Technical Report

Pattern Recognition and Image Processing Group
Institute of Computer Aided Automation
Vienna University of Technology
Favoritenstr. 9/1832
A-1040 Vienna AUSTRIA
Phone: amp;+43 (1) 58801-18351
Fax: amp;+43 (1) 58801-18392
E-mail: amp;zollnerh@prip.tuwien.ac.at
URL: amp; http://www.prip.tuwien.ac.at/

PRIP-TR-{085} August 25, 2003

A Calibration Technique for CCD Cameras using Pose Estimation

Helmut Zollner

Abstract

In this report we present a new calibration technique for off-the-shelf CCD cameras and acquisition systems. The estimated camera model includes 6 extrinsic and up to 8 intrinsic model parameters including radial and tangential distortion. Also an inverse distortion function for correction of distortion effects can be estimated. Planar and non-planar objects can be used as previously measured calibration targets. The method is suitable for single image and multiple image input datasets. Tools for Mathematica^{TM}, MATLAB^{TM}and Khoros^{TM}were developed and tested by the author at PRIP. In addition this report includes also a number of practical test results.

Contents

List of Figures

List of Tables

Preface

Structure of this work

The report is split into 5 main parts. Chapter 1 gives a brief introduction to camera calibration topic. The main motivation and the reasonable needs for calibrated cameras are explained. In Chapter 2 we give a broad overview of existing techniques and point out their benefits and drawbacks. In the following sections projection geometry of lens cameras and the mathematical representation of camera models is investigated. Chapter 3 presents the methods and algorithms of the implemented solution of which in Chapter 4 test results are shown. In Chapter 5 we draw our conclusions and give some outlook. Additionally we added 3 Appendices containing implementation details.

Contributions

- In-depth analysis of geometric and chromatic projection features of lens cameras
- Development of a new alternative approach to camera calibration by using classical pose estimation
- Implementation of the presented method in a Matlab^{TM}toolbox and application to a real world 3D acquisition system

Acknowledgements

The author would like to thank *Robert Sablatnig*, who reviewed this work, and *Martin Kampel*, who gave me a lot of inspiring tips and mental motivation to finish this work.

1 Introduction

Originally camera calibration is one of the main issues in the field of *photogrammetry*, a scientific discipline, that emerged at the time of the invention of photography long before computers and even electronics had been invented. Photogrammetry is primarily occupied with measurement by photography. Technologies for aerial and terrestrial survey were developed relying on expensive, state-of-the-art optical equipment. In order to measure characteristics of optics mechanical devices like the *goniometer* were invented ([1]). Before CCD [1] technology was available, only high quality photographic emulsions and filters were used in image acquisition. Photographic film preserves analog image information as negative image of the projection resulting from chemical reaction during light exposure. Shape and light information can be digitized by scanners and in the following be processed by computer software. This process is the main topic of *analytic photogrammetry*.
In 3D computer vision we use medium quality CCD cameras and off-the-shelf frame grabbers and video cards. Today digital CCD cameras are commonly used and quite affordable. But not every camera type will match the high opto-sensoric and mechanical standards of a photogrammetric acquisition system (e.g. web cams). These low-cost cameras are also more sensitive to environmental conditions like temperature, air pressure or mechanical exposure.
Basically one can distinguish between two different features of an existing camera-CCD-sensor system. First there are the *radiometric* properties of optics and sensor. Spectral irradiance, reflectance and response determine resolution of light intensity per channel. One can find a number of adequate radiometric calibration methods in literature (e.g. in [2]).

Secondly we have to be concerned about the geometric features of the camera lens projection. *Geometric calibration* of a camera system is needed to get an accurate approximation for the focal length and lens distortion of the optical part and for the horizontal and vertical resolution of the digital sensor element. In case we have information about the real world sizes of the sensor and the number of sensor elements in each row or column of the sensor matrix the resolution parameters are given. In most cases this data is made available by the manufacturer and so they are supposed to be known. But to maintain flexibility the proposed calibration method does not rely on that. Another important parameter is the pixel position of the point of intersection between the optical axis and the image plane — called principal point (abrev. p.p.). It is important to note that we have to distinguish the principal point from the optical center which lies also on the optical axis but within lens center.
Apart from the theoretical considerations there are also a number of practical requirements. The method should produce reliable results of a predefined and satisfying quality regardless of camera type and light source quality. Also the effort to produce a calibration object and its cost should be considered. Calibration equipment should be transportable and easily replaceable. Also a variety of different calibration targets should be possible to cover different kinds of scene spaces (it is well-known (see [3]) that, in order to obtain satisfactory and reliable calibra-

[1]CCD: Abbreviation for charge coupled device. A semiconductor image-sensing device that (a) is usually in the form of a plane rectangular matrix of microscopic, individual sensing elements, each of which corresponds to an image pixel, (b) is placed at the focal plane of an optical imaging system, and (c) converts the optical image into electrical signals which represent the information contained within each pixel, and which may be extracted sequentially and stored or otherwise processed in digital form for the purpose of transmitting or storing a digital representation of the optical image. Note: CCDs are commonly used as image sensors in professional and consumer television cameras and camcorders, and as image sensors in digital still cameras

tion results, it is necessary for the calibration object points to “fill up” the entire scene space).
This report solely deals with *geometric camera calibration*. So we do not care much about the electrical characteristics of the sensor elements or the reflectance properties of the calibration object. For our purpose only the real world ranges and sizes of the calibration target and an estimated value for the focal length of the lens system are needed. Problems due to faulty sensor elements are not considered.

2 Geometric calibration of CCD camera systems

Numerous methods were developed and applied to 3D vision acquisition systems. In this section we present a short, systematic classification of methods and required setups. Figure 2.1 gives a schematic overview. It will be explained more precisely in the following.

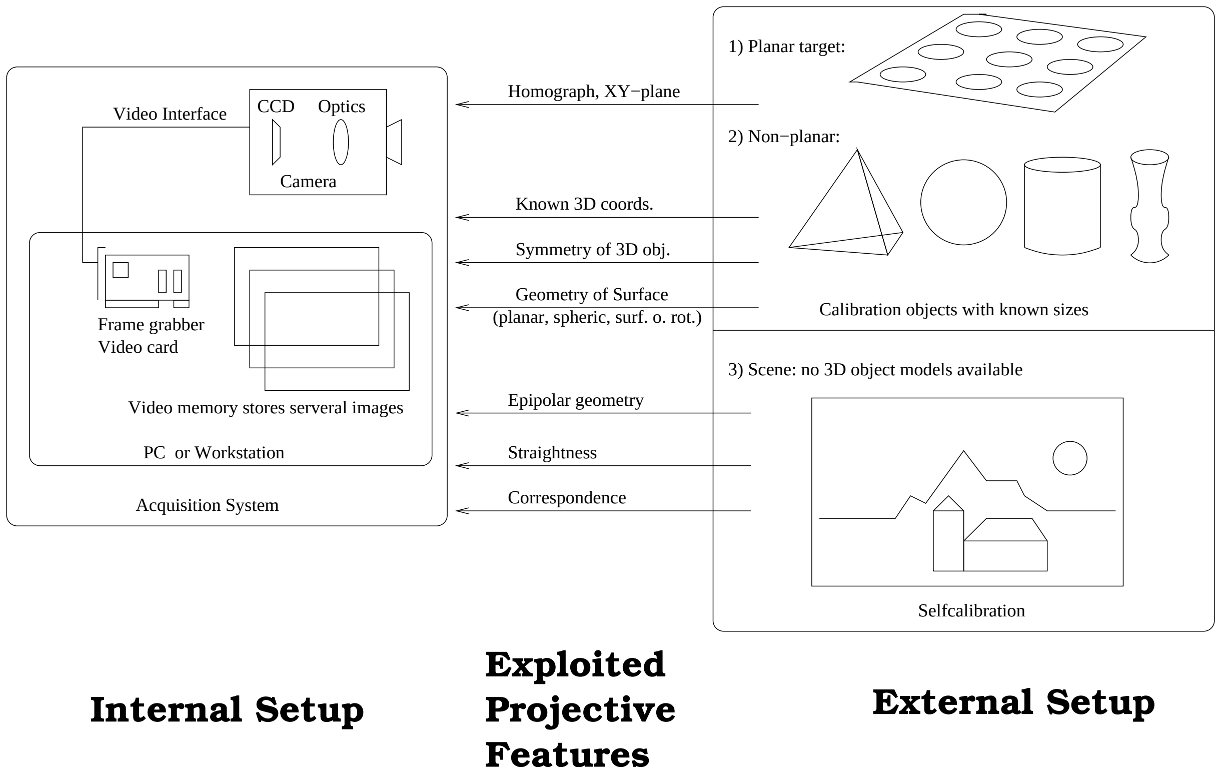


Figure 2.1: Entities and possible setups for camera calibration with required a-priori knowledge and exploited projective constrains

2.1 The big picture

The entities of concern in camera calibration are the *acquisition system*, that has to be calibrated, and the *real world environment*, that is recorded during the calibration process. The acquisition system applied for digital 3D vision tasks, consists of the following parts:

- a camera, with *optics* suitable for close range (0.5 - 5 meters) photography and with a typical field of view <25 degree, and a *CCD sensor* of known size and resolution. Depending on the type of sensor the CCD delivers gray-scale or RGB color image information.

- a *video interface*, that transmits the image information acquired from the sensor to another device. Older video standards (like Composite-Video, S-Video) work with analog transmission. In general digital transmission is more preferable due to signal quality.

Popular standards for digital video interfaces are Firewire™and USB 2.0™. In case of color imaging there are three independent RGB channels.

- a *PC or workstation* in which a *frame grabber* or *video card*[2] is integrated. In case of an analog interface the computer hardware has to re-digitize image information. This certainly lowers image quality. The resulting digital images do not necessarily have CCD sensor resolution. Indeed this is not true when using analog interfaces. Also digital image file format can have an effect on image quality. One should not use e.g. JPEG, since its compression mode is lossy an can produce unwanted artifacts.

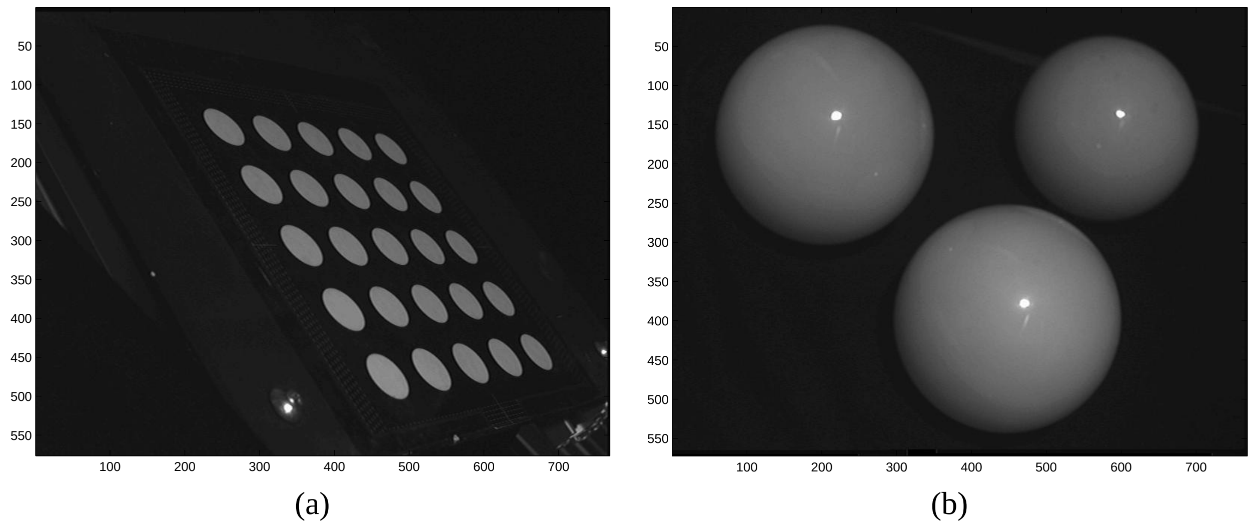


Figure 2.2: Digital image of (a) planar calibration object with circular control points (b) non-planar spheres

As second major aspect the structure of the recorded environment and its known objects classifies calibration methods. In Figure 2.1 a global disposition of types of calibration environments is given rightmost. A basic distinction should be made between *object calibration* with calibration objects,that have a known geometry and size, and *self-calibration* that uses invariant features of several views of a rigid 3D scene space and therefore does not rely on a pre-manufactured calibration objects. While self-calibration requires the change of orientation of the camera towards the scene[3], in object calibration the camera position is rigid and the position of the calibration objects is altered in case we use multiple views. Both approaches have their benefits and drawbacks. Object calibration generally is more accurate and can work with more complex camera models. In return the calibrated scene space is limited to the extent and positions of the calibration objects.
This may cause difficulties especially when it comes to *plane-based calibration*([4],[5],[3]),

[2]The basic difference between a frame grabber and a video card lies within performance. While a grabber needs at least one to several seconds to process one image (or frame), video card process in (almost) real time (>15 images per second)

[3]This does not have to mean, that we have to move the camera. In fact the scene can be re-positioned, if this does not affect its rigid appearance.

where the calibrated range [4] is defined by planar surfaces in 3D space. In this case a set of co-planar control points serves as calibration target. Normally these calibration points form a regular grid of known size and dimension and are defined by circles (see Figure 2.1a), squares or a checkerboard pattern. These geometric figures lose some geometric features by perspective projection, e.g. the center of gravity of circles and squares within the image is not identical to the position of its counterparts in the real world environment). A short summary of the pitfalls and solution methods of the contour-based control point detection problem is given in [6]. The geometric properties of the perspective projection of planar objects cause some useful constraints for geometric calibration (e.g. parallelism constraint in [3], simplified DLT in [5], simplified Kruppa equations in [4]). Therefore plane-based calibration is the method of choice from a numerical and mathematical point of view. Calibration on basis of a regular 3D-point-grid ([7]) can be seen as an extension to plane-based calibration, since it still uses co-planarity of planes together with orthogonality of grids. The constrains and algorithms used resemble that of plane calibration.
Though most approaches use planar calibration objects, some work has been done on calibration using spheres ([8], Figure 2.1b) and rotation symmetric bodies ([9]). Additionally in photogrammetry ([1]) calibration is also done using so-called *pass points*. The relative position of these pass points to each other is previously measured. It is sufficient to measure the distance for each pair in the pass point set to determine their absolute orientation in the camera coordinate system. This condition is met by our method purposed in Section 3 of this report.
The concept of self-calibration established just at the beginning of research in photogrammetry, but it is still an important topic in 3D vision. Some very interesting reports were published in recent years at INRIA ([10],[11]).
How can the method proposed be categorized? The method can be applied for plane-based calibration (as shown in test results) as well as for the non-planar case using pass points. Though it was not investigated, it should also be possible to calibrate with a 3D-point-grid.

2.2 Projective models

The projection of objects and shapes of the 3D environment to a specific 2 dimensional surface is described by the *projective model*. For some projective models certain constrains can be deduced from its *projective geometry*. The projective model of a CCD camera is determined by its optics and the relative orientation of the CCD towards the optical axis.
In the history of arts and to some extent in computer vision visualization of 3D worlds in pictures or on flat displays is done according to *perspective projection*, since perspective fulfills a key constraint of human visual perception, the so-called *straightness* constraint. Straightness guarantees that any straight line or connection in the 3D environment is projected as straight line in the 2D projection. Additionally all flat surfaces in the 3D environment retain their *flatness* property in the projection. For any sets of parallel lines or planes *vanishing points* can be constructed. For any set of correspondences of different views of a scene *epipoles* and resulting *epipolar lines* can be derived out of the image information. All this is directly implied by the straightness of perspective projection. Unfortunately one can not say, that perspective projec-

[4]We will explain this term more precisely in the following

tion is the only true and real projective model. Indeed, perspective produces some effects that do not naturally comply to human perception. Spheres e.g. generally do not preserve their circular shape, rotation symmetric bodies lose their symmetry property by perspective projection. Despite concentration on perspective model for machine vision was criticized by some authors (see e.g. [12], [13]), it is the standard model for calibration of close-range CCD cameras. This is due to the fact that the effects mentioned in the previous paragraph are very limited, if the field of view is smaller than $90°$.

2.2.1 Perspective Projection and the Pinhole Camera

Perspective projection is an abstract, mathematical concept. It describes the projection of any 3D point P onto a (virtual) projection plane, where it corresponds to an image point p. We denote the coordinates of the real world 3D point with (x_c, y_c, z_c) and that of the image point p with (u, v). The origin of 3D coordinate system lies at projection center C which is the exact position of the observer. Its XY-plane is parallel to the projection plane. The intersection between the projection plane and the line of sight is referred to as the *principle point (abrev. p.p.)*. In Figure 2.3 the geometry of perspective projection is illustrated. A specific projection is characterized by the following parameters:

- the distance of the virtual observer from the projection plane, called *focal length* f and
- the direction of the line of sight, which defines the *principal point (p.p. or* p_0*)* within the projection plane. We denote its image coordinates with (C_x, C_y). Additionally the line of sight is always a normal vector of the projection plane.

From Figure 2.3 it is easily seen that the triangles $\triangle(p, p_0, v_0)$ and $\triangle(P, P_0, y_0)$ are congruent and parallel to each other. From this the following relationship is derived:

$$\frac{x_c}{u} = \frac{y_c}{v} = \frac{R}{r}. \tag{2.1}$$

Also $\triangle(C, p, p_0)$ and $\triangle(C, P, P_0)$ are congruent from which follows:

$$\frac{R}{r} = \frac{z_c}{f}, \tag{2.2}$$

with $r = \sqrt{u^2 + v^2}$ and $R = \sqrt{{x_c}^2 + {y_c}^2}$.
From 2.1 and 2.2 we can now compute the image coordinates (u, v) of the projection of any given point P(x_c, y_c, z_c) with the following equation:

$$u = f\frac{x_c}{z_c} \qquad v = f\frac{y_c}{z_c} \tag{2.3}$$

As mentioned the perspective projection model is only a mathematical construction and has no exact physical model. The camera type which comes closest to perspective projection is the *pinhole camera*. This camera was indeed the first photographic device which was discovered[5].

[5]The earliest mention of this type of device was by the Chinese philosopher Mo-Ti (5th century BC). He formally recorded the creation of an inverted image formed by light rays passing through a pinhole into a darkened room. He called this darkened room a "collecting place" or the "locked treasure room."

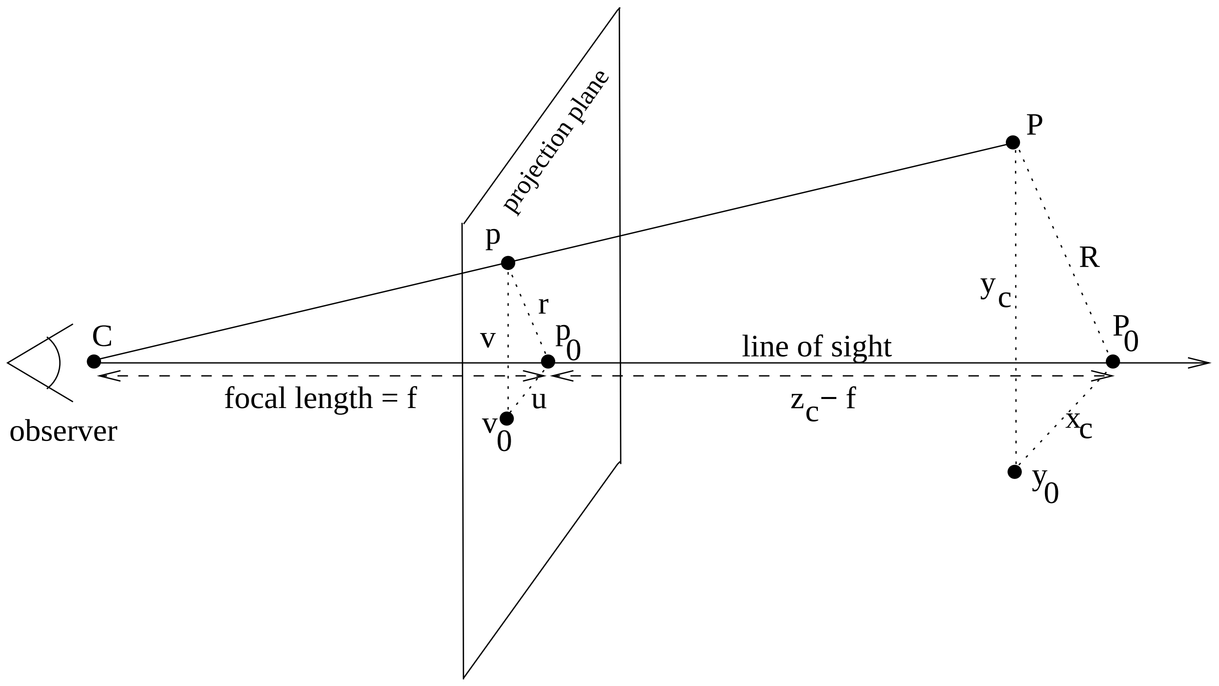


Figure 2.3: Geometry of perspective projection

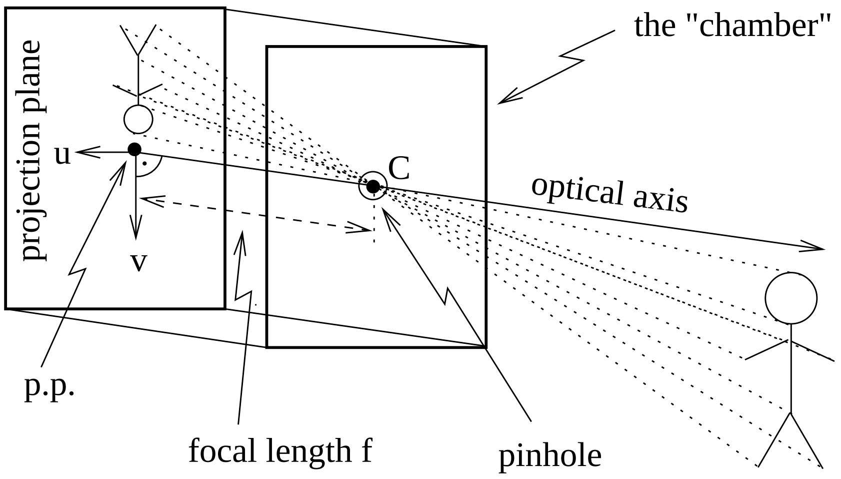


Figure 2.4: Schematic illustration of a pinhole camera

The device, also known as *camera obscura*[6], is a very simple construction. It basically consists of a photo-resist box, the so-called *chamber* in which a tiny *pinhole* is cut. As we see in Figure 2.4 all rays of light passing through the pinhole project an image onto the opposing plane inside the box. The image seems to be upside down an flipped and is generally very dark. Also blur is present due to the fact, that the rays do not exactly intersect in a single point. The laws of projection for the pinhole camera are same as for perspective projection. Why? Note that if we put the projection plane in Figure 2.3 behind the observer the image will also be inverted geometrically and the geometry itself is the same as for the pinhole camera changing nothing we derived from the congruence of the triangles we investigated. In fact for a given focal length and a line of sight there are always to possible projections — one in front and one behind the observer.

2.2.2 Lens projection model

If we want to build a pinhole camera, that has absolutely perfect perspective projection geometry, its pinhole must be of infinitesimal size, since all rays of light passing through the pinhole should intersect in one single dimensionless point. Physically this is not possible[7].
Since the tinier the pinhole the smaller the amount of light passing through will get, long exposure times are required. Therefore the use of pinhole cameras for cinematic photography or snap shots is ruled out. In order to build optics with better illumination the pinhole is replaced with an *optical lens*. In the following the main characteristics of optical refraction are discussed.

Refraction

If light rays pass the border surface of two physically different mediums, like air and glass, they are refracted according to the *law of Snellius*:

$$\frac{\sin\alpha}{\sin\beta} = \frac{n_1}{n_2}, \tag{2.4}$$

where α is the angle of incidence and β the angle of deflection, n_1 is the refraction index of the medium from where light rays are emitted, n_2 its counterpart for the other medium. The angles are measure with respect to the surface normals at the points were the light rays are refracted.

In Figure 2.5 refraction at spheric surfaces is illustrated. Let R be the radius and M be the center of the corresponding sphere and $\overline{OF}$ be a randomly chosen projection axis. Then a paraxial ray is refracted at S according to the law of Snellius with incident angle α and deflection angle β. The angle between the axis and the deflected ray is given by $\gamma = \alpha - \beta$, the distance of A from the axis is given by $h = R \cdot \sin\beta$. We now can make the following conclusions:

$$b = \frac{h}{\sin\gamma} = \frac{h}{\sin(\alpha-\beta)} \quad = \quad \frac{R\cdot\sin\beta}{\sin\alpha\cdot\cos\beta - \cos\alpha\cdot\sin\beta} \tag{2.5}$$

[6] The term "camera obscura" was first used by the German astronomer Johannes Kepler in the early 17th century. He used it for astronomical applications and had a portable tent camera for surveying in Upper Austria.

[7] Classical theories of optics fail at the point the pinhole is tiny enough. Indeed we are right in between quantum physics.

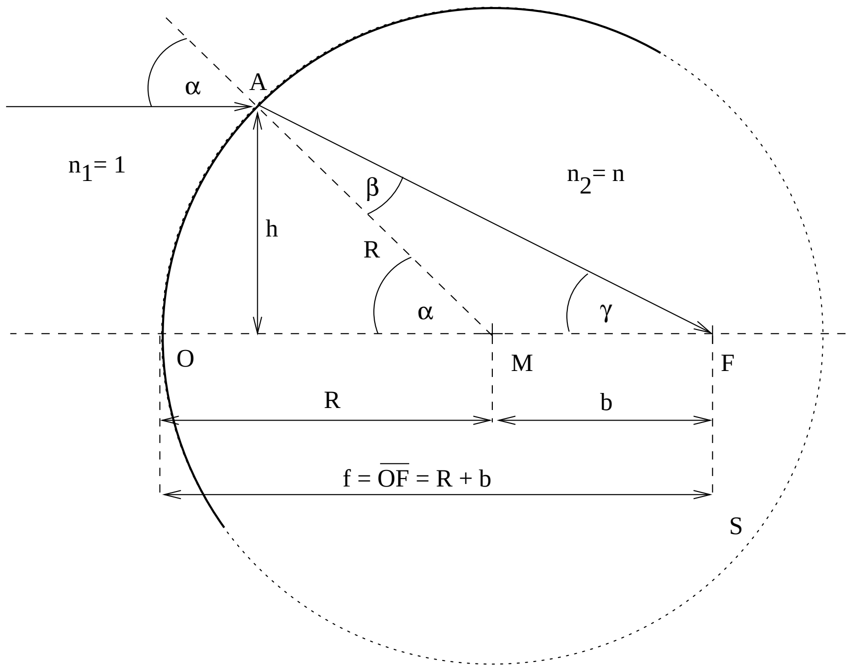


Figure 2.5: Definition of the focal length f in case of refraction at spheric surfaces.

A *focal point* F is now defined as the point where[8] all par-axial rays are bundled. We can formulate the following approximation for the distance of F — the so-called *focal length f* — of the spheric projection of S defined by $\overline{OF}$ and M with radius R:
If we assume that $\alpha, \beta \ll 1rad (\approx 57,2958°)$, then $\cos\alpha, \cos\beta \approx 1$ and we get:

$$f = R + b \ \simeq R \cdot \left[1 + \frac{\sin\beta}{\sin\alpha - \sin\beta}\right] \ = R \cdot \frac{\sin\alpha}{\sin\alpha - \sin\beta} \tag{2.6}$$

According to Equation 2.4 the *law of spheric refraction* can be formulated:

$$f \simeq R \cdot \frac{n_2}{n_2 - n_1} \tag{2.7}$$

If light rays are emitted from inverse direction the distance f' of focal point is approximated by:

$$f' \simeq R \cdot \frac{n_1}{n_1 - n_2} \tag{2.8}$$

Thin radial biconvex lens

The facing of radial lenses is defined by two opposing spheric surfaces with radius R_1 and R_2. According to their orientation towards the image plane their numerical values are signed. If $R_1 < 0$ and $R_2 > 0$, the lens is *biconcave*. If $R_1 > 0$ and $R_2 < 0$ the lens is *biconvex*. In case the thickness d of the lens is considerably smaller than the absolute values of R_1 and R_2, we

[8]Due to the fact that we draw our conclusion from an approximation, par-axial rays do not actually intersect in one point. In fact this is only a good guess for narrow angles of $\alpha \ll 1rad$

talk about a *thin lens*. In the other case this is a *thick lens*.
In Figure 2.6 the refraction of a light ray by a biconvex radial thin lens is illustrated. Additionally by R_1=R_2 the lens is symmetrical[9]. The point B is a projection of point A. Any ray of light passing through the lens is refracted at S_1 and S_2. Let the refraction index of the surrounding medium $n_1 = n_3 = 1$, and that of the lens medium $n_2 = n$. Therefore and according to equation 2.4 α_1 which is the angle of incidences for the refraction at S_1 equals α_2 which is the angle of deflection for the refraction at S_2. Also β_1 and β_2 are equal. This is true, because points A and B are the two focal points of S_1 and S_2 for the forward projection.
It can be derived from approximation 2.7 that the following relationships hold in this case:

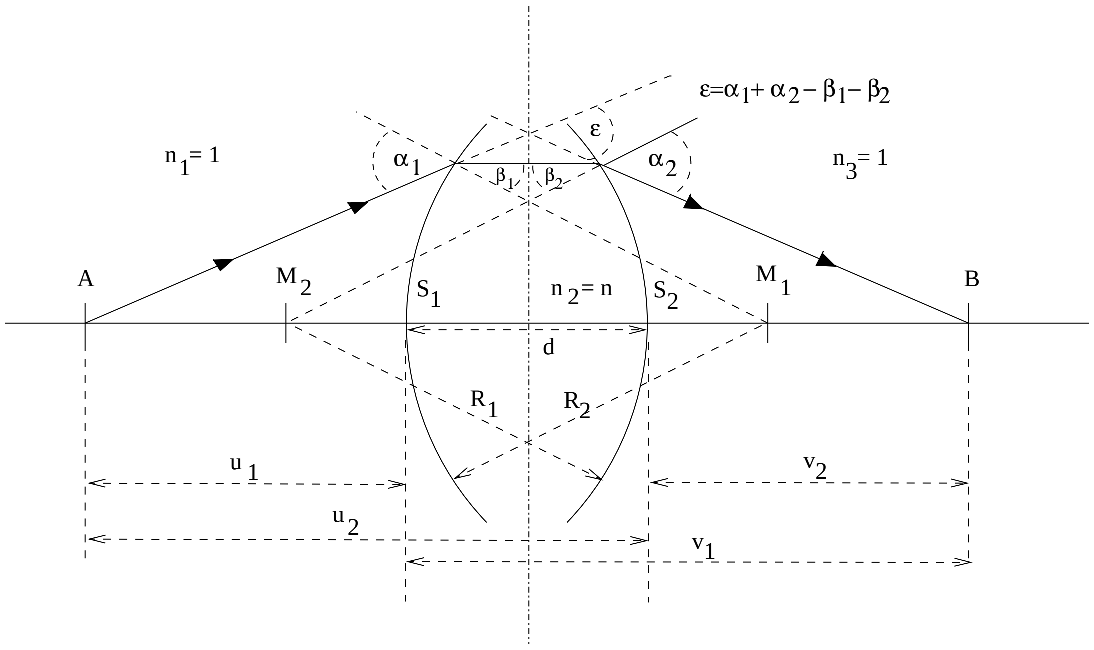


Figure 2.6: Illustration of the thin lens projection: an object point A is projected by the biconvex radial lens to the image point B. The course of one sample ray is depicted.

$$(n-1)\cdot\left[\frac{1}{R_1}-\frac{1}{R_2}\right]=\frac{1}{u_1}+\frac{1}{v_2} \tag{2.9}$$

and

$$\frac{1}{f}=(n-1)\cdot\left[\frac{1}{R_1}-\frac{1}{R_2}\right], \tag{2.10}$$

where $R_2 < 0$, because the lens is biconvex, and f is the distance of focal point from S_2. The relation 2.10 is also known as the *Lens Maker's formula*. If the lens is symmetric like in Figure 2.6 the focal lengths for both directions are equal. From equations 2.9 and 2.10 neglecting the thickness d by assuming that $u = u_1$ and $v = v_2$ we get:

$$\frac{1}{f}=\frac{1}{u}+\frac{1}{v} \tag{2.11}$$

[9]This is not a necessary condition for the following conclusions!

which is also known as the *Gaussian thin-lens equation*. The distances are now measured with respect to the center of the lens, which is is the equidistant point to M_1 and M_2. In Figure 2.7 geometry of lens projection according to equation 2.11 is depicted. It must be emphasized, that this is an ideal model, since two *approximations* were used in order to get the relation. Similarly

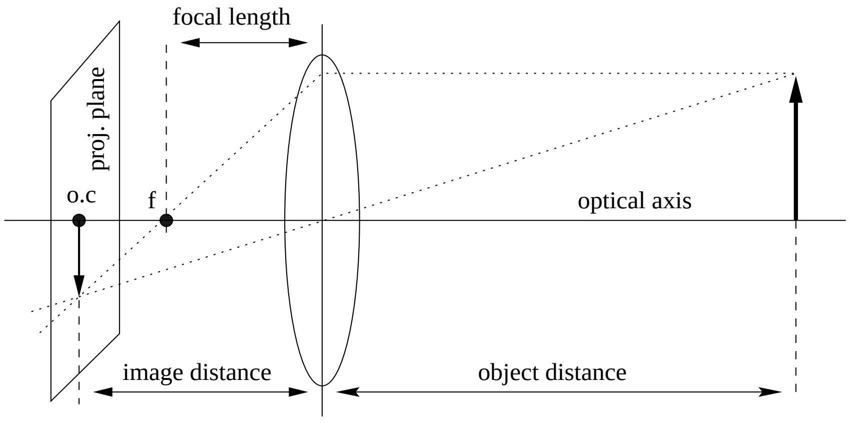


Figure 2.7: Schematic illustration of the camera lens projection

to the pinhole model all rays emitted from a planar surface with object distance U to the lens center are projected to points on a planar surface with image distance V to the lens center. If we assume that $U \to \infty$, then $V = F$. Therefore: if $U \gg V$ then $V \approx F$, which gives the thin lens model nearly a perspective projection geometry. As illustrated in Figure 2.8 all par-axial points

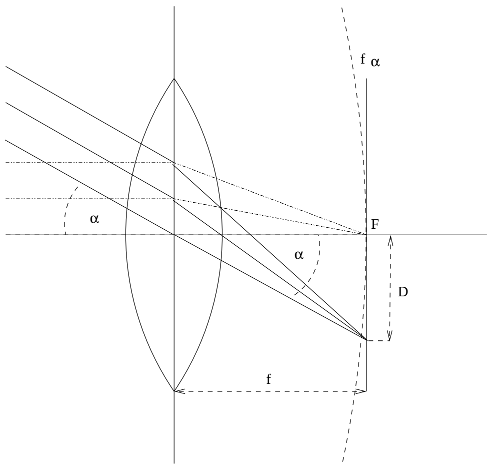


Figure 2.8: Projection of a parallel bundle of rays in the thin-lens model

(points that are parallel to the line of sight) are bundled in a focal point F with distance f from the lens center. Also a parallel bundle of rays[10] will intersect at one point that has a distance D

[10]In the pinhole model we can not really define a parallel bundle of rays, because the size of the pinhole is assumed to be infinitesimal.

from F. D is dependent on the angle of incidence α in the following way:

$$D = f \cdot \tan\alpha \approx f \cdot \alpha. \tag{2.12}$$

The approximation $f \cdot \alpha$ is adequate for smaller angles, and defines the spherical surface illustrated as dashed curve in Figure 2.8.
The scaling S_F of the projection is given by:

$$S_F = \frac{F}{U - F} \tag{2.13}$$

from which *Newton's projection formula can be derived*:

$$(U - F) \cdot (V - F) = F^2. \tag{2.14}$$

Thick lens

For thick lenses the projective geometry is considerably different. As depicted in Figure 2.9 any ray crossing the center of the lens is now deflected by a parallel displacement Δ. Its absolute value can be computed by:

$$\Delta = d \cdot \sin\alpha \cdot \left[1 - \frac{\cos\alpha}{\sqrt{n^2 - \sin^2\alpha}}\right] \tag{2.15}$$

Indeed the angle of incidence α measured with respect to H_1 and the angle of deflection α' measured with respect to H_2 are the same. This implies that there are two optical planes with two optical centers O_1 and O_2. Also the "real optical center" O is defined as the point where all crossing rays have equal angels of incidence and deflection. It can be derived from approximation 2.7 that the *Lens Maker's formula for thick lenses* is now:

$$\frac{1}{f} = (n-1) \cdot \left[\frac{1}{R_1} - \frac{1}{R_2} + \frac{(n-1) \cdot d}{n \cdot R_1 \cdot R_2}\right], \tag{2.16}$$

where d is the thickness of the lens. The distances h_1 and h_2 of the optical centers O_1 and O_2 to S_1 and S_2 (see Figure 2.9) is given by:

$$h_1 = -\frac{(n-1) \cdot f \cdot d}{n \cdot R_2}; \tag{2.17}$$

$$h_2 = -\frac{(n-1) \cdot f \cdot d}{n \cdot R_1} \tag{2.18}$$

In Figure 2.10 the construction of the projection of a thick lens is depicted. The difference to the thin lens model is that the thickness of the lens is not neglected. Therefore an additional length e is introduced designating the normal distance of the optical centers O_1 and O_2. If the thins lens is symmetric so that $f_1 = f_2$ and the optical planes H_1, H_2 are parallel then Equation 2.11 is valid also for thick lens[11].

[11]In this case we also assumed that the aperture is symmetric with respect to its position and size.

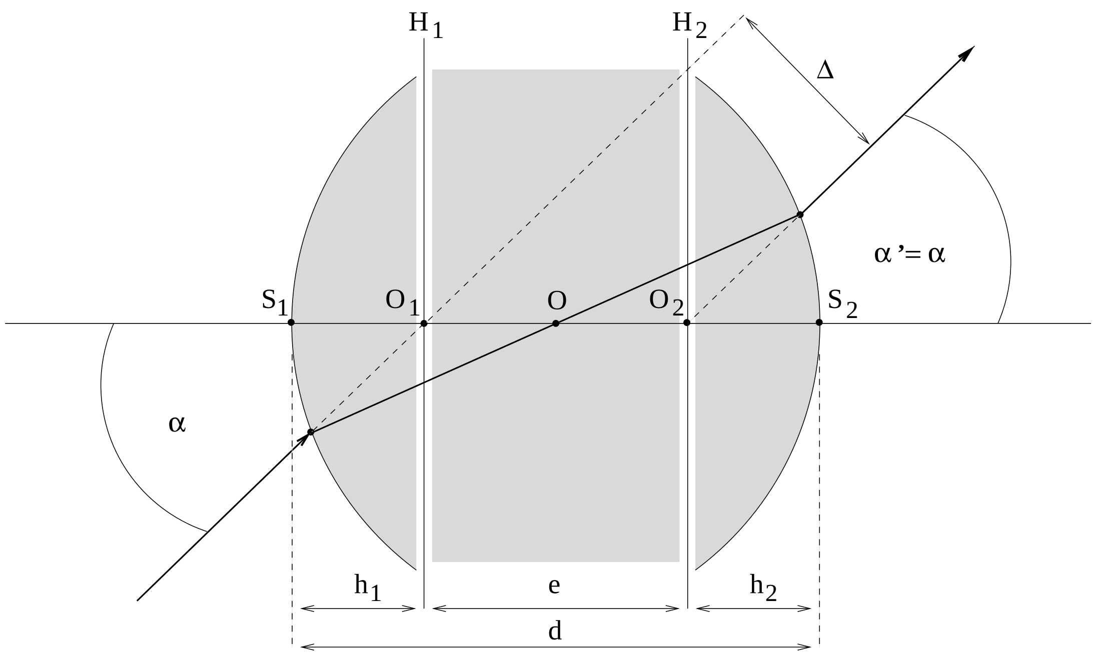


Figure 2.9: Thick lens projection: path of a central ray through the lens

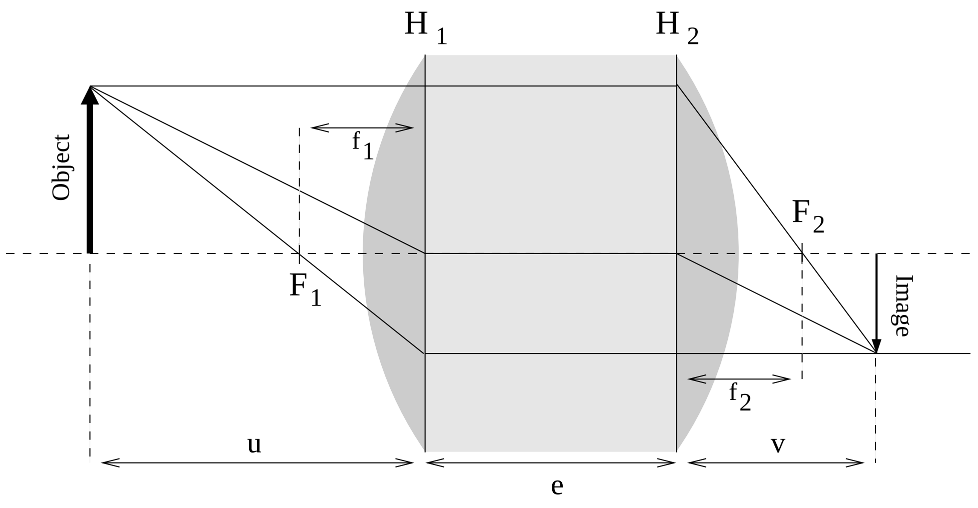


Figure 2.10: Geometric construction of a projection for thick lens: the rays are elongated by a distance e, geometry is basically the same as for thin-lens

Spherical Aberration

The Gaussian lens model is basically a rough approximation of the physical lens projection. Due to the bounding of refraction angles α and β in formula 2.7 the model gets inaccurate for rays that are remote to the optical axis. In fact the focal length is dependent on α. From Equations 2.7 the deviation of spheric refraction model from the physical model can be described functionally as follows:

$$f(h) = R \cdot \left[1 + \frac{1}{\sqrt{n^2 - \frac{h^2}{R^2}} - \sqrt{1 - \frac{h^2}{R^2}}}\right] \tag{2.19}$$

where h is the axial distance of the refraction point. This so-called *spherical aberration* effect is a systematic and radial-symmetric distortion effect. In principal it is possible to correct this effect by sophisticated lens systems. Due to lack of space we refer to [25] which is an in-depth analysis of applied optics and optical design.

Chromatic Aberration

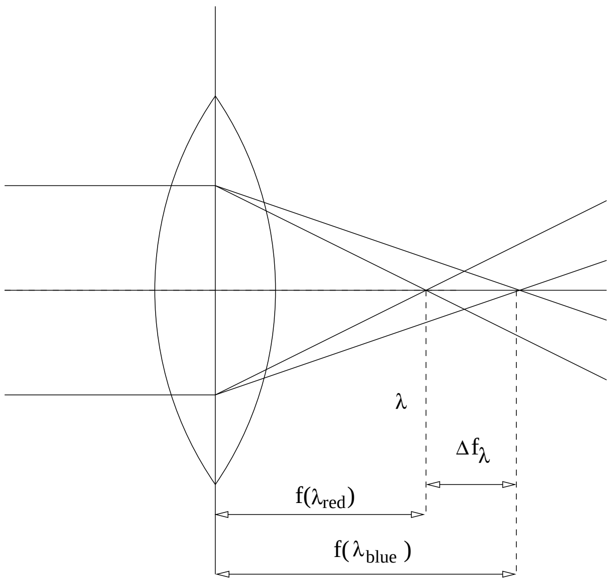


Figure 2.11: Effects of chromatic aberration: the focal length is dependent on the physical wavelength λ of the rays

The refraction index n in Equation 2.4 is only constant for a specific wavelength λ. Therefore the conclusions drawn so far are one valid for mono-chromatic light (LASER). The spectrum of visible light can be roughly specified with λ_{blue} = 380 nm to λ_{red} = 780nm. In Figure 2.11 the refraction of two white par-axial light rays containing the full visible spectrum at a radial biconvex lens is depicted. The focal lengths $f(\lambda)$ for $\lambda_{red} \leq \lambda \leq \lambda_{blue}$ will be within the range of Δf. In other words two par-axial rays will only intersect at a point with distance $f(\lambda)$, if they have the same wavelength λ. As mentioned the refraction index is functionally described by the chromatic refraction $n(\lambda)$. If $n(\lambda_{blue}) < n(\lambda_{red})$ the lens is characterized as

having *normal chromatic dispersion*.
In order to build an *achromatic lens* which features a very small Δf, a putty layer is applied un-evenly to the inside surface of a biconvex normally dispersive lens producing an asymmetric lens with inside radius:

$$R_2 = R_1 \cdot \frac{n_2(\lambda_{blue}) - n_2(\lambda_{red})}{n_1(\lambda_{blue}) - n_1(\lambda_{red})}, \tag{2.20}$$

where $n_1(\lambda)$ is refraction function of the biconvex lens and $n_2(\lambda)$ that of the putty material. R_1 is the radius of the spherical surfaces of the biconvex lens.

Astigmatism

Sometimes[12] the shape of thin lens in not perfectly radial, because the profiles of a pair of perpendicular meridians is not equal. In this case the focal length is also dependent on the position of the the rays seen from a frontal direction (identical to the line of sight). The dependency is known to be *radial-tangential*.

Depth of Sharpness and Distortion

Summing up the previous paragraphs the focal length f varies dependent on the following factors:

- *the angles of incident* of the rays emitted from a certain point with distance u from the lens center. This is due to *spherical aberration*
- *the wavelengths* of the rays emitted a point due to *chromatic aberration*.
- *the relative position* seen form a frontal direction, if the lens is not radial. This effect is known as *astigmatism*.

The thin-lens model suggests that the rays of each point of a plane perpendicular to the optical axis and with distance u to the lens center, will intersect in one single point at a planar area with distance v from the lens center. For all such projections the focal length f is assumed to be *constant*. As we have seen this is generally not the fact. Indeed for each ray emitted from some point the focal length is unique! Since photographic imaging is done by collecting the physical light information by a (usually) planar detector with has a defined distance to the optical center, the focal length is always under- or overestimated.

In Figure 2.12 the effects of over- or underestimating the focal length are depicted. Let O be a point with object distance u. We define to imaging planes for the projection with distances v' and v''. According to thin lens equation their focuses are defined by F' and F''. Let the physically correct focus for the projection of O be F. Then the rays from O passing the lens optics will intersect at a point with image distance v.

[12]The human eye possesses a natural astigmatism, since focal length is adjusted by squeezing the flexible eye lens.

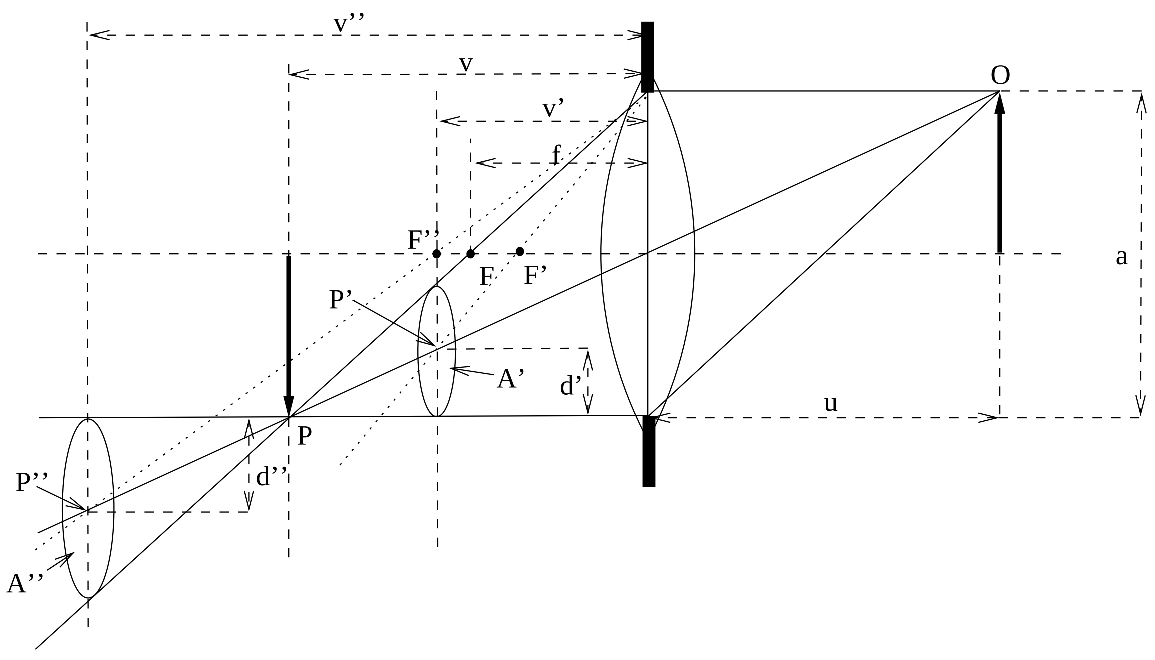


Figure 2.12: Effects of under- or over-focusing on sharpness and distortion

At image distance v' and v'' the rays will spread around a circular area[13] A' and A''. The diameters of the circular areas [14] are dependent on the deviation of the image distances from v (or alternatively from deviation of focal lengths to f) and the size of the *aperture a*. As long as these diameters are smaller than the granularity defined by the resolution of the detector, a picture of an object with distance u will still be sharp. If the diameters are bigger, blurring is present. The tolerance level of this effect is known as the *depth of sharpness*.
By the deviation of the focal length not only sharpness (appearance of the shape of its projection), but also the position of the projection within the image plane is affected. As depicted in Figure 2.12 the centers of the circular areas A' and A'' define approximately the positions P' and P'' of the projected point O as the planar detector will recognize them, indifferently, if blur is present or not. We notice that these deviations can be bigger than the granularity of the detector and thus will decrease by d' or increase by d'' the distance to the optical axis of the recorded image points. The effect is generally dependent on the deviation of the focal length and the size of the aperture. Since the focal length deviation is due to the spheric aberration, the absolute values of d' and d'' are increasing by the distance to the principal point (the intersection of the image plane with the optical axis). The projection of a straight line will appear slightly curved against or towards the p.p.. The effects are known as radial-symmetric *barrel* and *pincushion distortions*.
The effects of chromatic aberration are generally not systematic with respect to geometry and are dependent on the chromatic dispersion characteristics of the optics and the lightning and reflection features of the recorded scene. We generally suggest the lens system is achromatic

[13]The distribution of the rays is generally Gaussian like.
[14]If the lens are not radial the areas are ellipses.

and, if not, its effects should be normally distributed over the image area.
Astigmatism is responsible for systematic distortion effects that are not radial, but radial-tangential symmetric, which means that the projections of points equidistant to the optical axis are not equidistant to the p.p. within the image plane.

Calibration range

As we have seen distortion and sharpness are effects of optical aberration and are strongly interlinked with each other. In order to get a sharp image of a certain calibration object or a scene, we have to either adjust the nominal focal length, the image distance or the aperture. All tree adjustments implicitly define a volume with certain ranges within all rays emitted are projected sharply to the image plane. The calibrated focal length will be an average for the spatial volume we covered with our adjustment and the control points that were obeyed. Additionally the distortion estimates are also average values for the predefined range.

2.2.3 Projection models in photogrammetry

So far we have only dealt with single lens projection. Normally camera optics feature *lens systems* in order to correct aberration effects. A typical lens system consists of several lenses of various shapes and sizes. The lenses themselves are made of different sorts of glasses and materials.
In Figure 2.13 the ideal photogrammetric projection is illustrated. In principal the lens systems can be treated as a single thick lens. As discussed thick-lens projection features two optical principal planes. We denote the object-sided plane as H_1 and the image-sided one as H_2. The sum of the object distance u, the distance of the principal planes e and the image distance v defines the so-called *adjustment distance d*. In amateur photography this factor is used for the manual adjustment of the object distance, that is now measure with respect to the image plane. The optical system is assumed to be symmetrical with its aperture positioned in center of symmetry. This implies that the angles of incidence τ and the angle of deflection τ' of a central ray are equal. The optical axis is a surface normal to the image plane, since its p.p. is mutually the projection of K_1 and K_2.
The light that is passing through the optics is limited by the diameter of the aperture. The virtual picture of the aperture percept at the position of object bounds the bundle of rays which produces the effective image. This bundle of rays is generally referred to as *entrance pupil*, whereas the equivalent bundle of rays seen from the p.p. is called the *exit pupil*.
In practice the camera optics will never be perfectly symmetric. Indeed most photogrammetric cameras have very long and asymmetric optical systems with aperture s generally not positioned in center of the optics. As a consequence the entrance pupil will unlikely coincide with H_1, which results in a deviation of angles τ and τ'.
In Figure 2.14 the effect of this deviation is described graphically. In principal the reference axis of the inner orientation of the camera is not the optical axes $\overline{OO_P}'$ of the lens system. It is the object-sided principal ray H_S perpendicular to the image plane, that crosses the center O of the entrance pupil. The intersection between the image plane and $\overline{OO_P}'$ is called *principal auto-collimation point* H_A. The physical projection center lies within the center of the exit pupil O_P'. If we apply the pinhole model to the physical projection, then the two angle τ and

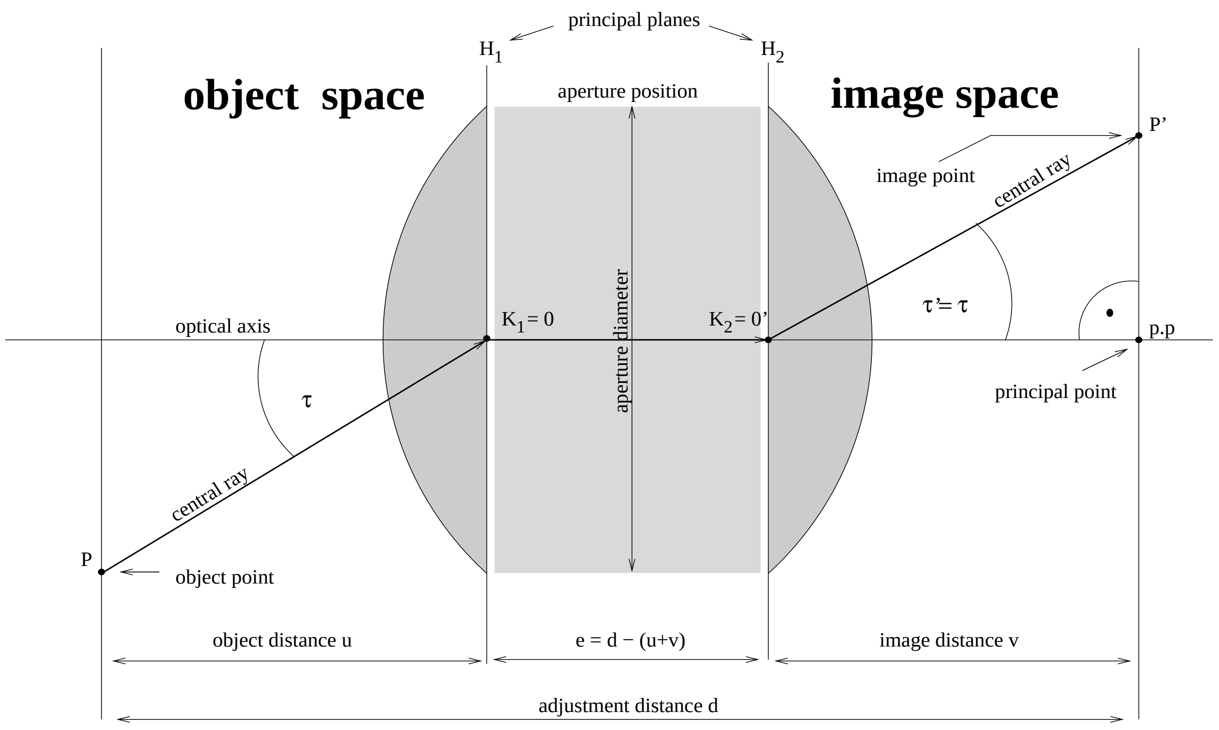


Figure 2.13: Ideal photogrammetric camera model

τ' are equal and the projection center is at $O_M{}'$, since we have to state a focal length c for the projection[15]. Now the expected distance $\rho' = c \cdot \tan \tau$ from H_A deviates from the the observed distance ρ by $\Delta\rho$, which is referred to as the *optical radial symmetric distortion*.

The question is now: Which factors determine $\Delta\rho$? In Figure 2.14 we see that the image plane seem to be considerably titled towards the optical axis. That does not necessary depict physical reality, (the position of the CCD sensor) but an imaginary image plane, valid only for a certain angle τ. The imaginary image plane is tangential at T to S, which is a spheric surface having O as radial center point. If τ is increased or decreased this will cause the imaginary image plane to be still tangential to S, increasing or decreasing $\Delta\rho$. If $\tau = 0°$ then the imaginary image plane is perpendicular to the optical axis, $\Delta\rho = 0$ and $O_M{}'$ lies on the optical axis . For any value of τ H_A will have the same position within the imaginary image plane being p.p. of the projection. Since c is supposed to be constant for all values of τ according to the pinhole model the normal distance of $O_M{}'$ to the image plane is also constant. Concluding from these observations we can state, that through the asymmetry of the lens system and its aperture, the physically correct projective surface would be spheric. The radial symmetric distortion emerges from the fact that the physical detector — a photographic film, or a CCD sensor — is usually planar.

[15]In photogrammetry the focal length is mostly called *camera constant c*. This is due to the fact that a focus or a focal length is a measure for the refraction on surfaces not especially for lenses and optics at all.

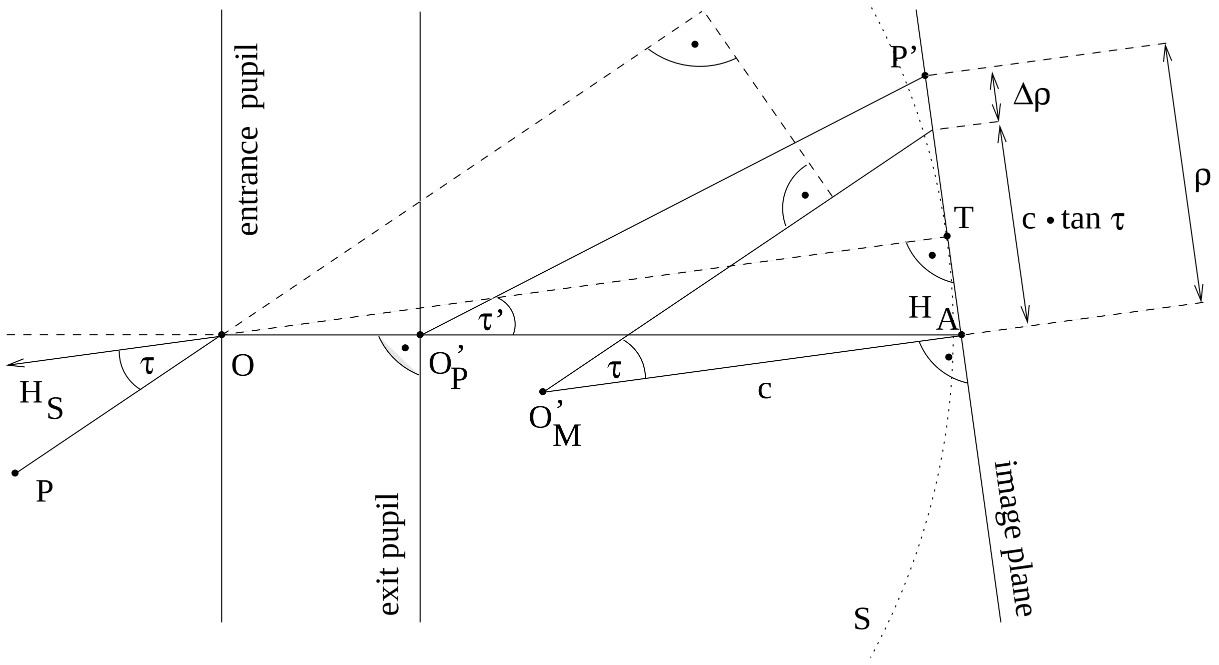


Figure 2.14: Schematic illustration of effects caused by assymmetric optics and de-centered apperture position

2.3 Calibration parameters

The calibration problem requires the identification of the camera model, which describes all the characteristics of the acquisition system with respect to an ideal projective geometry. With *camera model* we mean the set of mathematical relationships that link the 3D coordinates of a point in the scene space to the 2D coordinates of its projection on the acquired image. Such relationships can be defined in many different ways, as literature shows. Among them, a distinction could be made:

- Models defined by an operator (projection matrix) that link the coordinates of a 3D point to the coordinates of its projection in the image: The coefficients of such a projection matrix do not have any physical meaning, so the description of projection features of the camera system is *implicit*. To get the real world optical parameters like focal length and distortion factors further mathematical analysis of the projection matrix has to be done. A good example of a calibration method working on an *implicit camera model* is DLT (see [14]). Here a 3x4 projection matrix is used to project the 3D object coordinates in a homogeneous coordinate system to the 2D image coordinates also in a homogeneous coordinate system. To retrieve some of the real world physical parameters of the projection an RQ decomposition technique proposed by Melen in [15] can be used. The major disadvantage of this approach is that radial-symmetric distortion can not be estimated – indeed the computation effort is minimal.

- Models defined by a set of parameters describing *all optical and geometric parameters* of the camera explicitly. Therefore this kind of approach is called the *explicit model*. R. Y. Tsai's method ([3]) is a good example for that. The advantage of this approach is basically that all physical parameters can be estimated. It is also possible to use a-priory knowledge of the camera properties by keeping some of the model parameters constant.

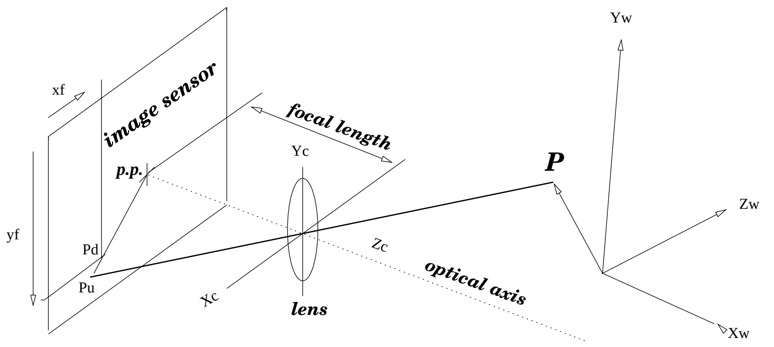


Figure 2.15: The adopted camera model: Point *P* with its object (world) coordinates (x_w, y_w, z_w) is transformed into the coordinate system of the camera (x_c, y_c, z_c), then it is projected into the image plane. Distortion causes the gap between P_d which is the real world projection point and the hypothetical projection P_u, that would be projected, if the lens system was perfectly linear (free of distortion).

The camera model that was adopted here, belongs to the later category. The illustration in Figure 2.15 sums up the whole camera model that provides a direct way of calculating the image coordinates of the projection of P, given its position in the scene space. This can be represented as a function *g* that maps a point *m* in the model space into a point *d* in the observation space. The function itself depends on e set of parameters that can be split up in two group:

- parameters describing the orientation of the object coordinate system to the camera coordinate system $(x_c, y_c, z_c) \to (x_w, y_w, z_w)$ — also called the *extrinsic parameters*. In the previous section we specified the *auto-collimation principal ray* H_S[16] as z-axis and the sensor surface to constitute the XY-plane of the camera coordinate system. The object coordinate system is freely chosen according to a set of defined world coordinates. It must be emphasized that both coordinate systems have to be simultaneously right-handed or left-handed in order to get the right physical orientation parameters.

- parameters describing the projection features of the camera-sensor system — the so called *intrinsic parameters*.

Extrinsic and intrinsic parameters can only be distinguished in the explicit model in a direct way. So explicit models are more suitable for calibration with multiple images of the calibration object, because the position independent intrinsic parameters have to converge to the same

[16]As described the direction of the auto-collimation principal ray in the object space is variable for any angles τ, but it is constant in the image space by definition (= direction of the normal distance of the center of the exit pupil to the sensor plane)

values for all views whereas the extrinsic parameters have to be estimated separately for each single view.
This set of equations is called *direct model*. For each fiducial point P_i the direct model provides image coordinates $d_{i,j}$ in the form

$$d_{i,j} = g(P_i, C, O_j); \tag{2.21}$$

where C is the set of camera specific intrinsic parameters and O_j is set of parameters describing the orientation of the object coordinate system in the jth view. $g : R^3 \rightarrow R^2$ is called the *complete camera model*.

2.3.1 Extrinsic parameters - outer orientation

The goal is to find a linear transformation between two coordinate systems. This is equivalent to a description of the relative spatial orientation of the camera lens and the calibration object. Geometrically all these orientations can be conceived as two linear transformations:

- a 3dimensional rotation *R* around all three axis
- a translation along a 3D translation vector $\vec{t}$

There are two mathematical representations for the rotation parameters:

- as a 3x3 rotation matrix
$$R = \begin{pmatrix} r_1 & r_2 & r_3 \\ r_4 & r_5 & r_6 \\ r_7 & r_8 & r_9 \end{pmatrix}$$
where all 9 coefficients are taken as calibration parameters or
- the *Euler Angles* φ, ϑ, ψ that can be derived from the coefficients $r_1..r_9$ as follows:
$$\varphi = \arctan\frac{r_6}{r_9} \qquad \vartheta = \arctan\frac{-r_3\cos\varphi}{r_1}$$
$$\psi = \arctan\frac{r_2}{r_1}$$

The matrix R can be expressed as a matrix multiplication of three rotation matrixes ($R_x \cdot R_y \cdot R_z$) which can are computed as follows:

$$R_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\varphi & -\sin\varphi \\ 0 & \sin\varphi & \cos\varphi \end{pmatrix} \quad R_y = \begin{pmatrix} \cos\vartheta & 0 & \sin\vartheta \\ 0 & 1 & 0 \\ -\sin\vartheta & 0 & \cos\vartheta \end{pmatrix}$$
$$R_z = \begin{pmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Therefore the rotation matrix can also be written as

$$\begin{pmatrix} \cos\psi\cos\vartheta & \sin\psi\cos\vartheta & -\sin\vartheta \\ -\sin\psi\cos\varphi + \cos\psi\sin\vartheta\sin\varphi & \cos\psi\cos\varphi + \sin\psi\sin\vartheta\sin\varphi & \cos\vartheta\sin\varphi \\ \sin\psi\sin\varphi + \cos\psi\sin\vartheta\cos\varphi & -\cos\psi\sin\varphi + \sin\psi\sin\vartheta\cos\varphi & \cos\vartheta\cos\varphi \end{pmatrix}$$

The translation vector

$$\vec{t} = \begin{pmatrix} t_1 \\ t_2 \\ t_3 \end{pmatrix}$$

adds another three calibration parameters.

Summing up we now have the following set of calibration parameters for the outer orientation:
Rotation:$r_1, r_2...r_9$, or alternatively φ, ϑ, ψ; Translation:t_1, t_2, t_3

The whole transformation from one coordinate system to another now looks like this:

$$\vec{P_c} = R \cdot \vec{P_w} + \vec{t} \tag{2.22}$$

where the $\vec{P_w}$ are the 3D object coordinates and $\vec{P_c}$ its representation in the target coordinate system (e.g. the camera coordinate system)

To simplify Equation 2.22, we can use *homogeneous 3D coordinates*. In this case the original Cartesian space is expanded by an additional dimension. For point coordinates the value of the added coordinate is usually defined as 1. Formally this has in consequence, that the origin of a Cartesian coordinates system, which is usually linearly dependent to all vectors in the Cartesian coordinate system, now is linearly independent to all transformed vectors of the original Cartesian space. Homogeneous[17] coordinates are commonly used in the world of computer graphics. The attraction of homogeneous coordinates is that in a 2- or 3- dimensional transformation of an input coordinate system or object into an output coordinate system or changed object we do not have to split our operation into a part with a multiplication for the rotation matrix and scale factor, and separately have an addition for the translation vector. Instead we simply employ only a matrix multiplication having a simple homogeneous coordinate for a point and output coordinates for the same point after the transformation, which leads to the following equation:

$$\vec{P_w} = \begin{pmatrix} x_w \\ y_w \\ z_w \\ 1 \end{pmatrix}, \quad \vec{P_c} = \begin{pmatrix} x_c \\ y_c \\ z_c \\ 1 \end{pmatrix} = \begin{pmatrix} r_1 & r_2 & r_3 & t_1 \\ r_4 & r_5 & r_6 & t_2 \\ r_7 & r_8 & r_9 & t_3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_w \\ y_w \\ z_w \\ 1 \end{pmatrix} \tag{2.23}$$

The right-most lowest element of the matrix in Equation 2.23 can alternatively be set to a different value than 1, which would add a scaling element to the transformation.
By determining a linear transformation from an object coordinate system to a camera coordinate system, we managed to transform the object coordinates so that all real world distances and sizes can be computed relatively to the camera position. The camera coordinate system itself can be oriented in several ways. In this case its z-axis is identical with the optical axis of the camera lens system. Its origin lies within the optical center of the lens (see Figure 2.15). One could also take the x or y-axis to represent the optical axis — however, this is uncommon.

[17]Homogeneous coordinates are sometimes also refered to as *projective coordinates* in order to point out, that this is not designating a single point in the Euclidean space, but a projection of a point in the projective space, which defines its projection only up to a scale factor

2.3.2 Intrinsic parameters - inner orientation

The projection geometry of the camera is described by the intrinsic model parameters. In Section 2.2.2 we described numerous properties of optical lens projection. Now a suitable mathematical model has to be defined.
A specific perspective projection with focal length f and the principal point with coordinates (C_x, C_y) being the origin of the image coordinate system can be formulated as matrix operation according to Equation 2.3 on page 6:

$$\begin{pmatrix} U \\ V \\ S \end{pmatrix} = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & f & 0 \end{pmatrix} \cdot \begin{pmatrix} x_c \\ y_c \\ z_c \\ 1 \end{pmatrix}, \tag{2.24}$$

where $u_i = (U + C_x)/S$ and $v_i = (V + C_y)/S$, if $S \neq 0$.
It is easily seen that this is not a *linear projection*. It causes the system of the equations, that represent the direct model to be non-linear. The coordinates (u_i, v_i) represent the physical coordinates of projection on the sensor plane — the so-called *retinal coordinates*. These coordinates have to be normalized to the resolutions s_x and s_y of the digital image information delivered by the sensor. Normally the origin of retinal coordinate system is not in the origin of the retinal plane, but at a certain point (C_x, C_y), the position of the p.p.. Considering that a simple but complete intrinsic camera model for the projection can be formulated:

$$\begin{pmatrix} U \\ V \\ S \end{pmatrix} = \begin{pmatrix} f \cdot s_x & 0 & C_x & 0 \\ 0 & f \cdot s_y & C_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} x_c \\ y_c \\ z_c \\ 1 \end{pmatrix}, \tag{2.25}$$

where $u_f = U/S$ and $v_f = U/S$, if $S \neq 0$.
This model represents the projective geometry of a camera, that comes very close to the pinhole-camera model. As we have seen lens projection does not comply to perspective projection geometry. Therefore a systematic radial-symmetric distortion effect will always be present.

Distortion parameters

The physical effects that causes geometric distortion can be identified as *spherical abberation* (see Section 2.2.2) and *a decentered aperture position* (see Section 2.2.3). Practically spheric aberration will occur when using *wide-angle optics*, because the rays of incidence of objects near to the border of the image range are considerably bigger than 1 rad. In contrast this effect can be neglected when using *telephoto optics*, that feature very long and asymmetric lens systems. In this case distortion is caused by the decentering of the aperture position. Anyhow, we must assume that radial-symmetric distortion is always present, since aberration depends generally on the manufactured quality of the optics.
There are physically correct mathematical formulas, that can exactly model both aberration effects and their superimposition. Due to the fact that these terms are mathematically very

complex and that there might be a number of other deviations from our ideal camera model[18], it is more convenient to give an estimate of the distortion function by describing it as a finite *polynomial Taylor series*:

$$D(r_i, \vec{\kappa}) = \frac{r_d}{r_i} = 1 + \kappa_1 {r_i}^2 + \kappa_2 {r_i}^4 + \ldots + \kappa_n {r_i}^{2n}, \tag{2.26}$$

where $r_i = \sqrt{{u_i}^2 + {v_i}^2}$ and $r_d = \sqrt{{u_d}^2 + {v_d}^2}$ being the distance of the distorted image point (u_d, v_d) to the p.p.. In [16] some alternative models for radial-symmetric distortion are dis-

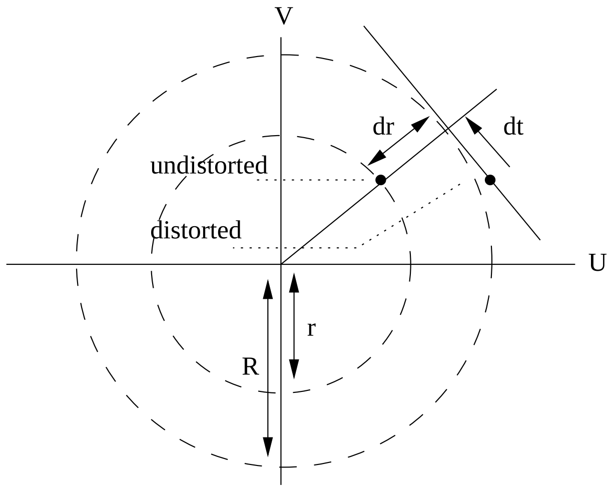


Figure 2.16: Schematic illustration of camera lens distortion

cussed. It is also mentioned that in general it is sufficient to use the first two terms of the Taylor series for the model function. Therefore we have

$$\begin{pmatrix} u_d \\ v_d \end{pmatrix} + \begin{pmatrix} D_x \\ D_y \end{pmatrix} = \begin{pmatrix} u_i \\ v_i \end{pmatrix}, \tag{2.27}$$
$$D_x = u_d(\kappa_1 r^2 + \kappa_2 r^4), \quad D_y = v_d(\kappa_1 r^2 + \kappa_2 r^4), \quad r = \sqrt[2]{u_d^2 + v_d^2}.$$

where (u_d, v_d) are the coordinates of the real-world distorted point and (D_x, D_y) is the relative distance to the undistorted point (u_i, v_i) in x and y direction. If just one term is used, the distortion function is defined by

$$D(r_i, \kappa_1) = \frac{r_d}{r_i} = 1 + \kappa_1 {r_i}^2. \tag{2.28}$$

The main advantage of using just one polynomial term is that there is a closed-form solution (see [16]) for derivation of the inverse model:

$$u_d = u_i \cdot D(r_i, \kappa_1), \qquad v_d = u_i \cdot D(r_i, \kappa_1) \tag{2.29}$$

[18]e.g. lens are not perfectly radial, transmission and dispersion features of the optics are estimates

In [17] a slightly different polynomial distortion function is described by Lenz:

$$u_i = \frac{2u_d}{1+\sqrt[2]{1-4\kappa {r_d}^2}} \qquad v_i = \frac{2v_d}{1+\sqrt[2]{1-4\kappa {r_d}^2}}. \tag{2.30}$$

Depending on the sign of the first order polynomial coefficient κ_1, the picture of a square will be distorted barrel-like, if κ_1 is negative, otherwise it will be distorted to a pincushion shape (see Figure 2.17).
Either we use Equations 2.27, 2.28 or 2.30 we can still not model radial-tangential distortion caused by astigmatism. In Figure 2.16 the effects of *radial-tangential distortion* are depicted. In general the direction of this type of distortion is defined by a tangent to the circle with radius R, which is the length of the position vector of an already radially distorted image point. Formally this can be described as follows:

$$\begin{pmatrix} u_d \\ v_d \end{pmatrix} + \begin{pmatrix} D_x \\ D_y \end{pmatrix} + \begin{pmatrix} Dt_x \\ Dt_y \end{pmatrix} = \begin{pmatrix} u_i \\ v_i \end{pmatrix}, \tag{2.31}$$
$$Dt_x = 2\tau_1 u_d v_d + \tau_2(r^2 + 2u_d^2)$$
$$Dt_y = 2\tau_2 u_d v_d + \tau_1(r^2 + 2v_d^2)$$

The two additional parameters for radial-tangential distortion are called τ_1 and τ_2, or alternatively τ_x and τ_y.

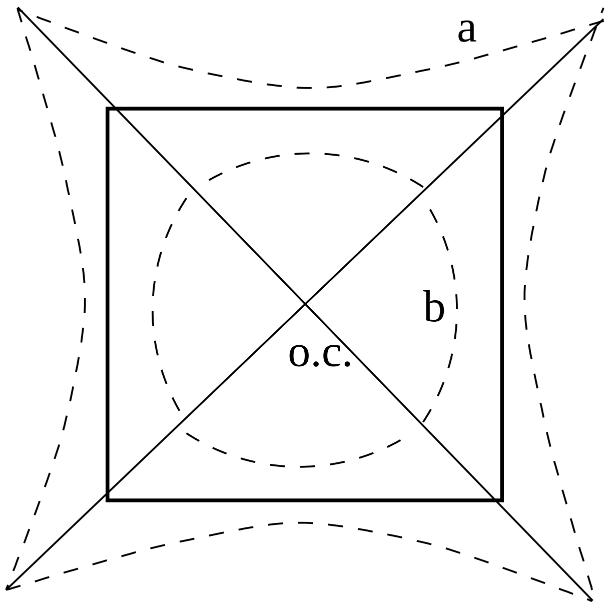


Figure 2.17: Schematic illustration of the barrel/pincushion-effect: The square around the optical center (p.p..) is extended to *a* or shrunken to *b* by radial lens distortion

Summing up we have now up to four different distortion parameters depending on whether we use Equation 2.27, 2.28, 2.30 or additionally 2.31. The more distortion parameters are calculated the more we expect the calibration procedure to be accurate. On the other hand this will consume more computation time caused by the increase of complexity. In fact this assumption does not always hold in practice. In the proposed solution Equation 2.31 is used to estimate the distortion effect, because it has shown that this method delivers considerably good results and is numerically stable (see [5]).

Sensor dependent parameters

So far we examined only effects that are determined by lens features. But there are also the sensor attributes that are important to know for correct image interpretation.
In the previous section we got the distorted point that should be projected to the projection plane. Ideally this projection plane should be identical with the sensor plane, where all digital sensor elements are placed. There, the projected light is recorded by a matrix of sensor elements. These sensor elements produce and analogue or digital (depends on the camera type) signal that is processed by the frame grabber module in case the signal is analogue.
First of all we have to deal with resolution parameters. Simply speaking: How large is the region in real that is represented by one pixel in a digital image frame?
Secondly we have also to determine where the computer image coordinates of the optical principal point (p.p.) are located in the pixel coordinate system. This is also vital for correct computation of the distortion effect, because all distortion depends on the distance between a point and the estimated position of the p.p. in the image [19]. We denote the pixel coordinates of the p.p. as (C_x, C_y) and take them as two additional camera parameters.

Figure 2.18 illustrates the meaning of the resolution factors d_x and d_y. Since the image is

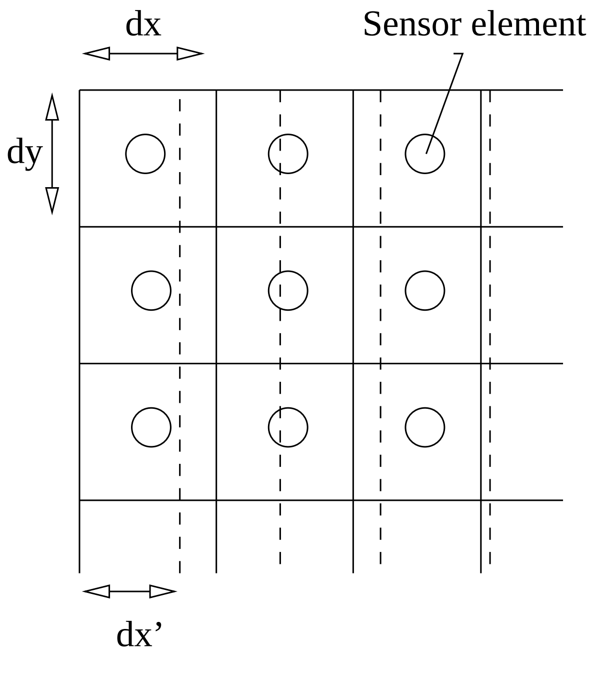


Figure 2.18: Horizontal and vertical resolution parameters

[19]It is well known that the center of distortion — the point where no distortion is present — is not equal to the p.p. in general. If we take that in account, we would still need a more complex camera model (see [1] for further details)

scanned line by line obviously, the distance between adjacent pixels d_y in the y direction is just the same as the center to center distance between adjacent CCD sensor elements in y direction. Therefore we can formulate the following relationship:

$$v_f = \frac{v_d}{d_y} + C_y$$

where v_f is the vertical pixel position of the distorted point u_f and C_y the y coordinate of the p.p..
In case the CCD camera delivers an analogue signal, a waveform is generated for each line of the image. Then it is sampled by the frame grabber into N_{fx} samples. Therefore, the horizontal resolution $d_{x'}$ (see Figure 2.18) directly relates to the pixel coordinate system in a way that:

$$u_f = \frac{u_d}{d_{x'}} + C_x, d_{x'} = \frac{d_x N_{cx}}{N_{fx}}$$

where N_{cx} is the number of sensor elements in x direction, and d_x is the mean distance between two adjacent sensor elements.
However, this is usually only true for *pixel symmetric CCDs and digital transmission*. When using analogue transmission and frame grabbers with low accuracy with respect hardware timing for scanning and digitization, an additional *uncertainty parameter* s_x has to be introduced. This is due to a variety of factors, such as slight hardware timing mismatch between image acquisition hardware and camera scanning hardware, or the imprecision of the timing of TV scanning itself. For example even a 1% difference in that can cause three to five pixels error for a full resolution frame. In case of a synchronous digital signal there is no need for the parameters s_x
Sometimes we do not have a priori knowledge of the sensor resolution (d_x,d_y), but of the digital image frame resolution (D_x, D_y). If we assume that one sensor element represents a square area within the projection plane then a pixel in the digital image will stand for a rectangular area with different range and area size if resolutions differ from each other. In this context we speak of rectangular pixels. A rectangle can be categorized by difference to a equal sized square. This is expressed by a parameter called *aspect ratio* or shortly s_{asp}. Formally the parameter s_{asp} has a resemblance to s_x in the camera model, but of course its meaning is different. On computing the camera model on the basis of digital image resolution the following formulas and parameters are important.

$$u_f = s_{asp}\frac{u_d}{D_x} + C_x, v_f = \frac{v_d}{D_y} \tag{2.32}$$

$$D_x = \frac{R_x}{N_{fx}}, D_y = \frac{R_y}{N_{fy}} \tag{2.33}$$

where N_{fx} and N_{fy} are the numbers of pixels in x and y direction of the digital image. Putting everything together now we have the following formulas and parameters:

	A priori known:
N_{fx}:	number of pixels in a line as sample by the frame grabber
N_{fy}:	number of pixels in a row as sample by the frame grabber
N_{cx}:	number of sensor elements in X direction
N_{cy}:	number of sensor elements in Y direction
d_x:	distance between adjacent sensor elements in X direction
d_y:	distance between adjacent sensor elements in Y direction
D_x:	proportionate distance of adjacent pixel centers in X direction
D_x:	proportionate distance of adjacent pixel centers in Y direction
R_x:	physical range of sensor element in X direction
R_y:	physical range of sensor element in Y direction
	Calibration parameters for estimation:
s_x:	uncertainty factor for horizontal scaling
s_{asp}	aspect ratio of x and y ranges
(C_x, C_y):	computer image coordinates for p.p.
	Coordinates:
(u_d, v_d):	real image coordinates in sensor plane
(u_f, v_f):	resulting computer image coordinates

Table 2.1: List of sensor dependent parameters

Sensor resolution parameters are known:

$$u_f = \frac{u_d s_x}{d_x\prime} + C_x, v_f = \frac{v_d}{d_y} + C_y \tag{2.34}$$

where $d_x{}' = \frac{R_x}{N_{cx}}$ and $d_y = \frac{R_y}{N_{cy}}$

Sensor size and image resolution is known:

$$u_f = \frac{u_d s_{asp}}{D_x} + C_x, v_f = \frac{v_d}{D_y} + C_y \tag{2.35}$$

where $D_x = \frac{R_x}{N_{fx}}$ and $D_y = \frac{R_y}{N_f x}$

All a priori parameters are known:

$$s_{norm} = \frac{d_x d_y}{D_x D_y}$$

Normalize focal length: $f' = \frac{f}{s_{norm}}$

Depending on the set of known a priori parameters either $\{s_x, C_x, C_y\}$ or alternatively $\{s_{asp}, C_x, C_y\}$ is the set of parameters, that is to be calibrated.

2.4 Estimation of the calibration parameters

The camera model consists of an extrinsic and an intrinsic model. The extrinsic model describes only the spatial orientation of some object to the camera optics and is only needed to calculate the whole projection of the object points. The the parameters of intrinsic model are invariant with respect to the pose of the camera towards a scene or an object and describe the common features of the CCD camera system.
Table 2.2 gives an overview of the whole projection and all calibration parameters needed to determine the camera model.

Step	Transformation	Operation	Parameters
1	$(x_w, y_w, z_w) \rightarrow (x_c, y_c, z_c)$	Rigid body transformation	$r_1, r_2 ... r_9$ or φ, ϑ, ψ and t_1, t_2, t_3
2	$(x_c, y_c, z_c) \rightarrow (u_i, v_i)$	Perspective projection	f
3	$(u_i, v_i) \rightarrow (u_d, v_d)$	Radial distortion	κ_1, κ_2 or κ
		Tangential distortion	τ_1, τ_2
4	$(u_d, v_d) \rightarrow (u_f, v_f)$	Scaling	$(C_x, C_y), s_x$

Table 2.2: Configuring the parameters of the calibration process

Step 1 is done by using the extrinsic camera model, Step 2-4 are calculated according to an intrinsic camera model of a kind that we presented in the previous sections.
What will be the result of the camera calibration? Assuming that we have already got the set *C* of all parameters for the extrinsic and intrinsic camera models we have to put our real world coordinates (x_w, y_w, z_w) of the object into the formulas of Step 1-4. Mathematically this can be represented by a vector function $g_C : R^3 \rightarrow R^2$ as follows:

$$g_C((x_w, y_w, z_w)) = (u_f, v_f)$$

2.4.1 Inverse Camera Model

In practice we need a model that expresses the the projection of a given 2D data set of image points to the control points of a 3D object. Therefore we have to find a mathematical method to compute the inverse function of g_C. In case we apply a distortion model with at least two radial distortion parameters like in Equation 2.31 this is not trivial, since there is no exact analytic method to the inverse mapping (the function g_C is not linear - an above all not linear in its parameters). The distortion function described by 2.31 de-linearizes the problem by causing the presented camera model to become a fifth order polynomial. The mapping from undistorted to distorted image coordinates $(u_i, v_i) \rightarrow (u_d, v_d)$ is given as follows:

$$\begin{aligned} u_d &= \kappa_2 u_i^5 + 2\kappa_2 u_i^3 v_i^2 + \kappa_2 u_i^4 v_i + \kappa_1 u_i^3 + \kappa_1 u_i v_i^2 + 3\tau_2 u_i^2 + 2\tau_1 u_i v_i + \tau_2 v_i^2 + u_i \\ v_d &= \kappa_2 u_i^4 v_i + 2\kappa_2 u_i^2 v_i^3 + \kappa_2 v_i^5 + \kappa_1 u_i^2 v_i + \kappa_1 v_i^3 + \tau_1 u_i^2 + 2\tau_2 u_f v_i + 3\tau_1 v_i^2 + v_i \end{aligned} \tag{2.36}$$

Nonlinear search is required to recover (u_i, v_i) from (u_d, v_d) in Equation 2.36. To avoid large computation effort some approximations for the inverse mapping were presented in literature ([5],[18],[19]). In the presented solution the approximation method in [5] is used.

3 Proposed solution

In order to build a robust and easily usable tool to calibrate the CCD cameras of a 3D image acquisition system at PRIP, the calibration method was developed and implemented. The main objective was to develop a software tool that can be used as a ready-to-use calibration toolbox in Matlab^{TM}. Additionally there are implementations for Khoros^{TM}and Mathematica^{TM}.
In this part only the fundamental ideas and algorithms are explained (for implementation details refer to Appendix A, B and C in this report). Furthermore no effort was made to create a specialized calibration technique that only works in some very limited arrangement. So different calibration parameter settings and even camera models can be used. The core of the work lies in the estimation of the outer orientation and the optimization procedure itself.

3.1 General description of the calibration method

Before the main calibration process can be started, the calibration points have to determined from the previously acquired image data. This tasks is generally referred to as *labeling* or in case of a 2D calibration target as *2D-grid labeling*. After labeling all calibration points one can assign all these points to the appropriate known world coordinates in all image data sets. In this report we present a robust labeling method that takes in account some odds concerning distortion effects.
The proposed solution of the calibration problem can be categorized as *nonlinear*. It uses several images (*multi-view technique*) to estimate all specified calibration parameters for a selectable camera model. The calibration object does not have to be a set of coplanar points. One could use any desired shape that is spanned by the calibration points. The calibration points should be numerous depending on the chosen camera model according to that the camera is calibrated.
The method differs from others presented in literature in the way how it estimates the outer orientation of the calibration object. In [20] the only start parameters needed are the scaling factors. Here we have to make an initial guess for all calibration parameters defined by the camera model. According to these start values the outer orientation of all views is calculated geometrically. We have to remark that the accuracy of these extrinsic parameters computed in this first step can not be determined because it heavily depends on the deviation of the start value for the focal length to the real world camera model, because this optimization step does not estimate the focal length. After all extrinsic parameters for all views are calculated the optimizer task tries to fit the whole camera model into the given image and real world data of the points. The method is described later on.

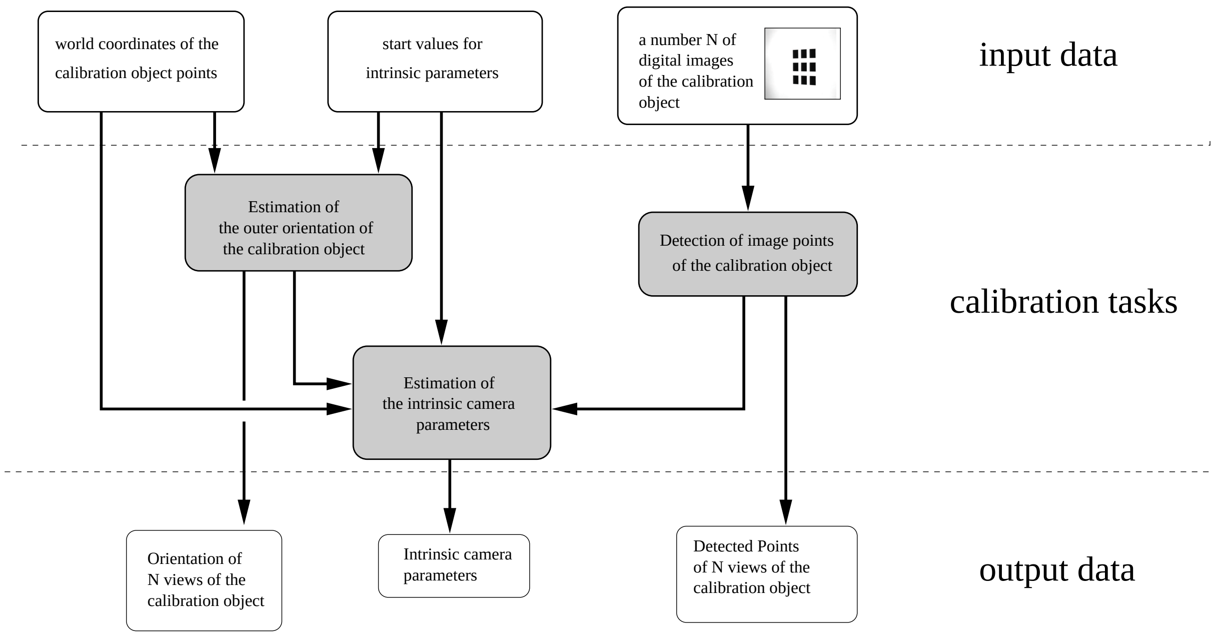

world coordinates of the calibration object points
start values for intrinsic parameters
a number N of digital images of the calibration object
input data
Estimation of the outer orientation of the calibration object
Detection of image points of the calibration object
calibration tasks
Estimation of the intrinsic camera parameters
Orientation of N views of the calibration object
Intrinsic camera parameters
Detected Points of N views of the calibration object
output data

Figure 3.1: Schematic diagram of the calibration process

So basically the calibration process can be divided into three different tasks (see Figure 3.1)

- *Detection of the calibration points in the images of all views of the object:* This is done by some 2D image processing algorithm (see Section 3.2) that computes the coordinates of the control points in the digital image with sub-pixel accuracy.
- *Estimation of the outer orientation of all views:* This tasks needs the world coordinates and the detected image coordinates of at least four calibration points per view, and the start values of the intrinsic parameters as input parameters. According to this data an approximation for the rotation matrix and the translation vector of the calibration object in each view is calculated. The robustness of the solution depends heavily on the number of points that were specified.
- *Final computation of all calibration parameters by the optimizer:* This is the most computation-intensive part of the calibration, because it requires the minimization of a system of functions that depend nonlinearly on a set of calibration parameters. This is done by a standard optimization algorithm called the *Levenberg-Marquardt method* that is described in [21] (see also Section 3.5).

3.2 Extraction and labeling of control points

After acquiring the gray scale image input data, a binary image is produced by a simple threshold algorithm which is referred to as *Otsu's method*. This binary image data are used to compute *connected components* (4- or 8 neighborhood). In order to avoid artifacts these components are filtered by size of area and their *solidity*, which is the ratio between the area defined by the convex hull of a component and its own area. Later criterion proved to be very useful for excluding artifacts of about the same area size, but with frayed edges. For the remaining components *centers of gravity* are computed as to stand for the extracted control points. A sample of this kind of preprocessing is shown in Figure 3.2.

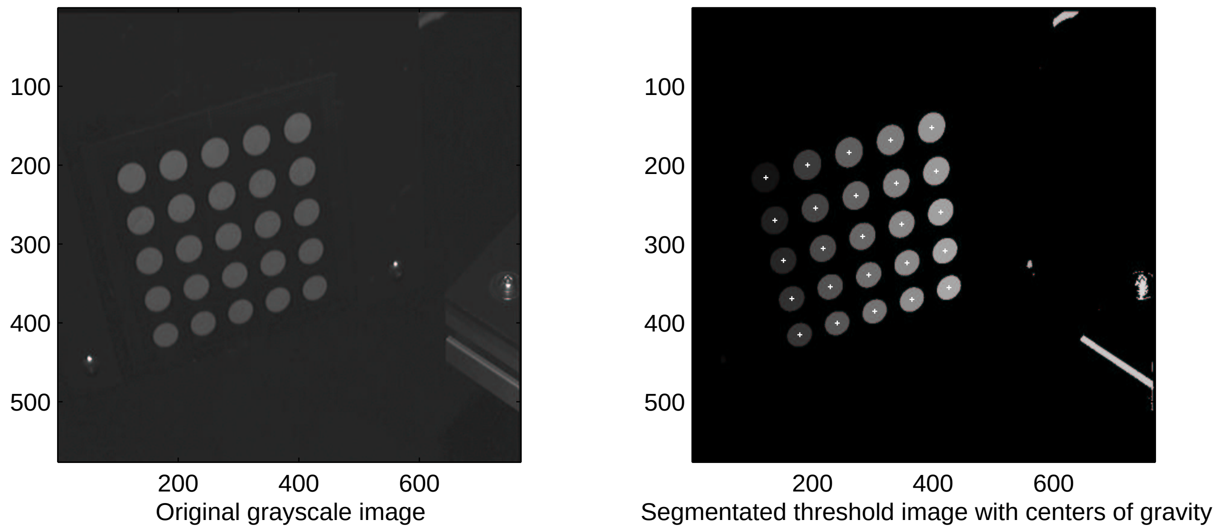


Figure 3.2: Preprocessing: (1) Threshold operation - (2) compute connected regions - (3) compute centers of gravity (4) filter according to area size and solidity

Grid labeling

Having extracted the location of the control points in the image data set, the next task is to match these image coordinates to the appropriate world coordinates for a known predefined calibration target. In this work we confine ourself to planar targets with regular grid patterns. Regular grids have one or more axes of symmetry which determines their nature (e.g. a square is a 4-symmetric pattern, a regular hexagonal pattern a 6-symmetric pattern). It is also postulated that the calibration object is visible on the image to its full extent, though not all calibration points have to be visible (e.g. some object lying on the calibration plane of which all corner points are visible).
One additional challenge arises from the presence of distortion in camera projection, because ,unlike in the pinhole projection model, straight lines are no longer straight. This prevents us from using a simple style grid matching algorithm which was applied by so many applications before.

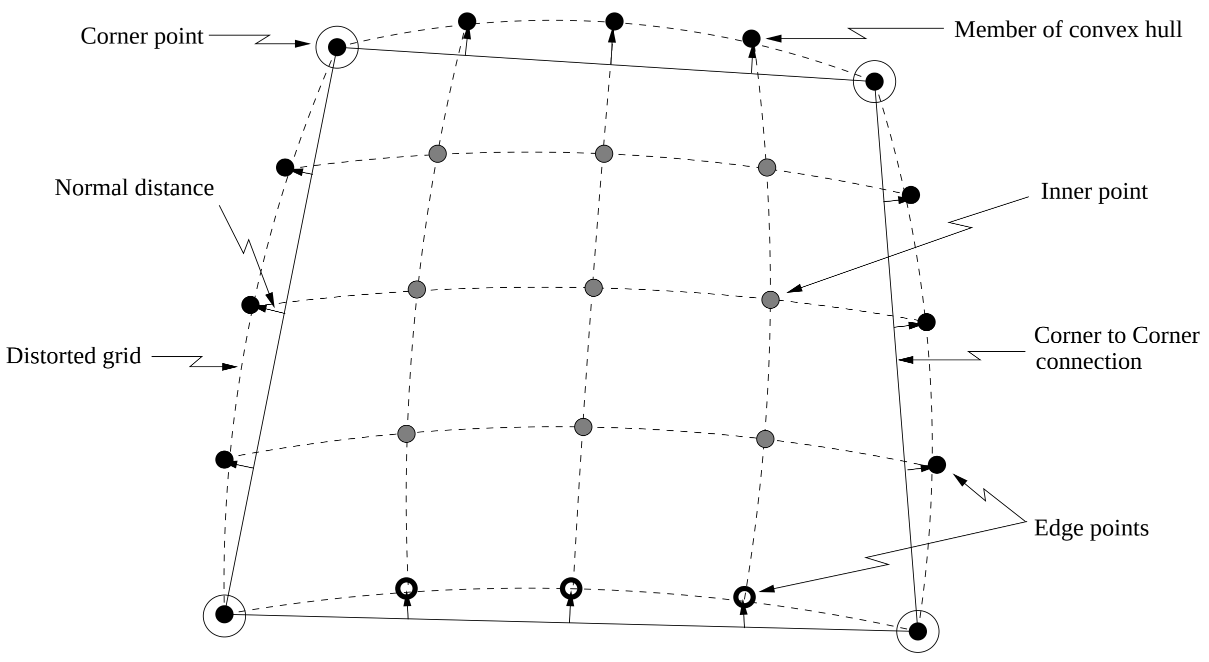


Figure 3.3: Labeling a 5x5 rectangular grid: Effects of image distortion - straight lines are no longer straight

Looking at the distorted projection of a number of points forming a regular grid the following observations are made (see Figure 3.3):

- All corner points are members of the convex hull.
- Some but not all edge points are members of the convex hull
- None of the inner points is a member of the convex hull, even if some control points on the edges are missing (if distortion is not extraordinary high)
- If we manage to detect edges and corners of the starting grid, we can iteratively solve the problem by taking the remaining inner points as the follow-up grid.
- Selecting the points with most acute angle between their adjacent points in the convex hull delivers the corner points of the grid. Moreover the number of corners is determined by the symmetry of the regular grid itself. Therefore this approach can be implemented for any regular grid, e.g. a hexagonal pattern.
- Points on the edges of the grid have minimal *normal distance* to the direct connection of the corners defining the specific edge. It is important in this case that the normal distance is measured with respect to the direction of the normal vector and is compared to the normal distance of any of the remaining corner points.
- If some of the inner points were mismatched as edge points in a first step, they can be relabeled in the following steps by comparing previous normal distances to current ones.

Taking into account these observation the following labeling algorithm was implemented:

1. Computation of the convex hull: This is done by implementing an algorithm proposed by Sedgewick in [22]. The points of the convex hull are sorted clockwise beginning at the point with minimal y location with index $i = 1..n_{ch}$.

2. Selection of the corner points: Get n corner points of the n-symmetrical grid by computing n most acute angles between all adjacent triplets of points of the convex hull. Order them as they appear in the clockwise ordered list of points in the convex hull. Take adjacent corners to define edges.

3. Selection of the edge points:

   (a) For all points P_i in the grid despite of corner points A and B defining edge $\overline{AB}$ compute normal distance given by:

   $$d(P_i) = (B - P_i) \cdot n_0 \cdot sgn(d(C)), \tag{3.1}$$

   $$n_0 = \frac{\vec{N}_{AB}}{\|\vec{N}_{AB}\|}, \vec{N}_{AB} = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \cdot (B - A),$$

   $$d(C) = (B - C) \cdot n_0,$$

   where n_0 is the normalized normal vector of $\overline{AB}$, C is another corner point and $d \in Z$

   (b) According to the dimensions of the grid we know the number m of edges points. Select m points with smallest normal distance.

4. Label all corners and edges according to the dimensions of the gird. Return to the first step with the remaining inner points

If the algorithm does not stop with a single point or a void dataset, we have to relabel the points in a backtracking manner. Therefore all distances for all points (except initial corner points) in every iterative step have to be stored and compared at the end.

3.3 Estimation of extrinsic parameters

The basic idea of this approach is that every 3D calibration object consists of a number of adjacent vertices spanned by its calibration points. Each vertex spanned by three calibration points can be taken as a calibration plane and associated to a specific object coordinate system. Because the relative orientations of all vertices to one another is known, the orientation of any 3D object can be derived immediately. The calibration object used in testing — a set of coplanar points on a calibration board — does not really make use of that special feature. But for comparison to other calibration methods a coplanar target was chosen.
So we have a set of vertices describing the calibration object, each consisting of 3 calibration points P1, P2, P3 of which we want to know the absolute orientation in respect to the camera position. The orientation is described by a rotation matrix R and a translation vector $\vec{t}$. In

photogrammetry literature this problem is usually called the *three point perspective pose estimation problem* for which a couple of solutions were developed by some mathematicians in the late 19th and early 20th century. For further historical details refer to [23].

3.3.1 Three Point Perspective Pose Estimation Problem

In order to compute the outer orientation we first determine the position of the three points P1, P2 and P3 with respect to the camera frame. To do so it is sufficient to know the absolute distances of the points S1, S2 and S3. In Figure 3.4 a schematic illustration of how the problem

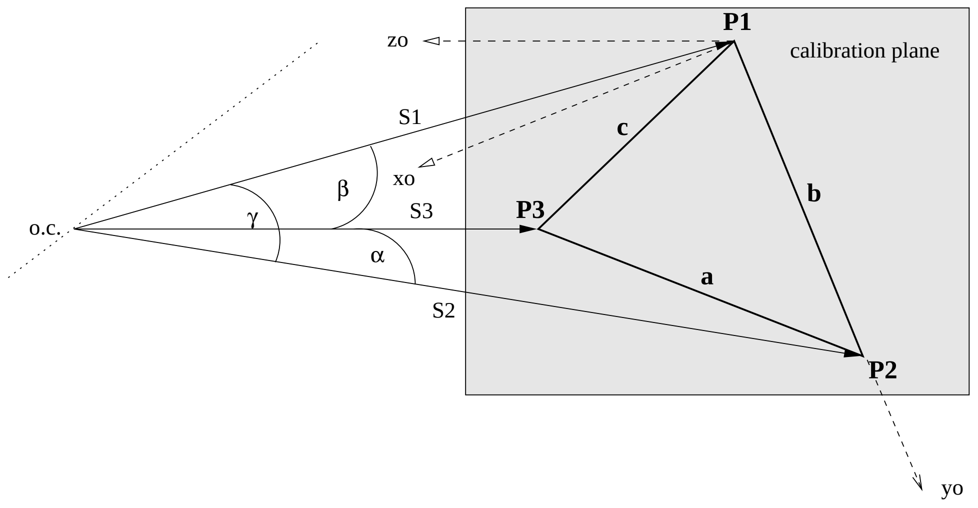


Figure 3.4: Schematic illustration of the three point space re-section problem.

can be set up is given. Here the point p.p. designates the optical center (center of perspective) which is the reference point for the origin in the camera coordinate system. The lengths of the vectors form the p.p. to the object points P1,P2 and P3 are the distances S1, S2 and S3 for those we want to have a solution. Knowing the world coordinates of the object points P1, P2, P3 we also know the side lengths of this triangle:

$$a = ||P2 - P3||, b = ||P1 - P3||, c = ||P1 - P2||. \tag{3.2}$$

Given the perspective projection — that means that we know the distances of the p.p. from the image plane (focal length f) — it is now possible to determine the positions of the 3 vertices constituted by the known triangle spanned by the object points P1,P2 and P3. There may be as many as four possible solutions for point positions in front of the p.p. and four corresponding solutions whose point positions are behind the center of perspective.
Now let the observed perspective projection of P1, P2 and P3 be p1, p2 and p3, respectively;

$$p_i = \begin{pmatrix} u_i \\ v_i \end{pmatrix}, i = 1, 2, 3$$

The coordinates of the real world object points are

$$P_i = \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix}, i = 1, 2, 3$$

By the perspective equations

$$u_i = f\frac{x_i}{z_i}, v_i = f\frac{y_i}{z_i}$$

the unit vectors j_i pointing from the center of perspective p.p. to the real world positions P_i of the observed points p_i are given by

$$j_i = \frac{1}{\sqrt{u_i^2 + v_i^2 + f^2}} \begin{pmatrix} u_i \\ v_i \\ f \end{pmatrix}, i = 1, 2, 3 \tag{3.3}$$

Then the p.p. together with the 3 object points form a tetrahedron with S_1, S_2, S_3, a, b and c as side lengths (see Figure 3.4) and the 3 vertices and the triangle as surface. Now let the angles at the p.p. opposite sides a,b and c be α, β and γ. Given the unit vectors the three angles can be determined as follows:

$$\cos\alpha = j_2 \cdot j_3, \quad \cos\beta = j_1 \cdot j_3, \quad \cos\gamma = j_1 \cdot j_2 \tag{3.4}$$

Summing up we have now the following known values:

- the side lengths a,b,c of triangle spanned by the three object points
- the angles α, β, γ at the p.p. opposite sides of the triangle

The distances S_1, S_2, S_3 are the 3 unknowns that have to be solved by an equation system in order to compute the real world positions of the object points according to the following formula:

$$P_i = S_i \cdot j_i, i = 1, 2, 3$$

If we look at the three vertices spanned by the p.p. and two of object points of the triangle in Figure 3.4, we can now proceed in setting up three equations according to the law of cosines:

$$S_2^2 + S_3^2 - 2S_2S_3\cos\alpha = a^2 \tag{3.5}$$
$$S_1^2 + S_3^2 - 2S_1S_3\cos\beta = b^2 \tag{3.6}$$
$$S_1^2 + S_2^2 - 2S_1S_2\cos\gamma = c^2 \tag{3.7}$$

By solving this quadratic equation system we have now simultaneously solved the point space re-section problem. In [23] six different methods of solving this equation system are presented. They were also implemented an tested giving results about the accuracy and robustness of the tested algorithms. In this case a method presented by a German mathematician called *Finsterwalder* yielded the best result. Therefore this exact solution method is also used in our calibration procedure.
We now give a short summary about the basic idea of Finsterwalder's solution which can be also read in [23].

Finsterwalder's solution

By changing the variables in Equation 3.5, 3.6 and 3.7 by

$$S_2 = uS_1, S_3 = vS_1$$

the three equations from above look like this:

$$S_1^2(u^2 + v^2 - 2uv\cos\alpha) = a^2 \tag{3.8}$$
$$S_1^2(1 + v^2 - 2v\cos\beta) = b^2 \tag{3.9}$$
$$S_1^2(1 + u^2 - 2u\cos\gamma) = c^2 \tag{3.10}$$

Therefore,

$$S_1^2 = \frac{a^2}{u^2 + v^2 - 2uv\cos\alpha} = \frac{b^2}{1 + v^2 - 2v\cos\beta} = \frac{c^2}{1 + u^2 - 2u\cos\gamma}$$

from which we can derive the following two equations

$$u^2 + \frac{b^2 - a^2}{b^2}v^2 - 2uv\cos\alpha + \frac{2a^2}{b^2}v\cos\beta - \frac{a^2}{b^2} = 0 \tag{3.11}$$
$$u^2 - \frac{c^2}{b^2}v^2 + 2v\frac{c^2}{b^2}\cos\beta - 2u\cos\gamma + \frac{b^2 - c^2}{b^2} = 0 \tag{3.12}$$

Now we introduce a new variable λ and multiply Equation 3.12 by it and add the result to Equation 3.11 to produce

$$Au^2 + 2Buv + Cv^2 + 2Du + 2Ev + F = 0 \tag{3.13}$$

where the coefficients depend on λ:

$$\begin{aligned}
A &= 1 + \lambda \\
B &= -\cos\alpha \\
C &= \frac{b^2 - a^2}{b^2} - \lambda\frac{c^2}{b^2} \\
D &= -\lambda\cos\gamma \\
E &= \left(\frac{a^2}{b^2} + \lambda\frac{c^2}{b^2}\right)\cos\beta \\
F &= \frac{-a^2}{b^2} + \lambda\left(\frac{b^2 - c^2}{b^2}\right)
\end{aligned}$$

Equation 3.13 is now considered as a quadratic equation in v. Solving for v, we have

$$\begin{aligned}
v_{large} &= \frac{-sgn(Bu + E)}{C}[|Bu + E| + \sqrt{(B^2 - AC)u^2 + 2(BE - CD)u + E^2 - CF}] \\
v_{small} &= \frac{C}{v_large}
\end{aligned} \tag{3.14}$$

Now we can ask, if a value for λ can be found which makes the term $(B^2 - AC)u^2 + 2(BE - CD)u + E^2 - CF$ a perfect square. In this case v can be expressed as a first order polynomial in terms of u. Geometrically that means that the solution to Equation 3.13 corresponds to two intersecting lines. Now the first order polynomial can be substituted back into Equation 3.11 or 3.12 which yields a quadratic equation that can be solved for u, and then using the value for u in the first order expression for v, a value for v can be determined. This calculation produces four

solutions since there are two first order expressions for v and two solutions for u in the resulting quadratic equation.
The value for λ which makes the term a perfect square now satisfies

$$(B^2 - AC)u^2 + 2(BE - CD)u + E^2 - CF = (up + q)^2$$

And so we can say,

$$\begin{aligned} B^2 - AC &= p^2 \\ BE - CD &= pq \\ E^2 - CF &= q^2 \end{aligned}$$

And because $p^2q^2 = (pq)^2$ we get,

$$(B^2 - AC)(E^2 - CF) = (BE - CD)^2$$

which represents a cubic equation for λ :

$$G\lambda^3 + H\lambda^2 + I\lambda + J = 0 \tag{3.15}$$

where

$$\begin{aligned} G &= c^2(c^2 \sin\beta - b^2 \sin\gamma) \\ H &= b^2(b^2 - a^2)\sin^2\gamma + c^2(c^2 + 2a^2)\sin^2\beta \\ &\quad +2b^2c^2(-1 + \cos\alpha\cos\beta\cos\gamma) \\ I &= b^2(b^2 - c^2)\sin^2\alpha + a^2(a^2 + 2c^2)\sin^2\beta \\ &\quad +2a^2b^2(-1 + \cos\alpha\cos\beta\cos\gamma) \\ J &= a^2(a^2\sin^2\beta - b^2\sin^2\alpha). \end{aligned}$$

By solving this equation for any root λ_0, p and q can be determined:

$$\begin{aligned} p &= \sqrt{B^2 - AC} \\ q &= sgn(BE - CD)\sqrt{E^2 - CF} \end{aligned}$$

Then we use these results in equation 3.14 to produce:

$$\begin{aligned} v &= \frac{-(Bu + E) \pm (pu + q)}{C} \\ &= \frac{-(B \mp p)u - (E \mp q)}{C} \\ &= um + n \end{aligned}$$

where

$$m = \frac{-B \pm p}{C}$$

and

$$n = \frac{-(E \mp q)}{C}$$

By substituting this back into Equation 3.12 and by simplifying there results

$$(b^2 - mc^2)u^2 + 2(c^2(\cos\beta - n)m - b^2\cos\gamma)u - c^2n^2 + 2c^2n\cos\beta + b^2 - c^2 = 0 \quad (3.16)$$

A numerically stable way to calculate the solutions for u is again to compute the smaller root in terms of the larger root:

$$\begin{aligned} u_{large} &= \frac{-sgn(B)}{A}\left[|B| + \sqrt{B^2 - AC}\right] \\ u_{small} &= \frac{C}{u_{large}} \end{aligned}$$

where

$$\begin{aligned} A &= b^2 - mc^2 \\ B &= c^2(\cos\beta - n)m - b^2\cos\gamma \\ C &= -c^2n^2 + 2c^2n\cos\beta + b^2 - c^2 \end{aligned}$$

Now that we have all solutions for u and v we can compute the four possible values for the distances S_1, S_2 and S_3 of the three object points.

3.3.2 Determination of the outer orientation

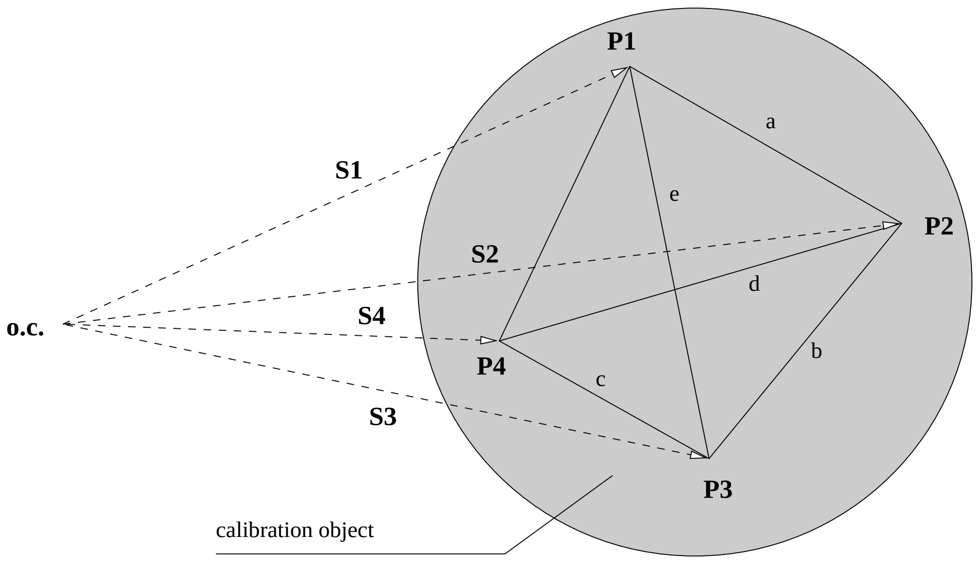


Figure 3.5: Estimation of the outer orientation by solving the three point re-section problem and using an additional fourth point

In the previous section we solved the problem of computing all possible positions of three already detected object points in front of the image plane. There can be up to four different

solutions for three points. But for our purpose we need to know where these object points are located in the scene space. To manage that we could use a fourth object point of which we know exactly the position relative to the other three object points. As depicted in Figure 3.5 we now have four vertices each consisting of the p.p. and two object points and a tetrahedron spanned by the object points themselves. For each side of the tetrahedron we now solve the three point re-section problem as described in the previous section. Each pair of adjacent vertices share a common side length S_i (i=1,..4) for which up to four different solutions per vertex were computed in the previous run. As we see in Figure 3.5 there are three adjacent vertices per side length S_i. So we have now up to twelve different solutions for S_i. By comparing the three tuples of up to four solutions for each vertex a quite robust estimation of the correct distance for each object point can be given in the following way:
Let A, B, C and D be the four sides of the tetrahedron spanned by the four object points P_1, P_2, P_3 and P_4.
Let $S_{i,j}(A)$ be the i-the solution of the distance of object point P_i computed by solving the three point re-section problem for the triangle A consisting of three object points including P_i.
Then we calculate an estimation for S_i by

$$S_i = median(S_{i,j_A}(A), S_{i,j_B}(B), S_{i,j_C}(C)) \tag{3.17}$$

where j_A, j_B, j_C are second order indices of the the tuple of solutions out of

$$\{(S_{i,j}(A), S_{i,k}(B), S_{i,l}(C))|i, j, k = 1, 2, 3, 4\}$$

for which

$$[S_{i,j}(A) - S_{i,k}(B)]^2 + [S_{i,k}(B) - S_{i,l}(C)]^2 + [S_{i,j}(A) - S_{i,l}(C)]^2$$

is minimal.
In practice this estimation has shown to be quite robust in terms of accuracy and reliability. Indeed we can extend the method introducing a fifth, a sixth or a any number of additional points in order to increase the robustness of the method. For further information about the numerical stability and accuracy of the selected algorithm read Appendix II and III in [23].

Now that we have the exact position of at least three object points of our calibration object we can compute the outer orientation of these points — that means the orientation of the plane spanned by the points relatively to the camera coordinate system described in previous sections — in the following way.
Let P_i' be the corresponding coordinates of P_i (object coordinates of the point) in the camera coordinate system. Then

$$P_i' = \begin{pmatrix} x_i' \\ y_i' \\ z_i' \end{pmatrix} = \begin{pmatrix} r_1 & r_2 & r_3 \\ r_4 & r_5 & r_6 \\ r_7 & r_8 & r_9 \end{pmatrix} \cdot \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix} + \begin{pmatrix} t_1 \\ t_2 \\ t_3 \end{pmatrix} = R \cdot P_i + \vec{t} \tag{3.18}$$

where P_i' is the point in the camera coordinate system, P_i the same point in the object coordinate system, R is the rotation matrix and T the translation vector (compare to 2.3.1 on page 21). In

[24] one can find a proof that the 9 parameters of the rotation matrix R are not independent. Indeed there exist some constraints as follows

$$r_1^2 + r_2^2 + r_3^2 = r_4^2 + r_5^2 + r_6^2 = r_7^2 + r_8^2 + r_9^2 = 1, \tag{3.19}$$

and

$$\begin{aligned} r_3 &= r_4 r_8 - r_5 r_9, && (3.20)\\ r_6 &= r_2 r_7 - r_1 r_8, && (3.21)\\ r_9 &= r_1 r_5 - r_2 r_4 && (3.22) \end{aligned}$$

Since any three points are coplanar and because of the constraints above we can assume that $z_i = 0, i = 1, 2, 3$. Therefore Equation 3.18 can be rewritten as

$$P_i' = \begin{pmatrix} x_i' \\ y_i' \\ z_i' \end{pmatrix} = \begin{pmatrix} r_1 & r_2 \\ r_4 & r_5 \\ r_7 & r_8 \end{pmatrix} \cdot \begin{pmatrix} x_i \\ y_i \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix} \tag{3.23}$$

By reducing the unknowns we now can set up an equation system that has a unique solution X for all the remaining unknowns as follows:

$$A \cdot X = B, \tag{3.24}$$

where

$$\begin{aligned} A &= \begin{pmatrix} x_1' & y_1' & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & x_1' & y_1' & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & x_1' & y_1' & 0 & 0 & 1 \\ x_2' & y_2' & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & x_2' & y_2' & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & x_2' & y_2' & 0 & 0 & 1 \\ x_3' & y_3' & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & x_3' & y_3' & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & x_3' & y_3' & 0 & 0 & 1 \end{pmatrix}, \\ X &= [r_1 r_2 r_4 r_5 r_7 r_8 t_x t_y t_z]^T, \\ B &= [x_1 y_1 z_1 x_2 y_2 z_2 x_3 y_3 z_3]^T. \end{aligned}$$

Now that we have all necessary values we can also compute the rest of the rotation parameters r_3, r_6 and r_9 by using the constraints formulated in the Equations 3.19 - 3.22.
The assumption that we get all object coordinate P_i in a way that all z-coordinates are zero requires an additional transformation in another coordinate system (where all three points are located in the xy-plane) before the described computation of the outer orientation can be done. It would also be appropriate to find a coordinate system that places one object point into the origin and a second one into the y-axis, since this would set a number of values in the equation system system to zero and therefore would increase numerical stability and accuracy. In Figure 3.5 this is schematically illustrated by the dotted lines x_0, y_0 and z_0 that represent the axis of such a coordinate system. In order to find a transformation matrix we can proceed in the following

way: We use homogeneous coordinates to describe the transformation from the original real world object coordinate system into the desired one by a single matrix C:

$$C = \begin{pmatrix} c_1 & c_2 & c_3 & c_x \\ c_4 & c_5 & c_6 & c_y \\ c_7 & c_8 & c_9 & c_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

where $c_1, ..., c_9$ denote the rotation and c_x, c_y, c_z the translation component. The coordinates for the solution of the equation system in 3.24 are computed by

$$\begin{pmatrix} x'_i \\ y'_i \\ z'_i \\ w_i \end{pmatrix} = C^{-1} \cdot \begin{pmatrix} x''_i \\ y''_i \\ z''_i \\ 1 \end{pmatrix}$$

where x''_i, y''_i, z''_i are the coordinates of the object points P_1, P_2 and P_3 in the original object coordinate system.
To place the origin in P_1 we set

$$\begin{pmatrix} c_x \\ c_y \\ c_z \end{pmatrix} = \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix}$$

Then defining the y-axis so that P_2 lies on it we set

$$\begin{pmatrix} c_2 \\ c_5 \\ c_8 \end{pmatrix} = \frac{P_2 - P_1}{\|P_2 - P_1\|}.$$

To make the z-axis perpendicular to the triangle we set

$$\begin{pmatrix} c_3 \\ c_6 \\ c_9 \end{pmatrix} = \frac{\begin{pmatrix} c_2 \\ c_5 \\ c_8 \end{pmatrix} \times (P_3 - P_1)}{\left\| \begin{pmatrix} c_2 \\ c_5 \\ c_8 \end{pmatrix} \times (P_3 - P_1) \right\|}$$

and to define the direction of x-axis:

$$\begin{pmatrix} c_1 \\ c_4 \\ c_7 \end{pmatrix} = \begin{pmatrix} c_2 \\ c_5 \\ c_8 \end{pmatrix} \times \begin{pmatrix} c_3 \\ c_6 \\ c_9 \end{pmatrix}$$

So putting everything together we now have the matrix

$$R_{final} = \begin{pmatrix} r_1 & r_2 & r_3 & t_x \\ r_4 & r_5 & r_6 & t_y \\ r_7 & r_8 & r_9 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot C^{-1} \tag{3.25}$$

describing the transformation of the original object coordinates of the object points into the camera coordinate system as previously defined.

3.4 Calibration parameters

In the previous section a method for estimating the extrinsic camera parameters was presented. Several calibration methods have been introduced for different purposes and requirements calibrating a subset of intrinsic parameters. We can presuppose that the vertical and horizontal resolution parameters are know. In fact we do not have to have exact values for them because they are linearly dependent to the focal length and the scaling factor (see [5] and Equation 2.34 on page 28).As a consequence, the value of the focal length only corresponds to its real world physical value, if resolution parameters are in compliance with the real world measurements.
As we discussed in Section 2.3, a basic distinction between an explicit and an implicit formulation of the projection features of a camera system can be made. In our case we only refer to a direct (explicit) model that describes these features by evaluating all necessary physical values to determine the projection. The pinhole model – translating the perspective projection directly – represents the ideal case of a perfect lens system with no distortion effects. Therefore, camera models basically differ in their perception and mathematical formulation of the nonlinear distortion effect that is the case in every real world camera projection. But also this kind of approach is just an approximation, since a camera lens can be de-formated in any possible way. Other camera parameters like resolution and the exact position of the principal point have also be taken into account, because they heavily depend on the distortion effect, too (the principal point is the only point in the image that is not distorted at all). To keep flexibility and to compare different kinds of camera models all of the three distortion models presented in Section 2.3.2 were implemented. Also the resolution uncertainty parameter S_x can be kept constant or variable depending on the kind of camera/sensor system that is used.
Table 3.1 sums up all possible distortion models for the camera models, that can be evaluated with this calibration software:

distortion parameters	focal length	S_x	(C_x, C_y)
radial and tangential: $\kappa_1, \kappa_2, \tau_1, \tau_2$	+	+/-	+/-
radial: κ_1, κ_2	+	+	+/-
Lenz's camera model: κ	+	+	+/-

Table 3.1: Parameters of the distortion model: the plus and minus signs in this table stand for the inclusion or exclusion (keeping it to a constant value) of the physical parameters in the calibration procedure.

3.5 Nonlinear estimation

The resulting pixel image coordinates (u_f, v_f) are then used to determine the difference to the points that where projected by the a real world CCD camera system and detected by some image processing algorithms. Mathematically we can now formulate an error function — also called the *merit function*.
For a set P of N calibration points and a projection g_C that depends on the calibration parameters

in C the residual is given by

$$\chi^2 = \sum_{i=1}^{N} \|p_i - g_C(P_i)\|^2 \tag{3.26}$$

where p_i is a point in real world image coordinates.(u_w, v_w) and P_i the corresponding coordinates in the 3D object (real world) coordinate system (x_w, y_w, z_w).

Our goal is to minimize χ^2 by variation of the values of the calibration parameters in C. Several optimization methods for this task can be used. Note that the function g_C is a non-linear projection. Therefore an iterative optimization technique for non-linear models like the *Levenberg-Marquardt method* (see 3.5 on page 47) has to be used.
In case we use a multi-image technique we have a number of camera models. For all V views an extrinsic model with parameters in R and $\vec{t}$ has to be determined. The intrinsic model is invariant for all views. We are using a set of functions $G = \{g_{C_i} : R^3 \to R^2 | i = 1..V\}$. So the *merit function* for the multi-view case is

$$\chi^2 = \sum_{j=1}^{V} \sum_{i=1}^{N} \|p_{ij} - g_{C_i}(P_j)\|^2, \tag{3.27}$$

where p_{ij} is the real world projected point of the 3D real world point P_j in the image of view v_i. For the implementation of the estimation process, that uses an optimization procedure that does not understand a model definition consisting of several functions, it is necessary to express the residuals as one function dependent on the parameter vector $\vec{x}$ as follows:

$$\chi^2(\vec{x}) = \sum_{j=1}^{V} \sum_{i=1}^{N} \|p_{ij} - g(P_j; \vec{x})\|^2. \tag{3.28}$$

where g is now a function of high dimensionality describing all outer orientations and the intrinsic camera parameters. Each additional view of the calibration object adds another 11 calibration parameters $(r_1...r_9, t_1, t_2, t_3)$ to the camera model. So the number of variants in the parametric fit is equal to 11N+I, where I is the number of intrinsic parameters. The optimization methods themselves use a set of M equations to fit the model into the given data set of M points p_{ij}. Obviously this equation system should be heavily over-determined so that the number of calibration points M should be considerably larger then 11N+I which is the dimension of the parameter vector $\vec{x}$. The disadvantage of that is that heavy increase of dimensionality in parameter space.
Since all camera models used are highly nonlinear we rely on heuristic search to find the most optimal parameter values for describing a specific camera system. The first step to solve this problem is to formulate an appropriate error function that has to be minimized to fit a set of adjustable camera parameters to a model, that summarizes the data given by the observations of the calibration object points. In Section 3.5 we have already given an error function χ^2 by Equation 3.28 (merit function), that evaluates a dimensionless value for the gap between the real world data points and the computed ones according to the estimated parameter values. In this case we do not search for a solution that is the global minimum of the error function but one that locally minimizes the error function within a certain region of the multi-dimensional

solution space. Limiting this region means to bound all parameters to certain intervals that are plausible. e.g. the focal length has to be within an interval of 30 to 80 mm, or the p.p. is located near the center of the image.
On the other hand by finding a set of appropriate start values for the outer and inner orientation of the camera system we define a starting vector X_0 for the following optimization procedure.
A classical approach to solve a nonlinear optimization problem is the *steepest descent method* (which, incidentally, goes back to Cauchy) where we start at the point X_0 and move from point X_i to the point X_{i+1} (i=0,...,N) along the line form X_i in the direction of the local downhill gradient $-\bigtriangledown \chi^2(X_i)$, as many times N as needed. In practice this method has shown to be a quite ineffective strategy, because every new gradient at the minimum point of any line minimization is perpendicular to the direction just traversed. This behavior causes an unacceptable amount of iterative steps in case of a narrow and long ”valley” in the error function, that is a huge gap between the steepness of two or more minimization variables, to reach the minimum.
A more efficient solution to the minimization problem is to prevent this situation by making the iteratively computed set of directions non-interfering, that means that minimization along one is not “spoiled” by subsequent minimization along another, so that interminable cycling through the set of directions can be avoided. Conventionally methods following this concept of non-interfering (conjugate) directions are called *conjugate gradient methods*.

Generally every multidimensional function $f(\vec{x})$ can be approximated at some particular point P by its Taylor series

$$f(\vec{x}) = f(\mathrm{P}) + \sum_{i=1}^{N} \frac{\partial f}{\partial x_i} + \frac{1}{2}\sum_{i,j=0}^{N} \frac{\partial^2 f}{\partial x_i \partial x_j} + ... \quad (3.29)$$

$$\approx c - b \cdot \vec{x} + \frac{1}{2}\vec{x} \cdot A \cdot \vec{x} \quad (3.30)$$

where

$$c \equiv f(\mathrm{P}), \quad b \equiv \nabla f|_{\mathbf{P}}, \quad [A]_{i,j} \equiv \left.\frac{\partial^2 f}{\partial x_i \partial x_j}\right|_{\mathbf{P}}$$

The matrix A whose components are the second partial derivative matrix of the function is called the *Hessian matrix* of the function at P. If we replace f(x) by the merit function χ^2 defined by our arbitrarily known camera model we can calculate the gradient components by

$$\beta_k \equiv -\frac{1}{2}\frac{\partial \chi^2}{\partial x_k}, \qquad k = 1, ...V \cdot N + C \quad (3.31)$$

and the components of the Hessian matrix

$$\alpha_{kl} \equiv -\frac{1}{2}\frac{\partial^2 \chi^2}{\partial x_k \partial x_l}, \qquad k, j = 1, ..., V \cdot N + C \quad (3.32)$$

where the rank of both is given by the number of views V, the number of extrinsic camera parameters N (6 or 11 depending on whether we use the Euler angles or a rotation matrix) and the number of intrinsic camera parameters C. Therefore, the derivation variables x_k are the

intrinsic and extrinsic camera parameters in the model.
The matrix $[\alpha] = \frac{1}{2}D$ in Equation 3.29 and is usually called the *curvature matrix*.
Using formula 3.29 the minimum can be found iteratively by

$$\vec{x}_{min} = \vec{x}_{cur} + D^{-1} \cdot \nabla\chi^2(\vec{x}_{cur}) \tag{3.33}$$

Rewriting this using the *curvature matrix* we get a set of linear equations

$$\sum_{l=1}^{V \cdot N+C} \alpha_{kl}\delta_{al} = \beta_k \tag{3.34}$$

This set is solved for the increments δ_{al} that, added to the current approximation x_{cur} gives the next approximation x_{min}.
This method is generally called the *inverse-Hessian method* and falls into the class of the *conjugate gradient methods*. Another way of iteratively finding the minimum point is the *steepest descent method* which only uses the first derivative term:

$$\delta_{al} = constant \times \beta_l \tag{3.35}$$

The problems in case of a narrow long "valley" of the *steepest descent method* we already discussed in a previous section. But also the *inverse-Hessian method* has its problems, because the inclusion of the second-derivative term can be destabilizing if the models fits badly (the value of the merit function is huge) or is contaminated by outlying points that are unlikely to be offset by compensating points of opposite sign.

Levenberg-Marquardt Method

To avoid the mal-behavior of both methods, Marquardt has put another elegant method, related to an earlier suggestion of Levenberg, for varying smoothly between the extremes of the inverse-Hessian method and the steepest descent method. There, the steepest descent method is used far from the minimum, switching continuously to the iterative approximation of the minimum by the inverse-Hessian method. In practice, this method has become the standard of nonlinear least-square problems (remember that the merit function computes a squared error).
The first consideration in this case is how the *constant* in Equation 3.35 should be quantified, even in order of magnitude. There is no information in the first derivative, but the Hessian matrix that gives at least some information about the order of magnitude.
Another question is drawn by considering the dimensionality of the components of the derivatives. The merit function χ^2 expresses a dimensionless value – a pure number. On the other hand, the components of the first-derivative term β_k have the reciprocal dimension of the original camera parameters x_k. Therefore, the constant of proportionality between β_k and x_k must have the dimensions of x_k^2. Examining the curvature matrix $[\alpha]$ only the reciprocals of its diagonal elements $1/a_{kk}$ possess the same dimensionality. Therefore this term must set the scale of the constant. In case also this scale is too big, we divide the constant by some (non-dimensional) fudge factor λ. Summing up we can replace Equation 3.35 by

$$\delta_{al} = \frac{1}{\lambda\alpha_{al}}\beta_l. \tag{3.36}$$

Secondly the Equations 3.36 and 3.34 can be combined if a new matrix $[\alpha']$ is defined by the following prescription

$$\begin{aligned} \alpha'_{jj} &\equiv \alpha_{jj}(1+\lambda) \quad (3.37) \\ \alpha'_{jk} &\equiv \alpha_{jk} \text{ and } j \neq k. \quad (3.38) \end{aligned}$$

Now we can replace Equation 3.34 and 3.36 by

$$\sum_{l=1}^{V \cdot N+C} \alpha'_{kl} \delta\alpha_l = \beta_k. \quad (3.39)$$

For a deeper understanding of the mechanism we can state that the matrix $[\alpha']$ is forced into being diagonally dominant, if the fudge factor λ is large. On the other hand if λ is small, then Equation 3.39 resembles more and more Equation 3.35.
Now Marquardt has given the following recipe:

- Compute $\chi^2(x_0)$
- Pick a modest value for λ – about 0.001
- †: Solve the linear equations of formula 3.39 for $\delta\vec{x}$ and evaluate $\chi^2(\vec{x}+\delta\vec{x})$.
- If $\chi^2(\vec{x}+\delta\vec{x}) \leq \chi^2(\vec{x})$, *increase* λ by a factor of 10 and go back to † or
- Else *decrease* λ by a factor of 10 and go back to †.

The iterative computation can be stopped when the value for χ^2 changes by an amount $\ll 1$ because further proceeding is generally wasteful and unnecessary since the minimum is at best only a statistical estimate of the parameters.
Once the minimum has been found, we can set $\lambda = 0$ and compute the matrix

$$[C] \equiv [\alpha]^{-1} \quad (3.40)$$

which contains the estimated *covariance matrix* of the standard errors in the fitted parameters in $\vec{x}$.

3.6 Computation of parameters for the inverse model

As discussed in Section 2.4.1 we need the inverse camera model to correct image distortion to get an adequate back-projection function. Several methods were proposed. In [19] the following approach is used to compute the corrected image points out of the distorted image points:

$$\vec{q_i} = \vec{q_d} - \delta(\vec{q_d} - \delta(\vec{q_d})) \quad (3.41)$$

where $\vec{q_i}$ is the vector for the corrected image coordinates, $\vec{q_d}$ the vector for distorted ones and $\delta : R^2 \rightarrow R^2$ the function of the distortion in image location $\vec{q}$. This method generally gives poor results with a residuals of about 0.1 pixels for typical lens distortion parameters according to [5].

A slightly different approach to the back-projection problem is the estimation of a model function determined by a set of non-physical or implicit parameters representing an inverse model. In [18] the following method is proposed to compute an accurate model function:

$$u_i = \frac{\sum\limits_{0\le j+k\le N} a_{jk}^{(1)} u_d^j v_d^k}{\sum\limits_{0\le j+k\le N} a_{jk}^{(3)} u_d^j v_d^k}, v_i = \frac{\sum\limits_{0\le j+k\le N} a_{jk}^{(2)} u_d^j v_d^k}{\sum\limits_{0\le j+k\le N} a_{jk}^{(3)} u_d^j v_d^k} \tag{3.42}$$

where (u_i, v_i) is a dense grid of of points in the image coordinate systems, N equals the order of the numerator and denominator polynomials of the fractions, (u_d, v_d) the corresponding distorted image coordinates and $\{a_{jk}^{(n)}\}$ is the set of model parameters for the inverse model.
(u_d, v_d) can be calculated by computing the distorted image coordinates according to the camera model described by Equation 2.36 on page 30. The grid of points should be large enough to result in an over-determined equation system defined by Equation 3.42 in order to solve for the parameters $a_{jk}^{(n)}, 1 \le n \le 3, 0 \le j+k \le N$. Obviously the required number of points is very high, since N should be at least be equal to the order of the polynomial in Equation 2.36 which is 5. Assuming N=5 we have 21 terms per set n of unknown parameters $a_{jk}^{(n)}$. Fortunately many of these parameters were found to be quite redundant by simulations ([5]). Indeed the following definition is sufficient to compensate for the distortion effect:

$$\begin{pmatrix} u_i \\ v_i \end{pmatrix} = \frac{1}{G} \begin{pmatrix} u'_d + u'_d(a_1 r_d^2 + a_2 r_d^4) + 2a_3 u'_d v'_d + a_4(r_d^2 + 2{u'_d}^2) \\ v'_d + v'_d(a_1 r_d^2 + a_2 r_d^4) + a_3(r_d^2 + 2{u'_d}^2) + 2a_4 u'_d v'_d \end{pmatrix} \tag{3.43}$$

$$\begin{aligned}
&\text{and} \\
G &= (a_5 r_d^2 + a_6 u'_d + a_7 v'_d + a_8) r_d^2 + 1 \\
&\text{where} \\
u'_d &= \frac{(u_d - C_x)}{d'_x s_x}, \\
v'_d &= \frac{(v_d - C_y)}{d'_y} \text{ and} \\
r_d &= \sqrt{{u'_d}^2 + {v'_d}^2}
\end{aligned}$$

The terms C_x, C_y, d'_x, d'_y and s_x stand for the intrinsic parameters described in Table 2.2 on page 29. Using this model we have reduced the number of parameters from 63 in Equation 3.42 to 8 ($\{a_1, a_2, \ldots, a_8\}$) for the inverse model. This greatly reduces computation effort without worsening results as it was proved by [5]. The parameters $a_1, \ldots, a_4$ of the inverse model resemble radial and tangential distortion parameters κ_1, κ_2, τ_1, τ_2.
In order to estimate inverse model parameters of Equation 3.43 we can solve the equation system iteratively using the *least square technique* or directly by projecting a number of additional virtual control points within the calibration plane. Experiments in [5] showed the number of these tie-points (u_{ii}, v_{ii}) and (u'_{di}, v'_{di}) is chosen best between 1000 and 2000 (40x40).
Now the following computation method will solve the problem:

Defining the following vectors and matrices . . .

$$\begin{aligned}
\vec{u_i} &= [-u_{d_i}' r_{d_i}^2, -u_{d_i}' r_{d_i}^4, -2u_{d_i}' v_{d_i}', -(r_{d_i}^2 + 2{u_{d_i}'}^2),\\
&\quad u_{ii} r_{d_i}^4, u_{ii} u_{d_i}' r_{d_i}^2, u_i v_d' r_d^2, u_i r_d^2]^T,\\
\vec{v_i} &= [-v_{d_i}' r_{d_i}^2, -v_{d_i}' r_{d_i}^4, -(r_{d_i}^2 + 2{v_{d_i}'}^2), -2u_{d_i}' v_{d_i}',\\
&\quad v_{ii} r_{d_i}^4, v_{ii} u_{d_i}' r_{d_i}^2, v_{ii} v_{d_i}' r_{d_i}^2, v_{ii} r_{d_i}^2]^T,\\
T &= [\vec{u_1}, \vec{v_2}, \ldots, \vec{u_i}, \vec{v_i}, \ldots, \vec{u_N}, \vec{v_N}]^T,\\
\vec{p} &= [a_1, a_2, \ldots, a_8]^T\\
\vec{e} &= [u_{d_1}' - u_{i1}, v_{d_1}' - v_{i1}, \ldots, u_{d_i}' - u_{ii}, v_{d_i}' - v_{ii}, \ldots, u_{d_N}' - u_{iN}, v_{d_N}' - v_{iN}]^T\\
i &= 1 \ldots N,
\end{aligned}$$

…the following relationship can be obtained by using eqn. 3.43:

$$\vec{e} = T\vec{p}$$

The vector $\vec{p}$ is now estimated in a least square sense:

$$\vec{p_{est}} = (T^T T)^{-1} T^T \vec{e}$$

The term $(T^T T)^{-1} T^T$ of this equation is also known as the *pseudoinverse* of the matrix T.

3.7 Determining start values

We have to determine a starting vector for the optimization process. Because of the complexity of the merit function there will be a number of local minimums. We are actually not looking for the smallest or *global* minimum, but one that is plausible. (e.g. the values for focal length or the uncertainty factor should not be magnitudes larger or smaller than expected). Because of the iterative nature of the proposed heuristic method, the initial parameter vector heavily determines the result (see [21]). In fact we expect or final estimation to be near the initial start vector. Assuming that the camera has in general a projection geometry that is not too different to the pinhole model a good guess for the start values is listed as follows:

- *focal length* f: is equal to the *nominal focal length* that can be read on the camera
- *distortion*: by setting all coefficients to zero we ignore the distortion effect at the start, assuming linear Euclidean perspective projection.
- *uncertainty factor* s_x: is initially set to one, assuming that the signal is perfectly synchronous, there is no difference between sensor resolution and resolution of the digital image.
- *principal point* (C_x, C_y): set in the center of the image frame

4 Test results

Due to different requirements in the history of this project several implementations of the presented calibration method were developed. In an early work a simulation for the Mathematica 4.0 (Wolfram Research) program was written. It showed some interesting results with respect to

the estimation of outer orientation by the Finsterwalder method. Unfortunately we had difficulties to get stable optimization results for the estimation of intrinsic parameters. Nevertheless the routines are available for practical use in conjunction with an additional Khoros environment that builds up a communication structure between the Mathematica Kernel and the Khoros cantata run time environment.
In order to establish an easy to use and portable calibration toolbox, all further programming was done with MATLAB. A simulation script as well as a real world calibration environment including grabbing and preprocessing of the image data was implemented, of which the most important experiments and results will be presented. In addition the MATLAB implementation was also made available for use with cantata (Khoros runtime environment) by implementing a kind of Khoros-to-MATLAB interface that establishes a glyph for starting and controlling a small-sized MATLAB-engine, and manages data exchange for both environments.

4.1 Tests with synthetic data in Mathematica^{TM}

First of all a hypothetic calibration object was determined. We have chosen a quite simple model, a 3x3 matrix of coplanar control points.

```
ControlPoints = {{-0.03,-0.03,0.}, {0.,-0.03,0.},{0.03,-0.03,0.},
                 {-0.03,0.,0.},{0.,0.,0.},{0.03,0.,0.},
                 {-0.03,0.03,0.},{0.,0.03,0.},{0.03,0.03,0.}};
```

Afterwards we computed a number of N=5 randomized outer orientations. The translation in x and y direction is in a range of ± 10 cm, in z direction between 0.5 m and 0.7 m. The rotation is limited to $\pm \Pi/4$ for every degree of freedom.

```
Translations = Table[{ Random[Real,{-0.1,0.1},2],
                       Random[Real,{-0.1,0.1},2],
                       Random[Real,{0.5,0.7},2]},{5}];
EulerAngles = Table[Random[Real,{-N[Pi]/4,N[Pi]/4},6],{5},{3}];
TransformationMatrices =
              Table[rtmat[Translations[[i]],EulerAngles[[i]]],{i,1,5}];
W0 ={0.,0.,0.}; T0 = {0.,0.,0.5};
Null= Table[rtmat[T0,W0]];
```

To create the images of the calibration object, we used a calibration setup taken from a real world CCD camera. These values are the ones that should be approximated by the calibration software.

```
        f=0.016;k1=60.;k2=300.; sx:=1.05;
        dx=0.0000086; dy = 0.0000083; Cx = 378.; Cy=291.;
        Params={f,k1,k2,sx,dx,dy,Cx,Cy};
```

The resulting projection is illustrated in Figure 4.1. The 9 control points are associated to the corner points of the squares.
In the next step we try to estimate the outer orientations of all views by Finsterwalder's algorithm. Therefore, we first have to define a set of vertices within the calibration plane together with a fourth reference point. We used the configuration illustrated in Figure 4.2. To simulate that we do not know the exact value for the focal length f at this point its value as input parameter for the optimization deviates slightly from its real value (about $\pm 5\%$) assuming a normal distribution.

```
OuterOrientations  = MultImagEstim[ImageCoords, ControlPoints ,
                                 NumOfPlanes,CalibP, Length[CalibP],
                                 Params,100.];
```

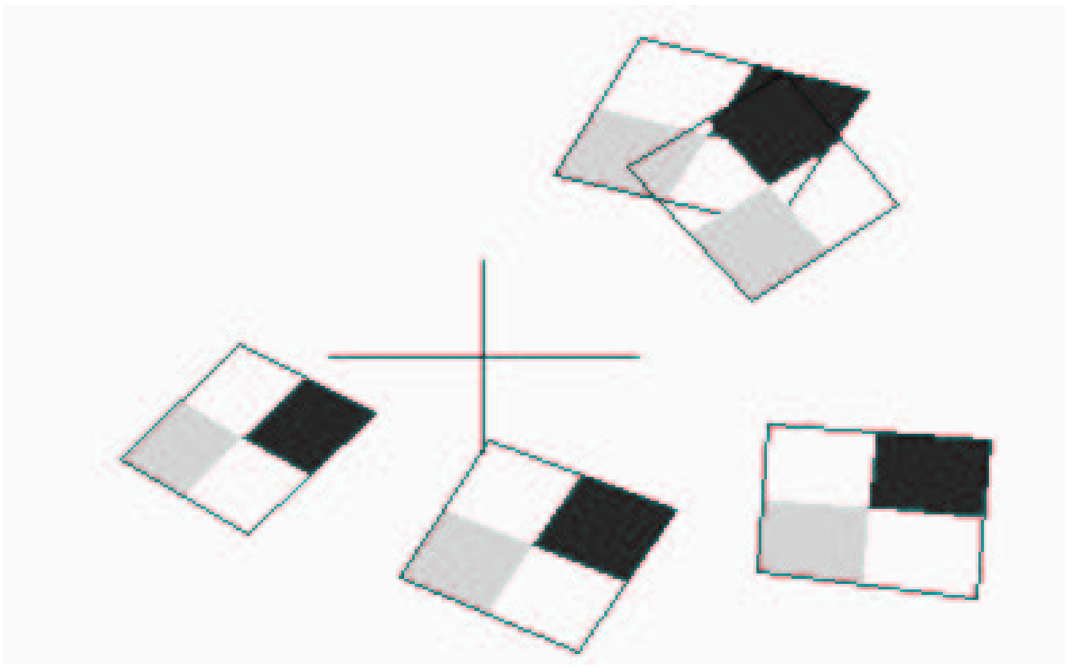


Figure 4.1: Projected image points of five views of the calibration plane

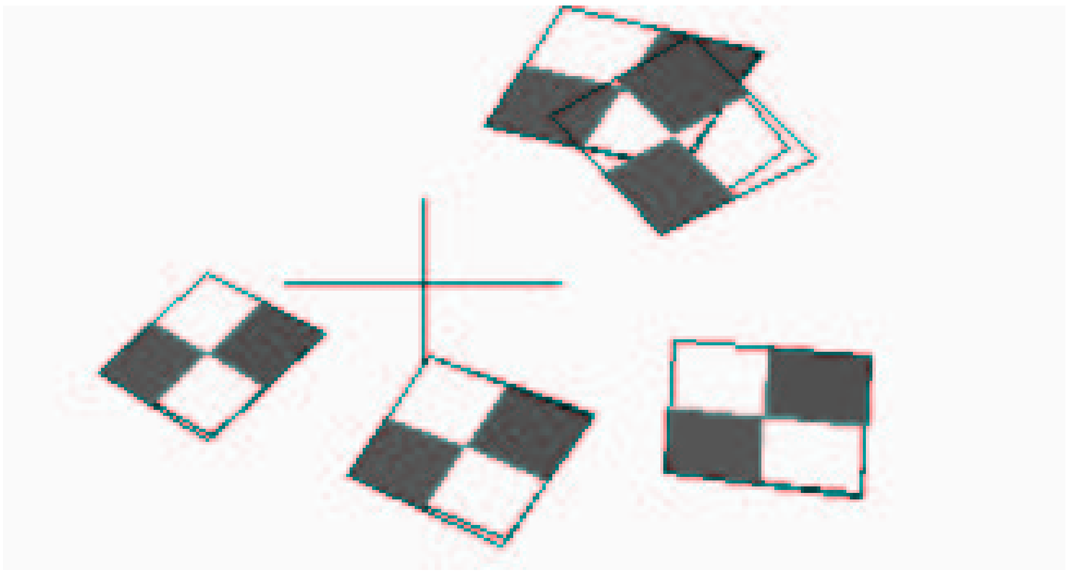


Figure 4.3: The resulting planes (orientations) and the original orientation

As we found out a maximal squared error of 100 guarantees a successful estimation procedure. This squared error is calculated by back-projection according to the determined extrinsic parameters. Figure 4.3 shows the result of an optimization run resulting in an squared error of 89.21.

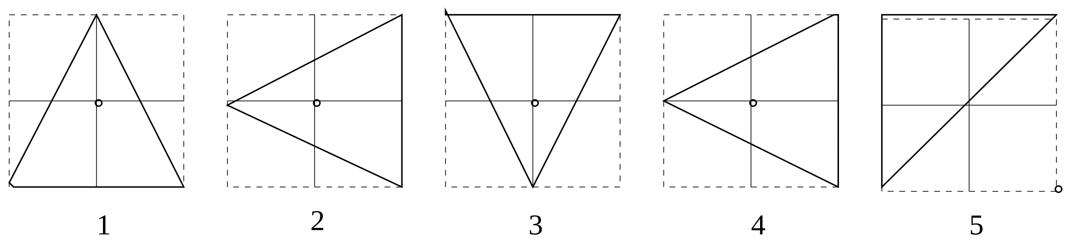


Figure 4.2: Five vertices together with their fourth reference point were chosen to get an estimation of the outer orientation. Since all points are coplanar the orientations of the planes defined by the vertices towards the abstract camera coordinate system has to be the same. The fourth reference point is depicted as a circle.

4.2 Tests with real world data using MATLAB

Though there also exists a simulation environment for the MATLAB implementation of the calibration routines, we skip the presentation of these results, because they are quite similar to those which were accomplished by real world experiments.
The setup for the experiments is described as follows:

- *Hardware:* For the acquisition of the images an of-the-shelf IMAC-S30 CCD camera was used. The analog signal was processed by a METEOR frame grabber card. Nominal focal length of the optics, the size and resolution of the CCD-sensor and the output resolution of the frame grabber card were known (see Table 4.1).

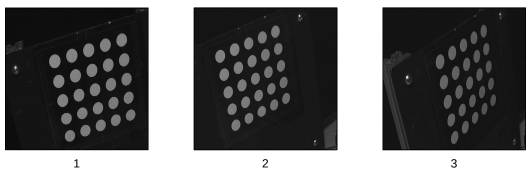
- *Calibration target, 3D table:* A simple plane with a black backdrop an bright white circle areas was used as calibration object. The dimensions of the circle area and its relative positions were measured. To avoid numerous artifacts and to simplify the extraction of the regions of interests (circle areas) the whole scene was dimmed out by photo-resist drapery. The range within images were taken can be roughly specified by 1x0.5x0.5 meters.

- *OS and Software:* The program was run on Linux machine using the MATLAB 6.2 engine. A third party driver for the frame grabber card was used that unfortunately delivered sometimes poor results (missing lines in frames, lower resolution than camera).

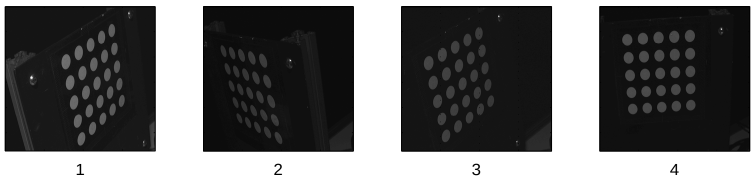
The chosen experiments are a representative example for the derivation of our most important conclusions that we draw from numerous others before. A set of 3, 4, 5 and 7 different images were chosen for comparison (see Figure 4.2) with the same camera at a fixed location.

nominal focal length:	16 mm	aspect ratio(Sx)	1.0
horizontal resolution:	sensor: 762 image: 768	vertical resolution:	sensor: 768 image: 576
sensor size		horizontal vertical	8.6 mm 6.6 mm

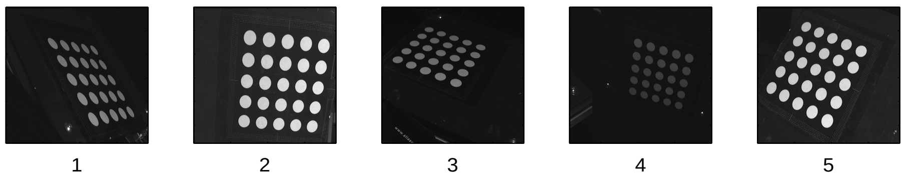
Table 4.1: Known camera and sensor parameters



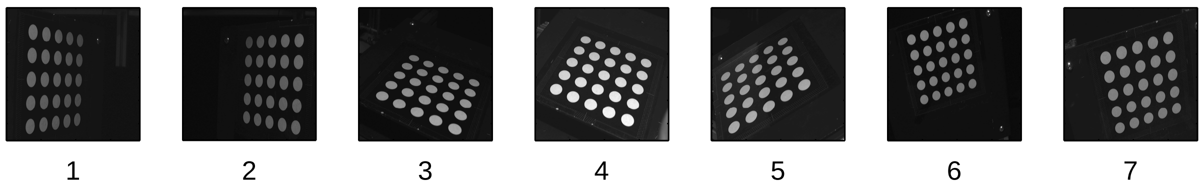
(a) Dataset 1



(b) Dataset 2



(c) Dataset 3



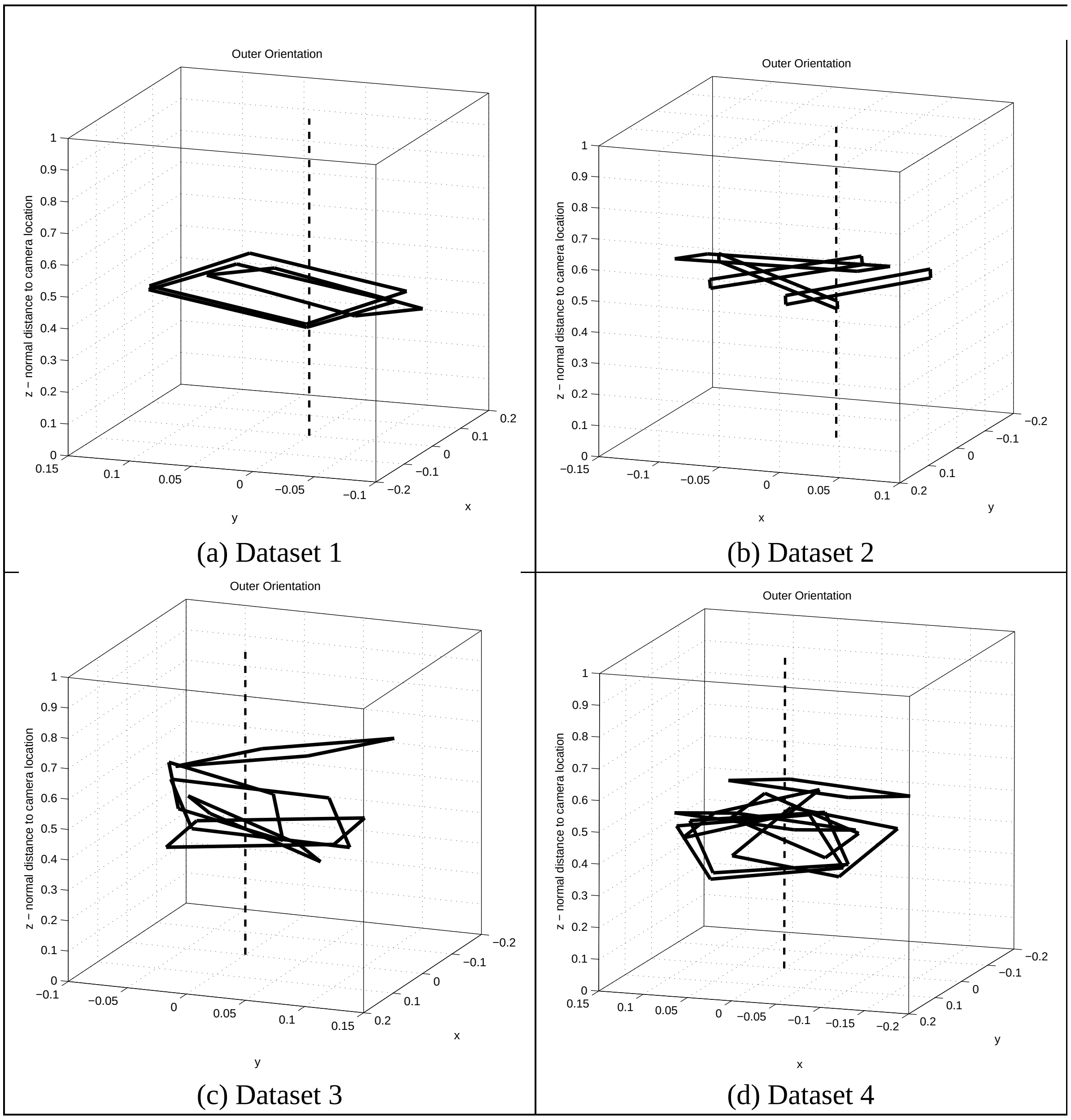
(d) Dataset 4

Figure 4.4: Sample of images for experiments - 4 datasets

With this four datasets and the a-priori known intrinsic parameters and additionally estimating the position of the principal point in the image center and neglecting distortion the following results were achieved:

4.2.1 Extrinsic parameters (Orientation)

In Figure 4.5 the computed orientations of the planes in the images is visualized per dataset. The line of sight is depicted as the dotted line, the position of the optical center of the camera lies in the origin of the xy-plane.



(a) Dataset 1 (b) Dataset 2

(c) Dataset 3 (d) Dataset 4

Figure 4.5: Orientations of the planes towards the camera position as computed by the calibration toolbox

The figures show that the images were taken in about the same range from the camera.

4.2.2 Unknown intrinsic parameters

In Table 4.2 the results for all 4 datasets are given. One may notice that some parameters like the principal point are more likely to deviate from other results than the real focal length or the radial distortion coefficients.

Dataset	1	2	3	4
S_x	1.0457	1.0452	1.0439	1.0508
f	17.1269	17.8233	17.3469	17.3575
C_x	355.5063	415.8188	377.1927	384.0515
C_y	344.4231	408.2650	292.9970	313.8536
κ_1	-0.0015	-0.0016	-0.0011	-0.0017
κ_2	1.4219e-05	5.0658e-06	-2.6071e-06	3.6567-05
τ_x	-1.5049e-05	-6.6540e-04	-1.2514e-04	-3.2130e-05
τ_y	2.2373e-04	-5.4902e-04	1.6888e-05	3.5260e-05
Error/point	0.395265	0.412533	0.223246	0.421950

Table 4.2: Computed intrinsic parameters and error after back-projection for the datasets. The average pixel error is 0.363302

Of course the table above is just a small sample of dozens of other experiments, but it gives an appropriate impression of the general features of the calibration method. The above results show that the calibration method is functionally correct and delivers a camera models that fits to the input data with sub-pixel accuracy. The focal length, the aspect ration (S_x) and first order radial distortion coefficient (κ_1) are less diversified than the coordinates of the principal point (C_x, C_y) and 2nd order radial, and tangential distortion coefficients. Because the latter parameters have generally very limited effects on the model itself, they can only be narrowed down properly by orders of magnitude. Additionally the model is normalized to the digital image resolution and not the sensor resolution, which would alter the value of the aspect ratio, the focal length and the principal point.

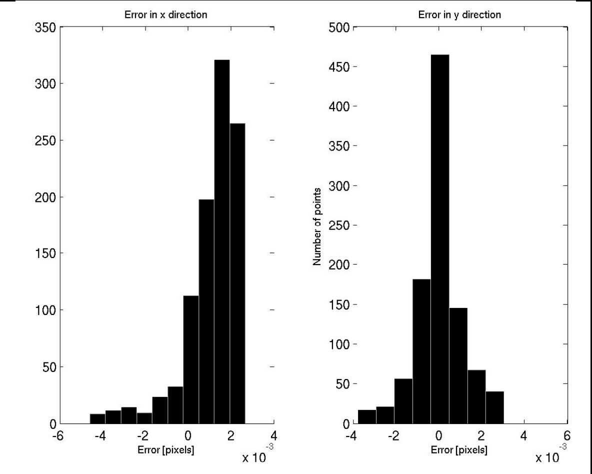
4.2.3 Inverse distortion parameters

After calibration of the real world camera parameter and an inverse model for distortion was computed using a normally distributed set of 1000 randomly chosen points. The parameters values are shown in Table 4.3. The resulting error histograms are shown in Figure 4.6.

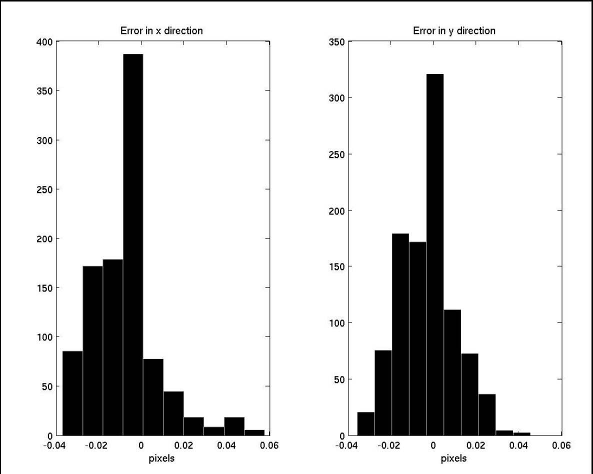
Dataset	1	2	3	4
a_1	-0.0033	-0.0029	-0.0023	-0.0036
a_2	0.0001	1.86e-05	-1.89e-05	1.58e-04
a_3	1.52e-05	6.71e-04	1.2440e-04	3.33e-05
a_4	-0.0002	0.0006	-1.68e-05	-3.64e-05
a_5	0.0001	2.33e-05	-2.13e-05	1.95e-04
a_6	1.19e-06	-4.01e-06	1.47e-07	2.76e-08
a_7	-1.26e-07	-5.66e-06	-1.11e-06	-9.32e-09
a_8	-0.0048	-0.0045	-0.0035	-0.0054

Table 4.3: Inverse distortion model: parameters of inverse distortion function defined by Equation 3.43

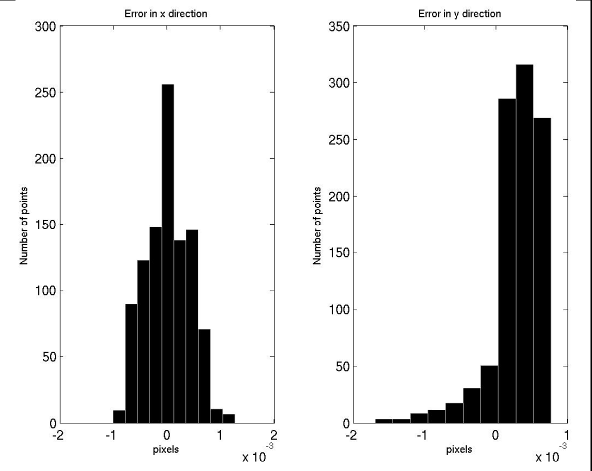
In all experiments the inverse models achieve sub pixel accuracy for all points. The histograms reach their peak around the zero error level as expected by using a normally distributed set of points. Despite of the parameters a_1 and a_8 all other parameters of the inverse model are diversified according to diversification of the original distortion model.



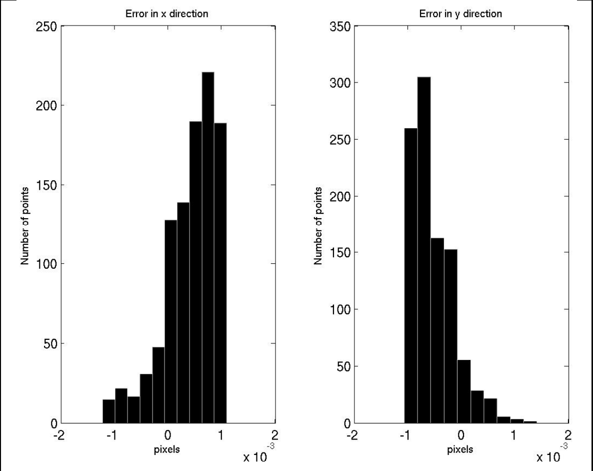
(a) Dataset 1



(b) Dataset 2



(c) Dataset 3



(d) Dataset 4

Figure 4.6: Inverse distortion model: histogram of errors by back-projection in x and y direction for 1000 randomly computed points distributed normally over the image area

4.2.4 Interpretation of experimental results

Considering the results of our experiments the following conclusions are drawn:

- The estimation of extrinsic parameters at the first stage of the calibration process is accurate enough to yield a good starting vector for the second stage optimization vector.
- The optimization results produce camera models that fit to selected input data images with sub-pixel accuracy. The values for the focal length, aspect ratio, position of the principal point and first order radial distortion are reliably estimated lying within about the same range. This indicates stability with respect to the physical model. Other model parameters are strongly diversified and are narrowed only in respect to their order of magnitude.
- The quality of the results does not depend on the number of images in the input data

set in general, but more obviously on the diversification of the images themselves. This is, because on taking several images of the calibration target we are virtually building a dataset of a three dimensional object of certain extent. This extent and the distribution of the control points within this 3 dimensional structure are the true quality determining factors for the resulting camera model. Therefore the camera model is always measured with respect to the range that is defined by the virtually created 3 d structure and can only be applied with expected accuracy on measurements and experiments that work within this range.

- In order to eliminate distortion effects for further processing an inverse camera model can be produced. The inverse models yield sub-pixel accuracy with great reliability. Despite of a_1 and a_8 the parameters of the inverse model are strongly diversified.

5 Conclusion and Outlook

In this work we presented an calibration technique that yields sufficiently accurate results for common purpose 3D vision tasks. The inverse distortion model it delivers is appropriate for eliminating distortion effects almost completely. A great advantage of this calibration method is its flexibility. Six degrees of freedom are allowed for the position of the calibration target. Also various calibration targets can be used, even objects with non-coplanar surfaces, which will be investigated in forthcoming experiments. Improvements should be made concerning the acquisition of the image data, which is of poor quality right now. In literature numerous methods for estimating focal length, principal point and distortion are presented. Some of these might be adequate to yield a more accurate start vector in the initial pre-optimization stage. Indeed, this will be a main issue in the forthcoming work.

References

[1] P. Kraus, K.and Waldhäusl, *Photogrammetry*, vol. 1 of *DÜMMLER-Books Geodetic Science*. Kaiserstrasse 31-37, 53113 Bonn: Ferd. Dümmlers Verlag, Germany, 4th ed., 1992.

[2] T. Mitsunaga and S. K. Nayar, “Radiometric self calibration,” in *Proc. of CVPR’99*, vol. 2, pp. 374–380, June 1999.

[3] R. Y. Tsai, “A versatile camera calibration technique for high accuracy 3d machine vision metrology using off-the-shelf tv cameras and lenses,” Tech. Rep. RC 51342, IBM, oct 1985.

[4] Z. Zhang, “A flexible new technique for camera calibration,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 11, pp. 1330–1334, 2000.

[5] J. Heikkila and O. Silven, “A four-step camera calibration procedure with implicit image correction,” in *IEEE Computer Society Conference on Computer Vision and Patern Recognition (CVPR’97)*, (San Juan, Puerto Rico), IEEE, 1997.

[6] H. Bakstein, “A complete dlt-based camera calibration with a virtual 3d calibration object,” Master’s thesis, Faculty of Mathematics and Physics, Charles University, Prague, 1999.

[7] X.-F. Zhang, A. Luo, W. Tao, and H. Burkhardt, “Camera calibration based on 3d-point-grid,” in *Proc. of ICIAP (1)*, pp. 636–643, 1997.

[8] G. Shivaram and G. Seetharaman, “Calibration of video cameras using spheres,” Tech. Rep. 98-1-1, Center for Advanced Computer Studies, University of Southwestern Louisiana, Lafayette, LA, U.S.A., February 1998.

[9] C. R. Wong K., Mendonca P., “Camera calibration from surfaces of revolution,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, pp. 147-161, February 2003.

[10] O. D. Faugeras, Q.-T. Luong, and S. J. Maybank, “Camera self-calibration: Theory and experiments,” in *Proc. of European Conference on Computer Vision*, pp. 321–334, 1992.

[11] M. Lourakis and R. Deriche, “Camera self-calibration using the kruppa equations and the svd of the fundamental matrix: The case of varying intrinsic parameters,” , Research Report, INRIA Sophia-Antipolis, 2000.

[12] M. Fleck, “Perspective projection: the wrong imaging model,” Tech. Rep. TR 95-01, Dept. of Computer Science, University of Iowa, 1995.

[13] S. Daniel and M. Fleck, “Nonparametric correction of distortion,”, Tech. Rep. TR 95-07, Dept. of Computer Science, University of Iowa, 1995.

[14] Y. I. Abdel-Aziz and H. M. Karara, “Direct linear transformation into object space coordinates in close-range photogrammetry,” in *Symposium on Close-Range Photogrammetry*, (Urbana, Illinois), pp. 1–18, University of Illinois, January 1971.

[15] T. Melen, *Geometrical modelling and calibration of video cameras for underwater navigation*. PhD thesis, Institutt for teknisk kybernetikk, Norges tekniske hogskole, 1994.

[16] F. Devernay and O. D. Faugeras, "Straight lines have to be straight," *Machine Vision and Applications*, vol. 13, no. 1, pp. 14–24, 2001.

[17] R. Lenz, "Linsenfehlerkorrigierte Eichung von Halbleiterkameras mit Standardobjektiven für hochgenaue 3D-Messungen in Echtzeit," in *Mustererkennung 1997* (E. Paulus, ed.), vol. 149, pp. 212–216, DAGM, Springer Verlage, 1987.

[18] G. Q. Wei and S. D. Ma, "A complete two-plane camera calibration method and experimental comparisons," in *4th International Conference on Computer Vision, pp. 234-257*.

[19] T. Melen, *Geometrical modelling and calibration of video cameras for underwater navigation*. PhD thesis, Norges tekniske hogskole, Institut for teknisk kybernetikk, 1993.

[20] R. Y. Tsai, "An efficient and accurate camera calibration technique for 3d-machine vision," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, (Miami Beach, Florida), pp. 364–374, IEEE, 1986.

[21] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C. The Art of Scientific Computing*, ch. 15. Cambridge University Press, 1996. ISBN 0-521-43108-5.

[22] R. Sedgewick, *Algorithms*. No. ISBN: 0.201-06673, Addison-Wesley, 2nd ed., 1992.

[23] R. M. Haralik, "Review and analysis of solutions of the three point perspective pose estimation problem," *International Journal of Computer Vision*, vol. 13, no. 3, pp. 123-168, 1994.

[24] S. Ganapaphy, "Decomposition of transformation matrices for robot vision," in *Proceedings of International Conference of Robotics and Automation*, pp. 130–139, 1984.

[25] A. E. Conrady, *Applied Optics and Optical Design*. Dover-Pubns, March 1992. ISBN 0-486-67007-4.

[26] S. Wolfram, *Mathematica book*. Addison-Wesley, 3rd ed., 1997. ISBN 3-8273-1036-9.

A Short description of the MATLAB toolbox

In order to make the implementation of the calibration method available for practical use a MATLAB script was written to adapt to the existing experimental setup. Its functionality spans from acquisition of the image data, extraction of control points to computation of the camera model and its inverse camera model. The functionality is determined by assign a string to the variable 'mode'. To abbreviate this, the more-or-less self explaining code of the script is printed below:

```
% definition of fileformat for center points

%roiextract.path = './work/work/';
%roiextract.filename = 'component_centers';
%roiextract.fileformat = '%d %f %f #%d points\n';

% apriori knowledge of intrinsic camera parameters

%cam.name = ('sony');
%cam.NDX = 768; cam.NDY = 576;
%cam.sx = 6.2031; cam.sy = 4.6514;
%cam.f = 8.5;

% definition of calibration target

%target.type = ('rectangle');
%N = 6; M=6; dn = 0.15; mindn = -0.30;
%dm = 0.15; mindm = -0.30;
%target.def = [N,M,dn,dm,mindn,mindm];
%target.world = compute_wpoints(target.type,target.def);

% simulation dataset
%numOfImages = 5;
DisplayAll = 1;
%MaximalAngle = [85,85,45];
%MaxTrans = [0.5,0.5,1.5];
%SetSeedIntrinsic = 0;
%SetSeedExtrinsic = 0;
%NoiseRate.FocalLength = 1;
%NoiseRate.Center = 0.8;
%NoiseRate.Asp = 0.001;


% options for precalibration
copts.est_extrinsic.method = 'finster';
copts.est_intrinsic.method = 'none';

if strcmp(mode,'init')
    par(1) = cam.NDX; par(2) = cam.NDY; par(3) = cam.sx; par(4) = cam.sy;
    par(5) = 1.0; % Aspect Ratio
    par(6) = cam.f;
    par(7) = cam.NDX./2; % Cpx
    par(8) = cam.NDY./2; % Cpy
    par(9:12) = 0.0;
    target.world = compute_wpoints(target.type,target.def);
elseif strcmp(mode,'add_centerpoints')
    if roiextract.path(length(roiextract.path))=='/'
        name = strcat(roiextract.path,roiextract.filename);
    else
        path = strcat(roiextract.path,'/');
        name = strcat(roiextract.path,roiextract.filename);
    end
    [n px py w] = textread(name,roiextract.fileformat);
    % filter
    j = 0;
    for i=1:length(n)
        if (w(i)>=roiextract.filtermin) & (w(i)<=roiextract.filtermax)
            j=j+1;
            images(imagenr,j,:) = [px(i) py(i)];
            numbers(imagenr,j) = [n(i)];
        end
    end
    numOfPoints = j;
    if (numOfPoints<length(target.world)) | (numOfPoints>length(target.world))
        error('Two few/many center points (%d) in image %d\n',numOfPoints,imagenr);
    end
elseif strcmp(mode,'calibrate')
    warning off MATLAB:nearlySingularMatrix;
    [iparamc,oparamc,epipoles,residual] = ...
        calibrate(par,target,images,DisplayAll,copts,nbr);
    pointError = sqrt(residual/sum(nbr));
 fprintf(1,'Absolute residual: %f\tError per point: %f\n', ...
             residual,pointError);
    intrinsic.Asp = iparamc(1);
    intrinsic.f = iparamc(2);
    intrinsic.Cx = iparamc(3);
    intrinsic.Cy = iparamc(4);;
    intrinsic.radial2 = iparamc(5);
    intrinsic.radial4 = iparamc(6);
    intrinsic.tangentialX = iparamc(7);
    intrinsic.tangentialY = iparamc(8);
    display(intrinsic);
elseif strcmp(mode,'showimages')
    numOfImagedata = size(img,3);
    figure;
    for i=1:numOfImagedata
        subplot(1,numOfImagedata,i);
        colormap(gray);
        imagesc(img(:,:,i));
        axis square
    end
elseif strcmp(mode,'inverse')
 a = invmodel(par,iparamc);
 fprintf(1,'Parameters for the inverse model:\n');
 display(a);

 % compute some random points
 r=[rand(1000,1)*768 rand(1000,1)*576];
 figure;
 plot(r(:,1),r(:,2),'rx');
 % Press enter
 pause
 clc
 d=imdist(par,iparamc,r);
 hold
 plot(d(:,1),d(:,2),'go');
 c=imcorr(par,iparamc,a,d);
 clf
 subplot(1,2,1)
 hist(r(:,1)-c(:,1))
 title('Error in x direction')
 xlabel('pixels')
 subplot(1,2,2)
 hist(r(:,2)-c(:,2))
 title('Error in y direction')
 xlabel('pixels')
elseif strcmp(mode,'grab')grabopts.ctrlcmd ='/opt/meteor/bin/vctrl';
    grabopts.resolutionX = '768';
    grabopts.resolutionY = '572';
    grabopts.colordepth = '24';
    grabopts.composite = '-p -i 0';
    grabopts.grabcmd ='/opt/meteor/bin/vcat';
    grabopts.rawfile ='$HOME/work/work/image.raw';
    grabopts.ppmfile ='./image.ppm';
    grabopts.convcmd ='/usr/bin/rawtoppm -bgr';
    display(grabopts);
    R = input('Select composite 1 or 2 [1]','s');
    if strcmp(R,'2')
        grabopts.composite = '-p -i 0';;
    else
        grabopts.composite = '-p -i 1';
    end
    notDone = 1; i = 0;
    while (notDone)
        i = i + 1;
        grabopts.changectrl = (i==1);
        img(:,:,i) = grab_image(grabopts);
        colormap(gray);
        R = sprintf('Image %d',i);
        xlabel(R);
        imagesc(img(:,:,i));
        axis square;
        fprintf(1,'Current # of frames: %d\n',i);
        R = input('Keep image? (y/n)','s');
        if strcmp(R,'n')
            i = i - 1;
        else
            R = input('Grab another image? (y/n)','s');
            notDone = strcmp(R,'y');
        end
    end
    numOfImages = i;
    fprintf(1,'%d images are kept.\n');
elseif strcmp(mode,'extract')
    if strcmp(target.type,'rectangle')
        numOfPoints = target.def(1) * target.def(2);
    elseif strcmp(target.type,'square')
        numOfPoints = target.def(1) * target.def(1);
    end
    [images(:,:,:),nbr] = roi_extract(img,numOfPoints);
end
```

B Manual for the Mathematica package

B.1 Computation of the rotation matrix, translation vector and the Euler angles

1. `yaw[q_Real]->Matrix[[4,4]]:` computes a 4x4 matrix describing a rotation around the x axis by the angle of q (in radians).

2. `pitch[q_Real]->Matrix[[4,4]]:` computes a 4x4 matrix describing a rotation around the y axis by the angle of q (in radians).

3. `tilt[q_Real]->Matrix[[4,4]]:` computes a 4x4 matrix describing a rotation around the z axis by the angle of q (in radians).

4. `transl[x_Integer,y_Integer,z_Integer]->Matrix[[4,4]]:` computes a 4x4 matrix describing a translation along a vector (x,y,z).

5. `rmat[rx_Real,ry_Real,rz_Real]->Matrix[[4,4]]:` computes a 4x4 matrix describing a rotation around the x, y and z axis by the angles of rx, ry and rz.

6. `rtmat[T_VectorQ,rx_Real,ry_Real,rz_Real]->Matrix[[4,4]]:` computes a 4x4 matrix describing a rotation around the x,y and z axis (by rx, ry and rz) and simultaneously a translation along a vector T.

7. `EulerAngles[R_MatrixQ]->{yaw,pitch,tilt}:` computes the Euler angles yaw, pitch and tilt by a given 4x4 rotation matrix including also a translation vector in the last column.

B.2 Further geometric functions

1. `SqrErr[{u1_,v1_},{u2_,v2_}]->Real:` computes the squared distance of two 2dimensional points.

2. `sqrdist3d [x1_,y1_,z1_,x2_,y2_,z2_]->Real:` returns the value for the squared distance of two 3dimensional points (x1, y1, z1) and (x2, y2, z2).

3. `VectorLength[x_VectorQ]->Real:` computes the length of a vector of any dimensionality

4. `Norm[x_VectorQ]->VectorQ:` returns the unit vector of a given vector x of any dimensionality.

5. `UnitVector[u_,v_,focus_]->{x_Real,y_Real,z_Real}:` computes the 3dimensional unit vector (x,y,z) of the viewing ray of a given image point (u,v) according to a given focus (focal length) describing a the projection.

6. `cosimg[P1_,P2_,focus_]->Real:` computes the angle between the viewing rays of two given 2dimensional points P1 and P2 reversing the projection given by the focus (focal length).

7. `finster[P1_,P2_,P3_,I1_,I2_,I3_,f_Real]->List[L1,L2,...]:` computes all possible spatial orientations (the distances to the focal center) of the three 3dimensional object points P1, P2 and P3 in front the image plane given their 2dimensional image coordinates I1, I2 and I3 and the focal length f, which defines the perspective projection, using the pinhole camera model. The solutions (there can be up to four) are returned as a list of up to four 3dimensional vectors L1, L2, L3 and L4. If there is less than four solutions also less than four vectors are included (that means that the number of elements in this list can vary).

8. `ObjCoordSys[{Pw1_,Pw2_,Pw3_,Pw4_}]->Matrix[[4,4]]:` returns a 4x4 transformation matrix from the given coordinate system of Pw1, Pw2 ... Pw4 to a coordinate system where Pw1 is identical with the origin, Pw2 lies on the y axis and Pw3 is within the xy plane.

B.3 Other functions

1. `SqrdErrSum[a_Real,b_Real,c_Real]->Real:` computes the sum of all squared differences of the values a, b and c.

2. `Lcube[d1_,d2_,d3_]->Real:` returns the median value of the triple out of all possible triples consisting of one element of each of the lists d1, d2 and d3, that has the minimal sum of squared differences between the three values.

B.4 Determination of the Outer orientation

1. `FourPointPoseEstim[{P1_,P2_,P3_,P4_},{I1_,I2_,I3_,I4},f_Real]->{d1,d2,d3,d4}:` returns the estimated distances d1, d2, d3 and d4 of four object points P1, P2, P3 and P4 from the focal point given the 2dimensional image points I1, I2, I3 and I4 and the focal length f of the projection.

2. `OuterOrientation[{P1_,P2_,P3_},{I1_,I2_,I3_},{S1_,S2_,S3_},f_Real]-> Matrix[[4,4]]:` returns the 4x4 transformation matrix from the coordinate system of three 3dimensional object points P1, P2 and P3 to the coordinate system defining the optical of the projection as the z axis and the projection plane as the xy plane. The parameters that have to be given are the 2dimensional image coordinates I1, I2 and I3 of the projected points, their distances from the focus S1, S2 and S3 and the focal length f, defining the perspective projection.

3. `TransCoord[P3d_,RT_]->{x,y,z}:` returns the 3D coordinates of the 3dimensional point P3d in a coordinate system defined by the 4x4 transformation matrix RT.

B.5 Camera model

1. `pinhole[{x_,y_,z_},f_Real]->{u,v}:` returns the projected image point (u,v) of a given 3D point (x,y,z) of perspective projection defined by f according to the pinhole camera model.

2. `distortion[{x_,y_},k1_Real,k2_Real]->{xd,yd}:` returns the distorted image point (xd,yd) of a given 2dimensional image point (x,y) according to the radial distortion parameters k1 and k2.

3. `scaling[{xd_,yd_},Sx_Real,Dx_Real,Dy_Real,Cx_,Cy_]->{X,Y}:` transforms the physical (optical) image coordinates (xd,yd) into the digital (machine depended) pixel coordinates (X,Y). The naming of the input parameters refers to that used previous sections.

4. `invScaling[P2d_,{Sx_Real,Dx_Real,Dy_Real,Cx_,Cy_]->{x,y}:` This is the inverse function of the previous one.

5. `project[{x_,y_,z_},{f_Real,k1_Real,k2_Real,Sx_Real,Dx_Real, Dy_Real,Cx_,Cy_,RT_MatrixQ]->(X,Y):` computes the pixel coordinates (X,Y) of the 3dimensional point (x,y,z) in the world coordinate system of the object according to the intrinsic camera parameters given by f, k1, k2, Sx, Dx, Dy, Cx, Cy and the extrinsic ones given by the transformation matrix RT. This routine simulates the hole camera model.

B.6 Estimation of start values for extrinsic camera parameters

1. `ImagEstim[PunktePK_,PunkteWK_,CalibP_,Params_]->{R1_Matrix[[4,4]], R2_Matrix[[4,4]],EA_Vector[[3]],TV_Vector[[3]]}:`computes the estimated outer orientation of the plane spanned by the first three points of CalibP. CalibP must contain at least four points (their indices) to compute an exact solution. PunktePK contains the pixel coordinates, PunkteWK the world coordinates of the points. Param is the set of start parameters that is used for the defined intrinsic camera model.
   The return values contain the parameters R2 that is the acctually computed outer orientation as a 4x4 rotation/translation matrix and R1 that is the transformation matrix normalizing the object

coordinate system. EA contains the 3 euler angles and TV the translation vector as they are defined by the transformation matrix R2.

2. `MultImagEstim[PunktePK_,PunkteWK_,NumOfPlanes_,TestP_, NumOfCalib_,Params_, MaxErr_]->List[NumOfPlanes,4]:` computes the estimated outer orientations of serval images of the calibration object defined by the 3D world coordinates in PunkteWK. PunktePK is a set of NumOfPlanes sets of the coresponding image coordinates (the numbering has to be consistent!). TestP is a set of NumOfCalib sets of at least four points (their indices in the point sets) for each of them the first three points are used as the reference coordinate system to compute an outer orientation (this is processed by the invocation of ImageEstim). By the assumption that all points in TestP are located in a single plane a robust estimation over all computed orientations per subset can be made to get the most likely values for the outer orientation of the calibration points in one image can be done. This process is used for all NumOfPlanes images to compute their extrinsic camera parameters. MaxErr defines the maximum residual error that is allowed by the estimation process.
   The function returns a list of extrinsic parameters for each image. The form of the return values in a parameter set is equal to the form in ImagEstim.

3. `BestFit[PunktePK_,PunkteWK_,Results_,Params_,MaxErr_]->{R1,R2,EA,TV}:` returns the optimal result out of results for the best that fits best in the model defined by the intrinsic camera parameters params and world and image coordinates of the calibration points. MaxErr defines the maximal tolerable residual of this fitting procedure. The form of the return values is equal to the ones returned by ImagEstim that accually invokes this procedure to produce a result.

B.7 Optimization

1. `ParameterEstim[PunkteWK_,NumOfPoints_,PunktePK_,RT_, NumOfPlanes_,Params_,Vars_,MaxIt_]-> List`: estimates the selected intrinsic camera parameters listed in Vars and returns them in the form $\{\{Var_1 \rightarrow Value_1\}, \{Var_2 \rightarrow Value_2\}, \ldots, \{Var_n \rightarrow Value_n\}\}$. The input parameters are the world coordinates PunkteWK of the NumOfPoints calibration points, their pixel image coordinates PunktePK in each of the NumOfPlanes images, the start values for all intrinsic camera parameters in the form $\{\{Par_1, Value_1\}, \ldots, \{Par_m, Value_m\}\}$, the start values for the extrinsic camera parameters RT as a list of 4x4 homogene transformation matrices and the maximum number of iterations MaxIt during the optimization procedure (Levenberg-Marquardt).

2. `MultiImagCalib[PunkteWK_,NumOfPoints_,PunktePK_, NumOfPlanes_,TestP_,Constants_,StartVars_,MaxErr_,MaxIt_]->List:` does basically the same as ParameterEstim. Additionally it also invokes MultiImageEstim with the parameters PunkteWK, NumOfPoints, PunktePK, NumOfPlanes and TestP to get the start values for the extrinsic camera model so the are not included in the input parameters. StartVars contains a list of the start values of all intrinsic camera parameter in the form of which is used in the function ParameterEstim for Vars. Constants is a list of intrinsic parameters that are actually no calibration parameters. MaxErr refers to the parameter in function MultiImagEstim and MaxIt to the one in ParameterEstim.

3. `SuccMultiImagCalib[PunkteWK_,NumOfPoints_,PunktePK_, NumOfPlanes_,TestP_,StartVars_,Succ_,MaxErr_,MaxIt_]->List:` estimates a subset of the intrinsic camera parameters. The optimization can be done in consecutive steps

using different subsets of intrinsic camera parameters defined Succ that is a list of sets of camera parameter symbols (f, sx, sy, dx, dy, k1, k2, Cx, Cy) . PunkteWK, NumOfPoints, PunktePK, NumOfPlanes, TestP and StartVars refer to the corresponding parameters in MultiImageCalib that is invoked by the procedure.

B.8 Package installation and using

All functions of the package are organized in so called *m-files*. These files include a Master m-file called *Master.m* file that declares all exportable functions and constants.

Installation To install the *m-files* of the *Calibration* package into your Mathematica system you first have to make a new directory named *Calibration* within the directiory you normally start the Mathematica kernel or in the Mathematica *AddOns|Autoload* directory if you want to load the package every time the kernel gets started or in the Mathematica *AddOns|ExtraPackages* directory. Then untar all *m-files* of the package and copy them to the previously created directory.

Load package into kernel To use the Mathematica function of the *Calibration* package you first have to load the package into the kernel and set the context path. This is done by the command

```
Get[Calibration] <CR> or
<<Calibration         or
Needs[Calibration] <CR>

in case you are not sure if the package was already loaded.
```

This loads all functions of the package and sets the context path to directory where the package files are copied. To load only a single function e. g. *SuccMultiImagCalib* you type

```
Get[Calibration‘SuccMultiImagCalib‘] or
<< Calibration‘SuccMultiImagCalib‘   or
Needs[Calibration‘SuccMultiImagCalib‘]
```

Now you can type in

```
$ContextPath <CR>
```

to see if the the package context was actually set.

Using the calibration commands To evaluate the result of a function by the given input parameters in the notebook click into the corresponding cell and press CRTL + Enter. No the cell will be evaluated producing occasionally different types of output cells. For further details refer to [26] Normally it only makes sense to you use the high level optimization functions for the purpose of calibration. These functions require can require a huge amount of input data. So it is quite unhandy to type in all input data manually in fact importing and exporting ASCII data files would be a more appropriate solution to that. For further details refer to [26].

B.9 Used standard packages and functions

Statistics‘NonlinearFit‘ implementation of the Levenberg-Marquardt optimization algorithm.

Statistics‘DescriptiveStatistics‘ several statistical functions like median and StandardDeviation.

C Manual for Khoros toolbox

C.1 General description

This calibration toolbox was designed to integrate the Mathematica package into the Khoros system. For this purpose, we have written the tasks *GetWorld* , *SetParameters* and *SetConfiguration*, that enables the user to control the input parameters of the calibration procedure interactively without editing text files. The generated output files of these tasks can be submitted to *MCalib* that finally processes the communication with the Mathematica kernel by loading the package and by the construction of the Mathematica commands needed to invoke the calibration process. After invocation the program waits for the final result from the kernel and translates the return package into the specified output file.

C.2 Installation remarks

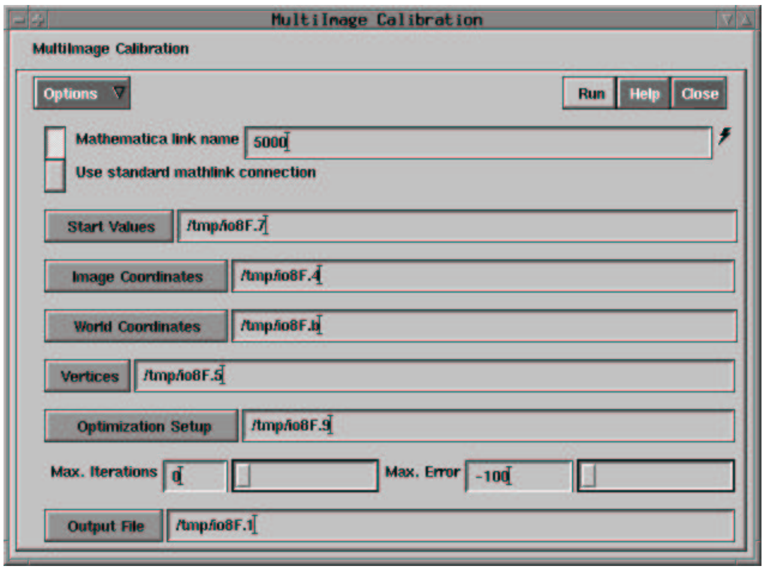
First it is necessary that the *Mathlink* C library is included in the local Mathematica installation. It contains all functions and data structures for communication with the Kernel and is used by *MCalib*. Then, if there are no binary already available and running, the tasks have to be compiled and linked. For this purpose have to specify the additional library path for the *Mathlink* library. This is done by editing the *Imake configuration file, toolbox.def*. Now, if for example the Mathlink path is */usr/local/mathlink* you can add the following line to configuration file,

```
/* -toolbox_library_path */
        TOOLBOX_LIBDIR +<= -L$(MULTICALIB_LIBDIR)
        TOOLBOX_LIBDIR +<= -L/usr/local/mathlink
/* -toolbox_library_path_end */
```

which makes this directory all search path for the toolbox libraries. After creating the imake and make files, the compilation of all tasks should work normally.

C.3 Description of the tasks

C.3.1 MultiCalib



Mathematica link name: 5000
Use standard mathlink connection
Start Values: /tmp/io8F.7
Image Coordinates: /tmp/io8F.4
World Coordinates: /tmp/io8F.b
Vertices: /tmp/io8F.5
Optimization Setup: /tmp/io8F.9
Max. Iterations: 0
Max. Error: -100
Output File: /tmp/io8F.1

Figure C.1: Pane of the MultiImageCalibration Task MCalib

General description This tasks communicates with the Mathematica kernel by sending commands for loading the calibration package and invoking the calibration routine with the desired input parameters. All input parameters are expected to be in ASCII format as described in Appendix C.4.

Input parameters

Mathematica link The mutual exclusive group in the upper half of the pane determines the link where the data between Khoros and Mathematica is exchanged. Giving the name explicitly enables the user to control and manipulate the data flow by the Mathematica front end (see Notebook file *communciation.nb*). In this case the Mathematica Front End should already be running or invoked during the calibration procedure. If you do not want to do this, select the flag for the standard mathlink connection. This will start the Mathematica kernel automatically in the background opening a pre-defined standard mathlink connection. Mathlink connections are generally based on a TCP like socket layer which enables the use of remote hosts, so that the kernel does not need to be started on the local machine.

Max. iterations determines the maximal number of iterations in the *Levenberg-Marquardt* optimization procedure.

Max. Error determines the maxmial error for the residual of the *Levenberg-Marquardt* that is will be accepted as final result.

Input files

Start Values defines the start values for the intrinsic camera parameters. (see Appendix C.4)

Image coordinates contains all detected calibration objects points and its image coordinates for each image. (see Section C.4

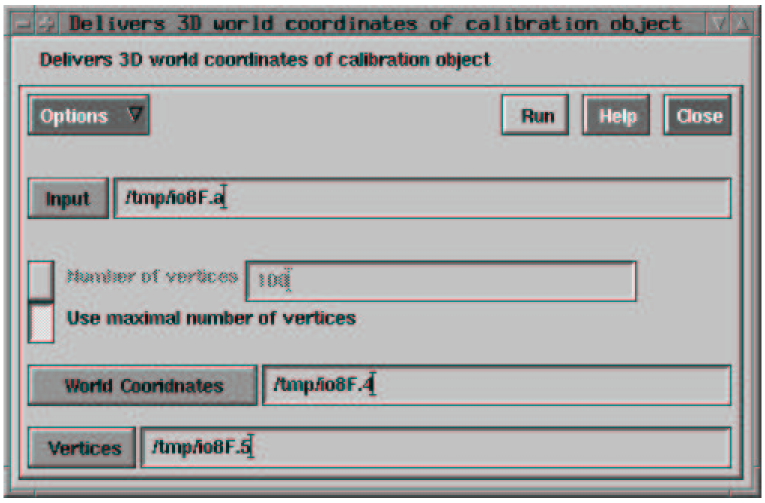
World coordinates contains all real world 3D coordinates of the calibration object points.

Vertices defines the sets of calibration points (the points numbered in the world coordinate specification file) according to them the outer orientation of the calibration object is computed by the *Finsterwalder's algorithm*. (see Appendix C.4)

Optimization Setup defines the parameters that have to be optimized by the calibration task. They are specified by a list that contains some elements of the set {*f, sx, k1, k2, tx, ty, Cx, Cy* }. (see Appendix C.4)

Output file contains the optimized intrinsic camera parameters. (see Appendix C.4)

C.3.2 GetWorld


Delivers 3D world coordinates of calibration object

Delivers 3D world coordinates of calibration object

Input /tmp/io8F.a

Number of vertices 100

Use maximal number of vertices

World Coordinates /tmp/io8F.4

Vertices /tmp/io8F.5

Figure C.2: Pane of the Calibration Object Geometry Task GetWorld

General description delivers the 3D world coordinates and the vertices suitable for *Finsterwalder's* optimization of the calibration object points.

Input parameters

Number of vertices specifies the number of vertices that are used for the estimation of the outer orientation of the calibration object in each frame. This number is bounded by the maximal number of vertices that is possible. In this case this has the same effect as enabling the second flag 'Use maximal number of vertices. The two flags are mutually exclusive of course.

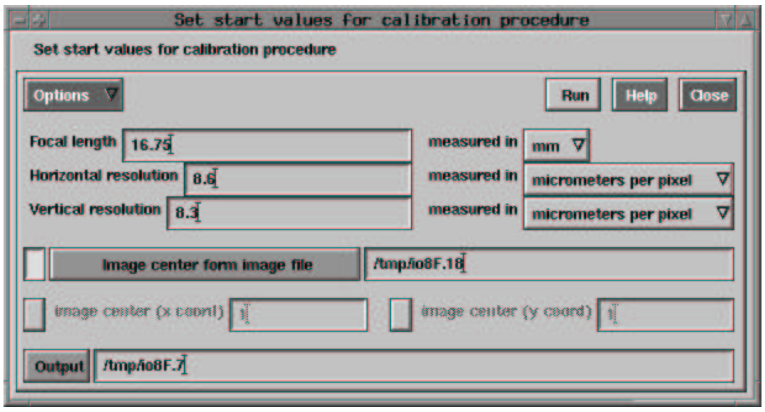
Input file Contains the world and image coordinates of all calibraiton points in all frames. This file is expected to be delivered by the point detection task. (see Section 3.2 on page 32

Output files

World Coordinates contains all world coordinates for the calibration points (see Appendix C.4)

Vertices contains a number of lists prepresenting different vertices spanned by the calibration points. (see Appendix C.4)

C.3.3 SetParameters


Set start values for calibration procedure

Set start values for calibration procedure

Focal length 16.75 measured in mm
Horizontal resolution 8.6 measured in micrometers per pixel
Vertical resolution 8.3 measured in micrometers per pixel
Image center form image file /tmp/io8F.18
image center (x coord) 1
image center (y coord) 1
Output /tmp/io8F.7

Figure C.3: Pane of the Parameter Set Task Set Parameters

General description This glyph is a tool for determining start values for the intrinsic camera parameters. It produces an output file that is compatible with the input file structure of the Start Values input file of *MCalib*

Input parameters

Focal length defines the start value for the focal length in the calibration procedure. The selection on the right hand side of the input field specifices the dimension.

Horizontal resolution defines the start value for the horizontal resolution. The selection on the right hand side of the input field specifices the dimension. The horizontal uncertainty scaling factor is also assumed to be one.

Vertical resolution defines the start value for the vertical resolution. The selection on the right hand side of the input field specifices the dimension.

Image center x coord. specifies the x coordinate of the image center (principal point). This input field is optional in case a reference image is specified. Then the image center is calculated automatically.

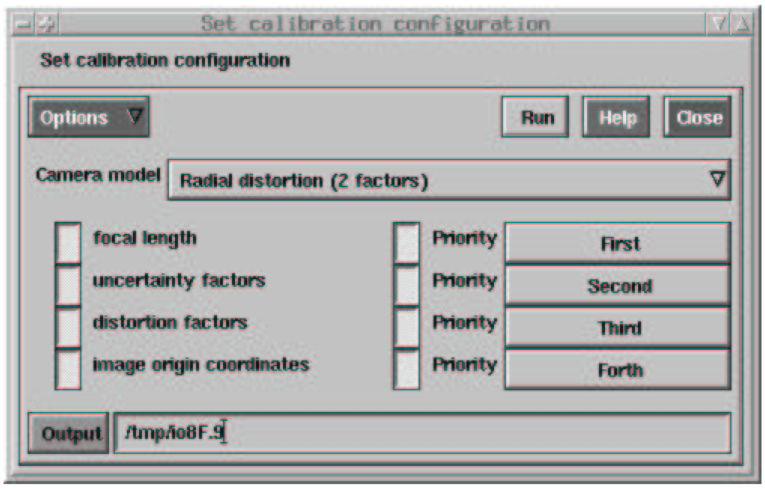
Image center y coord. specifies the y coordinate of the image center (principal point).This input field is optional in case a reference image is specfied. Then the image center is calculated automatically.

Input file

Image center from image file specifies the file name of the reference image from which the image center is taken as the location of the principal point. This specification is optional in optional and mutual exclusive to the input fields the image center coordinates.

Output file contains the start values for all intrinsic camera parameters in a form compatible with the input file structure of the Start Values input file of *MCalib*.

C.3.4 SetConfiguration



Camera model: Radial distortion (2 factors)

focal length	Priority First
uncertainty factors	Priority Second
distortion factors	Priority Third
image origin coordinates	Priority Forth

Output: /tmp/io8F.9

Figure C.4: Pane of the Configuration Task SetConfiguration

Input parameters

Camera model This selection defines the camera model used for calibration. The implemented camera models differ in the way distortion effects are modelled. (radial dist. according to Lenz, radial dist. according to Tsai, radial and tangential distrotion).

focal length This flag specifies if the value of the focal length should be optimized during the calibraiton procedure.

uncertainty factor This flag specifies if the value of the uncertainty scale factor s_x should be optimized during the calibraiton procedure.

distortion factors This flag specifies if the value of the distortion parameters ($\kappa, \kappa_1, \kappa_2, \tau_1, \tau_2$ depending on the camera model).

image origin coordinates This flag specifies if the position of the principal point in the image plane.

Priority flags Each selection field is assigned to the parameter flag on the left hand side. There are four priority levels defined. The reason for that is, that the optimization can also be done consecutivly with different sets of camera parameters to optimize. So the parameter with the highest priority is included first, while parameters with a lower priority are included in a consecutive run afterwords together with the ones that have higher priority. The reason for that is that the start values for the calibration procedure can be given in different qualities (accuracy), or that we are more interesseted in the accuracy of one parameter than in that for others.

Output file contains all configuration information for the *MCalib* task compatible with its input file specification (see Appendix C.4)

C.4 File format specification

In this section all used file formats are described in a formal manner. For all file types their standard naming format and their syntactical structure is given. In case the reader is not familiar with the formal notification as a gammar here are some short instruction how to read this defintion.
A syntactical definition of a file format defined as a language typically looks like this:

> language name := { {set of expressions}, {set of atoms}, {derivation rules}, start symbol }

The *set of expressions* contains all non-atomic syntactical elements of the language, where the *set of atoms* contains all *atomic* signs, also called *atoms*. An atom is syntactical elements that can not be derived. The derivation of the *non-atomic* elements is specified by a set of *derivation rules* which are typically in the following syntax:

$$< elementname > \Rightarrow < rulebody >$$

The *rule body* consists of a series of expressions that can be grouped by { . . . } and [. . .]. Where { . . . } means that the bracketed expressions can be repeated serveal times and [. . .] designates an optional expression. Also alternatives can be expressed by the notation

$$< expression1 > | < expression2 >.$$

So one of the two expressions can be specified in this case. Finally the *start symbol* is the syntactical element that has to be derived first by a syntactical derivation of a sentence (file) in that grammar.

C.4.1 Input files

Start Values: defines the start values for the intrinsic camera parameters. To identify the parmeters the following abbreviations for the intrinsic camera parameters (see also 2.3 on page 19) are specified:

f: focal length
sx: uncertainty factor for x resolution
sy: uncertainty factor for y resolution
k1,k2: radial distortion parameters;

(in case only *k1* is given, the Lenz's camera model is adopted)

tx,ty: tangential distortion parameters
dx: horizontal resolution of the sensor
dy: vertical resolution of the sensor
Cx,Cy: pixel coordinates of the principal point

Filename: $< filename >$.start

Language definition: START := {{ STARTFILE, FILEHEADER, FILEBODY, CAMTYPE, CAMID, PARAMETER, VALUE, NUM, FLOAT, DIGIT, LETTER},
{ "f","sx","sy","dx","dy", "k1","k2","tx","ty","Cx","Cy","."}, *Rules*, STARTFILE }
Rules := {

STARTFILE ⇒ FILEHEADER FILEBODY
FILEHEADER ⇒ CAMID CAMTYPE $< CR >$
FILEBODY ⇒ {PARAMETER VALUE}
CAMID ⇒ NUM
CAMTYPE ⇒ {LETTER |DIGIT}
PARAMETER ⇒

"f"|"sx"|"sy"|"dx"|"dy"|"k1"|"k2"|"tx"|"ty"|"Cx"|"Cy"

VALUE ⇒ NUM |FLOAT
NUM ⇒ DIGIT {DIGIT}
FLOAT ⇒ DIGIT {DIGIT}["."DIGIT]
DIGIT ⇒ 0-9
LETTER ⇒ a-z |A-Z |"-" |"/"
}

World Coordindates: defines the real world sizes and positions of the calibration object points.

Filename: $< filename >$.world

Language definition: WCOORD := { { WCOORDFILE,FILEHEADER, FILEBODY ,POINT, COORD, NUM, DIGIT},{"Index WorldX WorldY WorldZ $< CR >$" "."}, *Rules*, WCOORDFILE}
Rules := {

```
WCOORDFILE ⇒ FILEHEADER FILEBODY
FILEHEADER ⇒
    "Index WorldX WorldY WorldZ < CR >"
FILEBODY ⇒ {POINT}
POINT ⇒ NUM COORD COORD COORD
NUM ⇒ DIGIT {DIGIT}
COORD ⇒ DIGIT {DIGIT} ["." {DIGIT}]
DIGIT ⇒ 0-9
}
```

Image Coordinates: contains all detected calibration object points of a calibration object view in an image.

Filename: $< filename >$.image

Language definition: IMAGE := {{IMAGEFILE, FILEHEAD, FILEBODY, POINT,COORD}, {"Index ImageX ImageY $< CR >$","."}, *Rules*, IMAGEFILE}
Rules:= {

```
IMAGEFILE ⇒ FILEHEADER FILEBODY
FILEHEADER ⇒ "Index ImageX ImageY < CR >"
FILEBODY ⇒ {POINT}
POINT ⇒ NUM COORD COORD
NUM ⇒ DIGIT {DIGIT}
COORD ⇒ DIGIT {DIGIT} ["." {DIGIT}]
DIGIT ⇒ 0-9
}
```

Vertices: defines the sets of calibration points (the points numbered in the world coordinate specification file) according to them the outer orientation of the calibration object is computed by the *Finsterwalder's algorithm*. The file directly reffers to a specific world coordinate file (WCOORDFILE, see above).

Filename: $< filename >$.vertices

Since the data has a context to a specific calibration object and therefore to one particular world coordinate file (WCOORDFILE), it is quite appropiate to name the prefix after the same prefix $< filename >$ as in this file.

Language definition VERTICES := {{VERTFILE, FILEHEADER, FILEBODY, NUM, DIGIT}, {"First Second Third Fourth"}, *Rules*, VERTFILE }
Rules := {

VERTFILE ⇒ FILEHEADER FILEBODY
FILEHEADER ⇒ "First Second Third Fourth $< CR >$"
FILEBODY ⇒ {NUM NUM NUM NUM $< CR >$}
NUM ⇒ DIGIT {DIGIT}
}

Optimization Setup defines the parameters that have to be optimized by the calibration task. They are specified by a list that contains some elements of the set {*f, sx, sy, k1, k2, tx, ty, Cx, Cy* }.
There are some restrictions about the possible combinations of estimation parameters:

- if *k2* is included in the list also *k1* has to.
- if *tx* is included in the list also *ty* has to and vice versa.
- if *tx* and *ty* are included in the list also *k1* and *k2* have to.
- if *k1* is included in the list and *k2* is not, *tx* and *ty* have not to.
- if *Cx* is included in the list also *Cy* has to and vice versa.

Filename: $< filename >$.setup

Language definition: SETUP := { {SETUPFILE, FILEHEADER, FILEBODY, PARLIST, PARAMETER, NUM}, {"f","sx","sy","k1","k2", "tx","ty","Cx","Cy","Setup Information"}, *Rules*, SETUPFILE}
Rules := {

SETUPFILE ⇒ FILEHEADER FILEBODY
FILEHEADER ⇒ "Setup Information" $< CR >$
FILEBODY ⇒ NUM PARLIST {NUM PARLIST} $< CR >$
PARLIST ⇒ PARAMETER {PARAMETER}
PARAMETER ⇒ "f"|"sx"|"sy"|"k1"|"k2"|"tx"|"ty"|"Cx"|"Cy"
NUM ⇒ DIGIT {DIGIT}
}

C.4.2 Output files

Intrinsic camera parameters contains all values for the intrisic camera parameters computed by the optimizer. It also contains the information which parameters were accutally estimated or kept constant in the previous run.

Filename: $< filename >$.cam

Language definition: INTRINSIC:= {{IPARFILE, FILEHEADER, FILEBODY, PARAMETER, PARNAME, BOOLEAN, VALUE, NUM, FLOAT, DIGIT}, {"f","sx","sy","k1","k2","tx","ty","Cx","Cy",".","Parameter Value optimized", "True","False"}, *Rules*, IPARFILE}
Rules := {

IPARFILE $\Rightarrow$ FILEHEADER FILEBODY
FILEHEADER $\Rightarrow$ "Parameter Value optimized" $< CR >$
FILEBODY $\Rightarrow$ PARAMETER PARAMETER
PARAMETER $\Rightarrow$ PARNAME VALUE BOOLEAN $< CR >$
PARNAME $\Rightarrow$
"f"|"sx"|"sy"|"dx"|"dy"|"k1"|"k2"|"tx"|"ty"|"Cx"|"Cy"
BOOLEAN $\Rightarrow$ "True" |"False"
VALUE $\Rightarrow$ NUM |FLOAT
NUM $\Rightarrow$ DIGIT {DIGIT}
FLOAT $\Rightarrow$ DIGIT {DIGIT} ["."{DIGIT}]
DIGIT $\Rightarrow$ 0-9
}

Extrinsic parameters contains the parameters of the outer orientation in one view (image) of the calibration object (4x4 rotation/translation matrix).

Filename: $< filename >.< imagefile >$.extrinsic

The *imagefile* part of the filename designates the name of the view (image file) that was analysed.

Language definition: EXTRINSIC := {{EXTFILE, ROW, FLOAT},{"."}, *Rules*, EXTFILE}
Rules := {

EXTFILE $\Rightarrow$ ROW ROW ROW ROW
ROW $\Rightarrow$ FLOAT FLOAT FLOAT FLOAT $< CR >$
FLOAT $\Rightarrow$ DIGIT {DIGIT} ["." {DIGIT}]
DIGIT $\Rightarrow$ 0-9

Optimization statistics: contains the covariance matrix (see 3.40 on page 48) of the optimized parameters, the minimum and the number of iterations of the optimization procedure.

Filename: $< filename >$.statistic

Language definition: STATISTIC := {{STATFILE, FILEHEADER, FILEBODY, OPTSTAT, PARLIST, PARAMETER, MATRIX, ROW, NUM, FLOAT},{ "f","sx", "sy","k1","k2","tx","ty","Cx","Cy,".","Minimum", "Iterations","Parameters" },*Rules*, STATFILE }
Rules := {

STATFILE $\Rightarrow$ FILEHEADER FILEBODY
FILEHEADER $\Rightarrow$ OPTSTAT $< CR >$ "Parameters" PARLIST $< CR >$ MATRIX
OPTSTAT $\Rightarrow$ "Minimum" FLOAT "Iterations" NUM $< CR >$
PARLIST $\Rightarrow$ PARAMETER {PARAMETER}
PARAMETER $\Rightarrow$ "f"|"sx"|"sy"|"k1"|"k2"|"tx"|"ty"|"Cx"|"Cy"
MATRIX $\Rightarrow$ ROW {ROW}
ROW $\Rightarrow$ FLOAT {FLOAT}
NUM $\Rightarrow$ DIGIT {DIGIT}

```
FLOAT ⇒ DIGIT {DIGIT} ["." {DIGIT}]
DIGIT ⇒ 0-9
}
```