CENTER FOR
MACHINE PERCEPTION

CZECH TECHNICAL
UNIVERSITY

PhD THESIS

ISSN 1213-2365

Algebraic solutions to absolute pose problems

Martin Bujňák

bujnam1@cmp.felk.cvut.cz

CTU–CMP–2012–20

September 26, 2012

Available at
ftp://cmp.felk.cvut.cz/pub/cmp/articles/bujnak/Bujnak-TR-2012-20.pdf

Thesis Advisor: Tomáš Pajdla

I gratefully acknowledge EC project MRTN-CT-2004-005439 VISION-TRAIN, which supported my research.

Research Reports of CMP, Czech Technical University in Prague, No. 20, 2012

Published by

Center for Machine Perception, Department of Cybernetics
Faculty of Electrical Engineering, Czech Technical University
Technická 2, 166 27 Prague 6, Czech Republic
fax +420 2 2435 7385, phone +420 2 2435 7637, www: http://cmp.felk.cvut.cz

Algebraic solutions to absolute pose problems.

A Dissertation Presented to the Faculty of the Electrical Engineering of the Czech Technical University in Prague in Partial Fulfillment of the Requirements for the Ph.D. Degree in Study Programme No. P2612 - Electrotechnics and Informatics, branch No. 3902V035 - Artificial Intelligence and Biocybernetics, by

Martin Bujňák

Prague, September 2012

Thesis Advisor: **Ing. Tomáš Pajdla, Ph.D.**

Center for Machine Perception
Department of Cybernetics
Faculty of Electrical Engineering
Czech Technical University in Prague
Karlovo náměstí 13, 121 35 Prague 2, Czech Republic
fax: +420 224 357 385, phone: +420 224 357 465
http://cmp.felk.cvut.cz

Abstract

Estimating internal and external camera calibration is a very basic element in many computer vision applications. Camera localization, structure from motion, scene reconstruction, object localization, tracking and recognition are just a few examples of such applications. This thesis focuses on minimal algorithms for estimating camera calibration, i.e. algorithms which use all possible constraints and minimal number of inputs, for example point correspondences between 2D and 3D space, to calculate the camera pose and other camera parameters such as unknown focal length or coefficients modeling lens distortion.

In this work, we first study the absolute pose problem for a calibrated camera, which was an intensively studied problem in the past and many solutions were already developed. The problem itself can be formulated as a simple system of polynomial equations. Researches in the past focused on how to solve this problem, searched for different solutions, compared numerical stability, speed, or studied how to calculate the camera pose from more than three 2D-to-3D point correspondences. We review the state-of-the-art and present our own formulations to this problem based on the well known invariants and properties of the problem. We provide solutions to our formulations using different methods for solving system of polynomial equations.

Next we provide solutions to the absolute pose for a camera without complete internal calibration or for a camera where some additional information about the scene is known. In particular, absolute pose of a camera calibrated up to an unknown focal length or a camera with unknown focal length and unknown radial distortion. Furthermore, we describe special cases when some of the scene or camera priors are known, for example, scene is planar, scene is non-planar or when vertical direction of a camera is known from a gyroscope or a vanishing point. The main contribution of this thesis is in finding minimal or finding optimal solutions to these problems.

We formulate all studied problems, from very basic relations between 3D space and 2D measurements, show different formulations and how can different invariants be helpful in reducing the number of unknowns or to simplify the problem. All our formulations lead to systems of polynomial equations. We show how to solve these systems using methods for solving systems of polynomial equations, which we developed during our research. Solution the problems are evaluated with high level of detail and with focus on important properties such as numerical stability and resistance to noise in data. We compare our new solvers with the state-of-the-art on synthetic and real data.

In this thesis we further present a general method which speeds up most of the presented solvers and can be also used to speed up other solvers based on eigenvalue computation. We have also found the connection between methods for converting basis of an ideal to a basis with respect to the lexicographic ordering and calculation of the characteristic polynomial of an action matrix.

Acknowledgments

I would like to express my thanks to my colleagues at CMP who I had the pleasure of working with, especially to Zuzana Kúkelová for her ideas and collaborative efforts in a large part of my work.

I am greatly indebted to Tomáš Pajdla and Radim Šára for guiding me throughout my research. Their friendly support, patience and theoretical and practical help have been paramount to the successful completion of my PhD study. I also would like to thank to my family and to my friends for all their support that made it possible for me to finish this thesis.

I gratefully acknowledge EC project MRTN-CT-2004-005439 VISIONTRAIN, which supported my research.

Contents

Keywords: camera absolute pose, camera calibration, focal length, radial distortion, minimal problems

Table of Problems

Table of Scripts

Notation

h	scalar value
α	scalar scale factor
$\mathbf{h}$	column vector
$\mathtt{H}$	matrix
$[\mathtt{G} \mid \mathbf{h}], [\mathtt{G} \mid \mathtt{H}]$	matrix $\mathtt{G}$ augmented by column vector $\mathbf{h}$, or by $\mathtt{H}$
H	a set
$\mathbb{C}$	the set of complex numbers
$\mathbb{Z}_p$	the finite prime field $\mathbb{Z}/p$
$G \subset H$	G is a subset of H
$G \setminus H$	set difference
$G \setminus h$	set G excluding element h, i.e. $G \setminus \{h\}$
$\lvert H \rvert$	cardinality of H
$h \in H$	h is an element of the set H
$f(x_1, \ldots, x_n)$, $f(\mathbf{x})$	polynomial in variables $\mathbf{x} = (x_1, \ldots, x_n)$
$\mathbb{C}[x_1, \ldots, x_n]$	set of all polynomials in variables $x_1, \ldots, x_n$ with coefficients from $\mathbb{C}$
$\mathbf{x}^\alpha = x_1^{\alpha_1} \ldots x_n^{\alpha_n}$	monomial in variables $x_1, \ldots, x_n$
$\overline{f}^G$	reminder of f on division by a set of polynomials G
$\mathbf{0} = (0 \ldots 0)^\top$	vector of zeros ($\mathbf{0}_m$ denotes $m \times 1$ vector of zeros)
$\mathtt{0} = (\mathbf{0} \ldots \mathbf{0})$	matrix of zeros
$\mathtt{I}, \mathtt{I}_n$	Identity matrix, respectively $n \times n$ Identity matrix
$[\mathbf{h}]_\times = \begin{pmatrix} 0 & -h_3 & h_2 \\ h_3 & 0 & -h_1 \\ -h_2 & h_1 & 0 \end{pmatrix}$	skew-symmetric matrix
$\det(\mathtt{H})$	determinant of matrix $\mathtt{H}$
$\mathrm{tr}(\mathtt{H}) = \sum_i h_i^i$	trace of matrix $\mathtt{H}$
$\lVert \mathbf{h} \rVert = \sqrt{\sum_i h_i^2}$	Euclidean norm of $\mathbf{h}$
$\lvert h \rvert$	absolute value of h.
$\mathbf{h} \perp \mathbf{g}$	vector $\mathbf{h}$ is perpendicular to vector $\mathbf{g}$

Commonly used symbols and abbreviations

CG	Center of gravity
DIAC	Dual image of the absolute conic
DLT	Direct linear transform
EXIF	Exchangeable image file format
FGML	Faugére, Gianni, Lazard, and Mora algorithm for converting Gröbner basis
G-J	Gauss-Jordan elimination
GB	Gröbner basis
GCD	The greatest common divisor
IMU	Inertial measurement unit
RANSAC	Random Sampling Consensus
lex	Lexicographic ordering of monomials
grevlex	Graded reverse lexicographic ordering of monomials
GEP	Generalized eigenvalue problem
QEP	Quadratic eigenvalue problem
PEP	Polynomial eigenvalue problem
PnP	Perspective-N-Point problem
P3P	PnP using n=3 points for a calibrated camera
P4P	PnP using n=4 points
P4P+f	P4P with unknown focal length
P4P+f+k	P4P+f also with unknown radial distortion
uP2P	PnP from n=2 points for calibrated camera with known vertical direction
uP3P+f+k	PnP from n=3 points for radially distorted camera with unknown focal length and with known vertical direction

1 Introduction

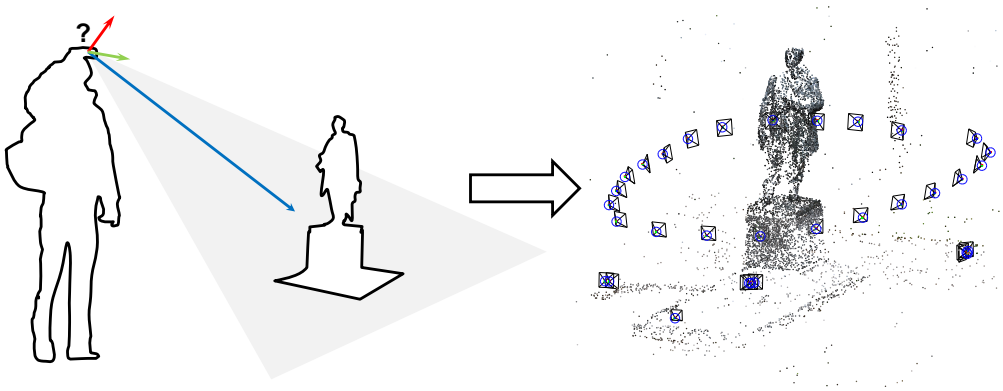


Figure 1.1: Illustration to the camera pose estimation problem.

Solving absolute camera pose, which means determining the position, orientation and possibly intrinsic camera parameters, has many application in various computer vision tasks, Figure 1.1. Indeed, one of the oldest papers considering this problem dates back to 1841 [42]. Since then, a huge number of solutions to this problem for three and more than three points has been published [37, 82, 88, 89, 5, 102, 105].

In this work we explore anatomy of camera pose estimation from a set of n correspondences between 3D points and their 2D projections. This problem is known as the Perspective-n-Point (PnP) problem. We concentrate on minimal solutions, i.e. when n is the smallest possible value sufficient to solve the problem. A number n of needed point correspondences between 2D and 3D depends on the number of additional constraints we have. For instance, Grunert [42] has shown that given three point correspondences, there can be up to four real solutions to the P3P problem if inner calibration of the camera is known. The minimal number of points needed to estimate the camera position and orientation for a fully uncalibrated camera is six - more precisely five and "half" points are sufficient. The linear solution to this problem exists and is known as Direct Linear Transform (DLT) [2, 86]. From a calculated camera we can extract camera rotation, translation and five elements of the camera calibration matrix [47].

Modern digital cameras have square pixels [47] and the principal point [47] close to the center of the image. For most of the applications this prior knowledge can be used and four out of the five internal calibration parameters can be safely set to known values (the skew to 0, the pixel aspect ratio to 1 and the principal point to the center of the image. Moreover, since most of the digital cameras put the information about the focal length into the image header (EXIF), it

is frequently assumed that this is a good approximation of the whole internal calibration of the camera and therefore the camera is considered calibrated. Adopting these calibration constraints has several advantages. First, the minimal number of points needed to solve the absolute pose of a camera is reduced. Secondly, since fewer parameters are estimated, the results are more stable.

In this work we show that four point correspondences are enough to estimate camera pose with unknown focal length and also with unknown radial distortion in the image plane. We published a general solution to the problem for the unknown focal length (P4P+f) in [12] and later Josephson et al. [56] included the radial distortion estimation (P4P+f+k) to this problem. In [14, 15] we have shown that by splitting the problem for planar and non-planar scenes, we can create much faster and more stable solvers. Indeed, taking the advantage of having scene priors, e.g. that a scene is planar or non-planar, which are also available in advance for absolute pose problems, can help finding more efficient solutions.

Another example of prior information which even more simplifies absolute pose solvers is knowledge of the camera vertical direction. This can be extracted, for example, from the vanishing point of verticals or read from a gyroscope mounted to the camera. We show that given the camera vertical direction, we can reduce the number of point correspondences between 2D and 3D space needed to calculate the camera pose of a calibrated camera to two point correspondences. Three point correspondences are sufficient for calculating absolute pose of camera with unknown focal length and radial distortion [66]. In both cases, the resulting solvers are very small and very fast. It sounds surprising that such a small number of point correspondences are sufficient to calculate the camera pose and calibration. In fact, knowing the vertical direction reveals roll and pitch angles and hence two out of the three rotation components. Importance and future potential of these algorithms grows since more and more consumer cameras and smart devices are equipped with IMUs (inertial measurement unit) and the vertical direction can be easily read from sensors or from image headers.

In this thesis we revisit our previously published work in more detail. We focus on mathematical formulations of problems, explain which formulations are suitable for further extending to a more complicated camera models. We formulate the problems we are solving as systems of polynomial equations. We show how to solve these systems either by hand using different algebraic methods or we show how to create a generating script for the automatic generator [64], which we developed. We demonstrate the usefulness of all presented algorithms on synthetic and real data. With a great help of the automatic generator [64] we could automatically create, verify, evaluate and compare thousands of different camera solvers.

The more we know about the scene, the more efficiently and robustly we can estimate the camera parameters. Fast development in sensors and consumer cameras, but also the existence of the Internet, affected the amount of image data which are easily accessible. Photo-sharing websites contain millions of images of various objects, landmarks or places on the earth. While few decades ago many computer vision problems were certainly unsolved and often thought unsolvable [47], today we are fighting with new challenges, for example, creating a 3D reconstruction of our world from available Internet image collections. Due to the amount of data which needs to be processed it is important to create fast solvers.

Although many solvers described in this thesis are already fast, i.e. their execution time is just a few microseconds, we can make them even faster, not only by optimizing their implementa-

tion, but also by exploiting further theoretical results to lower their computational complexity. In [16] we identified that eigenvalue computation is often the bottle-neck of solvers based on the Gröbner basis method, the polynomial eigenvalue method or generated using the automatic generator [64]. This includes problems presented not only in this thesis, but many other solvers in general. Since the most of the computer vision problems are restricted to the real field $\mathbb{R}$ with parameters constrained to some interval, e.g. focal length is positive and radial distortion coefficients are in the interval $(-1, 1)$, we can exploit these constraints to save computation effort. We show how to convert a problem to a simpler one in which eigenvalue and eigenvector computations are replaced by calculating roots of a single variable polynomial. This allows focusing on roots in a certain interval using, for example, Sturm's sequences [49]. We demonstrate in experiments that such an approach is numerically stable and modified solvers are several times faster than solvers based on the eigenvalue computation.

2 Contribution of the thesis

"The mere formulation of a problem is often far more essential than its solution, which may be merely a matter of mathematical or experimental skills. To raise new questions, new possibilities, to regard old problems from a new angle requires creative imagination and marks real advances in science."

Albert Einstein

The contribution of the thesis is related to estimating absolute camera pose from a minimal set of point correspondences between 2D and 3D space. This thesis also studies the absolute camera pose problem when some additional information is available, for example when scene is planar or when the camera vertical direction is known. The main contribution of this dissertation work is in formulating and developing new solutions to camera absolute pose problems:

- **Chapter 5** Absolute pose solvers for a calibrated camera. We show basic mathematical and algebraical relations, useful constraints and invariants and also how to build a system of polynomial equations which can be used to easily solve this P3P problem. We derive new solutions to P3P problem based on (1) distances between points in 3D space, (2) ratios of these distances, and we also derived (3) a direct solution to the problem (calculating camera matrix directly) based on the relation between projected points lying on the plane. Next, we build on these basic relations and derive more complicated solvers which gradually relax input calibration assumptions.

- **Chapter 6** Absolute pose solvers for a camera with unknown focal length is our next contribution. We present several different approaches to formulating and solving the problem. We exhaustively searched and analyzed over hundred thousands of possible solvers, picked the best solvers which can benefit from the fact that four point correspondences over-constrain this problem. We found solvers which provide much better performance on noisy data compared to the solvers presented in our previous work [12]. We also show that the problem can be simplified by considering planar and non-planar scenes separately. Moreover, this formulation allows calculation of the aspect ratio or the camera skew as well from the given set of four 2D-to-3D point correspondences.

- **Chapter 7** We developed a new fast absolute pose solver for a camera with unknown focal length and unknown radial distortion. We achieved speed and efficiency by splitting the problem into two sub-problems, i.e. for planar and non-planar scenes. We demonstrate in

experiments that the general purpose solver can be created by combining solvers for these two sub-problems.

- **Chapter 8** Absolute pose solver for a camera with known vertical direction, e.g. aligned with the ground plane, read from a vanishing point or from an inertial sensor. We derived closed form solutions to two previously unsolved problems. The first one is for completely calibrated camera where we show that two point correspondences are enough to calculate the camera pose. The second one uses three point correspondences to calculate the camera pose, unknown focal length and unknown radial distortion. Both solutions are very fast, efficient, numerically stable and closed-form.

In this work we show how to formulate and solve the above problems. People are often afraid of solving polynomials equations. This is true even for simple problems but, in fact, it is often sufficient to use simple methods to solve even complicated problems. Therefore, we show step-by-step how to get from a problem formulation to a final solution using popular algebraic methods. For more complicated problems we show how to use the automatic generator [64], which we recently developed. We also focus on technical details of the solver creation process, describe how to work in the finite prime field arithmetic, how to create a camera or 3D scenes in such fields, how to cope with over-constrained systems and also how to further tune solvers speed.

2.1 Optimization

Our next contribution in **Chapter 9** is in showing how the theory of solving system of polynomial equations using the action matrices is connected to the characteristic polynomial calculation and Cayley-Hamilton theorem. We developed a fast matrix version of popular FGML algorithm, which is used for converting Gröbner bases to Gröbner bases with respect to the lexicographic ordering. Next, we show how the conversion process is related to the calculation of the characteristic polynomial of an action matrix using Krylov's method. In fact, one of the polynomials in the lexicographic ordering is the characteristic polynomial of the action matrix. Based on this result we show how to build faster and numerically more stable Gröbner basis solvers.

2.2 Publications

The content of this thesis is based on the material published in the following papers:

M. Bujnak, Z. Kukelova, and T. Pajdla. A general solution to the p4p problem for camera with unknown focal length. In *CVPR'08*, 2008.

M. Bujnak, Z. Kukelova, and T. Pajdla. New efficient solution to the absolute pose problem for camera with unknown focal length and radial distortion. In *ACCV'10*, pages 11–24, 2010.

M. Bujnak, Z. Kukelova, and T. Pajdla. Efficient solutions to the absolute pose of cameras with unknown focal length and radial distortion by decomposition to planar and non-planar cases. *IPSJ Transaction on Computer vision and Applications*, 2012.

Z. Kukelova, M. Bujnak, and T. Pajdla. Closed-form solutions to the minimal absolute pose problems with known vertical direction. In *ACCV'10*, 2, pages 216–229, 2010.

M. Bujnak, Z. Kukelova, and T. Pajdla. Making Minimal Solvers Fast. In *CVPR'12*, 2012.

Additional papers:

Z. Kukelova, M. Bujnak, and T. Pajdla. Polynomial eigenvalue solutions to minimal problems in computer vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(7):1381–1393, 2012.

Z. Kukelova, M. Bujnak, and T. Pajdla. Automatic generator of minimal problem solvers. In *ECCV'08, Part III*, volume 5304 of *Lecture Notes in Computer Science*, pages 302–315, 2008.

Z. Kukelova, M. Bujnak, and T. Pajdla. Polynomial eigenvalue solutions to the 5-pt and 6-pt relative pose problems. In *BMVC'08*, 2008.

M. Bujnak, Z. Kukelova, and T. Pajdla. 3d reconstruction from image collections with a single known focal length. In *ICCV'09*, pages 1803–1810, 2009.

M. Bujnak, Z. Kukelova, and T. Pajdla. Robust focal length estimation by voting in multiview scenes. In *ACCV'09*, pages 13–24, 2009.

A. Torri, Z. Kukelova, M. Bujnak, and T. Pajdla, The six point algorithm revisited, Computer Vision in Vehicle Technology: From Earth to Mars. In *ACCV'10 CVVT Workshop*, pages 184–193, 2010.

M. Bujnak, and R. Sara. A Robust Graph-Based Method for The General Correspondence Problem Demonstrated on Image Stitching. In *ICCV'07*, pages 1–8, 2007

M. Bujnak, and R. Sara. An Efficient Deterministic Algorithm for The Sparse Correspondence. In *CVWW'08*, 2008.

3 State-of-the-Art

In this chapter we summarize the state-of-the-art of the camera absolute pose problem. The Perspective-n-Point (PnP) problem, i.e. the problem of determining the absolute position and orientation of a camera given a set of n 2D-to-3D point correspondences, is a well studied problem in both computer vision and photogrammetry with a broad range of applications in structure from motion [1, 4, 80] or recognition [70, 71].

The problem is called "space resection" in photogrammetry community and its first solution was published in 1841 by a German mathematician, which appears to be discovered sooner in 1795 by Lacroix, who was inspired by Monge [92]. Further and later, many solutions were published in the German and then American photogrammetry literature, and most recently in the current computer vision literature. However, the geometry of image itself, i.e. vanishing points and perspective, was understood by artists much sooner and dates back to $15th$ century. In fact the first publication on geometry of painting, "De pictura", was published by Italian architect and art theorist Leon Battista Alberti in 1435. He formulated the theory based on planar projections, and showed how the rays of light are passing from the viewer's eye to the landscape and how they strike the picture plane - the painting.

The problem is important in its own right as a core problem within the field of multiple view geometry [47] but it also appears as a subproblem for many other vision applications. Therefore many solutions to this problem and its various mutations exist.

Camera pose can be calculated from various kinds of image measurements, for example from a set of 2D projections of 3D points or 3D lines [30, 50, 69, 47], from a combination of points and lines [6], projections of known planar objects [104] like chessboards, coplanar circles [21], intersections of parallel lines (vanishing points) [47], edges of models [76] and more. In this thesis we focus on the camera pose estimation from a set of point correspondences between 3D space and 2D measurement on the image plane. On one hand, the estimation from more rich objects, like lines or circles, might appear more accurate, on the other we need to solve non-trivial low level computer vision tasks such as detection [94], and we need to compensate the fact that image plane might be affected by distortion from optics, leaving lines curved or circles deformed. Moreover, in case of methods like calibration from vanishing points we, need to detect parallel lines, which is itself a complicated computer vision task [10].

Depending on the number of measurements, i.e. point correspondences between 2D and 3D space, we can split camera pose algorithms into:

- minimal, which use the smallest possible set of measurements between 2D and 3D space to calculate camera pose and camera parameters and

- non-minimal, which use more measurements to simplify/linearize the task or to return the more precise result.

We can further divide algorithms according to the method used to solve them to:

- Non-Iterative, which usually formulate the problem as some system of equations, either linear or non-linear which is solved in a finite sequence of operations using linear algebra. These methods deliver exact solution in the absence of rounding errors.
- Iterative which search in a parameter space and optimize some objective function, for example the Euclidean distance between reprojected 3D points and known measurements. This is typically done in a sequence of improving approximate solutions and the optimization is terminated once a termination criteria is satisfied, e.g. a small change in the parameter space.

In this work we study minimal algorithms which we solve in closed form or using linear algebra. Minimal algorithms are usually used to filter out incorrect correspondences - outliers - for example using RANSAC paradigm [11, 47]. Once correct correspondences are identified we can use non-minimal algorithms to refine the final solution. To find correct correspondences, it is important to have a fast algorithm which is using as small number of measurements as possible to calculate the camera pose. It is because such an algorithm is executed inside the RANSAC loop many times and the number of iterations is exponentially proportional to the size of the minimal set [11, 47]. This is because the number of different m-tuples drawn from a set of size n is

$$\left(\begin{array}{c} n \\ m \end{array} \right) = \frac{n!}{(n-m)!m!} = \frac{(n-1)\dots(n-m)}{m!} \sim n^m. \tag{3.1}$$

On the other hand, a better estimate is expected when more than a minimal number of measurements is used. Given outlier free data, it is easy to optimize some objective function, for example above mentioned distance between reprojected 3D points and known measurements. Given a good initial estimate, for example from a minimal algorithm, methods such as Gauss-Newton iteration [87, 47] converge very quickly and return very accurate results. Also, it is much easier to incorporate additional constraints into the optimization process, such as parallel or orthogonal lines, lens distortion, compared to creating minimal solver for the same task.

Depending on the what we know about the camera we further distinguish situation when

- a camera internal calibration is known, i.e. only the camera position and orientation is unknown or
- a camera is not calibrated and calibration parameters are subject of interest as well.

In this work we first study the pose problem for the calibrated camera and then we extend theory for cameras with an unknown focal length, aspect ratio and unknown radial distortion. We further show how to benefit from knowing that scene is planar or non-planar or how to use known vanishing point or a camera vertical direction. We focus on central cameras or pin-hole camera model. Non-central cameras are out of the scope of this thesis.

Next, we review the state-of-the-art relevant to our work in the following three areas. We begin with a review of existing camera absolute pose solutions. The literature contains many solutions to absolute camera pose estimation for calibrated cameras. However, there is only limited prior work where this thesis contributes the most - in solving camera pose for cameras calibrated up to an unknown focal length, and cameras with unknown focal length and lens radial distortion. Minimal solutions to the problem of estimating camera pose when the vertical direction is known also have not been studied before.

3.1 Absolute pose of a calibrated camera

The first solution to this problem was published by German mathematician J. A. Grunert [42] in 1841, who formulated the problem in terms of three polynomial equations and solved it by a substitution approach in three steps. The more recent desire has been to solve the distance equations in fewer steps, as exemplified in the works of Finsterwalder and Scheufele (1937) [35], Merritt (1949) [81], Fischler and Bolles (1981) [37], Linnainmaa et al. (1988) [75], Grafarend et al. (1989) and Lohse (1990) [41].

Fischler and Bolles [37] were apparently not aware of the earlier American or even more earlier German solutions to the problem [44]. Their solution become popular in computer vision and they summarized the problem as follows.

Problem 1. *Given the relative spatial locations of n control points, and given the angle to every pair of control points $\mathbf{P_i}, \mathbf{P_j}$ from an additional point called the center of perspective $\mathbf{C}$, find the lengths of the line segments joining $\mathbf{C}$ to each of the control points, see Figure 5.2.*

In fact, they solved the same system as Grunert [42] using just a different substitution and showed that the problem has four geometrically feasible solutions. However this was known before since a fourth degree polynomial appears in the solution [42, 35] and such a polynomial can have all four roots real. The number of solutions for different problem formulations is analyzed in [55].

Haralick et al. [44] reviewed the major direct solutions up to 1991, including the six above algorithms. Each algorithm is carefully reviewed including numerical stability under different order of substitution steps and eliminations. The various forward and backward substitution steps inherent in the classical closed-form solutions of the above solutions are avoided in Awange et al. [8]. Here Gröbner basis was used to solve the problem starting from original Grunert equations [42]. Another deep analysis of the P3P problem was given in Gao and Tang [40] where Wu-Ritt's zero decomposition method was used to obtain a complete solution classification. A simple method which solves camera matrix directly was recently proposed by Kneip [58].

In 2010, Kukelova et al. [66] showed that two point correspondences are sufficient if a vertical direction of the camera is known. The vertical direction can be obtained either by direct physical measurement by, e.g. gyroscopes and inertial measurement units or from vanishing points constructed in images. Problem is reduced to finding roots of a quadratic degree polynomial.

3.1.1 Non-minimal solutions

The original Grunert's formulation appears almost in every P3P problem. In fact, the basic relation used in his formulation is based on the cosine law (Section 5.2, Figure 5.2). Quan and Lan [88] used Sylverter resultant and $x^2 = y$ substitution to build a 4^{th} degree polynomial to solve the cosine law equations for the P3P problem. Next, he has shown how to solve this problem linearly when four or five point correspondences are available. He constructed polynomials from different triplets of points and rewrote them into a matrix form with polynomial coefficients in the matrix and monomials in a vector. The matrix multiplied by the monomial vector gives the original polynomials. By analyzing the kernel or the coefficient matrix and exploiting monomial dependencies, he solved the problem linearly for 4 and 5 points. Technically it is possible to extend this solution to more than 5 points, but this solution is not very practical since the number of possible triplets grows exponentially with the number of input points. Ameller et al. [5] proposed another linear method from four points based on resultants. They used Bezout-Cayley-Dixon method to solve the minimal P3P case. In fact they formulated the problem as a generalized eigenvalue problem too. Their four point method works also for coplanar points and critical configurations but the relative error and failure rate is high. Zhi and Tang [105], Reid et al. [89], Wu and Hu [102] and other developed many variations to the P4P problem for planar as well as non-planar scenes. A linear solution whose computational complexity scales $O(n^2)$ for n points was proposed in [36]. The method is however unstable for noisy 2D locations. Recently, an accurate non-iterative solution with the complexity scaling linearly with the number of points was presented in [82].

3.1.2 Iterative methods

The importance of direct solution was less significant in the photogrammetry community where more iterative solutions were proposed [24, 25]. These iterative solutions, which were just repeated adjustments of the linearized equations required a good starting value. In most photogrammetry cases, an initialization which is good enough for the algorithm convergence is often available. Other non-linear schemes, most typically, the Gauss-Newton methods can be found in [90, 99] or [45]. Bundle adjustment [47] becomes today's standard non-linear refinement of an initial guess. A criteria function optimized using underlaying Levenberg-Marquard algorithm [87] is for absolute pose problem in general simple and convergence is very good.

These algorithms are usually non-minimal in terms of the number of correspondences between 2D and 3D space to guarantee existence of just a single solution.

To avoid calculating an initialization a class of approximate methods for pose estimation has been developed by relaxing the orthogonality constraint on rotation matrices and/or by simplifying the perspective camera model. DeMethon and Davis [29] proposed an iterative solution from four non-coplanar points which does not require any initialization. First, an orthographic problem is solved which is further optimized under the perspective camera model. Horaud et al. [51] analyze this solution in terms of convergence and they show that the iterative para-perspective has a better convergence. Oberkampt [85] initialized from scaled orthographic projection and solved pose from four or more coplanar points. Scherighofer and Pinz [91] showed that for

four coplanar and non-collinear points there are two distinct local minima of the error function - which even exist for cases with wide angle lenses and close range targets. They developed an algorithm which takes into account this fact.

Fast and globally convergent pose from video images was proposed by Lu et al. [77]. They showed that the pose estimation problem can be formulated as that of minimizing an error metric based on collinearity in object (as opposed to image) space. Using object space collinearity error, they derived an iterative algorithm which directly computes orthogonal rotation matrices and which is globally convergent. Lu et al. approach relies on an initial estimation of the camera pose based on a weak-perspective assumption, which can lead to instabilities when the assumption is not satisfied.

3.2 Absolute pose of an uncalibrated camera

A very simple algorithm for an uncalibrated camera pose problem can be derived directly from equations projecting 3D point to its 2D image. In fact, six point correspondences are sufficient to estimate all 11 elements of the projection matrix using linear algorithm. This method is known as direct linear method "DLT" in literature [47, 86, 2, 98]. We will return to this solution further in the text many times.

The pose estimation and the camera calibration are closely related. The main goal of camera calibration methods is to calculate the intrinsic parameters of the camera [101, 104, 33]. These methods are quite complex in terms of the number of parameters they estimate. It is because, besides the camera calibration matrix [47], they typically model various kinds of lens distortions with many unknown distortion coefficients. As a consequence, in order to get a reliable estimate of unknown parameters, much more than a minimal number of points is required.

3.3 Absolute pose with unknown focal length

The first solution to the P4P problem for a perspective camera with unknown focal length from four coplanar points was proposed by Abidi and Chandra [3]. They formulated the problem using areas of triangular subdivisions of a planar quadrangle and arrived to a closed form solution.

The first solution for non-planar scenes was presented by Triggs in [100]. This solution uses calibration constraints arising from using the dual image of the absolute quadric and solves resulting polynomial equations using multivariate resultant theory [26]. The solution works for non-planar scenes but fails for the planar ones. In this paper a solution which handles both planar and non-planar points and is based on eigendecomposition of multiplication matrices is proposed, but it is numerically unstable and not practical. In this work authors also provide a solution to the problem of estimating absolute pose of a camera with unknown focal length and unknown principal point from five 2D-to-3D correspondences. In 2001, Hu et al. [54] solved the coplanar P4P problem with two unknown effective focal lengths. They have shown that if 4 points are on a plane, then camera pose, effective focal length f, and camera aspect ratio can be determined uniquely and simultaneously. In fact, Zhang [104] derived the same equations 3 year earlier and used them for full camera calibration. Both methods transform points in the

space so that z becomes zero. Then, the projection equation can be expressed as a homography. We show how to derive P3P and P4P with unknown focal length this way later in the text. The first general solution working for both planar and non-planar scenes was proposed only recently in [12]. This solution is based on Euclidean rigidity constraint and results in the system of four polynomial equations in four unknowns which are solved using the Gröbner basis method [26]. Bujnak et al. [16] showed how to speed up this solution by more than factor of two by removing eigenvalue computation. In 2012, Bujnak et al. [15] presented two fast and simple algorithms for solving planar and non-planar cases separately. The planar case is transfered to the problem of homography estimation. The non-planar case uses the relationship between the dual image of the absolute conic (DIAC) and the world coordinate of camera optical center. Later in 2012, Gue et al. [43] derived solution for planar and non-planar configuration of points based affine invariant and DIAC too. This solution solves camera aspect ratio as well. Drawback of this method is symbolic calculation of determinant of a 8×8 matrix which is slow and numerically unstable in floating point arithmetics. A branch and bound approach to the camera absolute pose problem was presented in [22]. The solution is not minimal and the global solution is obtained under the L-infinity norm. On the other hand it is much slower compared to the previously mentioned methods.

3.4 Unknown focal length and radial distortion

Estimating the camera lens distortion is standard for many calibration methods such as popular Zhang [104] or Tsai [101] methods. Usually, an iterative numerical scheme is used to calculate the distortion coefficients of some lens distortion model. The problem of estimating absolute pose with unknown focal length from four 2D-to-3D correspondences is not minimal and one additional calibration parameter can be handled in this problem. In [56] authors included the radial distortion to the problem and proposed a method for solving absolute pose problem for a camera with radial distortion and unknown focal length from four point correspondences based on Gröbner bases. They have also demonstrated that considering radial distortion brings a significant improvement in many real applications. Their solution uses quaternions to parametrize rotations and one parameter division model for the radial distortion [38]. The problem is formulated as a system of five equations in five unknowns. These equations are quite complex and therefore the Gröbner basis method produces a relatively large solver (1134×720 matrix) which ran about $70ms$ at the time of its release. Therefore, the proposed solver is not really practical for real applications. Later Bujnak et al. [14] proposed two new solutions to this minimal problem. The first solver works for non-planar scenes and the second one for 3D point lying on a plane. Decomposing the general problem to the non-planar and planar case leads to a much simpler systems of polynomial equations and therefore also much simpler and more practical solutions. The same year, Kukelova et al. [66] showed that three point correspondences are sufficient minimum for a camera for which the vertical direction is known. The problem is reduced to finding roots of a second degree polynomial and solving two small systems of linear equations and can be efficiently solved in a closed-form.

4 Solving systems of polynomial equations

In this dissertation we focus on finding suitable formulations of problems rather than on finding new methods for solving underlying systems of polynomials equations. Reader should refer to more specialized literature when targeting the problem of finding solutions to systems of polynomial equations [26, 62, 64].

In some cases a basic knowledge of linear algebra is sufficient to understand solution methods. Even for more complicated problems it is often not necessary to understand whole theory of polynomial rings to capture the basic idea behind the solution. Hence we only briefly introduce some methods which we use in the text. Whenever the solution of an underlying system is not intuitive and straightforward, we provide a script for Gröbner basis solver generator by Kukelova et al. [64] which can be used to obtain the solver.

4.1 System of polynomial equations

The goal is to solve a system of polynomial equations

$$f_1(\mathbf{x}) = ... = f_m(\mathbf{x}) = 0 \tag{4.1}$$

which is given by a set of m polynomials $F = \{f_1, ..., f_m | \, f_i \in \mathbb{C}[x_1, ..., x_n]\}$ in n variables $\mathbf{x} = (x_1, ..., x_n)$ over the field of complex numbers.

Solving systems of algebraic polynomial equations is a very challenging problem. There doesn't exist one robust, numerically stable and efficient method for solving such systems in general case. Therefore, special algorithms have to be designed for specific problems.

Three important methods for solving systems of polynomial equations, which we will use further in the text, are the Gröbner basis method [26, 62], the Hidden variable resultant method [79, 27, 78, 62, 73, 72], and the polynomial eigenvalue method [9]. We next describe these three methods and propose solvers to our problems based on these methods.

4.2 Hidden variable method

One of the best known techniques for solving systems of polynomial equations is the hidden variable method [27]. The basic idea of this method is to consider one of the variables as a parameter and eliminate other variables from the system. To illustrate how this works, consider the system (4.1) of m polynomial equations in n variables $\mathbf{x} = (x_1, ..., x_n)$. In this system we can regard one of the variables, say x_1, as a parameter. Then, the system of polynomial equations

can be rewritten in a matrix form

$$\mathtt{M}(x_1)\,\mathbf{X} = \mathbf{0}, \tag{4.2}$$

where $\mathtt{M}(x_1)$ is a coefficient matrix depending on the hidden variable x_1, and $\mathbf{X}$ is a vector of all monomials including 1 in the remaining $n-1$ variables (in this case $x_2, x_3, \ldots, x_n$).

It can be easily shown that the monomial 1 is always present in the monomial vector $\mathbf{X}$ in (4.2) if the initial polynomial system (4.1) has a finite number of solutions. Otherwise, there must be a positive power of some variable different from x_1 in every monomial of each polynomial f_i from (4.1), i.e. all monomials have the form $x_1^{\alpha_1} x_2^{\alpha_2} \ldots x_n^{\alpha_n}$, where $\sum_{i=2}^{n} \alpha_i > 0$. In such a case the system (4.1) has an infinite number of solutions. It is because $(c, 0, \ldots, 0)$ for arbitrary $c \in \mathbb{C}$ are solutions of (4.1).

Assume that the number of equations in (4.2) equals the number of monomials in $\mathbf{X}$ (i.e. the matrix $\mathtt{M}(x_1)$ is square). Since we consider only systems with a finite number of solutions, i.e. $\mathbf{X}$ contains 1, the system in (4.2) has non-trivial solution if and only if the matrix $\mathtt{M}(x_1)$ is singular, i.e.

$$det\left(\mathtt{M}(x_1)\right) = 0. \tag{4.3}$$

The "resultant" equation (4.3) is a polynomial in single variable x_1 obtained by eliminating the other $n-1$ variables. This reduces the problem of solving a system of polynomial equations to solving the resultant equation (4.3) in x_1 and back-substituting solutions to $\mathtt{M}(x_1)$. Notice that the matrix $\mathtt{M}(x_1)$ can be even regular for systems with infinite number of solutions and the solution to (4.2) is trivial.

It may happen that the matrix $\mathtt{M}(x_1)$ is not square. Then we can add new equations by taking monomial multiples of the original equations (4.1), to this system. These new equations have the same solutions as the initial polynomial equations and if we are lucky gave us a square matrix. This method can be extended to several hidden variables [27, 62].

4.3 Gröbner basis method

The Gröbner basis method for solving systems of polynomial equations is based on polynomial ideal theory and multivariate polynomial division. It generates special bases of these ideals, called the Gröbner bases [26]. The polynomials

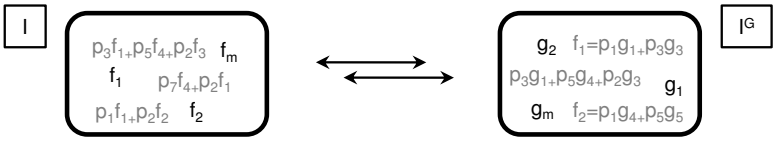
$$F = \{f_1, ..., f_m |\, f_i \in \mathbb{C}\,[x_1, ..., x_n]\}\,, \tag{4.4}$$

from the reference system (4.1) define an *ideal* I, which is a set of all polynomials that can be generated as polynomial combinations of the polynomials in F

$$I = \{\Sigma_{i=1}^{m} f_i\, p_i \,|\, p_i \in \mathbb{C}\,[x_1, ..., x_n]\}\,, \tag{4.5}$$

where p_i are arbitrary polynomials from $\mathbb{C}\,[x_1, ..., x_n]$.

In general, an ideal can be generated by many different sets of generators which all share the same solutions. A question arise whether we can construct the same ideal using a simpler polynomial basis.



Let's first define division by an ideal I in $\mathbb{C}[x_1, ..., x_n]$ as the division by the set F of generators of I. It is known that such multivariate polynomial division depends on the ordering of the polynomials in F and also on the monomial ordering used.

There is a special set of generators G, called a Gröbner basis of the ideal I, for which this division by the ideal I is well defined and doesn't depend on the ordering of the polynomials in G. This means that the remainder of an arbitrary polynomial $f \in \mathbb{C}[x_1, ..., x_n]$ under the division by G is uniquely determined. Furthermore, $f \in I$ if and only if the remainder of f on division by G is zero ($\overline{f}^G = 0$). This implies that for any two polynomials $f, g \in \mathbb{C}[x_1, ..., x_n]$ there holds

$$\overline{f}^G + \overline{g}^G = \overline{f+g}^G, \tag{4.6}$$

$$\overline{\overline{f}^G \overline{g}^G}^G = \overline{fg}^G. \tag{4.7}$$

Thanks to these properties of the Gröbner basis G, we can consider the space of all possible remainders under the division by I. This space is know as a *quotient ring* and we will denote it as $A = \mathbb{C}[x_1, ..., x_n]/I$. It is known that if I is a radical ideal [27] and the set of equations F has a finite number of solutions N, then A is a finite dimensional space with $dim(A) = N$.

In general, an ideal can be generated by the reduced Gröbner basis w.r.t. the lexicographic ordering of monomials, which generates the ideal I but is easy (often trivial) to solve since it contains a single variable polynomial [26]. Computing this basis and "reading off the solutions from it is one standard method for solving systems of polynomial equations [26]. Unfortunately, for larger systems of polynomial equations, and therefore for most computer vision problems, computing the Gröbner basis w.r.t. the lexicographic ordering is not feasible. It is because a computation of Gröbner basis is in general an EXPSPACE-complete problem. It may take very long time to compute this basis and huge space may be necessary for storing intermediate results [60].

Therefore, Gröbner basis solvers usually construct a Gröbner basis G under another ordering, e.g. the graded reverse lexicographic (grevlex) ordering, which is often easier to compute. Then, the properties of the *quotient ring* $A = \mathbb{C}[x_1, \ldots, x_n]/I = \{[f] : f \in \mathbb{C}[x_1, \ldots, x_n]\}$, which is the set of equivalence classes for congruence modulo I, with cosets $[f] = f + I = \{f + h : h \in I\}$ and $[f] = [g] \Leftrightarrow f - g \in I$, are used to get the solutions to the system (4.1) [26]. Usually the remainder of the polynomial f on the division by the Gröbner basis G, denoted as $\overline{f}^G$, is used as a standard representative of the coset $[f] \in A$, i.e. $\left[\overline{f}^G\right] = [f]$.

In the quotient ring A, the multiplication (action) matrix $\mathtt{M}_p$ [95] is constructed. It is the matrix of the linear operator $T_p\colon A \to A$ of the multiplication by a suitably chosen polynomial p w.r.t. some basis $B = \{[b_1], \ldots, [b_s]\}$ of A. Usually, the standard monomial basis $B = \left\{[\mathbf{x}^\alpha] : \overline{\mathbf{x}^\alpha}^G = \mathbf{x}^\alpha\right\}$ of A is used. Here $\mathbf{x}^\alpha$ represents a monomial $\mathbf{x} = x_1^{\alpha_1} x_2^{\alpha_2} ... x_n^{\alpha_n}$.

Then we can represent the multiplication mapping T_p by representing the image $T_p([b_i])$ of every basis element $[b_i]$, $i = 1 \ldots, s$ in terms of $B = \{[b_1], \ldots, [b_s]\}$

$$T_p([b_j]) = [p][b_j] = [pb_j] = \sum_{i=1}^{s} m_{ij}[b_i], \tag{4.8}$$

with $s \times s$ multiplication (action) matrix $\mathtt{M}_p := (m_{ij})$.

The solutions to the system of equations (4.1) can be read off directly from the eigenvalues and the eigenvectors of the action matrix (4.8). Therefore this action matrix can be viewed as a generalization of the companion matrix used in solving one polynomial equation in one unknown [26].

An important observation was used in [68]. This observation tells us, that we can construct the action matrix without explicitly computing a complete Gröbner basis G. All we need is to construct polynomials from the ideal I with leading monomials from the set $p \cdot B \setminus B$ and the remaining monomials from B. For more details about the form of these polynomials see [62, 68].

Since these polynomials are from the ideal I, we can generate them as polynomial combinations of the initial generators F. This can be done using several methods. One possible way is to start with F and then systematically generate new polynomials from I by multiplying already generated polynomials by individual variables and reducing them each time by the Gauss-Jordan (G-J) elimination. This method was, for example, used in [68] and resulted in several G-J eliminations. Another possible way is to generate all new polynomials in one step by multiplying polynomials from F with selected monomials and reducing all generated polynomials at once using one G-J elimination. This method was used in [19] and proved to be numerically more stable. Therefore, we use this method to construct the action matrix for our problems. In [12] we further extended this method by reducing the number of generated polynomials. This is done in a simple and intuitive way. Imagine that we have a set of monomials which should be used for multiplying initial polynomials F and in this way generate new polynomials. In most cases, this set of monomials contains all monomials up to some degree. All generated polynomials including initial polynomial equations F can be written in a matrix form

$$\mathtt{M}\,\mathbf{X} = \mathbf{0}, \tag{4.9}$$

where $\mathtt{M}$ is the coefficient matrix and $\mathbf{X}$ is the vector of all monomials. Consider that we know that after G-J elimination of this matrix $\mathtt{M}$ we obtain all polynomials that we need for constructing the action matrix. Since we know how these necessary polynomials should look (i.e. which monomials they contain), we can systematically reduce the number of generated polynomials in the following way:

1. For all rows from $\mathtt{M}$ starting with the last row r (with the polynomial of highest degree) do
   a) Perform G-J elimination on the matrix $\mathtt{M}$ without the row r.
   b) If the eliminated matrix contains all necessary polynomials $\mathtt{M} := \mathtt{M} \setminus r$.
   c) Go to step 1.

This elimination procedure is performed only once in the offline process. The online solver works with the eliminated set of polynomials which speeds up the process of creating the action matrix and also improves the numerical stability.

Naroditsky et al. describe in [83] another but similar method for reducing the number of polynomials in a Gröbner basis solver. In [17, 19, 20] authors show how to improve stability of such Gröbner basis solvers.

4.4 Polynomial eigenvalue method

This method is in some sense more straightforward and easier to implement than the Gröbner basis method since eigenvalue problems are well studied, easy to understand, and efficient and robust algorithms for solving these problems [9] can be directly used to solve concrete computer vision problems. The polynomial eigenvalue method is similar to the Hidden variable method presented in Section 4.2. This method also rewrites the input system (4.1) to the form (4.2). However, instead of solving the polynomial determinant it solves the system by transforming it to an eigenvalue problem.

Polynomial eigenvalue problems (PEP) are problems of the form

$$\mathtt{C}(\lambda)\,\mathbf{v} = \mathbf{0}, \tag{4.10}$$

where $\mathbf{v}$ is a vector of monomials in all variables except for λ and $\mathtt{C}(\lambda)$ is a matrix polynomial in variable λ defined as

$$\mathtt{C}(\lambda) \equiv \lambda^l \mathtt{C}_l + \lambda^{l-1}\mathtt{C}_{l-1} + \cdots + \lambda\mathtt{C}_1 + \mathtt{C}_0, \tag{4.11}$$

with $n \times n$ coefficient matrices $\mathtt{C}_j$ [9].

Polynomial eigenvalue problems (4.10) can be transformed to the standard generalized eigenvalue problems (GEP)

$$\mathtt{A}\,\mathbf{y} = \lambda\,\mathtt{B}\,\mathbf{y}, \tag{4.12}$$

which are well studied problems and there exist many efficient numerical algorithms for solving them [9]. To see how a PEP (4.10) can be transformed to a GEP (4.12), let us first consider an important class of polynomial eigenvalue problems, the quadratic eigenvalue problems (QEP) of the form

$$\left(\lambda^2\mathtt{C}_2 + \lambda\mathtt{C}_1 + \mathtt{C}_0\right)\mathbf{v} = \mathbf{0}, \tag{4.13}$$

where $\mathtt{C}_2, \mathtt{C}_1$ and $\mathtt{C}_0$ are coefficient matrices of size $n \times n$, λ is a variable called an eigenvalue and $\mathbf{v}$ is a vector of monomials in all variables except λ, called an eigenvector.

QEP (4.13) can be transformed to the generalized eigenvalue problem (4.12) with

$$\mathtt{A} = \begin{bmatrix} \mathtt{0} & \mathtt{I} \\ -\mathtt{C}_0 & -\mathtt{C}_1 \end{bmatrix}, \mathtt{B} = \begin{bmatrix} \mathtt{I} & 0 \\ 0 & \mathtt{C}_2 \end{bmatrix}, \mathbf{y} = \begin{bmatrix} \mathbf{v} \\ \lambda\mathbf{v} \end{bmatrix}. \tag{4.14}$$

Here 0 and $\mathtt{I}$ are $n \times n$ null and identity matrices, respectively. GEP (4.12) with matrices (4.14) gives equations $\lambda\mathbf{v} = \lambda\mathbf{v}$ and $-\mathtt{C}_0\mathbf{v} - \lambda\mathtt{C}_1\mathbf{v} = \lambda^2\mathtt{C}_2\mathbf{v}$, which is equivalent to (4.13). Note that

this GEP (4.14) has $2\,n$ eigenvalues and therefore by solving it we obtain $2\,n$ solutions to the QEP (4.13). Higher order eigenvalue problems can be transformed to GEP similarly and there is a nice relationship to constructing Frobenius companion matrices [26]. For more details refer to [67, 62].

If we are lucky, then equations (4.1) can directly be rewritten to a polynomial eigenvalue problem for some x_j, let say x_1,

$$\mathtt{C}\,(x_1)\,\mathbf{v} = \mathbf{0}, \tag{4.15}$$

where $\mathtt{C}\,(x_1)$ is a matrix polynomial (4.11) with square $m \times m$ coefficient matrices $\mathtt{C_j}$ and $\mathbf{v}$ is a vector of $s = m$ monomials in variables $x_2, \ldots, x_n$, i.e. monomials of the form $\mathbf{x}^\alpha = x_2^{\alpha_2} x_3^{\alpha_3} ... x_n^{\alpha_n}$.

Unfortunately, not all systems of polynomial equations (4.1) can be directly transformed to a PEP (4.15) for some x_j. Then, new polynomials have to be added in order to make matrices square. We show one possible way how to do this in Section 6.2. More can be found in [67, 62].

4.5 Automatic solver generator

In [64] we presented an automatic method for creating solvers for a given system of polynomial equations with symbolic coefficients. This method constructs a specialized efficient solver for a given symbolical system of polynomial equations based on Gröbner basis method described in Section 4.3. This is possible since many computer vision problems share the convenient property that the monomials which appear in the set of initial polynomials (4.1) are always the same irrespectively from the concrete coefficients arising from non-degenerate image measurements. It is not necessary to use general algorithms [26] for constructing Gröbner bases when solving these problems.

Therefore, most Gröbner basis solvers for computer vision problems consist of two phases. In the first "offline" phase, so-called "elimination templates" are found. These templates say which input polynomials should be multiplied with which monomials and then eliminated to obtain all polynomials from the grevlex Gröbner basis or at least all polynomials necessary for constructing an action matrix. This phase is for a one concrete problem performed only once.

The second "online" phase is much faster and uses elimination templates to construct the action matrix and to solve the input problem. In Chapter 9 we show how to speed-up most of solvers generated using the automatic generator [64].

Block-diagram process of creating Gröbner bases is shown in Figure 4.1. For more details please refer to [64].

4.6 Random instances in a finite prime field

The important part of the automatic generator pipeline (Figure 4.1) is the part where an instance of the input problem is created in a finite prime field $\mathbb{Z}_p$. In [64] we set random values to input parameters of the given problem. In this thesis we will however need to create random instances for cases where input parameters are not independent variables but variables with some hidden

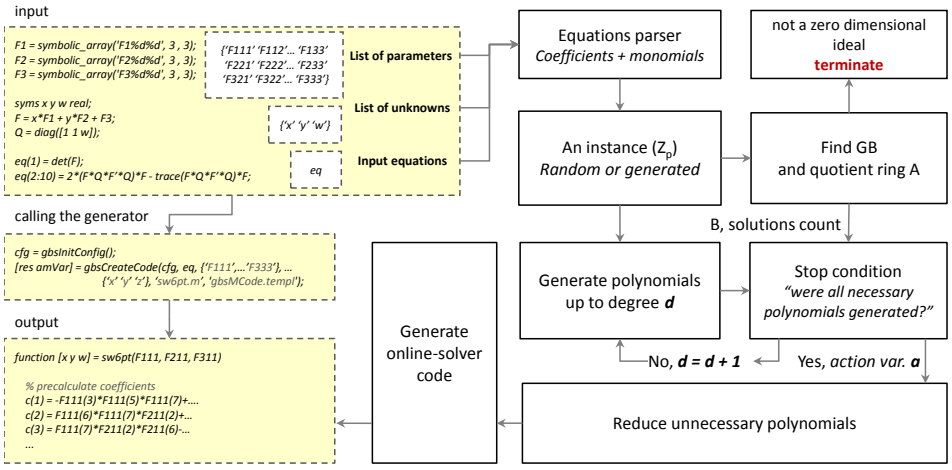


Figure 4.1: Automatic solver generator pipeline.

or latent relations. Therefore, setting completely random values to parameters of the problem would not preserve these latent relations and it should happen that we are solving some different problem, maybe without a solution. In fact, the random generator module used in the automatic generator [64] is just a simple function which creates equations such that all parameters are reasonably replaced with some random values.

```
function [eqi] = gbs_RandomInstanceZp(cfg, eq, known, unknown)
...
end
```

This function can be replaced with a custom random instance generator. Next we describe blocks which we will use later to create random instances which preserve latent relations between parameters. We show how to create proper $\mathbb{Z}_p$ instances, i.e. how to create a random camera, how to project 3D points, how to generate 3D scene, how to distort them in $\mathbb{Z}_p$ field etc.

4.6.1 Basic finite prime field arithmetics

Basics of finite field algebra are explained in [74]. We focus on how to algorithmically calculate basic elements such as inverse with respect to multiplication, how to find the matrix inverse, how to perform square root, project points, and so on.

A special attention must be paid to avoid integer overflow, especially when a bigger prime number p is used. In our experiments we use $p = 30097$ which fits into $15bits$. Hence the product of two such big numbers can fit into single $32bits$ integer. Since $Matlab$ uses double arithmetics we cannot make product of more than four such big numbers without rounding. Hence, modulo, i.e. $(\mathrm{mod}\ p)$, should be applied on intermediate results to avoid overflows.

Calculating scalar inverse in $\mathbb{Z}_p$

Let a be an integer and let p be a prime number. The Euclidean algorithm for finding the greatest common divisor of integers a and p, but also other algorithms [59], produce as a side product, two integers x and y (one of which is typically negative) that satisfy Bzout's identity

$$ax + py = \gcd(a, p). \tag{4.16}$$

The greatest common divisor (GCD) of a and p is one since p is the prime number. Hence congruence modulo p we get

$$1 = ax + py, \tag{4.17}$$

$$1 = ax \pmod{p}, \tag{4.18}$$

and thus $x \pmod{p}$ is the multiplicative inverse of a.

Gauss-Jordan elimination in $\mathbb{Z}_p$

From the algebra, inverse to additive operator for a given scalar value a is directly $(p - a) \pmod{p}$. Division a/b we can perform simply by multiplying a with multiplicative inverse of b, i.e. ab^{-1}. Now we are equipped with all operations that we need to perform Gauss-Jordan elimination, exactly in the same way as we would do it in the floating-point arithmetic, just by replacing arithmetics from the real field $\mathbb{R}$ with arithmetics from $\mathbb{Z}_p$ field. Intermediate results should be replaced with their $(\mathrm{mod}\ p)$ after every multiplication or division.

Matrix inverse in $\mathbb{Z}_p$

Gauss-Jordan elimination applied on a square matrix can be used to calculate the matrix inverse. This can be done by augmenting the square matrix with the identity matrix `I` of the same dimensions. Let's multiply such augmented matrix from the left with inverse to the matrix `M`:

$$[\,\mathtt{M}\,|\,\mathtt{I}\,] \Rightarrow \mathtt{M}^{-1}[\,\mathtt{M}\,|\,\mathtt{I}\,] \Rightarrow [\,\mathtt{I}\,|\,\mathtt{M}^{-1}\,]. \tag{4.19}$$

For a non-singular matrix `M`, there is just one way how can be matrix $[\,\mathtt{M}\,|\,\mathtt{I}\,]$ reduced to $[\,\mathtt{I}\,|\,\mathtt{M}^{-1}\,]$. Hence in order to calculate matrix inverse in the finite prime field $\mathbb{Z}_p$, it is sufficient to create the augmented matrix $[\,\mathtt{M}\,|\,\mathtt{I}\,]$, perform Gauss-Jordan elimination in $(\mathrm{mod}\ p)$ and read the right hand side of the reduced matrix $[\,\mathtt{I}\,|\,\mathtt{M}^{-1}\,]$.

Square root in $\mathbb{Z}_p$

It is known that there is no square root for each number in finite prime fields. Since we do not calculate square roots very frequently, we've chosen a very simple strategy for finding the square root. Consider that we want to find a square root of a number a. Basically, for every number in the interval $i \in \{1 \ldots p\}$ we simply verify whether $i^2 \pmod{p} - a = 0$. If we could not find such number i, then we indicate failure.

4.6.2 Random rotations in a finite prime field

For creating a rotation matrix we use Euler angles representation and create elementary rotation matrices rotating an arbitrary point around X, Y and Z axis:

$$R = \begin{bmatrix} c_\alpha & -s_\alpha & 0 \\ s_\alpha & c_\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_\beta & 0 & -s_\beta \\ 0 & 1 & 0 \\ s_\beta & 0 & c_\beta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_\gamma & -s_\gamma \\ 0 & s_\gamma & c_\gamma \end{bmatrix}, \tag{4.20}$$

where c_x respectively s_x stand for $\cos(x)$ respectively $sin(x)$.

In order to create correct rotation matrices, $c_x^2 + s_x^2 = 1$ must hold. This becomes tricky in $\mathbb{Z}_p$ since it is not easy to create $\sin$ and $\cos$ functions. On the other hand, we can use the above relation, randomly choose c_x value and calculate the s_x i.e $s_x = \sqrt{1 - c_x} \pmod p$. However it may happen that it was not possible to calculate the square root. Then we simply randomly assign a new value to c_x and try again. The following $Matlab$ code describes this procedure:

```
err = 1;
while err ~= 0

    c_x = floor(rand()*prime);
    [s_x err] = sqrtZp(mod(1 - c^2, prime), prime);
end
```

Once correct c_x and s_x values are found, rotation matrices (4.20) can be build easily. To avoid overflows during matrix multiplication it is important to modulo intermediate results after every scalar multiplication.

4.6.3 Creating a 3D scene in a finite prime field

Now it should be straightforward to create a synthetic 3D scene, a camera and 2D projections. In fact, we just need to create a camera matrix and 3D points. Then we simply project 3D points to the image plane.

Random 3D structure in $\mathbb{Z}_p$

The easy part of the scene generator is to creation of a 3D scene structure with N 3D points. In fact we just randomly distribute points in the 3D space using integer arithmetics within the range of the prime number and extend them to homogeneous coordinates.

```
M = floor( prime*rand(3, N) );
M(4, :) = 1;
```

Random camera in $\mathbb{Z}_p$

Let's create a random pinhole camera in $\mathbb{Z}_p$. First, we create a random 3D rotation matrix `R` using Euler angles as described in Section 4.6.2. Then, a camera position vector $\mathbf{t}$ is randomly chosen and finally a calibration matrix with a randomly selected focal length is created, i.e.

$$\mathtt{K} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{4.21}$$

The final camera projection matrix is created as $\mathtt{P} = \mathtt{K}\,[\mathtt{R}|\mathbf{t}]$.

```
R = CreateRandomRotationMatrixZp( prime );
t = floor( prime*rand( 3, 1 ) );
focal = floor( prime*rand );
% compose camera matrix
P = mod( diag( [focal focal 1] )*[R t], prime );
```

Notice the modulo which is called after multiplying camera calibration matrix with camera rotation and translation matrix. This is necessary to avoid integer overflows during the calculations.

Consistent 2D measurements in $\mathbb{Z}_p$

In order to create 2D measurements consistent with the 3D scene and the camera matrix we need to project 3D points to the camera image plane.

```
m = mod( P*M, prime );

% project to the camera image plane
for i = 1:N
    m(:, i) = mod( InvZp( m(3, i), prime ) * m(:, i), prime );
end
```

The above code transforms 3D points using matrix `P` into the camera coordinate frame. Then, every point is projected to the camera image plane. Note that instead of dividing coordinates $[x_i,\ y_i,\ w_i]^T$ by its homogeneous component w_i we multiply x_i and y_i with the inverse (`InvZp`) of it.

Radial distortion in $\mathbb{Z}_p$

Now consider image measurements distorted by some amount of radial distortion. Assume the division model [38] where the relationship between undistorted point $[x_u,\ y_u,\ 1]$ and distorted

point $[x_d,\ y_d,\ 1]$ can be expressed as

$$[x_u,\ y_u,\ 1] = \left[x_d,\ y_d,\ 1 + k\left(x_d^2 + y_d^2\right)\right], \tag{4.22}$$

where k is the distortion parameter.

One possible way of creating the distorted image is to distort image measurements, which we obtain by projecting 3D structure to the 2D image plane. We selected an alternative way since for distorting image points we need to calculate the inverse of (4.22). The inverse, i.e. the "distortion" function, contains square root and therefore it many times fails in $\mathbb{Z}_p$. It is therefore better to create randomly distorted 2D image points and then undistort them as follows

```
r = InvZp(mod(1 + r*mod(xd^2+yd^2, prime), prime), prime);
xu = mod(xd*r, prime);
yu = mod(yd*r, prime);
```

Finally, we get the 3D structure of the scene by back-projecting 2D points.

5 Absolute pose for a calibrated camera

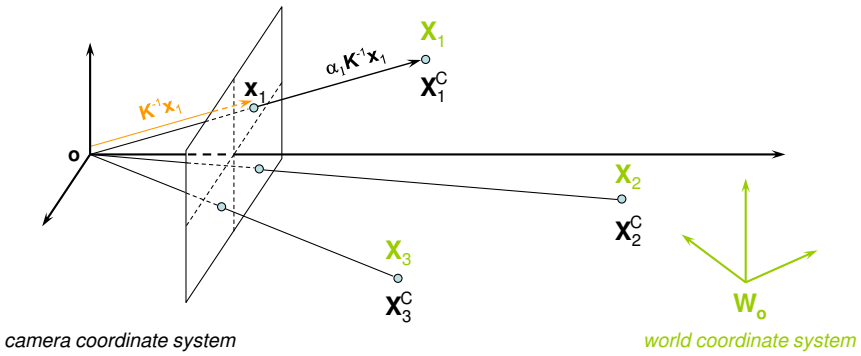


Figure 5.1: Camera pose problem illustration. Reference points $\mathbf{X}_i$ in the world coordinated system are projected to points $\mathbf{x}_i$ on the camera image plane.

Calculating the pose of a calibrated camera is one of the oldest computer vision problem dating to 1841 [42] but still a very important element of many computer vision algorithms. With modern cameras this problem appears more and more important since cameras become precise enough and allow approximating calibration parameters with zero skew, unit aspect ratio and principal point in the middle of the image. Moreover camera focal length is usually stored with the image data. Hence a good approximation of calibration parameters is provided for each image.

5.1 Problem formulation

The problem itself is a very well studied with tens of solutions. Figure 5.1 illustrates the problem. It is known that three point correspondences are sufficient to recover the camera rotation and translation in the case of calibrated camera – hence P3P– and there are up to four real solutions to the problem [47]. This problem can be formulated as follows.

Problem 2. *Given an upper triangular calibration matrix* $\mathtt{K}$ *[47] and a set of reference 3D points* $\mathbf{X}_i$ *corresponding to their images* $\mathbf{x}_i$ *measured on an image plane, we are looking for the camera rotation matrix* $\mathtt{R}$ *and the camera translation vector* $\mathbf{t}$ *satisfying the following relation for all input points:*

$$\alpha_i \mathbf{x}_i = \mathtt{K} \left[\mathtt{R} | \mathbf{t} \right] \mathbf{X}_i, \tag{5.1}$$

where α_i *are unknown scalar factors.*

Unless stated otherwise we will represent points by their homogeneous coordinates through this thesis.

5.2 Angles and Distances. Cosine law

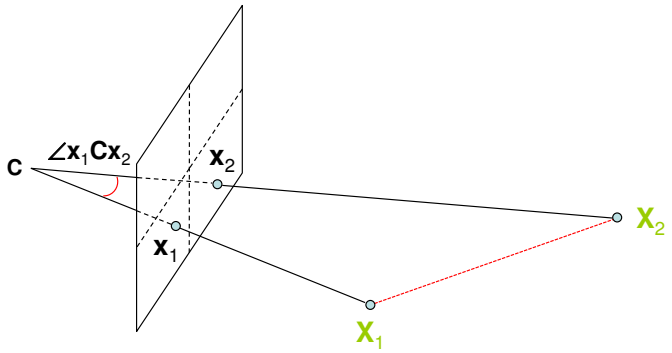


Figure 5.2: Illustration of the cosine law formula application to the P3P problem.

Another point of view at the problem is based on known distances and angles between measured points [42, 37]. Cosine law formula can greatly help to this view and the P3P problem can be formulated as follows.

Problem 3. *Given a calibrated camera positioned at C and correspondences $\mathbf{X}_i \leftrightarrow \mathbf{x}_i$ between 3D reference points and their images, we are looking for unknown camera-point distances $l_i = \|C - x_i\|$ and $l_j = \|C - x_j\|$, such that*

$$d_{ij} = l_i^2 + l_j^2 - 2\, l_i\, l_j\, cos\, \phi_{ij}, \tag{5.2}$$

where d_{ij} is a distance between i^{th} and j^{th} 3D reference points X_i and X_j, ϕ_{ij} is viewing angle subtended at the camera center by the i^{th} and j^{th} measured points.

See Figure 5.2 for an illustration. Equation (5.2) is an application of the cosine laws formula in the triangle given by the camera center and passing through image measurements. Note that l_i and l_j are the only unknowns in (5.2). Each pair of 2D-to-3D correspondences gives a singe constraint on the distance. Cosine of the viewing angle can be computed directly from the coordinates of image points once normalized ray of sights are calculated from image points using known calibration matrix $\mathtt{K}$:

$$cos\, \phi_{ij} = \frac{(\mathtt{K}^{-1}x_i)^\top(\mathtt{K}^{-1}x_j)}{\|\mathtt{K}^{-1}x_i\|\|\mathtt{K}^{-1}x_j\|}. \tag{5.3}$$

Given three 2D-to-3D point correspondences we get three 2^{nd} degree polynomial equations (5.2) in three unknown distances l_i. These equations can be solved using different methods for

solving systems of polynomial equations [62] or they can be transfered to a 4^{th} degree polynomial in a single variable by applying a set of clever operations and substitutions [42, 37]. In fact, different solutions to this system were published in the past [44]. Once the distances l_i are calculated we recover positions of the three 3D points in the camera coordinate system. Then, the camera matrix can be calculated simply by finding a rigid transformation between points in the camera coordinate system and points in the world coordinate system. It will be described in Section 5.3. Alternatively, in [37, 44], three distances l_i are used to calculate the camera center directly in the world coordinate system as an intersection of three spheres centered in 3D points $\mathbf{X}_i$. The local coordinate system of the camera is calculated later using rays of sight vectors.

Besides the cosine law mentioned above, many other formulations and solutions to the problem were developed in the past. It is quite complicated to extend the cosine law solution for a more complicated camera models. In next sections we describe different formulations and solutions to P3P problem which we further extend to cope with more complicated camera models Chapters 6 and 7. We show how to solve the absolute pose problem for a camera with an unknown focal length, camera with unknown focal length and unknown radial distortion in camera optics. Further we derive a solution for a camera with a known up direction, e.g. known from an inertia sensor or from a single known vanishing point.

5.3 Camera rigid motion

When looking back to the cosine-law solution of P3P it is interesting to observe that the camera matrix $[\mathtt{R}|\mathbf{t}]$ is not so important for solving P3P problem. Clearly, given depths of points, i.e. α_i factors in Equation (5.1), we can directly recover reference 3D points $\mathbf{X}_i$ in the camera coordinate system as

$$\alpha_i \mathtt{K}^{-1}\mathbf{x}_i = \mathbf{X}_i^C. \tag{5.4}$$

Then the camera pose, the rotation matrix $\mathtt{R}$ and the translation vector $\mathbf{t}$ can be directly recovered as a system of linear equations coming from the relation

$$\mathbf{X}_i^C = [R\,|\,t]\,\mathbf{X}_i. \tag{5.5}$$

Finding a rigid transformation between two sets of points is a well studied problem and optimal closed form solutions were already developed e.g. [7, 52, 32]. Hence the real question is how to calculate the α_i factors. Note that it also simplifies the problem since instead of calculating three unknown rotation angles and three scalars of the translation vector we need to recover just three unknown depths α_i. Next we describe three different formulations of P3P problem which can be easily used to compute the scalar factors α_i and hence to solve the whole P3P problem.

5.3.1 Distance between points

Distance between points is a popular invariant preserved under an unknown rigid transformation [47].

For P3P problem we have three reference 3D points in the world coordinate system. Since the camera motion is a rigid transform, distances between these 3D points in the world coordinate system and the camera coordinate system are preserved. Hence:

$$d_{ij}^2 = \|\mathbf{X}_i - \mathbf{X}_j\|^2 = \|\mathbf{X}_i^C - \mathbf{X}_j^C\|^2 = \|\alpha_i \mathtt{K}^{-1}\mathbf{x}_i - \alpha_j \mathtt{K}^{-1}\mathbf{x}_j\|^2. \tag{5.6}$$

Let's denote direction vectors of the camera rays toward the reference points (uncalibrated image measurements) as

$$\mathbf{x}_i^C = \mathtt{K}^{-1}\mathbf{x}_i = \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix}. \tag{5.7}$$

Now, given three 2D measurements $\mathbf{x}_i^C$ and three 3D points we obtain three 2^{nd} degree polynomial equations in three unknown depths – $\alpha_1, \alpha_2, \alpha_3$. By substituting (5.4 and 5.7) into Equation (5.6) we obtain

$$d_{ij}^2 = \|\mathbf{X}_i - \mathbf{X}_j\|^2 = \|\alpha_i \mathbf{x}_i^C - \alpha_j \mathbf{x}_j^C\|^2. \tag{5.8}$$

By further expansion of (5.8) we get

$$\begin{aligned} (\alpha_1 u_1 - \alpha_2 u_2)^2 + (\alpha_1 v_1 - \alpha_2 v_2)^2 + (\alpha_1 - \alpha_2)^2 - d_{12}^2 = 0, \\ (\alpha_1 u_1 - \alpha_3 u_3)^2 + (\alpha_1 v_1 - \alpha_3 v_3)^2 + (\alpha_1 - \alpha_3)^2 - d_{13}^2 = 0, \\ (\alpha_2 u_2 - \alpha_3 u_3)^2 + (\alpha_2 v_2 - \alpha_3 v_3)^2 + (\alpha_2 - \alpha_3)^2 - d_{23}^2 = 0. \end{aligned} \tag{5.9}$$

Bezout bound gives us that these three simple quadratic equations in three unknowns have at most $8 = 2 \times 2 \times 2$ solutions. However, since they have no linear terms, $\alpha_i \rightarrowtriangle -\alpha_i$ preserves the form, and the eight solutions occur in four pairs. Also note that only positive α_i are reasonable since they represent depths. System of equations (5.9) can be easily solved using the hidden variable method from Section 4.2, by manipulating with equations and using substitutions or just by putting these equations into the automatic generator [64] to solve them using the Gröbner basis method. A script for the solver generator is in Figure 5.3. Generated Gröbner basis solver returns all eight solutions and hence it is slower compared to the other solutions described in this chapter which compute only four solutions.

5.3.2 Ratios of distances

Ratios of distances is another good invariant preserved under a rigid transformation. First, let us return to Equation (5.8). Since all α_i are nonzero and positive, we can fix one of depths, e.g. α_1, and express all remaining depths α_i as $\alpha_i = \alpha_1 \lambda_i$. Note that $\lambda_1 = 1$. Then, Equation (5.8) can be further expanded to:

$$\begin{aligned} d_{ij}^2 = \|\mathbf{X}_i - \mathbf{X}_j\|^2 &= \|\alpha_1\,\lambda_i\,\mathbf{x}_i^C - \alpha_1\,\lambda_j\,\mathbf{x}_j^C\|^2 \\ &= \alpha_1^2\,\|\lambda_i\,\mathbf{x}_i^C - \lambda_j\,\mathbf{x}_j^C\|^2, \end{aligned} \tag{5.10}$$

```
m = gbs_Matrix('m', 2, 3); % three 2D measurements
d = gbs_Vector('d', 3);    % three known 3D distances
syms la lb lc real;        % three unknown lambda factors

a = [m(1,1) m(2,1) 1];
b = [m(1,2) m(2,2) 1];
c = [m(1,3) m(2,3) 1];
AB2 = sum((la*a - lb*b).^2);
AC2 = sum((la*a - lc*c).^2);
BC2 = sum((lb*b - lc*c).^2);

eq(1) = AB2 - d(1);        % three polynomial equations
eq(2) = AC2 - d(2);
eq(3) = BC2 - d(3);

unknown = {'la' 'lb' 'lc'};
known = {'m11', 'm12', 'm21', 'm22', 'm31', 'm32',
                                            'd1', 'd2', 'd3'};
[res export] = gbs_CreateCode('p3p_solver', eq,
                                            known, unknown);
```

Figure 5.3: A script for generating P3P solver using the distance invariant for our automatic generator [64].

for every $i \neq j$. If all 3D reference points are distinct, then for each $i \neq j$ we have both the left and the right side of Equation (5.10) nonzero and positive. Hence we can form the following ratios

$$r_{ijkl} = \frac{d_{ij}^2}{d_{kl}^2} = \frac{\|\mathbf{X}_i - \mathbf{X}_j\|^2}{\|\mathbf{X}_k - \mathbf{X}_l\|^2} = \frac{\alpha_1^2 \, \|\lambda_i \, \mathbf{x}_i^C - \lambda_j \, \mathbf{x}_j^C\|^2}{\alpha_1^2 \|\lambda_k \, \mathbf{x}_k^C - \lambda_l \, \mathbf{x}_l^C\|^2}. \tag{5.11}$$

in which α_1^2 on the right hand side cancels, thus yielding

$$r_{ijkl} \, \|\lambda_k \, \mathbf{x}_k^C - \lambda_l \, \mathbf{x}_l^C\|^2 = \|\lambda_i \, \mathbf{x}_i^C - \lambda_j \, \mathbf{x}_j^C\|^2. \tag{5.12}$$

This simple trick removes one of the unknown depths, α_1 in this particular example. Hence, given three 2D-to-3D point correspondences we get just two distinct 2^{nd} degree polynomial equations in two unknowns λ_1 and λ_2:

$$r_{1213}((u_1 - \lambda_3 u_3)^2 + (v_1 - \lambda_3 v_3)^2 + (1 - \lambda_3)^2) - ... \tag{5.13}$$
$$...(u_1 - \lambda_2 u_2)^2 - (v_1 - \lambda_2 v_2)^2 - (1 - \lambda_2)^2 = 0,$$
$$r_{2313}((u_1 - \lambda_3 u_3)^2 + (v_1 - \lambda_3 v_3)^2 + (1 - \lambda_3)^2) - ... \tag{5.14}$$
$$...(\lambda_2 u_2 - \lambda_3 u_3)^2 - (\lambda_2 v_2 - \lambda_3 v_3)^2 - (\lambda_2 - \lambda_3)^2 = 0.$$

Sylvester resultant [26] is a good choice for solving these two equations since there are only two unknowns. This system has Bezout bound of $4 = 2 \times 2$ solutions. Figure 5.4 shows a script for our automatic generator [64] which generates a source code solving these equations. The generated solver returns all four solutions to the problem.

```
m = gbs_Matrix('m', 2, 3);          % three 2D measurements
r = gbs_Vector('r', 2);             % two known ratios
syms lb lc real;                    % two unknown lambda factors

a = [m(1,1) m(2,1) 1];
b = [m(1,2) m(2,2) 1];
c = [m(1,3) m(2,3) 1];
AB2 = sum((a - lb*b).^2);
AC2 = sum((a - lc*c).^2);
BC2 = sum((lb*b - lc*c).^2);

eq(1) = AB2 - AC2 * r(1);           % two polynomial equations
eq(2) = BC2 - AC2 * r(2);

unknown = {'lb' 'lc'};
known = {'m11', 'm12', 'm21', 'm22', 'm31', 'm32', 'r1', 'r2'};
[res export] = gbs_CreateCode('p3p_solver_ratio', eq,
                                                  known, unknown);
```

Figure 5.4: A script generating P3P solver using ratios of distances invariant for our automatic generator [64].

5.4 Homography and camera absolute pose

Plane to plane homographies are well known for transferring images of points lying on a plane between images [47]:

$$\lambda \mathbf{x} = \mathtt{H}\mathbf{y}. \tag{5.15}$$

Since three points are always on a plane, we can calculate a homography between reference 3D points and their images in 2D. It is known [47] that four points are sufficient to calculate a homography using a linear algorithm [47]. We have only three point correspondences but our camera is calibrated and this information gives us an additional constraint. It is known that 3×3 homography matrix encodes camera parameters as well [47]. Let's investigate the structure of the homography matrix and see how to read rotation and translation parts of the camera from this homography matrix.

Without loss of generality we can assume that one of 3D points is at the origin, one at $[1, 0, 0]$ and all three are on the plane $z = 0$. We can always transform points to achieve that. Equation (5.1) then becomes:

$$\alpha_i \mathbf{x_i} = \mathtt{K}\,[\mathtt{R}|\mathbf{t}] \begin{bmatrix} x \\ y \\ 0 \\ 1 \end{bmatrix}. \tag{5.16}$$

It is clear that the third column of the rotation matrix R vanishes, and a 3×3 homography matrix H is sufficient to capture the relation between points on such 3D plane and on the camera image plane:

$$\alpha_i \mathbf{x}_i = \mathtt{K}\,[\mathbf{r}_1 \mathbf{r}_2 \mathbf{t}] \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \tag{5.17}$$

where $\mathbf{r}_i$ denotes the i^{th} column of the rotation matrix R. The third column of the rotation matrix can be easily computed as the cross product of first two matrix columns, i.e. $\mathbf{r_3} = \mathbf{r_1} \times \mathbf{r_2}$.

Since the camera calibration K is known we can further expand Equation (5.17) to:

$$\alpha_i \mathbf{x}_i^C = \mathtt{H}\mathbf{X}_i = [\mathbf{r}_1 \mathbf{r}_2 \mathbf{t}] \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}. \tag{5.18}$$

The last equation shows that the homography H contains two columns of the camera rotation matrix and the camera translation vector. Both are scaled by an unknown scalar value. By moving one of the 3D points to the origin and one to $[1, 0, 0]^\top$ we get:

$$\alpha_1 \mathbf{x}_1^C = [\mathbf{r}_1 \mathbf{r}_2 \mathbf{t}] \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \mathbf{t}, \tag{5.19}$$

$$\alpha_2 \mathbf{x}_2^C = [\mathbf{r}_1 \mathbf{r}_2 \mathbf{t}] \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} = \mathbf{r}_1 + \mathbf{t} = \mathbf{r}_1 + \alpha_1 \mathbf{x}_1^C. \tag{5.20}$$

Since the overall scale of the homography is not important we can fix scale so that $\alpha_1 = 1$. Then, homography H has the following form:

$$\mathtt{H} = \left[\left(\alpha_2 \mathbf{x}_2^C - \mathbf{x}_1^C \right) \;\; \mathbf{r}_2 \;\; \mathbf{x}_1^C \right]. \tag{5.21}$$

Homography H is now parametrized using the unknown scale factor α_2, and three values of the 2^{nd} column of the rotation matrix R. We can further reduce the number of unknowns by using the third correspondence and express the 2^{nd} column of the rotation matrix as a function of unknown α_2 and α_3:

$$\alpha_3 \mathbf{x}_3^C = \left[\left(\alpha_2 \mathbf{x}_2^C - \mathbf{x}_1^C \right) \;\; \mathbf{r}_2 \;\; \mathbf{x}_1^C \right] \begin{bmatrix} a \\ b \\ 1 \end{bmatrix} = a \left(\alpha_2 \mathbf{x}_2^C - \mathbf{x}_1^C \right) + b \mathbf{r}_2 + \mathbf{x}_1^C, \tag{5.22}$$

hence:

$$\mathbf{r}_2 = 1/b \left(\alpha_3 \mathbf{x}_3^C - a \left(\alpha_2 \mathbf{x}_2^C - \mathbf{x}_1^C \right) - \mathbf{x}_1^C \right). \tag{5.23}$$

Note that b component of the 3D point $\mathbf{X}_3$ must be non-zero. Otherwise all three 3D points $\mathbf{X}_1$, $\mathbf{X}_2$ and $\mathbf{X}_3$ lay on a line and camera pose cannot be uniquely determined.

Let's denote $\mathbf{c_a} = -a/b\,\mathbf{x}_2^C$, $\mathbf{c_b} = 1/b\,\mathbf{x}_3^C$ and $\mathbf{c_c} = 1/b\,\left(a\,\mathbf{x}_1^C - \mathbf{x}_1^C \right)$, then the homography $\mathtt{H}$ becomes:

$$\mathtt{H} = \left[\left(\alpha_2 \mathbf{x}_2^C - \mathbf{x}_1^C \right) \;\; \left(\alpha_2 \mathbf{c_a} + \alpha_3 \mathbf{c_b} + \mathbf{c_c} \right) \;\; \mathbf{x}_1^C \right]. \tag{5.24}$$

Note that until know we did not assume any specific property of our homography, such as it consist of the rotation and the translation components. Above equations hold for any general homography as well. Given an additional 4^{th} point we can compute unknown alpha factors linearly.

Returning back to original P3P problem, we know that our homography has a specific form, like outlined in Equation (5.17). The rotation part of the homography gives two additional non-linear constrains, i.e. rotation vectors are orthogonal and have the same length:

$$\mathbf{r}_1 \perp \mathbf{r}_2 \;\; = \;\; \left(\alpha_2 \mathbf{x}_2^C - \mathbf{x}_1^C \right) \perp \left(\alpha_2 \mathbf{c_a} + \alpha_3 \mathbf{c_b} + \mathbf{c_c} \right), \tag{5.25}$$

$$\|\mathbf{r}_1\| = \|\mathbf{r}_2\| \;\; = \;\; \|\alpha_2 \mathbf{x}_2^C - \mathbf{x}_1^C\| = \|\alpha_2 \mathbf{c_a} + \alpha_3 \mathbf{c_b} + \mathbf{c_c}\|. \tag{5.26}$$

The above constraints can be rewritten into two 2^{nd} degree polynomial equations and they are sufficient for calculating unknown scalar factors α_2 and α_3. Figure 5.5 shows a script for our automatic generator [64] which generates a source code solving these two equations and the whole P3P problem.

Let's solve this system of polynomial equations with the hidden variable method presented in Section 4.2. We hide one of the unknown alpha factors - say α_2 - and treat it as a "parameter". First, let's rewrite equations (5.25) and (5.26) to the form:

$$\left(\alpha_2 \mathbf{x}_2^C - \mathbf{x}_1^C \right)^\top \left(\alpha_2 \mathbf{c_a} + \alpha_3 \mathbf{c_b} + \mathbf{c_c} \right) = c_{20}\alpha_2^2 + c_{11}\alpha_2\alpha_3 + c_{10}\alpha_2 + c_{01}\alpha_3 + c_{00} = 0, \tag{5.27}$$

$$\begin{aligned} \left(\alpha_2 \mathbf{x}_2^C - \mathbf{x}_1^C \right)^\top \left(\alpha_2 \mathbf{x}_2^C - \mathbf{x}_1^C \right) - \left(\alpha_2 \mathbf{c_a} + \alpha_3 \mathbf{c_b} + \mathbf{c_c} \right)^\top \left(\alpha_2 \mathbf{c_a} + \alpha_3 \mathbf{c_b} + \mathbf{c_c} \right) = \\ d_{20}\alpha_2^2 + d_{11}\alpha_2\alpha_3 + d_{02}\alpha_3^2 + d_{10}\alpha_2 + d_{01}\alpha_3 + d_{00} = 0, \end{aligned} \tag{5.28}$$

where c_{ij} respectively d_{ij} are coefficients appearing in polynomials representing the orthogonality respectively the length constraint.

Next, rewrite the above polynomial equations into the matrix form $\mathtt{M}(\alpha_2)\mathbf{v} = 0$, where $\mathbf{v}$ is the monomial vector $\left[\alpha_3^2, \alpha_3, 1 \right]^\top$:

$$\mathtt{M}(\alpha_2)\mathbf{v} = \begin{bmatrix} 0 & c_{11}\alpha_2 + c_{01} & c_{20}\alpha_2^2 + c_{10}\alpha_2 + c_{00} \\ d_{02} & d_{11}\alpha_2 + d_{01} & d_{20}\alpha_2^2 + d_{10}\alpha_2 + d_{00} \end{bmatrix} \begin{bmatrix} \alpha_3^2 \\ \alpha_3 \\ 1 \end{bmatrix} = \mathbf{0}. \tag{5.29}$$

Given a square matrix $\mathtt{A}$, a homogeneous linear system $\mathtt{A}\mathbf{x} = 0$ has a non-trivial solution if and only if the determinant of the matrix $\mathtt{A}$ is zero. In order to use this property, we need to extend

```
x1 = gbs_Vector('x1', 3);     % 2D measurement
x2 = gbs_Vector('x2', 3);

ca = gbs_Vector('ca', 3);     % substitutions comming from
cb = gbs_Vector('cb', 3);     % the thrid point correspondence
cc = gbs_Vector('cc', 3);

syms a2 a3 real;              % unknown scales

R1 = a2*x2 - x1;              % columns of rotation matrix
R2 = a2*ca + a3*cb + cc;

eq(1) = transpose(R1)*R2;     % non-linear constraints
eq(2) = transpose(R1)*R1 - transpose(R2)*R2;

unknown = {'a2' 'a3'};
known = {'x11' 'x12' 'x13' 'x21' 'x22' 'x23' ...
         'ca1' 'ca2' 'ca3' 'cb1' 'cb2' 'cb3' ...
         'cc1' 'cc2' 'cc3'};
[res export] = gbs_CreateCode('p3p_solver_H', eq,
                                              known, unknown);
```

Figure 5.5: A script generating P3P solver using homography for our automatic generator [64].

our polynomial matrix $\mathtt{M}(\alpha_2)$ to a square matrix. Since the polynomial in Equation (5.27) does not contain 2^{nd} degree term, we can simply create a new row of matrix $\mathtt{M}(\alpha_2)$ by multiplying polynomial (5.27) by α_3. Note that the multiplication by α_3 does not change the number of solutions. We obtain:

$$\mathtt{M}'(\alpha_2)\mathbf{v} = \begin{bmatrix} 0 & c_{11}\alpha_2 + c_{01} & c_{20}\alpha_2^2 + c_{10}\alpha_2 + c_{00} \\ d_{02} & d_{11}\alpha_2 + d_{01} & d_{20}\alpha_2^2 + d_{10}\alpha_2 + d_{00} \\ c_{11}\alpha_2 + c_{01} & c_{20}\alpha_2^2 + c_{10}\alpha_2 + c_{00} & 0 \end{bmatrix} \begin{bmatrix} \alpha_3^2 \\ \alpha_3 \\ 1 \end{bmatrix} = \mathbf{0}. \tag{5.30}$$

Now, a non-trivial solution exists if and only if the determinant of matrix $\mathtt{M}'(\alpha_2)$ vanishes, i.e.

$$det\left(\mathtt{M}'(\alpha_2)\right) = 0. \tag{5.31}$$

The determinant is in this case a 4^{th} degree polynomial in α_2 whose roots can be found in a closed form [39], using Frobenius companion matrix [9] or real roots can be found numerically using Sturm bracketing [49]. Once α_2 factors are recovered it is straightforward to calculate α_3, camera rotation and translation vectors. In contrast to the previous P3P solutions this solution returns camera matrices directly.

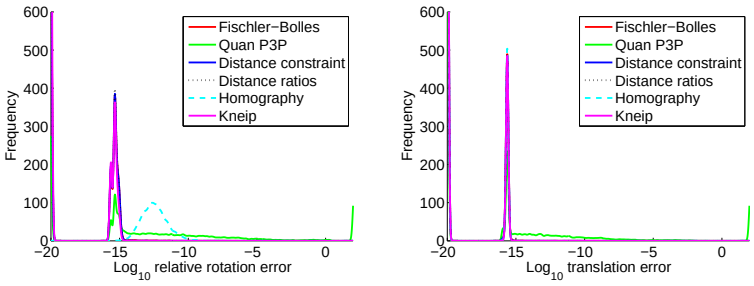


Figure 5.6: Numerical stability of the studied P3P solvers. The homography solver appears to be the least accurate, however it is far enough accurate for real use.

5.5 Synthetic experiments

In the synthetic experiment we focus only on the numerical stability of above P3P algorithms only. This is sufficient since all studied algorithms are minimal, i.e. they are using the minimal number of 2D-to-3D correspondences. Moreover, they are algebraically equivalent and particular solvers differ only in a formulation and solution method. It means, for example, that adding noise to image measurements or 3D points represents just another configuration between camera and the 3D space. Hence all solvers must behave the same way and again return the same solutions. Further comparisons with other algorithms appear in next chapters.

5.5.1 Numerical stability

In these test, the synthetic scene was generated randomly with 3D points randomly distributed in a cube. Each 3D point was projected by a randomly placed cameras such that all points were in the front of the cameras.

For each triplet of 2D-to-3D points we evaluated all solvers and calculated camera rotations and translations. If a solver produced more than one feasible solution, we selected the one closest to the ground truth camera. Let $\mathtt{R}$ be an estimated camera rotation and $\mathtt{R}_{gt}$ a ground-truth rotation. The rotation error is measured as an angle in the angle axis representation of the relative rotation $\mathtt{R}\,\mathtt{R}_{gt}^{-1}$ and the translation error as an angle between the ground-truth and the estimated translation vectors. Figure 5.6 shows results for 10000 executions of presented algorithms together with Fischler's [37], Quan's [88] algorithm and direct P3P solution by Kneip [58]. Quan's algorithm appears to be numerically less stable compared to other algorithms. This is caused by the fact, that an eight degree polynomial is build and its coefficients are very different in magnitude. Consequently roots solver balances on the edge of double precision arithmetics. All our studied algorithms are precise enough for practical use although the P3P solver through homography appears to be less accurate. Precision of the P3P homography solver is affected by the selection which points are transfered to the origin and to $[1, 0, 0]$. Nevertheless, algorithm precision is far enough good for real applications even without permuting point correspondences. Also it is less complex and faster compared to the solution by Kneip et al. [58].

5.6 Conclusion

In this chapter we focused on solutions to the well known P3P problem. This problem is a very old problem and many solutions were already published. However even the problem itself is easy to formulate and solve, more important question is how to extend P3P to a more general camera model. Following chapters target this question and show how to extend the above formulations for cameras with incomplete calibration or for cameras where some additional information is available, e.g. a vertical direction, a single vanishing point, etc. Each provided P3P formulation has some benefits when we are trying to extend it to more complicated camera models. Some formulations are faster, some are numerically more stable or easier to extend to a more complicated models. Some formulations are more complicated or even impossible to extend at all or result in huge, slow, and impractical solvers.

6 Absolute pose for a camera with an unknown focal length

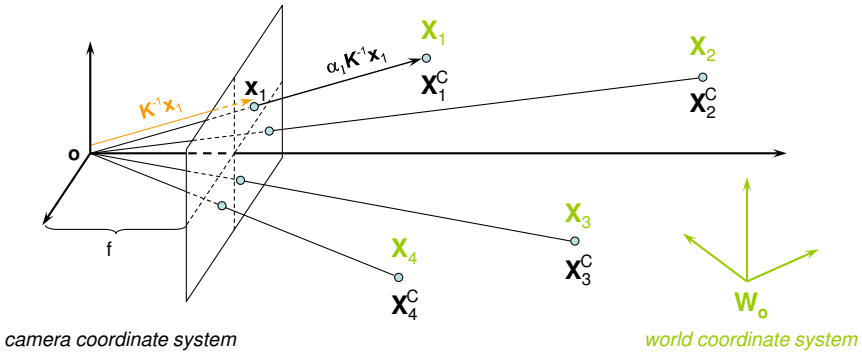


Figure 6.1: The camera and the world coordinate systems, image measurements ($\mathbf{x}$), ray direction vectors ($\mathtt{K}^{-1}\mathbf{x}$), and 3D points in the world coordinate system ($\mathbf{X}$) and in the camera coordinate system ($\alpha\,\mathtt{K}^{-1}\mathbf{x}_1$).

In this chapter, we consider a bit more complicated situation when the camera focal length is unknown. For consumer digital cameras we can assume that most of the calibration parameters are known or can be sufficiently well approximated. For example, cameras have square pixels and the principal point is close to the center of the image. For most of the applications this prior knowledge can be used and four out of the five internal calibration parameters can be safely set to the prior values. Focal length is the only missing, and for metric reconstruction very important, calibration parameter. This chapter deals with the situation when the camera focal length is the only unknown parameter in the camera calibration matrix. Figure 6.1 illustrates this problem.

In next sections we study several formulations of estimating the absolute pose of a camera with an unknown focal length (P4P+f). We compare several approaches, their pros and cons in terms of speed, accuracy, number of solutions and applicability. We show that four point correspondences between 3D space and 2D image measurements are enough to solve the unknown camera pose and the focal length and moreover over-constraint this problem. We use appropriate invariants to simplify the problem. Splitting the problem into two subproblems - (1) one for planar configuration where 3D points are on a plane and (2) one for non-coplanar 3D points also greatly simplifies this problem. Splitting does not cast any practical obstacle as 3D points are always known in advance and therefore we can determine if they are on a plane or not. We also study the numerical stability of algorithms on the boundary between planar and non-planar scenes to show good behavior of this approach.

6.1 Problem formulation

The relationship between corresponding measurements in a 2D image and 3D space is similar to Problem 2 where projection of 3D reference points, represented by homogeneous coordinates $\{\mathbf{X}_i\}_{i=1}^4$, by a projection matrix $\mathtt{P} = \mathtt{K}\,[\mathtt{R}\,|\,\mathbf{t}]$ is given as

$$\alpha_i\,\mathbf{x}_i = \mathtt{P}\,\mathbf{X}_i. \tag{6.1}$$

Instead of a fully calibrated matrix we have a one with the unknown focal length. Without loss of generality, and to simplify the following text, we can assume that calibration matrix $\mathtt{K}$ is simplified to the following form

$$\mathtt{K} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{6.2}$$

where f is the unknown focal length. This can be achieved easily by pre-multiplying 2D measurements with the inverse of a known calibration matrix with the focal length equal to one

$$\mathtt{K}_{known} = \begin{bmatrix} 1 & s & u \\ 0 & \alpha & v \\ 0 & 0 & 1 \end{bmatrix}, \tag{6.3}$$

where s is the camera skew, α is an aspect ratio and u, v are coordinates of the principal point [47]. The absolute pose problem for the camera with an unknown focal length can be formulated as follows.

Problem 4. *Let $\mathbf{X}_i$ with $i \in \{1, \ldots, 4\}$, be a set of four 3D points and let $\mathbf{x}_i$ be the corresponding projections measured in the image plane. Let $\mathtt{K}_{known}$ be a camera calibration matrix up to an unknown focal length. The goal is to find unknown camera rotation matrix $\mathtt{R}$, translation vector $\mathbf{t}$ and unknown focal length f such that*

$$\alpha_i\mathbf{x}_i = \mathtt{K}_{known}\mathtt{K}\,[\mathtt{R}|\mathbf{t}]\,\mathbf{X}_i, \tag{6.4}$$

holds for all input 2D-to-3D correspondences, where α_i are unknown scalar factors, and $\mathtt{K} = diag\,[f, f, 1]$ is a diagonal calibration matrix with unknown focal length.

Let's assume that we have pre-calibrated 2D points with known elements of the calibration matrix. Similarly as in the previous chapter, let $\mathbf{X}_i^C = [R\,|\,t]\,\mathbf{X}_i$ be affine coordinates of the set of the reference 3D points in the camera coordinate system. From (6.1), we get

$$\alpha_i\,\mathtt{K}^{-1}\mathbf{x}_i = \mathbf{X}_i^C, \tag{6.5}$$

and thus the direction vectors of the camera rays toward the reference points become

$$\mathbf{x}_i^C = \mathtt{K}^{-1}\mathbf{x}_i \simeq \begin{bmatrix} x_i \\ y_i \\ f \end{bmatrix}, \tag{6.6}$$

where x_i and y_i stand for the image coordinates and $\simeq$ denotes equality up to multiplication by a non-zero scale factor. Note that $\mathbf{x_i^C}$ lies on the line connecting the camera centre and $\mathbf{X_i^C}$ and from (6.5) we can see that $\alpha_i\mathbf{x_i^C} = \mathbf{X_i^C}$, vice versa point $\mathbf{X_i^C}$ is somewhere on this line.

6.1.1 Using distances

Following the idea we used in the previous chapter, we will solve unknown scale factors α_i which stretch camera rays $\mathbf{x}_i^C$, until they reach corresponding 3D points $\mathbf{X}_i^C$, Figure 6.1.

Again, we can use the fact that the transformation $[\mathtt{R} \,|\, \mathbf{t}]$, transforming $\mathbf{X}_i$ into $\mathbf{X}_i^C$ is Euclidean and thus preserves distances between 3D points. Hence, known distance between points $\mathbf{X}_i$ in the world coordinate system is the same as distance between points $\mathbf{X}_i^C$ in the camera coordinate frame. This can be express as

$$d_{ij}^2 = \|\mathbf{X}_i - \mathbf{X}_j\|^2 = \|\mathbf{X}_i^C - \mathbf{X}_j^C\|^2 = \|\alpha_i \, \mathbf{x}_i^C - \alpha_j \, \mathbf{x}_j^C\|^2, \tag{6.7}$$

where d_{ij} is a distance between i^{th} and j^{th} 3D point. Equation (6.7) looks the same as Equation (5.6) for P3P problem, however in this case we have an additional unknown focal length hidden in the camera measurements $\mathbf{X}_i^C$ and $\mathbf{x}_i^C$.

Using Equation (6.7) with different i and j, we can build a system of six 4^{th} degree polynomial equations in five unknowns - four unknown α_i and the unknown focal length f. We can further reduce the degree of the equations by one using the substitution $w = f^2$, since all f appear in even powers only.

Solving system of six polynomial equations is not always easy and moreover this system of equations is over-constrained and thus we have more equations than unknowns. We will next investigate several ways how to solve the system, go through different approaches and invariants, study what happens if we use different subsets of equations, focus on the stability, the number of solutions, size of solvers and re-formulate the original problem to make it easier to solve. We also review our solution previously published in [12].

In this solution we use the fact that Euclidean norm is invariant under a similarity transformation. Hence we can solve P4P problem with unknown focal length without solving the unknown rotation matrix and the translation vector, just by calculating unknown depths. Once all solutions to the system of polynomial equations (6.7) are found, i.e. the depths are found, we can calculate the camera pose and orientation. It is sufficient to recover 3D positions $\mathbf{X}_i^C$ of points $\mathbf{X}_i$ in the camera coordinate system using (6.5) and then search for a rigid transformation mapping all $\mathbf{X}_i$ to $\mathbf{X}_i^C$ [7, 52, 32].

6.1.2 Using ratios of distances

Similar to P3P solution in Section 5.3.2, we use the fact that all α_i are nonzero and positive. Hence we can select and fix one of the depths, say α_1, and express all remaining α_i as multiples of α_1, i.e. $\alpha_i = \alpha_1 \lambda_i$. Note that $\lambda_1 = 1$. Equation (6.7) become

$$\begin{aligned} d_{ij}^2 = \|\mathbf{X}_i - \mathbf{X}_j\|^2 &= \|\alpha_1 \, \lambda_i \, \mathbf{x}_i^C - \alpha_1 \, \lambda_j \, \mathbf{x}_j^C\|^2 \\ &= \alpha_1^2 \, \|\lambda_i \, \mathbf{x_i^C} - \lambda_j \, \mathbf{x_j^C}\|^2, \end{aligned} \tag{6.8}$$

for every $i, j \in \{1, \ldots, 4\}$. For distinct 3D reference points and for each $i \neq j$ we have both the left and the right side of Equation (6.8) nonzero and positive. Hence we can form the following

ratios

$$r_{ijkl} = \frac{\|\mathbf{X}_i - \mathbf{X}_j\|^2}{\|\mathbf{X}_k - \mathbf{X}_l\|^2} = \frac{\alpha_1^2 \, \|\lambda_i \, \mathbf{x}_i^C - \lambda_j \, \mathbf{x}_j^C\|^2}{\alpha_1^2 \|\lambda_k \, \mathbf{x}_k^C - \lambda_l \, \mathbf{x}_l^C\|^2}, \tag{6.9}$$

in which α_1^2 on the right hand side cancels, thus yielding

$$r_{ijkl} \, \|\lambda_k \, \mathbf{x}_k^C - \lambda_l \, \mathbf{x}_l^C\|^2 = \|\lambda_i \, \mathbf{x}_i^C - \lambda_j \, \mathbf{x}_j^C\|^2. \tag{6.10}$$

By exhausting all permutations without repeating $i, j, k, l \in \{1, \ldots, 4\}$ in Equation (6.10), we get a system of 15 polynomial equations in four unknowns $(\lambda_2, \lambda_3, \lambda_4, f)$. Note that each x_i^C contains one unknown parameter, which is the focal length f. Unknown f appears in even powers only, and thus the substitution $w = f^2$ reduces the total degree of these equations to three. We observed that only five out of these 15 equations are linearly independent. To get these five linearly independent equations in 20 monomials we performed Gauss-Jordan elimination. We will explain this process later in the text.

6.2 Hidden variable solver revisited

The hidden variable method, Section 4.2, offers a relatively easy way how to hide and solve unknown variable in a system of polynomial equations. One way how to solve our system (6.10) using the hidden variable method is to eliminate variables one by one. First, we need to rewrite system (6.10) into the matrix form and hide some variable - say f:

$$\mathtt{M}(f) \, \mathbf{X} = \mathbf{0}, \tag{6.11}$$

where $\mathtt{M}(f)$ is a coefficient matrix depending on the hidden variable f, and $\mathbf{X}$ is a vector of all monomials including 1 in the remaining variables - $\lambda_2, \lambda_3, \lambda_4$. Unfortunately, system (6.10) contains 10 unknown monomials after hiding the focal length f. This means, that matrix $\mathtt{M}(f)$ must have at least 10 linearly independent rows. However, only five out of fifteen equations from (6.10) are linearly independent. As a result we would need to generate a new set of polynomials so that the hidden variable method can be used. The problem is that adding new polynomials results in expansion of matrix $\mathtt{M}(f)$ and vector $\mathbf{X}$ and thus yielding a very high degree polynomial in one variable by computing the determinant of matrix $\mathtt{M}(f)$. Consequently, computation time as well as numerical stability of the solver are to large extent determined by the degree of this polynomial (4.3). Finding coefficients of this polynomial is equivalent to computing the determinant of the matrix $\mathtt{M}(f)$. This requires to multiply and sum numbers with different orders of magnitude. Thus "double" precision arithmetic may not be sufficient here. Further, finding the roots of a high degree polynomial involves finding the eigenvalues of a big companion matrix. Sturm bracketing [49] appears to be more stable then conventional eigenvalue methods and much faster since we can take advantage that the focal length is positive and finite.

To avoid the above mentioned problems we will not directly apply the hidden variable resultant method but we do some preprocessing to first simplify the problem.

Equation (5.12) from Section 4.2 looks the same as Equation (6.10) with exception that the unknown focal length is known for P3P problem. By fixing one of depths, say λ_1, in P4P equations (6.10), we get a system which has only three unknown depths $(\lambda_2, \lambda_3, \lambda_4)$. If we fix $i = 1$

and $k = 1$ in these equations (6.10), then we remove one variable from each equation (because $\lambda_1 = 1$). Now, each equation from (6.10) contains only two unknown depths (λ_j, λ_l) and the focal length f. Each of them can be easily removed using the hidden variable resultant (4.3). This yields a polynomial which we denote as $p3p_{j,l}(f)$ and which contains variable λ_l and fixed parameter f. We can look at it as on a parametric P3P problem. Given four 2D-to-3D point correspondences, we can build a system of 6 polynomial equations $p3p_{2,3}(f), p3p_{3,2}(f), p3p_{2,4}(f)$, $p3p_{4,2}(f), p3p_{3,4}(f), p3p_{4,3}(f)$ in four unknowns $(\lambda_2,\ \lambda_3,\ \lambda_4,\ f)$, where each equation contains just two of them - the focal length f and one of the unknown depth λ_i.

From this set of equations we can select a pair which shares the same unknowns, for instance, $p3p_{2,3}(f)$ and $p3p_{4,3}(f)$ in unknown λ_3 and f, and solve them using the hidden variable technique. In our case we "hide" the variable f. To obtain the square matrix $\mathtt{M}$ we need to add 6 equations (monomials multiples of equations $p3p_{2,3}(f)$ and $p3p_{4,3}(f)$), so getting a 8×8 resultant matrix (4.2) for unknown focal length.

Even calculating the determinant of a symbolic 8×8 matrix is a challenging task. In [12] we used rational arithmetic to calculate symbolic determinant and convert its coefficients to double precision floats before computing roots of determinant polynomial. Recently, Hartley et al. [46] proposed framework based on Toeplitz matrices and Levinson-Durbin algorithm for solving such tasks.

Alternatively, numerically stable solution can be obtained by transferring this problem to the polynomial eigenvalue problem [65, 67, 9] as described in Section 4.4. This can be achieved by splitting matrix $\mathtt{M}$ into

$$\mathtt{M} = \mathtt{M}_0 + f^2\mathtt{M}_1 + f^4\mathtt{M}_2 + f^6\mathtt{M}_3 + f^8\mathtt{M}_4, \tag{6.12}$$

and solve for example using $Matlab$ polyeig function. For more details refer to [62, 67, 65, 9].

Performance, precision and comparison of the hidden variable solver with other solver is discussed at the end of this chapter. It is easy to understand and to implement this solver, with algebraic tools for manipulating with symbolical terms. Drawback of this algorithm is lack of speed and numerical stability. On the other hand the algorithm somewhat benefits from the fact that the problem is over-constrained. This is visible in real and RANSAC experiments in the experiments section.

In the next section we use Gröbner basis method to solve the initial system of polynomial equations (6.10). This Gröbner basis solver is faster and more suitable for real-time applications. On the other hand it is much more difficult to implement and more difficult to understand it.

6.3 Gröbner basis solver revisited

Before diving into the Gröbner basis based solution to the camera absolute pose problem with an unknown focal length we should note that the automatic generator [64] can be easily used to create this solver. We will show how to create it in Section 6.4. Before that, we show how to use Gröbner basis method described in Section 4.3 to construct the solver by hand.

The Gröbner basis solver starts with 15 equations (6.10) in four unknowns $\lambda_2, \lambda_3, \lambda_4$ and $w = f^2$. Only five from these 15 equations are linearly independent so in fact, we have five equations in four unknowns. This is an over-constrained system of polynomial equations which

can be solved in several ways. We have decided to use four from these five equations to find solutions to the four unknowns and use the fifth equation to test for possible degeneracy and to select the best root.

We use the last four equations which we obtain after the G-J elimination of the initial 15 equations. The reason for this is that the system of the first four equations has 14 solutions and results in more complicated computations of the action matrix. On the other hand, the system of the last four equations has only 10 solutions and the action matrix can be obtained easier. By selecting different five-tuples we can get solvers with 10-18 solutions. We analyze in detail all possible combinations in next sections. In this section we review the solution presented we in [12].

To create the action matrix, we use the method described in Section 4.3. This method calls for generating polynomials from the ideal with leading monomials from the set $p \cdot B \setminus B$ and the remaining mononomials from B. The structure of these polynomials can be found for our problem once in advance. This is possible thanks to the fact that this problem, like many other in computer vision, has the convenient property that the monomials which appear in the initial system of polynomial equations are always the same irrespectively from the concrete coefficients arising from non-degenerate image measurements. Therefore, the leading monomials of the corresponding Gröbner basis, and thus the monomials in the basis B are generally the same. So they can be found once in advance.

To compute B, we solve our problem in a random chosen finite prime field $\mathbb{Z}_p$ where exact arithmetic can be used and numbers can be represented in a simple and efficient way. It speeds up computations, minimizes memory requirements and especially avoids numerical instability.

We use algebraic geometry software Macaulay 2, which can compute in finite fields, to solve the polynomial equations for many random coefficients from $\mathbb{Z}_p$, to compute the number of solutions, the Gröbner basis, and the basis B.

Using Macaulay2, we have found that our problem has 10 solutions and the basis $B = (\lambda_3^2, w\lambda_4, \lambda_2\lambda_4, \lambda_3\lambda_4, \lambda_4^2,\ w, \lambda_2, \lambda_3, \lambda_4, 1)$ w.r.t. the used graded reverse lexicographic ordering of monomials with $w > \lambda_2 > \lambda_3 > \lambda_4$. Once this is known, the action matrix for floating point coefficients can be created.

We construct the action matrix $\mathtt{M}_{\lambda_2}$ for multiplication by λ_2 because the creation of the action matrix for $w = f^2$ requires to generate polynomials of higher degrees.

Recall that the method described in Section 4.3 calls for generating polynomials with leading monomials from the set $\lambda_2 \cdot B \setminus B$ and the remaining monomials from B. To generate these polynomials from the ideal, we use the method described in Section 4.3 in which these polynomials are generated in one step by multiplying initial four polynomial equations with selected monomials and reducing all generated polynomials at once using one G-J elimination.

The set of monomials which should be used in this process for multiplying initial polynomial equations can be found once in advance. To find this set we used Macaulay 2. We have found that to obtain all necessary polynomials for crating the action matrix, we need to generate all monomial multiples of the initial four polynomial equations up to the total degree seven. This means that we need to multiply our four 3^{rd} degree polynomial equations with all monomials up to the degree four.

In this way we generate 276 new polynomials which, together with the initial four polynomial

equations, form a system of 280 polynomials in 283 monomials. Only 243 from these polynomials are linearly independent. So, in the next step we select only 243 linearly independent polynomial equations. Then we remove all unnecessary polynomials by the procedure described in Section 4.3 and obtain 154 polynomial equations in 180 monomials. These polynomial equations can be written in the matrix form (4.9). After the G-J elimination of the coefficient matrix `M` we obtain all polynomials with leading monomials from the set $\lambda_2 \cdot B \setminus B$ and the remaining monomials from B which we need for constructing the action matrix $\mathtt{M}_{\lambda_2}$.

Note that all steps until now (computation in Macaulay 2, finding the basis B and finding the polynomials form the ideal which should be generated to obtain all necessary polynomials for constructing the action matrix) were done only once by us to study this problem and provide the solution to it. The online solver consists only from one G-J elimination of the 154×180 coefficient matrix `M`. This matrix contains coefficients which arise from concrete image measurements. After the G-J elimination of this matrix, and using its rows which correspond to the polynomials with leading monomials from the set $\lambda_2 \cdot B \setminus B$, the action matrix $\mathtt{M}_{\lambda_2}$ can be created. The solutions to four unknowns $\lambda_2, \lambda_3, \lambda_4$ and $w = f^2$ can be found using eigenvectors of this action matrix.

6.4 Creating an optimal ratio solver

Let's have a look on how to solve equations (6.10) using our automatic generator [64]. Our goal is to create a solver - a function with inputs image measurements and distances between 3D points and outputs unknown focal length and unknown depths.

All these entities are unknowns or parameters for the automatic solver generator. Let's have a look how to create P4P+f solver step-by-step. First, define unknowns and known parameters.

```
% define unknown focal length and depths
syms f a2 a3 a4;

% declare variables holding known distances
% between 3D points
syms D12 D13 D14 D23 D24 D34;

% declare 4 image measurements
syms x1 y1 x2 y2 x3 y3 x4 y4;

% substitution variable w = f^2
syms w;
```

Now, parametrize four reference 3D points in the camera coordinate system using known image measurements, unknown depths α_i - `a2,a3,a4` and the focal length f. Note that α_1 equals one.

```
XC1 =  1*[x1; y1; f];
XC2 = a2*[x2; y2; f];
XC3 = a3*[x3; y3; f];
XC4 = a4*[x4; y4; f];
```

Now create the system of polynomial equations according to (6.10).

```
% express distances between 3D points in camera coordinate
% system
d12 = (sum((XC1-XC2).^2));
d13 = (sum((XC1-XC3).^2));
d14 = (sum((XC1-XC4).^2));
d23 = (sum((XC2-XC3).^2));
d24 = (sum((XC2-XC4).^2));
d34 = (sum((XC3-XC4).^2));

% ratios of distances rijkl = Dij/Dkl
eq(01) = D12*d13 - d12*D13;
eq(02) = D12*d14 - d12*D14;
eq(03) = D12*d23 - d12*D23;
eq(04) = D12*d24 - d12*D24;
eq(05) = D12*d34 - d12*D34;

eq(06) = D13*d14 - d13*D14;
eq(07) = D13*d23 - d13*D23;
eq(08) = D13*d24 - d13*D24;
eq(09) = D13*d34 - d13*D34;

eq(10) = D14*d23 - d14*D23;
eq(11) = D14*d24 - d14*D24;
eq(12) = D14*d34 - d14*D34;

eq(13) = D23*d24 - d23*D24;
eq(14) = D23*d34 - d23*D34;

eq(15) = D24*l34 - d24*D34;
```

The focal length f appears squared in all equations. Hence we can substitute it with $f^2 = w$ to lower total degree of the equations.

```
eq = expand(eq);
% substitute f^2 = w
for i=1:size(eq,2)
   str = char(expand(eq(i)));
   str = strrep(str, 'f^2', 'w');
   eq(i) = sym(str);
end
```

As the last step we need to create sets of known and unknown variables so that the solver generator knows how to treat these variables. The order of unknowns defined in $unknown$ array is important for the Gröbner basis method since it influences the form of necessary polynomials for creating the action matrix and also the whole elimination process. Thus for a different orderings of `w,a2,a3,a4` in the `unknown` field we may get different solvers with different properties [62]. We will return to this in more detail in the next section. Finally, we call the Gröbner basis solvers generator [64] to create the online solver.

```
unknown = {'w' 'a2' 'a3' 'a4'};
known =   {'D12' 'D13' 'D14' 'D23' 'D24' 'D34'
           'x1' 'y1' 'x2' 'y2' 'x3' 'y3' 'x4' 'y4'};

cfg = gbs_InitConfig();
cfg.GBrestrictEq = [4 8 9 11];
kngroups = [1 1 1 1 1 1 2 2 3 3 4 4 5 5];
[res export] = gbs_CreateCode('mbp4pf_ragio', eq,
                          known, unknown, kngroups, cfg);
```

Code snippet above creates a $Matlab$ solver for the studied problem. The code contains one special line of code:

```
cfg.GBrestrictEq = [4 8 9 11];
```

This line specifies which equations should be used for finding the Gröbner basis. This is important, and we will explain it now. Recall that the original set of equations (6.10) is over-constrained, i.e. we have more equations than variables. The automatic solver generator calculates a random instance of the input problem when it searches for Gröbner basis (Section 4.5). The way how it creates the random instance is important. First it takes all parameters of the system which are in this case 2D measurements and distances between 3D points. Then it creates random values in a finite field modulo a prime number and sets them to the known parameters. The problem is that the distances between 3D points and 2D measurements are not algebraically independent. Hence, by setting random values to the known parameters we get a system which does not have a solution. To overcome this problem we have selected equations $4, 8, 9$ and 11 which produce well-constrained system. A different set of equations would yield a different

result. For example running above $Matlab$ code script results in a solver with 14 solutions and elimination template with size 98×114.

Proper instance

Recall from Section 4.3 and Section 4.5 that the process of creating the Gröbner basis solver consists of the offline and the online phase. The online phase follows a solution (or elimination) path instantiated with some rational input values. The offline phase is used to find the "optimal" solution path in the solution space using one or more particular instances of the problem and the particular ordering of variables. Since our system is over-constrained, assigning random values to known variables, i.e. image measurement and distances between 3D points, will result in a system without a solution. As mentioned before, it is because the distances between 3D points and 2D measurements are not algebraically independent. Image measurements produce rays of sights which limit the way how should the 3D model look like in the camera coordinate system. These relations are not preserved when completely random data are used. Therefore we need to use a proper instance of this problem. By proper instance we mean data which satisfy all algebraical (latent) relations given by the problem. In this case, we can create a proper instance using the random scene generator operating in the finite prime field described in Section 4.6.

In the automatic generator [64], the random instance generator is just a function which returns equations where all known variables are replaced with some values. This instance generator can be set with the following line of code:

```
cfg.InstanceGenerator = @mbp4pf_instanceGenerator;
```

The instance generator can be easily created according to Section 4.6. In addition to the random scene generator from Section 4.6, `mbp4pf_instanceGenerator` calculates 2D measurements and distances between 3D points in the finite field arithmetics:

```
% calculate distances between ground true reference points
D12 = mod(sum((M(:,1)-M(:,2)).^2), prime);
D13 = mod(sum((M(:,1)-M(:,3)).^2), prime);
D14 = mod(sum((M(:,1)-M(:,4)).^2), prime);
D23 = mod(sum((M(:,2)-M(:,3)).^2), prime);
D24 = mod(sum((M(:,2)-M(:,4)).^2), prime);
D34 = mod(sum((M(:,3)-M(:,4)).^2), prime);
% set 2D measurements
x1 = m(1,1);
y2 = m(2,1);
x2 = m(1,2);
y2 = m(2,2);
x3 = m(1,3);
y3 = m(2,3);
x4 = m(1,4);
y4 = m(2,4);
```

Since we created a 3D scene, calculated distances and projections correctly, we get a proper random instance which preserves latent algebraic dependencies between variables. Hence we can include all five equations into the solver creation process.

Running above scripts using the automatic solver generator, i.e. the script with the proper random instance generator, produces a solver with 14 solutions and elimination template with size 76×90.

6.4.1 Reducing input equations

We observed that by reducing input equations using Gauss-Jordan elimination before calling the automatic solver generator we can obtain a smaller online solver. In fact only five out of 15 input equations are linearly independent. Gauss-Jordan elimination creates a simpler set of equations where each equation has a different leading monomial and each equation is sparser (with less monomials). A question arise: *what does it mean to perform Gauss-Jordan elimination on a set of polynomial equations?*

To explain the idea consider the following example. Let's have a set of equations:

$$2y^2 + x^2 - 6x = 0, \tag{6.13}$$

$$y^2 + x + 1 = 0. \tag{6.14}$$

By subtracting the second equation multiplied by two from the first one we eliminate y^2 in the first equation:

$$x^2 - 8x - 2 = 0, \tag{6.15}$$

$$y^2 + x + 1 = 0. \tag{6.16}$$

Now let's rewrite input equations (6.13) and (6.14) into the matrix form, i.e. a coefficient matrix multiplied with a monomial vector:

$$\begin{bmatrix} 1 & 2 & -6 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x^2 \\ y^2 \\ x \\ 1 \end{bmatrix} = \mathbf{0}, \tag{6.17}$$

which is equivalent matrix representation of polynomials (6.15) and (6.16).

After the Gauss-Jordan elimination of the coefficient matrix we get:

$$\begin{bmatrix} 1 & 0 & -8 & -2 \\ 0 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x^2 \\ y^2 \\ x \\ 1 \end{bmatrix} = \mathbf{0}. \tag{6.18}$$

Definition 1. *By Gauss-Jordan elimination of a set of input equations we understand:*

- *splitting input equations into a coefficient matrix and a monomial vector,*
- *performing the Gauss-Jordan elimination on the coefficient matrix,*
- *multiplying the reduced coefficient matrix with the monomial vector.*

Clearly, by using a different monomial ordering, we obtain a different coefficient matrices and thus a different elimination results.

We found that by using $grevlex$ ordering we obtain the best results for the absolute pose problem. Ordering of unknowns had only a minor impact on the size of the elimination temple.

The following code can be used to perform Gauss-Jordan elimination of an arbitrary set of polynomial equations with respect to $grevlex$ ordering:

```
% split equations into coefficients and monimials
[coefs monomials] = SplitEquations(eq, unknown);
monomials = [monomials 1];

% reduce coefficients matrix
reduced = rref(coefs);

% create reduced system
for i=1:size(eq,2)

   eq(i) = 0;
   for j=1:size(coefs,2)
      eq(i) = eq(i) + reduced(i,j)*monomials{j};
   end
   eq(i) = expand(eq(i));
end
```

By eliminating the initial polynomial equations using the Gauss-Jordan elimination we get a solver with 10 solutions and the elimination template reduces to size 53×63. The reduced system contains only five equations and the remaining equations canceled. Since the system is also over-constrained we selected four out of these five equations:

```
cfg.GBrestrictEq = [2 3 4 5];
```

6.4.2 Exhaustive search for the best solver

In previous section, we have identified many issues which had impact on the resulting online solver: choice of the subset of equations, ordering of unknown variables, performing Gauss-Jordan elimination before creating the solver, instantiating the problem with random values or using the proper instance. Also performing or not performing the substitution for $f^2 = w$

has impact on the final solution. All this has impact on the size, the speed and the precision of solvers. By counting all combinations of doing or not doing certain steps, we get over 7.7 million possible ways of how to create a solver. However there are much less unique solvers.

First, permuting unknowns $\alpha_2, \alpha_3, \alpha_4$ does not have any impact on the result since all unknown depths appear in equations with the same monomial structure. Therefore, only changing the position of w or f has impact on the computed Gröbner basis and solvers. It is sufficient to consider the following four orderings of variables:

$$\{f, \alpha_2, \alpha_3, \alpha_4\}, \tag{6.19}$$

$$\{\alpha_2, f, \alpha_3, \alpha_4\}, \tag{6.20}$$

$$\{\alpha_2, \alpha_3, f, \alpha_4\}, \tag{6.21}$$

$$\{\alpha_2, \alpha_3, \alpha_4, f\}. \tag{6.22}$$

Next, there is only a limited number of Gröbner bases for every instance of our problem. We should note here that by choosing a different ordering of variables, a different subset of equations, type of instance whether random or proper, etc., we are solving a slightly different system of polynomial equations defining a different ideal I. These ideals have different Gröbner bases in general. Finding these different bases of different ideals should not be confused with finding different Gröbner bases (reduced, non-reduced) of the same ideal, i.e. the ideal defined by one fixed system of polynomial equations.

By exhausting all combinations (ordering, subset of equations, precise or random instance) we found 21 different bases of the quotient ring $A = \mathbb{C}[x_1, ..., x_n]/I$ (Section 4.3) yielding $4, 5, 8, 10, 14, 16, 18, 20, 28, 32, 36$ solutions to the problem. Standard monomial bases corresponding to the underlying quotient rings are then:

$$1, w, \alpha_3, \alpha_4 \tag{6.23}$$

$$1, \alpha_2, w, \alpha_3, \alpha_4 \tag{6.24}$$

$$1, \alpha_2, \alpha_2 f, f, f\alpha_3, f\alpha_4, \alpha_3, \alpha_4 \tag{6.25}$$

$$1, \alpha_2, \alpha_2 f, f, f\alpha_3, f\alpha_4, f\alpha_4^2, \alpha_3, \alpha_4, \alpha_4^2 \tag{6.26}$$

$$1, \alpha_2, \alpha_2\alpha_4, w, w\alpha_4, \alpha_3, \alpha_3^2, \alpha_3\alpha_4, \alpha_4, \alpha_4^2 \tag{6.27}$$

$$1, \alpha_2, \alpha_2\alpha_4, \alpha_3, \alpha_3\alpha_4, w, w^2, w\alpha_4, \alpha_4, \alpha_4^2 \tag{6.28}$$

$$1, \alpha_2, \alpha_2 f, \alpha_3, \alpha_3 f, \alpha_4, \alpha_4 f, f, f^2, f^3 \tag{6.29}$$

$$1, \alpha_2, \alpha_2 w, \alpha_3, \alpha_3 w, \alpha_4, \alpha_4^2, \alpha_4 w, w, w^2 \tag{6.30}$$

$$1, \alpha_2, \alpha_2 w, \alpha_2\alpha_3, \alpha_2\alpha_4, w, w^2, w\alpha_3, w\alpha_4, \alpha_3, \alpha_3^2, \alpha_3\alpha_4, \alpha_4, \alpha_4^2 \tag{6.31}$$

$$1, \alpha_2, \alpha_2^2, \alpha_2 w, \alpha_2\alpha_3, \alpha_2\alpha_4, w, w^2, w\alpha_3, w\alpha_4, \alpha_3, \alpha_3^2, \alpha_3\alpha_4, \alpha_4, \alpha_4^2, \alpha_4^3 \tag{6.32}$$

$$1, \alpha_2, \alpha_2^2, \alpha_2 w, \alpha_2\alpha_3, \alpha_2\alpha_4, w, w^2, w\alpha_3, w\alpha_4, \alpha_3, \alpha_3^2, \alpha_3\alpha_4, \alpha_4, \alpha_4^2, \alpha_4^3 \tag{6.33}$$

$$1, \alpha_2, \alpha_2^2, \alpha_2\alpha_3, \alpha_2\alpha_4, \alpha_2 w, \alpha_3, \alpha_3^2, \alpha_3\alpha_4, \alpha_3 w, \alpha_4, \alpha_4^2, \alpha_4 w, w, w^2, w^3 \tag{6.34}$$

$$1, \alpha_2, \alpha_2^2, \alpha_2 w, \alpha_2\alpha_3, \alpha_2\alpha_4, w, w^2, w\alpha_3, w\alpha_4, w\alpha_4^2, \alpha_3, \alpha_3^2, \alpha_3\alpha_4, \ldots \tag{6.35}$$
$$\ldots \alpha_3\alpha_4^2, \alpha_4, \alpha_4^2, \alpha_4^3$$

$$1, \alpha_2, \alpha_2^2, \alpha_2\alpha_3, \alpha_2\alpha_4, \alpha_2 w, \alpha_3, \alpha_3^2, \alpha_3\alpha_4, \alpha_3 w, \alpha_3 w^2, \alpha_4, \alpha_4^2, \alpha_4 w, \ldots \tag{6.36}$$
$$\ldots \alpha_4 w^2, w, w^2, w^3$$

$$1, \alpha_2, \alpha_2 f, \alpha_2 f\alpha_3, \alpha_2 f\alpha_4, \alpha_2\alpha_3, \alpha_2\alpha_4, f, f^2, f^3, f\alpha_3, f\alpha_3^2, f\alpha_3\alpha_4, f\alpha_4, f\alpha_4^2, \ldots \tag{6.37}$$
$$\ldots \alpha_3, \alpha_3^2, \alpha_3\alpha_4, \alpha_4, \alpha_4^2$$

$$1, \alpha_2, \alpha_2^2, \alpha_2^2 f, \alpha_2 f, \alpha_2 f\alpha_3, \alpha_2 f\alpha_4, \alpha_2 f\alpha_4^2, \alpha_2\alpha_3, \alpha_2\alpha_4, \alpha_2\alpha_4^2, f, f^2, f^3, f\alpha_3, \ldots \tag{6.38}$$
$$\ldots f\alpha_3^2, f\alpha_3\alpha_4, f\alpha_3\alpha_4^2, f\alpha_4, f\alpha_4^2, f\alpha_4^3, \alpha_3, \alpha_3^2, \alpha_3\alpha_4, \alpha_3\alpha_4^2, \alpha_4, \alpha_4^2, \alpha_4^3$$

$$1, \alpha_2, \alpha_2^2, \alpha_2^2 f, \alpha_2\alpha_3, \alpha_2\alpha_3 f, \alpha_2\alpha_4, \alpha_2\alpha_4 f, \alpha_2 f, \alpha_2 f^2, \alpha_2 f^3, \alpha_3, \alpha_3^2, \alpha_3^2 f, \ldots \tag{6.39}$$
$$\ldots \alpha_3\alpha_4, \alpha_3\alpha_4 f, \alpha_3 f, \alpha_3 f^2, \alpha_3 f^3, \alpha_4, \alpha_4^2, \alpha_4^2 f, \alpha_4 f, \alpha_4 f^2, \alpha_4 f^3, f, f^2, f^3$$

$$1, \alpha_2, \alpha_2^2, \alpha_2^2 f, \alpha_2 f, \alpha_2 f\alpha_3, \alpha_2 f\alpha_3\alpha_4, \alpha_2 f\alpha_4, \alpha_2 f\alpha_4^2, \alpha_2\alpha_3, \alpha_2\alpha_3\alpha_4, \alpha_2\alpha_4, \ldots \tag{6.40}$$
$$\ldots \alpha_2\alpha_4^2, f, f^2, f^3, f\alpha_3, f\alpha_3^2, f\alpha_3^2\alpha_4, f\alpha_3\alpha_4, f\alpha_3\alpha_4^2, f\alpha_4, f\alpha_4^2, f\alpha_4^3, \alpha_3, \ldots$$
$$\ldots \alpha_3^2, \alpha_3^2\alpha_4, \alpha_3\alpha_4, \alpha_3\alpha_4^2, \alpha_4, \alpha_4^2, \alpha_4^3$$

$$1, \alpha_2, \alpha_2^2, \alpha_2^2 f, \alpha_2\alpha_3, \alpha_2\alpha_3 f, \alpha_2 f, \alpha_2 f\alpha_4, \alpha_2 f\alpha_4^2, \alpha_2\alpha_4, \alpha_2\alpha_4^2, \alpha_3, \alpha_3^2, \alpha_3^2 f, \ldots \tag{6.41}$$
$$\ldots \alpha_3^2 f\alpha_4, \alpha_3^2\alpha_4, \alpha_3 f, \alpha_3 f\alpha_4, \alpha_3 f\alpha_4^2, \alpha_3\alpha_4, \alpha_3\alpha_4^2, f, f^2, f^3, f^3\alpha_4, f^2\alpha_4, \ldots$$
$$\ldots f\alpha_4, f\alpha_4^2, f\alpha_4^3, \alpha_4, \alpha_4^2, \alpha_4^3$$

$$1, \alpha_2, \alpha_2^2, \alpha_2^2 f, \alpha_2\alpha_3, \alpha_2\alpha_3 f, \alpha_2\alpha_4, \alpha_2\alpha_4 f, \alpha_2 f, \alpha_2 f^2, \alpha_2 f^3, \alpha_3, \alpha_3^2, \alpha_3^2 f, \ldots \tag{6.42}$$
$$\ldots \alpha_3\alpha_4, \alpha_3\alpha_4^2, \alpha_3\alpha_4^2 f, \alpha_3\alpha_4 f, \alpha_3 f, \alpha_3 f^2, \alpha_3 f^3, \alpha_4, \alpha_4^2, \alpha_4^3, \alpha_4^3 f, \alpha_4^2 f, \alpha_4 f, \ldots$$
$$\ldots \alpha_4 f^2, \alpha_4 f^3, f, f^2, f^3$$

$$1, \alpha_2, \alpha_2^2, \alpha_2^2 f, \alpha_2^2 f\alpha_4, \alpha_2^2\alpha_4, \alpha_2 f, \alpha_2 f\alpha_3, \alpha_2 f\alpha_3\alpha_4, \alpha_2 f\alpha_4, \alpha_2 f\alpha_4^2, \alpha_2\alpha_3\alpha_4, \ldots \tag{6.43}$$
$$\ldots \alpha_2\alpha_3, \alpha_2\alpha_4, \alpha_2\alpha_4^2, f, f^2, f^3, f^3\alpha_4, f^2\alpha_4, f\alpha_3, f\alpha_3^2, f\alpha_3^2\alpha_4, f\alpha_3\alpha_4, \ldots$$
$$\ldots f\alpha_3\alpha_4^2, f\alpha_4, f\alpha_4^2, f\alpha_4^3, \alpha_3, \alpha_3^2, \alpha_3^2\alpha_4, \alpha_3\alpha_4, \alpha_3\alpha_4^2, \alpha_4, \alpha_4^2, \alpha_4^3$$

$$1, \alpha_2, \alpha_2^2, \alpha_2^2 f, \alpha_2\alpha_3, \alpha_2\alpha_3 f, \alpha_2\alpha_4, \alpha_2\alpha_4^2, \alpha_2\alpha_4^2 f, \alpha_2\alpha_4 f, \alpha_2 f, \alpha_2 f^2, \alpha_2 f^3, \ldots \tag{6.44}$$
$$\ldots \alpha_3, \alpha_3^2, \alpha_3^2\alpha_4, \alpha_3^2\alpha_4 f, \alpha_3^2 f, \alpha_3\alpha_4, \alpha_3\alpha_4^2, \alpha_3\alpha_4^2 f, \alpha_3\alpha_4 f, \alpha_3 f, \alpha_3 f^2, \alpha_3 f^3, \ldots$$
$$\ldots \alpha_4, \alpha_4^2, \alpha_4^3, \alpha_4^3 f, \alpha_4^2 f, \alpha_4 f, \alpha_4 f^2, \alpha_4 f^3, f, f^2, f^3$$

To be more precise, these standard monomial bases do not consist of monomials, e.g. $\alpha_4^3 f$ or $f\alpha_3\alpha_4$, but of cosets $\left[\alpha_4^3 f\right]$ or $[f\alpha_3\alpha_4]$. If it is clear from the context, we will identify costets in $A = \mathbb{C}[x_1, ..., x_n]/I$ with their monomial representatives and write x^α instead of $[x^\alpha] = x^\alpha + I$.

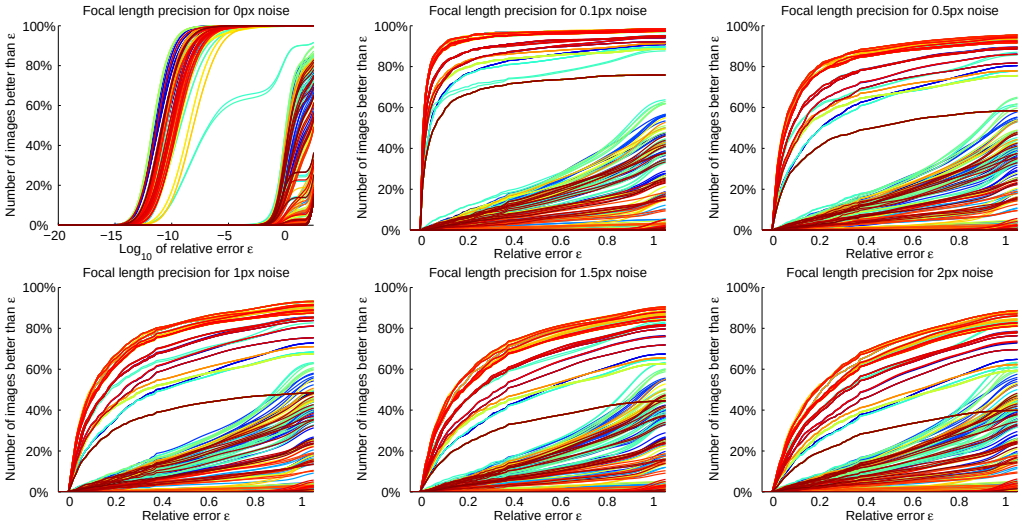


Figure 6.2: Precision of different P4P+f solvers based on the ratio constraints. Plots show the accuracy of the focal length estimation for different noise levels.

We fixed each out of these 21 bases and created different solvers for each possible combination - turning on/off substitution, doing/not doing initial Gauss-Jordan elimination, selecting random or proper instance, selecting different orderings of the unknowns and selecting different subsets of equations. This way we got 511 solvers. Note that not all of these 511 solvers produce correct solutions to the absolute pose problem. This is because the way how we created them was not always correct.

First, since the system of polynomial equations (6.10) is over-constrained and there are some dependencies between its coefficients, substituting random values into its known variables in the offline phase sometimes leads to a system which does not have a solution in general. It depends on a subset of equations which were selected. Some subsets of equations could be satisfied even with randomly instanced known variables. Secondly, a quotient ring basis could be obtained by using a random instantiation of known variables and the elimination template could be created with polynomials substituted with proper values. Vice versa, the proper instance could be used to find the quotient ring basis and a random instance could be used to create the elimination template. Such elimination templates [64] could be wrong. It is because algebraic dependencies might not be preserved. Then, some elements of the elimination template could become wrongly non-zero or vanish, while opposite was true during the template creation process in the offline phase. Consequently, the elimination template could have a different rank than expected on real data. This leads to incorrect action matrix and finally to incorrect results.

Solvers performance

To study the behavior of solvers we evaluated them on a set of synthetic scenes. Synthetic scenes were created by randomly distributing points within a 3D cube. Each 3D point was projected

by a randomly placed camera with a feasible position so that whole image plane was covered with image projections. Gaussian noise with standard deviation σ was added to the image points assuming 512×512 image size. A different amount of noise with $\sigma = \{0, 0.1, 0.5, 1, 1.5, 2\}$ image pixels was used in each experiment. For each noise level we generated 1000 random scenes. We focused on numerical stability and precision of each solver. Figure 6.2 displays results focusing on the precision of focal length estimation for all 511 solvers. Plots display the number of focal lengths (vertical axis) that were estimated with better relative precision than relative error threshold (horizontal axis). This means that the sooner a solver reaches 100%, the more precise the solver is. The relative error is measured as $(f - f_{gt})/f_{gt}$ where $f_{gt} = 1.5$ which corresponds to $50mm$ with respect to the $35mm$ film size.

Figure 6.2 (top-left) shows results for noise free data. Instantiation with a noise free data can be viewed as an instantiation with the proper data during the elimination template creation process. Hence it is expected that solvers which were generated using proper instances will behave well. However, some smaller differences may appear since the solver generator operates in a finite prime field rather than in the floating point arithmetics to avoid rounding and precision issues. Hence we can say that algebraic relations are preserved only up to the machine precision. There is a group of 155 solvers which behave accurately on noise free data. These solvers form the left group of plots in the top-left graph of Figure 6.2. It is interesting to see how differently solvers cope with increasing noise level in next figures. Most of solvers from the group of accurate 155 solvers from the top-left figure work well on noisy data too. They form a group of curves on the top of Figure 6.2 with non-zero noise level.

Table 6.1 shows details which are needed to create solvers using the solver generator. The table is ordered by the solver accuracy of estimation at 1 pixel noise level. Many solvers performed equally and in such cases the table displays parameters for the fastest solvers only. The table shows the size of the elimination template, the number of solutions and the percentage of images where the focal length was estimated with better relative error than $0.1, 0.2, 0.3$ and 0.4. Further, it indicates whether Gauss-Jordan elimination was used (GJ column), it shows the type of variable instantiating method and indices of equations which were used to build the elimination template. Substitution $w = f^2$ was used in all cases.

Table 6.1 shows how the selection of basis and the selection of equations significantly influence the way how solvers cope with increasing noise. Note that the speed of a solver is proportional to the size of its elimination template and to the number of solutions since computational complexity of the Gauss-Jordan elimination as well as the complexity of the eigenvalue decomposition grows cubically with the size of a matrix. It is sometimes better to choose a smaller, not so accurate, but faster solution and then further improve the solver results using a numerical optimization method such as Newton iteration [87]. We will study this in detail in Section 6.8.2. Moreover, Chapter 9 studies methods which can be used to significantly speed up the above solvers.

The best solvers

Detail of the most accurate and small and fast solutions follow:

- The smallest solver which belongs to the group of the most precise solvers:

size	sols	0.1	0.2	0.3	0.4	GJ	instance	equations
139×153	14	51%	67%	76%	81%	yes	random	$[1, 2, 3, 4]$
246×262	16	52%	67%	75%	80%	no	random	$[1, 3, 4, 5, 7, 8, 9, 13, 14, 15]$
236×252	16	51%	66%	75%	80%	no	proper	$[1, 2, 3, 15]$
86×96	10	48%	65%	74%	80%	yes	random	$[1, 2, 4, 5]$
210×228	18	46%	63%	73%	78%	no	random	$[1, 2, 13, 14]$
113×131	18	47%	62%	72%	77%	no	random	all
67×81	14	47%	62%	71%	76%	yes	random	all
83×101	18	40%	56%	65%	71%	yes	proper	all
65×79	14	39%	55%	63%	68%	yes	proper	all
104×131	28	39%	53%	60%	65%	yes	random	$[1, 2, 3, 4]$
96×114	18	33%	47%	56%	63%	no	random	all
53×63	10	31%	44%	53%	59%	yes	proper	all
71×81	10	31%	44%	51%	55%	yes	proper	all

Table 6.1: Details of selected ratio based solvers for P4P+f problem.

Solution size: 139×153,
Number of solutions: 14,
Quotient ring basis:$\{1, w, w\alpha_2, w\alpha_3, w\alpha_4, \alpha_2, \alpha_2^2, \alpha_2\alpha_3, \alpha_2\alpha_4, \alpha_3, \alpha_3^2, \alpha_3\alpha_4, \alpha_4, \alpha_4^2\}$,
Unknowns ordering: $w > \alpha_2 > \alpha_3 > \alpha_4$,
Monomials ordering: grevlex,
Instancing method: both proper or random can be used,
Equations used to build elimination template: $[1, 2, 3, 4]$,
Both substitution $w = f^2$ and initial Gauss-Jordan elimination were used.

- Still very precise solver which is small and about $4\times$ faster compared to the solver above:
  Solution size: 86×96,
  Number of solutions: 10,
  Quotient ring basis:$\{1, \alpha_2, \alpha_2\alpha_4, \alpha_3, \alpha_3\alpha_4, w, w^2, w\alpha_4, \alpha_4, \alpha_4^2\}$,
  Unknowns ordering: $\alpha_2 > \alpha_3 > w > \alpha_4$,
  Monomials ordering: grevlex,
  Instancing method: both proper or random can be used,
  Equations used to build elimination template: $[1, 2, 4, 5]$,
  Both substitution $w = f^2$ and initial Gauss-Jordan elimination were used.

The above parameters can be used to create both solvers with our automatic generator [64].

6.5 Creating an optimal distance solver

Now return back to the basic constraint from Equation (6.7). After expanding and substituting $w = f^2$ we obtain:

$$\alpha_i^2 x_i^2 - 2\alpha_i x_i \alpha_j x_j + \alpha_j^2 x_j^2 + \alpha_i^2 y_i^2 - 2\alpha_i y_i \alpha_j y_j + \alpha_j^2 y_j^2 + \alpha_i^2 w - 2\alpha_i w \alpha_j + \alpha_j^2 w - d_{12}^2 = 0. \quad (6.45)$$

Given four reference 3D points and four measurements, we get six such polynomial equations in five unknowns - four unknown depths α_i and the substituted focal length $w = f^2$. This system of polynomial equations is again over-constrained, i.e. one of the equations is not needed for solving the system. To be correct, all equations can be used to solve the system, however not all polynomial equations will be satisfied for noisy data. Thus finding a solution satisfying all six equations may not be always possible. As in the previous section, we can select five out of these six equations and solve them. There are 6 such choices and for each choice of input equations we get a solution which satisfies selected five-tuple of equations, however the remaining sixth equation will not be satisfied perfectly.

We focus on the Gröbner basis method only, since we found this system of polynomial equations too challenging for other methods. The reason is that the number of monomials in equations grows rapidly when adding multiples of the input equations and the resulting solvers are too big and hence impractical for real applications. We will use the advantage of our automatic generator [64] which allows automatic generation of thousands of solvers, which is not possible by hand. This way we can study all possible choices of equations, options for creating solver like selecting proper or random instance, doing/not doing the initial Gauss-Jordan elimination, substituting $w = f^2$ or not, changing ordering and so on.

6.5.1 Exhaustive search for the best solver

Let's first create a solver generator script for the given problem. We will build on the previous script for solvers based on the ratio of distances. Compared to the ratio based solvers, now we are solving depths of 2D image measurements directly, including the depth of the first point - α_1. Hence we must make this depth unknown too and re-parametrize the position of the first 3D point in the camera coordinate system.

```
% depth of the first point
syms a1;

XC1 = a1*[x1; y1; f];
```

Now we create the system of polynomial equations according to equations (6.7).

```
% express distances between 3D points in camera coordinate
% system
d12 = (sum((XC1-XC2).^2));
d13 = (sum((XC1-XC3).^2));
d14 = (sum((XC1-XC4).^2));
d23 = (sum((XC2-XC3).^2));
d24 = (sum((XC2-XC4).^2));
d34 = (sum((XC3-XC4).^2));
```

```
% distances between points in camera and world coordinate
% system must be the same - this defines our system of
% polynomial equations
eq(1) = d12 - D12;
eq(2) = d13 - D13;
eq(3) = d14 - D14;
eq(4) = d23 - D23;
eq(5) = d24 - D24;
eq(6) = d34 - D34;
```

The focal length f appears in its square only, hence we can optionally substitute $f^2 = w$. We will investigate behavior of solvers created both with and without substitution later.

```
eq = expand(eq);
% substitute f^2 = w
for i=1:size(eq,2)
   str = char(expand(eq(i)));
   str = strrep(str, 'f^2', 'w');
   eq(i) = sym(str);
end
```

Finally, we need to extend the set of unknown variables with the first depth - `a1`.

```
unknown = [unknown 'a1'];
```

Optionally, we can use the proper instance generator which we used in the previous section:

```
cfg.InstanceGenerator = @mbp4pf_instanceGenerator;
```

Solvers performance

As in the previous section, we exhaustively searched for all possible bases coming from the selections of different subsets of equations and by changing parameters of the generating script above (substitution, Gauss-Jordan elimination and so on). First, we obtained 21 unique bases giving $2, 4, 12, 20, 24$ and 40 solutions to this formulation of P4P+f problem. Then, we generated a solver for each Gröbner basis, again by evaluating all possible choices of parameters for the solver generating code. We obtained 242 solvers which we further evaluated. Figure 6.3 shows the accuracy of the focal length estimation for a noise free (top left), 0.1, 0.5 pixel noise (top-middle and right) and 1, 1.5 and 2 pixel noise (bottom row).

Table 6.2 shows details of best solvers ordered with respect to the accuracy of the focal length estimation in scenario with 1 pixel noise level. For solvers, which return the same results, we list

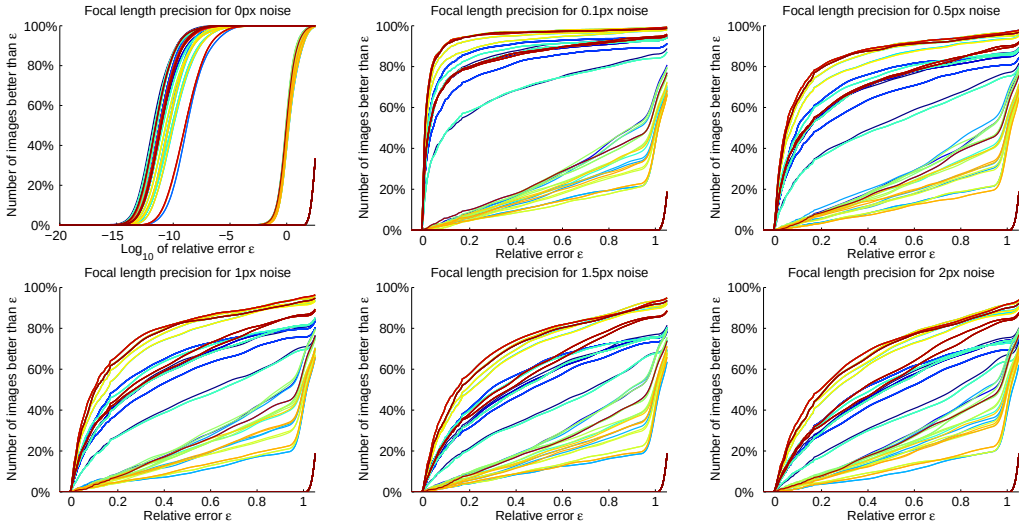


Figure 6.3: Precision of different P4P+f solvers based on the distance constraint. Plots show the accuracy of the focal length estimation for different noise levels.

only the fastest solver with the smallest elimination template (the first column of the table) or the smaller number of solutions (the second table column). The table further shows the percentage of images which were estimated with relative error smaller than $0.1, 0.2, 0.3$ and 0.4. It is visible that solvers based on the ratio constraint perform slightly better compared to the distance based solvers. Moreover, the best "ratio based" solver is more than $30\times$ faster compared to the best solver based on the distance constraint.

6.5.2 Single solution case

During the automatic solver generation, we have generated several linear, single solution solvers of the problem. In fact these solvers have two solutions, however only one of them is valid. These two solutions represent positive and negative depth pairs. Depending on the selected unknowns ordering, it can be any depth - α_i. For such simple case it is not necessary to create an action matrix to find the solution. Let's have a look at monomials of one of those solvers, for example the 193×195 solver. Its elimination template and monomials used can be exported during the solver creation. The last few monomials are

$$\ldots \; \alpha_3\alpha_2 \quad \alpha_3^2 \quad \alpha_4\alpha_1 \quad \alpha_4\alpha_2 \quad \alpha_4\alpha_3 \quad \alpha_4^2 \quad \alpha_4 \quad 1. \tag{6.46}$$

Reducing the elimination template during the online phase clears the left 193×193 sub-matrix to identity and makes the last three columns non-zero. These three columns correspond to monomials α_4^2, α_4 and 1. It means that the solution path obtained using the automatic generator in the offline phase contains a polynomial in a single variable; which can be found in the last row of the reduced elimination template. It is a second degree polynomial in some of the unknown depths depending on the variables ordering, e.g. α_4 in this case. Moreover, since it represents

size	sols	0.1	0.2	0.3	0.4	GJ	subst	instance	equations
561×581	20	51%	67%	77%	81%	no	yes	proper	$[1, 2, 3, 4, 5]$
256×276	20	48%	65%	75%	80%	yes	yes	random	$[1, 2, 3, 4, 5]$
167×179	12	48%	65%	74%	80%	yes	yes	random	$[1, 2, 3, 4, 5]$
224×244	20	46%	63%	72%	78%	yes	yes	proper	all
330×350	20	42%	58%	69%	75%	no	yes	random	all
92×96	4	36%	51%	59%	66%	yes	yes	proper	all
193×195	2	35%	49%	58%	64%	no	no	proper	all
237×249	12	32%	46%	55%	62%	no	yes	random	all
171×183	12	32%	45%	53%	60%	yes	yes	proper	all
76×78	2	30%	44%	53%	60%	yes	no	random	$[1, 2, 3, 4, 5]$
394×398	4	29%	43%	52%	59%	no	no	random	$[1, 2, 3, 4, 5, 6]$
76×78	2	29%	43%	52%	58%	yes	no	proper	$[1, 2, 3, 4, 5, 6]$
129×133	4	27%	39%	47%	54%	no	yes	proper	all
96×98	2	18%	28%	35%	41%	no	no	random	$[1, 2, 3, 4, 5]$
103×105	2	17%	27%	34%	39%	no	no	proper	$[1, 2, 3, 4, 5, 6]$

Table 6.2: Details of selected distance based solvers for P4P+f problem.

the pair of positive and negative solutions, the linear term is always zero. This is also visible in the reduced elimination template. Hence we have:

$$\beta_{[193,193]}\alpha_4^2 + \beta_{[193,195]} = 0, \tag{6.47}$$

where $\beta_{[i,j]}$ is the value in i^{th} row and j^{th} column of the matrix obtained after the Gauss-Jordan elimination of the elimination template. Consecutively, we do not need to create an action matrix in order to obtain a solution in the online phase. Instead, it is possible to read the solution directly from β_i factors after the Gauss-Jordan elimination of the elimination template. Remaining depths $\alpha_1, \alpha_2, \alpha_3$ can be read from rows above the last row:

$$\begin{aligned}
\alpha_4 &= \sqrt{-\beta_{[193,195]}}, \\
\alpha_3 &= -\beta_{[192,195]}/\alpha_4, \\
\alpha_2 &= -\beta_{[191,195]}/\alpha_4, \\
\alpha_1 &= -\beta_{[190,195]}/\alpha_4.
\end{aligned}$$

The focal length f can be extracted similarly, or calculated from one of the input equations:

$$f = \sqrt{-\frac{\alpha_1^2(y_1^2 + x_1^2) - 2\alpha_1\alpha_2(y_2y_1 - x_2x_1) + \alpha_2^2(y_2^2 + x_2^2) - d_{12}^2}{\alpha_2^2 - 2\alpha_1\alpha_2 + \alpha_1^2}}. \tag{6.48}$$

To conclude this section, one advantage of single solution solvers is their speed and further that solutions do not need to be filtered afterward to select the correct one. The drawback of these

solutions is their lower accuracy compared to other solvers. However, for some applications it is sometimes better to select a faster solution and then further improve its result using some numerical optimization method.

6.6 Rotation matrix parametrized using quaternions

In the previous section we used Euclidean rigidity constraint to get rid of the camera rotation and translation components. In this section we present a different formulation of P4P+f problem in which we use quaternions to parametrize the rotation matrix $\mathtt{R}$ as in [56].

Let $\mathbf{x}_i = [x_i, y_i, 1]^\top$ be an image measurement, and let $\mathbf{X}_i$ be its corresponding 3D point. Recall that the matrix equation capturing the relationship between projected $\mathbf{x}_i$ and $\mathbf{X}_i$ can be expressed as:

$$[\mathbf{x}_i]_\times \ \mathtt{K}\,[\mathtt{R} \,|\, \mathbf{t}]\mathbf{X}_i = \mathbf{0}, \tag{6.49}$$

where the calibration matrix $\mathtt{K} = diag\,[1, 1, w]$ for $w = 1/f$ and $\mathbf{t}$ is the camera translation vector and $[\mathbf{x}_i]_\times$ denotes the skew symmetric matrix representing the cross product. We parametrize the rotation matrix by quaternions [61]

$$\mathtt{R} = \begin{bmatrix} a^2+b^2-c^2-d^2 & 2bc-2ad & 2ac+2bd \\ 2ad+2bc & a^2-b^2+c^2-d^2 & 2cd-2ab \\ 2bd-2ac & 2ab+2cd & a^2-b^2-c^2+d^2 \end{bmatrix}. \tag{6.50}$$

Now, the projection matrix is parametrized using eight parameters - four quaternion components, three translation components and the focal length.

Reducing the number of unknowns

Since the projection matrix is defined up to scale, we can set one from the quaternion parameters to one, e.g. $a = 1$. Using parameterization (6.50) the projection equation (6.49) generates two linearly independent equations for every given 2D-to-3D correspondence in 7 unknowns – the quaternion parameters b, c, d, the translation parameters t_x, t_y, t_z and the calibration parameter $w = 1/f^2$.

These equations can be further simplified by appropriate choice of a coordinate system. Here we use the same choice of the coordinate system as in [56], however in this case also other choices can be used. Without the loss of generality, let the first two 3D points $\mathbf{X}_1, \mathbf{X}_2$ and the first image point $\mathbf{x}_1$ have the homogeneous coordinates:

$$\mathbf{X_1} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \mathbf{X_2} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \mathbf{x_1} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}. \tag{6.51}$$

This can be achieved easily since all four general 3D points and their projection can be transformed to (6.51) using a similarity transformation and without breaking the validity of the relationship (6.1). Using this assumption the projection equation (6.49) gives us two linearly

independent equations for the first point correspondence

$$t_y = 0, \tag{6.52}$$

$$t_x - w\,t_z = 0. \tag{6.53}$$

After substituting (6.52) and (6.53), i.e. $t_x - wt_z = 0$ and $t_y = 0$, into the projection equation (6.1) and after evaluating Equation (6.49) for the second point correspondence we obtain:

$$-2d - 2bc + y_2 w(2bd - 2c) + y_2 wtz = 0, \tag{6.54}$$

$$1 + b^2 - c^2 - d^2 - x_2 w(2bd - 2c) + w\,t_z - x_2 w\,t_z = 0, \tag{6.55}$$

$$-y_2(1 + b^2 - c^2 - d^2) + x_2(2d + 2bc) - y_2 w\,t_z = 0. \tag{6.56}$$

From this, we can express $w\,t_z$ as a function of b, c, d, we used Equation (6.56):

$$w\,t_z = -(1 + b^2 - c^2 - d^2) + \frac{x_2}{y_2}(2d + 2bc). \tag{6.57}$$

Finally, using relations (6.52), (6.53) and (6.57) we eliminated all three variables t_x, t_y and t_z from the projection matrix, which becomes:

$$\mathtt{P} = \begin{bmatrix} 1 + b^2 - c^2 - d^2 & 2bc - 2d & 2c + 2bd & w\,t_z \\ 2d + 2bc & 1 - b^2 + c^2 - d^2 & 2cd - 2b & 0 \\ w(2bd - 2c) & w(2b + 2cd) & w(1 - b^2 - c^2 + d^2) & w\,t_z \end{bmatrix}. \tag{6.58}$$

So far, we have used only the first point correspondence and one of the constraints from the second correspondence. Hence eight polynomial equations coming from relation (6.49) in four unknowns w, b, c and d remain. Indeed, only five of them are linearly independent polynomial equations. The maximal degree of these equations is again three.

Using the automatic generator [64] we can generate a linear solver for the above set of equations. We will show in experiments section that this solver, and its variation, does not perform very well and many times fails to deliver a real solution.

A different formulation which assumes that three points are on the plane $z = 0$ performs much better. Rotation component of projection matrix stays unchanged, but the translation vector become:

$$-2d - 2bc = t_y, \tag{6.59}$$

$$\frac{x_1}{y_1} t_y = t_x, \tag{6.60}$$

$$\frac{t_y}{y_1} = w\,t_z. \tag{6.61}$$

The projection matrix then becomes:

$$\mathtt{P} = \begin{bmatrix} 1 + b^2 - c^2 - d^2 & 2bc - 2d & 2c + 2bd & t_x \\ 2d + 2bc & 1 - b^2 + c^2 - d^2 & 2cd - 2b & t_y \\ w(2bd - 2c) & w(2b + 2cd) & w(1 - b^2 - c^2 + d^2) & t_z \end{bmatrix}, \tag{6.62}$$

where t_x, t_y and t_z are expressed using (6.59), (6.60) and (6.61).

Creating the solver

We used the automatic generator [64] with custom instance generator to find solutions to the previous formulations. The instance generator is similar to the one from Section 4.6. It consists of the following steps:

- create a random focal length, rotation matrix and translation vector,
- create projection matrix $\mathtt{P} = \mathtt{K}\,[\mathtt{R} \mid \mathbf{t}]$,
- project four 3D points where first two are as in (6.51),
- rotate 2D projections around the camera principal axis [47] so that the first 2D point becomes as in (6.51).

As in the previous sections, we performed precision analysis, selected different subsets of equations and selected different monomial orderings, and so on. We found a linear solver and solvers with $16, 20$ and 40 solutions. We got the linear solver using the first formulation (6.58). Its elimination template has size 113×115 and produces two solutions to the problem. Solutions are linked with the positive and the negative focal lengths where only the positive focal length is valid. We obtained the best results using the second quaternion formulation (6.62) and the solver which was generated using all equations. Its size is relatively small, just 65×81.

The size of the elimination template changes only marginally when changing the ordering of monomials. Also performing or skipping Gauss-Jordan elimination did not make the solver smaller, neither helped increasing its precision. Overall comparison with other P4P+f solvers, described in this chapter, can be found at the end of this chapter.

6.7 Projection matrix parametrized using a null space

In the previous sections we used depths to eliminate rotation and translation of the camera. Next we parametrized camera translation, rotation and focal length to build camera projection matrix $\mathtt{P}$ defined in (6.1). In this section we investigate the other direction and we will study the space where projection matrices live.

First, let's forget about structure of the projection matrix and let's have a look at the basic projection equation (6.1)

$$\lambda_i \, \mathbf{x}_i = \mathtt{P}\, \mathbf{X}_i. \tag{6.63}$$

We can rewrite this relation in terms of unknown camera elements

$$\mathtt{M}\mathbf{p} = \mathbf{0}, \tag{6.64}$$

where $\mathbf{p}$ is vector consisting of 12 elements of the projection matrix $\mathtt{P}$ and $\mathtt{M}$ is a $2N \times 12$ matrix for N 2D-to-3D point correspondences.

Each 2D-to-3D correspondence contributes with two rows to the matrix $\mathtt{M}$, the same way like in direct linear transform algorithm (DTL) [47]. Hence, given four non-degenerated point correspondences, we obtain eight linear homogeneous equations and matrix $\mathtt{M}$ becomes 8×12.

Consequently, $\mathtt{M}$ has a four dimensional kernel where solutions for $\mathbf{p}$, i.e. for the projection matrices $\mathtt{P}$, exist

$$\mathbf{p} = \sum_{i=1}^{4} \alpha_i \mathbf{p}_i. \tag{6.65}$$

In this way the projection matrix $\mathtt{P}$ is parameterized using four unknowns $\alpha_1, \alpha_2, \alpha_3$ and α_4. Since multiplying the camera matrix by a scalar value does not affect the projection, we will fix one of unknown α_i to 1, say $\alpha_4 = 1$.

Now, we will use the fact that our projection matrix has a certain structure to find unknowns $\alpha_1, \alpha_2, \alpha_3$. First, the three rows of the 3×3 submatrix of the projection matrix $\mathtt{P}$ are perpendicular and the first two rows of this submatrix have the same norm. These constraints follow from the fact that the 3×3 submatrix of the projection matrix $\mathtt{P}$ has the form $\mathtt{K}\,\mathtt{R}$, where $\mathtt{R}$ is a rotation matrix. In this way we obtain four quadratic equations in three unknowns $\alpha_1, \alpha_2, \alpha_3$

$$p_{11}p_{21} + p_{12}p_{22} + p_{13}p_{23} = 0, \tag{6.66}$$

$$p_{31}p_{11} + p_{32}p_{12} + p_{33}p_{13} = 0, \tag{6.67}$$

$$p_{31}p_{21} + p_{32}p_{22} + p_{33}p_{23} = 0, \tag{6.68}$$

$$p_{11}^2 + p_{12}^2 + p_{13}^2 - p_{21}^2 - p_{22}^2 - p_{23}^2 = 0, \tag{6.69}$$

where p_{ij} is the element at the i^{th} row and j^{th} column of the projection matrix $\mathtt{P}$.

A similar formulation has been used by Triggs in [100] where the properties of the dual image of the absolute quadric (DIAC) [47]

$$\omega \equiv \mathtt{P}\,\Omega\,\mathtt{P}^\top \simeq \mathtt{K}\,\mathtt{K}^\top \tag{6.70}$$

were used to constrain the search for the unknown combination of nullspace vectors $\mathbf{p}_i$. Triggs used a resultant to solve the problem. Both, our formulation above as well as Triggs's formulation, work only for a non-planar scene, i.e. when input 3D points are not on a single plane. In the next section we will build an algorithm which works for planar scenes only. Further, we will show how to mix the non-planar and the planar algorithm to build a general purpose algorithm. We will show that this algorithm works well even in near planar scenarios.

Now let's return back to the system of equations. Using constraints (6.66)-(6.69), i.e. that three rows of the 3×3 submatrix of the projection matrix $\mathtt{P}$ are perpendicular and that the first two rows of this submatrix have the same norm, we obtain four quadratic equations in three unknowns α_1, α_2 and α_3.

Again we have one more equation than unknowns. By selecting different subsets of constraints (6.66)-(6.69) we affect the final solution. For example, selecting (6.66)-(6.68) will result in projection matrix $\mathtt{P}$ whose 3×3 submatrix is orthogonal, but the length of rotation components will not be the same. Such solution is equivalent to assuming that the calibration matrix has unknown aspect ratio too:

$$\mathtt{K} = \begin{bmatrix} \alpha f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{6.71}$$

We used automatic solver generator [64] to create all possible solvers w.r.t. ordering, choices of equations, and using random and proper instances. This way we generated 120 unique solvers. Using different variables ordering did affect neither the precision nor the solver size. We evaluated all solvers using synthetically generated scenes with growing noise level. The following table shows the precision of solvers in configuration with 1 pixel noise for image with size 512×512 pixels. Solvers are ordered by their accuracy and the fastest solver from each group of solvers with same accuracy is shown:

size	solutions	0.1	0.2	0.3	0.4	instance	equations
26×34	8	44%	58%	67%	73%	random	$[1, 2, 3]$
30×34	8	32%	45%	54%	60%	random	all
33×34	1	21%	31%	37%	41%	proper	all

Using the solvers above we can obtain eight or a single solution for α_1, α_2 and α_3. The focal length can be calculated from the fact that 3×3 submatrix of the projection matrix `P` corresponds to the camera rotation multiplied by the calibration matrix, i.e. `KR`. Hence the length of the first row of this submatrix is f times smaller than the length of the third row, hence

$$f^2\, p_{11}^2 + f^2\, p_{12}^2 + f^2\, p_{13}^2 - p_{31}^2 - p_{32}^2 - p_{33}^2 = 0. \tag{6.72}$$

6.7.1 Solver for a planar scene

The solution to the planar case of the absolute pose problem for a camera with unknown focal length is very simple and can be formulated in a closed-form.

Without loss of generality, we can assume that all four 3D points $\mathbf{X}_i$ have the third coordinate $Z_i = 0$. This can be achieved simply by rotating 3D points. Now, looking at the expanded projection Equation (6.49)

$$\begin{bmatrix} 0 & -1 & y_i \\ 1 & 0 & -x_i \\ -y_i & x_i & 0 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ w\,r_{31} & w\,r_{32} & w\,r_{33} & w t_z \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \\ 0 \\ 1 \end{bmatrix} = \mathbf{0}. \tag{6.73}$$

we see that the third column in the projection matrix, i.e. r_{13}, r_{23} and wr_{33} vanishes. Hence it is sufficient to use elements of the first, the second and the fourth column of the projection matrix `P` to describe the projection. Note that this is in fact homography `H` between 2D points and 3D points in the plane, which can be linearly calculated from four points, which we have [47].

Solutions to the focal length and for the third column of the projection matrix `P` can be then easily calculated from the structure of the projection matrix and properties that columns of the rotation matrix are perpendicular and have the same norm, i.e.

$$\mathtt{H} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & t_1 \\ r_{21} & r_{22} & t_2 \\ w\,r_{31} & w\,r_{32} & t_3 \end{bmatrix}, \tag{6.74}$$

where $w = f^2$. Hence $f = ((h_{11}h_{12} + h_{21}h_{22})/(h_{31}h_{32}))^{1/2}$ and the third column of the rotation matrix is $r_3 = r_1 \times r_2$.

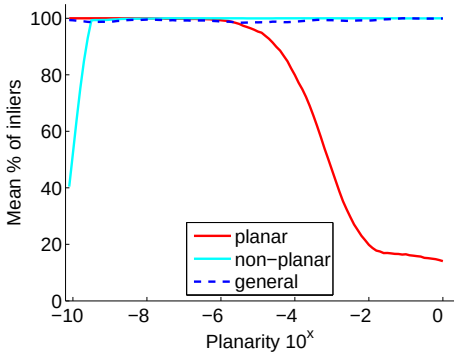


Figure 6.4: Results of the experiment on the near-planar scene for P4P+f planar and non-planar solvers.

6.7.2 General solver

In this section we study behavior of non-planar and planar solvers, described in previous sections, on near-planar scenes in order to show how to build a fast joined solver by combining two specialized solvers. For this purpose we created a synthetic scene where we could control scene planarity by a scalar value a. Given the planarity a, we construct a scene as follows. Assume we have a random synthetic scene generated like in Section 5.5 with virtual camera providing images with 512×512 pixels. Let's denote by ρ the plane created from the first three non-collinear 3D points and denote by s a normalizing scale. We calculate the scale s as the distance of the furthest point of these three points from their center of gravity CG. Then, we randomly generate the fourth point at the distance $(s\ a)$ from the plane ρ and such that it is not further than s from the center of gravity CG. Note that for planarity $a = 0$ we get four points on the plane and for $a = 1$ we obtain a well defined non-planar four-tuple of 3D points.

We did not contaminate the image points corresponding to these four 3D points by noise. We added noise with deviation of 0.5 pixels only to the remaining image points. In this way we created a scene with one uncontaminated four-tuple of 2D-to-3D point correspondences for which we can control their planarity by a scalar value a.

Next, for each given planarity value a, we created a scene and calculated the camera pose from the four-tuple of correspondences using the planar, the non-planar and the general solver [12]. Note that this four-tuple is not affected by noise and hence the only deviation from the ground truth solutions comes from the numerical instability of the solvers itself. To evaluate the impact of the instability on the solution we used the estimated camera and the focal length to project all 3D points to the image plane. Then we measured how many points were projected closer than one pixel to its corresponding 2D image - we call them inliers.

Figure 6.4 shows results for the planar (Red), the non-planar (Cyan) and the general (Dashed-Blue) solver. In this experiment we created 100 random scenes for each given planarity value a and evaluated all algorithms. The mean value is displayed.

The interesting point in Figure 6.4 is the intersection of the planar and the non-planar solvers.

This point defines the threshold of planarity below which it is better to use the planar algorithm and non-planar algorithm gives better precision above this threshold. It can be seen that the non-planar solver can be used immediately once the scene is not completely planar. If speed is important then non-planar solver should be used whenever $a > 10^{-7}$ since the planar solver is stable below this threshold, and it is much faster for near-planar scenes compared to the non-planar solver.

Also, a different strategy, which executes both the planar and the non-planar solvers simultaneously and select the better result, performs very well. It does not yield any slowdown since calculating planar solver takes less than 1μs. Moreover, the overall computation time is much smaller, compared to the general solver.

In the next section we will compare how this algorithm compete with other P4P+f solvers described in this chapter.

6.8 Algorithm comparisons

In this section we compare the performance of algorithms in synthetic and real experiments. For each type of constraint and method used in the previous sections, we selected several solvers for comparison.

Algorithms

Our aim was not to compare the best with the best, but to select some reasonable sample of solvers and study their performance in real applications. We selected solvers:

- the best solver of each type w.r.t. accuracy;
- some smaller, faster and yet still quite accurate solvers;
- next we selected solvers with the smallest elimination templates since we wanted to know if further non-linear optimization, such as Newton iteration, can be used to improve their results, while still be able to fit into a time budget of some of more accurate solvers;
- finally, we selected some linear solver or solver with the smallest number of solutions. Again we wanted to study its performance together with a non-linear optimization.

This way we selected:

- the ratio constraint bases solvers: (1) the best (139×153), (2) the 4^{th} best solver (86×96) which is still very accurate solver and several factors faster then the best solver, (3) the 7^{th} best solver (67×81) which is smaller and slightly less accurate then the previous solver and finally (4) solver (53×63) which is the fastest ratio based solver,
- the distance constraint based solvers: (1) the best (561×581), (2) the 3^{rd} best, smaller, yet still accurate solver (167×179), and two linear solvers (3) the 6^{th} best (92×96) solver and (4) the 10^{th} best (76×78) which is also the smallest distance based solver,

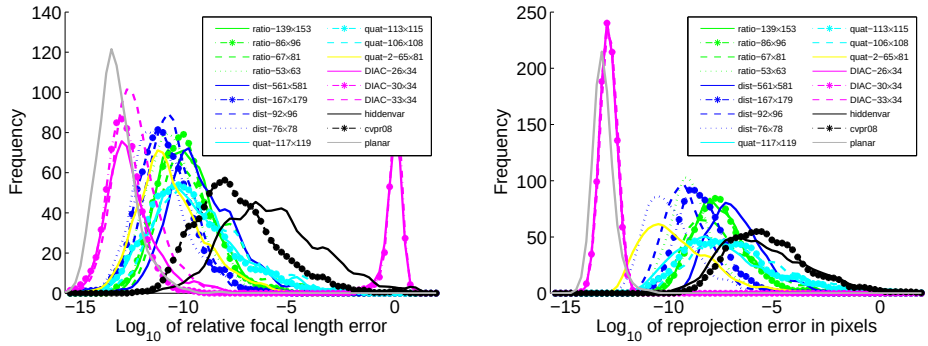


Figure 6.5: Numerical stability of investigated p4p solvers. Relative focal lengths error on the left and re-projection error on the right.

- quaternion solvers - (1)-(3) three linear solvers obtained by skipping different input equations and (4) the solver using all equations and giving 16 solutions (65×81),
- all three DIAC solvers from Section 6.7,
- the hidden variable solver,
- the original CVPR'08 solver [12],
- and the planar solver for planar scene experiment.

6.8.1 Synthetic datasets

In all synthetic experiments we created synthetic scenes as follows. First, a random 3D scene was created by randomly distributing points within a 3D cube, on a plane, or combination of both depending on the testing configuration. Next, we created cameras, which were randomly oriented and distributed in the 3D space. The focal length was either randomly selected or a fixed value was used. Each 3D point from the scene was projected by these cameras. Only points in the view frustum and in the front of camera were considered. Finally, Gaussian noise with standard deviation σ was added to the image points assuming 512×512 image size.

Numerical stability

In the first experiment we focus on the numerical stability of the selected solvers. We created a synthetic scene as described above. In this experiment we randomly created one thousand of scenes with five 3D points and a single camera. All solvers were evaluated using the same four out of these five points. When a solver returned multiple solutions we used the fifth 3D point to select the solution with the smallest re-projection error on all five points. Figure 6.5 displays relative focal length with respect to the ground truth focal length $(f - f_{gt})/f_{gt}$ on the left side and the sum of squared distances between re-projected 3D points and corresponding 2D points

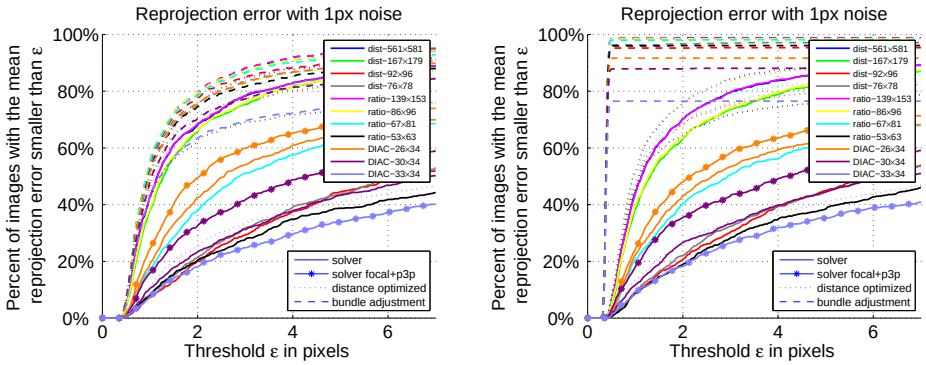


Figure 6.6: Synthetic noise experiment with 1px noise w.r.t. 512×512 image resolution. Left figure shows results with Bundle adjustment executed on four input point correspondences, right side with all 50 points.

on the right side. From the figure it is clear that except the hidden variable and DIAC solvers, all examined algorithms are sufficiently numerically stable. DIAC solvers failed to deliver a correct solution for planar scenes. For planar solver we generated pure planar scenes.

6.8.2 Algorithms accuracy

In previous experiment we used exact data - 2D and 3D point correspondences. Running solvers on precise data does not show differences w.r.t. accuracy between methods and also it does not uncover how important it is to select a proper subset of equations when solving this over-constraint problem. This experiment focuses on accuracy of the solvers when noise is added. Again we created a synthetic scene with 50 2D-to-3D point correspondences and added $1pixel$ noise into the 2D measurements. For comparison we selected the most accurate solvers and also fast solvers. A question arises whether fast solvers can be numerically improved to achieve the accuracy of slower but more accurate solvers. In this experiment we used ratio constraint, distance constraint and non-planar DIAC solvers. We omitted quaternion solvers since, as it will be shown later, the smallest quaternion gives the best result. On the other hand this solver is slower than other more accurate and smaller ratio constraint based solvers.

We executed each solver on the same data, calculated the camera pose and evaluated re-projection error on all 50 scene points. Figure 6.6 shows results for thousands random scenes. Each plot shows how many times (vertical) was average re-projection error smaller than a given threshold (horizontal). Colored solid lines show performance of directly evaluated solvers. Star lines show results, where we used focal length calculated by some solver, then we calibrated 2D measurements using this focal length, and used the calibrated P3P to calculate camera pose. Recall both ratio and distance constraint base problems estimate depths of image measurements and recover 3D model in camera coordinate system. This model is parametrized using four depths, Equation (6.7). We used Newton's method to minimize distances from Equation (6.7) on all six distance equations to improve solutions. As an initialization of the Newton's algorithm

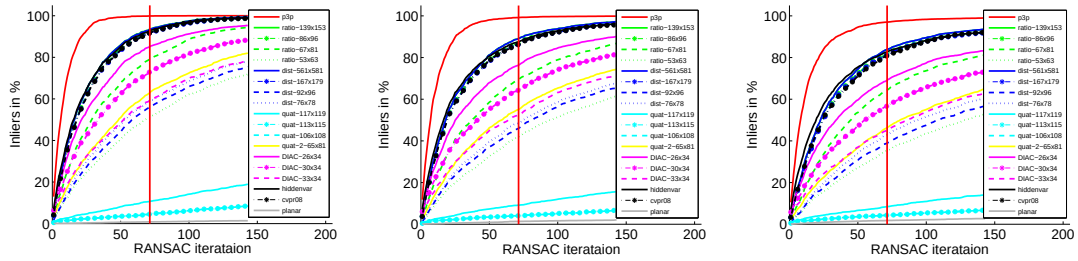


Figure 6.7: Results of 1000 execution of RANSAC algorithm on synthetic data with 50% outliers and noise level 0.5 (left), 1 (middle) and 2 image pixels (right).

we used depths estimated by solvers. Dotted lines show such optimized results. Note that we run the optimization only on four input points. It is a small optimization problem and converges very quickly. It takes around 20% of total solver time for the fastest solver.

From the plots it is visible that this simple optimization procedure helped to improve the accuracy, but fast small solvers did not achieve the accuracy of the best solvers without this optimization. It is also due to the fact, that small solvers sometimes fail to provide good initialization. This is also visible on results from bundle adjustment where we optimized camera pose, rotation and focal length, so that re-projected 3D space ($\mathbf{X_i}$) is as close as possible to 2D measurements ($\mathbf{x_i}$):

$$\epsilon = \sum_i \|\mathbf{x}_i - \frac{\mathtt{P}_{1,2}\mathbf{X}_i}{\mathtt{P}_3\mathbf{X}_i}\|^2, \tag{6.75}$$

where $\mathtt{P}_{1,2}$ and $\mathtt{P}_3$ denote the first two and third rows of matrix $\mathtt{P}$. The left graph in Figure 6.6 shows the result where we minimized the re-projection error of four points which where used to calculate the camera pose. The graph on the right displays the result where we used all 50 points in the optimization process. The quality is improved dramatically with the use of all points. However, there are visible differences for different solvers. There are two reasons for that: (1) a solver either failed to provide a real solution at all or (2) the solution was imprecise and optimization phase diverged.

Improving camera estimation accuracy with bundle adjustment yields more accurate results compared to a simple depth optimization result. On the other hand, it is much slower since bundle adjustment optimizes about twice more parameters compared to the simple approach - three rotation angles of the camera, 3D position vector and the camera focal length.

6.8.3 Synthetic noise tests

Previous experiments focused on numerical stability and accuracy of absolute pose solvers. In this section we study the performance of algorithms in real life scenarios, but still in a controlled environment with growing noise in image measurements. In this experiment we created synthetic scenes and added certain amount of noise to image measurements. Then, an even higher noise in order of 20 to 50 image pixels was added to 50% of image measurements to create wrong data (outliers). Finally we executed RANSAC on all algorithms with the following steps:

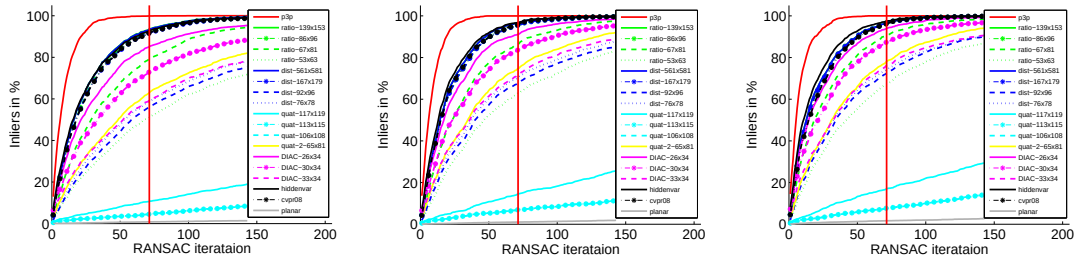


Figure 6.8: Results of 1000 execution of RANSAC algorithm on synthetic data with 50% outliers, 0.5 pixel noise and inlier threshold set to 1 (left), 1.5 (middle) and 2 pixels (right).

- randomly draw four 2D-to-3D correspondences,
- calculate the camera parameters using four correspondences using all solvers,
- calculate inlier sets for each camera, i.e. a set of points with are closer than a given threshold after re-projection using each estimated camera,
- remember size the of the largest inlier set for each solver,
- loop N times.

In other words, each algorithm in the RANSAC loop was executed on exactly the same data and under the same conditions. We are interested in how did the inlier set grow for each particular solver and how does the accuracy of each solver affect this curve. Figure 6.7 shows averaged results of 1000 executions of RANSAC algorithm for 0.5 (left), 1 (middle) and 2 (right) pixel noise level with respect to 512×512 image resolution. Threshold for inlier was set to 1 (left), 1.5 (middle) and 2.5 image pixels (right). For P3P algorithm (Chapter 5) we calibrated data using ground truth focal length.

Vertical red line indicates count of iterations after which there is a 99% probability that we randomly drawn a sample, which is not contaminated by an outlier [37]:

$$k = \frac{log(1-p)}{log(1-w^n)}. \tag{6.76}$$

In the expression above, $p = 0.99$ is the confidence we want (99% confidence level), $w = 0.5$ is a ratio of inliers and outliers (50% contamination). The above equation provides a rough estimate of the number of needed RANSAC iterations k [11] or, better said, its minimum boundary as it assumes that a camera model can be estimated perfectly given any inlier set of size n. In practice, this is not true and more iterations have to be done [11]. In our experiment we did twice more iterations. Figure 6.7 shows how does the precision of an algorithm influence the RANSAC algorithm convergence. Clearly, a more accurate algorithms gets to a much bigger inlier set much faster. Then it is questionable whether using a faster algorithm is a good option since finding and testing inlier set could be even more expensive than actual camera pose calculation.

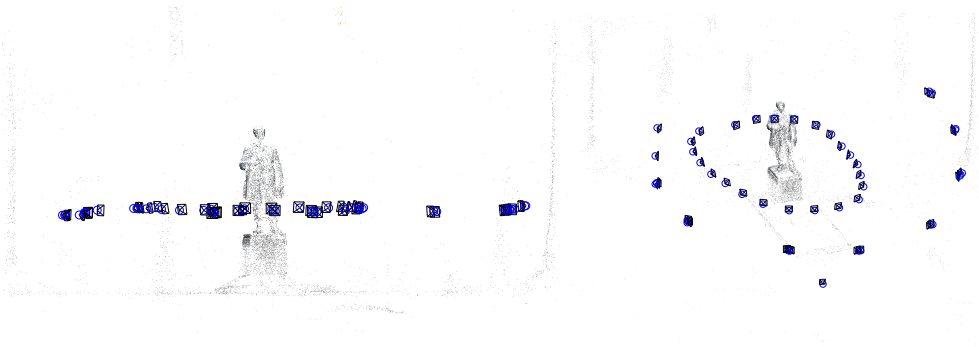


Figure 6.9: Reconstruction of statue from 46 images used as data source for real data experiment.

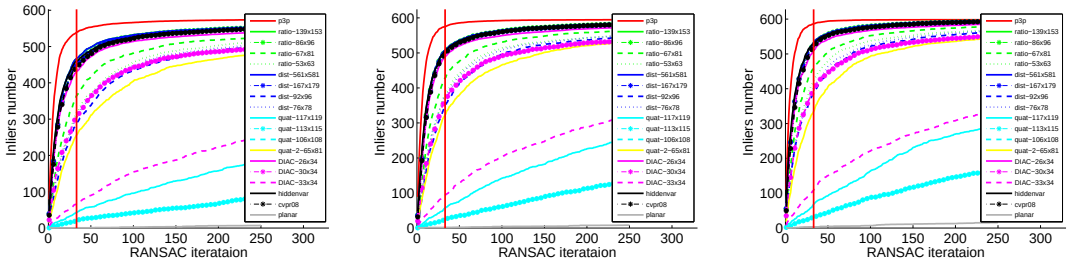


Figure 6.10: Results of 1000 execution of RANSAC algorithm on real data with 40% outliers and with inlier threshold set to 1 (left), 1.5 (middle) and 2 pixels (right).

Figure 6.8 shows results where we executed the RANSAC algorithm on data with 0.5 pixel noise level and with inlier threshold set to 1 (left), 1.5 (middle) and 2 pixels (right). It is visible that relaxing on an inlier threshold did not helped less accurate algorithms.

6.8.4 Real data experiment

The last experiment evaluates proposed algorithms on real data. We captured 46 images of a statue and calculated structure from motion using incremental bundle adjustment approach [13, 47]. Figure 6.9 shows 3D reconstruction of the dataset. We picked a camera with the smallest radial distortion, and took its 2D-3D image correspondences. We artificially increased the amount of outliers to 40% by adding significant noise to some data.

The experiment copies scenarios form the synthetic experiment described in previous section. We executed the same algorithms and replaced synthetic data with the real data. Figure 6.10 shows averaged results of 1000 runs of the RANSAC algorithm.

6.9 Conclusion

In this chapter we extended the previous chapter where we calculated camera rotation and translation and added calculation of an unknown focal length of the camera. We presented several methods how to solve the problem based on the distance, the ratios of distances, by parameterizing the camera rotation using quaternions and we show how to solve this problem for non-planar, planar scenes and near-planar scenes. Since the problem is over-constrained with four 2D-to-3D correspondences, we could create different solvers by choosing different subsets of constrains. By creating and testing several thousands of solvers we have shown that very different results can be obtained depending on the choice of constraints. In experimental section we analyzed the behavior of algorithms in both synthetic and real experiments and compared them to the state-of-the-art. Based on the results we recommend using ratios algorithm with template size 86×96 or combination of planar/non-planar algorithms with algorithm DIAC 26×34 followed by a numerical optimization.

7 An unknown focal length and a radial distortion

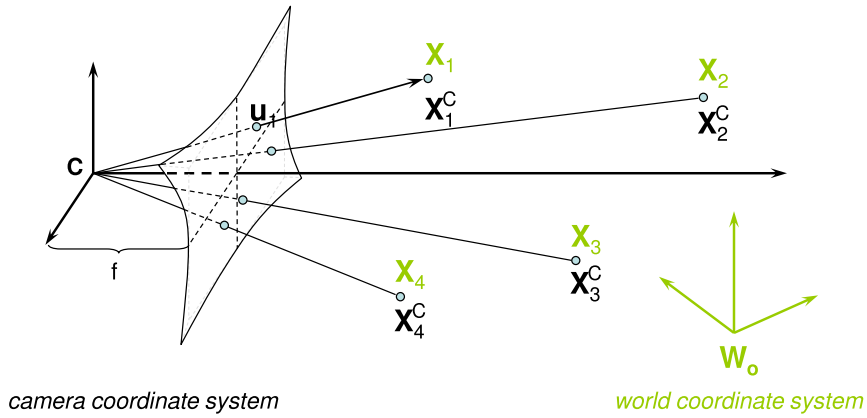


Figure 7.1: Absolute pose problem for a camera with an unknown radial distortion and focal length. Image plane is distorted by the unknown amount of radial lens distortion.

Including the radial distortion estimation into camera pose calculation in an early stage of the reconstruction process allows a broader variety of cameras to be handled compared to when only a focal length is calculated (Chapter 6). Solver estimating both radial distortion and focal length was first presented in [56]. In this chapter we study efficient solutions to this problem, i.e. the absolute pose problem for cameras with unknown focal length and radial distortion from four 2D-to-3D point correspondences, Figure 7.1. We propose to solve this problem separately for non-planar and for planar scenes. By decomposing the problem into these two situations we obtain simpler and more efficient solver than the previously known general solver [56].

In the next section we will formulate the problem, then we will show how to split this problem to planar and non-planar cases, and then how to create a general solver using these two solvers. Such combined solver gives comparable or even better results than the existing general solver for planar as well as non-planar scenes [56]. Finally, we demonstrate both in synthetic and real experiments the numerical stability, accuracy and significant speedup of new solver. Compared to the solver presented in [56], new solvers are about $40\times$ (non-planar) and $160\times$ (planar) faster.

7.1 Problem Formulation

For the sake of clarity, let us first start again with the standard pinhole camera model [47], where image projection $\mathbf{u}_i$ of a 3D reference point $\mathbf{X}_i$ can be written as

$$\lambda_i \, \mathbf{u}_i = \mathtt{P} \, \mathbf{X}_i, \tag{7.1}$$

with a 3×4 projection matrix $\mathtt{P}$, unknown scalars λ_i, image points $\mathbf{u}_i = [u_i, v_i, 1]^\top$ and 3D points $\mathbf{X}_i = [x_i, y_i, z_i, 1]^\top$ represented by their homogeneous coordinates.

The projection matrix $\mathtt{P}$ can be written as

$$\mathtt{P} = \mathtt{K} \, [\mathtt{R} \, | \, \mathbf{t}], \tag{7.2}$$

where $\mathtt{R} = [r_{ij}]_{i,j=1}^{3}$ is a 3×3 rotation matrix, $\mathbf{t} = [t_x, t_y, t_z]^\top$ contains the information about camera position and $\mathtt{K}$ is the calibration matrix of the camera.

In general, calibration matrix can be written as

$$K = \begin{bmatrix} f & s & u \\ 0 & af & v \\ 0 & 0 & 1 \end{bmatrix}, \tag{7.3}$$

where f represents the camera focal length, s is the skew, a the aspect ratio of the pixels and (u, v) are the coordinates of the principal point.

For simplicity assume that the calibration matrix $\mathtt{K}$ contains just an unknown focal length, hence $\mathtt{K} = diag\,[f, f, 1]$. Since the projection matrix is given only up to scale we can equivalently write $\mathtt{K} = diag\,[1, 1, w]$, for $w = 1/f$. This means that input measurements must be corrected to reflect these assumptions. However in experiments, no calibration is applied to the camera. Instead, we assumed square pixels, zero skew and centered principal point. Although these assumptions are not always strictly met, we show that good results are still obtained and by that we conclude that the proposed method is applicable to uncalibrated photographs.

Using these assumptions the projection equation (7.1) can be written as

$$\lambda_i \, \mathbf{u}_i = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ wr_{31} & wr_{32} & wr_{33} & wt_z \end{bmatrix} \mathbf{X}_i. \tag{7.4}$$

Image points are affected by some amount of radial distortion. Here we model the radial distortion by the one-parameter division model proposed by Fitzgibbon [38]. This model is given by formula

$$\mathbf{p}_u \sim \mathbf{p}_d / \left(1 + k r_d^2\right), \tag{7.5}$$

where k is the distortion parameter, $\mathbf{p}_u = [u_u, v_u, 1]^\top$ and $\mathbf{p}_d = [u_d, v_d, 1]^\top$ are the corresponding undistorted and distorted image points, and r_d is the radius of $\mathbf{p}_d$ w.r.t. the distortion center.

We assume that the distortion center is in the center of the image. Therefore $r_d^2 = u_d^2 + v_d^2$ and since scale is not important for homogeneous coordinates, we can express undistorted points as

$$\mathbf{u}_i = \left[u_i, v_i, 1 + k\left(u_i^2 + v_i^2\right)\right]^\top. \tag{7.6}$$

We first eliminate scalar values λ_i from the projection equation (7.4) by multiplying it with the skew symmetric matrix $[\mathbf{u}_i]_\times$. Since $[\mathbf{u_i}]_\times \mathbf{u_i} = 0$ we obtain matrix equation

$$\begin{bmatrix} 0 & -1 - k\,r_i^2 & v_i \\ 1 + k\,r_i^2 & 0 & -u_i \\ -v_i & u_i & 0 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ wr_{31} & wr_{32} & wr_{33} & wt_z \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ z_i \\ 1 \end{bmatrix} = 0, \tag{7.7}$$

for $\mathbf{X}_i = [x_i, y_i, z_i, 1]^\top$.

This matrix equation results in three polynomial equations from which only two are linearly independent. This is caused by the fact that the skew symmetric matrix $[\mathbf{u}_i]_\times$ has rank two.

Now we can formulate the problem of estimating the absolute pose of a camera with an unknown focal length and an unknown radial distortion as follows.

Problem 5. *Assume a camera with radial distortion that can be the modeled using the one parameter division model [38]. Let $\mathbf{X}_i$ be a set of four 3D points and let $\mathbf{u}_i$ be the corresponding projections measured on the distorted camera image plane. Let $\mathtt{K}_{known}$ be a camera calibration matrix known up to an unknown focal length and let k be an unknown scalar expressing the amount of distortion in the division model. The goal is to find the pose of the camera, i.e. its rotation matrix* R*, translation vector* $\mathbf{t}$*, its unknown focal length $f = 1/w$ and unknown parameter k such that Equation (7.7) is satisfied for all four input correspondences.*

In Section 6.7, we were solving a similar problem when image was not distorted by the radial distortion, i.e. $k = 0$. In such a case each point correspondence gave us two linear homogeneous equations in 12 elements of the projection matrix P, see Equation (6.64). We solved the problem by linearization of the projection equation (6.63) followed by four dimensional null space extraction and by calculating unknown α_i in a system of 2^{nd} degree polynomial equations.

Unfortunately, such parameterization cannot be used when image points are affected by radial distortion (7.6). It is because after linearization to $\mathtt{Mp} = 0$ we obtain 20 elements instead of 12. Thus the null space basis of the matrix M would be 12 dimensional too, since each 2D-to-3D correspondence contributes by two equations. The solution to the problem can be simplified by considering non-planar scenes and planar scenes separately. In Section 7.4.5 we will show how to create a general solver based on these two solvers.

7.2 Absolute pose for a camera with unknown focal length and radial distortion for a non-planar scene

Let us denote the elements of the projection matrix P as p_{ij}, where p_{ij} is the element from the i^{th} row and j^{th} column of the matrix P. The equation corresponding to the third row of the skew

symmetric matrix on the left in Equation (7.7) can be then written as

$$-v_i\left(p_{11}\,x_i + p_{12}\,y_i + p_{13}\,z_i + p_{14}\right) + u_i\left(p_{21}\,x_i + p_{22}\,y_i + p_{23}\,z_i + p_{24}\right) = \mathbf{0}. \tag{7.8}$$

This is a homogeneous linear equation in eight unknowns $p_{11}, p_{12}, p_{13}, p_{14}, p_{21}, p_{22}, p_{23}$ and p_{24}. As we have four 2D-to-3D point correspondences, we have four such equations. These four equations can be rewritten in the matrix form

$$\mathtt{M}\,\mathbf{v} = 0, \tag{7.9}$$

where $\mathtt{M}$ is a 4×8 coefficient matrix and $\mathbf{v} = [p_{11}, p_{12}, p_{13}, p_{14}, p_{21}, p_{22}, p_{23}, p_{24}]^\top$ is a 8×1 vector of unknowns. Therefore, we can express our eight unknowns in $\mathbf{v}$ as a linear combination of the four null space basis vectors $\mathbf{n}_i$ of the matrix $\mathtt{M}$ (7.9)

$$\mathbf{v} = \sum_{i=1}^{4} \alpha_i\,\mathbf{n}_i, \tag{7.10}$$

where α_i are new unknowns from which one can be set to one, e.g. $\alpha_4 = 1$.

In this way we obtain a parameterization of the first two rows of the projection matrix $\mathtt{P}$ with three unknowns α_1, α_2 and α_3.

To parameterize the third row of the projection matrix $\mathtt{P}$, we use one from the remaining two equations from the projection equation (7.7). When $u_i = 0$ we use the equation corresponding to the first row of (7.7) and when $v_i = 0$ the equation corresponding to the second row. In all remaining situations, which are most common, we can select the equation corresponding to u_i or v_i with the larger absolute value. Let us assume we are solving the situation when the second row of (7.7) is selected. This equation has the form

$$\left(1 + k\,r_i^2\right)\left(p_{11}\,x_i + p_{12}\,y_i + p_{13}\,z_i + p_{14}\right) - u_i\left(p_{31}\,x_i + p_{32}\,y_i + p_{33}\,z_i + p_{34}\right) = 0. \tag{7.11}$$

Equation (7.11) contains elements p_{31}, p_{32}, p_{33} and p_{34} from the third row of the projection matrix and elements p_{11}, p_{12}, p_{13} and p_{14} from the first row of $\mathtt{P}$ which are already parametrized with α_1, α_2 and α_3. We again have four equations of the form (7.11). Using (7.10) we can rewrite these equations as

$$\mathtt{A}\,[p_{31}, p_{32}, p_{33}, p_{34}]^\top = \mathtt{B}\,[\alpha_1, \alpha_2, \alpha_3, k\,\alpha_1, k\,\alpha_2, k\,\alpha_3, k, 1]^\top, \tag{7.12}$$

where $\mathtt{A}$ and $\mathtt{B}$ are coefficient matrices, $\mathtt{A}$ of size 4×4 and $\mathtt{B}$ of size 4×8.

If the matrix $\mathtt{A}$ has full rank, i.e. points X_1, X_2, X_3 and X_4 are not coplanar, we can write

$$[p_{31}, p_{32}, p_{33}, p_{34}]^\top = \mathtt{A}^{-1}\,\mathtt{B}\,[\alpha_1, \alpha_2, \alpha_3, k\,\alpha_1, k\,\alpha_2, k\,\alpha_3, k, 1]^\top. \tag{7.13}$$

This gives us a parameterization of the third row of the projection matrix $\mathtt{P}$ with four unknowns, $\alpha_1, \alpha_2, \alpha_3$ and k. Together with the parameterization of the first two rows (7.10) we obtain a parameterization of the whole projection matrix $\mathtt{P}$ with these four unknowns $\alpha_1, \alpha_2, \alpha_3$ and k.

With this parameterization of the projection matrix $\mathtt{P}$ in hand, we can now solve the absolute pose problem for the camera with unknown focal length and radial distortion.

To solve this problem we again, like in Section 6.7 of previous chapter, use constraints that the three rows of the 3×3 submatrix of the projection matrix $\mathtt{P}$ are perpendicular and that the first two rows of this submatrix have the same norm. These constraints result from the fact that the 3×3 submatrix of the projection matrix $\mathtt{P}$ has the form $\mathtt{K}\,\mathtt{R}$, where $\mathtt{R}$ is a rotation matrix. In this way we obtain four equations in four unknowns $\alpha_1, \alpha_2, \alpha_3, k$ (two from them quadratic and two cubic)

$$p_{11}p_{21} + p_{12}p_{22} + p_{13}p_{23} = 0, \tag{7.14}$$

$$p_{31}p_{11} + p_{32}p_{12} + p_{33}p_{13} = 0, \tag{7.15}$$

$$p_{31}p_{21} + p_{32}p_{22} + p_{33}p_{23} = 0, \tag{7.16}$$

$$p_{11}^2 + p_{12}^2 + p_{13}^2 - p_{21}^2 - p_{22}^2 - p_{23}^2 = 0. \tag{7.17}$$

To solve these four polynomial equations in four unknowns we use our automatic generator [64]. Using this automatic generator we obtain a solver for our equations consisting of single G-J elimination of a 136×152 matrix and the eigenvalue computation of a 16×16 matrix. This solver gives us 16 solutions for $\alpha_1, \alpha_2, \alpha_3$ and k from which we can create the projection matrix $\mathtt{P}$ using (7.10) and (7.13).

Finally we use the constraint that the squared norm of the first row of the 3×3 submatrix of the projection matrix $\mathtt{P}$ multiplied by w^2 is equal to the squared norm of the third row of this submatrix

$$w^2\, p_{11}^2 + w^2\, p_{12}^2 + w^2\, p_{13}^2 - p_{31}^2 - p_{32}^2 - p_{33}^2 = 0, \tag{7.18}$$

This is a quadratic equation in $w = 1/f$ from which the positive root give us a solution for the focal length f.

7.3 Absolute pose for a camera with unknown focal length and radial distortion for planar scene

In the planar case, i.e. when all four 3D points are on a plane, we can't directly use the parameterization presented in the Section 7.2 since it leads to a degenerate system. However, we can use a similar parameterization.

Without loss of generality let us assume that all four 3D points X_i have the third coordinate $z_i = 0$. In this case Equation (7.8) corresponding to the third row of the matrix equation (7.7) can be written as

$$-v_i\,(p_{11}\,x_i + p_{12}\,y_i + p_{14}) + u_i\,(p_{21}\,x_i + p_{22}\,y_i + p_{24}) = 0. \tag{7.19}$$

This is a homogeneous linear equation in only six unknowns $p_{11}, p_{12}, p_{14}, p_{21}, p_{22}$ and p_{24}. Since we have four 2D-to-3D point correspondences we have four such equations which can be again rewritten in the matrix form $\mathtt{M}\,\mathbf{v} = 0$, where $\mathtt{M}$ is a 4×6 coefficient matrix and $\mathbf{v} = [p_{11}, p_{12}, p_{14}, p_{21}, p_{22}, p_{24}]^\top$ is a 6×1 vector of unknowns. Therefore, in this case we can write

our unknowns in $\mathbf{v}$ as a linear combination of the two null space basis vectors $\mathbf{n}_1$ and $\mathbf{n}_2$ of the matrix $\mathtt{M}$

$$\mathbf{v} = \beta_1 \, \mathbf{n}_1 + \mathbf{n}_2, \tag{7.20}$$

where β_1 is a new unknown. Using (7.20) we obtain a parameterization of the first two rows of the matrix $\mathtt{P}$ (without the third column) with one unknown β_1.

To parametrize the third row we use one from the remaining two equations from the projection equation (7.7). Let's consider the equation corresponding to the second row of the projection equation (7.7). In this planar case this equation has the form

$$\left(1 + k\, r_i^2\right) \left(p_{11}\, x_i + p_{12}\, y_i + p_{14}\right) - u_i \left(p_{31}\, x_i + p_{32}\, y_i + p_{34}\right) = 0. \tag{7.21}$$

This equation contains elements from the first row of $\mathtt{P}$, which are already parameterized with β_1, and three elements p_{31}, p_{32} and p_{33} from the third row, which we want to parametrize. We again have four equations of the form (7.21). However, we will now use only three of them, e.g. equations corresponding to the first three 2D-to-3D point correspondences. Using (7.20) we can rewrite these three equations as

$$\mathtt{C} \left[p_{31}, p_{32}, p_{34}\right]^\top = \mathtt{D} \left[\beta_1, k\, \beta_1, k, 1\right]^\top, \tag{7.22}$$

where $\mathtt{C}$ and $\mathtt{D}$ are coefficient matrices, $\mathtt{C}$ of size 3×3 and $\mathtt{B}$ of size 3×4.

If the matrix $\mathtt{C}$ has full rank, i.e. points X_1, X_2 and X_3 are not collinear, we can write

$$\left[p_{31}, p_{32}, p_{34}\right]^\top = \mathtt{C}^{-1}\, \mathtt{D} \left[\beta_1, k\, \beta_1, k, 1\right]^\top. \tag{7.23}$$

In this way we obtain a parameterization of the third row (without the third column) of the projection matrix $\mathtt{P}$ with two unknowns, β_1 and k. Together with (7.20) we have parameterized the first, second and fourth column of the projection matrix $\mathtt{P}$ with β_1 and k.

In this case we can not use the constraints (7.14)-(7.17) on the rows of the projection matrix. It is because we do not have information about the third column of $\mathtt{P}$.

However, we can use the fact that columns of the rotation matrix are perpendicular and of the same norm. This gives us two equations of degree four in three unknowns β_1, k and $w = 1/f$

$$w\, p_{11}\, w\, p_{12} + w\, p_{21}\, w\, p_{22} + p_{31}\, p_{32} = 0, \tag{7.24}$$

$$w^2\, p_{11}^2 + w^2\, p_{21}^2 + p_{31}^2 - w^2\, p_{12}^2 - w^2\, p_{22}^2 - p_{32}^2 = 0. \tag{7.25}$$

Moreover, we have one more equation of the form (7.21), for the fourth 2D-3D point correspondence, which was not used in (7.22). This equation has the form

$$\left(1 + k\, r_4^2\right) \left(p_{11}\, x_4 + p_{12}\, y_4 + p_{14}\right) - u_4 \left(p_{31}\, x_4 + p_{32}\, y_4 + p_{34}\right) = 0, \tag{7.26}$$

and after using parameterization (7.20) and (7.23) of the unknowns $p_{11}, p_{12}, p_{14}, p_{31}, p_{32}$ and p_{34} it results in one quadratic equation in two unknowns β_1 and k.

Equation (7.26) together with equations (7.24) and (7.25) give us three equations in three unknowns β_1, k and $w = 1/f$ which we solve using the automatic generator [64]. In this case the resulting solver results in one G-J elimination of relatively small 12×18 matrix and gives up to 6 real solutions to β_1, k and f.

Finally, the third column of the projection matrix $\mathtt{P}$ can be obtained from the constraint that the columns of the rotation matrix are perpendicular and of the same norm.

7.4 Experiments

In this section we compare both the planar and the non-planar solvers described in above Sections 7.2 and 7.3 with the general solution to $P4P + f + k$ problem proposed in [56]. We compare algorithms on synthetically generated scenes and we show that all solvers return comparable results, while the new solvers are much faster.

We also study the performance of the solvers in near-planar scenes to show that the planar and the non-planar solvers can be easily combined into a general solver, which gives comparable or better results then the existing general solver [56] for all scenes, including near-planar ones.

Finally we show the performance of new combined general solver on real datasets and compare it with the general solver from [56].

7.4.1 Synthetic datasets

In the following synthetic experiments we use synthetically generated ground-truth 3D scenes. These scenes were generated using 3D points randomly distributed on a plane or in a 3D cube depending on the testing configuration. Each 3D point was projected by a camera with random feasible orientation and position and random or fixed focal length. Then we distorted image points using the division model (7.5) to generate noiseless distorted points. Finally, Gaussian noise with standard deviation σ was added to the distorted image points assuming a 1000×1000 pixel image.

7.4.2 Numerical stability

In the first experiment we study the behavior of all presented solvers on noise free data to evaluate their numerical stability.

In this experiment 1500 random scenes and feasible camera poses were generated. The radial distortion parameter was randomly drawn from the interval $k \in [-0.45, 0]$ and the focal length from the interval $f \in [0.5, 2.5]$ with respect to the unit size image.

Figure 7.2 shows results of the non-planar solver on non-planar scenes (Top) and of the planar solver on planar ones (Bottom). In both cases we compare our solvers (Red) with the general solvers from [56] (Blue). Plots show the log_{10} relative error of the focal length f obtained by selecting the real root closest to the ground truth value on the left and the log_{10} absolute error of the radial distortion parameter on the right.

From the figure it results that both the planar and the non-planar solvers are numerically more stable that the general algorithm from [56]. Also the number of failures or more precisely results with error greater than 10^{-5} is smaller compared to the general solver. While the general solver [56] "failed" in about 4.5%, combined solvers failed in less than 1% cases.

This difference in the number of "failures" will also be visible in the near-planar and real experiments.

Note that the general solution from [56] uses techniques for improving numerical stability of Gröbner basis solvers based on changing basis and QR decomposition [17] while we did not

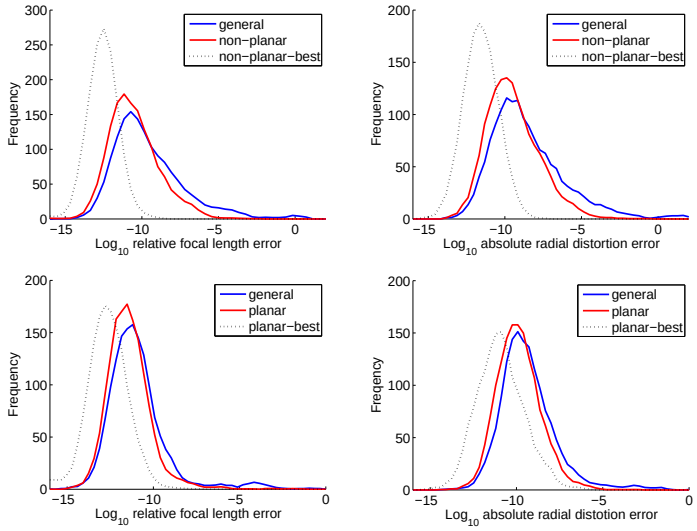


Figure 7.2: Log_{10} relative error of the focal length f and log_{10} absolute error of the radial distortion parameter k obtained by selecting the real root closest to the ground truth value for the non-planar solver (Top) and the planar solver (Bottom).

use such techniques. We therefore believe that such techniques can further improve numerical stability of our solvers as well. Numerical stability can be further improved by permuting input correspondences. This is partially visible in Figure 7.2 where the results denoted as non-planar-best and planar-best (Dashed black) correspond to the most accurate results obtained by permuting input points. This permutation of input data is in some sense similar to the permutation of columns of a coefficient matrix in the Gröbner basis solver and therefore it is also similar to the changing of basis used in [56]. However, these techniques for improving numerical stability [56, 17] are somewhat expensive and, as it will be shown in real experiments, they are not necessary.

7.4.3 Experiment with synthetic noise

In the next experiment we have tested behavior of all solvers in the presence of noise added to image points. We again compare our specialized solvers with the general solver [56].

For each noise level, from 0.0 to 2 pixels, 2000 estimates for random scenes and camera positions, focal length $f_{gt} = 1.5$ and radial distortion $k_{gt} = -0.2$, were made.

Results in Figure 7.3 are represented by the $Matlab$ boxplot function which shows values 25% to 75% quantile as a box with horizontal line at median. The crosses show data beyond 1.5 times the interquartile range. In this case the rotation error (Top left) was measured as the rotation angle in the angle-axis representation of the relative rotation $\mathtt{RR}_{gt}^{-1}$ and the translation error (Top middle) as the angle between ground-truth and estimated translation vector.

Our specialized radial distortion (P4P+f+k) solvers solve the same algebraical problem and have similar numerical stability as the general solver from [56]. Thus it is expected that these

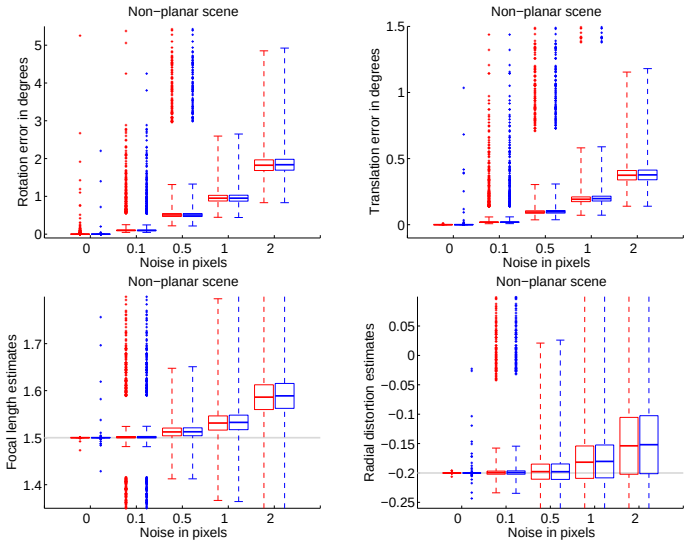


Figure 7.3: Error of rotation (Top left), translation (Top right), focal length estimates (Bottom left) and radial distortion estimates (Bottom right) in the presence of noise for the P4P+f+k non-planar solver (Red) and the general solver from [56] (Blue).

solvers will behave similarly also in the presence of noise. This is visible also from Figure 7.3 which shows results for specialized non-planar solver (Red) and the general solver from [56] (Blue). Similar results were obtained also for the planar solver and therefore we are omitting them.

7.4.4 Computational complexity

Splitting studied problem into two cases - planar and non-planar - results in much simpler systems of polynomial equations and therefore also in much simpler and more practical solvers. While the general solver [56] requires to perform LU decomposition of a 1134×720 matrix, QR decomposition of a 56×56 matrix and eigenvalue computations of a 24×24 matrix, non-planar solver requires only one G-J elimination of a 136×152 matrix and eigenvalue computations of a 16×16 matrix. The planar solver is even simpler and requires one G-J elimination of only 12×18 matrix. Moreover, specialized solvers return fewer solutions, 16 and 6 compared to 24 in [56]. Translating this in numbers, specialized solvers are about $40\times$ faster (non-planar) and $160\times$ faster (planar) than the general solver [56].

These facts are important in RANSAC and real applications in which the general solver from [56] was impractical due to the lack of speed.

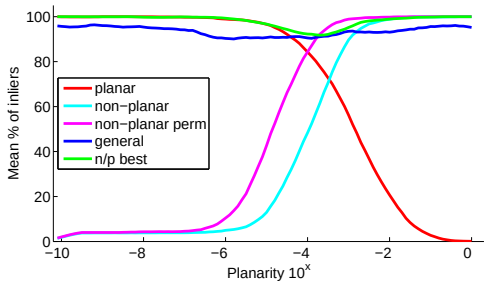


Figure 7.4: Results of experiment on the near-planar scene for focal and radial problem (P4P+f+k).

7.4.5 General Solver

In Section 6.7.2 we were building a general solver by combining two specific - the planar and the non-planar solvers for the problem with unknown focal length. In this section we will use the same methods to build the general solver for the radial distortion problem.

We will focus primarily on near-planar scenes as such scenes are on the border between planar and non-planar cases, i.e. a general solver has to make the decision whether to use the planar or the non-planar solver. For this purpose we will use the generator of near-planar scenes with controlled scene planarity by a scalar value a as in Section 6.7.2. For each given planarity value a, we created a scene, calculated the camera pose from the four-tuple of correspondences using the planar, the non-planar and the general solver [56]. Recall that this four-tuple is not affected by noise and hence the only deviation from the ground truth solutions comes from the numerical instability of the solvers itself. To evaluate the impact of the instability on the solution we used the estimated camera, the focal length and the radial distortion to project all 3D points to the image plane. Then we measured how many points were projected closer than one pixel to its corresponding 2D image - we call them inliers.

Figure 7.4 shows results for the planar (Red), the non-planar (Cyan) and the general (Blue) solver together with the results obtained from the non-planar solver obtained by permuting the input points (Magenta). The green curve shows results for the solver which executes both planar and non-planar solvers and select the better solution. Figure shows mean results obtained by evaluating 100 random scenes per each given planarity value a.

The most important parts in this Figure 7.4 are the intersections of the planar and the general solver and also the general and the non-planar solver. Ideally we would like to combine planar, general and non-planar solvers to gain maximal precision at maximal speed. It appears it is better to use the planar solver when the planarity value is smaller than $10^{-4.2}$ and the non-planar solver when the planarity is greater than $10^{-2.8}$. The general solver from [56] could be used in the interval $a \in \langle 10^{-4.2}, 10^{-2.8} \rangle$. However we found that it is faster and also results are better when both non-planar and planar algorithms are executed simultaneously in this interval and the better result is selected. This is visible in Figure 7.4 (Green) versus the general solver [56] (Blue).

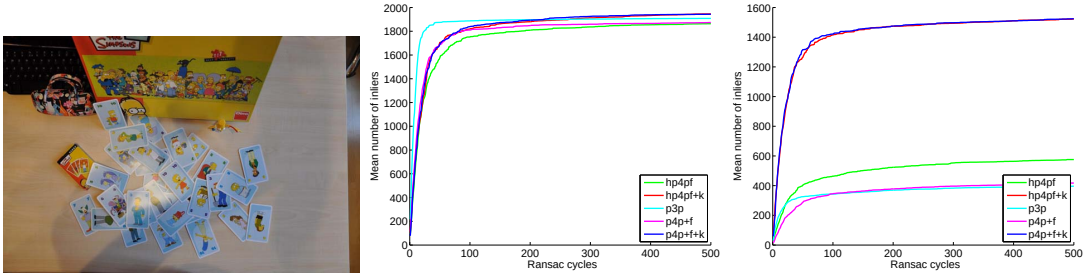


Figure 7.5: Example of an input image (left), RANSAC sampling history for the image without (middle) and with (right) strong radial distortion.

7.4.6 Real data experiment

This section demonstrates the applicability of algorithms studied in this chapter in a real setup. We created a simple scene with two dominant planar structures, Figure 7.5 (Left). Our intention was to show the behavior of combined planar/non-planar solver described in previous section in a real scene with sampling on the plane, near the plane and off the plane points.

First, we captured around 20 photos with a cellular phone and a digital camera to get images with different distortions. We used reconstruction pipeline like in [93] to create the 3D reconstruction, to get 2D-3D correspondences and to get the ground truth reference for the camera poses. Note that 2D correspondences are positions of detected feature points and not ideal projections of 3D points and hence they are noisy. We randomly modify 50% of 2D measurements to get 50% of outliers.

In this test we compare, the calibrated P3P solver (Cyan), the general P4P+f solver for the camera with unknown focal length [12] (Magenta), the general P4P+f+k solver from [56] (Blue), the combined P4P+f general solver from Section 6.7.2 (Green) and our new combined P4P+f+k general solver (Red). Locally optimized RANSAC estimator [23] was combined with all the examined solvers to calculate the camera pose of each camera using given 2D-3D correspondences.

Note that we did not calibrate radial distortion prior to calling P3P and P4P+f solvers. Since many images from the cell phone had strong radial distortion we cannot expect good results without correcting the distortion. The results show how are the radial distortion solvers useful in real applications. Figure 7.5 (Right) shows the difference in RANSAC convergence when using solvers with and without the radial distortion estimation and Figure 7.5 (Middle) results for not so distorted image.

We included the solver combining planar and non-planar solvers for the problem with unknown focal length (Section 6. 7.2) into this experiment to show how does this solver cope with radial distortion. It is visible that the combined solver outperforms the general P4P+f solvers (Chapter 6). This is because the combined solver does not produce strictly orthogonal rotation matrix and the skew and/or anisotropic scale in the matrix allows fitting to more image data. Therefore we need to make rotation part orthogonal to get a proper rotation matrix after all in-

liers are found. Alternatively, RANSAC can be used to find a good set of inliers and a different solver for fitting to all inliers, e.g. bundle adjustment initialized with the solver result.

7.5 Conclusion

To conclude, this chapter described efficient solvers to the absolute pose problem for the camera with unknown focal length and radial distortion from four 2D-to-3D point correspondences. Efficiency is achieved by decomposing the problem to non-planar scenes and to planar scenes and solving each sub problem separately using specialized solvers. We have shown when to use which solver and how to create a generalized solver by joining these specialized solvers. Experimental results demonstrate that the combined general solver gives comparable or better results at fraction of time compared to the existing general solver [56]. Moreover, even executing both the planar and the non-planar solver simultaneously and selecting the better result is faster and numerically more stable compared to the general solver [56] for all scenes including near-planar scenes.

8 Known vertical direction

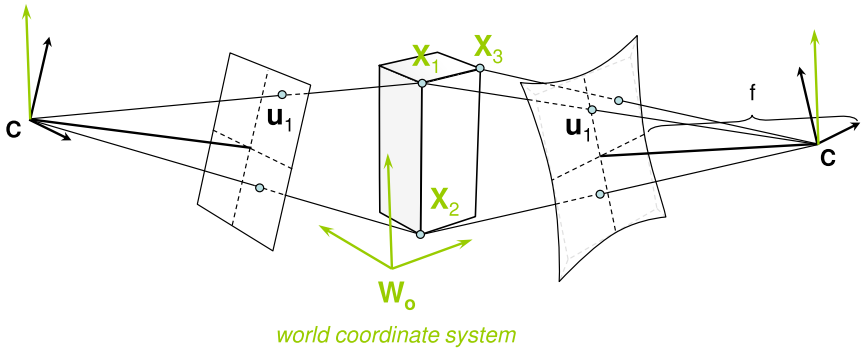


Figure 8.1: The camera pose estimated and pose of camera with unknown focal length and radial distortion from two, respectively three 2D-to-3D point correspondences and known vertical direction.

In this chapter we study camera pose estimation when vertical direction of a camera is known, see Figure 8.1.

Knowing the vertical direction is equivalent to knowing camera rotation around two axes what greatly simplifies the problem. This is important since more and more cameras become equipped with inertial measurement units (IMUs) like gyroscopes and accelerometers, compasses, or GPS devices. Information from these devices is often stored in the image header too. This could be even more observable in modern cellular phones and other smart devices.

GPS position accuracy is in order of meters and hence it is not sufficient to provide external camera calibration, i.e. the absolute position of the camera. Neither compass accuracy (which could provide cameras yaw) is good enough to provide camera orientation since its sensor is highly influenced by magnetic field disturbances.

On the other hand IMUs currently available (even the low cost ones) provide quite accurate roll and pitch angle, i.e. the vertical direction. The angular accuracy of roll and pitch angle in low cost IMUs like [103] is about $0.5°$, and in high accuracy IMUs is less than $0.02°$. While most of today cameras use information from gyroscopes and IMUs only to distinguish whether the image orientation is landscape or portrait, we show that knowing the camera "up vector" can radically simplify the camera absolute pose problem.

In the following sections we describe new simple closed-form solutions to two minimal absolute pose problems for cameras with known rotation around two axes, i.e. known vertical direction. In the first problem we estimate the absolute pose of a calibrated camera from two

2D-to-3D correspondences and a given vertical direction. In the second problem we assume camera with unknown focal length and radial distortion and we estimate its pose together with the focal length and the radial distortion from three 2D-to-3D correspondence and a given vertical direction. In both cases this is the minimal number of correspondences needed to solve these problems.

Both these problems result in solving a single polynomial equation of degree two in one variable and one, respectively two, systems of linear equations and can be efficiently solved in a closed-form way. By evaluating algorithms on synthetic and real data we show that both solutions are fast, efficient and numerically stable. Moreover, both algorithms are very useful in real applications like structure from motion, surveillance camera calibration or applications of registering photographs taken by a camera mounted on a car moving on a plane.

Our work builds on recent results from [57] where the efficient algorithm for relative pose estimation of a camera with known vertical direction from three point correspondences was presented. It was demonstrated that in the presence of good vertical direction information this algorithm is more accurate than the classical 5-point relative pose algorithm [84, 96] for calibrated cameras.

Next we provide the formulation of the two problems studied and further we show how they can be solved.

8.1 Problem formulation

In this section we formulate two problems of determining absolute pose of a camera given 2D-to-3D correspondences and the vertical direction, respectively rotation angles around two axes.

Problem 6. *Given the rotation of the calibrated camera around two axes and the images of two known 3D reference points, estimate the absolute position of the camera and the rotation of the camera around the third axis (y-axis).*

Problem 7. *Given the rotation of the camera with unknown focal length around two axes and the distorted images of three known 3D reference points, estimate the absolute position of the camera, the rotation of the camera around the third axis (y-axis), the unknown focal length and the parameter of radial distortion.*

In both problems we use the standard pinhole camera model [47] where the image projection $\mathbf{u}_i$ of a 3D reference point $\mathbf{X}_i$ is

$$\lambda_i \, \mathbf{u}_i = \mathtt{P} \, \mathbf{X}_i, \tag{8.1}$$

and $\mathtt{P}$ is a 3×4 projection matrix, λ_i is an unknown scalar value and points $\mathbf{u}_i = [u_i, v_i, 1]^\top$ and $\mathbf{X}_i = [X_i, Y_i, Z_i, 1]^\top$ are represented by their homogeneous coordinates.

The projection matrix $\mathtt{P}$ is in the form

$$\mathtt{P} = \mathtt{K} \, [\mathtt{R} \, | \, \mathbf{t}], \tag{8.2}$$

where $\mathtt{R}$ is a 3×3 rotation matrix, $\mathbf{t} = [t_x, t_y, t_z]^\top$ contains the information about camera position and $\mathtt{K}$ is the 3×3 calibration matrix of the camera.

We assume that we know the vertical direction, i.e. the coordinates of the world vector $[0, 1, 0]^\top$ in the camera coordinate system. This vector can be obtained from the vanishing point or read directly from the IMU after a very simple calibration of the IMU w.r.t. the world coordinate system. It is important that all what has to be known to calibrate the IMU w.r.t. the world coordinate system, is the direction of the "up-vector", i.e. the direction which is perpendicular to the "ground plane", i.e. the plane $Z = 0$ in the world coordinate system. This direction is easy to calibrate by reseting the IMU when lied down on the "ground plane". Note that this "ground plane" does not need to be horizontal.

Moreover, IMU's are often precalibrated to use the gravity vector as the up-vector. Then, the IMU is calibrated when the plane $Z = 0$ of the world coordinate system corresponds to the ground plane, i.e. if the coordinates $Z = 0$ are assigned to the points on the ground plane.

From the "up-vector" returned by an IMU, we can compute the rotation of the camera around two axes, in this case the x-axis (ϕ_x) and the z-axis (ϕ_z). Note that some IMUs return directly these two angles. We will denote this rotation as

$$\mathtt{R}_v = \begin{bmatrix} \cos\phi_z & -\sin\phi_z & 0 \\ \sin\phi_z & \cos\phi_z & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi_x & -\sin\phi_x \\ 0 & \sin\phi_x & \cos\phi_x \end{bmatrix}. \tag{8.3}$$

Therefore, the only unknown parameter in the camera rotation matrix $\mathtt{R}$ is the rotation angle ϕ_y around the y-axis, i.e. the vertical axis and we can write

$$\mathtt{R} = \mathtt{R}\left(\phi_y\right) = \mathtt{R_v}\,\mathtt{R_y}\left(\phi_y\right), \tag{8.4}$$

where $\mathtt{R_v}$ is the known rotation matrix (8.3) around the vertical axis and $\mathtt{R_y}\left(\phi_y\right)$ is the unknown rotation matrix around the y-axis of the form

$$\mathtt{R}_y = \begin{bmatrix} \cos\phi_y & 0 & -\sin\phi_y \\ 0 & 1 & 0 \\ \sin\phi_y & 0 & \cos\phi_y \end{bmatrix}. \tag{8.5}$$

With this parameterization, the projection equation (8.1) becomes

$$\lambda_i \mathbf{u}_i = \mathtt{K}\left[\mathtt{R}\left(\phi_y\right) \mid \mathbf{t}\right]\mathbf{X}_i = \mathtt{K}\left[\mathtt{R_v}\,\mathtt{R_y}\left(\phi_y\right) \mid \mathbf{t}\right]\mathbf{X}_i. \tag{8.6}$$

To "simplify" this new projection equation (8.6) we eliminate the scalar value λ_i and trigonometric functions $\sin$ and $\cos$.

To eliminate $\sin$ and $\cos$ we use the substitution $q = \tan\frac{\phi_y}{2}$ for which it holds that $\cos\phi_y = \frac{1-q^2}{1+q^2}$ and $\sin\phi_y = \frac{2q}{1+q^2}$. Therefore we can write

$$\left(1+q^2\right)\mathtt{R}_y\left(q\right) = \begin{bmatrix} 1-q^2 & 0 & -2q \\ 0 & 1+q^2 & 0 \\ 2q & 0 & 1-q^2 \end{bmatrix}. \tag{8.7}$$

We eliminate the scalar value λ_i by multiplying Equation (8.6) with the skew symmetric matrix $[\mathbf{u}_i]_\times$. Since $[\mathbf{u}_i]_\times \mathbf{u}_i = \mathbf{0}$ we obtain the matrix equation

$$[\mathbf{u}_i]_\times \, \mathtt{K} \, [\mathtt{R}_\mathtt{v} \, \mathtt{R}_\mathtt{y}(q) \mid \mathbf{t}] \mathbf{X}_i = \mathbf{0}. \tag{8.8}$$

Note that scalar value λ_i absorbed $\left(1+q^2\right)$ on the left hand side of Equation (8.7) and it vanishes together with λ_i. This matrix equation results in three polynomial equations from which only two are linearly independent.

Polynomial equations (8.8) are the basic equations which we use for solving our two problems. Next we describe how these equations look for our two problems and how they can be solved.

8.2 Absolute pose of a calibrated camera with known up direction

In the case of Problem 6 for calibrated camera the calibration matrix $\mathtt{K}$ is known and therefore the projection equation (8.8) $[\mathbf{u}_i]_\times \, \mathtt{K} \, [\mathtt{R}_\mathtt{v} \, \mathtt{R}_\mathtt{y}(q) \mid \mathbf{t}] \mathbf{X}_i = 0$ has the form

$$\begin{bmatrix} 0 & -1 & v_i \\ 1 & 0 & -u_i \\ -v_i & u_i & 0 \end{bmatrix} \begin{bmatrix} r_{11}(q) & r_{12}(q) & r_{13}(q) & t_x \\ r_{21}(q) & r_{22}(q) & r_{23}(q) & t_y \\ r_{31}(q) & r_{32}(q) & r_{33}(q) & t_z \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix} = \mathbf{0}, \tag{8.9}$$

where $\mathbf{u}_i = [u_i, v_i, 1]^\top$ is a calibrated image point and $r_{ij}(q)$ are the elements of the rotation matrix $\mathtt{R} = \mathtt{R}_\mathtt{v} \, \mathtt{R}_\mathtt{y}(q)$. Note that these elements are quadratic polynomials in q.

In this case we have four unknowns $t_x, t_y, t_z, q = \tan \frac{\phi_y}{2}$ and since each 2D-to-3D point correspondence gives us two constraints, i.e. two independent polynomial equations (8.9), the minimal number of point correspondences needed to solve this problem is two.

Let's see this in detail. Let $\mathbf{u}_i = [u_i, v_i, 1]$, be the image projection of the i^{th} known 3D reference point $\mathbf{X}_i = [X_i, Y_i, Z_i, 1]$. Without loss of generality assume that $\mathtt{R}_\mathtt{v}$ is the identity, i.e. image measurements are transformed with ${\mathtt{R}_\mathtt{v}}^\top$. Then, each 2D-to-3D correspondence plugged into Equation (8.9) gives us polynomial equations

$$2v_iX_iq - Y_i(1+q^2) + v_iZ_i(1-q^2) - t_y + v_it_z = 0, \tag{8.10}$$

$$X_i(1-q^2) - 2u_iX_iq - 2Z_iq - u_iZ_i(1-q^2) + t_x - u_it_z = 0, \tag{8.11}$$

$$-v_iX_i(1-q^2) + u_iY_i(1+q^2) + 2v_iZ_iq - v_it_x + u_it_y = 0. \tag{8.12}$$

These equations contain monomials $q^2, q, t_x, t_y, t_z, 1$. Recall that equations are linearly dependent because we multiplied the projection equation (8.6) with a rank two skew symmetric matrix. We will take the first two equations, since the third equation vanishes whenever the 2D measurement $\mathbf{u}_i$ is in the image center, i.e. $\mathbf{u}_i = [0, 0, 1]$. Given two 2D-to-3D point correspondences we get four equations in 5 unknown monomials q^2, q, t_x, t_y, t_z. We can use three of them to eliminate t_x, t_y and t_z from the fourth one.

In this way we obtain a single polynomial in one variable q of degree two. Such equation can be easily solved in a closed-form and generally results in two solutions for $q = \tan\frac{\phi_y}{2}$. Back-substituting these two solutions to the remaining three equations gives us three linear equations in three variables t_x, t_y and t_z. By solving these linear equations we obtain a solution to our problem.

8.3 Absolute pose of a camera with unknown focal length and radial distortion and known camera up direction

In Problem 7 we assume that the skew in the calibration matrix $\mathtt{K}$ is zero, the aspect ratio is one and the principal point is in the center of the image. Modern cameras are close to satisfying these assumptions and, as it will be shown in experiments, we are getting good results even when they are not exactly satisfied.

The only parameter from $\mathtt{K}$ which cannot be safely set to a known value is the focal length f. Here we assume that the focal length f is unknown and therefore the calibration matrix $\mathtt{K}$ has the form $diag\left[f, f, 1\right]$. Since the projection matrix is given only up to a scale we can equivalently write

$$K = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & w \end{bmatrix}, \tag{8.13}$$

for $w = 1/f$.

We further assume that the image points are affected by some amount of radial distortion. To model the distortion we use the one-parameter division model [38]

$$\mathbf{p}_u \sim \mathbf{p}_d / \left(1 + kr_d^2\right), \tag{8.14}$$

where k is the distortion parameter, $\mathbf{p}_u = \left[u_u, v_u, 1\right]^\top$, respectively $\mathbf{p}_d = \left[u_d, v_d, 1\right]^\top$, are the corresponding undistorted, respectively distorted, image points, and r_d is the radius of $\mathbf{p}_d$ w.r.t. the distortion center. We assume that the distortion center is in the center of the image and therefore $r_d^2 = u_d^2 + v_d^2$.

Since the projection equation (8.8) contains undistorted image points $\mathbf{u}$ but we measure distorted ones we need to include undistortion relation (8.14) into Equation (8.8).

Let $\left[u_i, v_i, 1\right]$ be the i^{th} measured distorted image point, projection of the 3D point $\mathbf{X}_i = \left[X_i, Y_i, Z_i, 1\right]$, and let $r_i^2 = u_i^2 + v_i^2$. Then $\mathbf{u}_i = \left[u_i, v_i, 1 + kr_i^2\right]^\top$ and the projection equation (8.8) for the i^{th} point has the form

$$\begin{bmatrix} 0 & -1 - kr_i^2 & v_i \\ 1 + kr_i^2 & 0 & -u_i \\ -v_i & u_i & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & w \end{bmatrix} \begin{bmatrix} r_{11}\left(q\right) & r_{12}\left(q\right) & r_{13}\left(q\right) & t_x \\ r_{21}\left(q\right) & r_{22}\left(q\right) & r_{23}\left(q\right) & t_y \\ r_{31}\left(q\right) & r_{32}\left(q\right) & r_{33}\left(q\right) & t_z \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix} = 0 \tag{8.15}$$

where again $r_{ij}\left(q\right)$ are the elements of the rotation matrix $\mathtt{R} = \mathtt{R_v}\,\mathtt{R_y}\left(q\right)$.

In this case we have six unknowns $t_x, t_y, t_z, q = \tan\frac{\phi_y}{2}, k, w = 1/f$ and since each 2D-to-3D point correspondence gives us two constraints the minimal number of point correspondences needed to solve this problem is three.

For three point correspondences the matrix equation (8.15) gives us nine polynomial equations from which only six are linearly independent. Let denote these nine polynomial equations $g_{1i} = 0, g_{2i} = 0$ and $g_{3i} = 0$ for $i = 1, 2, 3$, where $g_{1i} = 0$ is the equation corresponding to the first row in the matrix equation (8.15), $g_{2i} = 0$ to the second and $g_{3i} = 0$ to the third row.

Now consider the third polynomial equation from the matrix equation (8.15), i.e. the equation

$$g_{3i} = c_1\, q^2 + c_2\, q + c_3\, t_x + c_4\, t_y + c_5 = 0, \tag{8.16}$$

where $c_j,\ j = 1, \ldots, 5$ are known coefficients.

The polynomial equation (8.16) doesn't contain variables t_z, k and w. Since we have three 2D-to-3D correspondences we have three equations $g_{3i} = 0,\ i = 1, 2, 3$. Therefore, we can use two from these equations to eliminate t_x and t_y from the third one, e.g. from the $g_{31} = 0$.

In this way we obtain one polynomial equation in one variable q of degree two. Such equation can be easily solved in a closed-form and generally results in two solutions for $q = \tan\frac{\phi_y}{2}$. Backsubstituting these two solutions to the remaining two equations corresponding to the third row of (8.15), i.e. $g_{32} = 0$ and $g_{33} = 0$, gives us two linear equations in two variables t_x and t_y which can be again easily solved.

After obtaining solutions to q, t_x and t_y we can substitute them to the equations corresponding to the first row of (8.15) to get

$$g_{1i} = c_6\, t_z\, w + c_7\, w + c_8\, k + c_9 = 0, \tag{8.17}$$

for $i = i, \ldots, 3$ and known coefficients $c_j,\ j = 6, \ldots, 9$. Equations (8.17) can be solved in a linear way. This is because we can consider the monomial $t_z\, w$ as a new variable and solve a linear system of three equations in three variables. Solving this linear system together with solutions to q, t_x and t_y gives us in general two solutions to this problem of estimating absolute pose of camera with unknown focal length and radial distortion given camera rotation around two axes.

8.4 Experiments

We have tested both algorithms studied in this chapter on synthetic data (with various levels of noise, outliers, focal lengths and radial distortions and different angular deviation of the vertical direction) and on real datasets and compared them with algorithms studied in previous chapters, i.e. P3P algorithm for calibrated camera, the P4P+f algorithm for camera with unknown focal length and with the P4P+f+k algorithm for camera with unknown focal length and radial distortion. From the set of possible solvers for the P4P+f problem for camera with unknown focal length, we selected the baseline solver presented in [12]. For the problem with unknown radial distortion and focal length we compared results with Josephson et al. algorithm [56]. In all experiments we denote algorithms studied in this chapter as uP2P and uP3P+f+k.

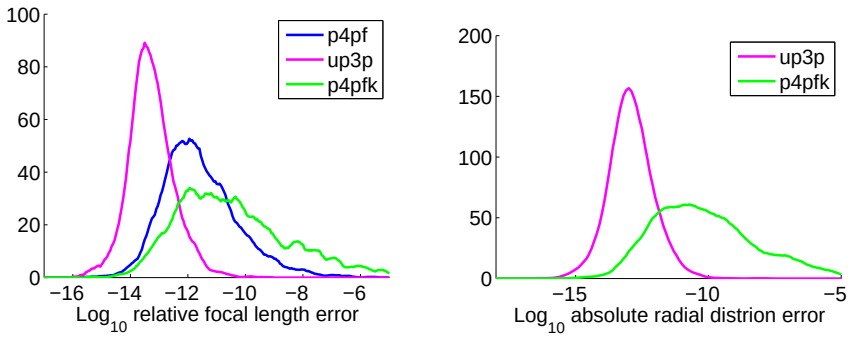


Figure 8.2: Log_{10} relative error of the focal length f (Left) and Log_{10} absolute error of the radial distortion parameter k (Right) obtained by selecting the real root closest to the ground truth value $f_{gt} = 1.5$ and $k_{gt} = -0.2$.

8.4.1 Synthetic data set

We initially studied algorithms on synthetically generated ground-truth 3D scenes. These scenes were generated randomly with the Gaussian distribution of points in a 3D cube. Each 3D point was projected by a camera, where the camera orientation and position were selected randomly but looking on the scene. Then the radial distortion using the division model [38] was added to all image points to generate noiseless distorted points. Finally, Gaussian noise with standard deviation σ was added to the distorted image points assuming a 1000×1000 pixel image.

8.4.2 Numerical stability

In the first synthetic experiment, we have studied the numerical stability of both proposed solvers on exact measurements.

In this experiment 1000 random scenes and camera poses were generated. The radial distortion parameter was set to $k_{gt} = -0.2$ and the focal length to $f_{gt} = 1.5$. These values were used to precalibrate the camera in case of calibrated uP2P algorithm, P3P and P4P+f algorithms (radial distortion only).

The rotation error was measured as the rotation angle in the angle-axis representation of the relative rotation $\mathtt{RR}_{gt}^{-1}$ and the translation error as the angle between ground-truth and estimated translation vector. In this case the rotation and translation errors for both algorithms using vertical direction were under the machine precision and therefore we were not able to properly display them here.

Figure 8.2 (Left) shows the log_{10} relative error of the focal length f obtained by selecting the real root closest to the ground truth value $f_{gt} = 1.5$. The log_{10} absolute error of the radial distortion parameter is in Figure 8.2 (Right).

Solvers uP3P+f+k for camera with unknown focal length and radial distortion and given vertical direction (Magenta) is much more numerically stable for both k and f compared to the both P4P+f+k (Green) and the P4P+f solver (Blue).

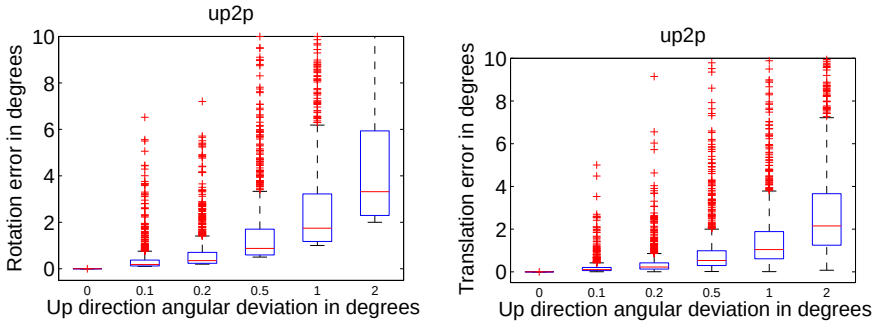


Figure 8.3: The influence of the accuracy of the vertical direction on the estimation of rotation (Left) and translation (Right) for calibrated uP2P algorithm.

8.4.3 Vertical direction angular deviation

In the real applications, the vertical direction obtained either by direct physical measurements by, e.g. gyroscopes and IMUs or from vanishing points, constructed in images is not accurate. The angular accuracy of roll and pitch angle in low cost IMUs like [103] is about $0.5°$, and in high accuracy IMUs is less than $0.02°$.

Therefore, in the next experiment we have tested the influence of the accuracy of the vertical direction on the estimation of rotation (Figure 8.3 (Left) and 8.4 (Top left)), translation (Figure 8.3 (Right) and 8.4 (Top right)), focal length (Figure 8.4 (Bottom left)) and radial distortion (Figure 8.4 (Bottom right)). In this case the ground truth focal length was $f_{gt} = 1.5$ and the radial distortion $k_{gt} = -0.2$.

For each angular deviation in the vertical direction, 1000 estimates of the calibrated uP2P (Figure 8.3) and the uP3P+f+k with unknown focal length and radial distortion (Figure 8.4), were made. The angular deviation in the vertical direction varied from $0°$ to $2°$. All results in Figures 8.3 and 8.4 are represented by the $Matlab$ function boxplot which shows values 25% to 75% quantile as a blue box with red horizontal line at median. The red crosses show data beyond 1.5 times the interquartile range.

The rotation error is measured as the rotation angle in the angle-axis representation of the relative rotation $\mathtt{RR}_{gt}^{-1}$ and the translation error as the angle between ground-truth and estimated translation vector. Focal length and radial distortion figures show directly estimated values.

It can be seen that the median values of estimated focal lengths and radial distortion parameters are very close to the ground truth values (Green horizontal line) even for relatively high error in the vertical direction. Note that the angular rotation and translation errors cannot be smaller than the angular error of the vertical direction.

8.4.4 Experiment with synthetic noise

Boxplots in Figure 8.5 show behavior of the calibrated uP2P solver (Cyan) and uP3P+f+k solver (Magenta), together with the P3P algorithm (Red), the P4P+f algorithm (Blue) and the P4P+f+k algorithm (Green) in the presence of noise added to image points.

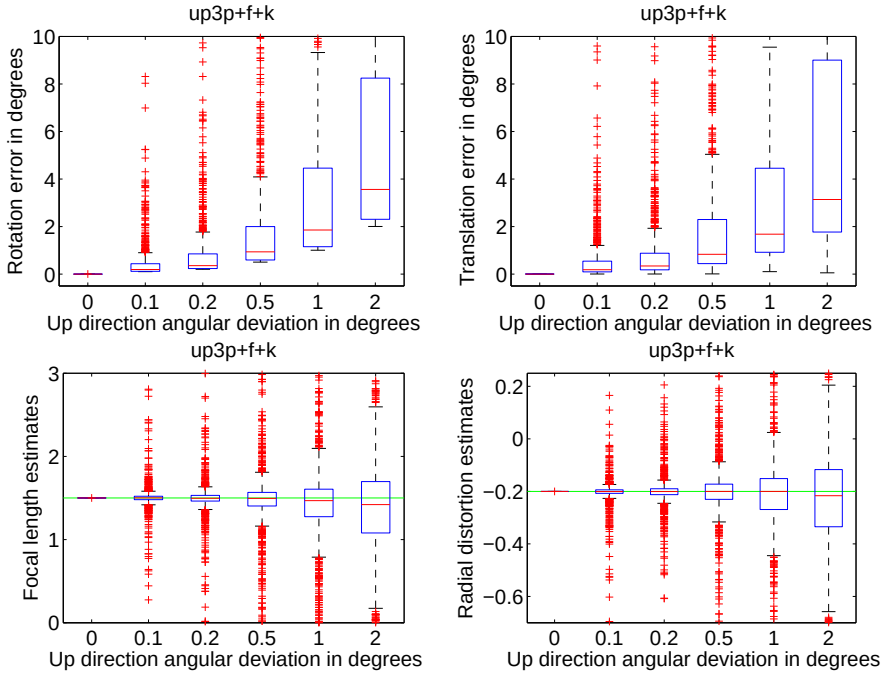


Figure 8.4: The influence of the accuracy of the vertical direction on the estimation of rotation (Top left), translation (Top right), focal length (Bottom right), radial distortion (Bottom right) for uP3P+f+k algorithm.

In this experiment for each noise level, from 0 to 2 pixels, 1000 estimates for random scenes and camera positions and $f_{gt} = 1.5$, $k_{gt} = -0.2$, were made.

Here it can be seen that since both algorithms with known vertical direction use less points than competing PnP solvers, they are a little bit more sensitive to the noise added to the image points. On the other hand, the difference is in the rotation error, see Figure 8.5 (Top left), where both solvers with known vertical direction outperform their competitors. This might be due to the fact that the rotation axis is fixed in both algorithms with known vertical direction. However, the proposed algorithms still provide good estimates also for translation, Figure 8.5 (Top right), focal length (Bottom left) and radial distortion (Bottom right) in the presence of noise.

8.4.5 RANSAC experiment

The final synthetic experiment shows behaviour of the studied algorithms within the RANSAC paradigm [37]. Figure 8.6 shows the mean value of the number of inliers over 1000 runs of RANSAC as a function of the number of cycles of the RANSAC. We compare calibrated uP2P algorithm (Cyan) with the P3P algorithm for calibrated camera (Red), and uP3P+f+k algorithm for camera with unknown focal length and radial distortion (Magenta) with the P4P+f algorithm (Blue) and the P4P+f+k algorithm (Green). Figure 8.6 (Left) shows results for 50% outliers, pixel noise $0.5px$, $f_{gt} = 1.5$, $k_{gt} = -0.2$ and the vertical direction angular deviation 0.02° and

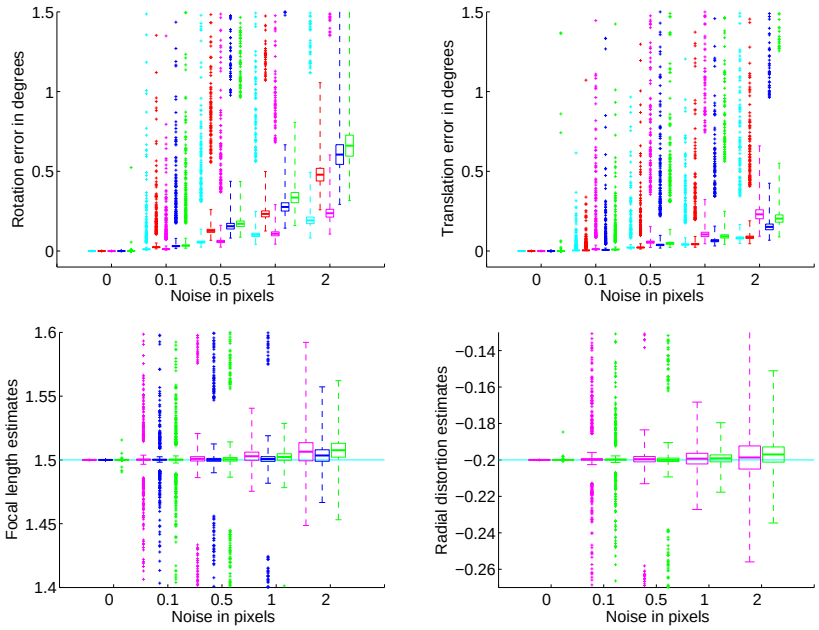


Figure 8.5: Error of rotation (Top left), translation (Top right), focal length estimates (Bottom left) and radial distortion estimates (Bottom right) in the presence of noise for calibrated uP2P algorithm (Cyan), the calibrated P3P algorithm (Red), uP3P+f+k algorithm (Magenta), the P4P+f algorithm (Blue) and the P4P+f+k (Green).

Figure 8.6 (Right) results for the same configuration but the vertical direction angular deviation 1°. These vertical direction angle deviations reflect accessible precisions using low cost and high cost IMUs. Vertical direction precision about 1° can be received by taking pictures by hand using a standard smartphone. As it can be seen both presented algorithms little bit outperform PnP solvers.

8.4.6 Computation times

Since both our problems result in solving one polynomial equation of degree two in one variable and one, respectively two, systems of linear equations and can be solved in a closed-form, they are extremely fast. The $Matlab$ implementations of both our solvers run about $0.1ms$. For comparison, our $Matlab$ implementation of the P3P algorithm [37] runs about $0.6ms$, the $Matlab$ implementation of the P4P+f algorithm [12] downloaded from [63] about $2ms$ and the original implementation of the P4P+f+k algorithm [56] even about $70ms$. Both new solvers for the problems with vertical direction run around $1\mu s$ in C on moderate machine.

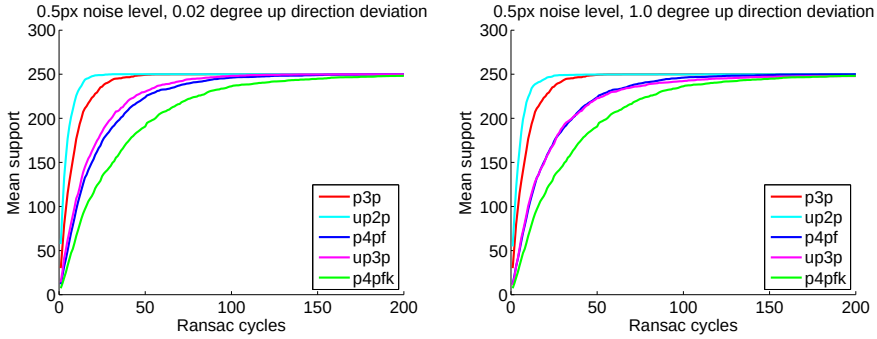


Figure 8.6: The mean value of the number of inliers over 1000 runs of RANSAC as a function of the number of cycles of the RANSAC.

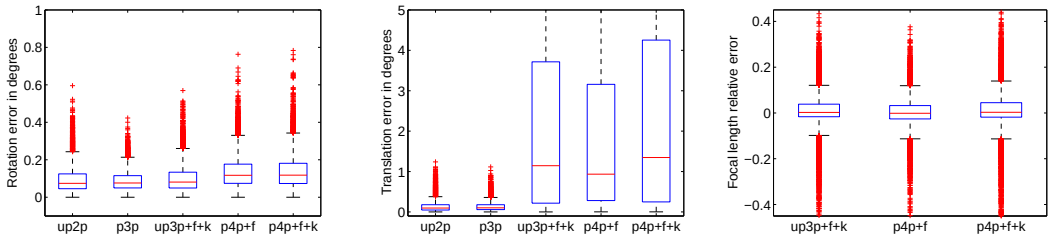


Figure 8.7: Results of the real experiment for vertical direction randomly rotated with standard deviation $0.5°$.

8.4.7 Real images and synthetically generated vertical direction

In the real experiment we created 3D reconstruction from a set of images captured by an off-the-shelf camera. We used reconstruction pipeline similar to one described in [93]. We take this reconstruction as the ground truth reference for further comparison even if it might not be perfect. To simulate measurements from a gyroscope we have extracted "vertical direction" from reconstructed cameras and randomly rotated them by a certain angle. This way we have created 1000 random vertical directions with normal distribution for maximal deviation of $0.02°$, $0.05°$, $0.5°$ and $1°$ to simulate industry quality, standard and low cost gyroscope measurements. Note that from the 3D reconstruction we have a set of 2D-to-3D tentative correspondences as well as correct correspondences. To make the registration more difficult we added more mismatches to the set of tentative correspondences for those cameras where the fraction of correct features was greater than 50%. We randomly connected 2D measurements and 3D points to make 50% of tentative correspondences wrong. Next, for each random vertical direction we executed locally optimized RANSAC [23] with both studied algorithms. Further we also evaluated 1000 runs of locally optimized RANSAC with all solvers, i.e. the calibrated P3P, P4P+f, P4P+f+k solvers and compared all results with the ground truth rotation, translation and the focal length. Note

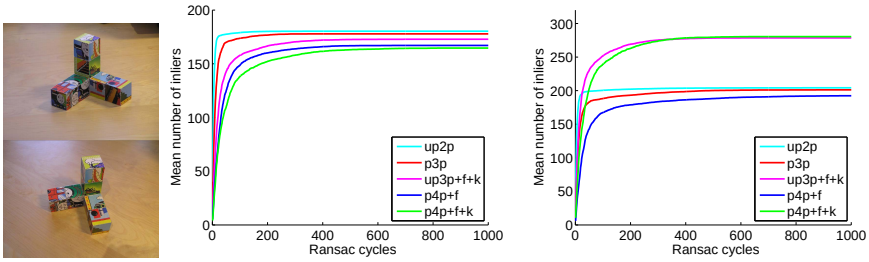


Figure 8.8: Real experiment: Example of input images (Left). Results from RANSAC on the image with small radial distortion (Center) and on the image with significant radial distortion (Right).

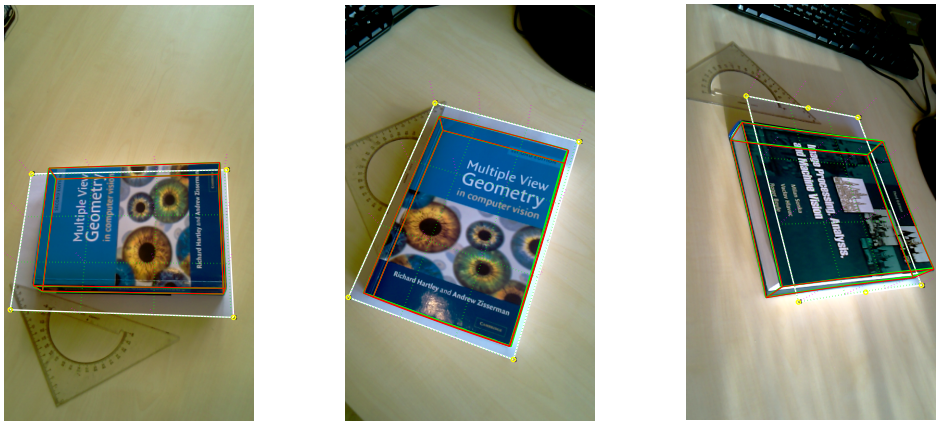


Figure 8.9: Results of the real experiment for real IMU sensor from HTC HD2.

that both calibrated uP2P algorithm and the P3P algorithm for calibrated camera require internal camera calibration. Hence we used the calibration obtained from the 3D reconstruction.

The results for vertical directions randomly rotated with standard deviation $0.5°$ are shown in Figure 8.7. It can be seen that algorithms uP2P and uP3P+f+k provide comparable estimates of rotation (Left), translation (Center) and focal length (Right) even with higher vertical direction error. Moreover, the proposed algorithms use less points, give only up to two solutions which have to be verified inside RANSAC loop and are considerably faster than competing PnP algorithms. All these aspects are very important in RANSAC and real applications. The mean value of the number of inliers over 1000 runs of RANSAC as a function of the number of samples of RANSAC for two cameras and vertical direction deviation $0.5°$ can be seen in Figure 8.8. It can be seen that both presented algorithms behave very well.

8.4.8 Real images with vertical direction from real IMU

In this experiment we used HTC HD2 phone device to obtain real images with IMU data. We took several images of a book placed on a A4 paper by hand. The image size was 800×600 pixels. We assigned 3D coordinates to the paper corners and manually clicked corresponding 2D points in all input images. First, RANSAC with the new solvers was used. Then, a non-linear refinement [47] optimizing the focal length, radial distortion, rotation and translation was performed. For calibrated uP2P algorithm, we used radial distortion and focal length calculated using non-linearly improved result obtained from uP3P+f+k.

Figure 8.9 shows the 3D model of the paper and the book back projected to images using the calculated camera poses. The green wire-frame model of the book shows the estimated camera from a minimal sample using new algorithms (uP2P (Left), uP3P+f+k (Center+Right)), red color represents model after the non-linear refinement. Magenta lines represent the gravity vector obtained from the phone. We have obtained quite accurate results (reprojection error about 1.5px) even for standard images taken by hand and without calibrating the relative position of the IMU sensor and camera inside the phone.

8.5 Conclusion

The vertical direction obtained by direct physical measurement by, e.g. gyroscopes and inertial measurement units (IMUs) or from vanishing points constructed in images, can radically simplify the camera absolute pose problem. In the case of the two problems studied in this chapter the information about the vertical direction leads to very fast, efficient and numerically stable closed-form solutions. For comparison, the algorithm for estimating absolute pose of a camera with unknown focal length and radial distortion without knowing the vertical direction presented by Josephson et al. [56] resulted to a huge computational problem - LU decomposition of 1134×720 matrix and further eigenvalue computations - which is less practical in real applications. Compared to the algorithms presented in previous Chapter 7, the algorithm with vertical direction is much faster.

Experiments also show that both algorithms, uP2P and uP3P+f+k, are very useful in real applications like structure-from-motion or for surveillance cameras, which are usually placed in urban area where vertical vanishing point can be calculated easily. Moreover, since still more and more cameras and smart devices are equipped with IMUs and can save information about the vertical direction, i.e. roll and pitch angles, to the image header, we see great future potential of these solvers.

9 Performance

We studied several absolute pose problems, played with problems formulations and focused on finding formulations which lead to the smallest yet accurate and numerically stable solvers. We solved the problems either in a close form, using Gröbner basis and the automatic generator [64] or using the polynomial eigenvalue computation. In this chapter we show how to speed up solvers which use eigenvalue computations, for example the Gröbner basis or the polynomial eigenvalue methods. We show how to convert the problem into a different one and replace computationally expensive eigenvalue computation by more efficient single variable polynomial roots calculation using robust numerical methods.

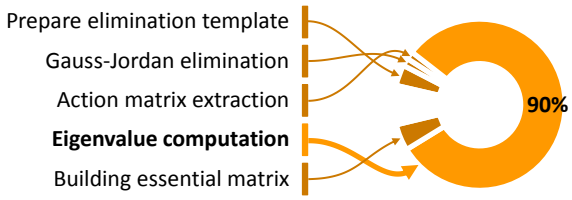


Figure 9.1: Time spent in various modules during the calculation of the five point relative pose algorithm [96].

In fact, not only the solvers studied in previous chapters, but many other important computer vision problems were solved using the Gröbner basis method or polynomial eigenvalue method [12, 13, 18, 68, 97, 96, 67] in the last few years. Minimal solvers are often used inside RANSAC [11] and they are used in large systems like SfM pipelines. Therefore, it is very important to maximize their efficiency.

This chapter focuses primarily on the Gröbner basis method, however, solvers based on polynomial eigenvalue method can be optimized in the same way. Recall that Gröbner basis solvers usually consist of Gauss-Jordan (G-J) elimination or LU decomposition of one or several matrices created using elimination templates [64] and by the eigenvalue computation of a multiplication (action) matrix [95]. It has been recently demonstrated that the numerical stability of Gröbner basis solvers can be improved, e.g. by reordering columns in elimination templates, using QR or LU decompositions or by "extending the basis" [19, 17, 20]. Several methods for reducing the size of elimination templates, and in this way speeding up Gröbner basis solvers and improving their stability, were proposed [83, 64].

However, it is usually not the G-J elimination or the LU decomposition of elimination templates but the eigenvalue computation which takes the biggest fraction of time in these solvers, see Figure 9.1.

Although the elimination templates are for some solvers quite large, they are relatively sparse

and thus can be efficiently eliminated using sparse methods. Therefore, this part of Gröbner basis solvers usually takes only a few microseconds.

Usually the eigenvalue and the eigenvector computation is the bottleneck taking, e.g. more than $50\mu s$ for 10×10, 140μ for 15×15 or even $250\mu s$ for 20×20 action matrices. The computation time depends on the size of the action matrix, and the size of the action matrix depends on the number of solutions of the problem. The number of solutions of a given formulation of a problem and therefore the size of the action matrix can't be often reduced. Moreover, the action matrices are relatively dense and hence sparse solvers do not help.

On the other hand, many solutions, which are obtained as eigenvalues of these action matrices, are often not feasible. They are either complex or out of the range of interest, e.g. too large or negative focal lengths, depths or radial distortion coefficients.

We next show how to use Gröbner basis solvers in such that only the promising interval of solutions is examined.

Problem 8. *Let us have a Gröbner basis solver which solves some problem using eigenvectors of some action matrix. Find a method for replacing the eigenvalue computation by calculation of roots of a single variable polynomial.*

This way we may save huge amount of work spent in calculating solutions which are often later dropped.

We show two different methods for obtaining a single variable polynomial. The first method is based on FGLM [34] algorithm for transforming the grevlex Gröbner basis to the lexicographic Gröbner basis which contains a single variable polynomial. Next we show how can the action matrix be helpful in removing polynomial division used in FGLM [34] algorithm. We also notice that complete grevlex Gröbner bases are often not needed to perform the transformation to a lexicographic Gröbner basis. Our modification is much more efficient than the standard FGLM [34] algorithm and results in a significant speed up of Gröbner basis solvers. We call this algorithm modified FGLM or "matrix" FGML.

We also show how the modified "matrix FGLM" algorithm is related to Krylov's algorithm [53] for computing the coefficients of the characteristic polynomial of an action matrix. It reveals the relationship between the standard FGLM algorithm [34], an action matrix and its characteristic polynomial. Since Krylov's algorithm [53] is suitable only for small or medium size action matrices, we finally present a method based on Danilevsky algorithm [28] for computing the coefficients of the characteristic polynomial, which is very efficient, numerically stable and can be used even for large problems.

We demonstrate the usefulness of both methods, the modified "matrix FGLM" algorithm and the Danilevsky algorithm [28] for computing the characteristic polynomial, by achieving a significant speedup of several important minimal computer vision problems.

For completeness, we also briefly describe Faddeev-Leverrier algorithm [31] for the characteristic polynomial coefficients computation. This algorithm is however slower and numerically less stable compared to the other methods.

9.1 Gröbner basis conversion method

One standard method for solving systems of polynomial equations is based on converting the Gröbner basis w.r.t. some monomial ordering which is easier to compute, e.g. the grevlex ordering, to a lexicographic Gröbner basis containing a single-variable polynomial. There exist several algorithms for Gröbner basis conversion, for example the well known FGLM algorithm [34]. We review the algorithm in the next section.

Finding a Gröbner basis w.r.t. the lexicographic ordering directly is often computationally more expensive than computing a Gröbner basis w.r.t. some other monomial ordering and converting it to a lexicographic Gröbner basis.

FGLM algorithm hasn't been used to solve minimal computer vision problems yet. It is probably because this method requires to perform quite inefficient and sometimes also numerically unstable polynomial division. Therefore, it was assumed that it is more efficient to find solutions to the initial system of polynomial equations by computing the eigenvalues and the eigenvectors of an action matrix constructed from, for example, a grevlex Gröbner basis.

9.1.1 Standard FGLM algorithm

The standard FGLM algorithm [34] starts with some Gröbner basis G of an ideal I and converts it to a lexicographic Gröbner basis G_{lex}, or a Gröbner basis w.r.t. some other monomial ordering, of the same ideal. For more details refer to [34, 62]. Here we provide just a brief overview.

The algorithm uses only two structures, a list $G_{lex} = \{g_1, \ldots, g_k\}$ which contains a subset of the lexicographic Gröbner basis at each stage of the algorithm and a list B_{lex} which contains a subset of the monomial basis of the quotient ring $A = \mathbb{C}[x_1, \ldots, x_n]/I$. Both these lists are initially empty. For each monomial $\mathbf{x}^\alpha \in \mathbb{C}[x_1, \ldots, x_n]$ in increasing lexicographic ordering, starting with $\mathbf{x}^\alpha = 1$, the algorithm performs these three steps:

1. For the input $\mathbf{x}^\alpha$, compute $\overline{\mathbf{x}^\alpha}^G$
   - If $\overline{\mathbf{x}^\alpha}^G = \sum_j c_j \overline{\mathbf{x}^{\alpha(j)}}^G$, where $\mathbf{x}^{\alpha(j)} \in B_{lex}$ and $c_j \in \mathbb{C}$, i.e. if $\overline{\mathbf{x}^\alpha}^G$ is linearly dependent on the remainders on division by G of the monomials in B_{lex}, then add polynomial $g = \mathbf{x}^\alpha - \sum_j c_j \mathbf{x}^{\alpha(j)} \in I$ to G_{lex} as the last element.
   - Otherwise add $\mathbf{x}^\alpha$ to B_{lex} as the last element.
2. Termination Test

   If a polynomial g was added to G_{lex}, and if $LT(g)$ is a power of the greatest variable in the lexicographic ordering, then STOP.
3. Next Monomial

   Replace $\mathbf{x}^\alpha$ with the next monomial, w.r.t. the lexicographic ordering, which is not divisible by any of the monomials $LT(g_i)$ for $g_i \in G_{lex}$. GOTO 1.

Note that for the division by G in this algorithm the original ordering (not the lexicographic ordering) is used and that for our applications it is sufficient to stop this algorithm after finding the first polynomial from G_{lex}, i.e. the single-variable polynomial.

This means that for the ordering of the variables $x_1 > x_2 > \cdots > x_n$, the FGLM algorithm starts with the monomial $\mathbf{x}^\alpha = 1$ and continues with monomials x_n, x_n^2, x_n^3 and so on in increasing lexicographic ordering and computes remainders $\overline{x_n^i}^G$ on division by G. After reaching x_n^s, where s is the number of solutions to the initial system of equations, it expresses $\overline{x_n^s}^G$ as a linear combination of $\overline{x_n^i}^G$, $i < s$. The coefficients of this linear combination are the coefficients of the single-variable polynomial form the lexicographic Gröbner basis G_{lex}.

9.1.2 Modified matrix FGLM algorithm

The standard FGLM algorithm [34] performs polynomial division which is the bottleneck of the method and may be for some problems less efficient than the "eigenvalue action matrix" method.

Here we show how to use action matrix powers instead of polynomial division. We propose a method, which we call the "matrix FGLM" algorithm, and which requires only matrix-vector multiplications for obtaining a single-variable polynomial. This method results in a significant speedup over the eigenvalue computation for many problems.

This method even doesn't require a complete grevlex Gröbner basis. All it needs is the action matrix $\mathtt{M}_{x_i}$ of the linear operator T_{x_i} of the multiplication by some variable x_i in $A = \mathbb{C}[x_1, \ldots, x_n]/I$ w.r.t. the standard monomial basis $B = \left\{ [\mathbf{x}^\alpha] : \overline{\mathbf{x}^\alpha}^G = \mathbf{x}^\alpha \right\}$ of A.

In this case the j^{th} column of the matrix $\mathtt{M}_{x_i}$ corresponds to the remainder $\overline{x_i b_j}^G$, where $[b_j] \in B$. For example, imagine that the basis $B = ([xy], [x], [y], [1])$ and that we have the action matrix $\mathtt{M}_x = (m_{ij})_{i,j=1,\ldots,4}$. Then we can obtain the remainder of x^2 on the division by G from this action matrix as $\overline{x^2}^G = m_{12}xy + m_{22}x + m_{32}y + m_{42}$.

Moreover, since the action matrix $\mathtt{M}_{x_i}$ represents the multiplication by $[x_i]$ in A, we can use it to obtain remainders $\overline{x_i^k}^G$ also for higher powers k.

Let l be the largest power such that $\left[x_i^l\right] \in B$. Then we have $\overline{x_i^k}^G = x_i^k$ for $k \leq l$ and for $k > l$ we can obtain $\overline{x_i^k}^G$ in the following way.

Let $\left[x_i^l\right] = [b_q]$ for some $[b_q] \in B$, i.e. $\left[x_i^l\right]$ is the q^{th} element of B. We set

$$\mathbf{v}_l = [0 \ldots 1 \ldots 0]^\top, \tag{9.1}$$

$$\mathbf{v}_{j+1} = \mathtt{M}_{x_i} \mathbf{v}_j, \quad j = l, \ldots, s-1, \tag{9.2}$$

where 1 in $\mathbf{v}_l$ is on the q^{th} place and s is the number of solutions of a given system of polynomial equations.

Then we have

$$\overline{x_i^k}^G = \mathbf{v}_k^\top \mathbf{b}, \quad k = l+1, \ldots, s, \tag{9.3}$$

where $\mathbf{b} = [b_1, \ldots\ b_s]^\top$, for $[b_j] \in B$.

Moreover, equations (9.1), (9.2) and (9.3) hold also for $l = 0$, i.e. for all remainders $\overline{x_i^k}^G$.

This means that we can compute the remainders $\overline{x_i^k}^G$ by a simple matrix-vector multiplication. This is much more efficient that the polynomial division performed in the standard FGLM [34] algorithm described in Section 9.1.1.

After obtaining $\overline{x_i^k}^G$ using the presented method we can continue with the standard FGLM algorithm and obtain the coefficients c_j of the single-variable polynomial by finding the coefficients of the linear combination

$$\overline{x_i^s}^G = \sum_{j=0}^{s-1} c_j \overline{x_i^j}^G. \tag{9.4}$$

The single-variable polynomial has then the form $x_i^s - c_{s-1}x_i^{s-1} - \cdots - c_1 x_i - c_0 = 0$.

This results in solving a simple system of s linear equations. After finding the coefficients of the single-variable polynomial we use Sturm-sequences [49] to find its roots on the interval which is feasible for the variable x_i, e.g. only positive values for the focal length. This gives us solutions to the variable x_i on this interval.

Solutions to the remaining variables can be obtained either by back-substituting obtained solutions to the initial equations and solving a simplified system, by computing the eigenvectors of the action matrix $\mathtt{M}_{x_i}$, which lies in the kernel of matrix $(\mathtt{A} - x_i\mathtt{I})$, or by performing the whole presented process also on action matrices for the remaining variables.

We next show that the presented "matrix FGLM" algorithm, which we have obtained using properties of the action matrix, is equivalent to the Krylov's algorithm [53] for computing the coefficients of the characteristic polynomial.

9.2 Characteristic polynomial method

Let $\mathtt{A}$ be an $n \times n$ matrix. To find its eigenvalues and eigenvectors we need to solve the matrix equation $\mathtt{A}\mathbf{x} = \lambda\mathbf{x}$, where λ is the eigenvalue corresponding to the eigenvector $\mathbf{x} \neq 0$. Therefore the eigenvalues λ must satisfy the equation $\det(\mathtt{A} - \lambda\mathtt{I}) = 0$. This is an n^{th} degree monic polynomial

$$p_{\mathtt{A}}(\lambda) = \lambda^n + p_{n-1}\lambda^{n-1} + \cdots + p_1\lambda + p_0 \tag{9.5}$$

in λ called the characteristic polynomial of the matrix $\mathtt{A}$. Its roots are the eigenvalues of the matrix $\mathtt{A}$.

Next we describe the methods for finding the coefficients p_i of the characteristic polynomial $p_{\mathtt{A}}(\lambda)$ for a given matrix $\mathtt{A}$. The first one is the Krylov's method [53], which is equivalent to the "matrix FGLM" algorithm presented in Section 9.1.2, the second one is the Danilevsky method [28], which is numerically the most stable. Finally, we briefly review the Faddeev-Leverrier algorithm [31].

9.2.1 Krylov's method

Krylov's method for computing the characteristic polynomial $p_{\mathtt{A}}(\lambda)$ uses Cayley-Hamilton theorem. According to this theorem the matrix $\mathtt{A}$ satisfies its characteristic polynomial, i.e.

$$p_{\mathtt{A}}(\mathtt{A}) = \mathtt{A}^n + p_{n-1}\mathtt{A}^{n-1} + \cdots + p_1\mathtt{A} + p_0\mathtt{I}_n = \mathtt{O}_n, \tag{9.6}$$

where $\mathtt{I}_n$ is the $n \times n$ identity matrix and $\mathtt{O}_n$ is the $n \times n$ zero matrix.

Let $\mathbf{v} \neq 0$ be a non-zero $n \times 1$ vector, e.g. $\mathbf{v} = [1\ 0 \ldots 0]^\top$. Then from (9.6) we get

$$\mathtt{A}^n\mathbf{v} + p_{n-1}\mathtt{A}^{n-1}\mathbf{v} + \cdots + p_1\mathtt{A}\mathbf{v} + p_0\mathbf{v} = \mathbf{0}. \tag{9.7}$$

This means that the vectors $\mathbf{v}_k,\ k = 1, \ldots, n$, defined as

$$\mathbf{v}_k = \mathtt{A}^k\mathbf{v} \tag{9.8}$$

are linearly dependent.

Therefore, to obtain the coefficients of the characteristic polynomial, it is sufficient to compute vectors $\mathbf{v}_k$ (9.8), $i = 1, \ldots, n$ and find the coefficients of the linear combination (9.7) by solving a system of n linear polynomial equations in n unknowns.

This is exactly what is done in the "matrix FGLM" algorithm presented in Section 9.1.2. We see that the sequence (9.8) of vectors $\mathbf{v}_k$, called the Krylov's sequence, is equivalent to the sequence of vectors (9.2), used in the "matrix FGLM" algorithm, for $\mathtt{M}_{x_i} = \mathtt{A}, s = n, l = 0$, and $\mathbf{v}_0 = \mathbf{v}$. The searched coefficients p_i of the characteristic polynomial are then the coefficients $-c_i$ from (9.4).

Krylov's method, as well as the "matrix FGLM" algorithm presented in Section 9.1.2, perform $\frac{3}{2}n^2(n+1)$ multiplications and divisions and work well for small or medium size matrices. For larger matrices numerical problems may appear.

We next describe another method for computing the coefficients of the characteristic polynomial $p_{\mathtt{A}}(\lambda)$ (9.5) which is more stable than the Krylov's method.

9.2.2 Danilevsky method

A very efficient and numerically stable method for computing the coefficients of the characteristic polynomial (9.5) was proposed by Danilevsky [28]. The main idea of this method is in transforming the matrix

$$\mathtt{A} = \begin{bmatrix} a_{1,1} & a_{1,2} & \ldots & a_{1,n} \\ a_{2,1} & a_{2,2} & \ldots & a_{2,n} \\ \ldots & \ldots & \ldots & \ldots \\ a_{n,1} & a_{n,2} & \ldots & a_{n,n} \end{bmatrix}, \tag{9.9}$$

to the matrix

$$\mathtt{P} = \begin{bmatrix} -p_{n-1} & -p_{n-2} & \ldots & -p_1 & -p_0 \\ 1 & 0 & \ldots & 0 & 0 \\ 0 & 1 & \ldots & 0 & 0 \\ \ldots & \ldots & \ddots & \ldots & \ldots \\ 0 & 0 & \ldots & 1 & 0 \end{bmatrix}. \tag{9.10}$$

The matrix $\mathtt{P}$ is known as the companion matrix of $p_{\mathtt{A}}(\lambda)$ (9.5), or the Frobenius form of $\mathtt{A}$.

Since the matrices $\mathtt{A}$ and $\mathtt{P}$ have the same characteristic polynomial $p_{\mathtt{A}}(\lambda)$, they are similar. This means that $\mathtt{P} = \mathtt{T}^{-1}\mathtt{A}\mathtt{T}$ for some regular transformation matrix $\mathtt{T}$.

In the Danilevsky method this transformation from the matrix $\mathtt{A}$ to $\mathtt{P}$ is done by $n-1$ similarity transformations $\mathtt{T}_i$ which successively transform the rows of $\mathtt{A}$, beginning with the last row, into the corresponding rows of $\mathtt{P}$. In fact this method applies a special form of the G-J elimination to transform $\mathtt{A}$ to its Frobenius form.

Here we describe only the first step of this method to form of the matrices $\mathtt{T}_{n-1}$ and $\mathtt{T}_{n-1}^{-1}$. In this first step we want to transform the last row of $\mathtt{A}$ to the last row of $\mathtt{P}$. Assuming $a_{n,n-1} \neq 0$, the transformations have the form

$$\mathtt{T}_{n-1} = \begin{bmatrix} 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ \dots & \dots & \ddots & \dots & \dots \\ -\frac{a_{n,1}}{a_{n,n-1}} & -\frac{a_{n,2}}{a_{n,n-1}} & \dots & \frac{1}{a_{n,n-1}} & -\frac{a_{n,n}}{a_{n,n-1}} \\ 0 & 0 & \dots & 0 & 1 \end{bmatrix}, \tag{9.11}$$

and

$$\mathtt{T}_{n-1}^{-1} = \begin{bmatrix} 1 & 0 & \dots & 0 & 0 \\ 1 & 1 & \dots & 0 & 0 \\ \dots & \dots & \ddots & \dots & \dots \\ a_{n,1} & a_{n,2} & \dots & a_{n,n-1} & a_{n,n} \\ 0 & 0 & \dots & 0 & 1 \end{bmatrix}. \tag{9.12}$$

The matrix $\mathtt{T}_{n-1}^{-1}\mathtt{A}\mathtt{T}_{n-1}$ has the desired form with the last row equal to the last row of $\mathtt{P}$ so the process can continue with the next row.

In this way we can easily construct $n-1$ similarity transformations $\mathtt{T}_i$ such that

$$\mathtt{P} = \mathtt{T}_1^{-1}\mathtt{T}_2^{-1}\dots\mathtt{T}_{n-1}^{-1}\mathtt{A}\mathtt{T}_{n-1}\dots\mathtt{T}_2\mathtt{T}_1. \tag{9.13}$$

Then the coefficients of the characteristic polynomial $p_{\mathtt{A}}(\lambda)$ (9.5) can be extracted from the matrix $\mathtt{P}$. Note that it is not necessary to perform full matrix multiplications when computing $\mathtt{T}_i^{-1}\mathtt{A}\mathtt{T}_i$ since matrices $\mathtt{T}_i$ are close to the identity matrix.

More about this method and solutions for cases when some pivots are equal to zero can be found in [28, 53]. The number of multiplications and divisions performed in this method is equal to $(n-1)\left(n^2+n-1\right)$, i.e. proportional to a single G-J elimination. This method is numerically more stable than the Krylov's method presented in Section 9.2.1 and its accuracy can be even increased by pivoting.

Moreover, Danilevsky method can be also used to find the eigenvectors of the matrix $\mathtt{A}$ by transforming the eigenvectors of the matrix $\mathtt{P}$, which have very simple form, using the transformations $\mathtt{T}_i$.

9.2.3 Faddeev-Leverrier algorithm

Let tr $(\mathtt{A})$ be the trace of a matrix $\mathtt{A}$, i.e. the sum of the diagonal elements. The Faddeev-Leverrier algorithm constructs a sequence of matrices $\mathtt{D_i}, i = 1, \ldots, n$ such that

$$\mathtt{D}_1 = \mathtt{I}, \tag{9.14}$$

$$\mathtt{D}_i = \mathtt{A}\mathtt{D}_{i-1} + p_{n-i+1}\mathtt{I}, \quad i = 2, \ldots, n, \tag{9.15}$$

with

$$p_{n-i} = -\frac{1}{i}\mathrm{tr}\left(\mathtt{A}\mathtt{D}_i\right), \quad i = 1, \ldots, n, \tag{9.16}$$

where p_i are searched coefficients of the characteristic polynomial $p_{\mathtt{A}}(\lambda)$ (9.5).

After obtaining the coefficients of the characteristic polynomial, we can find eigenvalues of the matrix $\mathtt{A}$ as its roots. Moreover, the matrices $\mathtt{D}_i$ constructed in this algorithm can be used to efficiently find eigenvectors of the matrix $\mathtt{A}$ as the columns of the matrices

$$\mathtt{C} = \sum_{i=1}^{n} \lambda^{n-i}\mathtt{D}_i. \tag{9.17}$$

Non-zero columns of the matrix $\mathtt{C}$ (9.17) are the eigenvectors corresponding to the eigenvalue λ. The Faddeev-Leverrier algorithm requires to perform $O\left(n^3(n-1)\right)$ multiplications in the case of $n \times n$ matrices and is useful for small or medium sized matrices.

We next briefly describe how the proposed methods for computing the coefficients of the characteristic polynomial can be used to solve systems of polynomial equations.

9.2.4 Characteristic polynomial of the action matrix

Let's consider that we have a Gröbner basis solver with constructed action matrix $\mathtt{M}_{x_i}$ for the variable x_i. We know that the eigenvalues of $\mathtt{M}_{x_i}$ give solutions to x_i and that the solutions to the remaining variables can be obtained from its eigenvectors.

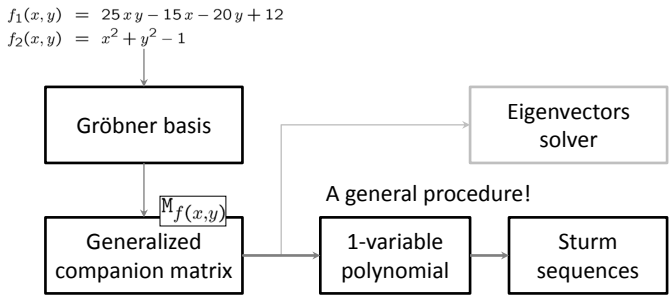


Figure 9.2: Illustration of solution paths.

Instead of computing these eigenvalues and eigenvectors using a standard numerical algorithm we can use, for example, the presented Danilevsky method (Figure 9.2). This is efficient especially when we know some constraints on the variable x_i.

After finding the coefficients of the characteristic polynomial $p_{\mathtt{M}_{x_i}}(x_i)$ of the action matrix $\mathtt{M}_{x_i}$, as described in Section 9.2.2, we can use Sturm-sequences [49] to find its roots on the interval which is feasible for the variable x_i, e.g. we look only for positive values for the focal length.

Solutions to the remaining variables x_j can be obtained either by back-substituting obtained solutions to the initial system of polynomial equations and solving the simplified system or by computing the eigenvectors of the action matrix $\mathtt{M}_{x_i}$ corresponding to obtained solutions.

9.3 Minimal solvers

To show the usefulness of the methods presented in Sections 9.1.2 and 9.2.2 we have used them to create efficient fast solvers to three important minimal problems. All these problems have been previously solved using the Gröbner basis "eigenvalue action matrix" method like solvers studied in previous chapters.

We next briefly describe these three problems and their existing Gröbner basis solutions which are the bases of our new fast solutions. Note that in all our solutions we only replace the eigenvalue action matrix computation, performed in the existing Gröbner basis solver, with the proposed "matrix FGLM" method, the Danilevsky method, or with the Faddeev-Leverrier algorithm for computing the coefficients of the characteristic polynomial, followed by the computation of the roots of the obtained single-variable polynomial using Sturm-sequences [49].

9.3.1 5-point relative pose

The problem of estimating relative pose of two calibrated cameras from five image point correspondences is one of the oldest and well-studied problems.

The state-of-the-art methods [84, 97] use the formulation which results in solving ten equations in three unknowns and gives ten solutions to this problem.

In this case, the Gröbner basis solution [96] is quite simple since it only performs G-J elimination of the 10×20 coefficient matrix representing the initial ten polynomials. Then, the action matrix is created from the rows of this eliminated matrix. In [96] the solutions are obtained from the eigenvalues and the eigenvectors of this action matrix. This is much less efficient than the state-of-the-art solver [84], which uses special structure of the initial equations to obtain almost a closed-form solution.

However, in experiments we show that by replacing the eigenvalue computation in the Gröbner basis solver [96] with the "matrix FGLM" method or the Danilevsky method for computing the coefficients of the characteristic polynomial, followed by the computation of its roots using the Sturm-sequences [49], we obtain comparable efficiency to the Nister's state-of-the-art solution [84].

9.3.2 6-point focal length problem

The problem of estimating relative camera pose for two cameras with unknown, but equal, focal length from minimal number of six point correspondences has 15 solutions. It leads to solving ten equations in three unknowns [97].

There exist several Gröbner basis solutions to this problem [97, 19, 64] which find the solutions by computing the eigenvalues of an action matrix using a standard eigenvalue method. The solution [64], which results in the smallest "elimination template", performs one G-J elimination of a 31×46 matrix. From the rows of this eliminated matrix the action matrix for the unknown focal length is created.

This action matrix is the input to methods presented in Section 9.2.

9.3.3 P4P+f problem

The last problem is the problem we solved in Chapter 6, i.e. the problem of estimating the absolute pose of a camera with unknown focal length from four 2D-3D point correspondences. We selected the solver based on the ratios of distances invariant with elimination template of size 86×96. Recall that this problem results in five equations in four unknowns and has ten solutions and hence results in a 10×10 action matrix. This is a Gröbner basis "eigenvalue action matrix" solver which performs a single G-J elimination of the 86×96 matrix and for which the action matrix for the unknown focal length is created. This action matrix is again the input to both our presented methods.

9.4 Polynomial roots calculation

The goal of previous sections was to produce an univariate polynomial $p(t) = c_0 + c_1 x + \cdots + c_{n-1}x^{n-1} + x^n$ which encodes a solution to a given problem. The reason for this was to replace the expensive eigenvalue and eigenvector calculations. Using Sturm's sequences is more convenient for many computer vision problems. It is because the action variable, the only unknown in the univariate polynomial, is usually connected with some physical attribute, e.g. focal length, radial distortion, distance, angle, etc. Hence most of the time we are interested only in real values and often in values in a certain interval only.

Let's briefly look at the Sturm's bracketing method. Given a polynomial $p(t)$ and its first order derivative $p'(t)$, let $g(t)$ be a square free polynomial, i.e. without repeated roots

$$g(t) = \frac{p(t)}{gcd(p(t), p'(t))}.$$

Then Sturm sequence is a finite sequence of polynomials defined as

$$\begin{aligned} p_0(t) &= g(t), \\ p_1(t) &= p_0'(t), \\ p_2(t) &= -rem\left(p_0(t), p_1(t)\right), \\ &\vdots \\ p_{n-1}(t) &= -rem\left(p_{n-3}(t), p_{n-2}(t)\right), \\ p_n(t) &= constant. \end{aligned}$$

where $rem\,(p, q)$ denotes the remainder on division of p by q. According to Sturm's theorem [49], if

- $(a, b]$ is an interval in $\mathbb{R}$,
- s_a is the number of sign changes in $p_0(a), p_1(a), \ldots, p_n(a)$,
- s_b is the number of sign changes in $p_0(b), p_1(b), \ldots, p_n(b)$,

then the number of distinct real roots of polynomial $p(t)$ on interval $(a, b] = s_b - s_a$, where $a < b$ are real numbers. Simply, by splitting interval $(a, b]$ we can isolate any root up to a given precision [49]. Note that Sturm's theorem deals with real roots only and disregards their multiplicities.

Alternatively, the companion matrix can be used to find roots of a polynomial. Idea behind the companion matrix is to build a matrix which has the same characteristics polynomial as a given input polynomial $p(t) = c_0 + c_1 x + \cdots + c_{n-1}x^{n-1} + x^n$. Then, eigenvalues of this matrix [26] are roots of the polynomial $p(t)$. The companion matrix has form:

$$\mathtt{A} = \begin{bmatrix} 0 & 0 & \ldots & 0 & -c_0 \\ 1 & 0 & \ldots & 0 & -c_1 \\ 0 & 1 & \ldots & 0 & -c_2 \\ \ldots & \ldots & \ddots & \ldots & \ldots \\ 0 & 0 & \ldots & 1 & -c_{n-1} \end{bmatrix}. \tag{9.18}$$

Of course, once the action matrix is constructed, it is faster to calculate eigenvalues directly rather then calculating its characteristic polynomial and finding roots of the characteristic polynomial using the companion matrix. However if we need to calculate roots of a single variable polynomial, e.g. when a Gröbner basis is already in the lexicographic ordering, or if we are interested in complex roots too, then the companion matrix offers more than just an algebraically correct solution. It is because the companion matrix is in a special form which offers a better starting point for many eigenvalue solvers. In particular, the companion matrix is in upper Hessenberg form [87]. Popular QR/QZ algorithms [87] first transform the input general matrices to Hessenberg form and then reduce the matrix using the shifted QR/QZ factorization to a triangular form. Transforming a matrix to the Hessenberg form simplifies algorithms and improves algorithm convergence. Since the companion matrix is already in the Hessenberg form, we can skip the matrix transformation step. This saves about 30% of computation time.

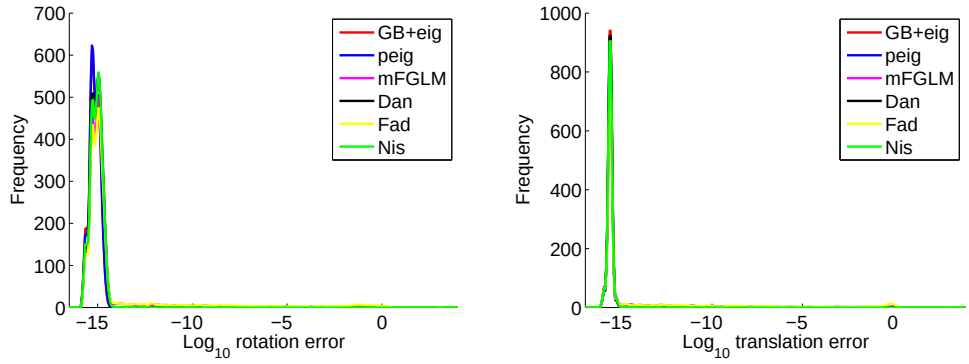


Figure 9.3: Numerical stability of studied 5-pt relative pose solvers. Peaks on the right hand side represent algorithm failure.

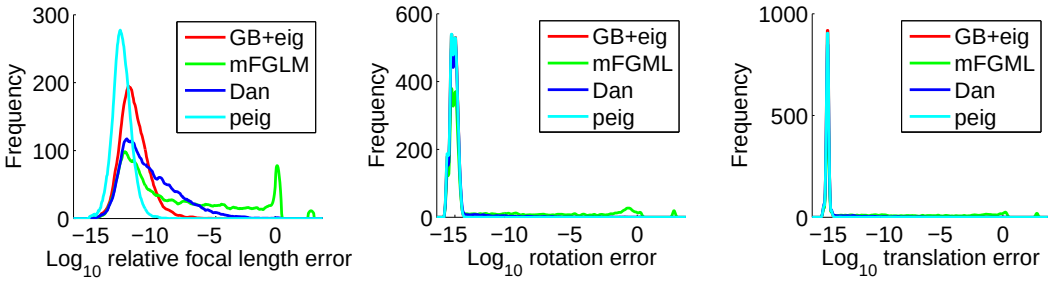


Figure 9.4: Numerical stability of studied 6-pt focal length solvers. Peaks on the right hand side represent algorithm failure.

9.5 Experiments

This section compares the speed and the numerical stability of the proposed solutions with the existing state-of-the-art solvers. Since all solvers are algebraically equivalent, we have evaluated them on synthetic noise free data only.

In all our experiments and performance tests we executed each algorithm $10k$ times on synthetically generated data. All scenes in these experiments were generated using 3D points randomly distributed in a 3D cube. Each 3D point was projected by a camera with random feasible orientation and position and a random or fixed focal length.

9.5.1 Numerical stability

We first study the numerical stability of the proposed approaches on three selected important computer vision problems. We collected several different publicly available solvers from the Internet [63], reimplemented several state-of-the-are methods [84, 12, 97], and compared them with our new solvers based on the methods presented in previous sections.

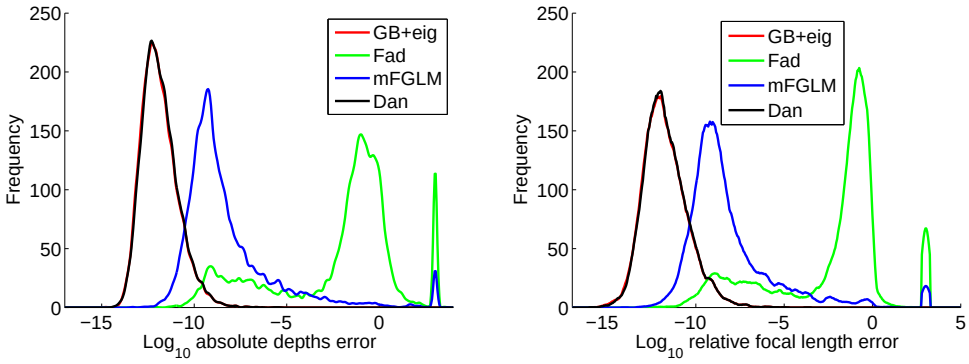


Figure 9.5: Numerical stability of studied P4P+f solvers. Peaks on the right hand side represent algorithm failure.

Figure 9.3 shows the comparison of several solvers to the 5-pt relative pose problem: GB+eig denotes the Gröbner basis "action matrix eigenvalue" solver [96], Nis - the Nisters solver [84], peig - the polynomial eigenvalue solver [65] , mFGLM - the solver based on the proposed "matrix FGLM" method, Fad - the solver based on Faddeev-Leverrier algorithm [31] and Dan - the solver based on the presented Danilevsky method.

The rotation error was measured as the angle in the angle axis representation of the relative rotation $\mathtt{R}\,\mathtt{R}_{gt}^{-1}$, where $\mathtt{R}_{gt}$ is the ground truth rotation and $\mathtt{R}$ the estimated rotation, and the translation error as an angle between the ground-truth and the estimated translation vectors.

All new fast solvers (mFGLM, Fad, Dan) behave competitively compared to the remaining solvers. Among these new solvers, the solver based on the Danilevsky method, Section 9.2.2, is the best in accuracy and as it will be seen later in speed too. This is true also for the remaining two tested problems, the 6-pt focal length an the P4P+f problem.

For the 6-pt focal length problem, we compared the Gröbner basis "action matrix eigenvalue" solution [64] (GB-eig), with the polynomial eigenvalue solution [65] (peig) and our two modifications of the Gröbner basis solver [64] using the modified "matrix FGML" method presented in Section 9.1.2 (mFGLM) and the Danilevsky method, Section 9.2.2 (Dan). We do not show plots for the Faddeev-Leverrier algorithm [31] since it was very unstable for this problem. Figure 9.4 shows that the "matrix FGML" method also suffers a little bit from numerical instability.

Finally, Figure 9.5 shows the comparison of several P4P+f solves, i.e. the Gröbner basis "action matrix eigenvalue" solver of size 86×96 from Section 6.4.2 (GB-eig), and our modifications of this solver using the modified "matrix FGML" method, Section 9.1.2 (mFGLM), the Faddeev-Leverrier algorithm [31] (Fad) and the Danilevsky method, Section 9.2.2 (Dan). It is clear that the Faddeev-Leverrier method suffers from a large numerical instability and cannot be used for solving this problem. The "matrix FGML" method failed several times, which is visible on the right hand side of both plots. The best results with respect to speed and numerical precision are again obtained using the Danilevsky method.

9.5.2 Speedup

In this experiment we are focusing on achieved speedup. We reimplemented most of the competing solvers in C++ and used the same math libraries in all tests. We omitted solvers which are known to be slower since our main focus is on speed. We have used Sturm sequences [49] to find real roots of a single-variable polynomial in all new solvers based on the "matrix FGML" method and the characteristic polynomial method. No special optimization, e.g. CPU intrinsic such as SSE were used. All tests were performed on Intel i7 Q720 1.6Ghz notebook.

Table 9.1 shows results for the 5-point relative pose problem:

Nister	GB+eig	Faddeev	mFGML	Danilevsky
10.6μs	61.2μs	17.2μs	13.7μs	14.2μs

Table 9.1: Speed comparison of the 5-pt relative pose solvers.

Our reimplementation of the Nister's algorithm [84] is the fastest for the 5-pt relative pose problem. It is because in this solution almost all computations can be done in a closed form. Our reimplementation of the Gröbner basis "action matrix eigenvalue" solver [64] (GB+eig) which uses a standard eigenvalue method [87] is almost $6\times$ slower due to eigenvalue and eigenvector computations. By replacing this eigenvalue computations with either the proposed "matrix FGML" method or the characteristic polynomial calculation using the Danilevsky method presented in Section 9.2.2 we achieved more than $4\times$ speed up.

Table 9.2 shows results for the 6-pt focal length problem. It can be seen that replacing the eigenvalue computations in the Gröbner basis solver [64] with the "matrix FGLM" method or the Danilevsky method resulted in almost $8\times$ speed up. Note that the "matrix FGML" algorithm is a little bit less stable compared to the remaining algorithms.

GB+eig	mFGML	Danilevsky
176.3μs	21.3μs	22.6μs

Table 9.2: Speed comparison of the 6-pt focal length solvers.

In this case we do not provide results for the Faddeev-Leverrier algorithm [31] because its numerical stability is poor and it failed to deliver any result most of the time.

Finally, the last Table 9.3 shows results for the absolute pose problem with unknown focal length.

GB+eig	sparse GB	Faddeev	mFGML	Danilevsky
127.4μs	82.9μs	53.3μs	48.5μs	48.1μs

Table 9.3: Speed comparison of the P4P+f solvers.

We also obtained significant speedup over the existing Gröbner basis "eigenvlaue action matrix" algorithm [12] (GB+eig). Here sparse GB corresponds to the Gröbner basis "eigenvlaue

action matrix" algorithm [12] with the sparse G-J elimination. Again the algorithm based on the Danilevsky method outperforms all the remaining algorithms both in numerical stability and speed.

The most efficient publicly available solvers of above problems we found at Hartley's web page [48] and minimal solvers site [63]. Hartley's implementation of Nister 5-pt algorithm solution runs 13.1μs. Hartley proposed a specialized 5-pt and 6-pt running 30.7μs and 112.7μs.

9.6 Conclusion

In this chapter we presented several methods for speeding up minimal solvers based on Gröbner basis and action matrix computation. We have shown that such solvers can be significantly sped up by replacing the eigenvalue computations with the computation of the characteristic polynomial of an action matrix followed by the calculation of its roots using Sturm-sequences. We demonstrated how effective the proposed methods are on three important computer vision problems.

10 Conclusion

In this thesis we studied the problem of estimating absolute pose of a camera from a minimal number of point correspondences between 2D and 3D space. We focused on solutions from a minimal number of point correspondences, that means, from the smallest number of points which are needed to actually solve the problem while using all possible constraints arising from the particular problem. We have shown how to reduce the number of point correspondences and how to create more efficient solvers when additional information about the scene is available. Problems we solved belong to the class of problems known as the Perspective-n-Point (PnP) problems.

For all problems we assumed a central camera, i.e. the pinhole camera model [47]. For some problems we assume that rays passing through the camera optics could be deformed by optics. As a mathematical model describing the distortion effect, we have chosen the division model [38]. The geometry of cameras, the projection and the scene invariants we are using have been known before. However, we have shown that the proper formulation of the problem is important. For example there exist many solutions for to P3P problem for a calibrated camera based on the cosine law constraints, but these solutions are very difficult to extend to cameras with unknown focal length. We have also shown that our own formulation of the P3P problem is a good starting point for further extensions to a more general camera models.

Our goal was to explain every problem we solved, describe basic relations, invariants and formulate the problem using polynomials. In Chapter 4 we described methods we used to solve underlying systems of polynomial equations. It is out of the scope of this thesis to describe these methods in detail and reader should refer to [62, 26, 27] where detailed description of the methods we used is provided. However, we sometimes included examples which should be useful to explain the methods or to catch the idea of the solution. For a more complicated systems of polynomial equations we show how to create a solver using the automatic generator [64], which we developed.

In Chapter 5 we described one of the oldest camera pose problem, the problem of estimation pose of a calibrated camera from three point correspondences. Besides the formulation based on the cosine law, which was used in the past, we provided three completely new formulations based on distances and ratios of distances and a very simple direct solution based on the relationship between a plane in 3D space and its projection. Using Euclidean rigidity constraint we derived simple systems of polynomial equations and we have shown how to solve these equations using the automatic generator [64]. For the solution based on a homography mapping, we have shown how the hidden variable method can be elegantly used to solve the camera matrix directly.

In Chapter 6 we have shown how easily can be our formulation of P3P problem extended to a camera with unknown focal length. This problem was not solved before for general scenes

and can be solved from four point correspondences between 2D and 3D space. Moreover, four points over-constraint this problem. This chapter shows that different invariants and different selection of equations greatly influence the quality of the result when noise is introduced into measurements. We presented several formulations of the problem, used different methods to solve underlying polynomial equations, created several thousands of solvers and extensively analyzed all these solvers in many experiments. In this chapter we have also shown step-by-step how to create a script for our automatic generator [64] which can be used to generate a source code of the solver. We have further shown how to setup the automatic solver generator [64] to create the most accurate and fast solvers.

In Chapter 7 we focused on a camera with unknown focal length and unknown radial distortion. The division model [38] was adopted for modeling the radial distortion. This problem can be solved using four point correspondences between 2D and 3D space too. We presented the first efficient solution to this problem by treating planar and non-planar scenes separately by specialized solvers. We have shown where to use which solver and how to create a generalized solver by combining these specialized solvers. Executing both the planar and the non-planar solver simultaneously and selecting the better result is much faster and numerically more stable compared to previous state-of-the-art solver by Josephson [56] for all scenes including near-planar ones.

Having some information about the camera vertical direction, obtained for example from gyroscopes, inertial measurement units (IMUs) or from a single vanishing point, can radically simplify the camera absolute pose problem. In Chapter 8 we developed completely new, very fast, efficient and numerically stable closed-form solutions for calibrated camera from two point correspondences and for radially distorted camera with unknown focal length from three points. Since still more and more cameras and smart devices are equipped with IMUs there is a great future potential in these simple solvers.

In Chapter 9 we presented several methods for speeding up minimal solvers based on Gröbner basis and action matrix computation. We showed that such solvers can be significantly sped up by replacing the eigenvalue computations with the computation of the characteristic polynomial of an action matrix followed by the calculation of its roots using Sturm-sequences. We have also shown the relationship between the standard FGLM algorithm [34], an action matrix and its characteristic polynomial.

The thesis contributes to solving not previously solved camera absolute pose problems. In particular, we describe general solution to the problem of estimating absolute pose of a camera which is calibrated up to an unknown focal length from four points, we developed a new more efficient solution to the problem for camera with unknown focal length and radial distortion. Further we solved previously unsolved absolute pose problems for a camera with known vertical direction from two point correspondences for a calibrated camera and from three for radially distorted camera with an unknown focal length. We demonstrated the quality and effectiveness of all proposed solvers on synthetic and real data. Finally we show how to create fast and efficient solvers by removing the eigenvalue and eigenvector completion.

Bibliography

[1] 2D3. Boujou - `http://www.2d3.com`.

[2] Y. I. Abdel-Aziz and H. M. Karara. Direct linear transformation from comparator coordinates into object space coordinates in close-range photogrammetry. In *Proceedings of the Symposium on Close-Range photogrammetry*, volume 1, page 18, 1971.

[3] M. A. Abidi and T. D. Chandra. A new efficient and direct solution for pose estimation using quadrangular targets: algorithm and evaluation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(5):534–538, May 1995.

[4] A. Akbarzadeh, J.-M. Frahm, P. Mordohai, B. Clipp, C. Engels, D. Gallup, P. Merrell, M. Phelps, S. Sinha, B. Talton, L. Wang, Q. Yang, H. Stewenius, R. Yang, G. Welch, H. Towles, D. Nister, and M. Pollefeys. Towards urban 3d reconstruction from video. In *3DPVT'06*, pages 1–8, June 2006.

[5] M.-A. Ameller, B. Triggs, and L. Quan. Camera Pose Revisited – New Linear Algorithms, 2000. Submitted to ECCV'00.

[6] A. Ansar and K. Daniilidis. Linear pose estimation from points or lines. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(5):578–589, May 2003.

[7] K. S. Arun, T. S. Huang, and S. D. Blostein. Least-squares fitting of two 3-d point sets. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-9(5):698–700, September 1987.

[8] J. L. Awange and Erik W. Grafarend. Groebner-basis solution of the three-dimensional resection problem. *Journal of Geodesy*, 77:327337, 2003.

[9] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. Van Der Vorst. *Templates for the solution of algebraic eigenvalue problems: a practical guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[10] J.-C. Bazin, S. Yongduek, C. Demonceaux, P. Vasseur, K. Ikeuchi, K. Inso, and M. Pollefeys. Globally optimal line clustering and vanishing point estimation in manhattan world. In *CVPR'12*, pages 638–645, June 2012.

[11] R. C. Bolles, P. Torr, D. Nister, J. Matas, O. Chum, and C. Stewart. 25 years of ransac. *Workshop in conjunction with CVPR'06*, June 2006.

[12] M. Bujnak, Z. Kukelova, and T. Pajdla. A general solution to the p4p problem for camera with unknown focal length. In *CVPR'08*, 2008.

[13] M. Bujnak, Z. Kukelova, and T. Pajdla. 3d reconstruction from image collections with a single known focal length. In *ICCV'09*, pages 1803–1810, 2009.

[14] M. Bujnak, Z. Kukelova, and T. Pajdla. New efficient solution to the absolute pose problem for camera with unknown focal length and radial distortion. In *ACCV'10*, pages 11–24, 2010.

[15] M. Bujnak, Z. Kukelova, and T. Pajdla. Efficient solutions to the absolute pose of cameras with unknown focal length and radial distortion by decomposition to planar and non-planar cases. *IPSJ Transactions on Computer Vision and Applications(CVA)*, 4:78–86, May 2012.

[16] M. Bujnak, Z. Kukelova, and T. Pajdla. Making Minimal Solvers Fast. In *CVPR'12*, 2012.

[17] M. Byröd. *Numerical Methods for Geometric Vision: From Minimal to Large Scale Problems*. PhD thesis, Centre for Mathematical Sciences LTH, Lund University, Sweden, 2010.

[18] M. Byröd, M. Brown, and K. Åström. Minimal solutions for panoramic stitching with radial distortion. In *BMVC'09*, 2009.

[19] M. Byröd, K. Josephson, and K. Åström. Improving numerical accuracy of gröbner basis polynomial equation solvers. In *ICCV'07*, 2007.

[20] M. Byröd, K. Josephson, and K. Åström. A column-pivoting based strategy for monomial ordering in numerical gröbner basis calculations. In *ECCV'08*, 2008.

[21] Q. Chen, H. Wu, and T. Wada. Camera calibration with two arbitrary coplanar circles. In *ECCV'04*, pages 521–532, 2004.

[22] K. Choi, S. Lee, and Y. Seo. A branch-and-bound algorithm for globally optimal camera pose and focal length. *Image and Vistion Computing*, 28(9):1369–1376, September 2010.

[23] O. Chum, J. Matas, and J. Kittler. Locally optimized ransac. In *DAGM-Symposium'03*, pages 236–243, 2003.

[24] E. Church. Revised geometry of the aerial photograph. In *Bulletin 15*, Syracuse, NY, 1945. Syracuse University Press.

[25] E. Church. Theory of photogrammetry. In *Bulletin 19*, Syracuse, NY, 1948. Syracuse University Press.

[26] D. Cox, J. Little, and D. O'Shea. *Using Algebraic Geometry*. Springer, 2nd edition, 2005.

[27] D.A. Cox, J. Little, and D.O'Shea. *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra*. Undergraduate Texts in Mathematics. Springer, 2010.

[28] A. M. Danilevskii. The numerical solution of the secular equation (russian). *Matem. sbornik*, 44:169–171, 1937. In Russian.

[29] D. F. Dementhon and L. S. Davis. Model-based object pose in 25 lines of code. *International Journal of Computer Vision*, 15:123–141, 1995.

[30] M. Dhome, M. Richetin, J. T. Lapreste, and G. Rives. Determination of the attitude of 3d objects from a single perspective view. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(12):1265–1278, December 1989.

[31] D.K. Faddeev and V.N. Faddeeva. *Computational methods of linear algebra*. W. H. Freeman, San Francisco, California, USA, 1963.

[32] O. D. Faugeras and M. Hebert. The representation, recognition, and locating of 3d objects. *International Journal of Robotics Research*, 5(3):27–52, September 1986.

[33] O. D. Faugeras and G. Toscani. Camera calibration for 3d computer vision. In *International Workshop on Machine Vision and Machine Intelligence*, pages 240–247, February 1987.

[34] J. Faugère, P. Gianni, D. Lazard, and F. Mora. Efficient computation of zero-dimensional gröbner bases by change of ordering. *Journal of Symbolic Computation*, 16(4):329–344, October 1993.

[35] S. Finstenvalder and W. Scheufele. Das rückwärtseinschneiden im raum. *Zum 75-Geburtstage Verlag Herbert Wichmann*, pages 86–100, 1937. Berlin, Germany.

[36] P. D. Fiore. Efficient linear solution of exterior orientation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(2):140–148, February 2001.

[37] M.A. Fischler and R.C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, June 1981.

[38] A. W. Fitzgibbon. Simultaneous linear estimation of multiple view geometry and lens distortion. In *CVPR'01*, volume 1, page 125, Los Alamitos, CA, USA, 2001. IEEE Computer Society.

[39] Quartic function. `http://en.wikipedia.org/wiki/Quartic_function`.

[40] X.S. Gao, X.R. Hou, J. Tang, and H.F. Cheng. Complete solution classification for the perspective-three-point problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(8):930–943, August 2003.

[41] E. Grafarend, P. Lohse, and B. Schaffrin. Dreidimensionaler rückwärtsschnitt, teil 1: Die projektiven gleichungen. In *Zeitschrift für Vermessungswesen*, page 137. Geodätisches Institut, Universität Stuttgart, 1989.

[42] J. A. Grunert. Das pothenot'sche problem, in erweiterter gestalt, nebst bemerkungen über seine anwendung in der. In *Archiv der Mathematik und Physik*, page 238248, 1841.

[43] Y. Guo. A novel solution to the p4p problem for an uncalibrated camera. *Journal of Mathematical Imaging and Vision*, pages 1–13, 2012.

[44] R. M. Haralick, Daniel D. Lee, K. Ottenburg, and Michael Nolle. Analysis and solutions of the three point perspective pose estimation problem. In *CVPR'08*, pages 592–598, June 1991.

[45] R. M. Haralick and L. G. Shapiro. *Computer and Robot Vision*. Addison Wesley Longman Publishing Co., Inc., 1st edition, 1992.

[46] R. Hartley and H. Li. An efficient hidden variable approach to minimal-case camera motion estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, preprint, 2012.

[47] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2nd edition, 2004.

[48] Richar Hartley. `http://users.cecs.anu.edu.au/~hartley/`.

[49] D. G. Hook and P. R. McAree. Graphics gems. In Andrew S. Glassner, editor, *Graphics gems*, chapter Using Sturm sequences to bracket real roots of polynomial equations, pages 416–422. Academic Press Professional, Inc., San Diego, CA, USA, 1990.

[50] R. Horaud, B. Conio, O. Leboulleux, and B. Lacolle. An analytic solution for the perspective 4-point problem. *Computer Vision, Graphics, and Image Processing*, 47(1):33–44, 1989.

[51] R. Horaud, F. Dornaika, and B. Lamiroy. Object pose: The link between weak perspective, paraperspective, and full perspective. *International Journal of Computer Vision*, 22(2):173–189, 1997.

[52] B. K. P. Horn. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America*, 4(4):629–642, April 1987.

[53] A.S. Householder. *The theory of matrices in numerical analysis*. Blaisdell book in the pure and applied sciences. Blaisdell Pub. Co., 1964.

[54] Z. Y. Hu, C. Lei, and F. C. Wu. A short note on p4p problem. *Acta Automatica Sinica*, 27(6):770–776, 2001.

[55] Z. Y. Hu and F. C. Wu. A note on the number of solutions of the noncoplanar p4p problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(4):550–555, April 2002.

[56] K. Josephson and M. Byröd. Pose estimation with radial distortion and unknown focal length. In *CVPR'09*, 2009.

[57] M. Kalantari, A. Hashemi, F. Jung, and J. Pi. Guedon. A new solution to the relative orientation problem using only 3 points and the vertical direction. *Journal of Mathematical Imaging and Vision*, 39(3):259–268, March 2011.

[58] L. Kneip, D. Scaramuzza, and R. Siegwart. A novel parametrization of the perspective-three-point problem for a direct computation of absolute camera position and orientation. In *CVPR'11*, pages 2969–2976, June 2011.

[59] D. E. Knuth. *The Art of Computer Programming: Seminumerical Algorithms*, volume 2. Addison Wesley Longman Publishing Co., Inc., 2nd edition, 1981.

[60] K. Kühnle and E.W. Mayr. Exponential space computation of gröbner bases. In *ISSAC'96*, pages 63–71, 1996.

[61] J. B. Kuipers. *Quaternions and Rotation Sequences: A Primer with Applications to Orbits, Aerospace and Virtual Reality*. Princeton University Press, August 2002.

[62] Z. Kukelova. *Algebraic methods in computer vision*. PhD thesis, Center for Machine Perception, Czech Technical University, Prague, Czech republic, 2012.

[63] Z. Kukelova, M. Bujnak, and T. Pajdla. Minimal problems in computer vision - `http://cmp.felk.cvut.cz/minimal/`.

[64] Z. Kukelova, M. Bujnak, and T. Pajdla. Automatic generator of minimal problem solvers. In *ECCV'08, Part III*, volume 5304 of *Lecture Notes in Computer Science*, pages 302–315, 2008.

[65] Z. Kukelova, M. Bujnak, and T. Pajdla. Polynomial eigenvalue solutions to the 5-pt and 6-pt relative pose problems. In *BMVC'08*, 2008.

[66] Z. Kukelova, M. Bujnak, and T. Pajdla. Closed-form solutions to the minimal absolute pose problems with known vertical direction. In *ACCV'10*, 2, pages 216–229, 2010.

[67] Z. Kukelova, M. Bujnak, and T. Pajdla. Polynomial eigenvalue solutions to minimal problems in computer vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(7):1381–1393, 2012.

[68] Z. Kukelova and T. Pajdla. A minimal solution to the autocalibration of radial distortion. In *CVPR'07*, 2007.

[69] R. Kumar and A. R. Hanson. Robust methods for estimating pose and a sensitivity analysis. *Computer Vision, Graphics, and Image Processing: Image Understanding*, 60(3):313–342, 1994.

[70] B. Leibe, N. Cornelis, K. Cornelis, and L. Van Gool. Dynamic 3d scene analysis from a moving vehicle. In *CVPR'07*, 2007.

[71] B. Leibe, K. Schindler, and L. Van Gool. Coupled detection and trajectory estimation for multi-object tracking. In *ICCV'07*, 2007.

[72] H. Li. A simple solution to the six-point two-view focal-length problem. In *ECCV'06 - Part IV*, pages 200–213, 2006.

[73] H. Li and R. Hartley. Five-point motion estimation made easy. In *ICPR'06*, volume 1, pages 630–633, 2006.

[74] R. Lidl and H. Niederreiter. *Finite fields*. Cambridge University Press, 1983.

[75] S. Linnainmaa, D. Harwood, and L. S. Davis. Pose determination of a three-dimensional object using triangle pairs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(5):634–647, September 1988.

[76] D. G. Lowe. Fitting parameterized three-dimensional models to images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(5):441–450, May 1991.

[77] C. Lu, G. D. Hager, and E. Mjolsness. Fast and globally convergent pose estimation from video images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(6):610–622, June 2000.

[78] D. Manocha. Solving systems of polynomial equations. *IEEE Computer Graphics and Applications*, 14:46–55, 1994.

[79] D. Manocha and J. F. Canny. Multipolynomial resultant algorithms. *Journal of Symbolic Computation*, 15:99–122, 1993.

[80] D. Martinec and T. Pajdla. Robust rotation and translation estimation in multiview reconstruction. In *CVPR'07*, 2007.

[81] E. Merritt. Explicit three-point resection in space. *Photogrammetric Engineering*, 15(4):649655, 1949.

[82] F. Moreno-Noguer, V. Lepetit, and P. Fua. Accurate non-iterative o(n) solution to the pnp problem. In *ICCV'07*, pages 1–8, October 2007.

[83] O. Naroditsky and K. Daniilidis. Optimizing polynomial solvers for minimal geometry problems. In *ICCV'11*, pages 975–982, 2011.

[84] D. Nistér. An efficient solution to the five-point relative pose problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(6):756–777, June 2004.

[85] D. Oberkampf, D. F. Dementhon, and L. S. Davis. Iterative pose estimation using coplanar feature points. *Computer Vision and Image Understanding*, 63(3):495–511, May 1996.

[86] American Society of Photogrammetry, C.C. Slama, C. Theurer, and S. W. Henriksen. *Manual of photogrammetry*. American Society of Photogrammetry, 1980.

[87] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 3rd edition, 2007.

[88] L. Quan and Z. Lan. Linear n-point camera pose determination. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(8):774–780, August 1999.

[89] G. Reid, J. Tang, and L. Zhi. A complete symbolic-numeric linear method for camera pose determination. In *ISSAC'03*, pages 215–223, 2003.

[90] G. H. Rosenfield. The problem of exterior orientation in photogrammetry. *Photogrammetric Engineering*, pages 536–553, 1959.

[91] G. Schweighofer and A. Pinz. Robust pose estimation from a planar target. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(12):2024–2030, 2006.

[92] D. E. Smith. *History of modern mathematics*, volume 1 of *Mathematical Monographs*. Ithaca, New York: Cornell University Library, 4th edition, 1906.

[93] N. Snavely, S. M. Seitz, and R. Szeliski. Modeling the world from internet photo collections. *International Journal of Computer Vision*, 80(2):189–210, 2008.

[94] M. Sonka, V. Hlavac, and R. Boyle. *Image processing, analysis and machine vision*. Thomson-Engineering, 3rd edition, 2008.

[95] H. J. Stetter. *Numerical Polynomial Algebra*. Society for Industrial and Applied Mathematics, 2004.

[96] H. Stewénius, C. Engels, and D. Nistér. Recent developments on direct relative orientation. *ISPRS Journal of Photogrammetry and Remote Sensing*, 60(4):284–294, 2006.

[97] H. Stewénius, D. Nistér, F. Kahl, and F. Schaffalitzky. A minimal solution for relative pose with unknown focal length. *Image and Vision Computing*, 26(7):871–877, July 2008.

[98] I. E. Sutherland. Three-dimensional data input by tablet. *Proceedings of the IEEE*, 62(4):453–461, April 1974.

[99] E.H. Tompson. The projective theory of relative orientation. *Photogrammetria*, pages 67–75, 1968.

[100] B. Triggs. Camera pose and calibration from 4 or 5 known 3d points. In *ICCV'99*, volume 1, page 278, 1999.

[101] R. Tsai. A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses. *IEEE Journal on Robotics and Automation*, 3(4):323–344, August 1987.

[102] Y. Wu and Z. Hu. Pnp problem revisited. *Journal of Mathematical Imaging and Vision*, 24:131–141, 2006.

[103] Xsens. `http://www.xsens.com`.

[104] Z. Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, November 2000.

[105] L. Zhi and J. Tan. A complete linear 4-point algorithm for camera pose determination. *Academy of Mathematics and System Sciences, Academia Sinica*, 21:239249, 2002.