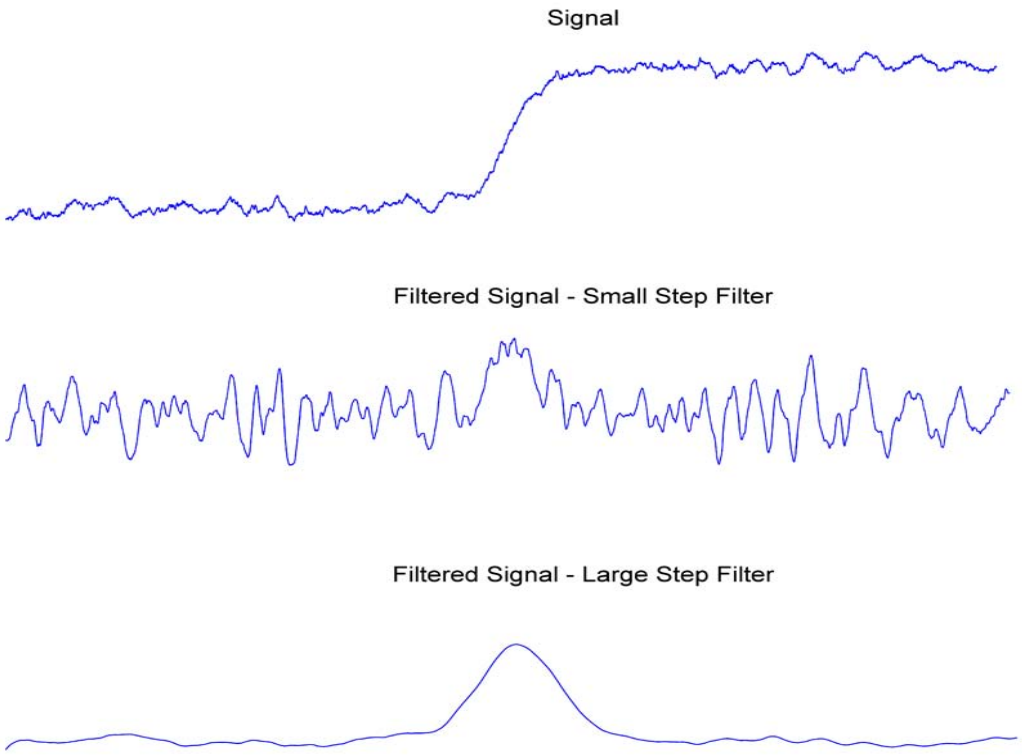

Figure 3.3: **Top:** A sample one dimensional signal with one or more edges. **Middle** The sample signal filtered with a *small* step edge. **Bottom** The sample signal filtered with a *large* step edge.

Image scale and the size of filters is naturally associated with image pyramids. An image pyramid is the successive down scaling of an image, c.f. Figure 3.4. This successive down scaling of an image is referred to as a pyramid, in that placing the images on top of each other will give a pyramid, due to the decreasing size of the images. The relationship to image scale and filter size, is that instead of making the filters larger the image can be made smaller. So in effect, running the same filter on a down sampled image is equivalent to working on a coarser scale. Although this is a somewhat crude technique for working with image scale, this is often used because reducing the image size reduces the computational requirements. This is opposed to increasing the filter size, which most often increases the computational requirements.

3.1.1 Gaussian Kernels

When dealing with image scale, or scale space a working horse is the Gaussian filter, c.f. Figure 3.5. A Gaussian filter is given by the following equation

$$g_\sigma = g(\mathbf{x}, \sigma) = \frac{1}{\left(\sqrt{2\pi\sigma^2}\right)^D} \exp\left(-\frac{||\mathbf{x}||_2^2}{2\sigma^2}\right) \quad , \tag{3.1}$$

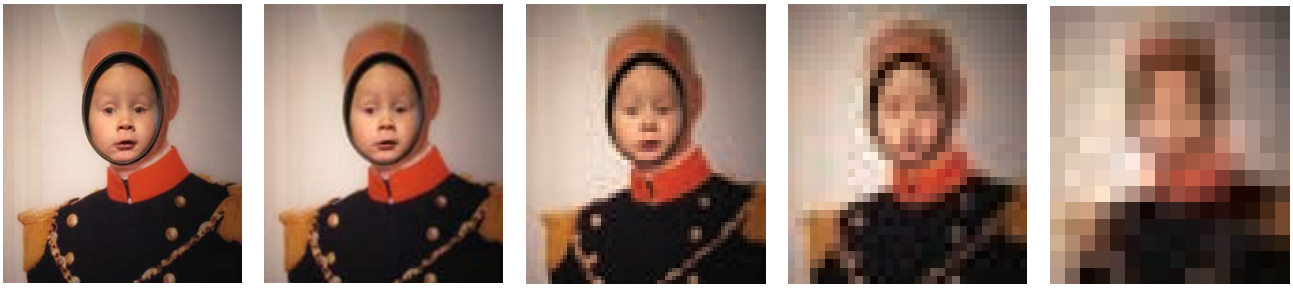

Figure 3.4: A successive down sampling of an image. If these images were placed on top of each other, with a pixel having the same size, they would form a pyramid. Hence the name image pyramid.

where g_σ is a short hand, D[1] is the dimension of $\mathbf{x}$ and σ controls the width of the kernel. The idea is that that if we want to apply a given filter f to a an image I via convolution, i.e.

$$I * f \ ,$$

The scale is typically adjusted by inserting a Gaussian filter, i.e.

$$I * g_\sigma * f \ . \tag{3.2}$$

Where the σ, is the measure of scale. This is opposed to altering the scale or size of the filter f itself. A main reason for using the Gaussian filter in this way is that is has a lot of very nice practical and theoretical properties, as are e.g. described in [25].

When doing feature extraction, a very commonly used filter, f, is a derivative of some order, as discussed in Section 3.2. For computational reasons, the order of operations in (3.2), is as follows

$$I * (g_\sigma * f) \ .$$

If f is a derivative, typical filters are shown in Figure 3.5.

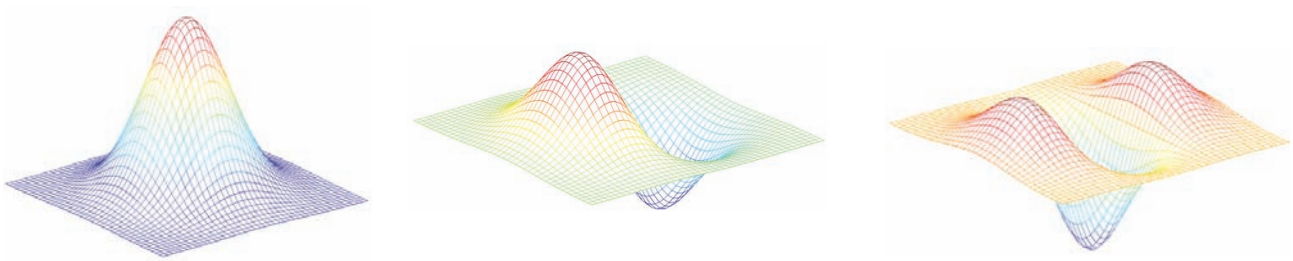

Figure 3.5: An example of a Gaussian kernel, and its derivative and second derivative wrt. the x axis.

3.1.2 Scale and Feature Extraction

This section presents a very rudimentary outline of the vast area of scale space theory [25]. For more information the interested reader is referred to [19, 25]. In the context of this text, the overall message of Section 3.1, is that scale matters for the output of most image algorithms, and should thus be considered.

In the context of feature extraction. Images are often times taking in a non controlled setting, in this case the scale is generally unknown. As an example consider the images in Figure 3.1. Here either of the three images could be taken with the task of extracting features of the leaves, resulting in the need to work at radically different scales. A technique, thus, often used is to do feature extraction at range of scales, with the aim of using the 'desired' scale in on instance.

3.2 The Aperture Problem

As a second general consideration, before specific feature extraction methodologies are introduced, the aperture problem will be covered. As illustrated in Figure 3.6, the location of a feature can only be uniquely determined

[1] Often one or two.

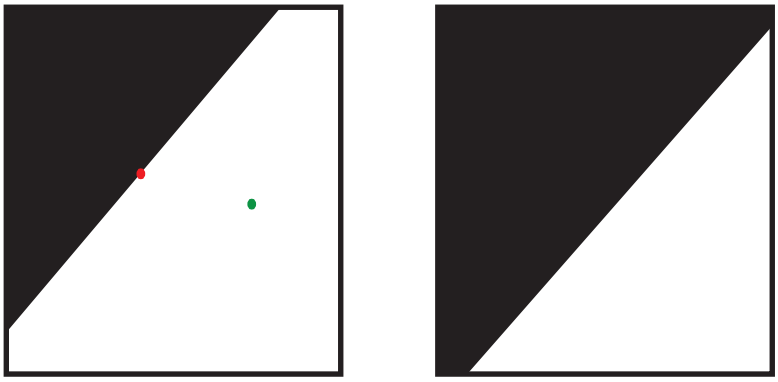

Figure 3.6: An illustration of the aperture problem. Two features (the red and green dots) are annotated in the left image. Consider finding these features in the right image, given that this image is a slightly transformed version of the left. The red dot – located on the line – must be located on the line in the right image, although it is undetermined where. As for the location of the green dot, all we can say is that it must be located in the white area of the right image. This illustrated that we can only uniquely determine the position of a feature in the direction of image gradients.

across an image gradient[2]. This is the statement of the aperture problem.

The implication of the aperture problem for feature extraction, is that we almost always would like a feature to be distinct and well localized. In this context good features are toughs that are located on image gradients. For point features there should be a large gradient in all/both directions, c.f. Figure 3.6. For line features, they should be located along the gradients. When facing noise in the image, the localization becomes more accurate the larger the image gradient, due to the improved signal to noise ration. This is a reason why many feature extraction algorithms build on gradients, or derivatives of some order.

3.3 Harris Corner Detector

A feature often extracted from images are corners. In view of the aperture problem, discussed above, the corner is a feature which has a high gradient in all directions. Probably the most popular corner detection algorithm is the Harris corner detector [13], which will be presented here. To derive this, consider the change in image intensity as a function of a shift in image position $(\Delta x, \Delta y)$, i.e.

$$I(x, y) - I(x + \Delta x, y + \Delta y) \ .$$

In order to have a strong corner this measure should be numerically large in all directions over a small weighted window. This measure is, thus, squared to attain only the numerical size, and is 'averaged' over a small region to ensure it robustness to noise[3]. In practise we achieve this by convolving with a Gaussian g_σ, and we consider the measure

$$c(x, y, \Delta x, \Delta y) = g_\sigma * (I(x, y) - I(x + \Delta x, y + \Delta y))^2 \tag{3.3}$$

[2]This is under the assumption that only local operations are used.
[3]If no smoothing occurs $\mathbf{C}(x, y)$ will also only have rank one.

for each image position x, y. Denoting the image derivatives in the x and y direction by I_x and I_y respectively, $c(x, y, \Delta x, \Delta y)$ can be expanded as follows

$$\begin{aligned} c(x,y,\Delta x,\Delta y) &= g_\sigma * (I(x,y) - I(x+\Delta x, y+\Delta y))^2 \\ &\approx g_\sigma * \left(I(x,y) - \left(I(x,y) + \begin{bmatrix} I_x(x,y) & I_y(x,y) \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} \right) \right)^2 \\ &= g_\sigma * \left(\begin{bmatrix} I_x(x,y) & I_y(x,y) \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} \right)^2 \\ &= \begin{bmatrix} \Delta x & \Delta y \end{bmatrix} \begin{bmatrix} g_\sigma * I_x(x,y)^2 & g_\sigma * I_x(x,y) I_y(x,y) \\ g_\sigma * I_x(x,y) I_y(x,y) & g_\sigma * I_y(x,y)^2 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} \\ &= \begin{bmatrix} \Delta x & \Delta y \end{bmatrix} \mathbf{C}(x,y) \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} \quad , \end{aligned} \tag{3.4}$$

Where $\approx$ denotes a first order approximation and

$$\mathbf{C}(x,y) = \begin{bmatrix} g_\sigma * I_x(x,y)^2 & g_\sigma * I_x(x,y) I_y(x,y) \\ g_\sigma * I_x(x,y) I_y(x,y) & g_\sigma * I_y(x,y)^2 \end{bmatrix} \quad . \tag{3.5}$$

This matrix, $\mathbf{C}(x, y)$, can be interpreted as approximating the average degree of change around the image position (or pixel) x, y. In a sense, it is a weighted variance matrix of the pairwise pixel differences. If we have a corner, with a large gradient in all directions, the rate of change should be large in all direction, implying that (3.4) should be large for $\Delta x, \Delta y$ pointing in any direction. This again implies that $\mathbf{C}(x, y)$ should have two large eigenvalues. If $\mathbf{C}(x, y)$, has one large and one small eigenvalue, the rate of change is large in one direction and not the other, indicating a line[4].

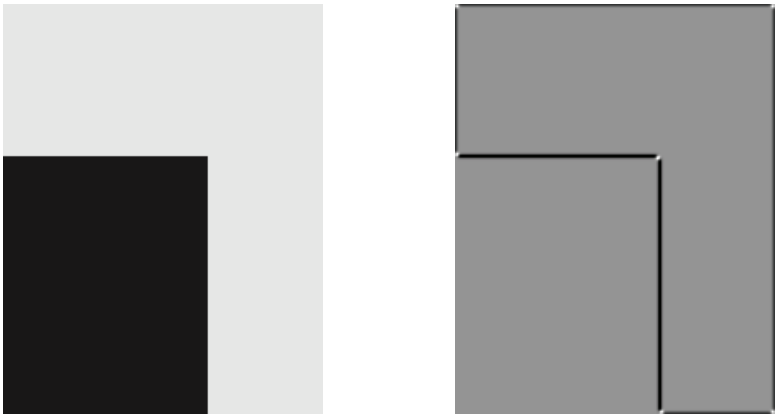

Figure 3.7: An example of $r(x, y)$. The right image is $r(x, y)$ computed on the left image. White is large positive values, black is large negative values, and grey are small values.

To operationalize the above, denote the two eigenvalues of the symmetric positive semidefinite matrix $\mathbf{C}(x, y)$ by λ_1 and λ_2. Then the corners are found at places where both eigenvalues are large, and not just one of them. The latter indicating a line. This holds for the following measure, where large values indicate a corner,

$$r(x,y) = \lambda_1 \lambda_2 - k(\lambda_1 + \lambda_2)^2 \quad , \tag{3.6}$$

for some scalar k, (a typical value for k is 0.06). The reader should verify that $r(x, y)$ will be

- Large and positive for two large eigenvalues.
- Large and negative for one large and one small eigenvalue.
- Small for two small eigenvalues.

[4]Please c.f. to Appendix A for an brief overview of eigenvalues.

This is illustrated in Figure 3.7. The Harris corner detector thus in essence works by finding large *positive* values of $r(x, y)$. The value of $r(x, y)$ can, however, be calculated much more efficiently from $\mathbf{C}(x, y)$ then indicated in (3.6). In deriving this computational trick, the notation is simplified by denoting the elements of $\mathbf{C}(x, y)$ by, a, b, c such that

$$\mathbf{C}(x, y) = \begin{bmatrix} a & c \\ c & b \end{bmatrix} \ .$$

From linear algebra, c.f. Appendix A, it is known that a relationship between the eigenvalues of a 2×2 matrix and the determinant and trace is given by

$$\begin{aligned} \lambda_1 \cdot \lambda_2 &= \det(\mathbf{C}(x, y)) = ab - c^2 \ , \\ \lambda_1 + \lambda_2 &= Trace(\mathbf{C}(x, y)) = a + b \ . \end{aligned}$$

Inserting this into (3.6) gives

$$\begin{aligned} r(x, y) &= \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2 \\ &= \det(\mathbf{C}(x, y)) - k \cdot Trace(\mathbf{C}(x, y))^2 \\ &= ab - c^2 - k(a + b)^2 \ . \end{aligned} \tag{3.7}$$

Which is computed efficiently compared to extracting eigenvalues of $\mathbf{C}(x, y)$.

At the outset the Harris corner detection algorithm thus consist of computing $r(x, y)$ and labelling the positions where

$$r(x, y) > \tau \ , \tag{3.8}$$

for some threshold τ as corners. As argued in Section 3.3.1, non-maximum suppression has to be applied. A typical value for τ is 0.8 times the maximum $r(x, y)$ value for the image in question, i.e.

$$\tau = 0.8 \max_{x,y} r(x, y) \ .$$

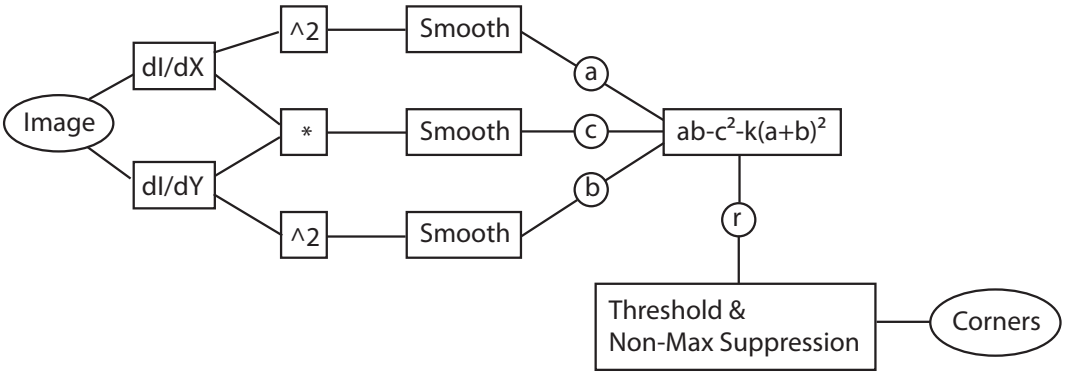

Figure 3.8: A flow diagram for the Harris Corner detection algorithm.

3.3.1 Non-Maximum Suppression

Smoothing is performed when calculating $\mathbf{C}(x, y)$ and subsequently $r(x, y)$. Images are also often smooth to some degree, therefore large values of $r(x, y)$ are usually surrounded by other large values of $r(x, y)$. Thus, if (3.8) is the only criteria used for detecting corners, it will be expected that several pixels will be marked as a corner, in an area around the corner. This is also observed in practise, and seen in Figure 3.7. We would, however, like a unique position of our features, and therefor (3.8), is often supplemented with the criteria of only keeping the local maxima. This is often referred to as non-maximum suppression, in that the non-maximum values are discarded or suppressed. Using a 4-neighborhood, the maximum criteria, (3.8) is supplemented with,

has the following form[5]

$$\begin{aligned} r(x,y) &> r(x+1,y) \quad \wedge \\ r(x,y) &\geq r(x-1,y) \quad \wedge \\ r(x,y) &> r(x,y+1) \quad \wedge \\ r(x,y) &\geq r(x,y-1) \end{aligned} \tag{3.9}$$

It is noted that there is a mix between $>$ and $\geq$ in (3.9), this is done in order to break the tie in case two neighboring pixels have the same values, but are local maximums otherwise. This concludes the Harris corner detection algorithm, and a flow diagram is given in Figure 3.8 along with and example in Figure 3.9.

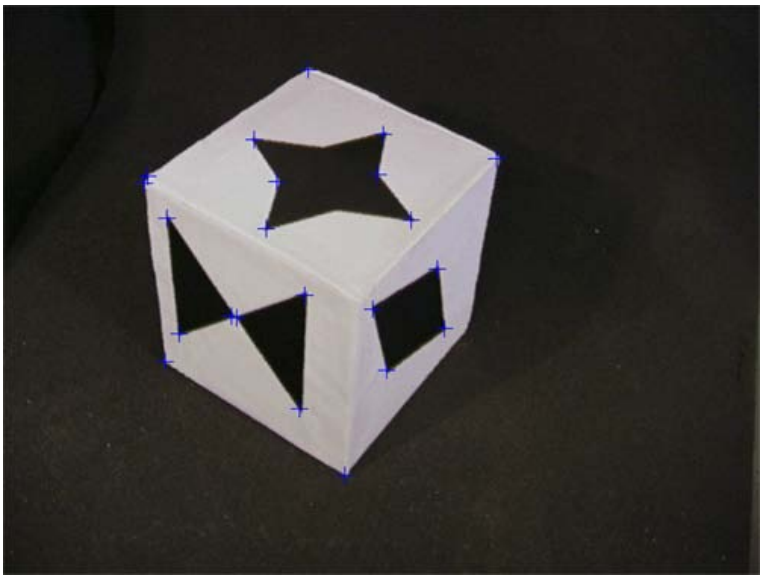

Figure 3.9: An example of applying the Harris corner detector.

3.3.2 Sub-pixel Accuracy*

Lastly, it is noted that once the Harris corners have been found, as described above, and illustrated in Figure 3.8, the position of the extracted corners can be refined to sub-pixel accuracy. This is often done when the extracted features are used to estimate camera geometry, where the extra accuracy makes a meaningful difference. Sub-pixel accuracy can be found by fitting a second order polynomial to the $r(x,y)$ score of the two neighbors, in the x and y direction respectively. The offset δ_x, δ_y from the extracted – integer valued – position, is then found as the optimum of these second order polynomials. Here the formula for δ_x will be derived. The derivation and formula are equivalent for δ_y.

A second order polynomial

$$f(x) = ax^2 + bx + c \ ,$$

should be fitted to the three points

$$f(-1) = r(x-1,y) = a - b + c \quad , \quad f(0) = r(x,y) = c \quad , \quad f(1) = r(x+1,y) = a + b + c \ ,$$

corresponding to the origin of the coordinate system being at (x,y), where the corner is extracted, c.f. Figure 3.10. It can easily be validated that

$$\begin{aligned} a &= \frac{f(1)+f(-1)}{2} - f(0) \\ b &= \frac{f(1)-f(-1)}{2} \\ c &= f(0) \ . \end{aligned}$$

[5] $\wedge$ denotes logical and.

Note, that since $f(0)$ is larger than both $f(1)$ and $f(-1)$, due to $r(x,y)$ being a local optimum, a is negative corresponding to f(x) having a unique optimum. The offset δ_x is found by setting the derivative of $f(x)$ equal to zero, i.e.

$$\begin{aligned}\frac{\partial f(x)}{\partial x} &= 2ax + b = 0 \quad \Rightarrow \\ \delta_x &= -\frac{b}{2a} = -\frac{\frac{f(1)-f(-1)}{2}}{2\left(\frac{f(1)+f(-1)}{2} - f(0)\right)} = -\frac{f(1)-f(-1)}{2f(1)+2f(-1)-4f(0)} \ .\end{aligned}$$

This scheme works well in practice, and the corner position, with sub-pixel accuracy, is given by $(x+\delta_x, y+\delta_y)$. However, if $r(x-1,y)$, $r(x,y)$ and $r(x+1,y)$ are too close in value, a may become so small that the scheme becomes numerically unstable. Thus if the absolute value of δ_x or δ_y becomes larger than 1, no offset is used, i.e. $\delta_x = 0$ and/or $\delta_y = 0$.

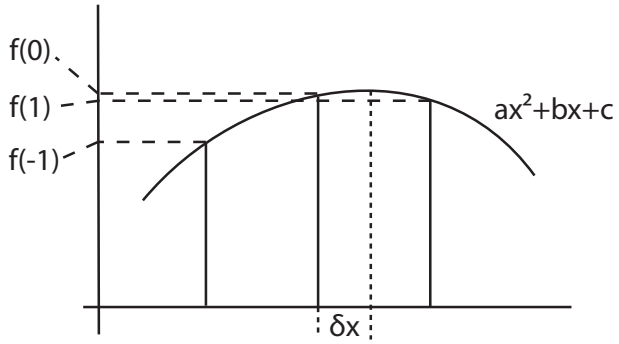

Figure 3.10: A schematic illustration of fitting a second order polynomial in order to find the sub-pixel offset Δ_x.

3.4 Blob Detection

Considering again the aperture problem, another entity which has a high gradient in all directions is the blob, i.e. an image location where the second derivative is large in both/all directions. For a 2D image the second order derivative or hessian, $\mathbf{H}$, is a 2×2 matrix

$$\mathbf{H} = \begin{bmatrix} I_{xx} & I_{xy} \\ I_{xy} & I_{yy} \end{bmatrix} \ ,$$

Here the the following short hand notation is used for ease of read, where $I(x,y)$ is the image intensity at position (x,y),

$$I_{xx} = \frac{\partial^2 I(x,y)}{\partial x^2} \quad , \quad I_{xy} = \frac{\partial^2 I(x,y)}{\partial xy} \quad , \quad I_{yy} = \frac{\partial^2 I(x,y)}{\partial y^2}$$

A main issue in designing a blob detector from the Hessian, is how to measure the size of the Hessian, such that a blob is determined in the presence of a large Hessian. In this regard there are two obvious or typical choices, namely the determinant and the trace of the Hessian, corresponding to the product and sum of the Hessian's two eigenvalues respectively. In the following these eigenvalues will be denoted by λ_1 and λ_2. The two measures are given by

$$\det(\mathbf{H}) = \det\left(\begin{bmatrix} I_{xx} & I_{xy} \\ I_{xy} & I_{yy} \end{bmatrix}\right) = I_{xx}I_{yy} + I_{xy}^2 = \lambda_1\lambda_2 \ . \tag{3.10}$$

$$Trace(\mathbf{H}) = Trace\left(\begin{bmatrix} I_{xx} & I_{xy} \\ I_{xy} & I_{yy} \end{bmatrix}\right) = I_{xx} + I_{yy} = \lambda_1 + \lambda_2 \ . \tag{3.11}$$

$$\tag{3.12}$$

These measures are then formed into detector by thresholding, doing non-maximum suppression and possibly determining sub-pixel accuracy as described in Section 3.3 (Section 3.3.1 and Section 3.3.2 specifically). It is noted that the trace of the hessian is also called the Laplacian, and is denoted by

$$\nabla^2 I = I_{xx} + I_{yy} = I * (\frac{\partial^2 g_\sigma}{\partial x^2} + \frac{\partial^2 g_\sigma}{\partial y^2}) \ . \tag{3.13}$$

The lapalacia filter is illustrated in Figure 3.11.

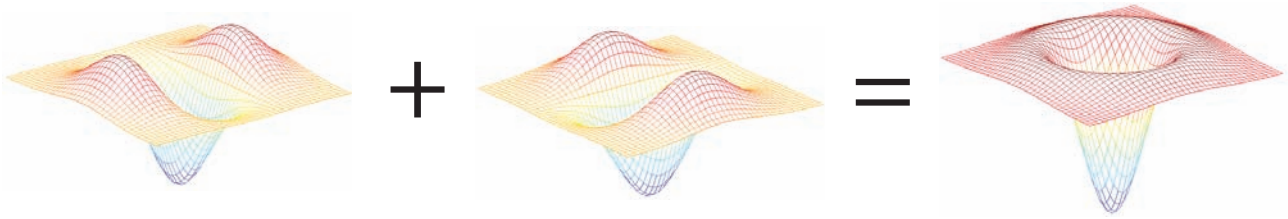

Figure 3.11: The Lapalcian (trace of Hessian) filter, seen as the addition of the $\frac{\partial^2 g_\sigma}{\partial x^2}$ and $\frac{\partial^2 g_\sigma}{\partial y^2}$ filters.

3.4.1 Difference of Gaussian (DoG)

Another very popular blob detector is the difference of Gaussian(DoG) detector, which is among others part of the very successful SIFT detector and descriptor framework [20]. The DoG detector applied to an image, $I(x, y)$, is formed by taking the difference of the image convolved with two Gaussian kernels of different width or scale, i.e.

$$I * g(s \cdot \sigma) - I * g_\sigma = I * \big(g(s \cdot \sigma) - g_\sigma\big) \ , \tag{3.14}$$

where s is a scalar and σ is the scale of the operation. An example of this filter is shown in Figure 3.12. A main motivation for the DoG filter is that it is a close approximation to the Laplacian, as seen by comparing the DoG filter in Figure 3.12 to the Laplacian in Figure 3.11. This approximation holds best for $s \approx 1.6$. The DoG filter is also used for other values of s, where it still keeps its Mexican hat like shape, which intuitively corresponds to what we are looking for when searching for a blob.

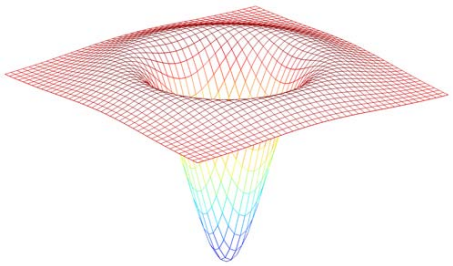

Figure 3.12: The Difference of Gaussian(DoG). Note the similarity to the kernel in Figure 3.11. In scale space the DoG detector is computed by first convolving with Gaussian kernels, c.f. Figure 3.13 , and the subtracting.

A reason for DoG detectors being popular, is that they are very efficient to implement in a scale space frame work, c.f. Section 3.1. That is, if we are to search for features at multiple scales, then the DoG feature detector is straight forward to implement. This is because the DoG occur as the difference between filter responses at different scales, and as such DoG in scale space are computed by convolving the image, I, with Gaussian kernels at different scales, and then computing the difference, c.f. Figure 3.13.

When the DoG features are extracted in scale, as done in the SIFT feature [20], then the non-maximum is not only done in the image plane as in Section 3.3.1, but also by considering a scale down and a scale up. This ensures that a feature has not only found it's right location but also it's right scale. The resulting features are seen in Figure 3.14.

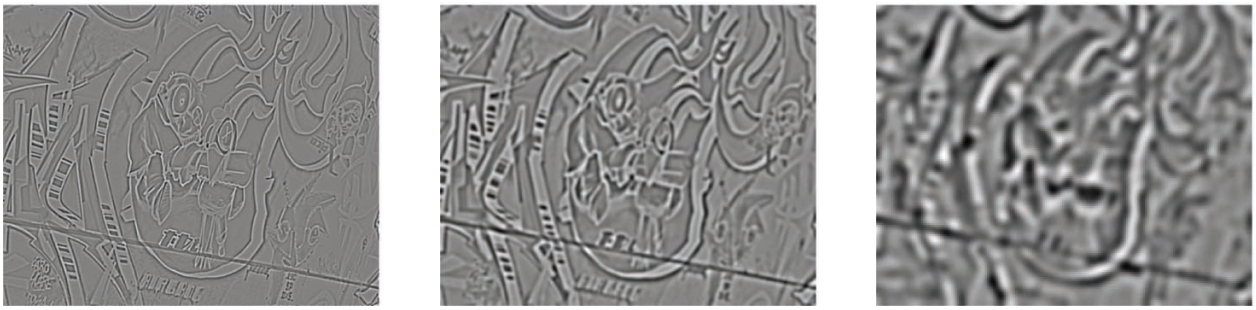

Figure 3.13: Result of applying the DoG filter to the image of Figure 3.14 at different scales, or places in the image pyramid.

Figure 3.14: An example of extracting the DoG features across scale corresponding to the so-called SIFT features [20].

3.5 Canny Edge Detector

Apart from point features we often want to extract edges or lines in an image. The most popular way of doing this is via the Canny edge detector [4], also sometimes referred to as the Canny-Deriche detector, because R. Deriche's method of for efficiently smoothing with a Gaussian kernel is often used [23][6].

The Canny edge detector uses the gradient magnitude, $\sqrt{I_x^2 + I_y^2}$, as an edge measure, c.f. Figure 3.15-middle-left. Non-maximum suppression is also preformed on this measure, c.f. Section 3.3.1. But differently from the point case, it should only be required that the and edge point is maximum *perpendicular* to the edge, i.e. in the direction of the gradient, i.e. (I_x, I_y) *and* $(-I_x, -I_y)$. The reason being that lines are extracted, and here we are looking for linked points, and as such should not only consider the only point on an edge with largest derivative. See Figure 3.15-middle-right.

As in the point case, thresholding is also applied, but here two thresholds are used, τ_1 and τ_2 with $\tau_1 > \tau_2$. The idea is that all pixels with a gradient magnitude larger than τ_1 are labelled as edges – provided that they pass the non-maximum suppression criteria. Whereas pixels with a gradient magnitude between τ_1 and τ_2 are labelled as edges, only *if* they are part of a line where part of it is above the τ_1 threshold. Again under the assumption that all edge pixels pass the non-maximum suppression criteria. The motivation is that if parts of a line becomes weak, e.g. due to noise, it should still be included. This can be seen as a sort of hysteresis. In practise this is done by extracting all possible edge pixels with a gradient magnitude above τ_2, c.f. Figure 3.15-bottom-left, and then only keeping the line segments where at least one pixel has a gradient magnitude above

[6] This technicality is not covered here.

τ_1, c.f. Figure 3.15-bottom-right. The latter operation can be performed via a connected components algorithm. The resulting edge segments are the output of the Canny edge detector.

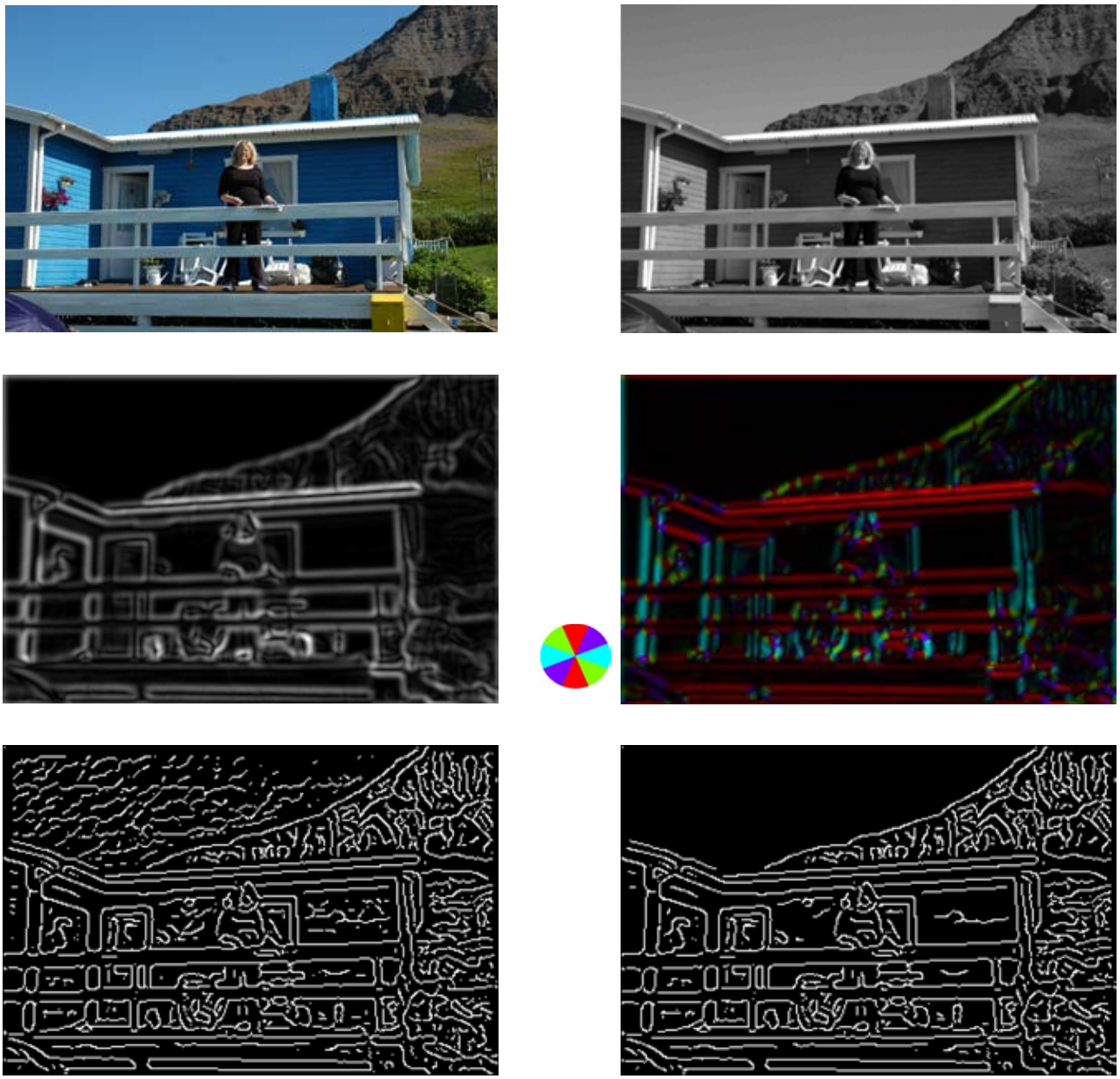

Figure 3.15: Illustration of the Canny edge detector. **Top:** The original color image, and the gray scaled version on which the operation is performed. **Middle:** The gradient magnitude $I_x^2 + I_y^2$, and the gradient magnitude with the orientations used to do non-maximum suppression. There is also a small color wheel specifying the relation between orientation and color. **Bottom:** The edges, passing the non-maximum suppression criteria, and with a gradient magnitude above τ_2, to the left. To the right the edge segments to from the left image with at least one pixel with a gradient magnitude above τ_1. The bottom right image is the result of the Canny edge detection on the top image.

Chapter 4

Image Correspondences

As mentioned in Chapter 3, this chapter is concerned with the correspondence problem. The correspondence problem is basically: *Find the correspondence between two – or more – images. Understood as determining where the same physical entities are depicted in the different images in question.* See Figure 4.1. This is a fundamental problem in image analysis, and a good general solution does not exist, although much progress is being, and has been made. This also implies that there is a multitude of solution schemes for finding the correspondence between images, which we will come nowhere near covering here. Here two methods will be presented for matching point features between images. If more than two images are to be matched, as is often the case, pair correspondences will be aggregated to a correspondence of the whole image set. It should also be mentioned that the correspondence problem is also know as tracking, registration and feature matching to mention a few of the other common used phrases.

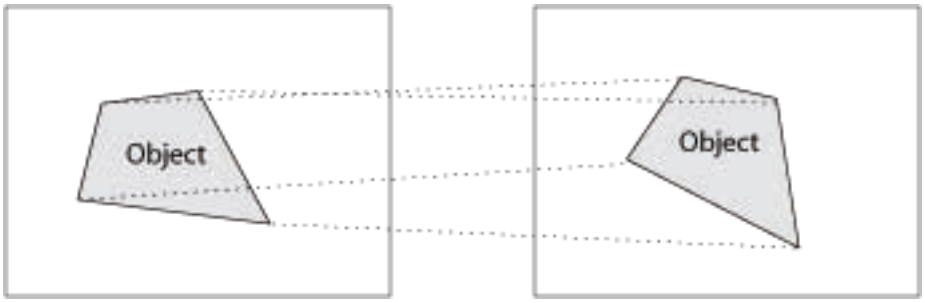

Figure 4.1: The general correspondence problem, where the relationships between the same entities of an object is mapped.

4.1 Correspondence via Feature Matching

As mentioned above, the method for finding the correspondence between images, which is mainly in focus here, is feature matching. Feature matching is a three stage technique composed of

1. Extract a number of, hopefully salient, feature from the images. Here it is assumed that these features are points, c.f. Chapter 3.
2. For each feature compute or extract a descriptor. This descriptor is typically based on a small window around the feature.
3. The correspondence between the two sets of features, and thus the images, is found by minimizing some distance between the two sets of descriptors. See Figure 4.2.

As might be expected, there are a multitude of ways of doing this. There are however some general comments that can be made as to what in general characterizes good strategies. As for what features to extract – which usually boils down to how to extract them, the features should be **unique** in the sense that it is clear where the feature is. This should be seen in relation to the aperture problem, as discussed in Section 3.2. For

if it is unclear where the feature is, then it will be unclear what went where, and as such violating the problem statement of finding the position of the same underlying identity in two or more images. This also holds practical implications for many applications of the correspondence problem, such as 3D reconstruction, estimation of camera movement etc.

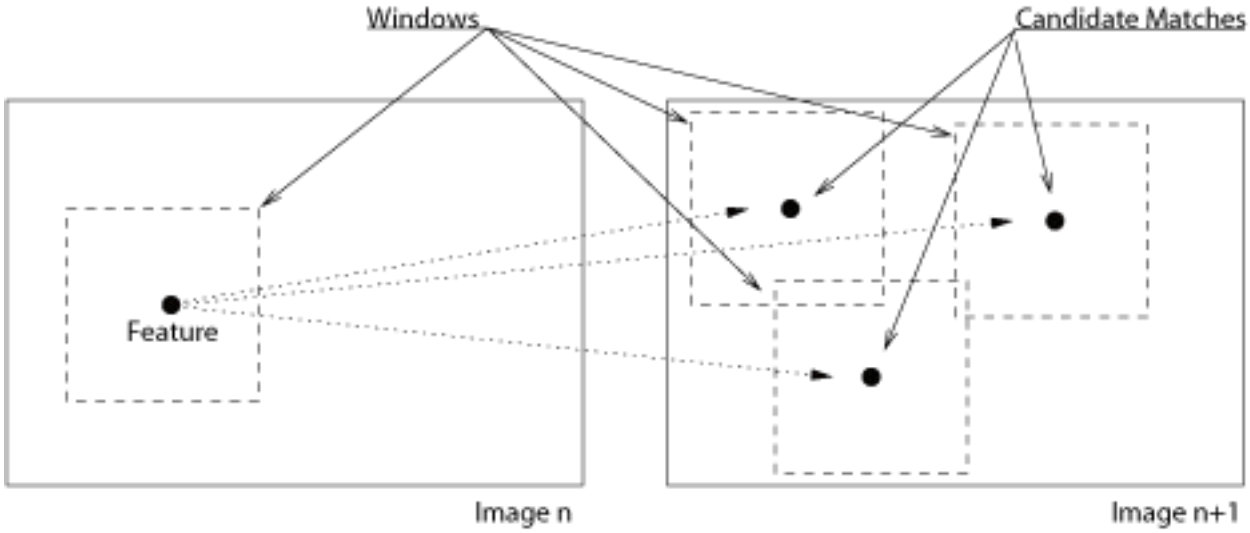

Figure 4.2: Matching features by finding the features with the most similar neighborhoods — defined by a window around the feature. Here 'Most Similar' is defined by the feature descriptors and a distance between them.

Another important issue to consider is that the features should be **repeatable**, in the sense that we should be able to find the same features if the images changed slightly — whatever 'slightly' means. The reasoning being that we should hopefully extract features corresponding to the same underlying 3D entities in different images. The same thing will seldom look exactly the same in two images due to changes in illumination, view point, internal camera setting and plain sensor noise. Thus some flexibility should be incorporated into ones feature extractor and e.g. requiring perfect correspondence with an image mask will in general not be a good idea.

Lastly the features should also be **distinguishable**, i.e. it should be plausible that we can find that exact image again in another image. The reason for this is obvious, since this is the task we want to perform with the features. And naturally the chosen image descriptors should capture this distinguishability. In the following two strategies for doing feature extraction and matching will be presented, to give the reader a feel for how such things could be done. It should, however, be kept in mind that this is only a small subset of the available methods. This is especially true if the problem domain is very limited, in which case special tailored solutions can be made. As an example consider finding the targets in Figure 4.3, here the associated number would be a very good descriptor, and a target like convolution kernel be a good feature extractor.

4.2 Feature Descriptor Examples

As mentioned in Section 4.1, after features are extracted, a descriptor is computed for each feature. These descriptors are then compared via some measure, by which the matching is made. Here two such descriptors, and associated measure are presented. These descriptors are calculated based on an area around the feature as illustrated in Figure 4.2.

4.2.1 Correlation

A typical similarity measures between image patches is correlation. This corresponds to using the raw numerical values of the window around the feature as a descriptor. Using correlation can be interpreted as seeing one patch as a kernel and convolving the other image with it. Correlation between two image patches, P_1 and P_2,

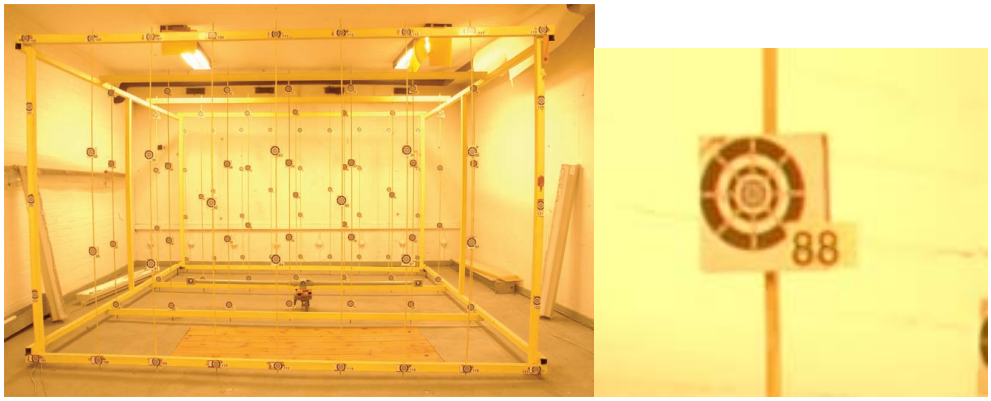

Figure 4.3: A image of a test field for calibrating cameras to the left. To the right an enlarged image section of the left image depicting the targets.

can be obtained as follows: [1]

1. Arrange the elements of P_1 and P_2 into vectors $\vec{x}_1$ and $\vec{x}_2$.
2. Calculate the mean of each vector, $\vec{x}_1$ and $\vec{x}_2$, μ_1 and μ_2, i.e. :

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i \quad .$$

3. Calculate the variance of each vector, $\vec{x}_1$ and $\vec{x}_2$, var_1 and var_2, i.e. :

$$var = \frac{1}{n-1}\sum_{i=1}^{n} (x_i - \mu)^2 \quad .$$

4. Calculate the covariance, cov, between the two vectors, $\vec{x}_1$ and $\vec{x}_2$, i.e.:

$$cov = \frac{1}{n-1}\sum_{i=1}^{n} (x_{1_i} - \mu_1)(x_{2_i} - \mu_2) \quad .$$

5. Then the cross-correlation, ρ, is given by:

$$\rho = \frac{cov}{\sqrt{var_1 \cdot var_2}} \quad . \tag{4.1}$$

Using correlation is a classical way of doing image feature matching, and works well in many – usually simpler cases – it however has some drawbacks. First, if there is no or little variance in an image, the denominator of (4.1) will be small or zero. In the later case (4.1) is undefined, in the first case noise will dominate. This will, however, seldom be an issue, since features with high gradients are chosen, and as such the local variance will be high. Secondly covariance is rather sensitive to rotation of the image, and other changes of view point. This is seen since pixels are compared one to one. An example of when this will work well or not is illustrated in Figure 4.4

4.2.2 SIFT Descriptors

A more complicated, but also more powerful way of computing feature descriptors are the so called SIFT descriptors [20]. They are also based on a square patch around the feature, as in the correlation case. To address the issue of invariance to rotation, this image patch is typically not aligned with the coordinate system of the image, instead it is aligned with the local image gradient, as illustrated in Figure 4.5. This has the effect, that if the image is rotated so is the image gradient, and as such this 'trick' ensures invariance to rotation, which is a benefit for natural images such as that of Figure 4.4. If the images are known *not* to rotate this 'trick' should

Figure 4.4: Three images of a building here at DTU. It is relatively easy to match the left and middle image via correlation, since the image motion is moderate and there is no or little rotation. Matching the left and right image will likely give problems using correlation due to the large amount of rotation.

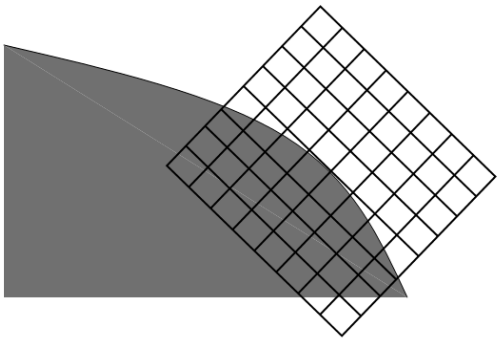

Figure 4.5: Here it is illustrated how the local patch is aligned with the local gradient in the image.

not be used, in that it also introduces extra ambiguity into the descriptor. I.e. if it is known that matches should look similar and are not rotated, this information should be used. Using such a rotated patch implies the need for interpolation – typically bilinear – in order to get the values on the image patch grid.

Once the patch rotation and interpolated values have been determined, the descriptor is calculated as indicated in Figure 4.7[2]. First the gradient of the patch is calculated, then this field is transformed into a modulus and argument representation, i.e.

$$mod = \sqrt{\frac{\partial I}{\partial x}^2 + \frac{\partial I}{\partial y}^2} \quad , \quad arg = atan2(\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}) \ .$$

The patch is then divided into a 4 by 4 grid. For each cell in this grid a 8 bin histogram is computed for the angles weighted by the modulus. That is for each pixel, the angle or argument gives one of the eight bins in the histogram, and this bin is then incremented by the modulus. This indicates that we have 8 values for each cell and we have $4 \cdot 4 = 16$ cells giving $8 \cdot 16 = 128$ values in all. It is these 128 values which compose the SIFT descriptor. Please note, that minor details have been left out, so for a more in depth description refer to [20]. These 128 dimensional SIFT descriptors need to be compared, in order to complete the matching. Noting that these 128 vector represent a histogram, they are seen to be χ^2-distributed, and as such a correct/good distance between two descriptors, a and b, is then,

$$d(a,b) = 2\sum_{i=1}^{128} \frac{(a_i - b_i)^2}{a_i + b_i} \ . \tag{4.2}$$

It should be noted that all entries of a and b are positive.

4.3 Matching via Descriptors

Following the outline of Section 4.1, the correspondence problems is then solved by matching pairs of descriptors, one from each image. To formalize, denote the two set of descriptors by A and B, then we want to find or

[1] There are faster numerical schemes then this, but this best conveys the idea.

[2] The image patch is not aligned to the local image gradient. This figure should only illustrate the computation of the SIFT descriptor.

compute a set of matches, $m_{ij} = (A_i, B_j)$, such that each element of A and B *only appears once*. It is noted, that the size of A and B are usually not the same. One way of doing this is by minimizing,

$$\min \sum_{m_{ij}} d(A_i, B_i) \ , \tag{4.3}$$

where $d(\cdot, \cdot)$ is the appropriate distance measure. This can be cast, with a bit of manipulation, as a linear assignment problem, which have efficient solvers c.f. e.g. [16]. Another strategy which often works better, in that it removes lower quality matches, is to define

$$Best(A_i) = \min_j d(A_i, B_i) \ , \tag{4.4}$$

i.e. $Best(A_i)$ is the best match for A_i in B. A similar measure is defined for the B_j. Then a match, m_{ij}, is included if for a given i and j

$$Best(A_i) = B_j \quad and \quad Best(A_j) = B_i \ . \tag{4.5}$$

That is A_i and B_j are each others most similar descriptors. As for efficiency of computation, the calculations in (4.4) can be somewhat time consuming when a lot of features are present in each image. In that these calculations have to be made for each possible descriptor pair, in the basic form. This is often done. However, sometimes more efficient data structures such as KD-trees have been used, especially the structure in [2] has been reported used.

4.4 Constrains in Search Space

An enhancement to the basic approach described in Section 4.1, is to constrain the search space. This is another way of saying, that only some features in one image can match a given feature in the other. There are two main reasons for this, firstly to limit the number of computations required. Secondly this is a way to incorporate prior knowledge or assumptions of how the images are formed, reducing the number of errors.

A very popular constrain is on the numerical size of the optical flow. Another is if the camera geometry is known, see Chapter 2, where the search space for a feature in one image can be constrained to a line. This constraint is expressed by the fundamental or essential matrix. This is a very popular approach which is often what makes a matching algorithm work in practice, see e.g. Figure 4.6.

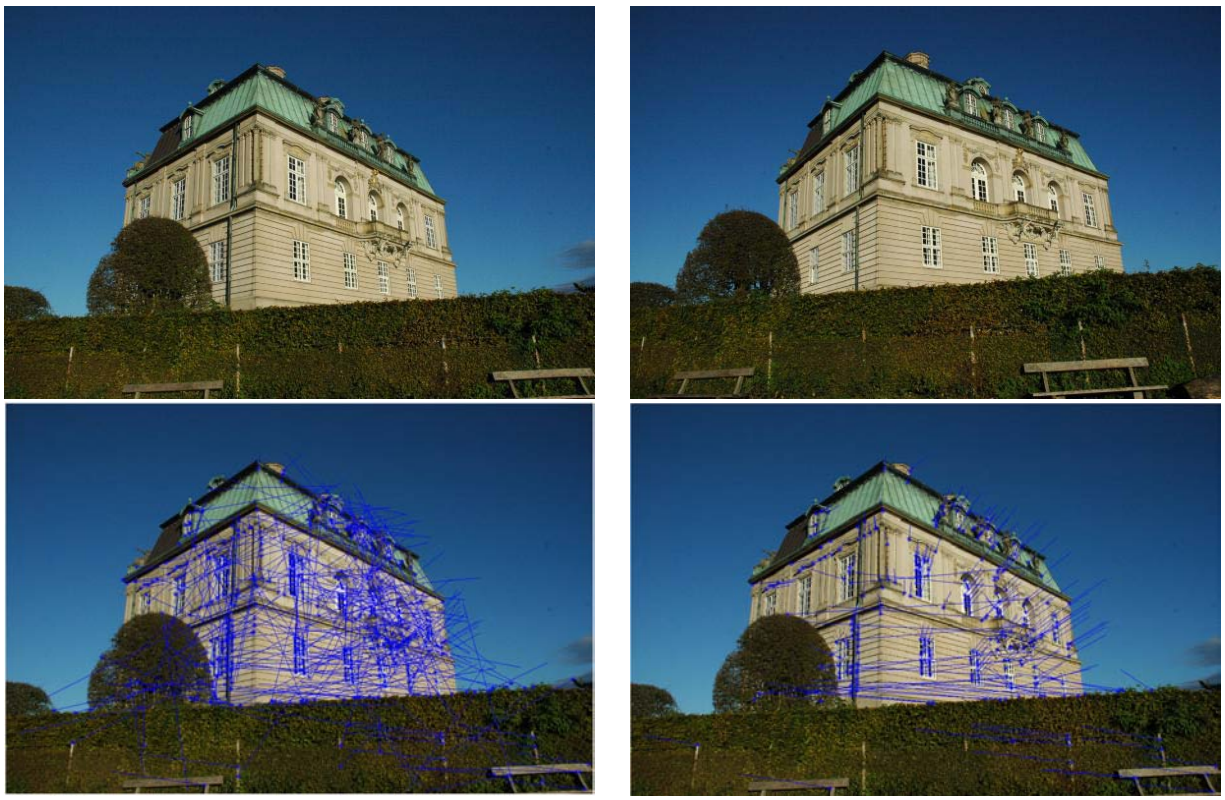

Figure 4.6: An example of the correspondence problem, where the two top images are to be matched. Features are extracted as Harris corners, and matched via correlation. In the bottom row the blue arrows indicates the displacements needed to get a match. On the left is an unconstraint matching, where as the camera geometry is used to the right.

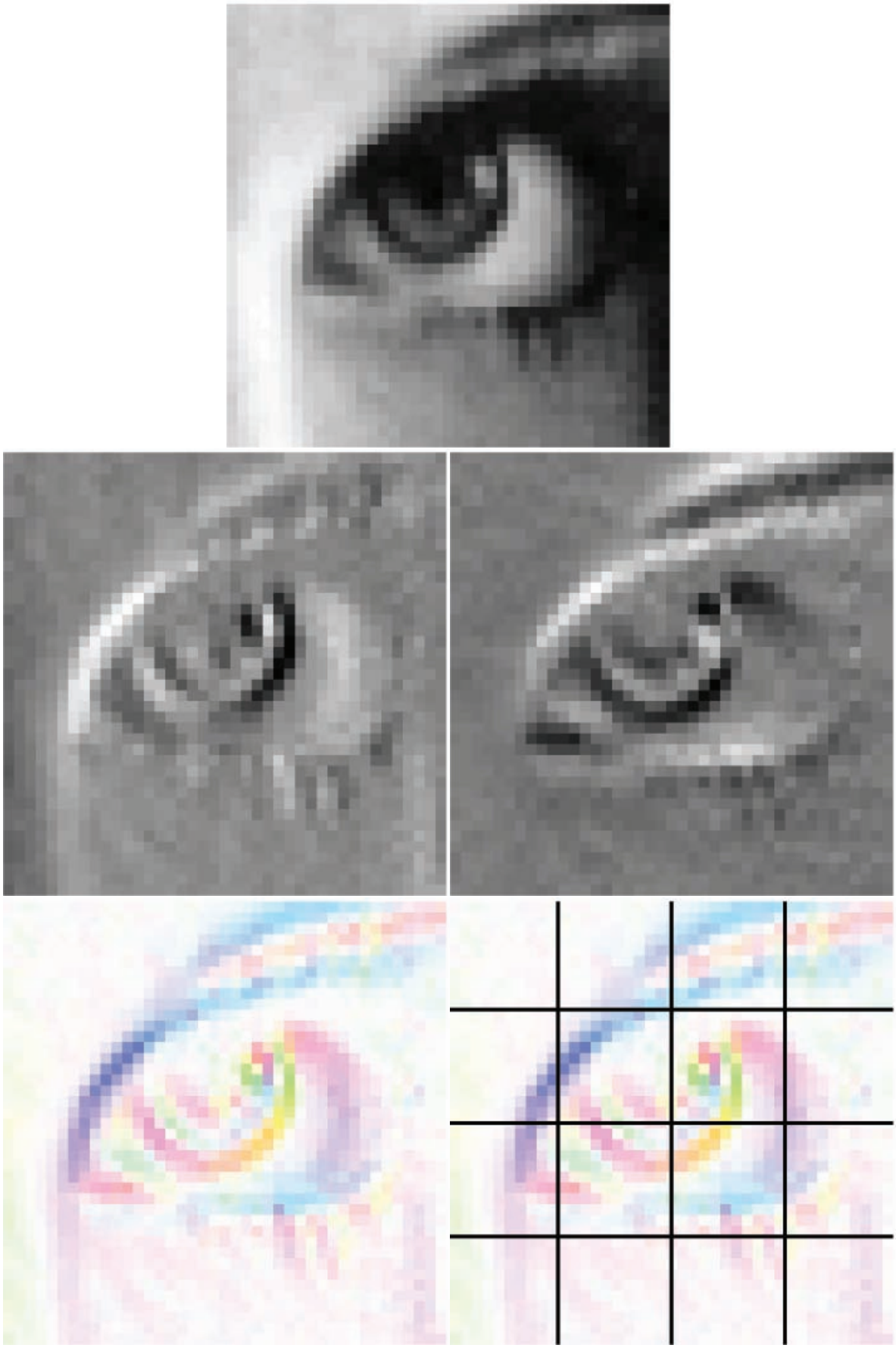

Figure 4.7: Illustration of how a SIFT descriptor is calculated. Given the patch, top row, the gradient is calculated, middle row. The gradient field is formed into a modulus and argument representation, bottom left. Lastly, bottom right, the patch is divided into a 4 by 4 grid where an 8 bin histogram is calculated over the angles weighted by the modulus. In the last row the angle is denoted by the color.

Part III

Appendices

Appendix A

A few Elements of Linear Algebra

This is an outline of a few elements from linear algebra used in this text, and which experience has shown needs explanation. This is, however, not a course-note in linear algebra. So if linear algebra is not clear and present[1], please look it up, e.g. in [6, 8], since it is a basis for much of the subject presented in this text.

A.1 Basics of Linear Algebra

This section presents a very concise overview of the basics of linear algebra, intended as a reference and brush up, e.g. if it is some time since the reader last worked with this subject.

1. A vector $\mathbf{v}$ is a collection of scalars, v_i in a row or a column, i.e.

$$\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} .$$

2. A vector can be transposed, switching between rows and columns i.e.

$$\mathbf{v}^T = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}^T = \begin{bmatrix} v_1, & v_2, & \ldots, & v_n \end{bmatrix} .$$

3. Two vectors $\mathbf{v}$ and $\mathbf{w}$ can be multiplied as follows

$$\mathbf{v^T w} = \begin{bmatrix} v_1, & v_2, & \ldots, & v_n \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} = \sum_{i=1}^{n} v_i w_i \ ,$$

   Assuming that they have the same length — here n.

4. Thus the two norm, or Euclidian distance, of a vector, $\mathbf{v}$ can be written as

$$||\mathbf{v}||_2 = \sqrt{\sum_{i=1}^{n} v_i^2} = \sqrt{\mathbf{v^T v}} \ .$$

[1] The first section is a small brush up though.

5. The angle, α, between two vectors $\mathbf{v}$ and $\mathbf{w}$ is given by

$$\cos(\alpha) = \frac{\mathbf{v^T w}}{||\mathbf{v}|| \cdot ||\mathbf{w}||} \ .$$

So if two vectors, $\mathbf{v}$ and $\mathbf{w}$, are orthogonal (perpendicular or 90 degrees to each other)

$$0 = cos(\frac{\pi}{2}) = \mathbf{v^T w} \ .$$

6. A Matrix $\mathbf{A}$ as an array of scalars, i.e (here an m by n matrix)

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \ ,$$

7. The transposed of a $m \times n$ matrix, $\mathbf{A}$, is a $n \times m$ matrix, $\mathbf{A^T}$. Here element a_{ji} in $\mathbf{A^T}$ is equal to element a_{ij} in $\mathbf{A}$.

8. We can view a matrix like a collection of vectors, e.g. each row, such that $\mathbf{a}_{i:}^T = [a_{i1}, \dots, a_{in}]^T$, and the $\mathbf{A}$ from above has the following form

$$\mathbf{A} = \begin{bmatrix} \mathbf{a}_{1:}^T \\ \mathbf{a}_{2:}^T \\ \vdots \\ \mathbf{a}_{m:}^T \end{bmatrix} \ .$$

Thus multiplication of a matrix by and a vector, $\mathbf{Av}$, is given by the vector times the individual rows (here the vector is multiplied to the right), i.e.

$$\mathbf{Av} = \begin{bmatrix} \mathbf{a}_{1:}^T\mathbf{v} \\ \mathbf{a}_{2:}^T\mathbf{v} \\ \vdots \\ \mathbf{a}_{m:}^T\mathbf{v} \end{bmatrix} = \begin{bmatrix} \sum_{j=1}^m a_{1j}v_j \\ \sum_{j=1}^m a_{2j}v_j \\ \vdots \\ \sum_{j=1}^m a_{nj}v_j \end{bmatrix} \ .$$

Here the vector $\mathbf{v}$ has to have length m and the resulting product will be a vector of length n (here $\mathbf{A}$ is an m by n matrix). Similarly we can multiply a vector with a matrix, $\mathbf{v^T A}$, (here the vector is multiplied to the left), as follows, denoting the the rows of $\mathbf{A}$ by $\mathbf{a}_{:j}$,

$$\mathbf{v^T A} = \begin{bmatrix} \mathbf{v}^T\mathbf{a}_{:1} & \mathbf{v}^T\mathbf{a}_{:2} & \dots & \mathbf{v}^T\mathbf{a}_{:n} \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^n v_i a_{i1} & \sum_{i=1}^n v_i a_{i2} & \dots & \sum_{i=1}^n v_i a_{in} \end{bmatrix} \ .$$

Here the vector $\mathbf{v}$ has to have length n and the resulting product will be a vector of length m (here $\mathbf{A}$ is an m by n matrix).

9. As matrix vector multiplication can be decomposed into vector-vector multiplication, by reducing the matrix into a collection of vectors, so can matrix-matrix multiplication, $\mathbf{AB}$. This is done by reducing $\mathbf{A}$ into it's rows, and $\mathbf{B}$ into it's columns. i.e.

$$\begin{aligned} \mathbf{A} &= \begin{bmatrix} \mathbf{a}_{1:}^T \\ \mathbf{a}_{2:}^T \\ \vdots \\ \mathbf{a}_{m:}^T \end{bmatrix} \ , \\ \mathbf{B} &= \begin{bmatrix} \mathbf{b}_{:1} & \mathbf{b}_{:2} & \dots & \mathbf{b}_{:n} \end{bmatrix} \ . \end{aligned}$$

Then

$$\begin{aligned}\mathbf{AB} &= \begin{bmatrix} \mathbf{a}_{1:}^T\mathbf{b}_{:1} & \mathbf{a}_{1:}^T\mathbf{b}_{:2} & \dots & \mathbf{a}_{1:}^T\mathbf{b}_{:l} \\ \mathbf{a}_{2:}^T\mathbf{b}_{:1} & \mathbf{a}_{2:}^T\mathbf{b}_{:2} & \dots & \mathbf{a}_{2:}^T\mathbf{b}_{:l} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{a}_{m:}^T\mathbf{b}_{:1} & \mathbf{a}_{m:}^T\mathbf{b}_{:2} & \dots & \mathbf{a}_{m:}^T\mathbf{b}_{:l} \end{bmatrix} \\ &= \begin{bmatrix} \sum_{k=1}^n a_{1k}b_{k1} & \sum_{k=1}^n a_{1k}b_{k2} & \dots & \sum_{k=1}^n a_{1k}b_{kl} \\ \sum_{k=1}^n a_{2k}b_{k1} & \sum_{k=1}^n a_{2k}b_{k2} & \dots & \sum_{k=1}^n a_{2k}b_{kl} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{k=1}^n a_{mk}b_{k1} & \sum_{k=1}^n a_{mk}b_{k2} & \dots & \sum_{k=1}^n a_{mk}b_{kl} \end{bmatrix}\end{aligned}$$

Here the dimensions of $\mathbf{A}$ are $m \times n$ and the dimensions of $\mathbf{B}$ are $n \times l$. It is noted the the number of columns of $\mathbf{A}$ and thee number of rows in $\mathbf{B}$ should be equal, for this multiplication to be possible/well-defined/allowed. This is due to the vectors $\mathbf{a}_{i:}$ and $\mathbf{b}_{:j}$ having to have the same length to be multiplied.

10. A matrix or a vector can be multiplied by a scalar, s, by multiplying all the elements with the scalar in question, i.e.

$$s\mathbf{v} = \begin{bmatrix} sv_1 \\ sv_2 \\ \vdots \\ sv_n \end{bmatrix} .$$

11. Addition of matrices or vectors, of the same dimensions, is done by adding the individual elements, e.g. (assuming the dimensions of $\mathbf{A}$ and $\mathbf{B}$ are $m \times n$)

$$\begin{aligned}\mathbf{A}+\mathbf{B} &= \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} + \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \dots & b_{mn} \end{bmatrix} \\ &= \begin{bmatrix} a_{11}+b_{11} & a_{12}+b_{12} & \dots & a_{1n}+b_{1n} \\ a_{21}+b_{21} & a_{22}+b_{22} & \dots & a_{2n}+b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}+b_{m1} & a_{m2}+b_{m2} & \dots & a_{mn}+b_{mn} \end{bmatrix} .\end{aligned}$$

12. Common rules for matrix — and thus vector[2] — arithmetic are (given that the elements have the appropriate dimensions)

- Matrix multiplication is associative:

$$\mathbf{A}(\mathbf{BC}) = (\mathbf{AB})\mathbf{C} .$$

- Matrix multiplication is distributive:

$$\begin{aligned}\mathbf{A}(\mathbf{B}+\mathbf{C}) &= \mathbf{AB}+\mathbf{BC} , \\ (\mathbf{A}+\mathbf{B})\mathbf{C} &= \mathbf{AC}+\mathbf{BC}\end{aligned}$$

- Matrix multiplication is compatible with scalar multiplication:

$$c(\mathbf{AB}) = (c\mathbf{A})\mathbf{B} = (\mathbf{A}c)\mathbf{B} = \mathbf{A}(c\mathbf{B}) = \mathbf{A}(\mathbf{B}c) = (\mathbf{AB})c .$$

- Matrix multiplication is *not* commutative in general, i.e.

$$\mathbf{AB} \neq \mathbf{BA} .$$

[2] A vector can be seen as a matrix with one dimension equal to one.

- The order of multiplication is reversed when transposing a multiplication, i.e.

$$(\mathbf{AB})^{\mathbf{T}} = \mathbf{B}^{\mathbf{T}}\mathbf{A}^{\mathbf{T}} \quad , \quad (\mathbf{ABC})^{\mathbf{T}} = \mathbf{C}^{\mathbf{T}}\mathbf{B}^{\mathbf{T}}\mathbf{A}^{\mathbf{T}} \ .$$

13. The inverse of square $n \times n$ matrix, $\mathbf{A}$, is a $n \times n$ matrix, $\mathbf{A^{-1}}$ with the properties that

$$\mathbf{A^{-1}A} = \mathbf{AA^{-1}} = \mathbf{I} \ ,$$

Where $\mathbf{I}$ is the identity matrix, i.e. a matrix with all zeros, except for the diagonal which is one. It is noted, that $\mathbf{A^{-1}}$ only exist if $\mathbf{A}$ has full rank, equivalent to it having a determinant different from zero.

14. The trace of a $n \times n$ square matrix, $\mathbf{A}$ is the sum of the diagonal elements, i.e.

$$Trace(\mathbf{A}) = \sum_{i=1}^{n} a_{ii} \ .$$

In terms of eigen values, λ_i, the trace is equal to the sum, i.e. $Trace(\mathbf{A}) = \sum_{i=1}^{n} \lambda_i$.

15. The determinant of a $n \times n$ square matrix, $\mathbf{A}$ is formally given by

$$\det(A) = \sum_{\sigma \in S_n} sgn(\sigma) \prod_{i=1}^{n} A_{i,\sigma(i)} \ .$$

where σ is a permutation of the numbers $1, \ldots, n$, and S_n is the set of all such permutations. The function sgn is plus or minus one, depending on if σ is an even or an odd permutation. For 2×2 matrices, this formula has the following appearance

$$\det(\mathbf{A}) = \det \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = a_{11}a_{22} - a_{12}a_{21} \ .$$

Similarly for 3×3 matrices

$$\begin{aligned} \det(\mathbf{A}) &= \det \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \\ &= a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32} - a_{11}a_{23}a_{32} - a_{12}a_{21}a_{33} - a_{13}a_{22}a_{31} \ . \end{aligned}$$

In terms of eigen values, λ_i, the determinant is equal to the product, i.e.

$$\det(\mathbf{A}) = \prod_{i=1}^{n} \lambda_i \ .$$

16. Some common rules for determinant arithmetic include

- The determinant of a matrix is equal to the determinant of it's transposed, i.e.

$$\det(A^T) = det(A) \ .$$

- The determinant of product of a matrix product is equal to the product of the two matrices determinant, i.e.

$$\det(AB) = \det(A)\det(B) \ .$$

- The determinant of an inverted matrix is equal to one divided by the determinant of the matrix, i.e.

$$\det(A^{-1}) = \frac{1}{\det(A)} \ .$$

The last rule implies, that if a matrix $\mathbf{A}$ is *invertible* it's determinant must be different from zero, i.e. it holds that

$$\det(\mathbf{A}) \neq 0 \quad \Leftrightarrow \mathbf{A^{-1}} exists \ .$$

17. The columns of a matrix, $\mathbf{A}$, can be linear dependent, e.g. if there exists α_j such that

$$a_{:1} = \sum_{j=2}^{n} \alpha_j a_{:j} \ ,$$

where $a_{:j}$ are the columns of $\mathbf{A}$, then $a_{:1}$ is linear dependent of the rest of the columns of $\mathbf{A}$. The largest set of columns of $\mathbf{A}$, such that no column is is linear dependent on the other is the *rank* of $\mathbf{A}$. This is also referred to the maximum number of linear *in*dependant columns of $\mathbf{A}$. The number of linear independent columns is equal to the number of linear independent rows, thus there is no need to talk about a column-rank and a row-rank. The rank of a matrix is also the dimension of the image of

$$f(\mathbf{v}) = \mathbf{A}\mathbf{v} \ .$$

If all a matrix rows or columns are linear independent it is said to have full rank.

18. Let $\mathbf{A}$ be a square $n \times n$ matrix with full rank, and an n-dimensional vector $\mathbf{b}$, then

$$\mathbf{A}\mathbf{x} = \mathbf{b} \ .$$

is a system or set of linear equations which uniquely determines $\mathbf{x}$, i.e.

$$\mathbf{x} = \mathbf{A^{-1}b} \ .$$

In practice A^{-1} would not be computed explicitly to solve for $\mathbf{x}$, instead a more suitable numerical routine would be used, e.g. LU-factorization.

19. A $m \times n$ matrix $\mathbf{A}$ can be seen as a mapping from $\Re^n$ to $\Re^m$. If $\mathbf{A}$ has a rank lower than n (this also includes $n > m$), $\mathbf{A}$ has a non-trivial (right) null space. The right null space is composed of the non-zero vectors, $\mathbf{x}$, for which i holds that

$$\mathbf{A}\mathbf{x} = \mathbf{0} \ .$$

The left null space can equivalently be defined as

$$\mathbf{x^T A} = \mathbf{0} \ .$$

Lastly it should be noted that the dimension of the right null space k has the following relationship with the rank of $\mathbf{A}$

$$k + rank(\mathbf{A}) = n \ .$$

A.1.1 Eigenvalues

A matrix, $\mathbf{A}$, can have many interpretations, e.g. as measured data or an operator. In the latter case it can be seen as a function of a vector $\mathbf{v}$, i.e.

$$f(\mathbf{v}) = \mathbf{A}\mathbf{v} \ .$$

When viewing a matrix as an operator, any $n \times n$ square matrix, $\mathbf{A}$, has eigenvalues, λ_i, and eigenvectors, $\mathbf{x_i}$, such that

$$\mathbf{A}\mathbf{x_i} = \lambda_i \mathbf{x_i} \quad , \quad ||\mathbf{x_i}||_2 > 0 \ .$$

That is, that for a set of non-zero vectors, $\mathbf{x_i}$, applying $\mathbf{A}$ to the vector is equivalent to scaling the vector by a factor of λ_i. For any set of eigenvector and eigenvalue, $\mathbf{x_i}$, λ_i it thus holds that

$$\mathbf{A}\mathbf{x_i} - \lambda_i \mathbf{x_i} = \mathbf{0} \Rightarrow (\mathbf{A} - \lambda_i \mathbf{I})\mathbf{x_i} = \mathbf{0} \ .$$

If $(\mathbf{A} - \lambda_i \mathbf{I})$ is invertible then

$$\mathbf{x_i} = (\mathbf{A} - \lambda_i \mathbf{I})^{-1}\mathbf{0} = \mathbf{0} \ ,$$

which is contrary to our requirement that $||\mathbf{x_i}||_2 > 0$. Thus $(\mathbf{A} - \lambda_i \mathbf{I})$ cannot be invertible, and

$$\det(\mathbf{A} - \lambda_i \mathbf{I}) = 0 \ .$$

Writing out the left side of this equation gives a polynomial in λ_i, which is called the *characteristic polynomial* for $\mathbf{A}$. Finding the roots of the characteristic polynomial, is one way of finding the eigenvalues. Once the eigenvalues have been found the corresponding eigenvectors can be found by solving the following system of linear equations

$$(\mathbf{A} - \lambda_i \mathbf{I})\mathbf{x_i} = \mathbf{0} \ .$$

The roots of the characteristic polynomial might very well be complex. However, for symmetric matrices, i.e. $\mathbf{A} = \mathbf{A^T}$, the eigenvalues are always real.

Eigenvalues and eigenvectors are a prime tools for the analysis of matrices. Among others the *rank* of a matrix is equal to the number of non-zero eigenvalues. The condition number of a matrix is equal to the ratio between the numerically largest and smallest eigenvalues. (This condition number is a prime indicator for the numerical stability of many matrix operations).

The eigne vectors, where the vector $\mathbf{v}$ is multiplied to the right of $\mathbf{A}$ are also called right eigenvectors. Correspondingly there also exist left eigenvectors. If nothing else is stated right eigenvectors are assumed. For symmetric matrices the right and left eigenvector are equal — as would be expected.

A square matrix, $\mathbf{A}$, can be decomposed or factored via its eigenvalues and vectors. Let $\mathbf{X}$ be the matrix where the j^{th} column is the j^{th} normalized eigenvector of $\mathbf{A}$. Let Λ be the diagonal[3] matrix where the diagonal element $\Lambda_{jj} = \lambda_j$. Then $\mathbf{A}$ can be decomposed into

$$\mathbf{A} = \mathbf{X}\mathbf{\Lambda}\mathbf{X^{-1}} \ .$$

For symmetric matrices $\mathbf{X}$ is a orthonormal matrix, i.e. all the eigenvectors are orthogonal (i.e. perpendicular) to each other, among others implying that $\mathbf{X^{-1}} = \mathbf{X^T}$. If the determinant is 1 (and not -1) this is a rotation matrix.

The eigen-decomposition implies, that the solution to

$$\max_{\mathbf{v}} ||\mathbf{Av}||_2 = \max_{\mathbf{v}} \mathbf{v^T A^T A v} \quad , \quad ||\mathbf{v}||_2 = 1 \ , \tag{A.1}$$

is $\mathbf{v}$ equal to the eigenvector corresponding tho the largest eigenvalue of $\mathbf{A^T A}$. The same holds for the equivalent minimization problem, where the eigenvector corresponding to the *smallest* eigenvalue is chosen.

Product Optimization – Argument*

To see this we will use Lagrange multipliers, whereby (A.1) becomes, where γ is used for the Lagrange multiplier,

$$\begin{aligned} 0 &= \frac{\partial}{\partial \mathbf{V}} \mathbf{v^T A^T A v} - \gamma(\mathbf{v^T v} - 1) \\ &= \mathbf{A^T A v} - \gamma \mathbf{v} \\ &= \left(\mathbf{A^T A} - \gamma \mathbf{I}\right) \mathbf{v} \ , \end{aligned}$$

implying that a necessary condition is $\mathbf{v}$ that is an eigenvector of $\mathbf{A^T A}$. Noting that $\mathbf{A^T A}$ is a symmetric matrix $\mathbf{X}$ must be orthonormal. Therefor, setting $\mathbf{v}$ equal to the j^{th} eigenvector of $\mathbf{A^T A}$, will make $\mathbf{Xv}$ equal to a vector that is all zeros, except for the j^{th} element, which will be 1. Letting

$$\begin{aligned} \mathbf{A^T A} &= \mathbf{X}\mathbf{\Lambda}\mathbf{X^T} \ , \\ \mathbf{v^T A^T A v} &= \mathbf{v^T X}\mathbf{\Lambda}\mathbf{X^T v} = ||\mathbf{\Lambda X^T v}|| = \sigma_j \ . \end{aligned}$$

since $\mathbf{X}$ is just a rotation, which does not the norm. Thus the largest (respectively smallest) value of (A.1) is obtained by choosing the largest (respectively smallest) σ_j. This corresponds to choosing the eigenvector corresponding to the largest(respectively smallest) eigenvalue of $\mathbf{A^T A}$.

[3] All but the diagonal elements are zero.

A.2 Linear Least Squares

A problem that we often want to solve is the so called least squares problem, i.e

$$\min_{\mathbf{x}} ||\mathbf{Ax} - \mathbf{b}||_2 \ . \tag{A.2}$$

To solve (A.2), denote the residual error vector ϵ by

$$\epsilon = \mathbf{Ax} - \mathbf{b} \ ,$$

And (A.2) is equivalent to

$$\min_{\mathbf{x}} ||\epsilon||_2 \ .$$

It is then seen that

$$\begin{aligned} ||\epsilon||_2 &= \epsilon^T \epsilon \\ &= (\mathbf{Ax} - \mathbf{b})^T (\mathbf{Ax} - \mathbf{b}) \\ &= \mathbf{x^T A^T Ax} - \mathbf{x^T A^T b} - \mathbf{b^T Ax} + \mathbf{b^T b} \\ &= \mathbf{x^T A^T Ax} - \mathbf{2b^T Ax} + \mathbf{b^T b} \ , \end{aligned}$$

Where it is seen that $\mathbf{x^T A^T b} = (\mathbf{x^T A^T b})^\mathbf{T} = \mathbf{b^T Ax}$, since this is a scalar, and the transposed of a scalar is the same scalar. To solve (A.2) we will set the derivative of $||\epsilon||_2$ with respect to $\mathbf{x}$ equal to zero, i.e.

$$\begin{aligned} 0 &= \frac{\partial}{\partial \mathbf{x}} \ ||\epsilon||_2 \\ &= \frac{\partial}{\partial \mathbf{x}} \ \mathbf{x^T A^T Ax} - \mathbf{2b^T Ax} + \mathbf{b^T b} \\ &= \mathbf{2A^T Ax} - \mathbf{2A^T b} \Rightarrow \\ \mathbf{A^T Ax} &= \mathbf{A^T b} \Rightarrow \\ \mathbf{x} &= (\mathbf{A^T A})^{-1} \mathbf{A^T b} \ . \end{aligned} \tag{A.3}$$

This assumes that $\mathbf{A^T A}$ has full rank and is invertible. If this is not the case the problem is not well constrained, and there exist several solution for $\mathbf{x}$. It is noted that (A.3) are often called the *normal equations*. A special version of (A.2) is when $\mathbf{b} = \mathbf{0}$, in which case the problem reduces to

$$\min_{\mathbf{x}} ||\mathbf{Ax}||_2 = \min_{\mathbf{x}} \mathbf{x^T A^T Ax} \ .$$

As seen in Appendix A.1.1, the solution is the eigenvector corresponding to the smallest eigenvalue of $\mathbf{A^T A}$. Usually the solution is found via singular value decomposition (SVD) of $\mathbf{A}$, since this is the most numerically stabile.

A.3 Rotation

A rotation, $\mathbf{R}$, is a square $n \times n$ matrix where,

- All rows, $\mathbf{r_i^T}$, are orthogonal to each other, and have norm 1,i.e.

$$\forall i \neq j \quad \mathbf{r}_i^T \mathbf{r}_j = 0 \quad and \quad \forall i \quad \mathbf{r}_i^T \mathbf{r}_i = ||r_i||_2^2 = 1 \ .$$

- All columns, $\mathbf{c_j}$, are orthogonal to each other, and have norm 1,i.e.

$$\forall i \neq j \quad \mathbf{c}_i^T \mathbf{c}_j = 0 \quad and \quad \forall i \quad \mathbf{c}_i^T \mathbf{c}_i = ||c_i||_2^2 = 1 \ .$$

- Has determinant $+1$.

The first two items are equivalent to

$$\mathbf{R^T R} = \begin{bmatrix} \mathbf{c}_1^T \\ \mathbf{c}_2^T \\ \vdots \\ \mathbf{c}_n^T \end{bmatrix} \begin{bmatrix} \mathbf{c}_1 & \mathbf{c}_2 & \dots & \mathbf{c}_n \end{bmatrix} = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix} = \mathbf{I} \ ,$$

$$\mathbf{R R^T} = \begin{bmatrix} \mathbf{r}_1^T \\ \mathbf{r}_2^T \\ \vdots \\ \mathbf{r}_n^T \end{bmatrix} \begin{bmatrix} \mathbf{r}_1 & \mathbf{r}_2 & \dots & \mathbf{r}_n \end{bmatrix} = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix} = \mathbf{I} \ ,$$

Thus the inverse of a rotation matrix is given by it's transposed, i.e.

$$\mathbf{R^{-1}} = \mathbf{R^T} \ . \tag{A.4}$$

A rotation matrix is thus an orthonormal basis, and it can be used as a basis shift from one orthonormal basis to another (to change between right handed Cartesian coordinate systems a translation is generally also needed since a rotation does not change the location of origo).

Matrices fulfilling the first two requirements above, can have a determinant of either -1 or $+1$, an are called *orthonormal matrices*. Thus a rotation is a special orthonormal matrix. An interpretation of a orthonomal matrix with determinant -1 is that it is a reflection, e.g.

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \ ,$$

which has a determinant of -1 and is a reflection about the x-axis, when applied to a vector, i.e.

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \mathbf{v} \ .$$

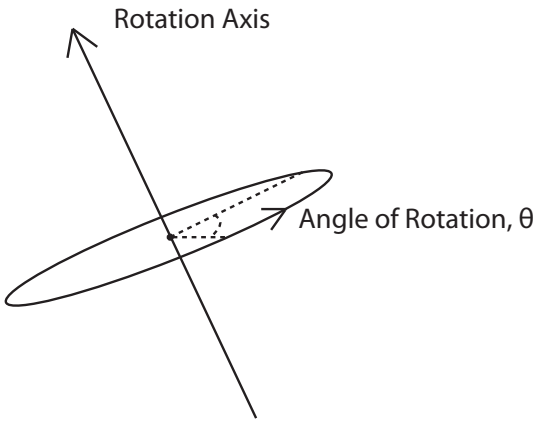

Figure A.1: A rotation matrix rotates all points with an angle of θ around an axis.

In 2D a rotation matrix can be parameterized as

$$\mathbf{R}(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \ ,$$

where

$$\mathbf{R}(\theta)\mathbf{v} \ ,$$

is a rotation of $\mathbf{v}$ by an angle of θ. Apart from the 2D rotation the 3D rotation is of special interest, since it represents rotations in the space we inhabit. It is used to represent the rotation of a vector, $\mathbf{v}$, around the origo, to $\tilde{\mathbf{v}}$ i.e.

$$\tilde{\mathbf{v}} = \mathbf{R}\mathbf{v} \ .$$

In general[4], a rotation matrix rotates all points with an angle of θ around an axis, see Figure A.1. This axis, $\mathbf{a}$, will have the property that

$$\mathbf{a} = \mathbf{R}\mathbf{a} \ ,$$

and is as seen to be an eigenvector of the rotation matrix – which also yeilds an algorithm for finding it. In general a rotation matrix will only have one real (and two complex conjugate) eigenvalues, and the axis, $\mathbf{a}$, is the one corresponding to the real eigenvalue.

Several rotations can be combined, i.e.

$$\mathbf{R} = \mathbf{R_1}\mathbf{R_2}\mathbf{R_3}\ldots$$

and the end result will still be a rotation, with a single axis of rotation.

A.3.1 Parametrization of Rotations in 3D

A rotation matrix has three degrees of freedom – two for the axis of rotation (a 3D vector with length 1), and one for the angle of rotation. Often times it is necessary to parameterize a rotation matrix in terms of such 3 parameters (sometimes four are used as explained bellow). One way of doing this is to compose the rotation matrix of rotations around the three coordinate axis, i.e.

$$\mathbf{R}(\omega,\phi,\kappa) = \mathbf{R_z}(\kappa)\mathbf{R_y}(\phi)\mathbf{R_x}(\omega) \ . \tag{A.5}$$

These rotations can easily be composed by generalizing the rotation in 2D, i.e.

$$\mathbf{R_x}(\omega) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\omega) & -\sin(\omega) \\ 0 & \sin(\omega) & \cos(\omega) \end{bmatrix} ,$$

$$\mathbf{R_y}(\phi) = \begin{bmatrix} \cos(\phi) & 0 & -\sin(\phi) \\ 0 & 1 & 0 \\ \sin(\phi) & 0 & \cos(\phi) \end{bmatrix} ,$$

$$\mathbf{R_z}(\kappa) = \begin{bmatrix} \cos(\kappa) & -\sin(\kappa) & 0 \\ \sin(\kappa) & \cos(\kappa) & 0 \\ 0 & 0 & 1 \end{bmatrix} .$$

Multiplying out (A.5) gives

$$\mathbf{R}(\omega,\phi,\kappa) = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} ,$$

where

$$\begin{aligned}
r_{11} &= \cos(\phi)\cos(\kappa) \\
r_{12} &= -\sin(\omega)\sin(\phi)\cos(\kappa) - \cos(\omega)\sin(\kappa) \\
r_{13} &= -\cos(\omega)\sin(\phi)\cos(\kappa) + \sin(\omega)\sin(\kappa) \\
r_{21} &= \cos(\phi)\sin(\kappa) \\
r_{22} &= -\sin(\omega)\sin(\phi)\sin(\kappa) + \cos(\omega)\cos(\kappa) \\
r_{23} &= -\cos(\omega)\sin(\phi)\sin(\kappa) - \sin(\omega)\cos(\kappa) \\
r_{31} &= \sin(\phi) \\
r_{32} &= \sin(\omega)\cos(\phi) \\
r_{33} &= \cos(\omega)\cos(\phi)
\end{aligned}$$

[4] When the angle of rotation is non-zero.

This parameterizations scheme is called *Euler angles*. There are, however, many orders in which there can be rotated around the axis, so there is a multitude of Euler angle parameterizations.

A problem with using Euler angels is the so-called Gimbal lock, which implies that for some ω, ϕ, κ values the parametrization looses a degree of freedom. This is another way of saying that this parametrization has a singularity. It can be proven, that such singularities, c.f. e.g. [3], will always exist when using only three parameters to parameterize a rotation. This, among others, has the effect that the derivative of the rotation matrix, e.g.

$$\frac{\partial \mathbf{R}(\omega, \phi, \kappa)}{\omega} \ ,$$

Becomes zero for certain ω, ϕ, κ configurations. This is e.g. a serious problem for many optimization problems involving rotations, e.g.

$$\min_{\omega,\phi,\kappa} \quad f(\mathbf{R}(\omega, \phi, \kappa)) \ ,$$

for some function f. To address this problem other parameterizations exist. Notably, quaternions, c.f. e.g. [1, 10], which use four parameters to parameterize, and the Rodrigues formula which use a local parametrization.

A.4 Change of Right Handed Cartesian Coordinate Frame

A cartesian coordinate system or frame is an orthonormal one, i.e. all the coordinate axis are orthogonal and have unit length. Right handed implies that the the z-axis is the cross product of the x-axis and y-axis. The center or origo of such a coordinate system must be given, but can be arbitrary — with respect to some global system. Assume that a point, $\mathbf{x}$ is given in the global[5] right handed cartesian coordinate system, then we can transform it into any other right handed Cartesian coordinate system, with coordinates $\mathbf{x}^\dagger$, via

$$\mathbf{x}^\dagger = \mathbf{R}^\dagger \mathbf{x} + \mathbf{t}^\dagger \ ,$$

Where $\mathbf{R}^\dagger$ is a rotation and $\mathbf{t}^\dagger$ is a translation. The columns of $\mathbf{R}^\dagger$ are the coordinate vectors of the new coordinate system. Note, that the coordinate axis of any right handed Cartesian coordinate system can be expressed as a rotation matrix in this way and vice versa. If the requirement that $\det \mathbf{R} = +1$ was not enforced we could also map to a left handed coordinate system. The translation vector, $\mathbf{t}^\dagger$, is equal to the origin of the new coordinate system in the global coordinate system. The reverse transformation is given by

$$\begin{aligned} \mathbf{x} &= \mathbf{R}^{\dagger\mathbf{T}} \mathbf{R}^\dagger \mathbf{x} \\ &= \mathbf{R}^{\dagger\mathbf{T}} (\mathbf{R}^\dagger \mathbf{x} + \mathbf{t}^\dagger - \mathbf{t}^\dagger) \\ &= \mathbf{R}^{\dagger\mathbf{T}} (\mathbf{x}^\dagger - \mathbf{t}^\dagger) \\ &= \mathbf{R}^{\dagger\mathbf{T}} \mathbf{x}^\dagger - \mathbf{R}^{\dagger\mathbf{T}} \mathbf{t}^\dagger \ . \end{aligned}$$

Here $\mathbf{R}^{\dagger\mathbf{T}}$ and $\mathbf{R}^{\dagger\mathbf{T}} \mathbf{t}^\dagger$ are a rotation and a translation respectively. Given another right handed Cartesian coordinate system where $\mathbf{x}$ has coordinates $\mathbf{x}^\ddagger$, the transforation to this coordinate system is given by

$$\begin{aligned} \mathbf{x}^\ddagger &= \mathbf{R}^\ddagger \mathbf{x} + \mathbf{t}^\ddagger \\ &= \mathbf{R}^\ddagger (\mathbf{R}^{\dagger\mathbf{T}} \mathbf{x}^\dagger - \mathbf{R}^{\dagger\mathbf{T}} \mathbf{t}^\dagger) + \mathbf{t}^\ddagger \\ &= \mathbf{R}^\ddagger \mathbf{R}^{\dagger\mathbf{T}} \mathbf{x}^\dagger - \mathbf{R}^\ddagger \mathbf{R}^{\dagger\mathbf{T}} \mathbf{t}^\dagger + \mathbf{t}^\ddagger \ . \end{aligned}$$

Where again, $\mathbf{R}^\ddagger \mathbf{R}^{\dagger\mathbf{T}}$ and $-\mathbf{R}^\ddagger \mathbf{R}^{\dagger\mathbf{T}} \mathbf{t}^\dagger + \mathbf{t}^\ddagger$ are a rotation and a translation respectively. Thus demonstrating that we can get from any right handed Cartesian coordinate system to another via a rotation and a translation. Furthermore, it is demonstrated how this rotation and rotation should look, given the two coordinate systems relation to the global coordinate system — or any other coordinate system for that matter.

A.5 Cross Product as an Operator

The cross product between two vectors $\mathbf{a}$ and $\mathbf{b}$, denoted $\mathbf{a} \times \mathbf{b}$, is given by

$$\mathbf{a} \times \mathbf{b} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \times \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} a_2 b_3 - a_3 b_2 \\ a_3 b_1 - a_1 b_3 \\ a_1 b_2 - a_2 b_1 \end{bmatrix} \ . \tag{A.6}$$

[5] Largely a matter of definition.

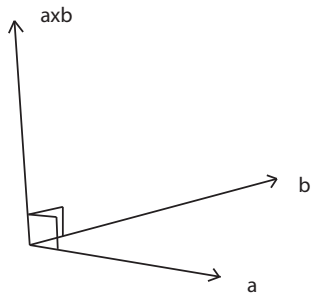

Figure A.2: The cross product $\mathbf{a} \times \mathbf{b}$ between vector $\mathbf{a}$ and $\mathbf{b}$. Note the right hand rule, which determines the direction of $\mathbf{a} \times \mathbf{b}$.

The cross product produces a vector, which is orthogonal to both $\mathbf{a}$ and $\mathbf{b}$, and has the length

$$\sin\theta ||\mathbf{a}||_2 ||\mathbf{b}||_2 \ ,$$

where θ is the angle between $\mathbf{a}$ and $\mathbf{b}$, see Figure A.2. It is therefor also refereed to as the sine product. If $s\mathbf{a} = \mathbf{b}$, where s is a scalar then the cross product is zero. This consistent with the sine of zero being zero. More formally

$$\mathbf{a} \times \mathbf{b} = \mathbf{a} \times s\mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \times \left(s \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \right) = s \begin{bmatrix} a_2 a_3 - a_3 a_2 \\ a_3 a_1 - a_1 a_3 \\ a_1 a_2 - a_2 a_1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} = \mathbf{0} \ .$$

The cross product of a vector, $\mathbf{a}$, with any other vector, $\mathbf{b}$, can be formulated as a matrix vector multiplication. In other words, the cross product of a vector, $\mathbf{a}$, with any other vector is a linear operator. To see this, we can write (A.6) as

$$\begin{bmatrix} a_2 b_3 - a_3 b_2 \\ a_3 b_1 - a_1 b_3 \\ a_1 b_2 - a_2 b_1 \end{bmatrix} = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = [\mathbf{a}]_\times \cdot \mathbf{b} \ , \tag{A.7}$$

where $[\mathbf{a}]_\times$ is the skew symmetric matrix in question.

A.6 SVD – Generalized Eigen Values*

The singular value decomposition(SVD) [11] is a generalization of the eigne value decomposition of square symmetric matrices to all matrices. That is that all matrices $\mathbf{A}$ can be written as (assume the dimensions of $\mathbf{A}$ is $m \times n$)

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\mathbf{T} \ , \tag{A.8}$$

where $\mathbf{U}$ is a $m \times m$ orthonormal matrix and $\mathbf{V}$ is a $n \times n$ orthonormal matrix . A orthonormal or unitary matrix is the same as a rotation matrix *without* the constraint that the determinant should be $+1$. That is that the determinant of a orthonormal matrix can be -1 or $+1$. It is noted that orthonormal matrices form are composed of a orthonormal basis, and thus $\mathbf{U}^\mathbf{T}\mathbf{U} = \mathbf{U}\mathbf{U}^\mathbf{T} = \mathbf{I}$ and $\mathbf{V}^\mathbf{T}\mathbf{V} = \mathbf{V}\mathbf{V}^\mathbf{T} = \mathbf{I}$. The matrix $\mathbf{\Sigma}$ is a $m \times n$ diagonal matrix, where the diagonal elements are the so-called singular values σ_i. The σ_i are positive and ordered such that

$$\sigma_1 \geq \sigma_2 \geq \ldots \sigma_k \geq 0 \quad , \quad k = \min(m, n) \ .$$

That is

$$\begin{aligned} \mathbf{\Sigma} &= \begin{bmatrix} \tilde{\mathbf{\Sigma}} & \mathbf{0} \end{bmatrix} \quad , \quad m \leq n \\ \mathbf{\Sigma} &= \begin{bmatrix} \tilde{\mathbf{\Sigma}} \\ \mathbf{0} \end{bmatrix} \quad , \quad m > n \\ \tilde{\mathbf{\Sigma}} &= diag(\sigma_1, \sigma_2, \ldots, \sigma_k) \ . \end{aligned}$$

This is seen to be a generalization of the eigendecomposition of of symmetric square matrices, where the resulting decomposition is also a orthonormal matrix of eigenvectors followed by a diagonal matrix of eigenvalues and lastly followed by the same orthonormal matrix of eigenvectors transposed. This also leads us to a way of deriving the singular value decomposition, noting that A^TA and AA^T are symmetric positive semi-definite[6] matrices,

$$\mathbf{A^TA} = (\mathbf{U\Sigma V^T})^T(\mathbf{U\Sigma V^T}) = \mathbf{V\Sigma^T U^T U\Sigma V^T} = \mathbf{V\Sigma^T\Sigma V^T} \ ,$$

Thus $\mathbf{V^T}$ are the eigenvectors of $\mathbf{A^TA}$. Similarly

$$\mathbf{AA^T} = (\mathbf{U\Sigma V^T})(\mathbf{U\Sigma V^T})^T = \mathbf{U\Sigma V^T V\Sigma^T U^T} = \mathbf{U\Sigma\Sigma^T U^T} \ ,$$

The eigenvectors of $\mathbf{AA^T}$ are equal to $\mathbf{U}$. Lastly the singular values are square root of the largest k eigenvalues of $\mathbf{AA^T}$ or $\mathbf{A^TA}$, subsequently ordered, i.e. (assuming the eigenvalues are ordered correctly)

$$\sigma_i = \sqrt{\lambda_i(\mathbf{AA^T})} = \sqrt{\lambda_i(\mathbf{A^TA})} \ .$$

The use of the SVD is widespread, in part because it is an efficient algorithm, and in part because it yields nice interpretations of matrices and good accompanying algorithms. When interpreting a matrix via SVD we first have to distinguish if the matrix models a function from $\Re^n \leftarrow \Re^m$, i.e. $\mathbf{w} = \mathbf{Av}$, or a collection of data points, i.e. each row of $\mathbf{A}$ is a measurement.

If we view a matrix, $\mathbf{A} = \mathbf{U\Sigma V^T}$, as a function, $\mathbf{w} = \mathbf{Av} = \mathbf{U\Sigma V^T v}$. Then, firstly, $\mathbf{v'} = \mathbf{V^T v}$ is a basis shift of $\mathbf{v}$, one can think of it as a rotation. Then $\mathbf{w'} = \mathbf{\Sigma v'}$ is a scaling of the coordinates of $\mathbf{v'}$, i.e. $\mathbf{w'_i} = \sigma_\mathbf{i}\mathbf{v'_i}$ for $i \in [1, \ldots, k]$. Lastly, $\mathbf{w} = \mathbf{Uw'}$ is again a basis shift of $\mathbf{w'}$. Any matrix function can, thus, in the *right* basis, be seen solely as a diagonal scaling, and $||\mathbf{w}||_2 \leq \sigma_1||\mathbf{v}||_2$. Further more, expressing $\mathbf{v}$ in terms of the basis $\mathbf{V}$, the part in the direction of the i^{th} row of $\mathbf{V}$ is scaled by a factor of σ_i. So if we want to maximize or minimize $\mathbf{Av}$ wrt. $\mathbf{v}$, we just have to set $\mathbf{v}$ equal to the first or last rows of $\mathbf{V}$ respectively. Lastly, the rows of $\mathbf{U}$ corresponding to zero eigenvalues or have an index larger then k are the null-space of the function, i.e. the basis of the parts of $\Re^m$ that $\mathbf{Av}$ cannot 'reach'.

If we view a matrix as a set of measurements, i.e.

$$\mathbf{A} = \begin{bmatrix} \mathbf{a_1} & \mathbf{a_2} & \ldots & \mathbf{a_n} \end{bmatrix} \ ,$$

where the rows $\mathbf{a_j}$ are measurements. Then apart from revealing the variance structure of the measurements, in that

$$\sum_{j=1}^{n} \mathbf{a_j a_j^T} = \mathbf{AA^T} = \mathbf{U\Sigma V^T V\Sigma^T U^T} = \mathbf{U\Sigma\Sigma^T U^T} \ ,$$

it also gives the 'best' low rank approximation. Charged with finding the q dimensional approximation that expresses most of $\mathbf{A}$ ($q < k$), the solution is given by

$$\mathbf{B} = \sum_{i=1}^{q} \mathbf{u_i}\sigma_\mathbf{i}\mathbf{v_i^T}$$

where $\mathbf{u_i}$ and $\mathbf{v_i}$ are the rows of $\mathbf{U}$ and $\mathbf{V}$ respectively. That is, that $\mathbf{B} = \mathbf{C}$ is the rank q matrix that minimizes

$$\min_m bC||\mathbf{A} - \mathbf{C}||_{Fro} \ ,$$

where $||\cdot||_{Fro}$ is the Frobenius norm. This can e.g. be used to fit a plane to a data set. Lastly, it should be mentioned that it is also possible to analytically compute the derivative of the SVD wrt. $\mathbf{A}$, c.f. [22].

The *Frobenius norm* of matrices is given by

$$\sum_{i=1}^{m}\sum_{j=1}^{n} a_{ij}^2 = Trace(\mathbf{A^TA}) = Trace(\mathbf{AA^T}) \ .$$

As seen above this norm is closely tied with the SVD and minimizes the sum of squared errors. It is noted that many other matrix norms exist

[6] All eigenvalues are positive or zero

A.7 Kronecker Product*

The Kronecker product[9, 28], between two matrices $\mathbf{A}$ and $\mathbf{B}$ is defined as the $mp \times nq$ matrix

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & \dots & a_{1n}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{m1}\mathbf{B} & \dots & a_{mn}\mathbf{B} \end{bmatrix} , \tag{A.9}$$

where $\mathbf{A}$ is an $m \times n$ matrix and $\mathbf{B}$ is a $p \times q$ matrix. In other words, the Kronecker product produces a matrix, consisting of blocks of the matrix $\mathbf{B}$, one block for each element of $\mathbf{A}$, and each block multiplied with the respective element of $\mathbf{A}$. As an example consider

$$\begin{bmatrix} 0.01 & 0.1 \\ 1 & 10 \end{bmatrix} \otimes \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} = \begin{bmatrix} 0.01 & 0.02 & 0.03 & 0.1 & 0.2 & 0.3 \\ 0.04 & 0.05 & 0.06 & 0.4 & 0.5 & 0.6 \\ 1 & 2 & 3 & 10 & 20 & 30 \\ 4 & 5 & 6 & 40 & 50 & 60 \end{bmatrix} .$$

Arithmetic with the Kronecker product often uses the *vectorization function*, vec. This function produces a vector from a matrix by stacking all the columns. Denote the columns of $\mathbf{A}$ by $\mathbf{c}_j$, then

$$vec(\mathbf{A}) = vec\left(\begin{bmatrix} \mathbf{c}_1 & \mathbf{c}_2 & \dots & \mathbf{c}_n \end{bmatrix}\right) = \begin{bmatrix} \mathbf{c}_1 \\ \mathbf{c}_2 \\ \vdots \\ \mathbf{c}_n \end{bmatrix} . \tag{A.10}$$

As an example consider

$$vec\left(\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}\right) = \begin{bmatrix} 1 \\ 4 \\ 2 \\ 5 \\ 3 \\ 6 \end{bmatrix} .$$

It is seen that $vec(\mathbf{v}) = \mathbf{v}$ for a vector $\mathbf{v}$, and thus $vec(\mathbf{Av}) = \mathbf{Av}$ since $\mathbf{Av}$ is a vector.

Common rules for Kronecker arithmetic include, where $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ and $\mathbf{D}$ are matrices of appropriate dimensions and s is a scalar.

1. $\mathbf{A} \otimes (\mathbf{B} + \mathbf{C}) = \mathbf{A} \otimes \mathbf{B} + \mathbf{A} \otimes \mathbf{C}$
2. $(\mathbf{A} + \mathbf{B}) \otimes \mathbf{C} = \mathbf{A} \otimes \mathbf{C} + \mathbf{B} \otimes \mathbf{C}$
3. $s(\mathbf{A} \otimes \mathbf{B}) = (s\mathbf{A}) \otimes \mathbf{B} = \mathbf{A} \otimes (s\mathbf{B})$
4. $(\mathbf{A} \otimes \mathbf{B}) \otimes \mathbf{C} = \mathbf{A} \otimes (\mathbf{B} \otimes \mathbf{C})$
5. $(\mathbf{A} \otimes \mathbf{B})(\mathbf{C} \otimes \mathbf{D}) = \mathbf{AC} \otimes \mathbf{BD}$
6. Generally the Kronecker product is *not* commutative, i.e. generally $\mathbf{A} \otimes \mathbf{B} \neq \mathbf{B} \otimes \mathbf{A}$
7. The transpose does *not* reverse the order, i.e. $(\mathbf{A} \otimes \mathbf{B})^T = \mathbf{A}^T \otimes \mathbf{B}^T$
8. If the inverse of $\mathbf{A}$ and $\mathbf{B}$ exist, then $(\mathbf{A} \otimes \mathbf{B})^{-1} = \mathbf{A}^{-1} \otimes \mathbf{B}^{-1}$
9. Given the square $m \times m$ matrix $\mathbf{E}$ and square $n \times n$ matrix $\mathbf{F}$, then $\det(\mathbf{E} \otimes \mathbf{F}) = \det(\mathbf{E})^n \det(\mathbf{F})^m$
10. Given $\mathbf{ABC} = \mathbf{D}$ then
$$vec(\mathbf{ABC}) = (\mathbf{C}^T \otimes \mathbf{A})vec(\mathbf{B}) = vec(\mathbf{D}) .$$

Here especially rule 10 is used in this text, in the following context; if $\mathbf{A}, \mathbf{B}$ and $\mathbf{C}$ are known as well as the relation

$$\mathbf{AXB} = \mathbf{C} \ ,$$

which we want to write as a linear equation in the elements of the unknown X, then rule 10 gives us

$$(\mathbf{B}^T \otimes \mathbf{A})vec(\mathbf{X}) = vec(\mathbf{C}) \ .$$

Here $\mathbf{C}$ might be a scalar, i.e. a 1×1 matrix.

A.8 Domain*

In all of this appendix it has been assumed that we were dealing with real $\Re$ numbers. The presented material applies almost unchanged for a variety of other domains. In particular every thing holds for complex numbers, $\mathcal{C}$, with the exception that transposed, $\mathbf{A}^T$, should be exchanged with the conjugate transpose (Hermitian adjoint), $\mathbf{A}^H$. The conjugate transpose is an operation where the matrix is first transposed, and then all the elements of the matrix are conjugated, i.e. the sign of the complex part changed. As an example consider

$$\begin{bmatrix} 1+i & 1+2i \\ -1+i & -1-i \\ i & 2 \end{bmatrix}^H = \begin{bmatrix} 1-i & -1-i & -i \\ 1-2i & -1+i & 2 \end{bmatrix} \ .$$

Bibliography

[1] T. Akenine-Möller and E. Haines. *Real-Time Rendering (2nd Edition)*. AK Peters, Ltd., 2002.

[2] S. Arya, D.M. Mount, N.S. Netanyahu, R. Silverman, and A.Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM (JACM)*, 45(6):891–923, 1998.

[3] S.I. Bishop, R.L. andGoldberg. *Tensor Analysis on Manifolds*. Dover Publications, 1980.

[4] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):679–698, 1986.

[5] J.M. Carstensen(Editor). *Image Analysis, Vision and Computer Graphics*. Tehnical University of Denmark, 2002.

[6] J. Eising. *Lineær Algebra*. Technical University of Denmark, Department of Mathematics, 1997.

[7] P. Favaro and S. Soatto. A geometric approach to shape from defocus. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(3):406–417, 2005.

[8] G.H. Golub and C.F. van Loan. *Matrix Computations*. Johns Hopkins University Press, 3rd edition, 1996.

[9] A. Graham. *Kronecker Products and Matrix Calculations with Applications*. John Wiley, New York, 1981.

[10] W. Hamilton. *Lectures on Quaternions*. Hodges and Smith & Co., Dublin, 1853.

[11] P.C. Hansen. *Rank-Deficient and Discrete Ill-Posed Problems. Numerical Aspects of Linear Inversion*. SIAM, Philadelphia, 1998.

[12] R.M. Haralick, Chung-Nan Lee, K. Ottenberg, and M. Nolle. Review and analysis of solutions of the three point perspective pose estimation problem. *International Journal of Computer Vision*, 13(3):331–356, 1994.

[13] C. Harris and M. Stephens. A combined corner and edge detector. In *Proc. Alvey Conf.*, pages 189–192, 1988.

[14] R. I. Hartley and A. Zisserman. *Multiple View Geometry – 2nd edition*. Cambridge University Press, The Edinburgh Building, Cambridge CB2 2RU, UK, 2003.

[15] J. Heikkila. Geometric camera calibration using circular control points. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(10):1066–1077, 2000.

[16] R. Jonker and A. Volgenant. A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing*, 38:325–340, 1987.

[17] F. Kahl, S. Agarwal, M.K. Chandraker, D. Kriegman, and S. Belongie. Practical global optimization for multiview geometry. *International Journal of Computer Vision*, 79(3):271–284, 2008.

[18] M. Levoy, S. Rusinkiewcz, M. Ginzton, J. Ginsberg, K. Pulli, D. Koller, S. Anderson, J. Shade, B. Curless, L. Pereira, J. Davis, and D. Fulk. The digital michelangelo project: 3d scanning of large statues. *Computer Graphics Proceedings. Annual Conference Series 2000. SIGGRAPH 2000. Conference Proceedings*, pages 131–44, 2000.

[19] T. Lindeberg. On the axiomatic foundations of linear scale-space: Combining semi-group structure with causality vs. scale invariance, 1997.

[20] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[21] D. Nister. An efficient solution to the five-point relative pose problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(6):756–770, 2004.

[22] T. Papadopoulo and M.I.A. Lourakis. Estimating the jacobian of the singular value decomposition: Theory and applications. In *Research report, INRIA Sophia-Antipolis, 2000. In preparation*, pages 554–570. Springer, 2000.

[23] Deriche R. Fast algorithms for low-level vision. *IEEE Trans. Pattern Anal. Mach. Intell.*, 12(1):78–87, 1990.

[24] S. Ray. *Applied Photographic Optics, Third Edition*. Focal Press, 2002.

[25] J. Sporring. *Gaussian Scale-Space Theory*. Kluwer Academic Publishers, 1997.

[26] H. Stewenius, D. Nister, F. Kahl, and F. Schaffilitzky. A minimal solution for relative pose with unknown focal length. *Image and Vision Computing*, 26(7):871–877, 2008.

[27] T. Tuytelaars and K. Mikolajczyk. Local invariant feature detectors: a survey. *Found. Trends. Comput. Graph. Vis.*, 3(3):177–280, 2008.

[28] C.F. van Loan. The ubiquitous kronecker product. *Journal of Computational and Applied Mathematics*, 123(1-2):85–100, 2000.

Index