Lecture Notes on Computer Vision

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

February 2, 2018

Contents

Preface

In it's present form this text is intended for the DTU courses 02504 and 02506. I would like to take this opportunity to thank my colleagues for invaluable help in completing these notes, by clarifying things for me and via proof reading. I would also like to express my appreciation to the students who have pointed out errors and come with suggestions for improvement.

/Henrik

Part I

View Geometry

Chapter 1

Single Camera Geometry - Modeling a Camera

Much of computer vision and image analysis deals with inference about the world from images thereof. In many cases this requires an understanding of the imaging process. Since computers are used, 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 thereof is described, and mathematically modeled. 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 [Hartley and Zisserman, 2003].

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

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

When dealing with the pinhole camera model, as will be done in the following, 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 [Hartley and Zisserman, 2003]. 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

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

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

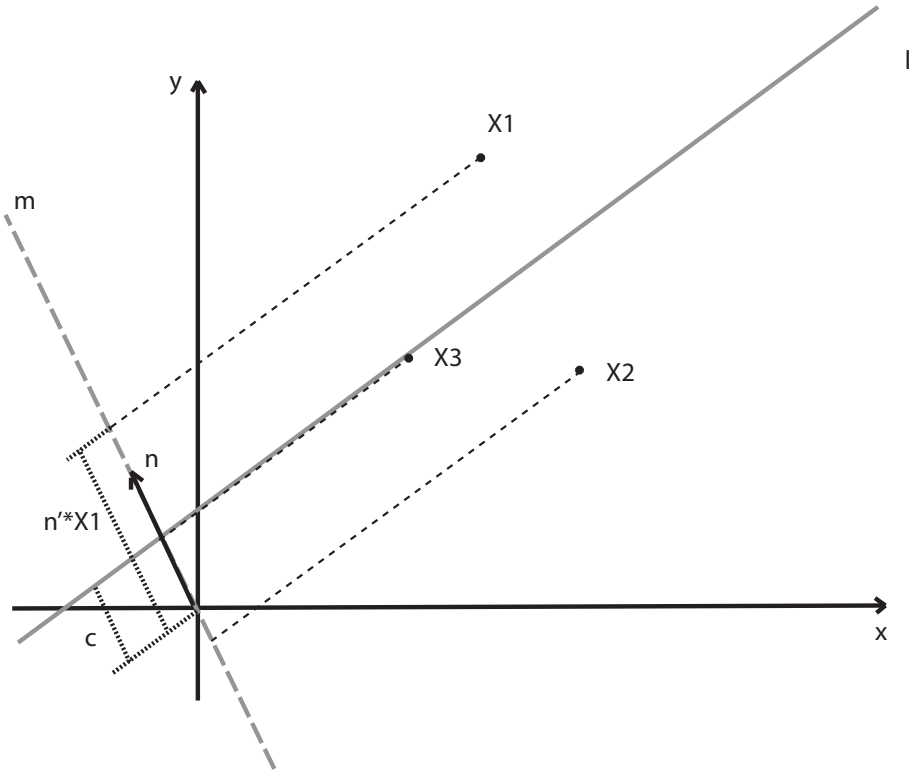


Figure 1.1: The distance from a point X_i to the line $\mathbf{l} = [\mathbf{n}^T, -c]^T$, is preserved by projecting the point onto a line normal to $\mathbf{l}$ (with direction $\mathbf{n}$). The distance is thus the inner product of X_i and $\mathbf{n}$ minus the projection of l onto its perpendicular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

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

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

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

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

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

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 Modeling 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 viewed 3D model 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 [Hartley and Zisserman, 2003, Ray, 2002].

1.2.1 What is a Good Model

As an example of *modeling* consider dropping an object from a given height and predicting it's position, which pretty much boils down to modeling 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

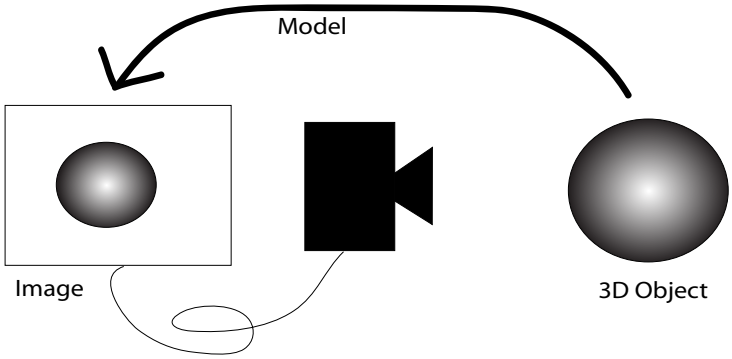


Figure 1.2: The required camera model should relate the 3D object viewed and the image generated by the camera.

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.

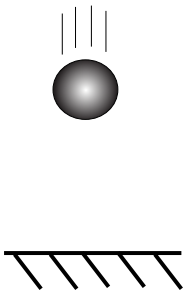


Figure 1.3: How fast will a dropped object accelerate?

So what is a good model? This answer depends on the purpose of the modeling. 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.

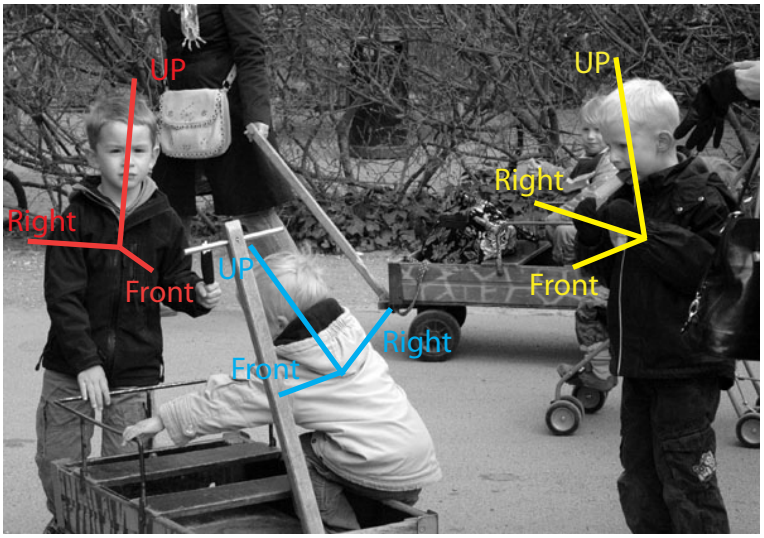


Figure 1.4: It is not only in camera geometry, where a multitude of reference frames exist. What is to the right of the boy to the left is in front of the boy to the right.

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.

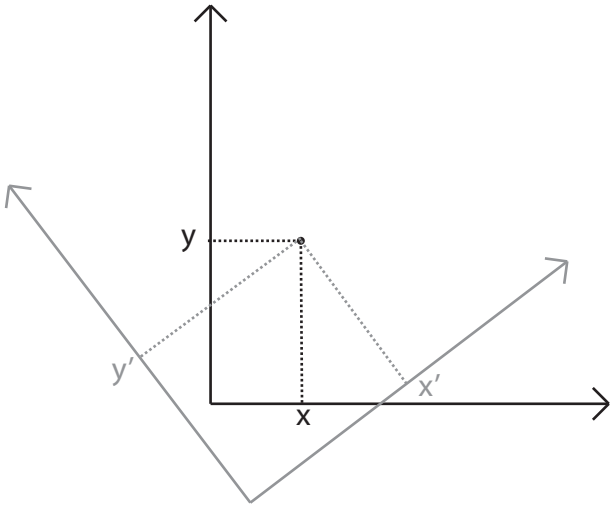


Figure 1.5: An example of a change of coordinate systems. The aim is to find the coordinates of the points in the gray coordinate system, i.e. (x', y'), given the coordinates of the point is the black coordinate system, i.e. (x, y). Note, that the location of the point does not change (in some sort of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

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

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)

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

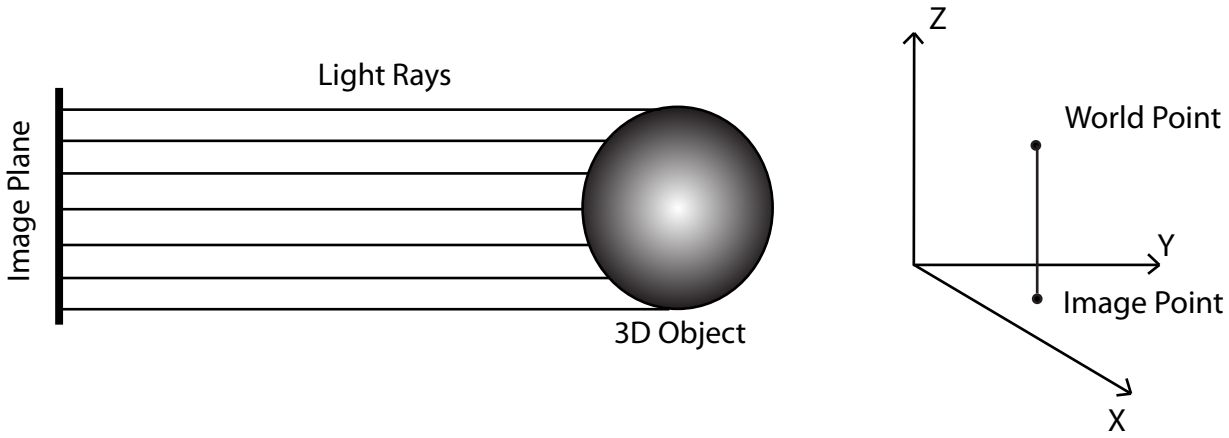


Figure 1.6: **Left:** Illustration of an orthographic camera, where it is assumed that light only travels in parallel lines, thereupon illuminating the photosensitive material. **Right:** This is mathematically equivalent to projecting a world coordinate along an axis, here the z-axis, onto a plane,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

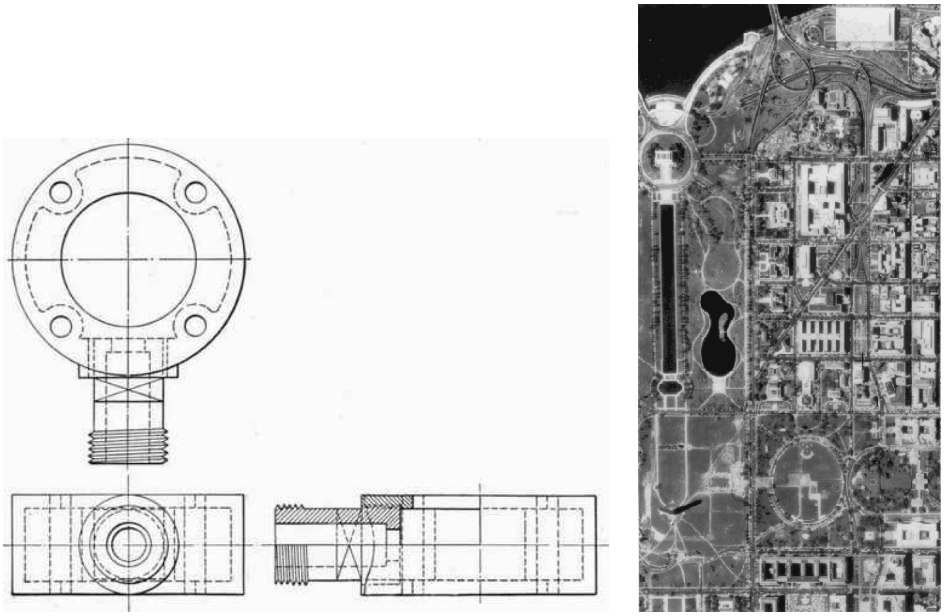


Figure 1.7: Examples of orthographic projections in technical drawings (**Left**) and maps (**Right**). The right image is taken from Wikipedia.

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 is 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 closer 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 manufacturers 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. Firstly, the model is derived in stages to better highlight the different aspects.

1.4.1 Derivation of the Pinhole Camera Model

The idea behind the pinhole camera model is the camera obscura. That 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

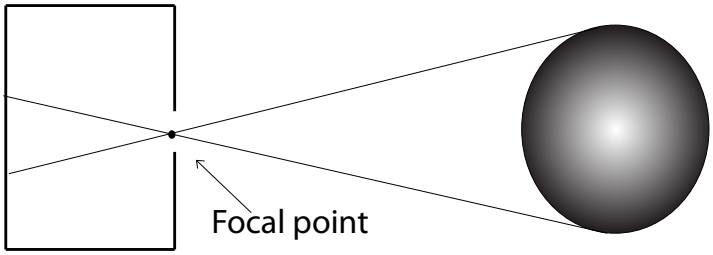


Figure 1.8: Illustration of an ideal pinhole camera, where it is assumed that light only passes through a tiny (i.e. pin) hole in the camera housing, and thus illuminating the photosensitive material.

xz-plane of this camera model, as seen in Figure 1.9. In this 2D world camera it is seen that

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

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, this is illustrated in Figure 1.10, and can be seen by the argument of Figure 1.9 and (1.18) being the same for the image plane in front or behind the optical center.

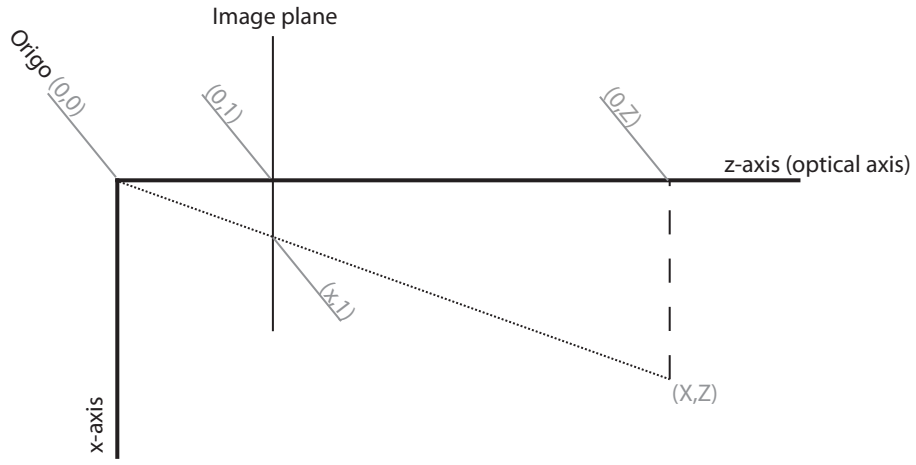


Figure 1.9: The point (X, Z) is projected onto the image plane with coordinates x. It is seen that the triangles $\triangle(0,0),(0,1),(x,1)$ and $\triangle(0,0),(0,Z),(X,Z)$ are scaled versions of each other , and thus $x/1 = X/Z$. Here the image plane is assumed to be unit distance from the origo,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

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

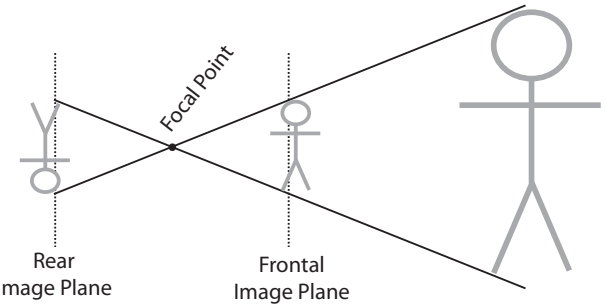


Figure 1.10: Except for a flipping of the image plane, a projection in front of or behind the focal point, or origo, are equivalent, as long as the distance from the focal point is the same.

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.

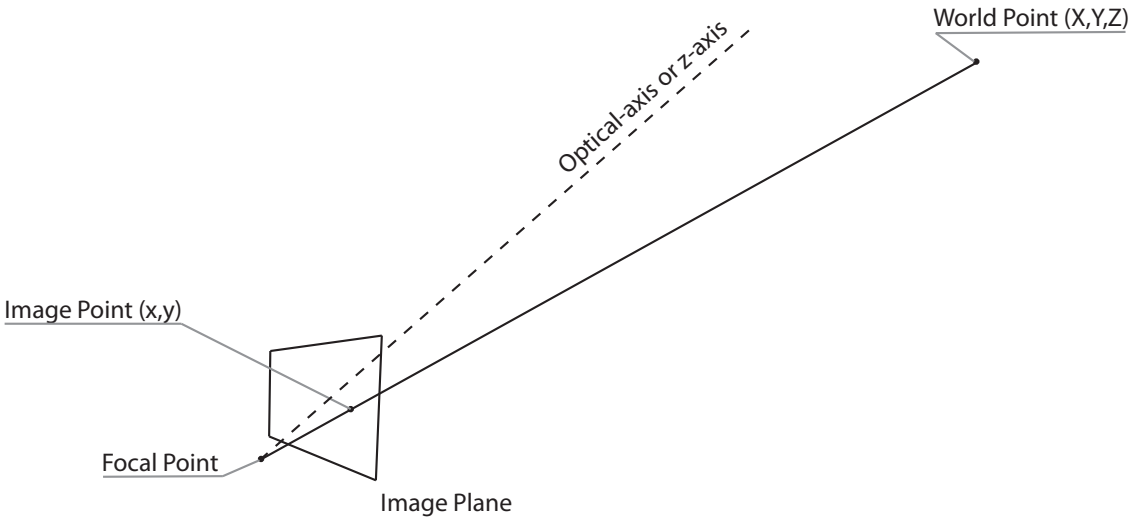


Figure 1.11: An illustration of a projection in 3D where a 3D world point (X, Y, Z), projects to a 2D image point (x, y). This projection can be found by determining where the straight line between the focal point and the 3D point (X, Y, Z) intersects the image plane.

1.4.2 Internal Parameters

So far we have not dealt with the internal part of the camera, such as lenses, film or censor 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

$$\mathbf{A} = \begin{bmatrix} f & f\beta & \Delta x \\ 0 & \alpha f & \Delta y \\ 0 & 0 & 1 \end{bmatrix} . \tag{1.23}$$

The typical way to extend this model is by 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 (1.23) 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 1.4.3.

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, i.e. 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

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

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 (1.23).

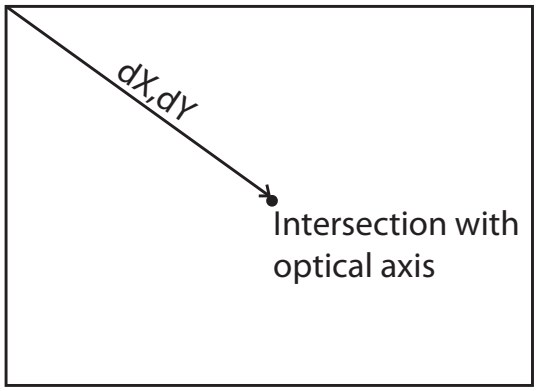


Figure 1.12: The translation vector, $[\Delta x, \Delta y]^T$, needed to get the upper left corner to the origo, is the vector from this image corner to the optical point.

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 modeled 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.

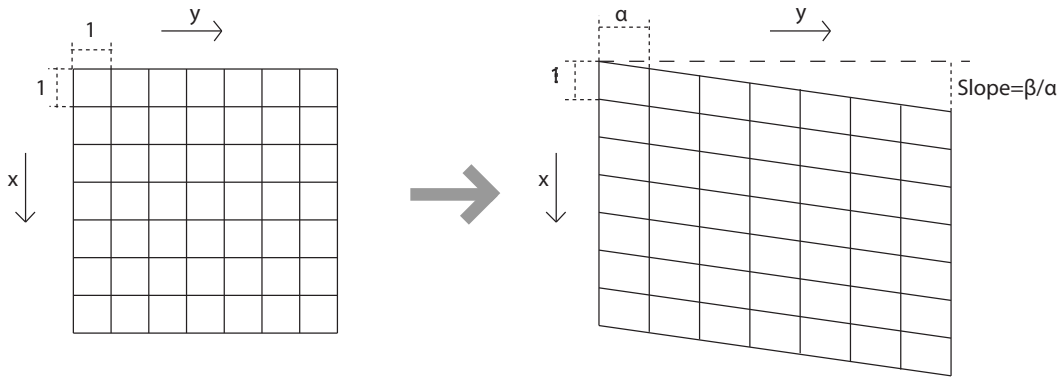


Figure 1.13: The scaling by a factor of α and a shearing of a factor of β is illustrated on a regular grid.

1.4.3 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.

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.

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, cf. [Carstensen(Editor), 2002].

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) , cf. 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 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

$$\begin{aligned} \tan\left(\frac{\theta}{2}\right) &= \frac{l/2}{f} \Rightarrow \\ \frac{\theta}{2} &= \arctan\left(\frac{l/2}{f}\right) \Rightarrow \\ \theta &= 2\arctan\left(\frac{l/2}{f}\right) \ . \end{aligned} \tag{1.24}$$

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

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

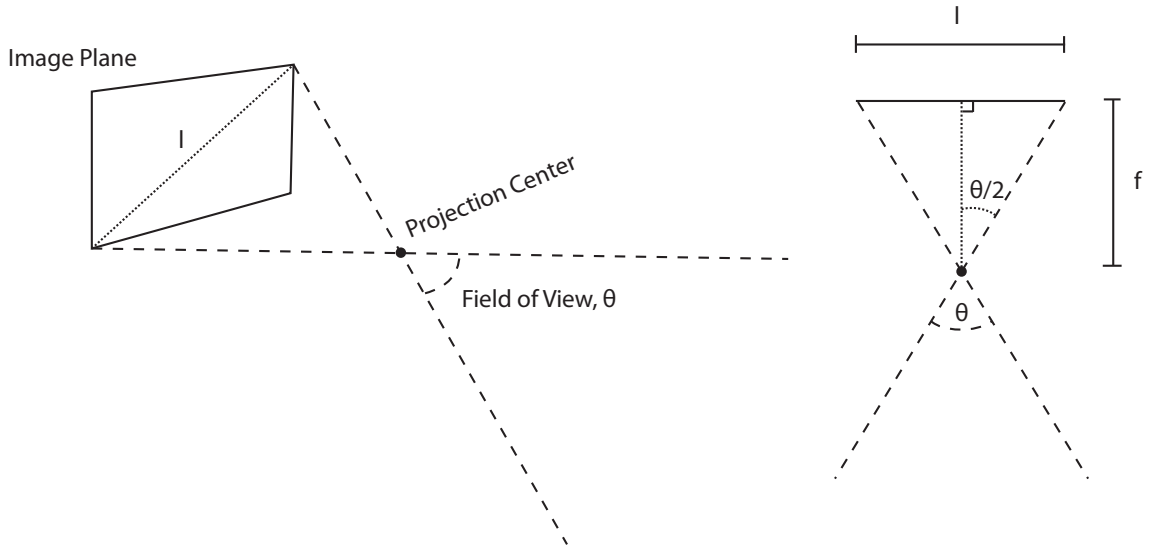


Figure 1.14: A schematic view of the pinhole camera model in relation to the field of view, θ. It is seen that we can form a right angled triangle, with one angle equaling $\theta/2$ and the length of the two sides f and $l/2$.

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, cf. (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

$$\mathbf{l^h}(\alpha) = \mathbf{P}(Q + \alpha S) = \mathbf{P}Q + \alpha \mathbf{P}S = \mathbf{q} + \alpha \mathbf{s} \ , \tag{1.25}$$

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 (1.25) 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.

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

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.

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

$$\mathbf{0} = \mathbf{A} \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} Q_c \quad \Rightarrow \quad \mathbf{0} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} Q_c \quad \Rightarrow \quad \mathbf{0} = \mathbf{R}\tilde{Q}_c + \mathbf{t} \quad \Rightarrow \quad \tilde{Q}_c = -\mathbf{R}^{\mathbf{T}}\mathbf{t} \ . \tag{1.26}$$

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 (1.26) 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);
```

1.4.4 Examples

Figure 1.15: Two images of the same scene, taken with a camera modeled 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, cf. Section 1.4, does not model a camera accurately 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, cf. Section 1.4.3. This error increases with the distance from the center of the image, i.e. near the edges.

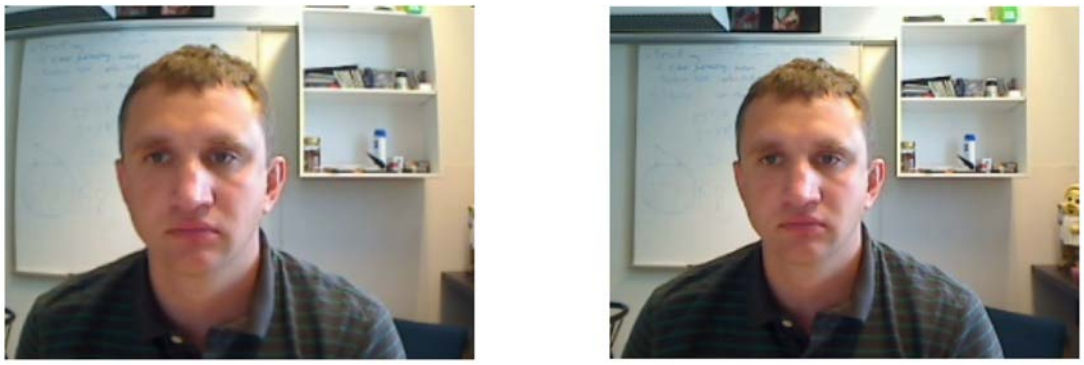


Figure 1.17: The effects of radial distortion, **Left:** An image with (allot) of radial distortion. Notice the rounding of the face, and that the edges are curved, especially close to the edges. **Right:** The same image without radial distortion. This image can be produced from the left image, if the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

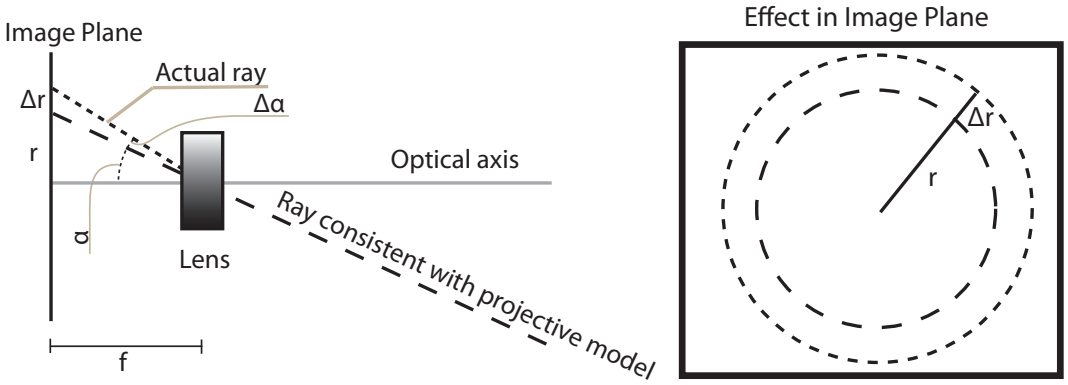


Figure 1.18: **Left:** Radial distortion is a nonlinear modeling of the effect, that a ray of light is 'bent' with the amount of $\Delta\alpha$, where $\Delta\alpha$ is a function of the angle α to the optical axis. Since the radius, r, and the angle α are related by $\alpha = \arctan\left(\frac{r}{f}\right)$, 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 internal camera model in two, i.e. $\mathbf{A}$ in (1.23). 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

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

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

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

Here the $\mathbf{A}$ of (1.23) 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

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

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 modeling $\Delta(r_i)$ is as a polynomial in r_i. There is no real physical rationale behind this modeling, 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 [Heikkila, 2000]. The effects of radial distortion have also had a nomenclature attached to them, namely *barrel distortion* 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*, cf. e.g. [Heikkila, 2000].

1.5.1 An Example

Here the example on page 24 is extended by assuming that the image in Figure 1.4.4 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, cf. Section 1.7.2.

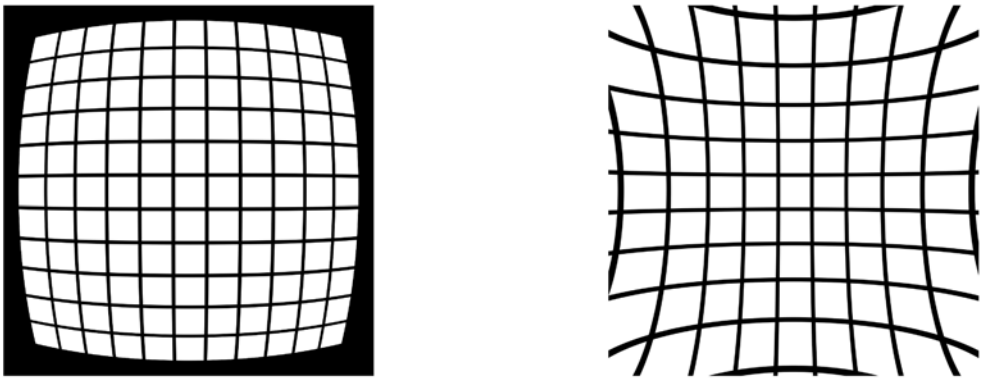


Figure 1.19: The effects of radial distortion on a regular grid, **Left:** Barrel distortion. **Right:** Pincushion distortion.

Then, according to the values supplied on page 24

$$\begin{aligned}
\mathbf{A_p} &= \begin{bmatrix} 2774.5 & 0 & 0 \\ 0 & 2774.5 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\
\mathbf{A_q} &= \begin{bmatrix} 1 & 0 & 806.8 \\ 0 & 1 & 622.6 \\ 0 & 0 & 1 \end{bmatrix} \\
\mathbf{A_p} \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} &= \begin{bmatrix} 2743.1 & -1.0 & 416.2 & -6051.6 \\ 2.3 & 2774.5 & -8.4 & 110.7 \\ -0.2 & 0.0 & 1.0 & 0.5 \end{bmatrix}
\end{aligned}$$

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 (cf. page 1.4.4)

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

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.

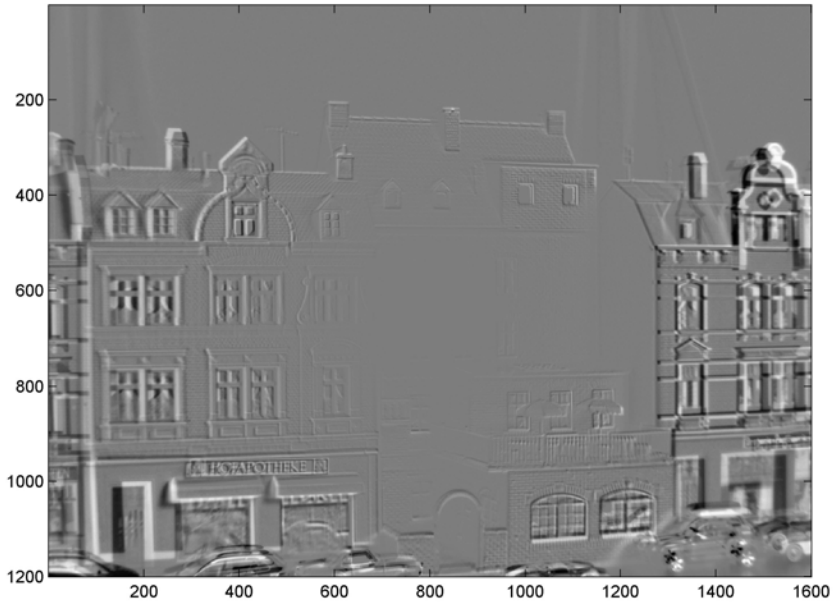


Figure 1.21: The difference image between the images in Figure 1.16 and Figure 1.20. Notice how the deviation of the two images increases away from the center, which is consistent with the effect of radial distortion being largest there.

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

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

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 cf. e.g. [Hartley and Zisserman, 2003]. 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, cf. Section 2.4.3. Therefore, define the function $\Pi(\mathbf{q_i})$, which takes a homogeneous coordinate and returns a inhomogeneous coordinate, i.e.

$$\begin{bmatrix} x \\ y \end{bmatrix} = \Pi\left(\begin{bmatrix} sx \\ sy \\ s \end{bmatrix}\right) = \begin{bmatrix} \frac{sx}{s} \\ \frac{sy}{s} \end{bmatrix} \quad . \tag{1.32}$$

The camera calibration problem thus becomes

$$\min_{\mathbf{P}} \sum_i \|\Pi(q_i) - \Pi(\mathbf{P}Q_i)\|_2^2 \quad . \tag{1.33}$$

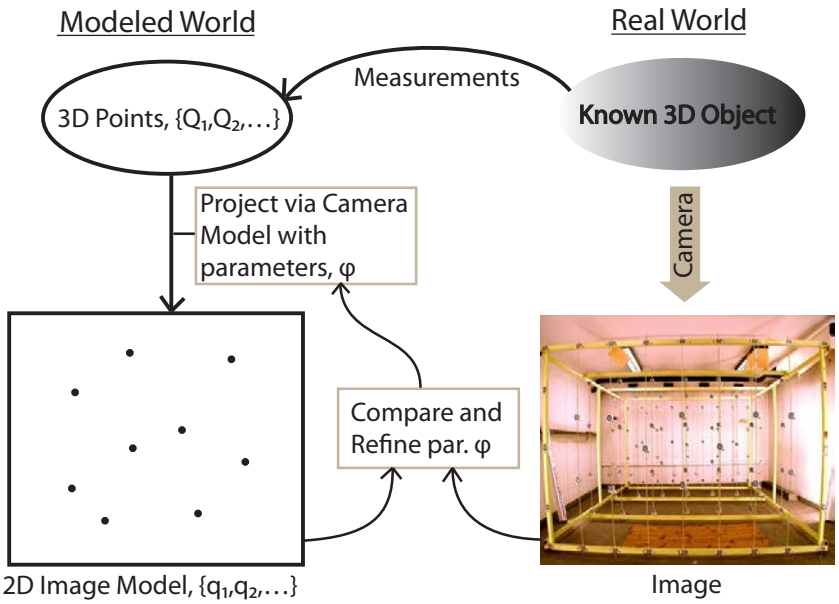


Figure 1.22: The camera calibration procedure works by modeling the 3D world via 3D point measurements. These measured 3D points are the projected into the model image plane via the camera model. These 2D model measurements are compared to the real image of the known 3D object. Based on this comparison the camera model parameters, φ , 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, cf. 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 (1.33) becomes ill-posed.

There are several free online software packages for doing camera calibration, e.g. an implementation of the method in [Heikkila, 2000] is available from the authors home page. 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 pattern from several angles.

1.7 End Notes

Here a few extra notes will be made on camera modeling, by briefly touching on what is modeled 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 modeled 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-Left. 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, cf. e.g. [Favaro and Soatto, 2005]. Along side the geometric camera properties, a lot of effort has also been used on modeling 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-Right.

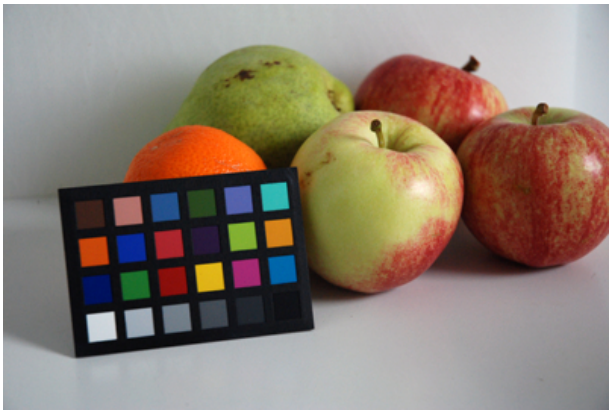



Figure 1.23: **Left:** Example of depth of field of a camera, note that it is only the flower and a couple of straws that are in focus. **Right:** Example of a color calibration card. This colors of the squares in the card a very well know. As such the image can be chromatically calibrated.

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) & (1.23)

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

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 modeling 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 [Hartley and Zisserman, 2003] 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. all these can be seen as comprised of points, but points are the simplest to work with. Thus, the theory developed for points will also generalize to most other features, and will give the basic understanding of 3D estimation.

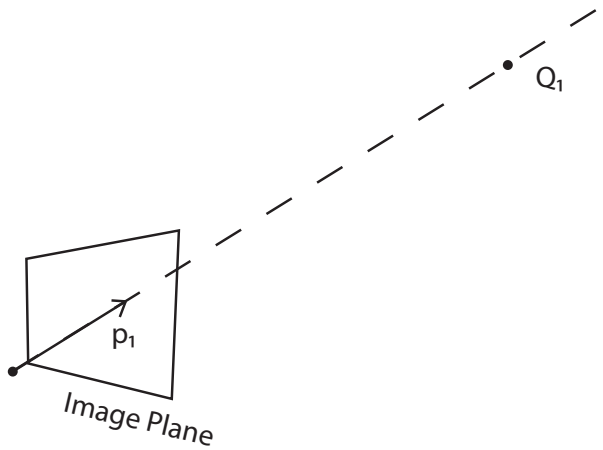


Figure 2.1: The back projection of the point $\mathbf{Q_1}$ is the dotted line in the direction of the vector $\mathbf{p_1}$. Thus the 3D point Q_1, which projects to $\mathbf{q_1}$, must lie on this back projected line.

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

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

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

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

which is equal to a line through $\mathbf{t}$ and with direction $\mathbf{R^T}\mathbf{p_i}$, 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 ray of light has hit it from a given direction, $\mathbf{p_i}$, but has no information on how far the light has traveled. 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 2.1. 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 to generalize from 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.

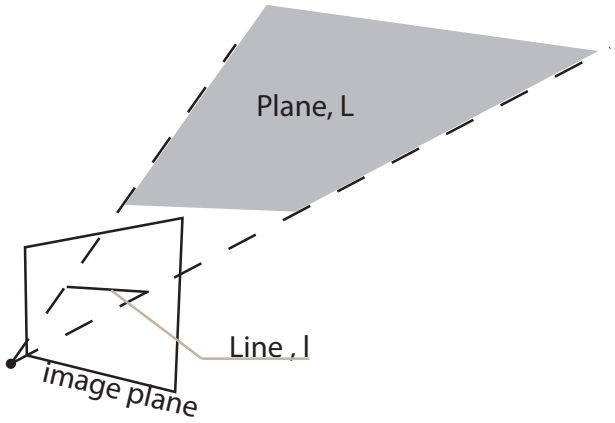


Figure 2.2: The image of a line back projects to a plane. All 3D points projecting to this 2D image line must thus be on this back projected plane.

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

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

[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 modeled 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 geometry, 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, cf. Appendix A.4. 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 $\mathbf{p_2} = \mathbf{A_2^{-1}} \mathbf{q_2}$.

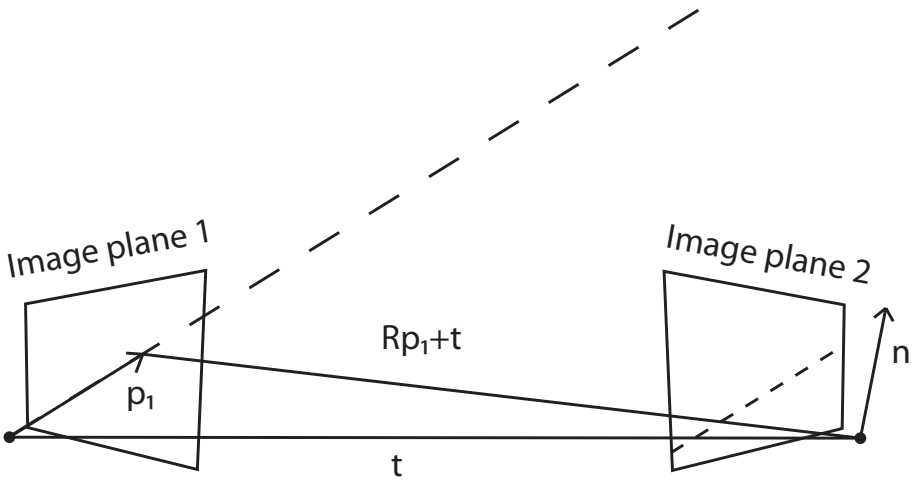


Figure 2.3: The two camera centers and the point $\mathbf{p_1}$ define a plane that also includes the back projection of point $\mathbf{q_1}$. The normal of this epipolar plane is n, and the intersection of this plane ad camera two's image plane is the projection of the back projected line of $\mathbf{q_1}$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

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

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) yields

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

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 to notation is abused somewhat, by letting $A_2^{-T} = \left(A_2^{-1}\right)^T$.

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, cf. 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 [Hartley and Zisserman, 2003].

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. A similar thing holds for image one, see Figure 2.5 and Figure 2.6. The projection of these camera centers are called the *epipoles*.

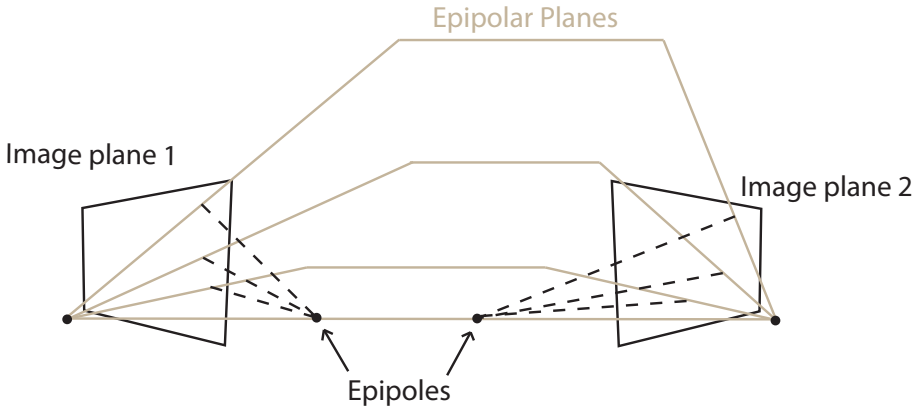


Figure 2.5: Since all the epipolar planes intersect in a common line – i.e. line connecting the two camera centers – all the epipolar lines will intersect in a common point, called the epipole. This epipole is often located outside the physical image.

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

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

Using that

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

Here it is used that the cross product between a vector and itself is zero, and that $[\mathbf{t}]_\times$ is skew symmetric such that $[\mathbf{t}]_\times = -[\mathbf{t}]_\times^T$. 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 symmetric 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 cf. [Hartley and Zisserman, 2003], 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 2D homogeneous coordinates $\mathbf{q_1}$ and $\mathbf{q_2}$, as follows[3]:

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

[3]Here we only consider 2D coordinates, but the concept of homographes generalizes to any dimension.

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 [Hartley and Zisserman, 2003].

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, cf. (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 $\mathbf{q_p} = [a, b, 1]^T$ denotes the homogeneous coordinate of (a, b). Assuming that the plane in question is viewed by a camera described by the pinhole camera model, cf. (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, cf. [Hartley and Zisserman, 2003], 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:

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

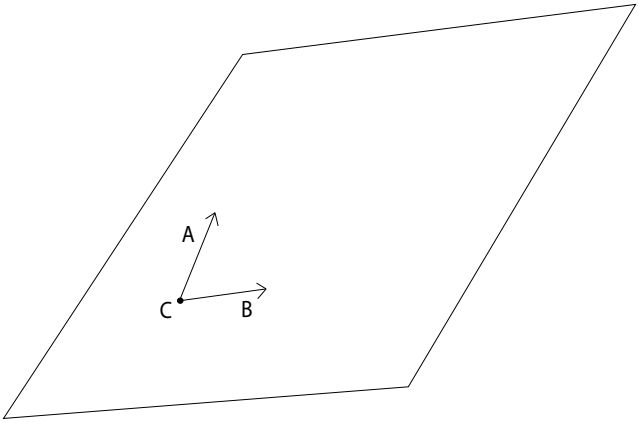


Figure 2.7: All point on a plane in 3D, can be represented by a point in the plane, C, plus a linear combination of two vectors in the plane, A and B. Here the linear combination of A and B consists of a local coordinate system.

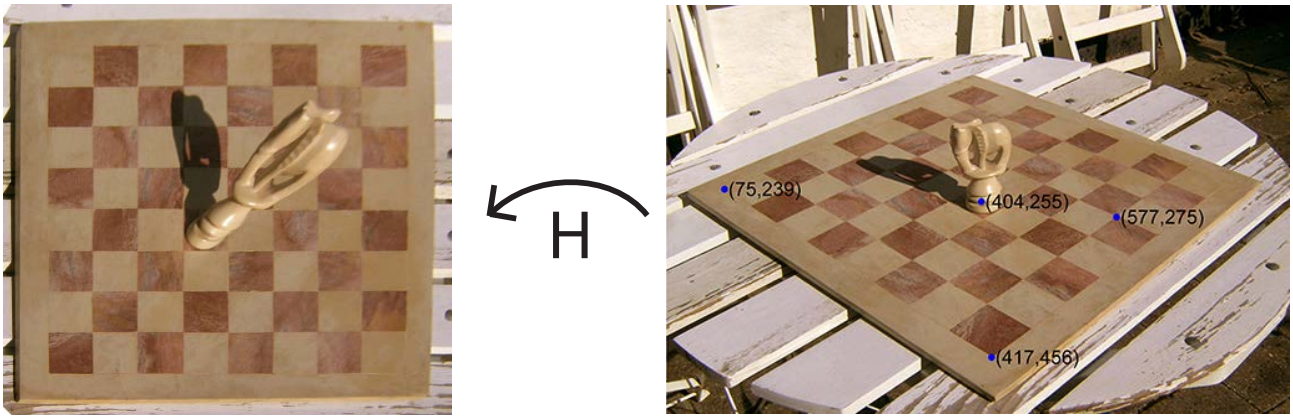


Figure 2.8: An image of a chess board to the right, and a warped version of it, via the homography $\mathbf{H}$, to the left. The left image is a pseudo view of how the chess board would look from straigh above, or a so called fronto parallel view.

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[4]. 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 the 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:

$$\mathbf{P_1} = \mathbf{A_1} \left[\begin{array}{cc} \mathbf{I} & \mathbf{0} \end{array} \right] \quad , \quad \mathbf{P_2} = \mathbf{A_2} \left[\begin{array}{cc} \mathbf{R} & \mathbf{0} \end{array} \right] \ .$$

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

$$\begin{aligned} \mathbf{q_1} &= \mathbf{A_1} \left[\begin{array}{cc} \mathbf{I} & \mathbf{0} \end{array} \right] Q \quad \Rightarrow \\ \mathbf{p_1} &= \mathbf{A_1^{-1} q_1} = \left[\begin{array}{cc} \mathbf{I} & \mathbf{0} \end{array} \right] Q = \tilde{Q} \ , \end{aligned}$$

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

$$\begin{aligned} \mathbf{q_2} &= \mathbf{A_2} \left[\begin{array}{cc} \mathbf{R} & \mathbf{0} \end{array} \right] Q \\ &= \mathbf{A_2 R} \tilde{Q} \\ &= \mathbf{A_2 R A_1^{-1} q_1} \end{aligned} \tag{2.19}$$

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.

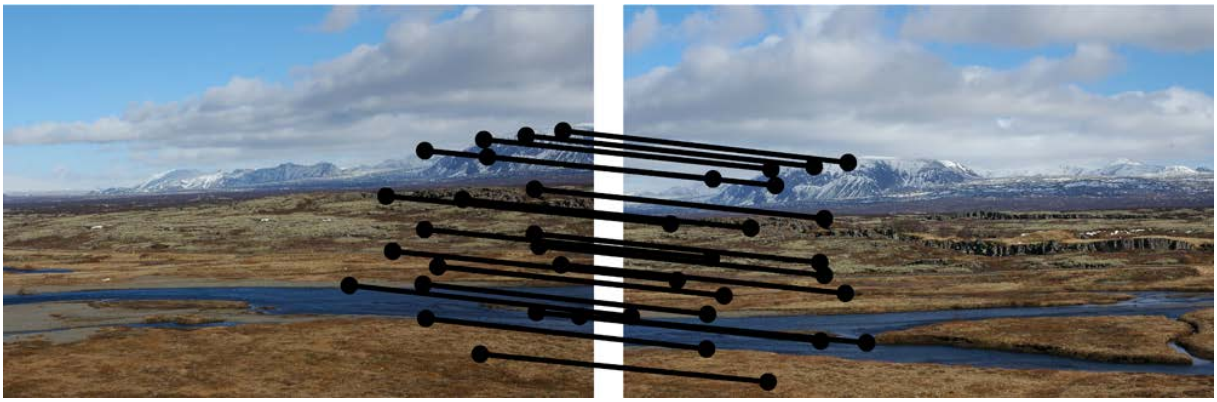


Figure 2.9: Two landscape images taken from the same spot, i.e. zero baseline. The black lines and dots illustrate the 20 points correspondences manually annotated.

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, cf. Figure 2.10. This result is generated from the two images in Figure 2.9, where 20 point correspondences have been annotated manually, and the method of Section 2.9 is used to estimate the needed homography. This homography is then used to warp the left image of Figure 2.9, cf. 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,

[4]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.

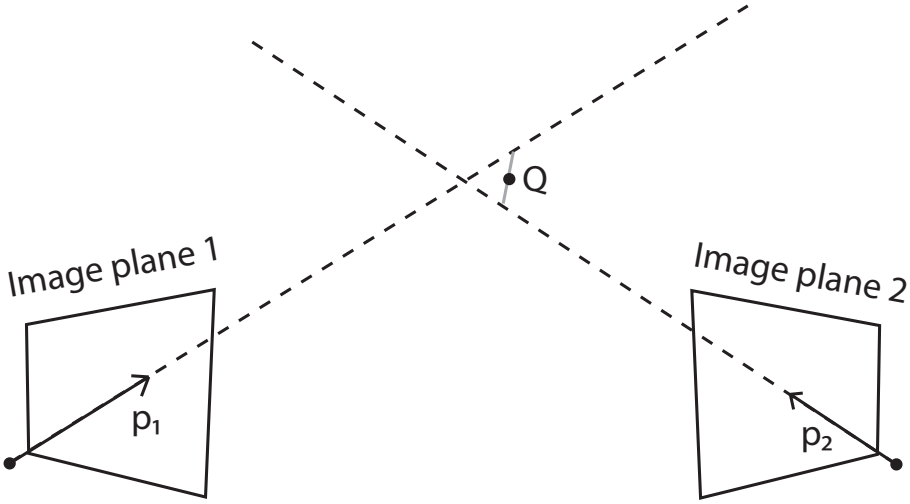


Figure 2.11: The result of point triangulation is the 3D point, Q, closest to the back projections of the observed 2D points.

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[5]

$$\mathbf{B} = \begin{bmatrix} P_1^3 x_1 - P_1^1 \\ P_1^3 y_1 - P_1^2 \\ P_2^3 x_2 - P_2^1 \\ P_2^3 y_2 - P_2^2 \\ \vdots \\ P_n^3 x_n - P_n^1 \\ P_n^3 y_n - P_n^2 \end{bmatrix} ,$$

then (2.22) is equivalent to

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

The extra constraint that $||Q|| \neq 0$ is , however, needed, since this is an uninformative – also called trivial – solution to all these estimation problems. More specifically, Q is a homogeneous representation of a 3D point, and thus defined up to scale, this scale should just not be zero. A practical way of ensuring this mathematically, is by requiring that

$$||Q|| = 1 \ ,$$

hereby fixing the scale. With noisy measurements (2.22) will however will not hold perfectly, and we instead solve

$$\min_Q \|\mathbf{B}Q\|_2^2 \quad \text{where} \quad ||Q|| = 1 \ . \tag{2.23}$$

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

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

Degeneracy

When solving the linear system (2.23), it is important that the linear system is well determined and well conditioned. Problems can arise if the distance between the cameras is very large or very small compared to the distance to the 3D point Q, cf. Section 2.5. For a given system of equations, $\mathbf{B}$, this can be checked by considering the singular values of $\mathbf{B}$, cf. the above MatLab code and Appendix A.6. For the system to be well conditioned the difference between the smallest, σ_k, and second smallest, σ_{k-1}, should be large. If e.g. both these singular values were close to zero, then this would be that same as the estimate of Q existed in a one dimensional subspace, etc.

2.4.2 Examples

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

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

and the corresponding cameras are given by

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

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

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.

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

$$\mathbf{B} = \begin{bmatrix} P_1^3 x_1 - P_1^1 \\ P_1^3 y_1 - P_1^2 \\ P_2^3 x_2 - P_2^1 \\ P_2^3 y_2 - P_2^2 \end{bmatrix} = \begin{bmatrix} -2068 & 212 & 2342 & -39501 \\ -631 & -3047 & -314 & -3582 \\ -3160 & 240 & -27 & -13571 \\ -298 & -3071 & -587 & 1371 \end{bmatrix} .$$

The solution to (2.23) is then given by

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

which is the linear estimate of the 3D point. This example is made from real data and thus have rounding errors, which becomes outspoken with the precision used here. A more theoretical example is thus, given two cameras described by[6]

$$\mathbf{P_1} = \begin{bmatrix} 800 & 0 & 300 & 0 \\ 0 & 800 & 400 & -2400 \\ 0 & 0 & 1 & 0 \end{bmatrix} , \quad \mathbf{P_2} = \begin{bmatrix} 800 & 0 & 300 & 0 \\ 0 & 800 & 400 & 2400 \\ 0 & 0 & 1 & 0 \end{bmatrix} .$$

In these two cameras a 3D point Q is depicted at

$$\mathbf{q_1} = \begin{bmatrix} 300 \\ 160 \\ 1 \end{bmatrix} , \quad \mathbf{q_2} = \begin{bmatrix} 300 \\ 640 \\ 1 \end{bmatrix} .$$

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

$$\mathbf{B} = \begin{bmatrix} P_1^3 x_1 - P_1^1 \\ P_1^3 y_1 - P_1^2 \\ P_2^3 x_2 - P_2^1 \\ P_2^3 y_2 - P_2^2 \end{bmatrix} = \begin{bmatrix} -800 & 0 & 0 & 0 \\ 0 & -800 & -240 & 2400 \\ -800 & 0 & 0 & 0 \\ 0 & -800 & 240 & -2400 \end{bmatrix} ,$$

and the solution to (2.23) is then given by

$$Q = \begin{bmatrix} 0 \\ 0 \\ 10 \\ 1 \end{bmatrix} ,$$

which can be verified by $\mathbf{B}Q = \mathbf{0}$.

[6]The only difference is the sign of 2400.

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

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

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, cf. 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, cf. Figure 2.13-Right, and unmodeled optical phenomena, such as radial distortion. Both these entities are captured well by a deviation in the point location, as in (2.24).

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 (1.33) in Section 1.6. What we want to minimize, w.r.t. Q, is

$$\min_Q \sum_{i=1}^n \left\| \begin{bmatrix} \varepsilon_i^x \\ \varepsilon_i^y \end{bmatrix} \right\|_2^2 = \min_Q \sum_{i=1}^n \|\Pi(q_i) - \Pi(\mathbf{P}Q_i)\|_2^2 \ . \tag{2.27}$$

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.

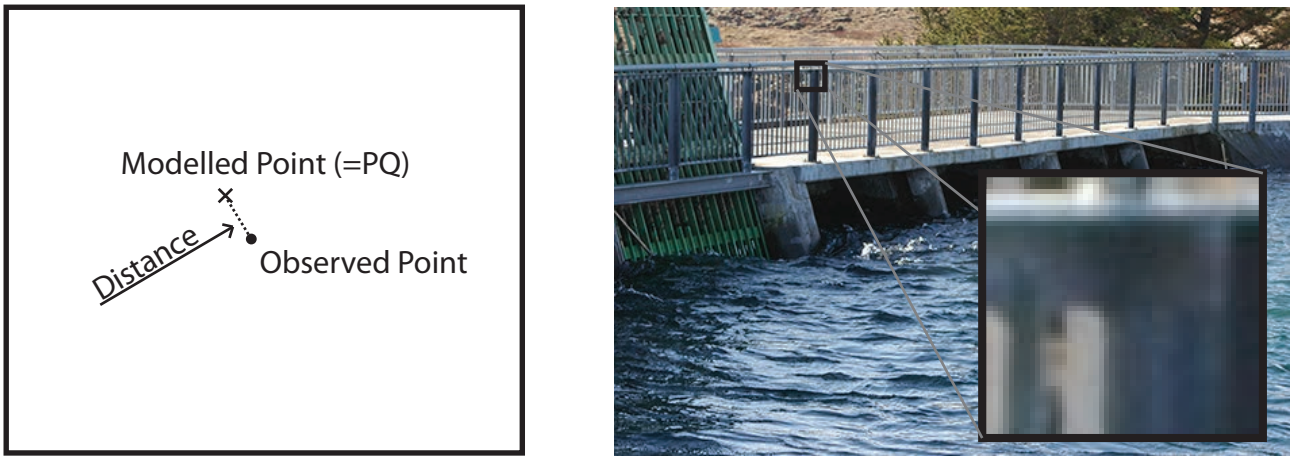


Figure 2.13: **Left:** A Schematic view of the distance to be minimized, namely the distance between the observed point at the one originating from the model. In the point triangulation case the model is $\mathbf{q} = \mathbf{P}Q$. **Right:** An illustration that even though an image apparently has many 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.

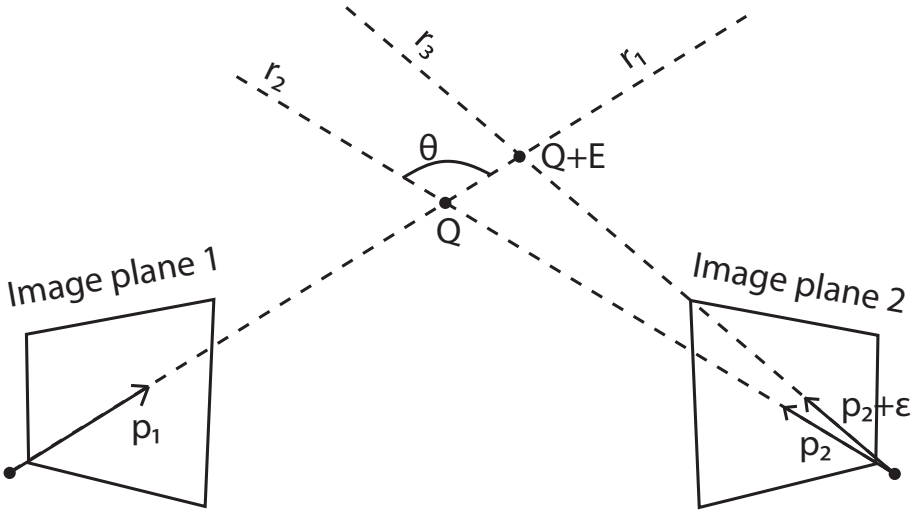


Figure 2.14: A schematic view of point triangulation of the 3D point $\mathbf{Q}$, from the projections $\mathbf{p_1}$ and $\mathbf{p_2}$ in cameras one and two respectively. To investigate the effect of an error in an image coordinate, we assume that $\mathbf{p_2}$ is offset by ε, resulting in an error on $\mathbf{Q}$ of $\mathbf{E}$. The optical rays resulting from three image coordinates are denoted by $\mathbf{r_1}$,$\mathbf{r_2}$ and $\mathbf{r_3}$. The angel between $\mathbf{r_1}$ and $\mathbf{r_2}$ is denoted by θ.

2.5 Camera Measurement Setup

When doing 3D measurements from cameraes, i.e. point triangulation, the accuracy naturally depends on the configuration or spacial relation of the cameras and the 3D points to be measured. To give an accurate estimate of the accuracy, the various sources of noise have to be accurately modeled and propagated to the 3D estimate. Some general considerations can however be made, and will be presented here, resulting in some rules of thumb useful for designing 3D measurement setups.

From the above presentation of the pinhole camera model, it is noted that most of sources of error can be seen as inaccurate image coordinate, $\mathbf{p_i}$, resulting in an angular error as seen in Figure 2.14. As illustrated in this figure, we investigate the effect of an error of the image coordinate[7] by adding an error, ε, to the projection in the second image, $\mathbf{p_2}$. As seen from Figure 2.15, the angle, θ, between the rays $\mathbf{r_1}$ and $\mathbf{r_2}$ is highly related to the spatial relation of the cameras and the 3D point. The effect of this angle will be investigated further here, thus consider Figure 2.16, which is a magnification of Figure 2.14 around the 3D point $\mathbf{Q}$.

[7]This error is along the epipolar line.

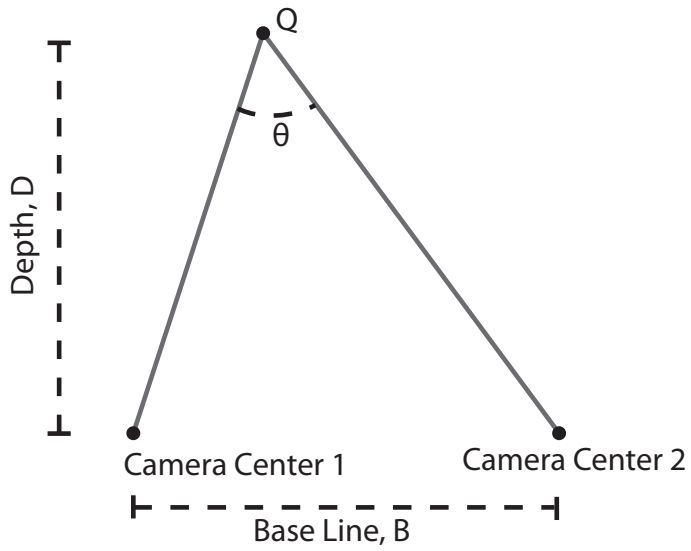


Figure 2.15: Illustration of the baseline, B, and depth, D, for a two camera setup.

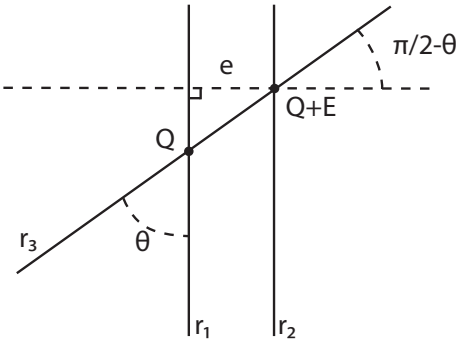


Figure 2.16: An amplification of Figure 2.14, around Q.Here it is seen that the closer θ is to $\frac{\pi}{2}$ the smaller $\mathbf{E}$ is relative to $\mathbf{e}$.

In Figure 2.16 it is assumed that the rays $\mathbf{r_1}$ and $\mathbf{r_2}$ are parallel, to simplify calculations. This is a reasonable assumption since ε is assumed small and only rules of thumb are derived. Denote the distance between $\mathbf{r_1}$ and $\mathbf{r_2}$ in the vicinity of $\mathbf{Q}$ by $\mathbf{e}$, then it is seen from Figure 2.16 that

$$\frac{\mathbf{e}}{\mathbf{E}} = \cos(\pi/2 - \theta) \Rightarrow \mathbf{E} = \frac{\mathbf{e}}{\cos(\pi/2 - \theta)} = \frac{\mathbf{e}}{\sin(\theta)} \ . \tag{2.28}$$

Since $\sin(a)$ is zero for $a = 0$ or $a = \pi$, the error $\mathbf{e}$ will be divided by a small number (equivalent to a large amplification) for θ close to zero or π. Thus the error, $\mathbf{E}$, is smallest when θ is close to $\pi/2$ or $90°$. The value of θ, is related to the baseline, B, and depth, D of the camera set up. Thus a general rule of thumb, used in photogrammetry [Carstensen(Editor), 2002], is that the relationship between baseline, B, and depth, D, should be

$$\frac{1}{3} < \frac{B}{D} < 3 \ . \tag{2.29}$$

Two notes should be made. Firstly, there will also be an error in image one, which will have a similar effect as that of image two, also dependent on θ. Secondly, when using more than two cameras, this rule of thumb indicates that for any 3D point there should be at least be a pair of cameras with a reasonable ratio between baseline and depth.

Naturally, the size of $\mathbf{e}$ and thus $\mathbf{E}$ will also depend on the size of the measurement setup it self, i.e. the length of the baseline and depth. This effect is, however, less subtle then that of the ratio between baseline and depth. Also the size of the setup is usually dictated by the size of the object (i.e. the collection of 3D points) to be measured, in that the whole object should be in the field of view.

2.6 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, cf. Chapter 6 . 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, cf. Figure 2.18-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.

2.6.1 Laser Scanners

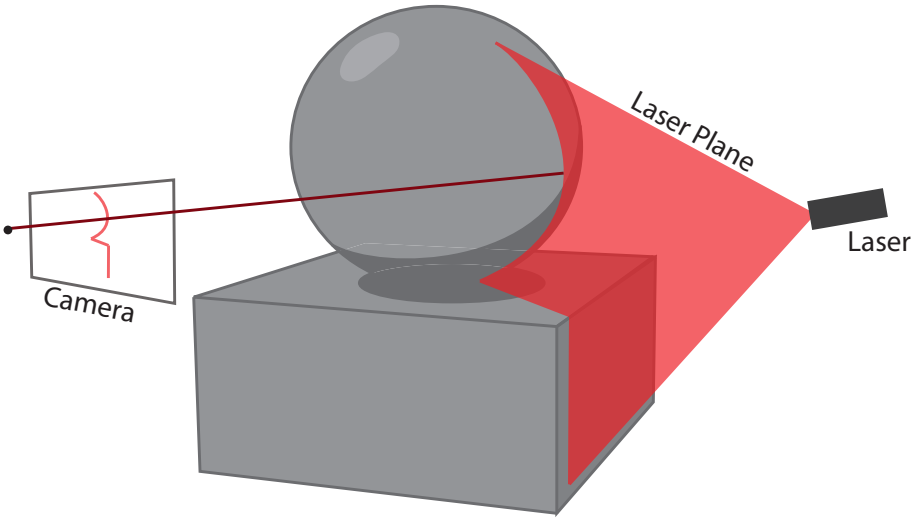


Figure 2.17: A schematic illustration of a laser scanner. The laser emits a laser plane that hits the surface and reflects light into the camera. The intersection of the laser plane and the back projected line defines a 3D point Q.

A laser scanner basically consists of a camera and a laser emitting a point or a plane. Here the laser plane version is considered, cf. Figure 2.17 and Figure 2.18. 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 2.17. 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.

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 derivation 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

$$\mathbf{B}Q = \begin{bmatrix} P^3 x - P^1 \\ P^3 y - P^2 \\ L^T \end{bmatrix} Q = 0 \ . \tag{2.30}$$

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 cf. [Levoy et al., 2000], as depicted in Figure 2.18.

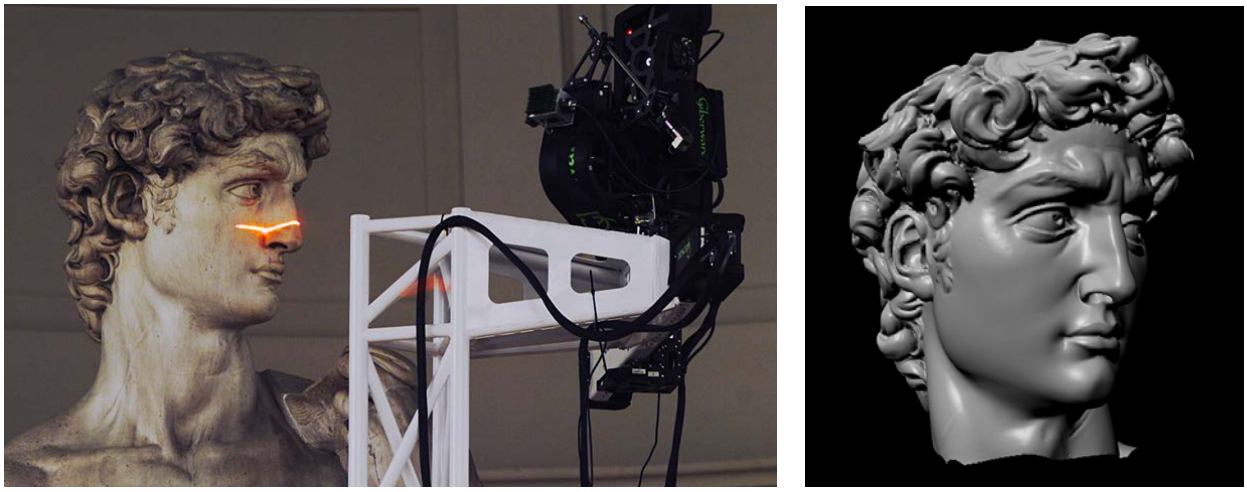


Figure 2.18: results from the digital michelangelo project cf. [Levoy et al., 2000]. **Left:** The laser plane and camera set up applied to the head of David. **Right:** The scanned result, where allot of 3D points have been scanned and used as vertices of a triangular mesh, which is rendered here.

2.7 Camera Resection

Just as a 3D point was estimated from the corresponding 2D projections in known cameras, cf. 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 (1.33). 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. For higher quality solutions, a non-linear optimization of (1.33) should thus follow. If resectioning is to be done for a camera with known internal parameters, special purpose algorithms exist, cf. e.g. [Haralick et al., 1994].

2.7.1 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)

$$
\begin{aligned}
&\mathbf{q_i} = \mathbf{P}Q_i \qquad \Rightarrow \\
&\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} = [\mathbf{q_i}]_\times \mathbf{P} Q_i \\
&= \begin{bmatrix} 0 & -1 & y_i \\ 1 & 0 & -x_i \\ -y_i & x_i & 0 \end{bmatrix} \begin{bmatrix} P_{11} & P_{12} & P_{13} & P_{14} \\ P_{21} & P_{22} & P_{23} & P_{24} \\ P_{31} & P_{32} & P_{33} & P_{34} \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix} \\
&= \begin{bmatrix} 0 & -1 & y_i \\ 1 & 0 & -x_i \\ -y_i & x_i & 0 \end{bmatrix} \begin{bmatrix} P_{11}X_i + P_{12}Y_i + P_{13}Z_i + P_{14} \\ P_{21}X_i + P_{22}Y_i + P_{23}Z_i + P_{24} \\ P_{31}X_i + P_{32}Y_i + P_{33}Z_i + P_{34} \end{bmatrix} \\
&= \begin{bmatrix} -P_{21}X_i + y_iP_{31}X_i - P_{22}Y_i + y_iP_{32}Y_i - P_{23}Z_i + y_iP_{33}Z_i - P_{24} + y_iP_{34} \\ P_{11}X_i - x_iP_{31}X_i + P_{12}Y_i - x_iP_{32}Y_i + P_{13}Z_i - x_iP_{33}Z_i + P_{14} - x_iP_{34} \\ -y_iP_{11}X_i + x_iP_{21}X_i - y_iP_{12}Y_i + x_iP_{22}Y_i - y_iP_{13}Z_i + x_iP_{23}Z_i - y_iP_{14} + x_iP_{24} \end{bmatrix} \\
&= \begin{bmatrix} 0 & -X_i & y_iX_i & 0 & -Y_i & y_iY_i & 0 & -Z_i & y_iZ_i & 0 & -1 & y_i \\ X_i & 0 & -x_iX_i & Y_i & 0 & -x_iY_i & Z_i & 0 & -x_iZ_i & 1 & 0 & -x_i \\ -y_iX_i & x_iX_i & 0 & -y_iY_i & x_iY_i & 0 & -y_iZ_i & x_iZ_i & 0 & -y_i & x_i & 0 \end{bmatrix} \mathrm{vec}(\mathbf{P}) \\
&= \mathbf{b_i^T} \cdot \mathrm{vec}(\mathbf{P}) \ . \qquad (2.31)
\end{aligned}
$$

Where

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

The first step of the calculations in (2.31) 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 (2.31).

As in Section 2.4.1 $\mathrm{vec}(\mathbf{P})$, and thus $\mathbf{P}$, is estimated from (2.31) 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

$$
\min_{\mathrm{vec}(\mathbf{P})} ||\mathbf{B} \cdot \mathrm{vec}(\mathbf{P})||_2^2 \quad \text{where} \quad ||\mathrm{vec}(\mathbf{P})|| = 1 \ . \qquad (2.32)
$$

Here the constraint on $||\mathrm{vec}(\mathbf{P})||$ is included to avoid the trivial null solution.

Even though the $\mathbf{b_i^T}$ are of dimension 3×12 they 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.

As mentioned is Section 2.4.1, it is important that the system of equations (2.32) are well formed. In the case camera resection[8], a necessary condition for the problem to be well formed, implying that $\mathbf{B}$ is well conditioned, is that the 3D points span the 3D space well. I.e. that the all the 3D points used are not located on or near a plane.

[8] Assuming a pinhole camera model with unknown internal parameters.

Use of the Kronecker Product

The amount of terms in (2.31) illustrates the power of using the Kronecker product, cf. 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 (cf. Appendix A.7)

$$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} = [\mathbf{q_i}]_\times \mathbf{P} Q_i = \left(Q_i^T \otimes [\mathbf{q_i}]_\times\right) \cdot \mathrm{vec}(\mathbf{P}) \ , \tag{2.33}$$

which is equivalent to (2.31). Actually using the Kronecker product as in (2.33) is often preferable to the formulation in (2.31), simply due to the number of typing mistakes avoided.

2.8 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.

2.8.1 The 8-Point Algorithm

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

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

Inserting this into (2.11), gives for each i

$$\begin{aligned} 0 &= \mathbf{q_{2i}^T F q_{1i}} \\ &= \begin{bmatrix} x_{2i} \\ y_{2i} \\ 1 \end{bmatrix}^T \begin{bmatrix} F_{11} & F_{12} & F_{13} \\ F_{21} & F_{22} & F_{23} \\ F_{31} & F_{32} & F_{33} \end{bmatrix} \begin{bmatrix} x_{1i} \\ y_{1i} \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} x_{2i} \\ y_{2i} \\ 1 \end{bmatrix}^T \begin{bmatrix} F_{11}x_{1i} + F_{12}y_{1i} + F_{13} \\ F_{21}x_{1i} + F_{22}y_{1i} + F_{23} \\ F_{31}x_{1i} + F_{32}y_{1i} + F_{33} \end{bmatrix} \\ &= F_{11}x_{1i}x_{2i} + F_{21}x_{1i}y_{2i} + F_{31}x_{1i} + F_{12}y_{1i}x_{2i} + F_{22}y_{1i}y_{2i} + F_{32}y_{1i} + F_{13}x_{2i} + F_{23}y_{2i} + F_{33} \\ &= \begin{bmatrix} x_{1i}x_{2i} & x_{1i}y_{2i} & x_{1i} & y_{1i}x_{2i} & y_{1i}y_{2i} & y_{1i} & x_{2i} & y_{2i} & 1 \end{bmatrix} \mathrm{vec}(\mathbf{F}) \\ &= \mathbf{b_i^T} \cdot \mathrm{vec}(\mathbf{F}) \ , \end{aligned} \tag{2.34}$$

where

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

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

$$\min_{\mathrm{vec}(\mathbf{F})} ||\mathbf{B} \cdot \mathrm{vec}(\mathbf{F})||_2^2 \ . \tag{2.35}$$

Since this algorithm does not exploit the the fact that the fundamental matrix has a determinant of zero, cf. 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. Also it is important that the problem and thus the linear system $\mathbf{B}$ is well formed, cf. Section 2.4.1. When estimating the fundamental matrix a necesary condition for a well formed problem is that the points viewed span 3D space well, i.e. that all the points are not located on or near a plane.

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 (2.35), will have a null space of dimension two. The solution to (2.35) 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

$$\det(\mathbf{F}) = \det(\alpha\mathbf{F}' + \mathbf{F}^\dagger) = 0 \ . \tag{2.36}$$

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 [Hartley and Zisserman, 2003].

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 outlined in Section 3.2.1. This algorithm requires 5 point correspondences, hance the algorithm is called the 5-point algorithm.

Use of the Kronecker Product

Just as in Section 2.7, the Kronecker product can make the calculations in (2.34) easier and less prone to typing mistakes, cf. 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 (cf. Appendix A.7)

$$0 = \mathbf{q_{2i}^T F q_{1i}} = \left(\mathbf{q_{1i}^T} \otimes \mathbf{q_{2i}^T}\right) \cdot \mathrm{vec}(\mathbf{F}) \ , \tag{2.37}$$

which is equivalent to (2.34).

2.8.2 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

$$0 = \tilde{\mathbf{q_{2i}}}^{\mathbf{T}} \tilde{\mathbf{F}} \tilde{\mathbf{q_{1i}}} = \mathbf{q_{2i}^T T_2^T} \tilde{\mathbf{F}} \mathbf{T_1 q_{1i}} \ , \tag{2.38}$$

implying that the fundamental matrix sought is

$$\mathbf{F} = \mathbf{T_2^T} \tilde{\mathbf{F}} \mathbf{T_1} \ . \tag{2.39}$$

2.8.3 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

$$d_{Samp}(\mathbf{F}, \mathbf{q_{1i}}, \mathbf{q_{2i}}) = \frac{\left(\mathbf{q_{2i}^T F q_{1i}}\right)^2}{(\mathbf{F q_{1i}})_1^2 + (\mathbf{F q_{1i}})_2^2 + (\mathbf{q_{2i}^T F})_1^2 + (\mathbf{q_{2i}^T F})_2^2} \tag{2.40}$$

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 [Hartley and Zisserman, 2003].

2.9 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.

2.9.1 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, where

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

The linear constraint such a point pair poses on $\mathbf{H}$ is derived as follows, inserting into (2.13)

$$
\begin{aligned}
\mathbf{q_{1i}} &= \mathbf{H q_{2i}} \quad \Rightarrow \\
\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} &= [\mathbf{q_{1i}}]_\times \mathbf{H q_{2i}} \\
&= \begin{bmatrix} 0 & -1 & y_{1i} \\ 1 & 0 & -x_{1i} \\ -y_{1i} & x_{1i} & 0 \end{bmatrix} \begin{bmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ H_{31} & H_{32} & H_{33} \end{bmatrix} \begin{bmatrix} x_{2i} \\ y_{2i} \\ 1 \end{bmatrix} \\
&= \begin{bmatrix} 0 & -1 & y_{1i} \\ 1 & 0 & -x_{1i} \\ -y_{1i} & x_{1i} & 0 \end{bmatrix} \begin{bmatrix} H_{11}x_{2i} + H_{12}y_{2i} + H_{13} \\ H_{21}x_{2i} + H_{22}y_{2i} + H_{23} \\ H_{31}x_{2i} + H_{32}y_{2i} + H_{33} \end{bmatrix} \\
&= \begin{bmatrix} -H_{21}x_{2i} + H_{31}x_{2i}y_{1i} - H_{22}y_{2i} + H_{32}y_{2i}y_{1i} - H_{23} + H_{33}y_{1i} \\ H_{11}x_{2i} - H_{31}x_{2i}x_{1i} + H_{12}y_{2i} - H_{32}y_{2i}x_{1i} + H_{13} - H_{33}x_{1i} \\ -H_{11}x_{2i}y_{1i} + H_{21}x_{2i}x_{1i} - H_{12}y_{2i}y_{1i} + H_{22}y_{2i}x_{1i} - H_{13}y_{1i} + H_{23}x_{1i} \end{bmatrix} \\
&= \begin{bmatrix} 0 & -x_{2i} & x_{2i}y_{1i} & 0 & -y_{2i} & y_{2i}y_{1i} & 0 & -1 & y_{1i} \\ x_{2i} & 0 & -x_{2i}x_{1i} & y_{2i} & 0 & -y_{2i}x_{1i} & 1 & 0 & -x_{1i} \\ -x_{2i}y_{1i} & x_{2i}x_{1i} & 0 & -y_{2i}y_{1i} & y_{2i}x_{1i} & 0 & -y_{1i} & x_{1i} & 0 \end{bmatrix} \cdot \mathrm{vec}(\mathbf{H}) \\
&= \mathbf{b_i^T} \cdot \mathrm{vec}(\mathbf{H}) \ . \qquad (2.41)
\end{aligned}
$$

Where

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

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

$$
\min_{\mathrm{vec}(\mathbf{H})} ||\mathbf{B} \cdot \mathrm{vec}(\mathbf{H})||_2^2 \quad \text{where} \quad ||\mathrm{vec}(\mathbf{H})| = 1 \ . \qquad (2.42)
$$

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 (2.41), cf. Appendix A.7, in that

$$
\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} = [\mathbf{q_{1i}}]_\times \mathbf{H q_{2i}} = \left(\mathbf{q_{2i}^T} \otimes [\mathbf{q_{1i}}]_\times \right) \cdot \mathrm{vec}(\mathbf{H}) \ .
$$

The minimum number of point correspondences needed to estimate a homography is four. A homography has 9 parameters, subtracting one for over all scale, we need 8 constraints – i.e. $\mathbf{H}$ and $s \cdot \mathbf{H}$ represent the same transformation, for any non-zero scalar s. Considering (2.41), it is seen that even though $\mathbf{b_i^T}$ is a 3×9 it only/maximally has rank two, since $[\mathbf{q_{1i}}]_\times$ is a rank two matrix. Thus $8/2 = 4$ $\mathbf{b_i^T}$ are needed, equivalent to four point correspondences. Again more correspondences will reduce noise and the numerical stability of the solution can be tested as described in Section 2.4.1.

As for degeneracy of the problem, i.e. a unique solution existing for $\mathbf{H}$. In the minimal case of four point correspondences, the problem is degenerate if three of the four points are locate on a line, $\mathbf{l}$, in one or both of the point sets. If more point correspondences are added the problem is still degenerate if all the added points are still located on this line, $\mathbf{l}$.

2.9.2 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

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

The minimization problem that we wish to solve is thus

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

which is a minimization problem in the elements of $\mathbf{H}$ and the $\mathbf{s_{2i}}$ over all $i \in \{1, \cdots, n\}$. The function $\Pi(\cdot)$, transforms from homogeneous to non homogeneous coordinates, cf. (1.32). A good solution to minimizing the statistical meaningful error in the images – assuming Gaussian distribution of the ε – is therefor solving

$$\min_{\mathbf{H}, \mathbf{s_{2i}}} \sum_{i=1}^{n} \left(\|\Pi(\mathbf{H}\mathbf{s_{2i}}) - \Pi(\mathbf{p_{1i}})\|_2^2 + \|\Pi(\mathbf{s_{2i}}) - \Pi(\mathbf{p_{2i}})\|_2^2 \right) \ . \tag{2.43}$$

The corresponding error measure is thus

$$\text{dist}(\mathbf{H}, \mathbf{p_{1i}}, \mathbf{p_{2i}}) = \min_{\mathbf{s_{2i}}} \|\Pi(\mathbf{H}\mathbf{s_{2i}}) - \Pi(\mathbf{p_{1i}})\|_2^2 + \|\Pi(\mathbf{s_{2i}}) - \Pi(\mathbf{p_{2i}})\|_2^2 \ , \tag{2.44}$$

which is an optimization problem in $\mathbf{s_{2i}}$, and thus somewhat computationally expensive. Thus sometimes the following approximation is sometimes used instead

$$\text{dist}_{app}(\mathbf{H}, \mathbf{p_{1i}}, \mathbf{p_{2i}}) = \|\Pi(\mathbf{H}\mathbf{p_{2i}}) - \Pi(\mathbf{p_{1i}})\|_2^2 + \|\Pi(\mathbf{H^{-1}}\mathbf{p_{1i}}) - \Pi(\mathbf{p_{2i}})\|_2^2 \ . \tag{2.45}$$

2.9.3 An example

As an example of how (2.45) is used, consider the example from Figure 2.9, where the estimated homography is given by

$$\mathbf{H} = \begin{bmatrix} 1.1478 & -0.0259 & -1950.8 \\ 0.0665 & 1.0796 & 28.142 \\ 0.0 & 0.0 & 1 \end{bmatrix} \ ,$$

and a pair of loosely annotated points is given

$$\mathbf{p_1} = \begin{bmatrix} 1003.9 \\ 1710.4 \\ 1 \end{bmatrix} \quad \text{and} \quad \mathbf{p_2} = \begin{bmatrix} 2715.6 \\ 1564.8 \\ 1 \end{bmatrix} \ ,$$

then the first term of (2.45) is computed by[9]

$$\begin{aligned} \Pi\left(\mathbf{H}\mathbf{p_2}\right) &= \Pi\left(\begin{bmatrix} 1125.7 \\ 1898.4 \\ 1.1 \end{bmatrix} \right) = \begin{bmatrix} \frac{1125.7}{1.1} \\ \frac{1898.4}{1.1} \end{bmatrix} = \begin{bmatrix} 1011.9 \\ 1706.5 \end{bmatrix} \\ \Pi\left(\mathbf{p_1}\right) &= \Pi\left(\begin{bmatrix} 1003.9 \\ 1710.5 \\ 1 \end{bmatrix} \right) = \begin{bmatrix} \frac{1003.9}{1} \\ \frac{1710.5}{1} \end{bmatrix} = \begin{bmatrix} 1003.9 \\ 1710.5 \end{bmatrix} \\ \|\Pi(\mathbf{H}\mathbf{p_{2i}}) - \Pi(\mathbf{p_{1i}})\|_2^2 &= \left\| \begin{bmatrix} 1011.9 \\ 1706.5 \end{bmatrix} - \begin{bmatrix} 1003.9 \\ 1710.5 \end{bmatrix} \right\|_2^2 = \left\| \begin{bmatrix} 8.0 \\ -4.0 \end{bmatrix} \right\|_2^2 = 8.9443 \ , \end{aligned}$$

[9] This is real data, so rounding errors occur.

and the second term by

$$\begin{aligned}
\Pi(\mathbf{H^{-1}p_{1i}}) &= \Pi\left(\begin{bmatrix} 2434.9 \\ 1410.7 \\ 0.89928 \end{bmatrix}\right) = \begin{bmatrix} \frac{2434.9}{0.89928} \\ \frac{1410.7}{0.89928} \end{bmatrix} = \begin{bmatrix} 2707.6 \\ 1568.6 \end{bmatrix} \\
\Pi\left(\mathbf{p_2}\right) &= \Pi\left(\begin{bmatrix} 2715.6 \\ 1564.8 \\ 1 \end{bmatrix}\right) = \begin{bmatrix} \frac{2715.6}{1} \\ \frac{1564.8}{1} \end{bmatrix} = \begin{bmatrix} 2715.6 \\ 1564.8 \end{bmatrix} \\
\|\Pi(\mathbf{H^{-1}p_{1i}}) - \Pi(\mathbf{p_{2i}})\|_2^2 &= \left\| \begin{bmatrix} 2707.6 \\ 1568.6 \end{bmatrix} - \begin{bmatrix} 2715.6 \\ 1564.8 \end{bmatrix} \right\|_2^2 = \left\| \begin{bmatrix} -9 \\ 3.8 \end{bmatrix} \right\|_2^2 = 9.7693 \ .
\end{aligned}$$

Thus in this example, the result of using (2.45) is

$$\begin{aligned}
\text{dist}_{app}(\mathbf{H}, \mathbf{p_{1i}}, \mathbf{p_{2i}}) &= \|\Pi(\mathbf{Hp_{2i}}) - \Pi(\mathbf{p_{1i}})\|_2^2 + \|\Pi(\mathbf{H^{-1}p_{1i}}) - \Pi(\mathbf{p_{2i}})\|_2^2 \\
&= 8.9443 + 9.7693 = 18.7136 \ .
\end{aligned}$$

2.10 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. The readers attention should, however, be guided towards two additional issue, of great practical importance. 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, cf. e.g. [Kahl et al., 2008]. 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, a few references are [Haralick et al., 1994, Nister, 2004, Stewenius et al., 2008].

Chapter 3

Further Issues on View Geometry

In the previous two chapters, Chapter 1 and Chapter 2, the 'standard' introductory parts of view geometry have been covered. In many common practical applications of camera or view geometry there are, however, some additional tools that are needed if good performance is to be obtained. Some of these are somewhat covered in the seminal text book [Hartley and Zisserman, 2003], and some are not, e.g. because they are of a more recent date. Central parts of these subjects will be outlined here. These revolve around the use of the geometry of cameras with *known* internal parameters, i.e. known $\mathbf{A}$ in (1.21), i.e.

$$\mathbf{P} = \mathbf{A} \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} ,$$

and the solution of the bundle adjustment problem as e.g. formulated in (1.33) and (2.27), where it is referred to as non-linear optimization. Covering these issues in sufficient detail to get working solutions is beyond the scope of any realistic length course in computer vision and thus of this text. But since these methods are needed to get good solutions, these are outlined here, such that the reader is aware of their existence and basic workings, and should be able to follow the provided references if details of the methods are needed.

3.1 The Geometry of Calibrated versus Uncalibrated Cameras

An advantage of the uncalibrated pinhole camera model, as described in Chapter 1, is that it has as many degree of freedom, i.e. twelve, as there are in the 3×4 matrix which can represent a pinhole camera, c.f. Section 1.4.3. This implies that any 3×4 matrix, of rank three, can correspond to a pinhole camera. This further implies that standard linear algebra can be used to e.g. estimate a camera as done in Section 2.7, because no further constraints are needed.

The calibrated pinhole camera model, where the internal parameters (and possible radial distortion) are assumed known, only has six degrees of freedom – three for the rotation $\mathbf{R}$ and three for the translation $\mathbf{t}$. Furthermore, the parametrization of the rotation matrix is highly non-linear, c.f. Appendix A.3. This implies, that the estimation of camera geometry with calibrated cameras is often more complicated, in that the solutions have to be on a non-linear manifold. Often, the solutions to these problems are achieved by solving multivariate higher order polynomials[1], c.f. [Nister, 2004, Nister and Stewenius, 2007].

Despite the added complexity, the calibrated camera model is often preferred in practice, because it's reduced degrees of freedom is in essence an expression of the physical system being better modeled. This e.g. has the effect of reducing the possible effects of noise. Thus, many computer vision systems, e.g. feature tracking as described in Section 6.4 become more stable by using calibrated cameras. In some cases, this is so even if the camera calibration information is relatively uncertain. In [Snavely et al., 2008] they e.g. propose using the camera calibration from the EXIF[2] file tags, with good results.

The distinction between using calibrated and uncalibrated cameras becomes especially apparent when both camera and scene parameters are to be estimated. This is the subject of *structure from motion* also known as *visual odometry*, [Nister and Engels, 2006, Pollefeys, 2000, Snavely et al., 2008], c.f. Chapter 6 XXXXX. In

[1] The solution of multivariate higher order polynomials is the subject of algebraic geometry c.f. [Cox et al., 2005, Cox et al., 2007].

[2] Exchangeable image file format (EXIF) is a tag attached to many images taken by consumer cameras, containing various information about the image taken. This tag e.g. often includes the focal length.

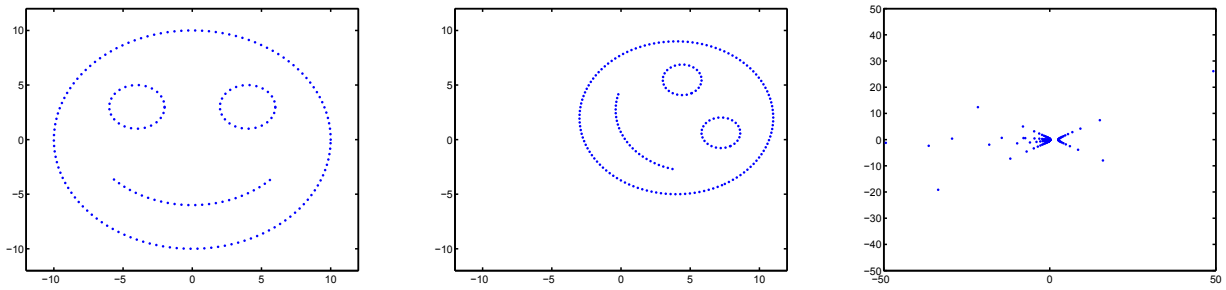


Figure 3.1: An illustration of the difference between the structure from motion ambiguities for calibrated, $\mathbf{H}_{cal}$, and uncalibrated, $\mathbf{H}$, cameras. **Left:** An original 2D (or planer 3D) point cloud. **Middle:** Transformation of the left image with all the possible elements of $\mathbf{H}_{cal}$, i.e. rotation, translation and scaling. **Right:** Transformation of the left image with a general $\mathbf{H}$.

this case, it is seen that for any full rank 4×4 matrix, $\mathbf{H}$, the uncalibrated pinhole camera model (1.21) can be manipulated in the following way

$$q_{ij} = \mathbf{P_i} Q_j = \mathbf{P_i} \mathbf{H}^{-1} \mathbf{H} Q_j = \tilde{\mathbf{P}}_\mathbf{i} \tilde{Q}_j \ , \tag{3.1}$$

where

$$\tilde{\mathbf{P}}_\mathbf{i} = \mathbf{P_i} \mathbf{H}^{-1} \quad \text{and} \quad \tilde{Q}_j = \mathbf{H} Q_j \ .$$

Since only the image points, q_{ij}, have been observed, there is nothing in the data to determine if $\mathbf{P_i}$ and Q_j or $\tilde{\mathbf{P}}_\mathbf{i}$ and $\tilde{Q}_j$ should be used. So in the uncalibrated case, it is seen, that $\tilde{\mathbf{P}}_\mathbf{i} = \mathbf{P_i} \mathbf{H}^{-1}$ is also a 3×4 rank three matrix, and the $\tilde{Q}_j$ still have a viable interpretation as homogeneous 3D points. Equation (3.1), thus, implies that a solution can only be estimated up to an arbitrary transformation of a full rank 4×4 matrix, $\mathbf{H}$, in the uncalibrated case. In other words, if a solution – consisting of the $\mathbf{P_i}$ and the Q_j – is computed in the uncalibrated case then we could just as well have computed a solution consisting of $\tilde{\mathbf{P}}_\mathbf{i}$ and $\tilde{Q}_j$, unless more information is available, c.f. [Hartley and Zisserman, 2003].

In the calibrated case, (3.1) in principle still holds, but if the $\mathbf{P_i}$ and the $\tilde{\mathbf{P}}_\mathbf{i}$ should both be consistent with specific internal camera calibration $\mathbf{A}_i$ then this constrains $\mathbf{H}$ to be of the form [Hartley and Zisserman, 2003]

$$\mathbf{H}_{cal} = \begin{bmatrix} \tilde{\mathbf{s}} \cdot \tilde{\mathbf{R}} & \tilde{\mathbf{t}} \\ 0\,0\,0 & 1 \end{bmatrix} \ ,$$

where $\tilde{s}$ is an arbitrary scale, $\tilde{\mathbf{R}}$ an arbitrary rotation and $\tilde{\mathbf{t}}$ an arbitrary translation. Thus, $\mathbf{H}_{cal}$ is reduced to being an arbitrary definition of the global coordinate system and an arbitrary scaling. The difference between a general $\mathbf{H}$ and a $\mathbf{H}_{cal}$ originating from a calibrated camera can be very significant, as seen in Figure 3.1. This Figure 3.1 hopefully gives an intuitive example of why a calibrated camera in general gives more stable solutions.

3.2 Estimating the Essential Matrix

A main practical difference between using calibrated or uncalibrated cameras, is if the epipolar geometry is represented by the fundamental matrix 2.9 or the essential matrix 2.7, i.e.

$$\mathbf{E} = [\mathbf{t}]_\times \mathbf{R} \ . \tag{3.2}$$

Often, this essential matrix, $\mathbf{E}$, needs to be estimated from point correspondences, in order to do feature tracking well, as described in Chapter 6. This is the subject of this section.

3.2.1 Additional Properties of the Essential Matrix

As outlined in Section 2.2, the essential matrix has 5 degrees of freedom(dof) and not 7 dof as the fundamental matrix. This implies that not all fundamental matrices, estimated as outlined in Section 2.8, can be interpreted as an essential matrix. Therefor, and due to efficiency reasons, a five point algorithm is often needed.

The extra constraints on the essential matrix comes from the fact that, it's singular- or eigenvalues have the form

$$\{\sigma, \sigma, 0\} \quad , \tag{3.3}$$

i.e. that it has a singular value of zero and the two others singular values are equal. To see this, note the the rotation, $\mathbf{R}$, in (3.2), does not affect the singular values of $\mathbf{E}$. Secondly, note that $[\mathbf{t}]_\times$ represents a cross product with $\mathbf{t}$, which has the singular value structure of (3.3). The reasoning being, that the the length of a cross product is proportional to the angle between the vectors, times the length of the two vectors. Thus, $[\mathbf{t}]_\times$ must have a null space in the direction of $\mathbf{t}$, and the two σ being equal to the length of $\mathbf{t}$.

It can be shown, c.f. [Nister, 2004], that a matrix $\mathbf{E}$ having the singular value structure (3.3) is equivalent to it fulfilling

$$\mathbf{E}\mathbf{E}^\mathbf{T}\mathbf{E} - \frac{1}{2}\text{trace}\left(\mathbf{E}\mathbf{E}^\mathbf{T}\right)\mathbf{E} = 0 \quad . \tag{3.4}$$

To see that a matrix with singular values equal to (3.3), and thus has the singular value decomposition(SVD) of

$$\mathbf{E} = \mathbf{U}\begin{bmatrix} \sigma & 0 & 0 \\ 0 & \sigma & 0 \\ 0 & 0 & 0 \end{bmatrix}\mathbf{V}^\mathbf{T} \quad , \tag{3.5}$$

has five dof, note that it can be specified by it left null space (rightmost column of $\mathbf{U}$) it's right null space (rightmost column of $\mathbf{V}$) and sigma. Since the matrices $\mathbf{U}$ and $\mathbf{V}$ have to be unitarian, i.e. the null space is a direction in 3D space, these null spaces have 2 dof. Thus there is 2+2+1=5 dof.

3.2.2 Algorithm Outline

The typical approach of a 5 point algorithms for estimating the essential matrix, cf. e.g. [Li and Hartley, 2006, Nister, 2004, Stewénius et al., 2006], starts by using five point correspondences to find linear constraints on the essential matrix, as done in (2.34). This gives a system of equations,

$$\mathbf{B} \cdot \text{vec}\left(\mathbf{E}\right) = 0$$

as in Section 2.8, but which has a 4(=9-5) dimensional solution space. That is, that a solution, $\mathbf{E}$, can be written as

$$\mathbf{E} = x\mathbf{X} + y\mathbf{Y} + z\mathbf{Z} + w\mathbf{W} \quad , \tag{3.6}$$

where $\{x, y, z, w\}$ are free scalar parameters and $\{\mathbf{X}, \mathbf{Y}, \mathbf{Z}, \mathbf{W}\}$ are 3×3 matrices spanning the null space of $\mathbf{B}$. To estimate $\{x, y, z, w\}$ and thus finding an estimate of the essential matrix, $\mathbf{E}$, (3.6) is inserted into (3.4). It should be noted, that the solution is only observable up to scale, and as such w can e.g. be equal to one. Inserting (3.6) into (3.4) gives a tenth degree polynomial in three or four variables. Good solutions for solving this are found in [Li and Hartley, 2006, Nister, 2004, Stewénius et al., 2006], but is beyond this text.

For an example of essential matrix estimation please refer to the end of Chapter 6 XXXX:(Insert section number)

3.2.3 Extracting Relative Orientation from the Essential Matrix

Another important property of the essential matrix, which will also be presented here without a derivation, is that the relative orientation of the two cameras can (almost) be extracted from the essential matrix. That is, given an essential matrix, the rotation and translation (direction not length) between the two cameras can be derived. Or more precisely, there are only four possible solutions, given the translation has fixed/unit length. These can be disambiguated by requiring that points should be in front of the camera. This in turn implies, that if we can estimate an essential matrix from point correspondences, we can also estimate the relative orientation between the cameras. This in turn implies, that we can start doing point triangulation of the point pairs.

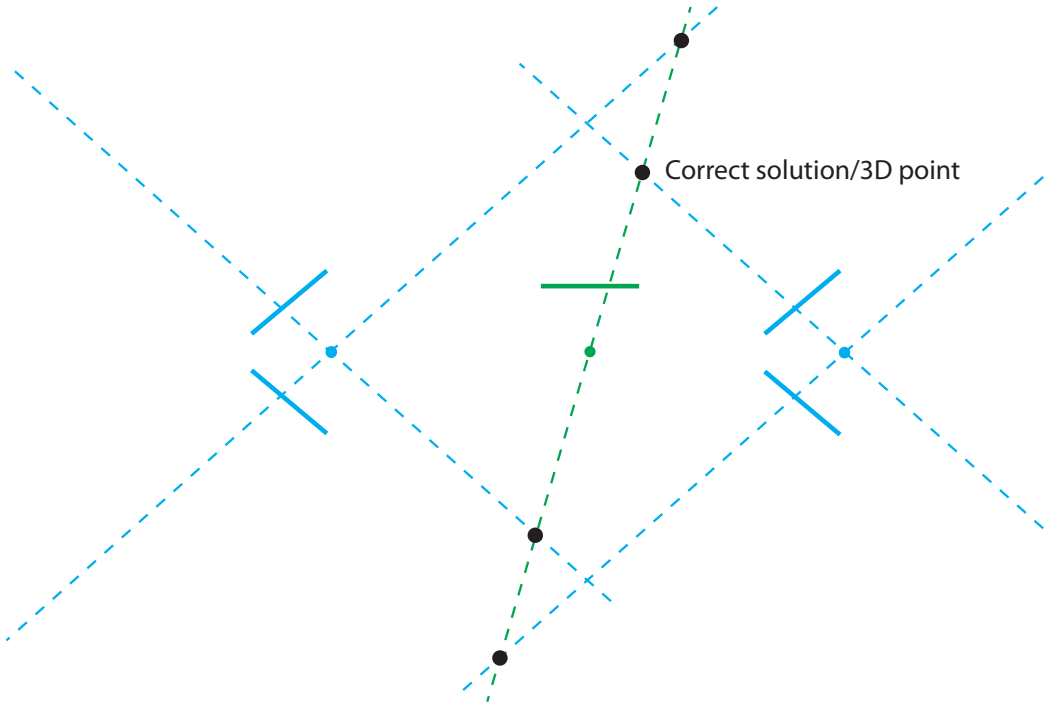


Figure 3.2: An abstract illustration of the four possible camera configurations for a given essential matrix $\mathbf{E}$, where the first camera $\mathbf{P_1}$ (in green) is kept fixed. The four possible solutions for camera two, $\mathbf{P_2}$, are depicted as blue camera centers and image plane. For a given point correspondence, denoted by the dashed lines (a green and a blue for each of the four possible camera configurations) it is possible to estimate a 3D point, denote by black dots, for each camera configuration. It is noted, that only one of these four 3D points is located *in front* of both cameras. This fact can be used to disambiguate (i.e. chose the right) the camera configuration. The four solutions are shown separately in Figure 3.3

For a derivation and proof of this result the interested reader is referred to [Hartley and Zisserman, 2003]. It states that if the SVD of the essential matrix, $\mathbf{E}$ is given by (3.5), and the first camera is given by

$$\mathbf{P_1} = \mathbf{A} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} ,$$

Then the second camera can be written as

$$\mathbf{P_2} = \mathbf{A} \begin{bmatrix} \mathbf{R_E} & \mathbf{t_E} \end{bmatrix} ,$$

where

$$\begin{aligned} \mathbf{R_E} &= \mathbf{U}\mathbf{D}\mathbf{V}^{\mathbf{T}} \quad \text{or} \quad \mathbf{R_E} = \mathbf{U}\mathbf{D}^{\mathbf{T}}\mathbf{V}^{\mathbf{T}} \quad \text{where} \\ \mathbf{D} &= \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ \mathbf{t_E} &= \mathbf{U} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad \text{or} \quad \mathbf{t_E} = -\mathbf{U} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} . \end{aligned} \tag{3.7}$$

That is two possibilities for $\mathbf{R_E}$ and two for $\mathbf{t_E}$, i.e. four($= 2 \cdot 2$) possibilities in all. Since $\mathbf{E}$ is defined up to a scale, the length of $\mathbf{t_E}$ is unobservable from $\mathbf{E}$, and and the solution of $\|\mathbf{t_E}\|_2 = 1$ is typically chosen. This corresponds to setting $\sigma = 1$ in (3.5). These four solutions are illustrated in Figure 3.2 and Figure 3.3, for a hypothetical essential matrix $\mathbf{E}$.

As demonstrated in Figure 3.2, a way of finding the correct relative orientation between the cameras, can be done via a point correspondence, c.f. e.g Chapter 6, by

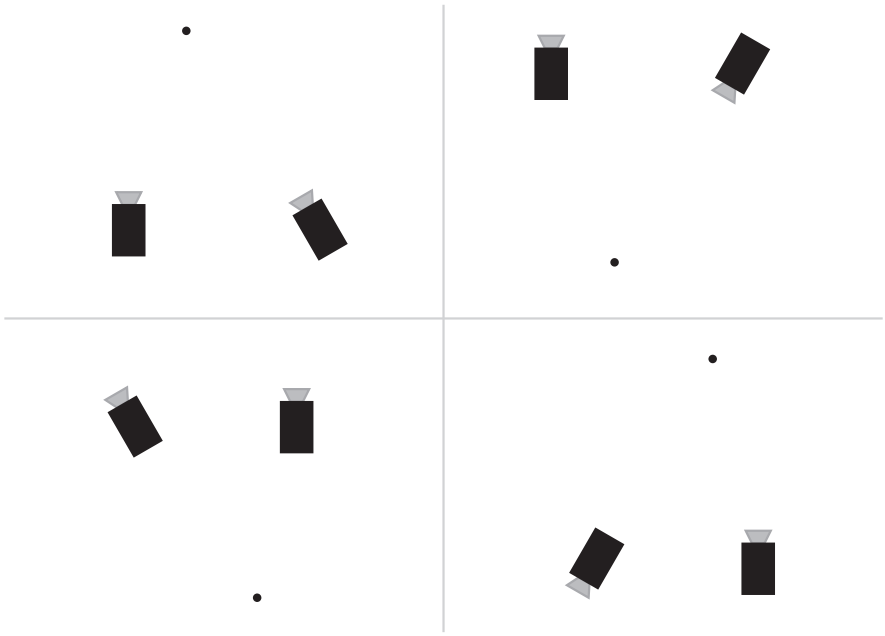


Figure 3.3: The four solutions from Figure 3.2, shown in a less abstract way, and separated out.

1. Estimate the 3D point, Q_i associated with that point correspondence, e.g. via the method of Section 2.4, for each of the for possible camera configurations.

2. Determine if that point is in front of the camera. This can be done by transforming the 3D point Q, into the coordinate system of the camera, and checking that the z-value is positive. This test can be performed by[3]
$$\begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} Q > 0 \ ,$$
where the camera is given by $\mathbf{P} = \mathbf{A}[\mathbf{R}\ \mathbf{t}]$.

Often it is expected, that some point correspondences are erroneous, and as such many points correspondences are often used, e.g. by finding the point camera configuration where most point correspondences are consistent with.

As an end note to this section it is noted, that the solution from Section 3.2.1, can be refined, by extracting the camera pairs from the essential matrix, and use this as input to a bundle adjustment routine, c.f. Section 3.4. Here the initial estimates of the 3D points are the ones used for disambiguating the relative orientation.

3.3 Calibrated Camera Resectioning

As with the epipolar geometry, camera resectioning, c.f. Section 2.7, becomes more complicated if the knowledge of the camera calibration, $\mathbf{A}$, is to be incorporated into the solution. Likewise, this is also done, in that it gives more robust algorithms. This is a well studied problem c.f. e.g [Ansar and Daniilidis, 2003, Fischler and Bolles, 1981, Gao et al., 2003, Haralick et al., 1994, Kneip et al., 2011].

I contrast to the resectioning of uncalibrated cameras, c.f. Section 2.7, only three 3D to 2D point correspondences, are needed as illustrated in Figure 3.4, which gives a maximum of four solutions, so often a fourth (or more) 3D to 2D correspondence is used for disambiguation.

The nature of the constraints from 3d to 2D point correspondences for calibrated cameraes, is illustrated in Figure 3.4. Denote the corresponding 3D to 2D pair by Q_i and q_i respectively, then the ray from the camera center C to the 3D point
$$x_i = \frac{Q_i - C}{\|Q_i - C\|} \ ,$$

[3] Some time the structure of $\mathbf{A}$ is such that points in front of the camera have a *negative* z-value, in which case $>$ should be changed to $<$. This structure of A in question is associated to which direction the cross product of the image coordinate axis point.

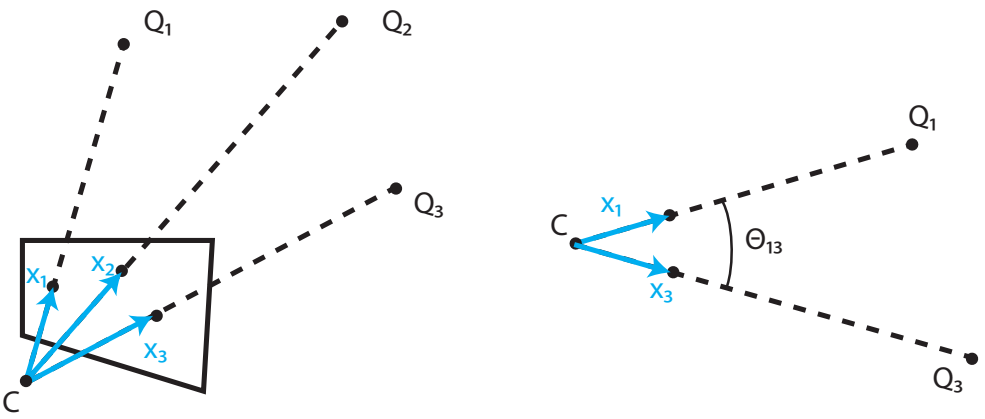


Figure 3.4: The nature of the constraints for calibrated camera resectioning are that the 3d to 2d point correspondences define rays, x_i, and that the angle between any ray pair, Θ_{ij}, can thus be computed.

can be found via

$$x_i = \frac{\mathbf{A}^{-1} q_i}{\|\mathbf{A}^{-1} q_i\|} \ ,$$

as seen in Figure 3.4-Left. This gives the angle between any two of these rays, c.f. Figure 3.4-Right, according to the following formula:

$$\Theta_{ij} = \text{acos}(x_i^T x_j) \ .$$

Consider a plane going through two of the 3D points, i.e. Q_i and Q_j. Then by using the inscribed angle theorem, as seen in Figure 3.5, it can be shown that the camera center C must be located on a circle in that plane for any given Θ_{ij}. This implies, that for given Q_i, Q_j and Θ_{ij} the camera center is constrained to lying on a surface defined by rotating this circle around the line through Q_i and Q_j, as depicted in Figure 3.5. This surface can be defined by a higher order polynomial in several variables.

For three 3D to 2D point correspondences, three 3D point pairs exist, and as such three such polynomial surfaces as describe directly above. The intersection of these surfaces is thus the possible locations of the camera center, C. The solution to which is can be provided by algebraic geometry [Cox et al., 2007]. Given the camera center, the rotation, $\mathbf{R}$, that maps the x_i in camera coordinates onto the x_i in global coordinates is 'easily' found via linear algebra.

For an example of calibrated camera resectioning please refer to the end of Chapter 6 XXXX:(Insert section number). It is noted, that calibrated camera resectioning from n 3d to 2d correspondences is also referred to as the perspective n point (PnP) problem, i.e. for three point correspondences it is called the P3P problem. Note also, that a solution to the P4P, involving only linear algebra, and giving a unique solution is presented in [Ansar and Daniilidis, 2003].

3.4 Bundle Adjustment

The term *bundle adjustment* [Triggs et al., 2000] refers to adjusting the bundles of light meeting at the camera center or focal point, and is in it's basic form associated with solving the problem

$$\min_{\mathbf{P_i}, Q_j} \sum_{ij} \left\| \begin{bmatrix} x_{ij} \\ y_{ij} \end{bmatrix} - \Pi(\mathbf{P_i} Q_j) \right\|_2^2 \ , \tag{3.8}$$

where (x_{ij}, y_{ij}) denote the observed 2D image points and $\Pi(\cdot)$ denotes the transformation between homogeneous and inhomogeneous coordinates, c.f. (1.32). Here, and elsewhere, bundle adjustment is also associated with solving or optimizing (3.8), where parts of the solution is known as in (1.33) and (2.27). Another common deviation from (3.8), is that a robust norm, instead of the 2-norm is used c.f. Section 5.4[4].

[4] A robust norm is often needed to have a practical work system, as elaborated upon in e.g. Section 5.4.

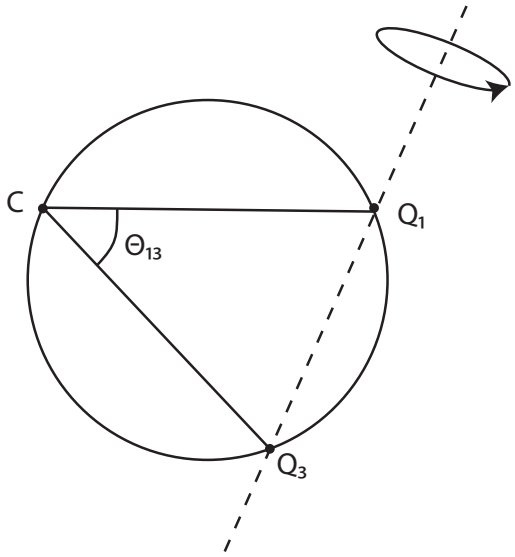


Figure 3.5: By use of the inscribed angle theorem, it is seen that given Q_i, Q_j and Θ_{ij} constrains the camera center C to lye on a specific circle in any plane through Q_i and Q_j.

A standard approach to solving the bundle adjustment problem (3.8), which will be presented here, is to cast it as a non-linear optimization problem, and solve it via the Levenberg Marquardt algorithm [Levenberg, 1944, Marquardt, 1963, Nocedal and Wright, 2000]. To do this the solution firstly has to be represented as a parameter vector, φ, which is done in a manner akin to the vectorization , vec($\mathbf{P}$), in (2.31).

Consider e.g. a toy example[5] with two cameras, $\mathbf{P_1}$ and $\mathbf{P_2}$, and three 3D points, Q_A, Q_B and Q_C. Associate the two cameras with the parameters

$$\varphi_1 = \text{vec}(\mathbf{P_1}) \quad \text{and} \quad \varphi_2 = \text{vec}(\mathbf{P_2}) \ ,$$

and with each of the three 3D points

$$\varphi_A = \begin{bmatrix} X_A \\ Y_A \\ Z_A \end{bmatrix} \quad , \quad \varphi_B = \begin{bmatrix} X_B \\ Y_B \\ Z_B \end{bmatrix} \quad \text{and} \quad \varphi_C = \begin{bmatrix} X_C \\ Y_C \\ Z_C \end{bmatrix} \ ,$$

for Q_A, Q_B and Q_C respectively. Then the following parameter vector, φ, defines the solution to the problem

$$\varphi = \begin{bmatrix} \varphi_1 \\ \varphi_2 \\ \varphi_A \\ \varphi_B \\ \varphi_C \end{bmatrix} \ , \tag{3.9}$$

where it is noted that all the φ_1, φ_2, φ_A, φ_B and φ_C are (column) vectors.

Secondly, to use Levenberg Marquardt, the objective function, i.e. (3.8), has to be represented as a sum of squares, i.e. $f(\varphi)^T f(\varphi)$ where $f(\varphi)$ is the error vector as a function of the parameter vector φ. When it is convenient, we for ease of notation simply write

$$f^T f \ . \tag{3.10}$$

When using the 2-norm, (3.8) is practically on this form[6], in that for a given camera $\mathbf{P_i}$ and 3D point Q_j the inner part of (3.8) becomes (where the rows of the camera matrix are denoted by a super script as in

[5] This example is so small that there are more free parameters than constraints and as such the solution is not well defined. It is chosen this way to keep the explanation simple.

[6] If robust norms are used, the mapping between ρ and the 2-norm has to be computed.

Section 2.4.1)

$$\left\| \begin{bmatrix} x_{ij} \\ y_{ij} \end{bmatrix} - \Pi(\mathbf{P}Q_i) \right\|_2^2 = \left\| \begin{bmatrix} x_{ij} \\ y_{ij} \end{bmatrix} - \begin{bmatrix} \frac{P_i^1 Q_j}{P_i^3 Q_j} \\ \frac{P_i^2 Q_j}{P_i^3 Q_j} \end{bmatrix} \right\|_2^2 =$$

$$\left\| \begin{bmatrix} x_{ij} - \frac{P_i^1 Q_j}{P_i^3 Q_j} \\ y_{ij} - \frac{P_i^2 Q_j}{P_i^3 Q_j} \end{bmatrix} \right\|_2^2 = \begin{bmatrix} x_{ij} - \frac{P_i^1 Q_j}{P_i^3 Q_j} \\ y_{ij} - \frac{P_i^2 Q_j}{P_i^3 Q_j} \end{bmatrix}^T \begin{bmatrix} x_{ij} - \frac{P_i^1 Q_j}{P_i^3 Q_j} \\ y_{ij} - \frac{P_i^2 Q_j}{P_i^3 Q_j} \end{bmatrix} .$$

Thus, any given 2D image observation (x_{ij}, y_{ij}), gives two elements of f corresponding to

$$\begin{bmatrix} x_{ij} - \frac{P_i^1 Q_j}{P_i^3 Q_j} \\ y_{ij} - \frac{P_i^2 Q_j}{P_i^3 Q_j} \end{bmatrix} . \tag{3.11}$$

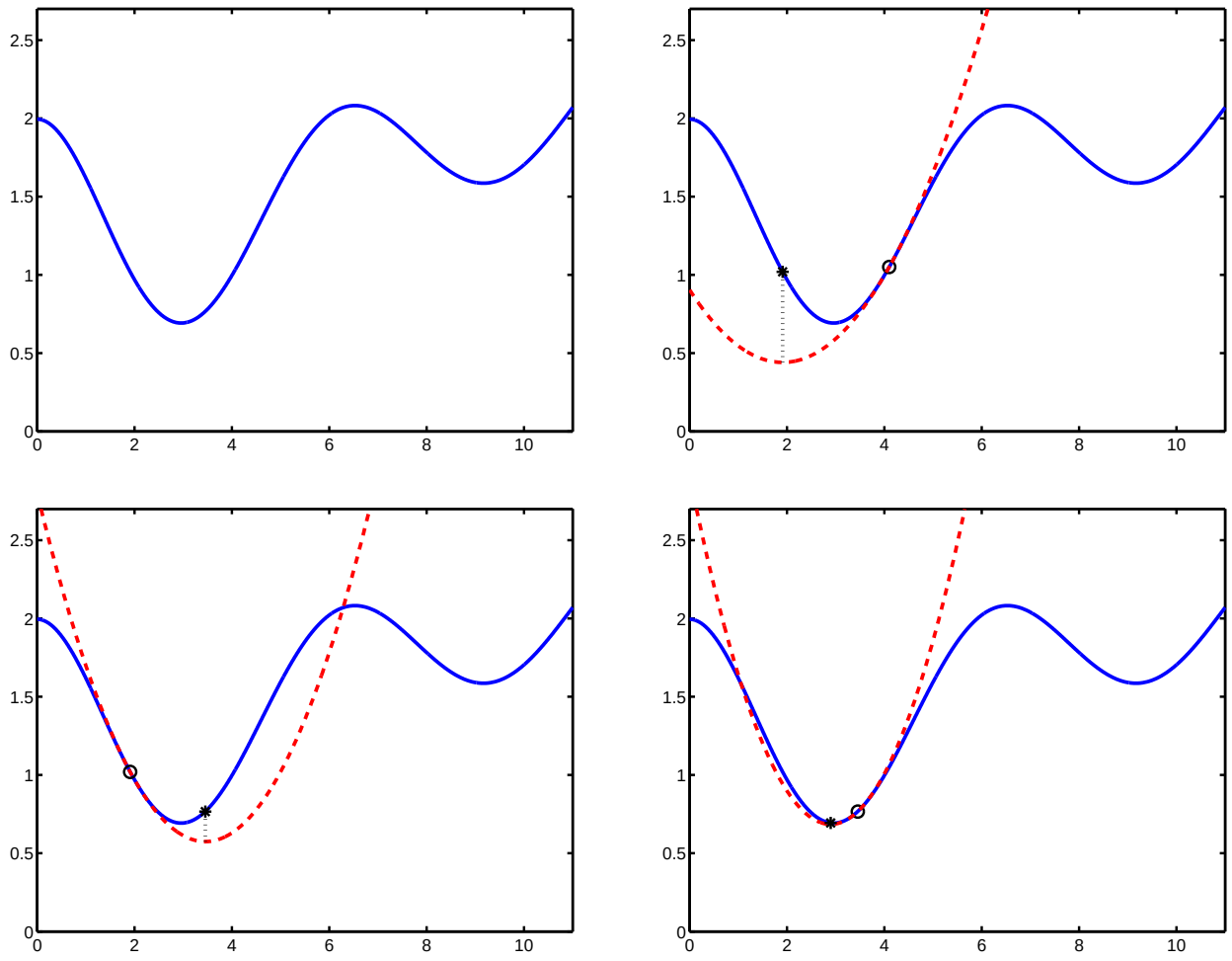


Figure 3.6: Three iterations of the Newton method. **Upper Left:** The function to be minimized. **Upper Right:** The first iteration. Starting from an initial estimate of $\varphi^0 = 4.1$ (round point) the red dashed second order polynomial is fitted, which has a minimum at 1.9 (start shaped point). **Lower Left:** Second iteration. Starting from $\varphi^1 = 1.9$, the fitted polynomial has a minimum at 3.45. **Lower Right:** The third iteration starting at $\varphi^2 = 3.45$ and finishing with $\varphi^3 = 2.9$.

3.4.1 The Levenberg-Marquardt Algorithm

The *Levenberg-Marquardt algorithm*, is an adaptation of the Newton method – also known as the Newton Raphson method – for finding extremal points[7] of univariate functions, or zero crossings of it's gradient. This is a property it shares with many other methods for solving non-linear least squares problems, i.e. minimization problems of the form of (3.10).

The Newton method is an iterative algorithm, starting from an initial guess φ^0, and then hopefully iterating closer to the φ that minimizes the given function (3.10). As shown in Figure 3.6, this is done by locally approximating the function by a second order polynomial, and then finding the minimum of that. This minimum of the second order polynomial is then the next φ in the iteration sequence.

The *Gauss-Newton method* is an extension of the Newton method to the multivariate non-linear least squares case, i.e. φ is a vector and not a scalar. This is done by approximating f at iteration k by

$$f(\varphi^k + \delta) \approx f(\varphi^k) + \mathbf{J}\delta \ ,$$

where $\mathbf{J}$ is the jacobian of f at φ^k, i.e. element l, m of $\mathbf{J}$ is given by

$$\mathbf{J}_{lm} = \frac{\partial f_l}{\partial \varphi_m^k} \ .$$

Then (3.10) is approximated by

$$\begin{aligned} f^T(\varphi^k + \delta) f(\varphi^k + \delta) &\approx \Big(f(\varphi^k) + \mathbf{J}\delta\Big)^T \Big(f(\varphi^k) + \mathbf{J}\delta\Big) \\ &= f^T(\varphi^k) f(\varphi^k) + 2\delta^T \mathbf{J}^T f(\varphi^k) + \delta^T \mathbf{J}^T \mathbf{J}\delta \ , \end{aligned} \tag{3.12}$$

which is a second order polynomial approximation to (3.10) at φ^k. The extremum point of (3.12) is found by taking its derivative wrt. δ and setting it to zero, i.e.

$$2\mathbf{J}^T f(\varphi^k) + 2\mathbf{J}^T \mathbf{J}\delta = 0 \quad \Rightarrow \quad \mathbf{J}^T \mathbf{J}\delta = -\mathbf{J}^T f(\varphi^k) \ . \tag{3.13}$$

The new iteration φ^{k+1} is then found as

$$\varphi^{k+1} = \varphi^k + \delta^k \ ,$$

where δ^k is the solution to (3.13).

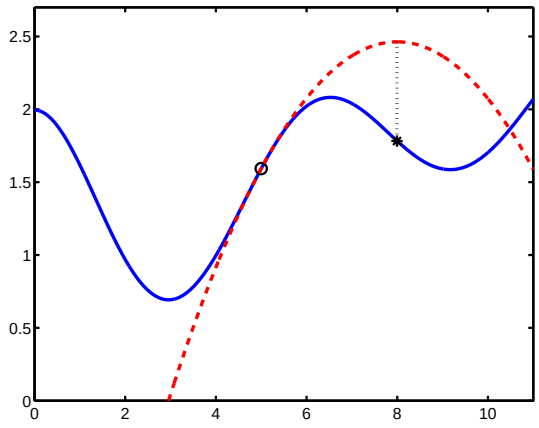


Figure 3.7: An illustration of what happens when the second order approximation does not fit well in the Newton method. Here $\varphi^0 = 5$.

The Newton and Gauss-Newton methods work fine if φ^0 is close enough to the minimum we are trying to find, as seen in Figure 3.6. If, however, this is not the case, both methods will *not* work very well as illustrated

[7] Local minima or maxima.

Given a function $f(\varphi)$, an initial guess φ^0 minimize

$$f^T f \ .$$

Set $\lambda = 1$, *Stop*=false and k=0.

1. Solve (3.14) for δ, i.e.
$$\left(\mathbf{J}^T\mathbf{J} + \lambda\mathbf{I}\right)\delta = -\mathbf{J}^T f(\varphi^k) \ .$$
2. if $f(\varphi^k + \delta) < f(\varphi^k)$
   - $\varphi^{k+1} = \varphi^k + \delta$.
   - *Stop*= $\frac{\|f(\varphi^k+\delta)-f(\varphi^k)\|_2^2}{\|f(\varphi^k+\delta)+f(\varphi^k)\|_2^2} < 1e^{-6}$.
   - $\lambda = \frac{1}{2}\lambda$.
   - $k = k + 1$.
3. else
   - $\lambda = 3\lambda$
4. If not *stop* go to 1.
5. Estimated solution to $f^T f$ is φ^k.

Table 3.1: Outline of the Levenberg-Marquardt algorithm.

in Figure 3.7. An underlying reason for this is that the second order approximation often only fits well near the minimum/extreamum. To address this issue, the Levenberg-Marquardt algorithm introduces a trust parameter λ, which can be incorporated into (3.13) as follows

$$\left(\mathbf{J}^T\mathbf{J} + \lambda\mathbf{I}\right)\delta = -\mathbf{J}^T f(\varphi^k) \ . \tag{3.14}$$

If there is allot of trust in (3.13) then λ is small and (3.14) approximates (3.13) well. If the trust is low then λ is large and the effect of the second order term is dampened. If λ is large enough (3.14) is approximated well by:

$$\delta = \frac{-1}{\lambda}\mathbf{J}^T f(\varphi^k) \ ,$$

which is seen to be a small step in the downward gradient direction, i.e.

$$\frac{\partial}{\partial\varphi^k} f^T(\varphi^k) f(\varphi^k) = 2\mathbf{J}^T f(\varphi^k) \ .$$

As such if λ is large enough $f(\varphi^{k+1}) \leq f(\varphi^k)$.

There are various proposed update rules for the trust parameter, λ, found in the literature, but common for them all is that an update step, i.e. $\varphi^{k+1} = \varphi^k + \delta^k$, is only performed if $f(\varphi^{k+1}) < f(\varphi^k)$. If not λ is increased and a new δ^k is computed until an update step will eventually happen in that λ becomes large enough. A simple version of the Levenberg-Marquardt algorithm is given in Table 3.1, although a more industrial strength version is recommended for real applications, c.f. e.g. [Nocedal and Wright, 2000].

In order to apply the Levenberg-Marquardt algorithm to the bundle adjustment problem the f and $\mathbf{J}$ have to be computed for a given φ. The computation of f is outlined in (3.11), and the computation of $\mathbf{J}$ is the derivative there of wrt. φ.

3.4.2 Solving $\mathbf{A}x = b$

In order to solve (3.14), a linear system of equations $\mathbf{A}x = b$ has to be solved where $\mathbf{A} = \mathbf{J}^\mathbf{T}\mathbf{J} + \lambda\mathbf{I}$. The computational complexity of this task is given by the dimensions of $\mathbf{A}$, and, thus, the number of parameters

in φ. The length of φ can easily be in the thousands for medium sized problems, e.g. 100 cameras with six parameters each and 3000 3D points with three parameters each, would give $100 \cdot 6 + 3000 \cdot 3 = 9600$. Solving such large linear systems without any special structure will cause possible computational and memory problems.

The $\mathbf{A}$ in the bundle adjustment problem, luckily, has a lot of structure that allows for faster computations. To see this, note each f_m is associated with a 2D image observation in a given image, therefore $\mathbf{J}_{ml}$ is zero if

- φ_l is associated with a camera, and f_m is associated with a 2D observation from a different camera.
- φ_l is associated with a 3D point, and f_m is associated with a 2D observation of another 3D point.

This implies, that $\mathbf{J}$ is very sparse, i.e. that most of its entries are zero. Now considering $\mathbf{J}^\mathbf{T}\mathbf{J}$, and thus $\mathbf{A}$ minus λ in the diagonal, it is seen that element (m, n) is associated with φ_m and φ_n. Elements (m, n) of $\mathbf{J}^\mathbf{T}\mathbf{J}$ are, thus, non-zero only if φ_m and φ_n have have non-zero 2D observations in common. As a consequence, elements (m, n) of $\mathbf{J}^\mathbf{T}\mathbf{J}$ are zero if

- φ_m and φ_n are associated with two different cameras, since two cameras do not have joint 2D measurements.
- φ_m and φ_n are associated with two different 3D points, since two 3D points do not have joint 2D measurements.
- φ_m is associated with a camera, and φ_n is associated with a 3D point not depicted/observed in that camera. The revers situation where φ_m is associated with a 3D point unseen by a camera associated with φ_n.

This implies that $\mathbf{J}^\mathbf{T}\mathbf{J}$ and thereby also $\mathbf{A}$ are very sparse. Popular ways of using this sparsity to solve $\mathbf{A}x = b$ is the Schur compliment, c.f. [Triggs et al., 2000], CGLS [Björck, 1996] and the LSQR [Paige and Saunders, 1982].

3.4.3 Gauge Freedoms

Another issue with solving the linear system $\mathbf{A}x = b$ is that $\mathbf{A}$ should be well conditioned and at least have full rank. For large enough λ the latter is not an issue, but for small λ it might be. Since the linear system $\mathbf{A}x = b$ (with x denoting δ) is a linearization of how change in x or δ improve the fit (3.8), a possibility of changing δ is such a way that the fit does not change, implies that $\mathbf{A}$ will be rank deficient – or possibly ill-conditioned for large enough λ. This will again give numerical problems.

A typical example of this is if four parameters (i.e. homogeneous) are used to denote a 3D point. In this case four parameters are used to describe a three degree of freedom(dof) phenomena, and by multiplying this homogeneous coordinate by a non-zero scale, the underlying solution and thus the fit stays the same. This implies that $\mathbf{A}$ will be rank deficient or at best ill-conditioned.

These type of over parameterizations of the solution is referred to as gauge freedoms [Meissl, 1982, Triggs et al., 2000] and should be avoided (by formulating the mathematical parameterizations of the problem well) or by doing numerial regularization of the solution. There is a theoretical link between this and the More-Penrose pseudo inverse [Meissl, 1982].

Another typical origin of gauge freedoms, is if no fixed measurements or datum exist, then e.g. the scale rotation and translation of a solution is undefined. In this cases an arbitrary scale, rotation and translation is often imposed on the solution to avoid the numerical issues associated with the gauge freedoms.

Part II

Image Features and Matching

Chapter 4

Extracting Features

Computer vision is mainly concerned with doing inference from images. One of the most typical strategies for doing this is by extracting features from images, which are believed to have some intrinsic meaning. The most common such features are points and lines, which are covered in this chapter, but higher order features such as e.g. circles – covered in Chapter 5 – and faces are also used.

One of the main used of feature is as one of several strategies for finding the correspondence between images, i.e. finding the 'same thing' in two or more images. This correspondence problem is the underlying theme of this part, where this Chapter 4 describes how to extract features, the next, Chapter 5, describes how the robust statistics needed for a reliable estimation and finally Chapter 6 describes how to estimate the correspondence between point features. Where point feature are the predominant features used when addressing the correspondence problem via feature matching. A good reference for feature extraction and correspondence estimation is [Tuytelaars and Mikolajczyk, 2008], recommended for readers seeking more information in this area.

The organization of this chapter is as follows; first, Section 4.1 and Section 4.2, present some general considerations in relation to feature extraction, upon which two point detection and a line detection scheme is presented in Sections 4.3, 4.4 and 4.5 respectively. This, naturally, does not cover the plethora of feature or even point based methods, and since the publication of [Tuytelaars and Mikolajczyk, 2008] much research focus has been on computationally fast methods. For an overview and evaluation of these, the reader is refereed to [Heinly et al., 2012].

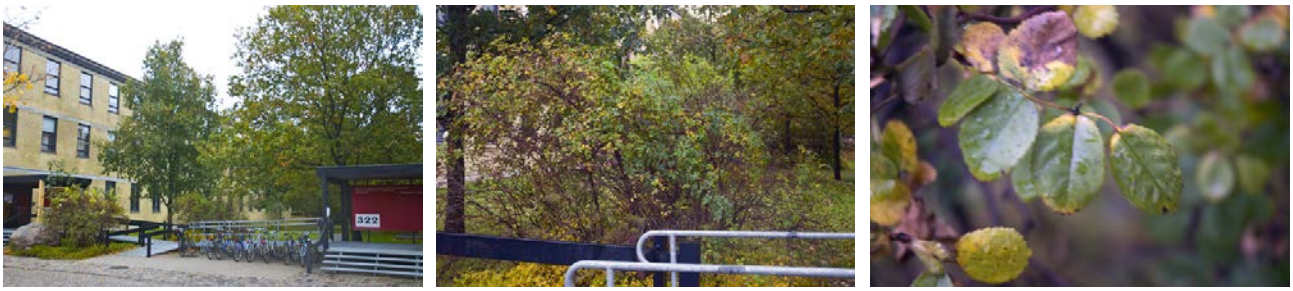


Figure 4.1: Things appear at a scale, as illustrated in the above images. This scale can also vary, as e.g. the leaves of the bush above. When extracting features, we often have to choose which scale we want to work at. This is naturally related to what we want to extract, e.g. the leaves, the branches or the trees.

4.1 Importance of Scale

The things or features that we want to extract from an image appear at a scale, cf. Figure 4.1. 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 4.2, where different scales extract different structures.

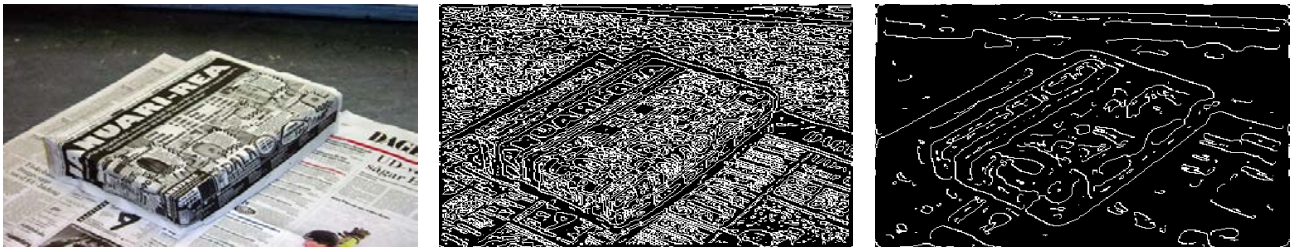


Figure 4.2: The maximum of the gradient magnitude is computed for the leftmost image and depicted in the middle and right images. The difference between the middle and right image is the scale at which the gradient is computed. Notice how different structures appear at the different scales.

To illustrate the need for choosing *image scale* in more detail, consider the one dimensional signal in Figure 4.3. When detecting edges on this signal using an *image 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.

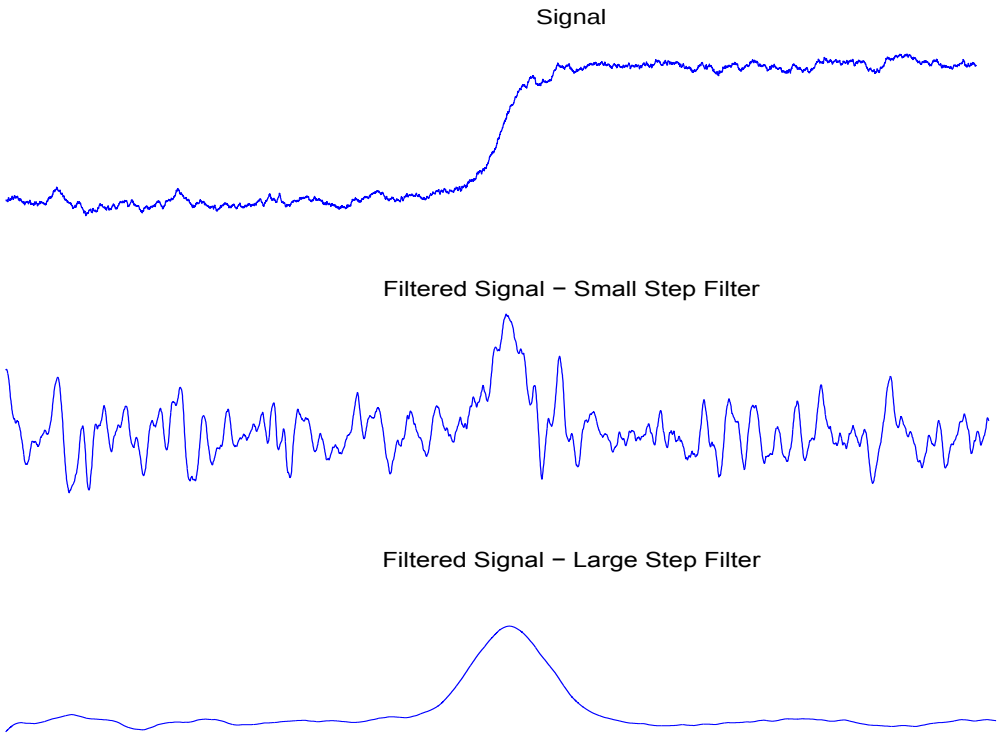


Figure 4.3: **Top:** A sample one dimensional signal with one or more edges. **Middle** The sample signal filtered with a *small* step edge. **Bottom** The sample signal filtered with a *large* step edge.

Image scale and the size of filters is naturally associated with image pyramids. An image pyramid is the successive down scaling of an image, cf. Figure 4.4. 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 – in pixels . 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.

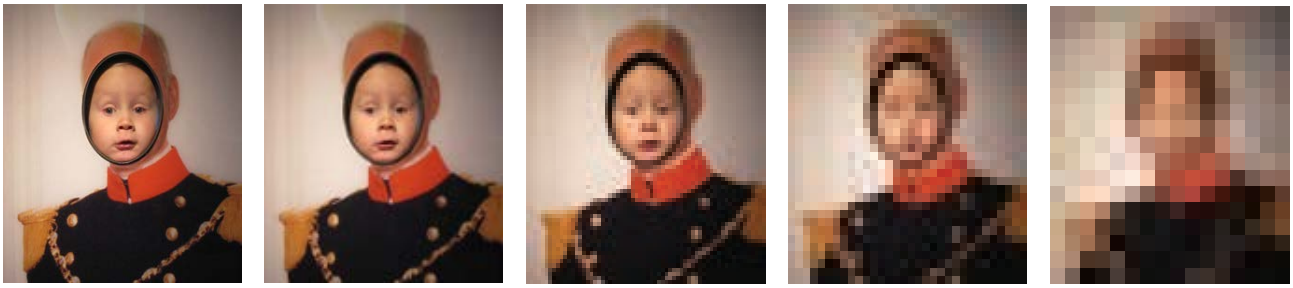


Figure 4.4: A successive down sampling of an image. If these images were placed on top of each other, with a pixel having the same size, they would form a pyramid. Hence the name image pyramid.

4.1.1 Gaussian Kernels

When dealing with image scale, or scale space a work-horse is the *Gaussian filter*, cf. Figure 4.5. A Gaussian filter is given by the following equation

$$g_\sigma = g(\mathbf{x}, \sigma) = \frac{1}{\left(\sqrt{2\pi\sigma^2}\right)^D} \exp\left(-\frac{\|\mathbf{x}\|_2^2}{2\sigma^2}\right) \quad , \tag{4.1}$$

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.

$$I * f \quad ,$$

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

$$I * g_\sigma * f \quad . \tag{4.2}$$

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 it has a lot of very nice practical and theoretical properties, as are e.g. described in [Sporring et al., 1997].

When doing feature extraction, a very commonly used filter, f, is a derivative of some order, as discussed in Section 4.2. For computational reasons, the order of operations in (4.2), is as follows

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

If f is a derivative, typical filters are shown in Figure 4.5.

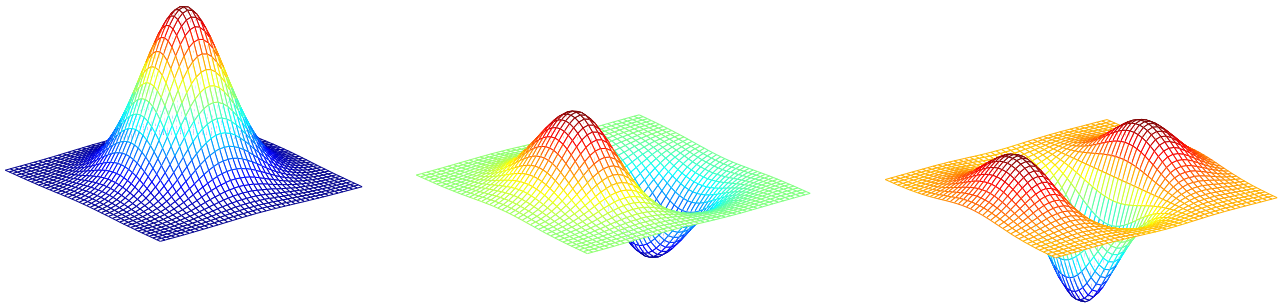


Figure 4.5: An example of a Gaussian kernel, and it's first and second derivative wrt. the x axis.

4.1.2 Scale and Feature Extraction

This section presents a very rudimentary outline of the vast area of scale space theory [Sporring et al., 1997]. For more information the interested reader is referred to [Lindeberg, 1997, Sporring et al., 1997]. In the context

[1] Often one or two.

of this text, the overall message of Section 4.1, 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 taken in a non controlled setting, in this case the scale is generally unknown. As an example consider the images in Figure 4.1. 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. An often used technique is thus to do feature extraction at range of scales. The idea is that the 'right' scale will be used in one instance, cf. e.g. Section 4.4.1.

4.2 The Aperture Problem

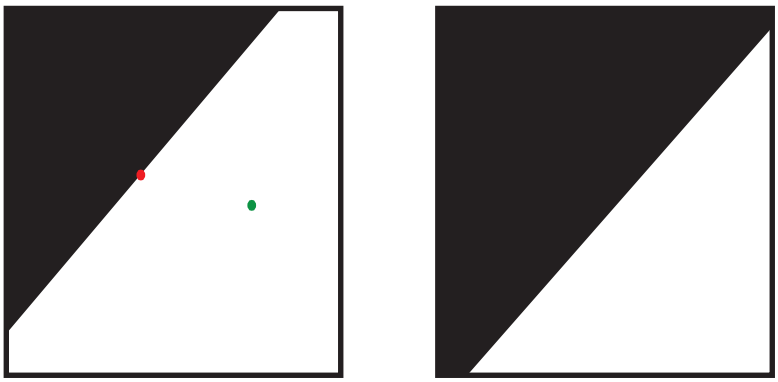


Figure 4.6: An illustration of the aperture problem. Two features (the red and green dots) are annotated in the left image. Consider finding these features in the right image, given that this image is a slightly transformed version of the left. The red dot – located on the line – must be located on the 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.

As a second general consideration, before specific feature extraction methodologies are introduced, the aperture problem will be covered. As illustrated in Figure 4.6, the location of a feature can only be uniquely determined 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 those that are located on image gradients. For point features there should be a large gradient in all/both directions, cf. Figure 4.6. 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.

4.3 Harris Corner Detector

A popular image feature is 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* [Harris and Stephens, 1988], 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

[2]This is under the assumption that only local operations are used.

to ensure it robustness to noise[3]. In practise we achieve this by convolving with a Gaussian g_σ, and we consider the measure

$$c(x, y, \Delta x, \Delta y) = g_\sigma * (I(x, y) - I(x + \Delta x, y + \Delta y))^2 \tag{4.3}$$

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

$$\begin{aligned}
c(x, y, \Delta x, \Delta y) &= g_\sigma * (I(x, y) - I(x + \Delta x, y + \Delta y))^2 \\
&\approx g_\sigma * \left(I(x, y) - \left(I(x, y) + \begin{bmatrix} I_x(x, y) & I_y(x, y) \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} \right) \right)^2 \\
&= g_\sigma * \left(\begin{bmatrix} I_x(x, y) & I_y(x, y) \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} \right)^2 \\
&= \begin{bmatrix} \Delta x & \Delta y \end{bmatrix} \begin{bmatrix} g_\sigma * I_x(x, y)^2 & g_\sigma * I_x(x, y) I_y(x, y) \\ g_\sigma * I_x(x, y) I_y(x, y) & g_\sigma * I_y(x, y)^2 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} \\
&= \begin{bmatrix} \Delta x & \Delta y \end{bmatrix} \mathbf{C}(x, y) \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} \ ,
\end{aligned} \tag{4.4}$$

where $\approx$ denotes a first order approximation and

$$\mathbf{C}(x, y) = \begin{bmatrix} g_\sigma * I_x(x, y)^2 & g_\sigma * I_x(x, y) I_y(x, y) \\ g_\sigma * I_x(x, y) I_y(x, y) & g_\sigma * I_y(x, y)^2 \end{bmatrix} \ . \tag{4.5}$$

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 directions, implying that (4.4) 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].

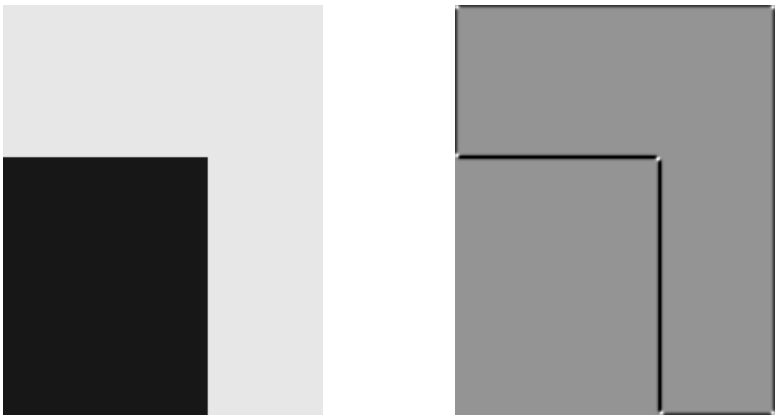


Figure 4.7: An example of $r(x, y)$. The right image is $r(x, y)$ computed on the left image. White is large positive values, black is large negative values, and grey are small values.

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,

$$r(x, y) = \lambda_1 \lambda_2 - k(\lambda_1 + \lambda_2)^2 \ , \tag{4.6}$$

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.

[3] If no smoothing occurs $\mathbf{C}(x, y)$ will also only have rank one, cf. (4.5).

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

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

This is illustrated in Figure 4.7. 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 (4.6). 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, cf. Appendix A, it is known that a relationship between the eigenvalues of a 2×2 matrix and the determinant and trace is given by

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

Inserting this into (4.6) gives

$$\begin{aligned} r(x, y) &= \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2 \\ &= \det(\mathbf{C}(x, y)) - k \cdot \text{trace}(\mathbf{C}(x, y))^2 \\ &= ab - c^2 - k(a + b)^2 \ . \end{aligned} \tag{4.7}$$

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

$$r(x, y) > \tau \ , \tag{4.8}$$

for some threshold τ as corners. As argued in Section 4.3.1, non-maximum suppression has to be applied. Typical values for τ is from 0.1 to 0.8 times the maximum $r(x, y)$ value for the image in question, and some times even lower. I.e. scaling by 0.8

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

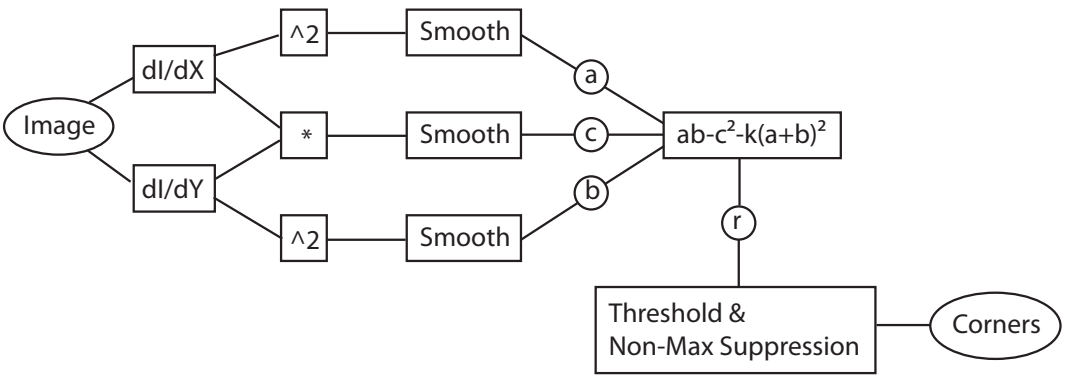


Figure 4.8: A flow diagram for the Harris Corner detection algorithm. Note, that a smoothing is performed in computing the image derivatives, c.f. Section 4.1.

4.3.1 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 (4.8) 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 4.7. We would, however, like a unique position of our features, and therefor (4.8), 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, (4.8) is supplemented with, has the following form[5]

$$\begin{aligned} r(x,y) &> r(x+1,y) \quad \wedge \\ r(x,y) &\geq r(x-1,y) \quad \wedge \\ r(x,y) &> r(x,y+1) \quad \wedge \\ r(x,y) &\geq r(x,y-1) \end{aligned} \tag{4.9}$$

It is noted that there is a mix between $>$ and $\geq$ in (4.9), this is done in order to break the tie in the case off two neighboring pixels having the same values, but are local maximums otherwise. This concludes the Harris corner detection algorithm, and a flow diagram is given in Figure 4.8 along with and example in Figure 4.9.

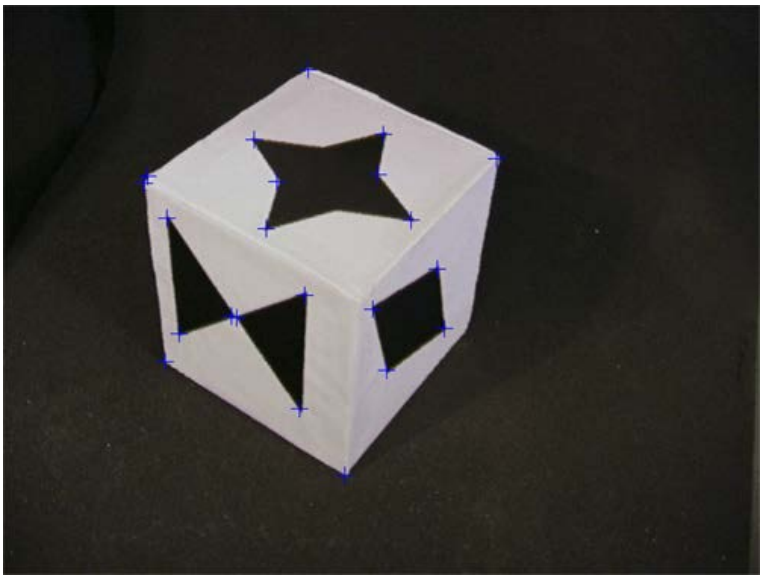


Figure 4.9: An example of applying the Harris corner detector.

4.3.2 Sub-pixel Accuracy*

Lastly, it is noted that once the Harris corners have been found, as described above, and illustrated in Figure 4.8, 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, cf. Figure 4.10. 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 $\delta_y = 0$.

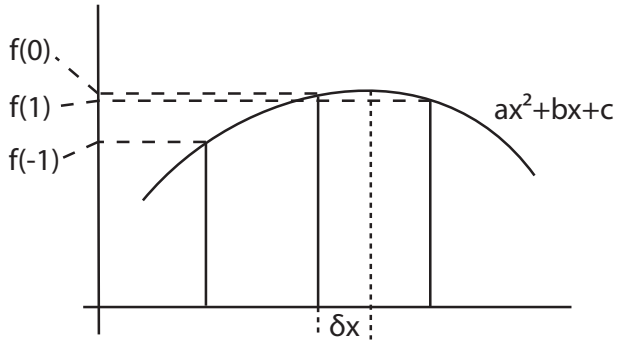


Figure 4.10: A schematic illustration of fitting a second order polynomial in order to find the sub-pixel offset Δ_x.

4.4 Blob Detection

Considering again the aperture problem, another entity which has a high degree of change 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

$$\det(\mathbf{H}) = \det\left(\begin{bmatrix} I_{xx} & I_{xy} \\ I_{xy} & I_{yy} \end{bmatrix}\right) = I_{xx}I_{yy} + I_{xy}^2 = \lambda_1\lambda_2 \ . \tag{4.10}$$

$$\text{trace}(\mathbf{H}) = \text{trace}\left(\begin{bmatrix} I_{xx} & I_{xy} \\ I_{xy} & I_{yy} \end{bmatrix}\right) = I_{xx} + I_{yy} = \lambda_1 + \lambda_2 \ . \tag{4.11}$$

$$\tag{4.12}$$

These measures are then formed into detector by thresholding, doing non-maximum suppression and possibly determining sub-pixel accuracy as described in Section 4.3 (Section 4.3.1 and Section 4.3.2 specifically). It is noted that the trace of the hessian is also called the Laplacian, and is denoted by

$$\nabla^2 I = I_{xx} + I_{yy} = I * \left(\frac{\partial^2 g_\sigma}{\partial x^2} + \frac{\partial^2 g_\sigma}{\partial y^2}\right) \ . \tag{4.13}$$

The lapalacia filter is illustrated in Figure 4.11.

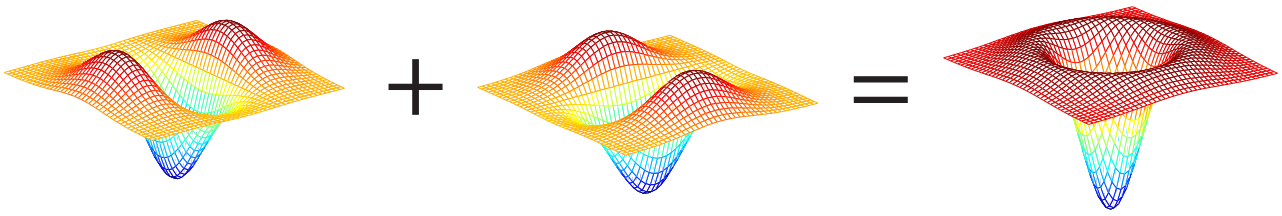


Figure 4.11: The Laplacian (trace of Hessian) filter, seen as the addition of the $\frac{\partial^2 g_\sigma}{\partial x^2}$ and $\frac{\partial^2 g_\sigma}{\partial y^2}$ filters.

4.4.1 Difference of Gaussian (DoG) & the SIFT Detector

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 [Lowe, 2004]. 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.

$$I * g_{s\cdot\sigma} - I * g_\sigma = I * \left(g_{s\cdot\sigma} - g_\sigma\right) \ , \tag{4.14}$$

where s is a scalar and σ is the scale of the operation. An example of this filter is shown in Figure 4.12. 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 4.12 to the Laplacian in Figure 4.11. 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.

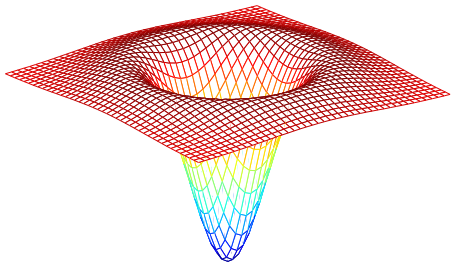


Figure 4.12: The Difference of Gaussian(DoG). Note the similarity to the kernel in Figure 4.11. In scale space the DoG detector is computed by first convolving with Gaussian kernels and then subtracting, cf. Figure 4.15.

A reason for DoG detectors being popular, is that they are very efficient to implement in a scale space frame work, cf. Section 4.1. That is, since the DoG feature detector is the difference between an image at two different scales, then if a scale space pyramid is already computed, extracting DoG features is a matter of subtracting images from each other. This is how it is done with the SIFT detector, and an example is illustrated in Figure 4.15.

In the SIFT case the the sampling in the scale space direction is typically set such that there are three samples per octave. Since an octave corresponds to a scale factor of two, s is given by

$$s = \sqrt[3]{2} \ .$$

SIFT descriptors are typically sought over several octaves, where each octave is processed separately. The Transition from one octave to the next is achieved by simply down-sampling the image with a factor of two. This makes the algorithm more efficient in that it reduces the number of pixels by a factor of $2 \cdot 2 = 4$.

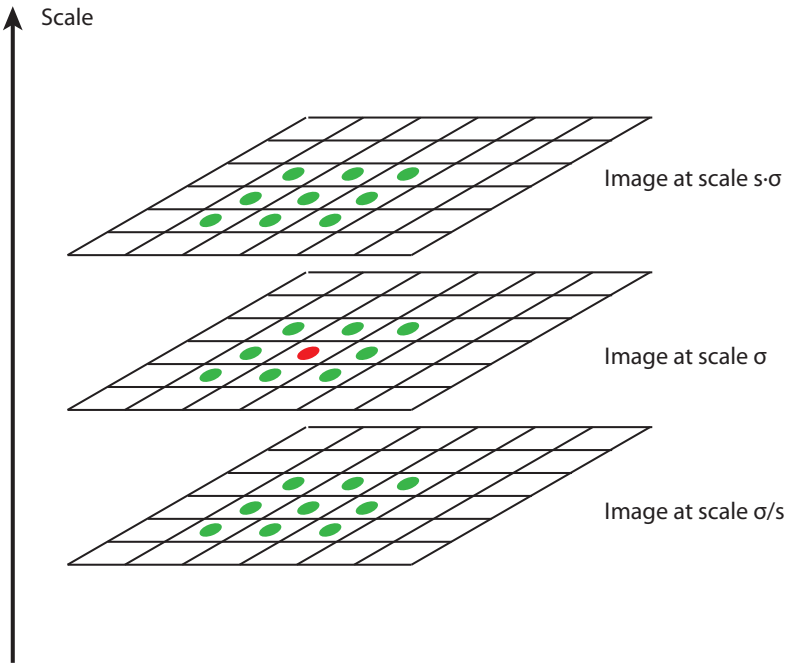


Figure 4.13: An illustration of non-maximum suppression in scale space. The red pixel is to be greater than all the green pixels (par some tie breaking mechanism). That is, it should fulfill the non-maximum suppresion criteria for its own image as well as be greater than the pixels one scale step up and down.

In the case of the SIFT detector, a point, (x, y), at a given scale, if the corresponding DoG value, ν, fulfills the following three criteria:

1. It's absolute value should be above a certain threshold, τ. That is $\nu < -\tau$ or $\tau < \nu$.

2. It should be a local optimum, i.e. it should pass a non-maximum suppression if ν is positive and a non-minimum suppression[6] test if ν is negative. As illustrated in Figure 4.13, this non-maximum suppression test should not only be performed in the given DoG image, but also an image up or down in the scale space.

3. As in the case of the Harris corner detector, a feature should not be too elongated indicating a line instead of a point feature.

The requirement of non-maximum suppression up and down in scale, is the reason that five (=3+2) instead of three DoG images are computed, as seen in Figure 4.15. The non-maximum suppression in scale is also a way of doing scale selection, i.e. finding the scale at which a feature is most dominant. There is, however, the issue that the DoG images at different scales are not directly comparable, in that the image signal naturally gets weaker at larger scales. As shown in [Lindeberg, 1998], this small scale bias can be corrected by making different polynomial approximations, depending on the filters used. In the case of DoG filters this implies that (4.14), should be updtaed to

$$\frac{I * \left(g_{s\cdot\sigma} - g_{\sigma}\right)}{s \cdot \sigma - \sigma} \quad . \tag{4.15}$$

An example of extracting SIFT features is seen in Figure 4.14. For more details on the SIFT detector please refer to [Lowe, 2004].

[6] In the remainder of this section non-maximum and non-minimum suppression is only referred to as non-maximum suppression for ease of read.

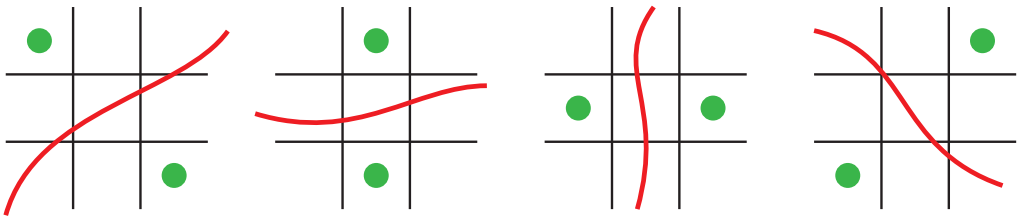
Figure 4.14: An example of applying the SIFT detector to the image from Figure 4.15. Note that several octaves are used.

4.5 Canny Edge Detector

Apart from point features we often want to extract edges or lines in an image. One of the most popular way of doing this is via the *Canny edge detector* [Canny, 1986], also sometimes referred to as the *Canny-Deriche detector*, because R. Deriche's method for efficiently smoothing with a Gaussian kernel is often used [Deriche, 1990][7].

The Canny edge detector uses the gradient magnitude, $\sqrt{I_x^2 + I_y^2}$, as an edge measure, cf. Figure 4.17-middle-left. Non-maximum suppression is also preformed on this measure, cf. Section 4.3.1. But differently from the point case, it should only be required that an 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 4.17-middle-right.

This non-maximum suppression is practically done by classifying an edges direction into one of four cases, as illustrated by the color wheel in Figure 4.17, as seen in Figure 4.17-middle-left. Following this classification the pixels most perpendicular to the edge are use for the comparison, as illustrated in the following figure



Where the edge is denoted by the red lines and the green dots are the pixel used for comparison. Note, that for the sake of illustration the lines have been draw as such, although in practice it is individual pixels classified as lines.

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. This is illustrated in Figure 4.16.

[7] This technicality is not covered here.

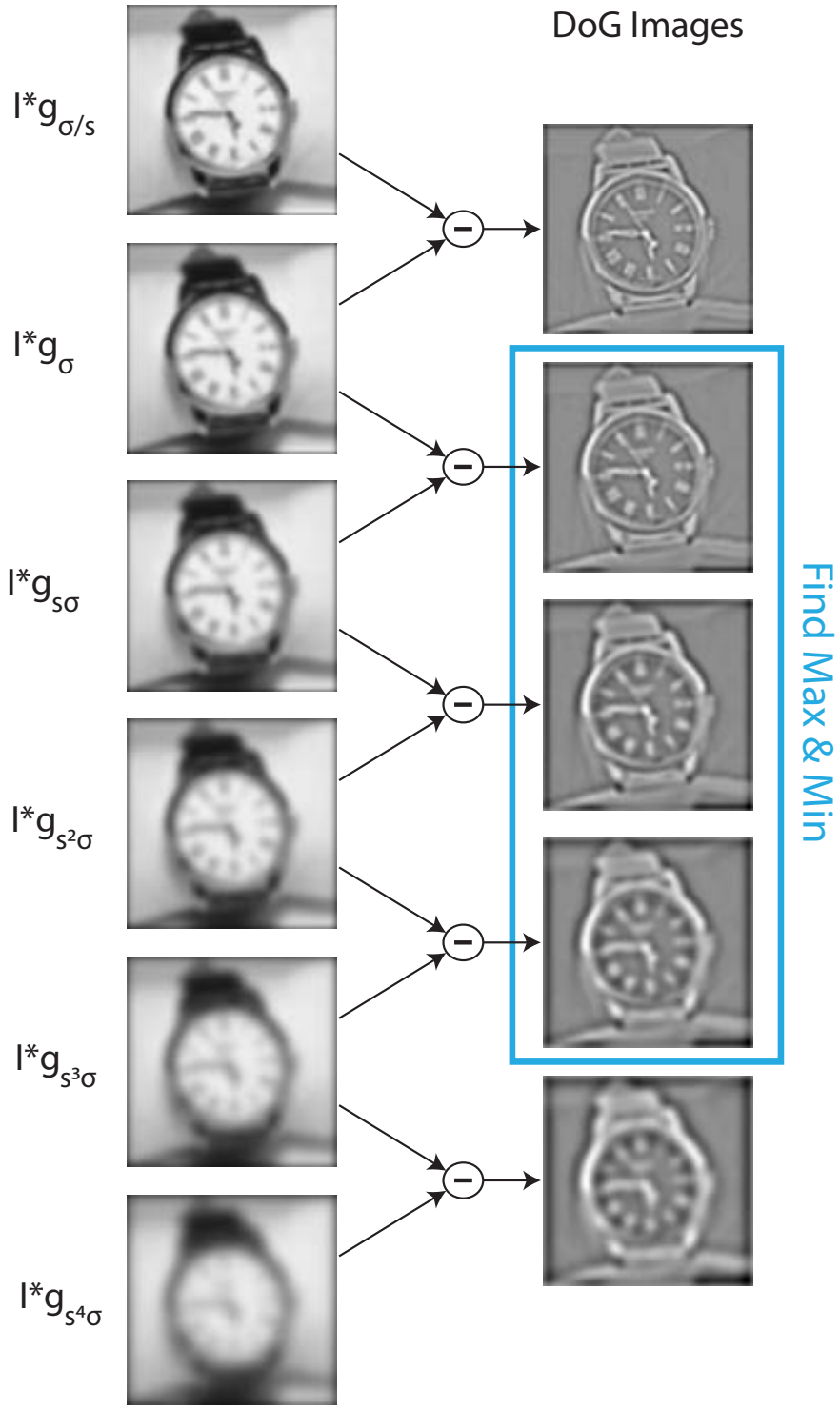


Figure 4.15: An illustration of how the DoG images are computed in the SIFT detector framework. Here three scale space samples are used per octave, as is the norm, implying that $s = \sqrt[3]{2}$. The reason that five Dog images are computed, to evaluate three images is, that in order to do non-maximum suppression the image one scale step up and down is needed.

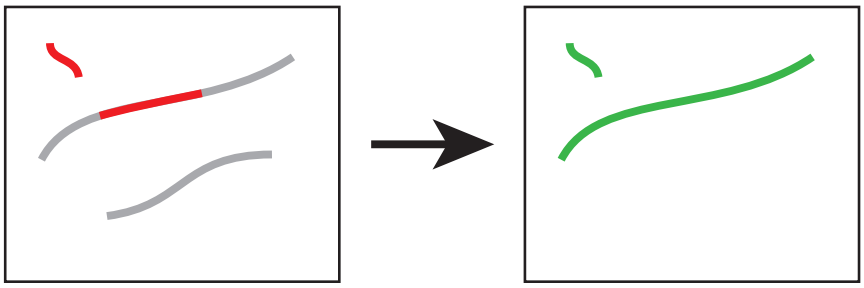


Figure 4.16: The use of two thresholds in the Canny edge detector. Assume that the in the left image the red pixels are edge pixels larger than τ_1 (and thus also τ_2) and the gray values are edge pixels larger that τ_2. but smaller than τ_1. Then the resulting Canny edges will be the green lines in the image to the right.

The motivation for these two thresholds 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, cf. Figure 4.17-bottom-left, and then only keeping the line segments where at least one pixel has a gradient magnitude above τ_1, cf. Figure 4.17-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. An outline of the Canny edge detector can be found in table 4.1.

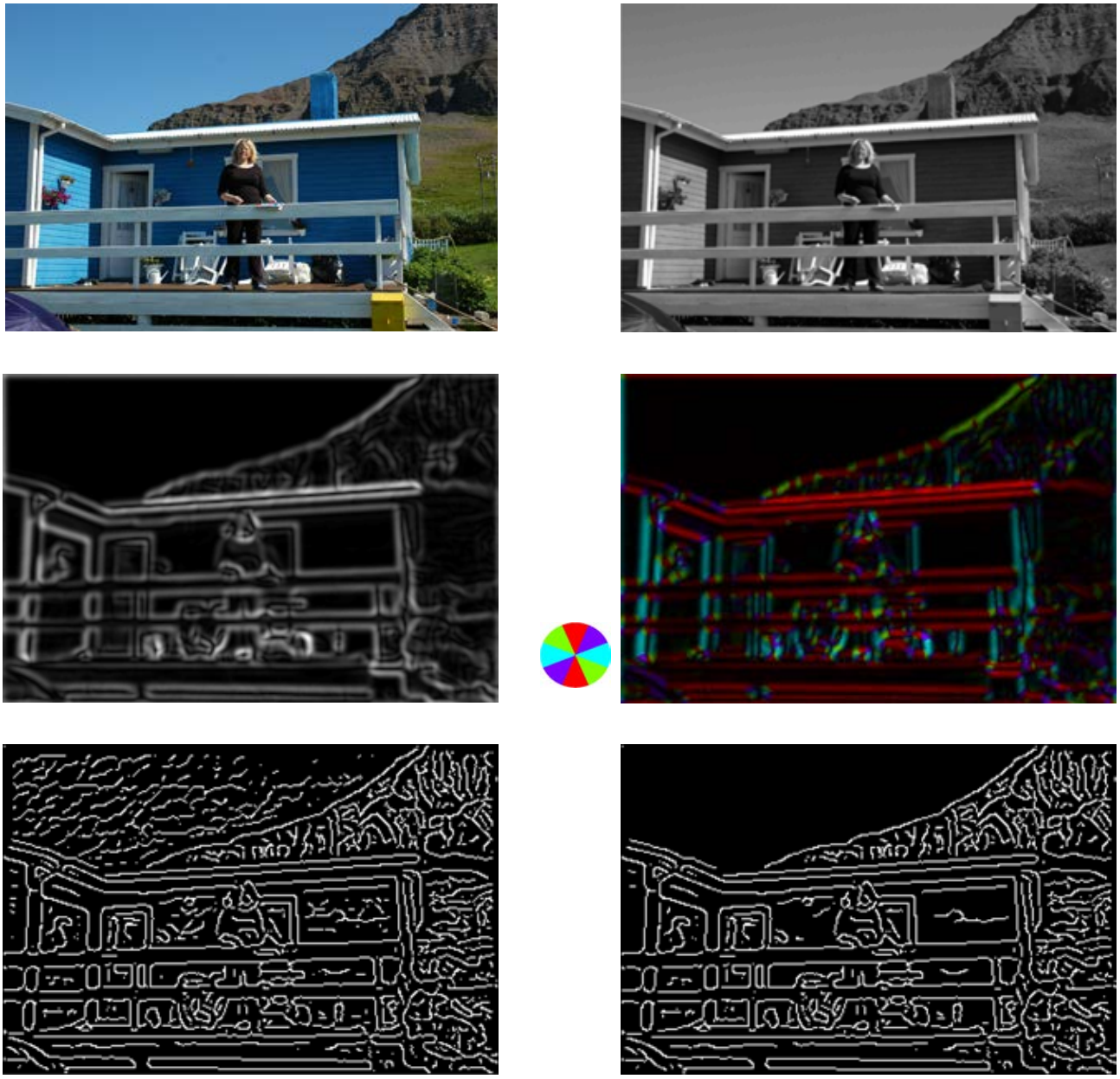


Figure 4.17: Illustration of the Canny edge detector. **Top:** The original color image, and the gray scaled version on which the operation is performed. **Middle:** The gradient magnitude $I_x^2 + I_y^2$, and the gradient magnitude with the orientations used to do non-maximum suppression. There is also 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.

Given an image I, two thresholds τ_1 and τ_2, and a scale σ

1. Compute the gradients, I_x and I_y, of the image I in both directions, using an appropriate scale, σ, wrt. to the Gaussian scale space, c.f. Section 4.1.

2. Compute an edge measure, E, in the form of the gradient magnetude, i.e.

$$E = \sqrt{I_x^2 + I_y^2} \ .$$

3. Perform non-maximum suppression on the edge measure, E, but only in the direction of the gradient, i.e. compare a position to it's neighbors in the directions of

$$\frac{1}{E}\begin{bmatrix} I_x \\ I_y \end{bmatrix} \quad \text{and} \quad \frac{-1}{E}\begin{bmatrix} I_x \\ I_y \end{bmatrix} \ ,$$

   where these directions have been clamped to being in one of four directions, corresponding to an eight neighborhood.

4. Threshold the part of the magnitude image E, which has passed the non-maximum suppression, by the two thresholds τ_1 and τ_2, where $\tau_1 > \tau_2$. Denote the resulting binary images by E_1 and E_2 respectively.

5. Extract the connected components from E_2, and denote the components contains at least one 'passed' pixel from E_1 as an edge.

Table 4.1: An outline of the Canny edge detector.

Chapter 5

Robust Model Fitting

When we want to extract non-local image features, such as lines, circles or ellipsoids, we are faced with the challenge that almost all images have a lot of disturbing noise and occlusions. As an example consider Figure 5.1, where we would be hard pressed to detect the black line based on a local edge detector. To address this we can in some cases, like the one in Figure 5.1, exploit that we are looking for feature that have a 'large support' in the image. I.e. we can consider the features as models that have to be fitted to the image data and then look for the model that fit the data best. In Figure 5.1 this corresponds to looking for the lines that correspond to most of the detected edge pixels.

A way to achieve this, is by considering the features as parameterized models that should be fitted to the image content, e.g. center and radius for a circle, and then search the parameter space for the best models. The archetypical way of doing this is via the Hough transform, presented in Section 5.1, aimed at dealing with lines. This will be generalized to arbitrary models in Section 5.2, and an alternative search strategy, named Ransac and better suited for higher dimensional models, will be presented in Section 5.3. These parameter search algorithms are often used as initializations for other robust fitting algorithms. Thus the subject of robust statistics is also briefly introduced, mainly through the presentation and motivation of robust norms, cf. Section 5.4.

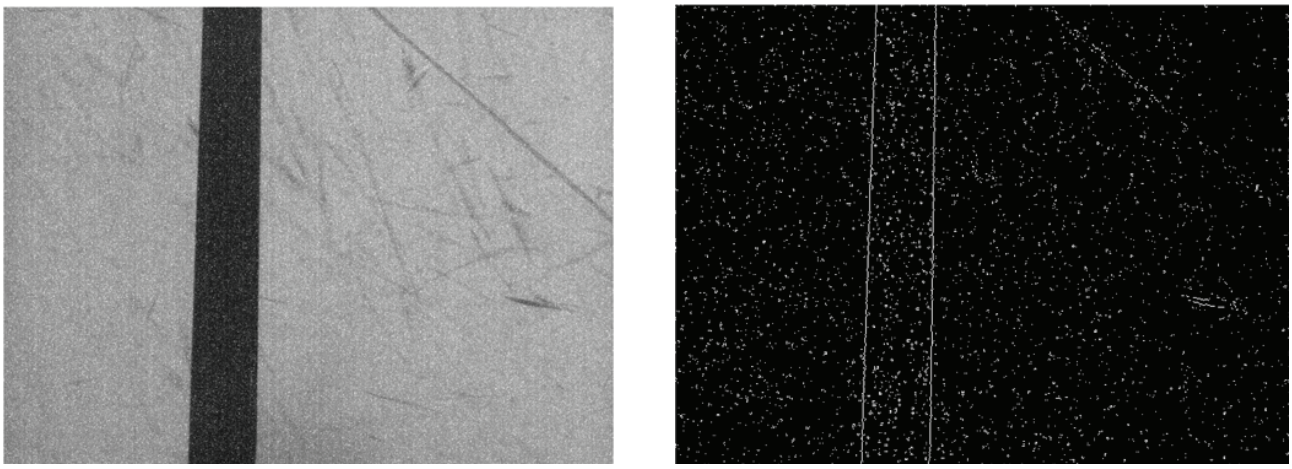


Figure 5.1: Suppose we have to find the black tape in the image to the left. This is a common task in many visual robot navigation systems – the image quality is also typical. The right image shows an edge filter applied to the left image. The edge filter is based on the Sobel operator.

5.1 Hough Transform

The original idea for the *Hough Transform* was first put forth by P.V.C. Hough in 1959, [Hough, 1962], as a method for machine analysis of bubble chamber photographs. It was developed into it's present day form by Duda and Hart [Duda and Hart, 1972] — as presented here. The Hough transform works by parameterizing all possible lines by (θ, r), where r denotes the minimum distance of the line to the origo, and $[\cos(\theta), \sin(\theta)]^T$

denotes the normal of the line, cf. Figure 5.2. In this notation the equation of the line is given by

$$\cos(\theta) \cdot x + \sin(\theta) \cdot y = r \ , \tag{5.1}$$

or in relationship to the homogeneous line representation in Section 1.1.

$$\mathbf{l} = [\cos(\theta), \sin(\theta), -r]^T \ .$$

The reason for using for this parametrization is that *all* possible 2D lines can be represented by two parameters, instead of the three that are often used.

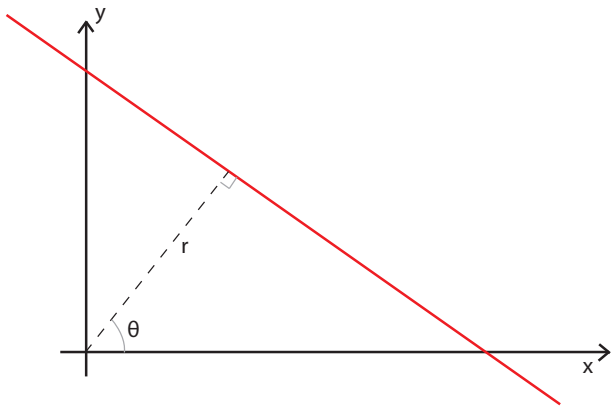


Figure 5.2: Any line in 2D can be represented as as it's minimum distance to the origo, r, the angle of the shortest distance to the line relative to the x-axis, θ, as described in (5.1). It is noted that θ is also the angle of the lines normal.

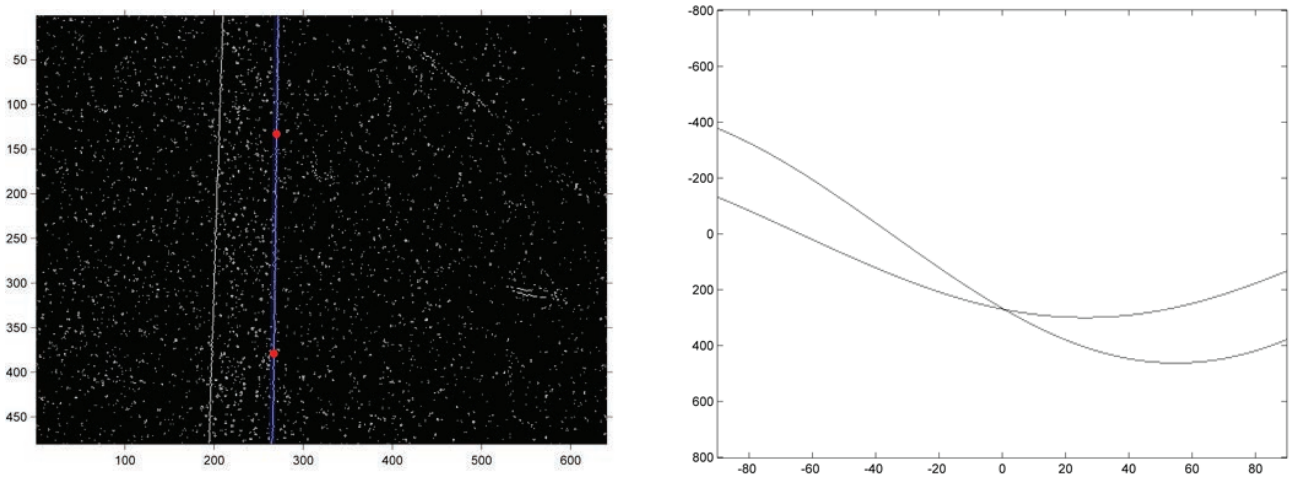


Figure 5.3: Two edge pixels in the edge image, denoted by red dots in the left image induce the two curves seen in the Hough space, seen to the right. The curves represent the parameterizations of all lines that can go through the points. The two curves in Hough space intersect in one point, corresponding to the line defined by the two points. This line is drawn in blue in the left image.

As mentioned, the aim of the Hough transform is finding the lines with most support among the edge pixels. Such support is found by discretizing the line parameter space (θ, r), and having each edge pixel vote for all possible lines that could go through it, cf. Figure 5.3. A way of doing this in practice, is by observing that for each edge pixel, (x_j, y_j), equation (5.1) defines a curve in (θ, r) space. For each discrete value of θ_i equation (5.1) gives a corresponding value of r_{ij}, i.e.

$$r_{ij} = \cos(\theta_i) \cdot x_j + \sin(\theta_i) \cdot y_j \quad \text{for} \quad \theta_i \in [-\frac{\pi}{2}, \ldots, \frac{\pi}{2}] \ . \tag{5.2}$$

Where r_{ij} is rounded of in a suitable manner. Then a vote is cast for point θ_i, r_{ij} in the the line parameter space (θ, r). This parameter space is represented as a matrix or an image initialized to all zeros before the voting

commences. An example of this is given in Figure 5.3, for two edge pixels of Figure 5.1, with the coordinates

$$[x_1, y_1] = [270, 133] \quad , \quad [x_2, y_2] = [267, 379] \ ,$$

giving rise to two curves in Figure 5.3-Right, and defined by (5.1). These two curves in *Hough space* intersect in one point, corresponding to the one line going through both points. This intersection in Hough space is given by

$$\theta = 0.0122 \quad , \quad r = 270 \ .$$

The equation of the line, in homogeneous coordinates is then given by

$$\mathbf{l} = \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \\ -r \end{bmatrix} = \begin{bmatrix} \cos(0.0122) \\ \sin(0.0122) \\ -270 \end{bmatrix} = \begin{bmatrix} 0.9999 \\ 0.0122 \\ -270 \end{bmatrix} \ .$$

The reason θ_j only runs from $-\pi/2$ to $\pi/2$, i.e. an arbitrary span of π, is that θ represents the angle of the lines normal, and a flip of that normal, i.e. adding π to θ, will still be a normal to the line. Thus a θ span of more that π would be redundant.

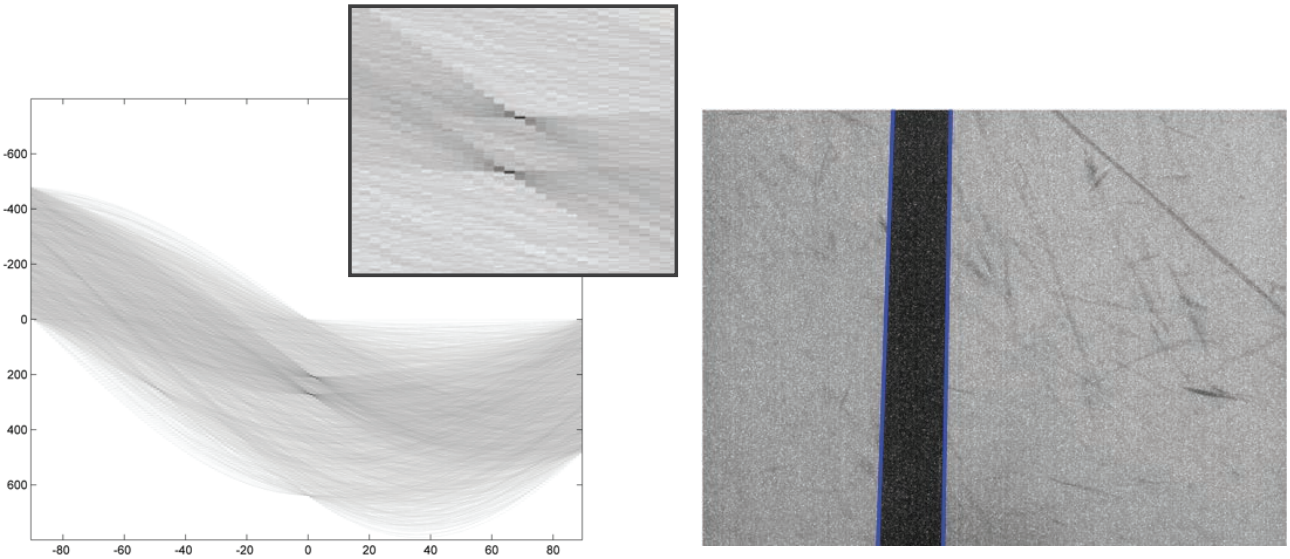


Figure 5.4: The result of applying the Hough transform to the image of Figure 5.1. To the left is seen the computed Hough Transform, with a zoom of the area around the two largest peaks. To the right the two lines corresponding to these peaks are drawn onto the original image.

The result of applying the Hough transform to the image of Figure 5.1 is given in Figure 5.4, where it is seen that it finds the boundaries of the black tape well. This is done by finding the two largest peaks in the Hough space, cf. Figure 5.4-Left, and computing the resulting lines as seen in Figure 5.4-Right. An example on a more complex image is given in Figure 5.5.

Considering the Hough transform in Figure 5.5 it can be observed that decision of what constitutes a peak is not very obvious. For all but the images of the 'simplest' complexity, this is a typical observation about the Hough transform. The reason being that there are a lot of lines, many being close in Hough space, thus 'crowding' the resulting Hough Transform. This issue is complicated by a peak in Hough space usually spanning more that one pixel/cell, due to image noise and Hough space quantization. A practical way to address this issue is to delete (i.e. set to zero) all the Hough values in a region around a detected peak after detection, such that no other peaks are detected there. This in essence corresponds to enforcing a certain distance between detected peaks. Also to lessen the effect of quantization in the Hough space, peak positions can be improved via interpolation.

An implication of the issues of peak detection implies that often the parameters of the Hough transform need to be tuned for complex images to give good results, i.e. parameters such as the quantization resolution and the minimum allow distance between peaks (typically a box corresponding to the max-norm). Another issue is how many peaks to find, the simplest strategy is to set a fix number, e.g. 50 as in Figure 5.5, and then find them extracting them one by one, taking the peak associated with the largest pixel value first. Another

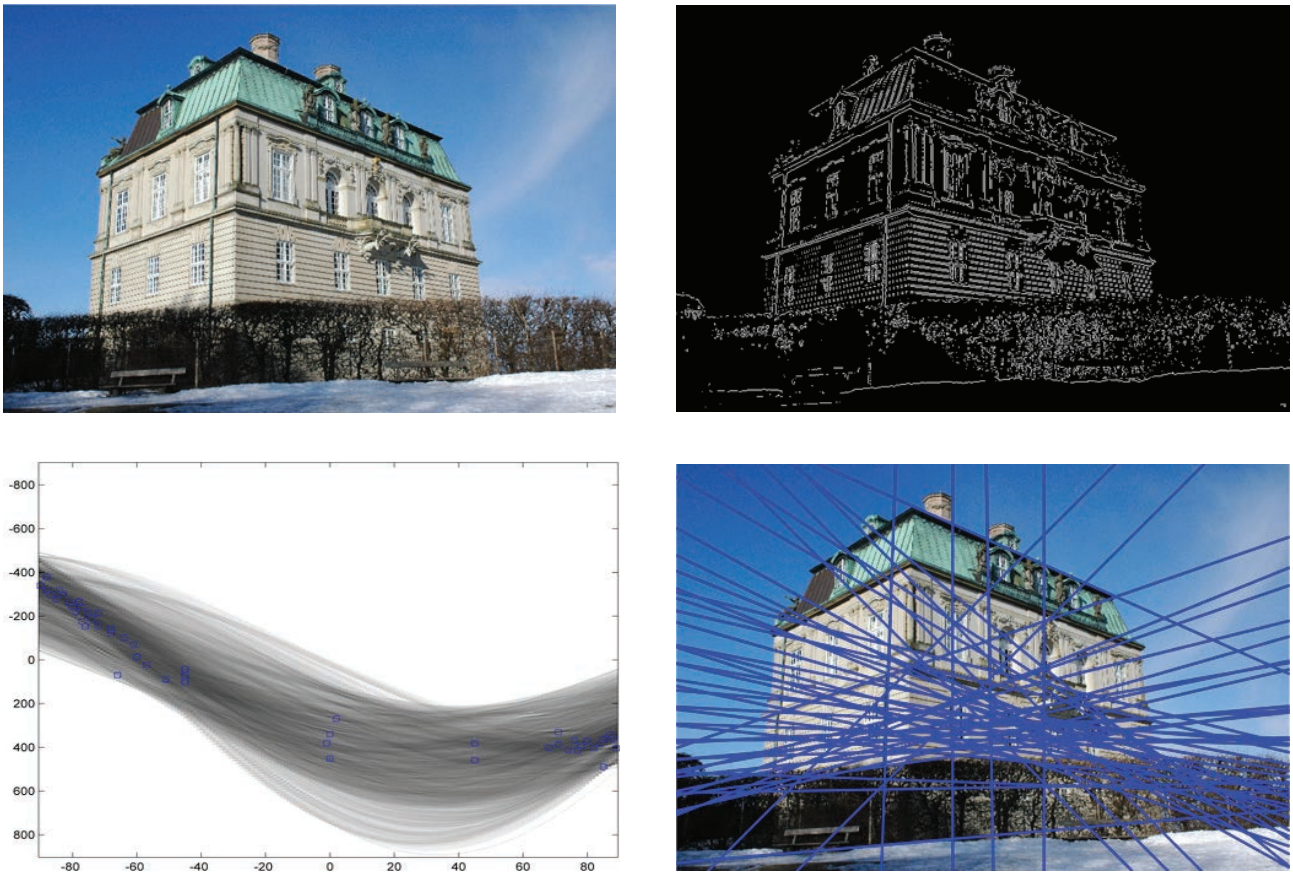


Figure 5.5: An example of applying the Hough transform to a more complex image. **Top Left:** The original image. **Top Right:** Extracted edges via the Sobel filter. **Bottom Left:** The Hough space representing the Hough transform of the edge pixels. The 50 largest peaks in the Hough space have been extracted and are marked with blue squares. **Bottom Right:** The 50 lines corresponding to the 50 extracted peaks in Hough space drawn onto the original image.

strategy is to only find peaks with pixel values over a certain threshold, e.g. 40% of the largest pixel value of the Hough transform. Here we start with the largest first such that if two peaks are too close the smallest will be pruned out. An outline of how to extract lines via the Hough transform is given in Table 5.1.

Lastly it should be mentioned that there is a vast body of work, including published research, on the Hough transform and it's generalized version presented in the next section. Much of this work deals with optimizing different parts of the implementation. Here only the basic algorithms are presented.

5.2 The Generalized Hough Transform

The Hough transform as described above, deals only with the estimation of lines. The approach, however, generalizes to a large class of arbitrary models often fitted to image data. The essence of this generalization is that the individual data points or observations can vote for the model parameters that are consistent with the data, and then the parameters with the most votes are chosen.

To formalize this generalization, instead of two parameter lines, cf. (5.1), we talk of *model classes* $\mathcal{M}(\phi)$ with parameter vectors ϕ. In the line case $\phi = [\theta, r]$, and

$$\mathcal{M}(\phi) = \cos(\phi_1) \cdot x + \sin(\phi_1) \cdot y = \phi_2 \quad .$$

Sometimes, as there is here, there is a need to distinguish between a model class, e.g. a line, and a instance of that model class, e.g. $x + y = 2$ – in such cases denoted a model. The model thus corresponds to a given instance of the model class corresponding to a parameter set ϕ. Often there is no need to distinguish and then both models and model classes are referred to as models.

The notion of edge pixels generalizes to data points $\mathbf{d}_i$. The generalization of the Hough algorithm in Table 5.1, is then for given model class, $\mathcal{M}$ and data $\mathbf{d}_i$:

1. **Extract Edge Pixels from the Input Image** This can e.g. be done by running an edge detection filter in the x and y direction of the image, I. Denote the filters by ∇_x and ∇_y. Then Threshold the gradient magnitude with the value τ. I.e. find pixels for which
$$(\nabla_x * I)^2 + (\nabla_y * I)^2 > \tau^2 \ .$$
A typical edge detecting filter is the Sobel filter, as used in the examples here.

2. **Initialize the Hough Space** Decide on a resolution for θ and r, allocate the corresponding matrix or image, and set al values to zero.

3. **Compute the Hough Transform** For each detected edge pixel cast a vote in Hough space for each line that goes trough it, using (5.2).[a]

4. **Extract Peaks** Do the following until enough peaks have been extracted:
   (a) Find largest value in the Hough space. This is the peak.
   (b) Save the (θ, r) vale of this peak, possibly after interpolation.
   (c) In a small area, e.g. 5 by 5 pixels, around the peak set all values of the Hough space equal to zero.

5. **Convert the Extracted Peaks to Lines** From the extracted peak (θ, r) values, compute the lines in your desired representation.

[a] This and the previous step can be completed using the Radon transform, cf. [Radon, 1986].

Table 5.1: Pseudo code for extracting lines via the Hough transform.

1. Discretize the parameter space into a D-dimensional data cube, where D is the dimension of ϕ. Set all elements of this data cube, equaling the Hough space, to zero.

2. For all data points $\mathbf{d}_i$, vote for the parameter configurations ϕ that are consistent with it. This is done by adding one to the relevant places in the data cube or Hough space.

3. Find the peaks in the Hough space, corresponding to ϕ_j, and output the $\mathcal{M}(\phi_j)$ as results.

This approach is seen to be very general and can e.g. be used to fit circle, ellipses, squares or other geometric shapes to edge data.

5.2.1 Example: Circle Fitting

An example of the generalized Hough transform is given in Figure 5.6, where circles are fitted to image gradients. So here the model class $\mathcal{M}$ is circles parameterized by $\phi = [a, b, R]$, in the following way

$$\mathcal{M}(\phi): \quad (x-a)^2 + (y-b)^2 = R \ , \tag{5.3}$$

where (x, y) are image coordinates. Thus (a, b) is the circle center and R it's radius.

Since ϕ in this case is three dimensional, the Hough space is also three dimensional. The discretization of the parameter space is such that all elements of ϕ have a resolution of one pixel, the range of (a, b) is the same as the image, and the range of R is 7 to 25. The data points $\mathbf{d_i}$ are in this case the edge pixels, see Figure 5.6-Top-Right.

The question is then, which circles are consistent with a given edge or data point $\mathbf{d_i}$? I.e. which ϕ should an edge vote for? To answer this, note first that for given (x, y) and R, (5.3) is a circle in (a, b) space with a center of (x, y) and a radius of R. When varying R, (5.3) then describes a cone in a, b, R space, equalling the Hough space, cf. Figure 5.7. So each edge pixel (x_i, y_i) should increase by one all the cells in the Hough space located on a cone. A slice through the three dimensional Hough space is seen in Figure 5.6-Bottom-Left.

Figure 5.6: An example of applying the generalized Hough transform with the aim of fitting circles to image gradients. **Top Left:** The original image. **Top Right:** Extracted edges via the Canny edge detector. **Bottom Left:** A cross section of the 3D Hough space, corresponding to a constant radius of 19 pixels. **Bottom Right:** The 11 most dominant circles in the Hough space drawn onto the original image.

As in the standard hough case, peaks are found, and a small area around them are set to zero to force the peaks apart. The result is seen in Figure 5.6-Bottom-Right. Here it is noted that the spider man head has a lot of edges, where enough of them are located on a circle for the algorithm to give a response.

5.2.2 Further Discussion

The Hough transformation and it's generalization is a very used technique in computer vision, especially in images with 'simple' content. A reason is that it is very robust toward random image noise e.g. arising from poor imaging conditions. A main issue with the generalized Hough transform is that the memory and computational power grows exponentially with the dimensions of the parameter space ϕ. So if we e.g. wanted to fit a seven dimensional model with just a resolution of 50 bins for each parameter we would need a Hough space with $50^7 = 7.8125e11$ elements. Thus Hough space techniques are typically only used for very low dimensional models. A way to deal with modest dimensional models is via the Ransac algorithm presented in the next Section, Section 5.3. Lastly it should be mentioned, that often both the standard and generalized Hough transforms are simply referred to as Hough transforms.

5.3 Random Sampling Consensus – Ransac

The *random sampling consensus* algorithm, referred to as *Ransac* [Fischler and Bolles, 1981], [Hartley and Zisserman, 2003], has successfully been applied to many computer vision problems. This algorithm can be seen as a modified version of the Hough transform, where only a sparse set of the Hough space is

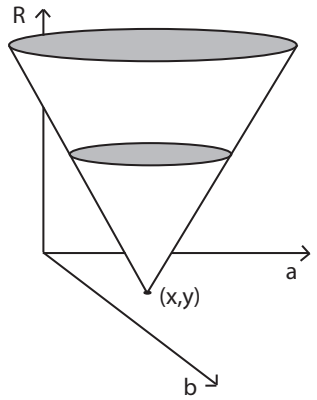


Figure 5.7: The points in Hough space representing models consistent with an edge pixel at (x, y), is a cone centered at (x, y).

Person	Height	Unit
1.	194	cm
2.	192	cm
3.	181	cm
4.	172	cm
5.	173	cm
6.	173	cm
7.	184	cm
8.	173	cm
9.	170	cm
10.	1.81	cm
11.	170	cm
12.	192	cm

Table 5.2: A collection of person heights. Note that measurement 10 is off by ca. a factor of a 100 indicating that meters instead of centimeters were uses as the unit of measurement.

computed, although the algorithm is typically presented in a different way.

5.3.1 Inliers and Outliers

The Ransac algorithm is used to fit a model to 'contaminated' data. It works by the assumption, that the given data points, $\mathbf{d}_i$, are divided into two parts *inliers* and *outliers*, where the inliers are the 'good' observations, which correspond to the underlying model we are trying to estimate, and the outliers are 'bad' or 'contaminated' observations. This is schematically illustrated in Figure 5.8. The aim is then to fit a model to the inliers and leave the outliers out of the estimation, the problem is however, that it is not known which data point, $\mathbf{d}_i$, are inliers and which are outliers.

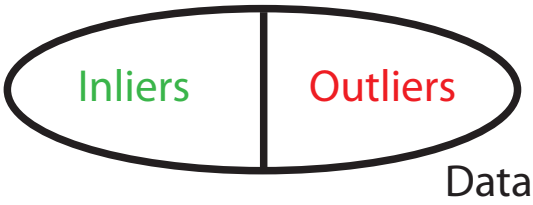


Figure 5.8: A schematic illustration of the assumption that the data, or data points, $\mathbf{d}_i$, are divided into inliers and outliers.

As an example consider the collection of person height measurements in Table 5.2, where there is a good indication of measurement 10 being erroneous, or an outlier. This is a typical recoding mistake of using the wrong units. If we based on these data were to calculate the average height, we would get $164.65\ cm$ with

Given a model class $\mathcal{M}(\phi)$, a set of data points, $\mathbf{d}_i$, $i \in \{1, \ldots, M\}$ and a threshold T, do the following for N iterations:

1. **Randomly Draw a Sample** of size S from $\mathbf{d}_i$, $i \in \{1, \ldots, M\}$, where each data point is selected with equal probability.
2. **Estimate a Model** based on the sample estimate a model, i.e. ϕ of $\mathcal{M}(\phi)$.
3. **Compute the Consensus** of the model ϕ by counting how many of the M data points for which
$$\text{dist}(\mathcal{M}(\phi), \mathbf{d}_i) < T \ , \tag{5.4}$$
for some distance function 'dist'.

The model parameters ϕ with the highest consensus is the resulting model. The sample size S should be minimal, in the sense that it should be large enough to estimate ϕ, but no larger.

Table 5.3: Pseudo code for the basic Ransac algorithm.

all the data and $179.45\ cm$ by omitting observation 10. In the current framework measurement 10 should be labelled as an outlier, and the rest of the observations as inliers. The difference in the two estimations of average height is very considerable, and illustrates why outliers are sought to be omitted from an estimation. Another point to note is that, apart from our prior understanding of the problem, what makes us identify measurement 10 as an outlier is because it 'sticks out' w.r.t. the other measurements.

In relation to the Hough transform, the Ransac algorithm only fits one model to the data. In this context the inliers are the data points, $\mathbf{d}_i$, that support the model to be fitted, in Hough space. The other data points are termed outliers. This also illustrates the symbioses there exists between the classification of data into inliers and outliers and the model that is fitted to the inliers. Explicitly, the data point classification into inliers and outlier is determined by the model, and the model is given by the data points defined as inliers.

5.3.2 Formulation of the Ransac Algorithm

The essence of the Ransac algorithm is a method for determining which data points are inliers by randomly proposing points in the Hough space to compute. Based on these random places in the Hough space, it then determines the model with the highest support in the data as the correct model. In relation to Ransac terminology a models support is termed consensus.

A cleaver thing about the Ransac algorithm is how it randomly draw's points to be evaluated in the Hough space. It does so by random sampling in the data, specifically it randomly draws a small set of data points, $\mathbf{d}_i$, to which it fits a model, i.e. it estimates the ϕ in the $\mathcal{M}(\phi)$. The aim is to get a data sample such that all the elements are inliers, for then the correct point in Hough space is computed. To increase the possibility of getting all inliers, the sample size, S, should be as small as possible, while still being large enough to estimate ϕ. So e.g. for a line S equals two 2D points, for a circle S equals three 2D points and for a plane in 3D S equals three 3D points.

To summarize; based on these samples, models are computed, and the consensus or support is computed for these models. This corresponds to finding a place in the Hough space ϕ, and computing the score. This process is done N times or iterations, and the ϕ with the highest consensus is chosen. The algorithm is outlined in Table 5.3. A last point for the general algorithm is that even for inliers some noise must be must be expected. This fact should be taken into consideration when computing the consensus, in that a perfect fit to the model should not be required. Instead data points, $\mathbf{d}_i$, within a certain distance of the model should be used as support when computing consensus, i.e. count the number of data points, $\mathbf{d_i}$, for which

$$\text{dist}(\mathcal{M}(\phi), \mathbf{d}_i) < T \ ,$$

for some threshold, T, and distance function appropriate for the model class and assumed noise structure.

5.3.3 Example: Robust Line Fitting

As an example of the Ransac algorithm consider the data in Figure 5.9, to which we want to fit a line robustly. Ten iterations are uses, i.e. $N = 10$, and each of the iterations are illustrated in Figure 5.11 and Figure 5.12. The size of the minimal set, S, is equal to two, since it takes two 2D points to define a line. The distance function used is the distance from point to line, cf. Section 1.1, and the threshold, T, is set to one. From Figure 5.12, it is seen that a maximum number of 30 inliers are found, which happens in iteration five. The result is seen in Figure 5.10

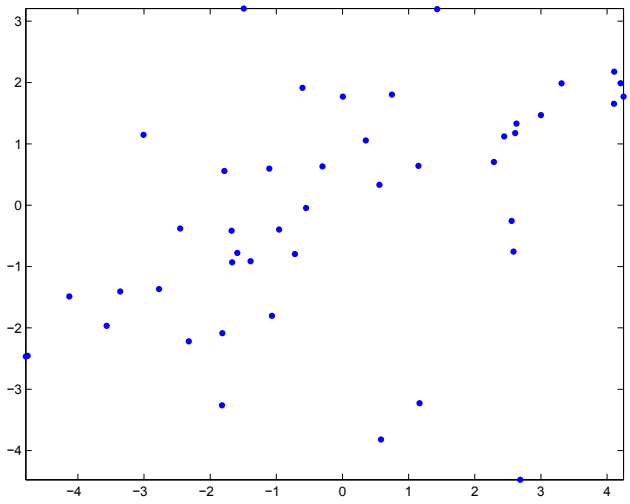


Figure 5.9: A data set of 45 2D points, to which we wish to fit a line.

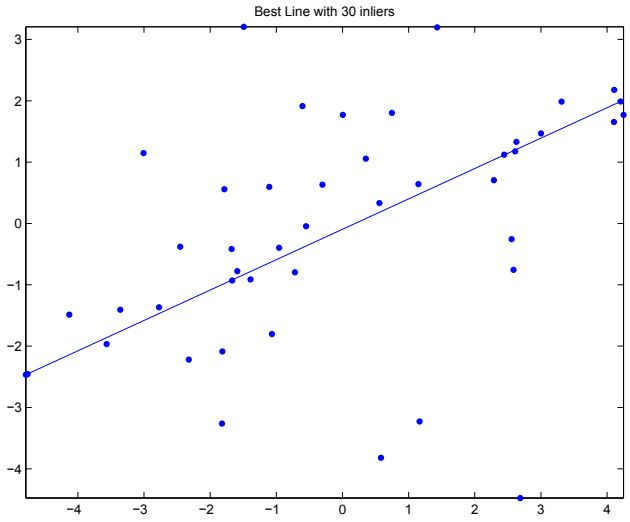


Figure 5.10: The solution to the problem in Figure 5.9, using ten iterations of Ransac as illustrated in Figure 5.11 and Figure 5.12.

5.3.4 Determining Required Number of Iterations N

An issue with using the Ransac algorithm is how many iterations one should use. The goal is to pick a subset of the data points which only contains inliers, so the question transforms to how many samples should one take in order to be sure to pick a sample containing only inliers. A simple answer would be to try out every possible solution, which will however, in most cases be computationally infeasible, and quite unnecessary.

	Probability of outlier ξ							
S	0.05	0.1	0.2	0.25	0.3	0.4	0.5	0.6
2	2	3	5	6	7	11	17	27
3	3	4	7	9	11	19	35	70
4	3	5	9	13	17	34	72	178
5	4	6	12	17	26	57	146	448
6	4	7	16	24	37	97	293	1123
7	4	8	20	33	54	163	588	2809
8	5	9	26	44	78	272	1177	7025

Table 5.4: Tabulation of some values for (5.5), for $p = 0.99$. If $p = 0.9999$ instead, these values should be multiplied by two since this is the ratio between $\log(1 - 0.9999)$ and $\log(1 - 0.99)$, which is the numerator of (5.5).

Another possibility, as used in the example of Section 5.3.3, is to make an educated guess. This, however, is somewhat unsatisfactory, and will make the algorithm less general, in that a new N should be supplied for each application.

A good solution for determining the number of needed iterations, [Hartley and Zisserman, 2003], is to assume that we know the probability of outliers in the data set[1], ξ. By requiring that with the probability of p we should pick at least one sample containing only inliers, a probabilistic calculation implies that the Ransac algorithm should run for N iterations, with

$$N = \left\lceil \frac{\log(1-p)}{\log(1-(1-\xi)^S)} \right\rceil \quad , \tag{5.5}$$

where S is the sample size (e.g. two for lines seven for the fundamental matrix). Typically p is set very close to one, with $p = 0.99$ being a typically used value. Here $\lceil \cdot \rceil$ denotes rounding up to the nearest integer. To give a feeling for the values of N some have been tabulated in Table 5.4. From this table it is seen that for a high percentage of outliers a lot of iterations, N, are needed. There are also reports that Ransac seems to work well until ca. 50% outliers – which is *probably* a good rule of thumb, and naturally depends highly on the application.

The calculations leading to (5.5), are as follows; since ξ is the probability of an outlier then $(1 - \xi)$ must be the probability of an inlier, and the probability of drawing S inliers must be $(1 - \xi)^S$. It follows that the probability of *not* drawing S inliers, i.e. drawing a sample with at least on outlier, must then be $(1 - (1 - \xi)^S)$. The probability of having such 'failures' happening N times is thus

$$(1 - (1 - \xi)^S)^N \quad ,$$

which corresponds to the probability that we will not succeed, i.e. not drawing a single sample with only inliers. In other words this is the probability of failure, which should equal $(1 - p)$, implying that

$$\begin{aligned} (1 - (1 - \xi)^S)^N &= (1 - p) \Rightarrow \\ \log\left((1 - (1 - \xi)^S)^N\right) &= \log(1 - p) \Rightarrow \\ N \log\left((1 - (1 - \xi)^S)\right) &= \log(1 - p) \Rightarrow \\ N &= \frac{\log(1-p)}{\log\left((1 - (1 - \xi)^S)\right)} \quad , \end{aligned}$$

which is equal to (5.5), apart from $\lceil \cdot \rceil$, which is needed in that we can only have integer values for N.

The method of (5.5) for determining the number of iterations N, requires an estimate of the probability or fraction of outliers ξ. Such an estimate is often not available. The Ransac itself algorithm, however, supplies such an upper bound estimate ξ, in that it keeps track of the largest consensus set found. This consensus set is a lower bound estimate of the number of inliers, which is adaptively increased as the algorithm runs. Thus an adaptive upper bound estimate, $\hat{\xi}$ of ξ is given by

$$\hat{\xi} = 1 - \frac{\text{Highest Consensus}}{M} \quad , \tag{5.6}$$

[1] A method for estimating this will be presented shortly.

Given a model class $\mathcal{M}(\phi)$, a set of data points, $\mathbf{d}_i$, $i \in \{1, \ldots, M\}$ and a threshold T, do the following:

- Initialize $\hat{N}$ to ∞
- Iterate until the number of iterations exceeds $\hat{N}$:
  1. **Randomly Draw a Sample** of size S from $\mathbf{d}_i$, $i \in \{1, \ldots, M\}$, where each data point is selected with equal probability.
  2. **Estimate a Model** based on the sample estimate a model, i.e. ϕ of $\mathcal{M}(\phi)$.
  3. **Compute the Consensus** of the model ϕ by counting how many of the M data points for which
  $$\text{dist}(\mathcal{M}(\phi), \mathbf{d}_i) < T \ , \tag{5.7}$$
  for some distance function 'dist'.
  4. **If a Better Model is Found** update the estimate of $\hat{N}$, by (5.6) into (5.5)

The model parameters ϕ with the highest consensus is the resulting model. The sample size S should be minimal, in the sense that it should be large enough to estimate ϕ, but no larger.

Table 5.5: Pseudo code for the adaptive Ransac algorithm.

which is again used to make an upper bound on the required number of iterations, $\hat{N}$, by plugging (5.6) into (5.5). The Ransac algorithm is then run until it performs a number of iterations exceeding $\hat{N}$. The adaptive Ransac algorithm is summarized in Table 5.5. It is noted, that if we are unlucky, and the Ransac algorithm does not pick a sample consisting of only inliers within the N iterations according to (5.5), then the best consensus set is likely to be poor and $\hat{N}$ is likely to be much larger then N. This will result in more iterations, and a higher probability of finding a 'good' sample.

Example from Section 5.3.3 Continued

As an example of estimating $\hat{N}$ with the proposed method, consider the above example from Section 5.3.3. Here there is a total of $M = 45$ data points, and the largest consensus set is 30 inliers, after iteration five. So according to (5.6)

$$\hat{\xi} = 1 - \frac{\text{Highest Consensus}}{M} = 1 - \frac{30}{45} = 0.3333 \ ,$$

and according to (5.5), with $p = 0.99$ and $S = 2$

$$\begin{aligned} \hat{N} &= \left\lceil \frac{\log(1-p)}{\log(1-(1-\xi)^S)} \right\rceil \\ &= \left\lceil \frac{\log(1-0.99)}{\log(1-(1-0.3333)^2)} \right\rceil \\ &= \lceil 7.8337 \rceil = 8 \ . \end{aligned}$$

This would be the estimate of $\hat{N}$ after iteration five. After iteration four the largest consensus set was equal to 15, achieved in the third iteration, implying that (with the same values for p and s)

$$\hat{\xi} = 1 - \frac{\text{Highest Consensus}}{M} = 1 - \frac{15}{45} = 0.6667 \ ,$$

implying that

$$\hat{N} = \left\lceil \frac{\log(1-0.99)}{\log(1-(1-0.6667)^2)} \right\rceil = \lceil 39.1071 \rceil = 40 \ .$$

In this examples the values of $\hat{N}$ after each iteration are given in Table 5.6.

Iteration:	0	1	2	3	4	5	6	7	8	9	10
$\hat{N}$	∞	63	63	40	40	8	8	8	8	8	8

Table 5.6: The estimate of $\hat{N}$ after each iteration in the example from Section 5.3.3, using the described adaptive method. Note that the algorithm would have stopped at iteration eight.

Model	dist$(\cdot)$	Codimension	T^2
Line	(1.12)	1	$3.84\sigma^2$
Fundamental Matrix	(2.40)	1	$3.84\sigma^2$
Essential Matrix	(2.40)	1	$3.84\sigma^2$
Homography	(2.45)	2	$5.99\sigma^2$
Camera Matrix	$(1.33)^2$	2	$5.99\sigma^2$

Table 5.7: Summary of how to adapt various view geometric models to Ransac, with a listing of a recommended distance measure, the codimension of the problem, and the relation between gaussian image noise with a standard variation of σ and the threshold value.

5.3.5 Relation to View Geometry

One of the main applications of the Ransac algorithm is in fitting view geometric models, cf. Chapter 2 to tracked features, as described in Chapter 6, in particular Section 6.4 to which the reader is referred for more examples on the use of Ransac. This is used e.g. to regularize feature tracing and is often required to construct a reliable feature tracker. Thus some special issue regarding the use of the Ransac algorithm to fit view geometric models will be covered here.

Firstly in relation to the distance measure dist$(\mathcal{M}(\phi), \mathbf{d}_i)$ needed in Table 5.3 and Table 5.5. Here the distance measures referred to in Table 5.7 are recommended. Other viable alternatives and models exist are sometimes preferred as discussed and presented in [Hartley and Zisserman, 2003].

A second issue, is regarding the thresholds T, for determining if an data point is an inlier or an outlier. It this regard there is often a lot of educated guessing taking place in regards to setting these threshold. If, however, the data points consists of pairs of 2D points with gaussian noise of variance σ^2 added, then the distances should be compared to a chi-squared distribution χ^2_m distribution where m is the codimension of the geometric model [Hartley and Zisserman, 2003]. Using a 95% confidence level the squared threshold and codimensions of the models are given in Table 5.7. The decision rule is thus that a point is an inlier if

$$\text{dist}(\mathcal{M}(\phi), \mathbf{d}_i)^2 < T^2 \quad . \tag{5.8}$$

5.3.6 Further Discussion

Two further points about the Ransac algorithm should be made at this point. Firstly, since the individual model estimates, $\mathcal{M}(\phi)$, to minimize S and thereby the needed number of samples N, there is no redundancy in the data to which the model is fitted. This in turn implies that there is no noise reduction, and the expected noise on the inliers will not be reduced. This can be seen in the example of Section 5.3.3, where the estimate of inliers is good, but the estimated line does not seem to be the best representation of these inliers. As such the Ransac algorithm should be followed by a polishing step, where either a standard non-robust estimation algorithm is used on the estimated inliers. Alternatively, and usually better, the result of the Ransac algorithm should be used as a starting guess to iterative optimization algorithms based on robust statistics, as covered in the next Section 5.4. Secondly the computational efficiency of the Ransac algorithm compared to the generalized Hough transform should be touched upon. Considering Table 5.4, it is seen that for a modest amount of outliers, Ransac can deal with models of a considerable higher dimension then the Hough transform.

5.4 Robust Statistics

In many statistic estimation problems the data contains outliers or extreme values. These troublesome data points originate in all from outright errors as in the example of Table 5.2, to the probability distribution function of the data having fatter tails then the assumed. In this regard the assumed probability distribution

function is usually Gaussian, in part because it implies easier computations. The field of robust statistics is a branch of statistics, aimed at dealing with these problems cf. e.g. [Huber, 1981], [Maronna et al., 2006], [Triggs et al., 2000]. Apart from the examples illustrated in this chapter, where image features and/or two view geometry is the subject, there is a multitude of other cases where robust statistics is an appropriate tool. Examples hereof include estimation of 3D models from hundreds of images as well as fitting sparse models used in e.g. image compression.

The basic approach in robust statistics is to use methods that are less sensitive to extreme values, if not completely insensitive like in the Ransac and Hough methods. This is to be seen in contrast to fitting by minimizing the squared residual error, which is the error function implied by assuming Gaussian distribution of the noise. This least squared error is very sensitive to extreme value, and as such other norms are considered, cf. Section 5.4.1. This, however, induces another issue namely that of computing solutions, in that one reason Gaussian noise is often assumed is that it induces computationally nice algorithms.

5.4.1 Robust Norms

The way statistical methods are made robust towards extreme values, is by using other norms for evaluating the residual error than the standard 2-norm. To formalize, assume that we want to fit a model $\mathcal{M}(\phi)$ to a set of data points, $\mathbf{d}_i$ $i \in \{1, \ldots, M\}$. Then the standard way of doing this is by minimizing the summed squared error, i.e.

$$\phi = \min_{\phi} \sum_{i=1}^{M} \text{dist}(\mathcal{M}(\phi), \mathbf{d}_i)^2 \quad , \tag{5.9}$$

for some appropriate distance function. As an example consider fitting a mean to a set of scalar values, as e.g. in Table 5.2, where $\mathcal{M}(\phi) = \mu$, (i.e. ϕ is set to μ to follow convention) the $\mathbf{d_i}$ are scalars and an appropriate distance function is

$$\text{dist}(\mathcal{M}(\phi), \mathbf{d}_i) = \frac{\mathbf{d}_i - \mu}{k} \quad , \tag{5.10}$$

where k is associated with the standard deviation or scale of the measurements. Then (5.9), becomes

$$\phi = \mu = \min_{\mu} \sum_{i=1}^{M} \left(\frac{\mathbf{d}_i - \mu}{k} \right)^2 \quad .$$

In robust statistics equation (5.9) is generalized to

$$\phi = \min_{\phi} \sum_{i=1}^{M} \rho \left(\text{dist}(\mathcal{M}(\phi), \mathbf{d}_i) \right) \quad , \tag{5.11}$$

where the $\rho(\cdot) : \Re \rightarrow \Re$ is the norm or error function. In the standard sum of squared errors case $\rho(x) = (x)^2$. It is this $\rho(\cdot)$ function that is modified in order to make the estimates more robust.

To illustrate the problem with the 2-norm in the face of extreme data, consider again fitting a mean value to the data of Table 5.2. In Table 5.8 the individual terms of the sum in (5.11) are tabulated for a $\mu = 179.45$, which coincides with the mean value without the outlier, i.e. measurement 10, and as such also an approximate value for the mean value we would want to estimate. In Table 5.8 it is seen that the outlier measurement overwhelms the estimation process by the weight it has in (5.11). More importantly this overwhelming impact is also present in the derivative, which denotes the gain or loss by moving closer to that observation. It is this extreme sensitivity to outliers we wish to address by using other $\rho(\cdot)$ functions.

Four of the most common alternative robust norms $\rho(\cdot)$, to the 2-norm are the 1-norm the Huber norm and the Cauchy norm, which will be covered briefly here[3]. The graph of the norms are found in Figure 5.13, and the extension of Table 5.8 to these alternative norms is found in Table 5.9. In this Table it should be noted the difference in sensitivity of the norms to the outlier. To finish of this example, the estimated mean with the different norms is shown in Table 5.10.

[3] Strictly speaking, some of these error measures are not norm in a mathematical sense, in that they do not fulfill the triangle inequality.

Person	$\mathbf{d_i}$	$\frac{\mathbf{d}_i-\mu}{k} = \frac{\mathbf{d}_i-179.45}{10}$	$\left(\frac{\mathbf{d}_i-\mu}{k}\right)^2$	$\frac{\partial}{\partial\mu}\left(\frac{\mathbf{d}_i-\mu}{k}\right)^2$
1.	194	1.45	2.12	-2.91
2.	192	1.25	1.57	-2.51
3.	181	0.15	0.02	-0.31
4.	172	-0.75	0.56	1.49
5.	173	-0.65	0.42	1.29
6.	173	-0.65	0.42	1.29
7.	184	0.45	0.21	-0.91
8.	173	-0.65	0.42	1.29
9.	170	-0.95	0.89	1.89
10.	**1.81**	**-17.76**	**315.58**	**35.53**
11.	170	-0.95	0.89	1.89
12.	192	1.25	1.57	-2.51

Table 5.8: A tabulation of the individual terms of (5.11) for the data in Table 5.2, as well as the derivative wrt. μ. Not how the outlier measurement 10 overwhelms the estimation. In this example k has been set to ten.

Person	2-Norm		1-Norm		Truncate Quadratic		Huber Norm		Cauchy Norm	
	ρ	$\frac{\partial\rho}{\partial\mu}$	ρ	$\frac{\partial\rho}{\partial\mu}$	ρ	$\frac{\partial\rho}{\partial\mu}$	ρ	$\frac{\partial\rho}{\partial\mu}$	ρ	$\frac{\partial\rho}{\partial\mu}$
1.	2.12	2.91	1.45	1.00	1.00	0.00	1.91	1.00	1.14	0.93
2.	1.57	2.51	1.25	1.00	1.00	0.00	1.51	1.00	0.95	0.97
3.	0.02	0.31	0.15	1.00	0.02	0.31	0.02	0.31	0.02	0.30
4.	0.56	-1.49	0.75	-1.00	0.56	-1.49	0.56	-1.49	0.44	-0.96
5.	0.42	-1.29	0.65	-1.00	0.42	-1.29	0.42	-1.29	0.35	-0.91
6.	0.42	-1.29	0.65	-1.00	0.42	-1.29	0.42	-1.29	0.35	-0.91
7.	0.21	0.91	0.45	1.00	0.21	0.91	0.21	0.91	0.19	0.75
8.	0.42	-1.29	0.65	-1.00	0.42	-1.29	0.42	-1.29	0.35	-0.91
9.	0.89	-1.89	0.95	-1.00	0.89	-1.89	0.89	-1.89	0.64	-1.00
10.	**315.58**	**-35.53**	**17.76**	**-1.00**	**1.00**	**0.00**	**34.53**	**-1.00**	**5.76**	**-0.11**
11.	0.89	-1.89	0.95	-1.00	0.89	-1.89	0.89	-1.89	0.64	-1.00
12.	1.57	2.51	1.25	1.00	1.00	0.00	1.51	1.00	0.95	0.97

Table 5.9: Expansion of Table 5.8 for the five robust norms presented. Here the last two column from Table 5.8 are repeated for each norm. Note that measurement 10. is the outlier.

Norm	Estimate
2-Norm	164.65
2-Norm (without outlier)	179.45
1-Norm (= median)	173
Truncated Quadratic	176.80
Huber Norm	177.15
Cauchy Norm	177.51

Table 5.10: Mean estimate for the data in Table 5.2, as a result of using the different robust norm mentioned here.

The 1-Norm

The 1-norm is equivalent to the absolute value, i.e.

$$\rho(x) = |x| \quad . \tag{5.12}$$

It grows much slower than the 2-norm, as illustrated in Table 5.9, and often computationally efficient algorithms using the 1-norm can be realized via linear programming. It can also be shown, as is done bellow, that using the one norm is related to and in many cases equivalent to taking the median, which is another popular approach for achieving robustness. A drawback of the 1-norm is that it is not differentiable everywhere, which will give some optimization approaches trouble. The 1-norm is also associated with sparse representation and the lasso approach [Hastie et al., 2001].

Truncated Quadratic

The truncated quadratic is given by

$$\rho(x) = \begin{cases} x^2 & x <= k \\ k^2 & x > k \end{cases} \quad . \tag{5.13}$$

As seen from Figure 5.13, this function corresponds to a 2-norm – consistent with a Gaussian error distribution – for data points with dist$(\mathcal{M}(\phi), \mathbf{d}_i) < k$, and is flat for data points with a larger distance. This is equivalent to ignoring all data points with dist$(\mathcal{M}(\phi), \mathbf{d}_i) > k$ since the first derivative is zero, and then there is no 'benefit' wrt. (5.11) in decreasing dist$(\mathcal{M}(\phi), \mathbf{d}_i)$ – unless that distance becomes less than k. Thus the truncated quadratic is equivalent to classifying all data points with distances larger than k as outliers and discarding them. This robust norm is thus in some sense equivalent to the norms used in the Ransac and Hough algorithms above.

A problem with the truncated quadratic is it's hard distinction between how inliers and outliers are determined, in that ones certainty in the choice of k is seldom very high. This is also illustrated in Table 5.9, where two of the inliers are termed as outliers, and as such the estimate is somewhat of, as seen in Table 5.10. An advantage, compared to the Huber and Cauchy norms, which to some extend address the certainty of k issue, is that the truncated quadratic is typically easier to implement.

Importance of Scale k

It is noted that there is two ways of including the scale parameter k, either as done in (5.10) and in the example of Table 5.8, where the distance is divided by it. In this case the k in (5.13) is set to one. Alternatively, the distance is not divided by k before the $\rho(\cdot)$ function is applied. In this case the scale is applied as indicated in (5.13).

Setting the scale parameter k as used in the truncated quadratic as well as the Huber and Cauchy norms, is an important aspect of using the robust norms. The reason being that it determines what is outlier and what are inliers and is as such highly related to the threshold T in the Ransac algorithm, and some of the same considerations apply. This is also the case for the Huber and Cauchy norms, albeit they do not totally discard 'outliers' but just gradually weigh them down.

Huber Norm

The Huber norm, also known as the Huber M-estimator [Huber, 1981], is depicted in Figure 5.13 and given by

$$\rho(x) = \begin{cases} x^2 & x <= k \\ 2k|x| - k^2 & x > k \end{cases} \quad . \tag{5.14}$$

There is a lot of theory related to mixture models associated to the Huber norm [Huber, 1981]. Other interpretations is that it addresses some of the issues of the 1-norm and the truncated quadratic. The Huber norm can be seen as a 1-norm that has been smoothed in the bottom, such that it is differentiable everywhere, and that it behaves as a least squares function for inliers. The latter is consistent with a Gaussian noise distribution. In relation to the truncated quadratic it is borderline convex, making for better performance of optimization algorithms, furthermore outliers still have some influence such that the choice of k is less harsh.

Cauchy Norm

The Cauchy norm is very popular in Bundle adjustment [Triggs et al., 2000], and is given by

$$\rho(x) = \log\left(1 + \frac{x^2}{k^2}\right) \quad . \tag{5.15}$$

This norm has many of the properties of the Huber norm, but as seen in Figure 5.13, the effect of outliers gradually decrease. In fact

$$\frac{\partial}{\partial x}\rho(x) \to 0 \qquad \text{as} \qquad x \to \infty \quad .$$

Equivalence of 1-Norm and Median*

In the scalar case as illustrated in the example of Table 5.2, using the one norm is equivalent to taking the median. To see this consider the error surface, $f(\mu)$, of (5.11), in this case

$$\begin{aligned} f(\mu) &= \sum_{i=1}^{M} |\mathbf{d_i} - \mu| \\ \frac{\partial}{\partial \mu} f(\mu) &= \sum_{i=1}^{M} \text{sign}(\mathbf{d_i} - \mu) \end{aligned}$$

Consider what happens with $\frac{\partial}{\partial \mu} f(\mu)$ as μ starts a a value lower then all the $\mathbf{d_i}$ and continuously gets larger until it has a value larger than all the $\mathbf{d_i}$. In this process $\frac{\partial}{\partial \mu} f(\mu)$ only changes value by two when it passes a data point, else it is constant. Secondly $\frac{\partial}{\partial \mu} f(\mu)$ is $2M$ when it is lower than all data points and $-2M$ when it is larger than all data points. So gradually increasing μ from a value which is smaller than all data points, $\frac{\partial}{\partial \mu} f(\mu)$ will be zero after passing half the data points. Thus it is after half the data points that $f(\mu)$ will achieve its minimum, and this minimum corresponds to the median.

5.5 End Notes

In this chapter the very popular methods of the Hough transform and Ransac have been presented. Also the general field of Robust statistics have been briefly touched upon. This field is rather extensive cf. e.g. [Hartley and Zisserman, 2003], [Huber, 1981], [Maronna et al., 2006] to give a few pointers, and is used extensively within computer vision, especially in more advanced topics than covered in this text, e.g. [Black and Rangarajan, 19 [Triggs et al., 2000]. This subject is also highly linked with optimization, since using robust statistical techniques often results in complicated optimization problems cf. e.g. [Boyd and Vandenberghe, 2004].

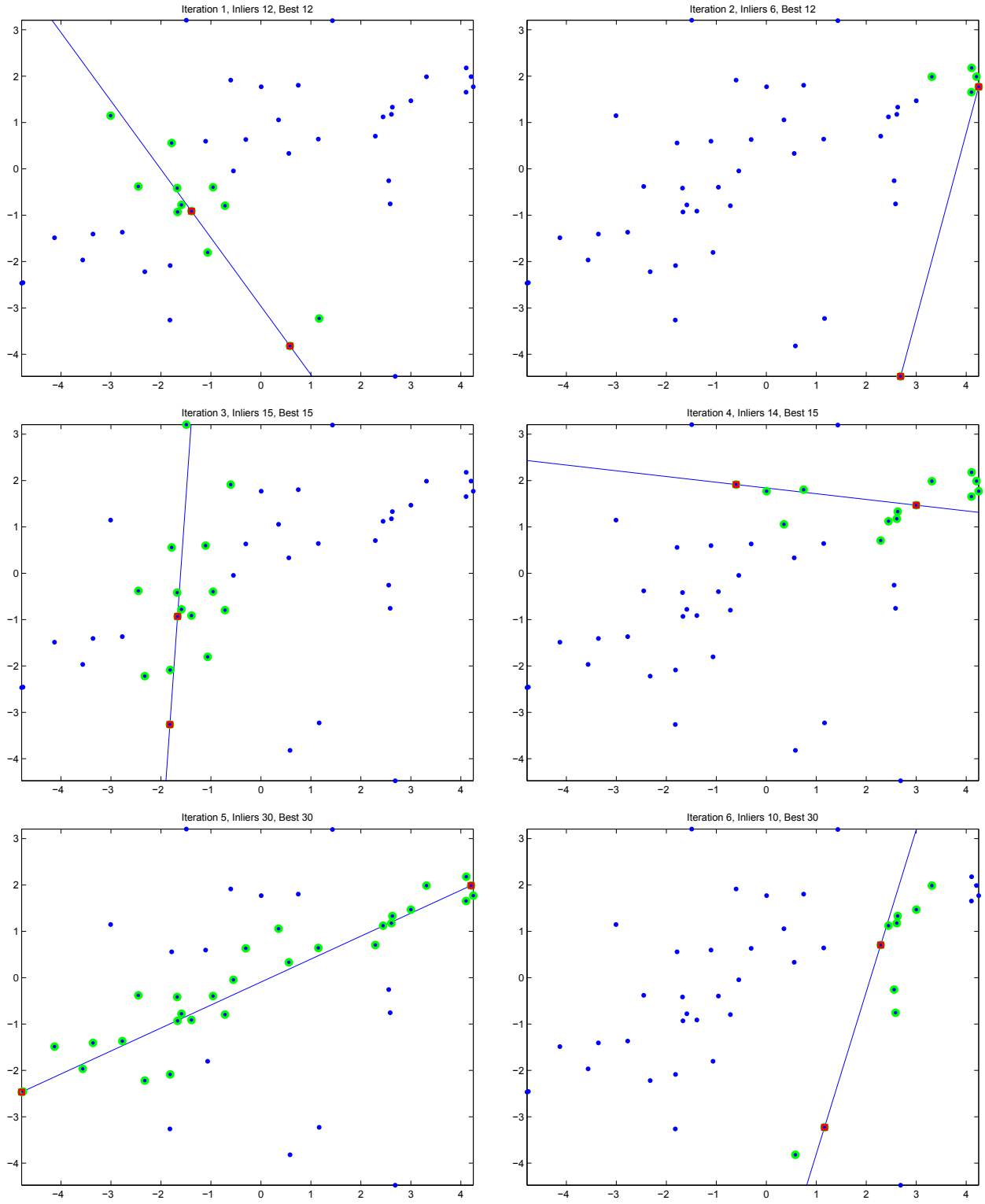


Figure 5.11: The six first iterations of the Ransac algorithm, on the problem from Figure 5.9, the last four iterations are found in Figure 5.12. The blue dots denote the data points. The red squares denotes the two points selected as the minimal sample, which the model in the shape of the blue line is fitted to. The dots with green circles are the inliers w.r.t. the model, or the consensus set. The title of each of the sub-figures denotes the iteration number, the number of inliers, and the largest number of inliers so far.

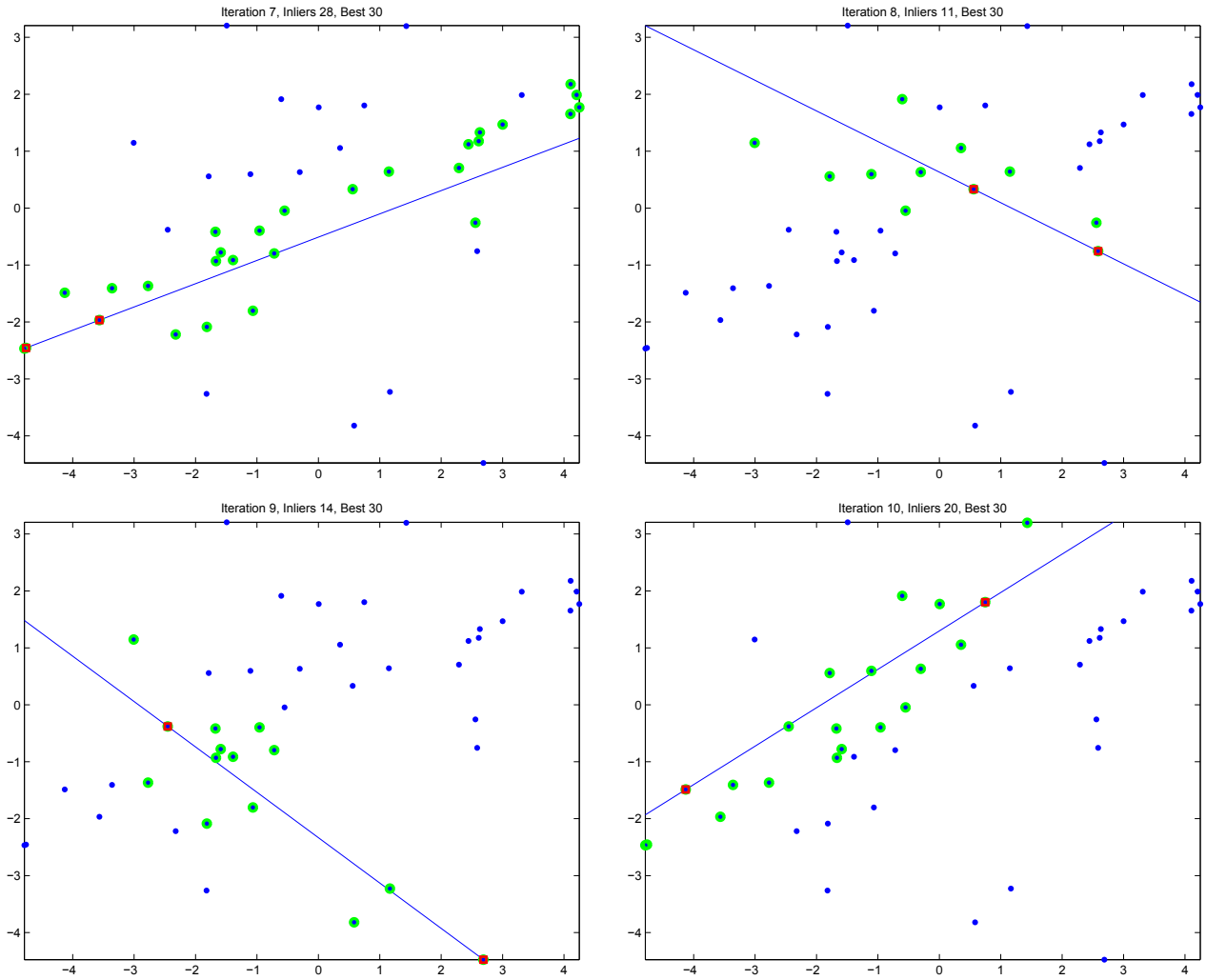


Figure 5.12: The continuation of Figure 5.11, with the last four iterations. The graphical notation is the same.

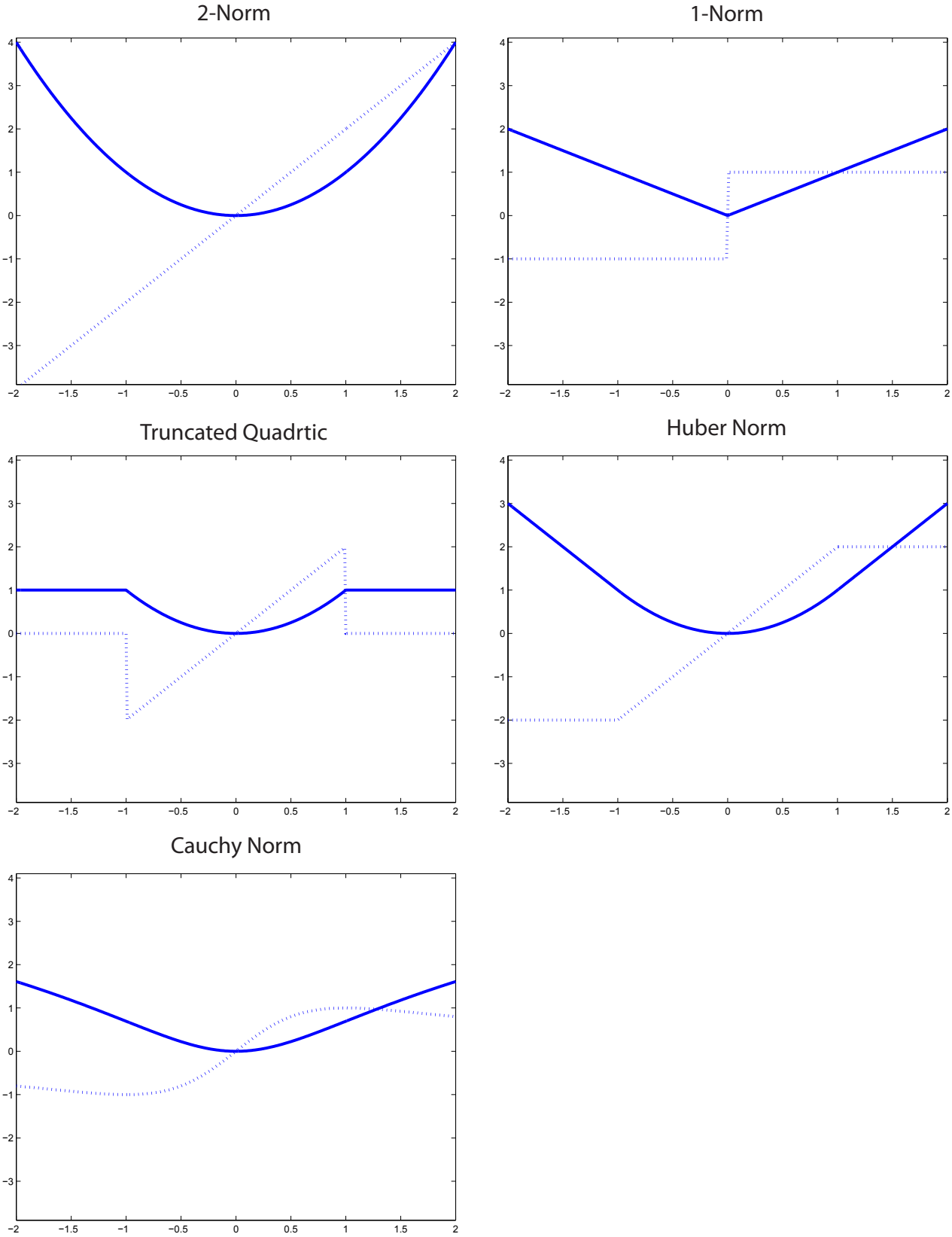


Figure 5.13: Example of robust norms (solid line), $\rho(\cdot)$, and their derivative (dotted line).

Chapter 6

Feature Based Image Correspondence

We often wish to do image analysis on more than one image, e.g. if we want to make computations on how events evolve over time, if we want to do 3D inference by use of view geometry as is the subject of Chapter 2, or if we want to search in image data bases and/or do object recognition. This typically involves finding the *correspondence* between images by solving the so called *correspondence problem*; *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 6.1. This is the subject of this chapter, where we take a feature based approach, consisting of first extracting salient features from the images in question, as described in Chapter 4, and then attempt to compute which of these extracted features match to which. This approach is also only carried out between pairs of images. If the correspondence between more than two images is sought, as is often the case, the solution from image pairs is aggregated to the entire set of images in question.

The image correspondence problem 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 implies that it is also known under a lot of different names e.g. *tracking*, *feature tracking*, *registration* and *feature matching* to mention a few. Also many other methods than the ones presented here exist, both feature based, cf. e.g. [Mikolajczyk and Schmid, 2005],[Tuytelaars and Mikolajczyk, 2008], and non-feature based. As for the latter these are typically area based such as optical flow, e.g. [Black and Rangarajan, 1996] [Horn and Schunk, 1981], and stereo, e.g. [Cox et al., 1996] [Roy and Cox, 1998] [Scharstein and Szeliski, 2002], where every pixel in the images is attempted matched, instead of just salient features as is done here. An outline of this chapter is as follows; firstly the general approach of feature based image correspondence is discussed in Section 6.1, upon which a select number of feature descriptors are presented in Section 6.2. These feature descriptors can be seen as an attempt to make a describing finger print of each feature, based on which as score of similarity can be computed. In the following Section 6.3 methods for computing matches based on these similarity scores are presented, and the chapter ends up with Section 6.4 where methods for regularizing or aiding the feature matching with view geometric methods are presented.

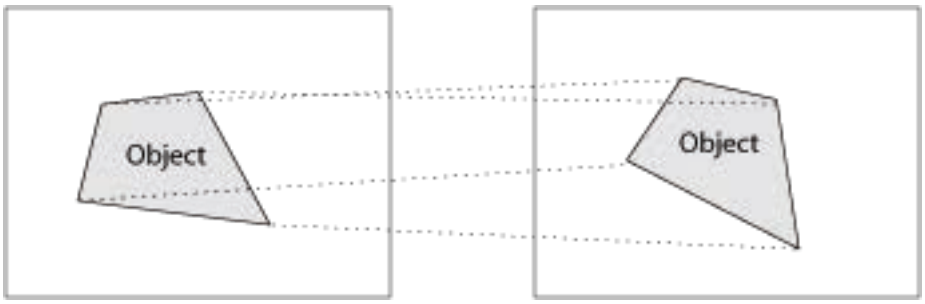


Figure 6.1: The general correspondence problem, where the relationships between the same entities of an object are mapped. Here the corners of the object correspond.

6.1 Correspondence via Feature Matching

As outlined above, the approach to image correspondence taken here is that of feature matching. This approach is taken because it is commonly used and has proven successful for many applications. Feature matching can be seen as a three stage process consisting of:

1. Extract a number of, hopefully salient, feature from the images. Here it is assumed that these features are points, cf. Chapter 4. Other types of features, e.g. lines, have been used, but the framework needed is much more complex, and might not be as robust in general.

2. For each feature compute or extract a descriptor, cf. Section 6.2. This descriptor is used as a distinguishing representation of the feature, and is usually based on a small patch , $\mathcal{P}$, around the feature point.

3. The correspondence between the two sets of features, and thus the images, is found by pairing features with similar descriptors. There are several strategies for doing this paring, which among others should ensure that one feature only has one correspondence. This is discussed in Section 6.3.

See Figure 6.2, for an schematic overview of this approach.

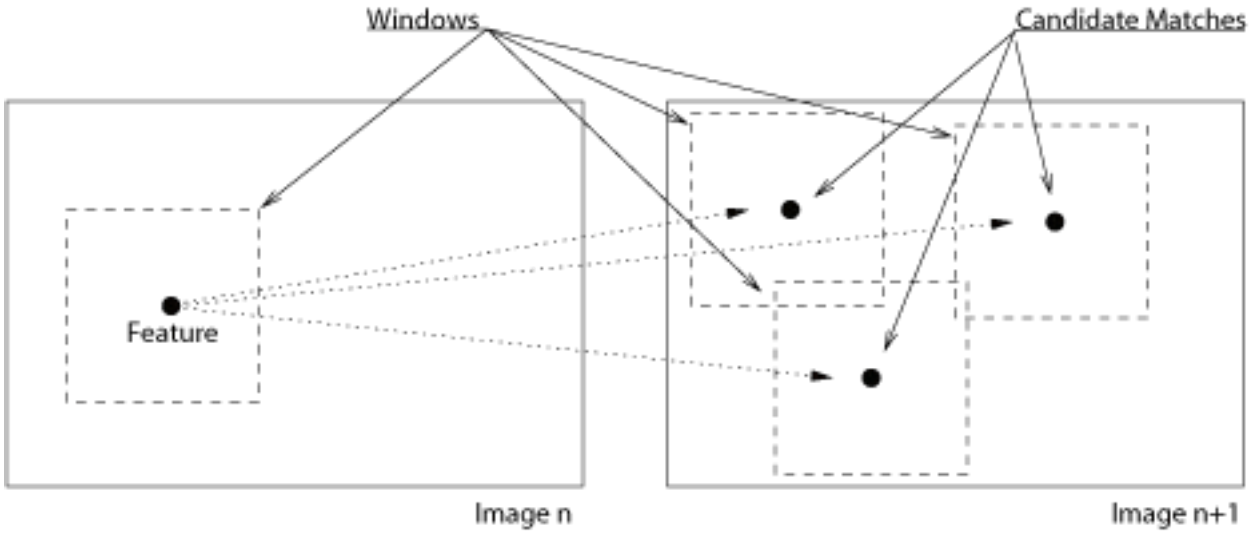


Figure 6.2: Matching features by finding the features with the most similar neighborhoods — defined by a window around the feature. Here 'Most Similar' is defined by the feature descriptors and a distance between them.

As stated, there are a multitude of ways of doing feature matching. There are however some general comments that can be made as to what in general characterizes good strategies. As for what features to extract, the features should be **unique** in the sense that it is clear where the features are. This should e.g. be seen in relation to the aperture problem, as discussed in Section 4.2. 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.

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 in both images, although two images are changed slightly relative to each other—whatever 'slightly' means. The issue is, that if the same 3D entities do not turn out as image features in both images, we can not match them. Issues in this regard are that the same thing will seldom look exactly the same in two images due to changes in illumination, view point, internal camera setting and 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 and their descriptors should be **distinguishable**, i.e. it should be able to distinguish between features via their descriptors. The motivation is the obvious that we wish to match the correct features to each other, which would be impossible if we could not distinguish one feature from the other.

In the following sections a few successful approaches to feature matching are presented, which to a reasonable extent fulfill the three criteria presented above. These strategies are, as mentioned, only a small subset of successfully used methods and are rather general in the problem domains they can address. If the problem domain is very limited, special tailored solutions can be made with success. As an example consider finding the targets in Figure 6.3, here the associated number would be a very good descriptor, and a target-like convolution kernel be a good feature extractor.

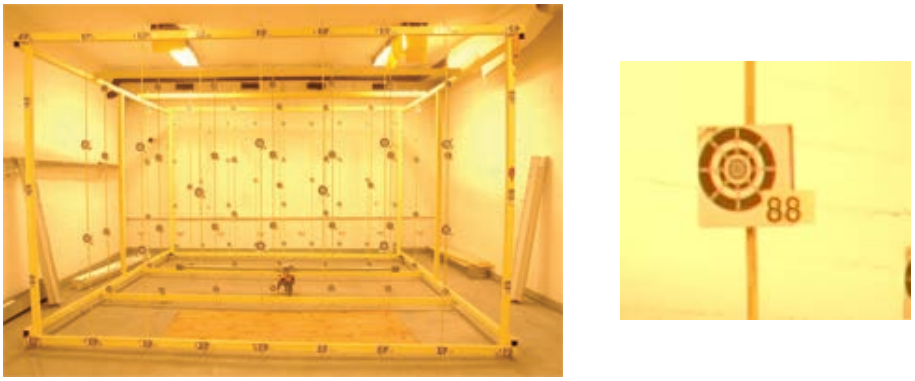


Figure 6.3: A image of a test field for calibrating cameras to the left. To the right an enlarged image section of the left image depicting the targets.

6.2 Feature Descriptors

Here we assume that the *feature descriptors* are based on a patch, $\mathcal{P}$, around the point features extracted, as illustrated in Figure 6.2. These descriptors implicitly include a metric for comparing such patches, such that a scalar can be produced that denotes how similar two patches are. for the first of the two presented descriptors the patch is the descriptor it self, and correlation is the metric. The second descriptor presented is the so called SIFT descriptor, which is at present close to the state of the art in feature matching, cf. [Dahl et al., 2011]. Some improvements to the SIFT descriptor include [Winder and Brown, 2007] and [Tola et al., 2010]. Here it is assumed that the images are gray values or single-colored. Often times this is not the case, and the images are color or RGB, in this case and the images are converted to gray scale images before matching begins.

6.2.1 Correlation

Using correlation to match point features is achieved by using the actual patches, $\mathcal{P}$, as descriptors, and then using the correlation between the pixel values as the associated metric, cf. Table 6.1. In relation to the location of the image features $\mathbf{x}$, these patches are centered on these, and are aligned with the image axis, as indicated in Figure 6.2. Using correlation can be interpreted as seeing one patch as a kernel and convolving the other image with it. It is noted that this technique is also referred to as *normalized cross correlation (NCC)* to emphasize that correlation is a normalized version of the covariance.

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 (6.1)[1] will be small or zero. In the later case (6.1) is undefined, in the first case noise will dominate. This will, however, seldom be an issue, since features with high gradients are chosen to address the aperture problem, 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 6.4.

Implementation Issues

When implementing the algorithm in Table 6.1, the data has to be traversed three times, once to compute the mean, and once to compute the variance, and once to compute the covariance. Since data access in large data

[1] Found at the bottom of Table 6.1.

To calculate the cross correlation, ρ, between to patches $\mathcal{P}_1$ and $\mathcal{P}_2$, given by a patch size $(2r+1) \times (2r+1)$ around the feature $\mathbf{x_1}$ and $\mathbf{x - 2}$

1. Get $\mathcal{P}_1$ by sampling the pixel values from the image in a square of size $(2r+1) \times (2r+1)$ around $\mathbf{x_1}$ in image 1. Do the similar thing with $\mathbf{x_2}$, $\mathcal{P}_2$ and image 2.

2. Arrange the elements of $\mathcal{P}_1$ and $\mathcal{P}_2$ into vectors $\mathbf{p}_1$ and $\mathbf{p}_2$.

3. Calculate the mean of each vector, $\mathbf{p}_1$ and $\mathbf{p}_2$, μ_1 and μ_2, i.e. :

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

4. Calculate the variance of each vector, $\mathbf{p}_1$ and $\mathbf{p}_2$, σ_1^2 and σ_2^2, i.e. :

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

5. Calculate the covariance, 'cov', between the two vectors, $\mathbf{p}_1$ and $\mathbf{p}_2$, i.e.:

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

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

$$\rho = \frac{\text{cov}}{\sqrt{\sigma_1^2 \cdot \sigma_2^2}} \quad . \tag{6.1}$$

Table 6.1: Computing the cross correlation, ρ, between two patches. Here ρ serves as the similarity score.

Figure 6.4: Three images of a building at the Technical University of Denmark. 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.

structures is often one of the bottlenecks i computational performance, this is unnecessary, since a method exists where the data only needs to be traversed once. To derive this define the variables

$$\varsigma_1 = \sum_{i=1}^{n} p_{1i} \quad , \quad \varsigma_2 = \sum_{i=1}^{n} p_{2i} \ ,$$

denoting the first moments of the data, and

$$\varsigma_{11} = \sum_{i=1}^{n} p_{1i}^2 \quad , \quad \varsigma_{12} = \sum_{i=1}^{n} p_{1i} p_{2i} \quad , \quad \varsigma_{22} = \sum_{i=1}^{n} p_{2i}^2 \ ,$$

denoting the second moments of the data. Then it is seen that the covariance can be written as

$$\begin{aligned}
\mathrm{cov}(\mathbf{p}_1, \mathbf{p}_2) &= \frac{1}{n-1} \sum_{i=1}^{n} (p_{1_i} - \mu_1)(p_{2_i} - \mu_2) \Rightarrow \\
(n-1)\mathrm{cov}(\mathbf{p}_1, \mathbf{p}_2) &= \sum_{i=1}^{n} (p_{1_i} - \mu_1)(p_{2_i} - \mu_2) \\
&= \sum_{i=1}^{n} (p_{1_i} p_{2_i} - \mu_1 p_{2_i} - p_{1_i} \mu_2 + \mu_1 \mu_2) \\
&= \sum_{i=1}^{n} p_{1_i} p_{2_i} - \sum_{i=1}^{n} \mu_1 p_{2_i} - \sum_{i=1}^{n} p_{1_i} \mu_2 + \sum_{i=1}^{n} \mu_1 \mu_2 \\
&= \varsigma_{12} - \mu_1 \sum_{i=1}^{n} p_{2_i} - \mu_2 \sum_{i=1}^{n} p_{1_i} + \mu_1 \mu_2 \sum_{i=1}^{n} 1 \\
&= \varsigma_{12} - \mu_1 \varsigma_2 - \mu_2 \varsigma_1 + \mu_1 \mu_2 n \\
&= \varsigma_{12} - \left(\frac{1}{n} \sum_{i=1}^{n} p_{1_i} \right) \varsigma_2 - \left(\frac{1}{n} \sum_{i=1}^{n} p_{2_i} \right) \varsigma_1 + \left(\frac{1}{n} \sum_{i=1}^{n} p_{1_i} \right) \left(\frac{1}{n} \sum_{i=1}^{n} p_{2_i} \right) n \\
&= \varsigma_{12} - \frac{1}{n} \varsigma_1 \varsigma_2 - \frac{1}{n} \varsigma_2 \varsigma_1 + \frac{1}{n} \varsigma_1 \frac{1}{n} \varsigma_2 n \\
&= \varsigma_{12} - \frac{1}{n} \varsigma_1 \varsigma_2 \Rightarrow \\
\mathrm{cov}(\mathbf{p}_1, \mathbf{p}_2) &= \frac{1}{n-1} \left(\varsigma_{12} - \frac{1}{n} \varsigma_1 \varsigma_2 \right) \ .
\end{aligned}$$

Since the variance of $\mathbf{p_1}$ is equal to $\text{cov}(\mathbf{p}_1, \mathbf{p}_1)$, and the variance of $\mathbf{p_2}$ equal $\text{cov}(\mathbf{p}_2, \mathbf{p}_2)$, we have

$$\begin{aligned} \text{cov}(\mathbf{p}_1, \mathbf{p}_2) &= \frac{1}{n-1}\left(\varsigma_{12} - \frac{1}{n}\varsigma_1\varsigma_2\right) \\ \sigma_1^2 &= \frac{1}{n-1}\left(\varsigma_{11} - \frac{1}{n}\varsigma_1^2\right) \\ \sigma_2^2 &= \frac{1}{n-1}\left(\varsigma_{22} - \frac{1}{n}\varsigma_2^2\right) \end{aligned}$$

Implying that

$$\rho = \frac{\text{cov}}{\sqrt{\sigma_1^2 \cdot \sigma_2^2}} = \frac{\varsigma_{12} - \frac{1}{n}\varsigma_1\varsigma_2}{\sqrt{\left(\varsigma_{11} - \frac{1}{n}\varsigma_1^2\right)\left(\varsigma_{22} - \frac{1}{n}\varsigma_2^2\right)}} \quad . \tag{6.2}$$

Since all the moments, ς_1, ς_2, ς_{11}, ς_{12}, ς_{22}, can be calculated in one pass through the data this gives a more efficient implementation of the covariance. It is noted, that this method can also be used to more efficiently calculate the variance of a large set of numbers, which at the same time computes the mean.

6.2.2 SIFT Descriptors

One of the best general purpose feature descriptors is the *SIFT descriptor*, [Lowe, 2004], which was proposed together with the SIFT feature extractor or detector, cf. Section 4.4.1, and thus bear the same name. This popular method has been patented by the author, and further improvements have been proposed, cf. e.g. [Winder and Brown, 2007] and [Tola et al., 2010]. The SIFT descriptor is a great piece of engineering and is comprised of considerable amount of stages. These stages will be covered here in some detail both to give an understanding of this important method, but also because it illustrates many of the problems and considerations relating to the correspondence problem. There are, however, many tricks to computing the SIFT descriptor in an 'optimal' way, and as such if an implementation of the SIFT features are aimed at, it is recommended considering an actual implementation, here `http://www.vlfeat.org/` is recommended. The computation of the SIFT descriptor can be divided into several stages, starting with getting a good patch, $\mathcal{P}$, of pixels. Please refer to Table 6.2 for an overview.

Getting a Good Patch $\mathcal{P}$

An important issue when designing feature descriptors, is what these descriptors should be invariant towards, i.e. which changes in the image should and should not influence the feature matching. As a relevant example; if a general purpose feature tracker is to be made, it should be taken into account that objects can vary in size between an image pair, i.e. as a result of zooming of the camera/observer moving between frames, cf. Figure 4.1. It is possible to construct size invariant descriptors by use of scale space cf. Section 4.1, as described bellow. The question is if it is desirable? In a general purpose setting it is, but if the size is known not to change (much) then being invariant towards scale, allowing the matching of mice and elephants, will be throwing away perfectly good information, and should as such ideally be avoided.

The SIFT descriptor is construed to be invariant towards scale, when it works together with the SIFT feature detector, Section 4.4.1, or any other feature detector with scale selection. The mechanism is that the detected SIFT *feature*, finds it's optimal scale, σ as part of the feature selection process. The hypothesis is then that if the image is enlarged with a factor of X[2], then the feature is found at a scale X times larger. The patches $\mathcal{P}$, are then sampled from an image at the optimal scale found by the detector, where the sampling distance for producing $\mathcal{P}$ is a function of the scale. Usually the SIFT features and SIFT descriptors are computed simultaneously, and as such images at appropriate scales for computing the descriptor are already available from the feature *detector* algorithm.

Since SIFT descriptors are aimed at being general purpose they are also aimed at being rotational invariant, e.g. a descriptor is in principle unchanged if you take an image in portrait instead of landscape mode. Again, if the rotation is known, then this property of the SIFT descriptor can be disabled or modified. A SIFT descriptor is made rotationally invariant by aligning it to the local dominant gradient around the feature point as illustrated in Figure 6.5. The concept is that if an image is rotated so is the gradient, as thus the patch will rotate with it.

[2] possibly smaller than one indicating shrinkage.

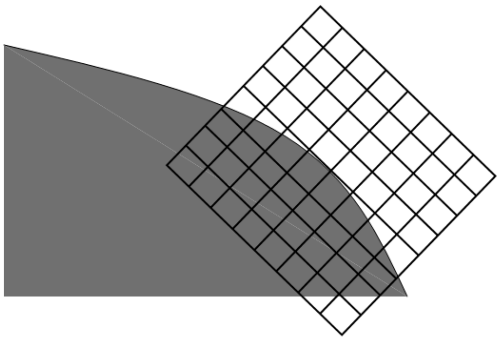


Figure 6.5: Here it is illustrated how the local patch is aligned with the local gradient in the image.

The orientation is computed by

1. Compute the modulus and argument of the image at scale σ in an appropriate area around the feature $\mathbf{x}$. A radius of 3σ is a good choice. I.e.

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

   Denote the positions in this area by $\mathbf{y}_i$ and the modulus and argument by 'mod_i' and 'arg_i'.

2. Make a 36 bin histogram of angles, corresponding to 10° each. The entries in each bin, $b(\theta)$, where θ is the appropriate angular range, is the sum

$$b(\theta) = \sum_{i:ang_i \in \theta} \text{mod}_i \exp\left(\frac{||\mathbf{y}_i - \mathbf{x}||^2}{\sigma^2}\right) \ , \tag{6.3}$$

   for pixels $\mathbf{y}_i$ with an argument in the appropriate 10° range, θ.

3. The orientation is the one corresponding to the bin with maximum value. This orientation value is improved via interpolation between the histogram bins.

4. **If** a bin exist, which has a value of more than 80% of the value of the maximum pixel, and is a local peak, two descriptors are computed for the feature, one for each orientation. In practice this is done by spawning a new feature $\tilde{\mathbf{x}}$ with the same location as $\mathbf{x}$ and a descriptor aligned with this alternative orientation.

The values of the orientation histogram, as expressed in (6.3), can be seen as a weighted averaging of the edge strengths around the feature location $\mathbf{x}$, with a gaussian weight. The reason that two descriptors are made in the special cases where two dominant orientations exist, is to avoid ambiguities. If e.g. two orientation peaks exist around a feature, then which is the largest might shift between images, and as such both orientations are utilized.

Compute a Grid of Histograms

Once a good patch, $\mathcal{P}$, is computed, the descriptor is constructed as illustrated in Figure 6.7. In particular the patch is divided into 4×4 cells[3] where an orientation histogram is calculated for each cell. In this case the orientation histograms only have eight orientation bins. The eight bins of these $16 (= 4 \times 4)$ cells are concatenated into a $128 (= 16 \cdot 8)$ bin histogram, which is the base SIFT descriptor.

These histograms are computed via the same basic formula as that for determining the orientation, namely (6.3). Implying that the edge strengths are weighted by a Gaussian weight as a function of the observations distance to $\mathbf{x}$. There is one small alteration, namely that when assigning a pixel $\mathbf{y}_i$ to a bins with center $\mathbf{q}$, its

[3] This cells are also bins and are often referred to as such. Here to help distinguish between the different bins we use the term cell for the divisions of the image.

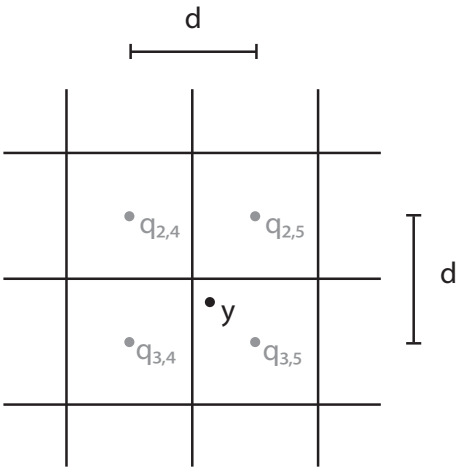


Figure 6.6: Illustration of the linear interpolation of a pixel $\mathbf{y_i}$ between its four nearest bins. The distance d denotes the spacing between the grid centers.

effect is divided among the four bins it is closest to vi linear interpolation. Thus the value of one of the eight bins with angular range θ of a cell with center $\mathbf{q}$ is given by

$$b(\mathbf{q},\theta) = \sum_{i:ang_i \in \theta} \max\left(\left(1 - \frac{||y_i - q||^2}{d^2}\right), 0\right) \mathrm{mod}_i \exp\left(\frac{||\mathbf{y}_i - \mathbf{x}||^2}{\sigma^2}\right) \quad , \tag{6.4}$$

where d is the distance between cell centers, cf. Figure 6.6. The basic SIFT descriptor, $\mathcal{D}$, is thus the concatenation of these sixteen (= four by four) histograms of eight bins each. I.e. a 128 dimension vector composed of

$$\mathcal{D} = [b(\mathbf{q}_1,\theta_1), b(\mathbf{q}_1,\theta_2), \ldots, b(\mathbf{q}_2,\theta_1), b(\mathbf{q}_2,\theta_2), \ldots] \quad . \tag{6.5}$$

This division into cells in (6.4), where the effects of the individual pixels is distributed via linear interpolation, has the effect that the SIFT descriptor, $\mathcal{D}$, is robust toward small spatial distortions of the image patch. I.e. the descriptor will not change that much if the location of the feature descriptor is wrong by one pixel, or as a result of small perspective distortion. This is an important fact in making matching with SIFT features much more repeatable and distinguishable, since such small distortions occur in real imaging applications, as a result of noise and as a change in view point.

Normalization and Truncation

Apart from view point change, another great change between images is that of illumination, and camera settings effecting the recorded illumination, such as shutter time and ISO values. To address this issue the the base SIFT descriptor is normalized, having the effect of removing an arbitrary scaling, i.e.

$$\mathcal{D} = \frac{\mathcal{D}}{||\mathcal{D}||_2} \quad . \tag{6.6}$$

To be robust towards specular effects and other types of outliers, values of $\mathcal{D}$ above 0.2 are truncated to 0.2, after normalization i.e.

$$\mathcal{D}_i = \begin{cases} \mathcal{D}_i & \mathcal{D}_i < 0.2 \\ 0.2 & \mathcal{D}_i \leq 0.2 \end{cases}$$

After this the descriptor is normalized again, thus ensuring that the length of all SIFT descriptors are one. This renormalized SIFT descriptor, $\mathcal{D}$ is the final or resulting descriptor.

Comparing SIFT Descriptors

The next question to arise is how such SIFT descriptors should be compared, i.e. how the similarity score should be computed. The most correct way of doing this is by noting, that since these 128 vectors, $\mathcal{D}$, represent

Given a feature coordinate $\mathbf{x}$ and an image smoothed to be at the same scale, σ, as the feature a SIFT descriptor is computed as follows:

1. Computet the gradient of the image at scale σ, i.e. $\frac{\partial I}{\partial x}$ and $\frac{\partial I}{\partial y}$. This calculation can be used for all descriptors at the same scale.
2. **Get the Patch** $\mathcal{P}$
   (a) Compute the local orientation at the feature $\mathbf{x}$. If several good orientation candidates are found several descriptors are computed.
   (b) Get the pixel values for the patch $\mathcal{P}$ by aligning a square of size $(2r+1) \times (2r+1)$ pixels with the computed local orientation. Here r is given by the cells size(typically $3\sigma \times 3\sigma$) and the number of cells (typically 4×4) of the descriptor.
3. **Compute Modulus and Argument of the Pixels in** $\mathcal{P}$
$$\text{mod} = \sqrt{\frac{\partial I}{\partial x}^2 + \frac{\partial I}{\partial y}^2} \quad , \quad \text{arg} = \text{atan2}(\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}) \ .$$
4. **Compute Histograms for Cells**
   (a) Divide the Patch into four by four cells of size $3\sigma \times 3\sigma$ each.
   (b) For each cell compute an eight bin histogram over the orientation, i.e. $45°$ p.r. bin. Where the entry is given by (6.4).
   (c) Concatenate these eight bin histograms from four by four cells into a $128 = 8 \cdot 4 \cdot 4$ histogram. This is the basis of the sift descriptor, $\mathcal{D}$.
5. **Normalize and Truncate** The descriptor vector $\mathcal{D}$, is normalized, i.e.
$$\mathcal{D} = \frac{\mathcal{D}}{||\mathcal{D}||_2} \ ,$$
then values larger than 0.2 truncated to 0.2, and the descriptor is normalized again.

Table 6.2: Outline of how to compute a SIFT descriptor.

histograms, they are seen to be χ^2-distributed, and as such a correct/good distance between two descriptors, a and b, is then,

$$\text{dist}(a,b) = 2\sum_{i=1}^{128} \frac{(a_i - b_i)^2}{a_i + b_i} \ . \tag{6.7}$$

Note that all entries of a and b are positive. For ease of computation, however, the 2-norm or 1-norm are often used, typically with good results. Using such norms also allows using efficient data structures such as KD-trees in the matching process.

Further Comments

As mentioned there a few subtleties left out of this presentation, for these refer to [Lowe, 2004], and/or a publicity available implementation. Also there are a lot of parameters and constants in the above presentation, such as the bin size, the number of bins, and the truncation threshold. The presented values for these are just suggestions, albeit good ones based on experiments cf. [Lowe, 2004], and can as such be changed.

6.3 Matching via Descriptors

Following the outline of Section 6.1, 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 compute a set of matches, $m_{ij} = (A_i, B_j)$, such that each element of A and B *at the most appears once*. It is noted, that the size of A and B are usually not the same.

The straight forward way of doing this is by minimizing,

$$\min \sum_{m_{ij}} \text{dist}(A_i, B_i) \ , \tag{6.8}$$

where $\text{dist}(\cdot, \cdot)$ is the appropriate metric or similarity measure. This can be cast, with a bit of manipulation, as a linear assignment problem, which have efficient solvers cf. e.g. [Jonker and Volgenant, 1987].

Another strategy which often works better, in that it removes lower quality matches, is to define

$$\text{Best}(A_i) = \min_j \text{dist}(A_i, B_j) \ , \tag{6.9}$$

i.e. $\text{Best}(A_i)$ 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

$$\text{Best}(A_i) = B_j \quad \text{and} \quad \text{Best}(A_j) = B_i \ . \tag{6.10}$$

That is A_i and B_j are each others most similar descriptors. This is the so called marriage problem. The analogy is that every man will only marry the woman he is most fond of, and every woman will only marry the man she is most fond of, so marriages will only occur if a man and a woman are each other favorites. Compared to (6.8), this also introduces symmetry into the matching process and removes some ambiguous matches.

Another method, which increases the quality of matches even higher, with the obvious trade off of getting fewer matches, is by requiring a certain level of uniqueness in the matches. This is done by defining the function

$$\text{NextBest}(A_i) = \min_j \text{dist}(A_i, B_j / \text{Best}(A_i)) \ ,$$

that is the next best match, or the best match to all features excluding the best match. Then the score

$$U(A_i) = \frac{\text{dist}(A_i, \text{Best}(A_i))}{\text{dist}(A_i, \text{NextBest}(A_i)} \ , \tag{6.11}$$

is a measure of how dominant or unique the best match is. If $\text{dist}(A_i, \text{Best}(A_i))$ is just a little smaller than $\text{dist}(A_i, \text{NextBest}(A_i)$, then it is close to ambiguous what is the best match, and $U(A_i)$ is close to one. If it, however, is clear which is the best batch then $\text{dist}(A_i, \text{Best}(A_i))$ will be much smaller than $\text{dist}(A_i, \text{NextBest}(A_i)$, and $U(A_i)$ will be much smaller than one. A good decision rule [Lowe, 2004], is to only accept a match if

$$U(A_i) = \frac{\text{dist}(A_i, \text{Best}(A_i))}{\text{dist}(A_i, \text{NextBest}(A_i)} < 0.6 \ . \tag{6.12}$$

Again the constant 0.6 can be subject of tuning, but 0.6 is by experience an acceptable value.

As for efficiency of computation, the calculations in (6.9) can be somewhat time consuming when a lot of features are present in each image. In that these calculations have to be made for all possible descriptor pairs, in the problems basic form. This is often done, however. Sometimes more efficient data structures such as KD-trees have been used, often at the cost of small approximation errors, to dramatically reduce the computational burden. In this regard especially the structures in [Arya et al., 1998] and [Muja and Lowe, 2009] have been used successfully.

6.4 Regularizing with View Geometry

In many cases the feature correspondence can be aided by the view geometry, cf. Chapter 2. As an example if the fundamental matrix were known between a pair of images, only feature pairs consistent with fundamental

matrix need to be considered. This will again greatly limit the possibilities for matches, making the chances of success much greater. See Figure 6.8.

If the fundamental matrix is unknown, but known to exist, i.e. the two images are taken at the same time, or it is known that the scene view is rigid and as such does not change over time, then the view geometry can be used to regularize the feature tracking anyway, cf. [Hartley and Zisserman, 2003]. The idea is as follows:

1. Extract features as described above, i.e. without the use of view geometry. In many cases this will contain allot of false matches as illustrated in Figure 6.8.

2. Considering the correct matches as inliers and the erroneous ones as outliers a fundamental matrix is fitted to the matched features via Ransac, cf. Section 2.8 and Section 5.3.

3. The estimated fundamental matrix computed via Ransac, is improved via the use of Robust norms.

4. The feature extraction is redone now constrained with the estimated fundamental matrix.

As an example see Figure 6.8. Naturally other models like the Homography or the essential matrix are used if they are appropriate. This approach of using two view geometry to regularize feature tracking with the help of robust model fitting, is instrumental in making general purpose tracking work in most practical cases.

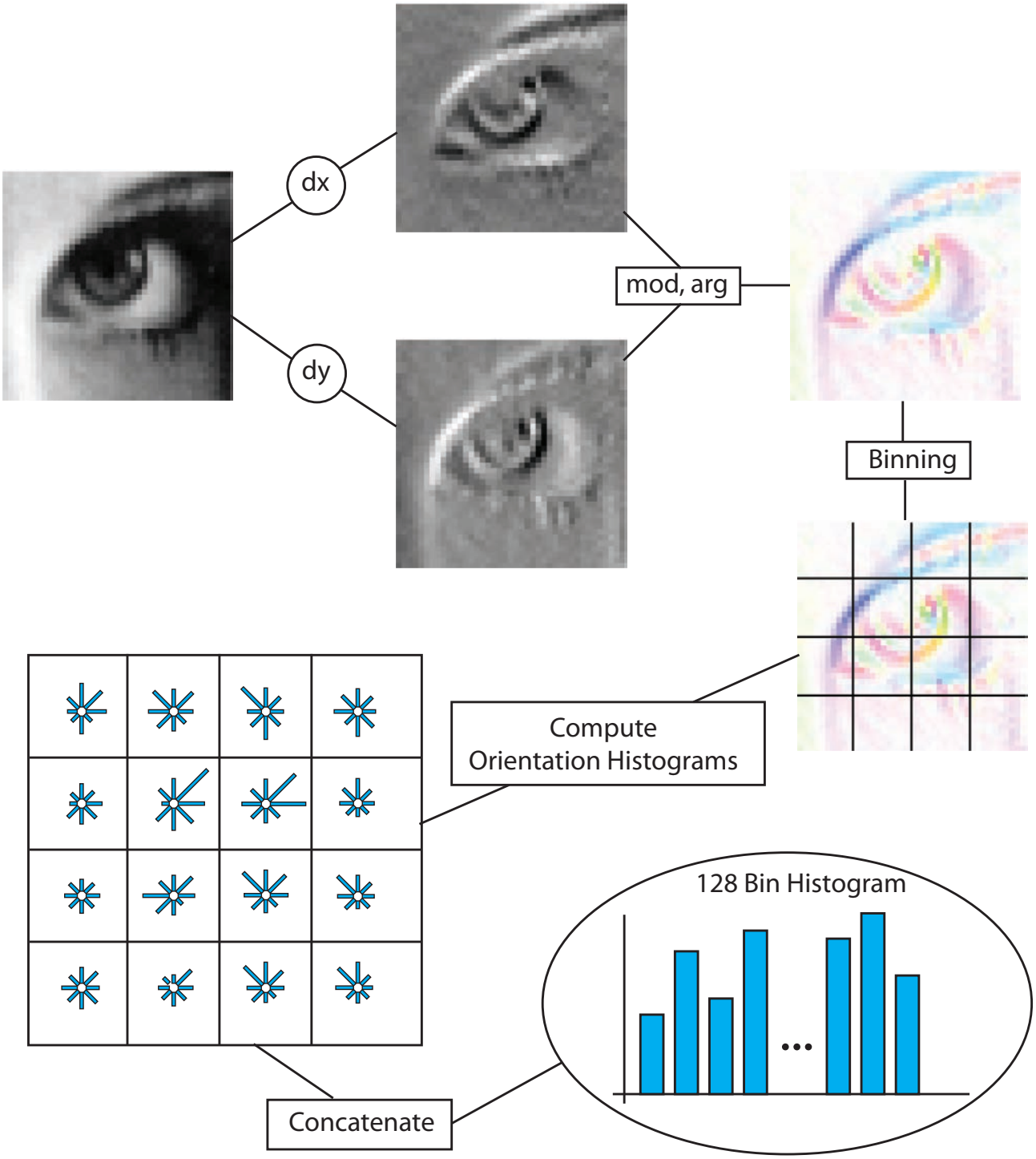


Figure 6.7: Illustration of how a SIFT descriptor is calculated. Given the patch, $\mathcal{P}$ the gradient is calculated in the x and y directions. The gradient field is then transformed into a modulus and argument representation, here the angle is denoted by color and modulus by intensity. Lastly the modulus/argument patch is divided into a 4 by 4 grid where an 8 bin histogram is calculated over the angles weighted by the modulus for each cell. These $16(= 4 \cdot 4)$ 8 bin histograms are merged into a $128(= 8 \cdot 16)$ bin histogram.

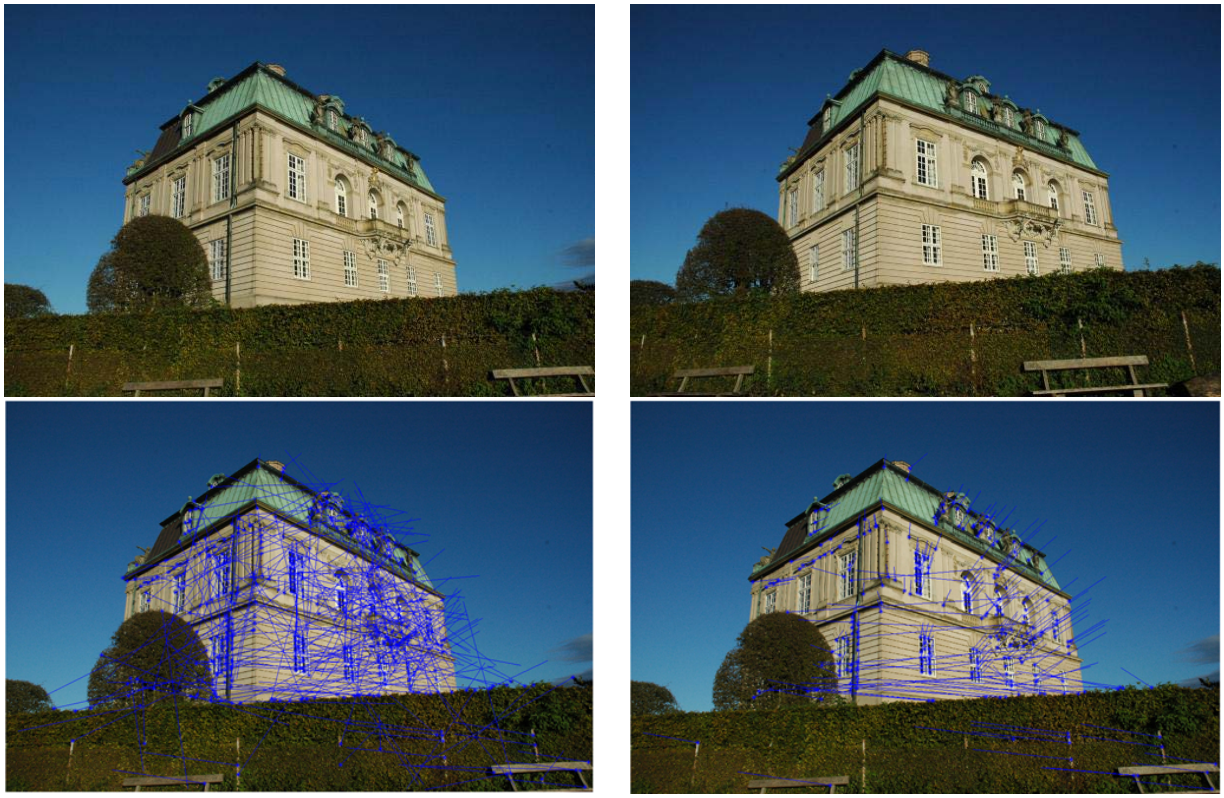


Figure 6.8: An example of the correspondence problem, where the two top images are to be matched. Features are extracted as Harris corners, and matched via correlation. In the bottom row the blue arrows indicates the displacements needed to get a match. On the left is an unconstraint matching, where as the camera geometry is used to the right. This geometry is estimated via Ransac.

Part III

Object Recognition and Segmentation

A main part of computer vision and image analysis deals with object recognition, localization, and (image) segmentation, where the aim is to answer question such as, ”is there a car in this image?”, if so ”where is it?” etc. The, by far, predominant method for answering these questions is by posing the problem as one of statistical classification – c.f. Appendix B. That is a process of extracting features from an image, or parts of an image, and based on these a statistical test is made to determine if a given object is present. As such this part could end here by simply refereing to recommendable textbooks on statistics and machine learning, e.g. [Hastie et al., 2009] or [Bishop, 2007]. There are, however, some commonalities or typical solution patterns, which are attempted covered in this part, by presenting typical case stories and some of the popular approaches for performing these statistical classifications in images.

Figure III.1: An illustration of the loos taxonomy of object recognition and segmentation. **a)** Image search, i.e. from a collection if images find the one containing a given object. **b)** Localization, i.e. set a bounding box around instances of the given object. **c)** Segmentation, i.e. find the pixels that are part of the given object. The face image and it's segmentation is supplied by Jens Fagertun.

As hinted, object recognition, and segmentation, consists of several different, albeit highly related tasks. This presentation, thus, starts with a loos taxonomy of this sub-field which is also reflected in the subjects of this parts chapters. Object recognition and segmentation can roughly be divided into the following three categories, also illustrated in the above figure,

Image Search or image database search, where the task is to find an image with a given object, e.g. part a) of the above figure, where the task could be to find an image containing a face. This is the subject of Chapter 7. To exemplify, a typical task is to find images of a given object in a large database of images.

Localization, where the task is to find the location – at the level of a bounding box – of where an object is. In relation to part b) of the above figure, the task is to find where in an image the face is located. This is the subject of Chapter XXX. This is usually used as a prepossessing step for segmentation.

Segmentation, or the classification of which pixels are part of a given object, as seen with the face in part c) of the above figure. This e.g. a very popular tool for photo editing, and in shape analysis of e.g. faces or biological sells. This is the subject in two chapters to be added at a later time, one on active appearance models and one on texture .

It is noted, that there is a clear hierarchy of this taxonomy. In that if it e.g. were known where in a set of images an object occurred, than an image search should only give positive results for images where that object occurred one or more times. The reason for not just making segmentation algorithms, and then deducing the results to the other tasks from there is a) that more computational efficient algorithms can be made for the simpler tasks, and b) that the simpler tasks are easier to solve. As for the latter the tasks of the taxonomy are often used in a sequence, where e.g. a segmentation algorithm is often initialized by a localization algorithm, and a localization algorithm is only run on images that have a given object.

There is also another important distinction between object recognition and segmentation tasks, namely that of recognition and that of localization, i.e. posing the question of either 'what' or 'where'. As humans this distinction might seem a bit peculiar. As an example, a typical approach for us to solve the task of determining if an image contained a car would consist of searching an image for a car and if we found it we would would simultaneously have solved the recognition an localization task. Computers, however, work in very different ways than humans, and object recognition and segmentation is still not a full solved problem. Remembering that a statistical approach is taken, the presence of roads and street furniture would be ques that a car is likely

to be found. This is opposed to an image with blue sky and birds. As such, considering a whole image, with out localization, will help the recognition task.

Lastly, it should be noted, that recognition tasks are often performed in hierarchy, corresponding to a hierarchy of the class that are in question. So if where e.g. to find the faces in an image and determine if they were smiling, i.e. find smiling and non smiling faces, then we could proceed by first recognizing and locating faces, and then determine if the individual faces were smiling. The last task, could e.g. be done by segmenting the face via an active appearance model, [Cootes et al., 1995], where by the shape of the mouth was also found.

Chapter 7

Image Search

Digital cameras have become ubiquitous, and with the cost of taking a picture approaching zero, massive collections of images are – and have been – emerging. A main challenge is to be able to retrieve the relevant images again. If, as an example, your hard disk is full with unordered photographs, how are you going to find the holiday shots at, say, the grand canyon? Or to mention one of the grand challenges in this field, can we make efficient internet-scale[1] image search?

Addressing these issues is a matter of image (database) search, and is the subject of this chapter. The predominant method for doing this by using the so called bag of words methods [Sivic and Zisserman, 2003], and outlined in Section 7.3. As hinted by the name this method is heavily inspired by text search. This section is followed by Section 7.4, outlining how such a bag of words model should be used to implement an image database. Prior to Section 7.3, an intuitive outline of statistical classification is given in Section 7.1 followed by Section 7.2 on clustering. These first two sections form the statistical foundations of the latter two, and are as such included here.

7.1 Statistical Classification

Classification, or statistical classification, basically decides which of two or more classes a given observation belongs to given the measured values of that observation. More formally, assume that an observation is given by an n dimensional vector $\mathbf{x}$, then we want to define a *classification function* $f(\mathbf{x})$ – also known as *decision rule* or *classifier* – that maps from the observations $\mathbf{x}$ to a specific class, c, i.e.

$$f : \mathbf{x} \to c \in \mathfrak{C} \ ,$$

where $\mathfrak{C}$ is the set of all relevant classes. The challenge is how to formulate this decision rule f?

In many walks of life decision rules are easy to come by. As an example consider doing quality control of 5 Ohm electronic resistors, which are required to be within a tolerance of five percent, then based on the resistance measurement of a given component, x, the following decision rule would apply:

$\mathbf{x} \leq 4.75$ Ohm	Classify resistor as 'Not OK'
$\mathbf{x} > 4.75$ Ohm **and** $\mathbf{x} < 5.25$ Ohm	Classify resistor as 'OK'
$\mathbf{x} \geq 5.25$ Ohm	Classify resistor as 'Not OK'

In this example $\mathfrak{C} = \{\text{'OK'}, \text{'Not OK'}\}$. In many cases of object recognition and localization, it is not this straight forward to formulate a decision rule or classifier, e.g. what pixel configuration should be classified as a car and which should not? In these cases we often resort to statistical methods, c.f. [Hastie et al., 2009], also known as machine learning, in order to solve the problem. That is, we assume that we have a data set of relevant observations $\mathbf{x}$, and hopefully a set of associated classes – i.e. tuples of $(\mathbf{x}, c)$ – and based on this we would like to estimate the classification function $f(\mathbf{x})$.

As a rather simple example, assume we have a mix of gymnastics and basketball players, and based on measurements of the athletes height and weight we would like to determine which sport they do. In relation to the formalism

$$\mathbf{x} \in \{\text{'Height'}, \text{'Weight'}\} \quad \text{and} \quad c \in \mathfrak{C} = \{\text{'Gymnastics'}, \text{'Basketball'}\} \ .$$

[1] Today internet based image search is mainly done by considering the words in the image caption.

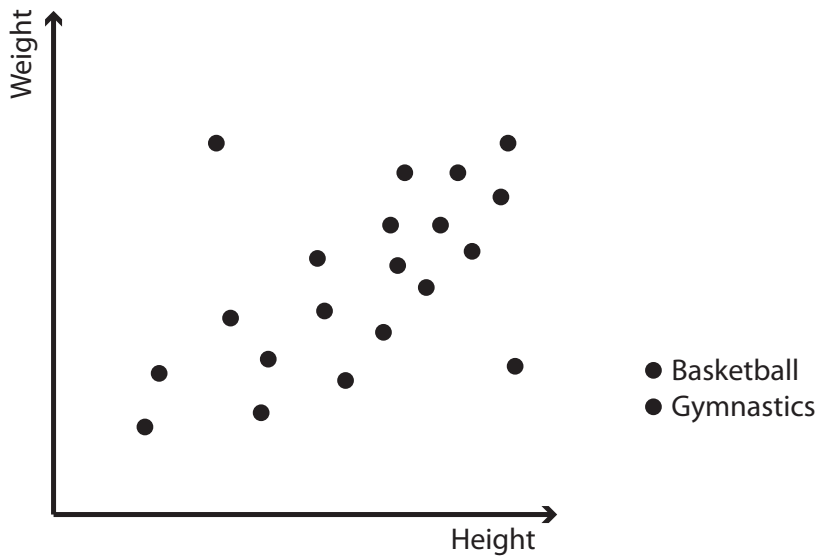


Figure 7.1: Corresponding height and weight measurements from twenty athletes doing either gymnastics or basketball.

To help us formulate the classification function, we have measurements of height and weight from twenty athletes, c.f. Figure 7.1. And as a further aid, we also have the sport associated with each of these twenty athletes, as seen in Figure 7.2-a. As illustrated in Figure 7.2, the classifier can then be seen as a segmentation of the Height$\times$Weight$\in \mathbb{R}^2$ into to disjunct parts. Considering e.g. Figure 7.2-b, the the observations to the right of the line are classified as basketball players and the ones to the left as gymnasts.

In cases such as Figure 7.2, the estimation of the classification function, $f(\mathbf{x})$, is typically cast as an optimization problem. That is, we define a quantitative measure of how well a given function performs, i.e. an objective function, and then try to find the best $f(\mathbf{x})$ according to this measure. A typical objective function would e.g. evolve around how many classifications of the given data, i.e. *training set*, were correct. To do this, however, the classification function, $f(\mathbf{x})$, needs to be parameterized, e.g. in Figure 7.2-b the linear classification function used is

$$f(\mathbf{x}) = \begin{cases} \text{Basketball} & \text{if} \quad \mathbf{n}^\mathbf{T}\mathbf{x} \geq \alpha \\ \text{Gymnastics} & \text{if} \quad \mathbf{n}^\mathbf{T}\mathbf{x} < \alpha \end{cases}, \tag{7.1}$$

where $\mathbf{n}$ is a 2D vector and α is a constant, and the parameters, $\mathbf{n}$ and α, are the ones that should be estimated. Choosing the parameterizations of $f(\mathbf{x})$ – i.e. the choice of model class c.f. Chapter 5 – also effects what partitions of the observation space are possible. That is if we chose a linear model such as (7.1), then we can only get linear partitions. As seen in Figure 7.2-c and Figure 7.2-d, more elaborate models (or more correctly model classes) can be used, typically achieving a better classification of the training data.

Achieving good performance on the training data is not the aim of the classification as such – the classification result is already known. The hope is, that classification function estimated via the training data *generalizes well* to new data or observations, unknown at training time. As seen in Figure 7.3, if a new observation is given, then the classification function of Figure 7.2-d would not classify it correctly – where as the ones from Figure 7.2-b and Figure 7.2-c would. In relation to Appendix B.5, the choice of model (class) is in part a question of not over or under fitting, in order for the estimated classification function to generalize as well as possible.

Throughout the relevant literature, there is a overwhelming wealth of methods for doing statistical classification, c.f. e.g. [Bishop, 2007, Hastie et al., 2009]. Probably, the main distinguishing factor between these methods is the model class used as classification functions, where a good deal of them challenge popular conceptions of what a function is. It should also be noted, that the the two class classification problems extend in a straight forward manner to multiple class problems, as illustrated in Figure 7.4. Multiple class classification problems occur frequently in computer vision where we, e.g. would like to determine if there is a a)car, b) person, c) bicycle, d) flower or e) non-of the above at a given location in an image.

A central difference between the two examples should be noted, namely, that it the resistor case we are observing – i.e. measuring – a quantity that is perfectly correlated to the classification, whereas this is far from

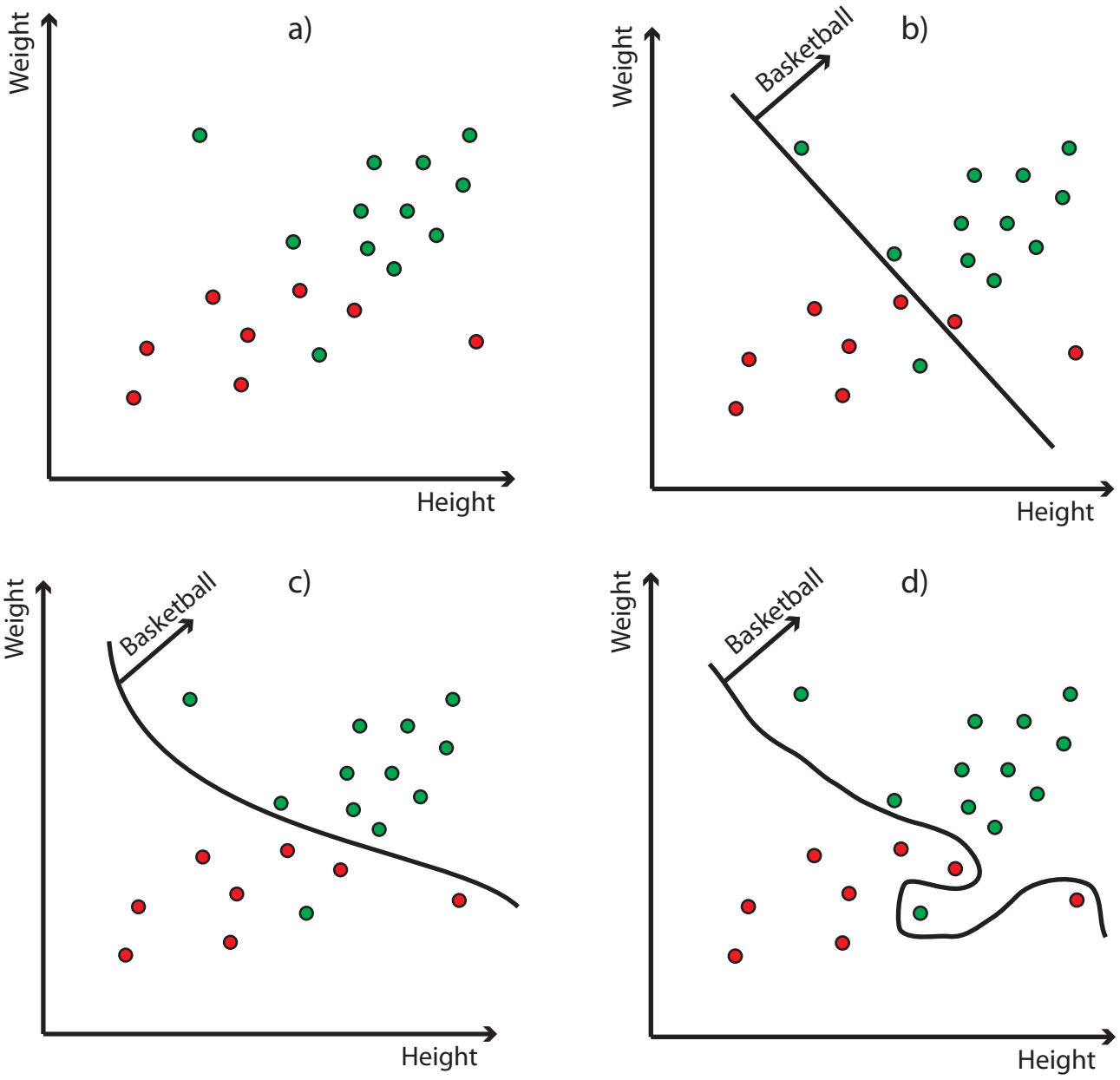


Figure 7.2: The associated classes for the measurements from Figure 7.1, where red dot denote gymnastics and green dots denote basketball. Figure a) is the raw data, and figures b), c) and d) have various classifiers of discission rules annotated.

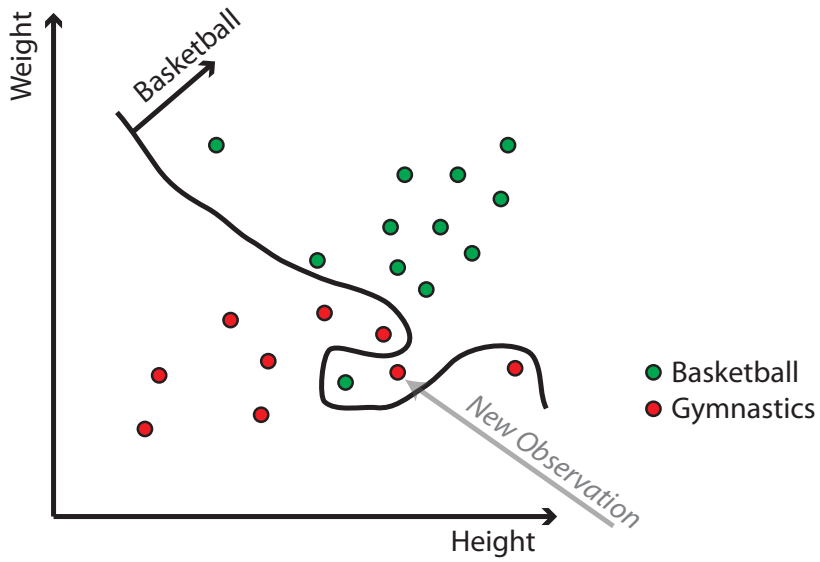


Figure 7.3: The aim of classification functions is that that they generalize well to new - unseen data.

the case in the athletes example. Stated differently, since there are other (uncorrelated) determining factors than height and weight when athletes choose a sport, it is unlikely that we can make a perfect classification based on only the two observation parameters given. In this case, it is similar to many classification problems in computer vision.

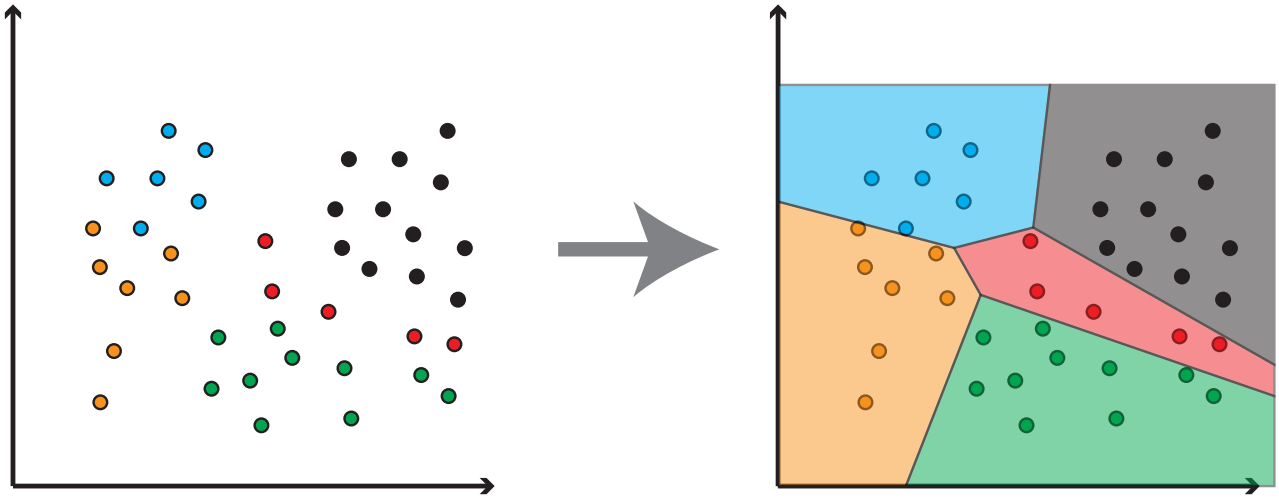


Figure 7.4: A stylized five class classification problem with training data to the left and the decision rule or classifier added to the right.

A last issue, is the distinction between *supervised learning* and *unsupervised learning*, which are both methods for estimating classification functions from given data. The difference is whether the classes associated with the observations are provided as part of the training data. That is if only the data from Figure 7.1 were provided the athlete example from above would be an *un*supervised learning problem, whereas Figure 7.2-a poses a supervised learning problem. Unsupervised learning is the subject of the next Section 7.2, and is also a general tool used for finding structure in large data sets, i.e. so called data mining.

7.2 Clustering

As mentioned above, clustering is a classification method for unsupervised learning, the underlying idea is that a given data set should be divided into k meaning full clusters or groups. Where k is a given scalar parameter.

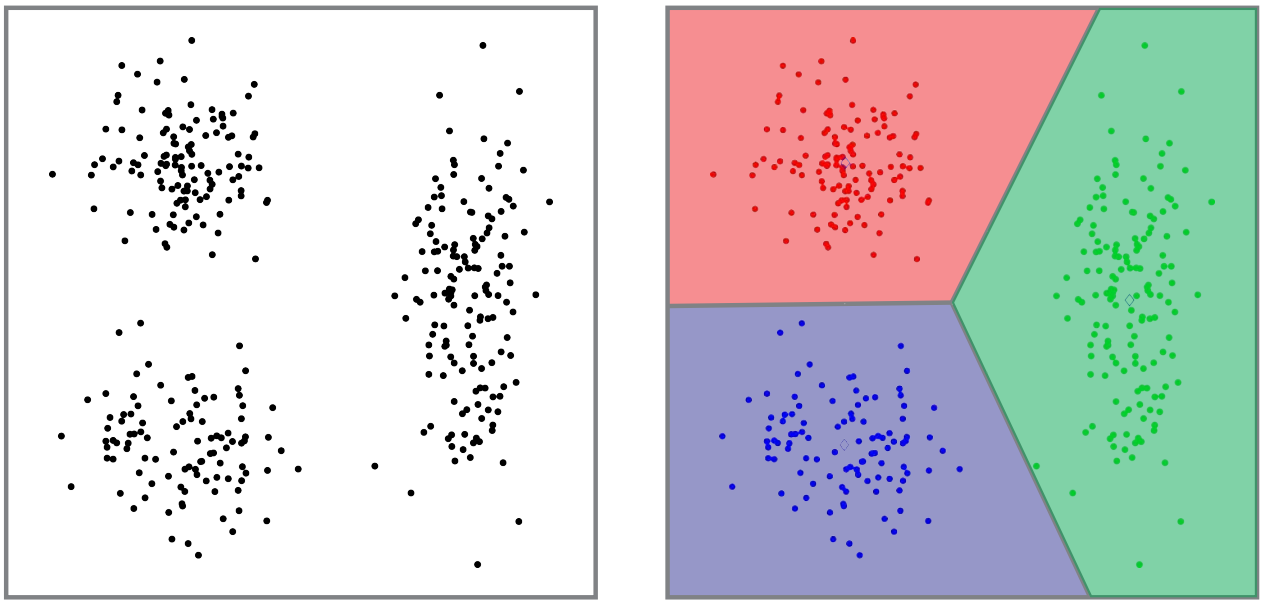


Figure 7.5: **Left:** A sample data set of 100 2D points, and to the **Right:** a clustering via this data, including the division of the observation space. The clustering is done via K-means clustering. Note in this example there are three clusters, so $k = 3$.

In terms of the previous section, this clustering should also divide the observation space into k disjunct parts – corresponding to the division of the given data set – and thus constitute a classification function.

As an example consider Figure 7.5-Left, where a set of 100 points are given. Most human observers would agree, that there are three clusters in this data set corresponding to the division shown in Figure 7.5-Right. The challenge of clustering algorithms is to formulate our human intuitive notion of clusters mathematically, and following that construct an algorithm for performing this division. A standard method of doing this is K-means clustering [Hastie et al., 2009].

7.2.1 K-Means Clustering

With k-means clustering, we are given n data points, x_i, which are to be divided into k ($k << n$) different clusters or groups, and the aim is to compute a good $f(\mathbf{x}_i) \to c \in \{1 \dots k\}$. As indicated by the methods name, this is done by considering the means μ_j, $j \in \{1 \dots k\}$, of the points classified as belonging to the j^{th} cluster. That is denote by J the indices of the data points mapped to cluster j, i.e.

$$v \in J \quad \Leftrightarrow \quad f(\mathbf{x}_v) = j \ ,$$

then

$$\mu_j = \frac{1}{|J|} \sum_{v \in J} \mathbf{x}_v \ . \tag{7.2}$$

The idea of the k-means clustering algorithm is then to find the clustering, defined by $f(\mathbf{x})$, such that the average distance from the data points $\mathbf{x}_i$ to their respective cluster mean, μ_j, is minimized. That is the following quantity is minimized

$$\min_{f(\mathbf{x}_i)} \sum_{j=1}^{k} \sum_{v \in J} \|\mathbf{x}_v - \mu_j\|_2 \ , \tag{7.3}$$

where

$$\sum_{v \in J} \|\mathbf{x}_v - \mu_j\|_2 \ ,$$

is the sum of the distances from data points to the mean of cluster j.

The k-means cluster algorithm attempts to minimize or solve (7.3) in a greedy and iterative fashion. Firstly, a guess of the cluster means, μ_j are made, e.g. by setting them equal to k random points from the data, $\mathbf{x}_i$. Then

Given a set of n data points, $\mathbf{x}_i$, and the number of clusters k.

1. Pick k data points at random, and set the cluster means, μ_j $j \in \{1, \ldots, k\}$, equal to these random data points.
2. Compute the classification of all the n data points according to (7.4), i.e.
$$\forall i \quad f(\mathbf{x}_i) = \min_{j \in k} \|\mathbf{x}_v - \mu_j\|_2 \ .$$
3. Recompute the means, μ_j, according to (7.2), i.e.
$$\mu_j = \frac{1}{|J|} \sum_{v \in J} \mathbf{x}_v \ ,$$
where
$$v \in J \quad \Leftrightarrow \quad f(\mathbf{x}_v) = j \ ,$$
4. If not converged go to step 2.

Table 7.1: Outline of the k-means clustering algorithm.

an intermediate choice of the classification function is

$$\forall i \quad f(\mathbf{x}_i) = \min_{j \in k} \|\mathbf{x}_v - \mu_j\|_2 \ . \tag{7.4}$$

That is that data point i is assigned to the cluster it is closest to, defined as the distance to that clusters mean. Given that the $f(\mathbf{x}_i)$ are redefined so is J from (7.2). The next step of the iteration process is then to recompute the means via (7.2). This iteration between computing (7.2) and (7.4), is then done until convergence. By convergence is meant that the classification function, $f(\mathbf{x}_i)$, does not change. The algorithm is outlined in Table 7.1. As an example the k-means clustering algorithm was used to compute the results of Figure 7.5.

The k-means algorithm a very popular approach to unsupervised learning, e.g. due to its simplicity, but there is no guarantee of it reaching a global optimum, e.g. solving (7.3). Many approaches have thus been proposed to doing k-means clustering, mainly differing in how the algorithm is initialize, e.g. first choice of means, μ_j, and running time. Since k-means clustering is often run on large data sets, running time is a crucial point. For further information the reader is referred to [Hastie et al., 2009].

7.2.2 Choice of Metric

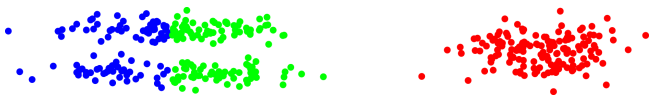


Figure 7.6: Severely changing the metric on the data set used in Figure 7.5, by scaling the axis. Note how this changes the clustering result.

An important aspect of unsupervised learning, which should be mentioned here, is the choice of metric, i.e. how distances are quantified or measured. As an example reconsider Figure 7.5. If we assume that one axis denoted time and another weight, then there is no clear way how these two different units of measurements should be compared. If the two measured entities were weighted differently relative to each other, we might achieve the result of Figure 7.6. Here it is noted that the result changes drastically. What is observed here is; that the basis for clustering the data points, in an unsupervised manner, is the notion of distance between the points. If it is not self evident how this distance measure is to be formulated, e.g. two variables scaled wrt. each other, then there is a subjective choice that needs to be made. In many cases, however, a natural distance is given, e.g. if all the dimensions if the measured data points are in the same units..

7.2.3 A Segmentation Example

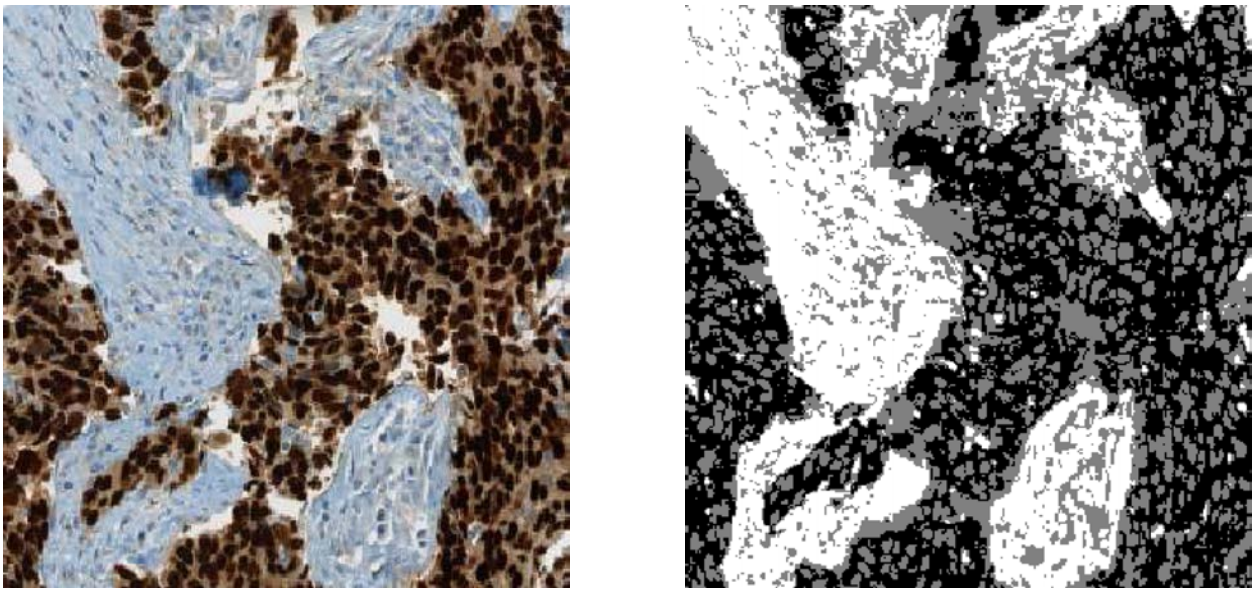


Figure 7.7: **Left:** A tainted cell sample, kindly supplied by Visionpharm. **Right:** A segmentation of the left image made by performing k-means clustering on the RGB values.

As an example of clustering, consider the segmentation problem of Figure 7.7-Left, where. Here the aim is to segment the image into the three visual clusters that intuitively exist in the image. The result of using k-means clustering is seen in Figure 7.7-Right. To do this each pixel was transformed into a data point $\mathbf{x}_i$, where the measure variable were the R, G and B channels of the image, i.e.

$$\mathbf{x}_i = \begin{bmatrix} R_i \\ G_i \\ B_i \end{bmatrix} ,$$

and the number of clusters were set to three, i.e. $k = 3$.

Figure 7.7 represents a typical image segmentation problem, and as such many other algorithms than k-means clustering are used, although it is one of the more widely used. One other approach to this type of clustering, should however be mentioned, due to it's wide use, namely the *Mean shift algorithm* [Comaniciu and P., 2002].

In relation to the choice of metric, the choice of color space is a design parameter that is often experimented with in these types of problems. That is, instead of using the RGB-color space as is done here the IHS[2]-color space might prove more successful. To see the relation between color space and metric, note that the distance between two given pixels RGB values and IHS values are generally different – assuming that the two-norm is used.

7.3 Bag of Words - A Descriptor for Images

Given the prerequisites in the previous two sections, this section will start on the core of this chapter, namely image database search. In particular, this is done by introducing how to construct a descriptor for an image. This should be seen as an analog to the feature descriptors presented in Chapter 6, where the aim of using these descriptors is to find features that resemble each other, and thereby doing feature matching. Here we instead want to do whole image matching, to find the image from an image database "closest to" a query image, as illustrated in Figure 7.8. To do this we need an operational formulation of the term "closest to", which is the aim of the image descriptors presented here. These image descriptors are referred to as bag of words.

Bag of words for images, as presented by [Sivic and Zisserman, 2003], are inspired by – and take their name from – a very successful technique from text search, cf. [Baeza-Yates and Ribeiro-Neto, 1999]. The

[2]Intensity Hue Saturation

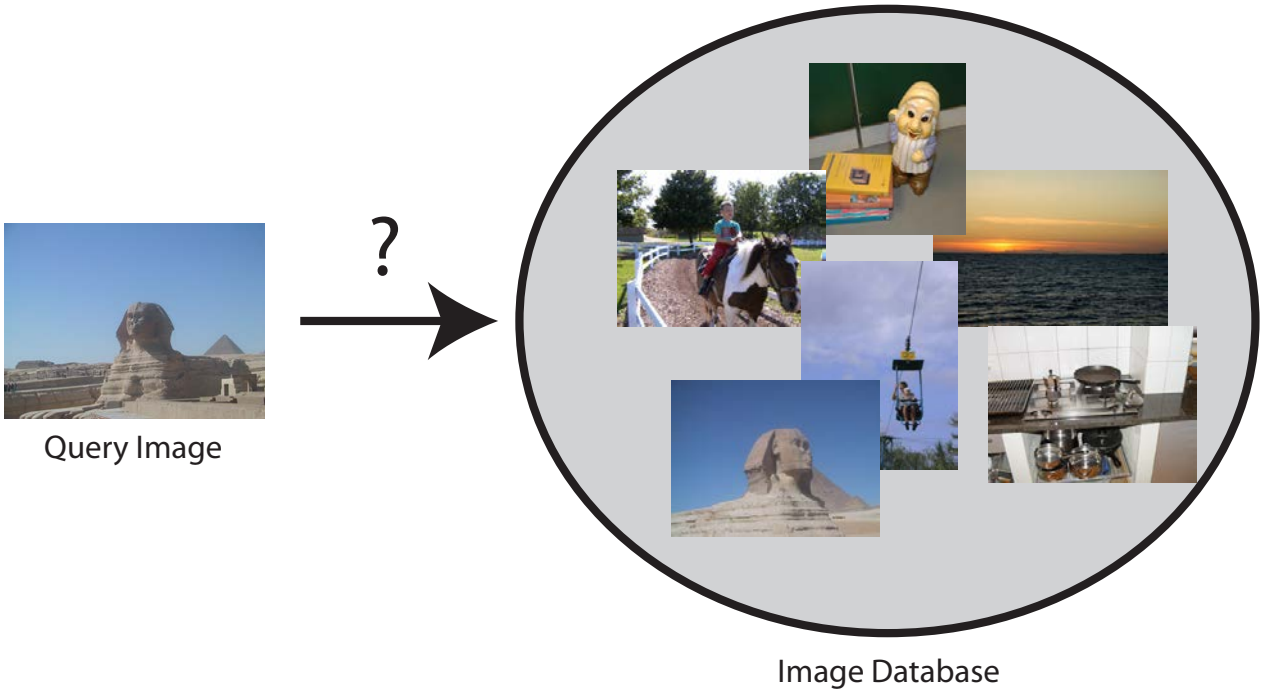


Figure 7.8: An illustration of image database search, where we want to find the image closest to the query image in the image database. To do this, we need an operational notion of closeness, which is what the bag of words supply.

bag of words approach to *text* document search, forms the histogram of all words in a text, e.g. counts how many times the words ”jump”, ”the” and ”daunt” occur – irrespectively of their ordering. These word counts form histograms, which are use as text document descriptors. An importance weighing is also performed[3], to reflect that some words are more discriminative than others, e.g. the occurrence or frequency of ”piston” in a text is more discriminative than the frequency of ”hello”. The similarity between texts is then computed by comparing these histograms in light of their importance weighting. As in image feature search, if we have a similarity measure, then we can find the most similar or closest text document, and then we can do database search.

An insight of [Sivic and Zisserman, 2003] was; that if we had an equivalence of words in images, then the successful bag of words technique from text search could be extended to images. Thus they introduced the concept of *visual words*, intuitively defined as small visual elements, such as eyes, ears, door handles etc. As an example of what is sought by visual words consider Figure 7.9 and Figure 7.10.

An issue with visual words is that they should be computed automatically, such that e.g. human annotation is not needed for every image in an image database. To achieve this automation [Sivic and Zisserman, 2003] assumed that a visual word corresponds to an image point feature, c.f Chapter 4. They also assume, that features belonging to the same visual word have similar descriptors, c.f. Chapter 6. The most populat choice of image feature seems to be the maximally stable extremal regions (MSER) [Matas et al., 2004, Nistér and Stewénius, 2008], and is the one used in [Sivic and Zisserman, 2003]. But for consistency with the rest of the text the SIFT detector is used, c.f. Section 4.4.1. Here, and throughout much of the literature, the chosen descriptor is the SIFT descriptor, cf. Section 6.2.2.

Visual words can thus be estimated from a training set of images, e.g. the image database it self, by the following algorithm:

1. Extract sift features and descriptors from *all* images, and store them as a combined data set. This are interpreted as a lot of data points in the 128 dimensional space of the SIFT descriptor.

2. Perform K-means clustering on this data set of SIFT descriptors from item one. Here the number of clusters k is chosen to being equal to the number of visual words.

[3]This sometimes extended further by rejecting the most common words such as ”the”, ”an” and ”is” altogether.

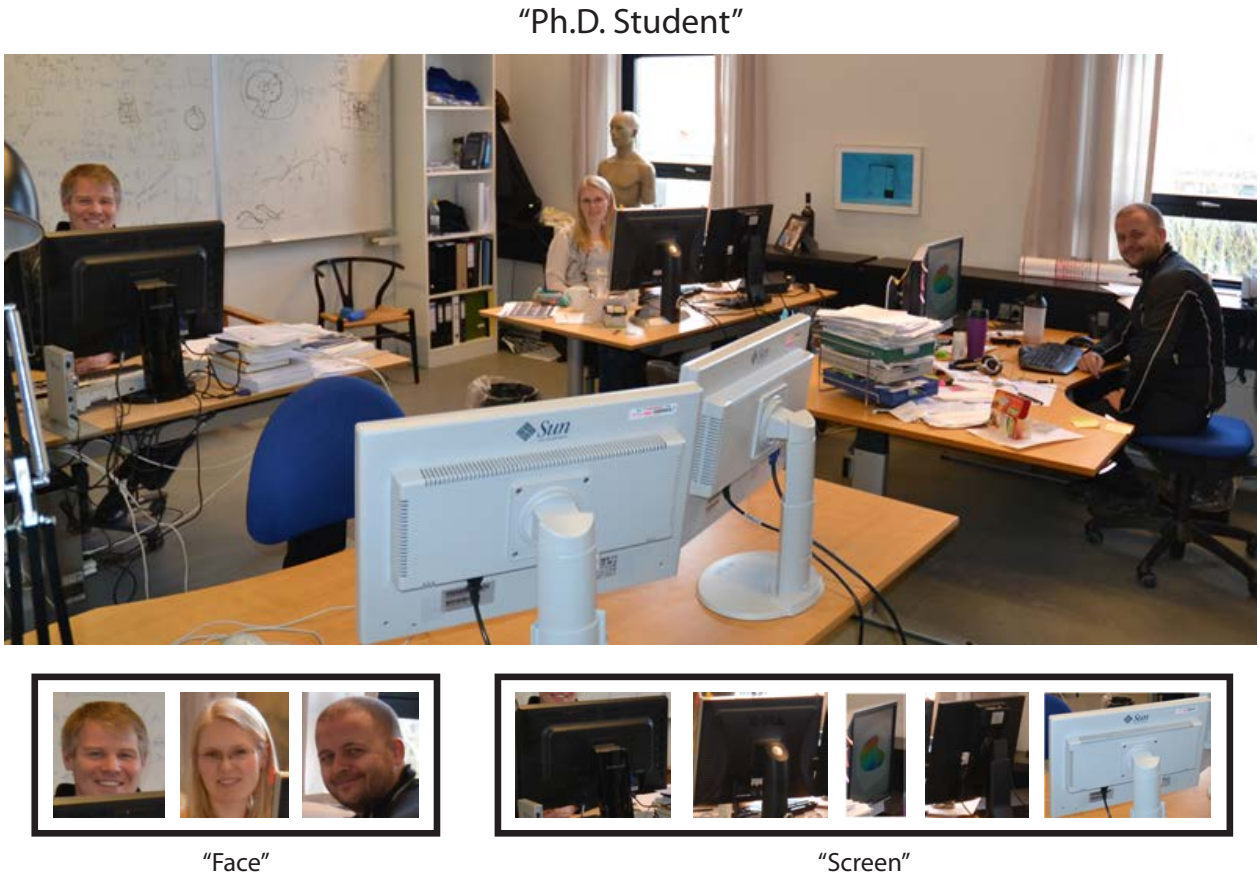


Figure 7.9: The visual words of "Face" and (computer) "screen" are good cues that an image is of a Ph.D. student.

3. Associate each cluster with a visual word.

This approach is illustrated in Figure 7.11. To extract visual words and a associated image descriptor from a given image, e.g. the query image, the following procedure is taken, c.f. Figure 7.12:

1. Extract SIFT features and SIFT descriptors.

2. For each SIFT descriptor, determine which cluster and thus visual word it belongs to.

3. Compute the histogram of visual words for the given image.

These histograms are then the image descriptors. This is similar to the SIFT feature descriptors which are also expressed as histograms. These descriptors can then be computed by considering the distance between histograms, which is correctly done as in (6.7)

$$\text{dist}(\mathbf{a}, \mathbf{b}) = 2 \sum_{i=1} \frac{w_i (a_i - b_i)^2}{a_i + b_i} \quad , \tag{7.5}$$

where $\mathbf{a}$ and $\mathbf{b}$ are two descriptors and the w_i are the importance weights, given by the inverse frequency of the visual words occurrence. In practice however, the 2-norm is used instead, although still weighted by the w_i.

Note, that the fact that the bag of words model work on the whole image has implications when searching. Since, e.g. images of boats will often contain water, images of cars often contain roads etc. The typical environment of objects, thus, act as ques for that object. As such, an image of a lake would score relatively high as an image of a boat, even if there was no boat. This is illustrated in Figure 7.9, where a screen is used as a visual word supporting that an image is of a Ph.D. student. Consider the same image as that of Figure 7.9 at lunch time; the Ph.D. students would be gone but the screens would still be there, thus supporting that this was an image of Ph.D. students – although there were none.

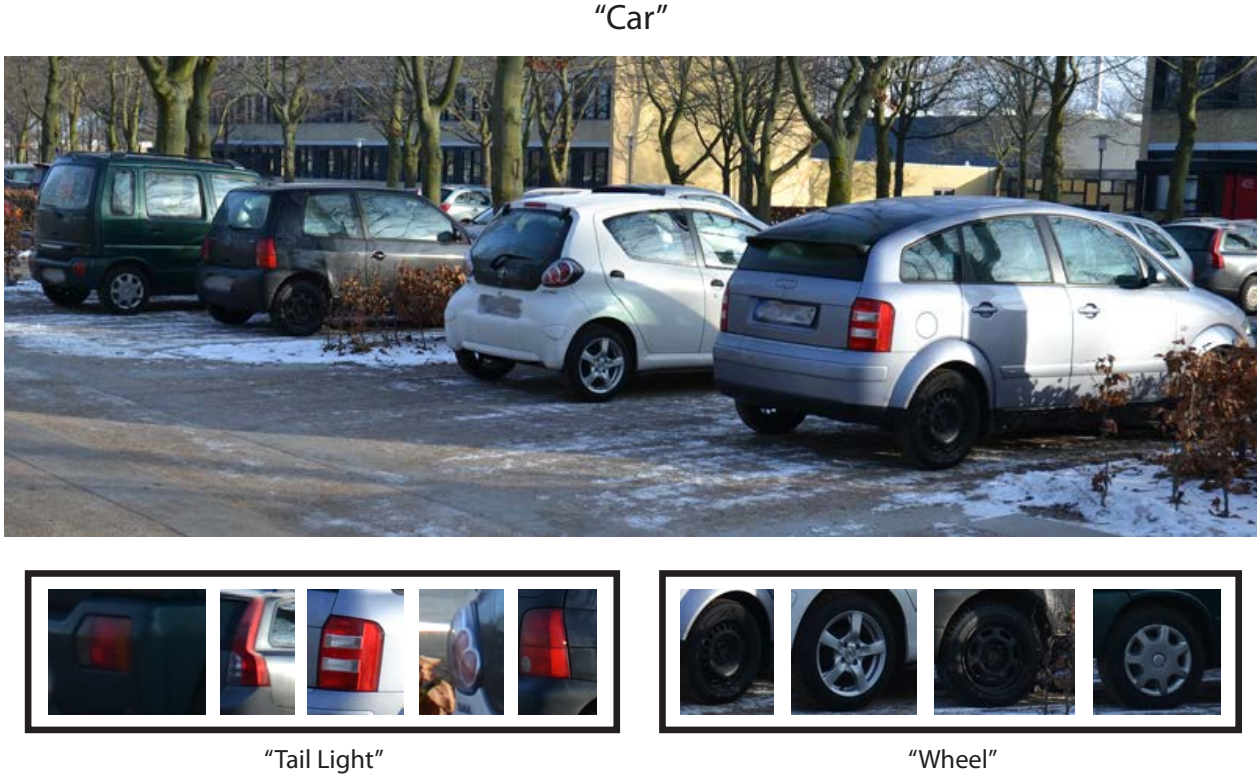


Figure 7.10: The visual words of "tail light" and wheel are good cues for an image being of a car.

7.4 Constructing a Searchable Image Database

The straight forward way to do image search based on the introduced image descriptor, is to compute the visual words and the image descriptors for all the images in the database, as described in the previous Section 7.3. These image descriptors are then stored in long list. To perform an image query, with a query image not in the database, simply:

1. Compute the image descriptor of the query image, given the visual words computed when the database was formed.

2. Compute the distance between the query image and each image in the database e.g. via (7.5).

As an illustration a small image database of 87 images and a query image was collected by sampling images from [Aanæs et al., 2012], as illustrated in Figure 7.15. The distances from the query image – i.e. image 0 – to each of the images in the image database is depicted in Figure 7.13. From Figure 7.13 it is seen, that the query image is closest to the three first images, corresponding to the three other images of the same scene. Thus the image search worked as expected in this small illustrative example. It is noted, that the fourth closest image is image 24, c.f. Figure 7.15, which can be seen to have many of the same visual elements or words as the query image.

In the previous example all the frequency weight were omitted, equivalent to setting them all to one. As stated above, this should be done, especially if the size of the image database gets above the toy stage, as in Figure 7.15. A question in this regards, how to specifically compute these weights. According to [Nister and Stewenius, 2006] a good choice is

$$w_i = \ln\left(\frac{N}{N_i}\right) \quad , \tag{7.6}$$

where N is the number of images in the database and N_i is the number of images with at least one occurrence of the visual word i.

An issue, regarding the simple image database approach mentioned above, and used for the example of Figure 7.13. Is that it becomes very inefficient as the size of the database grows. The nature of this issue, is

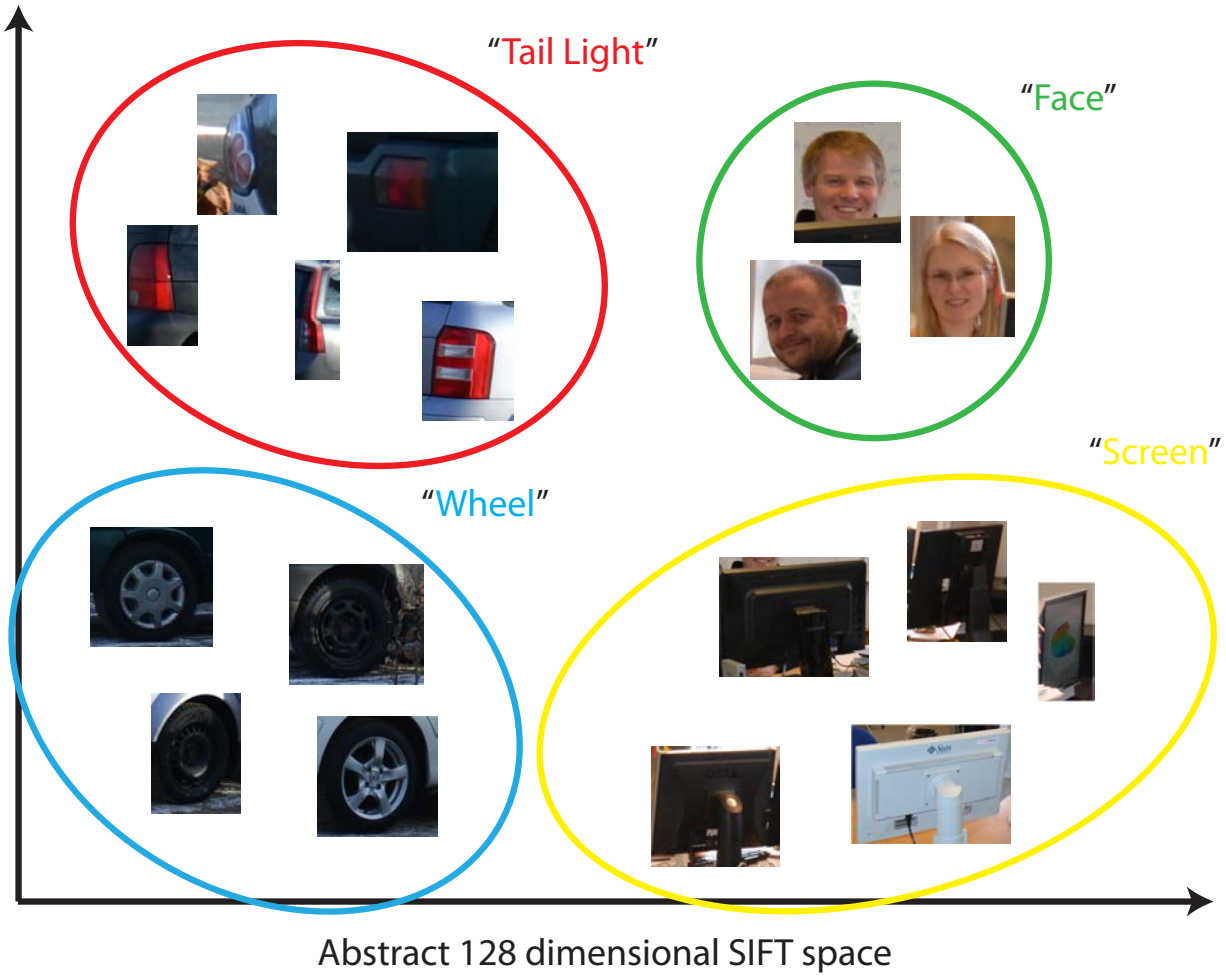


Figure 7.11: The result hoped for, if the sift features of the visual words from Figure 7.9 and Figure 7.10 were clustered.

twofold. Firstly a SIFT descriptor is compared to all visual clusters to find the closest one. Secondly, an image descriptor has to be compared to all the image descriptors in the database, and no tree based structure is used, as done in more traditional databases c.f. [Cormen et al., 2001]. A challenge in this regard, is that we are not looking for a image with a specific descriptor – i.e. key in database terms – but the descriptor closest to our query descriptor.

To address this issue [Nister and Stewenius, 2006] proposed reformulating the way´in which the clustering, and thus the visual words, are defined and computed. As illustrated in Figure 7.14, their idea was to do the clustering hierarchically, by first making one clustering of the data with e.g. 5 to 15 clusters, and the clustering the data point belonging to each cluster in an iterative fashion. Thus, in effect forming a tree data structure of the clusters and thus the visual words. So if a branching factor – i.e. number of clusters pr. iteration – was five and seven iterations were used this would give $5^7 = 78125$ visual words. Furthermore, given a descriptor from the query image, the closest visual words could be found by comparing it to five visual words at each of the seven layers, i.e. only $5 \cdot 7 = 35$ comparisons. Hereby, the visual word corresponding to a descriptor could be found by 35 comparisons in a vocabulary of 78125 words[4].

In addition, by using so called inverted files, the individual visual words vote for images they belong to, instead of directly comparing the image descriptor to all the descriptors in the database. Hereby, images of the database are only considered *if* they are relevant. The interested reader is refereed to [Nister and Stewenius, 2006] for more details.

[4]Often a larger branching factor and more iterations are made, implying much larger vocabularies.

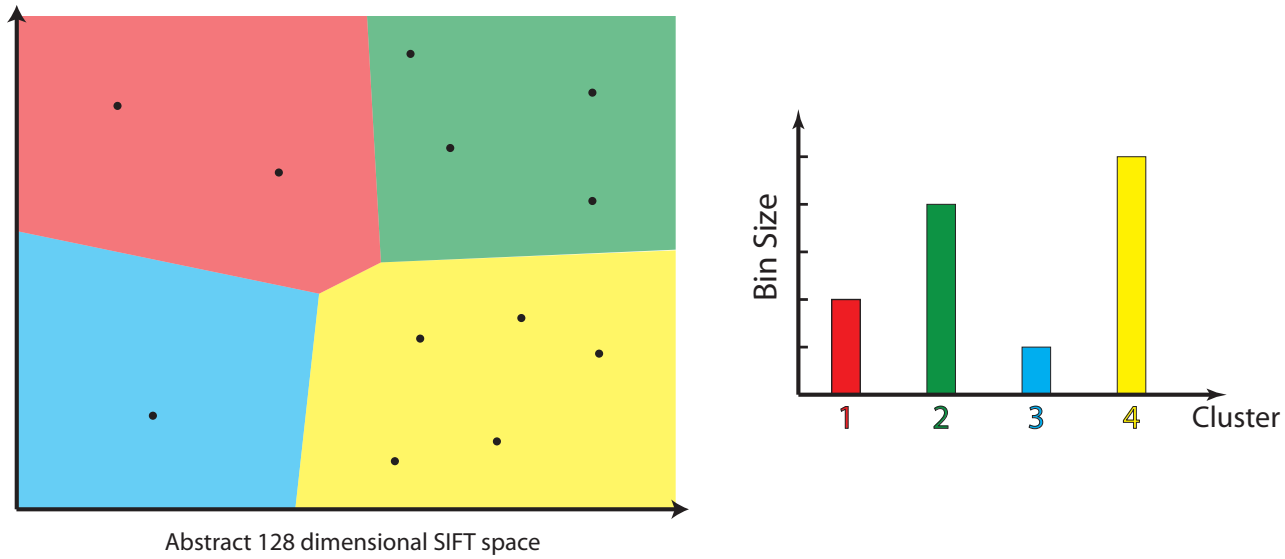


Figure 7.12: **Right:** the four regions of the clusters found in Figure 7.11, are denoted in the colors red green blue and yellow. Consider a new image with SIFT descriptors represented as the black dots. This would have two visual word classified as a ”tail light”, four visual words classified as a ”face”, one visual word classified as a ”wheel” and five visual words classified as a ”screen”. This results in the histogram to the **Left**, which is also a descriptor for this new image. As such this new image would be close to the image of Figure 7.9 than that of Figure 7.10.

7.5 End Notes

It should be mentioned, that the subject of image database search has received a vast amount of attention, in the research community, and as such there is a wealth of literature on the subject. Also in regards to applications, these go much beyond what has been mentioned here.

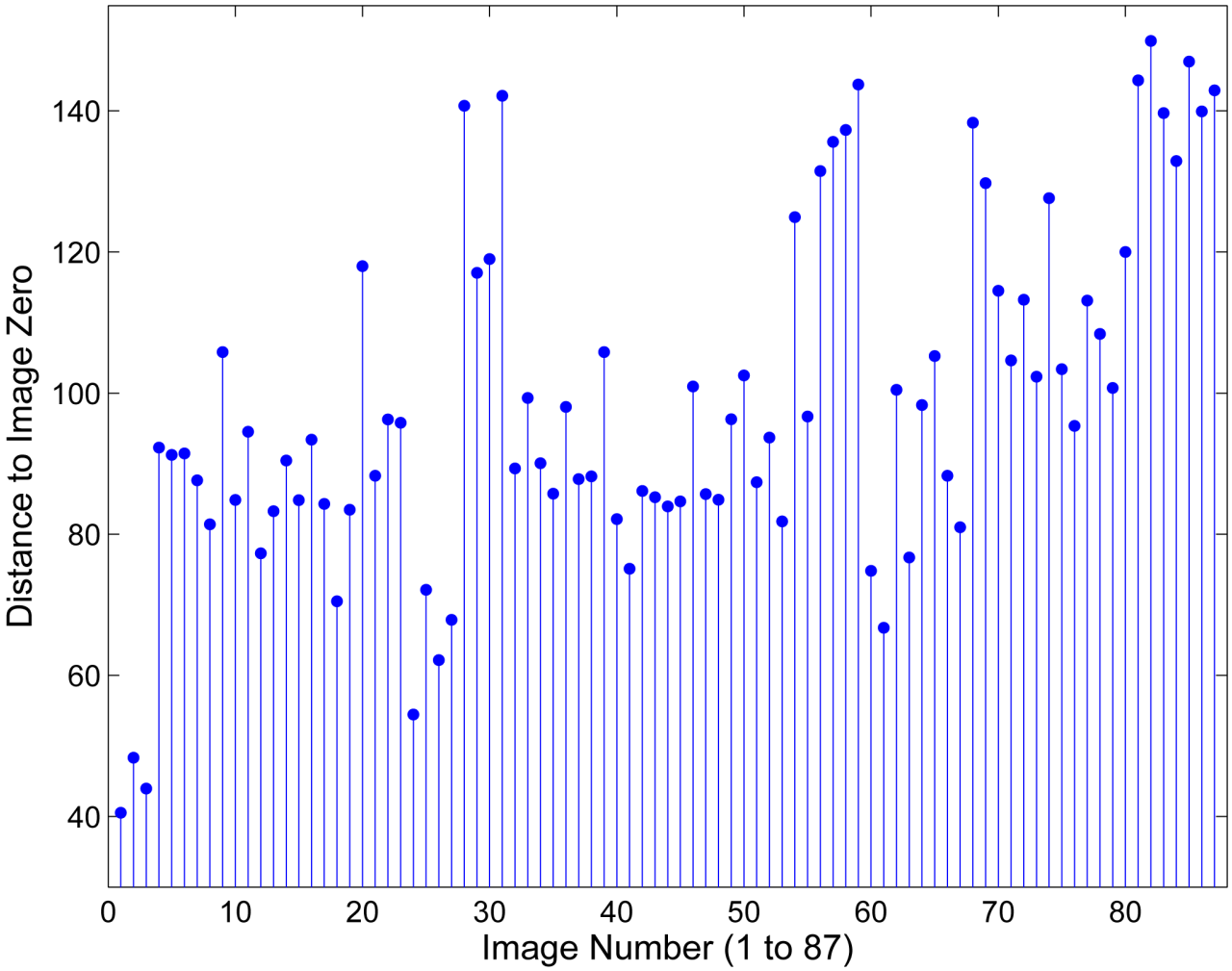


Figure 7.13: The results of the simple image database search, given the database of Figure 7.15. Note that the three closest images to the query image is the ones from of the same scene.

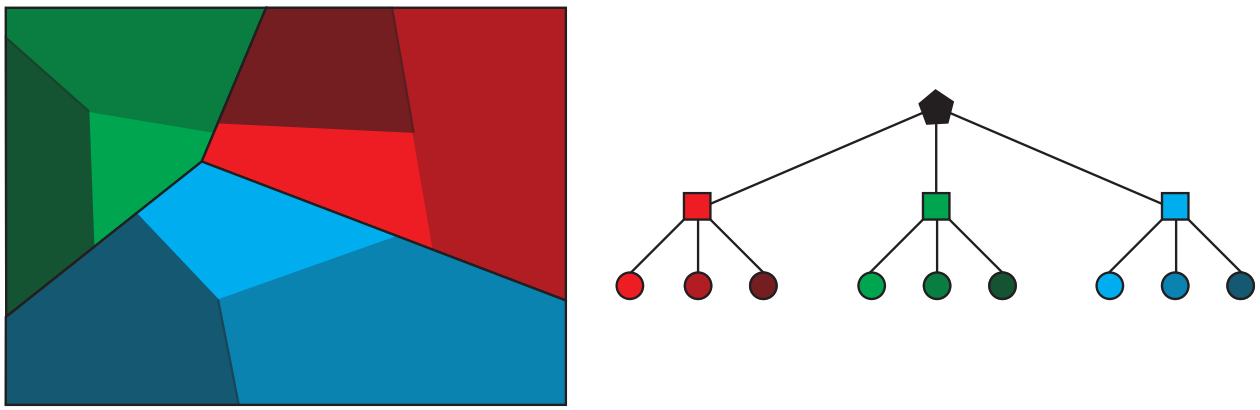


Figure 7.14: **Right:** An illustration of hierarchical clustering. A branch factor of three is used, for illustration purposes. The division of the observation space is denoted by red green and blue for the first iteration, and different shades of the respective colors for the second iteration. **Left:** The tree structure induced by the hierarchical clustering.

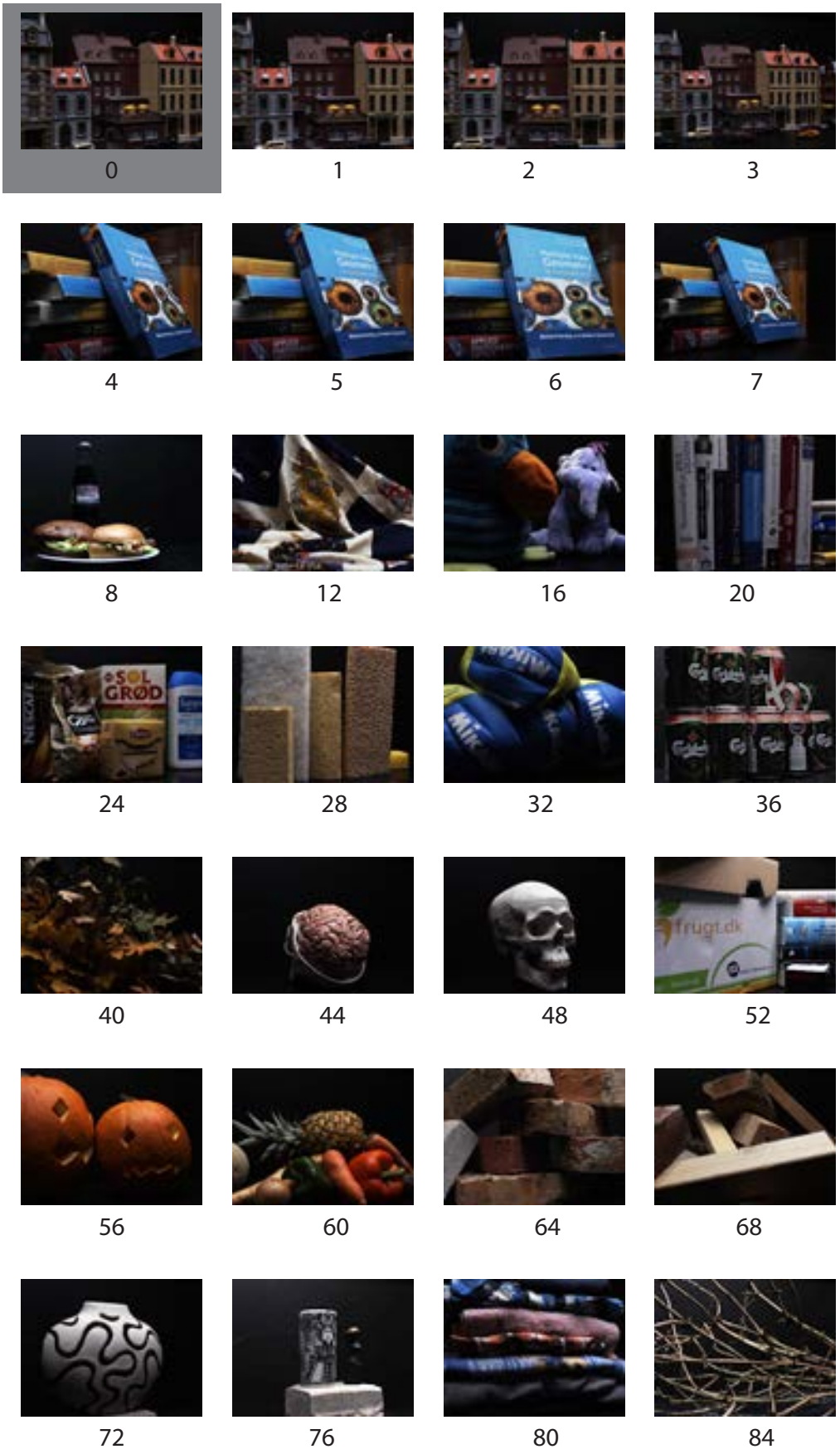


Figure 7.15: An image database of 88 images from the data set of [Aanæs et al., 2012]. Four images were collected from 22 different scenes. The quadruples are seen in image numbers 0 to 7. Due to space limitations only one image from each of the remanning scenes are shown. The image numbers correspond to Figure 7.13. In the example image 0 was the query image, and was *not* included in the database.

Part IV

Statistical Models of Images

Chapter 8

Markov Random Fields (MRF)

Many problems in computer vision are notoriously ridden by noise and indeterminacies. Algorithms for finding solutions to these problems often incorporate a priori knowledge or assumptions about the solutions e.g. as described in Appendix B. Such *regularization* or use of a priori knowledge or assumptions is often a necessary tool for getting useful results, albeit they give a bias towards the expected. As indicated in Figure 8.1, a typical sound way of expressing such priors is by modeling that neighboring pixels carry information about each other. This neighborhood information is typically formulated as a homogeneous constraint, e.g. that a pixel is similar to it's neighbor, which is the reasoning behind many smoothing schemes. A very expressive and general framework for formulating such regularization are the *Markov Random Field (MRF)* models, which are the subject of this chapter. Markov random fields have other uses in computer vision, especially in dealing with and describing texture, e.g. used in inpainting. This chapter is aimed at providing a rudimentary working knowledge of Markov random fields within image analysis. For a more in depth treatments of the matter, dedicated text books to the matter include [Li, 2009, Winkler, 2003]. Here the focus will be on formulating the MRF framework and on the use of graph cuts for finding the *maximum a posteriori probability (MAP)* estimate[1].

This chapter starts by presenting the MRF framework in Section 8.1. In Section 8.2 the Gibbs random fields are presented. This is a modeling framework that is equivalent to the MRF framework, but often more suitable when finding maximum a posteriori probability (MAP) solutions to the MRF. Sections 8.3, 8.4 and 8.5 describe a good solutions to a wide class of MAP-MRF problems using graph cuts. Graph cuts and maximum flow/minimum cut algorithms are outlined in Section 8.3, and it is demonstrated how these can be used to solve binary MAP-MRF problems in Section 8.4, which is extended to the multiple label case in Section 8.5. Section 8.6, briefly mentions other solutions to the MAP-MRF problems.

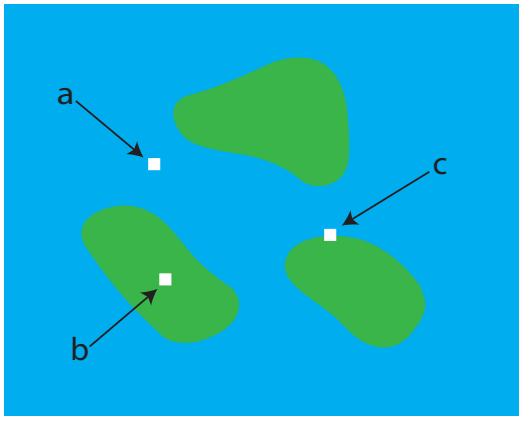


Figure 8.1: A blue and green stylized image of an archipelago, with three segments or pixels removed – denoted *a*,*b* and *c*. If the color of these three segments were to be replaced, most people would agree that segment *a* should be blue, segment *b* green and segment *c* is undecided or mixed. This illustrates, that we as a rule of thumb often assume a certain correlation between neighboring pixel, and this correlation is often in the form of a homogeneity assumption.

[1] MAP is closely related to the maximum likelihood(ML), cf. Appendix B

Binary:	{ true, false }
	{0, 1 }
Gray Byte:	$\{0, \ldots, 255\}$
RGB Byte:	$\{0, \ldots, 255\} \times \{0, \ldots, 255\} \times \{0, \ldots, 255\}$
Image Classification:	{ Land, Water}
	{ Object I, Object II, Back Ground}
Stereo Disparities:	{ Min. Disparity, . . . , Max. Disparity}

Table 8.1: Typical label sets, $\mathcal{L}$, from computer vision. Here $\times$, is used to denote tuples.

8.1 The Markov Random Field Framework

The *Markov random field* framework is quite compact and abstract, which is part of what makes it so useful and versatile. This, however, renderers the associated concepts and nomenclature relatively abstract, which experience has shown is a hinderance when teaching the subject. This is especially so for the concept of random fields, cliques and sites. Thus notion of a an image as a random variable will firstly be introduced, to give a concrete and specific example of these concepts.

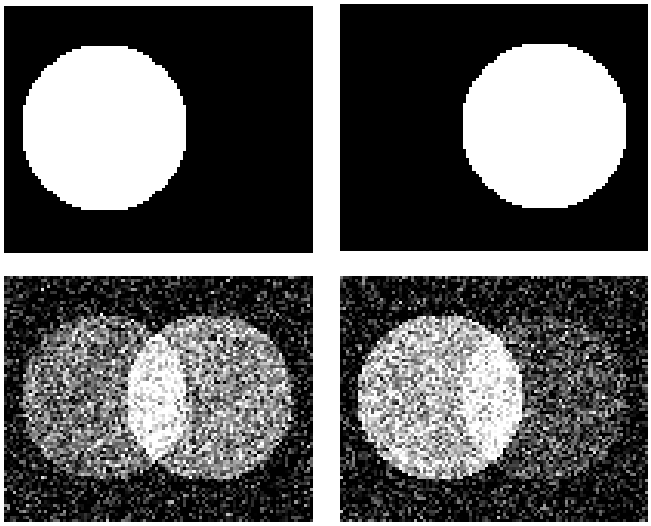


Figure 8.2: **Top Row:** Two input images, imA and imB. **Bottom Row:** Two realizations of the stochastic variable $\alpha \cdot \text{imA} + \beta \cdot \text{imB} + \varepsilon$, where α and β are stochastic scalar variables and ε is Normal distributed noise.

8.1.1 Random Fields – Images as Random Variables

As stated above, Markov random fields is a stochastic framework where images are treated as random variables following a distribution. Specifically, Markov random fields are a possible description or modeling of this distribution. A parallel could be made to e.g. the one dimensional normal distribution for scalars. So, just as classical multivariate statistics, cf. Appendix B, is an extension of introductory scalar statistics by grouping the scalar random variables in a vector, the random variables are now grouped in a matrix or pixel array. And just as in the classical multivariate case, there can be statistical interaction, e.g. correlations, between the variables. As an example, not particularly related to Markov random fields, consider the images of Figure 8.2. In this Figure two deterministic images, imA and imB, are given and used as inputs into a stochastic process given by

$$\alpha \cdot \text{imA} + \beta \cdot \text{imB} + \varepsilon \ ,$$

where α and β are stochastic scalar variables and ε is normal distributed noise. Two realizations of this stochastic variable – in the form of images – is illustrated at the bottom of Figure 8.2. Here, we in essence have a set of pixels each assigned a pixel value from a stochastic distribution, where there is clear correlation between the pixel values.

This leads us to the introduction of *random fields*, which are generalizations of random vectors, matrices images etc. Random fields are defined on a set of *sites*, $\mathcal{S}$, which is a generalization of pixels or vector elements.

Each of these sites have a stochastic variable, F_i, associated with it, where $i \in \mathcal{S}$. The realization of these stochastic variables, $f = \{f_1 \ldots, f_i, \ldots, f_m\}$, take on a value within a set of labels $\mathcal{L}$, i.e. $f_i \in \mathcal{L}$. These labels are a generalization of pixel values, and example label sets from computer vision are given in Table 8.1. Since f is a specific assignment of labels, it is among others referred to as a labeling. The set of random variables,

$$\mathcal{F} = \{F_1 \ldots, F_i, \ldots, F_m\}, \; i \in \mathcal{S} \;\; , \tag{8.1}$$

is called a random field [Li, 2009]. It is noted, that the arrangements of sites need not necessarily be on a regular lattice or grid, like the pixels of an image. The sites can be irregular and e.g. denote students in a lecture hall and the labels their majors.

8.1.2 Neighborhood Structure

As intuition dictates, Markov random fields are random fields with a Markov property. This Markov property originates from time series analysis and is named after A. Markov. In time series[2] the Markov property is; that all the information about an observation, x_t at time t, present in previous observations, is present in the immediately previous observation, i.e. x_{t-1}. This can be written formally, via probability, in the following manner

$$p(x_t|x_{t-1}, x_{t-2}, \ldots, x_2, x_1) = p(x_t|x_{t-1}) \;\; .$$

When extending this Markov property to a set of sites, we need to define a *neighborhood*, $\mathcal{N}_i$, of a site i. The collection of neighborhoods, $\mathcal{N}_i$, gives a *neighborhood structure*, $\mathcal{N}$. The Markov property for random fields – to be defined more rigorously in the following – is; the information about a site, i present in all other sites is present in the neighborhood of i, $\mathcal{N}_i$.

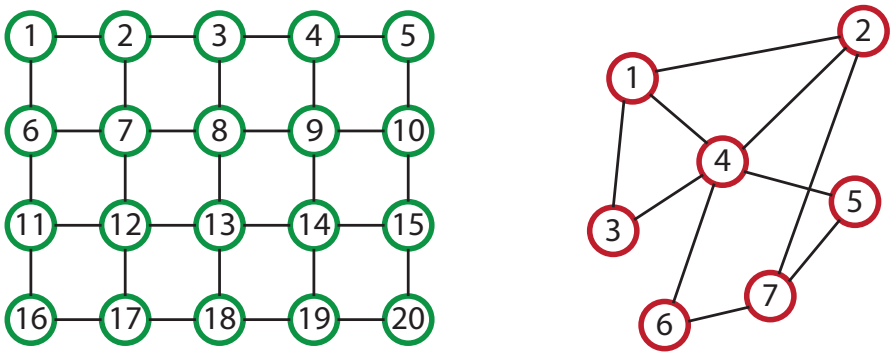


Figure 8.3: Depictions of sites (green and red circles) and neighborhood relations (black lines). **Left:** Regular grid with 4-neighborhood structure. **Right:** Irregular sites with *in*homogeneous neighborhood structure. A black line between two sites indicate that they are neighbors to each other.

A neighborhood structure is defined on a set of sites, and simply consists of a collection of neighborhoods defined for each site. These neighborhoods must follow the two simple rules; a) that no site can be a neighbor to it self, and b) that that if a site j is a neighbor of i then i is a neighbor of j. Both rules correspond with our intuitive understanding of neighbor. Formally a neighborhood structure, $\mathcal{N}$, is defined on a set of sites $\mathcal{S}$, by

$$\begin{aligned} \mathcal{N} &= \{\mathcal{N}_i | \forall i \in \mathcal{S}\} \\ i &\notin \mathcal{N}_i \\ j \in \mathcal{N}_i &\Leftrightarrow i \in \mathcal{N}_j \;\; . \end{aligned} \tag{8.2}$$

As an example consider Figure 8.3-Left where a sample neighborhood structure with 4-neighborhood is depicted, and e.g.

$$\mathcal{N}_8 = \{3, 7, 9, 13\} \quad , \quad \mathcal{N}_{12} = \{7, 11, 13, 17\} \quad \text{and} \quad \mathcal{N}_{16} = \{11, 17\} \;\; .$$

In Figure 8.3-Right an example of an irregular set of sites is presented with an accompanying inhomogeneous neighborhood structure. As an example from Figure 8.3-Right

$$\mathcal{N}_1 = \{2, 3, 4\} \quad , \quad \mathcal{N}_4 = \{1, 2, 3, 5, 6\} \quad \text{and} \quad \mathcal{N}_7 = \{2, 5, 6\} \;\; .$$

[2] Statistics dealing with observations ordered in time.

8.1.3 Markov Random Fields

Given this concept of neighborhoods and neighborhood structure, the Markov property for a set of sites is; that the information about the label of site i, f_i, from all other site labels is contained in the labels of the sites in the neighborhood of i. In terms of probability this can formally be stated as

$$p\left(f_i|f_j\ ,\ j \in \{\mathcal{S} \setminus i\}\right) = p\left(f_i|f_j\ ,\ j \in \mathcal{N}_i\right) \quad , \tag{8.3}$$

where $\mathcal{S}$ is the set of sites. For ease of notation the following convention in used

$$f_{\mathcal{N}_i} = f_j\ ,\ j \in \mathcal{N}_i\ .$$

Thus transforming the Markov property definition to

$$p\left(f_i|f_j\ ,\ j \in \{\mathcal{S} \setminus i\}\right) = p\left(f_i|f_{\mathcal{N}_i}\right) \quad , \tag{8.4}$$

Based on this definition a *Markov random field* is a random field with the Markov property, with the added condition that *all* configurations $\mathcal{F} = f$ should be possible, i.e.

$$\forall f \quad p(\mathcal{F} = f) > 0\ . \tag{8.5}$$

In this context it should be noted that configurations can have so small probabilities, e.g. $10e^{-8}$, that they are practically zero. For an overview of Markov random fields cf. Table 8.2. In summary, MRF is a local modeling framework formulated in terms of a neighborhood structure, and as such well suited for capturing the local homogeneity constraints needed for regularization, as described in the introduction to this chapter. The local property of the MRF framework does, however, *not* imply that it cannot be used to capture more global phenomena, as illustrated in the Ising example below.

8.1.4 The Ising Model – An Example

The *Ising model*, proposed by E. Ising to explain magnetic phenomena in iron in the 1920's, is one of the classical and simplest MRF models, which is also widely used in computer vision. This model is based on a binary labeling, i.e. the label set, $\mathcal{L}$, is given by

$$\mathcal{L} = \{0, 1\} \quad \text{or} \quad \mathcal{L} = \{-1, 1\}\ ,$$

the choice of which is of no fundamental consequence for the model. The typical $\mathcal{L}$ is $\{0, 1\}$ but the intuition of the model is easier understood for $\mathcal{L} = \{-1, 1\}$. So in this example $\mathcal{L} = \{-1, 1\}$ is used, in the rest of this text $\mathcal{L} = \{0, 1\}$.

The Ising model is based on a four neighborhood structure on a regular lattice as seen in Figure 8.3-Left. Setting this in an image framework the individual pixels are sites, indexed by i, $i \in \mathcal{S}$. The neighbors, $\mathcal{N}_i$, of a given pixel i, are the pixels above, below, to the right and left of it. This is excluding the cases where these neighboring pixels do not exist because pixel i is located on the border of the image, e.g. $i = 10$ in Figure 8.3-Left. The probability distribution function associated with this model is given by

$$p\left(f_i|f_{\mathcal{N}_i}\right) = \frac{\exp\left(\alpha_i f_i + \sum_{j\in\mathcal{N}_i} \beta_{ij} f_i f_j\right)}{\sum_{f_i\in\mathcal{L}} \exp\left(\alpha_i f_i + \sum_{j\in\mathcal{N}_i} \beta_{ij} f_i f_j\right)} \quad , \tag{8.6}$$

where the α_i and β_{ij} are given constants or parameters of the model, one for each site and neighborhood pair respectively. To dissect (8.6), note first that the denominator is a normalization constant summing over all possible instances of the numerator such that the combined probability will be one, i.e.

$$\sum_{f_i\in\mathcal{L}} p\left(f_i|f_{\mathcal{N}_i}\right) = p\left(-1|f_{\mathcal{N}_i}\right) + p\left(1|f_{\mathcal{N}_i}\right) = 1\ .$$

Furthermore note, that for a given f_i a high value of

$$\alpha_i f_i + \sum_{j\in\mathcal{N}_i} \beta_{ij} f_i f_j \quad , \tag{8.7}$$

Random Field:
Given a set of sites $i \in \mathcal{S}$ and corresponding random variables F_i associated with this set of sites. Let these random variables have the outcome f_i among set of labels $\mathcal{L}$. A random field is the this family of random variables, (8.1)

$$\mathcal{F} = \{F_1 \ldots, F_i, \ldots, F_m\},\ i \in \mathcal{S} \ .$$

Neighborhood Structure:
Given a set of sites $\mathcal{S}$, a neighborhood structure $\mathcal{N}$ is a collection of neighborhoods $\mathcal{N}_i$ defined by (8.2)

$$\begin{aligned} \mathcal{N} &= \{\mathcal{N}_i | \forall i \in \mathcal{S}\} \\ i &\notin \mathcal{N}_i \\ j \in \mathcal{N}_i &\Leftrightarrow i \in \mathcal{N}_j \ . \end{aligned}$$

Markov Property:
Given a random field, it has the Markov property if, by (8.4)

$$p\left(f_i | f_j\ ,\ j \in \{\mathcal{S} \setminus i\}\right) = p\left(f_i | f_{\mathcal{N}_i}\right) \ .$$

Markov Random Field:
A Markov random field is a random field with the Markov property *and* the positivity constraint, (8.5)

$$\forall f \quad p(\mathcal{F} = f) > 0 \ .$$

Table 8.2: Definition of Markov random fields and associated entities.

in (8.6) gives a high probability for that f_i. In this regard β_{ij} are seen as controlling homogeneity, in that for two neighboring pixels i and j, the possible values for $f_i f_j$ are

$$-1 \cdot -1 = 1 \quad , \quad -1 \cdot 1 = -1 \quad , \quad 1 \cdot -1 = -1 \quad , \quad 1 \cdot 1 = 1 \ .$$

Thus if i and j have the same label then $f_i f_j = 1$ and $f_i f_j = -1$ if i and j have opposite labels. As a consequence β_{ij} is added to (8.7) if i and j have the same label and $-\beta_{ij}$ if not. Thus for positive β_{ij} homogeneity is enforced, and the the larger the value β_{ij} the larger the effect. In this regard the α_i regulate the probability of pixel i having a positive or negative label, f_i, disregarding its neighbors.

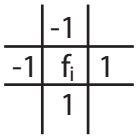


Figure 8.4: Setting used to adjust the α parameter for the samples of the Ising model shown in Figure 8.5.

The model is typically also formulated with a great deal of homogeneity in its parametrization, i.e. that all the α_i are the same and similarly for all the β_{ij}. In Figure 8.5 samples of the Ising model are shown. Here all β_{ij} corresponding to vertical neighbor pairs are set to the value β_v and all the β_{ij} corresponding to horizontal neighbor pairs are set to β_h. In these examples, all the α are set to zero, thus in the setting of Figure 8.4, where the left and upper neighbors are -1 and the right and lower neighbors are 1, there is an equal probability of pixel i having label, f_i, equal to -1 and 1. In particular, inserting these parameter values into (8.6)[3]

$$\begin{aligned}
p\left(-1|f_{\mathcal{N}_i} = \{-1,-1,1,1\}\right) &= \frac{\exp\left(\beta_v \cdot -1 \cdot -1 + \beta_v \cdot -1 \cdot 1 + \beta_h \cdot -1 \cdot -1 + \beta_h \cdot -1 \cdot 1\right)}{\sum_{f_i \in \{-1,1\}} \exp\left(\beta_v \cdot f_i \cdot -1 + \beta_v \cdot f_i \cdot 1 + \beta_h \cdot f_i \cdot -1 + \beta_h \cdot f_i \cdot 1\right)} \\
&= \frac{\exp\left(0\right)}{\sum_{f_i \in \{-1,1\}} \exp\left(0\right)} = \frac{1}{\sum_{f_i \in \{-1,1\}} 1} = \frac{1}{2} \\
p\left(1|f_{\mathcal{N}_i} = \{-1,-1,1,1\}\right) &= \frac{\exp\left(\beta_v \cdot 1 \cdot -1 + \beta_v \cdot 1 \cdot 1 + \beta_h \cdot 1 \cdot -1 + \beta_h \cdot 1 \cdot 1\right)}{\sum_{f_i \in \{-1,1\}} \exp\left(\beta_v \cdot f_i \cdot -1 + \beta_v \cdot f_i \cdot 1 + \beta_h \cdot f_i \cdot -1 + \beta_h \cdot f_i \cdot 1\right)} \\
&= \frac{\exp\left(0\right)}{\sum_{f_i \in \{-1,1\}} \exp\left(0\right)} = \frac{1}{\sum_{f_i \in \{-1,1\}} 1} = \frac{1}{2}
\end{aligned}$$

From the samples in Figure 8.5 a couple of things should be noted. Firstly, that this is a stochastic entity, so that even though the parameters are the same for all images in a row, the images are not the same. Secondly, it is seen that positive β enforce homogeneity or coherence in the pixel labels, and negative do the opposite. As an example, the first and second rows both have only positive β, but the numerical value is larger in the second row, where it is seen that the homogeneity is much larger. In the third and fourth rows one or both of the β have negative values, enforcing inhomogeneity in the corresponding direction. That is, that the third row has a line like pattern corresponding to one negative β and the fourth row has a checker board like pattern corresponding to both β being negative. Lastly, even though the Ising model is very local, in that a pixel only *directly* depends on its four neighbors, it indirectly has dependencies much further, e.g. in the second row a pixel is very likely to have the same label as one ten or twenty pixels away.

In regards to texture modeling, Figure 8.5 hopefully also hints at how MRF can be used to simulate and capture similar but not identical textures. It should also be mentioned, that the Ising model is a special case of the *auto models* [Li, 2009] and the *Potts model* [Potts, 1952]. The latter is the topic of Section 8.2.3.

8.2 Gibbs Random Fields & MAP-MRF

As is apparent from the above, MRF express distributions on fields, e.g. images as a dominant example. In computer vision we are, however, often not interested in sampling from a distribution, as done in Figure 8.5, but in finding the best or optimal solution to a problem, e.g. the optimal labeling in a segmentation problem. In

[3] There is nothing canonical about the order in which the neighbors are listed here.

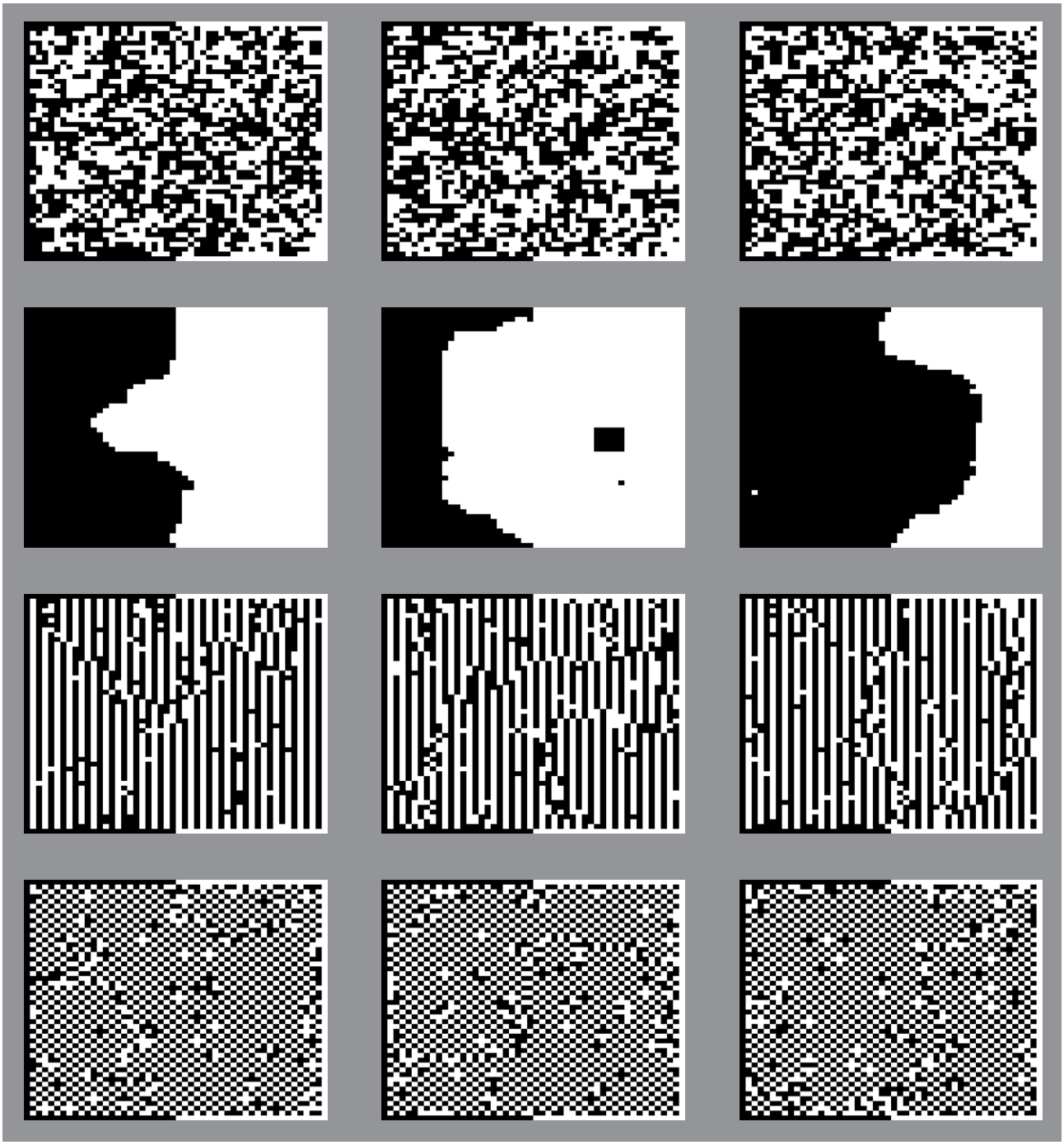


Figure 8.5: Samples from the Ising model (8.6), with different parameters of β_h and β_v for each row, $\alpha = 0$. In the first row $(\beta_h, \beta_v) = (0.1, 0.1)$, in the second row $(\beta_h, \beta_v) = (1, 1)$, in the third row $(\beta_h, \beta_v) = (-0.5, 0.5)$ and in the fourth row $(\beta_h, \beta_v) = (-0.5, -0.5)$. In all twelve samples the border was forced to be half white and half black to ensure dynamics in the samples. As such, the border pixels were not part of the stochastic sampling.

MRF this translates into finding the Maximum likelihood (ML) solution or maximum a posteriori probability (MAP) estimate[4], cf. Appendix B.

In order to compute these MAP estimates it is typically used that any given Markov random Field(MRF) is equivalent to a so called Gibbs random field. In that it is typically easier to find the optima of Gibbs random fields. Here these Gibbs random fields will be introduced, and used to find optimal solutions in the following

[4]ML and MAP are principally the same. ML is typically used by people you do not use priors and MAP by people who do. So ML is sometimes assumed to imply that priors are not used and vice versa. Such implications are not made in his text and the two terms are used interchangeably.

sections. Gibbs random fields are named after J. W. Gibbs and originate from statistical mechanics, and have more implications for the treatment of MRF than optimization cf. [Winkler, 2003].

The equivalence between the Markov and Gibbs random fields is established by the *Hammersley-Clifford theorem*, cf. [Clifford, 1990]. It states that; *a random field $\mathcal{F}$ on the sites $\mathcal{S}$ is a Markov random field with respect neighborhood structure $\mathcal{N}$ if and only if it is a Gibbs random with respect to the same neighborhood structure*. The proof of this theorem is beyond the scope of this text, and the interested reader is referred to e.g. [Li, 2009, Winkler, 2003].

8.2.1 Cliques

Gibbs random fields are defined in terms of *cliques*, which will be defined here. A clique is either all the single sites, known as one cliques, or sets of sites which are all neighbors to each other. Thus the set of one cliques, $\mathcal{C}_1$, is equal to the set of sites, i.e.

$$\mathcal{C}_1 = \mathcal{S} \ .$$

In line with this notation a taxonomy of multi-site cliques is given by the number of sites they contain. So the two cliques, $\mathcal{C}_2$, are pairs of sites, $\{i, j\}$, which are neighbors to each other, i.e.

$$i \in \mathcal{N}_j \quad \text{and} \quad j \in \mathcal{N}_i \ .$$

Due to the definition of neighborhood, cf. (8.2), these two statements are equivalent and

$$\mathcal{C}_2 = \{i, j | i \in \mathcal{S} \ , \ j \in \mathcal{N}_i\} \ .$$

Similarly the three cliques, $\mathcal{C}_3$, are given by

$$\mathcal{C}_3 = \{i, j, k | i \in \mathcal{S} \ , \ j \in \mathcal{N}_i \ , \ k \in \mathcal{N}_i \ , \ k \in \mathcal{N}_j\} \ ,$$

and so on. The complete set of cliques, $\mathcal{C}$, for a set of sites, $\mathcal{S}$, with a neighborhood structure,$\mathcal{N}$, is then given by

$$\mathcal{C} = \mathcal{C}_1 \cup \mathcal{C}_2 \cup \mathcal{C}_3 \cup \ldots \tag{8.8}$$

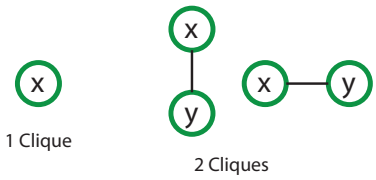


Figure 8.6: The clique types of Figure 8.3-Left.

As an example consider the neighborhood structures from Figure 8.3. For the regular grid with 4 connected neighborhood of Figure 8.3-Left, the clique types are shown in Figure 8.6. Here all the single sites will give rise to a one cliques, also all connected horizontal and vertical pairs give rise to two cliques. In this example there are no clique of order three or higher. To address a common misconception, it is noted that e.g. $(2, 6, 7)$ is *not* a clique even though both 2 & 7 and 6 & 7 are neighbors, this is so since 2 & 6 are *not* neighbors.

In the case of Figure 8.3-Right, there are the following cliques:

$$\begin{aligned} \mathcal{C}_1 &= \{1, 2, 3, 4, 5, 6, 7\} \\ \mathcal{C}_2 &= \{(1, 2), (1, 3), (1, 4), (2, 4), (2, 7), (3, 4), (4, 5), (4, 6), (5, 7), (6, 7)\} \\ \mathcal{C}_3 &= \{(1, 2, 4), (1, 3, 4)\} \ . \end{aligned}$$

Here it is noted that cliques are *un*ordered sets [Besag, 1974], such that e.g. $(1, 2)$ and $(2, 1)$ are not two distinct cliques in the above example.[5]

[5]There is however some uncertainty about cliques being ordered or unordered in the literature, c.f. [Li, 2009]. Here it is chosen to define them as unordered, because this is consistent with how MAP-MRF are typically implemented.

8.2.2 Gibbs Random Fields

Given the concept of cliques *Gibbs Random field* can be introduced. These are random fields, $\mathcal{F}$, with a Gibbs distribution, were the Gibbs distribution is given by

$$\begin{aligned} p(\mathcal{F}=f) &= Z^{-1}\cdot\exp\left(\frac{-U(f)}{T}\right) \\ Z &= \sum_{\text{all } f}\exp\left(\frac{-U(f)}{T}\right) \\ U(f) &= \sum_{c\in\mathcal{C}} V_c(f) \ . \end{aligned} \tag{8.9}$$

Here $U(f)$ is termed the energy function, which is to be formulated as a sum of *clique potentials* $V_c(f)$. Where $V_c(f)$ should be understood as an (potential) energy function of the labels assigned to the elements of clique c, given the labeling f[6]. The Z term is a normalization constant, which assures that the combined probability is one, i.e.

$$\sum_{\text{all } f} p(\mathcal{F}=f) = 1 \ .$$

The scalar variable T is interpreted as the temperature, and is very important for simulated annealing, cf. Section 8.6.

In essence the Gibbs random fields allow us to attach energy or desirability to different labeling of cliques. In image segmentation the one cliques are typically used to encapture the data term, which typically does not have any thing to do with the neighborhood structure. Similarly the higher order cliques are typically used to express our a priori assumption about a problem, e.g. homogeneity. As an example, the energy term, $U(f)$ corresponding to the Ising model above is[7]

$$U(f) = -\sum_{i\in\mathcal{C}_1}\alpha_i f_i - \sum_{(i,j)\in\mathcal{C}_2}\beta_{ij} f_i f_j \ , \tag{8.10}$$

which, when plugged into (8.9), is seen to correspond nicely with (8.6) for $T=1$[8]. In this case, the clique potentials are

$$\begin{aligned} V_i(f_i) &= -\alpha_i f_i \qquad \text{for} \qquad i\in\mathcal{C}_1 \\ V_{(i,j)}(f_i,f_j) &= -\beta_{ij} f_i f_j \qquad \text{for} \qquad (i,j)\in\mathcal{C}_2 \ . \end{aligned}$$

Since the exponential function is increasing, i.e. its gradient is always positive, it is seen that solving

$$\max_f p(\mathcal{F}=f) \ ,$$

in (8.9) is equivalent to solving

$$\max_f \frac{-1}{T}U(f) \ ,$$

for a fixed T. This is again equivalent to

$$\min_f U(f) \ .$$

Thus, we can chose to minimize an energy instead of maximizing a probability, or vice versa. from a practical point of view, Gibbs random fields are easier to work with, in that it is often easier to construct numerical algorithms to work with them. This is, thus, a main motivation for introducing the Gibbs random field.

[6] Strictly speaking the $V_c(f)$ should be positive, although this is often disregarded in computer vision. This lack of rigor can be mended by the addition of a constant, which will typically(all cases the author is aware of) have no practical effect on the solution, since we are only interested in the difference between energy states.

[7] Note that the change of sign is to counter the $\frac{-1}{T}$ term.

[8] There is a subtle point that the β should be multiplied by two for a perfect correspondence with (8.6). The reason being that the cliques are unordered sets, and e.g. β_{ij} should count in a version of (8.6) for both site i and cite j.

8.2.3 MAP-MRF Applied to Image Segmentation – The Potts Model

The Potts model, named after R.B. Potts, is a generalization of the Ising Model, c.f. Section 8.1.4, to more than two labels. As exemplified below, c.f. Figure 8.9, it is very often used as a prior or regularization term in computer vision for segmentation or labeling problems. That is, assume we have a pixel-wise classification function that supplies the probability or energy of a pixel having one of several labels (e.g. classes). This pixel-wise function can then be formulated as a one clique potential, whereby it fits well into the Gibbs random field framework. The Potts model can then be used as a homogeneity prior by use of two cliques, just as the two cliques did in the Ising model. The two clique potential in the Potts model is given by

$$V_{(i,j)}(f_i, f_j) = \begin{cases} -\tilde{\beta}_{ij} & \text{for} \quad f_i = f_j \\ \tilde{\beta}_{ij} & \text{for} \quad f_i \neq f_j \end{cases} \quad ,$$

where the $\tilde{\beta}_{ij}$ is a parameter denoting the amount of desired homogeneity. This two clique potential can be rewritten via the Kronecker delta function, $\delta_k(x)$, given by

$$\delta_k(x) = \begin{cases} 1 & \text{for} \quad x = 0 \\ 0 & \text{for} \quad x \neq 0 \end{cases} \quad .$$

In this case the two clique potential of the Potts model can be written as

$$V_{(i,j)}(f_i, f_j) = \tilde{\beta}_{ij} - 2 \cdot \tilde{\beta}_{ij}\delta_k(f_i - f_j) = 2 \cdot \tilde{\beta}_{ij} \left(1 - \delta_k(f_i - f_j)\right) + \tilde{\beta}_{ij} \quad .$$

Since we are interested in minimizing the energy, the problem is indifferent to the addition of a constant and we can remove it. The two clique potential of the Potts model can thus be written as

$$V_{(i,j)}(f_i, f_j) = \beta_{ij} \left(1 - \delta_k(f_i - f_j)\right) \quad , \tag{8.11}$$

with $\beta_{ij} = 2 \cdot \tilde{\beta}_{ij}$. As depicted in Figure 8.7, equation (8.11) can be interpreted as an error function applied to the difference between the labels, i.e. $f_i - f_j$. This, hopefully, illustrates that an extension of the Potts model to other types of regularization, can be made by use of other norms or error functions then the one used in (8.11). Typical other norms are treated in Section 5.4.1. Note, that *if* the error function or norm is changed, and thus also the two clique potential, we no longer refer to the model as a Potts model.

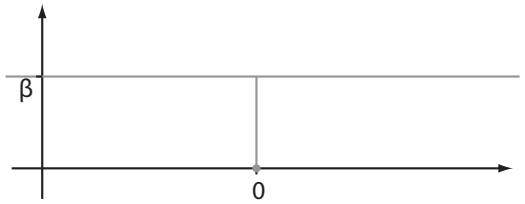


Figure 8.7: The error function of the difference between the labels, i.e. $f_i - f_j$, in (8.11). The function maps to β for all values other then zero, for which it maps to zero.

As an example, consider segmenting the image of Figure 8.8-Left, into three classes; (CSF) fluid, white matter and gray matter. In this example it is assumed, that the pixels belonging to each class follow Normal distributions with mean and standard deviation as specified in Table 8.3, c.f. Section B.4. These distributions are plotted in Figure 8.8-Right. From this plot it is seen that a maximum likelihood pr. pixel estimate would give the ranges or thresholds given in the right most column of Table 8.3. These thresholds are determined by the pixel value intervals in which a given label is most probable. The result of this pr. pixel classification is shown in Figure 8.9-Top Left, denoted $\beta = 0$.

As seen from the top left of Figure 8.9, the pixel wise classification gives a result with a lot of small isolated dots, which compared to our expectation are misclassifications. Apart from image noise, an explanation for this is that it is quite probable that pixels take on values where their true class is not the most probable, c.f. Figure 8.8-Right. As an example, a pixel of white matter might very well have a pixel value of 38, by which it would be classified as fluid on a per pixel basis. To address this issue, we can use the Potts model as a prior, hereby pixel classifications will be biased towards the classes of their neighbors, thus hopefully solving these borderline cases.

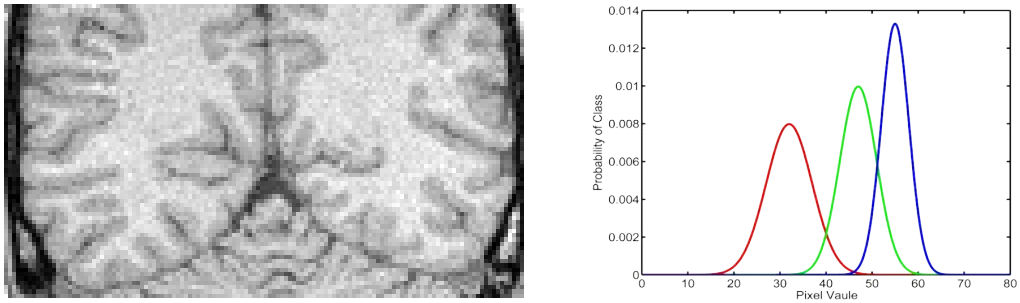


Figure 8.8: Brain segmentation example. **Left:** Image. Note, that the gray values have been stretched for better viewing. **Right** The assumed probability distribution functions (pdf) for the pixel values belonging to the three different classes; *red:* Fluid, *green:* Gray Matter and *blue:* White Matter, c.f. Table 8.3.

Class	Mean, μ	Std. Dev. σ	$pdf\,(x)$	$-\log(pdf\,(x))$	Range
Fluid	32	5	$\frac{1}{\sqrt{2\pi 5^2}}\exp^{-\frac{1}{2}\left(\frac{x-32}{5}\right)^2}$	$\log\left(\sqrt{2\pi 5^2}\right)+\frac{1}{2}\left(\frac{x-32}{5}\right)^2$	$0-40.0$
Gray Matter	47	4	$\frac{1}{\sqrt{2\pi 4^2}}\exp^{-\frac{1}{2}\left(\frac{x-47}{4}\right)^2}$	$\log\left(\sqrt{2\pi 4^2}\right)+\frac{1}{2}\left(\frac{x-47}{4}\right)^2$	$40.0-51.1$
White Matter	55	3	$\frac{1}{\sqrt{2\pi 3^2}}\exp^{-\frac{1}{2}\left(\frac{x-55}{3}\right)^2}$	$\log\left(\sqrt{2\pi 3^2}\right)+\frac{1}{2}\left(\frac{x-55}{3}\right)^2$	$51.1-255$

Table 8.3: Distributions of the pixel values corresponding to the three classes of Figure 8.8-Left. The right most column is the corresponding ranges of pixel values for which the given class is most probable.

To set up this Potts model we first have to compute the one clique potentials corresponding to the data term of Table 8.3. Assuming $T=1$. To do this the one clique potentials are set equal to $-\log(pdf\,(x))$ i.e.

$$\text{for}\quad i\in\mathcal{C}_1\quad,\quad V_i(f_i)=\begin{cases}\log\left(\sqrt{2\pi 5^2}\right)+\frac{1}{2}\left(\frac{x-32}{5}\right)^2 & \text{for}\quad f_i=\text{Fluid}\\ \log\left(\sqrt{2\pi 4^2}\right)+\frac{1}{2}\left(\frac{x-47}{4}\right)^2 & \text{for}\quad f_i=\text{Gray Matter}\\ \log\left(\sqrt{2\pi 3^2}\right)+\frac{1}{2}\left(\frac{x-55}{3}\right)^2 & \text{for}\quad f_i=\text{White Matter}\end{cases}.$$

In this way, if e.g. a pixel is labeled as Gray Matter it's contribution to the probability in (8.9) will be

$$\exp\left(\frac{-1}{T}V_i(f_i=\text{Gray Matter})\right)=\exp\left(\frac{-1}{1}\log\left(\sqrt{2\pi 4^2}\right)+\frac{1}{2}\left(\frac{x-47}{4}\right)^2\right)=\frac{1}{\sqrt{2\pi 4^2}}\exp^{-\frac{1}{2}\left(\frac{x-47}{4}\right)^2},$$

equaling the pdf for Gray Matter as expected. Combining this with (8.11), we get an energy function of

$$U(f)=\sum_{i\in\mathcal{C}}V_i(f_i)+\sum_{(i,j)\in\mathcal{C}_2}V_{(i,j)}(f_i,f_j)\ ,$$

giving a probability distribution function for the labeling of the image in Figure 8.8-Left. We are, however, not interested in this probability distribution as such, but more in finding the most probable or MAP labeling or classification. I.e. we want the same best labeling each time we do the classification in this manner. The result of solving this MAP-MRF problem is illustrated in Figure 8.9 for different values of β. The algorithm used is the subject of Section 8.5. In Figure 8.9 it is seen that the larger the value of β the more homogeneity amongst the neighbors is achieved, and for a large enough β all pixels will have the same label. Similarly for $\beta=0$ the effect of the two cliques are removed corresponding to the clique potential being zero in al cases. As for the value of β giving the best classification or regularization; this is a matter of modeling e.g. trial and error or data fitting. The subject of the latter is beyond the scope here, and the interested reader is referred to [Li, 2009, Winkler, 2003].

The Potts model as described above, can be seen in the context of Bayes's rule that states that, cf. (B.36)

$$p(m_i|d)\propto p(d|m_i)p(m_i)\ ,$$

where $p(d|m_i)$ is denoted the data term and $p(m_i)$ the prior term. In this regard the 1-cliques are the data term or our observation model, and the 2-cliques describes our homogeneity assumption or prior.

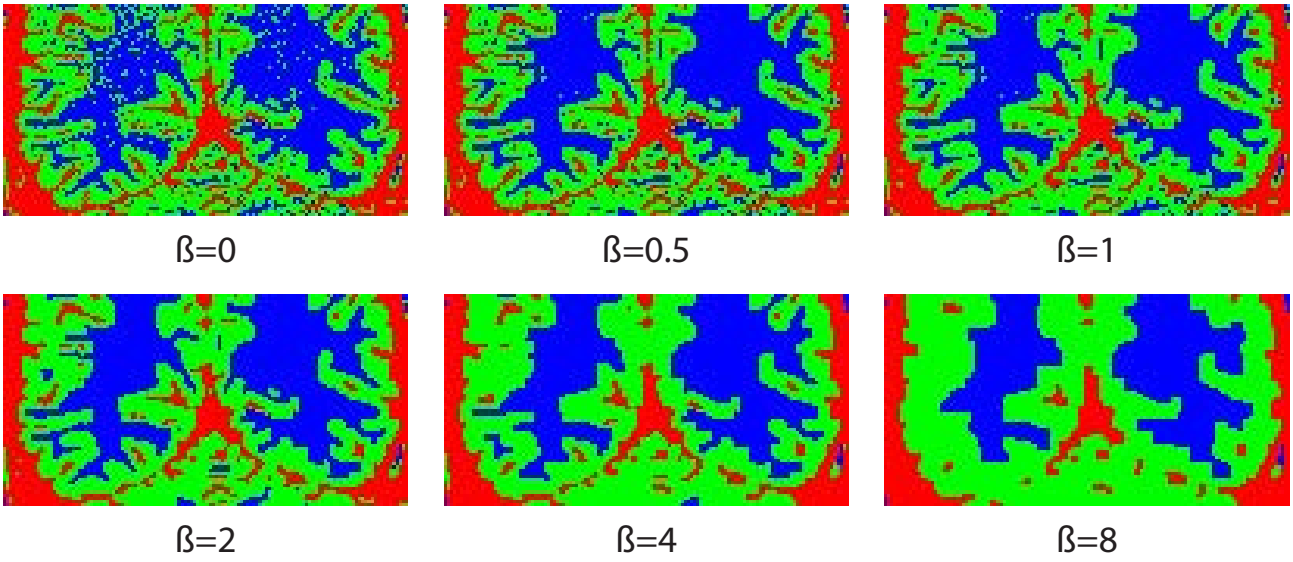


Figure 8.9: Labelings or classification of Figure 8.8-Left for different values of β in the Potts model. Note, that a β of zero corresponds to a removing the Potts regularization term equaling a pure pixel-wise classification.

8.3 Max Flow and Minimum Cut

In order to find the MAP solutions to MRF problems, as e.g. posed above, optimization algorithms are required. In the case of discrete labeled MRF problems, the probably most popular optimization schemes are based on the *maximum flow* and *minimum cut* algorithms of graphs. These algorithms will be outlined here, and the link to solving MAP MRF problems will be presented in the following two sections.The presentation level of max flow/ min cut algorithms here is only aimed at giving the intuitive understanding needed for use in a MRF setting. For further material on graph algorithms, and the max flow / min cut in particular, please refer to [Cormen et al., 2001]. Probably the state of the art algorithm for MRF computer vision problems is that of [Boykov and Kolmogorov, 2004], good implementations of which are freely available on the internet, e.g. via the boost library. Optimization methods not based on graph cuts will be covered briefly in Section 8.6.

The max flow problem is related to a (typically) directed graph as e.g. seen in Figure 8.10. A graph consist of a set of vertices, $v_i \in \mathcal{V}$, and a set of edges $e_{ij} \in \mathcal{V} \times \mathcal{V}$, connecting pairs of these vertices. If the edges, and thus the graph, are directed, then the edge from e_{ij} from v_i to v_j is not the same as the edge e_{ji} from vertex v_j to vertex v_i. Often times weights w_{ij} are associated with the edges, as indicated in Figure 8.10.

In the max flow problem one of the vertices is denoted the source, $s \in \mathcal{V}$, and one is denoted the sink, $t \in \mathcal{V}$. The aim is then to compute how much flow can be transported from source to sink through the graph, where the weights, w_{ij}, denote capacities. As an example we could consider the edges as valved water pipes, and their associated weights the transport capacity, e.g. liters pr. minute, of the respective pipes. The max flow problem would then be; how much water could be pumped through this pipe system from source, s, to sink t. More formally cf. [Cormen et al., 2001], denote the flow through edge e_{ij} by f_{ij}, then we want to maximize the flow out of the source, s (equivalent to the flow into the sink), i.e.

$$\max \sum_i f_{si} \ , \tag{8.12}$$

subject to the following conditions

1. **Capacity Constraint**, that a flow can not be negative or exceed its capacity, i.e.

$$\forall e_{ij} \qquad 0 \leq f_{ij} \leq w_{ij} \ . \tag{8.13}$$

2. **Flow Conservation**, that, apart from the source and the sink, flow cannot be created or disappear, implying that the flow into a vertex should equal the flow out of a vertex, i.e. for any non source or sink vertex

$$\forall v_i \in \mathcal{V} \backslash \{s,t\} \qquad \sum_j f_{ij} = \sum_k f_{ki} \ . \tag{8.14}$$

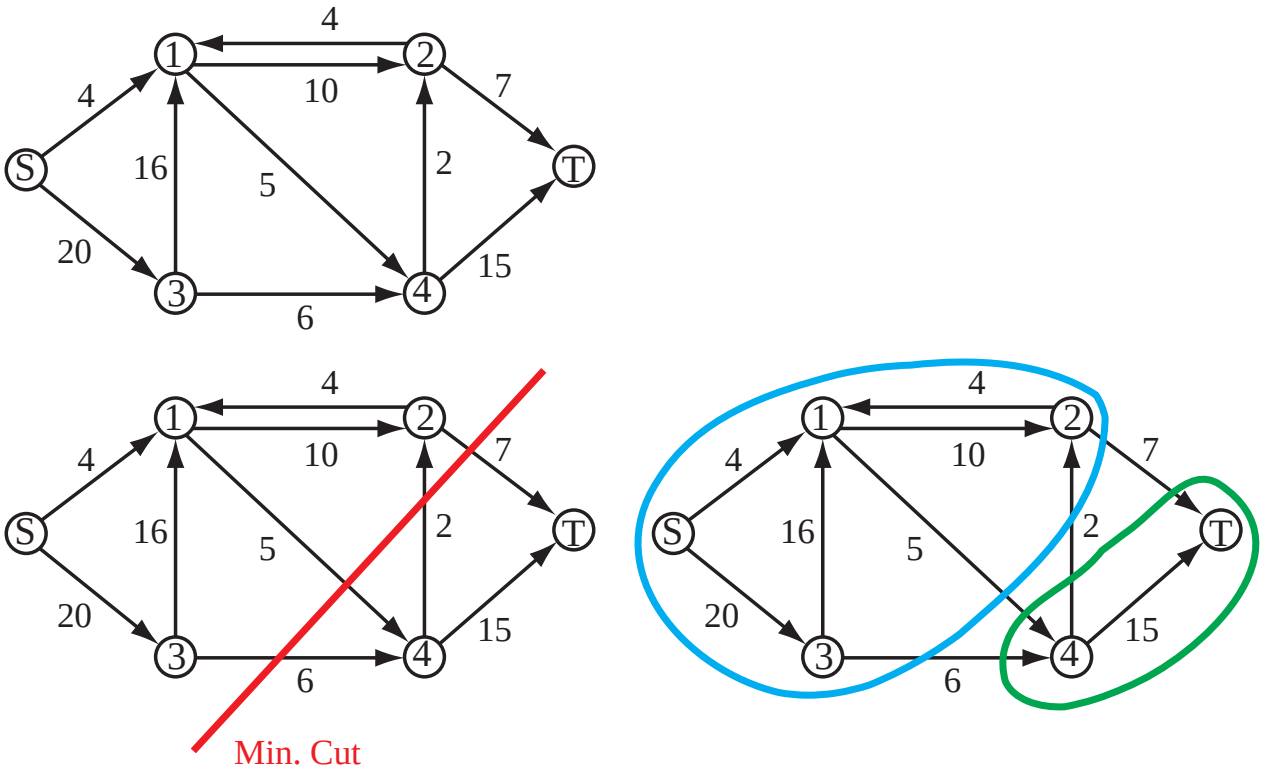


Figure 8.10: An example of a maximum flow/ minimum cut problem in the form of the directed graph illustrated at the **top** of the figure. The non-encircled numbers are edge weights. The minimum cut solution is illustrated in the bottom row, where **bottom-left** is the minimum cut with a capacity of 18, and **bottom-right** is the partition associated with this minimum cut.

8.3.1 The Ford Fulkerson Algorithm

Arguably the archetypical max flow algorithm is the Ford Fulkerson algorithm [Cormen et al., 2001] [Ford and Fulkerson, 1962]. This algorithm is covered here due to it's simplicity with the aim of giving an intuitive understanding of the max flow problem and it's close link to the min flow problem. As mentioned earlier the algorithm of [Boykov and Kolmogorov, 2004] is probably the state of the art, but not as straight forward as the Ford Fulkerson algorithm. In Figure 8.11 it is demonstrated how the Ford Fulkerson algorithm operates on the graph from Figure 8.10. The basic concept of this algorithm is; to start out with a graph with no flow, and then incrementally find paths through the graph where there is extra capacity. The steps of the algorithm are as follows, given a directed graph consisting of vertices, $v_i \in \mathcal{V}$, edges $e_{ij} \in \mathcal{V} \times \mathcal{V}$, and weights w_{ij}, associated with the edges:

1. Associate with each edge, e_{ij}, a residual flow, r_{ij}, denoted by the blue numbers in Figure 8.11. These residual flows, r_{ij}, denote how much residual capacity there is left in the edges. They are initialized to the weights, i.e. all $r_{ij} = w_{ij}$.

2. Find a path from source, s, to sink, t, via edges with *non-zero* residual flow, denoted by blue in Figure 8.11. This can be done efficiently via a depth first search, cf. [Cormen et al., 2001]. If such a path does not exist, it is not possible to push more flow through the system, and the algorithm terminates. If found, the path can be denoted by the vertices it pases through, i.e. $s, v_a, v_b, v_c \ldots, v_q, t$ and the residual flow of the paths edges is denoted by $\mathbf{r} = \{r_{sa}, r_{ab}, r_{bc}, \ldots, r_{qt}\}$.

3. Find the maximum residual flow, Δr, possible to 'push through' the path found in step 2. This is equal to the minimum of the residual flows of the found path, i.e.

$$\Delta r = \min_{r_{ij} \in \mathbf{r}} r_{ij} \quad .$$

4. Subtract the maximum residual flow, Δr, from all the residual flows in the path, i.e

$$\forall r_{ij} \in \mathbf{r} \quad , \quad r_{ij} = r_{ij} - \Delta r \quad .$$

Note that this will set at least one of the residual flows in the path to zero, thus removing at least one edge from the path searching in step 2. With a finite number of edges this also hints at a proof of termination.

5. Go to step 2.

Upon termination, it is not possible to push more flow through the graph from source, s, to sink, t, and the flow, f_{ij}, of the found max flow solution is given by:

$$f_{ij} = w_{ij} - r_{ij} \ .$$

To exemplify, consider the iterations of the Ford Fulkerson algorithm on the graph of Figure 8.10 as shown in Figure 8.11.

- In the **first** iteration the path is given by s, v_3, v_4, t, $\mathbf{r} = \{20, 6, 15\}$, and $\Delta r = 6$. So all the residuals of the path are lowered by 6, and the residual flow from v_3 to v_4 is set to zero.
- In the **second** iteration the path is given by s, v_1, v_2, t, $\mathbf{r} = \{4, 10, 7\}$, and $\Delta r = 4$. So all the residuals of the path are lowered by 4, and the residual flow from s to v_1 is set to zero. Note that there is an edge from v_2 to v_1 with weight 4, which is not part of the path, due to the directed nature of the graph.
- In the **third** iteration the path is given by s, v_3, v_1, v_4, t, $\mathbf{r} = \{14, 16, 5, 9\}$, and $\Delta r = 5$. So all the residuals of the path are lowered by 5, and the residual flow from v_1 to v_4 is set to zero.
- In the **fourth** iteration the path is given by s, v_3, v_1, v_2, t, $\mathbf{r} = \{9, 11, 6, 3\}$, and $\Delta r = 3$. So all the residuals of the path are lowered by 3, and the residual flow from v_2 to t is set to zero.
- After the fourth iteration it is not possible to find a path from source, s, to sink, t, through edges with *non-zero* residual flow, and the algorithm **terminates**.

It should be noted, that it is of no principal importance in what order the paths from source, s, to sink, t, are processed. Also, the algorithm is guaranteed to converge to the optimum, cf. [Cormen et al., 2001]. Further more this convergence is archived in polynomial time in the size of the problem.

8.3.2 Relation to Minimum Cut

What we are directly interested in, in relation to finding MAP solutions to MRF, is the minimum cut of a graph, cf. [Cormen et al., 2001]. It can, however, be shown, that finding the minimum cut of a graph is equivalent to find the maximum flow, and thus maximum flow algorithms such as the Ford Fulkerson algorithm are often used for finding the minimum cut[9].

A *graph cut*, or plainly *cut*, is a partition of the graphs vertices into to parts, $\mathcal{S}$ and $\mathcal{T}$, such that the source belongs to $\mathcal{S}$, and the sink belongs to $\mathcal{T}$. More formally

$$\begin{aligned} \mathcal{V} &= \mathcal{S} \cup \mathcal{T} \\ \emptyset &= \mathcal{S} \cap \mathcal{T} \\ s &\in \mathcal{S} \\ t &\in \mathcal{T} \ . \end{aligned} \tag{8.15}$$

The capacity of the cut, defines how much flow can come from $\mathcal{S}$ to $\mathcal{T}$. More formally, the edges beginning from a vertex in $\mathcal{S}$ and ending at a vertex in $\mathcal{T}$, can be expressed as

$$e_{ij} \qquad \text{for which} \qquad i \in \mathcal{S} \quad \text{and} \quad j \in \mathcal{T} \ .$$

Note that this does *not* include the edges starting in $\mathcal{T}$ and ending in $\mathcal{S}$. The capacity of a cut is then the sum of the weights of the edges from $\mathcal{S}$ to $\mathcal{T}$, i.e.

$$\text{Capacity} = \sum_{i \in \mathcal{S}, j \in \mathcal{T}} w_{ij} \ . \tag{8.16}$$

[9] Formally, it would be more correct to say that algorithms such as the Ford Fulkerson is both a maximum flow *and* a minimum cut algorithm.

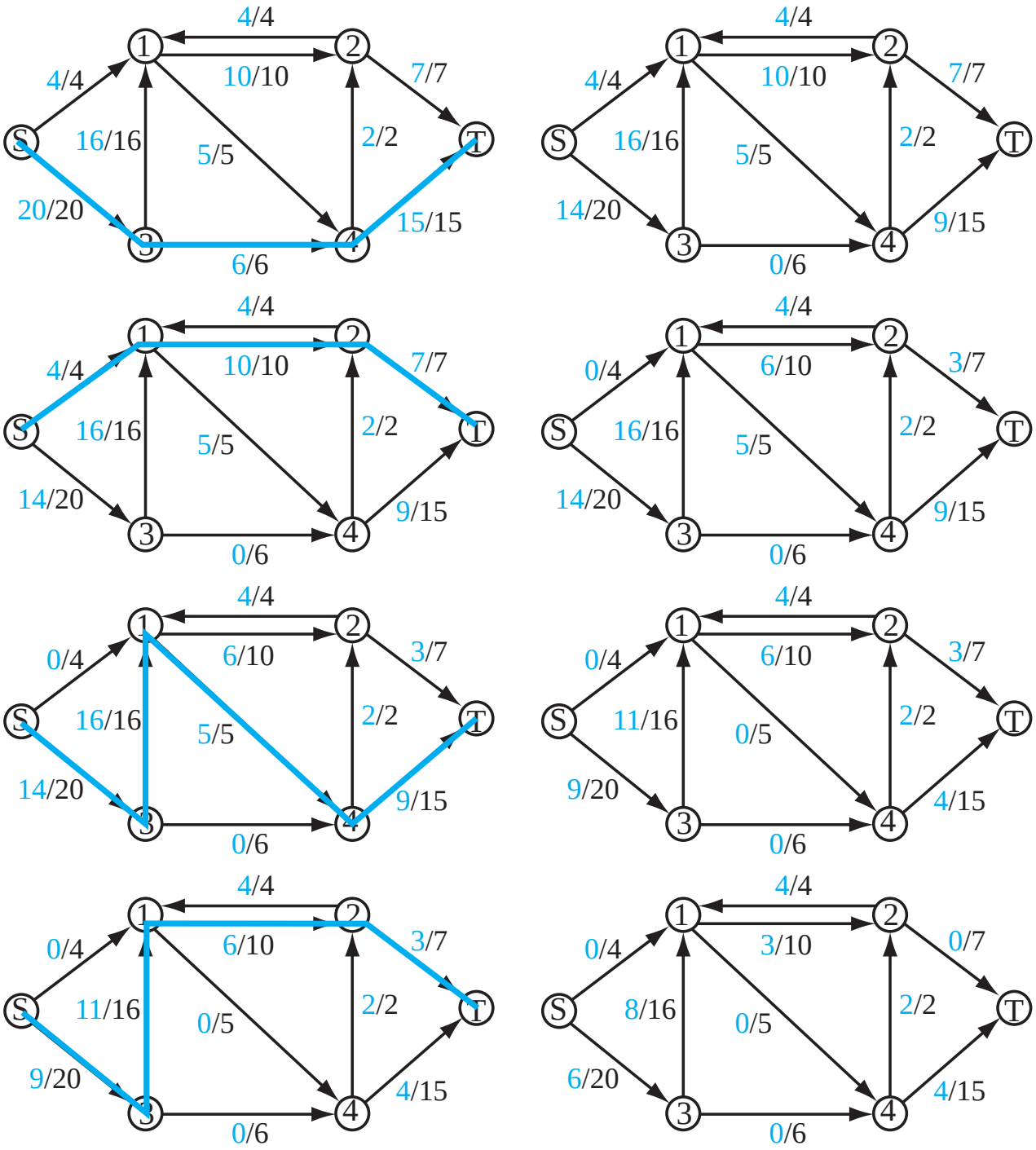


Figure 8.11: Example of running the Ford Fulkerson algorithm on the graph from Figure 8.10, with one iteration per row. Each edge has two numbers associted with it, the blue is the residual weights, and the black are the original edge weights or capacities. To the right the blue lines denote a residual path from source to sink, with non-zero residual weights. To the left, the residual flows have been lowered with Δr. Note, that after the last row or iteration it is impossible to find a path from source to sink through only edges with positive residual flow.

The *Minimum Cut* of a graph is then a cut with a minimum capacity. It is noted that several cuts can have the same minimum capacity, and as such several cuts of a graph can be minimal cuts.

Intuitively, a minimum cut is the bottle neck of the flow from source, s, to sink, t, in a graph. To see this – which also captures the essence of why minimal cut and maximum flow are equivalent – note, that the maximum flow of a graph can not be larger than the capacity of any cut. The reason being, that any flow passing from s to t must also pass from $\mathcal{S}$ to $\mathcal{T}$. The latter being equal to the capacity. Thus, the capacity of a minimum cut is an upper bound on the maximum flow, and is in fact equal to it.

In relation to the Ford Fulkerson algorithm, a minimum cut can be found by finding a partitioning of the graph such that all edges from S to T have a residual flow of zero, [Ford and Fulkerson, 1962]. This is e.g. seen by comparing resulting cut in Figure 8.10 with the result of the Ford Fulkerson algorithm, seen in the last row of Figure 8.11. We thus have efficient algorithms for finding a minimum cut of a graph, guaranteed to reach the global optimum.

8.4 Binary MRF and Graph Cuts

The minimum cut algorithm for graphs can be used to solve binary MAP-MRF problems efficiently, because binary MAP-MRF can be formulated as a graph cut problem, cf. [Kolmogorov and Zabih, 2004]. To give an intuitive understand of how, consider the simple 1D image[10] example seen in Figure 8.12-a. This 1D image has the following pixel values (numbering from left to right)

Pixel number	1	2	3	4	5	6
Pixel value	0.9	0.4	0.7	0.1	0.4	0.6

The binary labels used are $\mathcal{L} = \{0, 1\}$, and the 1-clique potential is given by:

$$V_i(f_i) = \begin{cases} \text{pixel value}_i & \text{for} \quad f_i = 0 \\ 1 - \text{pixel value}_i & \text{for} \quad f_i = 1 \end{cases} .$$

If we just consider the 1-clique potentials, the problem is equivalent to assigning a label of 0 to all pixels with pixel values above 0.5, with the result seen in Figure 8.12-c. This simple thresholding can also be formulated as a graph cut problem, as illustrated in Figure 8.12-b.

The graph in Figure 8.12-b contains eight vertices; a source, s, and a sink, t, and one pixel-vertex for each of the six pixels enumerated from one to six. The weights of the edges connecting the pixel-vertices to the sink, t, are set to the 1-clique potential corresponding to a 0 label. Similarly, the weights of the edges connecting the source, s, to the pixel-vertices, are set to the 1-clique potential corresponding to a 1 label.

The computed minimum cut of the graph in Figure 8.12-b is denoted by the red line. Given the special structure of the graph, a pixel-vertex will be in the same partition as the source if and only if it's 1-clique potential is lowest for a 0 label. To see this, note that the cut will be either above or below the pixel-vertex, depending solely on which edge cost, and thus 1-clique potential, is the lowest. The resulting segmentation, as seen in Figure 8.12-c, is thus achieved by assigning a 0 label to the pixels with a pixel-vertex in the source partition, and a 1 label to the pixels with a pixel-vertex in the sink partition.

Considering the example from Figure 8.12-b further, it is seen that the cuts capacity of this particular graph is equal to the energy

$$U(f) = \sum_{i \in \mathcal{C}} V_i(f_i) \ ,$$

of the formulated Gibbs random. Thus the minimum cut corresponds to the minimal possible energy, and thus the MAP-MRF solution.

The example from Figure 8.12-b, can be extended with a 2-clique potential of

$$V_{(i,j)}(f_i, f_j) = \begin{cases} 0 & \text{for} \quad f_i = f_j \\ 0.1 & \text{for} \quad f_i \neq f_j \end{cases} ,$$

where pixels located next to each other are assumed to be neighbors. The graph corresponding to this Gibbs random field is seen in Figure 8.12-d. Given a cut of the graph, the labeling of a pixel will again be

0 Label if the corresponding pixel-vertex is in the source partition, $\mathcal{S}$.

1 Label if the corresponding pixel-vertex is in the sink partition, $\mathcal{T}$.

With this labeling the cuts capacity is again equal to the energy, $U(f)$, of the formulated Gibbs random field. To see this, note that for the 1-cliques the situation is as with Figure 8.12-b. For the 2-cliques, note that non-zero potentials of 0.1 only happen if the label of two neighboring pixels differ. This is equivalent to the cut making

[10] A 1D image is chosen for ease of illustration.

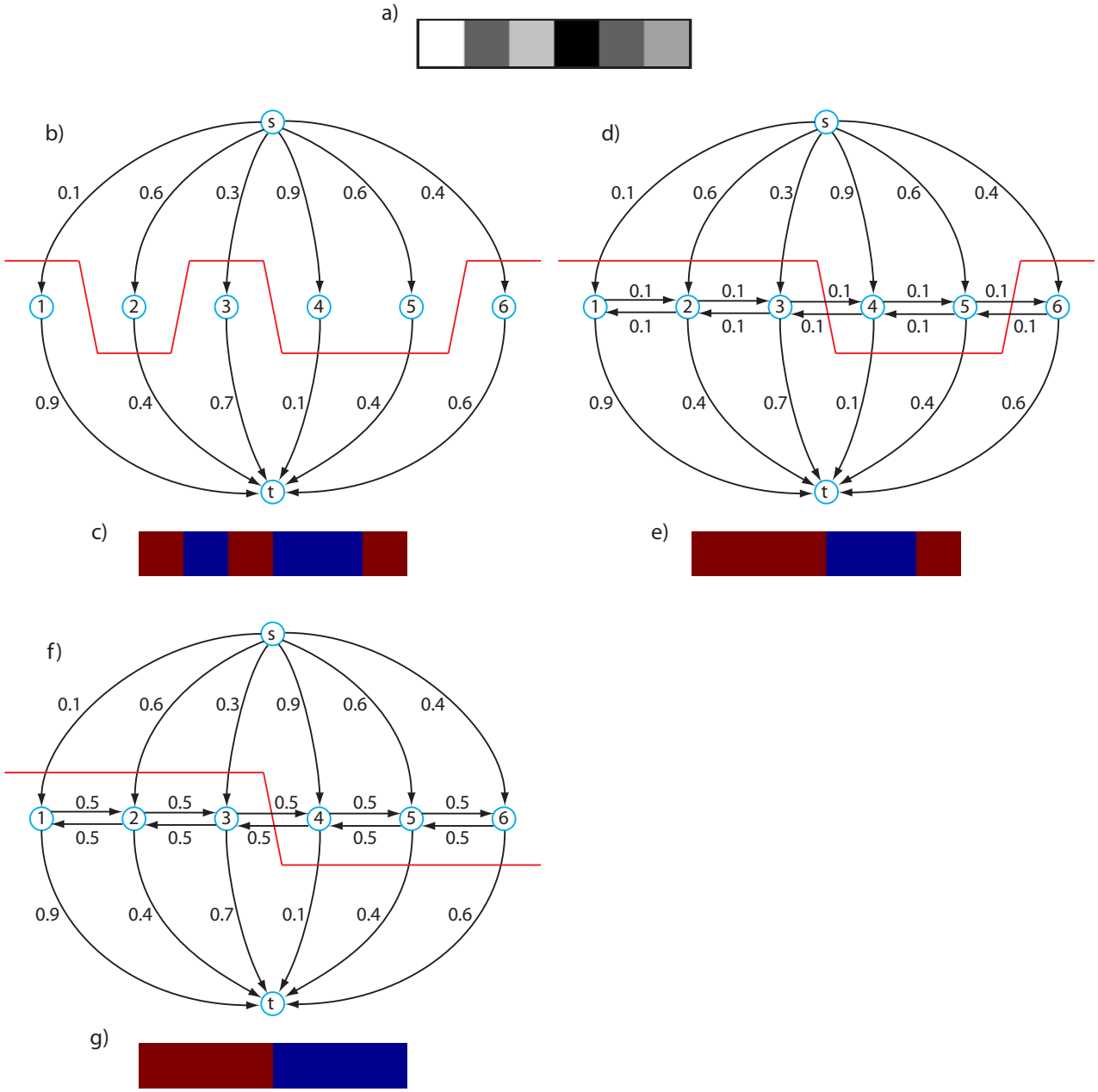


Figure 8.12: A 1D image, a), which is to be segmented by applying the min cut algorithm to a graph. If just the 1-cliques are used an appropriate MRF can be represented by the graph in b), and the resulting min cut gives rise to the segmentation in c). Here red denotes a 1 and a blue denotes a 0 label. In d) a smoothing 2-clique term is added to the graph, and thus the MRF, of b). The 2-clique term is represented by the edges between the pixel-vertices, with weight 0.1. The resulting minimum cut gives rise to the segmentation of e). Increasing the smoothing term from 0.1 to 0.5, gives rise to the graph in f) and the result in g). Note that the increased smoothing term actually gives a more homogenous solution.

a shift from lying above to lying below the pixel-vertices, or vice versa. In this case it will have to cross one of the edges with the cost of 0.1, thus adding this value to the cuts capacity. Thus the min cut will correspond to the minimum energy, $U(f)$, and thus a MAP-MRF solution.

The resulting labels from the example of Figure 8.12-d is seen in Figure 8.12-e. As a further illustration, in Figure 8.12-f the example from Figure 8.12-d is repeated but with the 0.1 in the 2-clique potential changed to 0.5. The result is seen in Figure 8.12-g.

8.4.1 Setting up the Graph

In the 1D examples presented immediately above, it should be noted that there is nothing fundamentally that limits the formulation to 1D, and that it could just as easily be extended to e.g. 2D or 3D. The clique potentials of binary MRF can, however, be more complicated than the ones presented above, and thus there is a need for a algorithm forming a binary MRF (in the form of a Gibbs random field) into a graph. Such an algorithm is proposed in [Kolmogorov and Zabih, 2004], and will be presented here for the 1-clique and 2-clique cases. Not all binary MRF are, however, representable as graphs, and the MAP solution to some are provably NP-hard, as described in the following.

In the general case the 2-clique potential can take on four distinct values, which in this section will be denoted K, L, M and N. Specifically, the general 2-clique potential in the case of binary labels is given by

$$V_{(i,j)}(f_i, f_j) = \begin{cases} K & \text{for} \quad f_i = 0, f_j = 0 \\ L & \text{for} \quad f_i = 0, f_j = 1 \\ M & \text{for} \quad f_i = 1, f_j = 0 \\ N & \text{for} \quad f_i = 1, f_j = 1 \end{cases} .$$

This can be arranged into a table as follows:

$V_{(i,j)}(f_i, f_j)$	$f_j = 0$	$f_j = 1$
$f_i = 0$	K	L
$f_i = 1$	M	N

Likewise, the two possible values of the 1-clique will be denoted A and B, i.e.

$$V_i(f_i) = \begin{cases} A & \text{for} \quad f_i = 0 \\ B & \text{for} \quad f_i = 1 \end{cases} ,$$

and

$V_i(f_i)$	
$f_i = 0$	A
$f_i = 1$	B

As mentioned, not all binary MRF are representable by graphs. An important theorem from [Kolmogorov and Zabih, 2 states that a MRF consisting only of 1-cliques and 2-cliques, is representable by a graph if and only if it is *regular function*[11], i.e. for all 2-cliques

$$K + N \leq M + L \ . \tag{8.17}$$

If the MRF is not regular, computing the MAP-MRF solution is NP-hard. The regularity condition (8.17), can basically be seen as a homogeneity constraint. It intuitively states that the cases of neighboring pixels having the same label should combined have a lower energy then toughs having different labels. Since we typically want to use MRF to formulate a homogeneity prior the regularity constraint, is often not an issue. Note that the term *submodular* is often used instead of regular, cf. [Boros and Hammer, 2002].

An Algorithm

Here the algorithm of [Kolmogorov and Zabih, 2004] for constructing a graph representing 1-clique and 2-clique MRF problems. Similarly to Figure 8.12, the graph should contain a vertex representing the source and another vertex representing the sink. In addition to this the graph should contain a site-vertex for each site, e.g. pixel. Then, also as in Figure 8.12, the one cliques are entered into the graph, as illustrated in Figure 8.13-Left, by for each $V_i(f_i)$

- Add an edge from the source, s, to site-vertex i with a weight of B.
- Add an edge from site-vertex i to the sink, t, with a weight of A.

To derive a method for the 2-cliques, the table (and thus the individual terms) for the 2-clique potential can be manipulated in the following way

[11] The concept of regular functions extends beyond MRF consisting only of 1-cliques and 2-cliques, but this is not covered here.

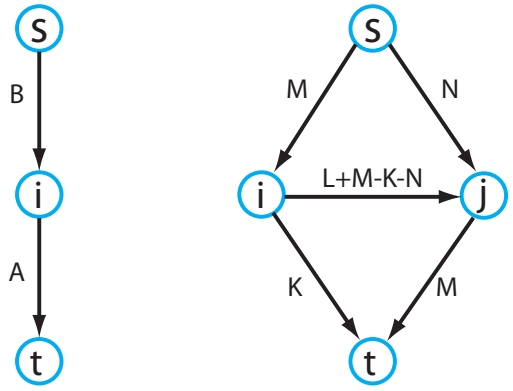


Figure 8.13: An illustration of how to implement graphs representing 1-cliques, **left**, and 2-cliques, **right**, cf. Table 8.4.

$$\begin{array}{|c|c|}\hline K & L \\ \hline M & N \\ \hline \end{array} = \begin{array}{|c|c|}\hline K & K \\ \hline M & M \\ \hline \end{array} + \begin{array}{|c|c|}\hline M & N \\ \hline M & N \\ \hline \end{array} + \begin{array}{|c|c|}\hline 0 & L+M-K-N \\ \hline 0 & 0 \\ \hline \end{array} - \begin{array}{|c|c|}\hline M & M \\ \hline M & M \\ \hline \end{array}$$

The terms on the right hand side are all readily implementable as edges, as seen in Figure 8.13-Right. Considering them one by one:

First Term is seen to be a function of site i's label only. This term can thus be implemented as the 1-clique potentials of site i.

Second Term is seen to be a function of site j's label only, and can be implemented as the 1-clique potentials of site j.

Third Term is seen to be zero, except when $f_i = 0$ and $f_j = 1$. This labeling happens only when site-vertex i is in the $\mathcal{S}$ partition and site-vertex j is in the $\mathcal{T}$ partition. Since the edges are directed, edges from site-vertex i to site-vertex j are part of the cut if and only if $f_i = 0$ and $f_j = 1$. Thus the third term is implemented as an edge from site-vertex i to site-vertex j with the weight $L + M - K - N$.

Fourth term is a constant, so it will be ignored, since it has no practical implications.

In constructing a graph via this algorithm, several edges are almost certain to be added between the same two vertices. This could e.g. happen if a site is both part of a 1-clique and a 2-clique. Since the flow is additive, such two edges can be collapsed into one with an edge weight of the combined weight, as illustrated in Figure 8.14. This has the advantage, that the graph will have fewer edges, and thus the minimum cut will be faster to compute. The graph construction algorithm is outlined in Table 8.4.

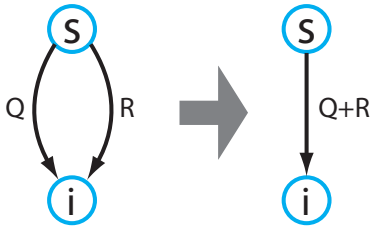


Figure 8.14: Since the max flow algorithm works with how much flow can be pushed between node, it is seen that the two subgraphs are equivalent in relation to this algorithm. Since the right graph is sparser, in terms of representation in software, compared to the left graph, the right graph is to be preferred due to computational efficiency.

When formulating the MRF as graphs, it should be noted, that edge weights are to be positive, or zero (corresponding to the edge being absent). This is part of the assumptions behind the max flow min cut problem formulation. In this regard it is noted that the third term above, i.e. $L + M - K - N$ is always non-negative due to the requirement that the 2-clique potentials are regular. Also if the edge weights are not non-negative by design of the MRF, this can often times be mended by adding a constant to the clique potentials.

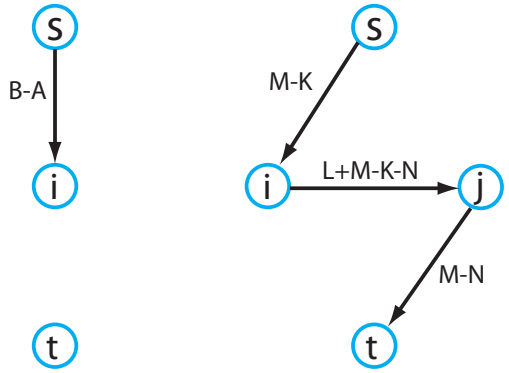


Figure 8.15: This is a sparse version of Figure 8.13, for $A \leq B$, $K \leq M$ and $N \leq M$, which has fewer edges and as such more computationally efficient. If these conditions do not hold other reductions are available, cf. [Kolmogorov and Zabih, 2004].

For a Gibbs random field, consisting only of one and two cliques, with n sites form a graph with $n + 2$ vertices, a source, t and a sink, t, plus n pixel-vertices enumerated from 1 to n. Denote the individual clique potentials by:

$V_i(f_i)$	
$f_i = 0$	A
$f_i = 1$	B

and

$V_{(i,j)}(f_i, f_j)$	$f_j = 0$	$f_j = 1$
$f_i = 0$	K	L
$f_i = 1$	M	N

For each clique potential add edges to the graph as illustrated in Figure 8.13 — alternatively Figure 8.15.
Given the computed solution of the min cut on the constructed graph, the sites with corresponding pixel-vertices in the same partition as the source, $\mathcal{S}$, are assigned a label of 0. The sites with a corresponding site-vertex in the same partition as the sink, $\mathcal{T}$ are assigned a label of 1.

Table 8.4: Pseudo code for construction a graph from a MRF as proposed by [Kolmogorov and Zabih, 2004].

A more subtle way to increase the computational efficiency, by reducing the number of edges, is illustrated in Figure 8.15, for the cases of $A \leq B$, $K \leq M$ and $N \leq M$. The underlying idea is; that since we can add a constant to the clique potentials with out changing the underlying MRF, we can add (negative) constants to the potentials such that some of the edge weights become zero. If edge weights become zero we can remove the edge. This must, however, happen under the condition that no edge weight must become negative, which is the reason for the conditions $A \leq B$, $K \leq M$ and $N \leq M$. If these specific conditions do not hold other similar modifications are possible, c.f. [Kolmogorov and Zabih, 2004]. Note that, many implementations of the graph cut algorithm from [Boykov and Kolmogorov, 2004], performs this optimization automatically.

8.4.2 Two Examples

As a first example consider the simple example from Figure 8.12-d is formulated in accordance with [Kolmogorov and Zab and Table 8.4 in Figure 8.16. As a more realistic example, Figure 8.17 contains the solution of the brain segmentation problem from Figure 8.8, but where only the two classes of Fluid and Gray Matter are used, the $\beta = 8$.

8.5 Fusion Moves – Extending Graph Cuts to More Than Two Labels

The graph cut algorithm presented in Section 8.4 is extremely powerful, in that it is guaranteed to find the global optimum of a certain class of regular MRF in an efficient manner. This is to be seen in contrast to many other

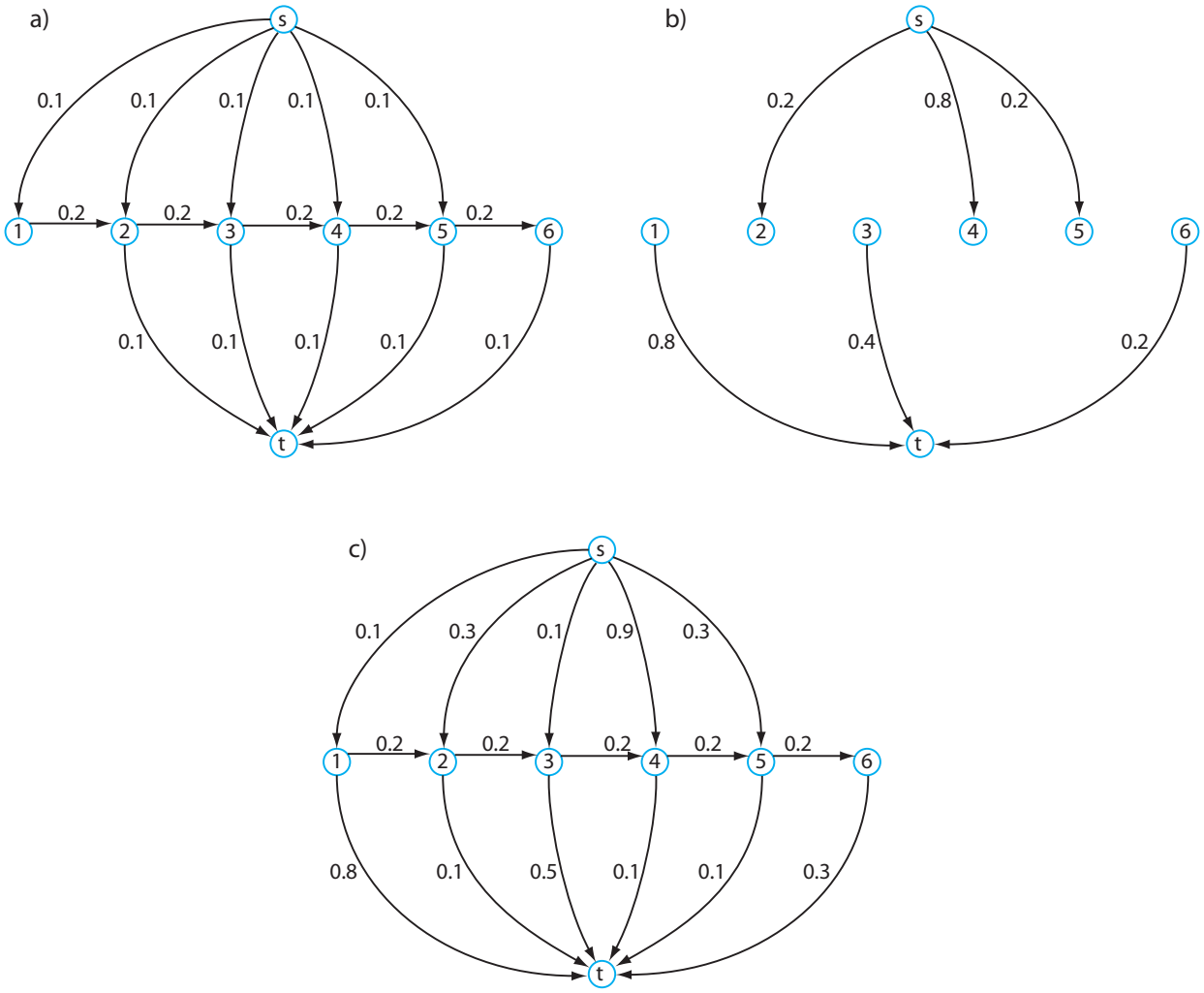


Figure 8.16: The example from Figure 8.12-d formulated in accordance with [Kolmogorov and Zabih, 2004] and Table 8.4. The graph in a) represents the one cliques and the graph in b) the two cliques. The sum of these two graphs is shown in c) and represents the same MRF as that of Figure 8.12-d.

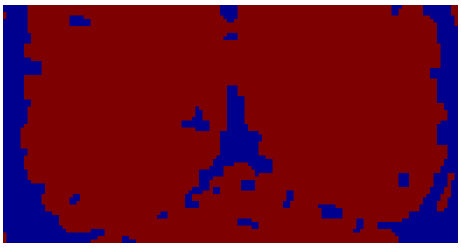


Figure 8.17: The solution to the brain segmentation problem from Figure 8.8, with only two labels Fluid (blue) and Gray Matter (Red).

MAP-MRF optimization methods, cf. Section 8.6. A serious limitation of this graph cut framework is that it is only able to deal with binary labels. In this section ways of addressing this limitation are described, although this happens at the expense of obtaining a guaranteed global optimal MAP-MRF solution. As the section tile indicates, this is basically done via the so-called fusion moves.

These fusion moves were introduced in [Boykov et al., 2001] and the theory was further refined, in e.g. [Kolmogorov and Zabih, 2004] and [Lempitsky et al., 2010]. The central idea behind the fusion moves, as illustrated in Figure 8.18, is that two set of labelings $g = \cup g_i$ and $h = \cup h_i$ can be fused into the labels

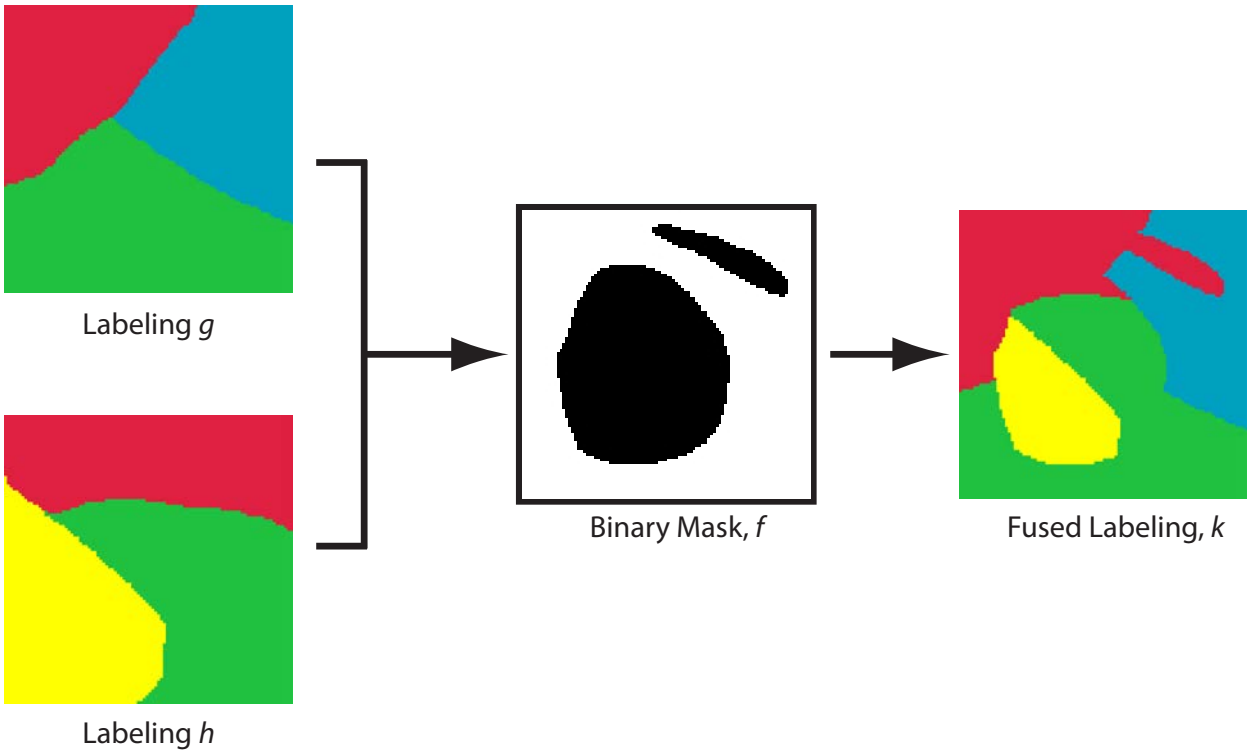


Figure 8.18: Illustration that two labelings, g and h can be combined into one, k, via a binary mask f. Given g and h, labeling k can be seen as a function of f.

$k = \cup k_i$, via a binary mask $f = \cup f_i$ as follows:

$$k = g \bullet f + h \bullet (1 - f) \ , \tag{8.18}$$

where $f_i \in \{0, 1\}$, and $\bullet$ denotes the Hadamard, i.e. element-wise, product. That is, (8.18) can be written as

$$\forall i, \quad k_i = g_i \cdot f_i + h_i \cdot (1 - f_i) \ .$$

This operation, (8.18), is termed the *fusion* of g and h, [Lempitsky et al., 2010]. Note that, given the two labelings, g and h, the fused labeling, k, is a function of the *binary* labeling f.

Consider a MRF with a set of possible labels $\mathcal{L}$, and two labelings g and h, such that

$$g_i \in \mathcal{L} \quad \text{and} \quad h_i \in \mathcal{L} \ .$$

Denote the energy function associated with the MRF, cf. (8.9), by

$$U(k) \ .$$

Let k be a fusion of g and h as a function of f, according to (8.18). Then we can formulate a binary MRF, with associated energy function $\tilde{U}(f)$ as follows

$$\tilde{U}(f) = U\left(g \bullet f + h \bullet (1 - f)\right) \ .$$

Given that the *binary* MRF, f, fulfills the necessary criteria, e.g. it is regular, a MAP solution can be computed for this binary MRF, e.g. via the graph cut algorithm in Table 8.4. Doing this is termed a *fusion move*. Note that, most properties of the binary MRF are inherited from the original MRF.

Assuming that we have a suitable method of proposing new labelings, the use of fusion moves give rise to the following algorithm, for optimizing a MRF[12], with the resulting labeling k and energy function $U(k)$

1. Initialize k, e.g. just using the 1-cliques as in Figure 8.12-b.

[12] Assuming a suitable MRF, e.g. that it is regular regular.

2. Propose a new labeling g, see e.g. Section 8.5.1.

3. Set k to the result of the fusion move of k and g, i.e.

   (a) Form the binary MRF
   $$\tilde{U}(f) = U\left(k \bullet f + g \bullet (1 - f)\right)$$

   (b) Find the MAP solution f^* to $\tilde{U}(f)$.

   (c) Update k with the result, i.e.
   $$k = k \bullet f^* + g \bullet (1 - f^*)$$

4. Goto 2, until convergence.

There are many proposed algorithms that fit into this generalized frame work, and there are practical methods to work around some of the instances where the binary MRF, f, is only almost regular. This is beyond the scope of this text, and the interested reader is referred to [Lempitsky et al., 2010].

8.5.1 Alpha Expansion

Probably the most popular fusion move algorithm is the alpha expansion algorithm [Boykov et al., 2001]. Here we consider a MRF for which

- The possible set of labels, $\mathcal{L} = \{\mathcal{L}_1, \mathcal{L}_2, \ldots \mathcal{L}_n\}$, is finite. Preferably, n, is not to large due to performance considerations.
- The associated energy function only contains 1-cliques and 2-cliques. This can be extended, e.g. to 3-cliques, but this is beyond the scope of this text.
- The 1-cliques are non-negative and the 2-cliques are metric in the labels, i.e.
  $$\begin{aligned} V_{(i,j)}(k_i, k_j) = 0 \quad &\Leftrightarrow \quad k_i = k_j \\ V_{(i,j)}(k_i, k_j) \leq V_{(i,j)}(k_i, k_l) &+ V_{(i,j)}(k_l, k_j) \ , \end{aligned} \tag{8.19}$$
  implying that the binary MRF derived from it will be regular. The second term is the triangle inequality.

The idea behind the alpha expansion (or α-expansion) is that we expand on label α iteratively. By this is meant, that we for all possible labels, $\mathcal{L}_r, r \in \{1, 2, \ldots, n\}$, iteratively try to fuse k with a g consisting only of label $\mathcal{L}_q$, i.e.
$$\forall i, \quad g_i = \mathcal{L}_q \ .$$
In a sense this is trying to expand the label $\mathcal{L}_q$ (or α) to the temporary solution k.

A technical issue in this regard is how to derive $\tilde{U}(f)$ from $U(k)$. To do this, note, that label f_i denotes if label k_i should be equal to k_i or to $g_i = \mathcal{L}_i$, i.e.
$$k_i = \begin{cases} g_i (= \mathcal{L}_q) & \text{for} \quad f_i = 0 \\ k_i & \text{for} \quad f_i = 1 \end{cases} \ .$$

Thus, for each 1-clique, $V_i(k_i)$, of $U(k)$ the binary MRF has a 1-clique, $\tilde{V}_i(f_i)$, given by
$$\tilde{V}_i(f_i) = \begin{cases} V_i(\mathcal{L}_q) & \text{for} \quad f_i = 0 \\ V_i(k_i) & \text{for} \quad f_i = 1 \end{cases} \ . \tag{8.20}$$

Similarly, for every 2-clique, $V_{(i,j)}(k_i, k_j)$, of $U(k)$ the binary MRF has a 2-clique, $\tilde{V}_{(i,j)}(k_i, k_j)$,given by
$$\tilde{V}_{(i,j)}(f_i, f_j) = \begin{cases} V_{(i,j)}(\mathcal{L}_q, \mathcal{L}_q) & \text{for} \quad f_i = 0, f_j = 0 \\ V_{(i,j)}(\mathcal{L}_q, k_j) & \text{for} \quad f_i = 0, f_j = 1 \\ V_{(i,j)}(k_i, \mathcal{L}_q) & \text{for} \quad f_i = 1, f_j = 0 \\ V_{(i,j)}(k_i, k_j) & \text{for} \quad f_i = 1, f_j = 1 \end{cases} \ . \tag{8.21}$$

Given a MRF with possible labels $\mathcal{L} = \{\mathcal{L}_1, \mathcal{L}_2, \ldots \mathcal{L}_n\}$, and an energy function $U(k)$ consisting only of non-negative 1-cliques and metric 2-cliques, a local optimal labeling k^* can be found by

1. Initialize k, e.g. just using the 1-cliques as in Figure 8.12-b.
2. For $q = 1$ to n do
   (a) For all i set $g_{=}\mathcal{L}_q$.
   (b) From the binary MRF and find the MAP solution f^*, as described in Table 8.4 with an energy function given by (8.20) and (8.21).
   (c) Update k with the result, i.e.
   $$k = k \bullet f^* + g \bullet (1 - f^*)$$
3. Goto 2, until convergence.
4. The local optimum $k^* = k$.

Table 8.5: Outline of the alpha expansion algorithm for MRF optimization.

It is noted that if $V_{(i,j)}(k_i, k_j)$ is metric, cf. (8.19), then the 2-clique, $\tilde{V}_{(i,j)}(f_i, f_j)$, of (8.21) is regular, in that from (8.17)

$$\begin{aligned} & K + N \leq M + L \qquad \Leftrightarrow \\ & V_{(i,j)}(\mathcal{L}_q, \mathcal{L}_q) + V_{(i,j)}(k_i, k_j) \leq V_{(i,j)}(k_i, \mathcal{L}_q) + V_{(i,j)}(\mathcal{L}_q, k_j) \qquad \Leftrightarrow \\ & V_{(i,j)}(k_i, k_j) \leq V_{(i,j)}(k_i, \mathcal{L}_q) + V_{(i,j)}(\mathcal{L}_q, k_j) \ , \end{aligned}$$

which is the triangle in equality and thus holds. Here it is used that according to (8.19)

$$V_{(i,j)}(\mathcal{L}_q, \mathcal{L}_q) = 0 \ .$$

In the case covered here, with only 1-cliques and 2-cliques, (8.20) and (8.21) define the binary MRF[13]. An outline for the alpha expansion algorithm is given is Table 8.5, which is a specific case of the fusion move algorithm above. A suitable stopping criteria in the third step of Table 8.5 is that no changes happened to k during the for loop of step two. This is equivalent to all $f_i^* = 1$ for all $q = 1$ to n.

As an example of the alpha expansion algorithm consider the example from Figure 8.8, where the solution in Figure 8.9 is computed via this algorithm. The individual fusion moves are illustrated in Figure 8.19.

Optimality

As mentioned in the start of this section, using fusion moves to handle MAP-MRF problems with more than two labels comes at the cost of the guarantee that the computed optimum is the global optimum

$$\min_k U(k) \ .$$

This is also true for the alpha expansion algorithm, however, it has the relatively strong property, [Boykov et al., 2001], that for the computed solution k^*

$$\min_k U(k) \leq 2cU(k^*) \ , \tag{8.22}$$

[13] This formulation is based on [Kolmogorov and Zabih, 2004], which is slightly different than the original formulation of the alpha expansion algorithm [Boykov et al., 2001].

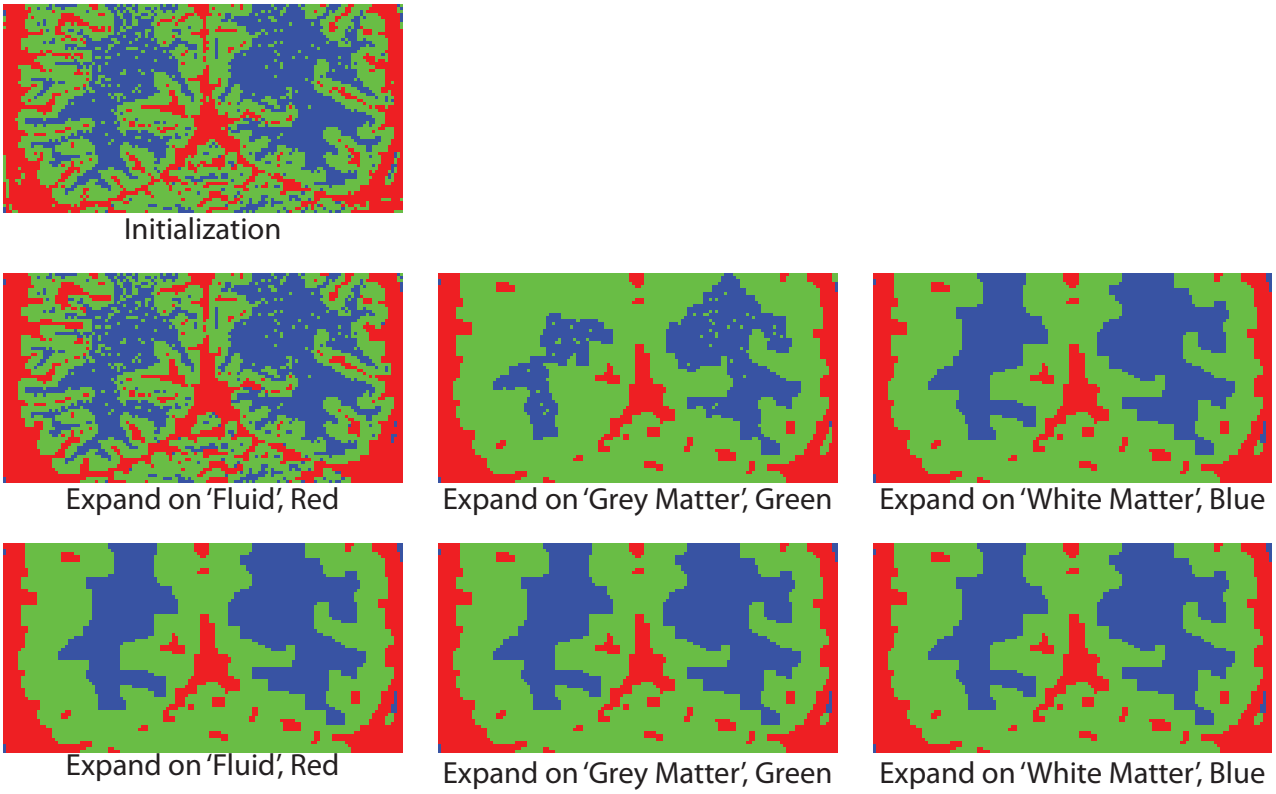


Figure 8.19: The individual expansion moves of the alpha expansion algorithm leading to the solution in Figure 8.9 for $\beta = 8$. It should be noted how fast the algorithm converges.

where c is a constant defined by the highest ratio of inhomogeneous 2-cliques. See [Boykov et al., 2001] for further details and the proof. However, in the often occurring simple case that that all inhomogeneous 2-cliques are equal, i.e.

$$\forall (i,j) \in \mathcal{C}_2 \quad f_i \neq f_j \Rightarrow V_{(i,j)}(f_i, f_j) = \Gamma > 0 \ ,$$

where Γ is a constant, then

$$c = 1 \ ,$$

and

$$\min_k U(k) \leq 2U(k^*) \ .$$

Thus, in this simple case the alpha expansion is guaranteed to perform at worse twice as bad as the global optimum, although it typically performs much better.

8.6 A Note on Other MAP-MRF Optimization Methods

Markov random fields and their MAP solution have received allot of attention in the literature, cf. [Li, 2009], and as such there is an abundance of numerical methods for computing the MAP-MRF solution, apart from the graph cut algorithms presented above. As an example, Gaussian image filtering can be formulated and a MAP-MRF problem, and the MAP-MRF solution of this problem can thus be found via linear filtering with a Gaussian kernel. Also, some MAP-MRF optimization problems are convex, e.g. [Aanaes et al., 2008], (or can be regarded as such) in which cases algorithms like gradient decent or Levenberg-Marquardt can be used [Nocedal and Wright, 2000].

In the immediately above mentioned examples, special structure is used to make efficient MAP-MRF solvers, this is, however, not always possible. To address this, much work has been done on extending the graph cut methodology, to deal with an ever increasing class of problems, cf. [Lempitsky et al., 2010]. Another class of solvers, which are applicable to all MAP-MRF problems, and predominantly used before the advent of the graph cut algorithms, are the stochastic sampling algorithms, cf. [Li, 2009, Winkler, 2003].

These stochastic sampling algorithms, work by the general scheme of

1. Initialize the labeling, k.
2. Randomly permute k into k'. This is typically done by sampling from a probability distribution, e.g. the normal distribution.
3. Accept the permutation, i.e. set $k = k'$, with a probability dependent on how $p(\mathcal{F} = k)$ compares to $p(\mathcal{F} = k')$, cf. (8.9).
4. Goto 2., until convergence or you run out of time. (These algorithms are often very slow)

Note that acceptance in step three can still occur if $p(\mathcal{F} = k) > p(\mathcal{F} = k')$, i.e. the permutation is worse. The likelihood of a worse permutation being chosen is dependent on temperature T. In general there is a strong link between the stochastic sampling algorithms and stochastic mechanics, as well as evolutionary biology theory.

In general stochastic sampling techniques are slow, and they have been demonstrated poor performance in some cases, [Greig et al., 1989]. They, however, do supply solutions to *all* MAP-MRF problems, and as such one should be aware of their existence when working with MRF in computer vision.

Part V

Optical 3D Estimation

Chapter 9

Active 3D Sensors

The estimation of objects 3D geometry is an important and largely used part of computer vision. In this chapter the active methods for doing this are covered. Here ”active”, as opposed to passive, implies that the methods and associated hardware actively emit light, making the 3D estimation problem easier to solve. This in turn makes the active methods more robust, and are as such the most widely used methods in industry.

9.1 Time of Flight Cameras

The fist 3D estimation method or technology considered is that of *time of flight*, cf. e.g. [Gudmundsson, 2010]. Time of flight, as the name implies, works by emitting light and measuring the time it takes for this light to travel the distance to an object and back again. Given that the speed of light is known, this translates into distances, and corresponds to radar with light instead of radio waves, cf. Figure 9.1. This technology is also referred to as *Lidar*. The typical way in which this time measurement is done is by considering the phases differences of the light emitted and received. Thus, aliasing can occur if the objects are further than a light wave away.

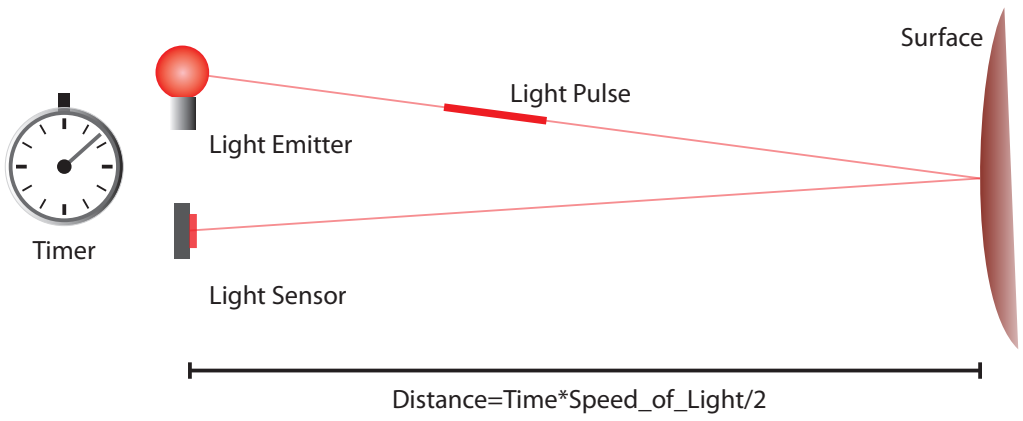


Figure 9.1: Illustration of the principle behind time of flight. A light source emits a light pulse, which hits a surface and is reflected back and hist a light sensor. Measuring the time it takes for the light pulse gives the distance from the time of flight camera (comprised of emitter and sensor) to the sensor, in that the distance traveled is equal to the time multiplied by the time of light. Since the pulse travels to the surface and back the distance traveled by the light has to be divided by two.

The light emitted is usually infrared, and the device used to measure the arrival of the light is similar to a normal camera chip. Thus a regular array of measurements are made, giving an image of 3D measurements. Please refer to Figure 9.2 for a depiction of a time of flight camera and output samples. Due to the arrival sensors similarity to an infrared camera, time of flight cameras typically also output infra red images. In particularly 3D estimate frames and infrared 2D frames are typically interlaced, such that e.g. the odd frames are 3D estimates and the even frames are 2D infrared images. Thus the output from a time of flight camera can be seen as textured 3D models, cf. Figure 9.2.

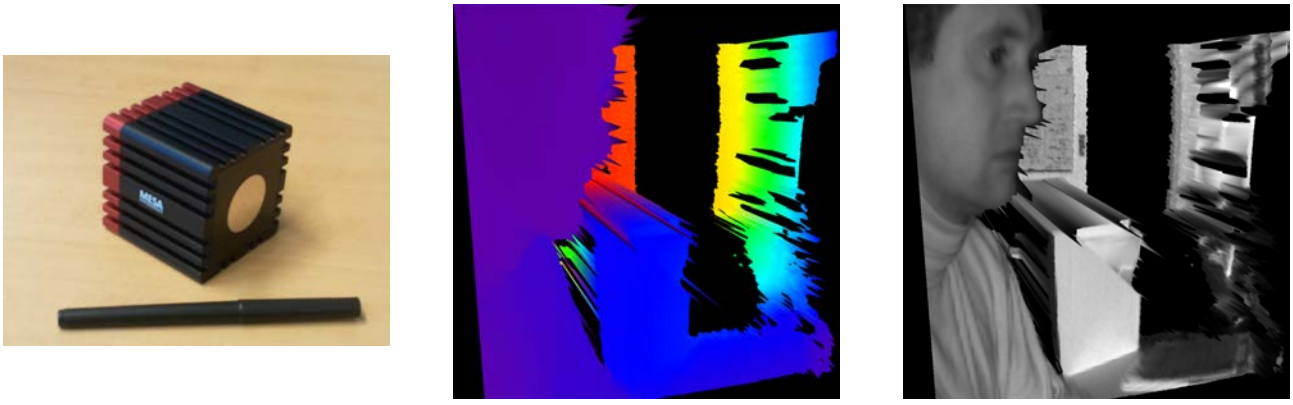


Figure 9.2: **Left:** The SR4000 time of flight camera from Mesa Imaging. **Middle:** A depth image from the SR4000 time of flight camera. It is a projected 3D model where the depth is indicated by the color. **Right:** The 3D model from the middle with the 3D infra red image is projected onto it as texture.

The advantage of time of flight cameras as a 3D capturing device, e.g. compared to passive methods and the other active methods presented here, is firstly it ease of use in that it is just a point and shoot device completely packed into a manageable casing. Also the frame rate is high compared to many of the other active methods. However, as seen from Figure 9.2, the 3D accuracy is decent but not great, e.g. compared to a laser scanner as presented in Section 9.2.

One of the main uses of time of flight cameras is for tracking purposes, because the depth measurement makes it trivial to distinguish between foreground and background objects. Examples include, human tracking used human computer interfaces such as smart rooms,[Gudmundsson, 2010], and for game controls, people detection aimed at making cars that can alert the driver if a care is about to hit a person.

9.2 Laser Scanners – part II

Another active sensor is the *laser scanner* [Levoy et al., 2000, Rusinkiewicz and Levoy, 2001] introduced in it's basic form in Section 2.6. In that section the laser scanner consists of a camera and a laser, both with known camera/optical geometry. Based on the camera observations (images) of the laser reflections of the object surface, the 3D coordinates of the reflecting points on object surface can be found via triangulation, as seen in Figure 2.17.

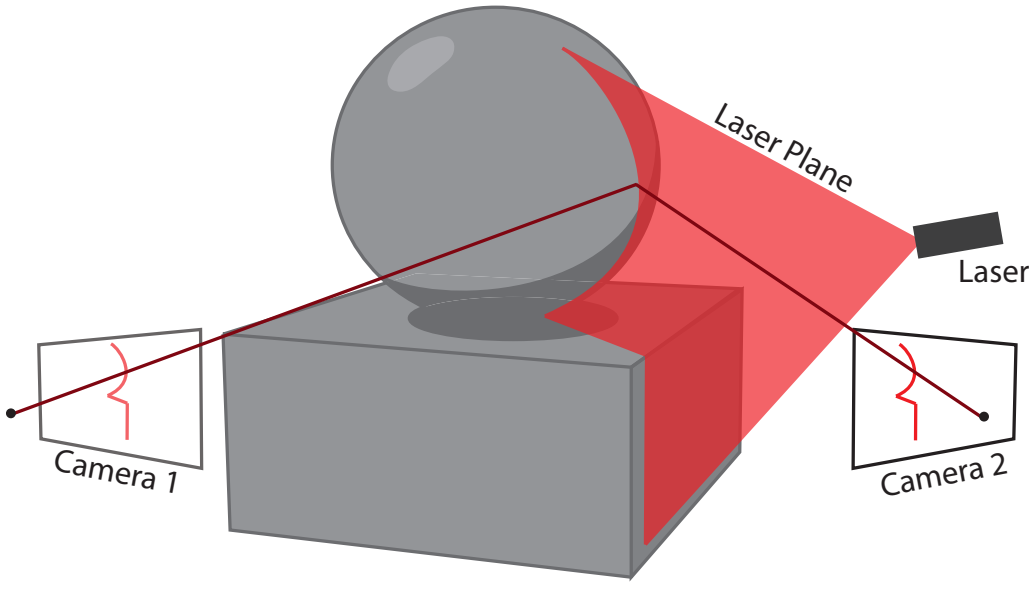


Figure 9.3: An extension of Figure 2.17, where another camera is added. Thus, the 3D reconstruction can be completed by triangulation between the two cameras, where the laser is used to establish the correspondence.

A variant of the setup from Section 2.6, shown in Figure 9.3, is that there are two instead of one cameras viewing the laser reflect of the object surface. Here the laser is used to establish point correspondences between the images of the two cameras, correspondences, which are then used to do 3D point triangulation, cf. Section 2.4. The cameras are assumed calibrated and their geometry known. The geometry of the laser plane is thus generally not used in the equations for 3D point estimation, although it arguably should be in order to reduce noise. An image of a laser plane reflecting of an object is seen in Figure 9.5-Right.

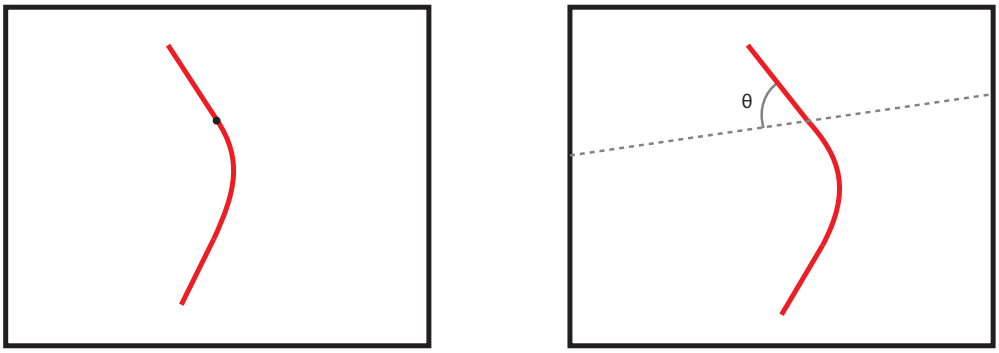


Figure 9.4: A point on the depicted laser curve in one image induces an epipolar line in the other. The intersection of this line and the laser curve gives the correspondence. It is noted that the angle, θ, between the epipolar line and the laser curve determines the accuracy by which this intersection can be found. Here a θ close to 90° is optimal.

An apparent problem with establishing the point correspondences between a pair of images viewing a reflected laser plane is, that the reflected laser plane form curves in the images. Thus a way of determining which points on the curve in one image correspond to points a point in the other is needed. The known calibration of the cameras provide such a method via the epipolar geometry, cf. Section 2.2, which can be computed from the calibrated cameras, cf. equation (2.10). As illustrated in Figure 9.4, a point in one image induces an epipolar line in the other. The intersection of this epipolar line and the curve from the reflected laser gives the point correspondence. The accuracy by which this intersection can be found – which directly influences the accuracy of the estimated 3D point – is a function of the angle θ between the epipolar line and the laser curve, in that the close θ is to a right angle the better. If, on the other hand, θ approaches zero, the epipolar line and the laser curve become co-linear and an intersection is not defined. Thus when designing the laser scanner hardware, it is important that the laser plane is close to perpendicular to the epipolar planes, i.e. that the line between the two camera centers is close to the normal of the laser plane.

Typically, we are not only interested in just the 3D estimate of the points located on a line/plane of the 3D object, and would like to a 3D estimate of most/all of the object. This is dealt with via a mechanical system, that either rotates/moves the object relative to the cameras and laser, as seen in Figure 9.5-Left, or moves the scanner (cameras and laser) such that it can cover a wide area. This mechanics can often be quite costly, if high precision is needed, and gives a relatively slow 3D estimation process. Laser scanners are, however, one of the most precise and robust optical 3D estimation technologies. An example of a laser scanned object is seen in Figure 9.5-Middle.

9.3 Image Rectification and Epipolar Geometry

As indicated above and in Chapter 6, epipolar geometry plays a central role when finding correspondences between image pairs. Often, the inclusion of this part of camera geometry into a correspondence algorithm implies searching along an epipolar line, which can be somewhat computationally cumbersome. One way to simplify this is if the two cameras producing the image pairs are fronto parallel, as illustrated in Figure 9.6. Here the cameras are placed side by side facing the same way. In this way the epipolar lines correspond to scan lines in the cameras. Thus the search along epipolar lines becomes equivalent to searching along scan lines, i.e. rows in the image data. This is computationally very simple and thus attractive.

Figure 9.5: **Left:** A scanner from Cyberware, the image is downloaded from http://www.cyberware.com/products/scanners/customScanners.html. Note the turn table installed such that the object can be rotated wrt. the laser. **Middle:** A sample laser scanned object in the form of the Stanford bunny. Image downloaded form http://graphics.stanford.edu/data/3Dscanrep/. **Right:** An example image of a laser plane reflecting of a 3D surface. Typically processing will be done so that the laser curve is the only significant light in the image, e.g. by doing the laser scanning in a dark box, and by putting narrow band optical filters in front of the cameras such that only light with the same color as the laser will come through.

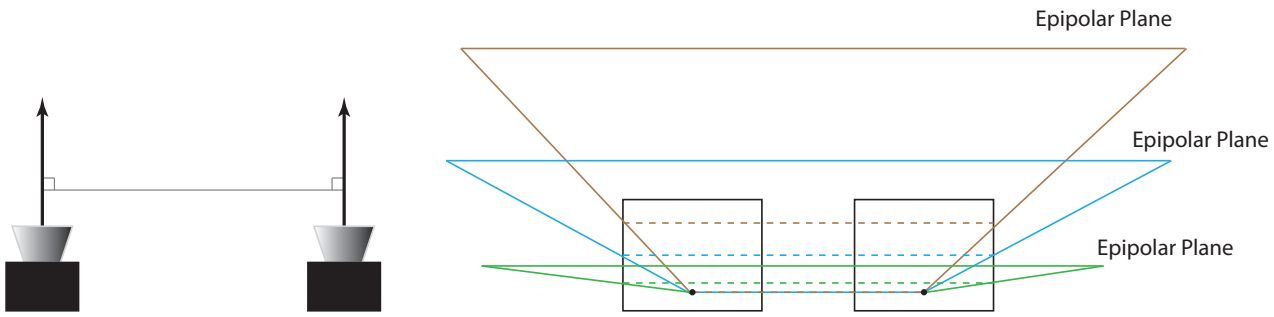


Figure 9.6: An illustration of two fronto parallel cameras, where the cameras are placed side by side facing the same way **left**. In this way the epipolar lines correspond to scan lines in the cameras **Right**. Note that the epipolar lines are the intersections of the epipolar planes with the image planes.

Due to the symmetry in the above argument, the scan lines can be paired two and two, such that all the pixels in a scan line all correspond to pixels in a specific scan line in the other image. Note that not all pixels have correspondences e.g. due to occlusions of the scene viewed. The pairwise correspondence of scan lines wrt. the epipolar geometry is particularly important for e.g. structured light algorithms, as presented in the next section. The reason being that finding correspondences of all possible pixels in an image can be subdivided into only finding correspondences between scan lines. I.e. a divide and conquer strategy can be used.

In practise, however, cameras are seldom fronto parallel. Even if they are physically set up to be – which is often a good idea for many 3D estimation purposes – this will seldom be achieved to the precision desired. Given the cameras orientation parameters, or even just the fundamental matrix, it is, however, possible to emulate a set of fronto parallel cameras by applying a homography to each of the images of the image pair, cf. Chapter 2. This is known as *image rectification*, and here an algorithm for achieving this is presented, given that the camera calibrations are known. An image rectification algorithm based on the fundamental matrix is given in [Hartley and Zisserman, 2003], and an alternative one is given in [Pollefeys et al., 1999], which is able to handle that the epipoles are located in the images themselves.

The essence of the the image rectification method presented here is to emulate alternative image planes for each of the two cameras, as seen in Figure 9.7. This is done by finding the mapping from the real image planes to the emulated ones. As shown in Section 2.3, this transformation can be described by a homography. As also seen in that section, this homography can be found if we have a basis and an origin of the plane to map to (the gray plane in Figure 9.7), and the camera calibrations[1]. Here we will denote this basis by A, B, as illustrated

[1] A pinhole camera is assumed.

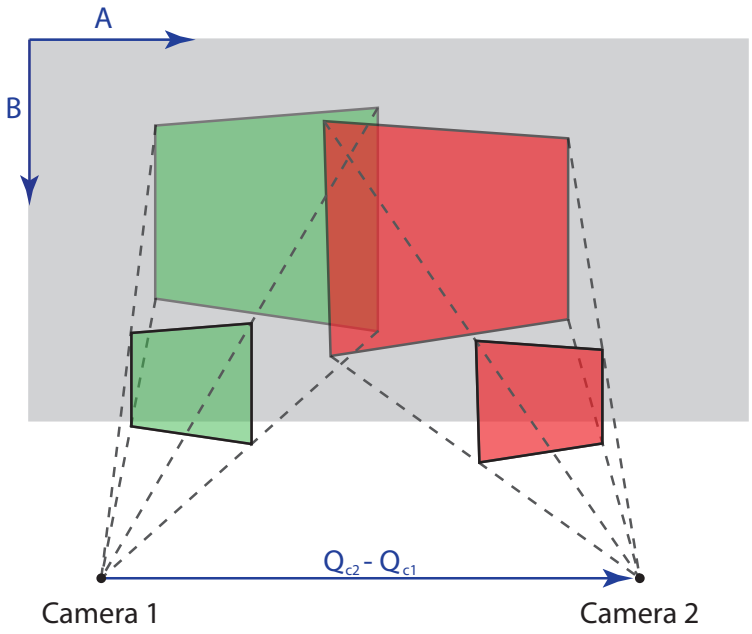


Figure 9.7: An illustration of the image rectification process. Given two images, here colored green and red, find a plane such that when the images are projected onto it the epipolar lines become parallel and along the scan lines.

in Figure 9.7, and the origin is denoted by C. In the following the algorithms for determining these A, B, C and a scale will be derived, thus defining the rectifying homography, given the camera matrices $\mathbf{P_1}$ and $\mathbf{P_2}$. Specifically

$$\mathbf{P_1} = \mathbf{A_1}[\mathbf{R_1}\ \mathbf{t_1}] \quad , \quad \mathbf{P_2} = \mathbf{A_2}[\mathbf{R_2}\ \mathbf{t_2}] \ .$$

Recall also, that any point (a, b) in the plane defined by A, B, C is equivalent to the 3D coordinate

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

Firstly a method for computing A will be derived. It is noted that the epipolar planes contain the centers of the two cameras. These camera centers, Q_{c1} and Q_{c2}, are according to Section 1.4.3, given by

$$Q_{c1} = -\mathbf{R_1^T t_1} \quad , \quad Q_{c2} = -\mathbf{R_2^T t_2} \ .$$

The vector A given by

$$A = \frac{Q_{c2} - Q_{c1}}{||Q_{c2} - Q_{c1}||_2} \ , \tag{9.1}$$

is thus a possible basis vector of any epipolar plane. It can be shown that, for any 3D point Q the point $Q + \alpha \cdot A$ will be on the same epipolar plane, for any scalar α. Specifically, denote the homogeneous coordinate of the epipolar plane defined by Q_{c1}, Q_{c2} and Q by L_e. Then the three points must be on this plane, i.e.

$$L_e^T Q_{c1} = 0 \quad , \quad L_e^T Q_{c2} = 0 \quad , \quad L_e^T Q = 0 \ .$$

Thus by direct insertion

$$L_e^T A = \frac{L_e^T (Q_{c2} - Q_{c1})}{||Q_{c2} - Q_{c1}||_2} = \frac{L_e^T Q_{c2} - L_e^T Q_{c1}}{||Q_{c2} - Q_{c1}||_2} = \frac{0 - 0}{||Q_{c2} - Q_{c1}||_2} = 0 \ .$$

And

$$L_e^T (Q + \alpha \cdot A) = L_e^T Q + \alpha \cdot L_e^T A = 0 \ .$$

So $Q + \alpha \cdot A$ will be on the the epipolar plane define by L_e. Therefor the A from (9.1) is set equal to the first basis vector A of our plane. To see this, note that any two points (a_1, b) and (a_2, b) – in the grey plane

of Figure 9.7 – with the same second coordinate b, will be on the same epipolar plane. Since epipolar lines are intersections of epipolar planes and the image plane, epipolar lines of images on the A, B, C plane have epipolar lines given by constant second coordinate b. Thus images of the A, B, C plane have epipolar lines corresponding to scan lines.

When the the two images are projected or warped onto plane A, B, C via a homography, whereby the warped images exist in the coordinate system of the plane. Thus, due to the construction of the first basis vector of the plane, A, epipolar lines are consistent with scan lines thus achieving the goal. As such the rest of the algorithm consists of finding good choices for B and C. The design criteria used here is that the images should be distorted as little as possible, such that the quality degradation of the rectified or warped images become as small as possible due to sampling issues.

The choice of the second plane basis B is based on the goal of the plane A, B, C should be as parallel to the original image planes as possible. From the derivation of the pinhole camera model in Section 1.4, it is seen that the normal to the original planes is equal to the 3^{rd} row of the respective rotation matrices, equaling the direction of the cameras optical axes'. The normal to the plane A, B, C is thus set as close to the average of these original image plane normals as possible[2], since parallel planes have equal normals. The average normal, N, is given by[3]

$$N = \frac{1}{2}\left(\mathbf{R_1}^T \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} + \mathbf{R_2}^T \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \right) \quad . \tag{9.2}$$

The basis vector B is thus chosen as perpendicular to N and A computed as the cross product between the two[4], following this it is normalized, i.e.

$$\begin{aligned} B' &= N \times A = \frac{1}{2}\left(\mathbf{R_1}^T \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} + \mathbf{R_2}^T \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \right) \times A \\ B &= \frac{B'}{||B'||} \quad . \end{aligned} \tag{9.3}$$

Note, the N will typically *not* become the normal to the plane A, B, C in that N and A are not perpendicular in general.

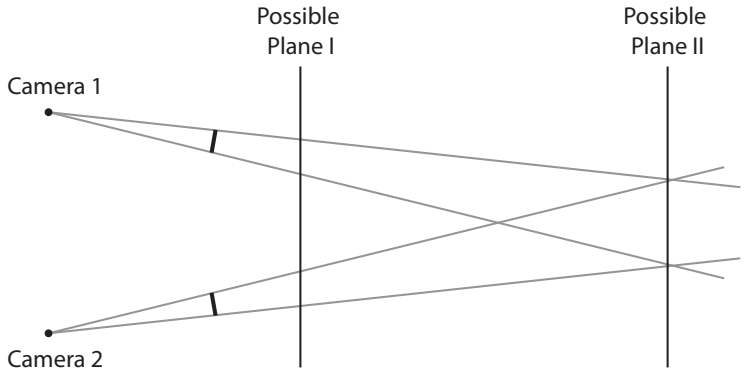


Figure 9.8: A diagram of two cameras, and two possible plane locations. It is seen that in the case of plane II the images from the two cameras are projected much closer together then in the case of plane I.

The origo, C of the plane can be determined more or less arbitrarily, as long as the two camera centers, Q_{c1} and Q_{c2}, are not located on the the plane A, B, C, whereby degeneracies will occur. This extends to the camera centers being located so close that numerical degeneracies will occur do to rounding errors. One solution is

$$C = \frac{Q_{c2} - Q_{c1}}{2} + N \cdot ||Q_{c2} - Q_{c1}||_2 \tag{9.4}$$

[2]It is assumed that this average is non-zero, for if the normals of the original planes pointed in opposite directions the images would likely not be seeing the same things.

[3]Not unit length in general.

[4]It is assumed that N and A are not collinear, in that this would almost certainly imply that one camera can se the other, in which case the method of [Pollefeys et al., 1999] would be recommended.

I.e. chose the point located half way between the two camera centers and move in the direction of N equal to the distance between the two camera centers. Possible subsequent algorithms, might, however, become simpler if the if the images projected onto the plane A, B, C are projected close to each other. Often the cameras are tilted towards each other such that there is a large common field of view. In this case, setting C equal to the 3D points closest to the optical axis' of the two cameras is often advisable. This point can be found by simple point triangulation, cf. Section 2.4. If the cameras are not tilted towards each other, the result of the triangulation might be at infinity or behind the cameras, neither of which is desirable.

As seen from Section 2.3, given A, B and C and the camera matrices the homographies from the plane A, B, C to the image planes are given by

$$\mathbf{P_1}\begin{bmatrix} A & B & C \\ 0 & 0 & 1 \end{bmatrix} \quad \text{and} \quad \mathbf{P_2}\begin{bmatrix} A & B & C \\ 0 & 0 & 1 \end{bmatrix} .$$

We are, however, seeking the inverse homogrphies from the image planes to the A, B, C plane. Thus a pair of *image rectifying homographies* are given by

$$\mathbf{H_1'} = \left(\mathbf{P_1}\begin{bmatrix} A & B & C \\ 0 & 0 & 1 \end{bmatrix}\right)^{-1} \quad \text{and} \quad \mathbf{H_2'} = \left(\mathbf{P_2}\begin{bmatrix} A & B & C \\ 0 & 0 & 1 \end{bmatrix}\right)^{-1} . \tag{9.5}$$

One issue, however, remains in determining the image rectifying homographies, namely scale. The length of the basis vectors A, B are set to one in the 3D coordinate system, mainly so that they have the same length. This implies that a pixel in the rectified images correspond to a unit square in the 3D coordinate system. This can give images of very few or very many pixels, i.e. if we set the 3D unit of measurement to a kilometer we would in many cases have a one pixel image. Thus a scale, s, should be applied to the homographies of (9.5), such that we get an acceptable resolution of the rectified images, i.e.

$$\mathbf{H_1} = \begin{bmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 1 \end{bmatrix}\mathbf{H_1'} \quad \text{and} \quad \mathbf{H_2} = \begin{bmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 1 \end{bmatrix}\mathbf{H_2'} . \tag{9.6}$$

The remaining question is then; how this scale s should be determined? The design criteria is, that we would ideally like a one to one correspondence between pixels in the original and the rectified images. In this way we would not under-sample and thereby throwing information away, and we would not over-sample whereby the rectified images would have too many pixels and subsequent algorithms in the processing pipeline would do an unnecessary amount of computations. Thus, s should be determined by the amount magnification the transformations $\mathbf{H_1'}$ and $\mathbf{H_2'}$ do. For a point $q = [x, y, 1]^T$ in the original images mapped by

$$\tilde{q} = \mathbf{H}q \ ,$$

to $\tilde{q} = [a, b, 1]^T$, the amount of magnification, m, can be approximated by the determinant of the map's jacobian, i.e.

$$m = \det\left(\begin{bmatrix} \frac{\partial a}{\partial x} & \frac{\partial a}{\partial y} \\ \frac{\partial b}{\partial x} & \frac{\partial b}{\partial y} \end{bmatrix}\right) . \tag{9.7}$$

A *possible* way of computing s is then to compute the magnification for the center pixel in each of the two images, denoted m_1 and m_2, and set s equal to

$$s = \frac{\alpha}{\min(|m_1|, |m_2|)} , \tag{9.8}$$

where α is a tuning parameter, e.g. set to $\alpha = 1.2$, to account for the center pixel not being the least magnified place of the original image. By using this scheme for determining s the magnification of the least magnified center pixel will be α. As an example of image rectification consider Figure 9.9. The image rectification algorithm is summarized in Table 9.1.

Lastly, as the camera matrices, $\mathbf{P_1}$ and $\mathbf{P_2}$, map from 3D points Q to image points q, i.e.

$$q_1 = \mathbf{P_1}Q \quad \text{and} \quad q_2 = \mathbf{P_2}Q \quad ,$$

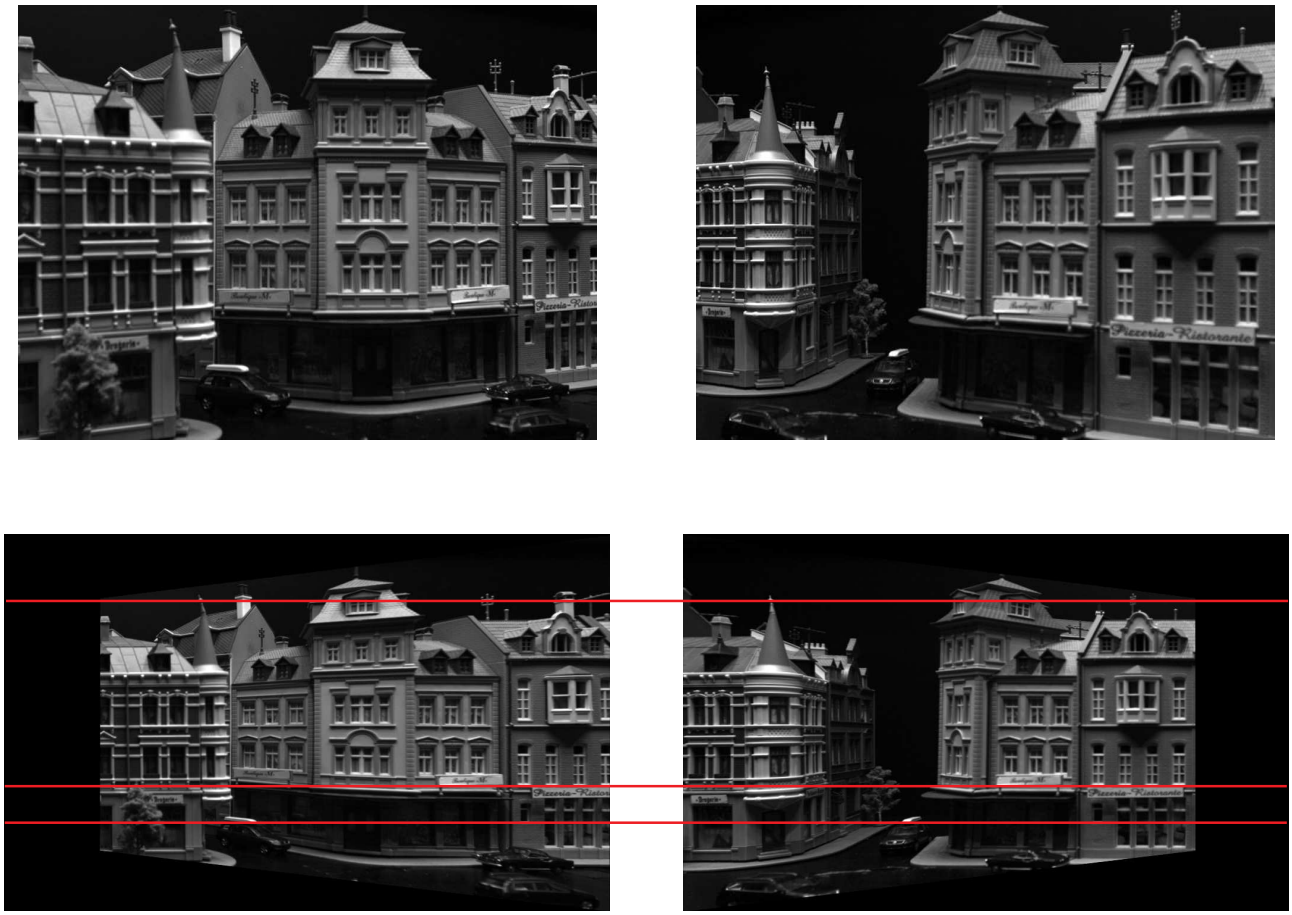


Figure 9.9: An example of image rectification. The input image pair is shown on the **top row**, and the rectified images are shown on the **bottom row**. Three sample epipolar lines are denoted on the rectified image pair in red. Note that they correspond to the scan lines. Please also verify that they are the epipolar lines in that corresponding features lye on the same lines.

the rectifying homographies map from image coordinates q to the rectified image coordinates $\tilde{q}$ on the plane A, B, C, i.e.

$$\tilde{q}_1 = \mathbf{H_1} q_1 \qquad \text{and} \qquad \tilde{q}_2 = \mathbf{P_2} q_2 \qquad .$$

These rectified image coordinates can be interpreted as emulated image planes and the camera matrices mapping from 3D coordinates, Q to these rectified image coordinates are given by

$$\begin{aligned} \tilde{q}_1 &= \mathbf{H_1} q_1 = \mathbf{H_1 P_1} Q = \tilde{\mathbf{P}}_1 Q \\ \tilde{q}_2 &= \mathbf{H_2} q_2 = \mathbf{H_2 P_2} Q = \tilde{\mathbf{P}}_2 Q \ . \end{aligned} \tag{9.9}$$

Thus defining the camera matrices for the rectified images.

9.4 Structured Light Scanners

Considering the plane based laser scanner presented in Section 9.2, it is somewhat limited that only one line is reconstructed per image frame pair, as illustrated in Figure 9.3. This also induces the need for a lot of mechanical motion, in that the object has to be moved relative to the laser for every frame. A method that uses all or most of the image sensor in every frame for doing 3D estimation thus has it's advantages. *Structured light scanners* are such methods. They can be seen as an extension of the two camera laser scanner in that instead of just projecting a plane of light onto the object they project an area of light. To do this a light projector is used instead of a laser. A device e.g. known from the delivery of Power Point presentations. A sample structured light scanner is seen in Figure 9.10. These structured light scanners produce very good 3D results as argued in [Scharstein et al., 2003], typically in a quality comparable to laser scanners, but typically much more efficient.

Given camera matrices

$$\mathbf{P_1} = \mathbf{A_1}[\mathbf{R_1}\ \mathbf{t_1}] \qquad \text{and} \qquad \mathbf{P_2} = \mathbf{A_2}[\mathbf{R_2}\ \mathbf{t_2}]\ .$$

Compute the camera centers

$$Q_{c1} = \mathbf{R_1^T t_1} \qquad \text{and} \qquad dQ_{c2} = \mathbf{R_2^T t_2}\ .$$

Find the basis A, B, C for the plane to project the images onto. Here A is found via (9.1),

$$A = \frac{Q_{c2} - Q_{c1}}{||Q_{c2} - Q_{c1}||_2}\ ,$$

furthermore B is found via (9.3),

$$\begin{aligned} B' &= \frac{1}{2}\left(\mathbf{R_1}^T \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} + \mathbf{R_2}^T \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}\right) \times A \\ B &= \frac{B'}{||B'||}\ . \end{aligned}$$

And C is found via (9.4) or preferably by point triangulation if eligible. Then the initial rectifying homographies are given by (9.5)

$$\mathbf{H_1'} = \left(\mathbf{P_1}\begin{bmatrix} A & B & C \\ 0 & 0 & 1 \end{bmatrix}\right)^{-1} \qquad \text{and} \quad \mathbf{H_2'} = \left(\mathbf{P_2}\begin{bmatrix} A & B & C \\ 0 & 0 & 1 \end{bmatrix}\right)^{-1}\ ,$$

determine the scale s, e.g. by (9.8), and apply it via (9.6)

$$\mathbf{H_1} = \begin{bmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 1 \end{bmatrix}\mathbf{H_1'} \qquad \text{and} \quad \mathbf{H_2} = \begin{bmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 1 \end{bmatrix}\mathbf{H_2'}\ .$$

Giving the final rectifying homographies. Then the associated camera matrices are given by (9.9)

$$\tilde{\mathbf{P}}_\mathbf{1} = \mathbf{H_1 P_1} \qquad \text{and} \qquad \tilde{\mathbf{P}}_\mathbf{2} = \mathbf{H_2 P_2}\ .$$

Table 9.1: Summary of the proposed image rectification algorithm.

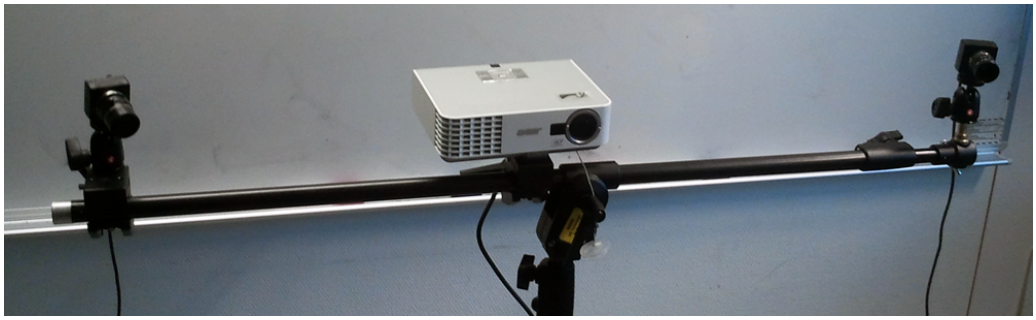


Figure 9.10: A basic structured light scanner consisting of two cameras and a light projector.

The issue with this extension from the laser scanner to structured light, is how correspondences between the image pairs are established. In the case of the laser scanner a search just had to be made along an epipolar line for the presence of reflected red laser light. In case of a whole area of the object being lit at the same time, as with structured light, this strategy is not useful any more. Thus, the light projected onto the object is structured – hence the name of the method – such that unique correspondences can be found along the epipolar lines, cf. Figure 9.17. A comparison of several common such structured light patterns can be found in [Batlle et al., 1998]. In the remainder of this section a structured light implementation using the hardware from Figure 9.10 and a Gray encoded pattern will be presented. Naturally many other structured light systems exist. The gray encoding is somewhat slow, but relatively easy to implement and robust.

9.4.1 Image Based Gray Encoding

Gray encoding is a binary coding scheme named after Frank Gray also known as reflected binary code. It has the property that only one bit will change value as the encoded value inceases by one. The four bit gray code is as follows:

0: 0000	8: 1100
1: 0001	9: 1101
2: 0011	10: 1111
3: 0010	11: 1110
4: 0110	12: 1010
5: 0111	13: 1011
6: 0101	14: 1001
7: 0100	15: 1000

This encoding can be used to encode image columns by projecting patterns as shown in Figure 9.11 onto a 3D object, resulting in captured images as seen in Figure 9.17. This is done one pattern at a time where each pattern encodes one bit. The link between the patterns and the encoding is illustrated in Figure 9.12. Here it is seen that black encode zeros and white ones. In this way the field of view of the projector is divided in to 2^b sections, where b is the number of bits/patterns used. Thus the part of the 3D objects surface in the field of view of the projector is ideally also divided into 2^b sections. These sections can be visually decoded by observing if they are lit or not by the projector, at the time of each patterns projection.

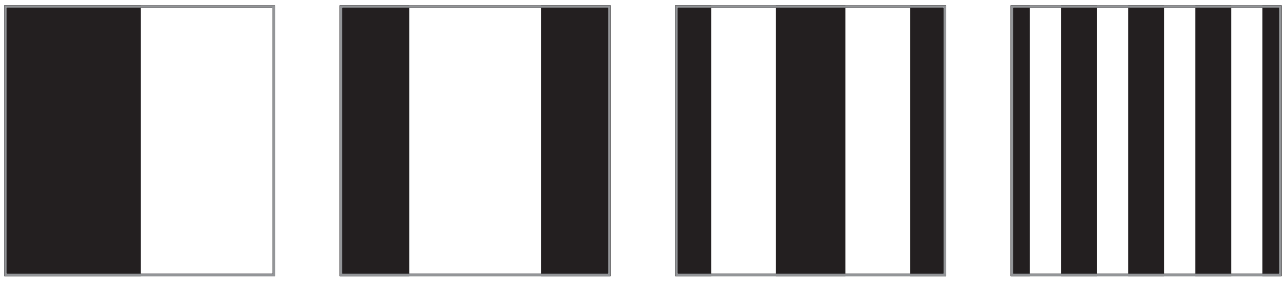


Figure 9.11: The first four image patterns used for Gray encoding, the link to the codes is seen in Figure 9.12.

In relation to 3D reconstruction the separating curves between the 2^b sections play the same role as the laser plane used in the laser scanner, cf. Figure 9.3. After a whole set of b patterns/frames this results in $2^b - 1$ lines. This is to be seen in contrast to the b lines obtained if a laser plane were used in b frames. If $b = 8$ then $2^b - 1 = 2^8 - 1 = 255$. As in the laser scanner case, point correspondences are found by the intersection of epipolar lines and the separating lines, cf. Figure 9.4.

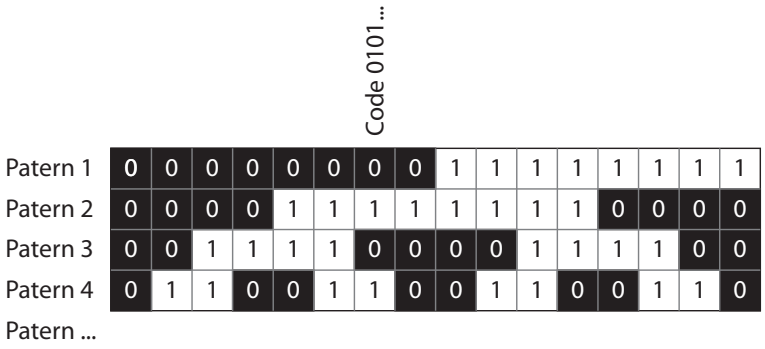


Figure 9.12: Link between Gray codes and image patterns.

The limit too how many patterns or bits, b, can be used is how narrow stripes can be detected and separated from each other. This has to be done by a camera viewing the 3D surface. The stripes should, thus, not get so narrow that they are 'drowned' by censor and quantization noise, in addition to the added complexity of the pattern being projected of a not necessarily smooth surface. In the below example $b = 8$ is used. In this sense the property of the Gray encoding, that it only changes one bit per value increment, is an advantage. For, as noted from Figures 9.11 and 9.12, this gives the double resolution compared to the stripe thickness, in that for every stripe of the last pattern there is a division further up, i.e. the last bit follows a sequence of two consecutive bits being equal. If a higher resolution is needed then dictated by the minimum stripe thickness, several scans can be made but with the stipe patterns shifted slightly between each scan.

Compared to other structured light encodings, cf. [Batlle et al., 1998], the Gray encoding is somewhat slow, in that a series of patters need to be projected in order to get a scan. This is compared to other patters where a complete scan can be made from *one* frame or pattern. This is done by utilizing that projected images can contain many other colors than black and white. The Gray encoding scheme's sole dependence on black and white images also makes it robust to interfering light and changing coloring of the 3D object. This is because only using black and white images in a sense maximizes the signal to noise ratio.

9.4.2 Implementation Example

As an example, an to give the reader an idea of how to construct an active scanner, a structured light scanner software algorithm aimed at the hardware in Figure 9.10, will be outlined here. The focus will be on the reconstruction algorithm given the input images of the right and left cameras, after the projector has projected the Gray coded patterns onto the object. A note should be attached to how the projected patters are generated. This can either be done by loading and displaying the respective images, but a considerable speed up can be achieved by having the graphics card compute the patters. In the following it is assumed that the cameras have been calibrated and that their camera matrices are known.

Occlusion Mask

The object viewed in this example is the garden gnome seen in Figure 9.13. The first task in the 3D reconstruction algorithm is to determine which parts of the object or scene are actually lit by the projector, i.e. where there is a significant difference between a balck and a white stripe being projected onto the scene. This is done by projecting a fully white and a fully black image and capturing images there of with the camera pair. These two set of images are then compared via subtraction, as seen in Figure 9.14. The pixels where there is an insignificant difference between the lit(white) and unlit(black) projections are deemed occluded. To account for noise this occlusion mask is filtering by an opening operation, other noise reduction techniques could be considered e.g. using a binary Markov random field. Refer to Figure 9.14-Right for the result.

Figure 9.13: The object scanned in this example in the form of a garden gnome. It is here seen from the left and the right camera, with the projector displaying a fully white image.

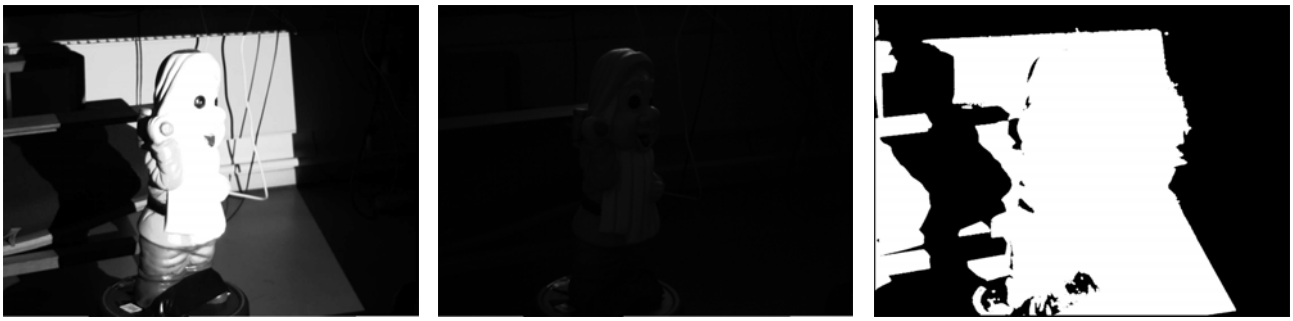


Figure 9.14: Images from the right camera after a fully white and a fully black projection onto the object, left and center. To the right the resulting occlusion mask is seen.

Encoding

Following this, the different stripe segments of the image are encoded by considering the captured images of the stripe patters projected onto the 3D object, cf. Figure 9.17. To increase the robustness, and to avoid tuning a threshold between white and black, the inverted patters are also projected onto the 3D object and depicted. These inverted patterns are made by switching white pixels to black and vise versa. The projection of these are the subtracted from each other forming a difference image. Projections of white stripes are then where the difference image has positive values and the image is not occluded, as determined by occlusion mask described above. Based on these eight set of difference images, corresponding to the eight patterns of Figure 9.17, an encoding of each non-occluded pixel can be computed as seen in Figure 9.15.

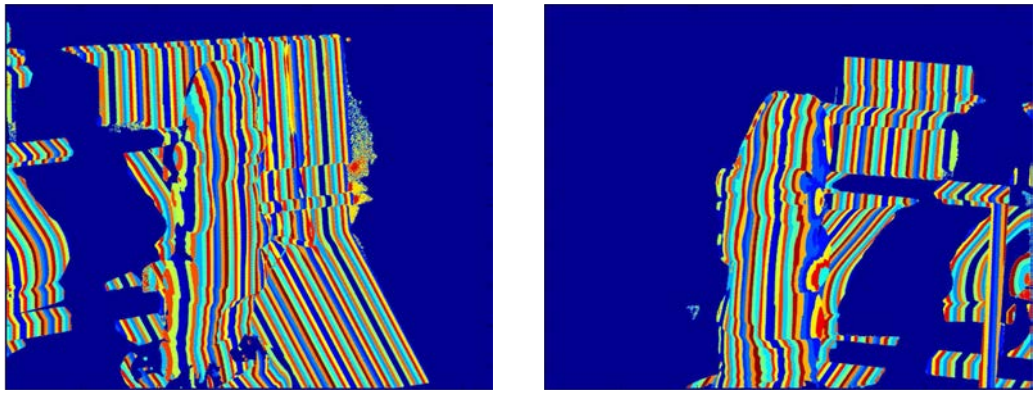


Figure 9.15: The encoding of the pixels of the right and left cameras.

When encoding the images, it is noted that numbering is needed to achieve a unique labeling and not because the numerical values in them selves matter. I.e. the purpose is to use these labelings to establish image correspondences between the right and left images later. As such it is *easier* to encode the labels via normal binary code then via Gray code. More formally, let i denote the pattern number and let p^i_{xy} denote the value

1. Sort both the RightEdge and LeftEdge data structures via the ordering relation of (9.11) .
2. Remove doubles from RightEdge and LeftEdge, i.e. entries with the same Label_1 and Label_2. This is done to remove the ambiguities there would otherwise pose. It is done easily, since after sorting such doubles will be located next to each other.
3. Reset the indices into the data structures, i.e. $i = 1$ and $j = 1$.
4. Repeat will $i \leq$LengthOf(RightEdge) and $j \leq$LengthOf(LeftEdge):
   (a) If $\text{RightEdge}_i == \text{LeftEdge}_j$ we have a match.
   (b) If $\text{RightEdge}_i < \text{LeftEdge}_j$ according to (9.11) do $i = i + 1$.
   (c) If $\text{RightEdge}_i > \text{LeftEdge}_j$ according to (9.11) do $j = j + 1$.

Table 9.2: Pseudo code for matching scan lines of the rectified encoded images.

of pixel (x, y) of the difference image corresponding to the i^{th} pattern. The the label of pixel (x, y) can be computed as

$$\text{Label}_{x,y} = \sum_{i=1}^{b} 2^{i \cdot (p^i_{xy} > 0)} \quad . \tag{9.10}$$

Were $p^i_{xy} > 0$ represents the indicator function such that $p^i_{xy} > 0$ is zero for negative or zero p^i_{xy} and one for positive p^i_{xy}.

Correspondence and 3D Point Triangulation

The next step is finding the image correspondences, where the defining points are the intersections between the epipolar lines and the separating curves between the stripes. To compute these correspondences the encoded images are rectified wrt. the camera geometry as described in Section 9.3. Thus the correspondence problem is reduced to matching scan lines.

For each scan line of a rectified encoded image the separating points between the stipes are found by traversing it and recording when a pixels label is different from it's left neighbor. Such a separating point is stored in a data structure with three elements per entry, namely the two labels and the column position. I.e. for the i^{th} edge of the right or left image respectively

$$\text{RightEdge}_i = [\text{Label}_1, \text{Label}_2, \text{Col}] \qquad \text{LeftEdge}_i = [\text{Label}_1, \text{Label}_2, \text{Col}] \quad .$$

For improved results, the column position, Col, should be refined by sub-pixel interpolation in the difference image. If this is to be done easily and efficiently the difference images should also be rectified.

To find the correspondences entries in the RightEdge and LeftEdge data structures with matching Label_1 and Label_2 have to be found. To do this both the RightEdge and the LeftEdge data structures are sorted via the ordering relation

$$\text{Edge}_i > \text{Edge}_j \quad \Leftrightarrow \quad \left(\text{Edge}_i.\text{Label}_1 \cdot 2^b + \text{Edge}_i.\text{Label}_2\right) > \left(\text{Edge}_j.\text{Label}_1 \cdot 2^b + \text{Edge}_j.\text{Label}_2\right) \quad . \tag{9.11}$$

The matches can then be found by relatively simple algorithm of Table 9.2. For each match the corresponding 3D points can be found via triangulation, cf. Section 2.4. Note that since the image coordinates come from the rectified images the camera matrices $\tilde{\mathbf{P}}_\mathbf{1}$ and $\tilde{\mathbf{P}}_\mathbf{2}$ should be used, cf. Table 9.1. The resulting 3D reconstruction of the garden gnome is seen in Figure 9.16.

Figure 9.16: The reconstructed result of the garden gnome, partly based on the structured light patterns from Figure 9.17. The reconstruction is based on five consecutive scans with eight bits or patterns. Each scan is shifted slightly relative to the previous to give a higher resolution.

9.5 A Note on Accuracy

In both the laser scanning case and in the structured light scanning case the notes made about accuracy in Chapter 2, also apply. Thus the base line depth ratio should be kept between a third and three, and radial distortion should be corrected for, if high precision is required.

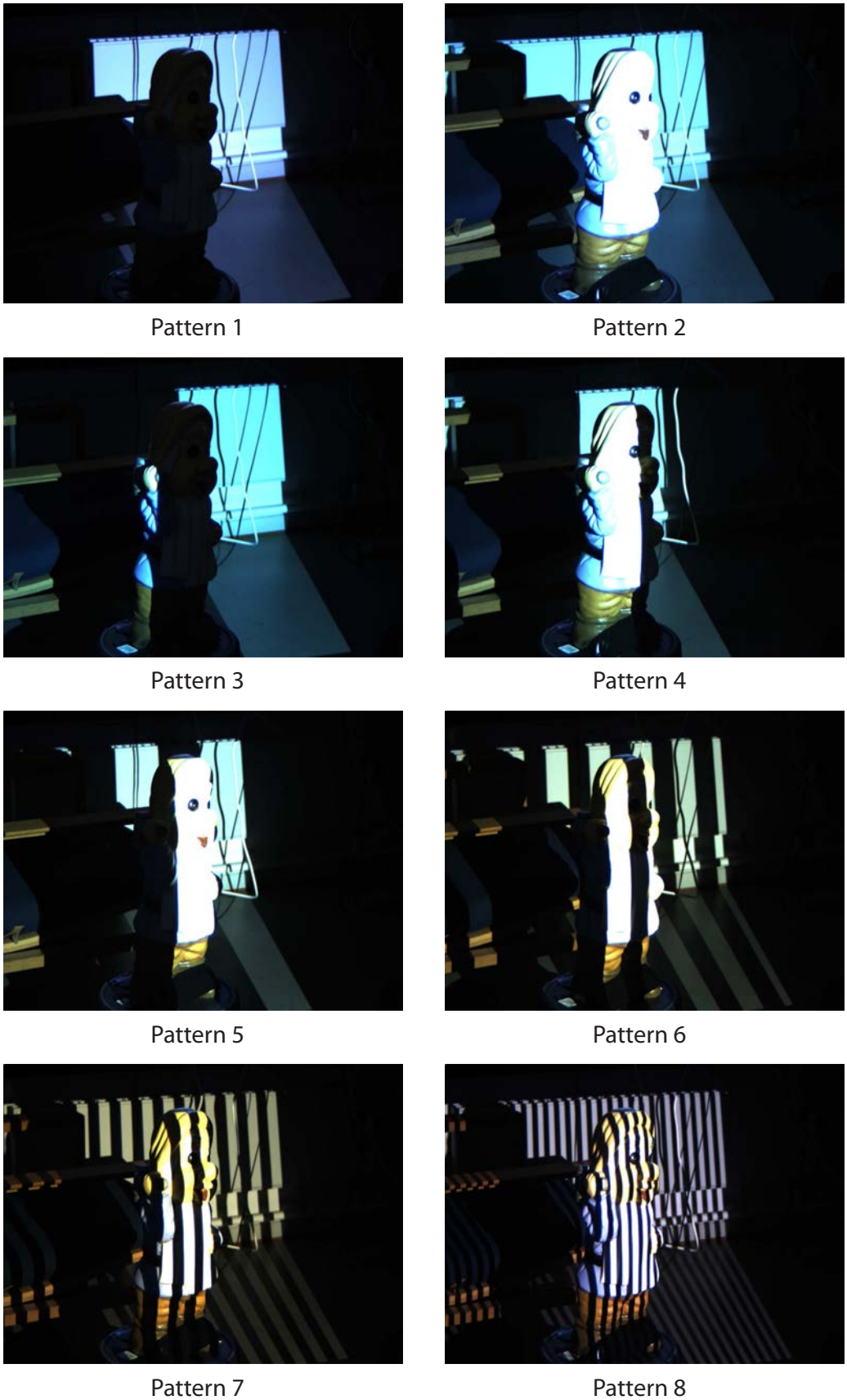


Figure 9.17: The images from the right camera when the eight stripe patterns are projected onto the 3D object.

Part VI

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 [Eising, 1997, Golub and van Loan, 1996], 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.

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

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

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

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

where σ is a permutation of the numbers $1, \ldots, n$, and S_n is the set of all such permutations. The function 'sign' 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

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

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.

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

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

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

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

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

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 this *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. This is sometimes called an *eigendecomposition*. 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

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

is $\mathbf{v}$ equal to the eigenvector corresponding tho the largest eigenvalue of $\mathbf{A^T}\mathbf{A}$. 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,

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

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

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

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 $\mathbf{A^T}\mathbf{A}$.

[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.

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

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 matrix*, $\mathbf{R}$, 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.

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

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

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

- 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.

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

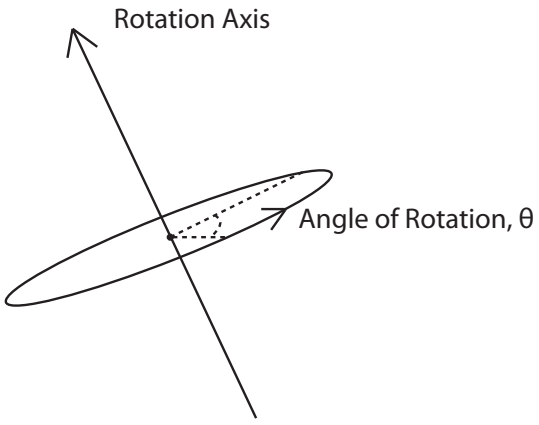


Figure A.1: A rotation matrix rotates all points with an angle of θ around an axis.

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, cf. e.g. [Bishop and Goldberg, 1980], 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, cf. e.g. [Akenine-Möller and Haines, 2002, Hamilton, 1853], 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.

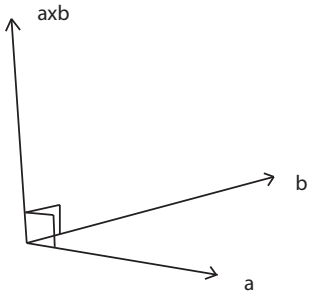


Figure A.2: The cross product $\mathbf{a} \times \mathbf{b}$ between vector $\mathbf{a}$ and $\mathbf{b}$. Note the right hand rule, which determines the direction of $\mathbf{a} \times \mathbf{b}$.

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

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

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

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

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

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

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

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

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

A.6 SVD – Generalized Eigen Values

The *singular value decomposition(SVD)* [Hansen, 1998] 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 $m \times n$)

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

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

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

[5] Largely a matter of definition.

That is

$$\begin{aligned}\mathbf{\Sigma} &= \left[\begin{array}{cc}\tilde{\mathbf{\Sigma}} & \mathbf{0}\end{array}\right] \quad , \quad m \leq n \\ \mathbf{\Sigma} &= \left[\begin{array}{c}\tilde{\mathbf{\Sigma}} \\ \mathbf{0}\end{array}\right] \quad , \quad m > n \\ \tilde{\mathbf{\Sigma}} &= \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_k) \ .\end{aligned}$$

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.

$$\mathbf{A} = \left[\begin{array}{cccc}\mathbf{a_1} & \mathbf{a_2} & \ldots & \mathbf{a_n}\end{array}\right] \ ,$$

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

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

[6] All eigenvalues are positive or zero

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. $\mathbf{A}$, cf. [Papadopoulo and Lourakis, 200

The *Frobenius norm* of matrices is given by

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

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

A.7 Kronecker Product

The *Kronecker product* [Graham, 1981, van Loan, 2000], 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 $vec(\mathbf{A}\mathbf{v}) = \mathbf{A}\mathbf{v}$ since $\mathbf{A}\mathbf{v}$ 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 $\mathbb{R}$ numbers. The presented material applies almost unchanged for a variety of other domains. In particular every thing holds for complex numbers, $\mathbb{C}$, 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

$$\begin{bmatrix} 1+i & 1+2i \\ -1+i & -1-i \\ i & 2 \end{bmatrix}^H = \begin{bmatrix} 1-i & -1-i & -i \\ 1-2i & -1+i & 2 \end{bmatrix} \ .$$

Appendix B

A Short Overview of Statistics

According to the Merriam Webster dictionary Statistics is " a branch of mathematics dealing with the collection, analysis, interpretation, and presentation of masses of numerical data", and as such statistics naturally plays a large role in computer vision. As with linear algebra cf. Appendix A, experience has shown that a brief overview of the field is called for, and as such given here. Thus this appendix, which attempts at giving a reasonably sized overview of the statistical concepts most central to computer vision, and not a text book. As a consequence this text is not as stringent as is typical in statistics textbooks, especially in the distinction between empirical or estimated statistics and the theoretically underlying statistics. If a proper textbook is sought, recommended introductory books include [Conradsen, 2002, Johnson, 2010] and more advanced [Anderson, 1984, Bishop, 2007, Hastie et al., 2009, MacKay, 2002], where especially [Hastie et al., 2009] is among this author's favorites.

B.1 Probability

A basic building block of statistics is probability, where the probability of an event e happening is denoted

$$p(e) \quad \text{where} \quad 0 \leq p(e) \leq 1 \ . \tag{B.1}$$

This should e.g. be understood as, if we conducted n experiments, we should expect e to happen $p(e) \cdot n$ times. If $n = 100$ then e would happen $p(e) \cdot 100\%$ of the time. As an example consider one of the classical examples from probability theory, namely flipping a coin. Assuming a 'fair' coin then

$$p(heads) = 0.5 \quad \text{and} \quad p(tails) = 0.5 \ .$$

An event happening with probability of one is the same as absolute certainty of that event happening, and if exactly one of several events, $e_1, \ldots, e_n$ will happen then

$$\sum_{i=1}^{n} p(e_i) = 1 \ . \tag{B.2}$$

In the case of flipping a coin which will end up heads or tails

$$p(heads) + p(tails) = 0.5 + 0.5 = 1 \ .$$

Expanding the above example, let us assume that we get 1\$ if the result is heads and nothing if the result is tails, then our expected earnings would be

$$p(heads) \cdot 1\$ + p(tails) \cdot 0 = 0.5 \cdot 1\$ + 0.5 \cdot 0 = 0.5\$ \ .$$

Conducting this experiment twice, i.e. two tosses, the expected outcome would be distributed as

1^{st} toss	2^{nd} toss	probability	profit
heads	heads	$0.5 \cdot 0.5 = 0.25$	2\$
heads	tails	$0.5 \cdot 0.5 = 0.25$	1\$
tails	heads	$0.5 \cdot 0.5 = 0.25$	1\$
tails	tails	$0.5 \cdot 0.5 = 0.25$	0\$

Here the rule that given two independent events e_1 and e_2 then the probability of them both happening is equal to[1]

$$p(e_1 \wedge e_2) = p(e_1) \cdot p(e_2) \ , \tag{B.3}$$

where independent implies that the occurrence of one event has no implication on the other event happening.

This leads to conditional probability, i.e. the probability that event e_2 will happen given that event e_1 has happened. This is denoted by

$$p(e_2|e_1) \ . \tag{B.4}$$

Considering the two toss example from above, then

$$p(profit = 2\$) = 0.25 \ ,$$

but given the first toss is a head then the probability of making a 2$ profit is equal to 0.5, i.e.

$$p(profit = 2\$|1^{st}toss = heads) = 0.5 \ ,$$

and

$$p(profit = 2\$|1^{st}toss = tails) = 0 \ ,$$

in that there is no way that a 2$ profit can be made if the first toss was tails. The formal definition of conditional probability is

$$p(e_2|e_1) = \frac{p(e_1 \wedge e_2)}{p(e_1)} \ . \tag{B.5}$$

Which states that the the probability that event e_2 will happen, given event e_1 has happened, is equal to the probability that both will happen corrected for the probability that e_1 will happen. This correction, is the division by $p(e_1)$. To exemplify (B.5)

$$p(profit = 2\$|1^{st}toss = heads) = \frac{p(profit = 2\$ \wedge 1^{st}toss = heads)}{p(1^{st}toss = heads)} = \frac{0.25}{0.5} = 0.5 \ ,$$

as indicated above. In terms of conditional probability independence is defined by

$$p(e_1|e_2) = p(e_1) \quad \text{and} \quad p(e_2|e_1) = p(e_2) \ . \tag{B.6}$$

B.2 Probability Distribution Functions (PDFs) and Histograms

The data in the above example where a coin is flipped twice, can also be illustrated as a histogram, as done in Figure B.1-Left, where the probability of the different outcomes are graphed. Extending the example to flipping the coin six times produces the histogram Figure B.1-Right.

Such histograms of the different possible outcomes probability, or more formally the underlying functions, are known as probability distribution functions abbreviated "pdf". Probability distribution functions constitute a central concept in statistics and probability theory and are used to described stochastic phenomena. Such phenomena are often referred to as processes and/or described as stochastic variables. These variables are said to follow a distribution, e.g. if a stochastic variable x_i was normal distributed with mean zero and variance one, cf. Section B.4, this could be written as

$$x_i \in N(0, 1) \ ,$$

where it is noted that the $\in$ symbol is used to denote the relationship instead of e.g. $=$. Much introductory statistics deals with defining and describing different such probability distribution functions, with various motivations and properties. Examples of these include the normal distribution, the binomial distribution, the Poisson distribution, the logarithmic normal distribution, the logistic distribution and the Cauchy distribution.

Many of these distributions, however, describe continuous random variables, i.e. instead of discrete events, like the flip of a coin, it is continues measurements like peoples heights or distances. Such continuous random variables are also described well by probability distribution functions. An issue with a continuous (outcome) domain is, however, that there are infinitely many outcomes, and as such the probability of each is typically

[1] Here $\wedge$ denotes logical *and*.

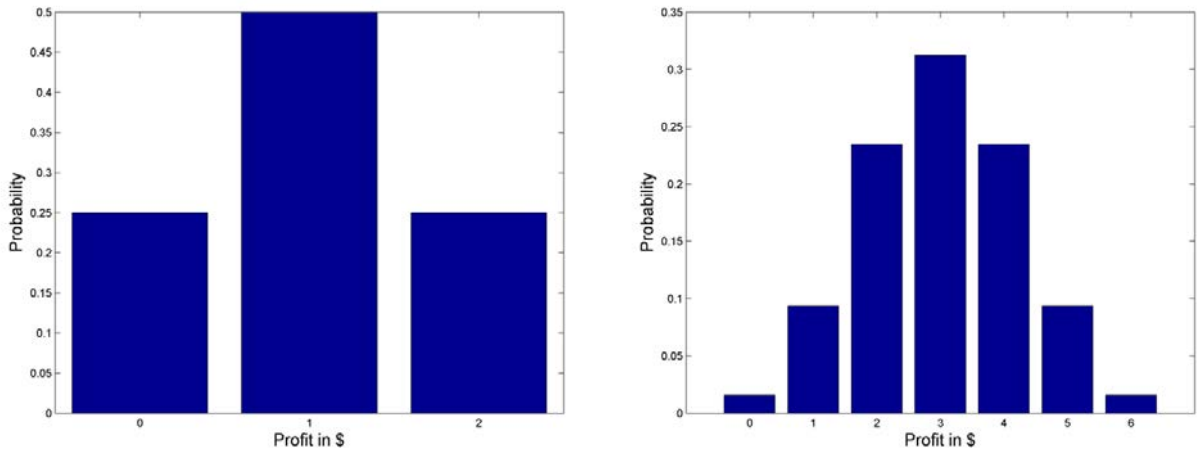


Figure B.1: Histogram or probability distribution function of the two coin flip example (**left**). And the same example to the **right** except for the coin being flipped six times. In both cases the outcome is the number of heads.

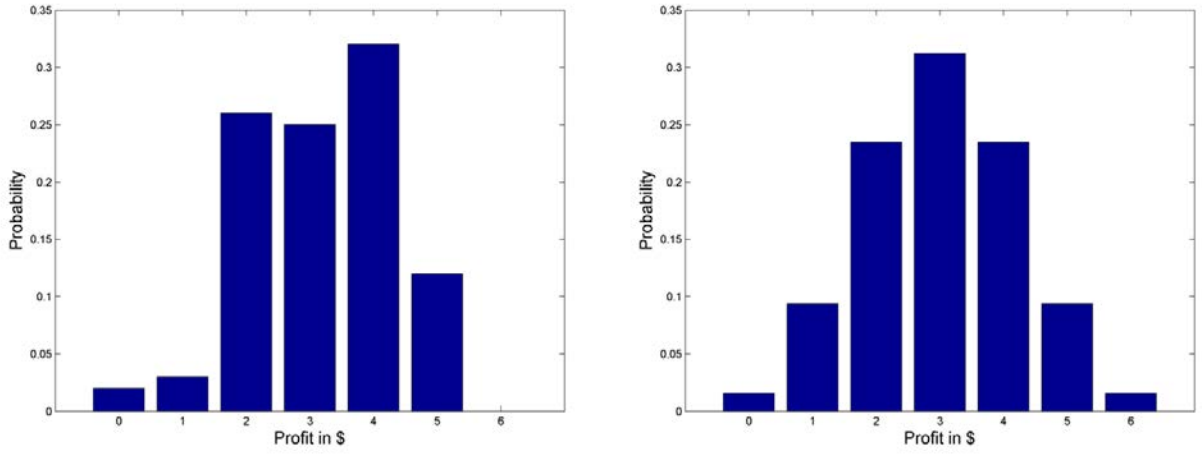


Figure B.2: The empirical pdf (**left**) and true pdf (**right**) for the example from Figure B.1, where a coin is flipped six times and the number of heads is the outcome. Note that with the empirical pdf, the underlying histogram is is normalized such that it complies with (B.9).

infinitely small. This problem is circumvented via a mathematical trick, consisting of first integrating and then differentiating again. More formally, consider the stochastic variable, x_i, with a continuous outcome, then integrating is equivalent to

$$f(y) = p(x_i \leq y) \ ,$$

i.e. the probability that x_i is less then y. Then the probability distribution function is given by

$$pdf(y) = \frac{\partial p(x_i \leq y)}{\partial y} \ . \tag{B.7}$$

Using a similar formalism, if x_i has a discrete outcome, e.g. the flip of a coin, the probability distribution function would be given by

$$pdf(y) = p(x_i = y) \ . \tag{B.8}$$

An example of a discrete pdf is given in Figure B.1, for the flip of a coin, and an example of a continuous pdf is the normal distribution, cf. Figure B.4-Left. It should be noted that since the stochastic variable x_i will have *one* value from it's pdf with absolute certainty then in line with (B.2)

$$\sum pdf(y) = 1 \qquad \text{or} \qquad \int pdf(y)\partial y = 1 \ , \tag{B.9}$$

if the pdf is discrete or continuous respectively.

For many stomatic variables or processes, it is not possible (or practical) to derive at a pdf axiomatically via logical reasoning as done in the case of flipping several coins. In such cases pdfs are often obtained by observing the stomatic variable and making a histogram. In such cases we talk of the empirical pdf. The underlying histogram is normalized such that it complies with (B.9). As an example the flipping of six coins was simulated 100 times, and the empirical pdf constructed, cf. Figure B.2. In these cases the empirical pdf is also a stochastic variable, or contaminated with noise, as can be seen in Figure B.2, where there is a discrepancy between the empirical and the true/theoretical pdf.

B.3 Mean and Variance

An often used technique in statistics is calculating – hopefully – meaningful numbers based on collections of data. Such a number is often referred to as a statistic. The most popular statistic is probably the mean value, which for n observations x_i is defined as

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i \ . \tag{B.10}$$

As an example consider the following ten measurements:

$$15.4311,\ 16.7329,\ 13.6509,\ 12.6682,\ 15.1532,\ 17.1888,\ 15.6237,\ 11.7019,\ 16.7428,\ 13.8032 \ . \tag{B.11}$$

Here

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i = \frac{1}{10}148.697 = 14.8697 \ .$$

An interpretation of the mean is that it in some sense describes the typical measurement. An obvious follow up question in this regard is how well the measurements are described by this mean, or how much do they on average deviate from it. That is we would like to make statistics on

$$x_i - \mu \ .$$

The obvious would be the mean of $x_i - \mu$, this is, however, is equal to

$$\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu) = \frac{1}{n}\sum_{i=1}^{n} x_i - \mu\frac{1}{n}\sum_{i=1}^{n} 1 = \mu - \mu = 0 \ .$$

Instead the mean of the deviation squared is considered, thus only the size, not the sign, of the deviation is considered. This statistic is termed the variance, σ^2, and is defined by:

$$\sigma^2 = \mathrm{var}(x) = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu)^2 \ . \tag{B.12}$$

The variance, in essence, estimates how much the measurements vary. The reason that $n-1$ and not n is divided by, is that one measurement or degree of freedom is used for the mean, μ. This implies that the variance of one observation is undefined. As an example, the variance of the measurements in (B.11) is given by

$$\sigma^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu)^2 = \frac{1}{10-1}\sum_{i=1}^{10}(x_i - 14.8697)^2 = \frac{1}{9}30.8268 = 3.4252 \ .$$

The standard deviation, given by

$$\sigma = \sqrt{\sigma^2} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu)^2} \ , \tag{B.13}$$

is reported instead of the variance.

B.3.1 Effects of Additions and Affine Transformation

Often measurements are added with each other and or have a scalar added or multiplied onto them. The effects of these operations on the mean and variance are covered here. Consider first an affine transformation, i.e. $y_i = \alpha x_i + \beta$, then

$$\mu_y = \frac{1}{n}\sum_{i=1}^{n} y_i = \frac{1}{n}\sum_{i=1}^{n}(\alpha x_i + \beta) = \alpha\frac{1}{n}\sum_{i=1}^{n} x_i + \beta\frac{1}{n}\sum_{i=1}^{n} 1 = \alpha\mu_x + \beta \ , \tag{B.14}$$

where

$$\mu_x = \frac{1}{n}\sum_{i=1}^{n} x_i$$

is the mean of x_i. In case of the variance

$$\begin{aligned}\sigma_y^2 &= \frac{1}{n-1}\sum_{i=1}^{n}(y_i - \mu_y)^2 = \frac{1}{n-1}\sum_{i=1}^{n}(\alpha x_i + \beta - \alpha\mu_x - \beta)^2 \\ &= \frac{1}{n-1}\sum_{i=1}^{n}(\alpha x_i - \alpha\mu_x)^2 = \frac{\alpha^2}{n-1}\sum_{i=1}^{n}(x_i - \mu_x)^2 = \alpha^2\sigma_x^2 \ ,\end{aligned} \tag{B.15}$$

where

$$\sigma_x^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu_x)^2$$

is the variance of x_i. Lastly, if two measurements, x_i and z_i, are added, then the mean becomes

$$\mu_{x+z} = \frac{1}{n}\sum_{i=1}^{n}(x_i + z_i) = \frac{1}{n}\sum_{i=1}^{n} x_i + \frac{1}{n}\sum_{i=1}^{n} z_i = \mu_x + \mu_z \ . \tag{B.16}$$

In case of addition and variance refer to Section B.3.4.

B.3.2 Expectation — the Theoretical Mean

In the above it is implied that the mean and variance are calculated based on observed data of a finite sample size. There is, however, also a need for calculating these quantities of distributions given by a pdf, e.g. as part of more rigorous probabilistic reasoning. It is noted, that the theoretical statistics are seldom achieved from samples, consider e.g. Figure B.2, also the numbers in (B.11), are drawn from a normal distribution with mean 14 and variance 4.

Starting with an example, assume the two coin flip example from above was conducted 100 times, and that these experiments were in perfect accordance with the theoretical pdf, then the outcome would be zero 25 times, one 50 times and two 25 times. Thus the mean would be

$$\mu = \frac{1}{100}(25\cdot 0 + 30\cdot 1 + 25\cdot 2) = 1 \ .$$

This suggests the follow the following algorithm for calculating the theoretical mean of variables belonging to a distribution with discrete scalar outcomes, x_i, where the x_i have probability $p(x_i)$

$$E(x) = \sum_i x_i \cdot p(x_i) \ . \tag{B.17}$$

This theoretical mean is also referred to as the expectation value, and is denoted $E(x)$. For continuous pdfs the expectation value is given by

$$E(x) = \int x \cdot p(x)\, dx \ . \tag{B.18}$$

In terms of these expectations the variance is given by, via the same line of reasoning as above

$$E\left((x - E(x))^2\right) \ . \tag{B.19}$$

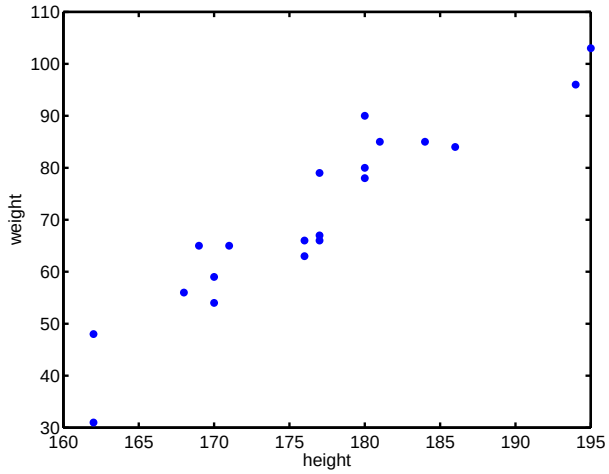


Figure B.3: Plot of corresponding height and weight of twenty people. Note that taller people tend to be heavier.

B.3.3 Paired Observations

Often measurements consists of multiple variables, and statistics for quantifying the interaction of these variables are thus useful. As an example, consider the measurements of height and weight of twenty people in the following table and illustrated in Figure B.3.

Person	Height cm	Weight kg	Person	Height cm	Weight kg
1	176	66	11	194	96
2	162	31	12	176	63
3	186	84	13	170	59
4	168	56	14	177	66
5	177	67	15	171	65
6	195	103	16	181	85
7	180	80	17	162	48
8	177	79	18	180	90
9	184	85	19	169	65
10	170	54	20	180	78

From Figure B.3, it is, as expected, seen that there is a tendency for taller people to be heavier. This tendency can be described as; if a person is taller than average then it is likely that that person is also heavier then average, or the mean, and similarly if a person is shorter than average. It is this relation we want to make a statistic for quantifying. The standard way of doing this is via covariance, which for two variables x_i and y_i is formulated as

$$\sigma_{xy} = \mathrm{cov}(x, y) = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \mu_x)(y_i - \mu_y) \quad , \tag{B.20}$$

where μ_x and μ_y are the the means of x_i and y_i respectively. The analogy with the variance is obvious in that a variables co-variance with it self is its variance, i.e.

$$\sigma_x^2 = \sigma_{xx} = \mathrm{cov}(x, x) = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \mu_x)(x_i - \mu_x) = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \mu_x)^2 = \sigma_x^2 \quad .$$

Putting (B.20) in the context of the tendency analogy of height and weight above, consider the product the two terms

$$(x_i - \mu_x)(y_i - \mu_y) \quad .$$

If both these terms have the *same* sign, this product is positive, and negative if they have *opposite* signs. In (B.20) these products of terms are summed up, and if there is an overweight of positive terms, corresponding to the tenancy we are trying to quantify, the covariance will be positive. On the other hand, if there is no tendency,

then we would expect an equal number of positive and negative terms, with numerical values such that they would approximately cancel out. This is thus a motivation for using covariance to quantify the interaction of variables. Continuing the example from Figure B.3, with x_i representing height and y_i weight, then[2]

$$\begin{aligned}
\mu_x &= \frac{1}{n}\sum_{i=1}^{n} x_i = \frac{1}{20}3535 = 176.75 \\
\mu_y &= \frac{1}{n}\sum_{i=1}^{n} y_i = \frac{1}{20}1420 = 71 \\
\sigma_x^2 &= \text{var}(x) = \frac{1}{n-1}\sum_{i=1}^{n} (x_i - \mu_x)^2 = \frac{1}{19}1515.8 = 79.78 \\
\sigma_y^2 &= \text{var}(y) = \frac{1}{n-1}\sum_{i=1}^{n} (y_i - \mu_y)^2 = \frac{1}{19}5754 = 302.84 \\
\sigma_{xy} &= \text{cov}(x,y) = \frac{1}{n-1}\sum_{i=1}^{n} (x_i - \mu_x)(y_i - \mu_y) = \frac{1}{19}2757 = 145.11
\end{aligned}$$

indicating a significant interaction or covariance between height and weight. It is also noted that two measurements can also have a negative interaction or covariance. As an extension of the height weight example, where there is a positive covariance, considered the interaction between height and head room[3] in a given car model, then we would assume that taller people would have less head room, and e.g. if the height is above average then the head room would be below, directly implying a negative covariance. Thus low interaction is the same as a small numerical covariance and a large interaction is the same as a high numerical covariance.

In the above example, if the height were measured in meters instead of centimeters – corresponding to multiplying the x_i variable with 0.01 – then the covariance calculations would look as follows,

$$\begin{aligned}
\mu_x &= \frac{1}{n}\sum_{i=1}^{n} x_i = \frac{1}{20}35.35 = 1.7675 \\
\mu_y &= \frac{1}{n}\sum_{i=1}^{n} y_i = \frac{1}{20}1420 = 71 \\
\sigma_x^2 &= \text{var}(x) = \frac{1}{n-1}\sum_{i=1}^{n} (x_i - \mu_x)^2 = \frac{1}{19}0.15158 = 0.007978 \\
\sigma_y^2 &= \text{var}(y) = \frac{1}{n-1}\sum_{i=1}^{n} (y_i - \mu_y)^2 = \frac{1}{19}5754 = 302.84 \\
\sigma_{xy} &= \text{cov}(x,y) = \frac{1}{n-1}\sum_{i=1}^{n} (x_i - \mu_x)(y_i - \mu_y) = \frac{1}{19}27.57 = 1.4511 \ .
\end{aligned}$$

This is in line with application of an affine transformation as described above, where the effect of an affine transformation on covariance is

$$\begin{aligned}
\text{cov}(\alpha x + \beta, y) &= \frac{1}{n-1}\sum_{i=1}^{n} (\alpha x + \beta - \mu_{\alpha x+\beta})(y_i - \mu_y) \\
&= \frac{1}{n-1}\sum_{i=1}^{n} (\alpha x + \beta - (\alpha\mu_x + \beta))(y_i - \mu_y) \\
&= \frac{1}{n-1}\sum_{i=1}^{n} \alpha (x - \mu_x)(y_i - \mu_y) \\
&= \alpha \cdot \text{cov}(x,y) \ . \qquad \text{(B.21)}
\end{aligned}$$

[2]The variances are computed for later use.

[3]Distance from head to roof.

Thus by changing the measurement unit from meters to centimeters the covariance between height and weight is decreased by a factor of 100. Thus there is a need for a unit free statistic of interaction, where the typical choice is *correlation*, which is given by

$$\text{corr}(x,y) = \frac{\sigma_{xy}}{\sqrt{\sigma_x^2}\sqrt{\sigma_y^2}} = \frac{\text{cov}(x,y)}{\sqrt{\text{var}(x)}\sqrt{\text{var}(y)}} \ . \tag{B.22}$$

Which is seen to be invariant to an affine transformation of the measurement variable, in that

$$\text{corr}(\alpha x+\beta) = \frac{\text{cov}(\alpha x+\beta,y)}{\sqrt{\text{var}(\alpha x+\beta)}\sqrt{\text{var}(y)}} = \frac{\alpha\cdot\text{cov}(x,y)}{\sqrt{\alpha^2\cdot\text{var}(x)}\sqrt{\text{var}(y)}} = \frac{\text{cov}(x,y)}{\sqrt{\text{var}(x)}\sqrt{\text{var}(y)}} = \text{corr}(x)$$

Thus the correlation between height and weight in the above example is, for height in cm

$$\text{corr}(x,y) = \frac{\sigma_{xy}}{\sqrt{\sigma_x^2}\sqrt{\sigma_y^2}} = \frac{145.11}{\sqrt{79.78}\sqrt{302.84}} = 0.9336 \ ,$$

and for height in m

$$\text{corr}(x,y) = \frac{\sigma_{xy}}{\sqrt{\sigma_x^2}\sqrt{\sigma_y^2}} = \frac{1.4511}{\sqrt{0.007978}\sqrt{302.84}} = 0.9336 \ ,$$

i.e. the same. Lastly, consider the collection of mean corrected measurements as vectors, i.e. $\tilde{\mathbf{x}} = [x_1 - \mu_x, \ldots, x_n - \mu_x]^T$ and $\tilde{\mathbf{y}} = [y_1 - \mu_y, \ldots, y_n - \mu_y]^T$, then

$$\begin{aligned}
\text{cov}(x,y) &= \frac{1}{n-1}\sum_{i=1}^{n}(x_i-\mu_x)(y_i-\mu_y) = \frac{1}{n-1}\tilde{\mathbf{x}}^T\tilde{\mathbf{y}} \\
\text{var}(x) &= \frac{1}{n-1}\sum_{i=1}^{n}(x_i-\mu_x)(x_i-\mu_x) = \frac{1}{n-1}\tilde{\mathbf{x}}^T\tilde{\mathbf{x}} \\
\text{var}(y) &= \frac{1}{n-1}\sum_{i=1}^{n}(y_i-\mu_y)(y_i-\mu_y) = \frac{1}{n-1}\tilde{\mathbf{y}}^T\tilde{\mathbf{y}} \\
\text{corr}(x,y) &= \frac{\text{cov}(x,y)}{\sqrt{\text{var}(x)}\sqrt{\text{var}(y)}} = \frac{\frac{1}{n-1}\tilde{\mathbf{x}}^T\tilde{\mathbf{y}}}{\sqrt{\frac{1}{n-1}\tilde{\mathbf{x}}^T\tilde{\mathbf{x}}}\sqrt{\frac{1}{n-1}\tilde{\mathbf{y}}^T\tilde{\mathbf{y}}}} = \frac{\tilde{\mathbf{x}}^T\tilde{\mathbf{y}}}{\sqrt{\tilde{\mathbf{x}}^T\tilde{\mathbf{x}}}\sqrt{\tilde{\mathbf{y}}^T\tilde{\mathbf{y}}}}
\end{aligned}$$

Thus variance and covariance can be seen in terms of inner products of the mean corrected measurements and the correlation as the normalized inner products, furthermore

$$\text{corr}(x,y) = \frac{\tilde{\mathbf{x}}^T\tilde{\mathbf{y}}}{\sqrt{\tilde{\mathbf{x}}^T\tilde{\mathbf{x}}}\sqrt{\tilde{\mathbf{y}}^T\tilde{\mathbf{y}}}} = \arccos\left(\angle(\tilde{\mathbf{x}},\tilde{\mathbf{y}})\right) \ ,$$

i.e. arcus/inverse cosine to the angle between $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{y}}$.

B.3.4 Effects of Additions and Affine Transformation – Continued

Let us consider addition in relation to covariance. From (B.21) it is seen that the addition of a constant has no effect, and we thus only consider the linear combination of two stochastic variable x_i and y_i, i.e.

$$\begin{aligned}
\text{cov}(\alpha x+\beta y,\zeta x+\xi y) &= \frac{1}{n-1}\sum_{i=1}^{n}\left(\alpha(x_i-\mu_x)+\beta(y_i-\mu_y)\right)\left(\zeta(x_i-\mu_x)+\xi(y_i-\mu_y)\right) \\
&= \frac{1}{n-1}\sum_{i=1}^{n}\left(\alpha\zeta(x_i-\mu_x)^2+\beta\xi(y_i-\mu_y)^2+(\alpha\xi+\beta\zeta)(x_i-\mu_x)(y_i-\mu_y)\right) \\
&= \frac{\alpha\zeta}{n-1}\sum_{i=1}^{n}(x_i-\mu_x)^2+\frac{\beta\xi}{n-1}\sum_{i=1}^{n}(y_i-\mu_y)^2+\frac{\alpha\xi+\beta\zeta}{n-1}\sum_{i=1}^{n}(x_i-\mu_x)(y_i-\mu_y) \\
&= \alpha\zeta\text{var}(x)+\beta\xi\text{var}(y)+(\alpha\xi+\beta\zeta)\text{cov}(x,y) \ .
\end{aligned} \tag{B.23}$$

Noting that $\text{var}(x) = \text{cov}(x, x)$, equation (B.23) implies that

$$\text{var}(\alpha x + \beta y) = \alpha^2 \text{var}(x) + \beta^2 \text{var}(x) + 2\alpha\beta \text{cov}(x, y) \ . \tag{B.24}$$

Extending to a linear combination of an arbitrary number of *independent* variables $\sum_{j=1}^{m} \alpha_j \mathbf{x_j}$ and $\sum_{j=1}^{m} \beta_j \mathbf{x_j}$ – independent implying that $\text{cov}(\mathbf{x}_i, \mathbf{x}_j) = 0$ for $i \neq j$ – equation (B.23) extends to

$$\text{cov}\left(\sum_{j=1}^{m} \alpha_j \mathbf{x_j}, \sum_{j=1}^{m} \beta_j \mathbf{x_j}\right) = \sum_{j=1}^{m} \alpha_j \beta_j \text{var}(\mathbf{x_j}) \ . \tag{B.25}$$

Similarly for (B.24)

$$\text{var}\left(\sum_{j=1}^{m} \alpha_j \mathbf{x_j}\right) = \sum_{j=1}^{m} \alpha_j^2 \text{var}(\mathbf{x_j}) \ . \tag{B.26}$$

B.4 The Normal or Gaussian Distribution

The most commonly used distribution is probably the normal or Gaussian[4] distribution, with a pdf illustrated in Figure B.4-Left and given by the following formula

$$pdf(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} \ , \tag{B.27}$$

where μ is the mean and σ is the standard deviation. If a variable, x_i follows a normal distribution it is e.g. denoted by

$$x_i \in N(\mu, \sigma^2) \ .$$

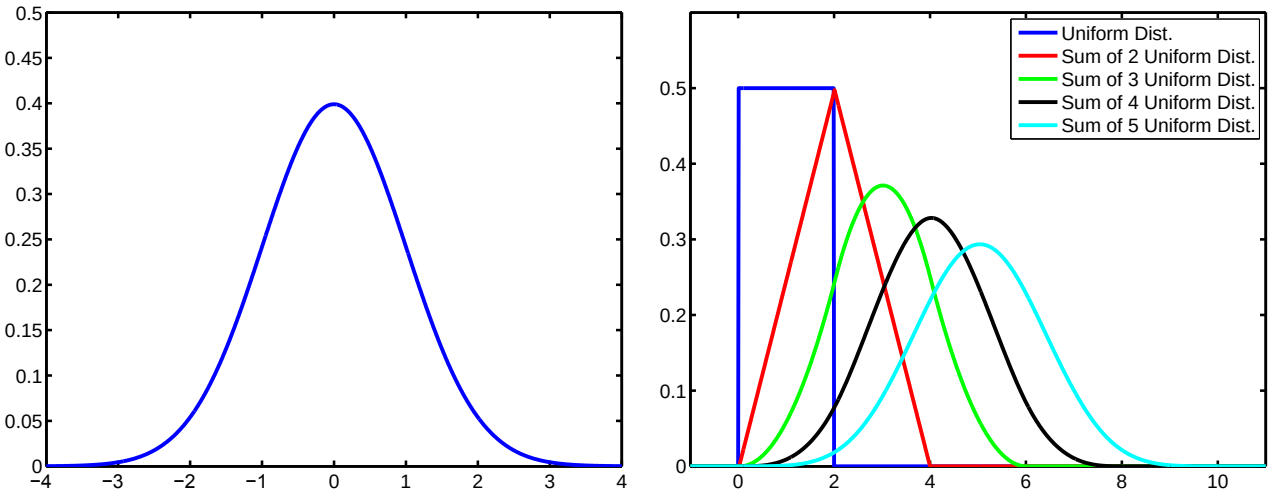


Figure B.4: **Left:** The probability distribution function (pdf) of the normal distribution with mean $\mu = 0$ and variance $\sigma^2 = 1$. **Right:** The pdf of a uniform distribution (dark blue), $U(0, 2)$ and the pdf of the sum of two or more independent variables, where each variable follows the uniform distribution, $U(0, 2)$. Note how fast it approximates the normal distribution.

B.4.1 Additivity and the Normal Distribution

One of the reasons for the normal distribution's popularity is its close link to adding stochastic variables. Firstly, the sum of two normal distributions is also a normal distribution, in that for two independent normal distributed variables $x_i \in N(\mu_x, \sigma_x^2)$ and $y_i \in N(\mu_y, \sigma_y^2)$

$$(x_i + y_i) \in N(\mu_x + \mu_y, \sigma_x^2 + \sigma_y^2) \ . \tag{B.28}$$

[4] After C.F. Gauss

The rules for addition and affine transformations wrt. mean and variance, derived in Section B.3.1, naturally also apply to the normal distribution, which is completely parameterized by the mean and variance. Thus for independent normal distributed variables $x_i \in N(\mu_i, \sigma_i^2)$

$$\left(\alpha + \sum_{i=1}^{n} \beta_i x_i\right) \in N\left(\alpha + \sum_{i=1}^{n} \beta_i \mu_i \ , \ \sum_{i=1}^{n} \beta_i^2 \sigma_i^2\right) \quad . \tag{B.29}$$

The link between addition of stochastic variables and the normal distribution is stronger than (B.29), in that the addition of any type of stochastic variables, y_i will tend towards a normal distribution. This is known as the **central limit theorem** and is a cornerstone in statistics. It states that for independent stochastic variables[5], y_i, with mean μ_i and variance σ_i^2, then for their sum, x_i

$$\begin{aligned} x &= \sum_{i=1}^{n} y_i \\ pdf(x) &\rightarrow N\left(\sum_{i=1}^{n} \mu_i \ , \ \sum_{i=1}^{n} \sigma_i^2\right) \quad \text{for} \quad n \rightarrow \infty \ . \end{aligned} \tag{B.30}$$

This states; that the sum of a large enough number of independent stochastic variables will be approximately normal distributed. Here numbers around twenty are typically 'large enough'.

As an example consider the *uniform distribution* $U(a, b)$, which has a pdf of the form

$$pdf(x) = \begin{cases} \frac{1}{b-a} & a < x < b \\ 0 & else \end{cases} \quad . \tag{B.31}$$

See Figure B.4-Right for an illustration. In this figure, Figure B.4-Right, the pdf of the sum of several independent variables following the uniform distribution is illustrated. I.e. let

$$x_1 \in U(0,2) \quad , \quad x_2 \in U(0,2) \quad , \quad x_3 \in U(0,2) \quad , \quad x_4 \in U(0,2) \quad , \quad x_5 \in U(0,2) \ ,$$

and let

$$y_i = \sum_{i=1}^{n} x_i \ .$$

Then the pdf's for y_1, y_2, y_3, y_4, y_5 are illustrated in Figure B.4-Right. The point to note here is how fast these pdf's look like the pdf of a normal distribution, thus illustrating the essence of the central limit theorem.

An important implication of the central limit theorem is in relation to the *mean* μ of any type of stochastic variables. This statistic is in it self also a stochastic variable, given by the sum of other stochastic variables, i.e.

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

Since the mean is also a stochastic variable, it also has a pdf, which is typically well approximated by a normal distribution. The quality of this approximation naturally depend on the number of measurements.

B.4.2 Multiplication of Stochastic Variables

If instead of a sum of independent variables we had a multiplicative structure, i.e. product of independent stochastic variables, the fact that

$$\log(a \cdot b) = \log(a) + \log(b)$$

can be used for statistical analysis. Given a set of independent stochastic variables x_i, their logarithm, $\log(x_i)$ will also be independent stochastic variables, and as such by the central limit theorem,

$$\begin{aligned} y &= \log\left(\prod_{i=1}^{n} x_i\right) = \sum_{i=1}^{n} \log(x_i) \\ pdf(y) &\rightarrow N(\mu, \sigma^2) \quad \text{for} \quad n \rightarrow \infty \ . \end{aligned} \tag{B.32}$$

[5]There are some mild conditions on the distribution on these variables, which very seldom have any practical importance.

This gives rise to the definition of the *logarithmic normal distribution* $LN(\alpha, \beta^2)$ defined by

$$x_i \in LN(\alpha, \beta^2) \Leftrightarrow \log(x_i) \in N(\alpha, \beta^2) \ . \tag{B.33}$$

A last important distribution associated with the normal distribution is the χ^2-distribution[6], which arises when the square is taken of normal distributed variables, i.e. let x_i be independently normal distributed variables with mean zero and variance one, then

$$y \in \chi^2(n) \qquad \text{for} \quad y = \sum_{i=1}^{n} x_i^2 \ .$$

Among others, the χ^2-distribution is relevant when doing statistics on variance; for more on this matter please refer to a statistics text book, e.g. one of the books noted in the introduction to this appendix.

B.5 A Word on Statistical Modeling

If you buy a lottery ticket you do not know if it will win you the grand prize. It is, however, typically very unlikely that it will. This illustrates that not having perfect knowledge is not the same as not having any knowledge at all. Statistics is a good method for utilizing such imperfect or uncertain knowledge. In computer vision there is a proliferation of imperfect knowledge. Thus a predominant use of statistics within the field is as a tool for making models of the phenomena observed, e.g. pixel intensities of depicted brain and of bone tissue in medical image analysis. These models are then typically used as basis of inference, e.g. which parts of an image depict brain matter.

One use of statistics is the integration of several uncertain cue – i.e. pieces of imperfect knowledge – to make better inference. As an example, we can use both shape and geometry when trying to match two images to each other. Another use is the integration of our a priory assumptions or expectations with the data driven cues, e.g. via Bayes rule cf. Section B.6. As an example, an image based face detection algorithm consider both the image gradients and our assumptions or a priori knowledge of how faces should look. This adding of a priori knowledge or assumptions is often referred to as *regularization*.

A large part of using statistics in practice is the construction of the stochastic models, at the basis of our inference. On approach for doing this is via axiomatic deduction as done in the multiple coin tossing example above, where a stochastic model is derived from simple assumptions, such as the coin being fair. This approach, although the most intellectually satisfying, is often impractical, since it requires a minute knowledge of the generating physics. As an example consider the appearance of a face, it would require ground breaking knowledge of genetics, anthropology, sociology etc. to construct the generative model of a face's appearance, and even if this were possible, the resulting model would likely be too complex to be practically useful.

An often practical alternative to the axiomatic derivation of stochastic models is the so called *black box modeling*. Here the generating process to be modeled is treated as a black box, and suitable mathematical functions are sought to describe the phenomena well enough without any consideration to the generative process. As an example we could fit a line to the height and weight data above and base inference on this. In this terminology the above mentioned axiomatically derived models are sometimes referred to as *white box models*, and some times hybrids between the two are used which are referred to as *grey box models*.

A concern with black box modeling is, if the fitted model is good enough, is it e.g. too simple and does not capture the phenomenon well enough or is it too complex and is capturing random noise. The first is referred to as *under fitting* and the latter as *over fitting*. A vast amount of methods and literature exist on this topic, cf. e.g. [Hastie et al., 2009].

Another concern often raised in regards to black box modeling is; if we just fit mathematical functions to a phenomena without regards to its underlying nature, then the fitted model can be seen as incorrect in that it does not capture the nature of the phenomenon. This is a valid concern, *but* often times, e.g. in computer vision, the task is to solve a problem and not to uncover its deepest nature, and as such black box models are suitable, especially if the white box models are unobtainable or just to complicated for practical use. This does not imply that white box models are not preferred, if all else is equal, in e.g. that the chance of over or under fitting is eliminated or highly reduced. This difference between black and white box modeling is also central to the distinction between basic science and engineering as discussed in Section 1.2.1.

[6] Pronounced chi-squared distribution.

In relation to modeling the rule of thumb or heuristic known as *Occam's razor*[7] should also be mentioned. Occam's Razor is attributed to the 14th-century Franciscan friar Father William of Ockham (d'Okham) who wrote "entities must not be multiplied beyond necessity". In modeling this implies that there should be a good and compelling reason to increase the modeling complexity. One reason is that simpler models are less likely to over fit, cf. e.g. [MacKay, 2002].

Deterministic, i.e. non-stochastic models naturally also exits, and there is thus a differentiation between stochastic and deterministic models. Sometimes, models can be split into a deterministic and stochastic part added together, i.e.

$$\text{Model}_{\text{Total}} = \text{Model}_{\text{Deterministic}} + \text{Model}_{\text{Stochastic}} \ .$$

This is e.g. used when simplified but operational models are made, e.g. the approximation of the height-weight relationship above as a line, then a noise term, ε, is added to denote the imperfection of the model, i.e.

$$\text{Weight} = \alpha \cdot \text{Height} + \varepsilon \ .$$

Where α is a scalar and ε is a noise term. This stochastic noise term ε is often not modeled via data fitting but as such assumed or guestimated , typically to be normal distributed, based on the assumption of additive noise, cf. Section B.4.1.

B.6 Baye's Rule

Consider again (B.5), defining conditional probability, i.e.

$$p(e_2|e_1) = \frac{p(e_1 \wedge e_2)}{p(e_1)} \ .$$

Since

$$p(e_1 \wedge e_2) = p(e_2 \wedge e_1) \ ,$$

it can be deduced that

$$\begin{aligned} p(e_1 \wedge e_2) &= p(e_2 \wedge e_1) \quad \Rightarrow \quad \text{by (B.5)} \\ p(e_2|e_1)p(e_1) &= p(e_1|e_2)p(e_2) \quad \Rightarrow \\ p(e_1|e_2) &= \frac{p(e_2|e_1)p(e_1)}{p(e_2)} \ . \end{aligned} \tag{B.34}$$

This result is known as *Baye's rule* or *Baye's theorem*, and is one of the cornerstones of statistics, especially the statistics used in computer vision. A typical way of presenting Bayes rule is by exchanging e_1 for m denoting that a given model is true and e_2 with d denoting the observed data, then Bayes rule has the form

$$p(m|d) = \frac{p(d|m)p(m)}{p(d)} \ . \tag{B.35}$$

As an example, suppose that at a given school there are 60% boys and 40% girls, and that 80% of the boys have short hair as do 40% of the girls. Given that we have observed a short haired person, the chance that it was a girl we observed is calculated as follows, where d is the fact that we observed a short haired person, and m is that the person was a girl:

$$\begin{aligned} p(m) = p(\text{girl}) &= 0.4 \\ p(\text{boy}) &= 0.6 \\ p(d|m) = p(\text{short hair}|\text{girl}) &= 0.4 \\ p(\text{short hair}|\text{boy}) &= 0.8 \\ p(d) = p(\text{short hair}) &= p(\text{short hair}|\text{girl}) \cdot p(\text{girl}) + p(\text{short hair}|\text{boy}) \cdot p(\text{boy}) \\ &= 0.4 \cdot 0.4 + 0.8 \cdot 0.6 = 0.64 \\ p(m|d) = p(\text{girl}|\text{short hair}) &= \frac{p(d|m)p(m)}{p(d)} = \frac{0.4 \cdot 0.4}{0.64} = 0.25 \ . \end{aligned}$$

[7] Often expressed in Latin as the lex parsimoniae, translating to "law of parsimony".

The chance that if we see a short haired person at the given school, then there is a 25% chance that that person is a girl. The chance of that person being a boy is thus $(1 - 0.25) = 0.75$, which can also be calculated as

$$\begin{aligned}
p(\text{girl}) &= 0.4 \\
p(m) = p(\text{boy}) &= 0.6 \\
p(\text{short hair}|\text{girl}) &= 0.4 \\
p(d|m) = p(\text{short hair}|\text{boy}) &= 0.8 \\
p(d) = p(\text{short hair}) &= p(\text{short hair}|\text{girl}) \cdot p(\text{girl}) + p(\text{short hair}|\text{boy}) \cdot p(\text{boy}) \\
&= 0.4 \cdot 0.4 + 0.8 \cdot 0.6 = 0.64 \\
p(m|d) = p(\text{boy}|\text{short hair}) &= \frac{p(d|m)p(m)}{p(d)} = \frac{0.8 \cdot 0.6}{0.64} = 0.75 \ .
\end{aligned}$$

Thus the odds of the observed person being a boy is thus three to one. In computer vision the task at hand is often; given some data, d, what is the most likely model, e.g. in the above case it is most likely that the observed person is a boy. Since we thus want to compare $p(m_i|d)$ for different models m_i, the probability of the data $p(d)$ is of no significance, and often hard to come by. Thus Baye's rule is often simplified to

$$p(m_i|d) \propto p(d|m_i)p(m_i) \ , \tag{B.36}$$

where $\propto$ denotes "proportional to".

Bayes rule is so common that the terms on the right side of (B.36) have had specific terms attached to them. The term $p(d|m_i)$ is called the *data term* or *observation model* in that this is the link between the model and the data, or the specification of what is expected to be observed for a given model, e.g. the number of boys with short hair. The term $p(m_i)$ is called the *a priori* or *prior term* or plainly just *the prior*. The prior has no connection with the observed data, and just specifies the likelihood of a given model, m_i, i.e. there are 40% girls. The use of priors is a strong tool in computer vision, in that this is where our prior assumptions can be formulated and put into our modeling framework, e.g. that faces are relatively symmetric, which greatly aids many tasks.

B.7 Statistical Estimation and Testing

The previous part of this appendix has mainly dealt with building up a framework for statistical modeling. Such frameworks are typically constructed in order to perform inference. As an example: If we have n measurements, x_i, capturing the effect of a possible process improvement, does this improvement have a positive effect on average? To answer this question, denote the ground truth average improvement by μ, then the statistical task is to determine if $\mu > 0$. To do this we compute

$$\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} x_i \ ,$$

the mean of the measurements x_i. Note, however, that μ and $\hat{\mu}$ will in general *not* be equal, since there is noise on the measurements x_i. As indicated in Section B.4.1, if n is large enough, then it can be assumed that $\hat{\mu}$ follows a normal distribution, furthermore it can be assumed that the mean of this normal distribution is μ[8], i.e.

$$\hat{\mu} \in N(\mu, \sigma^2) \ .$$

Let $\mu = 2$ and $\sigma^2 = 4$, then the pdf of $\hat{\mu}$ is shown in Figure B.5-Left. Furthermore let us stipulate that the process improvement in question has a positive effect on average if $\hat{\mu} > 0$. Since this is a toy example, we know that $\mu = 2$ and as such that the process has a positive effect on average. Nevertheless, as seen from Figure B.5-Left, there is a considerable probability of concluding otherwise based on our decision rule – which by the way is quite reasonable. This probability is equal to 0.16, which is equal to the red area in Figure B.5-Left. Thus there is a 16% change of reaching the wrong conclusion in this setup.

This example illustrates several key issues in statistical inference. Firstly, we do not have direct access to the underlying parameters of various distributions, e.g. the mean. We can only estimate these via *estimators*,

[8] See a introductory statistics textbook for a more in depth and rigorous derivation of this argument.

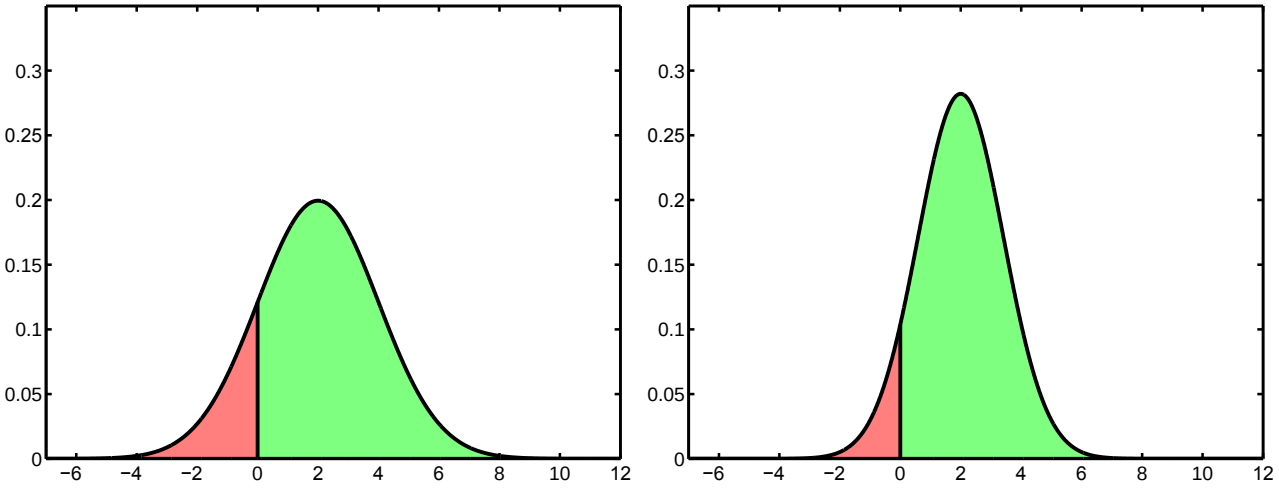


Figure B.5: The pdf of $\hat{\mu} \in N(\mu, \sigma^2)$, where the probability that $\hat{\mu} < 0$ is equal to the read area. In both cases $\mu = 2$. To the **left** $\sigma^2 = 4$, to the **right** $\sigma^2 = 2$ corresponding to twice as many samples being used to estimate $\hat{\mu}$, compared to the left.

e.g. the mean of the observations. There is a vast number of estimators of different properties in different distributions, and a vast theory surrounding these, but this is beyond the scope here. As a consequence, it is important to distinguish between a value and an estimate of it, this distinction is often made in the nomenclature by putting a $\hat{}$ over the estimate[9]. In the case of the mean

$$\mu = E(x_i) \qquad \text{and} \quad \hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} x_i \ ,$$

for a finite number of n measurements. It can, however, be shown that $\hat{\mu}$, in its above form, is central i.e. that

$$\hat{\mu} \to \mu \quad \text{for} \quad n \to \infty \ .$$

Secondly, in statistics few conclusions can be made with absolute certainty, or probability one. E.g. in the case from Figure B.5-Left, the probability of $\hat{\mu} < -6$ is 0.00003, which is small but still not zero. So in this case, we can not even conclude with absolute certainty that $\hat{\mu}$ will be above -6. This is dealt with by attaching a probability or *significance level* to a conclusion. E.g. for a given estimate $\hat{\mu}$ there is a 95% chance or significance that $\mu > 0$. Often times this is turned around, and a so called *null hypothesis* is formed, e.g.

$$H0: \quad \mu > 0 \ ,$$

and we require that the probability of this hypothesis being true is significant on a given level, e.g. a 90% level, implying that the probability of the hypothesis or conclusion being true is above 90%. Most introductory statistical textbooks are heavily concerned with ways to compute the probability of various common statistical hypothes, and the interested reader is referred there for more information on this subject.

A typical way of increasing the significance level of one's conclusions is by taking more samples. Consider e.g. in the case from Figure B.5-Left, and assume we had conducted the experiment twice, we i.e. had

$$\hat{\mu}_1 \in N(\mu, \sigma^2) \quad \text{and} \quad \hat{\mu}_2 \in N(\mu, \sigma^2) \ ,$$

and we constructed the new estimator

$$\hat{\mu}_3 = \frac{1}{2}\hat{\mu}_1 + \frac{1}{2}\hat{\mu}_2 \ .$$

Then by (B.29)

$$\hat{\mu} \in N\left(\frac{\mu}{2} + \frac{\mu}{2}\,,\, \frac{\sigma^2}{2^2} + \frac{\sigma^2}{2^2}\right) = N\left(\mu\,,\, \frac{\sigma^2}{2}\right) \ ,$$

i.e. the variance is halved by taking twice as many measurements, cf. Figure B.5-Right.

[9] This distinction is held a little lax in this appendix, as mentioned in its introduction.

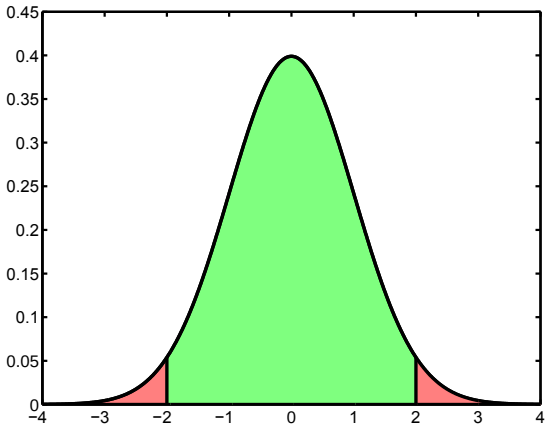


Figure B.6: The pdf of a normal distribution describing the test statistic Z. The read areas corresponding to $|Z| > 2$ correspond to 0.05 of the probability mass, corresponding to accepting $H0$ on a 95% level.

A very common analysis conducted is; given two processes, $x_i \in N(\mu_x, \sigma^2)$ and $y_i \in N(\mu_y, \sigma^2)$, is one better than the other, measured via the mean, i.e. the following hypothesis should be tested

$$H0: \quad \mu_x = \mu_y \qquad \text{alternatively} \qquad H1: \quad \mu_x \neq \mu_y \ .$$

Note that μ_x, μ_y and σ are unknown, and that only the x_i and y_i are known. A good test size or test statistic, Z, for evaluating $H0$ is the difference between the estimated means normalized by the estimated standard deviation, $\hat{\sigma}$, estimated as

$$\hat{\sigma}^2 = \frac{1}{n+m-1}\left(\sum_{i=1}^{n}(x_i - \mu_x)^2 + \sum_{i=1}^{m}(y_i - \mu_y)^2\right) \ ,$$

i.e.

$$Z = \frac{\hat{\mu_x} - \hat{\mu_y}}{\sqrt{\hat{\sigma}^2}} \tag{B.37}$$

Given that hypothesis $H0$ is true, Z will be an element of a Student's t-distribution. For enough samples this distribution becomes indistinguishable from becomes a normal distribution, with zero mean and unit standard deviation. If we want to accept hypothesis $H1$ on a 95% level, we note that this is equivalent to rejecting $H0$ on a 5% level. This is illustrated in Figure B.6, where the areas marked in red are 97.5% level for the hypothesis that $\mu_x < \mu_y$ and the 97.5% level for the hypothesis that $\mu_y < \mu_x$. Thus $H1$ is accepted on a 95% level if the numerical value of Z is larger than $1.960 \approx 2$. If we alternatively wanted to test for $\mu_x > \mu_y$, then via similar argumentation this hypothesis would be accepted on a 95% level if $Z > 1.645$ and on a 97.5% level if $Z > 1.960 \approx 2$.

B.7.1 Univariate Classification

Another type of statistical inference typically performed is classification, i.e. determining weather an observation, x belongs to one of two (or more) classes, A and B. Which is it most likely to belong to? One of two hypothesis is thus true

$$H0: \quad x \text{ belongs to } A \qquad \text{alternatively} \qquad H1: \quad x \text{ belongs to } B \ .$$

It is often assumed that A and B are described by normal distributions. As an example let us assume that observations of class A have the pdf $N(3,4)$ and that the observations of class B have a the pdf $N(7,4)$, i.e. the means are given by $\mu_A = 3$ and $\mu_B = 7$. This setting is illustrated in Figure B.7, and the two hypothesis become

$$H0: \quad x \in N(3,4) \qquad \text{alternatively} \qquad H1: \quad x \in N(7,4) \ .$$

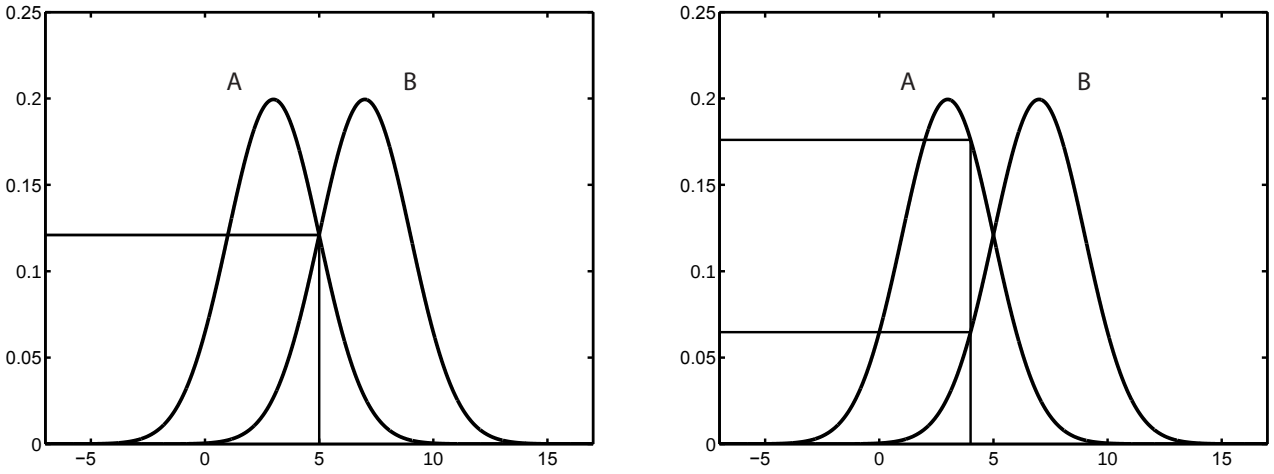


Figure B.7: Two normal distributed pdfs and a classification threshold or parameter τ. To the **left** the τ is chosen such that the most probable class is chosen. To the **right** a τ is chosen such that a higher probability is needed if class A should be chosen.

A straight forward decision rule or *classifier* is to choose $H0$ if $p(H0) > p(H1)$, which in the example corresponds to choose $H0$ if $x < \tau$ for $\tau = 5$ as seen in Figure B.7-Left. Again, this classification is not done with absolute certainty and the probability of making errors is equal to the red and blue areas of Figure B.8-Left.

Sometimes the consequence of of making classification errors differs between the different classes, and as such the classifier should be adapted. Consider e.g. that you should classify a parachute you are about to jump out of a plane with as either

$H0:$ Parachute is ok, so you jump. alternatively $H1:$ Parachute is *not* ok, so fly back down. .

The consequence if making a mistake in each of he cases is death or a missed parachute dive, and as such one would like a higher certainty before accepting $H0$. An example if this is illustrated in Figure B.7-Right where the probability of making classification errors is illustrated in Figure B.8-Right.

B.7.2 Type I & Type II Errors and ROC Curves

With statistical classification of inference, errors are generally a real possibility. A fact that has to be taken into consideration. Also, as in the parachute example, the consequence of different errors can vary considerably. Thus, in the framework of accepting or rejecting a hypothesis, $H0$, as illustrated above and often used, we talk of two types of errors:

- **Type I:** Erroneously accepting the hypothesis $H0$, also known as a **false positive**.
- **Type II:** Erroneously rejecting the hypothesis $H0$, also known as a **false negative**.

This implies that the relationship between the truth and our estimate can be characterized as done in Figure B.9. In relation to Figure B.8 and the hypothesis

$$H0: \quad x \text{ belongs to } A \ .$$

The probability of making a type I error is equal to the red area, and the probability of making a type II error is equal to the blue area.

It is naturally desirable to be able to evaluate a classifier's performance in terms of its type I and type II errors. But, as seen from the above, the classifier, and the relation between type I and type II errors, often depend on a tuning parameter, here τ. To quantify the performance of classifiers irrespectively of such tuning parameters, receiver operating characteristic (ROC), or simply ROC curves are often made. Here the false positive(=1-true negative) rate versus the true positive(=1-false negative) rate is plotted as a parameter curve

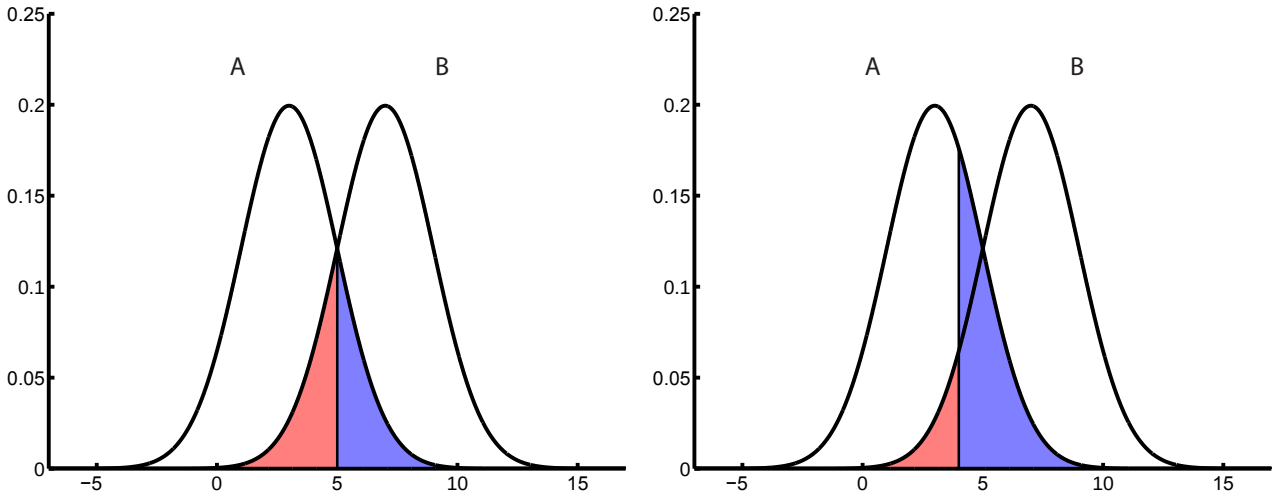


Figure B.8: The error rates from Figure B.7 is equal to the red and blue areas. Given the hypothesis $H0$: x belongs to A, then the probability of type I errors is equal to the red area, and the probability of type II errors are equal to the blue area.

		Truth	
		H0 is true	H0 is false
Estimate	H0 is true	True Positive	False Positive Type I Error
	H0 is false	False Negative Type II Error	True Negative

Figure B.9: Given a hypothesis $H0$ the relation ship between the truth and our estimate can be characterized as shown here.

of the tuning parameter as seen in Figure B.10. The area under this ROC curve is then used as a quantifiable performance measure, whereby the effect of the tuning parameter has been integrated out. The larger the area – which is between zero and one – the better, since if the curve can pass through $(0, 1)$, corresponding to a 100% true positive rate and a 0% false positive rate, it is a perfect classifier.

B.8 Model Fitting and Maximum Likelihood

Apart from testing, another large part of statistics is model fitting, i.e. given some noisy data, what is the most likely model for describing these data. A common way of doing this is to find this most likely model by optimizing over the free parameters in the model. This is the so called *maximum likelihood method*. As an example reconsider the ten measurements from (B.11), and assume that these are independent and drawn from a normal distribution with standard deviation two, i.e. $N(\mu, \sigma^2)$ for $\sigma = 2$. Then the model to be fitted to the

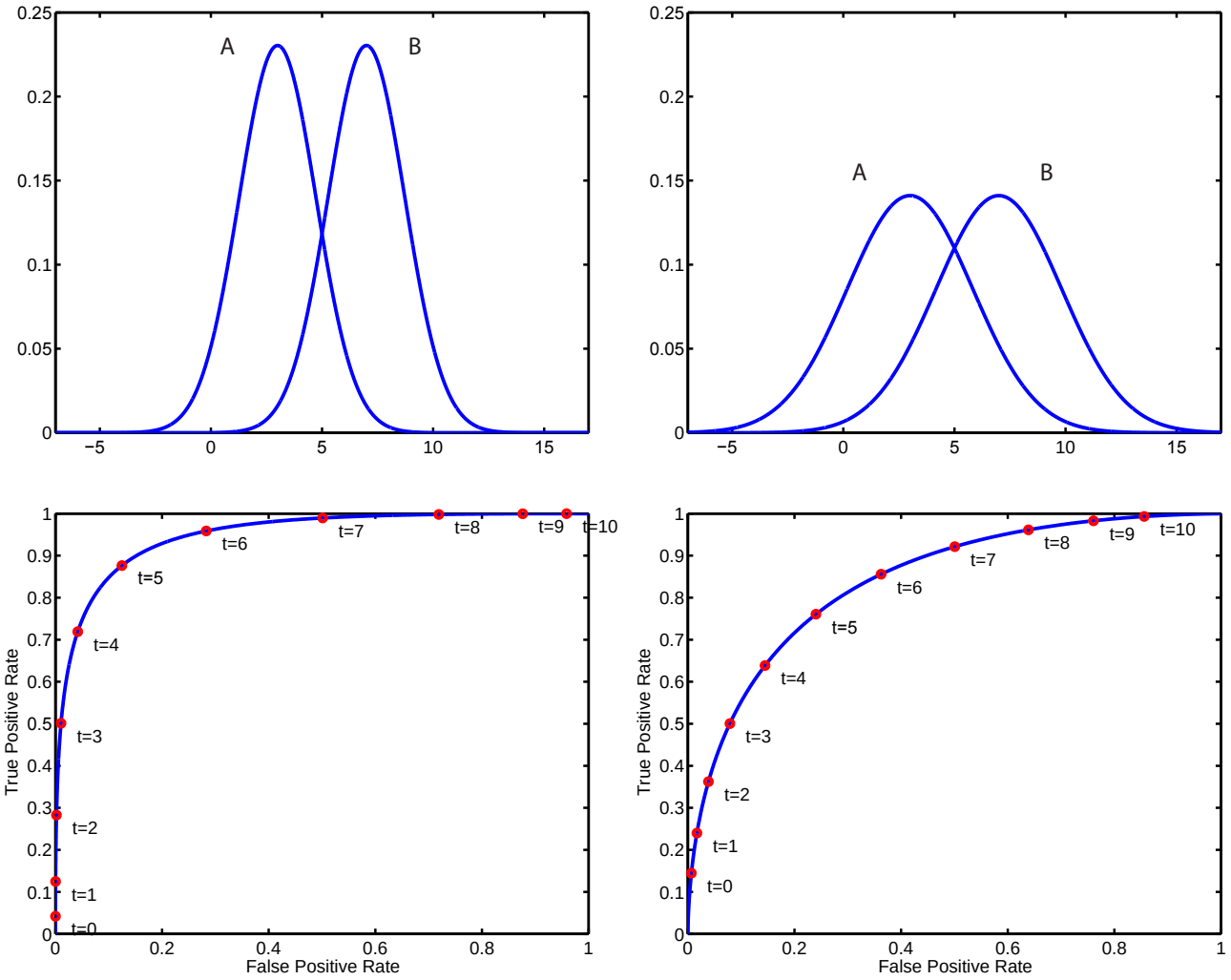


Figure B.10: The parameter curves for different classification thresholds τ (denoted t in the plots) are plotted, i.e. the ROC curves. The two sets of pdf are illustrated at the top, and the corresponding ROC curves bellow them. The hypothesis is $H0 :$ x belongs to A. It is seen that to the left the two pdfs are better separated than to the right, and as such the area under the ROC curve is larger to the left. Thus capturing that a better classification is possible in the left case due.

data is μ, and the likelihood of the model, $L(\mu)$, given the data, is

$$\begin{aligned}
L(\mu) &= \prod_{i=1}^{n} \left(x_i \in N(\mu, \sigma^2)\right) && \text{by (B.3)} \\
&= \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp^{-\frac{1}{2}\left(\frac{x_i-\mu}{\sigma}\right)^2} && \text{by (B.27)} \\
&\Rightarrow \\
\log\left(L(\mu)\right) &= \log\left(\prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp^{-\frac{1}{2}\left(\frac{x_i-\mu}{\sigma}\right)^2}\right) \\
&= \sum_{i=1}^{n} \log\left(\frac{1}{\sqrt{2\pi\sigma^2}} \exp^{-\frac{1}{2}\left(\frac{x_i-\mu}{\sigma}\right)^2}\right) \\
&= \sum_{i=1}^{n} \left(\log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) - \frac{1}{2}\left(\frac{x_i-\mu}{\sigma}\right)^2\right) \\
&= \sum_{i=1}^{n} \left(\log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)\right) - \frac{1}{2}\sum_{i=1}^{n}\left(\frac{x_i-\mu}{\sigma}\right)^2 \\
&= \sum_{i=1}^{n} \left(\log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)\right) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i-\mu)^2 \quad . && \text{(B.38)}
\end{aligned}$$

Where $n = 10$. Using the maximum likelihood method we want to find the μ that optimizes $L(\mu)$. Since $\log(\cdot)$ is an increasing function, i.e.

$$\log(a) > \log(b) \quad \Leftrightarrow \quad a > b \ .$$

Maximizing $\log(L(\mu))$ is equivalent to maximizing $L(\mu)$. As in this example, taking the logarithm often used to simplify the relevant expressions, and referred to as *log likelihood*. From (B.38) it is seen that in this example

$$\max_{\mu} L(\mu) \quad \Leftrightarrow \quad \min_{\mu} \sum_{i=1}^{n} (x_i - \mu)^2 \ ,$$

since in the last line of (B.38) it is only the rightmost term that contains μ, and that term has to be as small as possible to maximize $L(\mu)$, since there is a minus in front of it. Two things should be noted about this solution, firstly that the solution – obtained by differentiating and setting equal to zero – is

$$\begin{aligned} \frac{\partial}{\partial \mu} \sum_{i=1}^{n} (x_i - \mu)^2 &= -2 \cdot \sum_{i=1}^{n} (x_i - \mu) = 0 \qquad \Rightarrow \\ \sum_{i=1}^{n} x_i &= \sum_{i=1}^{n} \mu = n \cdot \mu \qquad \Rightarrow \\ \mu &= \frac{1}{n} \sum_{i=1}^{n} x_i \ . \end{aligned}$$

I.e. the *maximum likelihood estimate* (ML estimate) of μ is the mean as expected. This is nice, since another result would invalidate the method. The second thing to note is, that with normal distributed noise maximizing the likelihood is equivalent to minimizing the 2-norm, i.e. the normal distribution *induces* the two norm. This implies that given normal distributed noise, minimizing the Euclidean distance between the model and the data is equivalent to to finding the maximum likelihood. All other noise distributions induce norms as well, but no other well known distribution induces the 2-norm.

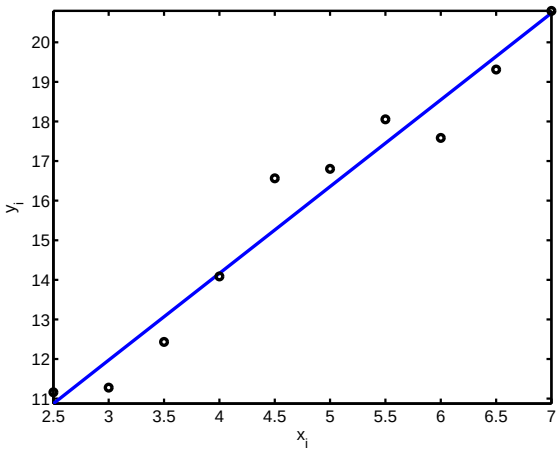


Figure B.11: Points given by $y_i = \alpha x_i + \beta + \varepsilon$ and the least squares fit, corresponding to a ML estimate if $\varepsilon \in N(0, \sigma^2)$

As a slight more elaborate example, consider fitting a line $y = \alpha x + \beta$ to some data points, i.e. our model is

$$\begin{aligned} y_i &= \alpha x_i + \beta + \varepsilon \qquad \text{with} \qquad \varepsilon \in N(0, \sigma^2) \qquad \Rightarrow \\ y_i &\in N(\alpha x_i + \beta, \sigma^2) \ . \end{aligned}$$

So now our model parameters to optimize over is α and β. Repeating the calculations from (B.38), we get

$$\log\left(L(\alpha, \beta)\right) = \sum_{i=1}^{n} \left(\log\left(\frac{1}{\sqrt{2\pi\sigma^2}} \right) \right) - \frac{1}{2\sigma^2} \sum_{i=1}^{n} (y_i - \alpha x_i - \beta)^2 \ .$$

Implying that

$$\max_{\alpha,\beta} L(\alpha,\beta) \quad \Leftrightarrow \quad \min_{\alpha,\beta} \sum_{i=1}^{n} (y_i - \alpha x_i - \beta)^2 \quad .$$

Which is seen to be a least squares problem solved via the *normal equations*, cf. Appendix A.2. This is also termed *regression*. As a numerical example consider the paired observations, also seen in Figure B.11

x_i	2.5	3.0	3.5	4.0	4.5	5.0	5.5	6.0	6.5	7.0
y_i	11.160	11.276	12.432	14.086	16.565	16.804	18.055	17.585	19.313	20.795

Here the ML estimate is given by $\hat{\alpha} = 2.1925$ and $\hat{\beta} = 5.3925$.

In is noted that in the derivation of the line fitting or regression above it is assumed that the y_i are noisy functions of x_i. This in turn implies that the x_i are noise free. This is often a good model, but not always. If noise on both the x_i and y_i is to be incorporated *orthogonal regression* should be performed. This corresponds to finding the line going through the mean in the direction of the first principal component, cf. Section B.9.2 and Figure B.13.

The result that ML estimates, in the face of normal distributed noise, induces the two-norm is another reason for the normal distribution's popularity – sometimes a reasonable noise model is just chosen. In image analysis we also often want a noise model consistent with the Euclidian distance or two norm, because this is a natural way to measure errors, e.g. deviations between image coordinates. Secondly, the induction of the two norm implies that we can use straightforward linear algebra methods to solve the estimation problems, which also has a lot of practical merit.

B.8.1 Comparing Histograms

Another frequently occurring statistic in computer vision is the histogram, as e.g. seen in Figure B.2, as such there is also a need for doing testing on and comparing of these histograms. To formalize, let a histogram of m observations with n bins be represented by the n-dimensional vector $\mathbf{x}$. Each element of $\mathbf{x}$, i.e. $\mathbf{x}_i$, contains the number of observations in bin i, and it e.g. follows directly that

$$\sum_{i=1}^{n} \mathbf{x}_i = m \quad .$$

Such an $\mathbf{x}$ describing a histogram is an element of a *polynomial distribution*. From here it follows, that if we want to test if this histogram is a realization of a discrete pdf with n bins, and the probability for an observation in bin i is given by p_i, then a good test size is given by

$$Z = \sum_{i=1}^{n} \frac{(\mathbf{x}_i - m \cdot p_i)^2}{m \cdot p_i} \quad . \tag{B.39}$$

Which is asymptotically $\chi^2(n-1)$ distributed. In computer vision we also often want to compare histograms, and e.g. find out which are most similar, implying a need for a metric or distance measure between histograms. A metric deduced from (B.39) between two histograms $\mathbf{x}$ and $\mathbf{y}$ is

$$\text{dist}(\mathbf{x},\mathbf{y}) = \sum_{i=1}^{n} \frac{(\mathbf{x}_i - \mathbf{y}_i)^2}{\frac{1}{2}(\mathbf{x}_i + \mathbf{y}_i)} \quad . \tag{B.40}$$

Where it is noted that $\frac{1}{2}(\mathbf{x}_i + \mathbf{y}_i)$ is the best estimate we have of the unknown $m \cdot p_i$. An interpretation of (B.40), is that it is the usual Euclidean distance between two vectors, but normalized by the average size of the elements. Thus a deviation of e.g. one between a pair of bins, $\mathbf{x}_i - \mathbf{y}_i$, is more grave if the average bin size $\frac{1}{2}(\mathbf{x}_i + \mathbf{y}_i)$ is five as compared to it being a hundred, which seams reasonable.

B.9 Multivariate Normal Distribution*

In the above the statistical methods introduced have mostly been on scalar measurements. Often, and especially in computer vision, measurements are multidimensional, and methods are needed to cope with this. I.e. instead

of observing scalars x_i we observe vectors $\mathbf{x}_i$. As indicated above, the normal distribution is very popular. As such this is the one which will be extended to the multivariate case here.

Starting with two introductory examples, based on $\mathbf{x}_i = \mathbf{A}\mathbf{z}_i$, where the elements of $\mathbf{z}_i$ are independent normal distributed variables with mean zero and variance one, i.e.

$$\mathbf{z}_{ij} \in N(0,1) \ ,$$

where $\mathbf{z}_{ij}$ is the j^{th} element of $\mathbf{z}_i$. In these introductory examples $j \in \{1,2\}$. Consider

$$\mathbf{x}_i = \mathbf{A}\mathbf{z}_i = \begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix} \mathbf{z}_i = \begin{bmatrix} 3 \cdot \mathbf{z}_{i1} \\ 2 \cdot \mathbf{z}_{i2} \end{bmatrix} \ ,$$

and by (B.29) and (B.25)

$$\begin{aligned} \sigma_{x1}^2 &= \text{var}(\mathbf{x}_{i1}) = 3^2 \cdot \text{var}(\mathbf{z}_{i1}) = 9 \\ \sigma_{x2}^2 &= \text{var}(\mathbf{x}_{i2}) = 2^2 \cdot \text{var}(\mathbf{z}_{i2}) = 4 \\ \sigma_{x1x2} &= \text{cov}(\mathbf{x}_{i1}, \mathbf{x}_{i2}) = 0 \end{aligned}$$

I.e. $\mathbf{x}_i$ is in this case a vector of independent normal distributed variables, formed by multiplying $N(0,1)$ distributed variables with a scalar. Consider instead,

$$\mathbf{x}_i = \mathbf{A}\mathbf{z}_i = \begin{bmatrix} 3 & 1 \\ -1 & 2 \end{bmatrix} \mathbf{z}_i = \begin{bmatrix} 3 \cdot \mathbf{z}_{i1} + 1 \cdot \mathbf{z}_{i2} \\ -1 \cdot \mathbf{z}_{i1} + 2 \cdot \mathbf{z}_{i2} \end{bmatrix} \ ,$$

and again by (B.29) and (B.25)

$$\begin{aligned} \sigma_{x1}^2 &= \text{var}(\mathbf{x}_{i1}) = 3^2 \cdot \text{var}(\mathbf{z}_{i1}) + 1^2 \cdot \text{var}(\mathbf{z}_{i2}) = 10 \\ \sigma_{x2}^2 &= \text{var}(\mathbf{x}_{i2}) = (-1)^2 \cdot \text{var}(\mathbf{z}_{i1}) + 2^2 \cdot \text{var}(\mathbf{z}_{i2}) = 5 \\ \sigma_{x1x2} &= \text{cov}(\mathbf{x}_{i1}, \mathbf{x}_{i2}) = 3 \cdot (-1) \cdot \text{var}(\mathbf{z}_{i1}) + 1 \cdot 2 \cdot \text{var}(\mathbf{z}_{i2}) = -1 \end{aligned}$$

Now $\mathbf{x}_i$ is a non trivial linear combination of the $\mathbf{z}_i$, and as such the elements of $\mathbf{x}_i$ are no longer independent, i.e. the correlation is non zero. In cases such as this, we can combine the variance structure into a matrix

$$\mathbf{\Sigma} = \begin{bmatrix} \sigma_{x1}^2 & \sigma_{x1x2} \\ \sigma_{x1x2} & \sigma_{x2}^2 \end{bmatrix} = \begin{bmatrix} 10 & -1 \\ -1 & 5 \end{bmatrix} \ .$$

Where it is noted that as a direct consequence of (B.25)

$$\mathbf{\Sigma} = \mathbf{A}\mathbf{A}^{\mathbf{T}} = \begin{bmatrix} 3 & 1 \\ -1 & 2 \end{bmatrix} \begin{bmatrix} 3 & -1 \\ 1 & 2 \end{bmatrix} = \begin{bmatrix} 10 & -1 \\ -1 & 5 \end{bmatrix} \ .$$

So Σ is symmetric and positive (semi)definite. Furthermore $\mathbf{x}_i$ is a multivariate normal distribution with mean $\mathbf{0}$ and variance $\mathbf{\Sigma}$, written as

$$\mathbf{x}_i \in N\left(\mathbf{0}, \mathbf{\Sigma}\right) \ .$$

The above example generalize to the definition of a multivariate normal distribution, by allowing for other dimensions than two and by adding a vector, μ, to $\mathbf{x}_i$ such that the mean becomes μ instead of $\mathbf{0}$, cf. the α in (B.29). The *definition of a multivariate normal distribution* is thus; given an n dimensional vector $\mathbf{z}_i$ of independent $N(0,1)$ distributed variables, an n-dimensional vector μ and an $n \times n$ matrix $\mathbf{A}$, then

$$\mathbf{x}_i = \mathbf{A}\mathbf{z}_i + \mu \ , \tag{B.41}$$

is normal distributed with variance $\mathbf{\Sigma} = \mathbf{A}\mathbf{A}^{\mathbf{T}}$ and mean μ, denoted as

$$\mathbf{x}_i \in N\left(\mu, \mathbf{\Sigma}\right) \ . \tag{B.42}$$

The elements of $\mathbf{\Sigma}$ correspond to the covariance (and variance) between the elements of $\mathbf{x}_i$, such element j,k of $\mathbf{\Sigma}$ is equal to the covariance between $\mathbf{x}_{ij}$ and $\mathbf{x}_{ik}$, i.e.

$$\mathbf{\Sigma} = \begin{bmatrix} \sigma_{x1}^2 & \sigma_{x1x2} & \cdots & \sigma_{x1xn} \\ \sigma_{x1x2} & \sigma_{x2}^2 & \cdots & \sigma_{x2xn} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{x1xn} & \sigma_{x2xn} & \cdots & \sigma_{xn}^2 \end{bmatrix} \ . \tag{B.43}$$

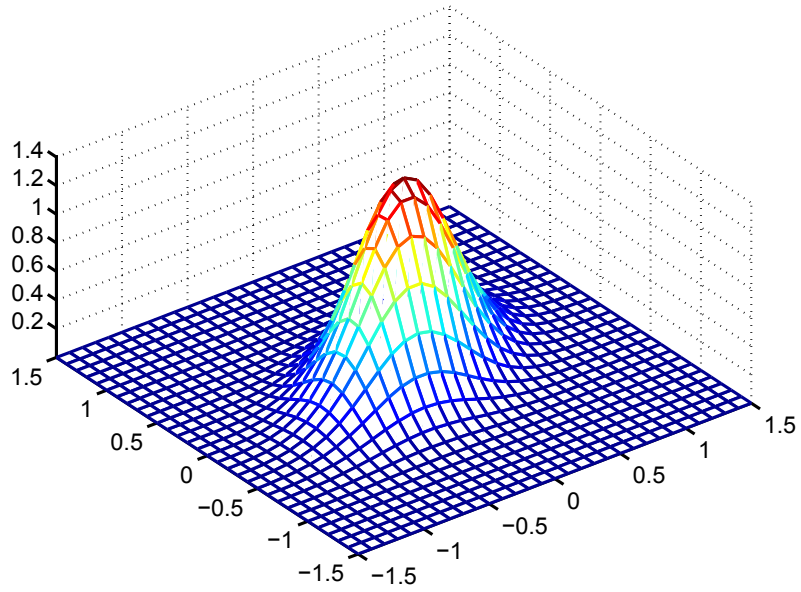


Figure B.12: Sample pdf for a 2 dimensional multivariate normal distribution.

Also the elements of μ are the means of the elements of $\mathbf{x}_i$, i.e. the mean of $\mathbf{x}_{ij}$ is μ_j.

As with scalar distributions multivariate distributions also have probability distribution functions or pdfs, in the case of the n-dimensional multivariate normal distribution this is

$$pdf(\mathbf{x}) = \frac{1}{(2\pi)^{n/2}\sqrt{\det(\mathbf{\Sigma})}} \exp\left\{-\frac{1}{2}(\mathbf{x}-\mu)^T \mathbf{\Sigma}^{-1}(\mathbf{x}-\mu)\right\} \tag{B.44}$$

Note that in the case of $n = 1$ and $\mathbf{\Sigma} = \sigma$ this reduces to (B.27). A sample distribution is illustrated in Figure B.12.

The link with the univariate or scalar case implies that estimates of mean, μ, and variance, $\mathbf{\Sigma}$, in the multivariate case follows directly from the estimates in the univariate. I.e. the estimate of the mean is

$$\hat{\mu} = \frac{1}{m}\sum_{i=1}^{m} \mathbf{x}_i \quad . \tag{B.45}$$

Given m observations. This is the vector version of estimating each element of μ individually. As for estimating the variance, the vector variant of estimating each element, $\mathbf{\Sigma}_{jk} = \sigma^2_{xjxk}$, of $\mathbf{\Sigma}$ independently is

$$\hat{\mathbf{\Sigma}} = \frac{1}{m-1}\sum_{i=1}^{m} (\mathbf{x}_i - \hat{\mu})(\mathbf{x}_i - \hat{\mu})^T \quad . \tag{B.46}$$

This is seen to be the normed sum of the outer product between $(\mathbf{x}_i - \hat{\mu})$ and it self. This way it is seen that

$$\hat{\mathbf{\Sigma}}_{jk} = \frac{1}{n-1}\sum_{i=1}^{m} (\mathbf{x}_{ij} - \hat{\mu}_j)(\mathbf{x}_{ik} - \hat{\mu}_k) \quad ,$$

as it should be. As this illustrates and as mentioned above, there is a strong link between the normal distribution and linear algebra, for more on this refer to [Anderson, 1984, Eaton, 1983], the latter reference makes a strong link between multivariate normal distributions and functional analysis.

To exemplify, reconsider the height and weight example of Figure B.3, where the observations are elements in a two dimensional normal distribution. To estimate μ and $\mathbf{\Sigma}$, we could just plug in the results from

Section B.3.3, but here (B.45) and (B.46) will be used to get the same result, i.e.

$$\begin{aligned}\hat{\mu} &= \frac{1}{m}\sum_{i=1}^{m}\mathbf{x}_i = \frac{1}{20}\begin{bmatrix}3535\\1420\end{bmatrix} = \begin{bmatrix}176.75\\71\end{bmatrix}\\ \hat{\mathbf{\Sigma}} &= \frac{1}{n-1}\sum_{i=1}^{m}\left(\mathbf{x}_i-\hat{\mu}\right)\left(\mathbf{x}_i-\hat{\mu}\right)^T = \frac{1}{19}\begin{bmatrix}1515.8 & 2757\\ 2757 & 5754\end{bmatrix} = \begin{bmatrix}79.78 & 145.11\\ 145.11 & 302.84\end{bmatrix} .\end{aligned}$$

Where x in Section B.3.3 is the first coordinate of $\mathbf{x}$ here, and y the second. Illustrating that the multivariate normal distribution is a conceptual and notational convenient way of handling a system of univariate normal distributions – typically dependent.

B.9.1 ML-Estimate and the Mahalanobis Distance

Consider using the ML method for finding the mean, μ, of a multivariate normal distribution, assuming that the variance, $\mathbf{\Sigma}$, is given. The likelihood function then is given by

$$\begin{aligned}L(\mu) &= \prod_{i=1}^{m}\frac{1}{(2\pi)^{n/2}\sqrt{\det\left(\mathbf{\Sigma}\right)}}\exp\left\{-\frac{1}{2}\left(\mathbf{x}_i-\mu\right)^T\mathbf{\Sigma}^{-1}\left(\mathbf{x}_i-\mu\right)\right\}\\ &\Rightarrow\\ \log\left(L(\mu)\right) &= \sum_{i=1}^{m}\log\left(\frac{1}{(2\pi)^{n/2}\sqrt{\det\left(\mathbf{\Sigma}\right)}}\right) - \frac{1}{2}\sum_{i=1}^{m}\left\{\left(\mathbf{x}_i-\mu\right)^T\mathbf{\Sigma}^{-1}\left(\mathbf{x}_i-\mu\right)\right\} ,\end{aligned}$$

leading to

$$\max_{\mu} L(\mu) \quad\Leftrightarrow\quad \min_{\mu}\sum_{i=1}^{m}\left\{\left(\mathbf{x}_i-\mu\right)^T\mathbf{\Sigma}^{-1}\left(\mathbf{x}_i-\mu\right)\right\} , \tag{B.47}$$

implying that the norm induced by the multivariate normal distribution is the weighted Euclidian or 2-norm, i.e.

$$||\mathbf{x}||_{\mathbf{\Sigma}^{-1}} = \mathbf{x}^T\mathbf{\Sigma}^{-1}\mathbf{x} . \tag{B.48}$$

It is noted, that the solution to (B.47) is the mean, found by differentiating and setting equal to zero, and thus consistent, i.e.

$$\begin{aligned}0 &= \frac{\partial}{\partial\mu}\sum_{i=1}^{m}\left\{\left(\mathbf{x}_i-\mu\right)^T\mathbf{\Sigma}^{-1}\left(\mathbf{x}_i-\mu\right)\right\}\\ &= \frac{\partial}{\partial\mu}\sum_{i=1}^{m}\left\{\mathbf{x}_i^T\mathbf{\Sigma}^{-1}\mathbf{x}_i - 2\mu^T\mathbf{\Sigma}^{-1}\mathbf{x}_i + \mu^T\mathbf{\Sigma}^{-1}\mu\right\}\\ &= \sum_{i=1}^{m}\left\{2\mathbf{\Sigma}^{-1}\mathbf{x}_i - 2\mathbf{\Sigma}^{-1}\mu\right\}\\ &\Rightarrow\\ 2\mathbf{\Sigma}^{-1}\sum_{i=1}^{m}\mathbf{x}_i &= 2\mathbf{\Sigma}^{-1}\sum_{i=1}^{m}\mu = 2\mathbf{\Sigma}^{-1}n\mu\\ &\Rightarrow\\ \mu &= \frac{1}{n}\sum_{i=1}^{m}\mathbf{x}_i .\end{aligned}$$

The distance in (B.48) is called the *Mahalanobis distance*, and is particularly useful when doing classification between classes described by multivariate normal distributions, with the same variance structure $\mathbf{\Sigma}$. This is also called *discriminant analysis*. As an example consider two classes A and B, described by multivariate normal distributions with means μ_A and μ_B respectively, and both with variance $\mathbf{\Sigma}$, and an observation $\mathbf{x}_i$. Consider the hypothesis, $H0$, that $\mathbf{x}_i$ belongs to A, then

$$H0:\ \mathbf{x}_i \in N\left(\mu_A,\mathbf{\Sigma}\right) \quad \text{alternatively} \quad H1:\ \mathbf{x}_i \in N\left(\mu_B,\mathbf{\Sigma}\right) .$$

If we wish to accept $H0$ if the probability of $H0$ is larger than the probability of $H1$, this corresponds to

$$\begin{aligned} pdf_A(\mathbf{x}_i) &\geq pdf_B(\mathbf{x}_i) \\ Z\exp\left\{-\frac{1}{2}\left(\mathbf{x}_i-\mu_A\right)^T\boldsymbol{\Sigma}^{-1}\left(\mathbf{x}_i-\mu_A\right)\right\} &\geq Z\exp\left\{-\frac{1}{2}\left(\mathbf{x}_i-\mu_B\right)^T\boldsymbol{\Sigma}^{-1}\left(\mathbf{x}_i-\mu_B\right)\right\} \\ 2\log(Z)+\left(\mathbf{x}_i-\mu_A\right)^T\boldsymbol{\Sigma}^{-1}\left(\mathbf{x}_i-\mu_A\right) &\leq 2\log(Z)+\left(\mathbf{x}_i-\mu_B\right)^T\boldsymbol{\Sigma}^{-1}\left(\mathbf{x}_i-\mu_B\right) \\ ||\mathbf{x}_i-\mu_A||_{\boldsymbol{\Sigma}^{-1}} &\leq ||\mathbf{x}_i-\mu_B||_{\boldsymbol{\Sigma}^{-1}} \\ \mathbf{x}_i^T\boldsymbol{\Sigma}^{-1}\mathbf{x}_i-2\mathbf{x}_i^T\boldsymbol{\Sigma}^{-1}\mu_A+\mu_A^T\boldsymbol{\Sigma}^{-1}\mu_A &\leq \mathbf{x}_i^T\boldsymbol{\Sigma}^{-1}\mathbf{x}_i-2\mathbf{x}_i^T\boldsymbol{\Sigma}^{-1}\mu_B+\mu_B^T\boldsymbol{\Sigma}^{-1}\mu_B \\ \mathbf{x}_i^T\boldsymbol{\Sigma}^{-1}\left(\mu_B-\mu_A\right) &\geq \frac{1}{2}\mu_A^T\boldsymbol{\Sigma}^{-1}\mu_A-\frac{1}{2}\mu_B^T\boldsymbol{\Sigma}^{-1}\mu_B \ . \end{aligned} \tag{B.49}$$

Where

$$Z=\frac{1}{(2\pi)^{n/2}\sqrt{\det\left(\boldsymbol{\Sigma}\right)}} \ .$$

Here it is seen that

$$\boldsymbol{\Sigma}^{-1}\left(\mu_B-\mu_A\right) \ ,$$

is a vector with the same dimension as $\mathbf{x}$ and

$$\frac{1}{2}\mu_A^T\boldsymbol{\Sigma}^{-1}\mu_A-\frac{1}{2}\mu_B^T\boldsymbol{\Sigma}^{-1}\mu_B \ .$$

Is a scalar. So the separating hyperplane, corresponding to the threshold τ in the 1D case, is given by

$$\mathbf{x}_i^T\boldsymbol{\Sigma}^{-1}\left(\mu_B-\mu_A\right)=\frac{1}{2}\mu_A^T\boldsymbol{\Sigma}^{-1}\mu_A-\frac{1}{2}\mu_B^T\boldsymbol{\Sigma}^{-1}\mu_B \ . \tag{B.50}$$

B.9.2 Principal Component Analysis

An analysis frequently made on normally distributed multivariate variables is that of an eigenvalue or singular value decomposition of the variance, cf. Appendix A.6. That is, decomposing the symmetric positive (semi) definite matrix $\boldsymbol{\Sigma}$ into

$$\boldsymbol{\Sigma}=\mathbf{R}\mathbf{D}\mathbf{R}^T \ ,$$

where $\mathbf{R}$ is a rotation matrix, made up of the eigenvectors of $\boldsymbol{\Sigma}$, and $\mathbf{D}$ is a diagonal matrix of the eigenvalues or singular values of $\boldsymbol{\Sigma}$, i.e.

$$\mathbf{D}=\begin{bmatrix} \delta_1 & 0 & \cdots & 0 \\ 0 & \delta_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \delta_n \end{bmatrix} \ .$$

Where the singular values δ_i are assumed ordered such that $\delta_1 \geq \delta_2 \geq \cdots \geq \delta_n$. Furthermore, let $\sqrt{\mathbf{D}}$ be the diagonal matrix of the square root of the singular values, i.e.

$$\sqrt{\mathbf{D}}=\begin{bmatrix} \sqrt{\delta_1} & 0 & \cdots & 0 \\ 0 & \sqrt{\delta_2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sqrt{\delta_n} \end{bmatrix} \ .$$

Leading to $\boldsymbol{\Sigma}$ being decomposed into

$$\boldsymbol{\Sigma}=\mathbf{R}\sqrt{\mathbf{D}}\sqrt{\mathbf{D}}^T\mathbf{R}^T=\mathbf{R}\sqrt{\mathbf{D}}\left(\mathbf{R}\sqrt{\mathbf{D}}\right)^T \ ,$$

which in relation to the definition of the multivariate normal distribution, where $\mathbf{\Sigma}$ is defined by $\mathbf{\Sigma} = \mathbf{A}\mathbf{A}^T$, implies that an $\mathbf{A}$ can be written as[10] $\mathbf{A} = \mathbf{R}\sqrt{\mathbf{D}}$. Inserting this into (B.41), yields[11]

$$\begin{aligned} \mathbf{x}_i &= \mathbf{A}\mathbf{z}_i + \mu \\ \mathbf{x}_i &= \mathbf{R}\sqrt{\mathbf{D}}\mathbf{z}_i + \mu \\ &\Leftrightarrow \\ \mathbf{R}^\mathbf{T}\mathbf{x}_i &= \sqrt{\mathbf{D}}\mathbf{z}_i + \mathbf{R}^\mathbf{T}\mu \end{aligned}$$

Thus, if the basis of $\mathbf{x}_i$ is changed to $\mathbf{R}^T$, i.e. by rotating $\mathbf{x}_i$ by $\mathbf{R}^T$, forming $\tilde{\mathbf{x}}_i$, would give the multivariate normal distribution

$$\tilde{\mathbf{x}}_i \in N\left(\mathbf{R}^T\mu, \mathbf{D}\right) \qquad \text{with} \qquad \tilde{\mathbf{x}}_i = \mathbf{R}^\mathbf{T}\mathbf{x}_i \ . \tag{B.51}$$

It is thus possible to find a rotation of, or new basis for, a set of multivariate normal distributed variables, such that the variance is given by an ordered diagonal matrix, $\mathbf{D}$. This is achieved by an eigenvalue or singular value decomposition of the variance $\mathbf{\Sigma}$. This analysis or decomposition is known as a *principal component analysis*. A reasonable question is then; why this is useful?

To answer this, note that due to the ordered nature of $\mathbf{D}$ the variance the first element of $\tilde{\mathbf{x}}_i$, i.e. $\tilde{\mathbf{x}}_{i1}$, is δ_1 and that no other element of $\tilde{\mathbf{x}}_i$ has a higher variance. It can in fact, by a slight extension of the above argument, be shown that the first column of $\mathbf{R}$, is the linear combination of the elements of $\mathbf{x}_i$, with unit length, that has the highest variance. Denoting this first column by $\mathbf{r}_{:1}$ it is seen that

$$\tilde{\mathbf{x}}_{i1} = \mathbf{r}_{:1}^T\mathbf{x}_i \ .$$

This extends such that the second column of $\mathbf{R}$, denoted $\mathbf{r}_{:2}$, is the linear combination of the elements of $\mathbf{x}_i$, with unit length *and* orthogonal to $\mathbf{r}_{:1}$, which has the highest variance. Thus if we want the k-dimensional subspace that captures the most of the variance of $\mathbf{x}_i$ we should choose the k first columns of $\mathbf{R}$. This can e.g. be useful for visualization purposes. The columns of $\mathbf{R}$ are called principal component loadings, so e.g. $\mathbf{r}_{:1}$ is called the first principle component loading etc.

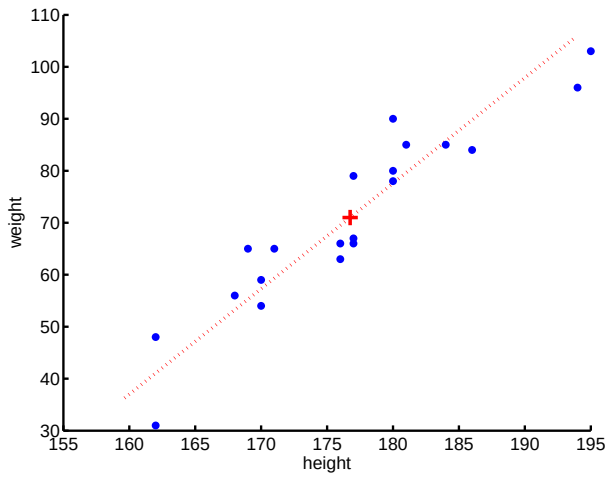


Figure B.13: The data from Figure B.3, with the first principal component denoted by the dotted red line and the mean by the red cross. This dotted red line corresponds to doing orthogonal regression of the data.

As an example consider the height weight example from Figure B.3, where it has been estimated that

$$\begin{aligned} \hat{\mathbf{\Sigma}} &= \begin{bmatrix} 79.78 & 145.11 \\ 145.11 & 302.84 \end{bmatrix} \\ &= \begin{bmatrix} 0.4419 & -0.8971 \\ 0.8971 & 0.4419 \end{bmatrix} \begin{bmatrix} 374.3259 & 0 \\ 0 & 8.2925 \end{bmatrix} \begin{bmatrix} 0.4419 & -0.8971 \\ 0.8971 & 0.4419 \end{bmatrix}^T \\ &= \mathbf{R}\mathbf{D}\mathbf{R}^T \ . \end{aligned}$$

[10] This $\mathbf{A}$ is not unique. This is seen by the taking the SVD of $\mathbf{A} = \mathbf{R}\sqrt{\mathbf{D}}\mathbf{R_2}$, where $\mathbf{R_2}$ is unobservable, in that $\mathbf{A}\mathbf{A}^\mathbf{T} = \mathbf{R}\sqrt{\mathbf{D}}\mathbf{R_2}\mathbf{R_2^T}\sqrt{\mathbf{D}}^\mathbf{T}\mathbf{R}^\mathbf{T} = \mathbf{R}\sqrt{\mathbf{D}}\sqrt{\mathbf{D}}^\mathbf{T}\mathbf{R}^\mathbf{T}$ since $\mathbf{R_2}\mathbf{R_2^T} = \mathbf{I}$.

[11] Noting that the inverse of a rotation matrix $\mathbf{R}$ is its transposed, i.e. $\mathbf{R}^{-1} = \mathbf{R}^\mathbf{T}$.

Indicating that $[0.4419, 0.8971]^T$ is the unit length linear combination that expresses most of the variation of the height weight measurements. This is illustrated in Figure B.13. From this figure it is also seen that the principal components can also be useful for interpreting the data, in that the first principal component can here be seen as an expression of size, which seams like a good simplified interpretation of what this data captures.

Bibliography

[Aanæs et al., 2012] Aanæs, H., Dahl, A., and Steenstrup Pedersen, K. (2012). Interesting interest points. *International Journal of Computer Vision*, pages 1–18.

[Aanaes et al., 2008] Aanaes, H., Sveinsson, J., Benediktsson, J., Nielsen, A., and Boevith, T. (2008). Model-based satellite image fusion. *IEEE Transactions on Geoscience and Remote Sensing*, 46(5):1336–1346.

[Akenine-Möller and Haines, 2002] Akenine-Möller, T. and Haines, E. (2002). *Real-Time Rendering (2nd Edition)*. AK Peters, Ltd.

[Anderson, 1984] Anderson, T. (1984). *An Introduction to Multivariate Statistical Analysis – 2nd Edition*. John Wiley and Sons, New York.

[Ansar and Daniilidis, 2003] Ansar, A. and Daniilidis, K. (2003). Linear pose estimation from points or lines. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(5):578–589.

[Arya et al., 1998] Arya, S., Mount, D., Netanyahu, N., Silverman, R., and Wu, A. (1998). An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM (JACM)*, 45(6):891–923.

[Baeza-Yates and Ribeiro-Neto, 1999] Baeza-Yates, R. and Ribeiro-Neto, B. (1999). *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[Batlle et al., 1998] Batlle, J., Mouaddib, E., and Salvi, J. (1998). Recent progress in coded structured light as a technique to solve the correspondence problem: a survey. *Pattern Recognition*, 31(7):963–982.

[Besag, 1974] Besag, J. (1974). Spatial interaction and the statistical analysis of lattice systems. *Journal of the Royal Statistical Society. Series B (Methodological)*, 36(2):192–236.

[Bishop, 2007] Bishop, C. (2007). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer.

[Bishop and Goldberg, 1980] Bishop, R. and Goldberg, S. (1980). *Tensor Analysis on Manifolds*. Dover Publications.

[Björck, 1996] Björck, Å. (1996). *Numerical Methods for Least Squares Problems*. SIAM, Philadelphia.

[Black and Rangarajan, 1996] Black, M. and Rangarajan, A. (1996). On the unification of line processes, outlier rejection, and robust statistics with applications in early vision. *International Journal of Computer Vision*, 19(1):57–91.

[Boros and Hammer, 2002] Boros, E. and Hammer, P. (2002). Pseudo-boolean optimization. *Discrete Appl. Math.*, 123:155–225.

[Boyd and Vandenberghe, 2004] Boyd, S. and Vandenberghe, L. (2004). *Convex Optimization*. Cambridge University Press.

[Boykov and Kolmogorov, 2004] Boykov, Y. and Kolmogorov, V. (2004). An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26:1124–1137.

[Boykov et al., 2001] Boykov, Y., Veksler, O., and Zabih, R. (2001). Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(11):1222–1239.

[Canny, 1986] Canny, J. (1986). A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):679–698.

[Carstensen(Editor), 2002] Carstensen(Editor), J. (2002). *Image Analysis, Vision and Computer Graphics*. Tehnical University of Denmark.

[Clifford, 1990] Clifford, P. (1990). Markov random fields in statistics. In *Disorder in Physical Systems: A Volume in Honour of John M. Hammersley*, pages 19–32. Oxford University Press.

[Comaniciu and P., 2002] Comaniciu, D. and P., M. (2002). Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24:603–619.

[Conradsen, 2002] Conradsen, K. (2002). *En introduktion til statistik - bind 2, 6. udgave*. Informatics and Mathematical Modelling, Technical University of Denmark, DTU.

[Cootes et al., 1995] Cootes, T. F., Taylor, C. J., Cooper, D. H., and Graham, J. (1995). Active shape models – their training and application. *Computer Vision, Graphics and Image Processing*, 61(1):38–59.

[Cormen et al., 2001] Cormen, T., Leiserson, C., Rivest, R., and Stein, C. (2001). *Introduction to algorithms*. MIT Press, 2nd edition.

[Cox et al., 2005] Cox, D., Little, J., and O'Shea, D. (2005). *Using Algebraic Geometry*. Springer, 2nd edition.

[Cox et al., 2007] Cox, D., Little, J., and O'Shea, D. (2007). *Ideals, Varieties, and Algorithms*. Springer Verlag.

[Cox et al., 1996] Cox, I., Higorani, S., S.B.Rao, and Maggs, B. (1996). A maximum likelihood stereo algorithm. *Computer Vision and Image Understanding*, 63(3):542–67.

[Dahl et al., 2011] Dahl, A., Aanæs, H., and Pedersen, K. (2011). Finding the best feature detector-descriptor combination. In *International Conference on 3D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT), 2011*, pages 318–325.

[Deriche, 1990] Deriche, R. (1990). Fast algorithms for low-level vision. *IEEE Trans. Pattern Anal. Mach. Intell.*, 12(1):78–87.

[Duda and Hart, 1972] Duda, R. and Hart, P. (1972). Use of the hough transformation to detect lines and curves in pictures. *Commun. ACM*, 15(1):11–15.

[Eaton, 1983] Eaton, M. L. (1983). *Multivariate Statistics, A Vector Space Approach*. John Wiley & Sons, Inc.

[Eising, 1997] Eising, J. (1997). *Lineær Algebra*. Technical University of Denmark, Department of Mathematics.

[Favaro and Soatto, 2005] Favaro, P. and Soatto, S. (2005). A geometric approach to shape from defocus. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(3):406–417.

[Fischler and Bolles, 1981] Fischler, M. and Bolles, R. (1981). Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395.

[Ford and Fulkerson, 1962] Ford, L. and Fulkerson, D. (1962). *Flows in Networks*. Princeton University Press.

[Gao et al., 2003] Gao, X., Hou, X., Tang, J., and Cheng, H. (2003). Complete solution classification for the perspective-three-point problem. *PAMI*, 25(8):930–943.

[Golub and van Loan, 1996] Golub, G. and van Loan, C. (1996). *Matrix Computations*. Johns Hopkins University Press, 3rd edition.

[Graham, 1981] Graham, A. (1981). *Kronecker Products and Matrix Calculations with Applications*. John Wiley, New York.

[Greig et al., 1989] Greig, D. M., Porteous, B. T., and Seheult, A. H. (1989). Exact maximum a posteriori estimation for binary images. *Journal of the Royal Statistical Society. Series B (Methodological)*, 51(2):pp. 271–279.

[Gudmundsson, 2010] Gudmundsson, S. (2010). *Time-of-Flight Cameras in Computer Vision*. PhD thesis, University of Iceland.

[Hamilton, 1853] Hamilton, W. (1853). *Lectures on Quaternions*. Hodges and Smith & Co., Dublin.

[Hansen, 1998] Hansen, P. (1998). *Rank-Deficient and Discrete Ill-Posed Problems. Numerical Aspects of Linear Inversion*. SIAM, Philadelphia.

[Haralick et al., 1994] Haralick, R., Lee, C.-N., Ottenberg, K., and Nolle, M. (1994). Review and analysis of solutions of the three point perspective pose estimation problem. *International Journal of Computer Vision*, 13(3):331–356.

[Harris and Stephens, 1988] Harris, C. and Stephens, M. (1988). A combined corner and edge detector. In *Proc. Alvey Conf.*, pages 189–192.

[Hartley and Zisserman, 2003] Hartley, R. I. and Zisserman, A. (2003). *Multiple View Geometry – 2nd edition*. Cambridge University Press, The Edinburgh Building, Cambridge CB2 2RU, UK.

[Hastie et al., 2001] Hastie, T., Tibshirani, J., and Friedman, J. (2001). *The Elements of Statistical Learning, Data Mining, Inference and Prediction*. Springer.

[Hastie et al., 2009] Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The Elements of Statistical Learning, Second Edition: Data Mining, Inference, and Prediction*. Springer, 2nd ed. 2009. edition.

[Heikkila, 2000] Heikkila, J. (2000). Geometric camera calibration using circular control points. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(10):1066–1077.

[Heinly et al., 2012] Heinly, J., Dunn, E., and Frahm, J. (2012). Comparative Evaluation of Binary Features. In *ECCV 2012*, pages 759–773+. Springer Berlin Heidelberg.

[Horn and Schunk, 1981] Horn, B. and Schunk, B. (1981). Determining optical flow. *AI*, 17(1-3):185–203.

[Hough, 1962] Hough, P. (1962). Method and means for recognizing complex patterns. U.S. Patent 3.069.654.

[Huber, 1981] Huber, P. (1981). *Robust Statistics*. John Wiley, New York.

[Johnson, 2010] Johnson, R. (2010). *Miller & Freund's Probability and Statistics for Engineers*. Prentice Hall, 8. edition edition.

[Jonker and Volgenant, 1987] Jonker, R. and Volgenant, A. (1987). A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing*, 38:325–340.

[Kahl et al., 2008] Kahl, F., Agarwal, S., Chandraker, M., Kriegman, D., and Belongie, S. (2008). Practical global optimization for multiview geometry. *International Journal of Computer Vision*, 79(3):271–284.

[Kneip et al., 2011] Kneip, L., Scaramuzza, D., and Siegwart, R. (2011). A novel parametrization of the perspective-three-point problem for a direct computation of absolute camera position and orientation. In *CVPR '11*, pages 2969–2976.

[Kolmogorov and Zabih, 2004] Kolmogorov, V. and Zabih, R. (2004). What energy functions can be minimized via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26:65–81.

[Lempitsky et al., 2010] Lempitsky, V., Rother, C., Roth, S., and Blake, A. (2010). Fusion moves for markov random field optimization. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32(8):1392–1405.

[Levenberg, 1944] Levenberg, K. (1944). A method for the solution of certain problems in least-squares. *Quart. J. of Appl. Math.*, 12:164–168.

[Levoy et al., 2000] Levoy, M., Rusinkiewcz, S., Ginzton, M., Ginsberg, J., Pulli, K., Koller, D., Anderson, S., Shade, J., Curless, B., Pereira, L., Davis, J., and Fulk, D. (2000). The digital michelangelo project: 3d scanning of large statues. *Computer Graphics Proceedings. Annual Conference Series 2000. SIGGRAPH 2000. Conference Proceedings*, pages 131–44.

[Li and Hartley, 2006] Li, H. and Hartley, R. (2006). Five-point motion estimation made easy. ICPR '06, pages 630–633. IEEE Computer Society.

[Li, 2009] Li, S. (2009). *Markov Random Field Modeling in Image Analysis*. Springer Verlag, 3rd edition.

[Lindeberg, 1997] Lindeberg, T. (1997). On the axiomatic foundations of linear scale-space: Combining semi-group structure with causality vs. scale invariance. Technical Report S-100 44, Department of Numerical Analysis and Computing Science, Royal Institute of Technology.

[Lindeberg, 1998] Lindeberg, T. (1998). Feature detection with automatic scale selection. *International Journal of Computer Vision*, 30(2).

[Lowe, 2004] Lowe, D. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110.

[MacKay, 2002] MacKay, D. J. C. (2002). *Information Theory, Inference & Learning Algorithms*. Cambridge University Press.

[Maronna et al., 2006] Maronna, R., Martin, D., and Yohai, V. (2006). *Robust Statistics: Theory and Methods*. Wiley-Blackwell.

[Marquardt, 1963] Marquardt, D. (1963). An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial and Applied Mathematics*, 11(2):431–441.

[Matas et al., 2004] Matas, J., Chum, O., Urban, M., and Pajdla, T. (2004). Robust wide-baseline stereo from maximally stable extremal regions. *Image and Vision Computing*, 22(10):761–767.

[Meissl, 1982] Meissl, P. (1982). *Least Squares Adjustment. A Modern Approach*. Geodetic Institute of the Technical University Graz.

[Mikolajczyk and Schmid, 2005] Mikolajczyk, K. and Schmid, C. (2005). A performance evaluation of local descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(10):1615–1630.

[Muja and Lowe, 2009] Muja, M. and Lowe, D. (2009). Fast approximate nearest neighbors with automatic algorithm configuration. In *International Conference on Computer Vision Theory and Application VISSAPP'09)*, pages 331–340.

[Nister, 2004] Nister, D. (2004). An efficient solution to the five-point relative pose problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(6):756–770.

[Nister and Engels, 2006] Nister, D. and Engels, C. (2006). Estimating global uncertainty in epipolar geometry for vehicle-mounted cameras. *Proceedings of the SPIE - The International Society for Optical Engineering*, 6230(1):62301L–1–12.

[Nister and Stewenius, 2006] Nister, D. and Stewenius, H. (2006). Scalable recognition with a vocabulary tree. *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 1 (CVPR'06)*, 2:2161–2168.

[Nister and Stewenius, 2007] Nister, D. and Stewenius, H. (2007). A minimal solution to the generalised 3-point pose problem. *Journal of Mathematical Imaging and Vision*, 27(1):67–79.

[Nistér and Stewénius, 2008] Nistér, D. and Stewénius, H. (2008). Linear time maximally stable extremal regions. In *Proceedings of the 10th European Conference on Computer Vision*, ECCV '08, pages 183–196.

[Nocedal and Wright, 2000] Nocedal, J. and Wright, S. (2000). *Numerical Optimization*. Springer.

[Paige and Saunders, 1982] Paige, C. and Saunders, M. (1982). Lsqr: An algorithm for sparse linear equations and sparse least squares. *ACM Trans. Math. Software*, 8(1):43–71.

[Papadopoulo and Lourakis, 2000] Papadopoulo, T. and Lourakis, M. (2000). Estimating the jacobian of the singular value decomposition: Theory and applications. In *Research report, INRIA Sophia-Antipolis, 2000. In preparation*, pages 554–570. Springer.

[Pollefeys, 2000] Pollefeys, M. (2000). Tutorial on 3d modeling from images. *In conjunction with ECCV 2000, Dublin, Ireland*.

[Pollefeys et al., 1999] Pollefeys, M., Koch, R., and Van Gool, L. (1999). A simple and efficient rectification method for general motion. *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, 1:496–501 vol.1.

[Potts, 1952] Potts, R. (1952). Some Generalized Order-Disorder Transformation. In *Transformations, Proceedings of the Cambridge Philosophical Society*, volume 48, pages 106–109.

[Radon, 1986] Radon, J. (1986). On the determination of functions from their integral values along certain manifolds. *IEEE Transactions on Medical Imaging*, 5(4):170–176.

[Ray, 2002] Ray, S. (2002). *Applied Photographic Optics, Third Edition*. Focal Press.

[Roy and Cox, 1998] Roy, S. and Cox, I. (1998). A maximum-flow formulation of the n-camera stereo correspondence problem. *Computer Vision, 1998. Sixth International Conference on*, pages 492 –499.

[Rusinkiewicz and Levoy, 2001] Rusinkiewicz, S. and Levoy, M. (2001). Efficient variants of the icp algorithm. *3-D Digital Imaging and Modeling, 2001. Proceedings. Third International Conference on*, pages 145–152.

[Scharstein and Szeliski, 2002] Scharstein, D. and Szeliski, R. (2002). A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision*, 47(3):7–42.

[Scharstein et al., 2003] Scharstein, D., Szeliski, R., and Coll, M. (2003). High-accuracy stereo depth maps using structured light. In *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings*, volume 1.

[Sivic and Zisserman, 2003] Sivic, J. and Zisserman, A. (2003). Video Google: A text retrieval approach to object matching in videos. In *Proceedings of the International Conference on Computer Vision*, volume 2, pages 1470–1477.

[Snavely et al., 2008] Snavely, N., Seitz, S., and Szeliski, R. (2008). Modeling the world from internet photo collections. *International Journal of Computer Vision*, 80(2):189–210.

[Sporring et al., 1997] Sporring, J., Nielsen, M., Florack, L., and Johansen, P. (1997). *Gaussian Scale-Space Theory*. Kluwer Academic Publishers.

[Stewénius et al., 2006] Stewénius, H., Engels, C., and Nistér, D. (2006). Recent developments on direct relative orientation. *ISPRS Journal of Photogrammetry and Remote Sensing*, 60(4):284–294.

[Stewenius et al., 2008] Stewenius, H., Nister, D., Kahl, F., and Schaffilitzky, F. (2008). A minimal solution for relative pose with unknown focal length. *Image and Vision Computing*, 26(7):871–877.

[Tola et al., 2010] Tola, E., Lepetit, V., and Fua, P. (2010). Daisy: an efficient dense descriptor applied to wide-baseline stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(5):815–830.

[Triggs et al., 2000] Triggs, B., McLauchlan, P. F., Hartley, R. I., and Fitzgibbon, A. W. (2000). Special sessions - bundle adjustment - a modern synthesis. *Lecture Notes in Computer Science*, 1883:298–372.

[Tuytelaars and Mikolajczyk, 2008] Tuytelaars, T. and Mikolajczyk, K. (2008). Local invariant feature detectors: a survey. *Found. Trends. Comput. Graph. Vis.*, 3(3):177–280.

[van Loan, 2000] van Loan, C. (2000). The ubiquitous kronecker product. *Journal of Computational and Applied Mathematics*, 123(1-2):85–100.

[Winder and Brown, 2007] Winder, S. and Brown, M. (2007). Learning local image descriptors. *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8.

[Winkler, 2003] Winkler, G. (2003). *Image Analysis, Random Fields and Markov Chain Monte Carlo Methods: A Mathematical Introduction (Applications of Mathematics)*. Springer Verlag.

Index