Numerical Methods for Geometric Vision: From Minimal to Large Scale Problems

Byröd, Martin

2010

Numerical Methods for Geometric Vision: From Minimal to Large Scale Problems

Martin Byröd

LUND UNIVERSITY

Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

Mathematics
Centre for Mathematical Sciences
Lund University
Box 118
SE-221 00 Lund
Sweden

http://www.maths.lth.se/

Doctoral Theses in Mathematical Sciences 2010:4
ISSN 1404-0034

ISBN ISBN 978-91-628-8091-0
LUTFMA-1041-2010

Printed in Sweden by Media-Tryck, Lund 2010

Preface

The central theme of this thesis is the numerical solution of problems in geometric computer vision. Given only a sequence of images of a scene we would like to infer the motion of the observer and the three dimensional layout of the scene. In applications, such problems come in a wide variety of sizes from setups with only a couple of cameras and a handful 3D points to scenarios with thousands of images and millions of 3D points. During the first part of this PhD I studied one extreme of this spectrum, where one asks for the absolute minimum information needed to solve particular instances of this problem, so called minimal configurations. These minimal problems typically lead to systems of polynomial equations that need to be solved. One conclusion of our work in this area is that the central computational tool for dealing with polynomial systems is numerical linear algebra. The contributions on this topic in essence deal with how to bring out the full potential of these tools.

Towards the end of my graduate studies I have investigated the other end of the spectrum; trying to push the state of the art in terms of how large problems we can handle on modern hardware. Large scale problems cannot be solved exactly and estimating 3D structure and camera locations thus turns into an estimation problem using noisy measurements, usually in the form of a non-linear least squares problem. Again it turns out that success or failure is largely in the hands of numerical linear algebra.

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

Main Papers

[12] M. Byröd, K. Åström, Conjugate Gradient Bundle Adjustment, *Submitted*, 2010.

[54] Z. Kukelova, M. Byröd, K. Josephson, T. Pajdla, K. Åström, Fast and robust numerical solutions to minimal problems for cameras with radial distortion, *Computer Vision and Image Understanding*, 2010.

[11] M. Byröd, K. Åström, Bundle Adjustment using Conjugate Gradients with Multiscale Preconditioning, *Proc. British Machine Vision Conference (BMVC)*, London, UK, 2009.

[13] M. Byröd, M. Brown, K. Åström, Minimal Solutions for Panoramic Stitching with Radial Distortion, *Proc. British Machine Vision Conference (BMVC)*, London, UK, 2009.

[47] K. Josephson, M. Byröd, Pose Estimation with Radial Distortion and Unknown Focal Length, *Proc. Conference on Computer Vision and Pattern Recognition (CVPR)*, Miami, Florida, USA, 2009.

[17] M. Byröd, K. Josephson, K. Åström, Fast and Stable Polynomial Equation Solving and its Application to Computer Vision, *in International Journal of Computer Vision*, 2009.

[16] M. Byröd, K. Josephson, K. Åström, A Column-Pivoting Based Strategy for Monomial Ordering in Numerical Gröbner Basis Calculations, *Proc. European Conference on Computer Vision (ECCV)*, Marseilles, France, 2008.

[18] M. Byröd, Z. Kukelova, K. Josephson, T. Pajdla, K. Åström, Fast and Robust Numerical Solutions to Minimal Problems for Cameras with Radial Distortion, *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Anchorage, Alaska, 2008.

[14] M. Byröd, K. Josephson, K. Åström, Fast Optimal Three View Triangulation, *Proc. Asian Conference on Computer Vision (ACCV)*, Tokyo, Japan, 2007.

[15] M. Byröd, K. Josephson, K. Åström, Improving Numerical Accuracy of Gröbner Basis Polynomial Equation Solvers, *Proc. International Conference on Computer Vision (ICCV)*, Rio de Janeiro, Brazil, 2007.

[48] K. Josephson, M. Byröd, F. Kahl, K. Åström, Image-Based Localization Using Hybrid Feature Correspondences, *ISPRS Workshop BenCOS at CVPR*, Minneapolis, USA, 2007.

Subsidiary Papers

[69] C. Olsson, M. Byröd, N. Overgaard, F. Kahl, Extending Continuous Cuts: Anisotropic Metrics and Expansion Moves, *Proc. International Conference on Computer Vision (ICCV)*, Kyoto, Japan, 2009.

[68] C. Olsson, M. Byröd, F. Kahl, Globally Optimal Least Squares Solutions for Quasiconvex 1D Vision Problems, *Proc. Scandinavian Conference on Image Analysis (SCIA)*, Oslo, Norway, 2009.

Organization

In summary, the thesis contains contributions in three main directions and is correspondingly divided into three different parts reflecting this.

Part I: Solving Polynomial Equations

Theoretical and algorithmic contributions are made which link algebraic geometry and numerical linear algebra. Algebraic geometry provides the theoretical foundations for dealing with multivariate polynomials. We extract the parts of this theory which deal specifically with solving systems of polynomial equations and rephrase it in the language of matrix operations. This provides certain freedom, which has allowed more creativity in the design of numerical algorithms.

Part II: Applications of Polynomial Equation Solving in Computer Vision

Here, the techniques from part I are put to use in practice. We consider classical problems in computer vision: triangulation, pose estimation, relative orientation and panoramic stitching, but with some new twists which require more advanced polynomial solving techniques than have previously been available. The material contains a mixture of novel contributions as well as more straightforward case studies. Where possible, we have tried to integrate these new solutions to classical problems in complete systems to evaluate the usefulness of the new methods in a practical context.

Part III: Bundle Adjustment

The material in this part of the thesis is of a different character than that of the preceding chapters. We consider large scale bundle adjustment, which refers to estimation of cameras and 3D points in a non-linear least squares framework. In other words, the topic is large scale unconstrained optimization. In large scale geometric estimation problems, bundle adjustment is typically a major computational bottleneck. Our basic line of attack here is to substitute the direct linear solver (Cholesky factorization) in standard the Gauss-Newton algorithm by iterative and approximate solvers in the form of conjugate gradient methods. This leads to difficult questions of preconditioning etc.

Acknowledgements

I would like to acknowledge the contribution of a number of people to the completion of this thesis. First of all, I would like to thank my supervisor, Kalle Åström, who has been a pleasure to work with these years; always enthusiastic, wise, friendly and ready to answer any question at any time. Kalle is one of the brightest people I know and also a great source of energy. A thank you also goes to the rest of the Mathematical Imaging Group and in particular to Klas Josephson for energy, ideas and collaborative efforts in a large part of my work as a PhD student. I would also like to thank my other co-authors in Lund: Fredrik Kahl, Carl Olsson and Niels Christian Overgaard.

Furthermore I would like to thank Zuzana Kukelova and Tomas Pajdla at the Czech Technical University in Prague for interesting discussions, useful comments and collaboration on two papers. I would also like to thank Matthew Brown at Ecole Polytechnique Fédérale de Lausanne, Switzerland for collaboration on one paper. A thank you also goes to Sameer Agarwal, Noah Snavely, Drew Steedly and Ni Kai for generously sharing their data.

Finally, I would like to thank our very helpful secretary Ann-Kristin Ottosson for aid in countless practical issues and of course lots of thanks to friends and family for everything else.

Contents

Chapter 1

Introduction

The general field of computer vision deals with the problem of making a computer "see". This might mean many things including recognizing objects, places and people. This thesis deals with the sub-field of geometric computer vision where one tries to extract geometric information about the world and the observer from a sequence of images. Estimation of scene structure and camera motion using only image data has been one of the central themes of research in photogrammetry, geodesy and computer vision. It has important applications within robotics, architecture, the movie industry, photography etc.

Given a set of images of a scene, it is possible to compute the 3D structure of the scene and the relative positions of the cameras. Examples of such reconstructions are shown in Figures 1.1 and 1.2. Initial reconstructions and outlier removal were in those cases performed using an algorithm by Enqvist *et al*, which at the time of writing has not yet been published, and the result was fine tuned using bundle adjustment as described in Part III of this thesis.

If all images were taken from the same point in space, it is possible to warp and align them to create one panoramic image. An example of this is shown in Figure 1.3. Read more about this in Chapter 9.

Two different images of a planar surface are related by a *homography* which can be described by a 3×3 matrix. An example is given in Figure 1, where a poster has been photographed from two different angles. Using feature matching techniques [61] it is possible to find potential corresponding points between the images, but such matches will contain many errors. Using the knowledge that the images should be related by a homography, we can look for such a transformation which agrees with a large number of the detected point matches. The remaining pairs of points which do not agree with this transformation can then be discarded as errors of the matching process.

The majority all of modern methods for the type of geometric computations described above can be broken down into two main parts: First point wise correspondence is established across views in the spirit of the homography example illustrated in Figure 1. This is largely a recognition and reasoning task. In the second step, we assume that image points are linked across views and this information is then used to infer the motion of the cameras and the locations of the points in 3D.

During the last decades, a very active research community has solved many problems in this field, but much remains to be done. Throughout this thesis,

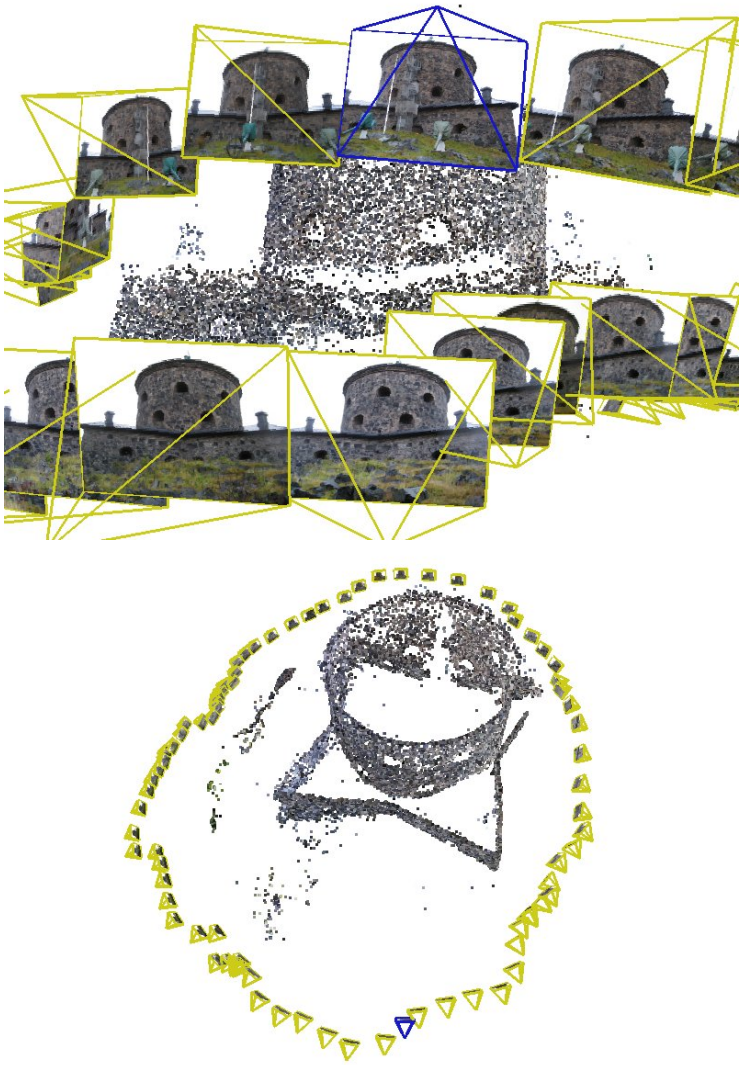

Figure 1.1: Reconstructed cameras and 3D points from images of Skansen Lejonet in Gothenburg.

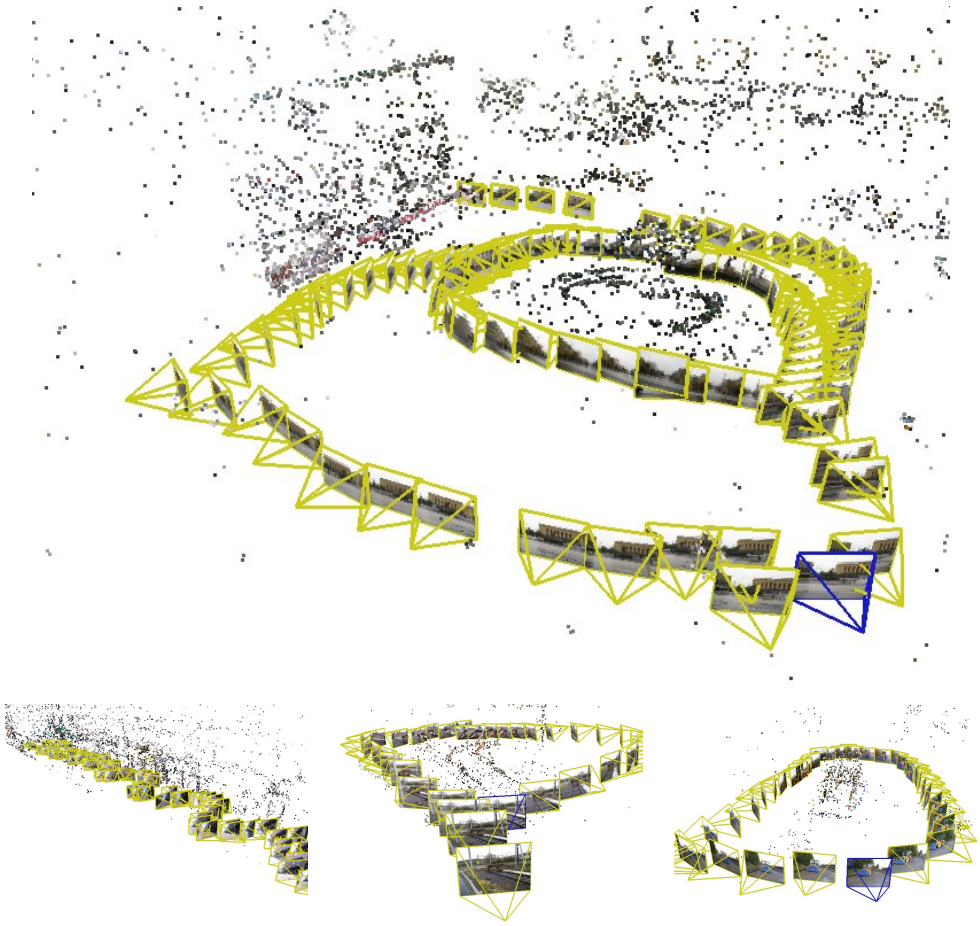

Figure 1.2: Top: Reconstruction from Götaplatsen in Gothenburg. Bottom: Reconstructions from Lilla fiskaregatan and two other locations in Lund.

Figure 1.3: Top: Nine images of a canal in Amsterdam. Bottom: All images were taken from the same point in space. Hence by computing the relative rotations between the views it was possible to project them onto an enclosing cylinder which could then be cut and unfolded into a single panoramic image.

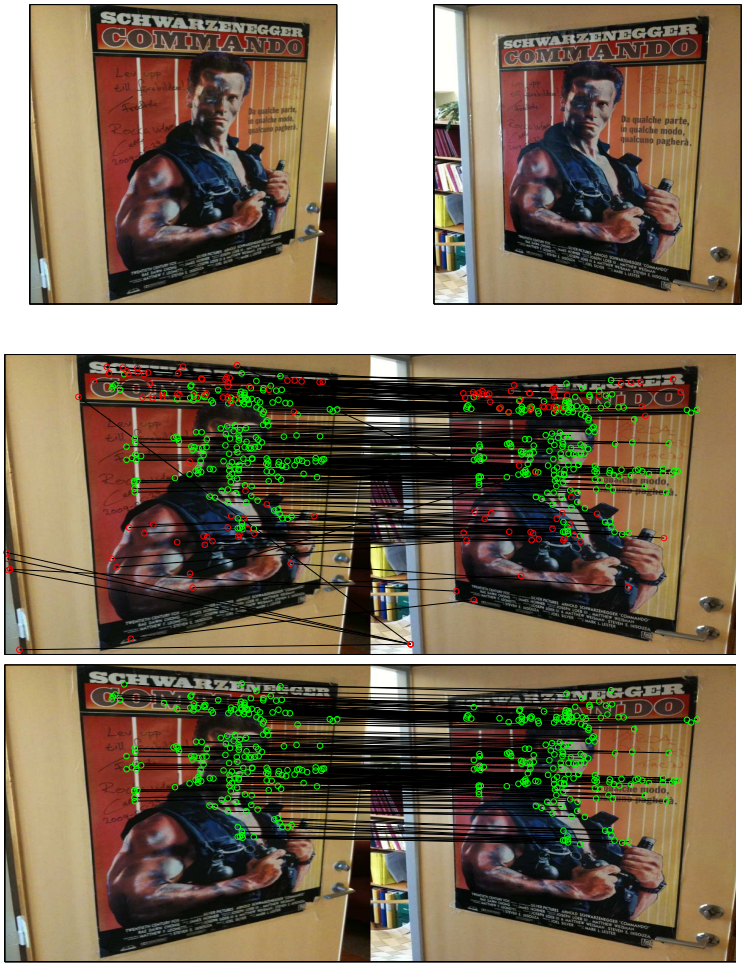

Figure 1.4: Top: Two photographs of a poster taken from different viewpoints. Since the majority of the scene is planar, the images are related by a homography. Middle: Potential matches have been found using feature matching techniques. Some are correct (green) and some are false (red). Bottom: Using the knowledge that the points should be related by a homography, we can detect and remove false matches.

we focus on the second of the two sub-problems mentioned above; We assume that point-correspondences across views are given and the question is thus how geometric information should be computed based on such data. In the thesis we consider two quite different settings. In the first, the number of constraints exactly matches the number of unknowns (minimal problems). With the right formulation, these setups can often be efficiently solved in a single step. In the second setting, we consider large scale problems with thousands of cameras and millions of 3D points. These two different settings have lead us to study two particular areas of applied mathematics: The numerical solution of systems of polynomial equations and sparse non-linear least squares problems.

1.1 Contributions

This section gives an overview of the contents of the thesis with a focus on the scientific contributions by the author and co-authors. Chapters 1, 2, 3, 11 and 14 are omitted in this overview since they consist mainly of background material and general discussions. I have collaborated with my supervisor Kalle Åström on all papers except [47] on pose estimation with radial distortion. I have collaborated with Klas Josephson on all papers except [13] on panoramic stitching and [11, 12] on bundle adjustment. For each of the contributions below, the author who took the first initiative in general also took a leading role in developing theory and algorithms for that contribution and is hence listed as first author.

Part I: Solving Polynomial Equations

Chapter 4 The main part of this chapter is based on [17]. Theoretical developments regarding systems of polynomial equations are made which are then used in Chapter 5 to derive practical algorithms for solving such systems. The concept of a solving basis is introduced and it is shown how a solving basis may be computed and used to solve a system of polynomial equations. With the new developments we no longer need strictly defined monomial orderings and proper Gröbner bases to solve a system of equations. The advantage of this is a larger freedom in how to design efficient numerical algorithms.

Chapter 5 Based on the publications [14, 15, 16], we introduce the *redundant solving basis method* and the *SVD* and *QR* methods for polynomial equation solving. In hindsight, the SVD and QR methods are quite similar. However, the more complicated SVD method was actually discovered first and it was only with the development of the cleaner QR method that we understood how to formulate these algorithms in terms of matrix operations. This cleaner formulation also made the close connection between these methods much more clear.

Part II: Applications of Polynomial Equation Solving in Computer Vision

Chapter 6 In this chapter we give a practical solution to L_2 optimal triangulation from three views. The problem was previously solved in [80], but due to numerical difficulties extremely slow emulated 128-bit numerics had to be used

which rendered that algorithm useless for any practical purposes. This chapter is based on [14].

Chapter 7 We study hybrid pose / relative pose estimation based on a mixture of correspondences to other views and to known 3D points in a model. This leads to a range of different minimal cases of which two are given numerical solutions. The chapter is based on material from [48].

Chapter 8 Two minimal cases for relative orientation with partial calibration and unknown radial distortion are solved and evaluated. The results were obtained in collaboration with Zuzana Kukelova and Tomas Pajdla at the Czech Technical University in Prague and were previously published in [18, 54].

Chapter 9 A minimal solver for rotation, focal length and radial distortion from three point-correspondences is derived and implemented using polynomial techniques. The application is panoramic image stitching. We show that the solver yields an improvement compared to the state of the art when integrated in a complete stitching pipeline. The work was done in collaboration with Matthew Brown, then at the University of British Columbia [13].

Chapter 10 In this chapter we study pose estimation for the case of unknown focal length and radial distortion. An interesting result is that modeling radial distortion improves accuracy considerably even for a standard lens SLR camera. The results were first published in [47].

Part III: Bundle Adjustment

Chapter 12 The chapter presents a relatively straightforward approach to Bundle Adjustment using conjugate gradients. However, care has been taken to adapt the conjugate gradient algorithm to the particular case of bundle adjustment in order to bring out its full potential. In particular, we (i) use a variant of the conjugate gradient method which allows us to avoid forming J^TJ, where J is the Jacobian, (ii) we propose a block QR factorization preconditioner tailored to the sparsity structure of the bundle adjustment Jacobian and (iii) we note that the preconditioned system has "property A", which allows us to roughly cut the work per iteration in half. A reworked version of this chapter is currently under review for publication [12].

Chapter 13 Despite some care put into preconditioning in the previous chapter, the CG based bundle adjustment procedure of the previous chapter often shows disappointingly slow convergence near the optimum. Here we present some ideas of a more speculative nature that suggest how the bundle adjustment system might be preconditioned in a more sophisticated way using multiscale representations. Preliminary results based on these ideas have been published in [11]. These results actually predate those presented in Chapter 12, but were placed later in the thesis for a more logical progression of the material.

Chapter 2

Preliminaries

This chapter introduces some background knowledge to facilitate the understanding of the remainder of the thesis. We start by presenting the basics of geometric computer vision including the linear pin-hole camera and the fundamental and essential matrices. We then give some elements of algebraic geometry used for polynomial equation solving.

2.1 Geometric Computer Vision

The general field of computer vision deals with the problem of making a computer "see". This might mean many things including recognizing objects, places and people. This thesis deals with the subfield of geometric computer vision, where one tries to extract geometric information about the world (scene) and the observer (camera) from a sequence of two or more images. This forms the basis for many applications; Stereo, 3D reconstruction, panoramic stitching, augmented reality, robotics, etc. See [40] for a thorough introduction to the subject. The fundamental entity in this process is the *camera*, which needs to be modeled in some sensible manner. The most popular way of doing this is to adopt the central projection principle which yields the *pin-hole camera* model. In geometric language, the pin-hole camera consists of a camera center t and a plane π (the image plane) with a local coordinate system. Projection of a world point $\mathbf{X}$ is done by intersecting the ray from t through $\mathbf{X}$ with π and the projected point $\mathbf{x}$ is simply obtained as the intersection, see Figure 2.1.

We now choose coordinate system in the world so that the camera is at the origin and place the origin $\mathcal{O}$ of π so that the axis from the camera center to $\mathcal{O}$ is perpendicular to π and then align the camera axis with the world coordinate Z-axis producing the schematic setup illustrated in Figure 2.2.

With this setup, we can use the top-triangle theorem of Euclidean geometry to derive the projection of a world point $\mathbf{X} = [X,\ Y,\ Z]^T$. From Figure 2.2 we easily see that we get the image coordinates

$$\begin{aligned} x &= \frac{X}{Z}, \\ y &= \frac{Y}{Z}. \end{aligned} \tag{2.1}$$

Now consider the more general case with a camera center $t \neq \mathbf{0}$ and a camera axis which is not aligned with the Z-axis (but still intersects the origin of the

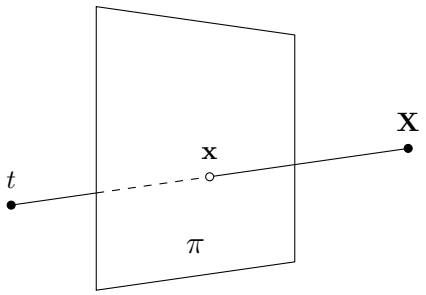

Figure 2.1: Projection of the point $\mathbf{X}$ onto the image plane π using the pin-hole camera model.

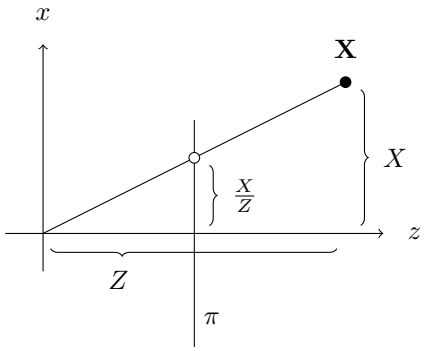

Figure 2.2: The setup with the camera axis aligned with the z-axis is convenient for deriving the central projection equations.

image plane). This can be brought back to the situation in Equation 2.1 by a Euclidean transformation

$$\mathbf{X}' = \begin{bmatrix} R & -Rt \end{bmatrix} \begin{bmatrix} \mathbf{X} \\ 1 \end{bmatrix}, \tag{2.2}$$

where R is the 3×3 rotation matrix which maps 3D points to the camera oriented coordinate system. In this coordinate frame we get $x = X'/Z'$ and $y = Y'/Z'$. We now switch to homogeneous coordinates which means that we extend the image coordinates $\mathbf{x}$ and the world coordinates $\mathbf{X}$ with a 1

$$u = \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}, \; U = \begin{bmatrix} \mathbf{X} \\ 1 \end{bmatrix}.$$

Denoting the matrix $[R \; -Rt]$ in equation 2.2 by P we thus get the familiar pin-hole projection equation

$$\lambda u = PU, \tag{2.3}$$

where the depth λ is now instead put on the left hand side.

2.1.1 The Calibration Matrix

The camera matrix P derived above is a 3×4 matrix with a special structure. If we let P be any 3×4 matrix we get a general projective camera. Using RQ

decomposition (QR decomposition ordered differently) we can write P as

$$P = K \begin{bmatrix} R & t \end{bmatrix}, \tag{2.4}$$

where R is an orthogonal matrix and K is upper triangular. It is common to write

$$K = \begin{bmatrix} f & fs & x_0 \\ 0 & f\gamma & y_0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{2.5}$$

where the parameters can then be interpreted as the focal length f, the principal point $[x_0, y_0]^T$ (the point of intersection between camera axis and image plane), the aspect ratio γ (scale ratio between the y-axis and the x-axis in the image) and the skew s which models non-orthogonal coordinate axes in the image. Of these parameters, the focal length f is the only parameter which is explicitly used in this thesis.

Typical assumptions are: (i) the camera is calibrated which means that K is known and we can then multiply the image coordinates with K^{-1} and assume that K is the identity matrix in (2.4), (ii) the camera is calibrated up to an unknown focal length f which means that we can assume

$$K = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{2.6}$$

or (iii) the camera is uncalibrated which means we have a general camera matrix P.

Algorithmically, as we will see, the uncalibrated case is often the easiest to work with since any partial or full calibration means that we have to introduce non-linear constraints which complicate the situation.

2.1.2 Epipolar Geometry

In the setting of two uncalibrated cameras P_1 and P_2, image coordinates x_1 and x_2 corresponding to a common world point X obey a bilinear constraint known as the epipolar constraint

$$x_1^T F x_2 = 0. \tag{2.7}$$

We work with homogeneous coordinates and F is thus a 3×3 matrix, which is known as the fundamental matrix and is uniquely determined by the camera matrices P_1 and P_2. An important property of F is that we always have $\det(F) = 0$. Conversely, any 3×3 matrix F with $\det(F) = 0$ is a fundamental matrix of some cameras P_1 and P_2.

Consider now two calibrated cameras P_1 and P_2 on the form (2.2). These uniquely determine a matrix E, called the essential matrix, which satisfies (2.7) for any corresponding points as well as $\det(E) = 0$. Moreover, since E is computed from two calibrated cameras it can be shown that the two nonzero singular values of E are equal which can be expressed as

$$2EE^TE - \mathrm{tr}(EE^T)E = 0, \tag{2.8}$$

known as the trace constraint for the essential matrix.

2.1.3 Structure from Motion

One of the main goals of geometric computer vision can be formulated as solving the *structure from motion* problem. The term structure from motion comes from the idea of inferring the structure (3D configuration) of an observed scene from the motion of a camera recorded as a sequence of images. In the general setting nothing is assumed to be known about the cameras. Algebraically formulated, we are given $m \cdot n$ image points $\mathbf{x}_{ij}$ captured by m unknown cameras P_i from n unknown world points $\mathbf{X}_j$. The problem is to determine the unknown cameras and world points satisfying

$$\lambda_{ij}\mathbf{x}_{ij} = P_i\mathbf{X}_j \tag{2.9}$$

for all i and j.

A typical structure from motion system consists of the following steps

1. Establish tentative point correspondences across views using some local patch descriptor. SIFT [61] is a popular choice. This step typically produces a large set of true as well as false correspondences.

2. Repeatedly compute fundamental/essential matrices from pairwise views using randomly selected small subsets of points and save the sets which are consistent with many of the other points (the RANSAC algorithm [30]).

3. Link matches from the previous steps between different pairs to form point tracks and incrementally or otherwise build a rough model by triangulating 3D points and adding new cameras using pose estimation.

4. Fine tune the reconstruction by employing a large scale optimization algorithm to minimize *e.g* the sum of squares of reprojection errors over all views for the points selected in the previous step.

The structure from motion problem is by no means solved and each of the steps above constitutes an active sub-field in its own right. The procedure above should only be seen as a rough sketch of how such a system works and there are other possible variations on this theme.

2.1.4 Minimal Problems

In the structure from motion system sketched in the previous section, step 2 involved computing camera geometries from small numbers of correspondences. The motivation for this is that a small set of correspondences is less likely to contain incorrect matches. It is therefore interesting to investigate what the minimal number of point correspondences is for a given geometric problem and to devise algorithms for solving them. Such problems are usually referred to as *minimal problems* or *minimal cases* and will occur frequently throughout this thesis.

Understanding the geometry and the number of solutions of minimal structure and motion problems has a long history. For instance, computing the fundamental matrix in the uncalibrated case requires a minimal set of seven points in two views and with this setup the problem has three solutions. This problem was studied and solved already in 1855, *cf* [20]. The corresponding calibrated case was in principle solved in 1913 [51] and later corrected by Demazure [24]. However, it was only recently that a practical numerical algorithm

for solving this problem was given [66, 77]. As mentioned above the study of minimal cases has got increased attention with its use in RANSAC algorithms to solve both for geometry and correspondence in numerous applications [40]. Briefly, the RANSAC procedure is as follows: Randomly sample a small subset of point-correspondences and estimate the geometric relations from them. Count how many of the remaining point-correspondences that are consistent with the estimated parameters. Repeat this until a parameter configuration has been found which agrees with enough observations. Keep these observations as inliers and discard the rest of the observations as outliers.

2.2 Algebraic Geometry for Equation Solving

In this section we review some of the classical theory of multivariate polynomials. We consider the following problem

Problem 1. Given a set of m polynomials $f_i(\mathbf{x})$ in s variables $\mathbf{x} = (x_1, \ldots, x_s)$, determine the complete set of solutions to

$$\begin{aligned} f_1(\mathbf{x}) &= 0, \\ &\vdots \\ f_m(\mathbf{x}) &= 0. \end{aligned} \tag{2.10}$$

We denote by V the zero set of (2.10). In general V need not be finite, but in this work we will only consider zero dimensional V, *i.e* V is a point set.

The general field of study of multivariate polynomials is algebraic geometry. See [23] and [22] for a nice introduction to the field and for proofs of all claims made in this section. In the language of algebraic geometry, V is an affine algebraic variety and the polynomials f_i generate an *ideal* $I = \Sigma_i h_i(\mathbf{x}) f_i(\mathbf{x})$, where $h_i \in \mathbb{C}[\mathbf{x}]$ are any polynomials and $\mathbb{C}[\mathbf{x}]$ denotes the set of all polynomials in $\mathbf{x}$ over the complex numbers.

One motivation for studying the ideal I is that it is a generalization of the set of equations (2.10). A point $\mathbf{x}$ is a zero of (2.10) iff it is a zero of I. Being even more general, we could ask for the complete set of polynomials which vanish on V. If I is equal to this set, then I is called a radical ideal.

We say that two polynomials f, g are equivalent modulo I iff $f - g \in I$ and denote this by $f \sim g$. With this definition we get the quotient space $\mathbb{C}[\mathbf{x}]/I$ of all equivalence classes modulo I. Let $[\cdot]$ denote the natural projection $\mathbb{C}[\mathbf{x}] \mapsto \mathbb{C}[\mathbf{x}]/I$, *i.e* by $[f_i]$ we mean the set $\{g_i : f_i - g_i \in I\}$ of all polynomials equivalent to f_i modulo I.

A related structure is $\mathbb{C}[V]$, the set of equivalence classes of polynomial functions on V. We say that a function F is polynomial on V if there is a polynomial f such that $F(\mathbf{x}) = f(\mathbf{x})$ for $\mathbf{x} \in V$. By equivalence we here mean equality on V (see Figure 2.3). If two polynomials are equivalent modulo I, then they are obviously also equal on V. If I is radical, then conversely two polynomials which are equal on V must also be equivalent modulo I. This means that for radical ideals, $\mathbb{C}[\mathbf{x}]/I$ and $\mathbb{C}[V]$ are isomorphic. Now, if V is a point set, then any function F on V can be identified with a $|V|$-dimensional vector $\mathbb{F}$ with $\mathbb{F}_i = F(v_i)$, where $v_i \in V$. Now, the unisolvence theorem for polynomials guarantees that any function can be interpolated exactly by a polynomial on a finite set of points, *i.e* there is a polynomial f such that $f(v_i) = \mathbb{F}_i$. This means

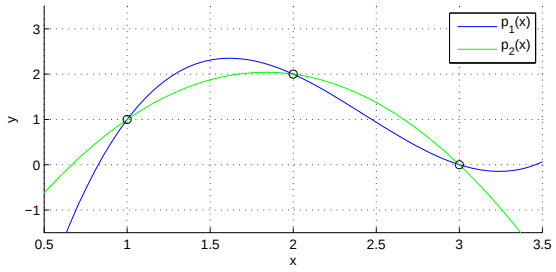

Figure 2.3: Given $V = \{1, 2, 3\}$, the two polynomials $p_1(x)$ and $p_2(x)$ shown in the figure are equivalent and hence represent the same equivalence class in $\mathbb{C}[V]$.

that any function F on V is a polynomial function and hence we get that $\mathbb{C}[V]$ is isomorphic to $\mathbb{C}^r$, where $r = |V|$.

2.2.1 The Action Matrix

Turning to equation solving, our starting point is the companion matrix which arises for polynomials in one variable. For a third degree polynomial

$$q(x) = x^3 + a_2x^2 + a_1x + a_0, \tag{2.11}$$

the companion matrix is

$$\begin{bmatrix} -a_2 & 1 & 0 \\ -a_1 & 0 & 1 \\ -a_0 & 0 & 0 \end{bmatrix}. \tag{2.12}$$

The eigenvalues of the companion matrix are the zeros of $q(x)$ and for high degree polynomials, this provides a numerically stable way of calculating the roots.

With some care, this technique can be extended to the multivariate case as well, which was first done by Lazard in 1981 [58]. For V finite, the space $\mathbb{C}[\mathbf{x}]/I$ is finite dimensional. Moreover, if I is radical, then the dimension of $\mathbb{C}[\mathbf{x}]/I$ is equal to $|V|$, *i.e* the number of solutions [23]. For some $p \in \mathbb{C}[\mathbf{x}]$ consider now the operation $T_p : f(\mathbf{x}) \mapsto p(\mathbf{x})f(\mathbf{x})$. The operator T_p is linear and since $\mathbb{C}[\mathbf{x}]/I$ is finite dimensional, we can select a linear basis $\mathcal{B}$ of polynomials for $\mathbb{C}[\mathbf{x}]/I$ and represent T_p as a matrix $\mathbf{m}_p$. This matrix is known as the action matrix and is precisely the generalization of the companion matrix we are looking for. In fact, in the example above, we can let the set $\{[x^2], [x], [1]\}$ be a basis for $\mathbb{C}[x]/\langle q(x)\rangle$, where $\langle q(x)\rangle$ denotes the ideal generated by $q(x)$. Representing $T_x : f(x) \mapsto xf(x)$ in this basis yields exactly the matrix in Equation 2.12. The eigenvalues of $\mathbf{m}_p$ are $p(\mathbf{x})$ evaluated at the points of V. Moreover, the eigenvectors of $\mathbf{m}_p^T$ equals the vector of basis elements evaluated on V. Briefly, this can be understood as follows: Consider an arbitrary polynomial $r(\mathbf{x}) = c^T\mathbf{b}$, where c is a vector of coefficients and $\mathbf{b}$ is a vector of polynomials forming a basis of $\mathbb{C}[\mathbf{x}]/I$. We then have

$$[p \cdot c^T\mathbf{b}] = [(\mathbf{m}_p c)^T\mathbf{b}] = [c^T\mathbf{m}_p^T\mathbf{b}]. \tag{2.13}$$

This holds for any coefficient vector c and hence it follows that $[p\mathbf{b}] = [\mathbf{m}_p^T\mathbf{b}]$, which can be written $p\mathbf{b} = \mathbf{m}_p^T\mathbf{b} + \mathbf{g}$ for some vector $\mathbf{g}$ with components $g_i \in I$. Evaluating the expression at a zero $\bar{\mathbf{x}} \in V$ we get $\mathbf{g}(\bar{\mathbf{x}}) = 0$ and thus obtain

$$p(\bar{\mathbf{x}})\mathbf{b}(\bar{\mathbf{x}}) = \mathbf{m}_p^T\mathbf{b}(\bar{\mathbf{x}}), \tag{2.14}$$

which we recognize as an eigenvalue problem for the matrix $\mathbf{m}_p^T$ with eigenvectors $\mathbf{b}(\bar{\mathbf{x}})$ and eigenvalues $p(\bar{\mathbf{x}})$. In other words, the eigenvectors of $\mathbf{m}_p^T$ yield $\mathbf{b}(\mathbf{x})$ evaluated at the zeros of I and the eigenvalues give $p(\mathbf{x})$ at the zeros. The conclusion we can draw from this is that zeros of I corresponds to eigenvectors and eigenvalues of $\mathbf{m}_p$, but not necessarily the opposite, *i.e* there can be eigenvectors/eigenvalues that do not correspond to actual solutions. If I is radical, this is not the case and we have an exact correspondence.

Note here that in a strict sense, a set of monomials $\mathcal{B}$ cannot form a basis for $\mathbb{C}[\mathbf{x}]/I$ since $\mathbb{C}[\mathbf{x}]/I$ is a space of equivalence classes. What we mean is that a set of monomials $\mathcal{B}$ are *representatives* of equivalence classes forming a basis of $\mathbb{C}[\mathbf{x}]/I$ or alternatively that the natural projection $[\cdot]$ of the monomials onto $\mathbb{C}[\mathbf{x}]/I$ form a basis. In the following we will, however, typically use the slightly incorrect but more readable terminology of referring to a set of monomials as a basis.

2.2.2 Gröbner Bases

We have seen theoretically that the action matrix $\mathbf{m}_p$ provides the solutions to a corresponding system of polynomial equations. The main issue is now how to compute $\mathbf{m}_p$. This is in general done by selecting a linear basis $\mathcal{B}$ for $\mathbb{C}[\mathbf{x}]/I$ and then computing $[p \cdot b_i]$ for each $b_i \in \mathcal{B}$. To do actual computations in $\mathbb{C}[\mathbf{x}]/I$ we need to represent each equivalence class $[f]$ by a well defined representative polynomial. The idea is to use multivariate polynomial division and represent $[f]$ by the remainder under division of f by I. Fortunately, for any polynomial ideal I, this can always be done and the tool for doing so is a Gröbner basis G for I [23]. The Gröbner basis for I is a canonical set of generators for I with the property that multivariate division by G, denoted $\overline{f}^G$, always yields a well defined remainder. By well defined we mean that for any $f_1, f_2 \in [f]$, we have $\overline{f_1}^G = \overline{f_2}^G$. The Gröbner basis is computed relative a monomial order and will be different for different monomial orders. As a consequence, the set of representatives for $\mathbb{C}[\mathbf{x}]/I$ will be different, whereas the space itself remains the same.

The linear basis $\mathcal{B}$ should consist of elements b_i such that the elements $\{[b_i]\}_{i=1}^r$ together span $\mathbb{C}[\mathbf{x}]/I$ and $\overline{b_i}^G = b_i$. Then all we have to do to get $\mathbf{m}_p$ is to compute the action $\overline{pb_i}^G$ for each basis element b_i, which is easily done if G is available.

Example 2. The following two equations describe the intersection of a line and a circle as illustrated in Figure 2.

$$\begin{aligned} x^2 + y^2 - 1 &= 0 \\ x - y &= 0. \end{aligned} \tag{2.15}$$

A Gröbner basis for this system is

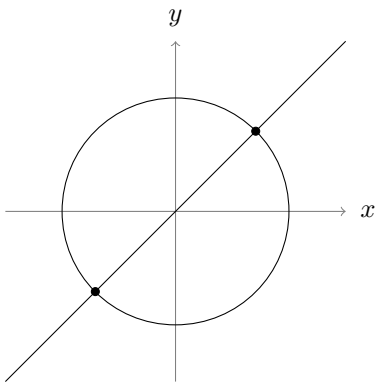

Figure 2.4: The intersection of a line and a circle can be formulated as a system of two polynomial equations. See Example 2.

$$\begin{aligned} y^2 - \tfrac{1}{2} &= 0 \\ x - y &= 0, \end{aligned} \tag{2.16}$$

from which we trivially see that the solutions are $\frac{1}{\sqrt{2}}(1,1)$ and $\frac{1}{\sqrt{2}}(-1,-1)$. However, it is nevertheless instructive to construct the action matrix. In this case $\mathcal{B} = \{y, 1\}$ are representatives for a basis for $\mathbb{C}[\mathbf{x}]/I$ and we have $T_x[1] = [x] = [y]$ and $T_x[y] = [xy] = [y^2] = [\frac{1}{2}]$, which yields the action matrix

$$\mathbf{m}_x = \begin{bmatrix} 0 & 1 \\ \frac{1}{2} & 0 \end{bmatrix}, \tag{2.17}$$

with eigenvalues $\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}}$. □

2.2.3 A Note on Algebraic and Linear Bases

At this point there is a potentially confusing situation since there are two different types of bases at play. There is the linear basis $\mathcal{B}$ of the quotient space $\mathbb{C}[\mathbf{x}]/I$ and there is the algebraic basis (Gröbner basis) G of the ideal I. To make the subsequent arguments as transparent as possible for the reader we will emphasize this fact by referring to the former as a *linear basis* of $\mathbb{C}[\mathbf{x}]/I$ and the latter as an *algebraic basis* of I.

2.2.4 Floating Point Gröbner Basis Computations

The well established Buchberger's algorithm is guaranteed to compute a Gröbner basis in finite time and works well in exact arithmetic [23]. However, due to round-off errors, it easily becomes unstable in floating point arithmetic and except for very small examples it becomes practically useless. The reason for this is that in the Gröbner basis computation, leading terms are successively eliminated from the generators of I by pairwise subtraction of polynomials, much like Gaussian elimination. This leads to cancellation effects where it becomes impossible to tell whether a certain coefficient should be zero or not.

A technique introduced by Faugere *et al* in [28] is to write the system of equations on matrix form

$$\mathbf{CX} = 0, \tag{2.18}$$

where $\mathbf{X} = \begin{bmatrix}\mathbf{x}^{\boldsymbol{\alpha}_1} & \dots & \mathbf{x}^{\boldsymbol{\alpha}_n}\end{bmatrix}^T$ is a vector of monomials with the notation $\mathbf{x}^{\alpha_k} = x_1^{\alpha_{k1}} \cdots x_s^{\alpha_{ks}}$ and $\mathbf{C}$ is a matrix of coefficients. Elimination of leading terms now translates to matrix operations and we then have access to a whole battery of techniques from numerical linear algebra allowing us to perform many eliminations at the same time with control on pivoting *etc.*

This technique takes us further, but for larger more demanding problems it is necessary to study a particular class of equations and use knowledge of what the structure of the Gröbner basis should be to design a special purpose Gröbner basis solver [76]. Typical examples from computer vision where this method can be applied are: essential matrix estimation [77], relative orientation for omnidirectional cameras [33], fundamental matrix estimation with radial distortion [55], optimal three view triangulation [80], *etc.* The typical work flow has been to study the particular problem at hand with the aid of a computer algebra system such as Maple or Macaulay2 [36] and extract information such as the leading terms of the Gröbner basis, the monomials to use as a basis for $\mathbb{C}[\mathbf{x}]/I$, the number of solutions, *etc* and work out a specific set of larger (Gauss-Jordan) elimination steps leading to the construction of a Gröbner basis for I.

Although, these techniques have permitted the solution to a large number of previously unsolved problems, many difficulties remain. Most notably, the above mentioned elimination steps (if at all doable) are often hopelessly ill conditioned [80, 56]. This is in part due to the fact that one has focused on computing a complete and correct Gröbner basis respecting a properly defined monomial order, which we show is not necessary.

In this work we move away from the goal of computing a Gröbner basis for I and focus on finding a representative of f in terms of a linear combination of a basis $\mathcal{B}$, since this is the key to constructing $\mathbf{m}_p$. We denote this operation $\overline{f}$ for a given $f \in \mathbb{C}[\mathbf{x}]$. Specifically, it is not necessary to be able to compute $\overline{f}$ for any $f \in \mathbb{C}[\mathbf{x}]$. To construct $\mathbf{m}_p$, we only need to worry about finding $\overline{f}$ for $f \in p\mathcal{B} \setminus \mathcal{B}$, which is an easier task. It should however be noted that the computations we do much resemble those necessary to obtain a Gröbner basis.

A further advantage of not having to compute a complete Gröbner basis is that we are not bound by any particular monomial order which as we will see, when used right, buys considerable numerical stability. In addition to this we introduce an object which generalizes the action matrix and can be computed even when a true linear basis for $\mathbb{C}[\mathbf{x}]/I$ cannot be used.

Drawing on these observations, we investigate in detail the exact matrix operations needed to compute $\overline{f}$ and thus obtain a procedure which is both faster and more stable, enabling the solution of a larger class of problems than previously possible. The theory behind these statements is explored in Chapter 4 and subsequently used in Chapter 5 to derive new stable algorithms for equation solving.

Part I

Solving Polynomial Equations

Chapter 3

Introduction

Numerous geometric problems in computer vision involve the solution of systems of polynomial equations. This is particularly true for so called minimal structure and motion problems [20, 51, 82]. Solutions to minimal structure and motion problems can often be used in RANSAC algorithms to find inliers in noisy data [30, 83, 84]. For such applications one needs to solve a large number of minimal structure and motion problems as fast as possible in order to find the best set of inliers. The minimal solutions then typically also serve as an initial estimate to be able to deploy a more sophisticated optimization algorithm, which relies on inlier free data and good initialization. There is thus a need for fast and numerically stable algorithms for solving particular systems of polynomial equations.

The state-of-the-art method for numerical solution of polynomial equations is based on calculations with Gröbner bases [76] and has many applications in computer vision, but also in other fields such as cryptology [29] and robotics [4]. A typical outline of such algorithms is that one first studies a specific geometric problem and finds out what structure the Gröbner basis of the ideal I has for that problem, how many solutions there are and what the degrees of monomials occurring in the Gröbner basis elements are. For each instance of the problem with numerical data, the process of forming the Gröbner basis follows the same steps and the construction of the Gröbner basis can be written down as a sequence of pre determined elimination steps using numerical linear algebra. The Gröbner basis can then be used to construct an *action matrix*, which represents multiplication in the quotient space $\mathbb{C}[\mathbf{x}]/I$. The solution to the problem is then obtained through an eigenvalue decomposition of the action matrix.

Currently, the limiting factor in using these methods for larger and more difficult cases is numerical problems. For example in [80], where an algorithm for optimal triangulation from three views is proposed, it was necessary to use emulated 128 bit numerics to make the system work, which made the implementation very slow. This thesis improves on the state of the art of these techniques making it possible to handle larger and more difficult problems in a practical way.

In the thesis we pin-point the main source of these numerical problems (the conditioning of a crucial elimination step) and propose a range of techniques for dealing with this issue. The main novelty is a new approach to the action matrix method for equation solving, relaxing the need of adhering to a properly

defined monomial order and a complete Gröbner basis. This unlocks substantial freedom, which is used in a number of different ways to improve stability.

Firstly, we show how the sensitive elimination step can be avoided by using an overly large/redundant linear basis for $\mathbb{C}[\mathbf{x}]/I$ to construct the action matrix. This method yields the right solutions along with a set of false solutions that can then easily be filtered out by evaluation in the original equations.

Secondly, we show how a true linear basis for $\mathbb{C}[\mathbf{x}]/I$ can be constructed from a redundant basis in such a way that good numerical precision is retained. This is done by attempting to find an optimal reordering or even linear combination of the monomials and we investigate what conditions such a reordering/linear combination needs to satisfy. We develop the tools needed to compute the action matrix in a general linear basis for $\mathbb{C}[\mathbf{x}]/I$ and propose two strategies for selecting this basis which enhances the stability of the solution procedure.

The first of these is a fast strategy based on QR factorization with column pivoting. The Gröbner basis like computations employed to solve a system of polynomial equations can essentially be seen as matrix factorization of an under-determined linear system. Based on this insight, we combine the robust method of QR factorization from numerical linear algebra with the Gröbner basis theory needed to solve polynomial equations. More precisely, we employ QR factorization with column pivoting in the above mentioned elimination step and obtain a simultaneous selection of linear basis and triangular factorization.

Factorization with column pivoting is a very well studied technique and there exist highly optimized and reliable implementations of these algorithms in *e.g* LAPACK [57], which makes this technique accessible and relatively straightforward to implement.

The second technique for basis selection goes one step further and employs singular value decomposition (SVD) to select a general linear basis of polynomials for $\mathbb{C}[\mathbf{x}]/I$. This technique is computationally more demanding than the QR method, but yields even better stability.

Finally, we show how a redundant linear basis for $\mathbb{C}[\mathbf{x}]/I$ can be combined with the above basis selection techniques. In the QR method, since the pivot elements are sorted in descending order, we get an adaptive criterion for where to truncate the Gröbner basis like structure by setting a maximal threshold for the quotient between the largest and the smallest pivot element. When the quotient exceeds this threshold we abort the elimination and move the remaining columns into the basis. This way, we expand the basis only when necessary.

Chapter 4

Theoretical Contributions

In this chapter we present a new way of looking at the action matrix method for polynomial equation solving. The advantage of the new formulation is that it yields more freedom in how the action matrix is computed allowing us to derive numerically more stable algorithms.

4.1 A New Approach to the Action Matrix Method

We start with a few examples that we will use to clarify the ideas of this chapter.

Example 3. In the five point relative orientation problem for calibrated cameras, [51, 24, 66, 77], the calculation of the essential matrix using 5 image point correspondences leads to 10 equations of degree 3 in 3 unknowns. These equations involve 20 monomials. By writing the equations as in (2.18) and using a total degree ordering on the monomials we get a coefficient matrix $\mathbf{C}$ of size 10×20 and a monomial vector $\mathbf{X} = [\mathbf{x}^{\boldsymbol{\alpha}_1} \dots \mathbf{x}^{\boldsymbol{\alpha}_n}]^T$ with 20 monomials. It turns out that if we partition the monomials so that $[\mathbf{C}_1 \ \mathbf{C}_2] \left[\begin{smallmatrix} \mathbf{X}_1 \\ \mathbf{X}_2 \end{smallmatrix}\right] = 0$, then the first 10×10 block $\mathbf{C}_1$ is in general of full rank and thus the first 10 monomials $\mathbf{X}_1$ can be expressed in terms of the last 10 monomials $\mathbf{X}_2$ as

$$\mathbf{X}_1 = -\mathbf{C}_1^{-1}\mathbf{C}_2\mathbf{X}_2. \tag{4.1}$$

This makes it possible to regard the monomials in $\mathbf{X}_2$ as representatives of a linear basis for $\mathbb{C}[\mathbf{x}]/I$. It is now straightforward to calculate the action matrix for T_x (the multiplication operator for multiplication by x) since monomials in the linear basis are either mapped to monomials already in the basis or to monomials in $\mathbf{X}_1$, which can be expressed in terms of the basis using (4.1). □

In this example the linear basis $\mathbf{X}_2$ can be thought of as a basis for the space of remainders after division with a Gröbner basis for one choice of monomial order and this is how these computations have typically been viewed. However, the calculations above are not really dependent on any properly defined monomial order and it seems that they should be meaningful irrespective of whether a true monomial order is used or not. Moreover, we do not use all the Gröbner basis properties.

Based on these observations we emphasize two important facts: (i) We are not interested in finding the Gröbner basis or a basis for the remainder space relative to some Gröbner basis *per se*; it is enough to get a well defined mapping $\overline{f}$ and (ii) it suffices to calculate $\overline{f}$ on the elements $x \cdot \mathbf{x}^{\alpha_i}$, *i.e* we do not need to be able to compute $\overline{f}$ for all $f \in \mathbb{C}[\mathbf{x}]$. These statements and their implications will be made more precise further on.

Example 4. Consider the equations

$$\begin{aligned} f_1 &= xy + x - y - 1 = 0, \\ f_2 &= xy - x + y - 1 = 0, \end{aligned} \tag{4.2}$$

with solutions $(-1,-1),(1,1)$. To this set we can add $f_3 = (f_1 - f_2)/2 = x - y$. Now let $\mathcal{B} = \{x, y, 1\}$ be a set of representatives for the equivalence classes in $\mathbb{C}[\mathbf{x}]/I$ for this system. The set $\mathcal{B}$ does not constitute a proper basis for $\mathbb{C}[\mathbf{x}]/I$ since the elements of $\mathcal{B}$ represent linearly dependent equivalence classes. They do however span $\mathbb{C}[\mathbf{x}]/I$. Now study the operator T_y acting on $\mathcal{B}$. We have $T_y(1) = y$, $T_y(x) = xy \sim x - y + 1$ and $T_y(y) = y^2 \sim xy \sim x - y + 1$ (where we used first yf_3 and then f_2) which gives a multiplication matrix

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

An eigendecomposition of this matrix yields the solutions $(-1,-1),(1,1),(-1,0)$. Of these the first two are true solutions to the problem, whereas the last one does not satisfy the equations and is thus a false zero. □

In this example we used a set of monomials $\mathcal{B}$ whose corresponding equivalence classes spanned $\mathbb{C}[\mathbf{x}]/I$, but were not linearly independent. However, it was still possible to express the image $T_y(\mathcal{B})$ in terms of $\mathcal{B}$. The elements of the resulting action matrix are not uniquely determined. Nevertheless we were able to use it to find the solutions to the problem. In this section we give general conditions for when a set $\mathcal{B}$ can be used to construct a multiplication matrix which produces the desired set of zeros, possibly along with a set of false zeros, which need to be filtered out.

More generally this also means that the chosen representatives of the linear basis of $\mathbb{C}[\mathbf{x}]/I$ need not be low order monomials given by a Gröbner basis. In fact, they need not be monomials at all, but could be general polynomials.

Drawing on the concepts illustrated in the above two examples we define a *solving basis*, similar to $\mathcal{B}$ in Example 4. The overall purpose of the definition is to rid our selves of the need of talking about a Gröbner basis and properly defined monomial orders, thus providing more room to derive numerically stable algorithms for computation of the action matrix and similar objects.

In the following we will also provide techniques for determining if a candidate basis $\mathcal{B}$ constitutes a solving basis and we will give numerically stable techniques for basis selection in too large (linearly dependent) solving bases, here referred to as redundant bases.

4.1.1 Solving Bases

We start off with a set of polynomial equations as in (2.10) and a assume a (point) set of zeros $V(f_1, \ldots, f_m)$. Given this we make the following definition

Definition 5. Consider a finite subset $\mathcal{B} \subset \mathbb{C}[\mathbf{x}]$ of the set of polynomials over the complex numbers. If for each $b_i \in \mathcal{B}$ and some $p \in \mathbb{C}[x]$ we can express pb_i as a linear combination of basis elements as

$$p(\mathbf{x})b_i(\mathbf{x}) = \Sigma_j m_{ij} b_j(\mathbf{x}), \tag{4.3}$$

for some (not necessarily unique) coefficients m_{ij} and where equality means equality on V, then we call $\mathcal{B}$ a *solving basis* for (2.10) w.r.t p. □

We now get the following for the matrix $\mathbf{m}_p$ made up of the coefficients m_{ij}.

Theorem 6. *Given a solving basis $\mathcal{B}$ for (2.10) w.r.t p, the evaluation of p on V is an eigenvalue of the matrix $\mathbf{m}_p$. Moreover, the vector $\mathbf{b} = (b_1, \ldots, b_r)^T$ evaluated on V is an eigenvector of $\mathbf{m}_p$.*

Proof. By the definition of $\mathbf{m}_p$, we get

$$p(\mathbf{x})\mathbf{b}(\mathbf{x}) = \begin{bmatrix} pb_1 \\ \vdots \\ fb_r \end{bmatrix} = \begin{bmatrix} \Sigma_j m_{1j} b_j \\ \vdots \\ \Sigma_j m_{rj} b_j \end{bmatrix} = \mathbf{m}_p \mathbf{b}(\mathbf{x}) \tag{4.4}$$

for $\mathbf{x} \in V$. □

As will become clear further on, when $\mathcal{B}$ is a true basis for $\mathbb{C}[\mathbf{x}]/I$, then the matrix $\mathbf{m}_p$ defined here is simply the transposed action matrix for multiplication by p.

Given a solving basis, the natural question to ask is: Under which circumstances may all solutions to the related system of equations be obtained from an eigenvalue decomposition of $\mathbf{m}_p$? We next explore some conditions under which this is possible. The general idea is that the vector of monomials $\mathbf{b}(\mathbf{x})$ evaluated at a zero $\bar{\mathbf{x}}$ is only useful if we can determine $\bar{\mathbf{x}}$ uniquely (or at least as member of a finite set of possible values) from $\mathbf{b}(\bar{\mathbf{x}})$. This is formalized in the following definition.

Definition 7. A solving basis $\mathcal{B}$ is called a *complete solving basis* if the inverse image of the mapping $x \mapsto \mathbf{b}(x)$ from variables to monomial vector is finite for all points. □

A complete solving basis allows us to recover all solutions from $\mathbf{m}_p$ as shown in the following theorem.

Theorem 8. *Let $\mathcal{B}$ be a complete solving basis for (2.10) and $\mathbf{m}_p$ as above and assume that for all eigenvalues λ_i we have $\lambda_i \neq \lambda_j$ for $i \neq j$. Then the complete set of solutions to (2.10) can be obtained from the set of eigenvectors $\{v_i\}$ of $\mathbf{m}_p$.*

Proof. The vector $\mathbf{b}(\bar{\mathbf{x}})$ for $\bar{\mathbf{x}} \in V$ is an eigenvector of $\mathbf{m}_p$. The number of eigenvectors and eigenvalues of $\mathbf{m}_p$ is finite so we can compute all eigenvectors $\{v_i\}$. This means that now $\{\mathbf{b}(\bar{\mathbf{x}})\}_{\bar{\mathbf{x}} \in V} \subset \{v_i\}$. Applying $\mathbf{b}^{-1}$ to v_i for all i thus yields a finite set of points containing V. Evaluation in (2.10) allows us to filter out the points of this set which are not solutions to (2.10) and keep only the true solutions. □

If the inverse image is not finite for some v_i so that we get a parameter family $\mathbf{x}$ corresponding to this eigenvector, then the correct solution can typically not be obtained without further use of the equations (2.10) as illustrated in the following example.

Example 9. Consider the polynomial system

$$\begin{aligned} y^2 - 2 &= 0 \\ x^2 - 1 &= 0 \end{aligned} \tag{4.5}$$

with $V = \{(1,\sqrt{2}),(-1,\sqrt{2}),(1,\sqrt{2}),(-1,-\sqrt{2}),\}$. Clearly, $\mathcal{B} = \{x, 1\}$ with monomial vector $\mathbf{b}(x,y) = \begin{bmatrix} x & 1 \end{bmatrix}^T$, is a solving basis w.r.t x for this example since $1 \cdot x = x$ and $x \cdot x = x^2 = 1$ on V. Hence, $\mathbf{b}(x,y)$ evaluated on V is an eigenvector of

$$\mathbf{m}_x = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \tag{4.6}$$

which is easily confirmed. However, these eigenvectors do not provide any information about the y-coordinate of the solutions. We could try adding y to $\mathcal{B}$ but this would not work since the values of xy on V cannot be expressed as a linear combination of x and y evaluated on V. A better choice of solving basis would be $\mathcal{B} = \{xy, x, y, 1\}$. □

At a first glance, Theorem 8 might not seem very useful since solving for x from $\mathbf{b}(x) = v_i$ potentially involves solving a new system of polynomial equations. However, it provides a tool for ruling out choices of $\mathcal{B}$ which are not practical to work with. Moreover, there is usually much freedom in the choice of $\mathcal{B}$. In general, $\mathcal{B}$ can be a set of polynomials. However, it is often practical to work with a basis of monomials. For each b_i we then get the following result

Corollary 10. *If $\mathcal{B}$ consists of monomials b_i on the form $b_i(x) = x_1^{\alpha_{i1}} \cdots x_s^{\alpha_{is}}$ and the $r \times s$ matrix A with $A_{ij} = \alpha_{ij}$ is of rank s, then all solutions to (2.10) can be obtained from the eigenvectors of $\mathbf{m}_{x_k}$.*

Proof. Taking the logarithm of $b_j(\mathbf{x})$ we get component wise

$$\log(b_i(\mathbf{x})) = \Sigma_j \alpha_{ij} \log(\tilde{x}_j), \tag{4.7}$$

where $\tilde{x}_j = \pm x_j$ if necessary. Using the matrix A, this can be written

$$\log(\mathbf{b}(\mathbf{x})) = A \begin{bmatrix} \log(\tilde{x_1}) \\ \vdots \\ \log(\tilde{x_s}) \end{bmatrix}. \tag{4.8}$$

If $\text{rank}(A) = s$ then we can solve linearly for $\log(\tilde{\mathbf{x}})$ and theorem 8 yields the conclusion. □

We get an even more convenient situation if the right monomials are included in $\mathcal{B}$:

Corollary 11. *If $\{1, x_1, \ldots, x_s\} \subset \mathcal{B}$, then all solutions to (2.10), can be directly read off from the eigenvectors of $\mathbf{m}_{x_k}$.*

Proof. Since the monomials $\{1, x_1, \ldots, x_s\}$ occur in $\mathcal{B}$, they enter in the vector $\mathbf{b}(x)$ and hence the mapping in Definition 7 is injective with a trivial inverse. □

The purpose of Theorem 6 and Corollaries 10 and 11 are to provide guarantees for when all information about the solutions can be obtained from the multiplication matrices. Phrased a little differently, the idea behind these results is to consider the relation between a solution point $\bar{\mathbf{x}}$ and monomial vector $\mathbf{b}(\bar{\mathbf{x}})$. We know that a solution point $\bar{\mathbf{x}}$ *always* corresponds to a vector $\mathbf{b}(\bar{\mathbf{x}})$ which is an eigenvector of the corresponding multiplication matrix. The only way we could miss some solutions would hence be if two different zeros $\bar{\mathbf{x}}_1$ and $\bar{\mathbf{x}}_2$ map to the same monomial vector. However, if the mapping $\mathbf{x} \mapsto \mathbf{b}(\mathbf{x})$ is injective, this cannot happen and we are safe.

The situation in Corollary 11 is certainly the most convenient one. However, even if not all variables are included as elements in $\mathcal{B}$, we can often still express each variable x_k as a linear combination of the basis elements $b_i(\mathbf{x})$ for $\mathbf{x} \in V$ by making use of the original equations. We thus again obtain a well defined inverse to the mapping in Definition 7.

Example 12. Consider the polynomial system (4.2) from Example 4. Subtracting f_1 and f_2 and dividing by 2 we get a third polynomial $f_3 = x - y$. Thus $\mathcal{B} = \{y, 1\}$ constitutes a solving basis w.r.t x since $T_x(1) = x = y$ (on V) and $T_x(y) = xy = x - y + 1 = 1$ (on V). The vector of monomials $\mathbf{b}(x, y) = \begin{bmatrix} y & 1 \end{bmatrix}^T$ seen as a mapping $\mathbf{b} : \mathbb{R}^2 \mapsto \mathbb{R}^2$ is not invertible since it does not give any information about the x coordinate. However, we can use $f_3 = x - y = 0$ to get the solutions from the eigenvectors. □

Finally, we show how the concept of solving basis connects to the standard theory of action matrices in the quotient space $\mathbb{C}[x]/I$.

Theorem 13. *If the ideal I generated by (2.10) is radical, then a solving basis $\mathcal{B}$ w.r.t to p for (2.10), with the additional properties that $\mathbf{b}(\mathbf{x})$ is injective and that all eigenvalues of $\mathbf{m}_p$ are distinct, spans $\mathbb{C}[x]/I$.*

Proof. Since I is radical, $\mathbb{C}[x]/I$ is isomorphic to $\mathbb{C}[V]$, the ring of all polynomial functions on V. Moreover, since V is finite, all functions on V are polynomial (see previous chapter) and hence $\mathbb{C}[V]$ can be identified with $\mathbb{C}^r$, where $r = |V|$. Consider now the matrix $B = \begin{bmatrix} \mathbf{b}(x_1), \ldots, \mathbf{b}(x_r) \end{bmatrix}$. Each row of B corresponds to a (polynomial) function on V. Hence, if we can show that B has row rank r, then these functions together span $\mathbb{C}[V]$ and we are done. Due to Theorem 6, all $\mathbf{b}(x_i)$ are eigenvectors of $\mathbf{m}_p$ with eigenvalues $p(x_i)$. By the assumption of distinct eigenvalues we have $p(x_i) \neq p(x_j)$ whenever $\mathbf{b}(x_i) \neq \mathbf{b}(x_j)$. Since $\mathcal{B}$ is a complete solving basis we have $\mathbf{b}(x_i) \neq \mathbf{b}(x_j)$ whenever $x_i \neq x_j$. This means that the r points in V correspond to distinct eigenvalues and hence, since eigenvectors corresponding to different eigenvalues are linearly independent, B has column rank r. For any matrix row rank equals column rank and we are done. □

The above theorem provides a correspondence between solving bases and linear bases for $\mathbb{C}[\mathbf{x}]/I$ and in principle states that under some extra restrictions, a solving basis is simply a certain choice of linear basis for $\mathbb{C}[\mathbf{x}]/I$ and then the matrix $\mathbf{m}_p$ turns into the transposed action matrix. However, relaxing these

extra restrictions we get something which is not necessarily a basis for $\mathbb{C}[\mathbf{x}]/I$ in the usual sense, but can still be used to construct a matrix $\mathbf{m}_p$ which encodes the solutions. This is what we call a solving basis. Using the concept of a solving basis provides two distinctive advantages:

(i) For a radical polynomial system with r zeros, $\mathbb{C}[\mathbf{x}]/I$ is r-dimensional, so a basis for $\mathbb{C}[\mathbf{x}]/I$ contains r elements. This need not be the case for a solving basis, which could well contain more than r elements, but due to Theorem 8 still provides the right solutions. This fact is exploited in Section 5.1.

(ii) Typically, the arithmetic in $\mathbb{C}[\mathbf{x}]/I$ has been computed using a Gröbner basis for I, which directly provides a monomial basis for $\mathbb{C}[\mathbf{x}]/I$ in form of the set of monomials which are not divisible by the Gröbner basis. In this work we move focus from Gröbner basis computation to the actual goal of expressing the products pb_i in terms of a set of linear basis elements and thus no longer need to adhere to the overly strict ordering rules imposed by a particular monomial order. This freedom is exploited in Sections 5.2.1 and 5.2.2.

Finally, (i) and (ii) are combined in Section 5.2.3.

4.1.2 Solving Basis Computations using Numerical Linear Algebra

We now describe the most straightforward technique for deciding whether a candidate basis $\mathcal{B}$ w.r.t one of the variables x_k, can be used as a solving basis and simultaneously calculate the action of T_{x_k} on the elements of $\mathcal{B}$.

We start by generating more equations by multiplying the original set of equations by a hand crafted (problem dependent) set of monomials. This yields additional equations, which are equivalent in terms of solutions, but hopefully linearly independent from the original ones. In Example 9, we could multiply by *e.g* $\{x, y, 1\}$, yielding $xy^2 - 2x, x^3 - x, y^3 - 2y, x^2y - y, y^2 - 2, x^2 - 1$.

Given a candidate for a linear basis $\mathcal{B}$ of monomials one then partitions the set of all monomials $\mathcal{M}$ occurring in the equations in to three parts $\mathcal{M} = \mathcal{E} \bigcup \mathcal{R} \bigcup \mathcal{B}$, where $\mathcal{R} = x_k\mathcal{B} \backslash \mathcal{B}$ is the set of monomials that need to be expressed in terms of $\mathcal{B}$ to satisfy the definition of a solving basis and $\mathcal{E} = \mathcal{M} \setminus (\mathcal{R} \bigcup \mathcal{B})$ is the set of remaining (excessive) monomials. Each column in the coefficient matrix represents a monomial, so we reorder the columns and write the system of equations as

$$\mathbf{C}\mathbf{X} = \begin{bmatrix} \mathbf{C}_\mathcal{E} & \mathbf{C}_R & \mathbf{C}_B \end{bmatrix} \begin{bmatrix} \mathbf{X}_\mathcal{E} \\ \mathbf{X}_\mathcal{R} \\ \mathbf{X}_\mathcal{B} \end{bmatrix} = 0, \tag{4.9}$$

reflecting the above partitioning. The $\mathcal{E}$-monomials are not used in the action matrix computation so we eliminate them by putting $\mathbf{C}_\mathcal{E}$ on row echelon form using LU factorization

$$\begin{bmatrix} \mathbf{U}_{\mathcal{E}1} & \mathbf{C}_{\mathcal{R}1} & \mathbf{C}_{\mathcal{B}1} \\ \mathbf{0} & \mathbf{C}_{\mathcal{R}2} & \mathbf{C}_{\mathcal{B}2} \end{bmatrix} \begin{bmatrix} \mathbf{X}_\mathcal{E} \\ \mathbf{X}_\mathcal{R} \\ \mathbf{X}_\mathcal{B} \end{bmatrix} = 0. \tag{4.10}$$

We now discard the top rows and provided that enough linearly independent equations were added in the first step so that $\mathbf{C}_{\mathcal{R}2}$ is of full rank, we multiply

by $\mathbf{C}_{\mathcal{R}2}^{-1}$ from the left producing

$$\begin{bmatrix}\mathbf{I} & \mathbf{C}_{\mathcal{R}2}^{-1}\mathbf{C}_{\mathcal{B}2}\end{bmatrix}\begin{bmatrix}\mathbf{X}_{\mathcal{R}}\\ \mathbf{X}_{\mathcal{B}}\end{bmatrix} = 0 \tag{4.11}$$

or equivalently

$$\mathbf{X}_{\mathcal{R}} = -\mathbf{C}_{\mathcal{R}2}^{-1}\mathbf{C}_{\mathcal{B}2}\mathbf{X}_{\mathcal{B}}, \tag{4.12}$$

which means that the $\mathcal{R}$-monomials can be expressed as a linear combination of the basis monomials. Thus $\mathcal{B}$ is a solving basis and the matrix $\mathbf{m}_{x_k}$ can easily be constructed as in (4.3). In other words, given an enlarged set of equations and a choice of linear basis $\mathcal{B}$, the full rank of $\mathbf{C}_{\mathcal{R}2}$ is sufficient to solve (2.10) via eigendecomposition of $\mathbf{m}_{x_k}$. The above method is summarized in Solving Basis Method and given the results of Section 4.1.1 we now have the following

Result 14. *The algorithm* Solving Basis Method *yields the complete set of zeros of a polynomial system, given that the postconditions are satisfied.*

Proof. The postcondition that $\mathbf{C}_{\mathcal{R}2}$ is of full rank ensures that $\mathcal{B}$ is a solving basis and Theorem 8 and Corollary 11 then guarantees the statement. □

Example 15. Consider the equations from Example 2. Multiplying the second equation by x and y yields the enlarged system

$$\begin{bmatrix}1 & 0 & 1 & 0 & 0 & -1\\ 1 & -1 & 0 & 0 & 0 & 0\\ 0 & 1 & -1 & 0 & 0 & 0\\ 0 & 0 & 0 & 1 & -1 & 0\end{bmatrix}\begin{bmatrix}x^2\\ xy\\ y^2\\ x\\ y\\ 1\end{bmatrix} = 0, \tag{4.13}$$

with $\mathcal{M} = \{x^2, xy, y^2, x, y, 1\}$. Choosing $\mathcal{B} = \{y, 1\}$ then gives $\mathcal{R} = \{xy, x\}$ and $\mathcal{E} = \{x^2, y^2\}$. After Step 9 and 10 of Algorithm Solving Basis Method we have $\mathbf{C}_{\mathcal{R}2} = \left[\begin{smallmatrix}2 & 0\\ 0 & 1\end{smallmatrix}\right]$ and $\mathbf{C}_{\mathcal{B}2} = \left[\begin{smallmatrix}0 & -1\\ -1 & 0\end{smallmatrix}\right]$ and inserting into (5.11) we obtain

$$\begin{bmatrix}xy\\ x\end{bmatrix} = \begin{bmatrix}0 & \frac{1}{2}\\ 1 & 0\end{bmatrix}\begin{bmatrix}y\\ 1\end{bmatrix}, \tag{4.14}$$

which then allows us to construct $\mathbf{m}_x$ for this example. □

SOLVING BASIS METHOD(F, $\mathcal{B}$, $\{L_i\}$)

// Compute a solving basis w.r.t x_k and use it to solve a polynomial system.
Input: List of equations $F = \{f_1, \ldots, f_m\}$, set of basis monomials $\mathcal{B}$ containing the coordinate variables $1, x_1, \ldots x_s$, m lists of monomials $\{L_i\}_{i=1}^m$.
Postcondition: $\mathbf{C}_{\mathcal{R}2}$ is of full rank, eigenvalues of $\mathbf{m}_{x_k}$ are distinct.

$F_{\text{ext}} \leftarrow F$
for $f_i \in F$
    for $\mathbf{x}^{\boldsymbol{\alpha}_j} \in L_i$
        $F_{\text{ext}} \leftarrow F_{\text{ext}} \bigcup \{\mathbf{x}^{\boldsymbol{\alpha}_j} \cdot f_i\}$
Construct coefficient matrix $\mathbf{C}$ from F_{ext}.
$\mathcal{M} \leftarrow$ The set of all monomials occurring in F_{ext}.
$\mathcal{R} \leftarrow x_k \cdot \mathcal{B} \setminus \mathcal{B}$
$\mathcal{E} \leftarrow \mathcal{M} \setminus (\mathcal{R} \bigcup \mathcal{B})$
Reorder and partition $\mathbf{C}$: $\tilde{\mathbf{C}} = [\mathbf{C}_{\mathcal{E}}\ \mathbf{C}_R\ \mathbf{C}_B]$.
LU-factorize to obtain $\mathbf{C}_{\mathcal{R}2}$ and $\mathbf{C}_{\mathcal{B}2}$ as in (4.10).
Use (4.12) to express $x_k \cdot \mathbf{x}^{\boldsymbol{\alpha}_i}$ in terms of $\mathcal{B}$ and store the coefficients in $\mathbf{m}_{x_k}$.
Compute eigenvectors of $\mathbf{m}_{x_k}$ and read off the tentative set of solutions.
Evaluate in F to filter out possible false zeros.

A typical problem that might occur is that some eigenvalues of $\mathbf{m}_{x_k}$ are equal, which means that two or more zeros have equal x_k-coordinate. Then the corresponding eigenvectors can not be uniquely determined. This problem can be resolved by computing $\mathbf{m}_{x_k}$ for several k and then forming a random linear combination $\mathbf{m}_{a_1 x_1 + \cdots + a_s x_s} = a_1 \mathbf{m}_{x_1} + \cdots + a_s \mathbf{m}_{x_s}$, which then with very small probability has two equal eigenvalues.

As previously mentioned, computing $\mathbf{m}_p$ for a larger problem is numerically very challenging and the predominant issue is expressing $p\mathcal{B}$ in terms of $\mathcal{B}$, via something similar to (4.12). The reason for this is that without proper care, $\mathbf{C}_{\mathcal{R}2}$ tends to become very ill conditioned (condition numbers of 10^{10} or higher are not uncommon). This was also the reason that extremely slow emulated 128 bit numerics had to be used in [80] to get a working algorithm.

In the next chapter we investigate techniques to circumvent this problem and produce a well conditioned $\mathbf{C}_{\mathcal{R}2}$, thus drastically improving numerical stability.

4.2 Related Work

The area of polynomial equation solving is currently very active. See *e.g* [19] and references therein for a comprehensive exposition of the state of the art in this field.

One of the oldest and still used methods for non-linear equation solving is the Newton-Raphson method which is fast and easy to implement, but depends heavily on initialization and finds only a single zero for each initialization. In the univariate case, a numerically sound procedure to find the complete set of roots is to compute the eigenvalues of the companion matrix. However, if only real solutions are needed, the fastest way is probably to use Sturm sequences [45].

In several variables a first method is to use resultants [23], which using a determinant construct enables the successive elimination of variables. However, the resultant grows exponentially in the number of variables and is in most

cases not practical for more than two variables. An alternative way of eliminating variables is to compute a lexicographical Gröbner basis for the ideal generated by the equations which can be shown to contain a univariate polynomial representing the solutions [23]. This approach is however often numerically unstable.

A radically different approach is provided by homotopy continuation methods [87]. These methods typically work in conjunction with mixed volume calculations by constructing a simple polynomial system with the same number of zeros as the actual system that is to be solved. The simple system with known zeros is then continuously deformed into the actual system. The main drawback of these methods is the computational complexity with computation times ranging in seconds or more.

At present, the best methods for geometric computer vision problems are based on eigendecomposition of a multiplication matrices representing multiplication in the quotient space $\mathbb{C}[\mathbf{x}]/I$ as discussed in the chapter. The factors that make this approach attractive is that it (i) is fast and numerically feasible, (ii) handles more than two variables and reasonably high degrees and (iii) is well suited to tuning for specific applications. To the authors best knowledge, this method was first used in the context of computer vision by Stewénius *et al* [76] even though Gröbner basis methods were used to some extent in [85] and were also mentioned in [44].

Chapter 5

Techniques for Polynomial Equation Solving

Drawing on the ideas introduced in the previous chapter, this chapter presents a range of techniques for improving the numerical stability of algorithms which rely on eigenvalue decomposition of a multiplication matrix. These techniques are based on efficient and numerically stable methods from numerical linear algebra. A benefit of this is that such routines have a relatively long history and are very well studied. Moreover, there exist highly optimized implementations in free code libraries such as LAPACK [57].

5.1 Using Redundant Solving Bases - The Truncation Method

As mentioned in Section 4.1.2, the sub matrix $\mathbf{C}_{\mathcal{R}2}$ which appears in Equation 4.10 is a large cause of numerical problems in the equation solving process. A typical situation with an ill conditioned or rank deficient $\mathbf{C}_{\mathcal{R}2}$ is that there are a few problematic monomials where the corresponding columns in $\mathbf{C}$ are responsible for the deteriorated conditioning of $\mathbf{C}_{\mathcal{R}2}$. A straightforward way to improve the situation is to simply include the problematic monomials in $\mathcal{B}$, thus avoiding the need to express these in terms of the other monomials. In practice this means that some columns of $\mathbf{C}_{\mathcal{R}}$ are moved into $\mathbf{C}_{\mathcal{B}}$. This technique is supported by Theorem 8, which guarantees that we will find the original set of solutions among the eigenvalues/eigenvectors of the larger $\mathbf{m}_p$ found using this redundant basis. The price we have to pay is performing an eigenvalue decomposition on a larger matrix.

Given a system of equations which has been expanded by multiplication with a set of monomials let, as before, $\mathcal{M}$ denote the complete set of monomials for this particular expansion. Not all monomials from $\mathcal{M}$ can be included in the basis $\mathcal{B}$ while still enabling the calculation of the necessary multiplication matrices. In general it is a difficult question exactly which monomials can be used or even if there *exists* a set $\mathcal{B}$ among $\mathcal{M}$, which can be used as a solving basis. One can however easily see that given $\mathcal{M}$, $\mathcal{B}$ has to be a subset of the following set, which we denote the *permissible* monomials:

Definition 16. The set of *permissible* monomials is the set

$$\mathcal{P} = \{b \in \mathcal{M} : pb \in \mathcal{M}\} \tag{5.1}$$

of monomials which stay in $\mathcal{M}$ under multiplication by p.

Note that $\mathcal{P}$ is not fixed for a certain system of equations, but also depends on the particular expansion which has been performed on that system. An example of how the redundant solving basis technique can be used is provided by the problem of L_2-optimal triangulation from three views [80]. The optimum is found among the up to 47 stationary points, which are zeros of a polynomial system in three variables. In this example an enlarged set of 255 equations in 209 monomials were used to get a Gröbner basis. Since the solution dimension r is 47 in this case, the 47 lowest order monomials were used as a basis for $\mathbb{C}[\mathbf{x}]/I$ in [80], yielding a numerically difficult situation. In fact, as will be shown in more detail in the experiments section, this problem can be solved by simply including more elements in $\mathcal{B}$. In this example, the complete permissible set $\mathcal{P}$ contains 154 monomials. By including all of these in $\mathcal{B}$ leaving 55 monomials to be expressed in terms of $\mathcal{B}$, we get a much smaller and in this case better conditioned elimination step. As mentioned above, this leads to a larger eigenvalue decomposition, but all true solutions can still be found among the larger set of eigenvalues/eigenvectors. This is illustrated in Figure 5.1, where the set of eigenvalues computed from $\mathbf{m}_{x_k}$ for one instance are plotted in the complex plane together with the actual solutions of the polynomial system.

5.2 Basis Selection

In the previous section we saw how it is possible to pick a "too large" ($> r$ elements) linear basis $\mathcal{P}$ and still use it to solve the equations. In this section we show how one can select a true (linearly independent) basis as a subset of $\mathcal{P}$ in a numerically stable way and thus gain both speed and stability. In the following, $\mathcal{P}$ denotes any subset of $\mathcal{M}$ with the property that the obtained $\mathbf{C}_{\mathcal{R}2}$ is of full rank, thus making $\mathcal{P}$ a solving basis.

Since the set V of zeros of (2.10) is finite with r points, $\mathcal{P}$ seen as a set of functions on V contains at most r linearly independent elements. It should therefore be possible to remove a subset $\mathcal{P}' \subset \mathcal{P}$ such that the elements in $\mathcal{P}'$ can be expressed as linear combinations of elements in $\mathcal{P} \setminus \mathcal{P}'$. By dropping $\mathcal{P}'$ from the solving basis, the set $\mathcal{B} = \mathcal{P} \setminus \mathcal{P}'$ would thus constitute a new tighter solving basis w.r.t the same multiplier p and ideal I as $\mathcal{P}$.

We now present two numerically stable techniques for constructing a true basis $\mathcal{B}$ from a redundant solving basis $\mathcal{P}$.

5.2.1 The QR Method

We start by selecting $\mathcal{P}$ as large as possible, still yielding a full rank $\mathbf{C}_{\mathcal{R}2}$ and form $[\mathbf{C}_{\mathcal{E}}\ \mathbf{C}_{\mathcal{R}}\ \mathbf{C}_{\mathcal{P}}]$. Any selection of basis monomials $\mathcal{B} \subset \mathcal{P}$ will then correspond to a matrix $\mathbf{C}_{\mathcal{B}}$ consisting of a subset of the columns of $\mathbf{C}_{\mathcal{P}}$.

By performing Gaussian elimination we again obtain (4.10), but with $\mathcal{B}$ replaced by $\mathcal{P}$, letting us get rid of the $\mathcal{E}$-monomials by discarding the top rows. Furthermore, the $\mathcal{R}$-monomials will all have to be expressed in terms of the

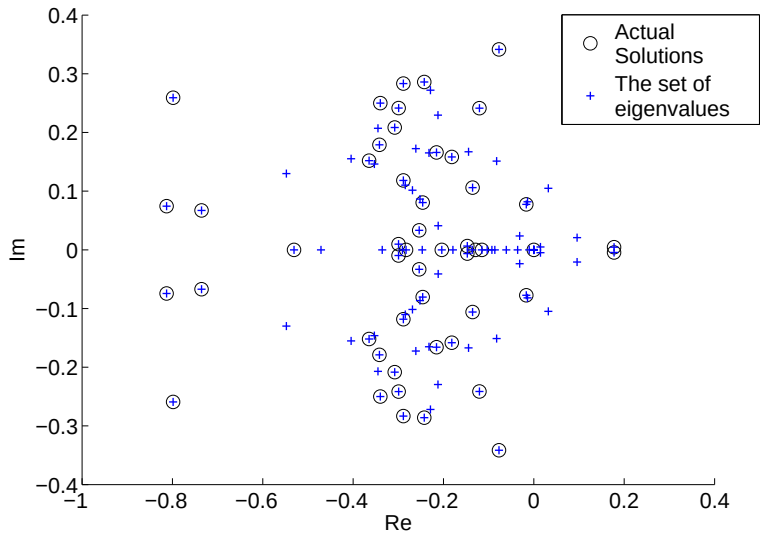

Figure 5.1: Eigenvalues of the action matrix using the redundant basis method and actual solutions to the polynomials system plotted in the complex number plane. The former are a strict superset of the latter.

$\mathcal{P}$-monomials so we continue the elimination putting $\mathbf{C}_{\mathcal{R}2}$ on triangular form, obtaining

$$\begin{bmatrix} \mathbf{U}_{\mathcal{R}} & \mathbf{C}_{\mathcal{P}1} \\ \mathbf{0} & \mathbf{C}_{\mathcal{P}2} \end{bmatrix} \begin{bmatrix} \mathbf{X}_{\mathcal{R}} \\ \mathbf{X}_{\mathcal{P}} \end{bmatrix} = 0. \tag{5.2}$$

At this point we could simply continue the Gaussian elimination, with each new pivot element representing a monomial expressed in terms of the remaining basis monomials. However, this typically leads to poor numerical performance since, as previously mentioned, the elimination might be very ill conditioned. This is where the basis selection comes to play.

As noted above we can choose which of the p monomials in $\mathcal{P}$ to put in the

basis and which to reduce. This is equivalent to choosing a permutation Π of the columns of $\mathbf{C}_{\mathcal{P}2}$,

$$\mathbf{C}_{\mathcal{P}2}\Pi = \begin{bmatrix} c_{\pi(1)} & \dots & c_{\pi(p)} \end{bmatrix} \tag{5.3}$$

and then proceed using standard elimination. The goal must thus be to make this choice so as to minimize the condition number $\kappa(\begin{bmatrix} c_{\pi(1)} & \dots & c_{\pi(p-r)} \end{bmatrix})$ of the first $p-r$ columns of the permuted matrix. In its generality, this is a difficult combinatorial optimization problem. However, the task can be approximately solved in an attractive way by QR factorization with column pivoting [35]. With this algorithm, $\mathbf{C}_{\mathcal{P}2}$ is factorized as

$$\mathbf{C}_{\mathcal{P}2}\Pi = \mathbf{Q}\mathbf{U}, \tag{5.4}$$

where $\mathbf{Q}$ is orthogonal and $\mathbf{U}$ is upper triangular. By solving for $\mathbf{C}_{\mathcal{P}2}$ in (5.4) and substituting into (5.2) followed by multiplication from the left with $\begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{Q}^T \end{bmatrix}$, we get

$$\begin{bmatrix} \mathbf{U}_{\mathcal{R}} & \mathbf{C}_{\mathcal{P}1}\Pi \\ \mathbf{0} & \mathbf{U} \end{bmatrix} \begin{bmatrix} \mathbf{X}_{\mathcal{R}} \\ \Pi^T\mathbf{X}_{\mathcal{P}} \end{bmatrix} = 0. \tag{5.5}$$

We observe that $\mathbf{U}$ is in general not square and write $\mathbf{U} = \begin{bmatrix} \mathbf{U}_{\mathcal{P}'2} & \mathbf{C}_{\mathcal{B}2} \end{bmatrix}$, where $\mathbf{U}_{\mathcal{P}'2}$ is square upper triangular. We also write $\mathbf{C}_{\mathcal{P}1}\Pi = \begin{bmatrix} \mathbf{C}_{\mathcal{P}'1} & \mathbf{C}_{\mathcal{B}1} \end{bmatrix}$ and $\Pi^T\mathbf{X}_{\mathcal{P}1} = \begin{bmatrix} \mathbf{X}_{\mathcal{P}'1} & \mathbf{X}_{\mathcal{B}} \end{bmatrix}^T$ yielding

$$\begin{bmatrix} \mathbf{U}_{\mathcal{R}} & \mathbf{C}_{\mathcal{P}'1} & \mathbf{C}_{\mathcal{B}1} \\ \mathbf{0} & \mathbf{U}_{\mathcal{P}'2} & \mathbf{C}_{\mathcal{B}2} \end{bmatrix} \begin{bmatrix} \mathbf{X}_{\mathcal{R}} \\ \mathbf{X}_{\mathcal{P}'} \\ \mathbf{X}_{\mathcal{B}} \end{bmatrix} = 0. \tag{5.6}$$

Notice here that $\mathcal{P}$ has now split into the set $\mathcal{B}$ which is the new smaller basis and $\mathcal{P}'$, which can now be expressed in terms of the elements in $\mathcal{B}$. Finally the expression

$$\begin{bmatrix} \mathbf{X}_{\mathcal{R}} \\ \mathbf{X}_{\mathcal{P}'} \end{bmatrix} = -\begin{bmatrix} \mathbf{U}_{\mathcal{R}} & \mathbf{C}_{\mathcal{P}'1} \\ \mathbf{0} & \mathbf{U}_{\mathcal{P}'2} \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{C}_{\mathcal{B}1} \\ \mathbf{C}_{\mathcal{B}2} \end{bmatrix} \mathbf{X}_{\mathcal{B}} \tag{5.7}$$

is analogous to (4.12) and amounts to solving r upper triangular equation systems which can be efficiently done by back substitution.

The reason why QR factorization fits so nicely within this framework is that it simultaneously solves the two tasks of reduction to upper triangular form and numerically sound basis selection and with comparable effort to normal Gaussian elimination.

Furthermore, QR factorization with column pivoting is a widely used and well studied algorithm and there exist free, highly optimized implementations, making this an accessible approach.

Standard QR factorization successively eliminates elements below the main diagonal by multiplying from the left with a sequence of orthogonal matrices (usually Householder transformations). For matrices with more columns than rows (under-determined systems) this algorithm can produce a rank-deficient $\mathbf{U}$ which would then cause the computations in this section to break down. QR with column pivoting solves this problem by, at iteration k, moving the column with greatest 2-norm on the last $m-k+1$ elements to position k and then eliminating the last $m-k$ elements of this column by multiplication with an orthogonal matrix Q_k.

5.2.2 The SVD Method

By considering not only monomial bases, but more general polynomial bases it is possible to further improve numerical stability. In this subsection it is shown how the singular value decomposition (SVD) can be used to construct a basis for $\mathbb{C}[\mathbf{x}]/I$ as r linearly independent linear combinations of elements in a solving basis $\mathcal{P}$.

As in Section 5.2.1 we start out by selecting an as large as possible (redundant) solving basis and perform preliminary matrix operations forming (5.2), where the aim is now to construct a linearly independent basis from $\mathcal{P}$. We do this by performing an SVD on $\mathbf{C}_{\mathcal{P}2}$, writing

$$\mathbf{C}_{\mathcal{P}2} = \mathbf{U}\Sigma\mathbf{V}^T, \tag{5.8}$$

where $\mathbf{U}$ and $\mathbf{V}$ are orthogonal and Σ is diagonal with typically r last elements zero $\Sigma = \left[\begin{smallmatrix}\Sigma' & \mathbf{0}\\ \mathbf{0} & \mathbf{0}\end{smallmatrix}\right]$ for a system with r solutions.

Now inserting this into (5.2) and multiplying from the left with $\left[\begin{smallmatrix}\mathbf{I} & \mathbf{0}\\ \mathbf{0} & \mathbf{U}^T\end{smallmatrix}\right]$, we get

$$\begin{bmatrix}\mathbf{U}_{\mathcal{R}} & \mathbf{C}_{\mathcal{P}1}\mathbf{V}\\ \mathbf{0} & \Sigma\end{bmatrix}\begin{bmatrix}\mathbf{X}_{\mathcal{R}}\\ \mathbf{V}^T\mathbf{X}_{\mathcal{P}}\end{bmatrix} = 0. \tag{5.9}$$

The matrix $\mathbf{V}$ induces a change of basis in the space spanned by $\mathcal{P}$ and we write $\tilde{\mathbf{X}}_{\mathcal{P}} = \mathbf{V}^T\mathbf{X}_{\mathcal{P}} = \left[\mathbf{x}'_{\mathcal{P}}\ \mathbf{x}_{\mathcal{B}}\right]^T$, where $\mathcal{P}'$ and $\mathcal{B}$ are sets of polynomials. Using this notation we get

$$\begin{bmatrix}\mathbf{U}_{\mathcal{R}} & \mathbf{0} & \tilde{\mathbf{C}}_{\mathcal{P}1}\\ \mathbf{0} & \Sigma' & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \mathbf{0}\end{bmatrix}\begin{bmatrix}\mathbf{X}_{\mathcal{R}}\\ \mathbf{X}_{\mathcal{P}'}\\ \mathbf{X}_{\mathcal{B}}\end{bmatrix} = 0, \tag{5.10}$$

where Σ' is diagonal with $n-r$ nonzero diagonal entries. The zeros above Σ' enter since Σ' can be used to eliminate the corresponding elements without affecting any other elements in the matrix. In particular this means that we have

$$\begin{cases}\mathbf{X}_{\mathcal{P}'} &= \mathbf{0}\\ \mathbf{X}_{\mathcal{R}} &= -\mathbf{U}_{\mathcal{R}}^{-1}\tilde{\mathbf{C}}_{\mathcal{P}1}\mathbf{X}_{\mathcal{B}}\end{cases} \tag{5.11}$$

on V, which allows us to express any elements in $\mathrm{span}(\mathcal{M})$ in terms of $\mathbf{X}_{\mathcal{B}}$, which makes $\mathcal{B}$ a solving basis.

Computing the action matrix is complicated slightly by the fact that we are now working with a polynomial basis rather than a monomial one. To deal with this situation we introduce some new notation. To each element e_k of $\tilde{\mathcal{P}} = \mathcal{P}' \bigcup \mathcal{B}$ we assign a vector $v_k = \left[0\ \ldots\ 1\ \ldots\ 0\right]^T \in \mathbb{R}^{|\tilde{\mathcal{P}}|}$, with a one at position k. Similarly, we introduce vectors $u_k \in \mathbb{R}^{|\mathcal{R}\bigcup\tilde{\mathcal{P}}|}, w_k \in \mathbb{R}^{|\mathcal{B}|}$ representing elements of $\mathcal{R}\bigcup\tilde{\mathcal{P}}$ and $\mathcal{B}$ respectively. Further we define the linear mapping $R : \mathrm{span}(\mathcal{R}\bigcup\tilde{\mathcal{P}}) \mapsto \mathrm{span}(\mathcal{B})$, which using (5.11) associates an element of $\mathrm{span}(\mathcal{R}\bigcup\tilde{\mathcal{P}})$ with an element in $\mathrm{span}(\mathcal{B})$. We represent R by a $|\mathcal{B}| \times |\mathcal{R}\bigcup\tilde{\mathcal{P}}|$ matrix

$$\mathbf{R} = \left[-\tilde{\mathbf{C}}_{\mathcal{P}'}^T\mathbf{U}_{\mathcal{R}}^{-1T} \quad \mathbf{0} \quad \mathbf{I}\right], \tag{5.12}$$

acting on the space spanned by the vectors u_k.

We also introduce the mapping $M_p : \text{span}(\mathcal{P}) \mapsto \text{span}(\mathcal{R} \bigcup \mathcal{P})$ given by $M_p(f) = p \cdot f$ with the representation

$$(\mathbf{M}_p)_{ij} = I(x^{\alpha_i} = p \cdot x^{\alpha_j}), \tag{5.13}$$

where $I(\cdot)$ is the indicator function.

$\mathbf{M}_p$ represents multiplication by p on $\mathcal{P}$. In the basis $\tilde{\mathcal{P}}$ induced by the change of basis $\mathbf{V}$ we thus get

$$\tilde{\mathbf{M}}_p = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{V}^T \end{bmatrix} \mathbf{M}_p \mathbf{V}. \tag{5.14}$$

Finally, we get a representation of the multiplication mapping from $\mathcal{B}$ to $\mathcal{B}$ as

$$\tilde{\mathbf{m}}_p^T = \mathbf{R}\tilde{\mathbf{M}}_p\mathbf{L}, \tag{5.15}$$

where $\mathbf{L} = \left[\begin{smallmatrix} \mathbf{0} \\ \mathbf{I} \end{smallmatrix}\right]$ simply interprets the $w_k \in \mathbb{R}^{|\mathcal{B}|}$ vectors as $\mathbb{R}^{|\tilde{\mathcal{P}}|}$-vectors. The transpose on $\tilde{\mathbf{m}}_p$ in the above equation shows up because we derived the expression using the representation vectors u_k, v_k, w_k rather than directly with monomial vectors.

An eigendecomposition of $\tilde{\mathbf{m}}_p^T$ yields a set of eigenvectors $\tilde{v}$ in the new basis. It remains to inverse transform these eigenvectors to obtain eigenvectors of $\mathbf{m}_p^T$, which is the corresponding multiplication matrix in the space span$(\mathcal{P})$. Writing

$$\begin{bmatrix} \mathbf{X}_{\tilde{\mathcal{P}}} \\ \mathbf{X}_{\mathcal{B}} \end{bmatrix} = \begin{bmatrix} \mathbf{V}_1^T \\ \mathbf{V}_2^T \end{bmatrix} \mathbf{X}_{\mathcal{P}},$$

we get

$$\tilde{\mathbf{m}}_p^T = \mathbf{V}_2^T \mathbf{m}_p \mathbf{V}_2. \tag{5.16}$$

Assume now that $\tilde{v}$ is an eigenvector of $\tilde{\mathbf{m}}_p^T$. Using the above expression, we can see directly that $\tilde{v} = \mathbf{V}_2^T v$ is an eigenvector for $\tilde{\mathbf{m}}_p^T$ matrix iff v is an eigenvector of $\mathbf{m}_p^T$. This yields

$$v = \mathbf{V}_2 \tilde{v} \tag{5.17}$$

and hence we have a way of going back to our original basis where we can read off the solutions to our equations.

As will be seen in the experiments, the SVD method is somewhat more stable than the QR method, but significantly slower due to the costly SVD factorization.

5.2.3 Basis Selection and Adaptive Truncation

We have so far seen three techniques for dealing with the case when the sub matrix $\mathbf{C}_{\mathcal{P}2}$ is ill conditioned. By the method in Section 5.1 we avoid operating on $\mathbf{C}_{\mathcal{P}2}$ altogether. Using, the QR and SVD methods we perform elimination, but in a numerically much more stable manner. One might now ask whether it is possible to combine these methods. Indeed it turns out that we can combine either the QR or the SVD method with a redundant solving basis to get an adaptive truncation criterion yielding even better stability in some cases. The

way to do this is to choose a criterion for early stopping in the factorization algorithms. The techniques in this section are related to truncation schemes for rank-deficient linear least squares problems, *cf* [50].

A neat feature of QR factorization with column pivoting is that it provides a way of numerically estimating the conditioning of $\mathbf{C}_{\mathcal{P}2}$ simultaneously with elimination. By design, the QR factorization algorithm produces an upper triangular matrix $\mathbf{U}$ with diagonal elements u_{ii} of decreasing absolute value. The factorization proceeds column wise, producing one $|u_{ii}|$ at a time. If $\text{rank}(\mathbf{U}) = r$, then $|u_{rr}| > 0$ and $u_{r+1,r+1} = \cdots = u_{nn} = 0$. However, in floating point arithmetic, the transition from finite $|u_{ii}|$ to zero is typically gradual passing through extremely small values and the rank is consequently hard to determine. For robustness it might therefore be a good idea to abort the factorization process early. We do this by setting a threshold τ for the ratio $|\frac{u_{11}}{u_{ii}}|$ and abort the factorization once the value exceeds this threshold. A value of $\tau \approx 10^8$ has been found to yield good results[1]. Note that this produces an equivalent result to carrying out the full QR factorization and then simply discarding the last rows of $\mathbf{U}$. This is practical since off-the-shelf packages as LAPACK only provide full QR factorization, even though some computational effort could be spared by modifying the algorithm so as not to carry out the last steps.

Compared to setting a fixed (redundant) basis size, this approach is beneficial since both rank and conditioning of $\mathbf{C}_{\mathcal{P}2}$ might depend on the data. By the above method we decide adaptively where to truncate and therefore how large the linear basis for $\mathbb{C}[\mathbf{x}]/I$ should be.

In the context of the SVD we get a similar criterion by looking at the singular values instead and set a threshold for $\frac{\sigma_1}{\sigma_i}$, which for $i = \text{rank}(\mathbf{C}_{\mathcal{P}2})$ is exactly the condition number of $\mathbf{C}_{\mathcal{P}2}$.

5.3 Other Techniques

We end the part on techniques in this chapter with two less involved but still useful ideas.

5.3.1 A Single Elimination Step

In previous works which have been more closely connected to classical algebraic geometry using properly defined monomial orders *etc*, a Gröbner basis for the particular ideal has typically been obtained by successive elimination and addition of equations [76, 55]. This is also more similar to how the original Buchberger's algorithm for computing a Gröbner basis works. We strongly advocate avoiding this and instead first adding all equations and then doing the full elimination in one go. The reason for this is that, as mentioned often in this text, the eliminations tend to be ill conditioned. If several elimination steps are interleaved with addition of new equations, numerical errors accumulate and the algorithms easily become unstable.

[1] Performance is not very sensitive to the choice of τ and values in the range 10^6 to 10^{10} yield similar results.

5.3.2 Using Eigenvalues Instead of Eigenvectors

In the literature, the preferred method of extracting solutions using eigenvalue decomposition is to look at the eigenvectors. It is also possible to use the eigenvalues, but for a problem with s variables, this seemingly requires us to solve s eigenvalue problems since each eigenvalue only gives the value of one variable. However, there can be an advantage with using the eigenvalues instead of eigenvectors. If there are multiple eigenvalues (or almost multiple eigenvalues) the computation of the corresponding eigenvectors will be numerically unstable. However, the eigenvalues can usually be determined with reasonable accuracy. In practice, this situation is not uncommon with the action matrix.

Fortunately, we can make use of our knowledge of the eigenvectors to devise a scheme for quickly finding the eigenvalues of any action matrix on $\mathbb{C}[\mathbf{x}]/I$. From Section 2.2 we know that the right eigenvectors of an action matrix is the vector of basis elements of $\mathbb{C}[\mathbf{x}]/I$ evaluated at the zeros of I. This holds for *any* action matrix and hence all action matrices have the same set of eigenvectors. Consider now a problem involving the two variables x_i and x_j. If we have constructed $\mathbf{m}_{x_i}$, the construction of $\mathbf{m}_{x_j}$ requires almost no extra time. Now perform an eigenvalue decomposition $\mathbf{m}_{x_i} = \mathbf{V}\mathbf{D}_{x_i}\mathbf{V}^{-1}$. Since $\mathbf{V}$ is the set of eigenvectors for $\mathbf{m}_{x_j}$ as well, we get the eigenvalues of $\mathbf{m}_{x_j}$ by straightforward matrix multiplication and then element wise division from

$$\mathbf{m}_{x_j}\mathbf{V} = \mathbf{V}\mathbf{D}_{x_j}. \tag{5.18}$$

This means that with very little extra computational effort over a single eigenvalue decomposition we can obtain the eigenvalues of all action matrices we need.

5.4 Experimental Validation

In this section we evaluate the numerical stability of the proposed techniques on a range of typical geometric computer vision problems. The experiments are mainly carried out on synthetic data since we are interested in the intrinsic numerical precision of the solver. By intrinsic precision we mean precision under perfect data. The error under noise is of course interesting for any application, but for minimal data this is an effect of the problem formulation and *not* of the particular equation solving technique.

In Section 5.4.1 all the main methods (standard, truncated, SVD and QR) are tested on the problem of optimal triangulation from three different views. This problem was first studied in [80] where emulated 128 bit arithmetics was necessary to get usable results. Later, with the techniques presented in this thesis the problem was given an efficient implementation in standard IEEE double precision. Details of this are given in Chapter 6. However, this example provides such a nice illustration of the relative benefits and drawbacks of the different techniques so we take the liberty of borrowing some of the results and present them already in this section.

Apart from the triangulation example, the improved methods are tested on the problems of relative pose with unknown but common focal length [78] and relative pose for generalized cameras [79]. Significant improvements in stability are shown in all cases.

5.4.1 Optimal Three View Triangulation

The triangulation problem is formulated as finding the world point that minimizes the sum of squares of the reprojection errors in the three views. We do this by computing the gradient of the sum of squares error and setting it to zero. This yields three sixth degree polynomial equations in three variables (the X, Y and Z coordinates of the unknown point) and using *e.g* a computer algebra system one can check that the system has 47 (real and complex) zeros. After some preliminary manipulations described in Chapter 6 we expand the set of equations up to degrees 9 (see the beginning of Section 4.1.2) yielding 225 equations in 209 different monomials.

The synthetic data used in the validation was generated with three randomly placed cameras at a distance around 1000 from the origin and a focal length of around 1000. The unknown world point was randomly placed in a cube with side length 1000 centered at the origin. The methods have been compared on 100,000 test cases.

Numerical Experiments

The first experiment investigates what improvement can be achieved by simply avoiding the problematic matrix elimination using the techniques of Section 5.1. For this purpose we choose the complete set of permissible monomials $\mathcal{P}$ as a redundant basis and perform the steps in the algorithm Solving Basis Method. In this case we thus get a redundant basis of 154 elements and a 154×154 multiplication matrix to perform eigenvalue decomposition on. In both cases the eigenvectors are used to find the solutions. The results of this experiment are shown in Figure 5.2. As can be seen, this relatively straightforward technique already yields a large improvement in numerical stability.

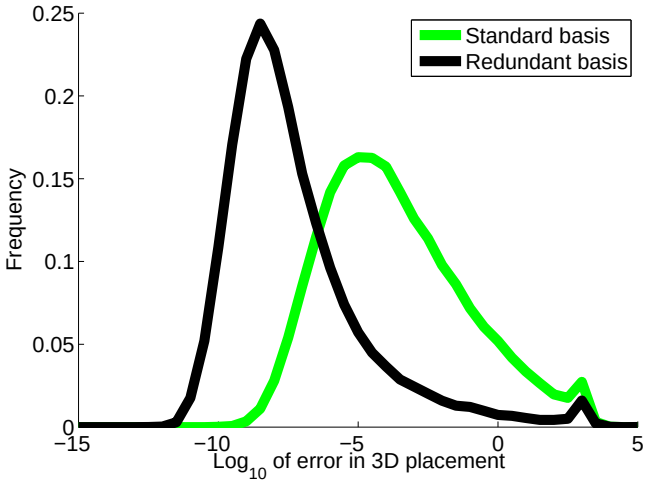

Figure 5.2: Histogram of errors over 100,000 points. The improvement in stability using the redundant basis renders the algorithm feasible in standard IEEE double precision.

Looking closely at Figure 5.2 one can see that even though the general stability is much improved, a small set of relatively large errors remain. It is unclear

what causes these errors. However, by doing some extra work using the QR method of Section 5.2.1 to select a true basis as a subset of $\mathcal{P}$, we improve stability further in general and in particular completely resolve the issue with large errors, *cf* Figure 5.3. Moreover, we get a smaller eigenvalue decomposition and hence reduce computational complexity.

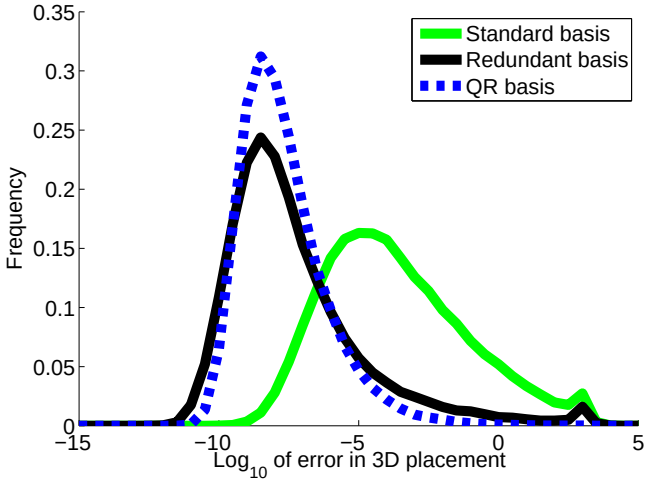

Figure 5.3: Histogram of errors for the standard, redundant basis and QR methods. The QR method improves stability in general and in particular completely removes the small set of large errors present in both the standard and redundant basis methods.

In Figure 5.4, the performance of the QR method is compared to the slightly more stable SVD method which selects a polynomial basis for $\mathbb{C}[\mathbf{x}]/I$ from the monomials in $\mathcal{P}$. In this case, errors are typically a factor ~ 5 smaller for the SVD method compared to the QR method.

The reason that a good choice of basis improves the numerical stability is that the condition number in the elimination step can be lowered considerably. Using the basis selection methods, the condition number is decreased by about a factor 10^5. Figure 5.5 shows a scatter plot of error versus condition number for the three view triangulation problem. The SVD method displays a significant decrease and concentration in both error and condition number. In theory, there could be many possible sources of numerical error in the complete solving procedure. It is therefore interesting to note that to a reasonable approximation we have a linear trend between the final error and the condition number of $\mathbf{C}_{\mathcal{R}2}$. This can be seen since we have a linear trend with slope one in the logarithmic scale. Moreover, we have a y-axis intersection at about 10^{-13}, since the coordinates are around 1000 in magnitude this means that we have a relative error $\approx 10^{-16}\kappa = \epsilon_{mach}\kappa$. This observation justifies our strategy of minimizing the condition number of $\mathbf{C}_{\mathcal{R}2}$.

As mentioned in Section 5.3.2, it might be beneficial to use the eigenvalues instead of eigenvectors to extract solutions.

When solving this problem using eigenvalues there are two extra eigenvalue problems of size 50×50 that need to be solved. The impact of the switch from eigenvectors to eigenvalues is shown in Figure 5.6. For this example we gain

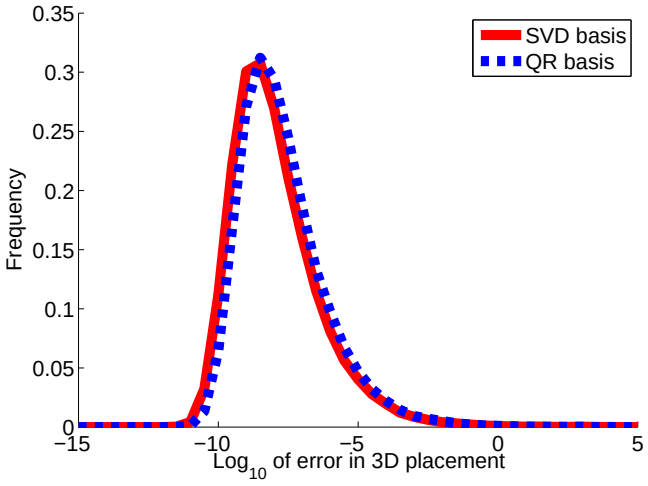

Figure 5.4: Comparison between the SVD and QR methods. The SVD method improves somewhat over the QR method at the cost of the computationally more demanding SVD factorization.

some stability at the cost of having to perform three eigenvalue decompositions (one for each coordinate) instead of only one. Moreover, we need to sort the eigenvalues using the eigenvectors to put together the correct triplets.

However, we can use the trick of Section 5.3.2 to get nearly the same accuracy using only a single eigenvalue decomposition. Figure 5.7 shows the results of this method. The main advantage of using the eigenvalues is that we push down the number of large errors considerably.

Finally we study the combination of basis selection and early stopping which yields a redundant solving basis for the three view triangulation problem. The basis size was determined adaptively as described in Section 5.2.3 with a threshold $\tau = 10^8$. Table 5.1 shows the distribution of basis sizes obtained when this method was used. Since the basis is chosen minimal in 94% of the cases for the SVD-method and 95% for the QR method the time consumption is almost identical to the original basis selection methods, but as can be seen in Table 5.2 the number of large errors are reduced. This is probably due to the fact that truncation is carried out only when the matrices are close to being singular.

	50	51	52	53	54	≥ 55
SVD	94.0	3.5	0.8	0.4	0.3	1.0
QR	95.0	3.0	0.7	0.3	0.2	0.8

Table 5.1: Basis sizes for the QR and SVD methods with variable basis size. The table shows the percentage of times certain basis sizes occurred during 100,000 experiments.

To conclude the numerical experiments on three view triangulation two tables with detailed error statistics are given. The acronyms *STD, QR, SVD* and *TRUNC* respectively denote the standard method, QR method, SVD method and redundant basis method. The suffixes *eig, fast* and *var* respectively denote

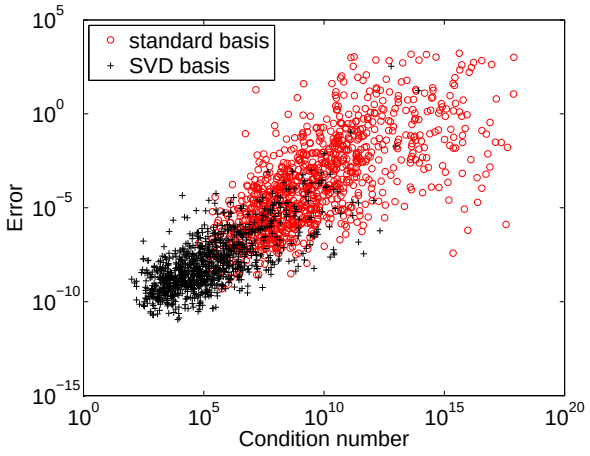

Figure 5.5: Error versus condition number for the part of the matrix which is inverted in the solution procedure.

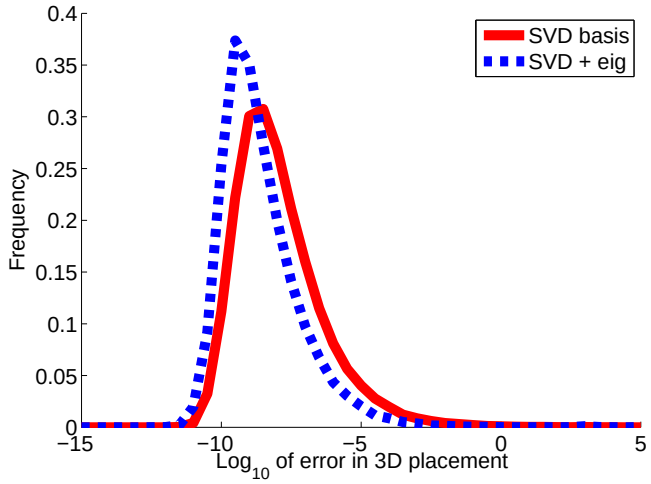

Figure 5.6: Error histograms showing the difference in precision between the eigenvalue and eigenvector methods.

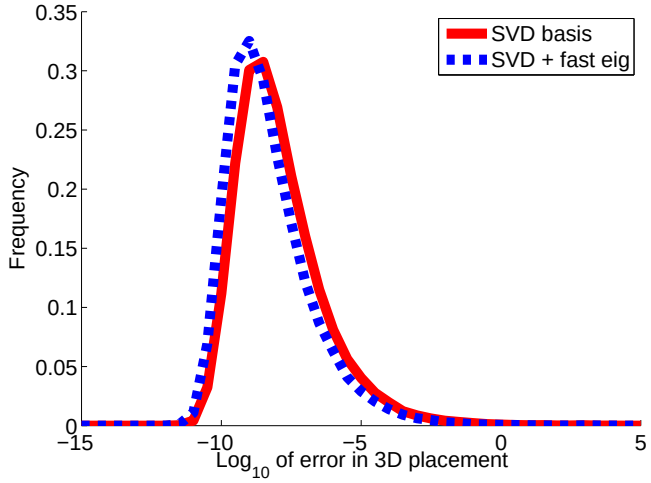

Figure 5.7: This graph shows the increase in performance when the fast eigenvalue method is used instead of the eigenvector method.

the eigenvalue method, the fast eigenvalue method (Section 5.3.2) and the use of a variable size basis (Section 5.2.3). Table 5.2 shows how many times the error was larger than some given levels for several solvers. This is interesting for example in the context of RANSAC a process. As can be seen, the QR-method with adaptive basis size is the best method for reducing the largest errors but the SVD-method with the use of eigenvalues is the best in general. Table 5.3 shows the median and the 95:th percentile errors for the same methods as in the previous table. Notable in here is that the 95:th percentile is improved with as much as factor 10^7 and the median with a factor 10^5. The SVD-method with eigenvalues is shown to be the best but the QR-method gives almost as good results.

Method	$> 10^{-3}$	$> 10^{-2}$	$> 10^{-1}$	> 1
STD	35633	24348	15806	9703
STD:eig	29847	19999	12690	7610
SVD	1173	562	247	119
SVD:eig	428*	222*	128*	94
SVD:fast	834	393	178	94
SVD:var+fast	730	421	245	141
TRUNC	6712	4697	3339	2384
TRUNC:fast	5464	3892	2723	2015
QR	1287	599	269	127
QR:eig	517	250	149	117
QR:fast	1043	480	229	106
QR:var+fast	584	272	141	71*

Table 5.2: Number of errors out of 100,000 experiments larger than certain levels. The QR-method with adaptive basis size yields the fewest number of large errors. The best result in each column is marked with an *.

Method	95th	50th
STD	$1.42 \cdot 10^{1}$	$9.85 \cdot 10^{-5}$
STD:eig	$5.30 \cdot 10^{0}$	$3.32 \cdot 10^{-5}$
SVD	$1.19 \cdot 10^{-5}$	$6.09 \cdot 10^{-9}$
SVD:eig	$1.20 \cdot 10^{-6}*$	$1.29 \cdot 10^{-9}*$
SVD:fast	$4.37 \cdot 10^{-6}$	$2.53 \cdot 10^{-9}$
SVD:var+fast	$2.34 \cdot 10^{-6}$	$2.50 \cdot 10^{-9}$
TRUNC	$6.55 \cdot 10^{-3}$	$1.40 \cdot 10^{-8}$
TRUNC:fast	$1.87 \cdot 10^{-3}$	$3.27 \cdot 10^{-9}$
QR	$1.78 \cdot 10^{-5}$	$1.06 \cdot 10^{-8}$
QR:eig	$1.70 \cdot 10^{-6}$	$2.08 \cdot 10^{-9}$
QR:fast	$6.97 \cdot 10^{-6}$	$3.64 \cdot 10^{-9}$
QR:var+fast	$3.41 \cdot 10^{-6}$	$3.61 \cdot 10^{-9}$

Table 5.3: The 95th percentile and the median error for various methods. The improvement in precision is up to a factor 10^7. The SVD method gives the best results, but the QR-method is not far off. The best result for each column is marked with an *.

Speed Comparison

The main motivation for using the QR-method rather than the SVD-method is that the QR-method is computationally less expensive. To verify this the standard, SVD and QR-methods were run and the time was measured. Since the implementations were done in Matlab it was necessary to take care to eliminate the effect of Matlab being an interpreted language. To do this only the time after construction of the coefficient matrix was taken into account. This is because the construction of the coefficient matrix essentially amounts to copying coefficients to the right places which can be done extremely fast in *e.g* a C language implementation.

In the routines that were measured no subroutines were called that were not built-in functions in Matlab. The measurements were done with the Matlab profiler.

The time measurements were done on an Intel Core 2 2.13 GHz machine with 2 GB memory. Each algorithm was executed with 1000 different coefficient matrices constructed from the same type of scene setups as previously. The same set of coefficient matrices was used for each method. The result is given in Table 5.4. Our results show that the QR-method is approximately three times faster than the SVD-method but 50% slower than the standard method. The reason that the redundant basis method is more than twice as slow as the QR method is the larger eigenvalue decomposition which dominates the computation time.

Method	Time per call / ms	Relative time
SVD	66.89	1
TRUNC	55.84	0.83
QR	24.45	0.37
STD	16.44	0.25

Table 5.4: Time consumption in the solver part for the three different methods. The time is an average over 1000 function calls.

5.4.2 Relative Pose with Unknown Focal Length

Relative pose for calibrated cameras is a well known problem and the standard minimal case for this is five points in two views. There are in general ten solutions to this problem. For the same problem but with unknown focal length, the corresponding minimal case is six points in two views, which was solved by Stewénius *et al* using Gröbner basis techniques [78].

Following the same recipe as Stewénius *et al* it is possible to express the fundamental matrix as a linear combination,

$$F = F_0 + F_1 l_1 + F_2 l_2. \tag{5.19}$$

Then setting $f^{-2} = p$ one obtains nine equations from the constraint on the essential matrix [71]

$$2EE^T E - \mathrm{tr}(EE^T)E = 0. \tag{5.20}$$

A 10th equation is obtained by making use of the fact that the fundamental matrix i singular, *i.e* $\det(F) = 0$. These equations involve the unknowns p, l_1 and l_2 and are of total degree 5. The problem has 15 solutions in general.

We set up the coefficient matrix $\mathbf{C}$ by multiplying these ten equations by p so that the degree of p reaches a maximum of four. This gives 34 equations in a total of 50 monomials.

The validation data was generated with two cameras of equal focal length of around 1000 placed at a distance of around 1000 from the origin. The six points were randomly placed in a cube with side length 1000 centered at the origin. The standard, SVD, and QR-methods have been compared on 100,000 test cases and the errors in focal length are shown in Figure 5.8. In this case the QR-method actually yields slightly better results than the SVD-method.

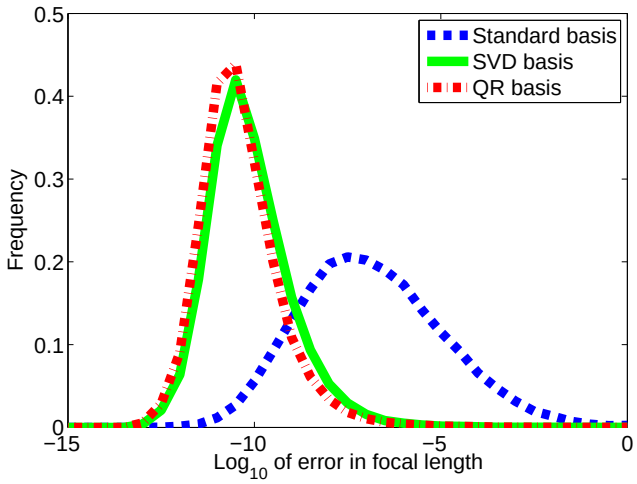

Figure 5.8: The error in focal length for relative pose with two semi calibrated cameras with unknown but common focal length.

5.4.3 Relative Pose for Generalized Cameras

Generalized cameras provide a generalization of the standard pin-hole camera in the sense that there is no common focal point through which all image rays pass, *cf* [72]. Instead the camera captures arbitrary image rays or lines. Solving for the relative motion of a generalized camera can be done using six point correspondences in two views. This is a minimal case which was solved in [79] with Gröbner basis techniques. The problem equations can be set up using quaternions to parameterize the rotation, Plücker representation of the lines and a generalized epipolar constraint which captures the relation between the lines. After some manipulations one obtains a set of sixth degree equations in the three quaternion parameters v_1, v_2 and v_3. For details, see [79]. The problem has 64 solutions in general.

To build our solver including the change of basis we multiply an original set of 15 equations with all combinations of $1, v_1, v_2, v_3$ up to degree two. After this we end up with 101 equations of total degree 8 in 165 different monomials.

We generate synthetic test cases by drawing six points from a normal distribution centered at the origin. Since the purpose of this investigation is not to study generalized cameras under realistic conditions we have not used any particular camera rig. Instead we use a completely general setting where the cameras observe six randomly chosen lines each through the six points. There is also a random relative rotation and translation relating the two cameras. It is the task of the solver to calculate the rotation and translation.

The methods have been compared on a data set of 10,000 randomly generated test cases. The results from this experiment are shown in Figure 5.9. As can be seen, a good choice of basis yields drastically improved numerical precision over the standard method.

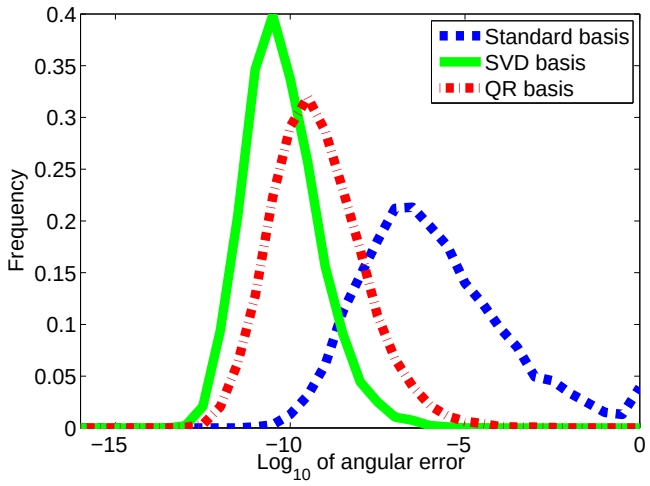

Figure 5.9: The angular error for relative pose with generalized cameras.

5.5 Discussion

We have introduced some new theoretical ideas as well as a set of techniques designed to overcome numerical problems encountered in state-of-the-art methods for polynomial equation solving. We have shown empirically that these techniques in many cases yield dramatic improvements in numerical stability and further permits the solution of a larger class of problems than previously possible.

The techniques for solving polynomial equations that are used in this work can be summarized as follows. The original equations are first expanded by multiplying the polynomials with a set of monomials. The resulting equations is expressed as a product of a coefficient matrix $\mathbf{C}$ and a monomial vector $\mathbf{X}$. Here we have some freedom in choosing which monomials to multiply with. We then try to find a solving basis $\mathcal{B}$ for the problem. For a given candidate basis $\mathcal{B}$ we have shown how to determine if $\mathcal{B}$ constitutes a solving basis. If so then we can use numerical linear algebra to construct the action matrix and get a fast and numerically stable solution to the problem at hand. However, we do not know (i) what monomials we should multiply the original equations with

and (ii) what solving basis $\mathcal{B}$ should be used to get the simplest and most numerically stable solutions. Are there algorithmic methods for answering these questions? For a given expansion $\mathbf{CX}$ can one determine if this allows for a solving basis? Such questions can be answered to some extent using existing theory for Gröbner bases and exact arithmetic. However in the context of general (possibly overcomplete) bases for $\mathbb{C}[\mathbf{x}]/I$ and non-strict monomial orderings there is so far much less we can say. A concise theoretical understanding and practical algorithms for these problems would certainly be of great aid in the work on polynomial problems and is an interesting subject for future research.

Part II

Applications of Polynomial Equation Solving in Computer Vision

Chapter 6

Optimal Triangulation

In this chapter we consider the problem of globally optimal triangulation from three separate views. Whereas, the two-view case has a relatively simple closed form solution, the three-view case has just the right complexity to make it an excellent target for the techniques introduced in Chapter 5. For four or more views though, optimization by solving a polynomial is still more or less infeasible.

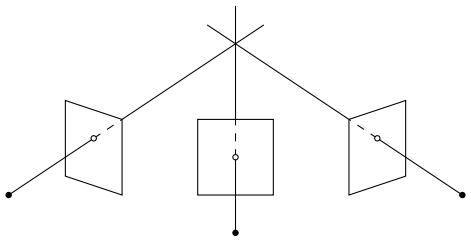

Figure 6.1: The unknown location of a point can be reconstructed using its projection in a sequence of images if the location and orientation of the cameras are known. This is usually called triangulation.

6.1 Introduction

Triangulation, referring to the act of reconstructing the 3D location of a point given its images in two or more known views, is an important part of numerous computer vision systems. Albeit conceptually simple, this problem is not completely solved in the general case of n views and noisy measurements.

There exist fast and relatively robust methods based on linear least squares [40]. These methods are however sub-optimal. Moreover, the linear least squares formulation does not have a clear geometrical meaning, which means that in unfortunate situations, this approach can yield very poor accuracy.

The most desirable, but non-linear, approach is instead to minimize the L_2 norm of the reprojection error, *i.e* the sum of squares of the reprojection errors. The reason for this is that the L_2 optimum yields the maximum likelihood estimate for the 3D point under the assumption of independent Gaussian noise on the image measurements [39]. This problem has been given a closed form

solution[1] by Hartley and Sturm in the case of two views [39]. However, the approach of Hartley and Sturm is not straightforward to generalize to more than two views.

In the case of n views, the standard method when high accuracy is needed is to use a two-phase strategy where an iterative scheme for non-linear least squares such as Levenberg-Marquardt (Bundle Adjustment) is initialized with a linear method [86]. This procedure is reasonably fast and in general yields excellent results. One potential drawback, however, is that the method is inherently local, *i.e* finds local minima with no guarantee of being close to the global optimum.

An interesting alternative is to replace the L_2 norm with the L_∞, norm *cf* [49]. This way it is possible to obtain a provably optimal solution with a geometrically sound cost function in a relatively efficient way. The drawback is that the L_∞ norm is suboptimal under Gaussian noise and it is less robust to noise and outliers than the L_2 norm.

The most practical existing method for L_2 optimization with an optimality guarantee is to use a branch and bound approach as introduced in [2], which, however, is a computationally expensive strategy. [2]

In this work, we propose to solve the problem of L_2 optimal triangulation from three views using a method introduced by Stewénius *et al* in [80], where the optimum was found by explicit computation of the complete set of stationary points of the likelihood function. This approach is similar to that of Hartley and Sturm [39]. However, whereas the stationary points in the two view case can be found by solving a sixth degree polynomial in one variable, the easiest known formulation of the three view case involves solving a system of three sixth degree equations in three unknowns with 47 solutions. Thus, we have to resort to more sophisticated techniques to tackle this problem.

Stewénius *et al* used algebraic geometry and Gröbner basis techniques to analyze and solve the equation system. However, as previously mentioned, Gröbner basis calculations are known to be numerically challenging and they were forced to use emulated 128 bit precision arithmetics to get a stable implementation, which rendered their solution too slow to be of any practical value.

Using the new techniques presented in this thesis, we are now able to give the Gröbner basis method a fast implementation using standard IEEE double precision. By this we also show that global optimization by calculation of stationary points is indeed a feasible approach and that Gröbner basis like techniques provide a powerful tool in this pursuit.

6.2 Three View Triangulation

The main motivation for triangulation from more than two views is to use the additional information to improve accuracy. In this section we briefly outline the approach we take and derive the equations to be used in the following sections.

[1] The solution is actually not *entirely* on closed form, since it involves the solution of a sixth degree polynomial, which cannot in general be solved on closed form. Therefore one has to go by *e.g* the eigenvalues of the companion matrix, which implies an iterative process.

[2] Since the main part of the material of this chapter was written, a faster version of the branch and bound algorithm for L_2 optimal triangulation has been published [62] that probably has comparable running time to the method presented here, even though exact running times are not available for the case of three views. However, the new branch and bound method also generalizes to n views and is therefore probably a more practical choice.

This part is essentially identical to that used in [80]. We assume a linear pinhole camera model, *i.e* projection in homogeneous coordinates is done according to $\lambda_i x_i = P_i X$, where P_i is the 3×4 camera matrix for view i, x_i is the image coordinates, λ_i is the depth and X is the 3D coordinates of the world point to be determined. In standard coordinates, this can be written as

$$x_i = \frac{1}{P_{i3}X} \begin{bmatrix} P_{i1}X \\ P_{i2}X \end{bmatrix}, \tag{6.1}$$

where *e.g* P_{i3} refers to row 3 of camera i.

As mentioned previously, we aim at minimizing the L_2 norm of the reprojection errors. Since we are free to choose coordinate system in the images, we place the three image points at the origin in their respective image coordinate systems. With this choice of coordinates, we obtain the following cost function to minimize over X

$$\varphi(X) = \frac{(P_{11}X)^2 + (P_{12}X)^2}{(P_{13}X)^2} + \frac{(P_{21}X)^2 + (P_{22}X)^2}{(P_{23}X)^2} + \frac{(P_{31}X)^2 + (P_{32}X)^2}{(P_{33}X)^2}. \tag{6.2}$$

The approach we take is based on calculating the complete set of stationary points of $\varphi(X)$, *i.e* solving $\nabla\varphi(X) = 0$. By inspection of (6.2) we see that $\nabla\varphi(X)$ will be a sum of rational functions. The explicit derivatives can easily be calculated, but we refrain from writing them out here. Differentiating and multiplying through with the denominators produces three sixth degree polynomial equations in the three unknowns of $X = [X_1\ X_2\ X_3]^T$. To simplify the equations we also make a change of world coordinates, setting the last rows of the respective cameras to

$$P_{13} = [1\ 0\ 0\ 0],\ P_{23} = [0\ 1\ 0\ 0],\ P_{33} = [0\ 0\ 1\ 0]. \tag{6.3}$$

Since we multiply with the denominator we introduce new stationary points in our equations corresponding to one of the denominators in (6.2) being equal to zero. This happens precisely when X coincides with the plane through one of the focal points parallel to the corresponding image plane. Such points have infinite/undefined value of $\varphi(X)$ and can therefore safely be removed.

To summarize, we now have three sixth degree equations in three unknowns. The remainder of the theoretical part of the chapter will be devoted to the problem of solving these.

6.3 A Numerical Solution to the Three View Triangulation Problem

As discussed in Section 6.2, we optimize the L_2 cost function by calculation of the stationary points. This yields three sixth degree polynomial equations in $X = [X_1\ X_2\ X_3]^T$. In addition to this, we add a fourth equation by taking the sum of our three original equations. This cancels out the leading terms, producing a fifth degree equation which will be useful in the subsequent calculations [80]. These equations generate an ideal I in $\mathbb{C}[X]$. We start this section out by going through the previous method of trying to compute a Gröbner basis for I and explain where this method runs into problems. This serves as a

starting point for employing the methods of Chapter 5 to get a fast and stable algorithm.

First, however, we need to deal with the problem where one or more of $X_i = 0$. When this happens, we get a parametric solution to our equations. As mentioned in Section 6.2, this corresponds to the extra stationary points introduced by multiplying up denominators and these points have infinite value of the cost function $\varphi(X)$. Hence, we would like to exclude solutions with any $X_i = 0$ or equivalently $X_1X_2X_3 = 0$. The algebraic geometry way of doing this is to calculate the *saturation* $\text{sat}(I, X_1X_2X_3)$ of I w.r.t $X_1X_2X_3$, consisting of all polynomials $f(X)$ s.t. $(X_1X_2X_3)^k \cdot f \in I$ for some k.

Computationally it is easier to calculate $\text{sat}(I, X_i)$ for one variable at a time and then joining the result. This removes the same problematic parameter family of solutions, but with the side effect of producing some extra (finite) solutions with $X_i = 0$. These do not present any serious difficulties since they can easily be detected and filtered out.

Consider one of the variables, say X_1. The ideal $\text{sat}(I, X_1)$ is calculated in three steps. We order the monomials according to X_1 but take the monomial with the highest power of X_1 to be the smallest, *e.g* $X_1X_2^2X_3 \geq X_1^2X_2^2X_3$. With the monomials ordered this way, we perform a few steps of the Gröbner basis calculation, yielding a set of generators where the last elements can be divided by powers of X_1. We add these new equations which are "stripped" from powers of X_1 to I.

More concretely, we multiply the equations by all monomials creating equations up to degree seven. After the elimination step two equations are divisible by X_1 and one is divisible by X_1^2.

The saturation process is performed analogously for X_2 and X_3 producing the saturated ideal I_{sat}, from which we extract our solutions.

The final step is to calculate a multiplication matrix for I_{sat}, at this point generated by a set of nine fifth and sixth degree equations. To be able to do this we multiply with monomials creating 225 equations in 209 different monomials of total degree up to nine. The last step thus consists of putting the 225 by 209 matrix $\mathbf{C}$ on reduced row echelon form.

This last part turns out to be a delicate task though due to generally very poor conditioning. In fact, the conditioning is often so poor that round-off errors in the order of magnitude of machine epsilon (approximately 10^{-16} for doubles) yield errors as large as 10^2 or more in the final result. This is the reason one had to resort to emulated 128 bit numerics in [80].

Using the new techniques for computing the action matrix though, we can now more or less completely avoid these conditioning problems. By extensive experimentation (see Section 5.4) we have found that using the QR method (Section 5.2.1) with an adaptive basis size (Section 5.2.3) yields the best stability/speed trade-off, see Table 6.1.

6.4 Experiments

The algorithm described in this chapter has been implemented in Matlab which suggests that further gains in speed could be made by implementing it in *e.g* C. However, the main time consuming parts of the algorithm are the LU and QR factorizations and the eigenvalue decomposition of the action matrix and

	QR	Standard	Standard, 128 bit
Running time:	14ms	10ms	30s
Stability:	Good	Very poor	Good

Table 6.1: Overview of running time and stability characteristics for the new QR-based algorithm, for the previous method in double precision and for the previous method implemented in emulated 128 bit arithmetics. The previous method is only stable in the higher precision, which makes it very slow (a factor 300 slower). Using the QR method we get a fast and stable algorithm in standard double precision.

Matlab uses LAPACK and BLAS for these operations which contain state-of-the-art implementations of the above mentioned linear algebra operations. Care has been taken to make the Matlab code for the remaining operations as efficient as possible.

Experimental results for the triangulation problem have already been presented in Chapter 5, but we repeat some of them here for completeness of the chapter with the purpose of demonstrating the speed and numerical precision of the method. We have run the algorithm on both real and synthetically generated data using a 2.0 Ghz AMD Athlon X2 64 bit machine. With this setup, triangulation of one point takes approximately 13 milliseconds using the new method. This is to be contrasted with the previous implementation by Stewénius *et al* [80], which needs 30 seconds per triangulation with their setup. The branch and bound method of [2] is faster than [80] but exact running times for triangulation are not given in [2]. However, based on the performance of this algorithm on similar problems, the running time for three view triangulation is probably at least a couple of seconds using their method.

6.4.1 Synthetic Data

To evaluate the intrinsic numerical stability of the solver the algorithm has been run on 100,000 randomly generated test cases. World points were drawn uniformly from the cube $[-500, 500]^3$ and cameras were placed randomly at a distance of around 1000 from the origin with focal length of around 1000 and pointing inwards. We compare the approach presented here to that of [80] implemented in double precision here referred to as the standard method since it is based on straightforward Gröbner basis calculation. A histogram over the resulting errors in estimated 3D location is shown in Figure 6.2. As can be seen, the error is typically around a factor 10^5 smaller with the new method.

Since we consider triangulation by minimization of the L_2 norm of the error, ideally behavior under noise should not be affected by the algorithm used. In the second experiment we assert that the algorithm behaves as expected under noise. We generate data as in the first experiment and apply Gaussian noise to the image measurements in 0.1 pixel intervals from 0 to 5 pixels. We triangulate 1000 points for each noise level. The median error in 3D location is plotted versus noise in Figure 6.3. There is a linear relation between noise and error, which confirms that the algorithm is stable also in the presence of noise.

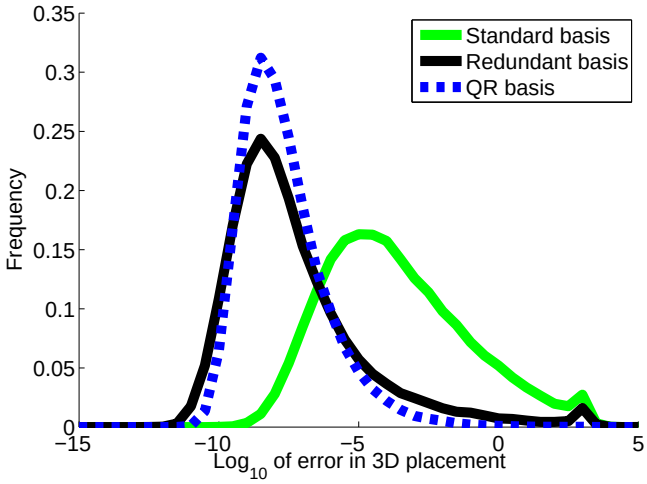

Figure 6.2: Histogram of errors for the standard, redundant basis and QR methods. The QR method improves stability in general and in particular completely removes the small set of large errors present in both the standard and redundant basis methods. Compared to the standard method, precision is improved by about a factor 10^5.

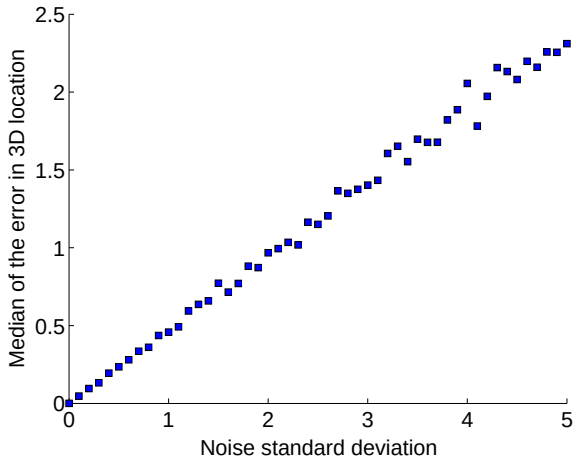

Figure 6.3: Error in 3D location of the triangulated point X as a function of image-point noise. The behavior under noise is as expected given the problem formulation.

6.4.2 A Real Example

Finally, we evaluate the algorithm under real world conditions. The Oxford dinosaur [25] is a familiar image sequence of a toy dinosaur shot on a turn table. The image sequence consists of 36 images and 4983 point tracks. For each point visible in three or more views we select the first, middle and last view and triangulate using these. This yields a total of 2683 point triplets to triangulate

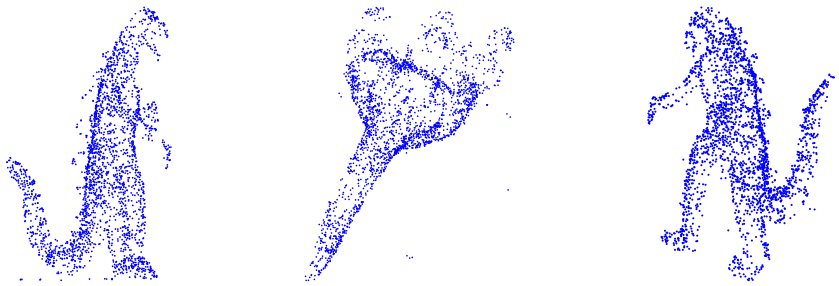

Figure 6.4: The Oxford dinosaur reconstructed from 2683 point triplets using the QR-method with variable basis size. The reconstruction was completed in approximately 34 seconds.

from. The image sequence contains some erroneous tracks which we deal with by removing any points reprojected with an error greater than two pixels in any frame. The whole sequence was processed in approximately 34 seconds and the resulting point cloud is shown in Figure 6.4.

We have also run the same sequence using the previous method implemented in double precision, but the errors were too large to yield usable results. Note that [80] contains a successful triangulation of the dinosaur sequence, but this was done using extremely slow emulated 128 bit arithmetic yielding an estimated running time of 20h for the whole sequence.

6.5 Conclusions

In this chapter we have shown how a typical problem from computer vision, triangulation, can be solved for the globally optimal L_2 estimate using Gröbner basis like techniques. With the new techniques for equation solving, we have taken this approach to a state where it can now have practical value in actual applications. In all fairness though, linear initialization combined with bundle adjustment will probably remain the choice for most applications since this is still significantly faster and gives excellent accuracy. However, if a guarantee of finding the provably optimal solution is desired, we provide a competitive method.

More importantly perhaps, by this example we show that global optimization by calculation of the stationary points using Gröbner basis techniques is indeed a possible way forward. This is particularly interesting since a large number of computer vision problems ultimately depend on some form of optimization.

Chapter 7

Hybrid Minimal Problems

Camera estimation tasks are usually divided into (i) absolute orientation estimation, where a set of world points with known coordinates correspond to a set of image points and the task is to determine the exact pose of the camera and (ii) relative orientation estimation, where two or more cameras view the same scene and a set of corresponding points between images are given. In this chapter we investigate the mathematics of cases that fall in between those two extremes, *i.e* we consider a camera which captures some points with known 3D coordinates and some points which are not known but give partial information since they are seen by other known cameras. The application we have in mind is global image-based localization.

7.1 Introduction

Localization refers to the ability of automatically inferring the pose of an observer relative a model [7]. Solving this problem using an image-based approach amounts to first establishing tentative correspondences between an input image and the model, filtering out outliers and computing the camera location and orientation. The need to understand and solve minimal setups thus arises in a manner very similar to that of Chapter 8. The model or the map of the environment can be anything from a single room in a building to a complete city. In general, one image will be used as a query image, but in principle several images can be used as input. No prior knowledge of the observer's position is assumed and therefore the problem is often referred to as global localization whereas local versions assume an approximate position. The mapping of the environment can be regarded as an off-line process since it is generally done once and for all. Such a mapping can be done using standard Structure from Motion (SfM) algorithms [40], or by some other means.

The key idea of this work is to use a mixture of 2D and 3D features simultaneously for localization. A 3D feature here refers to a point with known location in the room with associated features. A 2D feature is a feature point detected in one view with known camera matrix, this gives a feature with the 3D position known up to a line in 3D. If one were to rely solely on 3D matches, one is restricting the set of possible correspondences to fewer correspondences and a relatively rich 3D model would be required in order to be successful. On

the other hand, using only 2D features requires relatively many correct correspondences to generate a single hypothesis. In addition, with existing methods such as the seven point algorithm for two views [40], one is limited to picking all the 2D correspondences from one single image in the model. Again, one is restricting the set of correspondences to a smaller subset. Furthermore, the absolute scale cannot be recovered solely from 2D correspondences of one query image and one model image.

Using hybrid correspondence sets for generating hypotheses gives a number of advantages. We can make use of all possible correspondences simultaneously, even from different 2D model images. Compared to approaches using only 2D correspondences, the scale relative to the 3D map can be recovered and, more importantly, the number of correspondences is smaller which is preferable when using RANSAC. One can argue that in most cases, traditional methods would work fine. However, if one accepts possibly somewhat longer computation times, using hybrid correspondence sets (as well as traditional ones) provides a strictly greater chance of obtaining an outlier-free point set and there is hence no reason why the extra information should not be used.

The main contributions of this chapter are:

1. A complete list of minimal hybrid cases is given. For most the number of possible solutions is given.

2. Algorithms for efficiently computing the solutions of two of the minimal cases are given. One of these cases was only solvable using the techniques of Chapter 5.

7.2 Problem Formulation

With the localization application in mind, we are interested in solving the following problem:

Under the assumption that for a query image, there are m potential correspondences to image points in views with known absolute orientation and n potential correspondences to scene points with known 3D coordinates, find the largest subset of the correspondences that admits a solution to the absolute orientation problem within a specified accuracy.

The method that we use to solve the localization problem is based on hypothesize-and-test with RANSAC [30] and local invariant features [61]. This involves solving minimal structure and motion problems with hybrid correspondence sets.

7.3 Minimal Hybrid Correspondence Sets

The classical absolute orientation problem (also known as camera resectioning) for calibrated cameras for three known points can be posed as finding the matrix $P = [R\ t]$, such that $\lambda_i u_i = PU_i$, $i = 1, 2, 3$. Here R is a 3×3 rotation matrix and t is a 3-element translation vector. Thus, the camera matrix encodes six

degrees of freedom. Each point gives two constraints and therefore three points form a minimal case. In general there are four possible solutions [40].

We will study the absolute orientation problem for both calibrated cameras as above, for the case of unknown focal length and for the uncalibrated camera case. Furthermore we will consider both known 3D-2D correspondences (U_i, u_i) as above and 2D-2D correspondences (v_i, u_i) with features v_i in other views. Here we will assume that the camera matrices of the other views are known, so that a 2D-2D correspondence can be thought of as a 3D-2D correspondence where the unknown 3D point U_i lies on a line expressed in Plücker coordinates. Here, **the (m,n) case** denotes the case of m 2D-2D correspondences and n 3D-2D correspondences. Notice that each 2D-2D correspondence imposes one constraint and each 2D-3D correspondence imposes two constraints. We begin with an overview of all possible minimal cases in this family and some brief comments for each case. After that we go into some more detail on how they can be analyzed and solved.

Calibrated Cameras For calibrated cameras there are six degrees of freedom, three for orientation and three for position. One way of parameterizing the camera matrix is to use a quaternion vector $[a\ b\ c\ d]^T$ for rotation, *i.e*

$$P = \begin{bmatrix} a^2+b^2-c^2-d^2 & 2bc-2ad & 2ac+2bd & x \\ 2ad+2bc & a^2-b^2+c^2-d^2 & 2cd-2ab & y \\ 2bd-2ac & 2ab+2cd & a^2-b^2-c^2+d^2 & z \end{bmatrix}. \tag{7.1}$$

Potential minimal cases are:

The (0,3) case. This is the well known resectioning problem, *cf* [40] with up to four solutions in front of the camera.

The (2,2) case. This case is given a numerical solution in Section 7.4.2. The algorithm works equally well if the 2D-2D correspondences are to the same or to different cameras. There are up to 16 solutions.

The (4,1) case. There are two cases here. In the first case all 2D-2D correspondences are to the same view. In this first case the problem can be solved by first projecting the 3D point in the known camera and then using the five point algorithm to solve for relative orientation, (hence up to 10 solutions), cf. [52]. The scale is then fixed using the 2D-3D correspondence. The second case is when the 2D-2D correspondences are to at least two different views. This is studied in this chapter and there are up to 32 solutions for this case. No numerical algorithm is presented.

The (6,0) case. This cannot be solved for absolute orientation if all points are from the same model view. However, if the correspondences come from different views, it is equivalent to the relative orientation problem for generalized cameras, *cf* [79], which has up to 64 solutions.

Unknown Focal Length For calibrated cameras with unknown focal length there are seven degrees of freedom, three for orientation, three for position and one for the focal length. One way of parameterizing the camera matrix is as

$$P = \begin{bmatrix} a^2+b^2-c^2-d^2 & 2bc-2ad & 2ac+2bd & x \\ 2ad+2bc & a^2-b^2+c^2-d^2 & 2cd-2ab & y \\ 2f(bd-ac) & 2f(ab+cd) & f(a^2-b^2-c^2+d^2) & fz \end{bmatrix}. \tag{7.2}$$

Potential minimal cases are

The (1,3) case. This case is given a numerical solution in Section 7.4.4. There are 36 solutions.

The (3,2) case. This is studied in this chapter and we have found the number of solutions to be up to 40 in the general case. No numerical algorithm is presented.

The (5,1) case. There are two cases here. In the first case all 2D-2D correspondences are to the same view. In this first case the problem can be solved by first projecting the 3D point in the known camera and then using the six point algorithm to solve for relative orientation and focal length [78]. The scale is then fixed using the 2D-3D correspondence. There are up to 15 solutions. The second case is when the 2D-2D correspondences are to at least two different views. This is studied in this chapter and with the aid of Macaulay2, we have found that there are potentially up to 112 solutions. No numerical algorithm is presented.

The (7,0) case. This cannot be solved for absolute orientation if all points are to the same view. For the case of correspondence to different views, finding the number of solutions is an open problem. For this case the calculations turned out to be too complex to handle for the computer algebra system.

Uncalibrated Cameras For the uncalibrated camera case there are 11 degrees of freedom. Each 2D-2D correspondence gives one constraint and each 2D-3D correspondence gives two constraints. Potential minimal cases are

The (1,5) case. This can be solved by hand-calculations as follows. Using the five 3D-2D correspondences, the camera matrix can be determined up to a one-parameter family $P = P_1 + \nu P_2$, where P_1 and P_2 are given 3×4 matrices and ν is an unknown scalar. The remaining 2D correspondence can be parameterized as a point on a line $U = C + \mu D$ for some unknown parameter μ. The projection equation gives $\lambda u = PU = (P_1 + \nu P_2)(U_1 + \mu U_2)$. Using resultants, it follows easily that there are two solutions for the unknowns λ, ν, μ.

The (3,4) case. There are eight solutions, unless all four 2D-2D correspondences are from the same model view, in which the standard seven-point-two-view algorithm can be used. There are then up to three solutions. No numerical algorithm is presented.

The (1+2k,5-k) case with $k = 2, 3, 4$. These cannot be solved for absolute orientation if all points originate from one model view. However, for the case of correspondences from different model views, there are $2^{(1+2k)}$ solutions. The solutions procedure is analogous to the (1,5) case above and can be obtained using resultants. No numerical algorithm is presented.

Summary We conclude this section by summarizing all the minimal cases for hybrid 2D and 3D feature correspondences, see Table 7.1. We state an upper bound on the number of physically realizable solutions. In practice though, as we shall see later in Section 7.4.3, the number of plausible solutions is much smaller. In the next section, we give the remaining justifications to these claims. This will also lead to efficient algorithms for computing the solutions. Algorithms in Matlab for solving the (2,2) and (1,3) cases, that later are evaluated in this paper, are available for download at `http://www.maths.lth.se/vision/downloads/`.

2D-2D corresp.	2D-3D corresp.	number of solutions	camera setting
0	3	4	calibrated
2	2	16	calibrated
4	1	32 or 10*	calibrated
6	0	64	calibrated
1	3	36	unknown focal
3	2	40	unknown focal
5	1	112 or 15*	unknown focal
7	0	?	unknown focal
1	5	2	uncalibrated
3	4	8 or 3*	uncalibrated
$1+2k$	$5-k$	2^{1+2k}	uncalibrated

Table 7.1: Minimal hybrid cases for structure from motion. The number of solutions indicates an upper bound of the number of physically realizable solutions. The solution numbers marked with asterisk "*" correspond to cases where all 2D-2D correspondences originate from a single (model) view, whereas for other cases, it is implicitly assumed that the correspondence set covers multiple views. Note that one case is still an open problem (marked with "?").

7.4 Solving Hybrid Minimal Cases with Algebraic Geometry

As we have seen in previous chapters, minimal structure and motion problems typically boil down to solving a system of polynomial equations in a number of unknowns. This is the case for all problems studied in this chapter as well. We now give the details concerning how the various hybrid problems are formulated and solved using a combination of algebraic geometry and numerical linear algebra. We in turn consider calibrated, semi-calibrated and uncalibrated cameras.

7.4.1 Symbolic Calculations

For a specific application problem the structure of the polynomial system is fixed. Thus the number of solutions to a structure and motion problem typically depends only on the type of problem at hand.

To analyze the problems listed in this chapter we have used the computer algebra system Macaulay2 [36] which uses Gröbner Basis techniques with integer coefficients by projecting the equations from $\mathbb{C}[\mathbf{x}]$ to $\mathbb{Z}_p[\mathbf{x}]$, where p represents a large prime number and computing the Gröbner basis there.

In [44] it is shown that an upper bound on the number of solutions to a problem can be found by solving a single instance of the problem. That is, if you find the solution to a problem for one instance that gives an upper bound on the number of solutions in the general case. There can still exist degenerate configurations with a higher number of solution but the probability to end up in such a solution is very small for random coefficients. Based on this theoretical result, it is enough to work with integer coefficient in $\mathbb{Z}_p[\mathbf{x}]$ to determine the

number of solutions.

7.4.2 Calibrated Cameras

A calibrated camera can be parameterized using quaternions as shown in (7.1). We now study the (2,2) case in more detail, *i.e* assume that we have two correspondences between image points and scene points

$$u_1 \sim PU_1, \quad u_2 \sim PU_2.$$

Since there is a freedom in choosing coordinate systems both in the scene and in the images, without loss of generality we can assume

$$U_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \; u_1 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \; U_2 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \; u_2 = \begin{bmatrix} 1 \\ 0 \\ u \end{bmatrix}.$$

This gives us the following constraints

$$\begin{gathered} x = 0, \quad y = 0, \quad ad = -bc, \\ z = u(a^2 + b^2 - c^2 - d^2) - 2bd + 2ac. \end{gathered}$$

As the overall scale of the camera matrix is irrelevant, one can set $a = 1$ and eliminate d as $d = -bc$. This makes it possible to parameterize the camera matrix as

$$P = \begin{bmatrix} (1+b^2)(1-c^2) & 4bc & 2c(1-b^2) & 0 \\ 0 & (1-b^2)(1+c^2) & -2b(1+c^2) & 0 \\ -2c(1+b^2) & 2b(1-c^2) & (1-b^2)(1-c^2) & z \end{bmatrix}.$$

By setting $a = 1$ two things happen. First the scale of the camera matrix is fixed, hence the left-hand 3×3 sub matrix in (7.1) will only be a rotation matrix up to scale. This will not have any further impact on the problem since the measurement equations are homogeneous. The second consequence is that solutions with $a = 0$ will not be included. The probability of obtaining $a = 0$ by chance is very small, but there might be problems as well if a is close to zero. However, as the synthetic experiments will show this causes no serious problems for the end result.

Assume now that we have two correspondences between image points and points that have been seen in only one other model image. This gives two points on the viewing line C_i and D_i associated to a point v_i in the query image. If the line is represented with Plücker coordinates [40] and the camera is converted to the corresponding Plücker camera the constraints above converts to a single equation. It is furthermore easy to see that every nonzero element in the Plücker camera has a common factor $1 + b^2$. After removing the common factor, the constraint polynomials (p_1, p_2) are of order 2 in b and order 4 in c.

The dimension of the quotient space $\mathbb{C}[b, c]/I$ is 16 with $I = \{p_1, p_2\}$ which can be checked with computer algebra. By multiplying the polynomials with $\{1, b, c, bc\}$ we obtain 8 equations in 24 monomials. It is then possible to express 8 of the monomials in terms of the remaining 16 monomials

$$\{bc^4, b^3c^2, c^4, bc^3, b^2c^2, b^3c, c^3, bc^2, b^2c, b^3, c^2, bc, b^2, c, b, 1\},$$

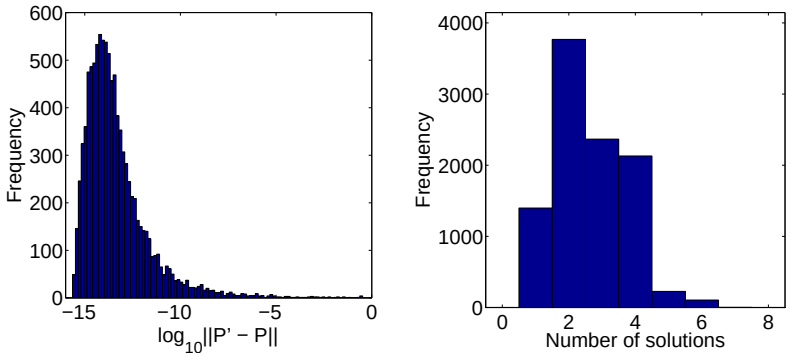

Figure 7.1: Statistics from the evaluation of the solver for the (2,2) case for calibrated cameras. The solver was run on 10,000 randomly generated cases. Left: Histogram over the error in Frobenius norm between the estimated camera P' and the true camera P. The error is plotted on a logarithmic scale. Right: Histogram over the number of real valued solutions yielding positive depths.

which then form a basis for the quotient space $\mathbb{C}[b,c]/I$. From this it is straightforward to construct the 16×16 multiplication matrix $\mathbf{m}_c$. From the eigenvalue decomposition of the matrix $\mathbf{m}_c$ the 16 solutions are obtained. Since this problem is of relatively low degree with only two variables, the eliminations are well conditioned as they are and there is no pressing need to apply any stabilizing techniques.

Gröbner basis calculations with a computer algebra system show that there are 32 solutions for the (4,1) case, but we have not implemented a numerical solver for this case.

7.4.3 Experimental Results for the (2,2) Case

The purpose of this section is to evaluate the stability of the algorithm for solving the $(2,2)$ case. To this end we use synthetically generated data in the form of randomly generated cameras and points. This allows us to measure the typical errors and the typical number of plausible solutions, over a large range of cases.

The point features are drawn uniformly from the cube ± 500 units from the origin in each direction. The cameras (two known and one unknown) are generated at approximately 1000 units from the origin pointing roughly in the direction of the center of the point cloud.

The algorithm has been run on 10,000 randomly generated cases as described above. To evaluate the accuracy of the solution we take the minimal error (over the plausible solutions) of the Frobenius norm $\|P' - P\|$ of the difference between the estimated camera P' and the true camera P. The cameras were normalized by setting the last element to one. The result is illustrated in Figure 7.1. As can be seen, the errors typically stay as low as 10^{-15} to 10^{-10}, but occasionally larger errors occur. However, since the solver is used as a subroutine in a RANSAC engine, which relies on solving a large number of different instances, these very rare cases with poor accuracy are not a serious problem.

As shown in Section 7.4 the (2,2) calibrated case in general has 16 solutions. Since obviously only one of these solutions is the correct one it is interesting to investigate how many plausible solutions are typically obtained. With plausible solutions we mean real valued camera matrices which yield positive depths for all four problem points. In Figure 7.1 a histogram which shows the typical number of plausible solutions is given. As can be seen the most common situation is one to four plausible solutions. In one of the 10,000 cases, the algorithm was unable to find a real solution with positive depths for all points. This is probably due to numerical problems when the points and/or cameras are unfortunately positioned (two or more real solutions irrespective of the sign of the depths were found in all cases). In three of the cases seven solutions were found and in one case eight plausible solutions were found. The average number of plausible solutions was 2.6 and the average number of real solutions was 6.4. In some of the cases all 16 solutions were real.

7.4.4 Unknown Focal Length

For the case of unknown focal length we have one additional unknown and we thus need one extra constraint. There are several interesting minimal cases: (1,3), (3,2) and (5,1). However, for the last case (assuming that all the five points were in correspondence with the same view) one could solve the relative orientation problem using the six point algorithm [78] and then fix the scale using the known 3D correspondence.

Using (7.2) as parameterization for the camera matrix and assuming that two of the 3D point correspondences are with

$$U_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}, U_2 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}, u_1 = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

it is possible to eliminate $y = 0$ and $x = zf = g(b, c, d, f)$. We fix the scale by setting $a = 1$. For both the $(1, 3)$ case and the $(3, 2)$ case we get polynomial constraints in the five remaining unknowns (b, c, d, z, f). Calculations with computer algebra show that there are 36 solutions for the $(1, 3)$ case, 40 solutions to the $(3, 2)$ case and 112 in the $(5, 1)$ case.

A numerical algorithm for the $(1, 3)$ case has been obtained as follows. The above parameterization gives four equations in four unknowns. The unknowns are the three quaternion parameters and the focal length. The equation derived from the line correspondence is of degree 6 and those obtained from the 3D points are of degree 3. The coefficient matrix $\mathbf{C}$ is then constructed by expanding all equations up to degree 10. This means that the equation derived from the line is multiplied with all monomials up to degree 4, but no single variable in the monomials is of higher degree than 2. In the same manner the point correspondence equations are multiplied with monomials up to degree 7 but no single variable of degree more than 5. The described expansion gives 980 equations in 873 monomials. All of these equations were necessary to get a working solution to the problem.

Whereas the $(2, 2)$ case was reasonably well conditioned in itself, the $(1, 3)$ case is significantly more complicated as can be seen by the number of equations

that need to be generated and we were not even able to construct a numerical solver using previous methods. As it turned out, truncation had to be used (*i.e* a redundant basis for $\mathbb{C}[\mathbf{x}]/I$) to avoid a rank deficient elimination step.

Proceeding as in Chapter 5 we partition and reorder the monomials into excessive monomials ($\mathcal{E}$), monomials to reduce ($\mathcal{R}$) and permissible basis monomials ($\mathcal{P}$). In this problem $\mathbf{C}_\mathcal{P}$ corresponds to all monomials up to degree 4 except f^4 where f is the focal length, which gives 69 columns in $\mathbf{C}_\mathcal{P}$. The part $\mathbf{C}_\mathcal{R}$ corresponds to the 5:th degree monomials that appear when the monomials in $\mathcal{P}$ are multiplied with the first of the unknown quaternion parameters.

7.4.5 Experimental Results for the (1,3) Case

The synthetic examples for the (1,3) problem were generated in the same manner as for the (2,2) case. Here, this gives one unknown camera with three point correspondences and one line correspondence. The experiment was run 10,000 times.

Figure 7.2 gives the distribution of relative errors in the estimated focal length. It can be seen that both the SVD method and the faster QR method give useful results. We emphasize that we were not able to construct a solver with the standard method and hence no error distribution for that method is available.

In Figure 7.3 the distribution of basis sizes is shown for the QR method with variable basis size. For the SVD method the basis size was identical to the QR method in over 97% of the cases and never differed by more than one element.

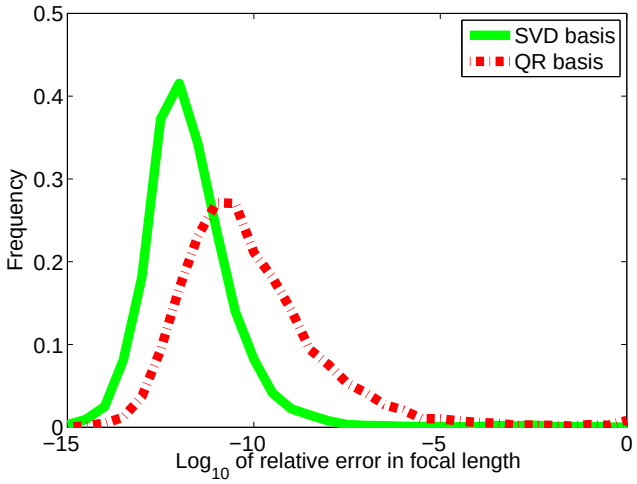

Figure 7.2: Error histogram for the (1,3) case with unknown focal length. The standard method is omitted since we did not manage to construct a standard solver due to numerical problems.

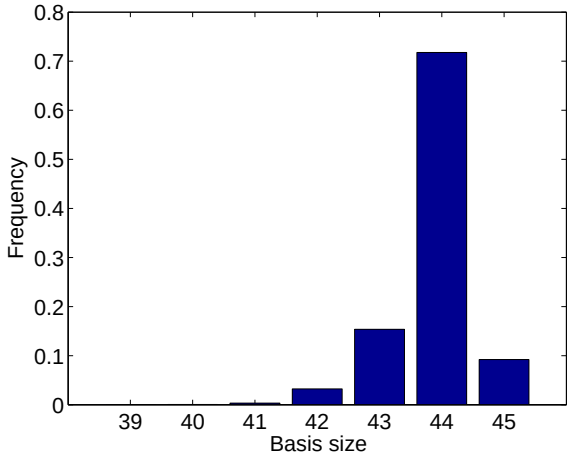

Figure 7.3: The distribution of basis sizes for the (1,3) problem with unknown focal length solved with the QR method with variable basis size. The number of solutions are 36 and since we always add three dimensions to the truncated ideal the minimal possible basis size is 39.

7.5 Conclusions

In this chapter we have presented new minimal cases for the resection problem. These use a mixture of correspondences to known 3D points and correspondences to points that have only been found in one image in the model. In all except one of these minimal cases we have given an upper bound on the possible number of solutions with Gröbner basis techniques. In two of the cases we have also presented and evaluated numerical solvers. The first of these cases is the (2,2) problem that finds the pose for a calibrated camera. The solution with Gröbner basis techniques leads to a fast and numerically stable algorithm. We also present a solver for the (1,3) case for cameras with unknown focal length. This problem is significantly more complicated than the (2,2) problem but we can still present a numerically sound and computationally efficient algorithm with Gröbner basis methods.

Chapter 8

Epipolar Geometry Under Radial Distortion

In this chapter we study the problem of estimating relative camera motion between two frames in the presence of potentially heavy radial distortion. Efficient and reliable solutions to the relative motion problem serve as the core of many computer vision systems. Traditionally, one has assumed a linear camera model and at best compensated for radial distortion towards the end of the process. In this chapter it is indicated how radial distortion can be taken into account already from the outset. In particular, two minimal cases of structure from motion with radial distortion are derived and solved.

8.1 Introduction

Estimating camera motion and inner calibration parameters from sequences of images is a challenging computer vision problem with a broad range of applications. Typically one starts with a noisy set of tentative image point correspondences. The first step then is to make decisions about inliers and outliers and get a good initial estimate to be able to deploy a more sophisticated optimization algorithm on the set of all inliers.

Two robust and widely used techniques for this purpose are RANSAC [30] and kernel voting [59], both relying on solving a large number of instances of the underlying problem, each with a small number of point correspondences. There is thus a need to develop fast and stable algorithms for solving geometric computer vision problems with a minimal number of points. Typically this amounts to solving a system of polynomial equations in several variables.

Traditionally, minimal problems have been formulated assuming a linear pinhole camera model with different restrictions on the inner calibration parameters *etc.* However, for some cameras such fish-eye lenses this can be insufficient and one might need to handle strong radial distortions already from the outset.

Solving for the fundamental matrix under radial distortion was first studied in [5], where a *non-minimal* algorithm based on 15 point correspondences was given for a pair of uncalibrated cameras. More recently, in [55, 56], a number of different *minimal* problems with radial distortion have been studied and practical solutions have been given in some cases.

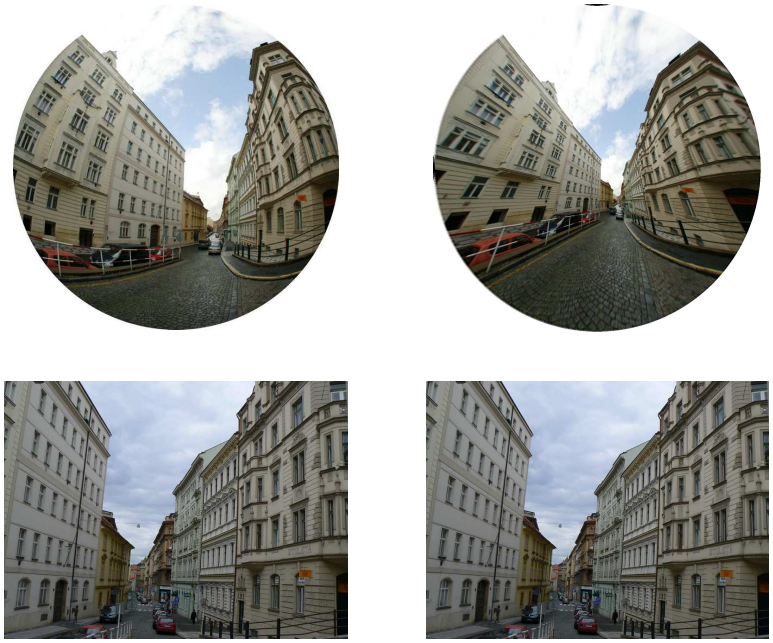

Figure 8.1: Left: Input images with different radial distortions. Right: Images corrected using the method described here. Images from sources with different amounts of distortion are shown. Top: 66% cutout from an omnidirectional image. Bottom: standard perspective camera.

Leveraging on the new techniques presented in this thesis, fast and numerically stable algorithms for two minimal problems with radial distortion previously unsolved in floating point arithmetic are formulated and solved:

1. The problem of estimating a one-parameter radial distortion model and epipolar geometry from image point correspondences in two uncalibrated views with different radial distortions in each image.

2. The problem of estimating a one-parameter radial distortion model and epipolar geometry from image point correspondences in two partially calibrated views.

These two problems were previously studied in [56] and found to be numerically very challenging. In [56] the authors provide solutions to these problems computed in exact rational arithmetic only. This results in very long computational times and is not usable in practical applications. Here we show that these two problems can be efficiently solved in floating point arithmetic.

The speed and intrinsic numerical stability as well as robustness to noise of the proposed algorithms is demonstrated using both synthetic data and real images.

8.2 Uncalibrated Case

In this case, we study the situation with two uncalibrated cameras and two different unknown radial distortion parameters. We use the same formulation of the problem as in [56]. This formulation assumes a one-parameter division model [31] given by the formula

$$u \sim x/(1 + \lambda|x|^2) \tag{8.1}$$

where $u = (u_1, u_2, 1)^T$ and $x = (x_2, x_2, 1)^T$ are the corresponding undistorted, resp. distorted, image points, and $|x|$ (with a slight abuse of notation) is the radius of x w.r.t. the distortion center.

The minimal set of constraints needed to uniquely solve for relative motion for uncalibrated cameras with different radial distortion λ_1 and λ_2 is 9 point correspondences with epipolar constraints

$$u_i^T(\lambda_1) F u_i'(\lambda_2) = 0, \quad i = 1, \ldots, 9 \tag{8.2}$$

and the singularity of the fundamental matrix F

$$\det(F) = 0. \tag{8.3}$$

Assuming $f_{3,3} \neq 0$ we can set $f_{3,3} = 1$ and obtain 10 equations in 10 unknowns.

By linear elimination, these 10 equations can be reduced to 4 equations in 4 unknowns (one of 2^{nd} degree, two of 3^{rd} degree and one of 5^{th} degree). For more details see [56] where it was shown that this problem has 24 solutions.

The numerical solver is constructed starting with the four remaining equations in the four unknowns $f_{3,1}$, $f_{3,2}$, λ_1 and λ_2. The first step is to expand the number of equations, as outlined in Section 4.1.2, by multiplying them by a handcrafted set of monomials in the four unknowns, in this case yielding 393 equations in 390 monomials. See Section 8.2.1 for details.

We now stack the coefficients of the equations in a matrix $\mathbf{C}$. Following this, we order the monomials and correspondingly the columns of $\mathbf{C}$ as in (4.9). The sets $\mathcal{E}$ and $\mathcal{R}$ depend on which variable is used to create the multiplication matrix. For this problem $f_{3,1}$ was used as multiplier variable. The classical method is thereafter to choose the linear basis $\mathcal{B}$ of $\mathbb{C}[\mathbf{x}]/I$ to be the 24 lowest monomials (w.r.t some monomial order). This is enough to get a solution to the problem, but we can use the methods of Chapter 5 to select a basis of linear combinations of monomials from a larger set and thereby improve numerical stability. Empirically, we have found that the linear basis can be selected from the set of all monomials up to degree four excluding the monomial λ_1^4. The set $\mathcal{R}$ then consists of monomials of degree five that are reached when the monomials of degree four are multiplied with $f_{3,1}$. The set $\mathcal{E}$ contains the remaining 285 monomials.

Putting the part of $\mathbf{C}$ corresponding to $\mathcal{E}$ and $\mathcal{R}$ on triangular form by means of an LU decomposition now produces Equation 4.10. We can then remove all equations that include excessive monomials and still have enough information to construct the multiplication matrix.

Finally, we use the QR method to select a linear basis for $\mathbb{C}[\mathbf{x}]/I$ and construct the matrix $\mathbf{m}_{f_{3,1}}$ from which the solutions are extracted.

8.2.1 Details on the Expansion Step for the Uncalibrated Case

We have found in experiments that to compute reduction to the solving basis as described in Chapter 4 and hence obtain the multiplication matrix $\mathbf{m}_{f_{3,1}}$, we need to generate polynomials up to total degree eight. Thus, the 2^{nd} degree polynomial has to be multiplied with all monomials up to degree six and corresponding numbers for the 3^{rd} and 5^{th} degree polynomials.

Further investigations have shown that not exactly all monomials up to degree eight are needed, so in the implementation, the 2^{nd} degree polynomial was only multiplied with monomials up to degree five and each variable was not allowed to got to a higher degree than four. Furthermore, λ_1 was not multiplied with higher degrees than two. For the other polynomials it was possible to limit the degree of each individual variable to one lower than the total degree.

These multiplications yield 393 equations in 390 monomials. Without the last fine tuning of the degrees, the number of equations and monomials will be larger but all extra monomials will be in the set $\mathcal{E}$ and will make no real differences to the solver except slightly longer computation times.

8.3 Calibrated Case

We now turn to the setup with two calibrated cameras and one common unknown radial distortion parameter. To solve the corresponding minimal problem, we make use of the epipolar constraint for 6 point correspondences

$$u_i^T(\lambda)\, E\, u_i'(\lambda) = 0, \quad i = 1, \ldots, 6, \tag{8.4}$$

the singularity of the essential matrix E

$$\det(E) = 0 \tag{8.5}$$

and the trace condition, which says that two singular values of the essential matrix are equal

$$2\left(EE^T\right)E - trace(EE^T)E = 0. \tag{8.6}$$

Again assuming $e_{3,3} \neq 0$, we can set $e_{3,3} = 1$ and obtain 16 equations in 9 unknowns. In analogy with the uncalibrated case, these equations can be rewritten as 11 polynomial equations in 4 unknowns (one of 3^{rd} degree, four of 5^{th} degree and six of 6^{th} degree). In [56] it was shown that this problem has 52 solutions.

The numerical solution of this problem largely follows that of the uncalibrated version. In the first expansion, all equations are multiplied with monomials to reach degree eight. This gives 356 equations in 378 monomials. As in the uncalibrated case it is possible to reduce the number of monomials by fine tuning the degrees we need to go to, in this case yielding 320 equations in 363 monomials.

The next step is to reorder the monomials and columns as in equation (4.9). Once again, the linear basis of $\mathbb{C}[\mathbf{x}]/I$ can be constructed from the monomials of degree four and lower. $\mathcal{R}$ will then consist of those monomials of degree five that are reached when the degree four monomials are multiplied with the variable $e_{3,1}$, which is used as multiplier variable.

As before, $\mathbf{C}$ is transformed to triangular form by LU decomposition and after that we only consider those equations that do not include any of the monomials in $\mathcal{E}$. Hence $\mathbf{C}$ holds all necessary information to choose representatives in $\mathbb{C}[\mathbf{x}]/I$ and create the multiplication matrix with respect to multiplication by $e_{3,1}$.

8.4 Experiments

We have tested the algorithms for the uncalibrated and calibrated minimal problems on synthetic images with various levels of noise, outliers and radial distortions as well as on real images.

The algorithms proposed here are significantly more stable than the algorithms presented in [56] which ran in exact rational arithmetic only. Since doing the computations in exact arithmetic is extremely slow (minutes instead of milliseconds), such implementations are of very little practical value and we have hence not evaluated the numerical performance of these.

Both problems are solved by finding the roots of a system of polynomial equations which means that we obtain several potentially correct answers, 52 in the calibrated case and 24 in the uncalibrated case. In general we obtain more than one real root, in which case we need to select the best one, *i.e* the root which is consistent with most measurements. To do so, we treat the real roots obtained by solving the equations for one input as real roots from different inputs and use kernel voting [59] for several inputs to select the best root among all generated roots. The kernel voting is done using a Gaussian kernel with fixed variance. The estimate of λ_1 and λ_2 in the uncalibrated case and λ in the calibrated case is found as the position of the largest peak [59, 55].

8.4.1 Tests on Synthetic Images

For both problems treated here, the same synthetic experiments were carried out to evaluate the quality of the solvers.

In all simulated experiments we generate synthetic data using the following procedure:

1. Generate a 3D scene consisting of 1000 points distributed randomly within a cube. Project $M\%$ of the points onto the image planes of the two displaced cameras. Let each such pair of projections be stored as a match. In both image planes, generate $(100 - M)\%$ random points distributed uniformly in the image and let random pairs of these represent mismatches.
2. Apply different radial distortions to the undistorted correspondences in each image and in this way generate noiseless distorted points.
3. Add Gaussian noise of standard deviation σ to the distorted points.

Uncalibrated case

In the first two experiments we study the robustness of the algorithm for the uncalibrated case to Gaussian noise added to the distorted points.

The first experiment investigates the estimation error of λ as a function of noise. The ground truth radial distortions parameters were $\lambda_1 = -0.2$, $\lambda_2 =$

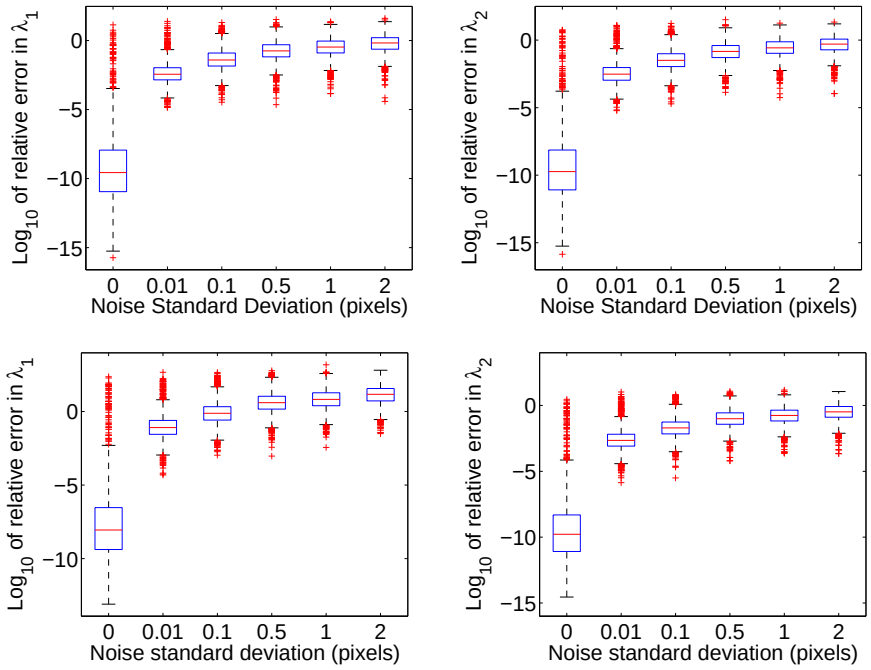

Figure 8.2: Uncalibrated case: Relative errors of estimated values (Left) $\overline{\lambda_1}$ and (Right) $\overline{\lambda_2}$ as a function of noise. Ground truth (Top) $\lambda_1 = -0.2$, $\lambda_2 = -0.3$ and (Bottom) $\lambda_1 = -0.01$, $\lambda_2 = -0.7$. Blue boxes contain values from 25% to 75% quantile.

-0.3 in the first case and $\lambda_1 = -0.01$, $\lambda_2 = -0.7$ in the second case. See Figure 8.2. The noise varied from 0 to 2 pixels. For each noise level relative errors for 2000 λs (estimated as closest values to the ground truth value from all solutions) were computed. The results in Figure 8.2 for the estimated $\overline{\lambda_1}$ (Left) and $\overline{\lambda_2}$ (Right) are presented 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.

For noiseless data we obtain very accurate estimates of radial distortion parameters even for very different λs. For larger noises the $\log_{10}$ relative errors are much higher (mostly around 10^{-1}). However obtained λs are still satisfactory and mostly differ from the ground truth value in the second decimal place. The main point is however not to use a minimal point set to get a good estimate, but to repeatedly draw minimal configurations from a larger set of potential matches and then use *e.g* kernel voting to get a more reliable estimate. Finally, the result can be further enhanced using the obtained estimate as a good starting guess in a large scale bundle adjustment. The effect of kernel voting is studied in the second experiment.

In this experiment we did not select the root closest to the ground truth value for each run of the algorithm, instead we used kernel voting to select the best λs among all generated roots from several runs. The ground truth radial distortion parameters were as in the previous experiment ($\lambda_1 = -0.2$, $\lambda_2 = -0.3$

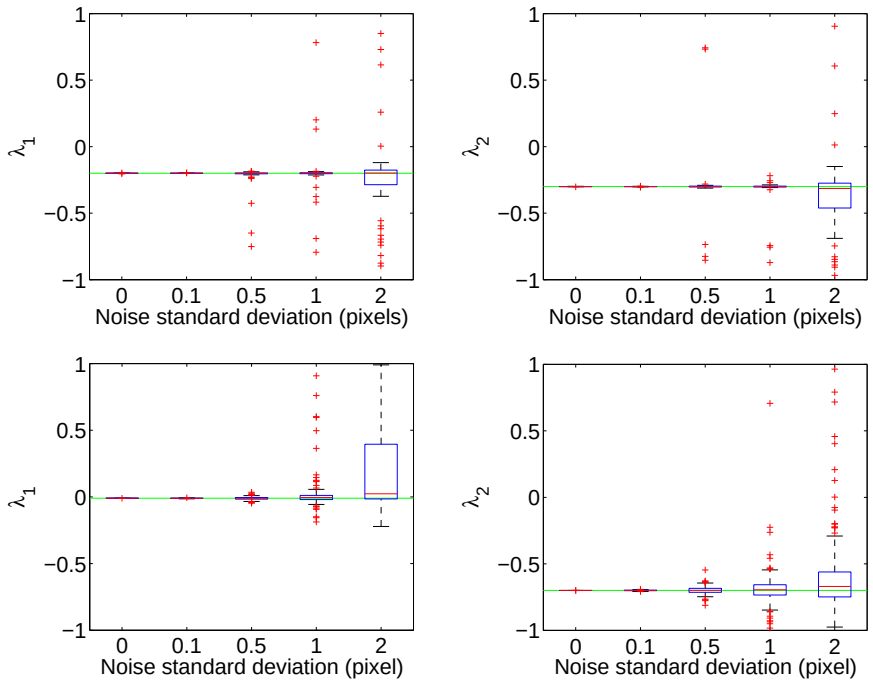

Figure 8.3: Uncalibrated case, kernel voting: Estimated (Left) $\overline{\lambda_1}$ and (Right) $\overline{\lambda_2}$ as a function of noise, (Top) ground truth $\lambda_1 = -0.2$, $\lambda_2 = -0.3$ (green lines), 90% of inliers and 100 samples in kernel voting and (Bottom) ground truth $\lambda_1 = -0.01$, $\lambda_2 = -0.7$, 100% of inliers and 50 samples in kernel voting.

in the first case and $\lambda_1 = -0.01$, $\lambda_2 = -0.7$ in the second case) and the level of noise varied from 0 to 2 pixels. Moreover, in the first case there were 10% of outliers in the image (M=90).

The testing procedure was as follows:

1. Repeat K times (We use K from 50 to 100).
   (a) Randomly choose 9 point correspondences from a set of N potential correspondences (6 point correspondences for the calibrated case).
   (b) Normalize image point coordinates to $[-1, 1]$.
   (c) Find 24 (54) roots using the presented algorithm.
   (d) Select the real roots in the feasible interval, *i.e* $-1 < \lambda_1, \lambda_2 < 1$ and the corresponding F's.
2. Use kernel voting to select the best root.

Figure 8.3 shows λs computed using the algorithm for the uncalibrated case as a function of noise. In the first case with outliers Figure 8.3 (Top) 100 λs were estimated using kernel voting for roots computed from 100 ($K = 100$) 9-tuples of correspondences randomly drawn for each noise level. In the second case Figure 8.3 (Bottom) 200 λs were estimated using kernel voting for roots computed from 50 ($K = 50$) 9-tuples of correspondences. This means that for

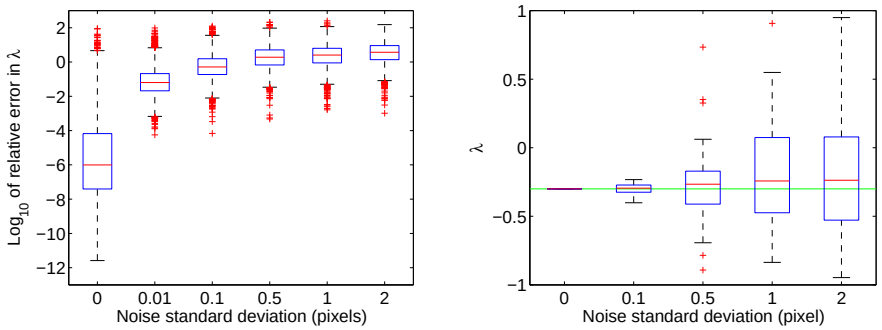

Figure 8.4: Calibrated case: (Left) relative errors of $\overline{\lambda}$ as a function of noise, ground truth $\lambda = -0.3$. (Right) kernel voting: Estimated $\overline{\lambda}$ using kernel voting for roots computed from 200 6-tuples of correspondences randomly drawn for each noise level. Ground truth $\lambda = -0.3$ (green line).

each noise level the algorithm ran 10,000 times in both cases. The results are again presented by the Matlab function *boxplot*.

Calibrated case

The same synthetic experiments were carried out for the calibrated solver.

The results of the first experiment which shows relative errors of the estimated $\overline{\lambda}$ as a function of noise are shown in Figure 8.4. The ground truth radial distortion was $\lambda = -0.3$. For noiseless data we again obtain very precise estimates of radial distortion parameter λ. For larger noise levels the $\log_{10}$ relative errors are slightly larger than for the uncalibrated case. However, using kernel voting we can still obtain good estimates. This is shown by the second experiment.

In this experiment λ was estimated 50 times using kernel voting for roots computed from 200 6-tuples of correspondences randomly drawn for each noise level, Figure 8.4. The median values for $\overline{\lambda}$ are again very close to the ground truth value $\lambda = -0.3$ for all noise levels from 0 to 2 pixels. However the variances of this for the calibrated case are larger, especially for higher noise levels, than the variances for the uncalibrated case. This means that for good estimates of λ this algorithm requires more samples in the kernel voting procedure than in the uncalibrated case.

8.4.2 Time Consumption

To evaluate the speed of the new algorithms, reasonably optimized versions for both the uncalibrated and calibrated cases were implemented. The implementation was done in Matlab so rewriting the algorithm in a compiled language such as C should reduce the execution time further.

The algorithm was run 10,000 times and the time consumption was measured using the Matlab profiler. The experiments were performed on an Intel Core 2 CPU 2.13 GHz machine with 2 GB of memory. The estimated average execution time for solving one instance of the uncalibrated problem was 16 milliseconds and the corresponding time for the calibrated problem was 17 milliseconds.

Figure 8.5: Real data, 60% cutouts from omnidirectional images. (Left) Input images with different radial distortions for camera 1 (Top) and camera 2 (Bottom). (Right) Corrected images.

These results are to be compared with the execution times given for the same problem in [56], where solutions were computed in exact rational arithmetic. There, the processing time for one problem instance was 30s for the uncalibrated case and 1700s for the calibrated case.

8.4.3 Tests on Real Images

The algorithm for uncalibrated cameras with different radial distortions has been tested on several different sets of images. In the first experiment the input images with different relatively large distortions in each image, Figure 8.5 (Left), were obtained as 60% cutouts from fish-eye images taken with two different cameras with different radial distortions. Tentative point matches were then found by the wide base-line matching algorithm [63]. They contained correct as well as incorrect matches. Distortion parameters λ_1 and λ_2 were estimated using the algorithm for uncalibrated cameras with different radial distortions and the kernel voting method for 100 samples. The input images (Left) and corrected images (Right) are presented in Figure 8.5. Figure 8.6 shows the distribution of real roots for images from Figure 8.5, from which $\lambda_1 = -0.301$ and $\lambda_2 = -0.368$ were estimated. The peaks from kernel voting are sharp and the λ's are estimated accurately.

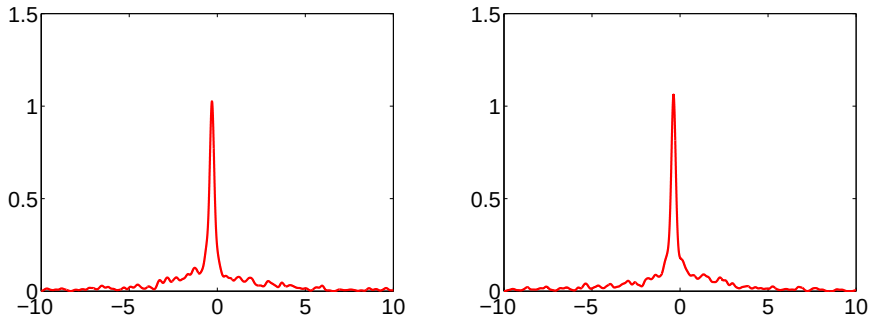

Figure 8.6: Distribution of real roots obtained by kernel voting for images in Figure 8.5. Estimated $\lambda_1 = -0.301$ and $\lambda_2 = -0.368$.

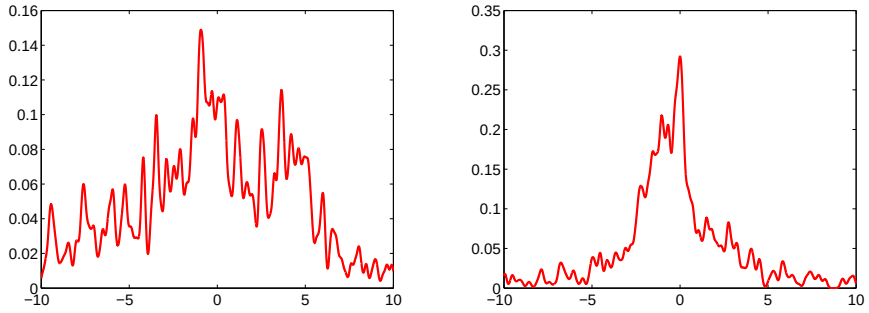

Figure 8.7: Distribution of real roots obtained by kernel voting for images in Figure 8.1. Estimated $\lambda_1 = -0.926$ and $\lambda_2 = 0.0025$.

In the second experiment the algorithm was tested on images with significantly different distortions. The left image Figure 8.1 (Left), was obtained as a 66% cutout from a fish-eye image and the right image was taken with a standard perspective camera. Since these images had a rather large difference in radial distortion, the tentative point correspondences contained a larger number of mismatches. Distortion parameters λ_1 and λ_2 were again estimated using the algorithm for uncalibrated cameras with different radial distortions and the kernel voting method. The input (Left) and corrected (Right) images are presented in Figure 8.1. Figure 8.7 shows the distribution of real roots for these images from which $\lambda_1 = -0.926$ and $\lambda_2 = 0.0025$ were estimated. As can be seen the peaks obtained by kernel voting are not so sharp but still sufficient to get good estimates of the λ's even from only 100 samples.

8.5 Discussion

In this chapter we have given fast and robust algorithms for two minimal problems previously unsolved in floating point arithmetic. The two problems of simultaneously solving for relative pose and radial distortion were, due to numerical problems, previously solved in exact rational arithmetic only, yielding

them too time consuming to be of practical value. With the floating point algorithm presented here we have reduced the computation time from minutes to milliseconds. Moreover, we have verified that this is done without loss of numerical precision by extensive experiments both on synthetic and real images.

In the experiments we have also demonstrated that the radial distortion estimation is reasonably robust both to outliers and noise when kernel voting is used over several runs. Finally we have shown that large differences in distortion between the two images can be handled.

Chapter 9

Panoramic Stitching

In this chapter, we present a solution to panoramic image stitching of two images with coinciding optical centers, but unknown focal length and radial distortion. The algorithm is a direct application of polynomial techniques introduced in the previous chapters and operates with a minimal set of corresponding points (three) which means that it is well suited for use in any RANSAC style algorithm for simultaneous estimation of geometry and outlier rejection. The proposed algorithm has been integrated in a complete multi image stitching system and we evaluate its performance on real images with lens distortion. We demonstrate both quantitative and qualitative improvements compared to state of the art methods.

9.1 Introduction

Given a sequence of images taken from a single point in space, but with varying orientations, it is possible to map the images into a common reference frame and create a perfectly aligned larger photograph with a wider field of view. This is normally referred to as panoramic image stitching. In this chapter we extend previous work to account for camera distortion throughout the stitching process. This is in contrast to most previous approaches which have assumed a traditional pin-hole camera model. Stitching images with large radial distortion is useful in a practical context, as it allows the user to create 360 degree panoramas with wide angle lenses (often suffering from heavy radial distortion), using only a few exposures. Furthermore, radial distortion occurs frequently in both cheap consumer cameras and high-end lenses depending on the type of lens *etc.*

In essence, a typical stitching pipeline consists of the following three parts

1. Image matching: Point matches across images are established and an initial estimate of the image geometry is computed. A RANSAC type algorithm is a popular choice here [30].

2. Bundle adjustment: The estimate of inner and outer calibration parameters is refined using non-linear optimization.

3. Rendering: The estimated camera parameters are used to project the images into a common reference frame.

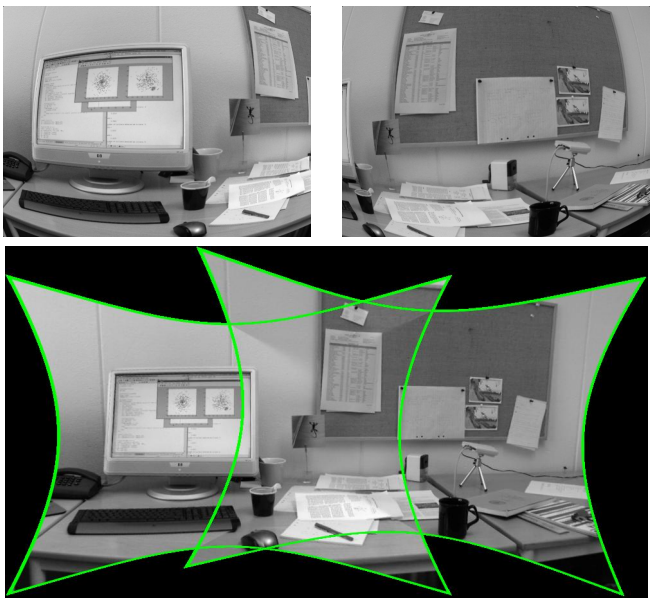

Figure 9.1: Top: Two images with heavy radial distortion taken with a common focal point. Bottom: The same two images after rectification and alignment using the stitching pipeline presented here.

Here, we mainly deal with Step 1. At the core of the RANSAC loop is an algorithm which solves for calibration and geometry given a small set of corresponding points. As discussed in previous chapters, ideally one would like a solver which operates with the minimum possible number of correspondences. For instance, consider two images taken with a pin-hole camera calibrated up to focal length. We then need to estimate rotation (3 dof) and focal length (1 dof). Each point match yields two constraints, which means that the minimal solver should use two points. This problem was solved by Brown *et al* in [8]. The rationale for using a minimal point set is that a smaller number of points yields a smaller probability of selecting a set contaminated by outliers. Furthermore, since we are solving directly for the parameters of interest, there is no need for an error-prone autocalibration process to extract the underlying camera parameters needed for multi-view non-linear techniques (i.e. bundle adjustment) to proceed.

9.1.1 Relation to Previous Work

The problem of image stitching is relatively well studied and a good overview of the literature and techniques can be found in the tutorial by Szeliski [81]. A complete stitching system representative of the state of the art in this area was presented by Brown and Lowe in [9].

A direct inspiration for this work is the two-point algorithm for estimating rotation and focal length by Brown *et al* [8]. This algorithm does however not handle any distortion and we show that for non-standard lenses, this might be insufficient.

A related algorithm which *does* account for radial distortion due to Fitzgibbon [31] estimates homography and radial distortion using five correspondences.

Two disadvantages of this approach is that (i) a homography is usually too general since in most cases one can assume square pixels and zero skew *etc* and (ii) the algorithm is not minimal. Our algorithm operates with three correspondences making it easier to find outlier-free sets.

Most closely related to our approach is the work by Jin [46]. Jin formulates the same problem as we do. However, Jin notes that solvers of polynomial equations are often numerically unstable and therefore abandons a direct solution approach. Instead he resorts to an iterative optimization based scheme. This is problematic since (i) convergence to a solution cannot be guaranteed (local minima) and (ii) even if a solution is found, this will only be one of the possible solutions and one cannot be sure to have found the right one. The actual problem has 18 (possibly complex) solutions and the only way to resolve this ambiguity is to test with additional points. Indeed, Jin reports poor performance of his algorithm for moderate to heavy distortions.

In this work we make use of the new polynomial techniques presented in this thesis to provide a numerically stable true solver for the polynomial system, which is guaranteed to find all solutions.

9.2 Models for Panoramic Stitching

We consider a setup with two cameras P_1 and P_2 with a common focal point. We fix a coordinate system where the common focal point coincides with the origin and such that the first 3x3 part of the matrix P_1 is the identity. Moreover, we have a set of world points $\{X_j\}$ and corresponding image projections $\{u_{1j}\}$ and $\{u_{2j}\}$. In most cases it is beneficial for stability to assume some partial calibration. A common choice is to assign square pixels, zero skew and centered principal point [40]. With this assumption we obtain the following relations

$$\lambda_{1j}u_{1j} = KX_j, \quad \lambda_{2j}u_{2j} = KRX_j, \tag{9.1}$$

where $K = \begin{bmatrix} f & & \\ & f & \\ & & 1 \end{bmatrix}$, R is a rotation matrix and the λs are the depths. By normalizing to remove the dependence on λ_{ij} and solving for X_j we can write down the constraints

$$\frac{\langle K^{-1}u_{1j}, K^{-1}u_{1k}\rangle^2}{|K^{-1}u_{1j}|^2|K^{-1}u_{1k}|^2} = \frac{\langle X_j, X_k\rangle^2}{|X_j|^2|X_k|^2} = \frac{\langle K^{-1}u_{2j}, K^{-1}u_{2k}\rangle^2}{|K^{-1}u_{2j}|^2|K^{-1}u_{2k}|^2}, \tag{9.2}$$

where R has vanished from the right hand side since the scalar products and norms are invariant to rotations. The expression is squared to remove the square roots from the vector norms in the denominators. In the above equation f only occurs in even powers and hence we set $p = f^2$. Moreover we multiply through with p^2 to remove any $1/p^2$ terms. Finally we multiply up the denominators. At a first glance, this seems to yield a 4th degree polynomial in p but the 4th degree terms cancel out leaving a 3rd degree polynomial in p. We next show how to modify this expression to include radial distortion.

9.2.1 A Three Point Minimal Solution for Distortion and Focal Length

Let x denote measured image coordinates affected by radial distortion and let u denote the corresponding pin-hole coordinates. As before, we model radial

distortion using Fitzgibbon's division model

$$|x| = (1 + \lambda|x|^2)|u|, \tag{9.3}$$

where $|\cdot|$ is the vector length and λ is the radial distortion coefficient. This means that in homogeneous coordinates we can write

$$u \sim x + \lambda z, \tag{9.4}$$

where $z = [\ 0 \quad 0 \quad x_1^2 + x_2^2\ ]^T$.

We now simply insert (9.4) into (9.2) and obtain a polynomial of degree 3 in p and degree 6 in λ (the 8th and 7th degree terms in λ cancel out). One more unknown means that we need more constraints. With an additional point we can form three independent constraints of type (9.2). This situation is actually a little unsatisfactory since we cannot make use of all available information. Using all three constraints would yield an overdetermined system and hence there would be no solution in general. One possibility would be to introduce an extra unknown, but we found no natural way to do this and instead settled for selecting two of the three constraints to get a system of two equations in two unknowns. The experiments confirm that this strategy works well. We used the computer algebra software Macaulay2 [36] to check solvability and number of solutions for the system which is 18 in this case.

9.2.2 Alternative Minimal Setups for Distortion and Focal Length

In addition to the setup mentioned above, there are three other related possible formulations of the problem. As mentioned, three points yield six constraints so we could actually solve for rotation, focal length, distortion and one additional parameter. Two possible choices here could be either to let the focal length vary between the images or to let the distortion vary. This can be done using the exact same equations as above, but inserting one more unknown. We have checked these two formulations using Macaulay2 and found that the fixed focal length, varying distortion case has 96 solutions and that the varying focal length, fixed distortion case has 62 solutions. Both of these formulations seem a bit artificial though, since it is generally impossible to change the focal length without affecting the distortion and vice versa.

Perhaps a bit more interesting is the case where we have two different focal lengths f_1, f_2 *and* two different distortions λ_1, λ_2. This problem could in principle be solved using four correspondences with the same formulation as for the three-point algorithm. Using Macaulay2 we have found that this case has 52 solutions. Apart from the fact that more parameters makes the formulation inherently less stable and hence more sensitive to noise, this setup still looks potentially interesting. However, the larger number of solutions and higher complexity of the equations in general seem to make this problem significantly more difficult to handle numerically. We managed to write a solver for this case, but in the process had to use an expanded coefficient matrix (see the next section) of about 1000×2000 elements. This makes the solver too slow (seconds per iteration compared to milliseconds for the three point solver) to be of practical value. Hence we have not performed any further experiments with the four point case.

9.3 A Numerical Solution using Gröbner Basis Techniques

In the current application, we are faced with a system of two equations in two unknowns (f, λ) occurring up to degrees 3 and 6 respectively and 18 solutions. To handle this system we start off with the basic method described in Chapter 5. We order the monomials in *grevlex* order and multiply the two equations with all possible monomials up to degree 8, yielding a 90×132 coefficient matrix $\mathbf{C}$.

However, with the straightforward method of [76] we were not able to solve the problem and we had to employ the redundant solving basis method. As described in Chapter 4, with this method one "pretends" to have a system with more solutions which is easier to solve. This produces all the right solutions along with a set of false solutions which have to be filtered out by evaluation in the original equations. By using this technique and setting the solution set to 25 zeros (18 for the true system), we were able to get a stable solution.

An interesting comparison would be to run the automatic solver generator by Kukelova *et al* [53]. This solver does not include any of the stabilizing methods mentioned above and might therefore fail, but this is yet to be investigated.

The algorithm has been implemented in MATLAB, which is not ideal for speed. However, the running time is dominated by an LU factorization and an eigenvalue decomposition which are fast in MATLAB so our implementation should not be too far behind a fully native implementation. The running time is about 13 milliseconds/instance on a standard 2 *Ghz* machine. The code is available for download at `http://www.maths.lth.se/vision/downloads`.

9.4 System Overview

The image stitching system implemented for this study follows the typical pattern of modern geometric computer vision systems. We start off by finding matching points pair wise across images using the SIFT descriptor/detector [61] together with RANSAC for outlier rejection [30]. Thereafter we perform first a pair-wise and subsequently a global bundle adjustment step to get an accurate estimate of geometry and calibration parameters [40]. Finally we render the images onto an enclosing cylinder which can be cut and unfolded to the final panoramic image.

9.5 Experiments

In this section, we study the basic properties of the new algorithm on synthetic data and also assess its performance as part of a complete stitching system. For this purpose we have collected two data sets using a lens with significant non-linear distortion. The data sets referred to as *City* and *University* consist of 9 and 10 photographs respectively and both cover 360 degrees. In addition, a reference set called *Canal* consisting of 8 images was shot with a low distortion lens. The final result after matching, bundle adjustment and basic blending is shown in Figure 9.6. In all cases with image data we normalized the pixel coordinates to make the width of the image fall in the interval $[-1, 1]$. This makes values for λ independent of image resolution.

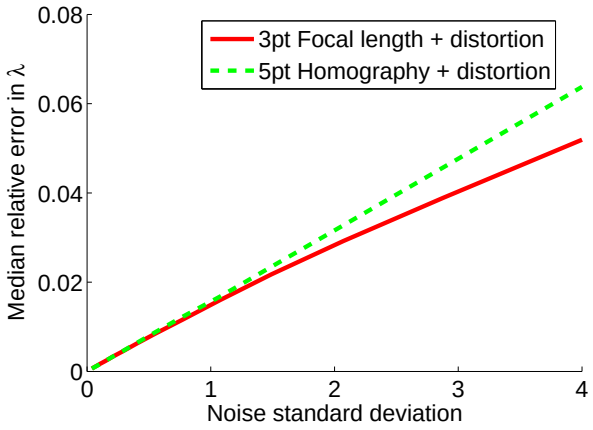

Figure 9.2: Error versus noise on synthetic data for the new three point algorithm and the five point algorithm for distortion and homography. Despite being over determined, the five point algorithm shows a slightly larger sensitivity to noise, probably due to the fact that the underlying model has more degrees of freedom.

9.5.1 Robustness to Noise

We first did a basic sanity check of our new algorithm on synthetic data to study its behavior under noise compared to Fitzgibbon's five-point algorithm for homography and distortion. Since Fitzgibbon's algorithm estimates more degrees of freedom than needed to express a transformation with focal length, rotation and distortion, we expect to see some more sensitivity to noise than with our exact solver. For this experiment we randomly generated two views separated by a random rotation and drew three and five world points respectively from a normal distribution. The points were projected into the two views to form image point correspondences and a distortion of $\lambda = -0.5$ was applied. Finally varying degrees of noise (equivalent to the interval 0 to 4 pixels in an 800 pixels wide image) was added to the projected coordinates and the distortion parameter was estimated using each algorithm. This experiment was repeated 10000 times for each noise level and median errors were calculated. The median error was chosen since both algorithms (and in particular the five point algorithm) occasionally produce gross errors for unfortunate point configurations. This makes the average errors uninformative. The results are shown in Figure 9.2. As expected, both algorithms work well at low noise levels, but the five-point algorithm is slightly less robust at high noise levels.

9.5.2 Relation to Jin's Work

Since the work of Jin [46] is most closely related to the work presented here, a direct comparison would have been ideal, but we have not been able to obtain an implementation of Jin's method which is a little unsatisfactory. However, this should not be too serious, since under the assumption that Jin's method finds the right solution, the results should be virtually identical to ours. The problem is that Jin's method is *not* guaranteed to produce the desired solution. Figure

1 in [46] shows statistics of how often Jin's algorithm finds the correct solution. For distortions below -0.2 this rate is down to below 40%. In comparison, our solver is guaranteed to find the right solution for all distortions with no serious sacrifice in speed.

9.5.3 Performance in RANSAC

The main motivation for the three point algorithm presented in this here is that it can be used to improve the RANSAC part in a stitching pipeline. With a refined inner step for geometry estimation, we hope to recover a larger proportion of inliers, at a higher rate and to a higher precision. In the next experiment we study the rate at which inliers are discovered as the RANSAC loop progresses. In addition to the five point algorithm, we now also compare our algorithm to the two-point algorithm of Brown *et al* [8], which solves for focal length but not distortion. Brown *et al* show in [8] that their algorithm is superior to the standard four point DLT algorithm for estimating a homography and hence we omit a comparison with the DLT. We fixed the threshold for outlier rejection to 3 pixels and ran each algorithm in turn for 400 RANSAC iterations, keeping track of the largest inlier set found so far. We repeated this 100 times on noisy point matches from the *City* data set and computed averages. To study the influence of varying degrees of outlier contamination we also repeated the whole process for cases with 10%, 25% and 50% outliers. The results of this experiment are shown in Figure 9.3. As can be seen, the two-point algorithm is not competitive on this sequence and recovers half as many or fewer inliers compared to our algorithm in all cases. The behavior of the five-point algorithm is more interesting. For the case with very few inliers its performance in terms of inliers is virtually as good as for the minimal algorithm. This is because the quality of the inlier point matches is quite high in terms of pixel accuracy, which means that as long as we find a set of good quality inliers we are well served by either algorithm. For the case with 25% outlier rate we already observe a significant difference and for outlier rates of 50% and more our algorithm is clearly superior. The running time for the five point algorithm is slightly lower at around 10 milliseconds/instance in our implementation compared to 13 milliseconds for the three point algorithm. With a moderate degree of outliers in the process, this speed gain is easily eaten up by the extra RANSAC iterations required.

Although the two-point algorithm recovers less inliers than *e.g* the three point algorithm, it still finds a substantial number of correct matches. However, the problem is that these matches are exactly the matches which agree with the assumption of zero distortion. In Figure 9.4 one can see the qualitative difference between correspondences produced by the three point method (with distortion) and the two-point method (no distortion). Whereas the three point solver produces matches well spread out over the images, the two-point solver recovers points grouped together near the centers of the images where the projection is reasonably well approximated by a pin-hole camera. This is problematic for two reasons, (i) the initial estimate of camera parameters will be poor and (ii) points located closely together make for poor conditioning of the bundle adjustment step.

It should be mentioned that there are other possible ways around this problem. One could *e.g* set an artificially high threshold of outlier rejection hoping

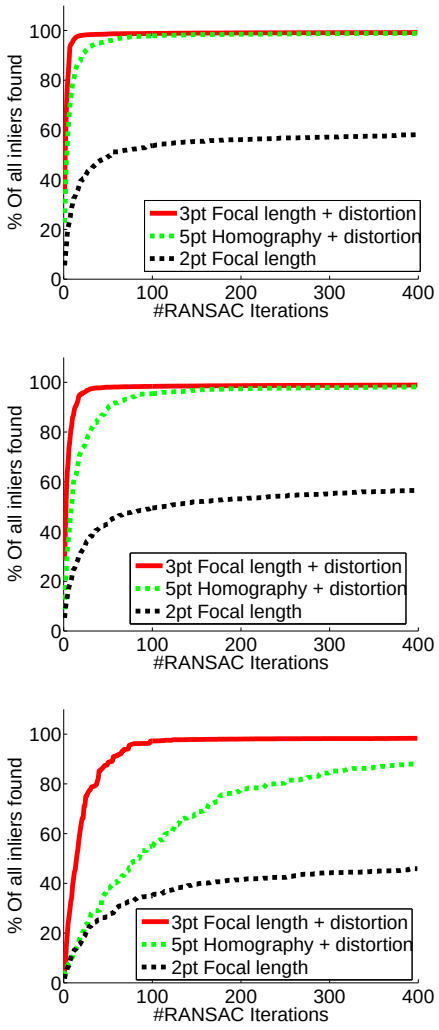

Figure 9.3: Number of inliers found as a function of the number of RANSAC iterations for different percentages of outliers. From top to bottom, the algorithms have been run on examples with 10%, 25% and 50% outliers taken from the *City* data set. In all cases the RANSAC algorithm was run 100 times and mean values were calculated. As can be seen, for moderate to large numbers of outliers, the minimal solver is superior to the overdetermined solver for homography and distortion. In neither case is the two-point solver for focal length competitive. This is expected since the two-point solver assumes zero distortion.

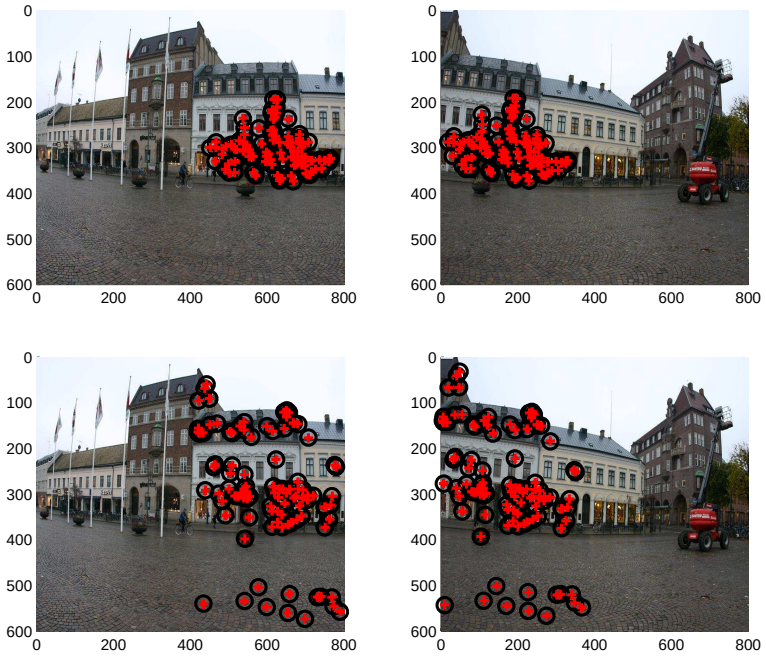

Figure 9.4: Point matches generated using the 2-point algorithm (top), versus our new 3-point algorithm including radial distortion (bottom). Note that the 2-point algorithm is only able to find matches in the central, undistorted portion of the images whereas the 3-point algorithm finds matches all the way to the image edge. This allows for a much more robust image alignment procedure in the presence of radial distortion.

Figure 9.5: Top row: Close-ups on two mistakes made by Autostitch on the sequence *City*. Bottom row: Results obtained using the system presented here.

to recover more inliers, but at the same time increasing the risk of accepting a false match as an inlier. One could also look for inliers in multiple passes by alternating bundle adjustment and inlier selection, but this process is more costly and prone to ending up in local minima. For comparison we ran the Autostitch software by Brown and Lowe [9] on the *City*, and *University* data sets. Despite not explicitly accounting for radial distortion, Autostitch was actually able to stitch together both sequences. However the final result contains visible artifacts which using the system presented in this thesis we were able to avoid. Close-ups of two examples are shown in Figure 9.5.

9.6 Conclusions

We have presented a solution to the problem of estimating rotation, focal length and radial distortion from two images of the same scene undergoing pure rotation using the minimal setup with three point correspondences. The main contribution is that compared to a previous method for this problem, we are able to guarantee that the correct solution is found for all cases. Moreover, we have shown that including radial distortion at the RANSAC stage is beneficial compared to distortion free approaches in terms of number of inliers found and overall precision. An advantage of our algorithm is the ability to recover inliers evenly over the whole image where an algorithm which does not model distortion will only keep point matches close to the centers of the images. Having point matches in the center as well as close to the edges improves recognition performance as well as stability in the subsequent bundle adjustment stage. Compared to a non-minimal algorithm, we in particular do much better at higher outlier

(a) *City*

(b) *University*

(c) *Canal*

Figure 9.6: 360 degree panoramic stitching of the sequences *City*, *University* and *Canal* using the system described in this chapter. The first two sequences were shot using a fish-eye lens while the last sequence was shot with a normal lens. The stitching pipeline includes the following steps: A RANSAC stage where good point matches are established and an initial guess for geometry and calibration is estimated, a pair wise bundle adjustment step to polish the initial estimate, a global bundle adjustment step to further refine internal and external calibration parameters and finally a rendering step with basic blending.

rates since a smaller correspondence set yields a smaller risk of hitting an outlier or a poor quality match.

Finally, we have investigated the practical value of the algorithm on realistic data sets and demonstrated qualitative improvements in the end result compared to a recently published stitching system.

Chapter 10

Pose Estimation

In this application chapter we study the problem of pose estimation, while taking radial distortion and a potentially large number of outliers into account. We give an algorithm that solves for radial distortion, focal length and camera pose using a minimal set of four point correspondences between 3D world points and image points. We use a RANSAC loop to find a set of inliers and an initial estimate for bundle adjustment. As in the preceding chapters, the main advantage compared to previous methods is that the presented minimal solver allows us to handle large radial distortions already at the RANSAC stage. We demonstrate that with the inclusion of radial distortion in an early stage of the process, a broader variety of cameras can be handled than was previously possible. In the experiments, no calibration has been applied to the cameras. Instead we assume square pixels, zero skew and centered principal point and then proceed to estimate only position, orientation, focal length and radial distortion. Although the assumptions are not strictly true, we show that good results are still obtained and thus conclude that in practice, the proposed method is applicable to uncalibrated photographs.

10.1 Introduction

The ability to find the position and the direction in which a camera points based on image information is a classic problem in computer vision. The typical way to solve the problem is to find correspondences between an image taken with a camera with unknown position and a three dimensional model. This method has for example been used in Photo tourism [74]. In this paper we choose to follow the same outline of the algorithm but add one extra component to the model, radial distortion. The enhancement with radial distortion makes it possible to use photos taken with fisheye lenses and other heavily distorted images. See Figure 10.1 for an example.

The oldest papers on localization are from the time before the research field of computer vision existed. Already in 1841 Grunert [37] showed that there can be up to four real solutions to the problem of pose estimation if inner calibration of the camera is known and there are three correspondences between image points and known three dimensional points. For an easier description of the problem and how to solve it, [38] is recommended.

Figure 10.1: Left: An image taken with a fisheye lens. Right: The same image rectified when kernel voting is used to determine the radial distortion

If the inner (linear) calibration is unknown one needs six correspondences between the image and the 3D model. In that case a linear method to find the camera position exists [40]. This method usually gives poor results since digital cameras have square pixels and the principal point close to the center of the image. By not imposing these assumptions on the camera model, too many degrees of freedom are used which makes the model unnecessarily sensitive to noise. These assumptions can however be incorporated and the problem is then to find the pose along with an unknown focal length. In 1995 Abidi and Chandra [1] presented a solution to this problem that worked on planar scenes. Four years later Triggs [85] gave a solution to the same problem that worked well on non-planar scenes. In the same paper he also presented a solution to the same problem but without any assumptions on the principal point of the camera. In 2008 the latest paper [10] on this problem was presented. In this paper Bujnak *et al* presents a solution that works on both planar and non-planar data. In that solution Gröbner basis methods were used to solve the system of polynomial equations that arises in their solution. Gröbner bases were also mentioned in the paper by Triggs and to the authors knowledge this is the first paper in the computer vision community that uses Gröbner basis methods to solve a system of polynomial equations. This is also the method that will be used in this paper to solve the systems of polynomial equations arising in the problem.

The problem of pose estimation with unknown focal length is not a true minimal case with four points, hence no exact solution can be found. In [10] the fact that the problem is over constrained is resolved by ignoring one equation in an early step of the solver and then using the last equation to verify which of the multiple solutions to use. An alternative method to find the focal length is the (2,2) solver in Chapter 7. There a correspondence to another image replaced one of the correspondences to a three dimensional point. That method can also be used for the four points problem if one of the points is substituted by an arbitrary line through that point. In this work we include radial distortion into the model. This adds one degree of freedom and hence the four points problem becomes minimal.

Our contribution here is to incorporate radial distortion already in the RANSAC step in the problem of absolute pose estimation.

10.2 The Camera Model

As in the previous chapters, we employ the standard pinhole camera model [40] with the projection equation

$$\mu \mathbf{u} = P\mathbf{X}, \tag{10.1}$$

Where, P is the camera matrix and μ is the depth (we have used μ for the depth here since λ will be used for the distortion parameter). The camera matrix is now factorized as,

$$P = K[R \mid t], \tag{10.2}$$

where R is a rotation matrix and holds the information in which direction the camera is pointing and t gives information of camera position. K is the calibration matrix of the camera. Given our basic assumptions the matrix has the form:

$$K = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{10.3}$$

where f represents the focal length of the camera.

As in the previous chapters, the pinhole camera model is extended with radial distortion suggested by Fitzgibbon [31]. Let $\mathbf{x}$ denote the image coordinates affected by distortion and let $\mathbf{u}$ be the undistorted coordinates. Then in homogeneous coordinates we have $\mathbf{u} \sim \mathbf{x} + \lambda \mathbf{z}$, where $\mathbf{z} = [0\ 0\ x_1^2 + x_2^2]^T$ and λ is the distortion parameter. To get a consistent radial distortion independent of image size all image coordinates are initially scaled with a factor of

$$scale = \frac{2}{\max(width, height) - 1}, \tag{10.4}$$

which maps all image coordinates to be between minus one and one.

10.3 Pose with Radial Distortion

The setup with unknown radial distortion, focal lengths and pose has eight degrees of freedom; one distortion parameter, the focal length, three translation parameters and three rotation angles. To simplify the calculations we write the calibration matrix as

$$K = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1/f \end{bmatrix} \tag{10.5}$$

and substitute $1/f$ by w. This can be done since the camera matrix is only given up to scale.

The rotation is parameterized with quaternions. This gives the following rotation matrix,

$$R = \begin{bmatrix} a^2 + b^2 - c^2 - d^2 & 2bc - 2ad & 2ac + 2bd \\ 2ad + 2bc & a^2 - b^2 + c^2 - d^2 & 2cd - 2ab \\ 2bd - 2ac & 2ab + 2cd & a^2 - b^2 - c^2 + d^2 \end{bmatrix}. \tag{10.6}$$

Finally, the translation is parameterized by a vector $t = \begin{bmatrix} x & y & z \end{bmatrix}^T$. Using this model, the projection equation becomes

$$\mu(\mathbf{x} + \lambda \mathbf{z}) = P\mathbf{X}. \tag{10.7}$$

At this stage the number of unknowns is nine. But since the camera matrix is only defined up to scale, the number of unknowns can be reduced by one by setting the quaternion parameter a to one. This will result in the rotation matrix also including a scale factor and that the scale of the camera matrix will be fixed. Setting $a = 1$ could potentially yield a poorly conditioned problem for setups where a is close to zero compared to the other rotation parameters. We have however not noted any such problems in the experiments, but this issue could possibly be studied further.

The number of unknowns is now down to eight. Every correspondence between an image point and a world point will give rise to three equations and one additional unknown. Hence four correspondences are necessary to solve the problem. This is a true minimal case were all equations are necessary and in the next two sections we go into detail on how to solve this problem.

10.4 Solving the Minimal Setup

To solve the system generated by (10.7) the equations are first simplified using the freedom in choice of coordinate system. In three dimensional space any similarity transform can be applied without affecting the solutions of the equations. This freedom is used to put the first 3D point at the origin and the second at $\begin{bmatrix}1 & 0 & 0\end{bmatrix}$. In the image space, only rotation and scaling is allowed (not translation) since the focal length is unknown. Hence, the first point is moved to $\begin{bmatrix}1 & 0\end{bmatrix}$. To summarize, the following will hold for every problem setup,

$$\mathbf{X}_1 = \begin{bmatrix}0\\0\\0\\1\end{bmatrix}, \quad \mathbf{X}_2 = \begin{bmatrix}1\\0\\0\\1\end{bmatrix}, \quad \mathbf{x}_1 = \begin{bmatrix}1\\0\\1\end{bmatrix}. \tag{10.8}$$

This choice of coordinate system leads to several simplifications. Firstly, we can express the translation coordinate x in measured image points and the quaternion parameters as follows,

$$x = g_1(a, b, c, d) = \frac{\hat{x}_{21}}{\hat{x}_{22}}(2ad + 2bc) - (a^2 + b^2 - c^2 - d^2). \tag{10.9}$$

Here, $\hat{x}_{22}$ and $u\hat{x}_{22}$ are the coordinates of the second image point. Secondly, y can be set to zero. Finally, the product between the inverted focal length and z can be expressed in the quaternion parameters and the distortion parameter as

$$zw = x(1 + \lambda), \tag{10.10}$$

where x is given by Equation (10.9).

The next step is to include the last two point correspondences and the last information from the second point $\mathbf{x}_2$. This is done by eliminating μ in equation (10.7). The elimination is done by multiplying $P\mathbf{X}$ with the following matrix from the left,

$$B = \begin{bmatrix}0 & -x_3 & x_2\\ -x_3 & 0 & x_1\\ -x_2 & x_1 & 0\end{bmatrix}, \tag{10.11}$$

where $x_3 = 1 + \lambda(x_1^2 + x_2^2)$. This is a rank 2 matrix so not all rows need to be used from the equation $BP\mathbf{X} = 0$. For the second image point only the second row of B is used and for the other two the first and the last row are used. This results in five equations in the five unknowns b, c, d, w and λ.

10.5 Gröbner Basis Solver

To solve the system of polynomial equations constructed above, Gröbner basis methods are used.

As previously discussed, the first step in constructing a Gröbner basis solver is to find out the number of solutions of the system. This can be done once and will hold for all geometrical setups of the same minimal problem. For this case, the computer algebra software Macaulay 2 [36] reports 24 solutions with the given formulation. However, the focal length occurs only in even powers so we will never obtain more than 12 geometrically plausible solutions.

The second step is to expand the initial set of equations. This is done by multiplying the initial equations with a set of monomials. This results in more linearly independent equations with the same solution set and by that it is possible to construct the Gröbner basis. In the problem at hand, the two original equations of lowest degree resulting from multiplication with the last row of B in equation (10.11), are multiplied with λ and w. After that all the nine equations, at this stage, are multiplied with all monomials up to degree four in the unknowns. The result of this expansion is 1134 equations and 720 different monomials. We write this as,

$$C_{\text{exp}}\mathbf{X}_{\text{exp}} = 0, \tag{10.12}$$

where C_{exp} is a 1134×720 matrix holding all coefficients and $\mathbf{X}_{\text{exp}}$ is a 720 elements long column vector with all occurring monomials.

Here, we apply the QR method (with column pivoting and adaptive truncation) described in Chapter 5 . A truncation threshold of 10^{-8} was used.

To construct the multiplication matrix, the permissible monomials and the multiplier variable need to be given. In this application we choose all monomials up to degree three to be in the permissible set and b to be the multiplier variable. The number of permissible monomials with the given choice is 56.

Matlab code for the solver used in this paper is available online at `http://www.maths.lth.se/vision/downloads`.

10.6 Experiments on Synthetic Data

In this section we study some basic properties of the presented algorithm on synthetic data. We start off with a straightforward test on noise free data to check stability and the distribution of plausible solutions. In this experiment, random scenes were generated by drawing four points uniformly from a cube with side length 1000 centered at the origin. A camera was then placed at a distance of 1000 from the origin pointing approximately at the center. The camera was calibrated except for the focal length that was set to around 1000. Radial distortion was then added to the projected points where the distortion parameter was uniformly drawn from the interval $[-0.5, 0]$. Our new minimal

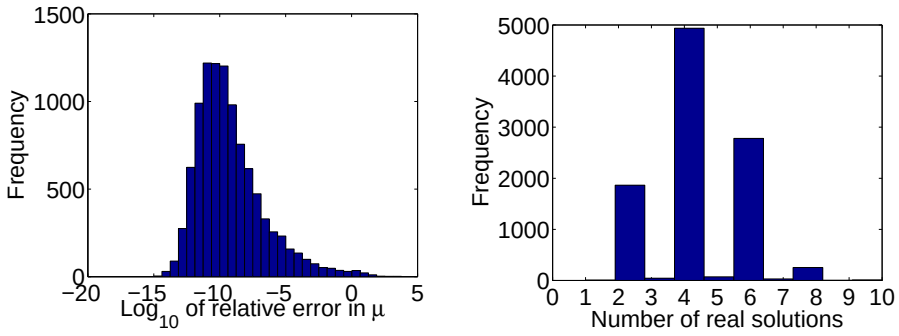

Figure 10.2: Left: Histogram of errors over 10000 runs on noise free data. Right: Histogram over the number of solutions with real positive focal lengths found on the same data.

solver was run on 10000 such instances. Figure 10.2 displays the results of this experiment. The numerical error stays low for most cases. A small number of examples show larger errors, but these do not pose any serious problem since the intended application is RANSAC where lots of instances are solved and only the best one is kept. As previously mentioned, the largest possible number of plausible solutions (real positive focal length) is 12. However, the largest observed number of plausible solutions for the 10000 random instances was 10 and in all but a few exceptions we got 6 solutions or fewer.

To verify that the solver does give accurate results and not just adapts to noise we made an experiment where we measured the relative error in focal length as a function of noise. The setup was the same as in the previous experiment and the standard deviation of the noise was varied between (the equivalent of) zero and three pixels on a 1000×1000 pixel image. For each noise level, 1000 problem instances were tested. The results are given in Table 10.1 and show that our method is robust to noise. Even with as large errors as three pixels, the median error in focal length is less than seven percent.

Noise	Median	75th percentile
0.0	$1.5 \cdot 10^{-11}$	$5.1 \cdot 10^{-10}$
0.5	$1.4 \cdot 10^{-2}$	$4.1 \cdot 10^{-2}$
1.0	$2.3 \cdot 10^{-2}$	$6.8 \cdot 10^{-2}$
2.0	$5.2 \cdot 10^{-2}$	$1.5 \cdot 10^{-1}$
3.0	$6.7 \cdot 10^{-2}$	$1.5 \cdot 10^{-1}$

Table 10.1: The relative error of the focal length for different levels of noise. The noise is given in pixels.

The time consumption of the solver was also measured. On an Intel Core 2 machine with clock rate of 2.13 GHz the average time for a call over 1000 tests was 60 ms in our Matlab implementation.

The next synthetic experiment was designed to investigate how important it is to include radial distortion in the minimal solver. To do that, a setup with 80 inliers and 120 outliers was constructed. Radial distortion was then added to all image points. Three different levels of radial distortion were used, 0, -0.07

and -0.2. Zero distortion was included to test our algorithm compared to a method that assumes no radial distortion. A distortion of -0.07 was used since the normal lens later used in the real experiments has roughly this distortion. This lens is shipped with a consumer level SLR camera. The last value, -0.2, corresponds to the distortion of the fisheye lens used later in the experiments. Noise corresponding to one pixel in a 1000×1000 image was added to each image point. We ran RANSAC on this data and the number of inliers was counted. In the RANSAC loop a point was considered to be an inlier if the reprojection error was less than 0.01 times the mean value of all coordinates of all points given that the origin is in the center of the image. One hundred individual scenes were used for each distortion level. All distortion levels were tested both on the proposed method and on the method of Bujnak *et al* [10]. The algorithm of Bujnak *et al* solves for pose and focal length using four points. The results of this experiment are shown in Figure 10.3 with increasing radial distortion from top to the bottom. Our method is plotted with a solid blue line and Bujnak's in dashed red. The results show as expected that for zero radial distortion it is slightly better not to estimate it. The two other plots show that the use of radial distortion estimation gives a large boost in performance for cases where non-negligible distortion is present. Note especially the large difference even with the small distortion of a standard SLR camera lens.

10.7 Experiments on Real Data

The real world experiments were done in a leave one out manner: We first created a model of a scene using the Photo tourism system "bundler" [74]. To build the model, 93 images from a shopping street were used covering around one hundred meters. An example of one of those images is shown to the right in Figure 10.4. In all these images a regular lens was used. For 29 of these images a second image was taken from the exact same position (a tripod was used to fixate the position) with a fisheye lens. See Figure 10.4 (left) for an example. Then one image at a time (of those images with a corresponding fisheye image) were removed from the model. The pose of the removed image was estimated using the proposed method both for the fisheye image and the regular image. The positions were then compared with the positions estimated by Photo tourism. Note that Photo tourism does not give an exact solution and the authors do not know the precision, but it will still be used as ground truth in this work. The results of this experiment were also compared with the method by Bujnak *et al*.

The pose estimation procedure is carried out in the typical manner. First SIFT features are computed from the image for which the pose should be estimated. Thereafter potential correspondences between the image and the model are established using a nearest neighbor lookup. A point is considered a correspondence if the distance to the closest point times 0.9 is not smaller than the distance to the second closest point. Finally, a RANSAC stage removes false correspondences and local optimization is performed.

The first evaluation on real data is an inliers versus RANSAC iterations comparison. The threshold for a point to be considered an inlier is the same as in the corresponding synthetic experiment. In Figure 10.5 the result of this study is shown. To the left is the result using the fisheye lens and to the right

is the result for the regular lens. The graphs show an average over one hundred trials for the images in Figure 10.4. It is obvious that the use of radial distortion boosts the performance significantly. In some of the tests the method without distortion almost fails to get more inliers than the minimal set. This shows that the use of radial distortion already in the RANSAC step is an important way to increase the performance of the pose estimation. The result confirms what we found in the synthetic experiments.

We next evaluate the proposed method compared to the Photo tourism reconstruction. To do this, the inliers, position, focal length and radial distortion given by the RANSAC step are used for local optimization. The optimization is done for all the unknown parameters. The result is compared with the result when Bujnak's method is used. For that method the same local optimization is performed with the radial distortion initiated with $\lambda = 0$. The scale of the model in this experiment is adjusted so that the errors roughly correspond to meters in camera position. Each of the 29 camera positions used in the experiment is estimated one hundred times so the pose estimation has been performed 2900 times. In Figure 10.6 the result of this experiment is shown. To the left is the result when the fisheye lens is used and to the right is the result for the regular lens.

The precision of Photo tourism that is used for the error measurements is unknown to the authors. Due to that, the result for the smallest errors are hard to interpret. We estimate that on this data set, Photo tourism achieves roughly an accuracy of one to a couple of meters. Thus error measurements below that are not reliable. Nevertheless, one can see clearly that our new minimal algorithm gives much more accurate results compared to the previous method which does not take distortion into account in the RANSAC process.

The results for fisheye lenses was also compared with the result when the regular lens was used. In Figure 10.7 the outcome of that comparison is shown. In the figure, the blue solid line shows the result with the fisheye lens and the red dashed line shows the result with the regular lens. The plot shows that the amount of radial distortion gives almost no impact on the result.

The last experiment is a kernel voting experiment where the distorted image in Figure 10.1 (left) was used. The image was localized 500 times with the minimal solver and the results of the estimations of the radial distortion were used in a kernel voting scheme to find the radial distortion. The results of the kernel voting is shown in Figure 10.8. The peak of the curve is at $\lambda = -0.20$ and that value was used to remove the distortion from the original fisheye image. The undistorted image is shown in Figure 10.1 (right). Notice how the curved lines in the original image have been straightened in the undistorted image. This shows that the estimated radial distortion is reasonably accurate.

10.8 Conclusions

In this chapter a method to estimate the position, rotation, focal length and radial distortion from a minimal set of correspondences to a 3D model has been presented. The parameterization used in this paper gives a system of polynomial equations which we have solved with Gröbner basis methods. This gives a fast and numerical stable method that can be used in a RANSAC loop.

Previous methods have not taken radial distortion into account during the

RANSAC process and in this paper it is shown that the benefits of using radial distortion in the core of the RANSAC loop are significant. This is shown both on synthetic and real data. The large improvements with the fisheye lens come as no surprise due to the heavy radial distortion in that case. More surprising are the large improvements for a regular lens of the SLR camera. The reason for this improvement is probably that there is some radial distortion even in those kinds of lenses and evidently, that distortion can have a large impact on the estimated position.

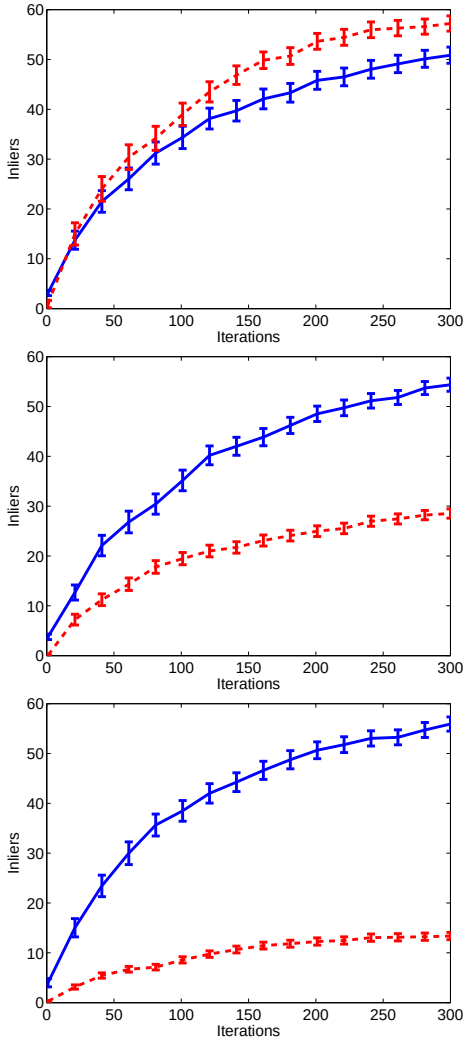

Figure 10.3: Number of inliers given the number of RANSAC iterations for an example with 80 inliers and 120 outliers. Noise was set to correspond to one pixel in a 1000×1000 pixels image. The distortion parameter, λ, was fixed to, from top to bottom, 0, -0.07, -0.2 and one hundred examples were evaluated for each level of distortion. The blue solid line is the method of this paper and the dashed red line is the method proposed by Bujnak *et al.*

Figure 10.4: Test images used for the experiment whose results are shown in Figure 10.5. The images were taken at the exact same position.

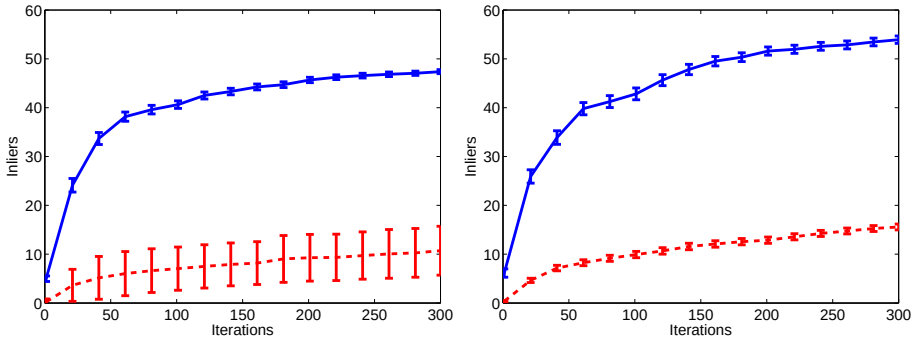

Figure 10.5: Number of inliers versus the number of RANSAC iterations. To the left, a fisheye lens was used and to the right a regular lens was used. The blue solid line is for the method proposed in this paper and the dashed red line is for the method which does not include distortion.

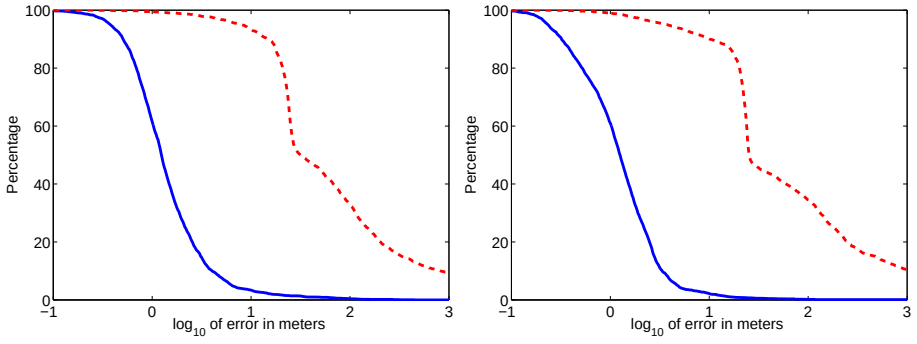

Figure 10.6: The percentage of images with an estimated position further away then a given distance to the position given by Photo tourism. The error is roughly given in meters. Notice the logarithmic scale. The blue solid line is for the proposed method and the red dashed represents method without distortion. The left plot is for the fisheye lens and the right is for a regular lens.

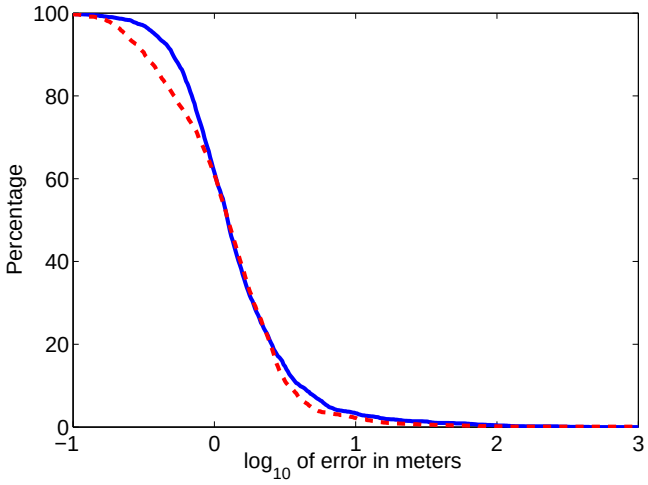

Figure 10.7: Percentage of images estimated with an error (in meters) lower than a varying threshold. The blue solid line represents the distorted images and the red dashed line shows the result for images taken with a regular lens.

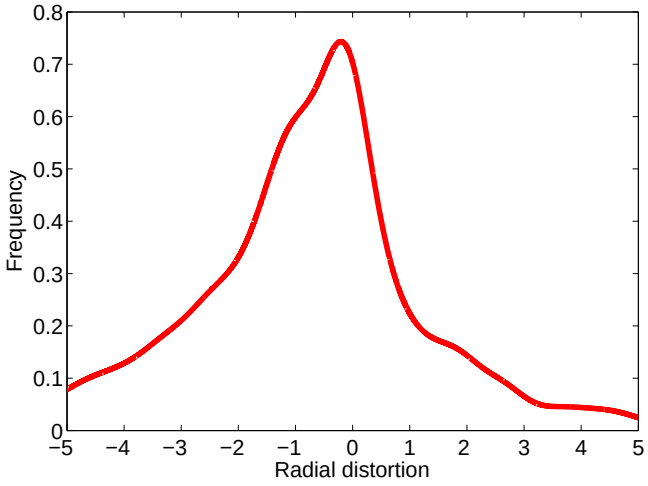

Figure 10.8: Result after kernel voting for radial distortion. The standard deviation of the Gaussian kernel was fixed to 1/3 and the peak of the curve is at $\lambda = -0.20$.

Part III

Bundle Adjustment

Chapter 11

Background and Related Work

Bundle adjustment refers to the accurate refinement of camera and 3D structure parameters based on minimization of the reprojection errors. Typically bundle adjustment is applied once a rough initial reconstruction has been found. Such initial estimates may be obtained in a number of different ways including the use of minimal solvers in a RANSAC framework, incremental resection-intersection approaches, linear estimation of fundamental or trifocal tensors *etc* or using factorization techniques. However, bundle adjustment is much more than only a final polishing step. As noted in among other [26, 86], bundle adjustment is critical in any incremental or online structure from motion system to prevent build-up of errors and will often make the difference between success and complete failure in the reconstruction process.

In this chapter we cover basic aspects of bundle adjustment and provide some motivation for the original research presented in the following chapters.

11.1 Introduction

It is nowadays safe to say that bundle adjustment is a critical component of any image based 3D reconstruction system. This includes both large scale off line batch systems and *e.g* real time SLAM [1] systems.

In bundle adjustment, the reconstruction task is cast as a non-linear optimization problem. The 3D structure and camera parameters are simultaneously refined by adjusting them to minimize the discrepancy between the observations and the image of the 3D model projected into the estimated cameras.

Recently there has been an increased interest in solving for the geometry of very large camera systems with applications such as modelling of large photo collections [74] and urban 3D reconstructions [64, 21]. In trying to achieve such large scale reconstructions, the bundle adjustment stage is commonly a bottle neck and with methods in use today time and memory requirements typically grow cubically in the number of cameras and features [86]. To meet the demand for dealing with increasingly large systems there is thus a need to

[1] Simultaneous localization and mapping

develop methods, which potentially scale better with problem size. This will be the topic of Chapters 12 and 13.

11.2 Problem Formulation

We consider a setup with m cameras $C = (C_1, \ldots, C_m)$ observing n points $U = (U_1, \ldots, U_n)$ in 3D space. An index set $\mathcal{I}$ keeps track of which points are seen in which views by $(i, j) \in \mathcal{I}$ *iff* point j is seen in image i. If all points are visible in all views then there are mn projections. This is not the case in general and we denote the number of image points $n_r = |\mathcal{I}|$. The observation model $f(C_i, U_j)$ yields the 2D image coordinates of the point U_j projected into the view C_i. The input data is a set of observations $\hat{f}_{ij}$ such that

$$\hat{f}_{ij} = f(C_i, U_j) + \eta_{ij}, \tag{11.1}$$

where η_{ij} is measurement noise drawn from a suitable distribution. The unknown parameters $x = (C, U)$ are now estimated given the set of observations by adjusting them to produce a low re-projection error as realized in the following non-linear least squares problem

$$x^* = \underset{x}{\operatorname{argmin}} \sum_{(i,j)\in\mathcal{I}} \|\hat{f}_{ij} - f(C_i, U_j)\|^2. \tag{11.2}$$

This cost function can be motivated in a statistical sense by assuming that the errors are *iid* samples from a Gaussian distribution. Estimating x in a maximum likelihood (ML) sense

$$x^* = \operatorname{argmax} \prod_{(i,j)\in\mathcal{I}} \mathbf{P}[f(C_i, U_j) = \hat{f}_{ij} | C_i, U_j], \tag{11.3}$$

then yields (11.2) if one takes the usual route of minimizing the negative log likelihood instead. Later on we will see how other statistical assumptions on the noise might actually be more reasonable. Luckily these can be fit into the same optimization framework and since the least squares formulation is the cleanest to work with we will stick with that one for now.

11.3 Overview of Optimization Strategies

Arguably, the most popular algorithm for dealing with the non-linear least squares problem is the Gauss-Newton algorithm. Rewriting (11.2) in tidier notation, our task is to solve the following optimization problem

$$x^* = \underset{x}{\operatorname{argmin}}\, r(x)^T r(x), \tag{11.4}$$

where r is the vector of individual residuals $r_{ij} = f(C_i, U_j) - \hat{f}_{ij}$ stacked in a column vector. A second order Taylor expansion of $c(x) = r(x)^T r(x)$ around x yields

$$c(x + \delta x) \approx r^T r + 2((\partial_x r)^T r)^T \delta x + \delta x^T (\sum_i (\partial_x^2 r_i) r_i + (\partial_x r)^T (\partial_x r)) \delta x. \tag{11.5}$$

By introducing $J(x) = \partial_x f = \partial_x r$ and $H_i(x) = \partial_x^2 f_i = \partial_x^2 r_i$ for the Jacobian and Hessian respectively, we obtain

$$c(x + \delta x) \approx r^T r + 2(J(x)^T r)^T \delta x + \delta x^T (\sum_i H_i(x) r_i + J(x)^T J(x)) \delta x. \quad (11.6)$$

If we have a reasonable starting guess so that the residuals r_i are small, then the contribution of the term $\sum_i H_i(x) r_i(x)$ to the Hessian will be small compared to $J^T J$ and can thus be dropped from the expression yielding

$$c(x + \delta x) \approx r^T r + 2(J^T r)^T \delta x + \delta x^T J^T J \delta x. \quad (11.7)$$

Differentiating w.r.t δx and setting the expression equal to zero now yields the equation for the update step in the Gauss-Newton algorithm

$$J(x)^T J(x) \delta x = -J(x)^T r(x). \quad (11.8)$$

The recursion

$$\begin{cases} J(x^k)^T J(x^k) \delta x^k = -J(x^k)^T r(x^k) \\ x^{k+1} = x^k + \delta x^k \end{cases} \quad (11.9)$$

is then iterated until convergence.

Alternatively, we can do a first order expansion inside the norm in the non-linear sum of squares expression to arrive at a linear least squares problem

$$\min_x \|r(x + \delta x)\|^2 \approx \|r(x) + J(x) \delta x\|^2, \quad (11.10)$$

where solving for δx in the usual least squares sense yields exactly (11.8).

11.3.1 The Levenberg-Marquardt Algorithm

One of the main issues in an implementation of the Gauss-Newton algorithm is solving the update equation (11.8). Apart from the fact that this might be a very large system of equations, the system matrix $J^T J$ might not have full rank, or it might have something very close to a non-trivial nullspace. If this is the case then we cannot reliably solve for the update step (at least not using standard methods). A common solution, known as the Levenberg-Marquardt algorithm, is to add a *damping* term λI to $J^T J$ and solve the damped system

$$(J^T J + \lambda I) \delta x = -J^T r, \quad (11.11)$$

which guarantees a unique solution to the update equation (see pseudo code). The larger λ is chosen the closer one gets to a step in the negative gradient direction, hence one can see this approach as interpolating between the Gauss-Newton step and the steepest descent step. Moreover, the larger λ is the shorter the update step will be. It can therefore be a good idea to set λ to a larger value during the initial iterations when one is far from the optimum and the residuals are large. As x gets closer to the optimum, λ can then be decreased to have faster convergence. One such strategy by Nielsen [41], which is also used in the SBA package [60] is given in pseudo code in the LEVENBERG-MARQUARDT procedure. What happens in each iteration is that the state vector and the damping parameter λ gets updated. The line $x_{\text{new}} = x + \delta x$ should be interpreted as "x is updated with δx" and depending on which parameterization is used, this

step might involve some non-linear manipulations. The damping parameter λ is updated by multiplying with a factor between $\frac{1}{3}$ and 2 depending on how well the predicted linear decrease in residual norm agrees with the actual decrease. The degree of correspondence is measured by ρ. If an *increase* in objective function value is obtained then the step is rejected, λ is multiplied by ν and ν is multiplied by 2. This means that λ will grow very quickly if a sequence of rejected steps would occur.

The L-M algorithm is a continuous optimization algorithm and hence needs a stopping criterion to terminate. Common choices include terminating once the magnitude of the gradient falls below a certain threshold, the relative change of residual error falls below a threshold or a maximum number of iterations is reached. Since convergence is quadratic near the optimum (for zero damping and small residuals), the thresholds can usually be set very low without adding many extra iterations. It is often a good idea to combine a threshold criterion with a bound on the number of iterations to avoid extremely long running times in unfortunate cases with poor convergence.

LEVENBERG-MARQUARDT($x_0, \hat{f}$)

$x = x_0$
$r = f(x) - \hat{f}$
$\nu = 2$
Initialize λ
while not converged
  compute $J(x)$
  solve for δx: $(J^T J + \lambda I)\delta x = -J^T r$
  $x_{\text{new}} = x + \delta x$
  $r_{\text{new}} = f(x_{\text{new}}) - \hat{f}$
  if $\|r_{new}\| < \|r\|$
    $\rho = (\|r\|^2 - \|r_{\text{new}}\|^2)/(\delta x^T(\lambda \delta x - J^T r))$
    $r = r_{\text{new}}$
    $x = x_{\text{new}}$
    $\lambda = \lambda * \max(\frac{1}{3}, 1 - (2\rho - 1)^3)$
    $\nu = 2$
  else
    $\lambda = \nu\lambda$
    $\nu = 2\nu$

11.3.2 Trust Regions and Powel's Dog Leg Method

In the Gauss Newton method we are dealing with a non-linear least squares problem by solving a sequence of linear approximations. It is natural to ask how far these can be trusted. In the trust region method a region of trust Ω for the linear approximation is estimated after each iteration. Typically this is done by comparing the decrease in residuals predicted by the linear approximation to the actual decrease obtained by the update step. The idea is then to solve the following constrained least squares problem in each iteration

$$\begin{aligned} &\min_{\delta x} \|J\delta x + r\| \\ &\text{s.t. } \|\delta x\| \in \Omega \end{aligned} \tag{11.12}$$

A common choice for Ω is a ball of radius Δ centered around the current parameter vector x, where Δ is updated after each iteration. There is a close connection to the Levenberg-Marquardt algorithm with this type of trust region. If δx of the non-damped equations fall outside the trust region, then one can show that there is a damping parameter λ such that the solution to the damped system (11.11) is exactly the solution to (11.12) *cf* [67] and there are strategies for selecting such λs. However, once we are sufficiently close to the optimum so that δx lies within the trust region, then the damped system will not yield the same result, causing slower convergence. Based on the insight that the damped G-N step can be seen as an interpolation between steepest descent and the regular G-N step, Powel suggested the following strategy: Compute the G-N step $\delta x_{\mathrm{G-N}}$. If the G-N step is inside Ω then we are done. Otherwise compute the steepest descent step $\alpha\delta x_{\mathrm{SD}}$ (where α is the optimal step length in the steepest descent direction). If this step also falls outside Ω then simply truncate it to be inside Ω and take this as δx_{DL}. If not then move on to take a linear combination of $\alpha\delta x_{\mathrm{SD}}$ and $\delta x_{\mathrm{G-N}}$, $\delta x_{\mathrm{DL}} = \alpha\delta x_{\mathrm{SD}} + \beta(\delta x_{\mathrm{G-N}} - \alpha\delta x_{\mathrm{SD}})$, with β selected such that $\|\delta x_{\mathrm{DL}}\| = \Delta$.

11.4 Sparsity Structure of the Jacobian

The system matrix in (11.8) is $N \times N$ with N equal to the number of unknowns. In problems with a large number of points and cameras this can lead to extremely large equation systems. Solving a linear equation system in general has time complexity $\mathcal{O}(N^3)$ and this step thus forms the major computational bottle neck in the algorithm. To handle the complexity it is vital to make use of the special sparsity structure of the Jacobian. If we partition the Jacobian into a camera part J_C and a point part J_P and order the residuals first by 3D point and then by image so that

$$r = [r_{11}, r_{12}, \ldots, r_{1n_1}, r_{21}, r_{22}, \ldots, r_{2n_2}, \ldots, r_{m1}, r_{m2}, \ldots, r_{mn_m}],$$

then the Jacobian will have the following block structure

$$J = [J_C \; J_P] = \begin{bmatrix} A_1 & B_1 & & & \\ A_2 & & B_2 & & \\ \vdots & & & \ddots & \\ A_n & & & & B_n \end{bmatrix}, \tag{11.13}$$

where

$$A_j = \begin{bmatrix} A_{1j} & & & \\ & A_{2j} & & \\ & & \ddots & \\ & & & A_{mj} \end{bmatrix} \tag{11.14}$$

and

$$B_j = \begin{bmatrix} B_{1j} \\ B_{2j} \\ \vdots \\ B_{mj} \end{bmatrix}. \tag{11.15}$$

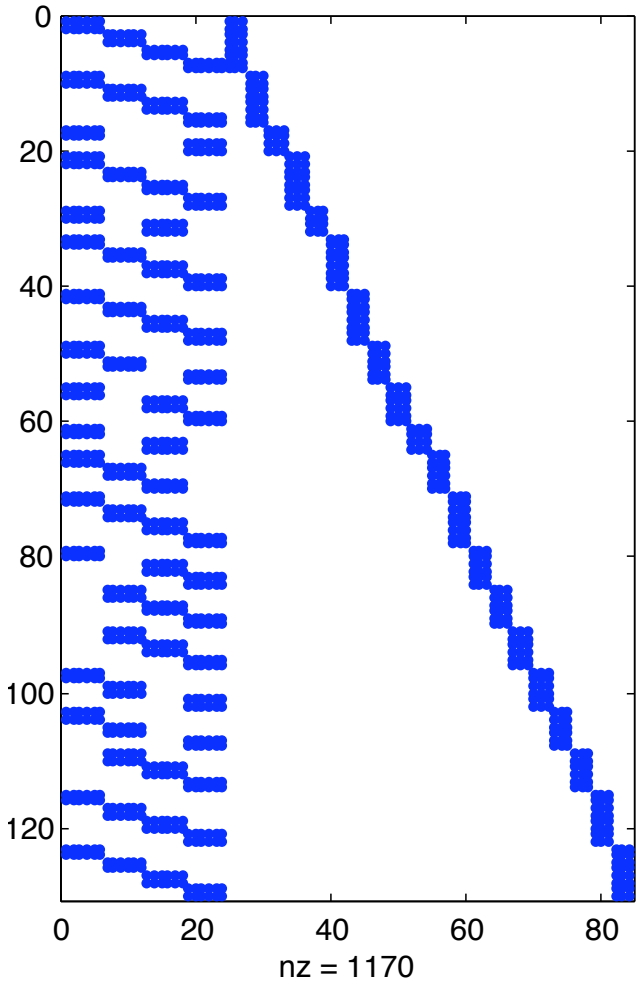

Figure 11.1: Sparsity pattern of the Jacobian for an example with $m = 4$ cameras and $n = 20$ 3D points. Not all points are visible in all views.

Here, the A blocks correspond to camera parameters and the B blocks correspond to 3D point variables. Each block row in (11.15) and (11.14) corresponds to one image projection f_{ij} and thus to two rows in J. If we parameterize a camera with 6 parameters (3 for position and 3 for orientation) and a 3D point with 3 parameters (X, Y, Z) then the size of an A_{ij} block will be 2×6 and a B_{ij} block will have dimension 2×3. This means that each observed image point adds 18 entries to J.

The particular structure of the Jacobian is due to the fact that an image point (feature) f_{ij} depends on the parameters of a single camera C_i and a single 3D point U_j only. Hence any partial derivative $\partial_{x_{i'j'}} f_{ij}$ will vanish where either $i' \neq i$ or $j' \neq j$. In practice, there will be missing block rows in (11.15) and (11.14) owing to the fact that not all cameras observe all 3D points. An example of the sparsity pattern of J can be found in Image 11.1.

The sparsity pattern of J induces some interesting structure in the system matrix J^TJ for the normal equations of the update step as well.

$$J^TJ = \begin{bmatrix} J_C^TJ_C & J_C^TJ_P \\ J_P^TJ_C & J_P^TJ_P \end{bmatrix} \tag{11.16}$$

where

$$J_C^TJ_C = \begin{bmatrix} \sum_{j=1}^n A_{1j}^TA_{1j} & & & \\ & \sum_{j=1}^n A_{2j}^TA_{2j} & & \\ & & \ddots & \\ & & & \sum_{j=1}^n A_{mj}^TA_{mj} \end{bmatrix}, \tag{11.17}$$

$$J_P^TJ_P = \begin{bmatrix} \sum_{i=1}^m B_{i1}^TB_{i1} & & & \\ & \sum_{i=1}^m B_{i2}^TB_{i2} & & \\ & & \ddots & \\ & & & \sum_{i=1}^m B_{in}^TB_{in} \end{bmatrix} \tag{11.18}$$

and

$$J_C^TJ_P = [A_1^TB_1\ A_2^TB_2 \dots A_n^TB_n] = \begin{bmatrix} A_{11}^TB_{11} & A_{12}^TB_{12} & \dots & A_{1n}^TB_{1n} \\ A_{21}^TB_{21} & A_{22}^TB_{22} & & \\ \vdots & & \ddots & \\ A_{m1}^TB_{m1} & & & A_{mn}^TB_{mn} \end{bmatrix} \tag{11.19}$$

Note here that block ij will be missing from (11.19) whenever point j is not visible in view i. An example of the structure of J^TJ is shown in Figure 11.2.

11.4.1 Solving the Sparse Normal Equations

We now show how the sparsity patterns of J and J^TJ can be used to simplify the procedure of solving for δx in (11.8). We introduce $U = J_C^TJ_C$, $V = J_P^TJ_P$ and $W = J_C^TJ_P$ thus obtaining

$$\begin{bmatrix} U & W \\ W^T & V \end{bmatrix} \begin{bmatrix} \delta x_C \\ \delta x_P \end{bmatrix} = -J^Tr = \begin{bmatrix} b_C \\ b_P \end{bmatrix}. \tag{11.20}$$

Next we perform block-wise Gaussian elimination from the bottom up producing

$$\begin{bmatrix} U - WV^{-1}W^T & 0 \\ W^T & V \end{bmatrix} \begin{bmatrix} \delta x_C \\ \delta x_P \end{bmatrix} = \begin{bmatrix} b_C - WV^{-1}b_P \\ b_P \end{bmatrix}, \tag{11.21}$$

which can be solved for δx in two steps by first solving

$$(U - WV^{-1}W^T)\delta x_C = b_C - WV^{-1}b_P \tag{11.22}$$

and then substituting the obtained value of δx_C into

$$V\delta x_P = b_P - W^T\delta x_C \tag{11.23}$$

and solving for δx_P. This procedure is often referred to as Schur complementation and the matrix $S = U - WV^{-1}W^T$ is called the Schur complement. Note

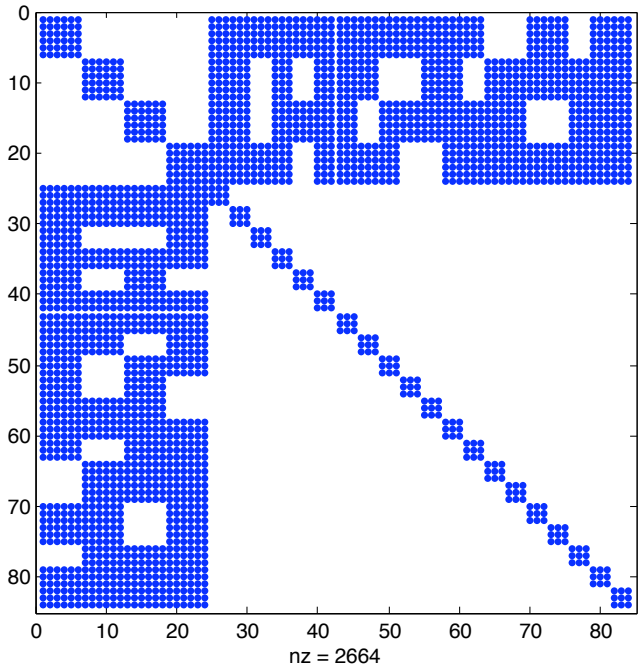

Figure 11.2: Sparsity pattern of $J^T J$ for the same example as in Figure 11.1

here that multiplication with V^{-1} is easily computed since V is block diagonal. This reduces the computational load from solving a $(6m + 3n) \times (6m + 3n)$ system to solving a $6m \times 6m$ system followed by a quick substitution and block diagonal solve. In applications m is usually much smaller than n (about a factor 100 is common) so this typically means substantial savings. For systems with up to a couple of hundred cameras, the most expensive step actually often lies in forming $WV^{-1}W^T$, since W is often quite dense. However, for larger problems the cost of solving the Schur system will dominate the computations.

11.4.2 Complexity and Storage of the Different Steps

To design an efficient bundle adjustment algorithm it is important to have an understanding of the time and memory complexity of the different steps in the iteration. The steps of the Levenberg-Marquardt algorithm which involve significant computations are in order of increasing complexity: computing r, computing J and solving for the update step δx. Computing r and J is linear in the number of image projections n_r with J requiring roughly 10 times that of r in time and storage (depending on how many parameters the camera model has etc.).

Since solving (11.8) is the dominant part of the algorithm it is worth breaking this step down further. Denote by n_c and n_P the number of camera and 3D point parameters respectively ($n_c = 6$ in the case of calibrated cameras). Computing $J^T J$ can be said to consist of three parts; computing the block diagonal matrices U and V and computing the off-diagonal blocks W. By inspecting Equations 11.17, 11.18 and 11.19 we see that each image projection f_{ij} will re-

Operation	Time complexity	Memory complexity	Relative time
r	$\mathcal{O}(n_r)$	$\mathcal{O}(n_r)$	0.1
J	$\mathcal{O}(n_r)$	$\mathcal{O}(n_r)$	1
J^TJ	$\mathcal{O}(n_r)$	$\mathcal{O}(n_r)$	2-3
$WV^{-1}W^T$	$\mathcal{O}(nl^2)$	$\mathcal{O}(nq^2)$	10*
Solving for δx_C	$\mathcal{O}(m^3)$	$\mathcal{O}(m^3)$	N/A
Solving for δx_P	$\mathcal{O}(n+n_r)$	$\mathcal{O}(n+n_r)$	1

Table 11.1: Complexity of the various steps that make up one bundle adjustment iteration. *This figure is only relevant under certain circumstances as discussed in the text and may vary depending on problem structure.

quire computing the $n_C \times n_C$ block $A_{ij}^T A_{ij}$ in U, the $n_P \times n_P$ block $B_{ij}^T B_{ij}$ in V and the $n_C \times n_P$ block $A_{ij}^T B_{ij}$ in W. Hence we can conclude that forming J^TJ is also $\mathcal{O}(n_r)$ for time and storage. Empirically, J^TJ takes about a factor 2-3 to compute compared to J, but in an implementation J and J^TJ can beneficially be computed simultaneously to optimize cache usage.

Dealing with V^{-1} is best done via Cholesky factorization of V, which is $\mathcal{O}(n)$ since V is block diagonal. As previously mentioned, computing the part $WV^{-1}W^T$ of the Schur complement can be a rather demanding step. The Schur complement will have a block-sparse structure with non-zero $n_C \times n_C$ blocks for each pair of cameras (C_i, C_j) which share 3D points. Each such block will be an outer product over camera parameters summed over all shared 3D points. This means that estimating the complexity of this step is a little more involved. If we can assume a representative track length l (the typical number of views which see a given 3D point) then computing $WV^{-1}W^T$ has time complexity $\mathcal{O}(l^2n)$. This will behave a little differently depending on the problem type. If we consider a setup where we cover a successively larger area with cameras, but with the same type of imagery and roughly equal spacing between images, then we can expect the typical track length to stay approximately constant and the number of 3D points to grow linearly with the number of cameras yielding $\mathcal{O}(n)$ or alternatively $\mathcal{O}(m)$ complexity. However, if we consider a case where we cover roughly the same geographic are with an increasing number of cameras, then with some mild assumptions we will see a track length which grows in proportion to the number of new views yielding a faster growing complexity. In practice, the time needed to compute the Schur complement is commonly about a factor 10 longer than that needed to compute J.

The memory complexity for the Schur complement is slightly different. The storage required will be proportional to the number of connected views which cannot be directly calculated from a typical track length. However we can introduce a typical number of neighboring views q which behaves qualitatively in the same manner as l in the above two cases yielding linear/quadratic complexity in the cases with large sparsely covered area vs small densely covered area.

For moderate to large size problems the dominant step in the procedure is solving Equation 11.22, which in the general case has time complexity $\mathcal{O}(m^3)$. Finally (11.23) with complexity $\mathcal{O}(n+n_r)$ is done easily by applying V^{-1} block wise.

11.5 Handling Gauge Freedoms

An issue that must be dealt with when designing a bundle adjuster is the choice of coordinate system. Depending on the problem type the cost function $c(x)$ will be invariant to a group of coordinate transformations such as *e.g* translation, rotation and scale in the case of calibrated cameras. Choosing a particular coordinate frame is known as selecting the *gauge* to work in and the invariance to coordinate transformations is called *gauge freedom*.

Let n_g be the number of degrees of freedom in coordinate system (7 in the calibrated case), then there is an n_g dimensional manifold $\mathcal{N}$ parameterized by $z \in \mathbb{R}^{n_g}$ locally around the current parameter vector x such that for $\forall x' \in \mathcal{N}$, $c(x) = c(x')$. This simply means that if for instance x' were obtained by say translating all coordinates in x by the same amount in a given direction, then the value of the cost function would not change.

While mathematically harmless, this presents an algorithmic problem. Consider $g(z) \in \mathcal{N}$ and denote $\partial_z g(z) = G$ (which is an $n_r \times n_g$ matrix). We then have $\partial_z c(g(z)) = J(y)G = 0$, *i.e* G is an n_g-dimensional null space of J. In particular this causes $J^T J$ to be rank deficient and the consequence is that out-of-the-box Cholesky factorization will break down. There are basically two different strategies for handling this problem. We can either (i) fix the gauge explicitly via parameterization or implicitly by constraining the optimization or (ii) leave the gauge free and try to deal with a rank deficient $J^T J$.

The most common approach is to select a number of parameters equal to the number of gauge freedoms and simply fix these, which would fall into category (i). In the calibrated case one can for instance set one of the camera matrices P_i to $P_i = [I\ 0]$ and fix the distance between this camera and any arbitrary other one to the unit distance. This is sometimes referred to as the trivial gauge and often works well in practice. A potential problem can occur here if the gauge fixing camera happens to be relatively ill determined relative to the other cameras in the starting guess. Correcting this camera then amounts to adjusting all other cameras (since this one is fixed) potentially resulting in poor conditioning.

Other more sophisticated ways of fixing the gauge involve more global parameterizations or adding constraints to the update vector (*e.g* no translation, rotation, scaling) and solving the resulting constrained optimization problem. The issue with these approaches is that they tend to ruin the sparsity structure of $J^T J$, thus significantly increasing time and memory requirements.

An easy way out is available if one uses damping to solve for δx. Then the damped matrix $J^T J + \lambda I$ will have full rank irrespective of any gauge freedoms and one can often simply forget that there exists such a thing as gauge freedoms. Once the optimization is finished one can then map the result to any suitable coordinate frame. This approach is however not recommended since the damping term slows convergence and if the damping term is used to amend the gauge problem, then it will not be possible to let the damping term go to zero near the optimum.

11.6 Parameterization

In general, the state x of the bundle optimization will live on some non-linear manifold. To compute the Jacobian *etc* for optimization we need a parameterization of this. However, it is not necessarily needed to provide a global parameterization. A local parameterization around the current state will do just as fine as long as we can update the current state with a local displacement δx. The main issue in parameterizing the system of cameras typically lies in how to handle rotations. Rotations in $\mathbb{R}^3$ form a 3-dimensional manifold (Lie group) known as SO(3) and some options for parameterization are quaternions, Euler angles and exponential maps.

As long as the update steps are not too large and singularities in the parameterizations can be avoided, then the choice of parameterization is usually not critical. In this work we have chosen exponential maps as the tool for parameterizing rotations. Let Q be a 3×3 rotation matrix. Then there is a unique antisymmetric 3×3 matrix $X = \begin{bmatrix} 0 & x_1 & x_2 \\ -x_1 & 0 & x_3 \\ -x_2 & -x_3 & 0 \end{bmatrix}$ such that $Q = e^X$. However, we can also obtain a local parameterization around a rotation Q_0 as $Q(x) = Q_0 e^X$, with $Q(\mathbf{0}) = Q_0$. This can now be differentiated w.r.t x as

$$\partial_{x_1} Q(x) = [Q_1\ Q_2\ Q_3] \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} = [-Q_2\ Q_1\ \mathbf{0}], \tag{11.24}$$

$$\partial_{x_2} Q(x) = [Q_1\ Q_2\ Q_3] \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{bmatrix} = [-Q_3\ \mathbf{0}\ Q_1], \tag{11.25}$$

$$\partial_{x_3} Q(x) = [Q_1\ Q_2\ Q_3] \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix} = [\mathbf{0}\ -Q_3\ Q_2]. \tag{11.26}$$

11.7 Robust Error Functions

As mentioned above, the least squares formulation corresponds to an assumption of Gaussian noise on the image measurements. In practice this assumption is often violated. The most important deviation from the Gaussian noise model is the occurrence of *outliers* which is a statistical term referring to samples which deviate markedly from the rest of the data or from the observation model. In the Gaussian distribution such samples are extremely unlikely and are thus heavily penalized in the cost function to the extent that a single bad outlier may ruin the whole reconstruction. Typically the estimation is done in to stages with a first step of outlier removal with some combinatorial/randomized algorithm such as RANSAC [30] for outlier removal followed by the bundle adjustment step. Nevertheless, outliers may find their way into the bundle adjustment stage. The most common source of outliers are errors in the matching algorithm which produce incorrect correspondences between points in images. Such outliers typically produce very large errors and are often relatively easy to detect. More challenging are correspondences which are correct, but of low quality perhaps due to severe viewpoint or lighting changes. For these reasons it is usually a good idea to consider distributions with heavier tails than the Gaussian. These induce cost functions, known as robust norms $\rho(r)$, which do not penalize large errors as heavily as the L_2 norm; typically with the form

$$c_\rho(x) = \sum_{ij \in \mathcal{I}} \rho_{ij}(\|f_{ij} - \hat{f}_{ij}\|), \tag{11.27}$$

although there are some alternative forms. The Gauss-Newton approximation can be generalized to such robust cost functions yielding the robust Gauss-Newton algorithm. This can be done in a few different ways yielding different update rules with varying convergence properties. However, since robust statistics in bundle adjustment is not a central topic in this thesis we refer the reader to *e.g* [86] for details. Two common robust norms are the Huber norm (hybrid L_1/L_2)

$$\rho_H(r_i) = \begin{cases} |\frac{r_i}{2\gamma}|^2 & |r_i| \leq \gamma \\ |r_i| + \frac{1}{4} - \gamma & |r_i| > \gamma \end{cases}$$

and the Cauchy norm

$$\rho_C(r_i) = \ln(1 + |\frac{r_i}{\gamma}|^2).$$

Chapter 12

Iterative and Approximate Solutions to the Normal Equations

In this chapter, new techniques are introduced for fast solution of the bundle adjustment problem using iterative linear solvers. As mentioned in Chapter 11, classical bundle adjustment runs into difficulties on large scale problems. In the standard Gauss-Newton method the dominant step is forming and solving the normal equations typically using (sparse) Cholesky factorization, which has cubic complexity in the number of variables.

However, it has been hypothesized that for large problems the method of *conjugate gradients* could be a better choice [86, 65]. So far, this has not been observed and one has mostly obtained rather disappointing convergence rates. This is likely due to a number of reasons. Firstly, it is difficult to select suitable *preconditioners*, which are widely agreed to be necessary for the conjugate gradient method to work well [42]. Secondly, the conjugate gradient method needs to be modified to show its full potential on the least squares problem. To the best knowledge of the author, this has not yet been done in the context of bundle adjustment. Thirdly, bundle adjustment is a non-linear problem usually solved by a sequence of linear approximations. Thus the conjugate gradient algorithm can be applied at two different levels; in the non-linear outer iteration or in an inner iteration as a linear solver for the normal equations. The best approach here is to go for a hybrid of the two via inexact Gauss-Newton methods.

We will begin this chapter by a brief review of the linear and non-linear conjugate gradient algorithms. After this we will address the above mentioned issues. Our main contributions are:

- We apply the CGLS algorithm (instead of the standard CG algorithm), which allows us to avoid forming $J^T J$, where J is the Jacobian, thus saving time and space and improving precision.
- A QR factorization based block-preconditioner, which can be computed in roughly the same time it takes to compute the Jacobian.
- We note that the preconditioned system has "property A" in the sense of

Young [88], allowing us to cut the work per iteration in roughly half.

- An experimental study which sheds some new light on when iterative solvers for the normal equations may be successfully used.

12.1 The Linear and Non-Linear Conjugate Gradient Algorithms

The conjugate gradient algorithm is an iterative method for solving a symmetric positive definite system of linear equations

$$Ax = b, \tag{12.1}$$

introduced by Hestenes and Stiefel [43, 35]. It is also a member of the wider family of Krylov subspace methods. In its basic form it requires only multiplication of the matrix A with a vector, *i.e* no matrix-matrix multiplications and no matrix factorizations. The basic way to apply the conjugate gradient algorithm to the bundle adjustment problem is to form the normal equations $J^T J\delta x = -J^T r$ and set $A = J^T J, b = -J^T r$.

A neat way of approaching iterative methods for symmetric linear systems is to consider minimization of the quadratic form

$$q(x) = \frac{1}{2}x^T Ax - b^T x. \tag{12.2}$$

The gradient of $q(x)$ is easily seen to be the residual of (12.1), $\nabla q(x)^T = Ax - b$, and by setting the gradient equal to zero we see that the minimizer of $q(x)$ is $x^* = A^{-1}b$. This means that to solve (12.1) we can instead solve (12.2) using any optimization method we like. Applying straightforward steepest descent yields the iteration

$$x^{k+1} = x^k + \alpha^k(b - Ax^k) = x^k + \alpha^k s^k,$$

where α^k is a suitable step length at iteration k and $s^k = b - Ax^k$ [1]. This is known as the Richardson iteration and with the right step length this procedure will converge, but typically at a very slow rate. Much faster convergence can be obtained by searching in a direction p^k which is a combination of the negative gradient s^k and the previous search direction p^{k-1}, $p^k = s^k + \beta^{k-1}p^{k-1}$. In the conjugate gradient algorithm, β^{k-1} is selected so that search directions p^k are mutually A-orthogonal meaning $\langle p^i, p^j \rangle_A = {p^i}^T Ap^j = 0,\ \forall i \neq j$ (see pseudo code below). Due to this, in theory the optimum will be found in at most n iterations when A is of dimension $n \times n$. Due to round-off errors this is not the case in practice and what we obtain is an approximation. However, if A is reasonably conditioned, the approximate solution x^k will often reach machine precision with $k \ll n$, thus constituting an effective way to solve (12.1) to the same precision as a direct method.

[1] $b - Ax^k$ is usually denoted r^k in the context of conjugate gradients, but to avoid confusion we reserve r^k for the residuals of the main least squares problem.

CONJUGATE GRADIENT ALGORITHM(x^0, A, b)

// An initial solution x^0 (possibly zero) has to be provided
$s^0 = b - Ax^0, p^0 = s^0, k = 0$
while $|s^k| >$ threshold
$\alpha^k = \frac{{s^k}^T s^k}{{p^k}^T A p^k}$
$x^{k+1} = x^k + \alpha^k p^k$
$s^{k+1} = s^k - \alpha^k A p^k$
$\beta^k = \frac{{s^{k+1}}^T s^{k+1}}{{s^k}^T s^k}$
$p^{k+1} = s^{k+1} + \beta^k p^k$
$k = k + 1$

As mentioned, the conjugate gradient algorithm was originally introduced to solve a system of linear equations, via optimization of the associated quadratic form. However, Fletcher and Reeves generalized the procedure to non-quadratic functions yielding the non-linear conjugate gradients algorithm [32]. Here, only the function $f(x)$ and its gradient $\nabla f(x)$ are available (and not the matrix A). This forces a couple of modifications to the above algorithm. Firstly, in the linear version, the step length α^k is computed analytically to yield the minimum in the search direction p^k. In the non-linear case α^k has to be found using line search. Secondly, s^{k+1} can not be found by updating s^k and hence has to be computed anew at each iteration as $s^{k+1} = \nabla f(x^{k+1})$. Thirdly, β^k can now be computed in a couple of different ways, yielding slightly different behavior. In the original paper by Fletcher and Reeves, the exact same formula as in the linear algorithm was used. However, a slightly different version known as Polak-Ribiere is now more popular

$$\beta^k = \frac{{s^{k+1}}^T (s^{k+1} - s^k)}{{s^k}^T s^k}.$$

12.2 Conjugate Gradients for Least Squares

A naive implementation of the conjugate gradient algorithm for the normal equations would require forming $A = J^T J$ which as discussed in Chapter 11 is a relatively expensive operation. However, we can rewrite the updating formulas for α^k and s^{k+1} as

$$\alpha^k = \frac{{s^k}^T s^k}{(Jp^k)^T (Jp^k)}, \tag{12.3}$$

$$s^{k+1} = s^k - \alpha^k J^T (Jp^k), \tag{12.4}$$

implying that we only need to compute the two matrix-vector multiplications $w^k = Jp^k$ and $J^T w^k$ in each iteration. The resulting algorithm is known as CGLS [6]. The conjugate gradient method belongs to the wider family of Krylov subspace optimizing algorithms. An alternative to CGLS is the LSQR algorithm by Paige and Saunders [70], which is based on Lanczos bidiagonalization. Mathematically CGLS and LSQR are equivalent, but LSQR has in some cases been observed to be slightly more stable numerically. However, in our bundle adjustment experiments these two algorithms have produced virtually identical results. Since LSQR requires somewhat more storage and computation than CGLS we have stuck with the latter.

12.3 Inexact Gauss-Newton Methods

As previously mentioned, there are two levels where we can apply conjugate gradients. Either we use linear conjugate gradients to solve the normal equations $J^T Jdx = -J^T r$ and thus obtain the Gauss-Newton step or we apply non-linear conjugate gradients to directly solve the non-linear optimization problem. Solving the normal equations at each step gives us the good convergence properties of the Gauss-Newton algorithm, but at the expense of running potentially very many conjugate gradient iterations. Applying the non-linear version allows us to quickly take many non-linear steps, but we are likely to need many of these as well and at each step the gradient has to be recomputed. For large systems, computing the gradient will itself be relatively expensive.

However, by making use of the fact that we are dealing with a non-linear *least squares* problem, we can strike a balance between these two approaches. Since $c(x) = r^T(x)r(x)$, we get $\nabla c(x) = -J^T(x)r(x)$ and we see that computing ∇c implies computing the Jacobian J of r. Once we have computed J (and r) we might as well run a few more iterations keeping these fixed. But, since the Gauss-Newton step is anyway an approximation to the true optimum, there is no need to solve the normal equations very exactly and it is likely to be a good idea to abort the linear conjugate gradient method early, going for an approximate solution. This leads to the topic of inexact Newton methods (see *e.g* [67] for more details). In these methods a sequence of stopping criteria are used to abort the inner iterative solver for the update step early. The logical termination quantity here is the relative magnitude of the residual of the normal equations $|s^k|$ (not to be confused with the residual of the least squares system r). A common choice is to terminate the inner CG iteration when

$$\frac{|s^k|}{|\nabla c(x_j)|} < \eta_j,$$

where the sequence $\eta_j \in (0, 1)$ is called a *forcing sequence*. Simply selecting $\eta_j < \eta < 1$, $\forall j$ will ensure convergence, but not at any particular rate. If $\eta_j \to 0$ then we are guaranteed superlinear convergence. We have chosen $\eta_j = \min(0.5, \sqrt{\|\nabla c(x_j)\|})$ as recommended in [67].

12.4 Preconditioning

The success of the conjugate gradient algorithm depends largely on the conditioning of the matrix A. Whenever the condition number $\kappa(A)$ is large convergence will be slow. In the case of least squares, $A = J^T J$ and thus $\kappa(A) = \kappa(J)^2$, so we will almost inevitably face a large condition number [2]. In these cases one can apply *preconditioning*, which in the case of the conjugate gradient method means pre-multiplying from left and right with a matrix E to form

$$E^T AE\hat{x} = E^T b.$$

The idea is to select E so that $\hat{A} = E^T AE$ has a smaller condition number than A. Often E is chosen so that EE^T approximates A^{-1} in some sense. Explicitly

[2] Note that even if we avoid forming $A = J^T J$ explicitly, A is still implicitly the system matrix and hence it is the condition number $\kappa(A)$ we need to worry about.

forming $\hat{A}$ is expensive and usually avoided by inserting $M = EE^T$ in the right places in the conjugate gradient method obtaining the *preconditioned conjugate gradient method*. Two useful preconditioners can be obtained by writing $A = L + L^T - D$, where D and L are the diagonal and lower triangular parts of A. Setting $M = D^{-1}$ is known as Jacobi preconditioning and $M = L^{-T}DL^{-1}$ yields Gauss-Seidel preconditioning.

12.4.1 Block QR Preconditioning

The Jacobi and Gauss-Seidel preconditioners alone do not make use of the special structure of the bundle adjustment Jacobian. Assume for a moment that we have the QR factorization of J, $J = QR$ and set $E = R^{-1}$. This yields the preconditioned normal equations

$$R^{-T}J^TJR^{-1}\delta\hat{x} = -R^{-T}J^Tr,$$

which by inserting $J = QR$ reduce to

$$\delta\hat{x} = -R^{-T}J^Tr$$

and $\delta\hat{x}$ is found in a single iteration step (δx is then obtained from $\delta x = R^{-1}\delta\hat{x}$). Applying R^{-1} is done very quickly through back-substitution. The problem here is of course that computing $J = QR$ is exactly the sort of expensive operation we are seeking to avoid. However, we can do something which is similar in spirit. Consider again the partitioning $J = [J_C,\ J_P]$ used in Chapter 11. Using this, we can do a block wise QR factorization in the following way:

$$J_C = Q_CR_C,\ J_P = Q_PR_P.$$

Due to the special block structure of J_C and J_P respectively we have

$$R_C = R(J_C) = \begin{bmatrix} R(\tilde{A}_1) & & & \\ & R(\tilde{A}_2) & & \\ & & \ddots & \\ & & & R(\tilde{A}_n) \end{bmatrix}$$

and

$$R_P = R(J_P) = \begin{bmatrix} R(B_1) & & & \\ & R(B_2) & & \\ & & \ddots & \\ & & & R(B_n) \end{bmatrix},$$

where

$$\tilde{A}_k = \begin{bmatrix} A_{k1} \\ A_{k2} \\ \vdots \\ A_{kn} \end{bmatrix}.$$

In other words, we can perform QR factorization independently on the block columns of J_C and J_P, making this operation very efficient (linear in the number

of variables) and easy to parallelize. The preconditioner we propose to use thus becomes

$$E = \begin{bmatrix} R(J_C)^{-1} & \\ & R(J_P)^{-1} \end{bmatrix}.$$

Similar preconditioners were used by Golub *et al* in [34] in the context of satellite positioning.

A useful property of this preconditioner is that there is no need to form $J^T J$, which takes time and may introduce round-off errors. To precondition the inexact trust region method described above, one can simply insert $J(Ex)$ instead of Jx wherever a multiplication between J and a vector x occurs.

12.4.2 Property A

A further important aspect of the bundle adjustment Jacobian is that the preconditioned system matrix $\hat{J}^T\hat{J}$ has "property A" as defined by Young in [88].

Definition 17. The matrix A has "property A" *iff* it can be written

$$A = \begin{bmatrix} D_1 & F \\ F^T & D_2 \end{bmatrix}, \tag{12.5}$$

where D_1 and D_2 are diagonal.

The benefit is that for any matrix possessing "property A", the work that has to be carried out in the conjugate gradient method can roughly be cut in half as showed by Reid in [73]. This property can easily be seen to hold for $\hat{J}^T\hat{J}$:

$$\hat{J}^T\hat{J} = \begin{bmatrix} R(J_C) & \\ & R(J_P) \end{bmatrix}^{-T} \begin{bmatrix} J_C^T J_C & J_C^T J_P \\ J_P^T J_C & J_P^T J_P \end{bmatrix} \begin{bmatrix} R(J_C) & \\ & R(J_P) \end{bmatrix}^{-1} = \begin{bmatrix} Q_C^T Q_C & Q_C^T Q_P \\ Q_P^T Q_C & Q_P^T Q_P \end{bmatrix},$$

where $Q_C^T Q_C$ and $Q_P^T Q_P$ are both identity matrices and $Q_P^T Q_C = (Q_C^T Q_P)^T$. Partition the variables into camera and point variables and set $s^k = \begin{bmatrix} s_C^k \\ s_P^k \end{bmatrix}$. Applying Reid's results to our problem yields the following: By initializing so that $\delta x_C = 0$ and $\delta x_P = -J_P^T r$, we will have $s_C^{2m} = s_P^{2m+1} = 0$. We can make use of this fact in the following way (where for clarity, we have dropped the subscript j from the outer iteration):

INNER CG LOOP USING "PROPERTY A"(J, r)

$\eta = 0.1$
$\delta x_C^0 = 0,\ \delta x_P^0 = -J_P^T r,\ \hat{r}^0 = -r - J\delta x^0,\ p^0 = s^0 = J^T\hat{r}^0,$
$\gamma^0 = {s^0}^T s^0,\ q^0 = Jp^0,\ k = 0$
while $\|s^k\| > \eta\|s^0\|$

$\alpha^k = \frac{\gamma^k}{{q^k}^T q^k}$
$\delta x^{k+1} = \delta x^k + \alpha^k p^k$
$\begin{cases} s_C^{k+1} = -\alpha^k J_C^T q^k,\ s_P^{k+1} = 0 & k \text{ odd} \\ s_P^{k+1} = -\alpha^k J_P^T q^k,\ s_C^{k+1} = 0 & k \text{ even} \end{cases}$
$\gamma^{k+1} = {s^{k+1}}^T s^{k+1}$
$\beta^k = \frac{\gamma^{k+1}}{\gamma^k}$
$p^{k+1} = s^{k+1} + \beta^k p^k$
$\begin{cases} q^{k+1} = \beta^k q^k + J_C s_C^{k+1} & k \text{ odd} \\ q^{k+1} = \beta^k q^k + J_P s_P^{k+1} & k \text{ even} \end{cases}$

12.5 Experiments

For evaluation we compare three different algorithms on synthetic and real data. Standard bundle adjustment is performed using the Levenberg-Marquardt algorithm and Cholesky factorization of the Schur complement to solve the normal equations. We henceforth denote this algorithm DBA for direct bundle adjustment. Secondly, we study a straightforward adaptation of the conjugate gradient algorithm to bundle adjustment by using J^TJ as the system matrix and the block diagonal of J^TJ as a preconditioner. We simply refer to this algorithm as CG. Finally, we denote the conjugate gradient method tailored to bundle adjustment as proposed in this chapter CGBA for conjugate gradient bundle adjustment.

In all cases we apply adaptive damping to the normal equations as suggested in [41]. In the case of CGBA, we never form J^TJ and we instead apply damping by using the damped Jacobian

$$J_\lambda = \begin{bmatrix} J \\ \lambda I \end{bmatrix},$$

which can be factorized in the same manner as J for preconditioning.

For clarity, we focus on calibrated cameras only in this work. Including additional parameters such as focal length and distortion parameters presents no problem and fits into the same general framework without modification.

12.5.1 Synthetic Data: When is the CG Algorithm a Good Choice?

An common statement is that standard bundle adjustment is good for small to medium size problems and that Conjugate Gradients should probably be the way to go for large and sparse problems. This is not quite true as we will show with a couple of synthetic experiments. In some cases CG based bundle adjustment can actually be a better choice for quite small problems. On the other hand it might suffer from hopelessly slow convergence on some large very sparse setups. Theoretically, the linear CG algorithm converges in a number of iterations proportional to roughly the square root of the condition number and a large condition number hence yields slow convergence. Empirically, this happens in particular for sparsely connected structures where unknowns in the camera-structure graph are far apart. Intuitively such setups are much less "stiff" and can undergo relatively large deformations with only little effect on the reprojection errors.

To capture this intuition, we have simulated two qualitatively very different scenarios. In the first setup, points are randomly located inside a sphere of radius one centered at the origin. Cameras are positioned uniformly around the sphere at around two length units from the origin pointing roughly towards the origin. There are 10 times as many points as cameras and each camera sees 100 randomly selected points. Due to this, each camera shares features with a large percentage of the other cameras. In the second experiments, points are arranged along a circular wall with cameras on the inside of the wall pointing outwards. There are four points for each camera and due to the configuration of the cameras, each camera only shares features with a small number of other

cameras. For each scenario we have generated a series of configurations with increasingly many cameras and points (from 10 to about 500 cameras). One example from each problem type can be seen in Figure 12.1. For each problem instance we ran both standard bundle adjustment with Cholesky factorization (DBA) and the Conjugate Gradient based bundle adjustment procedure proposed here (CGBA) and recorded the total time until convergence. Since the focus of this experiment was on iterative versus direct solvers, we omitted the comparison CG method. The results of this experiment are perhaps somewhat surprising. For the sphere problem, CGBA is orders of magnitude faster for all but the smallest problems, where the time is roughly equal. In fact, the empirical time complexity is almost linear for CGBA whereas DBA displays the familiar cubic growth. For the circular wall scenario the situation is reversed. While CGBA here turns out to be painfully slow for the larger examples, DBA seems perfectly suited to the problem and requires not much more than linear time in the number of cameras. Note here that the Schur complement in the sphere setup is almost completely dense whereas in the wall case it is extremely sparse. The radically different results on these data sets can probably understood like this. Since the CG algorithm in essence is a first order method "with acceleration", information has to flow from variable to variable. In the sphere case, the distance between cameras in the camera graph is very small with lots of connections in the whole graph. This means that information gets propagated very quickly. In the wall problem though, cameras on opposite sides of the circular configuration are very far apart in the camera graph which yields a large number of CG iterations. For the direct approach "stiffness" of the graph does not matter much. Instead fill-in during Cholesky factorization is the dominant issue. In the wall problem, the Schur complement will have a narrow banded structure and is thus possible to factorize with minimal fill-in.

12.5.2 Community Photo Collections

In addition to the synthetic experiments, we have compared the algorithms on four real world data sets based on images of four different locations downloaded from the Internet: The St. Peters church in Rome, Trafalgar square in London, the old town of Dubrovnik and the San Marco square in Venice. The unoptimized models were produced using the systems described in [74, 75, 3].

The models produced by these systems initially contained a relatively large number of outliers, 3D points with extremely short baselines and very distant cameras with a small field of view. Each of these elements can have a very large impact on the convergence of bundle adjustment (both for iterative and direct solvers). To ensure an informative comparison, such sources of large residuals and ill conditioning were removed from the models. This meant that approximately 10% of the cameras, 3D points and reprojections were removed from the models.

In addition, we used the available calibration information to calibrate all cameras before bundle adjustment. In general this gave good results but for a very small subset of cameras ($< 0.1\%$) the calibration information was clearly incorrect and these cameras were removed as well from the models.

For each data set we ran bundle adjustment for 50 iterations and measured the total time and final RMS reprojection error in pixels. All experiments were done on a standard PC equipped with 32GB of RAM to be able to process large

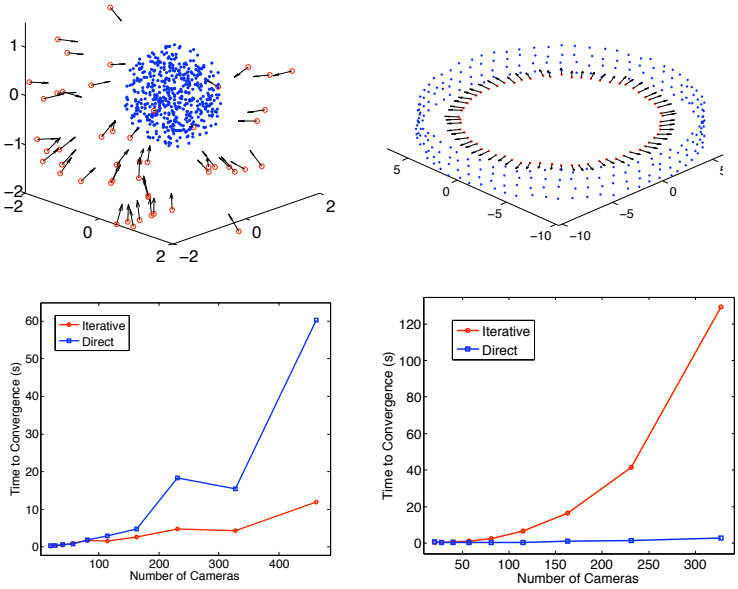

Figure 12.1: Top-left: An instance of the sphere problem with 50 cameras and 500 3D points. Top-right: Points arranged along a circular wall, with 64 cameras viewing the wall from the inside. Bottom-left: Time to convergence vs. number of cameras for the sphere problem. This configuration is ideally suited to CG based bundle adjustment which displays approximately linear complexity. Bottom-right: Time vs. problem size for the circular wall. The CG based solver takes very long to converge, whereas the direct solver shows an almost linear increase in complexity, far from the theoretical $\mathcal{O}(N^3)$ worst case behavior.

Data set	m	n	n_r	Algorithm	Total Time	Final Error (Pixels)
St. Peter	263	129502	423432	DBA	113s	2.18148
				CGBA	441s	2.23135
				CG	629s	2.23073
Trafalgar	2897	298457	1330801	DBA	68m	1.726962
				CGBA	18m	1.73639
				CG	38m	1.75926
Dubrovnik	4564	1307827	8988557	DBA	307m	1.015706
				CGBA	130m	1.015808
				CG	236m	1.015812
Venice	13666	3977377	28078869	DBA	N/A	N/A
				CGBA	230m	1.05777
				CG	N/A	N/A

Table 12.1: Performance statistics for the different algorithms on the four community photo data sets.

data sets. For the CG based solvers, we used a constant $\eta = 0.1$ forcing sequence and set the maximum number of linear iterations to 100. The results can be found in Table 12.1. Basically, we observed the same general pattern for all four data sets. Due to the light weight nature of the CG algorithms, these showed very fast convergence (measured in seconds) in the beginning. At a certain point close to the optimum however, convergence slowed down drastically and in none of the cases did either of the CG methods run to complete convergence. This is likely to correspond to the bound by the condition number of the Jacobian (which we were not able to compute due to the sizes of these problems). In other words, the CG algorithms have problems with the eigenmodes corresponding to the smallest singular values of the Jacobian. This situation makes it hard to give a fair comparison between direct BA and BA based on an iterative linear solver. The choice has to depend on the application and desired accuracy. In all cases, CGBA was about two times faster than CG as expected and in general produced slightly more accurate results.

For the Venice data set, we were not able to compute the Cholesky factorization of the Schur complement since we ran out of memory. Similarly, there was not enough memory in the case of CG to store both J and $J^T J$. While Cholesky factorization in this case is not likely to be feasible even with considerably more memory, a more clever implementation would probably not require both J and $J^T J$ and could possibly allow CG to run on this instance as well. However, as can be seen from the other three examples, the relative performance of CG and CGBA is pretty constant so this missing piece of information should not be too serious.

As observed in the previous section, problem structure largely determines the convergence rate of the CG based solvers. In Figure 12.2, sparsity plots for the Schur complement in each of the four data sets is shown. To reveal the structure of the problem we applied reverse Cuthill-McKee reordering (this reordering was also applied before Cholesky factorization in DBA), which aims at minimizing the bandwidth of the matrix. As can be seen, this succeeds quite well in the case of St. Peter and Trafalgar. In particular in the Trafalgar

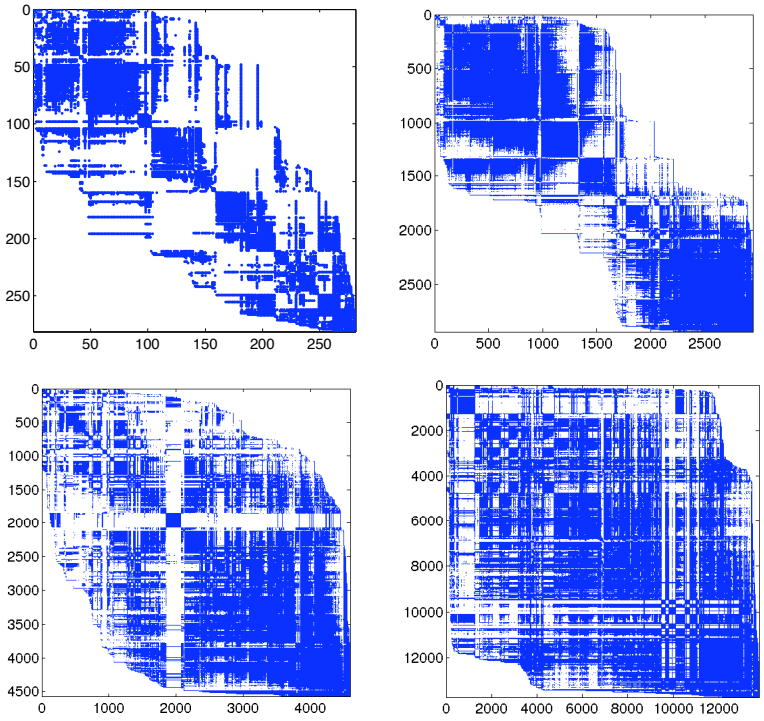

Figure 12.2: Sparsity plots for the reverse Cuthill-McKee reordered Schur complements. Top-left: St. Peter, top-right: Trafalgar, bottom-left: Dubrovnik, bottom-right: Venice

case, two almost independent sets are discovered. As discussed in the previous section, this is a disadvantage for the iterative solvers since information does not propagate as easily in these cases. In the case of Dubrovnik and in particular Venice, the graph is highly connected, which is beneficial for the CG solvers, but problematic for direct factorization.

12.6 Conclusions

In its current state, conjugate gradient based bundle adjustment (on most problems) is not in a state where it can compete with standard bundle adjustment when it comes to absolute accuracy. However, when good accuracy is enough, these solvers can provide a powerful alternative and sometimes the only alternative when the problem size makes Cholesky factorization infeasible. A typical application would be intermediate bundle adjustment during large scale incremental SfM reconstructions. We have in this chapter presented a new conjugate gradient based bundle adjustment algorithm (CGBA) which by making use of "Property A" of the preconditioned system and by avoiding $J^T J$ is about twice as fast as "naive" bundle adjustment with conjugate gradients and more precise. An interesting path for future work would be to try and combine the largely orthogonal strengths of the direct versus iterative approaches. One such idea

would be to solve a simplified (skeletal) system using a direct solver and use that as a preconditioner for the complete system.

Chapter 13

Multiscale Preconditioning

In this chapter we present some results on how the bundle adjustment problem might be preconditioned using domain knowledge. These results were first published in [11], prior to those of Chapter 11. However, we felt that placing them here would give the thesis a more natural progression.

As previously mentioned, bundle adjustment problems are often very large, commonly involving thousands of variables. The traditional Levenberg Marquardt algorithm with a direct sparse solver can be efficiently adapted to the special structure of the problem and works well for small to medium size setups. However, for larger scale configurations the cubic computational complexity makes this approach prohibitively expensive. The natural step here is to turn to iterative methods for solving the normal equations as was done in the previous chapter. However, as was noted, this works well in some cases where the structure makes the problem reasonably well conditioned. In other cases, convergence is disappointingly slow with "bottom up" preconditioners such as the block QR preconditioner introduced there. In the field of large scale numerical linear algebra, there seems to be a rather wide spread consensus that finding the right preconditioners is *the* most important factor for the success of any iterative linear solver. In this chapter, we take a "top down" approach to the problem and ask what domain knowledge can buy us in the case of bundle adjustment. Our basic empirical finding is that convergence is in general fast for small local deformations whereas the real problem lies in more global coarse scale deformations. To address this issue we have experimented with various ways of introducing different representations of the problem which are able handle these deformations in a better way.

In this chapter we make use of multiscale representations, derived from the underlying geometric layout of the problem and show how these can be used to dramatically increase the power of straightforward preconditioners such as Gauss-Seidel.

13.1 Multiscale Preconditioning

In this section we discuss how a multiscale representation can be used to accelerate convergence. The conjugate gradient method is invariant under orthonormal changes of basis. This means that a perfect orthogonal transformation taking

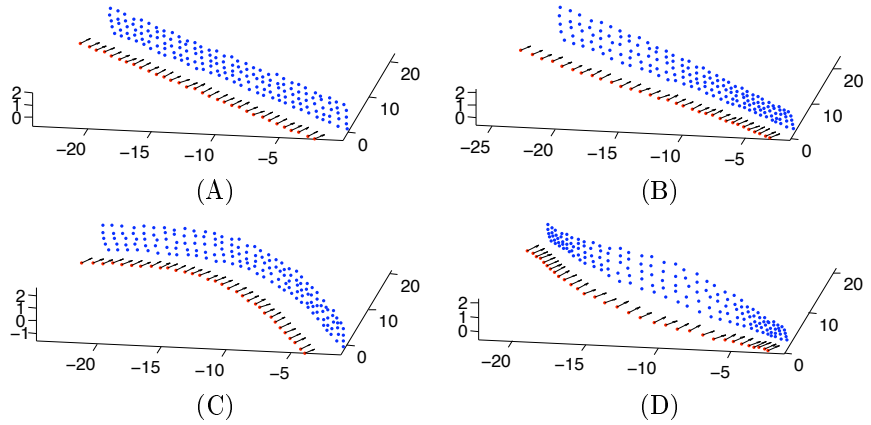

Figure 13.1: (A) Synthetic example with a 32 meter long wall viewed by a sequence cameras. (B) Deformation corresponding to the smallest (non-zero) singular value. This deformation has approximately the effect of a linearly varying sca le along the wall. (C) The next smallest eigenmode corresponding to bending of the wall. (D) The third slowest mode corrsponding to some kind of combination of bending and varying scale.

$J^T J$ to diagonal form would in principle not improve convergence. What we will show is how one can improve convergence rates considerably by combining changes of basis with standard preconditioners. For intuition, consider the left singular vectors of J. Using these as basis vectors would take $J^T J$ to diagonal form and then Jacobi preconditioning would produce the identity matrix leading to convergence in one step in the conjugate gradient method. Of course, the singular vectors are way to expensive to compute, but if we could somehow approximate them, then we should be in a good position. Empirically, large singular values correspond to components representing very local displacements (fine scale) in only a few variables, whereas small singular values correspond to more global (coarse scale) deformations. In the experiments section we consider a synthetic example where a 32 meter long wall is viewed by a sequence of translating cameras. Figure 13.1 shows this example together with deformations corresponding to the three lowest eigenmodes (excluding the seven gauge freedoms) of the Jacobian.

To explicitly tackle this situation we have experimented with various multiscale representations of the problem. These can *e.g* be obtained by hierarchically splitting the set of unknowns. In each step the set of unknown variables is split into two (approximately equally sized) pieces. This gives a dyadic multiscale representation of the problem.

In our changes of basis we have experimented with various approaches. The first approach we tried was using basis vectors corresponding to translation and counter-translation as illustrated in figure 13.2.b and c. The basis is similar to that of the Haar basis, but each division has three basis vectors corresponding to the three translation directions. By proper weighting of the vectors the basis can be made orthogonal. In addition to translation we optionally also add rotation and scaling to a component.

After experimenting with the Haar like basis, we tried a more straightfor-

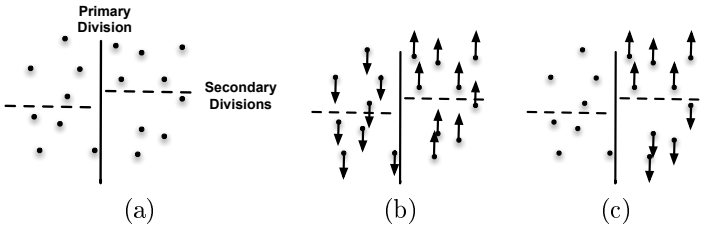

Figure 13.2: Illustration of a multiscale basis at a coarser scale (b) and at a finer scale (c), where points represent camera locations and/or 3D points. The points and/or cameras are hierarchically split into a dyadic basis.

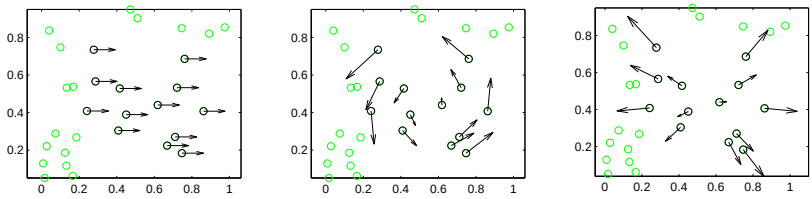

Figure 13.3: Illustration of displacement basis vectors for a specific subset of points (*e.g* camera centers). From left to right: translation, rotation and scaling.

ward multiscale representation by simply letting all elements within a division translate, rotate and scale in the same direction. The Haar representation is in a sense more sophisticated since it by construction yields an orthogonal basis, whereas the simpler representation is highly correlated and a priori we therefore felt that the Haar representation should perform better. To our surprise we have however not been able to observe this so far. On the contrary, the straightforward multiscale representation actually seems to perform slightly better and this is therefore the one which has been used in the experiments.

13.1.1 Constructing A Multiscale Representation for Bundle Adjustment

We now turn to a more detailed discussion of how the multiscale representation can be obtained. To get a manageable sized problem, we factor out the 3D point variables leaving only the camera variables. Now, given a set of cameras with approximately known camera centers $t_1, \cdots, t_m$ we construct a multiscale representation matrix P using a hierarchical binary partitioning of the cameras; At the top level the cameras are split into two groups and these are then recursively split into successively finer groups until some minimum size is reached. We have experimented with various ways to do this partitioning *e.g* using the camera graph and graph clustering algorithms, but so far simple k-means clustering based on the camera locations with two clusters at each level seems to yield the best results. We feel that this is not the end of the story and there should be room to do something more clever on this point.

For each partition $c_i \subset \{t_1, \ldots, t_m\}$, we now add a set of basis vectors x_i, y_i, z_i representing translational displacement to the basis P. For instance,

the basis vector x_i would consist of ones for each position corresponding to an x coordinate of $t_i \in c_i$ and zeros otherwise. Optionally, we also add basis vectors corresponding to rotation in three different planes, $t_i^{xy}, t_i^{yz}, t_i^{zx}$ and scaling s_i. See Figure 13.3 for an illustration of these basis vectors.

The basis vectors are collected in a matrix

$$P = [x_1, y_1, z_1, \ldots, x_m, y_m, z_m, \ldots], \tag{13.1}$$

used to allow multiscale preconditioning. By changing basis according to

$$\tilde{A}_s = P^T A_s P, \quad x = P\tilde{x}, \quad \tilde{b} = P^T b \tag{13.2}$$

we obtain

$$\tilde{A}_s \tilde{x} = \tilde{b}, \tag{13.3}$$

where A_s is the Schur complement $A_s = A - BC^{-1}B^T$ discussed in Chapter 11. We can now write $\tilde{A}_s = \tilde{L} + \tilde{D} + \tilde{L}^T$ and apply Jacobi or Gauss-Seidel preconditioning to $\tilde{A}_s$.

We have found that the best results are obtained when the partitioning is done all the way down to single cameras. At the finest level scaling does of course not apply and what we get there is thus simply the standard basis. This obviously yields an overcomplete basis $\mathcal{P}$ and empirically this seems to be important to obtain good convergence

13.1.2 Efficient Implementation of the Multiscale Transformation

At a first glance, the step 13.2 might look exceedingly expensive since it involves two matrix-matrix multiplications (cubic complexity) to obtain $\tilde{A}_s$. However, since this is a multiscale transformation it should not be implemented as a matrix multiplication. For instance, the Haar wavelet transformation $\hat{x} = P_{\text{haar}}x$ of a vector is of linear complexity (and not quadratic complexity as normal square matrix-vector multiplication). Furthermore, there is actually a way to avoid two dimensional transformation of A_s (both columns and rows). Writing the Jacobian in the form $J = [J_C \; J_P]$ we get

$$\begin{aligned} A_s &= J_C^T J_C - J_C^T J_P (J_P^T J_P)^{-1} J_P^T J_C \\ &= J_C^T (I - J_P (J_P^T J_P)^{-1} J_P^T) J_C \\ &= J_C^T T_P J_C \\ &= (T_P J_C)^T (T_P J_C) \\ &= J_{sC}^T J_{sC}, \end{aligned}$$

where T_P is the projection matrix onto the orthogonal complement of the columns of J_P (and hence symmetric with $T_P^2 = T_P$). This means that we can write the Schur complement A_s as the inner product of a "Schur Jacobian" with itself $A_s = J_{sC}^T J_{sC}$. Using CGLS instead of the normal CG algorithm (as described in the previous chapter), we can thus avoid forming A_s explicitly. A new complication now is however how to apply Gauss-Seidel preconditioning. As it is usually written Gauss-Seidel preconditioning requires the upper (or lower) triangular part of A_s. Fortunately, Björck *et al* have showed how Gauss-Seidel preconditioning can be applied in the context of least squares without explicitly forming the normal equations [6]. By applying the preconditioner in an incremental fashion, we can avoid computing $J_{sC}^T J_{sC}$.

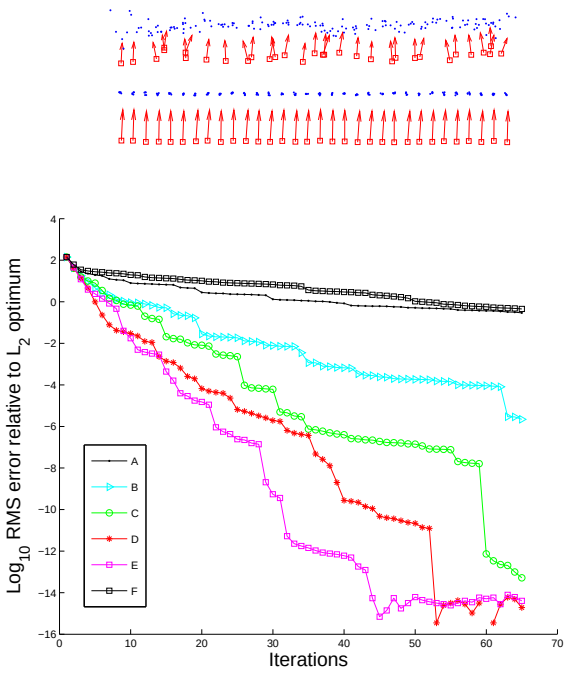

Figure 13.4: Top: The synthetic wall problem viewed from above with ground truth below and perturbed starting guess before bundle adjustment above. Bottom: Log_{10} residual error relative to the optimal solution versus number of iterations for the conjugate gradient method with various forms of preconditioning. A: Jacobi, B: Gauss-Seidel (GS), C: Multiscale representation + GS, D: Multiscale with rotation + GS, E: Multiscale with rotation and scaling + GS F: Multiscale + Jacobi.

13.2 Experimental verification

In a first synthetic experiment we have simulated a long wall (32 meter) with cameras viewing the wall at roughly every meter. In this experiment we calculated the ground truth estimate (not the ground truth reconstruction) by exhaustive Gauss-Newton iterations. A starting guess was chosen so that the error was proportional to $1/s_i$ in the direction v_i, where s_i are the singular values of the Jacobian at the optimum and v_i are corresponding basis vector. This simulates the effect that we may be far off in the directions that are most difficult to estimate. In the experiment we have first reduced the problem to that of only cameras as in Chapter 11. In Figure 13.4 the convergence of different methods are compared. In the figure, the logarithm of the relative difference between the residual error and the optimal residual error is shown as a function of optimization steps. In each step of the algorithms a new residual and Jacobian is calculated followed by 10 iterations of the conjugate gradient method with different choices of bases and preconditioners.

In the figure, curve A illustrates the convergence of the original equation with the Jacobi preconditioner and as can be seen, convergence is quite slow.

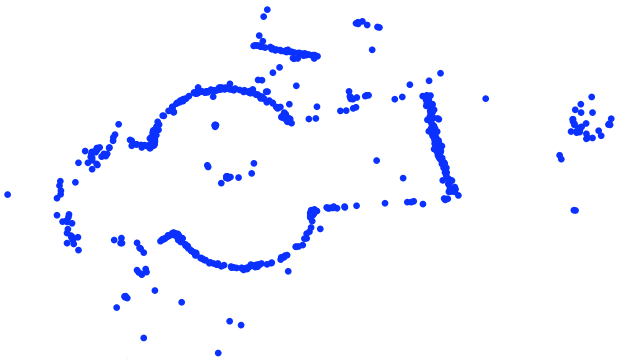

Figure 13.5: Topview of the reconstructed 3D points in the St. Peter data set.

The convergence improves with Gauss-Seidel preconditioning as is illustrated by curve B, but the real boost in convergence is obtained when multiscale representations are combined with Gauss-Seidel preconditioning (curves C, D and E). For all of these there is a steady drop in the RMS error relative to the optimum and convergence within machine precision is achieved after 40-60 iterations. In this experiment we have tried all three approaches multiscale representation with only translations (curve C), with translations and rotations (curve D) and with translations, rotations and scale (curve E). As can be seen, each additional type of large scale deformation additionally facilitates convergence to the optimum. Curve F shows the convergence with multiscale representation and Jacobi only preconditioning. Surprisingly, multiscale together with this most basic form of preconditioning actually does *worse* than only Jacobi preconditioning. This suggests that on its own, the multiscale representation is not sufficiently similar to the singular vectors of the Jacobian and the additional Gauss-Seidel step is needed to bring out its potential.

13.2.1 The St. Peters Basilica

In addition to the synthetic data set we have run the proposed method on a dataset constructed from 285 real photographs of the St. Peters Basilica in Rome, containing 142283 3D points and 466222 image measurements. This data set was used in [65] to evaluate an out of core approach to bundle adjustment. A top view of the reconstructed point cloud of the dataset is shown in Figure 13.5.

On this dataset, we again computed a ground truth estimate by running normal bundle adjustment until complete convergence. Figure 13.6 shows the relative difference to the optimum on a logscale versus the number of iterations. As in the synthetic experiment, we again see a drastic improvement in convergence with the proposed method for preconditioning. Note that on this more difficult data, the Gauss-Seidel preconditioner was not able to improve convergence much on its own.

Ni *et al* optimized the sequence in 49 minutes on a standard PC. After removing 5 images from the data set which did not see any feature points of the model we optimized the set using our approach. The total running time was about 20 minutes, probably to slightly lower accuracy. However, for reference

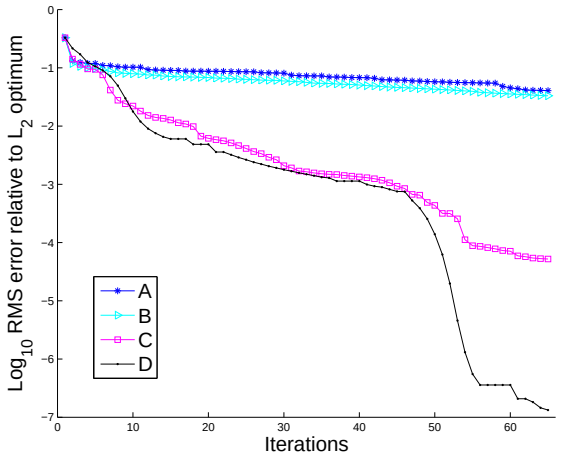

Figure 13.6: Log_{10} Bundle adjustment of the St. Peter data set: Residual error relative to the optimal solution versus number of iterations. A: Jacobi, B: GS, C: Multiscale with rotation and scaling + GS, D: Levenberg Marquardt.

we also made an implementation of standard bundle adjustment using Matlab's sparse direct routines for linear systems and this solver optimized the set in also about 20 minutes to full accuracy. Since running time depends on a large number of fine implementation details, especially for the preconditioned conjugate gradient method and multiscale representations, the results should only be seen as preliminary.

13.3 Conclusions

In this paper we have studied how multiscale representations can be used in conjunction with standard preconditioners for conjugate gradient algorithms for solving large sparse bundle adjustment problems. Our intuition about the problem is that iterative solvers often have convergence problems due to difficulties with large scale, slowly varying deformations. We have tried to tackle this problem by explicitly introducing variables representing various deformations on different scales. The algorithms have been tested on both real and synthetic data sets and the results confirm our hypothesis in the sense that vastly improved convergence rates can be obtained this way.

Since the work presented in this chapter was first published, some effort has been put into obtaining an efficient implementation of the multiscale representation, hoping to beat bundle adjustment based on direct solution of the normal equations for large problems. So far though, that goal has not been reached. One reason for this is that to obtain the kind of powerful improvement in convergence showed in the experiments in this chapter, it seems necessary to include all scales down to partitions with only a couple of cameras. This make the transformed Schur Jacobian $J_{sC}P$ very large in memory.

We still find the results interesting though since they show that large improvements in convergence can be obtained by simply representing the problem

in a different way. We hope that these results might open up the possibility for designing new efficient bundle adjustment algorithms, possibly enabling the solution of problems that were previously out of reach. More investigation is, however, needed in order to exploit these results and to obtain efficient algorithms.

Chapter 14

Conclusions

This chapter concludes the thesis with some closing remarks and some possible future research directions. Since the two main parts of the thesis (polynomial equations and bundle adjustment) are relatively independent the concluding discussion has been split over two sections; one for each topic.

14.1 Polynomial Equations

In a sense, the title of the main paper for this part of the thesis, "Fast and Stable Polynomial Equation Solving and Its Application to Computer Vision" is a bit misleading. The words "fast and stable" seem to imply that we now have the tools to easily and efficiently solve most polynomial equations. This is far from true. Whereas we have come a long way as demonstrated by many examples in this thesis, many problems still remain way out of reach. However, in the cases we have encountered so far, numerical stability has no longer been the limiting factor. Instead what sets the limit for what we can solve numerically is the sheer size of the matrices occurring in the computations, leading to infeasible time and memory requirements. We can typically deal with systems of up to 50 or sometimes even 100 solutions, but above that our methods are simply insufficient. Alternatively, these problems are perhaps inherently so difficult that no really efficient methods to solve them exist.

There are however many interesting questions directly connected to the methods presented in this thesis that still have not been answered. The central theme of this work has been to generalize the action matrix method and exploit as many previously overlooked opportunities to improve speed and stability as possible. The most important discovery here is arguably the large freedom in how a basis can be selected from the set of all monomials $\mathcal{M}$ occurring in an expanded set of equations. This is also where most topics still to be explored can be found. For instance, given an expanded set of equations, one would like to know if it is at all possible to construct a solving basis for this set of equations and in that case how it should be chosen. A solid theoretical understanding of this question and efficient and reliable algorithms for answering this for particular cases would be immensely helpful in applications. Furthermore, except for manual testing, we have no real guidance in how to construct the expanded set of equations. Currently, this is largely an empirical process done by hand. What

degrees should we go to? Should we go to the same degrees for all equations? For all variables? Are there any bounds on what degrees we will need to go to? These questions are most likely very difficult to answer and have been studied for quite a long time in the algebraic geometry community.

This thesis discusses both stability and speed, but looking at the main contributions and the experiments it is evident that numerical stability has been the main focus. Since huge amounts of data is typically paired with real time requirements in computer vision applications, speed is however always of high priority. An interesting topic which has not been much explored yet in computer vision applications of polynomial solvers is real root extraction. It is not uncommon in a case with say 50 solutions that only a handful of these are real. It seems that an algorithm which computes only these could be much faster. A promising possibility here is to compute a total degree Gröbner basis and then convert it to a lexicographical Gröbner basis using the FGLM algorithm [27]. This way one obtains a one-variable polynomial for which the real roots can be bracketed very efficiently using Sturm sequences [45]. This would then have to be done once for each variable.

To summarize, we have introduced a range of techniques which have enlarged the class of problems that can now be handled successfully. However, approaching a particular problem by formulating it as a system of polynomial equations still comes with a degree of uncertainty. It is difficult to tell a priori what the outcome will be in terms of number of solutions, speed and stability. Under the right circumstances, solving a polynomial equation is by far the best method, especially in terms of speed. In other cases the polynomial system is simply too complex to yield anything valuable. Due to this, so far the main application in computer vision of these techniques has been to solve minimal problems of structure from motion. Only time will tell whether the strategy of formulating a given problem as a polynomial equation system will have a broader use.

14.2 Bundle Adjustment

The motivation for studying large scale bundle adjustment is that it constitutes a major computational bottle neck. For structure from motion reconstructions in the order of $10^3 - 10^4$ cameras or more nearly all the computational time can easily be spent doing bundle adjustment. The bundle adjustment step is also the hardest step to parallelize and memory requirements can be extreme. State-of-the-art methods for bundle adjustment in general rely on solving a system of linear equations involving all variables in the system in each iteration. Direct solution of a linear system has time and memory complexity $\mathcal{O}(N^3)$, where N is the number of variables and one can thus easily appreciate the need for alternatives when N grows. We have studied a class of algorithms (conjugate gradient methods) where the computationally most demanding step only involves a matrix-vector product. For a general square matrix, this operation is $\mathcal{O}(\mathcal{N})^{\in}$. However, in bundle adjustment the matrix in this operation (the Jacobian) has a sparsity structure which (under some quite reasonable assumptions) actually makes this operation $\mathcal{O}(N)$. While this looks very promising, the problem is of course that this says nothing about the number of iterations needed to reach the solution. What we have observed in our research is that if one wants to find the optimal solution within machine precision, it seems very

hard (if at all possible) to keep the number of iterations low enough to beat the standard approach with a direct solver. In that sense, the outcome of this part of our research is a disappointment. The reason that the direct solver approach is so hard to beat in practice is probably due to two reasons: (i) modern sparse direct solvers are extremely efficient and good at making use of problem structure, which in practice often means an empirical complexity which is lower than $\mathbf{O}(N^3)$ and (ii) the bundle adjustment problem seems to be naturally ill conditioned (except for some special cases) which makes it a hard target for iterative linear solvers.

There are however also reasons to be optimistic. First of all, even if the conjugate gradient (CG) methods show slow convergence near the optimum, they can at least still *handle* the type of very large problems we are interested in. As was shown in Chapter 12 it is possible, in a relatively small amount of time, to obtain a result which is within subpixel precision of the output from direct bundle adjustment. Thus perhaps the tradeoff is time/memory vs accuracy. The *Venice* data set studied in Chapter 12 is large enough and structured in such a way that we are nowhere near doing bundle adjustment with Cholesky factorization in that case. Still we can run our CG based solver and obtain a good reconstruction (in terms of reprojection errors) in a reasonable amount of time.

Secondly, there are some interesting paths for future work and some indications that the slow convergence of the CG solvers can actually be overcome. Chapter 13 shows that it actually *is* possible to get much better convergence with the right preconditioning. The problem there is of course that the preconditioner itself is too expensive in terms of both memory and computation time to be really practical. The interesting result here is not just that we have found a preconditioner (recall that *e.g* using the inverse of $J^T J$ would be the perfect preconditioner), but *how* it was constructed. In essence what we showed is that by simply parameterizing the problem in a different way it is possible to obtain much improved convergence rates. The hope here is naturally to find a way to compute and apply such parameterizations much more efficiently.

An interesting observation in Chapter 12 is that direct and iterative bundle adjustment respectively show their strengths on quite different types of problems. Bundle adjustment with a direct solver shines where there is much sparsity and a highly structured variable graph since this allows factorization with little fill-in. CG based bundle adjustment on the other hand works best for highly connected problems where the distance in the variable graph between any two unknowns is small - precisely the type of structure which produces large amounts of fill-in during Cholesky factorization and easily causes that approach to break down. As previously mentioned it would thus be interesting to try and combine these largely orthogonal strengths of the direct versus iterative approaches. One such idea would be to solve a simplified (skeletal) system using a direct solver and use that as a preconditioner for the complete system.

Bibliography

[1] M. Abidi and T. 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, 1995.

[2] S. Agarwal, M. K. Chandraker, F. Kahl, D. J. Kriegman, and S. Belongie. Practical global optimization for multiview geometry. In *Proc. 9th European Conf. on Computer Vision, Graz, Austria*, pages 592–605, 2006.

[3] S. Agarwal, N. Snavely, I. Simon, S. M. Seitz, and R. Szeliski. Building rome in a day. In *Proc. 12th Int. Conf. on Computer Vision, Kyoto, Japan*, 2009.

[4] A. Almadi, A. Dhingra, and D. Kohli. A gröbner-sylvester hybrid method for closed-form displacement analysis of mechanisms. *Journal of Mechanical Design*, 122(4):431 – 438, 12 2000.

[5] J. Barreto and K. Daniilidis. Fundamental matrix for cameras with radial distortion. In *IEEE International Conference on Computer Vision*, Beijing, China, 2005.

[6] Å. Björck. *Numerical methods for least squares problems.* SIAM, Society for Industrial and Applied Mathematics, Philadelphia, Pa., 1996.

[7] J. Borenstein, B. Everett, and L. Feng. *Navigating Mobile Robots: Systems and Techniques.* A. K. Peters, Ltd., Wellesley, MA, 1996.

[8] M. Brown, R. Hartley, and D. Nister. Minimal solutions for panoramic stitching. In *Proceedings of the International Conference on Computer Vision and Pattern Recognition (CVPR07)*, Minneapolis, June 2007.

[9] M. Brown and D. G. Lowe. Automatic panoramic image stitching using invariant features. *Int. J. Comput. Vision*, 74(1):59–73, 2007.

[10] M. Bujnak, Z. Kukelova, and T. Pajdla. A general solution to the p4p problem for camera with unknown focal length. In *Proc. Conf. Computer Vision and Pattern Recognition, Anchorage, USA*, 2008.

[11] M. Byröd and K. Åström. Bundle adjustment using conjugate gradients with multiscale preconditioning. In *Proc. British Machine Vision Conference, London, United Kingdom*, 2009.

[12] M. Byröd and K. Åström. Conjugate gradient bundle adjustment. Submitted for review, 2010.

[13] M. Byröd, M. Brown, and K. Åström. Minimal solutions for panoramic stitching with radial distortion. In *Proc. British Machine Vision Conference, London, United Kingdom*, 2009.

[14] M. Byröd, K. Josephson, and K. Åström. Fast optimal three view triangulation. In *Asian Conference on Computer Vision*, 2007.

[15] M. Byröd, K. Josephson, and K. Åström. Improving numerical accuracy of gröbner basis polynomial equation solvers. In *Proc. 11th Int. Conf. on Computer Vision, Rio de Janeiro, Brazil*, Rio de Janeiro, Brazil, 2007.

[16] M. Byröd, K. Josephson, and K. Åström. A column-pivoting based strategy for monomial ordering in numerical gröbner basis calculations. In *The 10th European Conference on Computer Vision*, 2008.

[17] M. Byröd, K. Josephson, and K. Åström. Fast and stable polynomial equation solving and its application to computer vision. *Int. Journal of Computer Vision*, 84(3):237–255, 2009.

[18] M. Byröd, Z. Kukelova, K. Josephson, T. Pajdla, and K. Åström. Fast and robust numerical solutions to minimal problems for cameras with radial distortion. In *Proc. Conf. Computer Vision and Pattern Recognition, Anchorage, USA*, 2008.

[19] E. Cattani, D. A. Cox, G. Chèze, A. Dickenstein, M. Elkadi, I. Z. Emiris, A. Galligo, A. Kehrein, M. Kreuzer, and B. Mourrain. *Solving Polynomial Equations: Foundations, Algorithms, and Applications (Algorithms and Computation in Mathematics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[20] M. Chasles. Question 296. *Nouv. Ann. Math.*, 14(50), 1855.

[21] N. Cornelis, B. Leibe, K. Cornelis, and L. V. Gool. 3d urban scene modeling integrating recognition and reconstruction. *Int. Journal of Computer Vision*, 78(2-3):121–141, July 2008.

[22] D. Cox, J. Little, and D. O'Shea. *Using Algebraic Geometry*. Springer Verlag, 1998.

[23] D. Cox, J. Little, and D. O'Shea. *Ideals, Varieties, and Algorithms*. Springer, 2007.

[24] M. Demazure. Sur deux problemes de reconstruction. Technical Report 882, INRIA, Rocquencourt, France, 1988.

[25] http://www.robots.ox.ac.uk/∼vgg.

[26] C. Engels, H. Stewénius, and D. Nistér. Bundle adjustment rules. In *In Photogrammetric Computer Vision*, 2006.

[27] J. Faugére, P. Gianni, and T. Lazard, D. och Mora. Efficient computation of zero-dimensional gröbner-bases by change of ordering. *Journal of Symbolic Computation*, 16(4):329–344, 1993.

[28] J.-C. Faugère. A new efficient algorithm for computing gröbner bases (f_4). *Journal of Pure and Applied Algebra*, 139(1-3):61–88, 1999.

[29] J.-C. Faugère and A. Joux. Algebraic cryptanalysis of hidden field equation (hfe) cryptosystems using gröbner bases. In *CRYPTO 2003*, pages 44–60, 2003.

[30] M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–95, 1981.

[31] A. W. Fitzgibbon. Simultaneous linear estimation of multiple view geometry and lens distortion. In *Proc. of Computer Vision and Pattern Recognition Conference*, pages 125–132, 2001.

[32] R. Fletcher and C. M. Reeves. Function minimization by conjugate gradients. *The Computer Journal*, 7(2):149–154, February 1964.

[33] C. Geyer and H. Stewénius. A nine-point algorithm for estimating paracatadioptric fundamental matrices. In *Proc. Conf. Computer Vision and Pattern Recognition, Minneapolis, USA*, June 2007.

[34] G. H. Golub, P. Manneback, and P. L. Toint. A comparison between some direct and iterative methods for certian large scale godetic least squares problems. *SIAM J. Sci. Stat. Comput.*, 7(3):799–816, 1986.

[35] G. H. Golub and C. F. van Loan. *Matrix Computations*. The Johns Hopkins University Press, 3rd edition, 1996.

[36] D. R. Grayson and M. E. Stillman. Macaulay2, a software system for research in algebraic geometry. http://www.math.uiuc.edu/Macaulay2/.

[37] J. A. Grunert. Das pothenot'sche problem, in erweiterter gestalt; nebst bemerkungen über seine anwendung in der geodäsie. *Archiv der Mathematik und Physik*, 1:238–248, 1841.

[38] R. M. Haralick, C. Lee, K. Ottenberg, and M. Nölle. Analysis and solutions for the three point perspective pose estimation problem. In *Proc. Conf. Computer Vision and Pattern Recognition*, pages 592–598, 1991.

[39] R. Hartley and P. Sturm. Triangulation. *Computer Vision and Image Understanding*, 68:146–157, 1997.

[40] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2004. Second Edition.

[41] H.B.Nielsen. Damping parameter in marquardt's method. Technical Report IMM-REP-1999-05, Technical University of Denmark, 1999.

[42] M. T. Heath. *Scientific Computing : An introductory Survey*. McGraw-Hill, 1996.

[43] M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49:409–436, Dec 1952.

[44] R. Holt, R. Huang, and A. Netravali. Algebraic methods for image processing and computer vision. *IEEE Transactions on Image Processing*, 5:976–986, 1996.

[45] D. G. Hook and P. R. McAree. Using sturm sequences to bracket real roots of polynomial equations. *Graphics gems*, pages 416–422, 1990.

[46] H. Jin. A three-point minimal solution for panoramic stitching with lens distortion. *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8, June 2008.

[47] K. Josephson and M. Byröd. Pose estimation with radial distortion and unknown focal length. In *Proc. Conf. Computer Vision and Pattern Recognition, Miami, Florida, USA*, 2009.

[48] K. Josephson, M. Byröd, F. Kahl, and K. Åström. Image-based localization using hybrid feature correspondences. In *The second international ISPRS workshop BenCOS 2007, Towards Benchmarking Automated Calibration, Orientation, and Surface Reconstruction from Images*, 2007.

[49] F. Kahl. Multiple view geometry and the L_∞-norm. In *International Conference on Computer Vision*, pages 1002–1009, Beijing, China, 2005.

[50] I. Karasalo. A criterion for truncation of the QR-decomposition algorithm for the singular linear least squares problem. *BIT Numerical Mathematics*, 14(2):156–166, June 1974.

[51] E. Kruppa. Zur ermittlung eines objektes aus zwei perspektiven mit innerer orientierung. *Sitz-Ber. Akad. Wiss., Wien, math. naturw. Kl. Abt*, IIa(122):1939–1948, 1913.

[52] E. Kruppa. Zur Ermittlung eines Objektes aus zwei Perspektiven mit innerer Orientierung. *Sitz-Ber. Akad. Wiss., Wien, math. naturw. Kl. Abt*, IIa(122):1939–1948, 1913.

[53] Z. Kukelova, M. Bujnak, and T. Pajdla. Automatic generator of minimal problem solvers. In *Proc. 10th European Conf. on Computer Vision, Marseille, France*, 2008.

[54] Z. Kukelova, M. Byröd, K. Josephson, T. Pajdla, and K. ström. Fast and robust numerical solutions to minimal problems for cameras with radial distortion. *Computer Vision and Image Understanding*, 114(2):234–244, 2010.

[55] Z. Kukelova and T. Pajdla. A minimal solution to the autocalibration of radial distortion. In *In Proc. Conf. Computer Vision and Pattern Recognition*, 2007.

[56] Z. Kukelova and T. Pajdla. Two minimal problems for cameras with radial distortion. In *Proceedings of The Seventh Workshop on Omnidirectional Vision, Camera Networks and Non-classical Cameras (OMNIVIS)*, 2007.

[57] Lapack - linear algebra package. http://www.netlib.org/lapack, 2008.

[58] D. Lazard. Resolution des systemes d'equations algebriques. *Theor. Comput. Sci.*, 15:77–110, 1981.

[59] H. Li and R. Hartley. A non-iterative method for lens distortion correction from point matches. In *Workshop on Omnidirectional Vision*, Beijing China, Oct. 2005.

[60] M. I. A. Lourakis and A. A. Argyros. Sba: A software package for generic sparse bundle adjustment. *ACM Trans. Math. Softw.*, 36(1):1–30, 2009.

[61] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. Journal of Computer Vision*, 60(2):91–110, 2004.

[62] F. Lu and R. Hartley. A fast optimal algorithm for l_2 triangulation. In *ACCV*, pages 279–288, 2007.

[63] J. Matas, O. Chum, M. Urban, and T. Pajdla. Robust wide-baseline stereo from maximally stable extremal regions. *Image Vision Computing*, 22(10):761–767, 2004.

[64] A. F. Mordohai. Towards urban 3d reconstruction from video, 2006.

[65] K. Ni, D. Steedly, and F. Dellaert. Out-of-core bundle adjustment for large-scale 3d reconstruction. In *Proc. 11th Int. Conf. on Computer Vision, Rio de Janeiro, Brazil*, pages 1–8, 2007.

[66] D. Nistér. An efficient solution to the five-point relative pose problem. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 26(6):756–770, 2004.

[67] J. Nocedal and S. J. Wright. *Numerical optimization*. Springer, Berlin, 2. ed. edition, 2006.

[68] C. Olsson, M. Byröd, and F. Kahl. Globally optimal least squares solutions for quasiconvex 1d vision problems. In *Scandinavian Conference on Image Analysis*, 2009.

[69] C. Olsson, M. Byröd, N. C. Overgaard, and F. Kahl. Extending continuous cuts: Anisotropic metrics and expansion moves. In *International Conference on Computer Vision*, 2009.

[70] C. C. Paige and M. A. Saunders. Lsqr: An algorithm for sparse linear equations and sparse least squares. *ACM Trans. Math. Softw.*, 8(1):43–71, 1982.

[71] J. Philip. A non-iterative algorithm for determining all essential matrices corresponding to five point pairs. *Photogrammetric Record*, 15(88):589–599, Oct. 1996.

[72] R. Pless. Using many cameras as one. In *Proc. Conf. Computer Vision and Pattern Recognition, Madison, USA*, 2003.

[73] J. K. Reid. The use of conjugate gradients for systems of linear equations possessing "property a". *SIAM Journal on Numerical Analysis*, 9(2):325–332, 1972.

[74] N. Snavely, S. M. Seitz, and R. Szeliski. Modeling the world from internet photo collections. *International Journal of Computer Vision*, 80(2):189–210, 2007.

[75] N. Snavely, S. M. Seitz, and R. Szeliski. Skeletal sets for efficient structure from motion. In *Proc. Conf. Computer Vision and Pattern Recognition, Anchorage, USA*, 2008.

[76] H. Stewénius. *Gröbner Basis Methods for Minimal Problems in Computer Vision*. PhD thesis, Lund University, Apr. 2005.

[77] H. Stewénius, C. Engels, and D. Nistér. Recent developments on direct relative orientation. *ISPRS Journal of Photogrammetry and Remote Sensing*, 60:284–294, June 2006.

[78] H. Stewenius, D. Nister, F. Kahl, and F. Schaffilitzky. A minimal solution for relative pose with unknown focal length. *Image and Vision Computing*, 26(7):871–877, 2008.

[79] H. Stewénius, D. Nistér, M. Oskarsson, and K. Åström. Solutions to minimal generalized relative pose problems. In *Workshop on Omnidirectional Vision*, Beijing China, Oct. 2005.

[80] H. Stewénius, F. Schaffalitzky, and D. Nistér. How hard is three-view triangulation really? In *Proc. Int. Conf. on Computer Vision*, pages 686–693, Beijing, China, 2005.

[81] R. Szeliski. *Image Alignment and Stitching*. World Scientific, 2007.

[82] E. H. Thompson. A rational algebraic formulation of the problem of relative orientation. *Photogrammetric Record*, 14(3):152–159, 1959.

[83] P. Torr and A. Zisserman. Robust parameterization and computation of the trifocal tensor. *Image and Vision Computing*, 15(8):591–605, 1997.

[84] P. Torr and A. Zisserman. Robust computation and parametrization of multiple view relations. In *Proc. 6th Int. Conf. on Computer Vision, Mumbai, India*, pages 727–732, 1998.

[85] B. Triggs. Camera pose and calibration from 4 or 5 known 3d points. In *Proc. 7th Int. Conf. on Computer Vision, Kerkyra, Greece*, pages 278–284. IEEE Computer Society Press, 1999.

[86] B. Triggs, P. McLauchlan, R. Hartley, and A. Fitzgibbon. Bundle adjustment: A modern synthesis. In *Vision Algorithms: Theory and Practice*, LNCS. Springer Verlag, 2000.

[87] J. Verschelde. Phcpack: A general-purpose solver for polynomial systems by homotopy continuation. *ACM Transactions on Mathematical Software*, 25(2):251–276, 1999.

[88] D. M. Young. *Iterative solution of large linear systems*. Academic Press, New York, 1971.