A Custom Computing Framework for Orientation and Photogrammetry

by

Paul D. Fiore

B.S., University of Massachusetts at Amherst (1986)
M.S., Tufts University, Medford, Massachusetts (1988)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2000

Signature of Author .....
Department of Electrical Engineering and Computer Science
April 27, 2000

Certified by .....
Berthold K. P. Horn
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

A Custom Computing Framework for Orientation and Photogrammetry

by

Paul D. Fiore

Submitted to the Department of Electrical Engineering and Computer Science
on April 27, 2000, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

There is great demand today for real-time computer vision systems, with applications including image enhancement, target detection and surveillance, autonomous navigation, and scene reconstruction. These operations generally require extensive computing power; when multiple conventional processors and custom gate arrays are inappropriate, due to either excessive cost or risk, a class of devices known as Field-Programmable Gate Arrays (FPGAs) can be employed. FPGAs offer the flexibility of a programmable solution and nearly the performance of a custom gate array.

When implementing a custom algorithm in an FPGA, one must be more efficient than with a gate array technology. By tailoring the algorithms, architectures, and precisions, the gate count of an algorithm may be sufficiently reduced to fit into an FPGA. The challenge is to perform this customization of the algorithm, while still maintaining the required performance. The techniques required to perform algorithmic optimization for FPGAs are scattered across many fields; what is currently lacking is a framework for utilizing all these well known and developing techniques. The purpose of this thesis is to develop this framework for orientation and photogrammetry systems.

Thesis Supervisor: Berthold K. P. Horn
Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

I would like to thank my advisor, Prof. Berthold Horn, for his enthusiasm and advice, and for his patience in listening to my many ideas for all these years. I doubt that I could have found another advisor with as much breadth of knowledge and encouragement of my diverse research interests.

I would like to thank Prof. George Verghese for his patience and guidance; I have greatly benefited from our collaboration. I would also like to thank Prof. Alan Oppenheim for his sage advice throughout my academic career at MIT.

Generous financial support for my work was provided by Sanders, a Lockheed Martin company, of Nashua NH. In particular at Sanders, I would like to thank Dr. Cory Myers and Dr. Webster Dove for their efforts to secure funding for my education. I would also like to thank Dr. Stephen Lang, formerly of Sanders, and Prof. Donald Tufts of the University of Rhode Island, for their encouragement of my efforts to pursue the doctoral degree.

Lastly, and most importantly, I would like to thank my wife Denise and my children, Evan and Alyssa, for their understanding of my need to return to school. Their love, companionship, and encouragement was crucial at every step of the way.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

There is great demand today for real-time computer vision systems, with interest being generated by consumer, industrial, and military applications. We continually witness the increasing usage of electronic imaging systems to improve or automate tasks in our daily lives. In the past, size, weight, cost, and power were hindrances to the widespread deployment of these systems. Now, the availability of inexpensive, high-performance processing brings the possibility of satisfying our ever-increasing performance and cost demands.

Many of these systems are designed to be carried long distances, flown on small unmanned aerial vehicles, or left in a remote location with only a battery for a power source. In all these instance, efficiency of processing is paramount to long mission life. Also of great importance in many problems is robustness and reliability; there is a great penalty for erroneous results. This requirement can drive designers to more computationally-intensive algorithms, further straining the bounds of real-time processing.

Real-time vision systems have been proposed or are currently in use in the following applications:

- Biometrics and access control (ATMs, "smart doors")
- Optical target detection and classification (battlefield processing, near-earth asteroid surveys)
- Image enhancement and understanding (stellar imaging, financial document verification, law-enforcement)
- Scene reconstruction (real-estate, cartography, battle-damage assessment)
- Autonomous vehicle navigation (collision avoidance, robotic transportation, UAVs)
- Intelligent Transportation (Obstacle detection, lane tracking, adaptive cruise control)
- Surveillance and site monitoring (traffic monitoring, activity detection, security cameras)
- Telecommunications (internet video)
- Consumer electronics (video gaming, portable TVs, camcorders, toys)
- Entertainment industry (special effects, motion capture)
- Augmented reality (virtual worlds, image stabilization, flight simulation)
- Medical (CAT, MRI, image guided surgery, optical tomography)

Autonomous navigation is needed for unmanned aerial vehicles. The latest research is focusing on "micro air-vehicles" (MAVs), which can be extremely small fixed-wing aircraft [39], or rotary-wing aircraft, such as the Firefly, a micro-helicopter [154]. Image processing and onboard scene reconstruction has been proposed for flight plan verification and obstacle avoidance. One of the latest experimental vehicles, the MicroStar [25] has a total wingspan of six inches. Clearly, the computational system for onboard image processing must be extremely efficient.

For automobiles, there is great desire to incorporate computer vision processing [19], thereby creating "intelligent transportation systems." Calculation of the time-to-collision and the direction of motion via the focus-of-expansion are two examples of processing that can be used to increase safety; however, the results must be available in a timely fashion. These applications require the determination of camera motion from image sequences, which is termed the *orientation* problem.

Closely related to the orientation problem is the subject of *photogrammetry*, which attempts to perform three dimensional reconstruction of scenes, given photographic images and camera locations. Scene understanding and reconstruction from images is used in model generation for virtual walkthrough in real-estate, law-enforcement, military applications [143], and even the entertainment industry [45]. Algorithms for performing metric reconstructions from images typically require an immense amount of linear algebraic operations.

Orientation and photogrammetry generally require extensive processing power; but when the fastest digital signal processing (DSP) microprocessor is not fast enough, the traditional alternatives have been to add multiple DSP processors or to use custom gate arrays. Multiple DSP microprocessor approaches are expensive, require many components, and consume too much power. The development and debugging of multi-processor software can also be time consuming and problematic. Custom gate array solutions deliver the performance but sacrifice flexibility and require a large engineering investment; the cost, risk, and development time are prohibitive in many applications.

The past decade has seen the emergence of a family of devices that address these shortcomings: the Field-Programmable Gate Array (FPGA). These are an intermediate solution for implementing computationally intensive algorithms [70], [80]. FPGAs offer the flexibility of a programmable solution and nearly the performance of a custom gate array.

The ability to reconfigure an FPGA of course requires extensive on-chip silicon area. For a typical FPGA, most of the silicon is devoted to reprogrammability and routing. This limits the number of user-programmable gates to roughly an order of magnitude less than the number available to a custom gate array technology.

Therefore, when implementing a custom DSP algorithm in an FPGA, one must be more efficient than with a custom gate array technology. By tailoring the algorithms, architectures, and precisions, the gate count of an algorithm may be sufficiently reduced to fit into an FPGA. The challenge is to perform this customization of the algorithm, while still maintaining the required performance.

The techniques required to perform algorithmic optimization for FPGAs are scattered across many diverse fields: numerical analysis, linear algebra, tensor analysis, parallel and distributed computing, computer arithmetic, digital hardware design, digital signal processing, detection and estimation, array processing and subspace tracking, image processing, pattern recognition, sensor fusion, and computer vision. What is currently lacking is a framework for utilizing all these well known and developing techniques.

This thesis develops just such a framework for the computationally intensive orientation

and photogrammetry problems. The key contribution will be a theoretical framework for mapping orientation and photogrammetry algorithms to FPGA hardware. To show the contribution, we will explore the reduction of several key algorithms to an FPGA architecture.

Today, advances will be made by combining hard-earned knowledge from many fields, not just from a specialty area. This leads to the central tenet of this thesis: *Advances in affordable real-time vision systems will occur only when the algorithm domain and device technology are considered simultaneously.*

This form of simultaneous optimization has traditionally not been thought of as a self-contained discipline. Rather, it gets lumped under the heading of "system engineering". Often, new and efficient mappings of algorithms receive less attention than more conventional literature in the specialty fields, because the results are viewed by algorithm specialists as merely an "implementation," whereas the computational specialists view the results as a "special case".

Rather than viewing this customized computing as an "orphan discipline," we now see that in fact this approach is the heart of the meaning of engineering: *use the most appropriate algorithms, approximations, and implementations that you can to make the best design possible*. Indeed, the realization is growing among the engineering community that custom computing solutions to DSP and computer vision problems is a field worthy of study in its own right. For example, the keynote speech [155] at the 1999 *Intl. Conf. on Acoustics, Speech, and Signal Processing*, traditionally a conference on signal processing theory, was on exactly this topic. Additionally, several conferences with this focus (i.e. *IEEE Workshop on Signal Processing Systems*, the *IEEE Workshop on FPGA Custom Computing Machines*, and the *ACM Intl. Symp. on FPGAs*), even refereed journals (i.e. *The Journal of VLSI Signal Processing*) are increasing in popularity.

To be successful, one must become adept at device-specific algorithmic optimization. Clearly, generic methods to optimize any hardware design should be considered; this thesis will propose several novel techniques in this regard. Just as clearly, one must be skilled in the algorithmic field at hand; this thesis examines the orientation and photogrammetry fields, which heretofore have not been extensively examined in light of the custom computing paradign.

In the remainder of this chapter, we highlight the key concepts in orientation and photogrammetry, and then present an outline of the thesis.

1.1 Orientation and Photogrammetry

Determining orientation is the process of locating the position and attitude of two or more objects in space. Various types of information may be available to aid in this process, leading to the different categories of orientation detailed below and in subsequent thesis chapters. Photogrammetry is the process of determining the structure of a scene from image or motion data. Orientation and photogrammetry can thus be considered duals of each other, in the sense that if one is known, solving for the other is greatly simplified.

1.1.1 Types of Orientation

Relative orientation is the process of determining camera motion from two views. For this problem, it is assumed that the camera has been move through a rigid environment. We therefore must determine the rotation and translation (up to a scale factor) that the camera

has undergone. Relative orientation finds use is automated vehicle attitude tracking; it is also a critical component of photogrammetric processing.

Given a sufficient number of matched points, the relative orientation between the two images can be calculated using a geometric formulation. Horn [90], Dron [48], and Hartley [78] address the various computational and stability issues involved. Recent work includes that of Coorg [34] and some Total Least Squares (TLS) approaches [34], [44], [146].

Clearly, erroneous matches can cause extremely poor relative orientation performance. Work work has been done on detecting and removing these outliers, with robust statistics concepts being the most used approaches [71], [75], [77], [128], [170].

Absolute orientation seeks to align two (or more) sets of feature points, whose three dimensional locations have been measured or reconstructed. Horn [89] gave a closed-form quaternion solution, and matrix square-root [92] solution. Arun, Huang, and Blostein [10], and Umeyama [189] provide a closed-form singular value decomposition (SVD) approach. Recent work [87] addresses the absolute orientation problem under nonisotropic location uncertainty. This approach can lead to a generalization of absolute orientation, to include relative and exterior orientation.

Exterior orientation is the process of determining camera location, given feature point locations in the image, as well as the absolute coordinates of the feature points in the scene. This finds application in navigation problems.

Interior orientation is the process of determining the internal configuration of the camera. Generally, we are concerned with obtaining the effective focal length, the principal point, and perhaps the pixel aspect ratio. While not strictly a motion vision task, knowledge of the interior orientation is essential if meaningful Euclidean motion estimation and scene reconstructions are to be attempted.

Determination of camera motion from image sequences continues to be an active area of research. Methods for determining camera motion fall into three classes: feature point matching, optical flow, and direct methods We discuss each of these categories next.

1.1.2 Feature Point Matching

In this method, points in each image that correspond to the same physical point are located. Given a sufficient number of matched points, the relative motion of between the two images can be calculated using straightforward geometrical ideas.

The difficulty in using feature point matching is the features themselves. One or several *feature extractors* are required. These are generally heuristic-based algorithms that attempt to find interesting regions of the images. Often, edges, line segments and corners are the desired features. Also, more complicated methods, that take into account the "trackability" of the image region, have been employed [179], [181].

The amount of motion during the times when the images were taken can also effect the amount of required processing. If the motion is very small (i.e. subpixel), then associating feature points between images is not a difficult task. One could simply associate a feature point in the new image with the closest feature point in the old image.

If the motion is on the order of several pixels to the tens of pixels, the possibility of misassociating features can occur. Assuming the motion does not drastically warp the pattern of the feature points, then one can apply *image registration* techniques [13], [29], [184]. Depending on the quality of the feature extractor or the images themselves, one can have spurious and missing points in the feature point sets. This is not likely to be a problem in the case of very small camera motion, but it is known to lead to significantly

more complicated algorithms in the case of larger motions [53], [182]. Recently, Pilu [149] presented a surprisingly simple registration technique based on the SVD.

For extremely large camera motion between images, significant warping of the feature point patterns may occur. The epipolar constraint is often used to reduce the search region for point matches [51], [129], [158]. Also, large parts of the scene may not be visible in one or the other image, further complicating matters. Correlation type processing and model-based template matching [107], [194], as well as operator assistance are typically the approaches used in this situation.

Of interest in the feature based approach is the quality of the estimate of the features. In many real-time and low cost or low power systems, the amount of computation that one can afford may be limited. One may intentionally use a feature extractor with higher feature pixel location error, in an attempt to reduce computations. The robustness of the feature point matching algorithm, and of the motion recovery algorithm in the presence of feature pixel location noise is therefore of interest.

With the feature point locations in the images and the correct associations, one can recover the translational and rotational parameters of the camera motion. In general, only five correspondences are needed [51] (with a very weak constraint on their locations). If eight correspondences are available, then a set of linear equations results [77], [78], [88], [185]. Often, one uses many more points than this to reduce the effect of feature point location errors.

1.1.3 Optical Flow

The second class of algorithms for recovery of camera motion are based on the calculation of *optical flow*, which is the apparent motion of brightness patterns in an image sequence due to camera motion. Horn and Schunck [93] first proposed the optical flow concept. Optical flow may be used to calculate camera motion, or perhaps more distilled information, such as the "time-to-collision" [8]. Recent work on optical flow based methods include multiscale regularization approach of Luettgen, Karl, and Willsky [119].

The implicit assumption employed when utilizing optical flow is that it in some ways mimics the motion field of the scene. One must thus first compute the optical flow before it can be used to recover camera motion. In general, this is not a trivial operation. Also, the amount of camera motion between images must be small, or one will suffer aliasing during the calculations of the brightness gradients.

As shown by Horn [88], we can easily derive the optical flow constraint equation $E_x u + E_y v + E_t = 0$ at each point in the image. Here, E_x, E_y, and E_t are the spatial and time derivatives of the image brightness, and u and v are the components of the optical flow. We must impose an addition constraint, the "smoothness" assumption, to enable a solution. We thus minimize,

$$\int\int (E_x u + E_y v + E_t)^2 \text{ dx dy} + \alpha \int\int (u_x^2 + u_y^2 + v_x^2 + v_y^2) \text{ dx dy} \tag{1.1}$$

where α must be determined interactively to trade off the degree of smoothing (a recent paper has an automated method for this [137]).

With the optical flow in hand, one must then actually calculate the motion parameters. Horn [88] shows how this is done for the general case of rotation and translation, as well as for simplifying cases with either translation or rotation. For the general case, one has to solve for the six parameters iteratively. The rotational parameters are linear functions of the

translational parameters, so these are easily obtained for each iteration. The translational parameters satisfy second order equations, so a numerical iteration is required.

1.1.4 Direct Methods

The third and final class of algorithms for recovery of camera motion only use spatial and time derivatives of image brightness data. These methods are also termed *direct methods* because we directly calculate motion parameters without first calculating optical flow. This approach seems attractive because complicated feature extractors and optical flow calculation can be avoided. Here of course, as with the optical flow, the camera motion must be small during the time between images, or aliasing will result.

Horn and Weldon [94] analyze in detail directly using the spatial and temporal brightness gradients. They show that for restricted motions (i.e. pure rotation or pure translation) that very simple algorithms result for recovery of the motion parameters. Also, geometric insight is given to explain why some parameters can be better estimated than others. Negahdaripour and Horn [136] analyze the case of a planar scene in great detail. Several iterative as well as a closed form essential matrix solution are given.

Recent work on the direct methods include the work of Gupta and Kanal [74], who show how to perform gradient-based direct methods without actually calculating gradients. Other recent work includes [73].

Optical flow and direct methods are popular because the problematic feature extraction step is eliminated. The drawback of these methods using conventional DSP microprocessors is the extremely high processing load of performing linear algebraic operations on all the image pixels [48]. Of course, custom computational processors, based on gate arrays and FPGAs, have been proposed for these operations.

1.1.5 Photogrammetry

Closely related to the orientation problem is the subject of photogrammetry, which attempts to perform three dimensional reconstruction of scenes, given photographic images and camera locations. Although well-established, there is continuing interest in this field [46], [132], [164]. As an example, two views of a synthetic scene are shown in Figure 1-1. Corresponding feature points were extracted and matched, and are shown in the figure. Conventional relative orientation was used to first determine the camera displacement. Then, a geometric reconstruction was performed, as shown in Figure 1-2. Good results were obtained here because the images were relatively free of noise; significantly worse reconstructions occur with real images, unless more computationally intensive approaches are used.

With relative orientation known or estimated, one can formulate the 3D reconstruction problem as the approximate intersection of multiple rays in space. A different approach (taken by Thompson [178]) is to combine known or estimated relative orientation with photometric stereo ideas.

1.1.6 Multiple Views

Researchers have sought to use multiple views to further improve the accuracy of the estimated camera motion, and for scene reconstruction. Accurate scene reconstruction can also be used to aid on obstacle avoidance for moving platforms, and for recognition of known objects in the environment.

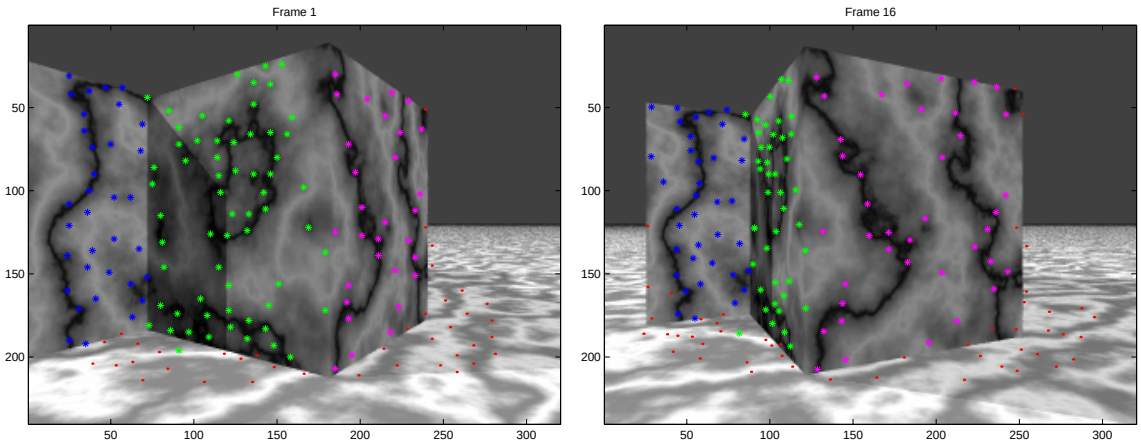

Figure 1-1: Two views from a sequence (unknown motion). Marked points in images were identified and associated by a feature extractor.

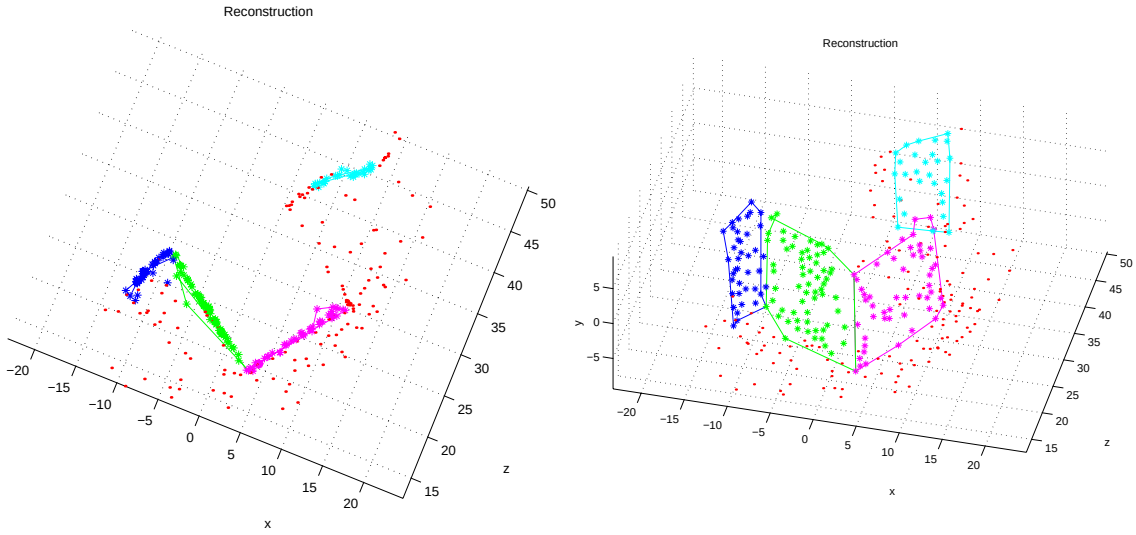

Figure 1-2: Metric reconstruction of marked points.

Historically, scene reconstruction was first performed by photogrammetrists, with the emphasis placed on accuracy. A typical application was to build maps from aerial photographs. Since data collection was (and still is) rather expensive, the emphasis on highly accurate methods is understandable.

More recently, the computer vision community has taken great interest in this problem, but from a different viewpoint. The emphasis is now placed on fast and robust methods, in order to construct systems that can act in real-time.

A type of multiple view orientation problem arises when an image sequence is captured from a moving video camera. Recursive estimation techniques are the appropriate computational model for this situation. Heel [83] formulates the problem as an iterated, extended Kalman filter, with the reconstructed pixel depths as the state variables. Broida, Chandrashekhar, and Chellappa [28] and Azerbayejani and Pentland [12] track feature points. A review of other recursive techniques can be found in [4]. A subspace-tracking approach under the assumption of nonaccelerating motion is taken by Chaudhuri, *et al.* [31]. In the

array processing field, a related method by Tufts, Real, and Cooley [188] promises to significantly reduce computational throughput of subspace tracking with minimal degradation on performance. Multiple view orientation and reconstruction, using tensor concepts are also receiving attention [169].

Lately, there has also been a great deal of interest in using *projective geometry* for camera motion and scene reconstruction. These methods typically try to solve the problem using uncalibrated cameras. Faugeras *et al.* [52] outline the workings of a system to the computer-aided modeling of urban scenes. Robust estimation of matches and projective geometric methods are extensively used. Heyden [86] uses the properties of projective transformations to determine the minimal number N of point correspondences visible in M images required for reconstruction using uncalibrated cameras. Basically, Heyden shows that using only linear constraints, the (N, M) pairs $(8, 2)$, $(7, 3)$, and $(6, 4)$ are possible, and if all linear and nonlinear constraints are used, the pairs $(7, 2)$ and $(6, 3)$ are possible. In contrast to these minimal point/view combination approaches, we are more interested in this thesis in using a large number of points and images for robust orientation and photogrammetry, and the efficient implementation of algorithms utilization this data.

1.2 Overview of the Thesis

The purpose of this thesis is to develop a framework for custom computing optimization of orientation and photogrammetry systems. To achieve this, we apply DSP, estimation, and novel hardware design and optimization techniques to these motion vision problems. Applying these ideas to motion vision problems will allow a reduction in computations and the required accuracy.

To demonstrate the usefulness of this framework, the mapping of orientation algorithms into modern digital logic devices will be explored. Recent advances in cost and density of FPGAs make feasible high performance custom computing solutions to a variety of signal and image processing problems. A systematic approach to understanding and applying optimization techniques will enable the rapid and affordable development of computer vision systems, able to take advantage of the current revolution in the paradigm of custom computing.

Here we briefly outline the remainder of the thesis.

In the introduction of each chapter, the relationship of the chapter in the context of the thesis is discussed. Also, related and previous work is reviewed.

We begin in Chapter 2 with a review of the relevant mathematical background for the projective camera model and the perspective projection. The first goal of this chapter is to provide the mathematical details of motion recovery from feature point matches and from image gradients. Also, the various orientation problems are reviewed, and a new, efficient approach to exterior orientation is introduced.

In Chapter 3 we discuss several exciting new techniques for optimizing custom hardware designs. These techniques are crucial to the successful implementation of real-time orientation and photogrammetry systems. A review of real-time technology is first given.

In order to reduce hardware requirements, one must round or truncate data at various points throughout an algorithm. Many operations such as multiplication or accumulation can greatly expand the wordlength of the data. If these operations are cascaded, then without some form of wordlength reduction, excessive hardware growth results. Wordlength reduction introduces noise into the data stream, so the designer must balance the need for an

efficient implementation with output quality. This chapter explores systematic approaches for optimal wordlength allocation for orientation and photogrammetry.

To achieve this, a new hardware optimization technique based on the Markov Chain Monte Carlo (MCMC) method is introduced and explored. Also in this chapter is a usable new iteration that bounds the achievable cost and quality for a particular algorithm. In many cases, this algorithm can be used to initialize the MCMC approach.

Following these general purpose optimization techniques, two new specialized optimization techniques are introduced and explored. The first method concerns the reduction of the complexity of the addition operation. The second method discusses rearranging the computations in an algorithm to minimize the number of expensive multiplications. Because most algorithms can be decomposed into multiplications and additions, both of these operations are critical to mapping an orientation algorithm to real-time hardware.

While Chapter 3 is more concerned with gaining speed at the possible expense of accuracy, Chapter 4 investigates the opposite approach. In Chapter 4, we review estimation theory and focus on techniques for constrained estimation. A new Constrained Maximum Likelihood method is introduced, and its relationship to orientation is explored. These concepts are then applied to the problem of motion estimation from image gradients. A new, custom FPGA design is introduced and analyzed using the techniques from the previous chapters to implement this high computational burden in real-time.

In Chapter 5 we explore the topic of orientation and photogrammetry using multiple (more than two) data sets. This topic is currently receiving a great deal of interest due to the possibility of obtaining more accurate scene reconstructions than are possible with only two data sets. Recent work has attempted to cast photogrammetry and other orientation problems as an absolute orientation problem, subject to non-isotropic noise. This chapter explores the unification of multiview orientation data using error ellipses into a common processing framework, so that a suitable custom computing architecture can be formulated. Both of these approaches offer high accuracy, at the cost of a slow, iterative solution method. In constrast to these approaches, we next offer a fast, linear approach to the multiple view problem that appears to work well in practice. An FPGA architecture that accelerates this approach is also given.

Finally, Chapter 6 provides a summary of the thesis, as well as possible future research directions.

Chapter 2

Mathematical Background

2.1 Introduction and Chapter Overview

In this chapter, we develop the mathematical background necessary for structure, motion, and orientation calculations. We review the relevant known results and methods, and additionally provide new error analyses, and even some new algorithms.

This chapter is organized as follows. In Section 2.2 we review the process of image formation via the perspective projection. Two images leads to the concept of epipolar geometry, which plays an important role in the development of structure and motion recovery algorithms. Methods that use matching feature points locations in the images in conjunction with rigid motion equations are reviewed.

Section 2.3 reviews an alternative approach, based not on explicit feature matches but rather on image brightness gradient measurements. These so-called "direct" methods can be thought of as the limiting case of a two-view formulation, with the time (or motion) between view approaching zero. These methods are popular because the problematic feature extraction and matching step is eliminated. In the past, acceptance of these direct methods has been hindered because of the computational load required to support real-time operation. Although the required operations are simple, every pixel is processed. Also, the gradients that are required must be calculated numerically from noisy data, a notoriously risky proposition (although there has been recent work in optimal differentiating filters [50] and in totally avoiding the gradient calculation altogether [74]). Another problem is that the small (infinitesimal) motion assumption translates into a high frame rate requirement, further increasing the computational requirements. However, it is exactly in this situation, a large number of simple operation, that custom computing approaches excel.

Section 2.4 next explores the use of multiple (greater than two) camera views. Researchers have recently been intensively studying the multiple view problem with the hope of overcoming the known deficiencies of using only two views. In this section, Stein's tensorial approach is explored.

The next section, Section 2.5, treats a different orientation problem, that of absolute orientation. In some sense, absolute orientation is seemingly unrelated to the problem of determining camera position and attitude from images. Nonetheless, absolute orientation is extremely useful in formulating multiple view orientation problems.

The next section, Section 2.6 covers exterior orientation, which can be viewed as a mixture of relative and absolute orientation. Because of this, exterior orientation is perhaps the most interesting orientation problem, because all the other ones might be viewed as limiting cases. Traditionally, exterior orientation refers to determining camera position from one image and one set of known three-dimensional landmarks. The ability to perform this calculation in real-time is important for robotic hand-eye calibration, mobile robots

in known environments, and in unmanned aerial vehicles navigation. With multiple photographs, exterior orientation can be generalized into what is known as photogrammetry. The emphasis is now on accurate reconstruction of the scene. This section, reviews the basic photogrammetric approach, and covers some newer techniques. Section 2.7 covers camera calibration, which in some ways is just a generalization of exterior orientation.

2.2 Feature Based Relative Orientation

To aid in our analysis, we will adopt a viewer-based coordinate system, as shown in Figure 2-1. We assume a *perspective projection*. The *center of projection* lies at the origin, and the image plane is parallel to the xy-plane, at a principal distance of $f = 1$. We assume that the camera pixels are arranged in a grid aligned with the x and y axes. The center-to-center spacing of the pixels is Δp. We assume the environment lies in front of the xy-plane, with positive Z values, so that the optical axis is pointing in the same direction as $\hat{\mathbf{z}}$, the unit vector in the direction $(0, 0, 1)^T$.

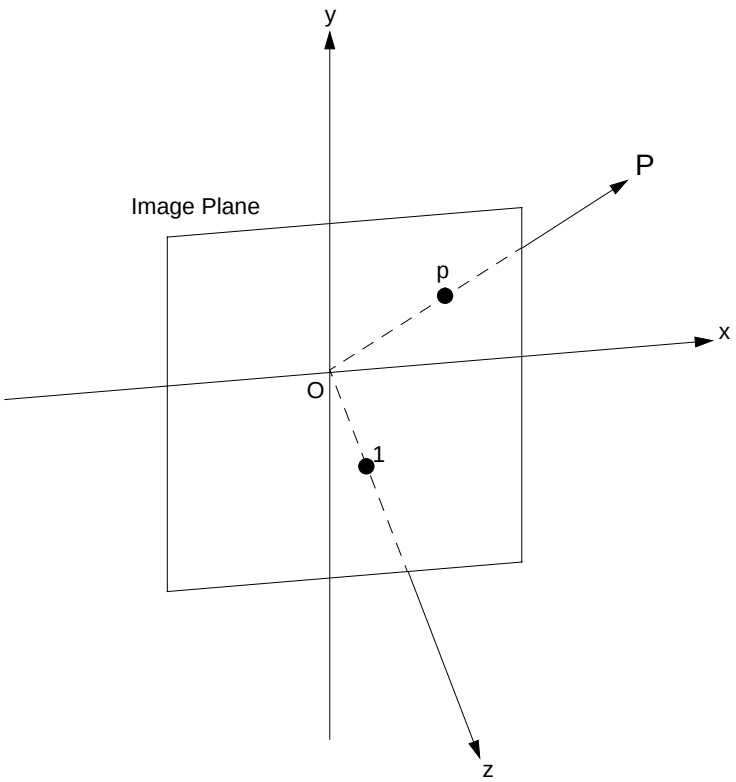

Figure 2-1: Viewer-based coordinate system used for perspective projection.

A point in the environment

$$\mathbf{P} = [X \; Y \; Z]^T \tag{2.1}$$

has a projection onto the image plane

$$\mathbf{p} = [x \; y \; 1]^T. \tag{2.2}$$

These points are related by the perspective projection equation [88]

$$\mathbf{p} = \frac{\mathbf{P}}{\mathbf{P} \cdot \hat{\mathbf{z}}}, \tag{2.3}$$

where $\hat{\mathbf{z}}$ is the unit vector along the positive z-axis. The line connecting $\mathbf{p}$ and $\mathbf{P}$ passes through the origin, and thus perspective projection represents an imaging model equivalent to an ideal pinhole. Rewriting (2.3) in terms of the coordinates,

$$x = \frac{X}{Z}, \quad y = \frac{Y}{Z}, \tag{2.4}$$

which demonstrates the main complicating factor in perspective geometry: the observed data is nonlinear in the quantities of interest.

The *field of view* (FOV) of the system is the maximum angle θ_v that a point $\mathbf{P}$ can make with the optical axis and still have a projection onto the pixels in the image plane. Since the camera pixels are typically arranged in a square, the angles that the boundary pixels make with the optical axis are not identical. We shall assume that the field of view is the cone of angles made by the circle inscribed inside the square of pixels.

Assume now that we have two images taken with the camera at two different positions, as shown in Figure 2-2. Each camera identifies a 3D reference frame, whose origin is the center of projection, and whose z-axis is the optical axis of the camera. This type of arrangement is known as an *epipolar geometry*. The points at which the line through the centers of projection intersects the image planes are known as *epipoles*. By construction, the left epipole $\mathbf{e}_l$ is the image of the projection center of the right camera, and vice versa.

A point $\mathbf{P}$ in the environment is labeled $\mathbf{P}_l = [X_l \ Y_l \ Z_l]^T$ in the left camera coordinate system and $\mathbf{P}_r = [X_r \ Y_r \ Z_r]^T$ in the right camera coordinate system. The vectors $\mathbf{p}_l = [x_l \ y_l \ 1]^T$ and $\mathbf{p}_r = [x_r \ y_r \ 1]^T$ refer to the projections of $\mathbf{P}$ onto the left and right image planes, respectively. If we assume that the environment is rigid, then we can always relate $\mathbf{P}_l$ and $\mathbf{P}_r$ by a rotation followed by a translation, or

$$\mathbf{P}_r = \mathbf{R}\mathbf{P}_l + \mathbf{t}, \tag{2.5}$$

where $\mathbf{R}$ is a 3×3 orthonormal rotation matrix and $\mathbf{t}$ is a translation vector. Estimating $\mathbf{R}$ and $\mathbf{t}$ from sets of matching $\mathbf{P}_l$ and $\mathbf{P}_r$ pairs is the Absolute Orientation problem. Estimating the motion parameters from matching pairs of $\mathbf{p}_l$ and $\mathbf{p}_r$ is the Relative Orientation problem.

In Figure 2-2, the plane defined by $\mathbf{P}$, $\mathbf{O}_l$, and $\mathbf{O}_r$ is known as the *epipolar plane* of the point, and the intersection of this plane with an image plane is called the *epipolar line* of the point. Since all rays include the center of projection, all epipolar lines of one camera go through the epipole.

From Figure 2-2, we see that the rays $\mathbf{p}_l$, $\mathbf{p}_r$, and $\mathbf{t}$, must all be coplanar. The expression of this fact in algebraic terms, known as the *coplanarity constraint*, is central to most motion estimation formulations. Since these vectors are coplanar, the parallelepiped formed by the three vectors must have zero volume. Since the volume is equal to the triple product of the vectors [88], [174], we transform $\mathbf{p}_r$ to the coordinate system of $\mathbf{p}_l$ and write

$$[\mathbf{p}_l \mathbf{t} \mathbf{p}'_r] = 0 \tag{2.6}$$

where $[\cdot]$ is the triple product and $\mathbf{p}'_r = \mathbf{R}\mathbf{p}_r + \mathbf{t}$. $\mathbf{R}$ is a rotation matrix and $\mathbf{t}$ is the baseline translation, and $\mathbf{p}'_r$ is result of transforming $\mathbf{p}_r$ to the coordinate system of $\mathbf{p}_l$. Expanding

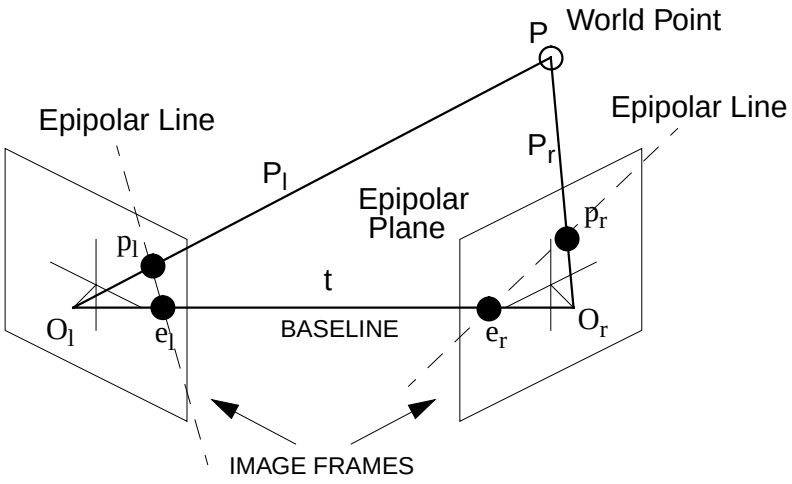

Figure 2-2: Epipolar Geometry

the triple product in (2.6) and substituting for $\mathbf{p}'_r$ gives

$$\begin{aligned} 0 &= [\mathbf{p}_l \mathbf{t} \mathbf{p}'_r] \\ &= \mathbf{p}_l \cdot (\mathbf{t} \times \mathbf{p}'_r) \\ &= \mathbf{p}_l \cdot (\mathbf{t} \times (\mathbf{R}\mathbf{p}_r + \mathbf{t})) \\ &= \mathbf{p}_l \cdot (\mathbf{t} \times \mathbf{R}\mathbf{p}_r + \mathbf{t} \times \mathbf{t})) \\ &= \mathbf{p}_l \cdot (\mathbf{t} \times \mathbf{R}\mathbf{p}_r) \\ &= \mathbf{p}_l \cdot (\mathbf{B}\mathbf{R}\mathbf{p}_r) \\ &= \mathbf{p}_l^T \mathbf{B}\mathbf{R}\mathbf{p}_r \\ &= \mathbf{p}_l^T \mathbf{E}\mathbf{p}_r, \end{aligned} \tag{2.7}$$

where we have set $\mathbf{E} = \mathbf{BR}$ and we have used the skew-symmetric representation of the cross product with $\mathbf{t}$, with

$$\mathbf{B} = \begin{bmatrix} 0 & -t_3 & t_2 \\ t_3 & 0 & -t_1 \\ -t_2 & t_1 & 0 \end{bmatrix}. \tag{2.8}$$

The matrix $\mathbf{E}$ is known as the *essential matrix*; it contains all the (unknown) motion parameters. Since $\mathbf{B}$ is rank two, $\mathbf{E}$ is also rank two. It is also known that the two nonzero singular values of $\mathbf{E}$ are equal to each other [51]. Notice that

$$\mathbf{E}^T\mathbf{t} = \mathbf{R}^T\mathbf{B}^T\mathbf{t} = \mathbf{R}^T(-\mathbf{t} \times \mathbf{t}) = 0, \tag{2.9}$$

so that $\mathbf{t}$ is perpendicular to each column of $\mathbf{E}$. Since $\mathbf{E}$ is rank two, we can thus recover the baseline translation $\mathbf{t}$ to within a scale factor by taking the cross product of any two columns of $\mathbf{E}$. With $\mathbf{t}$ known, the rotation $\mathbf{R}$ can be recovered using, for example, quaternions [91].

2.2.1 Recovering the Motion Parameters

In this section we review the known methods for solving the overdetermined set of equations

$$\mathbf{p}_{il}^T \mathbf{E} \mathbf{p}_{ir} \approx 0. \tag{2.10}$$

for all i. In (2.10), $\mathbf{p}_{il}$ is the i^{th} point in the left image, matched to the i^{th} point in the right image, $\mathbf{p}_{ir}$.

Linear Algorithms

A simple algorithm, known as the "linear" or "8-Point" algorithm, first due to Stefanovic [166] and later by Longuet-Higgins [117], ignores all nonlinear constraints on $\mathbf{E}$. Clearly, for noisy data, this will lead to extremely poor performance. Nevertheless, this approach typically is the starting point for more accurate methods.

In order to place approximately equal weight on the approximation errors in (2.10), a good heuristic, documented first by Chaudhuri and Chatterjee [30] and later elaborated by Hartley [78], is to normalize the feature point locations $\mathbf{p}_{il}$ and $\mathbf{p}_{ir}$ so that their magnitudes equal one. If this is not done, then feature points near the edge of the image dominate the error norm, so the linear algorithm tends to ignore most point matches near the center of the image, leading to poor performance. Let us denote the normalized feature points by $\tilde{\mathbf{p}}_{il} = \mathbf{p}_{il}/\|\mathbf{p}_{il}\|$ and similarly $\tilde{\mathbf{p}}_{ir} = \mathbf{p}_{ir}/\|\mathbf{p}_{ir}\|$.

We thus attempt the minimization

$$\mathbf{E}_{est} = \arg \min_{\|\mathbf{E}\|_F^2=1} \sum_{i=1}^{N} \left(\tilde{\mathbf{p}}_{il}^T \mathbf{E} \tilde{\mathbf{p}}_{ir} \right)^2 \tag{2.11}$$

where the elements of $\mathbf{E}$ are otherwise unconstrained. The normalization of $\mathbf{E}$ is necessary because (2.11) is homogeneous in the elements of $\mathbf{E}$. Writing the elements of $\mathbf{E}$ as

$$\mathbf{E} = \begin{bmatrix} e_1 & e_2 & e_3 \\ e_4 & e_5 & e_6 \\ e_7 & e_8 & e_9 \end{bmatrix}, \tag{2.12}$$

and defining $\mathbf{e} = [e_1, \ \ldots, \ e_9]^T$, we rearrange (2.11) as

$$\begin{aligned} \mathbf{e}_{est} &= \arg\min_{\|\mathbf{e}\|_2^2=1} \textstyle\sum_{i=1}^{N} \mathbf{e}^T \mathbf{a}_i \mathbf{a}_i^T \mathbf{e} \\ &= \arg\min_{\|\mathbf{e}\|_2^2=1} \mathbf{e}^T \left(\textstyle\sum_{i=1}^{N} \mathbf{a}_i \mathbf{a}_i^T \right) \mathbf{e} \\ &= \arg\min_{\|\mathbf{e}\|_2^2=1} \mathbf{e}^T \mathbf{A}^T \mathbf{A} \mathbf{e}, \end{aligned} \tag{2.13}$$

where

$$\mathbf{a}_i \stackrel{\triangle}{=} \tilde{\mathbf{p}}_{il} \otimes \tilde{\mathbf{p}}_{ir}, \quad \mathbf{A} \stackrel{\triangle}{=} \begin{bmatrix} \mathbf{a}_1^T \\ \vdots \\ \mathbf{a}_N^T \end{bmatrix}, \tag{2.14}$$

and where "$\otimes$" is the Kronecker product. The solution to (2.13) is simply the eigenvector corresponding to the minimum eigenvalue of $\mathbf{A}^T\mathbf{A}$. Alternatively, we could set $\mathbf{e}$ equal to the right singular vector associated with the minimum singular value of $\mathbf{A}$. Another alternative is to fixed $e_9 = 1$ and solve the linear equations $\mathbf{A}\mathbf{e} \approx \mathbf{0}$ in the (ordinary) least squares sense. Partitioning $\mathbf{A} = [\mathbf{A}(:, 1:8) \mid \mathbf{A}(:, 9)]$, the solution found this way is obtained as

$$\mathbf{e}(1:8)_{est} = -\mathbf{A}(:, 1:8)^{+} \mathbf{A}(:, 9). \tag{2.15}$$

With the vector $\mathbf{e}$ now obtained, we now proceed to the recovery of the baseline and rotation parameters, $\mathbf{B}$ and $\mathbf{R}$. Most authors first form $\mathbf{E}$ from $\mathbf{e}$, and then attempt to enforce some of the nonlinear constraints on $\mathbf{E}$, by zeroing the smallest singular value of $\mathbf{E}$.

This enforces the rank-2 constraint, but does nothing about enforcing constraint that $\mathbf{R}$ be a rotation matrix, or that $\mathbf{B}$ be skew-symmetric.

With $\mathbf{E}$ now formed, there are several approaches to recovering $\mathbf{B}$ and $\mathbf{R}$. Tsai and Huang [185] show that we may recover $\mathbf{B}$ and $\mathbf{R}$ from the SVD $\mathbf{USV}^T$ of $\mathbf{E}$. Their solution is that there are two solutions for the rotation:

$$\mathbf{R} = \mathbf{U} \begin{bmatrix} 0 & -1 & \\ 1 & 0 & \\ & & s \end{bmatrix} \mathbf{V}^T, \tag{2.16}$$

or

$$\mathbf{R} = \mathbf{U} \begin{bmatrix} 0 & 1 & \\ -1 & 0 & \\ & & s \end{bmatrix} \mathbf{V}^T, \tag{2.17}$$

with $s = det(\mathbf{U})det(\mathbf{V})$, and one solution for the translation (up to a scale factor)

$$\begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix} = \alpha \begin{bmatrix} E(1,:) \cdot E(2,:)/E(2,:) \cdot E(3,:) \\ E(1,:) \cdot E(2,:)/E(1,:) \cdot E(3,:) \\ 1 \end{bmatrix}. \tag{2.18}$$

Only one of the two solutions for $\mathbf{R}$, together with the appropriate sign for α, will yield positive depth values.

Horn [90] recovers the baseline and rotation from $\mathbf{E}$ in a slightly simpler fashion, without resorting to the SVD. His solution is

$$\mathbf{t}\mathbf{t}^T = \frac{1}{2}\text{tr}(\mathbf{E}\mathbf{E}^T)\mathbf{I} - \mathbf{E}^T\mathbf{E} \tag{2.19}$$

which easily gives the translation vector, and

$$(\mathbf{t} \cdot \mathbf{t})\mathbf{R} = \mathit{Cofactors}(\mathbf{E})^T - \mathbf{BE}, \tag{2.20}$$

where $\mathit{Cofactors}(\mathbf{E})$ is the matrix of cofactors of $\mathbf{E}$. Again, multiple solutions may be obtained from these equations.

Many researchers have observed that poor performance is likely to be obtained from these "linear" approaches. Hartley [78] attributes most of this to the lack of normalizing the observation vectors to unit magnitude.

A reasonably simple approach that has gained some attention recently is to use Total Least Squares (TLS) in solving for the motion parameters. Actually, setting $\mathbf{e}$ equal to the minimum singular value of $\mathbf{A}$ is a TLS approach (which does not perform very well at low SNR). Slightly better results were obtained by Diamantaras *et al.* [44] and Papadimitriou *et al.* [146], by using a small rotation angle assumption. Letting the incremental rotation angles ω_x, ω_y and ω_z be small, the input to their solution method is the set of 2D displacements $[u_i,\ v_i] = [x_{il} - x_{ir},\ y_{il} - y_{ir}]$. Using simple geometry, these displacements can be approximated as

$$\begin{aligned} u_i &= (t_1 - x_i t_3)/Z_i + \omega_x x_i y_i - \omega_y(1 + x_i^2) + \omega_z y_i \\ v_i &= (t_2 - y_i t_3)/Z_i - \omega_y x_i y_i + \omega_x(1 + y_i^2) - \omega_z x_i. \end{aligned} \tag{2.21}$$

When the unknown depths Z_i are eliminated from these equations, we obtain the linear equation $\mathbf{A}\boldsymbol{\theta} = 0$, with $\boldsymbol{\theta} = [t_1,\ t_2,\ t_3,\ t_1\omega_x, t_2\omega_y,\ t_3\omega_z,\ t_1\omega_y + t_2\omega_x,\ \ t_1\omega_z + t_3\omega_x,\ \ t_2\omega_z + t_3\omega_y]^T$ and $\mathbf{A}$ a function of the observed data. TLS is used to find the optimal $\boldsymbol{\theta}$, and the translation and rotation parameters can then be recovered from $\boldsymbol{\theta}$.

Nonlinear Algorithms

While the linear and TLS approaches are straightforward and computationally efficient, their performance at lower SNRs will generally be unsatisfactory. This is because the least squares constraint is not being enforced at the most appropriate place in the equations. In fact, if all the constraints are enforced, then only 5 point pairs are required to determine orientation (Faugeras [51] provides a summary of this "5-point" algorithm). Recently, Philip [148] presented a "linear" 5-point algorithm that would appear to work correctly only with 5 points and no noise (and therefore makes the algorithm of theoretical interest only).

With these limitations in mind, several researchers have proposed exact solutions to the motion problem. These algorithms are iterative in nature, and are essentially steepest descent minimizations (although other authors [98] attempt to solve the multivariate polynomial system symbolically).

Horn [91] provides a unit quaternion based approach (see Appendix B for details on quaternion algebra). He rewrites the coplanarity constraint in terms of quaternions. Minimizing the squares of the volumes that result from deviations from true coplanarity, the minimization problem becomes

$$\overset{\circ}{\mathbf{d}}_{est}, \overset{\circ}{\mathbf{q}}_{est} = \arg\min_{\overset{\circ}{\mathbf{d}}, \overset{\circ}{\mathbf{q}}} \sum_{i=1}^{N} w_i \left(\overset{\circ}{\mathbf{p}}_{ri} \overset{\circ}{\mathbf{d}} \cdot \overset{\circ}{\mathbf{q}} \overset{\circ}{\mathbf{p}}_{li} \right)^2, \tag{2.22}$$

where $\overset{\circ}{\mathbf{d}}$ and $\overset{\circ}{\mathbf{q}}$ are unit quaternions, $\overset{\circ}{\mathbf{p}}_{ri}$ is a quaternion version of the image measurement ($\overset{\circ}{\mathbf{p}}_{ri} = (0, \mathbf{p}_{ri})$, and similarly for $\overset{\circ}{\mathbf{p}}_{li}$), and where w_i is a weighting factor. We also have the orthogonality condition $\overset{\circ}{\mathbf{d}} \cdot \overset{\circ}{\mathbf{q}} = 0$. A Newton-Raphson approach is used to perform the minimization. The unit quaternions $\overset{\circ}{\mathbf{d}}$ and $\overset{\circ}{\mathbf{q}}$ are renormalized and re-orthogonalized after each iteration. There are 11 variables in the optimization, (each unit quaternion is four variables, and the three constraints contribute three Lagrange multipliers).

Mcilrath (nee Dron) [48] provides another iterative approach. In this iteration, first the baseline estimate is held fixed, and the rotation matrix is estimated, then the rotation matrix is held fixed, and the baseline estimated. Estimation of the baseline with a fixed rotation is actually a simple problem. Substituting $\mathbf{E} = \mathbf{BR}$ into (2.11),

$$\mathbf{B}_{est} = \arg\min_{\|\mathbf{t}\|_2^2 = 1} \sum_{i=1}^{N} \left(\tilde{\mathbf{p}}_{il}^T \mathbf{B} \mathbf{R} \tilde{\mathbf{p}}_{ir} \right)^2. \tag{2.23}$$

With $\mathbf{R}$ fixed, we may rearrange this into a simple quadratic minimization

$$\mathbf{B}_{est} = \arg\min_{\|\mathbf{t}\|_2^2 = 1} \mathbf{t}^T (\mathbf{A}^T \mathbf{A}) \mathbf{t}, \tag{2.24}$$

where the elements of $\mathbf{A}$ are known functions of the data and the fixed rotation matrix $\mathbf{R}$. We find the baseline estimate simply as the eigenvector associated with the minimum eigenvalue of $\mathbf{A}^T\mathbf{A}$.

With the baseline is fixed, we must now estimate the rotation matrix. Unfortunately, no simple solution method (such as an EVD or SVD approach) is known. Mcilrath calculates an incremental adjustment $\delta\mathbf{R}$ to the current rotation matrix by determining the optimal axis/angle representation. Let ω be a vector representing the incremental rotation. Then the rotation angle is $\delta\theta = \|\omega\|$ and the axis of the rotation is given by $\hat{\omega} = \omega/\delta\theta$. Let the residual of each equation at the current estimate be denoted by r_i, and define the quantities

$\mathbf{a}_i = (\mathbf{t} \times \mathbf{p}_{li}) \times (\mathbf{R}\mathbf{p}_{ri})$. With these definitions, Mcilrath shows that the optimal update can be found from

$$\omega = \left(\sum_{i=1}^{N} \mathbf{a}_i \mathbf{a}_i^T \right)^{-1} \left(\sum_{i=1}^{N} r_i \mathbf{a}_i \right). \tag{2.25}$$

With ω for the iteration determined, the optimal matrix $\delta\mathbf{R}$ is calculated using Rodrigues formula, or from the quaternion formulation (given in Appendix B). The new $\mathbf{R}$ is then calculated, and we then repeat the main iteration by recalculating $\mathbf{B}$.

2.3 Gradient Based Relative Orientation

Movement of a camera through an environment, or movement by objects in the environment, change the projection locations of points in the environment. The instantaneous direction and magnitude of the projection point motion is termed the *motion field*. This is a vector field, confined to the image plane. Generally, we cannot observe the motion field, but rather the *optical flow*. This is the apparent motion of brightness patterns in the image plane. When the optical flow used, we generally assume that it is identical to the motion field, although it is possible that the two fields differ. Assume that a camera moves continuously through a rigid environment. The instantaneous motion of an environmental point can be described by two vectors. The first vector, $\boldsymbol{\omega}$, is instantaneous rotational velocity. The entire environment is assumed to rotate about the line through the origin in the direction of $\boldsymbol{\omega}$. The magnitude of $\boldsymbol{\omega}$ is taken to the angular rotation rate, in rads/sec. The second vector is the instantaneous translational velocity, $\mathbf{t}$. Any motion can be described by first rotating points $\mathbf{P}$ about $\boldsymbol{\omega}$, followed by translation $\mathbf{t}$. The time derivative of the location of $\mathbf{P}$ is calculated as [94], [136]

$$\mathbf{P}_t = -\boldsymbol{\omega} \times \mathbf{P} - \mathbf{t}. \tag{2.26}$$

The motion of $\mathbf{P}$ induces motion of its projection $\mathbf{p}$ (which we will also denote by $\mathbf{r} = \mathbf{p} = [x\ y\ 1]^T$) in the image plane. This derivative is

$$\mathbf{r}_t = \frac{d\mathbf{r}}{dt} = \frac{d}{dt}\left(\frac{\mathbf{P}}{\mathbf{P}\cdot\hat{\mathbf{z}}}\right) = \frac{\hat{\mathbf{z}} \times (\mathbf{P}_t \times \mathbf{r})}{\mathbf{P}\cdot\hat{\mathbf{z}}}. \tag{2.27}$$

In (2.27), we made use of the identity $\mathbf{a} \times (\mathbf{b} \times \mathbf{c}) = (\mathbf{c}\cdot\mathbf{a})\mathbf{b} - (\mathbf{a}\cdot\mathbf{b})\mathbf{c}$. Substitution of (2.3) into (2.26) and (2.26) into (2.27) gives

$$\mathbf{r}_t = -\left(\hat{\mathbf{z}} \times \left(\mathbf{r} \times \left(\mathbf{r} \times \boldsymbol{\omega} - \frac{1}{\mathbf{P}\cdot\hat{\mathbf{z}}}\mathbf{t}\right)\right)\right). \tag{2.28}$$

Next, we make use of the brightness values. If we assume a moving environmental point maintains steady brightness E, then its projection into the image plane also maintains steady brightness as it wanders about the image. This is the *brightness change constraint equation* (BCCE), which is

$$\frac{dE}{dt} = 0 \tag{2.29}$$

for any small patch of image. Expanding the total derivative in (2.29) gives [88],

$$\frac{\partial E}{\partial \mathbf{r}} \cdot \frac{d\mathbf{r}}{dt} + \frac{\partial E}{\partial t} = 0. \tag{2.30}$$

Here, $\partial E/\partial \mathbf{r} = E_{\mathbf{r}}$ is the spatial brightness gradient, $d\mathbf{r}/dt = \mathbf{r}_t$ is the point velocity, and $\partial E/\partial t = E_t$ is the brightness gradient with respect to time. Substituting $\mathbf{r}_t$ given in (2.28) into (2.30) and simplifying gives [136]:

$$E_t - (((E_{\mathbf{r}} \times \hat{\mathbf{z}}) \times \mathbf{r}) \times \mathbf{r}) \cdot \boldsymbol{\omega} + \frac{1}{\mathbf{P} \cdot \hat{\mathbf{z}}}((E_{\mathbf{r}} \times \hat{\mathbf{z}}) \times \mathbf{r}) \cdot \mathbf{t} = 0. \tag{2.31}$$

Defining $c = E_t$, $\mathbf{s} = (E_{\mathbf{r}} \times \hat{\mathbf{z}}) \times \mathbf{r}$, and $\mathbf{v} = -\mathbf{s} \times \mathbf{r}$, allows us to write (2.31) more compactly as

$$c + \mathbf{v} \cdot \boldsymbol{\omega} + \frac{1}{\mathbf{P} \cdot \hat{\mathbf{z}}}\mathbf{s} \cdot \mathbf{t} = 0. \tag{2.32}$$

This is valid for any three dimensional rigid scene.

The vectors $\mathbf{s}$ and $\mathbf{v}$ are derived solely from the image data. Clearly, from their defining equations, we have $\mathbf{s} \cdot \mathbf{r} = 0$, $\mathbf{s} \cdot \mathbf{v} = 0$, and $\mathbf{v} \cdot \mathbf{r} = 0$. These vectors may be written out explicitly as [94]

$$\mathbf{s} = \begin{bmatrix} -E_x \\ -E_y \\ xE_x + yE_y \end{bmatrix}, \tag{2.33}$$

and

$$\mathbf{v} = \begin{bmatrix} E_y + y(xE_x + yE_y) \\ -E_x - x(xE_x + yE_y) \\ yE_x - xE_y \end{bmatrix}. \tag{2.34}$$

$\mathbf{s}$ and $\mathbf{v}$ indicate the directions for which translation and rotation will give the maximal temporal brightness change.

Several special cases for solving (2.32) for $\boldsymbol{\omega}$ and $\mathbf{t}$ have been examined:

- Pure rotation (Horn and Weldon [94]),
- Pure translation (Horn and Weldon [94]),
- Planar scene (Negahdaripour and Horn [136]),
- Quadratic patches (Negahdaripour [135]).

All of these start with a least squares formulation, where we attempt to perform the minimization

$$\arg\min_{\boldsymbol{\omega},\mathbf{t},Z} \int\int \left(c + \mathbf{v} \cdot \boldsymbol{\omega} + \frac{1}{Z}\,\mathbf{s} \cdot \mathbf{t}\right)^2 \mathrm{dx}\ \mathrm{dy} \tag{2.35}$$

through choice of $\boldsymbol{\omega}$, $\mathbf{t}$, and Z.

2.3.1 Pure Rotation

In this instance $\|\mathbf{t}\| = 0$, so (2.35) becomes

$$\arg\min_{\boldsymbol{\omega},\mathbf{t},Z} \int\int (c + \mathbf{v} \cdot \boldsymbol{\omega})^2 \,\mathrm{dx}\ \mathrm{dy} \tag{2.36}$$

This is a straightforward linear least squares problem, with solution given by

$$\left[\int\int \mathbf{v}\mathbf{v}^T \mathrm{dx}\ \mathrm{dy}\right] \boldsymbol{\omega} = -\left[\int\int E_t \mathbf{v}\ \mathrm{dx}\ \mathrm{dy}\right], \tag{2.37}$$

which is just a 3×3 set of linear equations. Clearly, the ability to recover the components of $\boldsymbol{\omega}$ accurately depends on the condition number of the matrix on the left hand side of (2.37) (defined to be $\mathbf{M_1}$). Horn and Weldon [94] showed that the the condition number of the integral of $\mathbf{v}\mathbf{v}^T$ on average is

$$\text{Cond}(\mathbf{M_1}) = \frac{3/2}{\sin^2 \theta_v} - \frac{1}{2} \tag{2.38}$$

which can be very large for a small field of view. They also show that on average $\mathbf{M_1}$ is diagonal, with the magnitude of the first two diagonal elements greater than the magnitude of the third. This means that the third component of $\boldsymbol{\omega}$ will generally be estimated with worse accuracy than the other two components.

2.3.2 Pure Translation

In this instance $\|\boldsymbol{\omega}\| = 0$, and (2.35) becomes

$$\arg \min_{\boldsymbol{\omega},\mathbf{t},Z} \int\int \left(c + \frac{1}{Z}\, \mathbf{s} \cdot \mathbf{t} \right)^2 \text{dx dy} \tag{2.39}$$

If we do not know the depth Z of $\mathbf{R}$, then (2.39) is ill-posed, because we can set $Z = -\mathbf{s}\cdot\mathbf{t}/c$ at every point, thus making the integral zero. However, if we do know the depth at every point, then (2.39) again becomes a linear least squares estimation problem, with solution given by

$$\left[\int\int \frac{1}{Z^2} \mathbf{s}\mathbf{s}^T \text{dx dy} \right] \mathbf{t} = - \left[\int\int \frac{1}{Z} E_t \mathbf{s} \text{ dx dy} \right], \tag{2.40}$$

which is just a 3×3 set of linear equations. As with the pure rotation case, the condition number of the integral of $\mathbf{s}\mathbf{s}^T$ controls the accuracy with which $\mathbf{t}$ may be recovered. Horn and Weldon [94] showed that on average the matrix is diagonal, with diagonal elements 1, 1, and $\tan^2 \theta_v/2$. Conditioning is best when $\theta_v = 54.74\ldots$ degrees. When the field of view is larger than this, the z-component of $\mathbf{t}$ is recovered more accurately than the other two components. When the field of view is smaller, the third component is recovered less accurately.

When Z is not known, in [94] the authors find the $\mathbf{t}$ that minimizes the quadratic form

$$\mathbf{t}^T \left[\int\int \frac{1}{E_t^2} \mathbf{s}\mathbf{s}^T \text{dx dy} \right] \mathbf{t}. \tag{2.41}$$

The solution is well known to be the eigenvector associated with the minimum eigenvalue of the matrix in (2.41).

2.3.3 Planar Scene

For simple planar scenes

$$\mathbf{P} \cdot \mathbf{n} = 1, \tag{2.42}$$

where $\mathbf{n}$ is a vector normal to the plane. The quantity $1/\|\mathbf{n}\|$ is the distance from the origin to the plane. With the restriction of (2.42) and using (2.3), (2.32) becomes [136]:

$$c + \mathbf{v} \cdot \boldsymbol{\omega} + (\mathbf{r} \cdot \mathbf{n})(\mathbf{s} \cdot \mathbf{t}) = 0. \tag{2.43}$$

This is the brightness change constraint equation for a planar scene. The minimization of (2.35) now becomes

$$\arg\min_{\boldsymbol{\omega},\mathbf{t},\mathbf{n}} J = \int\int (c + \mathbf{v}\cdot\boldsymbol{\omega} + (\mathbf{r}\cdot\mathbf{n})(\mathbf{s}\cdot\mathbf{t}))^2 \,\mathrm{dx}\,\mathrm{dy} \tag{2.44}$$

At the minimum, all the gradients must be zero. This leads to the following nonlinear equations

$$\begin{aligned} \mathbf{0} &= \textstyle\int\int [c + \mathbf{v}\cdot\boldsymbol{\omega} + (\mathbf{r}\cdot\mathbf{n})(\mathbf{s}\cdot\mathbf{t})]\,\mathbf{v}\,\mathrm{dx}\,\mathrm{dy} \\ \mathbf{0} &= \textstyle\int\int (\mathbf{r}\cdot\mathbf{n})\,[c + \mathbf{v}\cdot\boldsymbol{\omega} + (\mathbf{r}\cdot\mathbf{n})(\mathbf{s}\cdot\mathbf{t})]\,\mathbf{s}\,\mathrm{dx}\,\mathrm{dy} \\ \mathbf{0} &= \textstyle\int\int (\mathbf{s}\cdot\mathbf{t})\,[c + \mathbf{v}\cdot\boldsymbol{\omega} + (\mathbf{r}\cdot\mathbf{n})(\mathbf{s}\cdot\mathbf{t})]\,\mathbf{r}\,\mathrm{dx}\,\mathrm{dy}. \end{aligned} \tag{2.45}$$

Negahdaripour and Horn [136] propose several solutions to this set of equations. These are iterative schemes where the unknowns are partitioned into two sets. In one iteration, the first set of unknowns is held constant, and the second set is calculated based on these values. Then the second set is held constant, and the first set is calculated. This iteration proceeds until convergence. For the translation-only case, Frumkin [65] designed a specialized time-to-collision circuit based on these equations.

Recovering Motion Parameters - Iterative Solution

For the problem in (2.45), an iterative method is attractive because it leads to simultaneous linear equations. For example, if we partition the unknowns into the two sets $\{\boldsymbol{\omega}, \mathbf{t}\}$ and $\{\mathbf{n}\}$, we can rearrange (2.45) as

$$\begin{aligned} \left[\textstyle\int\int \mathbf{v}\mathbf{v}^T \mathrm{dx}\,\mathrm{dy}\right]\boldsymbol{\omega} \;+\; \left[\textstyle\int\int (\mathbf{r}\cdot\mathbf{n})(\mathbf{v}\mathbf{s}^T) \mathrm{dx}\,\mathrm{dy}\right]\mathbf{t} &= -\textstyle\int\int c\mathbf{v}\,\mathrm{dx}\,\mathrm{dy} \\ \left[\textstyle\int\int (\mathbf{r}\cdot\mathbf{n})(\mathbf{s}\mathbf{v}^T) \mathrm{dx}\,\mathrm{dy}\right]\boldsymbol{\omega} \;+\; \left[\textstyle\int\int (\mathbf{r}\cdot\mathbf{n})^2(\mathbf{s}\mathbf{s}^T) \mathrm{dx}\,\mathrm{dy}\right]\mathbf{t} &= -\textstyle\int\int c(\mathbf{r}\cdot\mathbf{n})\mathbf{s}\,\mathrm{dx}\,\mathrm{dy} \end{aligned} \tag{2.46}$$

and

$$\left[\textstyle\int\int (\mathbf{s}\cdot\mathbf{t})^2(\mathbf{r}\mathbf{r}^T) \mathrm{dx}\,\mathrm{dy}\right]\mathbf{n} = -\textstyle\int\int [c + \mathbf{v}\cdot\boldsymbol{\omega}](\mathbf{s}\cdot\mathbf{t})\,\mathrm{dx}\,\mathrm{dy}. \tag{2.47}$$

Equations (2.46) and (2.47) are iteratively solved through the sequential solution of the linear equations

$$\begin{bmatrix} \mathbf{M_1} & \mathbf{M_2} \\ \mathbf{M_2}^T & \mathbf{M_4} \end{bmatrix} \begin{bmatrix} \boldsymbol{\omega} \\ \mathbf{t} \end{bmatrix} = - \begin{bmatrix} \mathbf{d_1} \\ \mathbf{d_2} \end{bmatrix} \tag{2.48}$$

and

$$\mathbf{N_4}\mathbf{n} = -\mathbf{g}, \tag{2.49}$$

where the definitions of $\mathbf{M_1}$, $\mathbf{M_2}$, $\mathbf{M_4}$, $\mathbf{M_4}$,$\mathbf{N_4}$, $\mathbf{d_1}$, $\mathbf{d_1}$, and $\mathbf{g}$ are clear from (2.46) and (2.47).

This solution method was simulated using a synthetic scene. Figure 2-3 shows the evolution of the $\boldsymbol{\omega}$, $\mathbf{t}$, and $\mathbf{n}$ estimates after every evolution of (2.48) and (2.49). As seen in the figure, the rotational parameters in this case were quick to converge to the correct values. The parameters $\mathbf{t}$ and $\mathbf{n}$ took much longer to converge. In fact, the inherent scale ambiguity in $\mathbf{t}$ and $\mathbf{n}$ causes poor conditioning of the matrices during the iterations. We simply fix the last element of $\mathbf{n}$ to be one during the iterations, and rescale at the end.

The number of multiplications per iteration is shown in [136] to be 279, if certain tensors are precalculated before the iterations commence. $165N$ multiplications are required initially to form these tensors, where N is the number of pixels involved in the calculation.

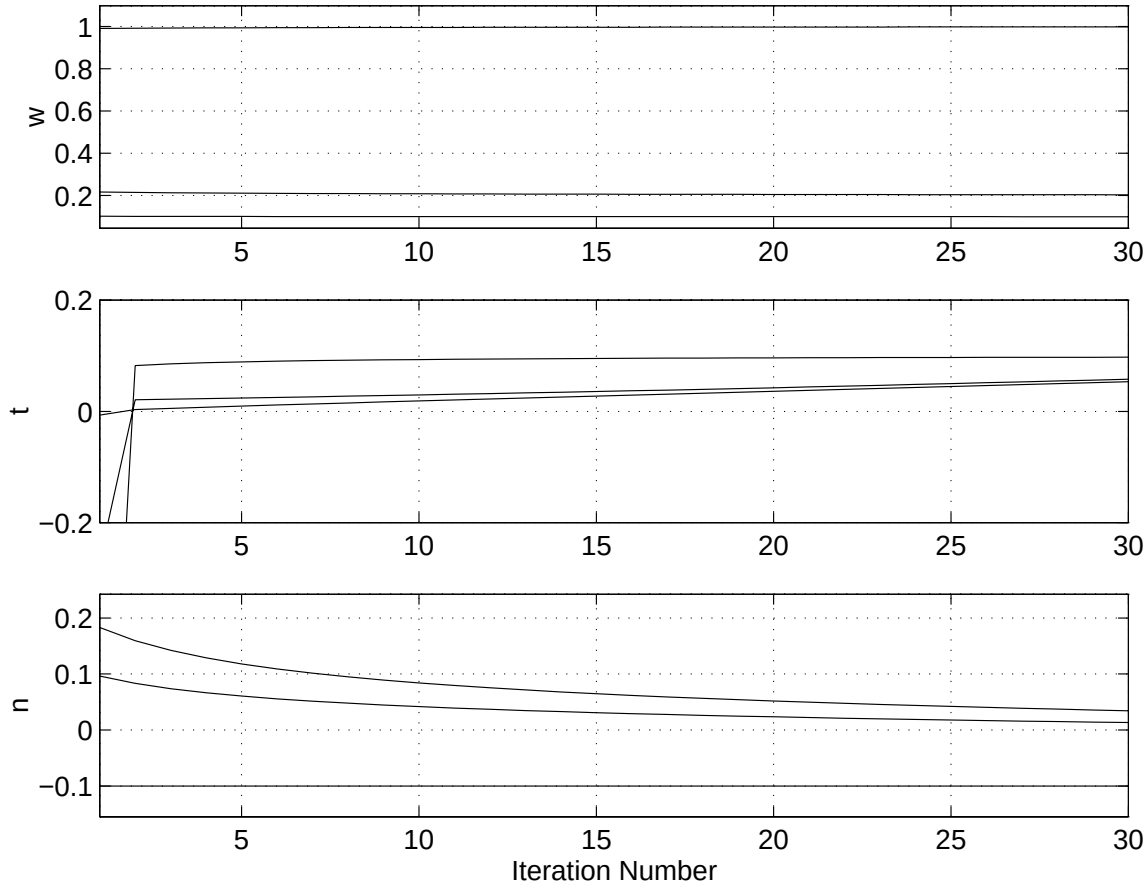

Figure 2-3: Iterative solution for motion parameters. True parameters are $\boldsymbol{\omega} = [1\ 0.2\ 0.1]^T_,$ $\mathbf{t} = [.1\ .1\ 0.1]^T_,$ and $\mathbf{n} = [0\ 0\ -0.1]^T$.

If N is large compared to the number of iterations I (i.e. if we use all the pixels in an image), then the amount of computation in the iterations is negligible. We note that a closed form solution presented below requires $44N$ multiplications for a preprocessing step.

Of interest is the question of convergence rate of the above scheme. The authors in [136] observed that in some situations very few iterations were required for convergence, while in others (namely, $\mathbf{n}$ and $\mathbf{t}$ nearly parallel), several hundred iterations were required.

This form of numerical estimation, where an estimation problem is decomposed into subsets of parameters that are sequentially updated while the remaining parameters are held fixed, has been treated by Velez-Reyes and Verghese [193]. They show that the partitioning of the Hessian matrix of the iteration determines the convergence rate. For the iteration in (2.46) and (2.47), we have partitioned the unknown parameter vector $\theta = [\mathbf{w}^T\ \mathbf{t}^T\ \mathbf{n}^T]^T$ into two sets $\theta_\alpha = [\mathbf{w}^T\ \mathbf{t}^T]^T$ and $\theta_\beta = \mathbf{n}$. The Hessian matrix of the cost function J is the

9×9 matrix

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 J}{\partial \boldsymbol{\omega} \partial \boldsymbol{\omega}} & \frac{\partial^2 J}{\partial \boldsymbol{\omega} \partial \mathbf{t}} & \frac{\partial^2 J}{\partial \boldsymbol{\omega} \partial \mathbf{n}} \\ \frac{\partial^2 J}{\partial \mathbf{t} \partial \boldsymbol{\omega}} & \frac{\partial^2 J}{\partial \mathbf{t} \partial \mathbf{t}} & \frac{\partial^2 J}{\partial \mathbf{t} \partial \mathbf{n}} \\ \frac{\partial^2 J}{\partial \mathbf{n} \partial \boldsymbol{\omega}} & \frac{\partial^2 J}{\partial \mathbf{n} \partial \mathbf{t}} & \frac{\partial^2 J}{\partial \mathbf{n} \partial \mathbf{n}} \end{bmatrix}. \tag{2.50}$$

Each entry shown in (2.50) is in fact a 3×3 matrix. These are given by:

$$\frac{\partial^2 J}{\partial \boldsymbol{\omega} \partial \boldsymbol{\omega}} = \int \int \mathbf{v}\mathbf{v}^T \mathrm{dx}\ \mathrm{dy} \tag{2.51}$$

$$\frac{\partial^2 J}{\partial \boldsymbol{\omega} \partial \mathbf{t}} = \int \int (\mathbf{r} \cdot \mathbf{n}) \mathbf{v}\mathbf{s}^T \mathrm{dx}\ \mathrm{dy} \tag{2.52}$$

$$\frac{\partial^2 J}{\partial \boldsymbol{\omega} \partial \mathbf{n}} = \int \int (\mathbf{s} \cdot \mathbf{t}) \mathbf{v}\mathbf{r}^T \mathrm{dx}\ \mathrm{dy} \tag{2.53}$$

$$\frac{\partial^2 J}{\partial \mathbf{t} \partial \boldsymbol{\omega}} = \left(\frac{\partial^2 J}{\partial \boldsymbol{\omega} \partial \mathbf{t}} \right)^T \tag{2.54}$$

$$\frac{\partial^2 J}{\partial \mathbf{t} \partial \mathbf{t}} = \int \int (\mathbf{r} \cdot \mathbf{n})^2 \mathbf{s}\mathbf{s}^T \mathrm{dx}\ \mathrm{dy} \tag{2.55}$$

$$\frac{\partial^2 J}{\partial \mathbf{t} \partial \mathbf{n}} = \int \int (\mathbf{s} \cdot \mathbf{t})(\mathbf{r} \cdot \mathbf{n}) \mathbf{s}\mathbf{r}^T \mathrm{dx}\ \mathrm{dy} \tag{2.56}$$

$$\frac{\partial^2 J}{\partial \mathbf{n} \partial \boldsymbol{\omega}} = \left(\frac{\partial^2 J}{\partial \boldsymbol{\omega} \partial \mathbf{n}} \right)^T \tag{2.57}$$

$$\frac{\partial^2 J}{\partial \mathbf{n} \partial \mathbf{t}} = \left(\frac{\partial^2 J}{\partial \mathbf{t} \partial \mathbf{n}} \right)^T \tag{2.58}$$

$$\frac{\partial^2 J}{\partial \mathbf{n} \partial \mathbf{n}} = \int \int (\mathbf{s} \cdot \mathbf{t})^2 \mathbf{r}\mathbf{r}^T \mathrm{dx}\ \mathrm{dy}. \tag{2.59}$$

The behavior of $\mathbf{H}$ near the true solution governs the convergence rate of the iteration. Because of the way we partitioned θ, we perform a similar block partitioning on $\mathbf{H}$:

$$\mathbf{H} = \begin{bmatrix} \mathbf{H}_{\alpha,\alpha} & \mathbf{H}_{\alpha,\beta} \\ \mathbf{H}_{\beta,\alpha} & \mathbf{H}_{\beta,\beta} \end{bmatrix}. \tag{2.60}$$

where $\mathbf{H}_{\alpha,\alpha}$ is a 6×6 matrix, $\mathbf{H}_{\alpha,\beta}$ is a 6×3 matrix, $\mathbf{H}_{\beta,\alpha} = \mathbf{H}_{\alpha,\beta}^T$ and $\mathbf{H}_{\beta,\beta}$ is a 3×3 matrix, The iterative method estimates θ at iteration $k+1$ from the estimate at iteration k via a nonlinear mapping M. Fast convergence of this iterations equates to small spectral radius of the Jacobian matrix of M. The authors in [193] show that this spectral radius is given by

$$\rho = \left\| \mathbf{H}_{\beta,\beta}^{-1/2} \mathbf{H}_{\beta,\alpha} \mathbf{H}_{\alpha,\alpha}^{-1} \mathbf{H}_{\alpha,\beta} \mathbf{H}_{\beta,\beta}^{-1/2} \right\|_2^2, \tag{2.61}$$

where $\| \cdot \|_2$ is the induced 2-norm.

Because of the complexity of (2.61), it does not appear that we can get intuition concerning the convergence rate of the iteration symbolically. However, we can evaluate (2.50) numerically under different motion parameter sets. Of interest then is the "shape" of $\mathbf{H}$ at or near the true solution. This will be governed by the condition number of the matrix, which is the ratio of the maximum singular value to the minimum singular value. This measures the maximum ratio of growth that two equal size vectors can experience when multiplied by $\mathbf{H}$. It is of importance here because it measures the sensitivity of the error metric (2.44) to the values of the true parameters $\boldsymbol{\omega}$, $\mathbf{t}$, and $\mathbf{n}$. A large condition number indicates that some parameters have almost no influence on the error metric, and hence can not be reliably estimated.

Figure 2-4 shows the condition number of the leading submatrices of $\mathbf{H}$ at the solution for the iteration shown in Figure 2-3. When the number of parameters equals nine, the condition number of the entire $\mathbf{H}$ given in (2.50) is shown to be quite large. This is because of the inherent scale ambiguity in $\mathbf{t}$ and $\mathbf{n}$. If we fix the last element of $\mathbf{n}$ during the iterations to one, the Hessian matrix is now 8×8, and the condition number drops approximately thirteen orders of magnitude, as shown in Figure 2-4. If we somehow knew $\mathbf{t}$ and $\mathbf{n}$, and were only required to solve for $\boldsymbol{\omega}$, then as shown in Figure 2-4, $\mathbf{H}$ would be quite well conditioned. This is borne out by the simulation shown in Figure 2-3, where the rotational parameters are seen to be estimated quickly and reliably.

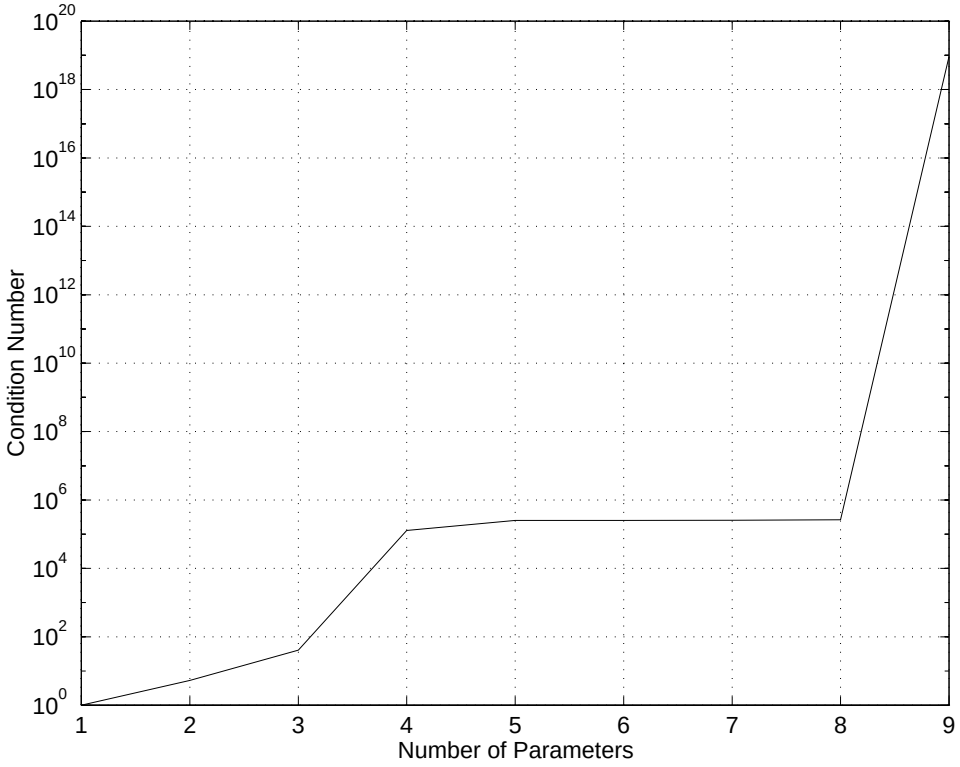

Figure 2-4: Condition number of leading submatrices of $\mathbf{H}$ at true solution $\boldsymbol{\omega} = [1\ 0.2\ 0.1]^T$, $\mathbf{t} = [.1\ .1\ 0.1]^T$ and $\mathbf{n} = [0\ 0\ -0.1]^T$.

Recovering Motion Parameters - Closed Form Solution

Negahdaripour and Horn [136] also show that the brightness change constraint equation can lead to a closed form solution. Define the skew-symmetric matrix $\mathbf{\Omega}$ as

$$\mathbf{\Omega} = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix}. \tag{2.62}$$

It is shown in [136] that (2.43) becomes

$$c + \mathbf{r}^T \mathbf{E} \mathbf{s} = 0, \tag{2.63}$$

where

$$\mathbf{E} = \begin{bmatrix} e_1 & e_2 & e_3 \\ e_4 & e_5 & e_6 \\ e_7 & e_8 & e_9 \end{bmatrix} = -\mathbf{\Omega} + \mathbf{n}\mathbf{t}^T. \tag{2.64}$$

The matrix $\mathbf{E}$ is referred to as the *essential matrix* since this matrix contains all the information about the planar motion. Since the equation is linear in the unknowns $\{e_i\}$, the data c, $\mathbf{r}$, and $\mathbf{s}$ from several points can be used to solve for the components of $\mathbf{E}$.

There are nine elements in $\mathbf{E}$, but only eight degrees of freedom since we can replace $\mathbf{n}$ and $\mathbf{t}$ by $k\mathbf{n}$ and $\mathbf{t}/k$, respectively. We make use of this by noting

$$\mathbf{E}^* \stackrel{\triangle}{=} \mathbf{E} + \mathbf{E}^T = \mathbf{n}\mathbf{t}^T + \mathbf{t}\mathbf{n}^T \tag{2.65}$$

is a rank-2 matrix so $\det(\mathbf{E} + \mathbf{E}^T) = 0$. Because $\mathbf{r}^T\mathbf{s} = 0$, we can alter $\mathbf{E}$ in (2.63) by any multiple of the identity matrix and still have an exact equality:

$$c + \mathbf{r}^T(\mathbf{E} + l\mathbf{I})\mathbf{s} = 0. \tag{2.66}$$

With $\mathbf{E}' \stackrel{\triangle}{=} \mathbf{E} + l\mathbf{I}$, (2.66) becomes

$$c + \mathbf{r}^T \mathbf{E}' \mathbf{s} = 0. \tag{2.67}$$

Thus, we first find any $\mathbf{E}'$ that satisfies (2.67) for all image points, and then we determine l such that $\det(\mathbf{E}^*) = 0$.

Because the unknowns in (2.67) are linear in the constraints, we can rearrange (2.67) as

$$\mathbf{a}^T \mathbf{e}' = -c, \tag{2.68}$$

where

$$\mathbf{e}' = [e'_1 \ \ldots \ e'_9]^T, \tag{2.69}$$

and

$$\mathbf{a} = \mathbf{r} \otimes \mathbf{s} = [r_1 s_1 \ r_1 s_2 \ r_1 s_3 \ r_2 s_1 \ r_2 s_2 \ r_2 s_3 \ r_3 s_1 \ r_3 s_2 \ r_3 s_3]^T. \tag{2.70}$$

Since there are only eight degrees of freedom in solving for the $\{e'_i\}$, we can set $e'_9 = 0$ without loss of generality. A minimum of eight image points with constraints of the form (2.68) are used to solve for the eight unknowns $e'_1, \ldots e'_8$, although generally one would use more points and solve for the unknowns in the least squares sense. This would provide for robustness in the presence of noise in the measurements, which is likely to occur because

of approximate gradient calculations and limited image pixel brightness accuracy. Stacking up all the $\mathbf{a}^T$ into a matrix $\mathbf{A}$ and all the c into a vector $\mathbf{c}$ gives

$$\mathbf{A}\mathbf{e}' = -\mathbf{c}. \tag{2.71}$$

This overdetermined set of linear equations is generally solved using least squares. We will analyze this approach in more detail in a later section.

Once the $\{e_i'\}$ are determined, we construct $\mathbf{E}'$ and determine its eigenvalues. Since $\det(\mathbf{E} + \mathbf{E}^T) = \det(\mathbf{E}' + \mathbf{E}'^T - 2l\mathbf{I}) = 0$, $2l$ must be an eigenvalue of $\mathbf{E}' + \mathbf{E}'^T$. In [136], the authors use the middle eigenvalue l of $\mathbf{E}'$. $\mathbf{E}$ is formed via $\mathbf{E} = \mathbf{E}' - l\mathbf{I}$.

With $\mathbf{E}$ now known, we can reconstruct the motion parameters. Let $\sigma = |\mathbf{n}||\mathbf{t}|$, the product of the magnitudes of $\mathbf{n}$ and $\mathbf{t}$, and $\tau = \hat{\mathbf{n}} \cdot \hat{\mathbf{t}}$. It is shown in [136] that the eigenvalues of $\mathbf{E}^*$ are $d_1 = \sigma(\tau - 1)$, $d_2 = 0$, and $d_3 = \sigma(\tau + 1)$, so that we may recover σ and τ via

$$\sigma = \frac{d_3 - d_1}{2}, \tag{2.72}$$

and

$$\tau = \frac{d_3 + d_1}{2\sigma}. \tag{2.73}$$

The eigenvectors of $\mathbf{E}^*$ are $\mathbf{u_1}$, $\mathbf{u_2}$, and $\mathbf{u_3}$, and in terms of these eigenvectors it is shown in [136] that

$$\begin{aligned} \hat{\mathbf{n}} &= \sqrt{\tfrac{1}{2}(1+\tau)}\mathbf{u_3} - \sqrt{\tfrac{1}{2}(1-\tau)}\mathbf{u_1}, \\ \hat{\mathbf{t}} &= \sqrt{\tfrac{1}{2}(1+\tau)}\mathbf{u_3} + \sqrt{\tfrac{1}{2}(1-\tau)}\mathbf{u_1}, \end{aligned} \tag{2.74}$$

The rotation parameters are then found via

$$\mathbf{\Omega} = \sigma\hat{\mathbf{n}}\hat{\mathbf{t}}^T - \mathbf{E}. \tag{2.75}$$

2.4 Multiple View Geometry

While it is possible to estimate motion and structure from two views, estimation performance may be improved by using more images. That is, we expect more accurate motion and reconstruction results if we can enforce rigidity constraints across many camera views, rather than just two. Figure 2-5 illustrates this concept. If we had only two views from cameras very close to each other, then we expect large uncertainty in the location of many of the world points. Many views, from a diversity of directions, will allow more accurate reconstruction.

A type of multiple view orientation problem arises when an image sequence is captured from a moving video camera. Recursive estimation techniques are the appropriate computational model for this situation. Heel [82], [83] formulates the problem as an iterated, extended Kalman filter (IEKF), with the reconstructed pixel depths as the state variables. Broida, Chandrashekhar, and Chellappa [28] and Azerbayejani, Pentland, and Jebara [12], [97] track feature points. Lee and Kay [112] use a quaternion representation to formulate a linear Kalman filter for feature point tracking. A review of other recursive techniques can be found in [4] and [165]. A subspace-tracking approach under the assumption of nonaccelerating motion is taken by Chaudhuri *et al.* [31]. In the array processing field, a related method by Tufts, Real, and Cooley [188] promises to significantly reduce computational throughput of subspace tracking with minimal degradation on performance.

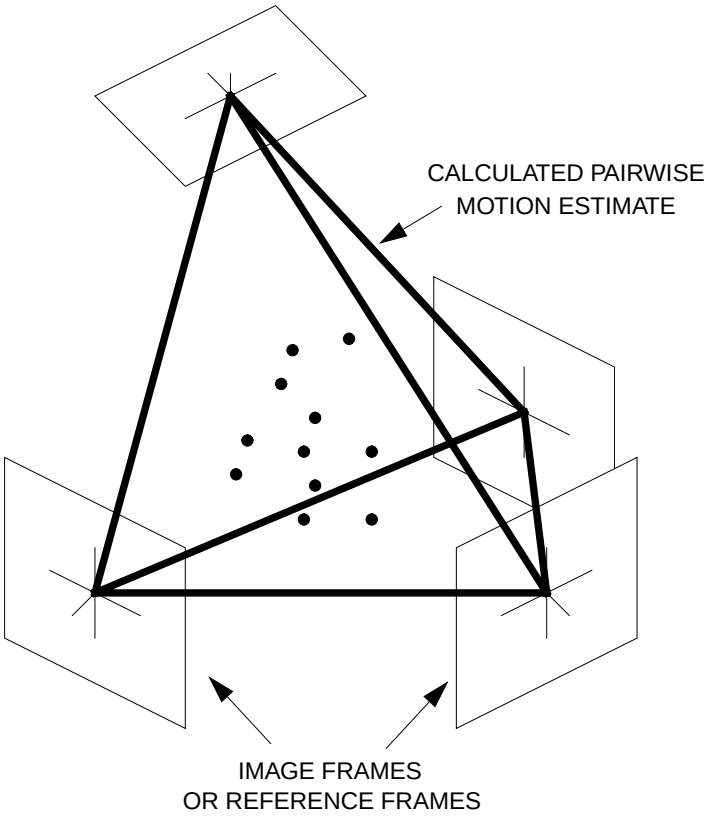

Figure 2-5: Multiple View Geometry.

Batch estimation techniques are also of interest. Iterative least squares methods (i.e. the "bundle adjustment" method) have been extensively used (see Appendix C and [67], [163]). Bergendahl [18] used three images in the design and implementation of an adaptive cruise control system. Coorg [34] uses images augmented with positional information from GPS.

Recently, a tensorial representation has been used with some success (see Shashua [159], [160], Stein [167], [168], [169], and Avidan and Shashua [11]). It will be convenient to use the *projective coordinates* to represent a world point $\mathbf{P}$:

$$\mathbf{P} = [X \; Y \; Z \; 1]^T \equiv [x \; y \; 1 \; \rho]^T, \tag{2.76}$$

with ρ equal to the inverse depth of the point. Assume we have three cameras (shown in Figure 2-6. We represent image points in the reference camera frame by $\mathbf{p}$. The corresponding points in the other camera frames are $\mathbf{p}'$ and $\mathbf{p}''$. Under perspective projection, we have

$$\begin{array}{ll} \mathbf{p} & = [\mathbf{I} \; \mathbf{0}] \, \mathbf{P}, \\ \mathbf{p}' & = [\mathbf{I} \; \mathbf{0}] \, \mathbf{P}', \\ \mathbf{p}'' & = [\mathbf{I} \; \mathbf{0}] \, \mathbf{P}''. \end{array} \tag{2.77}$$

For any line $\mathbf{s}'$ through $\mathbf{p}'$, we have $\mathbf{s}'^T \mathbf{p}' = 0$. Similarly, we also have $\mathbf{s}''^T \mathbf{p}'' = 0$.

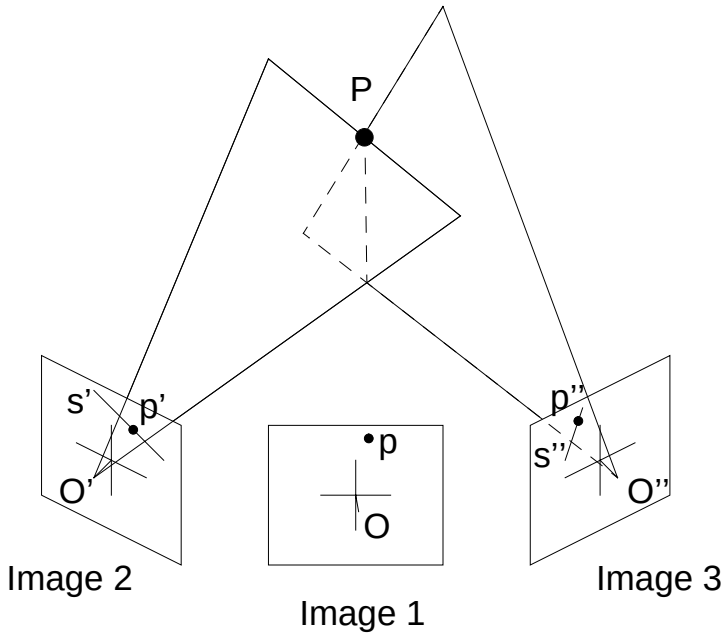

Figure 2-6: Three views. Reference frame is Image 1. $\mathbf{s}'$ is any line through $\mathbf{p}'$, and $\mathbf{s}''$ is any line through $\mathbf{p}''$.

Transferring the second camera's reference frame to the original camera,

$$\mathbf{P}' = \begin{bmatrix} \mathbf{R}' & \mathbf{t}' \\ \mathbf{0}^T & 1 \end{bmatrix} \mathbf{P}. \tag{2.78}$$

where $\mathbf{R}'$ is the rotation from the first frame to the second frame, and $\mathbf{t}'$ is the translation from the first frame to the second frame. We obtain a similar equation for the third camera frame. Combining (2.77) and (2.78),

$$\begin{aligned} \mathbf{p}' &= [\mathbf{I}\ \mathbf{0}]\,\mathbf{P}' \\ &= [\mathbf{I}\ \mathbf{0}] \begin{bmatrix} \mathbf{R}' & \mathbf{t}' \\ \mathbf{0}^T & 1 \end{bmatrix} \mathbf{P} \\ &= [\mathbf{R}'\ \mathbf{t}']\,\mathbf{P} \\ &= [\mathbf{R}'\ \mathbf{t}'] \begin{bmatrix} \mathbf{p} \\ \rho \end{bmatrix} \\ &= \mathbf{R}'\mathbf{p} + \rho\mathbf{t}'. \end{aligned} \tag{2.79}$$

Similarly, we obtain $\mathbf{p}'' = \mathbf{R}''\mathbf{p} + \rho\mathbf{t}''$. Premultiplying by $\mathbf{s}'^T$ and $\mathbf{s}''^T$ gives

$$\begin{aligned} \mathbf{p}' &= \mathbf{R}'\mathbf{p} + \rho\mathbf{t}' \\ \mathbf{s}'^T\mathbf{p}' &= \mathbf{s}'^T\mathbf{R}'\mathbf{p} + \rho\mathbf{s}'^T\mathbf{t}' = 0, \end{aligned} \tag{2.80}$$

and

$$\begin{aligned} \mathbf{p}'' &= \mathbf{R}''\mathbf{p} + \rho\mathbf{t}'' \\ \mathbf{s}''^T\mathbf{p}'' &= \mathbf{s}''^T\mathbf{R}''\mathbf{p} + \rho\mathbf{s}''^T\mathbf{t}'' = 0. \end{aligned} \tag{2.81}$$

Eliminating ρ from these equations results in

$$\mathbf{s}'^T\mathbf{t}'\mathbf{s}''^T\mathbf{R}''\mathbf{p} - \mathbf{s}''^T\mathbf{t}''\mathbf{s}'^T\mathbf{R}'\mathbf{p} = 0. \tag{2.82}$$

This can be written compactly using tensor notation as

$$\mathcal{T}_{ijk} p_i s'_j s''_k = 0, \tag{2.83}$$

where $\mathcal{T}$ is the third order tensor representing the bilinear function of the camera motion parameters:

$$\mathcal{T}_{ijk} = t'_j R''_{k,i} - t''_k R'_{j,i}. \tag{2.84}$$

In (2.83), a repeated subscript in a product indicates an implicit summation over that subscript, an operation known as *tensor contraction*. For example, writing out (2.83) in detail,

$$\sum_{i=1}^{3} \sum_{j=1}^{3} \sum_{k=1}^{3} \mathcal{T}_{ijk} p_i s'_j s''_k = 0. \tag{2.85}$$

Since (2.83 - 2.85) hold for any lines $\mathbf{s}'$ and $\mathbf{s}''$, we are free to pick convenient ones. The *standard form* is to pick lines parallel to horizontal and vertical:

$$\mathbf{s}' = \begin{bmatrix} -1 \\ 0 \\ x' \end{bmatrix} \quad or \ \mathbf{s}' = \begin{bmatrix} 0 \\ -1 \\ y' \end{bmatrix}, \tag{2.86}$$

and

$$\mathbf{s}'' = \begin{bmatrix} -1 \\ 0 \\ x'' \end{bmatrix} \quad or \ \mathbf{s}'' = \begin{bmatrix} 0 \\ -1 \\ y'' \end{bmatrix}. \tag{2.87}$$

Choosing one form from each equation in all possible ways results in four combinations. When each combination is substituted into (2.84), we obtain the four trilinear equations:

$$\begin{array}{ll} x''\mathcal{T}_{i13}p_i - x'x''\mathcal{T}_{i33}p_i + x'\mathcal{T}_{i31}p_i - \mathcal{T}_{i11}p_i & = 0 \\ y''\mathcal{T}_{i13}p_i - x'y''\mathcal{T}_{i33}p_i + x'\mathcal{T}_{i32}p_i - \mathcal{T}_{i12}p_i & = 0 \\ x''\mathcal{T}_{i23}p_i - y'x''\mathcal{T}_{i33}p_i + y'\mathcal{T}_{i31}p_i - \mathcal{T}_{i21}p_i & = 0 \\ y''\mathcal{T}_{i23}p_i - y'y''\mathcal{T}_{i33}p_i + y'\mathcal{T}_{i32}p_i - \mathcal{T}_{i22}p_i & = 0. \end{array} \tag{2.88}$$

In (2.88), note the implicit summation over the i index (i.e. $\mathcal{T}_{i13}p_i = \mathcal{T}_{113}p_1 + \mathcal{T}_{213}p_2 + \mathcal{T}_{313}p_3$). While the set of equations (2.88) may appear complicated, they are in fact *linear* in the unknowns $\mathcal{T}_{ijk}$. Thus, given enough point correspondences across three frames, we may form a set of overdetermined linear equations, and solve for the tensor of motion parameters $\mathcal{T}_{ijk}$. Since each point correspondence gives 4 equations, a minimum of 7 point correspondences across three views are required to solve for the 27 unknowns $\mathcal{T}_{ijk}$. Predictably, there are some nonlinear admissibility constraints among the 27 elements of $\mathcal{T}_{ijk}$. In an approach similar to the 8-point algorithm for two views, given noisy measurements, one would solve for $\mathcal{T}_{ijk}$ using the linear equations (2.88), and use this as a starting point for a nonlinear optimization algorithm.

2.4.1 Gradient-Based Approach

The foregoing description described a feature-based approach to three-view orientation. Stein [167] actually adapted this to a gradient-based approach. The basic idea is to use photometric constraints to develop an equation for a line in a second image corresponding to each point in a first image.

In this section, quantities associated with the second image will have a prime attached, while quantities associated with the third image will have a double-prime attached. Starting with (2.32), for the motion between the first and second image, we have

$$0 = E'_t + \mathbf{v}^T \boldsymbol{\omega}' + \rho \mathbf{s}^T \mathbf{t}', \tag{2.89}$$

where ρ is the inverse depth, and $\mathbf{s}$ and $\mathbf{v}$ are as defined in (2.33) and (2.34). Similarly, we obtain for the motion between the first and third image,

$$0 = E''_t + \mathbf{v}^T \boldsymbol{\omega}'' + \rho \mathbf{s}^T \mathbf{t}''. \tag{2.90}$$

Eliminating the unknown ρ between these two equations gives

$$E''_t \mathbf{s}^T \mathbf{t}' - E'_t \mathbf{s}^T \mathbf{t}'' + \mathbf{s}^T \left(\mathbf{t}' \boldsymbol{\omega}''^T - \mathbf{t}'' \boldsymbol{\omega}'^T \right) \mathbf{v} = 0, \tag{2.91}$$

which is known as the "15-parameter" model. We may solve for the unknowns in (2.91) through linear equations by ignoring some constraints. The 15 unknowns are $\mathbf{t}'$, $\mathbf{t}''$, and the nine elements of the matrix $\left(\mathbf{t}' \boldsymbol{\omega}''^T - \mathbf{t}'' \boldsymbol{\omega}'^T \right)$. By treating the unknown matrix elements as independent quantities (thus ignoring constraints), we obtain a linear equation for each pixel in the image.

For general motions, Stein [167] shows that the set of linear equations (2.91) has rank of at most 13. This complicates recovery of the motion parameters $\mathbf{t}'$, $\mathbf{t}''$, $\boldsymbol{\omega}'$, and $\boldsymbol{\omega}''$, but Stein provides an effective algorithm. However, in the case of collinear motion ($\mathbf{t}' \propto \mathbf{t}''$), this algorithm fails. Stein therefore mainly restricts his attention to the case of a moving stereo rig, in which case the "motions" are guaranteed to be noncollinear.

2.5 Absolute Orientation

Absolute orientation seeks to align two (or more) sets of feature points, whose three dimensional locations have been measured or reconstructed in two reference frames (see Figure 2-7). Horn [89] gave a closed-form quaternion solution, and matrix square-root [92] solution. Arun, Huang, and Blostein [10], and Umeyama [189] provide a closed-form singular value decomposition (SVD) approach. Although the previous approaches implictly assume that any location errors are assigned to one data set, Goryn and Hein [69] show that in fact that these algorithms do not need to be modified when both data sets are subject to error. Recent work [87] addresses the absolute orientation problem under nonisotropic location uncertainty. This approach can lead to a generalization of absolute orientation, to include relative and exterior orientation.

We present a very compact proof and algorithm for the exact solution of the absolute orientation problem. Using simple linear algebraic identities, we show that the product of the orthonormal matrices associated with the SVD of the data matrix is the required rotation matrix, a result first obtained by Arun *et al.* [10]. Note that there is a version of absolute orientation with an unknown scale factor s; it is useful in sequential tracking. Here, we simply treat the case where $s = 1$, because recovery of an unknown s is easy.

Let one set of three dimensional feature point locations be denoted as $\mathbf{l}_i$ and the other set as $\mathbf{r}_i$, for $i = 1, \ldots, N$. We seek the optimal translation $\mathbf{t}$ and rotation $\mathbf{Q}$ that approximately satisfies $\mathbf{l}_i = \mathbf{Q}\mathbf{r}_i + \mathbf{t}$. Summing these equations for all i and dividing by N shows that the optimal translation is found via $\mathbf{t} = \mathbf{l}_0 - \mathbf{Q}\mathbf{r}_0$, where $\mathbf{l}_0 = \sum_i \mathbf{l}_i / N$ and $\mathbf{r}_0 = \sum_i \mathbf{r}_i / N$ are the centroids of the two data sets. Combining this with our original equation gives

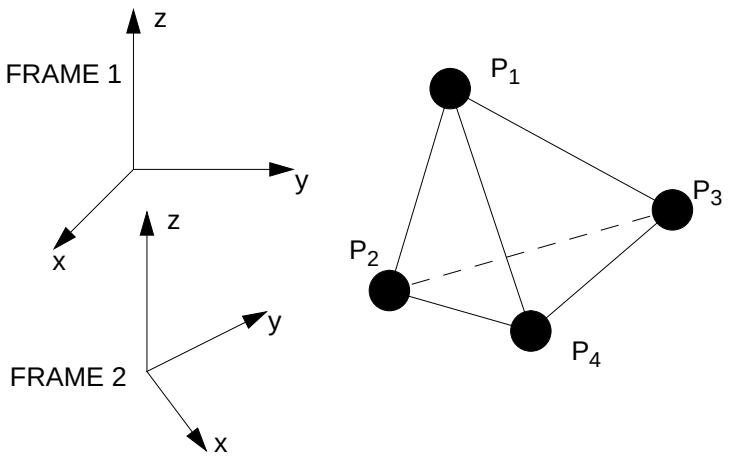

Figure 2-7: The absolute orientation problem.

$\mathbf{l}_i - \mathbf{l}_0 = \mathbf{Q}(\mathbf{r}_i - \mathbf{r}_0)$. Letting $\tilde{\mathbf{l}}_i = \mathbf{l}_i - \mathbf{l}_0$ and $\tilde{\mathbf{r}}_i = \mathbf{r}_i - \mathbf{r}_0$, we must now find the best rotation matrix $\mathbf{Q}$ that approximately satisfies $\tilde{\mathbf{l}}_i = \mathbf{Q}\tilde{\mathbf{r}}_i$, for all i. Let $\mathbf{L}$ be the $3 \times N$ matrix formed by stacking the points $\tilde{\mathbf{l}}_i$ side by side, and $\mathbf{R}$ be the matrix formed by stacking the points $\tilde{\mathbf{r}}_i$ similarly. We desire to minimize the sum of the square of the errors, or $\sum_{i=1}^{N} \|\tilde{\mathbf{l}}_i - \mathbf{Q}\tilde{\mathbf{r}}_i\|_2^2 = \|\mathbf{L} - \mathbf{QR}\|_F^2 = \|\mathbf{L}^T - \mathbf{R}^T\mathbf{Q}^T\|_F^2$, where $\|\cdot\|_F$ is the Frobenius norm. We could stop right there and claim that this exact problem is known as the "Orthogonal Procrustes Problem," and whose solution is given in the book by Golub and Van Loan [68] as

$$\mathbf{Q} = \mathbf{V}\mathbf{U}^T, \tag{2.92}$$

where $\mathbf{U}$ and $\mathbf{V}$ are derived from the singular value decomposition (SVD) $\mathbf{USV}^T$ of the matrix $\mathbf{RL}^T$. However, for completeness, we will expand upon the solution given in [68] and give a few more details.

We will use two simple facts about the trace of a matrix. The first is $\mathrm{tr}(\mathbf{A}^T\mathbf{A}) = \|\mathbf{A}\|_F^2$. The second fact is that if matrices $\mathbf{A}$ and $\mathbf{B}$ are related by a similarity transform, $\mathbf{B} = \mathbf{XAX}^{-1}$, then $\mathrm{tr}(\mathbf{A}) = \mathrm{tr}(\mathbf{B})$. Using our two trace formulas,

$$\begin{aligned} \|\mathbf{L}^T - \mathbf{R}^T\mathbf{Q}^T\|_F^2 &= \mathrm{tr}\left((\mathbf{L} - \mathbf{QR})(\mathbf{L}^T - \mathbf{R}^T\mathbf{Q}^T)\right) \\ &= \mathrm{tr}(\mathbf{LL}^T) + \mathrm{tr}(\mathbf{QRR}^T\mathbf{Q}^T) - 2\mathrm{tr}(\mathbf{QRL}^T) \\ &= \mathrm{tr}(\mathbf{LL}^T) + \mathrm{tr}(\mathbf{RR}^T) - 2\mathrm{tr}(\mathbf{QRL}^T). \end{aligned} \tag{2.93}$$

Since the first two terms on the righthand side of (2.93) do not depend on $\mathbf{Q}$, minimizing (2.93) is equivalent to maximizing $\mathrm{tr}(\mathbf{QRL}^T)$. Let $\mathbf{RL}^T = \mathbf{USV}^T$ be the singular value decomposition, with $[\mathbf{S}]_{ii} = \sigma_i \geq 0$. Now, $\mathrm{tr}(\mathbf{QRL}^T) = \mathrm{tr}(\mathbf{QUSV}^T) = \mathrm{tr}(\mathbf{V}^T\mathbf{QUSV}^T\mathbf{V}) = \mathrm{tr}(\mathbf{V}^T\mathbf{QUS})$. Letting $\mathbf{Z} = \mathbf{V}^T\mathbf{QU}$, we have $\mathrm{tr}(\mathbf{V}^T\mathbf{QUS}) = \mathrm{tr}(\mathbf{ZS}) = \sum z_{ii}\sigma_i \leq \sum \sigma_i$. Thus, to maximize the trace $\mathrm{tr}(\mathbf{V}^T\mathbf{QUS})$, we must set $z_{ii} = 1$, which means that $\mathbf{Z} = \mathbf{I}$. Therefore, $\mathbf{I} = \mathbf{V}^T\mathbf{QU}$, or $\mathbf{Q} = \mathbf{VU}^T$, thus completing the proof.

Haralick [75] derives this using a Lagrange multiplier approach. However, our derivation is much shorter and perhaps more intuitive. Also, the "B" matrix in [75] plays the role of our matrix product $\mathbf{RL}^T$ but it is not immediately obvious that this is so.

Horn *et al.* [92] solve the absolute orientation problem using a matrix square root approach. One must first multiply the data matrix by its transpose, before performing a

an eigendecomposition. Thus, the condition number of the matrix is essentially squared, pushing the matrix closer towards singularity. Also, the method must be modified in the case of a low rank data matrix. Our solution presented here avoids this condition number doubling, and needs no modification in case of a low rank matrix.

Horn [89] also presents a closed form solution using unit quaternions. This method requires the solution of a 4×4 eigenvalue problem. Our solution here requires the solution of a 3×3 SVD problem. Thus, slightly fewer operations are required, and we expect more accurate results because the SVD is inherently more stable than the eigenvector calculation.

To summarize the algorithm, first remove the centroids from the data sets, form the matrices of match points $\mathbf{L}$ and $\mathbf{R}$, form the product $\mathbf{R}\mathbf{L}^T$ and take the singular value decomposition $\mathbf{U}\mathbf{S}\mathbf{V}^T = \mathbf{R}\mathbf{L}^T$. The optimal rotation is then $\mathbf{Q} = \mathbf{V}\mathbf{U}^T$, and the optimal baseline translation is the difference in the centroids, with one of them rotated.

If the datasets are very corrupted with noise, then occasionally the $\mathbf{Q}$ produced by the algorithm will be a reflection matrix and not a rotation matrix. Umeyama [189] proves that the following simple fix is in fact optimal. The error arises whenever $det(\mathbf{U})det(\mathbf{V}) = -1$. If this occurs, simply form $\mathbf{Q}$ as

$$\mathbf{Q} = \mathbf{V} \begin{bmatrix} 1 & & \\ & 1 & \\ & & -1 \end{bmatrix} \mathbf{U}^T. \tag{2.94}$$

While it is by no means obvious, the Orthogonal Procrustes algorithm provides a maximum likelihood solution to the absolute orientation problem (assuming Gaussian perturbations to the feature point locations). We will prove this fact in Chapter 5.

Wang and Jepson [199] present a method that derives the translation vector without first solving for the rotation. They first form a weighted sum of the rigid motion equations

$$\begin{aligned} \textstyle\sum_{i=1}^N c_i \mathbf{l}_i &= \mathbf{Q} \textstyle\sum_{i=1}^N c_i \mathbf{r}_i + \mathbf{t} \textstyle\sum_{i=1}^N c_i, \\ \mathbf{L}\mathbf{c} &= \mathbf{Q}\mathbf{R}\mathbf{c} + \mathbf{t}\mathbf{1}^T\mathbf{c} \end{aligned} \tag{2.95}$$

To eliminate the rotation, we require the c_i's to satisfy $\mathbf{R}\mathbf{c} = \mathbf{0}$, or $\mathbf{c}$ must be in the null space of the data matrix $\mathbf{R}$. Since (2.95) is homogeneous in $\mathbf{c}$, we are free to choose the normalization. For simplicity, we choose this as $\mathbf{1}^T\mathbf{c} = 1$. The weight vector $\mathbf{c}$ is therefore found by combining the normalization constraint and the null space constraint, yielding

$$\mathbf{c} = \begin{bmatrix} \mathbf{R} \\ \mathbf{1}^T \end{bmatrix}^+ \begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix} \tag{2.96}$$

$\mathbf{t}$ is now recovered by

$$\mathbf{t} = \mathbf{L}\mathbf{c}. \tag{2.97}$$

With $\mathbf{t}$ now calculated, we can recover $\mathbf{Q}$ by subtracting $\mathbf{t}$ from every point in the left hand data set $\mathbf{l}_i$ and then performing either the Orthogonal Procrustes Problem, or the quaternion eigenvalue problem.

With $\mathbf{Q}$ and $\mathbf{t}$ now calculated, we may desire to calculate updated feature point locations. These refined feature point estimates would be valuable in photogrammetric applications, and also in refining structure derived from stereo reconstructions. In the development here (and in fact in the quaternion approach [89]) it would appear that all the error has to be

assigned to the $\tilde{\mathbf{l}}_i$. In situations where the $\tilde{\mathbf{r}}_i$ are noise-free, this is correct; when both sets subject to noise, we must develop an alternate approach.

Assume that the true locations are $(\mathbf{l}_i)_t$ and $(\mathbf{r}_i)_t$, and the required perturbations are $\Delta\mathbf{l}_i$ and $\Delta\mathbf{r}_i$. We have $\mathbf{l}_i = (\mathbf{l}_i)_t + \Delta\mathbf{l}_i$, and $\mathbf{r}_i = (\mathbf{r}_i)_t + \Delta\mathbf{r}_i$. Substituting these into the equation $(\mathbf{l}_i)_t = \mathbf{Q}(\mathbf{r}_i)_t + \mathbf{t}$ gives

$$\begin{aligned} \mathbf{l}_i + \Delta\mathbf{l}_i &= \mathbf{Q}(\mathbf{r}_i)_t + \mathbf{Q}\Delta\mathbf{r}_i + \mathbf{t}, \\ [\mathbf{I} \ -\mathbf{Q}] \begin{bmatrix} \Delta\mathbf{l}_i \\ \Delta\mathbf{r}_i \end{bmatrix} &= \mathbf{t} - [\mathbf{I} \ -\mathbf{Q}] \begin{bmatrix} \mathbf{l}_i \\ \mathbf{r}_i \end{bmatrix}, \\ \begin{bmatrix} \Delta\mathbf{l}_i \\ \Delta\mathbf{r}_i \end{bmatrix} &= [\mathbf{I} \ -\mathbf{Q}]^+ \left(\mathbf{t} - [\mathbf{I} \ -\mathbf{Q}] \begin{bmatrix} \mathbf{l}_i \\ \mathbf{r}_i \end{bmatrix} \right), \end{aligned} \tag{2.98}$$

where the last line gives the desired perturbations. Note that we cannot recover the perturbation components in the null space of the matrix $[\mathbf{I} \ -\mathbf{Q}]$.

2.6 Exterior Orientation

Exterior orientation [88] is the process of determining the location and attitude of a camera, given three-dimensional feature point locations and their corresponding two-dimensional image points.

While this problem is more than 150 years old, there is recent renewed interest because of automated navigation and model-based vision systems. Previous work [76], [163] focused on working with a small number of correspondences, and iterative Newton-Raphson approaches [67], [77] to solve sets of polynomial equations (which can also be solved using continuation methods [133]).

Recently, there has been great interest in the computer vision literature in methods to determine orientation that do not rely on Newton-Raphson iterations. This is mainly a result of the need for real-time computations. Iterative method often have uncertain execution times, and perhaps worse, uncertain convergence properties. Since many applications, such as mobile robotics and automotive systems have hard, real-time requirements, this type of uncertainty is generally not acceptable.

At the extreme in terms of efficiency are the "linear" methods. In these, conventional linear algebra is used to solve for unknowns. The problem with these methods is that not all geometric constraints are satisfied by a solution. At high SNR, these methods generally perform well, but at low SNR, their performance degrades rapidly. Recent examples of this approach are given by Quan and Lan [152], Triggs [183], and Ji *et al.* [99].

Lately, there has been work in noniterative, nonlinear methods. These are often touted by their practitioners as "closed-form" or "direct" methods [144]. These methods attempt to capture more of the problem constraints by formulating and solving nonlinear equations. These take the form of high order multivariate polynomial equations using spherical trigonometry [145], resultants [210], Grobner bases [7], [115], [156], group theoretic methods [100], Clifford algebras [3], [110], [111], [118], Grassman algebras, and even tensor formulations [150].

For a single photograph, the exterior orientation problem is given by

$$l_j \begin{bmatrix} x_j \\ y_j \\ f \end{bmatrix} = \mathbf{R}(\mathbf{a}_j + \mathbf{t}), \quad j = 1, \ldots, n. \tag{2.99}$$

In (2.99), the pair (x_j, y_j) denotes the camera feature point locations, $\mathbf{a}_j$ the three-dimensional feature point locations in the world coordinate frame, $\mathbf{R}$ is a rotation matrix, $\mathbf{t}$ is a translation vector (which is in fact the location of the camera center of projection in the world coordinates), and the parameters l_j are unknown scales. For multiple photographs and a single set of three-dimensional feature points, we have the classical *bundle adjustment* problem, where we must determine all the camera location parameters, as well as updated feature point locations. Appendix C contains a detailed writeup of this technique. For the remainder of this section, we examine a recent polynomial-based method, as well as a new linear approach.

2.6.1 Resultants and Exterior Orientation

Quan and Lan [152] have formulated the recovery of exterior orientation parameters in terms of *resultants* [210], which is a symbolic method of solving sets of multivariate polynomial equations. If we let $f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_0$ and $g(x) = b_m x^m + b_{m-1} x^{m-1} + \cdots + b_0$ be polynomials in a single variable x, with $a_n \neq 0$ and $b_m \neq 0$, the resultant of $f(x)$ and $g(x)$ with regard to x is the determinant of the $(m+n) \times (m+n)$ matrix

$$\mathbf{D} = \begin{bmatrix} a_n & a_{n-1} & \cdots & a_0 & & \\ & \ddots & & & \ddots & \\ & & a_n & a_{n-1} & \cdots & a_0 \\ b_m & b_{m-1} & \cdots & b_0 & & \\ & \ddots & & & \ddots & \\ & & b_m & b_{m-1} & \cdots & b_0 \end{bmatrix}, \tag{2.100}$$

where the "a" row is repeated m times and the "b" row is repeated n times.

The resultant has the following property: the determinant $\det(\mathbf{D})$ equals zero if and only if $f(x)$ and $g(x)$ have a common root. This fact can be used to solve sets of multivariate polynomial equations as follows. Let the multiple variables be represented as $\mathbf{x} = [x_1, \ldots, x_n]^T$, and assume that there is a common root to the set of equations. First, select a variable (say, x_1) to eliminate, and represent two of the equations as polynomials in this variable. The other unknowns, x_2 through x_n are treated as parts of the coefficients of the various powers of x_1 in each of the equations. Write out the resultant with regard to x_1 of the two equations symbolically. The elements of the resultant matrix will consist of constants and the other variables x_2 through x_n, but not x_1. Because the set equations are assumed to have a common root $\mathbf{x}$, there must be a common root x_1 to the two equations, and hence the determinant of the resultant must equal zero. By symbolically calculating the determinant of the resultant and setting it equal to zero, we obtain another polynomial equation, with x_1 eliminated. This new polynomial can be combined with similar ones created by the same resultant-determinant process, to eliminate another variable, say x_2.

Eventually, if there are enough equations, we can obtain a polynomial equation which contains only a single variable, say x_n. The solutions to this univariate polynomial equation

can be then found by standard methods [151]. These solutions are candidate solutions to the entire set of equations. Candidate solutions for all the variables x_1 through x_n are obtained this way, and then the correct combinations must be found through direct substitution. This process involves a finite (although probably large) number of operations. Ideally, the designer would have *a priori* knowledge with which to eliminate many of the solution combinations. Clearly, the efficacy of using resultants depends on the number of variables and equations, as well as the order of the polynomials.

For computer vision and signal processing applications, we may be interested in *approximately* solving a set of multivariate polynomial equations, rather than exactly so. This situation occurs when the equation coefficients are formed from noisy observations, so we do not expect that there is an exact, common solution. In this situation, it would appear that resultants are not directly applicable. However, by *selectively* requiring exact solutions to small subsets of the equations, we may use resultants to form sets of univariate polynomial equations, which may then be approximately solved using linear methods. This is exactly the approach taken by Quan and Lan in their solution of the exterior orientation problem.

Quan and Lan consider triples of feature points p_1, p_2, and p_3, with known three-dimensional location in the world coordinate system, and known camera image locations. Of course, both sets of observations are contaminated with some amount of noise, so that approximate solutions must be sought. Consider the triangle formed by p_2 and p_3 and the camera center of projection. In the triangle, let the length of the side from the center of projection to p_2 be denoted by x_2, and similarly for p_3 and x_3. Also, let the (known) distance between points p_2 and p_3 be denoted by d_{23}, and let the (known) angle formed by the two rays from the center of projection to p_2 and p_3 be denoted by θ_{23}. Elementary geometry gives us the relation

$$d_{23}^2 = x_2^2 + x_3^2 - 2x_2x_3c_{23}, \tag{2.101}$$

where $c_{23} \triangleq \cos\theta_{23}$. We can in fact determine c_{23} without resorting to trigonometric function evaluation by noting that the cosine of the angle formed by the rays is the inner product of the rays (normalized in length):

$$c_{23} = \frac{[u_2, v_2, 1]^T \cdot [u_3, v_3, 1]^T}{\|[u_2, v_2, 1]^T\| \|[u_3, v_3, 1]^T\|}, \tag{2.102}$$

where $[u_2, v_2, 1]^T$ are the image plane coordinates of p_2, and $[u_3, v_3, 1]^T$ are the image plane coordinates of p_3.

By considering the pair p_1 and p_3, we may obtain an equation similar to (2.101):

$$d_{13}^2 = x_1^2 + x_3^2 - 2x_1x_3c_{13}. \tag{2.103}$$

We can use resultants to eliminate x_3 from these two equations. The resultant with regard to x_3 is

$$\mathbf{D}_1 = \begin{bmatrix} 1 & -2x_1c_{13} & x_1^2 - d_{13}^2 & 0 \\ 0 & 1 & -2x_1c_{13} & x_1^2 - d_{13}^2 \\ 1 & -2x_2c_{23} & x_2^2 - d_{23}^2 & 0 \\ 0 & 1 & -2x_2c_{23} & x_2^2 - d_{23}^2 \end{bmatrix}. \tag{2.104}$$

Using a symbolic algebra program, the determinant of $\mathbf{D}_1$ is calculated and set equal to zero, resulting in the equation

$$x_2^4 + (a_1x_1)\,x_2^3 + \left(a_2x_1^2 + a_3\right)x_2^2 + \left(a_4x_1^3 + a_5x_1\right)x_2 + \left(x_1^4 + a_6x_1^2 + a_7\right) = 0, \tag{2.105}$$

where we have collected terms for a polynomial in x_2. The coefficients $a_1, \ldots, a_7$ are given by

$$\begin{aligned} a_1 &= -4c_{23}c_{13}, \\ a_2 &= 4c_{23}^2 + 4c_{13}^2 - 2, \\ a_3 &= -4c_{23}^2 d_{13}^2 + 2d_{13}^2 - 2d_{23}^2, \\ a_4 &= -4c_{13}c_{23}, \\ a_5 &= 4c_{13}c_{23}d_{13}^2 + 4c_{23}c_{13}d_{23}^2, \\ a_6 &= -2d_{13}^2 + 2d_{23}^2 - 4c_{13}^2 d_{23}^2, \\ a_7 &= -2d_{13}^2 d_{23}^2 + d_{13}^4 + d_{23}^4. \end{aligned} \tag{2.106}$$

We may now combine (2.105) and the equation

$$d_{12}^2 = x_1^2 + x_2^2 - 2x_1 x_2 c_{12} \tag{2.107}$$

to eliminate x_2. The resultant is

$$\mathbf{D}_2 = \begin{bmatrix} 1 & a_1 x_1 & a_2 x_1^2 + a_3 & a_4 x_1^3 + a_5 x_1 & x_1^4 + a_6 x_1^2 + a_7 & 0 \\ 0 & 1 & a_1 x_1 & a_2 x_1^2 + a_3 & a_4 x_1^3 + a_5 x_1 & x_1^4 + a_6 x_1^2 + a_7 \\ 1 & -2x_1 c_{12} & x_1^2 - d_{12}^2 & 0 & 0 & 0 \\ 0 & 1 & -2x_1 c_{12} & x_1^2 - d_{12}^2 & 0 & 0 \\ 0 & 0 & 1 & -2x_1 c_{12} & x_1^2 - d_{12}^2 & 0 \\ 0 & 0 & 0 & 1 & -2x_1 c_{12} & x_1^2 - d_{12}^2 \end{bmatrix}. \tag{2.108}$$

The determinant of $\mathbf{D}_2$ is calculated and set equal to zero, resulting in the equation

$$b_8 x_1^8 + b_6 x_1^6 + b_4 x_1^4 + b_2 x_1^2 + b_0 = 0. \tag{2.109}$$

The coefficients b_1, b_3, b_5, and b_7 are all exactly zero. The remaining even coefficients are incredibly complicated expressions of the a_i's, c_{12}, and d_{12} (i.e. the measurements). See Appendix D for the expressions.

We may actually forego this last symbolic determinant evaluation and calculate the coefficients b_0, b_2, b_4, b_6, and b_8 numerically. We simply plug in at least five different values of x_1 into (2.108), numerically evaluate the determinant for these values, and fit a polynomial of the form shown on the left-hand side of (2.109). This results in a Vandermonde system of equations, which can be solved in the least squares sense for the coefficients.

Quan and Lan then consider another triplet containing p_1, say (p_1, p_2, p_4), obtaining another equation of the form (2.109). By consider all such triplets containing p_1, we obtain a set of $M = (N-1)(N-2)/2$ quartic polynomial equations in x_1^2, where N is the number of feature points. Since the coefficients are corrupted by measurement noise, we attempt to find an x_1 such that

$$\begin{bmatrix} b_0^{(1)} & b_2^{(1)} & b_4^{(1)} & b_6^{(1)} & b_8^{(1)} \\ b_0^{(2)} & b_2^{(2)} & b_4^{(2)} & b_6^{(2)} & b_8^{(2)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ b_0^{(M)} & b_2^{(M)} & b_4^{(M)} & b_6^{(M)} & b_8^{(M)} \end{bmatrix} \begin{bmatrix} 1 \\ x_1^2 \\ x_1^4 \\ x_1^6 \\ x_1^8 \end{bmatrix} \approx \mathbf{0}, \tag{2.110}$$

where $b_i^{(j)}$ indicates the b_i obtained by considering the j^{th} triplet containing p_1. Quan and Lan resort to the SVD, by calculating the right singular vector $\mathbf{v}$ corresponding to the minimum singular value. They then set x_1^2 equal to the ratio of two consecutive entries

of $\mathbf{v}$, say $x_1^2 = v_2/v_1$. Clearly, this approach will suffer at high noise levels because the elements of $\mathbf{v}$ are not guaranteed to follow a geometric progression, as would be required by the exact solution.

Several variations on this approach are obvious. For example, we could average the ratios of consecutive elements of $\mathbf{v}$. Also, denoting the matrix in (2.110) by $\mathbf{A}$, we could find the optimal x_1 by performing the minimization

$$\arg\min_{x_1} \left[1 \; x_1^2 \; x_1^4 \; x_1^6 \; x_1^8\right] \mathbf{A}^t\mathbf{A} \left[1 \; x_1^2 \; x_1^4 \; x_1^6 \; x_1^8\right]^T. \tag{2.111}$$

This is simply a polynomial in x_1, which may be minimized by standard one-dimensional polynomial root finding algorithms. Finally, rather than evaluating all M triples, we could randomly select a smaller set of triples. This in fact becomes imperative for increasing N, because of the large number (i.e. $O(N^5)$) of operations involved.

2.6.2 A New, Efficient, Linear Method

This section presents a new algorithm for the solution of the exterior orientation problem. Orthogonal decompositions are used to first isolate the unknown depths of feature points in the camera reference frame, which allows the problem to be reduced to an absolute orientation with scale problem. The SVD is then used to solve the absolute orientation problem.

There do not appear to be any non-iterative algorithms that enforce all nonlinear constraints on the unknown motion and length parameters. In contrast, absolute orientation has several closed-form, optimal solution methods. Wang and Jepson [199] solve the absolute orientation problem by forming linear combinations of constraint equations in such a way as to first eliminate the unknown rotation and then similarly to eliminate the unknown translation.

We use a very similar trick here, but for the exterior orientation problem. We form a set of linear combinations of the data from which the unknown translation, rotation, and scale are eliminated. This allows us to recover the point depths in the camera reference frame using orthogonal decompositions. The problem is thus reduced to an absolute orientation problem, which we solve using the basic approaches of [10] and [189].

Derivation of the New Method

Assume that we know the noise-free three-dimensional feature point locations, denoted by $\mathbf{a}_i$, and also the corresponding camera feature point locations, denoted by the pair (x_i, y_i), for $i = 1, \ldots, N$. We seek the optimal translation vector $\mathbf{t}$, rotation matrix $\mathbf{R}$, scale s, and projective parameters l_i that best satisfy

$$l_i \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = s\mathbf{R}(\mathbf{a}_i + \mathbf{t}), \quad i = 1, \ldots, N. \tag{2.112}$$

The parameters l_i and scale factor s will be unique only up to an arbitrary common scale factor. For notational ease, we define the data matrices

$$\mathbf{P} = \begin{bmatrix} \mathbf{a}_1 & \cdots & \mathbf{a}_N \\ 1 & \cdots & 1 \end{bmatrix}, \tag{2.113}$$

and

$$\mathbf{D} = \begin{bmatrix} x_1 & \cdots & x_N \\ y_1 & \cdots & y_N \end{bmatrix}. \tag{2.114}$$

We first solve for the parameters l_i by seeking multiple linear combinations of the N equations (2.112) so that the right hand sides become zero. To this end, we seek a full column rank weight matrix $\mathbf{W} = [w_{ij}]$ such that

$$\sum_{i=1}^{N} w_{ij} = 0, \tag{2.115}$$

$$\sum_{i=1}^{N} w_{ij}\mathbf{a}_i = \mathbf{0}, \tag{2.116}$$

and

$$\sum_{i=1}^{N} w_{ij}^2 = 1. \tag{2.117}$$

for all j. The weights $\mathbf{W}$ can be easily found by collecting the linear constraints (2.115) and (2.116) into the equation

$$\mathbf{PW} = \mathbf{0}. \tag{2.118}$$

The matrix $\mathbf{P}$ is of size $4 \times N$, and we assume here that it is rank-4 (and treat the rank-deficient case later in the section). Let $\mathbf{P} = \mathbf{USV}^T$ be the SVD, and let $\mathbf{V}$ be partitioned as $\mathbf{V} = [\mathbf{V}_1|\mathbf{V}_2]$. $\mathbf{V}_2$ is the $N \times N-4$ matrix of right singular vectors corresponding to the null space of $\mathbf{P}$. We see then that the columns of $\mathbf{V}_2$ are linearly independent (and, in fact, orthonormal). We may therefore use $\mathbf{W} = \mathbf{V}_2$.

We form $N-4$ linear combinations of the N equations (2.112). For the j^{th} linear combination, we use the j^{th} column of $\mathbf{W}$. We thus have

$$\begin{aligned} \textstyle\sum_{i=1}^{N} w_{ij} l_i \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} &= \textstyle\sum_{i=1}^{N} w_{ij} s\mathbf{R}(\mathbf{a}_i + \mathbf{t}) \\ &= s\mathbf{R} \textstyle\sum_{i=1}^{N} w_{ij}\mathbf{a}_i + s\mathbf{Rt} \sum_{i=1}^{N} w_{ij} \\ &= s\mathbf{R0} + s\mathbf{Rt} \cdot 0 \\ &= \mathbf{0}, \end{aligned} \tag{2.119}$$

for $j = 1, \ldots, N-4$. We stack these $N-4$ equations, and isolate the l_i into a vector $\mathbf{l} = [l_1, \ldots, l_N]^T$:

$$\begin{bmatrix} w_{1,1} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} & \cdots & w_{N,1} \begin{bmatrix} x_N \\ y_N \\ 1 \end{bmatrix} \\ \vdots & & \vdots \\ w_{1,N-4} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} & \cdots & w_{N,N-4} \begin{bmatrix} x_N \\ y_N \\ 1 \end{bmatrix} \end{bmatrix} \mathbf{l} = \mathbf{0} \tag{2.120}$$

We can reduce this equation by noting that every third row of the matrix in (2.120) is in fact a column of $\mathbf{W}$. This means that $\mathbf{l}$ is required to be in the left null space of $\mathbf{W}$, which by virtue of (3.63), is spanned by $\mathbf{P}^T$. We therefore have that

$$\mathbf{l} = \mathbf{P}^T \boldsymbol{\alpha}, \tag{2.121}$$

for an unknown 4×1 vector $\boldsymbol{\alpha}$. This allows us to rewrite the constraint (2.120) as

$$\mathbf{C}\boldsymbol{\alpha} \triangleq \begin{bmatrix} w_{1,1} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} & \dots & w_{N,1} \begin{bmatrix} x_N \\ y_N \end{bmatrix} \\ \vdots & & \vdots \\ w_{1,N-4} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} & \dots & w_{N,N-4} \begin{bmatrix} x_N \\ y_N \end{bmatrix} \end{bmatrix} \mathbf{P}^T \boldsymbol{\alpha} = \mathbf{0}. \tag{2.122}$$

The best $\boldsymbol{\alpha}$ is found from the SVD of the matrix product $\mathbf{C}$ in (2.122), by choosing the left singular vector corresponding to the minimum singular value of $\mathbf{C}$. Alternatively, we can premultiply by $\mathbf{C}^T$ and seek the eigenvector corresponding to the minimum eigenvalue of the matrix $\mathbf{C}^T\mathbf{C}$. After some manipulation, we can write this matrix compactly as

$$\mathbf{C}^T\mathbf{C} = \mathbf{P}\left(\mathbf{W}\mathbf{W}^T \odot \mathbf{D}^T\mathbf{D}\right)\mathbf{P}^T, \tag{2.123}$$

where we use the symbol "$\odot$" to denote componentwise multiplication (also known as the "Hadamard product" or "Schur product").

In (2.123), $\mathbf{W}$ appears only through the product $\mathbf{W}\mathbf{W}^T$. We can form this product more efficiently than by calculating $\mathbf{W}$ from the SVD of $\mathbf{P}$. We could calculate this product as

$$\mathbf{W}\mathbf{W}^T = \mathbf{I} - \mathbf{V}_3\mathbf{V}_3^T, \tag{2.124}$$

where $\mathbf{I}$ is the identity matrix and $\mathbf{V}_3$ is an $N \times 4$ matrix that forms an orthonormal basis for the columns of $\mathbf{P}^T$. Such a basis is easily calculated using a Gram-Schmidt or QR procedure [173], at significantly less cost than an SVD for large N.

Note that since $\mathbf{C}$ is $2(N-4) \times 4$, we will require $N \geq 6$ points for a unique minimal $\boldsymbol{\alpha}$ to be found. Since it is possible to calculate exterior orientation from 3 points [67], this indicates that some nonlinear constraints are being ignored by the algorithm.

Once $\boldsymbol{\alpha}$ is determined, the l_i are found from (2.121). With the vector $\mathbf{l}$ thus recovered (up to a scale factor) we are now in a position to apply the absolute orientation with scaling algorithm to (2.112). Denoting the left hand sides of (2.112) as

$$\mathbf{b}_i = l_i \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}, \tag{2.125}$$

(2.112) becomes

$$\mathbf{b}_i = s\mathbf{R}(\mathbf{a}_i + \mathbf{t}), \quad i = 1, \dots, N. \tag{2.126}$$

Summing these equations for all i and dividing by N shows that the optimal translation is found via

$$\mathbf{t} = s^{-1}\mathbf{R}^T\mathbf{b}_0 - \mathbf{a}_0, \tag{2.127}$$

where $\mathbf{b}_0 = \sum_i \mathbf{b}_i/N$ and $\mathbf{a}_0 = \sum_i \mathbf{a}_i/N$ are the centroids of the two data sets. Combining this with our original equation gives $\mathbf{b}_i - \mathbf{b}_0 = s\mathbf{R}(\mathbf{a}_i - \mathbf{a}_0)$. Letting $\tilde{\mathbf{b}}_i = \mathbf{b}_i - \mathbf{b}_0$ and $\tilde{\mathbf{a}}_i = \mathbf{a}_i - \mathbf{a}_0$, we must now find the best rotation matrix $\mathbf{R}$ that approximately satisfies $\tilde{\mathbf{b}}_i = s\mathbf{R}\tilde{\mathbf{a}}_i$, for all i. Because the rotation matrix does not change the length of the vectors $\tilde{\mathbf{a}}_i$, we immediately solve for the optimal least squares scale s using

$$s = \frac{\sum_i \|\tilde{\mathbf{a}}_i\| \|\tilde{\mathbf{b}_i}\|}{\sum_i \|\tilde{\mathbf{a}}_i\|^2}. \tag{2.128}$$

Let $\mathbf{B}$ be the $3 \times N$ matrix formed by stacking the points $\tilde{\mathbf{b}}_i$ side by side, and $\mathbf{A}$ be the matrix formed by stacking the scaled points $s\tilde{\mathbf{a}}_i$ similarly. We desire to minimize the sum of the square of the errors, or $\sum_{i=1}^{N} \|\tilde{\mathbf{b}}_i - s\mathbf{R}\tilde{\mathbf{a}}_i\|_2^2 = \|\mathbf{B} - \mathbf{R}\mathbf{A}\|_F^2$, where $\|\cdot\|_F$ is the Frobenius norm. This problem is known as the "Orthogonal Procrustes Problem," whose solution is given by Golub and Van Loan [68] as

$$\mathbf{R} = \mathbf{V}\mathbf{U}^T, \tag{2.129}$$

where $\mathbf{U}$ and $\mathbf{V}$ are the left and right singular vectors of the SVD $\mathbf{U}\mathbf{S}\mathbf{V}^T$ of the matrix $\mathbf{A}\mathbf{B}^T$. Actually, if the data sets are extremely noisy, it is possible that $\mathbf{R}$ obtained by (2.129) may be merely orthonormal, rather than a rotation matrix. The method of Umeyama [189] corrects this situation.

The efficient linear algorithm for exterior orientation is thus summarized as:

1. Calculate an orthonormal basis $\mathbf{V}_3$ for the columns of $\mathbf{P}^T$ using the Gram-Schmidt procedure,
2. Calculate $\mathbf{W}\mathbf{W}^T$ from (2.124),
3. Form the 4×4 matrix $\mathbf{C}^T\mathbf{C}$ in (2.123) and set $\boldsymbol{\alpha}$ equal to the eigenvector corresponding to the minimum eigenvalue,
4. Calculate the vector of parameters $\mathbf{l}$ from (2.121),
5. Calculate the centroids $\mathbf{a}_0$ and $\mathbf{b}_0$, and form the data sets $\tilde{\mathbf{a}}_i$ and $\tilde{\mathbf{b}}_i$,
6. Calculate the scale s using (2.128),
7. Form the matrices $\mathbf{A}$ and $\mathbf{B}$, and calculate the SVD of $\mathbf{A}\mathbf{B}^T$,
8. Calculate the rotation estimate using (2.129),
9. Calculate the translation estimate using (2.127).

We note also that a small cost savings can be had if multiple image frames are available for the single set of three-dimensional feature points. The calculation of $\mathbf{W}\mathbf{W}^T$ in (2.124) is independent of the image point locations and can therefore be calculated once for multiple frames. This can serve as an efficient initialization for a bundle adjustment procedure in photogrammetric applications [163].

Rank Deficient Case

When $\mathbf{P}$ has rank less than four, as was assumed in the previous section, we must slightly modify the algorithm. It is easily shown that $\mathbf{P}$ will be rank-3 if and only if all the points $\mathbf{a}_i$ are coplanar (and not all collinear). In this situation, $\mathbf{V}_1$ is now $N \times 3$. We replace $\mathbf{P}^T$ in (2.121)-(2.124) by $\mathbf{V}_1$, and solve for the now 3×1 vector $\boldsymbol{\alpha}$. Since $\mathbf{C}$ is now $2(N-3) \times 3$, $N \geq 4$ known coplanar points are necessary for a unique minimal $\boldsymbol{\alpha}$. As a practical matter, for geometries where the N points are *nearly* coplanar, the algorithm as given in the previous section will exhibit unstable performance, due to the near-repeated minimal eigenvalue of $\mathbf{C}$. Therefore, the planar algorithm given this section should be used when $\mathbf{C}$ exhibits nearly equal minimal eigenvalues.

Simulations and Performance Evaluation

In this section, we compare the new algorithm to that of [152]. Additionally, we also compare to another, very simple, linear method. Starting with (2.112), we rearrange the equation so that all the unknowns are collected into a single vector:

$$\left[\begin{array}{ccc|c|c|ccc} -x_1 & & & & & \mathbf{a}_1^T & & \\ -y_1 & & & \multicolumn{2}{c|}{\multirow{2}{*}{$\mathbf{I}$}} & & \mathbf{a}_1^T & \\ -1 & & & & & & & \mathbf{a}_1^T \\ & \ddots & & \vdots & & & \vdots & \\ & & -x_N & & & \mathbf{a}_N^T & & \\ & & -y_N & \multicolumn{2}{c|}{\mathbf{I}} & & \mathbf{a}_N^T & \\ & & -1 & & & & & \mathbf{a}_N^T \end{array}\right] \left[\begin{array}{c} l_1/s \\ \vdots \\ l_N/s \\ \mathbf{t}' \\ \mathbf{r} \end{array}\right] = \mathbf{0}. \tag{2.130}$$

where $\mathbf{t}' = \mathbf{R}\mathbf{t}$, and $\mathbf{r} = [r_1, \ldots, r_9]^T$ is a vectorized (row scanned) version of $\mathbf{R}$. We solve for the unknowns by finding the right singular vector $\mathbf{v}$ corresponding to the minimum singular value of the matrix in (2.130). Alternatively, to reduce the computational load, we may premultiply by the matrix transpose in (2.130) and set $\mathbf{v}$ equal to the eigenvector corresponding to the minimum eigenvalue. Since the matrix in (2.130) is $3N \times (N+12)$, we require $N \geq 6$ for a unique minimum. We extract the l_i/s estimates from $\mathbf{v}$, and then solve the resulting absolute orientation with scale problem with these estimates.

For the tests, a random set of two-dimensional feature points and l_i's were generated, along with a random rotation and translation (care was used so that the points were in front of the camera). The corresponding three-dimensional points were calculated. Noise was then added independently to both the two-dimensional and three-dimensional data.

The metrics that were measured are the error in the rotation matrix components (the Frobenius norm is reported), the angular error (in degrees) of the corresponding unit quaternion associated with the rotation, the relative error in magnitude of the translation vector, and the angular error (in degrees) of the translation vector.

The signal-to-noise ratio (SNR) used in setting the noise level was calculated by first measuring the average spread of the two-dimensional and three-dimensional components. This number was then used as a multiplier for a scaled Gaussian random noise generator. Monte Carlo runs were then performed, and results for constant SNR were combined by taking the median of the various errors. Figures 2-8 and 2-9 show examples of the relative feature point displacements in the image and in space corresponding to the worst SNR level used in the Monte Carlo runs.

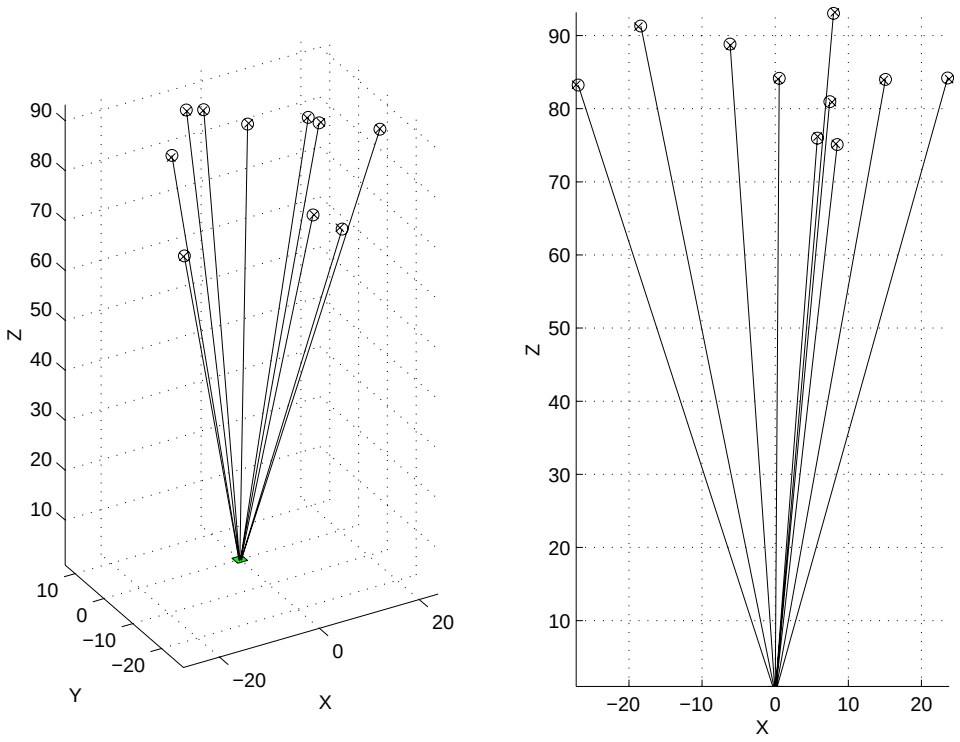

Figure 2-8: Two views of a sample three-dimensional feature point scene. Noise-free "$\circ$" and noisy "$\times$" feature point locations corresponding to 35dB SNR.

Figures 2-10 shows a comparison of the simple linear method, the method of [152], and the new method. 50 Monte Carlo runs per SNR were performed, and the median errors were plotted. Results for $N = 8$ are shown; all of the methods show improved performance with the addition of more points. Figures 2-11 and 2-12 show the performance of the new method for a more challenging scenario, where the field of view is narrower than in Figure 2-10.

Figure 2-13 shows a comparison of the number of operations required by the methods (as determined by the MATLAB "flops" command), as a function of the number of feature points. Clearly, the new method requires far fewer computations. This shows that for the method of [152] to be practical for large N, one must forego forming all triplets and select smaller (perhaps random) subsets of triplets.

It is straightforward to derive the operations counts of the methods. For the new method, the Gram-Schmidt and absolute orientation procedures requires $O(N)$ operations, and all the other steps are $O(N^2)$ in complexity (with small multiplicative constants). The method of [152] as formulated evaluates $O(N^2)$ triplets for each of N points, leading to $O(N^3)$ behavior. The simple linear method requires $O(N^2)$ operations to form the symmetric product of the matrix in (2.130), and anywhere from $O(N^2)$ to $O(N^3)$ operations to find the minimum eigenvector, depending on the method used.

Linear methods, such as the one presented here and in [152] involve a trade-off between robustness and computational cost. Because nonlinear constraints are ignored by linear algorithms, it is crucial to understand the behavior of these algorithms in the presence of feature location noise. For our algorithm, the main entry point for deviations from optimal behavior will be in the solution of the vector $\boldsymbol{\alpha}$ as the minimizer of the quadratic form

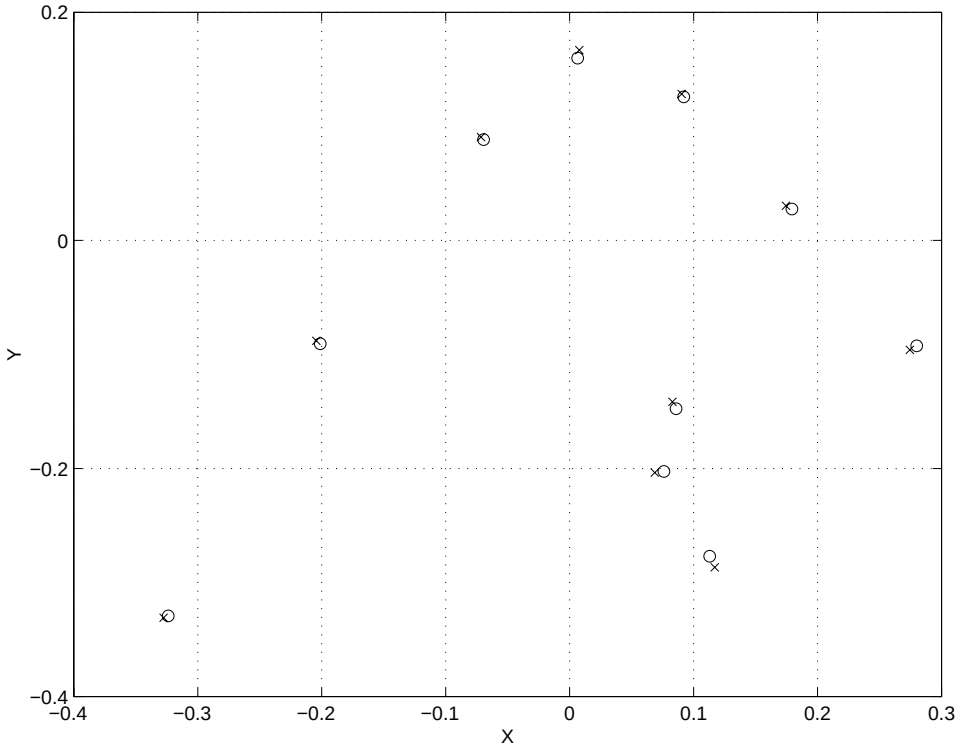

Figure 2-9: Image feature points corresponding to three-dimensional scene in Figure 2-8. The image feature points were independently perturbed; "$\circ$" indicates the true location, "$\times$" indicates the perturbed location.

$\boldsymbol{\alpha}^T\mathbf{C}^T\mathbf{C}\boldsymbol{\alpha}$. In the noise-free case, $\boldsymbol{\alpha}$ will be the unique eigenvector corresponding to the zero-eigenvalue of $\mathbf{C}^T\mathbf{C}$. The nonlinear constraints are not needed in this case because $\boldsymbol{\alpha}$ is necessarily in the null space. As noise is allowed to increase, the zero-eigenvalue increases, and the corresponding zero-eigenvector will change direction. In this situation, the nonlinear constraints are not satisfied, and more importantly, deviations from satisfying the nonlinear constraints are not penalized by any linear method.

We thus become interested in the sensitivity of eigenvectors to noise. Golub and Van Loan [68] showed that the eigenvector sensitivity is proportional to the inverse distance between the zero-eigenvalue and the other eigenvalues of the matrix. Therefore, for a matrix of constant Frobenius norm, the most stable zero-eigenvector computation is had when all other eigenvalues of the matrix are equal. Because we may arbitrarily scale the matrix, the quantity of interest is thus the ratio of the maximum eigenvalue to the second smallest eigenvalue. We loosely term this ratio the "condition" number of the matrix (as does Hartley [78] in a similar computation to analyze the stability of the "8-point algorithm" for relative orientation). Figure 2-14 shows this condition number of the matrix $\mathbf{C}^T\mathbf{C}$ as a function of the number of points N.

Figure 2-15 shows a scatter plot of the median angular error of the translation estimate as a function of the condition number. Here, the condition number is calculated without the addition of noise; it is a function of the scene geometry only. The translation angular error was calculated with a small amount of additive noise; we are in effect linearizing the sensitivity calculation about the noise free operating point. What the figure shows is that

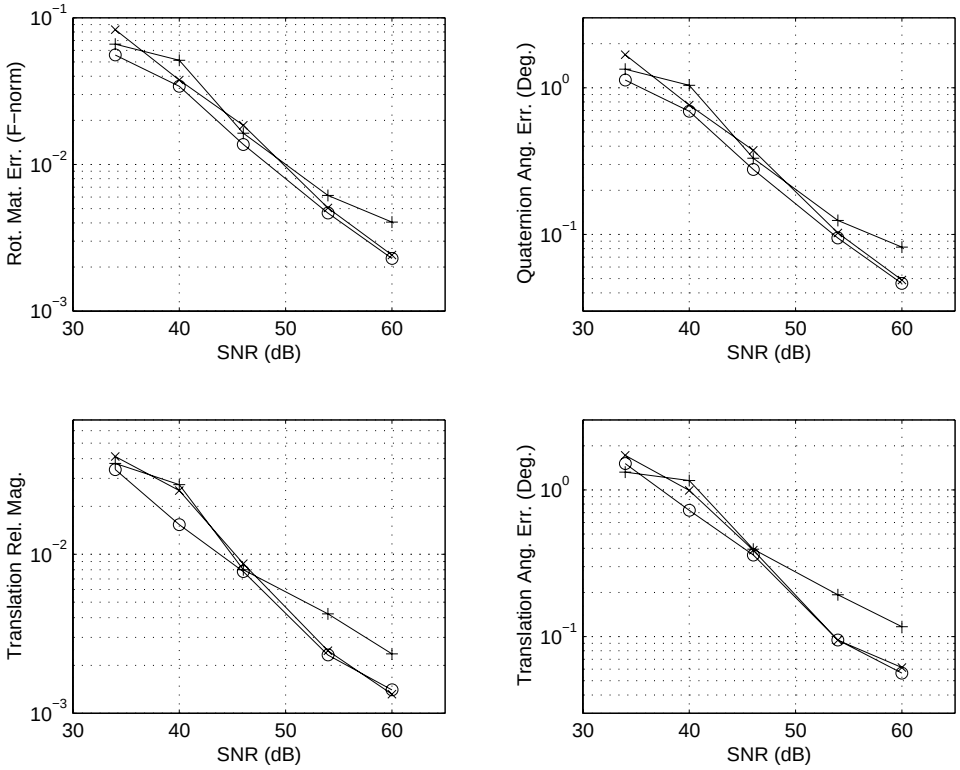

Figure 2-10: Performance comparison for method of [152] (plotted as "+"), the simple linear method (plotted as "$\circ$"), and new method (plotted as "$\times$"). Shown are results for $N = 8$ points. The average field of view was 80 degrees, and the average l_i was approximately 80, and the average scene depth was approximately 10.

the error does increase with the condition number of the scene.

Summary

In this section, we have developed a computationally efficient method for exterior orientation which uses well known numerical, linear algebraic methods. Comparisons to several existing methods showed nearly the same robustness to feature point location noise, but with a significantly reduced computational cost. The method extends to multiple image frames with some additional cost savings through shared computations. The method can be used as is, or be used to initialize a more exact, iterative procedure.

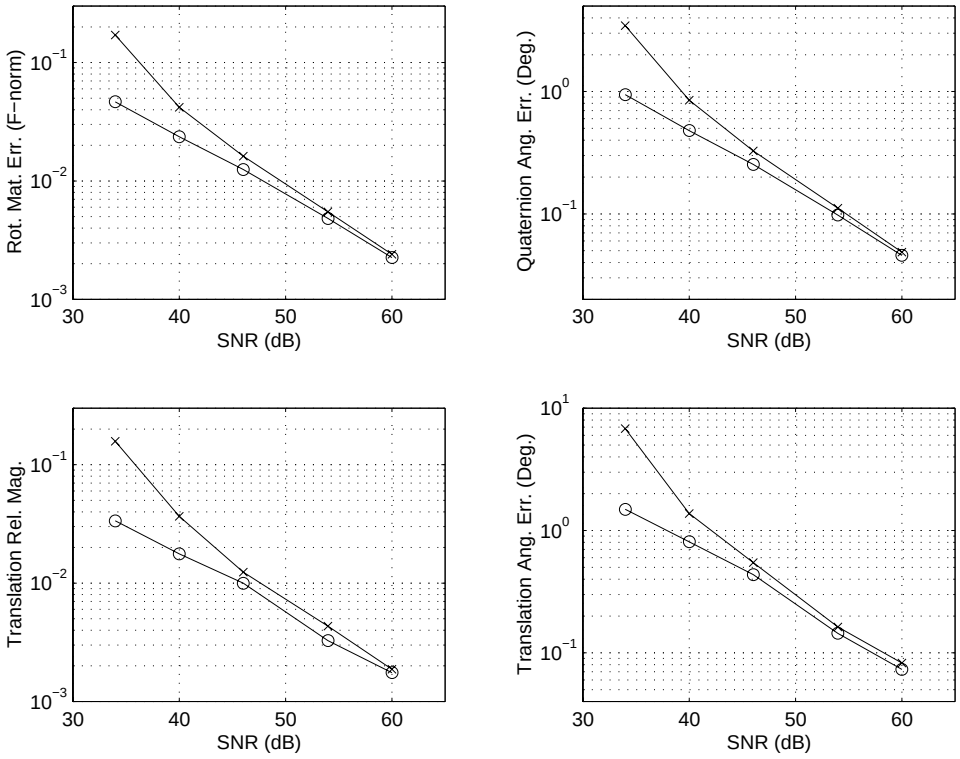

Figure 2-11: Performance comparison of the simple linear method (plotted as "$\circ$"), and new method (plotted as "$\times$"). Shown are results for $N = 20$ points. The average field of view was 45 degrees, the average l_i was approximately 150, and the average scene depth was approximately 20. 100 Monte Carlo runs per point were performed.

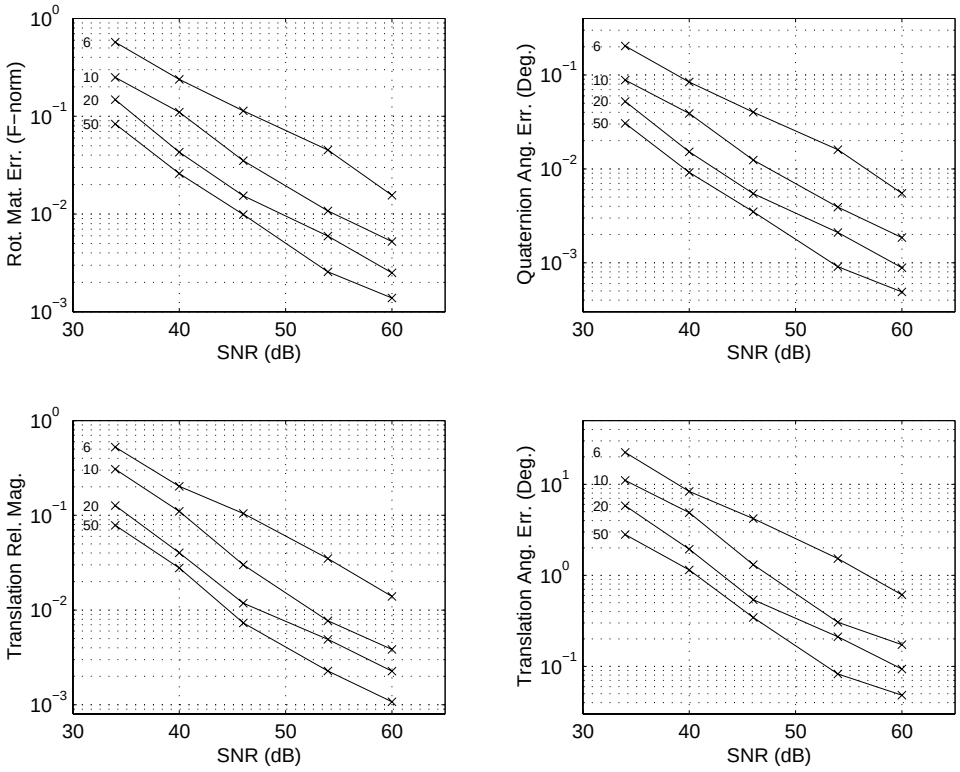

Figure 2-12: Performance of the new method. Shown are results for $N = 6$, 10, 20, and 50 points. The average field of view was 45 degrees, the average l_i was approximately 150, and the average scene depth was approximately 20. 100 Monte Carlo runs per point were performed.

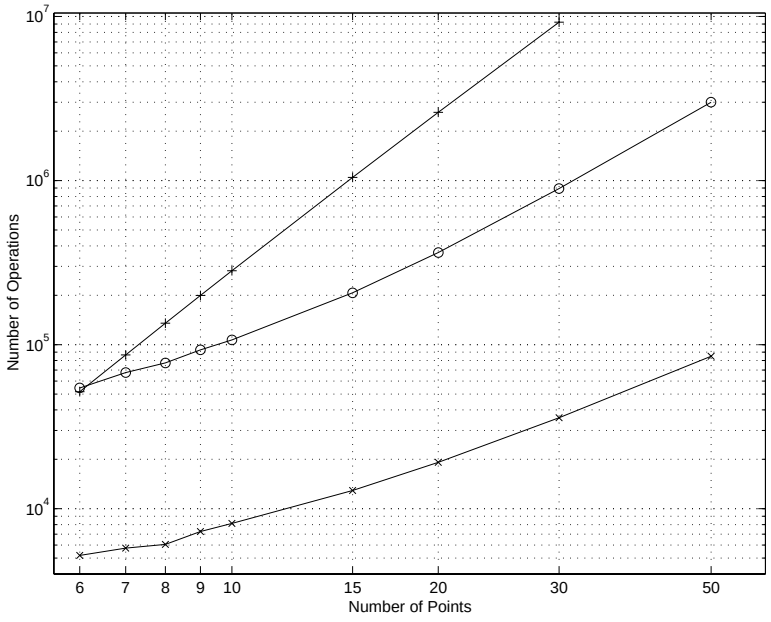

Figure 2-13: Operations count of the method of [152] (plotted as "+"), simple linear method (plotted as "∘") and new method (plotted as "×").

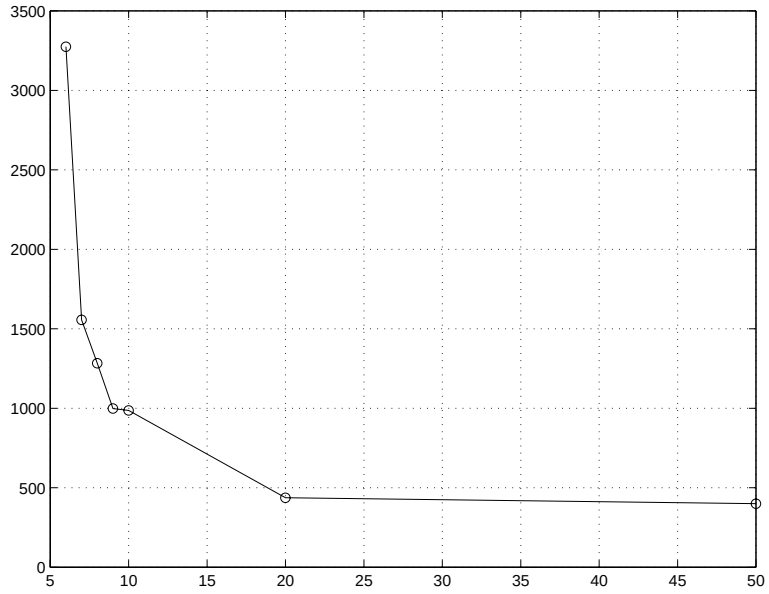

Figure 2-14: Condition number of $\mathbf{C}^T\mathbf{C}$ as a function of N (same scene generation parameters as in Figure 2-12).

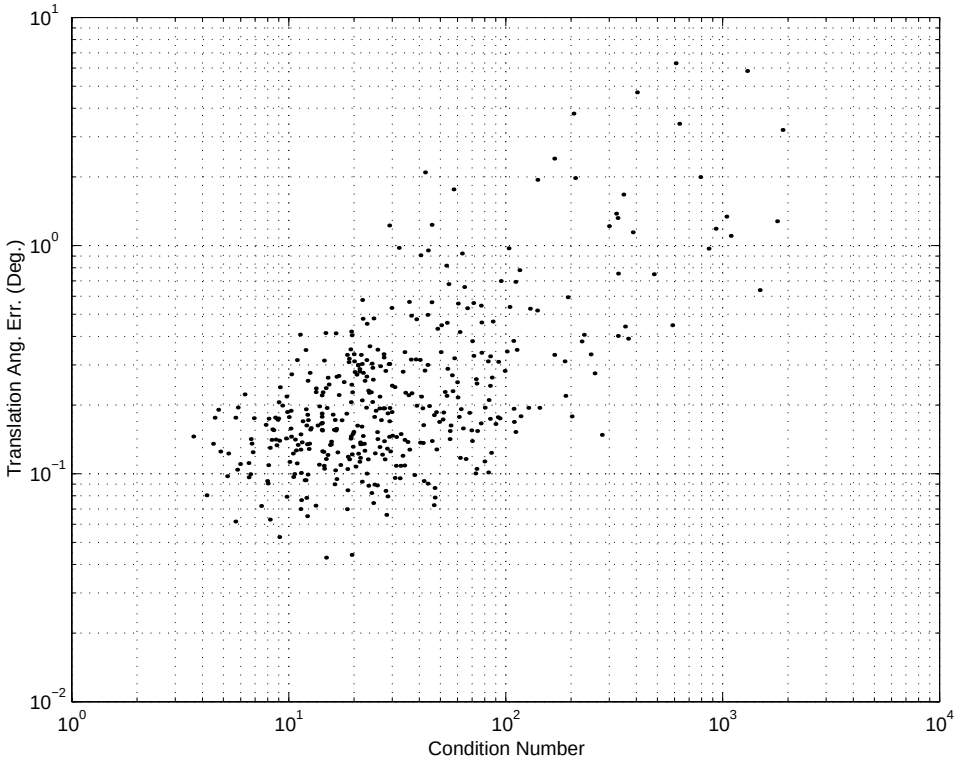

Figure 2-15: Scatter plot of noise-free condition number vs. 50dB SNR translation angular error for $N = 20$. 400 random configurations plotted (same scene generation parameters as in Figure 2-12).

2.7 Camera Calibration and Interior Orientation

When working with real camera images, we are often presented with just the pixel brightness values, and is generally unsafe to assume that the camera was perfect. Even if it were, we do not know the focal length (i.e. zoom) that the photo was taken. Also, the array of pixel data, may have been clipped in an off-centered manner, so that the principal point may not lie in the exact center of the array. If we to this the fact that the pixels are generally rectangular, of unknown aspect ratio, we see then that there are several parameters of the camera that really must be known before any type of exterior or relative orientation is attempted. Once these parameters are determined, it is a simple matter to normalize the image coordinates to a standard camera.

This problem is referred to as the *interior orientation* [88] or camera calibration problem. The method we summarize here follows Trucco and Verri [184]. It is a linear method, based on projective geometry, so it should be used only on clean calibration patterns (i.e. low noise environments).

The approach consists of two steps:

1. Estimate the projection matrix from the world coordinate system to the image coordinates.
2. Compute the interior orientation parameters in closed-form from the projection matrix.

The following describes these operations in more detail.

The interior orientation parameters that we are interested in are the principal distance, f, the principal point (o_x, o_y), and the pixel dimensions s_x and s_y. It will be convenient to define

$$f_x = \frac{f}{s_x}, \quad f_y = \frac{f}{s_y}. \tag{2.131}$$

Assume that we have a known calibration pattern, where easily visible feature points have known 3D locations $\mathbf{a}_i$. We are free to choose any convenient frame of reference for these locations. Typically, the calibration pattern has an obvious, convenient reference frame.

The images of these feature points are related projectively to the image points via

$$\begin{bmatrix} u_i \\ v_i \\ w_i \end{bmatrix} = \mathbf{M} \begin{bmatrix} \mathbf{a}_i \\ 1 \end{bmatrix}, \tag{2.132}$$

with

$$x_i = \frac{u_i}{w_i} = \frac{[m_{11}\ m_{12}\ m_{13}\ m_{14}] \begin{bmatrix} \mathbf{a}_i \\ 1 \end{bmatrix}}{[m_{31}\ m_{32}\ m_{33}\ m_{34}] \begin{bmatrix} \mathbf{a}_i \\ 1 \end{bmatrix}} \tag{2.133}$$

and

$$y_i = \frac{v_i}{w_i} = \frac{[m_{21}\ m_{22}\ m_{23}\ m_{24}] \begin{bmatrix} \mathbf{a}_i \\ 1 \end{bmatrix}}{[m_{31}\ m_{32}\ m_{33}\ m_{34}] \begin{bmatrix} \mathbf{a}_i \\ 1 \end{bmatrix}}. \tag{2.134}$$

The matrix $\mathbf{M}$ is defined up to an arbitrary scale factor and therefore has only 11 degrees of freedom. Since we get two linear equations for every feature point, at least six feature points can be used to form a set of homogeneous linear equations in the elements of $\mathbf{M}$:

$$\mathbf{A}\mathbf{m} = \mathbf{0}, \tag{2.135}$$

where

$$\mathbf{A} = \begin{bmatrix} \mathbf{a}_1^T & 1 & \mathbf{0}^T & 0 & -x_1\mathbf{a}_1^T & -x_1 \\ \mathbf{0}^T & 0 & \mathbf{a}_1^T & 1 & -y_1\mathbf{a}_1^T & -y_1 \\ & & & \vdots & & \\ \mathbf{a}_N^T & 1 & \mathbf{0}^T & 0 & -x_N\mathbf{a}_N^T & -x_N \\ \mathbf{0}^T & 0 & \mathbf{a}_N^T & 1 & -y_N\mathbf{a}_N^T & -y_N \end{bmatrix}, \tag{2.136}$$

and $\mathbf{m} = [m_{11},\ m_{12},\ \ldots, m_{34}]^T$. The optimal $\widehat{\mathbf{m}}$ estimate, determined up to an unknown scale factor, is found as the right singular vector corresponding to the minimal singular value of $\mathbf{A}$.

Once $\widehat{\mathbf{m}}$ is thus determined, we must recover the interior orientation parameters from it elements. It is easily derived [184] from the perspective projection equations that the elements of $\mathbf{M}$ can be written in terms of the interior and exterior orientation parameters as

$$\mathbf{M} = \begin{bmatrix} -f_x r_{11} + o_x r_{31} & -f_x r_{12} + o_x r_{32} & -f_x r_{13} + o_x r_{33} & -f_x t_1 + o_x t_3 \\ -f_y r_{21} + o_y r_{31} & -f_y r_{22} + o_y r_{32} & -f_y r_{23} + o_y r_{33} & -f_y t_2 + o_y t_3 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix}. \tag{2.137}$$

The negatives appear in the equation because generally both x and y coordinates of the pixel values are flipped with respect to the coordinates of the viewer-centered coordinate system. For convenience, define

$$\widehat{\mathbf{m}}_1 = \begin{bmatrix} \widehat{m}_{11} \\ \widehat{m}_{12} \\ \widehat{m}_{13} \end{bmatrix},\ \widehat{\mathbf{m}}_2 = \begin{bmatrix} \widehat{m}_{21} \\ \widehat{m}_{22} \\ \widehat{m}_{23} \end{bmatrix},\ \widehat{\mathbf{m}}_3 = \begin{bmatrix} \widehat{m}_{31} \\ \widehat{m}_{32} \\ \widehat{m}_{33} \end{bmatrix},\ \widehat{\mathbf{m}}_4 = \begin{bmatrix} \widehat{m}_{14} \\ \widehat{m}_{24} \\ \widehat{m}_{34} \end{bmatrix}. \tag{2.138}$$

Simple algebra shows that

$$o_x = \frac{\widehat{\mathbf{m}}_1^T \widehat{\mathbf{m}}_3}{\widehat{\mathbf{m}}_3^T \widehat{\mathbf{m}}_3}, \tag{2.139}$$

$$o_y = \frac{\widehat{\mathbf{m}}_2^T \widehat{\mathbf{m}}_3}{\widehat{\mathbf{m}}_3^T \widehat{\mathbf{m}}_3}, \tag{2.140}$$

$$f_x^2 = \frac{\widehat{\mathbf{m}}_1^T \widehat{\mathbf{m}}_1}{\widehat{\mathbf{m}}_3^T \widehat{\mathbf{m}}_3} - o_x^2, \tag{2.141}$$

$$f_y^2 = \frac{\widehat{\mathbf{m}}_2^T \widehat{\mathbf{m}}_2}{\widehat{\mathbf{m}}_3^T \widehat{\mathbf{m}}_3} - o_y^2. \tag{2.142}$$

The denominator in the above expression is needed to remove the arbitrary scale factor in $\widehat{\mathbf{m}}$.

To illustrate this method, Figure 2-16 shows a picture of a Rubik's Cube, obtained from an uncalibrated camera. Because we know that the various romboids in the picture are actually squares, we can set up a three-dimensional coordinate system along the edges of the cube, and can therefore guess the three-dimensional coordinates of the various corners

in the image. In the figure, the circle marked "15" is the origin, "25" is the unit x direction, "14" is the unit y direction, and "11" is the unit z direction.

The corners in the image were marked by hand, so that the image feature locations are not particularly accurate. The above algorithm was run, with the result that the principal point was about $(42, 109)$, and s_x and s_y were nearly identical. With the calibration parameters now in hand, the exterior orientation algorithm of Section 2.6.2 was run. This algorithm yielded

$$\mathbf{R} = \begin{bmatrix} -0.9387 & 0.3442 & -0.0190 \\ 0.0419 & 0.1688 & 0.9848 \\ 0.3422 & 0.9236 & -0.1729 \end{bmatrix}, \tag{2.143}$$

and

$$\mathbf{Rt} = \begin{bmatrix} -2.3566 \\ -0.3677 \\ 23.5270, \end{bmatrix} \tag{2.144}$$

which is the translation from the camera origin to three-dimension coordinate system origin, measured in the camera reference frame. This says that the camera was about 23 units away from the Rubik's cube, We also note that the condition number, defined in Section 2.6.2, for this problem was approximately 4.2, indicating that the efficient exterior orientation method was also extremely stable for this situation.

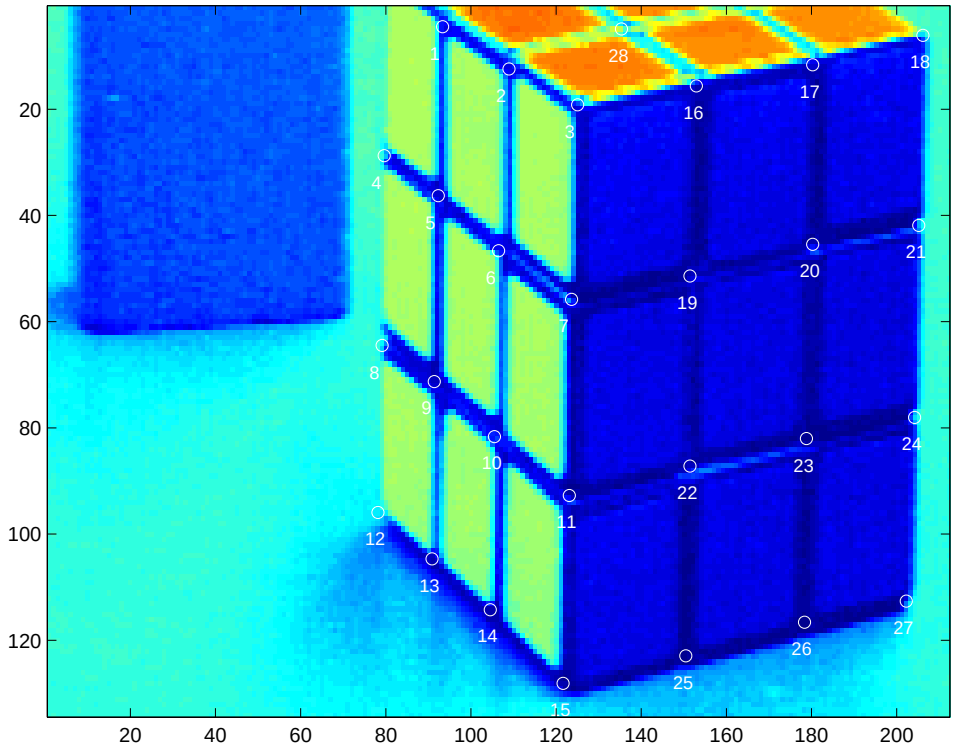

Figure 2-16: Image used for camera calibration.

2.8 Summary

This chapter was concerned with giving a detailed mathematical introduction to the structure, motion, and orientation recovery problem. At first glance, this recovery would seem to be a straightforward geometry problem; however, the diversity of problem formulations, and the complexity of the solution approaches given in this chapter should have dispelled this notion.

This chapter relates to the main theme of the thesis by giving a solid introduction to the various orientation problems, as well as the two dominant modeling approaches, feature-based and gradient-based methods. We focused on the perspective projection; we should note that other approximate image formation models are receiving a great deal of interest, because the equations are much easier to solve [5], [6], [180].

For two data sets, the transformation in three-dimensional coordinates due to motion is described by (2.5). For images, (2.6) and (2.7) describe the coplanarity constraint for matched feature points. Section 2.2.1 describes the simple, approximate linear methods for recovering the motion, as well as the accurate, nonlinear iterative methods.

The other major model, the gradient based methods for relative orientation, was treated in Section 2.3. The brightness change constraint equation (2.29) and (2.30), and its specializations are the important concepts here. We saw that the planar scene situation admitted a closed-form solution; we will treat this case more extensively in Chapter 4, by analyzing the accuracy of the approach, as well as offering an efficient hardware realization for the algorithm.

Estimation of motion from three perspective views was also introduced in Section 2.4, for both feature-based and gradient-based approaches. This multiple view situation will be treated more extensively in Chapter 5, where we are more concerned about accurately combining information, by taking into account the relative information content in different types of observations.

Section 2.5 treated absolute orientation, an important operation for registering models and observations. Using the Orthogonal Procrustes algorithm, the absolute orientation problem is easily and exactly solved. This ease of solution is in contrast to all other orientation problems, and in fact has led researchers to investigate generalizations of the absolute orientation; we investigate this in greater detail in Chapter 5.

Section 2.6 dealt with the exterior orientation problem and photogrammetry, mathematically described by (2.99). The resultant approach for exterior orientation given in Section 2.6.1 served as an introduction to the use of multivariate polynomials in comper vision. The classical bundle adjustment technique (Appendix C) is closely related to this problem. While exact in the maximum likelihood sense, the complexity of the bundle adjustment method gives little insight into the orientation problem; we will reexamine this issue in greater detail in Chapter 5 by adopting an estimation-theoretic approach. Standard estimation theory concepts, such as error ellipses and Fisher information matrices, will give a more intuitive development, as well as allowing for a generalization to many different types of data sets.

While mainly a review chapter, there were several novel ideas put forth:

- the convergence rate analysis of iterative solution for a moving planar scene (Section 2.3.3),
- the linear exterior orientation method (Section 2.6.2), which compared favorably in both computation cost and estimation performance to the current resultant-based

approach.

The development of these items required use of signal processing concepts to understand the effects of noise; more extensive use of these concepts will be made in Chapters 4 and 5.

Chapter 3

Efficient Real-Time Computation and Field Programmable Gate Arrays

3.1 Introduction and Chapter Overview

To date, all of the research effort on orientation and photogrammetry has been algorithmic in nature. The computations were designed to be run on a conventional, general purpose computer or workstation. With the availability of reprogrammable devices comes the opportunity to implement efficient, custom, real-time versions of these algorithms, but only if the challenges inherent in optimizing the hardware design can be overcome.

Researchers have recognized the importance of customized implementation for DSP, image processing, and computer vision problems [66], [201], [202]. Recently, Ratha and Jain [153] describe the mapping of filtering, segmentation, and fingerprint classification to an FPGA-based computer. Efficient FPGA implementations of feature point selection and tracking for image sequences has also received attention. Fiore *et al.* [59] approximated an eigenvalue-based window tracking algorithm, obtaining nearly identical performance with just a handful of additions and comparisons. Benedetti and Perona [17] had a less efficient approximation, requiring multiplications and squarings. Extremely high-performance template matching has been demonstrated using FPGA computers for Synthetic Aperture Radar (SAR) target detection [62], [107], [194].

Lightweight, power-efficient processing is being increasingly demanded for incorporation into video processing and image understanding systems. Several ongoing DARPA programs [138], [143] are researching efficient methods, from low-level devices to advanced algorithms, for gathering, transmitting, and understanding image data.

Techniques for reduced precision computation have received recent attention. Care must be used by the designer when simple wordlength reduction methods, such as truncation, are used. Simple methods can introduce biases and excessive noise. Several authors have introduced schemes that reduce the bias of truncation with little additional cost; see [54] for details.

Wordlength reduction introduces noise into the data stream, so the designer must balance the need for an efficient implementation with output quality. To perform this tradeoff, Barnes *et al.* [14] modeled the effect of truncation as injection of additive white uniform noise. The algorithm wordlengths can be determined by minimizing either the cost or variance metric, subject to constraints on the other metric. Generally, this is a nonlinear optimization problem (although Fiore and Lee [61] derived the optimal wordlengths in closed-form for a standard adaptive LMS filter). Choi and Burleson [32] used this prob-

lem formulation, and proposed a branch-and-bound solution method. Simulation-based approaches have perhaps received the most attention; indeed, this is the approach used in a commercially-available CAD tool [72]. Recently, Fiore [56] has investigated Pareto-optimal solutions, which attempt to simultaneously minimize both metrics.

The remainder of this chapter is organized as follows. Section 3.2 reviews the technologies used to implement real-time systems. Older devices (essentially software-based) are reviewed first in order to set the stage for the newer class of custom computing devices. Because the ever-increasing device density seems to engender a similar increase for demanding processing, it appears we will never have "enough" processing power. For custom computing then, we will periodically be faced with the wordlength optimization problem. To this end, Section 3.3 introduces a new and powerful approach, based on the Markov Chain Monte Carlo (MCMC) method. By using this well known algorithm, we are guaranteed, with probability one, to find all interesting optimal designs. Experience has shown that the algorithm gives good results fast. Also in Section 3.3 is a usable new iteration that bounds the achievable cost and quality for a particular algorithm, which is very useful information. In many cases, this algorithm can be used to initialize the MCMC approach.

While Section 3.3 is concerned with globally optimizing a particular design, Section 3.4 is concerned with a more local optimization approach, which I have termed "lazy rounding." This new heuristic approach is based on optimizing the wordlength reduction of the addition operation; despite its simplicity, the approach has received a great deal of interest [58], [54]. We derive the method in this section and also apply it to the design of truncated multipliers. Since the majority of custom computing designs appear to be centered around custom additions and multiplications, this technique is highly relevant to the custom computing field.

Section 3.5 touches upon another aspect of a successful mapping of an algorithm to custom hardware: that of rearranging the computation to minimize the number of expensive multiplications. We show in this section how this problem is intimately related to that of decomposing tensors into sums of rank-1 outer products. Also, due to the extreme difficulty of this task in general, we explore a new, heuristic approach.

In Section 3.6 we apply these optimization concepts to the efficient FPGA calculation of a critical algorithm for orientation problems, the singular value decomposition. We will make use of the specialized structure given in Section 3.6 to accelerate algorithms for multiple view orientation in Chapter 5. Section 3.7 examines the mapping of a particular preprocessing stage for absolute orientation.

Finally, Section 3.8 summarizes the main results of this chapter, and put the contribution of these ideas into the proper perspective.

3.2 Real-Time Technology

Real-time system designers face a constant struggle to balance efficiency, speed, and upgradability. In this section, we briefly review some of the technologies that are used to implement real-time systems, with special attention paid to the newest class of device, the field programmable gate array.

3.2.1 Microprocessors

A stored program computer provides a highly versatile real-time system, allowing the execution of a wide range of algorithms. This approach has the benefit of minimization an

amortized non-recurring engineering (NRE) cost, because one hardware design may be used for many applications. This minimization is the primary driving force behind the popularity of the stored-program computer, and the most visible member of that family, the microprocessor.

Unfortunately, the price of versatility is (comparatively) low-performance. Because a stored-program is being executed, large amounts of chip area must be devoted to the program sequencer, or controller. Only a fixed number of instructions are supported. Also, very rigid structuring of data accesses between memory and computational units are required. Of late, the area requirements of the controller have increased dramatically, due to the inclusion of onboard local program and data memory, or caches. These caches are used in an attempt to reduce much of the overhead associated with stored-program execution. They also in effect provide more wires from the memory to the processing units, thereby allowing the processing units to operate closer to full capacity.

The multitasking nature of the environments where most microprocessors are used, namely personal computers, also has a detrimental influence the performance of these chips. Specifically, the overhead in terms of speed and area of supporting dynamic linking and virtual memory access can greatly reduce the throughput capability of the device.

Generally, the arithmetic-logic unit (ALU) of a microprocessor can perform only a small set of basic operations. Typically, designers of these chips focus on operations that are most likely to be used in a variety computing fields, such as computer graphics, financial computing, networking (and Internet access), and embedded process controllers. The basic arithmetic operations that support all these application areas are addition/subtraction and multiplication/division. Binary operations, such as bit-wise ANDing, ORing, etc., as well as bit rotations and shifts are generally supported as well. It has only been in the last ten years that there has been a high enough gate count on a single chip to support the larger operations of multiplication and division at near single-cycle operation. Generally, the sustained execution of these operations falls far short of the peak execution rate, usually because of the bussing and memory structure.

General purpose microprocessors tend to have full implementations of floating point arithmetic, because most users do not understand or (justifiably) cannot be bothered with the intricacies of fixed point algorithm implementation. While convenient for software developers, floating point functionality comes at a price: it is area consuming, and rarely single cycle. Because of the large size of these units, even for simple operations such as addition, it is nearly impossible to achieve performance speedup of an algorithm through parallelization of floating point units.

Another problem with general purpose microprocessors is that the advertised processing rate is often difficult to achieve, because of the necessity of using compilers. Compilers translate high level programs into machine executable code. The skill of the compiler designer is often the deciding factor that makes or breaks the reputation of a given processor. Real-time performance is often impossible to achieve from a high-level language implementation, because the general purpose compiler for a new device is rarely fully optimized.

Despite all these drawback, the general purpose microprocessor has flourished; undeniably, they are the single most useful integrated circuit ever devised. Because of their popularity, manufacturers have been able overcome the aforementioned limitations through massive investment in process improvement; the clock speed and gate count of these devices are always at the cutting edge of technology, because the return on investment is unmatched.

3.2.2 DSP Microprocessors

Very few general purpose microprocessors have direct hardware implementations of arithmetic and mathematical operations beyond the basic arithmetic operations, simply because it is not cost effective to do so. Since these devices generally do not satisfy the needs of small, efficient real-time signal processing systems, in the last decade, a new type of device emerged: the digital signal processing (DSP) microprocessor.

Of late, the telecommunications industry has been the driving force behind this type of processor, with many varieties and subspecies of DSP microprocessors flourishing. The DSP microprocessor closely resembles a conventional, general purpose microprocessor in that it executes stored programs, has a program controller, ALU, and memory interface. The main difference is that its ALU is optimized for signal processing. Sustained, single cycle multiply-accumulate operations are the norm. This is generally accomplished through specialized memory access instructions, which are designed to keep the ALU processing at 100 percent capacity.

Of course, sacrifices must be made in order to achieve this type of guaranteed throughput. Generally, multitasking and virtual memory support are eliminated, as well as some other niceties to support complicated operating systems. There are many devices that forego floating point functionality, supplying only fixed point processing. These devices exist because the systems they are targeted to require either very low power operation, or must be extremely inexpensive to manufacture. The companies using these devices are willing to expend the additional design effort to field products with these devices, because the extra development cost will be recouped quickly by a larger market share.

It should also be clear that the division between general purpose and DSP microprocessors is not sharp; rather, a spectrum of devices exist. DSP algorithms may be run on all these devices, but with differing levels of design effort, final unit cost, and capabilities.

3.2.3 Application-Specific Integrated Circuits (ASICs)

While the DSP microprocessor revolutionized the design and development of real-time signal processing systems, there are many applications which require processing throughput far beyond what may be supplied by a DSP microprocessor. Many radar and image processing algorithms, for instance, must operate in throughput ranges up to four orders of magnitude larger than may be supplied by a single device. Very few customers have the ability to buy and maintain systems consisting of large numbers of microprocessors. Instead, a different processing paradigm is employed: application-specific integrated circuits (ASICs).

ASICs are custom hardware circuits that provide precisely the functionality needed for a specific task. By carefully tuning the ASIC design for the specific task, gains in efficiency, cost, size, and power consumption over microprocessors may be had.

ASICs, and their older cousins, fully-custom designed integrated circuits, are not new; by definition, they predate microprocessors. More importantly, DSP engineers traditionally sought a solution among ASICs when faced with a large, processing intensive, real-time design problem. The advent of the DSP microprocessor ate away at the low-end market segment, and indeed continues to do so. Despite this, the ASIC design industry continues to thrive, solely because there exist processing tasks that cannot be handled (and never will be handled), by DSP microprocessors. Indeed, performance speed up of thousands, or even millions, times that achievable by a general purpose microprocessor for specific problems have been achieved by custom ASIC designs.

ASICs are applied to tasks which justify the increased cost of designing specialized logic. All modern personal computers have one or several ASICs to perform tasks such as video control, sound generation, hard disk accessing, and memory management. The final designed ASIC is often very inexpensive to manufacture; products with a large potential market are therefore the driving force behind the ASIC industry.

For signal and image processing applications, ASICs are very successfully applied to problems which exhibit a high degree of parallelism. Systolic arrays, whose characteristics include a regular, repetitious structure, usually simple processing elements, and mostly local interconnections, are often implemented in ASICs.

Another class of DSP designs which find common ASIC implementations are those that consist of a deep processing pipeline. In these problems, pipeline stages correspond to different steps of an algorithm; When processing of a piece of data is completed at one stage of the pipeline, the data is then moved on to the next pipeline processing stage.

Many DSP tasks can be naturally decomposed into systolic or pipelined computational structures, and for many years, DSP implementation was solely the province of ASIC designers.

The usual analogy is that of an assembly line. In contrast, a general purpose microprocessor may be viewed analogously to a craftsman; the craftsman can do a good job on nearly any task, whereas the assembly line produces essentially only one item, but at fantastic rates of speed. Also, it is very expensive to retool the assembly line to produce a different product. Finally, to minimize expenses, great pains are taken to ensure smooth flow of the assembly line, with no idle workers. For ASICs and general purpose microprocessors, the same comments apply. To get fantastic performance on essentially one algorithm, an ASIC is the answer. However, the end volume must justify the cost. The ASIC had better solve the correct problem, because it cannot generally be used for other problems.

Even as there was a continuum of devices among general purpose and DSP microprocessors, the boundary between ASICs and microprocessors is not really sharp. Many vendors offer standardized microprocessor "cores," which may be incorporated into a custom ASIC.

While attractive from a computational throughput viewpoint, the drawback of ASICs are their NRE cost and design time. Clearly, with more flexibility available over microprocessors, comes the burden of actually specifying the contents of the ASIC. Thus, experienced teams are generally required to successfully complete an ASIC in a timely manner. Also, these skills are relatively specialized; while it is common for many people to write software on a daily basis, comparatively few people design ASICs.

Hardware synthesis tools ease this development process. However, the comments made earlier about compilers apply here as well. In fact, the problem is actually worse, because there are many more degrees of freedom in the design.

We close this section by noting that many computer vision tasks are suitably implemented an *analog* VLSI. Mcquirk *et al.* [126], [127] implemented a focus-of-expansion (FOE) analog array. McIlrath (nee Dron) [47], [48] built an analog network for edge detection and image reconstruction. McIlrath [125] is constructing a specialized CCD imaging array, which incorporates processing directly on the imager. Horn [88] proposed an a resistive network for recovering "lightness," which is related to the reflectance of scene objects. Comments made earlier about about the development cost and performance of ASICs also apply to the analog devices: the devices offer the lowest power and highest performance of any approach, but the circuits are so specialized that it is often not economical to build them.

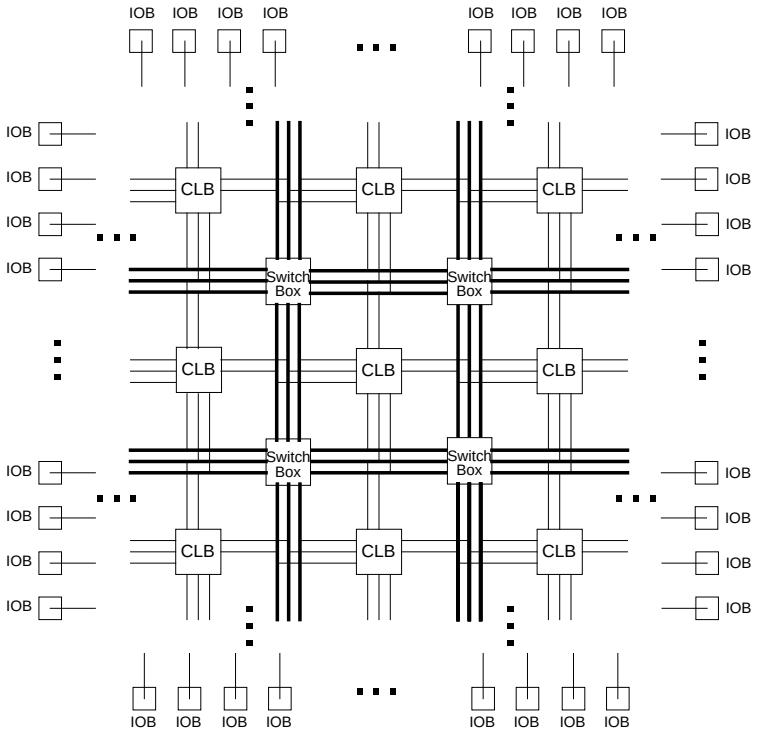

Figure 3-1: A typical FPGA device configuration.

3.2.4 Field Programmable Gate Arrays

The high risk, large up-front cost, and long design time of ASICs prompted the development of a whole new industry and device family: field programmable gate arrays (FPGAs). Actually, FPGAs have their roots in older programmable logic devices (PLDs). In their early years, FPGAs were seen merely as large PLDs. Within the past five or so years, however, DSP designers have increasingly used FPGAs to take the place of ASICs for some processing tasks [109]. Hauck [80], Mangione-Smith *et al.* [123], and Goslin [70] provide readable summaries of FPGA technology. Other work includes that of Mirsky and DeHon [131], and Whittig [204]. FPGAs are ideally suited for pipelined, dataflowgraph applications. Many researchers have mapped high performance DSP applications into these devices. Some examples are:

- Filters and FFTs [55], [70], [106], [130], [197],
- Frequency estimation [60], [186],
- IR detection [24],
- Feature detection and tracking [17], [59],
- Template Matching [62], [107], [194],
- Segmentation [153].

Unlike ASICs, which are programmed at the factory, the function implemented by an FPGA can be modified repeatedly by the user, long after the device has left the factory.

For many arithmetic intensive applications, such as signal and image processing, an ASIC would provide significant computational speedup. Unfortunately, due to the cost and development effort, such a custom ASIC is never created. These applications are then relegated to slower performance software implementations on DSP microprocessors. FPGAs correct this problem by providing near ASIC performance with the flexibility of software.

There are many arithmetic operations which could benefit from custom implementation, and which have known, efficient, special purpose implementations, but for which custom ASICs are not available. For instance, functions not provided by the ALU of a microprocessor (i.e. square roots, CORDIC, and eigenvalue calculations), operations with non standard wordlength requirements (very low or very high precision), and complex functions composed of specific operation sequences (i.e. matrix multiplication). These operations are all typically very time consuming in software; an FPGA could greatly improve the performance, without the associated cost and risk of an ASIC design.

FPGAs devices are typically organized as an array of small configurable logic blocks (CLB), embedded in a sea of routing resources (as shown in Figure 3-1). The CLBs are generally identical in terms of their capabilities.

Around the perimeter of the device are usually input/output logic blocks (IOBs). In addition to merely providing connections to the package pins, the IOBs typically have the ability to to perform simple functions, such as complementing, registering, tristating, as well as providing for the different voltage levels associated with different logic families.

The routing resources of FPGAs connect the internal processing logic blocks to each other and the IOBs. Typically, multiple levels of routing are arranged into a hierarchy. At the lowest level, a complete nearest neighbor interconnection is usually provided. Next, long lines which traverse the length and width of the device are provided. Finally, a small number of global routes are provided. Critical signals, such as clock and reset lines, are typically hardwired by the manufacturer. To connect logic blocks to the various routing wires, embedded in the logic array is another array of routing blocks. The logic blocks are directly connected to the routing blocks, as are the long lines. The routing blocks are then configured to provide the desired connections.

The reconfigurability of FPGAs has only recently been been used as part of the real time system design. Real time system designers make use of the reconfiguration capability in several ways. The least demanding method is to switch between functions on command, which is the hardware equivalent of finishing one program and loading another. Reconfiguration times on the order of a second are generally acceptable for a wide range of applications. The ability to execute multiple hardware functions in a time share mode can thus been seen to reduce the system cost because multiple single use ASICs are not required. Clearly, though, the throughput of the FPGA will not in any way compare to that achievable by multiple, custom ASICs, so this approach must be considered carefully.

Most FPGA devices in operation today require a second or more to switch from one configuration to another (this presupposes that the configurations have been previously designed). A reconfiguration time on the order of a second is perfectly suitable for testing alternative circuit designs, or for occasionally upgrading products in the field There are newer devices that can actually reconfigure must faster; one experimental device can actually reconfigure the entire array in one clock cycle. A very hot topic in the FPGA research community are devices that continually adapt the hardware function they implement in real time as a function of the data and processing environment. While the mechanics of

rapid configuration are non-trivial, the biggest challenge with this data-dependent, dynamic reconfiguration is the control strategies required for specifying the new design.

The various levels of reconfigurability found in FPGAs comes at a great price: reconfigurability costs gates. In fact, very large portions of the devices are dedicated to the reconfiguration and routing (in some cases, almost 90 percent). Thus, FPGAs will always be limited to an order of magnitude smaller usable gate count than an identical ASIC technology. FPGAs also suffer in terms of speed as compared to ASICs. Because of the fixed, general purpose routing structure on the FPGA, routing between CLBs will in general be slower than a custom route on an ASIC. For similar reasons, the power dissipation of FPGAs compared to a custom ASIC design is also much higher.

Today's FPGAs also have other limitations; not all computations are well suited to FPGA implementation. FPGAs excel at bit-level operations, such as comparisons and integer arithmetic, but are a poor match for high precision floating point operations. Only recently have there been devices with large, onboard RAM. Most FPGAs require external memories for intermediate result storage. Algorithms with branching are an extremely poor match to an FPGA implementation (i.e. resources must be allocated to all branch possibilities, resulting in many unused gates for a given piece of data).

The challenge of using FPGAs is thus clear: one must seek algorithms which can be decomposed into low precision operations, that utilize as much spatial locality in data movement as possible, and that can be deeply pipelined or parallelized to limit I/O operations. The benefits that accrue if this can be done are great: dramatic decreases in cost, size, and power dissipation, with an increase in throughput has been demonstrated over microprocessors for appropriately mapped functions. Compared to ASICs, dramatically reduced development risk, time, and cost, as well as the ability to field upgrade virtually for free, are the benefits that FPGAs offer.

Real time designers have come to realize that a powerful approach is to include microprocessors, FPGAs, and external memory in a design. This combination allows the strengths of each device to be used to the fullest.

3.2.5 FPGA Details

The FPGAs that we consider here have CLBs arranged in a rectangular array, embedded in a sea of routing resources. Usable gate counts of over 1 million gates are commonly available, with system clock speeds of 50MHz, and over 500 I/O pins.

Figure 3-2 shows a simplified view of a CLB for the most popular family of FPGAs, the Xilinx 4000 (XC4000) family [207]. The CLB contains two 4-input lookup tables, (the F and G lookup tables, or LUTs), and one 3-input lookup table (the H LUT). The F and G LUTs can both implement any function of four variables. When the F, G, and H LUTs are used together, any function of five boolean variables may be implemented, and some functions of nine variables as well. Unused inputs can be treated as "don't cares," so functions with fewer variables than stated above may be implemented.

Each CLB contains two flip-flops, which may be bypassed if desired. The flip-flops have clock lines, clock enables, and asynchronous sets and clears (not shown in Figure 3-2). Both the registered and unregistered outputs are available for use every clock cycle. Each flip-flop may be preloaded to a zero or one at powerup.

The XC4000 family has a built-in fast ripple-carry chain, connecting the F and G LUTs. This capability allows a single CLB to implement two bits of a ripple carry adder (RCA). The carry chain is so fast that it is extremely difficult to justify using any other method

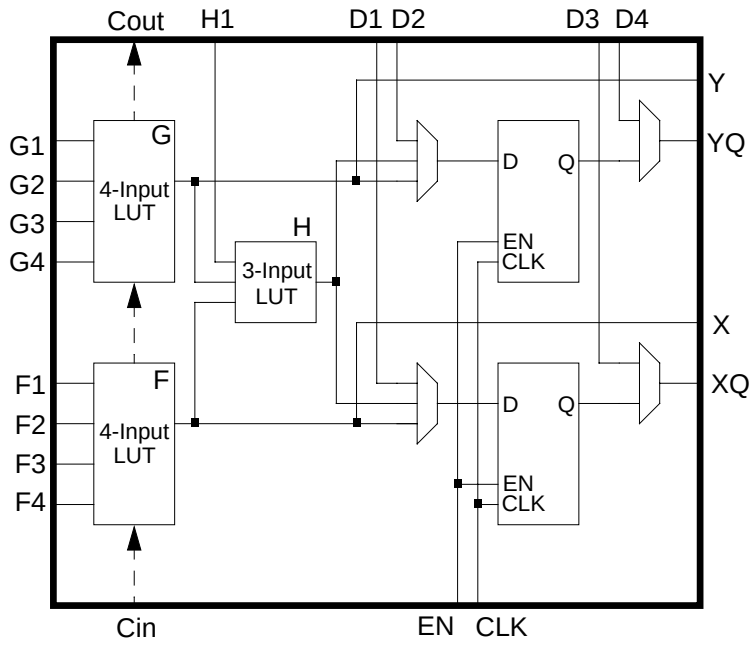

Figure 3-2: Simplified view of a Xilinx 4000 CLB.

of "fast" addition, such as carry lookahead adders (CLA), carry save adders (CSA), or conditional-sum adders [35]. Of course, this fast carry chain consumes silicon area and dissipates power, so in effect, algorithms that do not use it are wasteful. Control modes are available to allow the combined F and G LUTs to implement subtractors, adder/subtractors, incrementers, and 1's complementers, and 2's complementers. Figure 3-3 shows the basic implementation of the RCA.

The LUTs and multiplexers (MUXs) shown in Figure 3-2 are configured during powerup by the device to the settings specified by the user's design software. Thus, the LUTs are really RAMs, which are loaded with the appropriate bit patterns to implement the user's boolean functions. The control lines for the MUXs are also driven by hidden RAM bits, loaded at powerup, and generally not alterable during operation.

The RAM implementation of the LUTs also leads to a very powerful capability of the XC4000 family: hardware hooks are in-place so that each F and G LUT may be configured as a user accessible 16×1 RAM. Essentially, the F and G inputs become the address lines of the RAM, and the auxiliary D inputs become the read/write, and data input signals. (This capability is not shown in Figure 3-2.)

A related capability that the XC4000 family possesses is to configure the F and G LUTs as "RAM-pipes." This is essentially means that both a read and a write may occur in one clock cycle, which allows the construction of a fixed length shift register. Each LUT is able to implement a one bit wide shift register, of up to 16 delays. Note that the internal shifts are not accessible in this mode. This shifting capability can lead to a large savings in flip-flop usage.

Multipliers are also commonly implemented in FPGAs; Figure 3-4 shows a 4×4 signed Pezaris array multiplier [121], built from Full Adder cells, AND gates, and NAND gates. Figure 3-5 shows a floorplan of the 4×4 signed multiplier. The grey shaded areas in the figure indicate how basic building blocks and interconnect are to be replicated for different

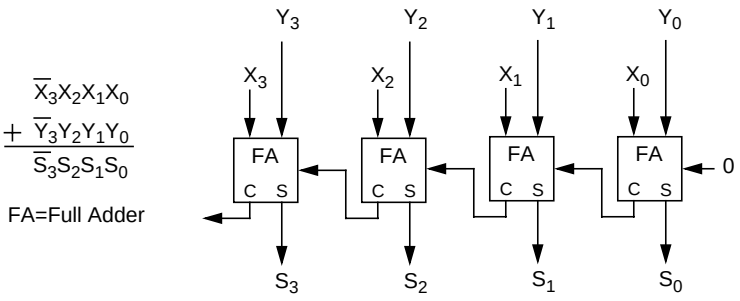

Figure 3-3: Ripple carry adder in Xilinx 4000 FPGA. Each "FA" block is a full-adder cell, which adds the three input bits to produce the sum bit (S) and the carry bit (C). The Xilinx 4000 CLB can fit two FAs.

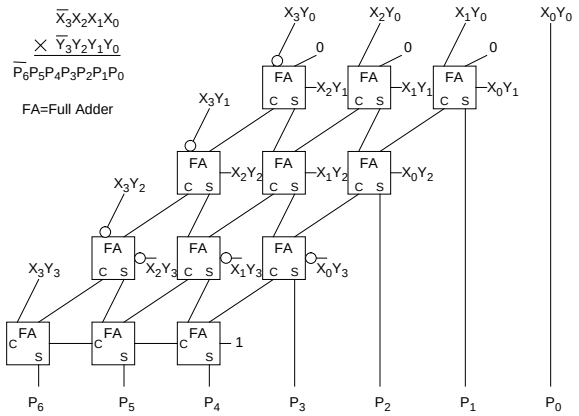

Figure 3-4: Signed, nonpipelined Pezaris array multiplier.

sized multipliers. Figure 3-6 shows this in more detail. In this figure, the numbers inside the grey shaded areas are the row and column sizes in CLBs needed for the region. The text accompanying each area is the required functionality inside each CLB within the area. Figures 3-4, 3-5, and 3-6 show how to floorplan the multiplier, and what logic functions belong in each CLB, as a function of operand size.

These figures show the calculation of all the product bits, which is often unecessary because of observation noise; in Section 3.4, we will investigate optimal truncation approaches that save on hardware cost and yet maintain desired rounding noise characteristics.

Figure 3-7 shows an extremely simplified view of a portion of the routing structure of the XC4000 family. Between each row and column of CLBs are a series of wires, categorized by the number of rows (or columns) that are skipped before interruption. At the appropriate intersection of horizontal and vertical lines, are switch matrix units (not shown in Figure 3-7). This switches are configured at powerup to connect the horizontal and vertical channels. Clearly, the routing and switch units occupy a large percentage of real estate on the chip.

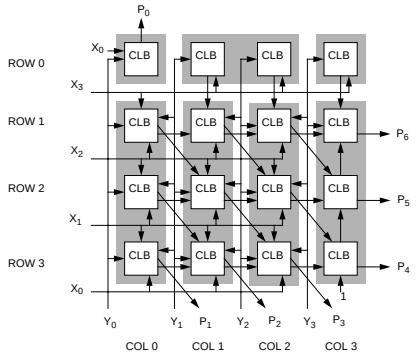

Figure 3-5: Floorplan of 4×4 signed multiplier in XC4000 technology.

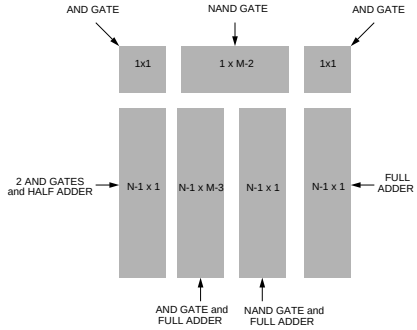

Figure 3-6: Replication pattern for an $N \times M$ signed multiplier.

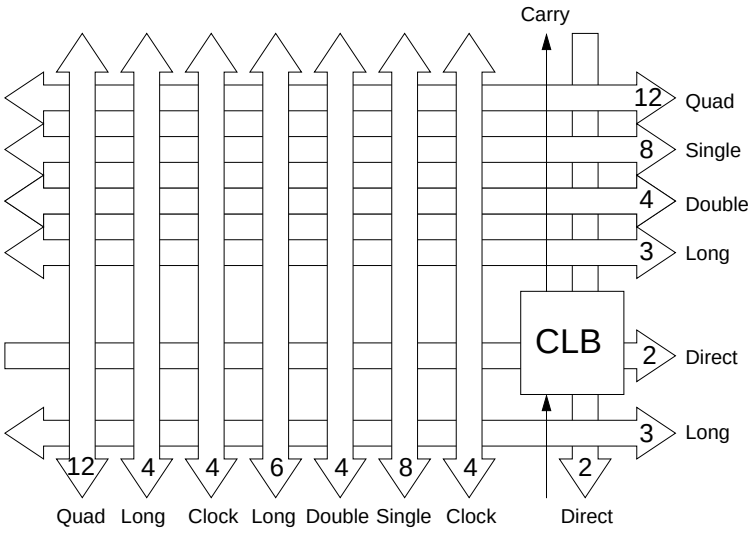

Figure 3-7: Simplified view of a Xilinx 4000 routing structure.

3.3 Wordlength Optimization Methods for FPGAs

In order to reduce hardware requirements, one must round or truncate data at various points throughout an algorithm. Many operations such as multiplication or accumulation can greatly expand the wordlength of the data. If these operations are cascaded, then without some form of wordlength reduction, excessive hardware growth results.

Wordlength reduction introduces noise into the data stream, so the designer must balance the need for an efficient implementation with output quality. To perform this tradeoff, we model the effect of truncation as the injection of additive white uniform noise [142] (Barnes *et al.* [14] showed conditions for which this is appropriate).

In this section, we address the problem of finding good wordlength combinations for DSP and computer vision flowgraphs. Each arc i of the flowgraph will have a wordlength b_i bits associated with it. We desire to find vectors $\mathbf{b} = [b_1 \ \ldots \ b_n]^T$ of wordlengths so that good combinations of a given cost function $C(\mathbf{b})$ and a given performance function $V(\mathbf{b})$ result.

This problem has been treated at a rudimentary level in the literature. For linear systems (i.e. filters, FFTs), analysis has been extensively used to determine the effect of roundoff errors [9], [41], [142], [205]. For nonlinear problems, locally greedy approaches that use a sensitivity measure to determine the greatest cost reduction are typical [55], [102].

Simulation-based approaches have perhaps received the most attention; indeed, this is the approach used in a commercially-available CAD tool [72]. A simulation-only approach can be time-consuming, and is highly dependent on the inputs signals presented to it. Also, the simulations must be repeated for every proposed wordlength combination [108]. Kim *et al.* [104] and Sung and Kum [175] use a full precision simulation to gather statistics about the distribution of numbers on each arc in the flowgraph. The range is then estimated using these statistical measures. While range information is necessary, the *sensitivity* of the function to rounding on a particular arc is also of importance. Choi and Burleson [32] properly account for this sensitivity by using the allowable noise variance as a design constraint.

3.3.1 Determination of Performance Function $V(\mathbf{b})$

For simplicity, we will use rounding noise variance in the output of the calculation as the performance function $V(\mathbf{b})$. Clearly, other metrics may be used, such as mean square error (MSE), and signal-to-noise ratio (SNR). Using variance rather than MSE is appropriate if there is no bias introduced by the wordlength reduction. Care must be used by the designer when simple wordlength reduction methods, such as truncation, are used. Fortunately, for adders and multipliers, which are common DSP building blocks, several authors have introduced wordlength reduction schemes that reduce the bias of truncation with little additional cost; see [43], [54], [105], [116], [161] for details.

In order to determine the performance function $V(\mathbf{b})$, we use classical results from numerical analysis. We wish to examine the sensitivity of the calculation $y = f(\mathbf{x})$. Let the vector $\mathbf{x}$ represent both the primary inputs to the algorithm as well as the additional inputs created by the injection of additive noise. The change in the output y as a function of a change in the inputs is approximately calculated by expanding $f(\mathbf{x})$ in a Taylor series and keeping only the first order terms,

$$\Delta y = \sum_j \frac{\partial f(\mathbf{x})}{\partial x_j} \Delta x_j. \tag{3.1}$$

This shows that partial derivative of the calculation acts as a "transfer function," which determines the contribution of noise to the output of the calculation. Since the partial derivative varies, depending on the values of the primary inputs, the transfer function changes, depending on the particular operating point.

Analysis of the absolute error is appropriate for fixed point implementations, which is what we focus on in this thesis. As an aside, we note that it is straightforward to show similar results for relative error propagation, which is the appropriate measure for floating point implementations. Indeed, Stoer and Bulirsch [172] give the following formula for relative error propagation:

$$\epsilon_y = \sum_j \frac{x_j}{f(\mathbf{x})} \frac{\partial f(\mathbf{x})}{\partial x_j} \epsilon_{x_j}, \tag{3.2}$$

where ϵ_{x_j}, is the relative error in an input x_j, and ϵ_y is the relative error in the output y.

If one were willing to map the input space of $\mathbf{x}$, the worst case transfer functions could be calculated. This will of course be computationally prohibitive except for small or specialized problems. The following is a usable suboptimal approach. For each injection point, obtain the transfer function from the injection point to the output, while maintaining all the auxiliary paths from the other primary inputs. Next, derive what the worst case inputs would be that maximize the first derivative of the transfer function. Use this maximal derivative to determine the constant scale factor in determining the output noise contribution.

The gain coefficients of the transfer function are therefore

$$s_i = \max_{x_i} \left| \frac{\partial f(\mathbf{x})}{\partial x_i} \right|. \tag{3.3}$$

Elementary probability gives the output noise variance as

$$\sigma_y^2 = \sum_i s_i^2 \sigma_{x_i}^2, \tag{3.4}$$

where $\sigma_{x_i}^2$ is the rounding noise variance of an input x_i. If arc i has maximum absolute value M_i, we first round this up to the next power of two to prevent overflows:

$$\tilde{M}_i = 2^{\lceil log_2(M_i) \rceil}. \tag{3.5}$$

Now, rounding arc i to b_i bits introduces uniform noise with range $[-\tilde{M}_i 2^{-b_i},\ \tilde{M}_i 2^{-b_i}]$. The variance of this noise is [14]

$$\sigma_{x_i}^2 = \frac{1}{12} \tilde{M}_i^2 2^{-2b_i}. \tag{3.6}$$

$$\begin{aligned} V(\mathbf{b}) \ \stackrel{\triangle}{=} \sigma_y^2 \ &= \textstyle\sum_i \frac{1}{12} s_i^2 \tilde{M}_i^2 2^{-2b_i}, \\ &= \textstyle\sum_i a_i 2^{-2b_i}, \end{aligned} \tag{3.7}$$

with $a_i \stackrel{\triangle}{=} s_i^2 \tilde{M}_i^2 / 12$.

We shall use the form of $V(\mathbf{b})$ in (3.7) to find optimal designs. The ranges M_i and the maximum absolute slopes s_i must be determined in order to calculate the a_i. In simple cases, these values can be hand calculated. However, for complicated flowgraphs, this becomes tedious. One may therefore need to resort to search-based methods to approximately determine these values.

The approach described above relies on maximum absolute slopes. Functions with points where the slope is infinite will thus present a problem to this method. In this situation,

two approaches may be taken. The first is to simply bound the independent variables away from the singular point. The second approach is to use an upper bound on the wordlength that may be known a priori.

3.3.2 Pareto-Optimal Designs

Here, we investigate Pareto-optimal solutions, which attempt to simultaneously minimize both $C(\mathbf{b})$ and $V(\mathbf{b})$ metrics.

Consider a known cost function $C(\mathbf{b})$ and known variance function $V(\mathbf{b})$. We propose many random combinations $\mathbf{b}$, calculate $C(\mathbf{b})$ and $V(\mathbf{b})$, and plot the variance versus the cost (the association between the particular $\mathbf{b}$ that generated each data point is maintained). The combinations of interest are those on the boundary of the set of data points closest to the coordinate axes.

These points are known as *Pareto-optimal* points [20]. A point is Pareto-optimal if no other point has both lower cost and variance. This curve provides a designer flexibility in choosing designs, when one is not quite sure how much cost can be tolerated (or how much output noise can be tolerated). Clearly, given the lower bounding curve, if one finally is also given a hard cost constraint, then the optimal design (i.e. minimum variance subject to not exceeding the cost constraint) is easily determined. Another situation where all the lower bounding points are needed is when one is populating a design library.

Note that if we relax the integer constraint on the elements of $\mathbf{b}$, a convex continuous lower bounding curve usually results (as long as $C(\mathbf{b})$ and $V(\mathbf{b})$ are continuous). With integer constraints, the true lower bounding curve may not be convex (and is certainly not continuous).

We can find the lower bounding curve on the Pareto-optimal solutions. Consider convex combinations of the $C(\mathbf{b})$ and $V(\mathbf{b})$:

$$g(\mathbf{b};\alpha) \stackrel{\triangle}{=} C(\mathbf{b}) + \alpha V(\mathbf{b}), \quad \alpha \geq 0. \tag{3.8}$$

If we minimize this with respect to $\mathbf{b}$,

$$\mathbf{b}_{opt}(\alpha) = \arg\min_{\mathbf{b}} g(\mathbf{b};\alpha), \tag{3.9}$$

then $\mathbf{b}_{opt}(\alpha)$ is a Pareto-optimal solution. To see this, assume that the solution $\mathbf{b}_{opt}(\alpha)$ were not Pareto-optimal. By definition, there exists a $\mathbf{b}$ such that both $C(\mathbf{b}) < C(\mathbf{b}_{opt})$ and $V(\mathbf{b}) < V(\mathbf{b}_{opt})$ are true. Multiplying the second inequality by α and adding to the first inequality, we obtain

$$g(\mathbf{b};\alpha) < g(\mathbf{b}_{opt};\alpha), \tag{3.10}$$

which contradicts (3.9).

Thus, (3.9) may be used to derive a lower bounding curve on the Pareto-optimal solutions. We first relax the integrality constraint on $\mathbf{b}$, and also relax any other constraints that may be placed on the wordlengths. Then, α is initialized to zero, $\mathbf{b}_{opt}(\alpha)$ is found via (3.9), and $C(\mathbf{b}_{opt}(\alpha))$ is plotted against $V(\mathbf{b}_{opt}(\alpha))$. We then slightly increase α, and repeat the procedure. Each minimization is initiated by the previous value of $\mathbf{b}_{opt}(\alpha)$ to speed the optimization. The method is robust because $V(\mathbf{b})$ is a convex function (due to the form (3.7)), and experience has shown that $C(\mathbf{b})$ is generally convex as well. This implies that $g(\mathbf{b};\alpha)$ is a convex function, so minimization is straightforward.

3.3.3 Lower Bound Minimization Algorithm

For a fixed α we desire to perform the minimization

$$\mathbf{b}^* = \arg\min_{\mathbf{b}} C(\mathbf{b}) + \alpha V(\mathbf{b}), \tag{3.11}$$

where we relax the integer constraints on $\mathbf{b}$. Taking gradients and setting equal to zero,

$$\nabla C(\mathbf{b}) + \alpha \nabla V(\mathbf{b}) = \mathbf{0}. \tag{3.12}$$

Using (3.7) (the definition of $V(\mathbf{b})$) and rearranging (3.12),

$$a_i 2^{-2b_i}(-2\ln 2) = -\frac{1}{\alpha}\nabla C(\mathbf{b})_i, \quad i = 1, \ldots, N. \tag{3.13}$$

Taking logarithms and rearranging gives

$$b_i = -\frac{1}{2}\log_2 \frac{\nabla C(\mathbf{b})_i}{2\alpha a_i \ln 2}, \quad i = 1, \ldots, N. \tag{3.14}$$

This is a set of nonlinear equations for the optimal $\mathbf{b}$. To actually solve them, we define an iteration

$$b_i^{(k+1)} = -\frac{1}{2}\log_2 \frac{\nabla C(\mathbf{b})_i^{(k)}}{2\alpha a_i \ln 2}, \quad i = 1, \ldots, N, \tag{3.15}$$

where k is the iteration index.

Convergence Rate

Recall that the convergence of an iteration of the form $\mathbf{x}^{(k+1)} = \mathbf{f}\left(\mathbf{x}^{(k)}\right)$ to the solution $\mathbf{x}^* = \mathbf{f}(\mathbf{x}^*)$, is governed by the Jacobian of $\mathbf{f}(\mathbf{x})$. We can easily see this by relating the error of the k^{th} iteration to the error of the $(k+1)^{st}$. Let the error at the k^{th} step be defined as

$$\mathbf{e}^{(k)} = \mathbf{x}^{(k)} - \mathbf{x}^*, \tag{3.16}$$

and similarly for $\mathbf{e}^{(k+1)}$. Linearizing $\mathbf{f}(\mathbf{x})$ about $\mathbf{x}^*$ gives,

$$\mathbf{f}(\mathbf{x}) \approx \mathbf{f}(\mathbf{x}^*) + \left.\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right|_{\mathbf{x}^*} (\mathbf{x} - \mathbf{x}^*). \tag{3.17}$$

Evaluating at $\mathbf{x} = \mathbf{x}^{(k)}$ and rearranging,

$$\begin{aligned} \mathbf{f}\left(\mathbf{x}^{(k)}\right) - \mathbf{f}(\mathbf{x}^*) &\approx \left.\tfrac{\partial \mathbf{f}}{\partial \mathbf{x}}\right|_{\mathbf{x}^*} \left(\mathbf{x}^{(k)} - \mathbf{x}^*\right), \\ \mathbf{x}^{(k+1)} - \mathbf{x}^* &\approx \left.\tfrac{\partial \mathbf{f}}{\partial \mathbf{x}}\right|_{\mathbf{x}^*} \left(\mathbf{x}^{(k)} - \mathbf{x}^*\right), \\ \mathbf{e}^{(k+1)} &\approx \left.\tfrac{\partial \mathbf{f}}{\partial \mathbf{x}}\right|_{\mathbf{x}^*} \mathbf{e}^{(k)}. \end{aligned} \tag{3.18}$$

Or finally,

$$\mathbf{e}^{(k+1)} \approx \mathbf{J}\mathbf{e}^{(k)}, \tag{3.19}$$

where $\mathbf{J} = \left.\partial \mathbf{f}/\partial \mathbf{x}\right|_{\mathbf{x}^*}$ is the Jacobian matrix of the iteration. An iteration of the form $\mathbf{x}^{(k+1)} = \mathbf{f}\left(\mathbf{x}^{(k)}\right)$ will converge if the magnitude of the largest eigenvalue of $\mathbf{J}$ is less than one at the equilibrium point $\mathbf{x}^*$.

To calculate the Jacobian of the iteration (3.15), let

$$\mathbf{f}_i(\mathbf{b}) = -\frac{1}{2}\log_2 \frac{\nabla C(\mathbf{b})_i}{2\alpha a_i \ln 2}, \quad i = 1, \ldots, N. \tag{3.20}$$

Taking derivatives and cancelling common terms gives the elements of the Jacobian matrix as

$$J_{i,j} = \frac{\partial \mathbf{f}_i(\mathbf{b})}{\partial b_j} = -\frac{1}{(2\ln 2)} \frac{\partial \nabla C(\mathbf{b})_i}{\partial b_j} \frac{1}{\nabla C(\mathbf{b})_i}, \quad i, j = 1, \ldots, N. \tag{3.21}$$

For most flowgraphs, $\mathbf{J}$ is a sparse matrix. This is because two nodes i and j can only appear together in a term of $C(\mathbf{b})$ if the two nodes are connected to a common functional block. For i and j such that the two nodes are not connected to a common functional block, $\nabla C(\mathbf{b})_i$ does not depend on j, so that $\partial \nabla C(\mathbf{b})_i / \partial b_j = 0$, and hence $J_{i,j} = 0$.

It is interesting to note that $\mathbf{J}$ does not depend on either the variance injection scale factors a_i or the particular value of α. For cost functions $C(\mathbf{b})$ that are separable, in that they can be written as a sum of terms which only depend on one variable, (3.21) gives $\mathbf{J} = \mathbf{0}$. This really means that we can independently solve for each component of $\mathbf{b}$ in (3.14). For the special case of a linear cost function,

$$C(\mathbf{b}) = \sum_{i=1}^{N} c_i b_i, \tag{3.22}$$

we can actually get a closed-form expression for the optimal $\mathbf{b}$ from (3.15):

$$b_i = -\frac{1}{2}\log_2 \frac{c_i}{2\alpha a_i \ln 2}, \quad i = 1, \ldots, N. \tag{3.23}$$

A similar result was derived by Fiore and Lee [61], where the α played the role of a Lagrange multiplier in the constrained optimization problem: minimize $C(\mathbf{b})$, subject to $V(\mathbf{b}) = A$.

Eigenvalue Upper Bound

If we desire to monitor the iterations when solving (3.11) using the iteration (3.15), we can calculate $\mathbf{J}$ using (3.21) and then find the largest eigenvalue at every step. A more efficient alternative is to make use of the inequality [171]

$$\|\mathbf{J}\|_2^2 \leq \|\mathbf{J}\|_1 \|\mathbf{J}\|_\infty, \tag{3.24}$$

where $\|\mathbf{J}\|_p$ is the induced p-norm of a matrix $\mathbf{J}$. In particular,

$$\|\mathbf{J}\|_1 = \max_j \sum_{i=1}^{N} |J_{ij}|, \tag{3.25}$$

$$\|\mathbf{J}\|_\infty = \max_i \sum_{j=1}^{N} |J_{ij}|, \tag{3.26}$$

and $\|\mathbf{J}\|_2$ is equal to the largest singular value σ_1 of $\mathbf{J}$. It is also known that $\sigma_1 > |\lambda_1|$, where λ_1 is the largest eigenvalue of $\mathbf{J}$.

Given all this, we can calculate an upper bound on λ_1 of $\mathbf{J}$ by calculating (3.25) and (3.26), taking the product of these two quantities, and then comparing this to one. If the product is less than one, then the iteration (3.15) is stable. If the product is greater than one, it may indicate that the iteration is unstable. In this case, a more careful calculation of the eigenvalue λ_1 is required. For sparse $\mathbf{J}$, the calculation of (3.25) and (3.26) will be particularly efficient, requiring only absolute values and additions.

Quadratic Cost Function

The special case where $C(\mathbf{b})$ is a quadratic form also deserves special attention. We make the reasonable assumption that all $C_{i,j} \geq 0$. In this case, $C(\mathbf{b})+\alpha V(\mathbf{b})$ is a convex function, and therefore has only one minimum. We have

$$C(\mathbf{b}) = \mathbf{b}^T \mathbf{C} \mathbf{b}, \tag{3.27}$$

$$\nabla C(\mathbf{b})_i = 2 \sum_{k=1}^{N} C_{i,k} b_k, \tag{3.28}$$

and

$$\frac{\partial \nabla C(\mathbf{b})_i}{\partial b_j} = 2C_{i,j}. \tag{3.29}$$

Substituting into (3.21), we obtain

$$J_{i,j} = -\frac{C_{i,j}}{(2\ln 2)\sum_{k=1}^{N} C_{i,k} b_k}, \quad i,j = 1,\ldots,N. \tag{3.30}$$

Summing over j,

$$\sum_{j=1}^{N} |J_{i,j}| = \frac{\sum_{j=1}^{N} C_{i,j}}{(2\ln 2)\sum_{k=1}^{N} C_{i,k} b_k} \leq \frac{\sum_{j=1}^{N} C_{i,j}}{(2\ln 2)\min(\mathbf{b})\sum_{k=1}^{N} C_{i,k}} = \frac{1}{(2\ln 2)\min(\mathbf{b})}, \ \ i = 1,\ldots,N. \tag{3.31}$$

Since (3.31) held for all i, it holds for the i that maximizes

$$\sum_{j=1}^{N} |J_{i,j}|, \tag{3.32}$$

or, in other words,

$$\|\mathbf{J}\|_\infty \leq \frac{1}{(2\ln 2)\min(\mathbf{b})}. \tag{3.33}$$

We can similarly obtain a bound on $\|\mathbf{J}\|_1$. Summing over i, we have

$$\sum_{i=1}^{N} |J_{i,j}| = \sum_{i=1}^{N} \left(\frac{C_{i,j}}{(2\ln 2)\sum_{k=1}^{N} C_{i,k} b_k} \right) \leq \frac{1}{(2\ln 2)\min(\mathbf{b})} \sum_{i=1}^{N} \left(\frac{C_{i,j}}{\sum_{k=1}^{N} C_{i,k}} \right), \quad j = 1,\ldots,N. \tag{3.34}$$

Therefore,

$$\|\mathbf{J}\|_1 = \max_j \sum_{i=1}^{N} |J_{i,j}| \leq \frac{1}{(2\ln 2)\min(\mathbf{b})} \max_j \sum_{i=1}^{N} \left(\frac{C_{i,j}}{\sum_{k=1}^{N} C_{i,k}} \right). \tag{3.35}$$

A more convenient, but slightly looser bound can be obtained via

$$\sum_{k=1}^{N} C_{i,k} \geq C_{i,j}. \tag{3.36}$$

Using this, the bound in (3.35) becomes

$$\|\mathbf{J}\|_1 \leq \frac{N}{(2\ln 2)\min(\mathbf{b})}. \tag{3.37}$$

Combining the bounds (3.33) and (3.37) with (3.24), we obtain the sufficient condition for stability for a quadratic cost function:

$$\min(\mathbf{b}) \geq \frac{\sqrt{N}}{2 \ln 2}. \tag{3.38}$$

For example, with $N = 64$, if all components of $\mathbf{b}$ have at least 6 bits, the iteration is converging.

3.3.4 Sampling Approaches for Feasible Designs

We are given functions $C(\mathbf{b})$ and $V(\mathbf{b})$ and wish to find Pareto-optimal integer n-dimensional vectors $\mathbf{b}$. We assume there is a maximum wordlength allowable, so that $1 \leq b_i \leq N$. Typical values for the maximum wordlength is $N = 32$ or $N = 64$. The value n may be small or quite large. In the absence of other constraints, there are a large number of possible combinations, in fact N^n of them. This is exponential in terms of n, so we do not expect to be able to exhaustively search the space. Also note that, for $N = 32$, a value of $n \geq 20$ would result in more than $2^{100} \approx 10^{30}$ combinations, which effectively places a limit on the size of n for exhaustive search.

Often, we may wish to consider constraining the allowable $\mathbf{b}$ combinations in some way. For example, a square law device cannot produce an output with more than twice the number of input bits. It seems reasonable then to prune the feasible region using these "natural" constraints, if it is convenient to do so.

There are other types of constraints that we may not be able to ignore. For example, the design options for a particular component may not allow arbitrary wordlength combinations on its inputs. Another way that this may arise is that we do not have a formula for the cost and variance for all $\mathbf{b}$ combinations (perhaps we were lazy in compiling these formulas, a very real possibility if the formulas are generated by computer). Also, we may have some idea of where the optimal cost/variance points lie in the plane; this manifests itself as a (rather complicated) constraint on the allowable $\mathbf{b}$'s. Finally, if we are considering partitioning the design into several FPGAs using a particular cut of the flowgraph, there may be constraints on the number of wires that may cross the cuts, and also constraints on the total cost within a particular FPGA.

The foregoing arguments lead us to strongly consider sampling from a feasible subset defined by a set of constraints. Since $C(\mathbf{b})$ and $V(\mathbf{b})$ can be arbitrarily complicated, we can in general do no better than sampling uniformly over the feasible region, at least initially. One could perform local greedy optimizations to slightly improve a solution found by uniform sampling.

We now demonstrate a naive Monte Carlo approach for uniform sampling, which we will improve upon in the next section. The algorithm may not produce points that are truly Pareto-optimal, and we have no way to detect this (in polynomial time); however, we shall prove that with high probability, all Pareto-optimal points will be found if we sample enough times.

Magnitude Calculation Example

In this section we derive optimal designs for the magnitude operation $y = \sqrt{a^2 + b^2}$. Figure 3-8 shows a flowchart of this operation, with the wordlength definitions b_1 through b_6. We further assume that the inputs are bounded by $0 \leq a \leq 1$ and $1/2 \leq b \leq 1$. This is done to

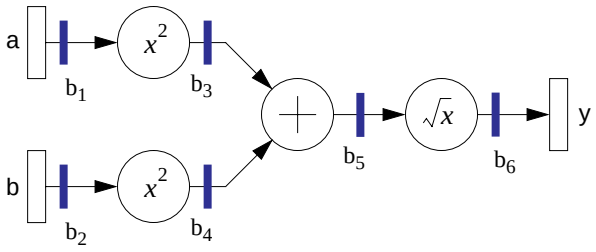

Figure 3-8: Flowchart for magnitude calculation.

avoid the infinite slope at $a = b = 0$. For each noise injection point, we must find the worst case absolute slope and maximum arc absolute value. Table 3.1 tabulates these calculations.

Coeff.	$f(x)$	$f'(x)$	a,b	x	max f'	a_i	M_i	m_i	Val.
α_1	$\sqrt{x^2+b^2}$	$\frac{x}{\sqrt{x^2+b^2}}$	$b = 1/2$	1	$\frac{2}{\sqrt{5}}$	$\frac{4}{5}$	1	1	$\frac{1}{15}$
α_2	$\sqrt{a^2+x^2}$	$\frac{x}{\sqrt{a^2+x^2}}$	$a = 0$	1	1	1	1	1	$\frac{1}{12}$
α_3	$\sqrt{x+b^2}$	$\frac{1}{2\sqrt{x+b^2}}$	$b = 1/2$	0	1	1	1	1	$\frac{1}{12}$
α_4	$\sqrt{a^2+x}$	$\frac{1}{2\sqrt{a^2+x}}$	$a = 0$	1/4	1	1	1	1	$\frac{1}{12}$
α_5	$\sqrt{x}$	$\frac{1}{2\sqrt{x}}$		1/4	1	1	2	4	$\frac{1}{3}$
α_6	x	1		1/4	1	1	$\sqrt{2}$	4	$\frac{1}{3}$

Table 3.1: Gain coefficient calculation for magnitude example.

Thus, from Table 3.1, we have that the output variance is given by

$$\sigma^2 = \frac{4}{5}\sigma_1^2 + \sigma_2^2 + \cdots + \sigma_6^2, \tag{3.39}$$

where σ_i is the noise injected at point i. Writing this in terms of the wordlengths b_i using (3.6) and (3.7),

$$V(\mathbf{b}) = \frac{1}{15}2^{-2b_1} + \frac{1}{12}\left(2^{-2b_2} + 2^{-2b_3} + 2^{-2b_4}\right) + \frac{1}{3}\left(2^{-2b_5} + 2^{-2b_6}\right) \tag{3.40}$$

Now that we have the variance equation, let us postulate a cost function. A reasonable one is

$$C(\mathbf{b}) = b_1^2 + b_2^2 + \max(b_3, b_4) + 1 + b_5 b_6. \tag{3.41}$$

The b_1^2 and b_2^2 terms in this cost function result from implementing the squaring function with a Pezaris array multiplier (see Section 3.2.5 and [121]). The "max" term assumes using a ripple carry adder. The multiplicative term $b_5 b_6$ comes from the square-root implementation given by Hwang ([96], Figure 11.2), and is only approximate.

For the purposes of this section, this cost function is good enough. For any real FPGA design, each of these terms would have additional scale factors to reflect the relative cost of adders, multipliers, etc. Also, for different regimes of $\mathbf{b}$, the implementation style may change. For example, we may switch from a Pezaris array to some other multiplier configuration [57], [121], [140], [195], resulting in a cost function that has a very complicated (but known) formula.

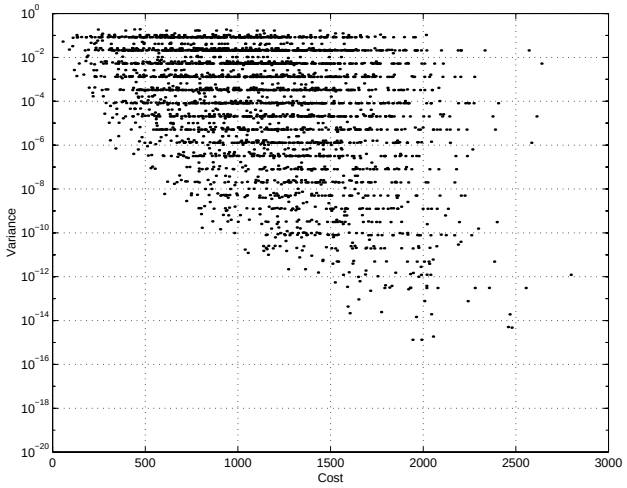

Figure 3-9: Cost vs. variance for naive Monte Carlo random sampling.

For the magnitude design, thirty thousand random **b** vectors were drawn uniformly from $1 \leq b_i \leq 32$. Only about ten-thousand actually satisfied the natural wordlength constraints (the rest were discarded). The cost versus variance for the feasible points is plotted in Figure 3-9. Note that most of the sample points are dominated by the comparatively few Pareto-optimal points.

3.3.5 Markov Chain Monte Carlo Method

The major problem with the previous naive sampling approach was that constraints on the wordlength allocations made it difficult to sample uniformly from the set of feasible allocations. If the constraints made the feasible set small, then many random samples were wasted because they fell into the infeasible region. Another problem is that most of the random samples are not even close to the Pareto-optimal lower boundary, and are thus wasted. We therefore look for a method to uniformly sample from a potentially arbitrary-shaped feasible region. This section is concerned with using random walks on Markov chains to perform this sampling.

Markov Chain Basics

A Markov chain is a discrete-time stochastic process for which the state at time t depends only on the state at time $t-1$. The relationship between one time step and the next is probabilistic. The probability that the chain is in state j at time t, given that the chain is in state i at time $t-1$, is given by

$$P(S_t = j \mid S_{t-1} = i) = P_{ij}, \tag{3.42}$$

where the numbers P_{ij} are given. These numbers are collected into a *transition matrix* $\mathbf{P}$. Clearly, $0 \leq P_{ij} \leq 1$ and $\sum_i P_{ij} = 1$. We will concern ourselves here only with Markov chains with a finite number of states, that are aperiodic and ergodic [134]. For these chains, after a suitable number of time steps, the probability that the chain is in state i is defined as π_i. We collect these into a row vector π, called the *stationary distribution*, or steady

state probability vector. If at some time step we are in the stationary distribution π, then at the next step we will still be in the stationary distribution, yielding

$$\pi = \pi \mathbf{P}. \tag{3.43}$$

A result, known as the Fundamental Theorem of Markov chains, states that the stationary distribution π is unique for the Markov chains we consider here.

A random walk on a graph is closely related to the state sequence on a Markov chain. Given and undirected (multi)graph G with k states, we can construct its underlying Markov chain M as follows. Replace each undirected arc by two directed arcs (one in each direction). The transition probability on an arc leaving vertex v is set to be $1/d(v)$, where $d(v)$ is the degree of (number of arcs leaving) vertex v. Often, we associate one or more self-loops with each vertex of G (thus increasing the total degree and changing the transition probabilities of M accordingly). This will be done to remove any potential periodicity in M.

Let m be the total number of edges in G. Then it is shown in [134] that

$$\pi_v = d(v)/2m. \tag{3.44}$$

From (3.44), if we can construct a graph where $d(v)$ is constant for all v, then $\mathbf{P}$ will be symmetric. In this case $\mathbf{P}$ is called *doubly stochastic* and the stationary distribution π will in fact be a uniform distribution, $\pi = [1, \ldots, 1]/k$. This is the crucial construction we need to enable (near) uniform sampling from a state space. We construct an appropriate graph G (and hence a Markov chain M), run M for "a long time" to ensure that we are in the stationary distribution, and then take as our sample the state we are in at that time.

The key question of course is: "How long is a long time?" To answer this, we examine the eigenvalues λ_i of $\mathbf{P}$. We order the eigenvalues $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_k$, and it is easy to show that $\lambda_1 = 1$ corresponds to the eigenvector π. If we start in an initial distribution $\mathbf{q}^{(0)}$ and run M for t cycles, then the distribution at time t is $\mathbf{q}^{(t)} = \mathbf{q}^{(0)}\mathbf{P}^t$. From elementary linear algebra, the eigenvalues of $\mathbf{P}^t$ are λ_i^t. All eigenvalues less than one approach zero exponentially fast, so that $\mathbf{q}^{(t)} \to \pi$. Clearly, the magnitude of λ_2 controls the convergence rate.

We define the *relative pointwise distance* (RPD) $\Delta(t)$ at a time t. Letting $\mathbf{q}(t)$ be the actual distribution of M at time t, the RPD is then

$$\Delta(t) = \max_i \frac{|q_i(t) - \pi_i|}{\pi_i}. \tag{3.45}$$

The speed at which $\Delta(t)$ goes to zero for increasing t thus determines how long we must run M to ensure that we are in the stationary distribution.

Sinclair and Jerrum [162] prove a key result that relates $\Delta(t)$ to λ_2:

$$\Delta(t) \leq \frac{\lambda_2^t}{\min_{j \neq 1} \pi_j}. \tag{3.46}$$

For a doubly stochastic $\mathbf{P}$, $\pi_j = 1/k$, and this implies

$$\Delta(t) \leq k\lambda_2^t, \tag{3.47}$$

which is a slightly tighter bound than the one derived in [134]. The practical importance of this relationship is that we will only need $O(\log(k))$ steps to reach a desired level of RPD.

Multidimensional Nearest Neighbor Mesh for Uniform Generation

Rather than the naive Monte Carlo sampling algorithm used in Section 3.3.4, instead we use a special Markov chain, that will easily incorporate any constraints on the **b**. The states of the chain are the allowable combinations of **b** that satisfy all constraints. Transitions in the chain are allowed to either increment or decrement one element of the current **b** vector, provided that the new state is feasible. Also, we allow self-loops; a state may transition to itself.

With no constraints, this Markov chain is a multidimensional nearest neighbor mesh in n dimensions. With constraints, we merely delete states that are not feasible, so that the chain is a subset of the nearest neighbor mesh.

Constraining the transitions to the nearest neighbors is a conservative approach. It relies on the observation that the feasible region will usually be convex or nearly-convex (if we relax integrality constraints). Under this condition, we have a good chance that a local move will not attempt to cross a boundary. If we were to increase the allowable distance that a transition could take, we would increase the probability that a boundary is crossed, thus wasting a step.

With no self-loops, this chain would be bipartite (i.e. consider the checkerboard for $n = 2$). Self-loops remove this problem, as well as provide a way to adjust transition probabilities. For uniform generation of feasible states, we must get the transition probabilities correct. Consider the underlying undirected graph G of the Markov chain M. Nodes in the "interior" of the feasible region have $2n$ neighbors, and one self-loop arc. Thus, the maximum degree is $2n + 1$. For any states on the boundary of the region, we add as many self-loops as necessary to bring the degree up to $2n + 1$. We now have that the transition matrix for M is doubly stochastic, so that the steady state distribution is uniform. Figure 3-10 shows an example of this chain with no constraints for $n = 2$.

We note that with too many constraints, the feasible region might not consist of a single connected component. Since there could be a large number of states, it is probable that we would not be able to detect this situation (in polynomial time).

As a practical matter, we do not generate and store the chain. Also, for boundary states, we do not need to augment the number of self-loops beyond the one already present. When we are in a state, we randomly pick one of the $2n+1$ destinations. If the destination happens to be infeasible, we stay in the current state, and try again.

This approach was tried for the magnitude design. Figure 3-11 shows the results of this method for ten-thousand time steps. The natural wordlength constraints for the magnitude operations were used to define a feasible region for this example.

To show the power of the Markov chain approach, a constraint on the allowable cost and variance combinations was added to the design problem. Figure 3-12 shows the results. In the figure, the upper bounding curve is the cost/variance constraint. The cost/variance combinations achieved by the random walk are denoted by the dots. The curve in the middle of the dots is the set of Pareto-optimal solutions given by a naive sampling approach. Notice the vast improvement this constraint affords; we only generate random samples near the true Pareto-optimal boundary. It would appear that the added upper bounding constraint did not disconnect the feasible region, because we seem to have good coverage in Figure 3-12 (although this is not a proof).

The drawback of this approach is that one must guess at the location of the upper bounding curve. Figure 3-12 actually suggests an iterative approach that appears to reduce the number of random samples needed. We iteratively guess at an upper bounding curve,

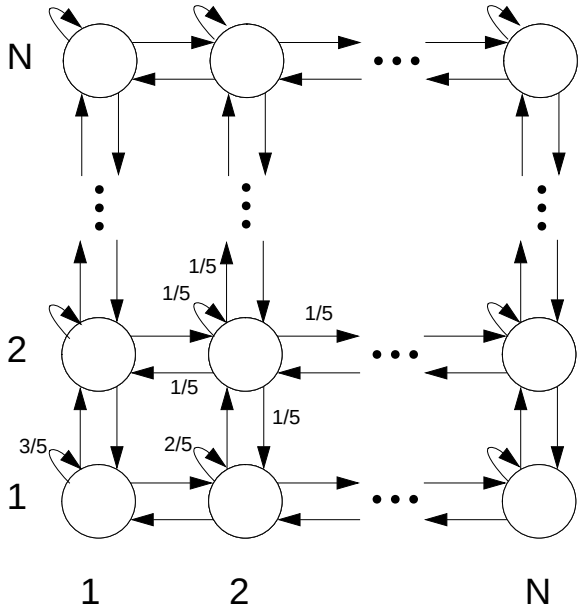

Figure 3-10: Underlying Markov chain for $n = 2$. All horizontal and vertical arcs have transition probability equal to 1/5. The self-loops have different probabilities, depending on their location within the array.

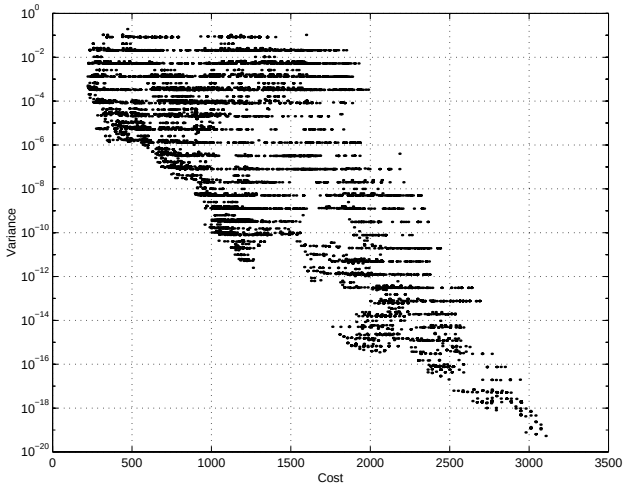

Figure 3-11: Cost vs. variance for Markov chain random sampling.

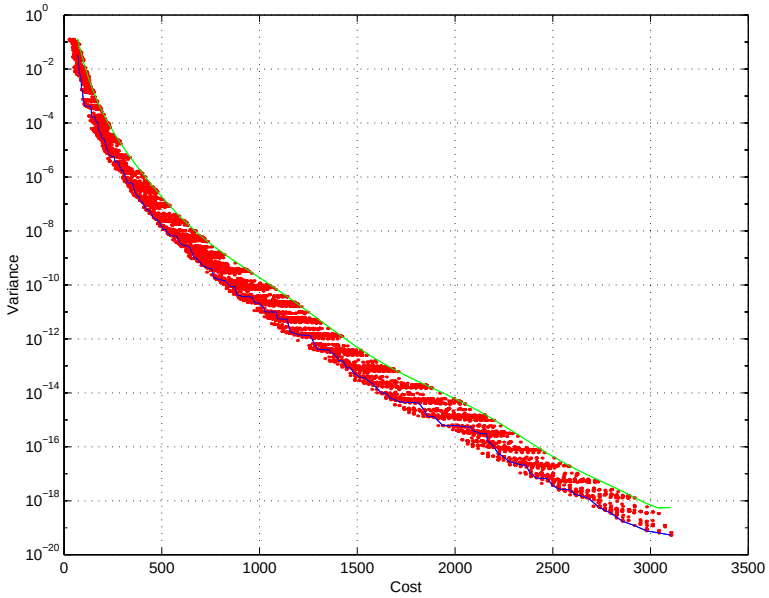

Figure 3-12: Markov chain random sampling with cost/variance constraint.

and run the Markov chain to improve our guess at the set of Pareto-optimal solutions. From this current set, we adjust the upper bounding curve, and repeat the iteration. At each step, we run the Markov chain for only a fraction of the number of steps that we did in Figures 3-9 and 3-11. While the total number of samples will be about the same, the iterative approach will (hopefully) take more of these samples in the good region of the cost/variance plane.

This approach was used to generate Figure 3-13. Each of five iterations used two-thousand time steps. At the end of the iteration, the Pareto-optimal points were found. A polynomial was passed through these points, and this polynomial was raised up in variance slightly so that the feasible region would be less likely to be disconnected. All of the tested **b** combinations are plotted, with a solid line drawn through the final optimal set. Clearly, the algorithm avoided very bad points in the upper right hand corner of the plot.

Partitioning

The Markov chain approach can be easily extended to perform *partitioning* in addition to wordlength allocation. In this problem, we allow the design to occupy several FPGAs, and the signal flowgraph is divided up among the FPGAs. To be feasible, a partition must not overload any one FPGA, and must not require more interconnections among the FPGAs than are available.

Partitioning can be a difficult problem. If we fix the wordlength allocations, (thereby also fixing the costs), then partitioning becomes an integer programming problem. To be required to solve this for every attempted wordlength allocation would result in a very slow algorithm.

To more efficiently compute good partitions, we extend the state space of the Markov chain. In addition to the current wordlength allocation, the state will be augmented with one decision variable per functional unit in the flowgraph (i.e. for the magnitude calculation, there are four functional units). The value of a new decision variable indicates the FPGA

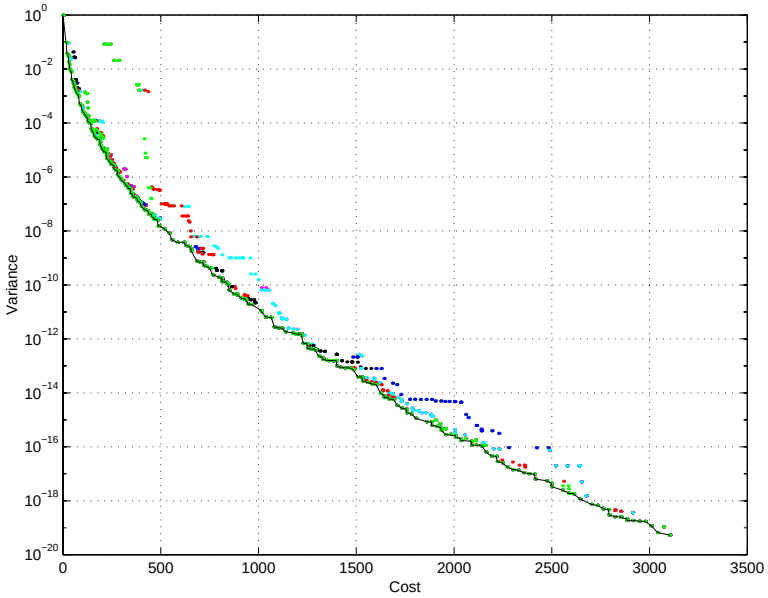

Figure 3-13: Iterated Markov chain random sampling.

to which the functional unit is assigned. As before, we randomly walk around the feasible region, collecting Pareto-optimal solutions. As the partition changes, the set of constraints will change, because different sets of wires are now involved in different cuts. The Markov chain approach easily handles this situation.

3.3.6 Time Required to Reach the Stationary Distribution

We saw in the preceding section that a Markov chain approach could profitably be employed to find good designs. Since we assume no *a priori* knowledge of the locations of these good designs, we constructed the chain so that eventually, uniform sampling from the feasible region resulted. In practice, we do not run for a long time, and then take only one sample. Instead, the current implementation is analyzed at every step, and good ones are retained. The time to reach the stationary distribution is still of interest, because we want to be guaranteed that by the end of a run, we would visit interesting states with equal probability.

In this section, we analyze the time required to reach the stationary (uniform) distribution. We first analyze the case of $n = 1$ (i.e. only one wordlength to assign). We then extend this result to more general n, using a separability argument.

Case $n = 1$

Consider the case $n = 1$. This corresponds to the 1D Markov chain, shown in Figure 3-14. The number of states N is the maximum number of bits we will allow for the wordlength. Clearly, the stationary distribution is uniform. The time to reach the stationary distribution

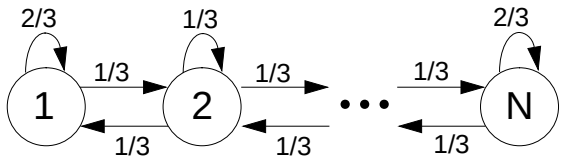

Figure 3-14: Underlying Markov chain for $n = 1$.

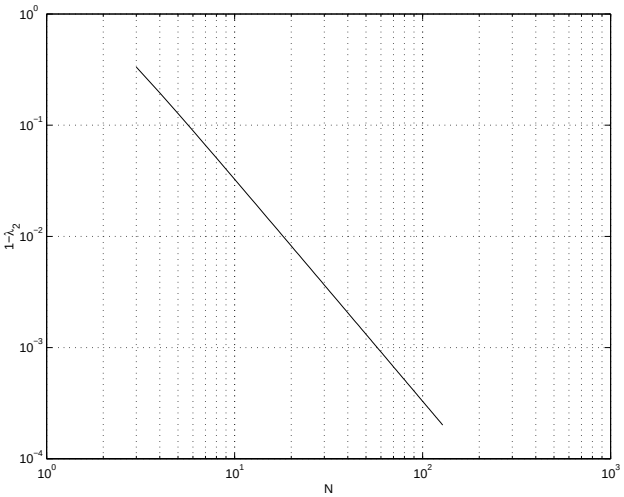

Figure 3-15: Difference between 1st and 2nd eigenvalues for 1D Markov chain.

is dependent on the second largest eigenvalue λ_2 of the transition matrix $\mathbf{P}$, where

$$\mathbf{P} = \begin{bmatrix} \frac{2}{3} & \frac{1}{3} & & & & \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & & & \\ & \ddots & \ddots & \ddots & & \\ & & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & \\ & & & \frac{1}{3} & \frac{2}{3} \end{bmatrix}. \tag{3.48}$$

The closer λ_2 is to the maximum eigenvalue (=1), the longer it takes to reach the stationary distribution. We can easily calculate $1 - \lambda_2$ for various values of N; these are plotted in Figure 3-15.

Figure 3-15 shows that the eigenvalue separation with respect to N is linear on a log-log scale, over the range of interest in N. From the plot we thus have

$$\begin{aligned} \log_{10}(1-\lambda_2) &\approx -2\log_{10}(N) + 0.5, \\ 1-\lambda_2 &\approx 10^{-2\log_{10}(N)+0.5}, \\ \lambda_2 &\approx 1 - \sqrt{10}N^{-2}. \end{aligned} \tag{3.49}$$

This result can be compared with a theoretical calculation of the eigenvalue spread. A Markov chain will mix rapidly if it is unlikely to get stuck in any subset of the states. This property is measured by a quantity known as the *conductance* of the chain. For any

connected subset S and its complement $\bar{S}$ of states in the chain, we define the *capacity* C_S as

$$C_S = \sum_{i \in S} \pi_i, \tag{3.50}$$

and the *ergodic flow* F_S,

$$F_S = \sum_{i \in S, j \in \bar{S}} p_{ij} \pi_i. \tag{3.51}$$

C_S is the probability of being in the set S, and F_S is the probability of leaving S. Defining the conductance of the set S as $\Phi_S = F_S / C_S$, we see that Φ_S is the conditional probability of leaving S given that we are in S. The minimum of the Φ_S over all sets S in the chain (subject to $C_S \leq 1/2$) is defined to be the global conductance Φ of the chain. Clearly, Φ is a measure of the likelihood of getting stuck in some set of states, and should be related somehow to λ_2. It is proved in [162] that this relationship is

$$\lambda_2 \leq 1 - \frac{\Phi^2}{2}. \tag{3.52}$$

For the 1D chain of Figure 3-14, we can find the minimum Φ_S by maximizing the number of states in S while minimizing the probability of leaving S (equivalently, maximizing C_S and minimizing F_S). It is easily seen that $S = \{1, \ldots, N/2\}$ will produce the minimum Φ. This is calculated as

$$C_S = \frac{1}{2}, \quad F_S = \frac{1}{3N}, \quad \Phi = \frac{2}{3N}. \tag{3.53}$$

Combining this with (3.52) gives

$$\lambda_2 \leq 1 - \frac{2}{9N^2}, \tag{3.54}$$

which is compatible with the result given in (3.49).

We have thus shown that the eigenvalue spread is polynomial in $1/N$. This is quite encouraging, because it implies very fast convergence to the stationary distribution.

With the value of λ_2 as a function of N known, we can now calculate the relative pointwise distance (RPD) $\Delta(t)$ as a function of the number of steps t, and $k = N$. Recall that

$$\Delta(t) \leq N \lambda_2^t. \tag{3.55}$$

If we require the RPD to be less than some δ, then it is sufficient to require

$$\begin{aligned} N\lambda_2^t &\leq \delta \\ t \ln(\lambda_2) &\leq \ln(\delta/N) \\ t &\leq \tfrac{\ln(\delta) - \ln(N)}{\ln(\lambda_2)}. \end{aligned} \tag{3.56}$$

We can use (3.49) and (3.56) to plot t as a function of δ and N; this is shown in Figure 3-16. This figure shows that for reasonable wordlengths (i.e. $N \leq 32$), at most a few thousand steps are required to guarantee that we have reached the stationary distribution.

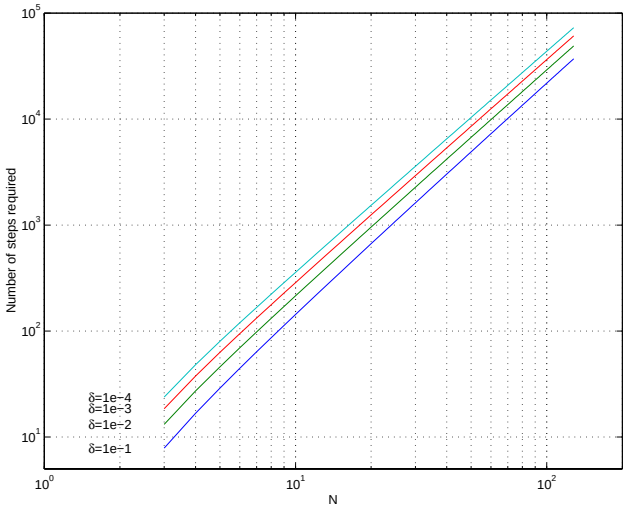

Figure 3-16: Number of steps required to reach uniform stationary distribution for 1D Markov chain.

Case $n > 1$, No Constraints

Realistic design problems of course have $n > 1$. For example, the magnitude calculation problem had $n = 6$. The underlying Markov chain is now an n-dimensional nearest-neighbor mesh. The $n = 2$ case is shown in Figure 3-10. By choosing the self-loop probabilities correctly, it is clear that this Markov chain again has a uniform stationary distribution.

We could go through an analysis similar to the $n = 1$ case. However, this would be tedious because the $\mathbf{P}$ matrix structure becomes complicated. Also, it seems that such an approach would be unenlightening for understanding the behavior as n increases. We can, however, avoid this complexity by realizing that the Markov chain is "separable," in the sense that we are able to consider only one dimension at a time.

For simplicity, we consider only interior nodes (i.e. not on the boundary) of the multidimensional Markov chain. Since the number of nodes on the boundary is insignificant compared to the number of nodes in the interior (for large enough N), we can ignore the fact that the self-loop transitions probabilities change slightly.

Consider movement in the direction corresponding to the first dimension (i.e. horizontally in Figure 3-10). Our current location in any of the other dimensions has no effect on our ability to move in the direction of the first dimension.

We can consider "lifting" a one-dimensional Markov chain out of this multidimensional one in the following manner: Whenever a transition along the first dimension is chosen, make the corresponding transition in the 1D Markov chain. If any other dimension is chosen, the 1D Markov chain performs a self-loop.

On average, we will make a transition in the lifted Markov chain twice for every $2n + 1$ transitions in the multidimensional Markov chain. This corresponds to rescaling t in (3.56) to $t/(n + 1/2)$ (N and λ_2 remain as defined for the 1D case). Therefore, for the multidimensional Markov chain, to reach an RPD of δ, the expected number of required times steps is

$$t = \frac{\ln(\delta) - \ln(N)}{\ln(1 - \sqrt{10}N^{-2})} \left(n + \frac{1}{2}\right). \tag{3.57}$$

For the magnitude calculation example, we had $n = 6$, $N = 32$, yielding $t = 15764$ from

(3.57). We saw in practice that 10,000 to 20,000 steps gave reasonably uniform coverage over the cost/variance plane for the naive sampling approach, thereby giving us confidence in our analysis.

We can also derive a theoretical bound for this multidimensional Markov chain, and show that (3.57) is compatible with this bound. We again use the global conductance argument to estimate an upper bound on λ_2. We need to find a set of states S of the Markov chain with a large C_S and a small F_S. It is not too difficult to see that one such S is the set of states with $b_1 \leq N/2$. That is, S will contain half of the states of the chain. We partition S from $\bar{S}$ by selected for S all the states with the first index less than or equal to $N/2$. Now, C_S=1/2, and F_S will be determined by the size of the boundary between S and $\bar{S}$. The steady state distribution of being in any state is $1/N^n$, and the boundary consists of N^{n-1} states, so we obtain

$$F_S = \frac{1}{2n+1}\frac{N^{n-1}}{N^n} = \frac{1}{2n+1}\frac{1}{N}, \tag{3.58}$$

$$\Phi = \frac{2}{(2n+1)N}, \tag{3.59}$$

and finally

$$\lambda_2 \leq 1 - \frac{2}{(2n+1)^2 N^2}. \tag{3.60}$$

This of course is extremely encouraging, because the eigenvalue separation is a (low-order) polynomial function of the inverse of both n and N (rather than exponential), so we expect rapid mixing. We can check to see that the bound (3.57) is compatible with (3.60) (and perhaps tighter). From (3.56) and (3.60), we have

$$t \leq \frac{\ln(\delta) - \ln(N)}{\ln(1 - \frac{2}{(2n+1)^2 N^2})}. \tag{3.61}$$

We use the approximation $\ln(1-x) \approx x/(x-1)$ for small x to rewrite (3.61) as

$$t \leq (\ln(N) - \ln(\delta)) \left(\frac{(2n+1)^2 N^2}{2} - 1 \right). \tag{3.62}$$

We can perform similar manipulations on the time bound in (3.57) to obtain

$$t = (\ln(N) - \ln(\delta)) \left(n + \frac{1}{2} \right) \left(N^2/\sqrt{10} - 1 \right). \tag{3.63}$$

Clearly, the right-hand side of (3.63) is larger than the right-hand side of (3.62) for large enough n, so our calculated result agrees with the (loose) theoretical upper bound.

Case $n > 1$, With Constraints

As mentioned previously, there frequently are inherent constraints on the wordlengths. For example, the magnitude operation had a constraint $b_2 \leq 2b_1$ (as well as other constraints). Additionally, with the iterated Markov chain approach, we add a constraint on the achievable cost and variance. The underlying Markov chain for the problem with constraints can thus be more general than the regular lattice structure shown in Figure 3-10. Figure 3-17 gives an example Markov chain with constraints on the allowable wordlengths.

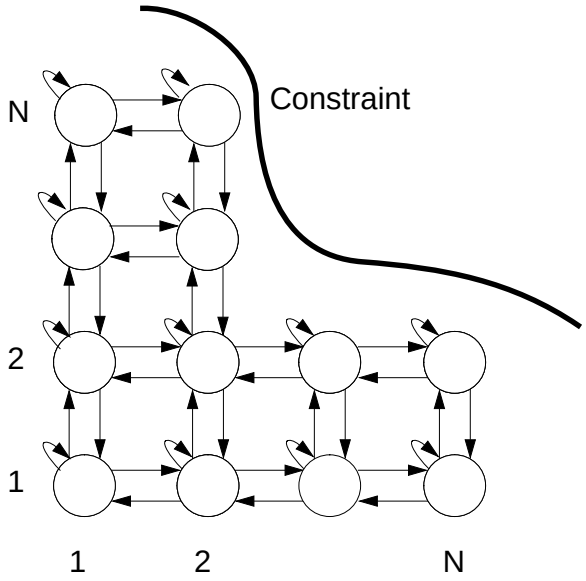

Figure 3-17: Markov chain with constraints on allocated wordlengths.

We can construct the Markov chain corresponding to the constraints as follows. Start with the n-dimensional lattice and remove nodes that are in the infeasible region. Any arcs between feasible and infeasible nodes become self-loops for the feasible node. By appropriately weighting the self-loop arcs, a uniform stationary distribution is obtained. Clearly, one would not wish to explicitly perform this construction, because it would require an exponential amount of work.

It is not guaranteed that the Markov chain so formed is connected, even for a convex feasible region. To test for connectivity requires time linear in the number of nodes, which here can be exponential time. We therefore must assume connectivity, but we cannot verify this assumption. This renders our algorithm Monte Carlo in nature. The practical effect of having a disconnected graph is that we may not discover good existing cost/variance combinations by a particular random walk.

For this situation, analyzing the number of steps required for convergence to the stationary distribution seems like it could be very complicated in general. We will therefore only make a few observations. For situations where the feasible region is convex and consists of a large number of nodes, it is reasonable to assume that most nodes are in the interior of the chain. Many reasonable hardware related constraints result in convex sets (i.e. hyperplane boundaries resulting from linear inequalities give half-space feasible sets). The number of boundary points in this situation is thus negligible in comparison to the number of interior points. This same argument led to the bound on t in (3.57) in the previous section, so (3.57) applies here as well.

For nonconvex constraints, or for problems where many constraints significantly prune the feasible region, the number of boundary points is not negligible compared to the number of interior points. About all we can hope for in this case is that the total number of feasible points is small (at least compared to the full exponential set). This would mitigate the effect of a bad asymptotic running time.

3.3.7 Simulated Annealing Approach

In the preceding random sampling approaches, we attempted to uniformly sample from the feasible region because we did not know where the optimal points were to be found.

Typically in optimization, one would actually use cost information to direct the search, resulting in a local search method. The drawback to this approach is that local searches get stuck in local minima, rather than a global minimum. A method known as *Simulated Annealing* attempts to solve this problem by occasionally allowing moves to a higher cost state, rather than always downhill. As the algorithm progresses, the allowable probability of moving uphill is slowly decreased, so that the algorithm converges to the global minimum.

Following the derivation in [21], we formulate this method as a random walk on a Markov chain. We use our nearest neighbor mesh Markov chain with feasibility constraints, as in Section 3.3.5, although we will modify the transition rules slightly. Also, we will not be seeking Pareto-optimal solutions here; rather, we shall assume a maximum allowable variance constraint and attempt to minimize the cost.

Recall from Section 3.3.5 that we would select a feasible neighbor $\mathbf{y}$ of the current state $\mathbf{x}$ with probability $1/(2n+1)$. Here, we modify this rule slightly. We perform an initial selection of $\mathbf{y}$ according to the above rule. We then test if the cost $C(\mathbf{y})$ at $\mathbf{y}$ is at least as good as the cost at $\mathbf{x}$ (i.e. $C(\mathbf{y}) \leq C(\mathbf{x})$). If so, then we take the step. If, however, the cost is higher at $\mathbf{y}$, then we move to $\mathbf{y}$ with probability

$$e^{(C(\mathbf{x})-C(\mathbf{y}))/T}, \tag{3.64}$$

and stay in $\mathbf{x}$ otherwise. T is a positive constant, usually referred to as the "temperature," in analogy with the physical annealing method used in manufacturing. T controls the probability that a step will increase the current cost. When T is large, we have a high probability that a step towards increasing cost will be allowed; in fact, at $T = \infty$, we are back to our original Markov chain of Section 3.3.5. When T is small, steps towards increasing cost are very unlikely; at $T = 0$, only downhill steps are allowed, corresponding to a conventional local search method.

Now let us examine the steady state probabilities π of this chain. Define the (constant) normalizing term

$$A = \sum_{\mathbf{z} \in F} e^{-C(\mathbf{z})/T}, \tag{3.65}$$

where F is the set of feasible states. It is shown in [21] that the steady state probabilities are given by

$$\pi(\mathbf{x}) = \frac{e^{-C(\mathbf{x})/T}}{A}. \tag{3.66}$$

For very small T, this means that if we let the algorithm run long enough, almost all of the steady state probability is concentrated in states at which the cost is minimized globally.

To avoid being stuck in a local minimum for a prolonged period, we start with a high temperature, and slowly decrease it, according to

$$T(t) = \frac{T_0}{\log t}, \tag{3.67}$$

where T_0 is the initial temperature, and t is the time step number.

To test this approach we set a variance upper bound of 10^{-8} and attempt to minimize the cost. A value $T_0 = 250$ was used to set the initial temperature. Figure 3-18 shows

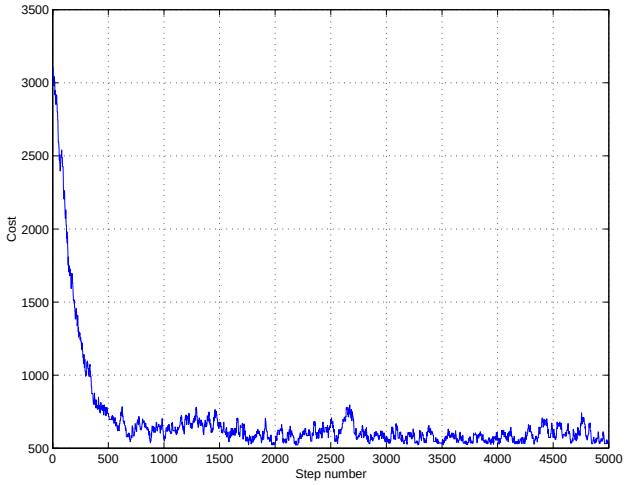

Figure 3-18: Simulated annealing cost at each time step.

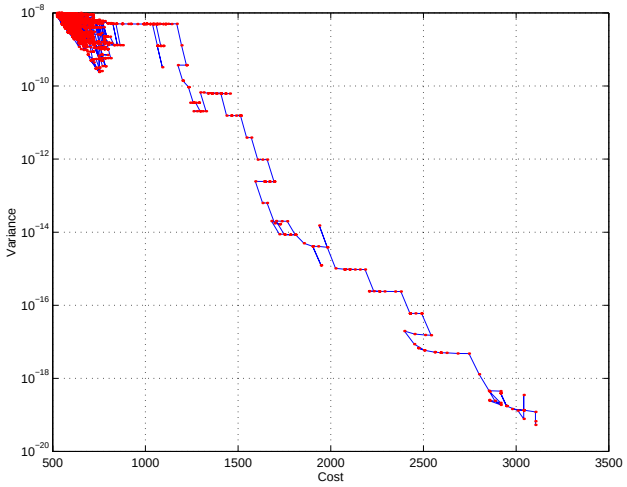

Figure 3-19: Simulated annealing cost/variance path.

the cost of the first 5000 steps of this Markov chain simulation, which clearly increases occasionally. Figure 3-19 shows the cost versus variance of these steps. The lowest cost implementation that was found over the steps was $\mathbf{b} = [12, 13, 14, 13, 14, 14]^T$. It had a cost of 523 and a variance of 9.25×10^{-9}. This is in close agreement with Figure 3-13.

3.3.8 Summary

Section 3.3 was concerned with formulating and effectively solving the wordlength design problem for algorithms where approximate calculations are allowed. The first order approximation embodied by the maximum gradient calculation in Section 3.3.1 allowed us to easily write a variance performance function $V(\mathbf{b})$. When combined with a cost function, we saw the desirability of finding Pareto-optimal designs in Section 3.3.2, and an effective algorithm for finding a lower bounding curve on these designs in Section 3.3.3. To find actual designs, a randomized search method was proposed and analyzed in Section 3.3.5. This algorithm solves the wordlength design problem that, remarkably, still confounds most

hardware designers. The algorithm can be easily tailored to incorporate various constraints that arise in real systems, such as partitioning amongst multiple devices fixed input/output wordlengths and formats, coarse granularity of available library cell wordlengths, or getting close to a fixed cost/variance target.

3.4 Reducing Bias: Lazy Rounding

We have seen that intermediate results DSP hardware frequently must be truncated or rounded to maintain reasonable wordlengths. Noise and bias are introduced into the signal due to these operations. For the addition operation, we introduce and investigate two methods which have reduced variance and bias and yet maintain the computational simplicity of truncation. Essentially, each method drives the least significant carry input of the adder string with a very simple boolean function.

3.4.1 Introduction

In this section, we mainly focus on reducing the wordlength of the addition operation. This has practical importance for FPGA-based DSP systems. For instance, in one very popular FPGA family [207], the ripple carry adder (RCA) is highly optimized. For this family, we look for methods that make use of two-input adders, rather than multirow compression [195] or merged arithmetic [176] approaches.

In the following, we first introduce and analyze two simplified rounding schemes for addition operations. The methods are then applied to parallel multiplication. This work was first introduced by Fiore [54], and expanded in [58], where the application of the method to IIR filtering and complex multiplication was also treated.

3.4.2 Lazy Rounding Schemes

For two's complement data, simply truncating the intermediate result involves the minimal amount of hardware, but it introduces a bias in the data. Rounding is unbiased (for infinite precision data) and has the same noise variance as truncation. However, rounding introduces an adder into the data path, which may be an excessive amount of hardware. For finite precision data, rounding is actually slightly biased. The so called "round-to-nearest-even" method, which is the IEEE standard [64], can be used to overcome this problem. However, in addition to an extra adder, there is zero detection logic to conditionally add another bit.

If the intermediate result to be rounded is going to be added to another value that is also going to be rounded, then there are two new methods presented here that can be applied. The first method, "lazy rounding" (LR), consists of simply ORing the most significant bits (MSBs) to be eliminated and applying this result to the carry-input of the adder chain. This is shown in Figure 3-20. Flow diagrams for all the methods considered in this paper are shown in Figure 3-21. In this and subsequent figures, truncation is denoted by "$\lfloor\ \rfloor$," and rounding is denoted by "$[\ ]$."

Note that we are assuming that the two inputs to be rounded are statistically uncorrelated, as is often done to simplify the analysis (a more detailed study of this model is given in [14]). For either truncation or rounding, the error is thus assumed to be a uniform random variable, in an interval equal in length to one bit of precision [142]. For our purposes, we assume rounding and truncation to integer values. The error due to rounding a value y, is $e_r = y - [y]$, and is assumed to have a uniform probability density function

(PDF) between $-1/2$ and $1/2$. Similarly, the error due to truncation, $e_t = y - \lfloor y \rfloor$, is also a uniform random variable between 0 and 1.

When we add two such uncorrelated random variables, the error PDF is the convolution of the two input PDFs [147]. We therefore expect triangle-shaped error PDFs for the sum of errors for simple rounding or truncation before addition. Figure 3-22 shows the error PDFs of the methods. The figure was obtained by simulation, but it is relatively straightforward to derive the PDFs analytically. This figure assumes that a large number of bits are being eliminated from the wordlength. Results will also be shown for the case where only a small number bits are being eliminated from the wordlength.

Since the spread of the PDF for the LR scheme is smaller than that of either truncation or rounding, we have hopes that the error variance is less (it is). There is a bias with this scheme, but it is 1/4 that of truncation.

The error PDF of LR in Figure 3-22 leads us to consider another new rounding scheme, dubbed "really lazy rounding" (RLR). With LR, since we are ORing two random bits, we will add a zero into the adder chain one-fourth of the time. Noting that the bias of LR is also one-fourth, we realize that if we also add a one into the chain for the $(0, 0)$ case as well, the bias will go to zero. This means we are simply forcing a one into the adder chain all the time (as shown in Figure 3-21). We can also obtain this result by simply attempting to correct for the bias of truncation. From Figure 3-22, we must add a one to remove the bias.

The error PDF for RLR is also shown in Figure 3-22. It is unbiased, and has the same variance as both rounding and truncation. Thus, if one is rounding a large number of bits, RLR can be used in place of rounding. Table 3.2 summarizes the error mean, variance, and mean square error of the four methods.

The above results must be modified slightly if we are eliminating only a small number of bits from the wordlength. This is because the error PDFs become discrete probabilities rather than continuous functions. For example, consider the error variance of LR, which asymptotically gives a signal-to-noise ratio (SNR) gain over the other methods of $10 \log_{10} \frac{1}{6} / \frac{10}{96} = 2.04$dB. Figure 3-23a shows the actual SNR gain as a function of the number of bits eliminated. As this number increases, the SNR gain approaches that of the infinite precision case. Figure 3-23b shows the error variances of the methods as a function of bits eliminated. Figure 3-23c shows the biases of the methods. It is interesting to note that in addition to having the minimum variance, the LR scheme has zero bias when exactly two bits are eliminated. Finally, Figure 3-23d shows the mean square errors of the methods. These figures give guidance to the designer who must balance bias, SNR performance, and hardware size.

Other mean/variance combinations can also be achieved by randomly utilizing more than

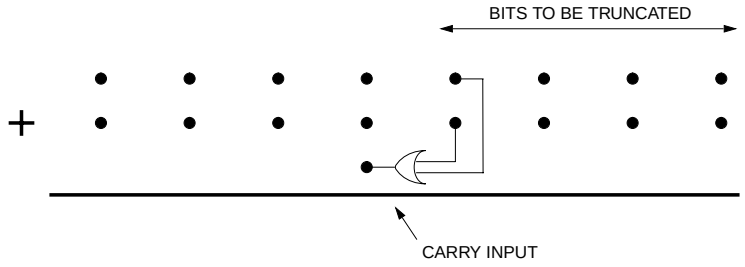

Figure 3-20: Lazy rounding for the sum of two uncorrelated numbers.

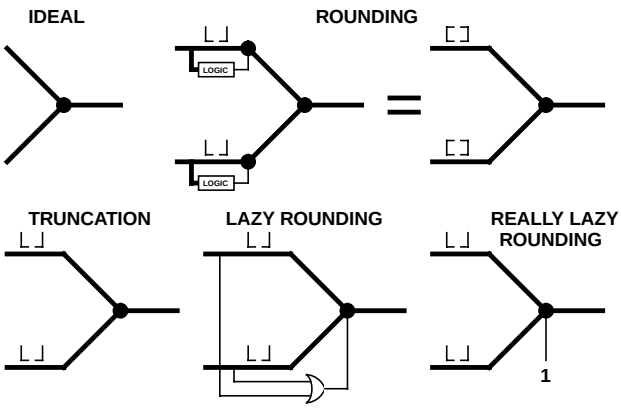

Figure 3-21: Flow diagrams for various wordlength reduction schemes. The filled dot indicates addition.

one of the methods in Table 3.2. Let the proportion of times that we perform truncation, LR, and RLR by denoted by θ_t, θ_l, and θ_r respectively, with $\theta_t + \theta_l + \theta_r = 1$. The overall mean μ and variance σ^2 are calculated from

$$\begin{bmatrix} \mu \\ \sigma^2 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & \frac{1}{4} & 0 \\ \frac{1}{6} & \frac{10}{96} & \frac{1}{6} \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \theta_t \\ \theta_l \\ \theta_r \end{bmatrix}. \tag{3.68}$$

Alternatively, desired values of μ and σ^2 can be given, and then (3.68) can be solved for the θ's. If the solution results in all of the θ's being between zero and one, then the desired values of μ and σ^2 are feasible.

We can also solve for the combination of θ_t, θ_l, and θ_r that gives the minimum mean square error, $\mu^2 + \sigma^2$. It is not difficult to show that this is achieved when $\theta_t = 0$, $\theta_l = 1/2$, and $\theta_r = 1/2$. With these values, we find that $\mu = 1/8$, $\sigma^2 = 0.1354$, and the mean square error equals 0.1510. Thus, alternating between LR and RLR achieves a lower mean square error than using either method singly.

3.4.3 LR Parallel Multiplier Design

High speed multiplication is a common signal processing operation. Many researchers have documented parallel multiplication methods [38], [121], [57]. For example, Dadda's approach [38] is to form the diamond-shaped tableaux of partial product bits, and then to

Table 3.2: Asymptotic Error Mean and Variance of Rounding Schemes.

Method	Mean	Var.	MSE	Work
Rounding	0	1/6	1/6	Extra Adder
Truncation	1	1/6	7/6	None
Lazy Rounding	1/4	10/96	1/6	OR Gate
Really Lazy Rounding	0	1/6	1/6	$C_{in} = 1$

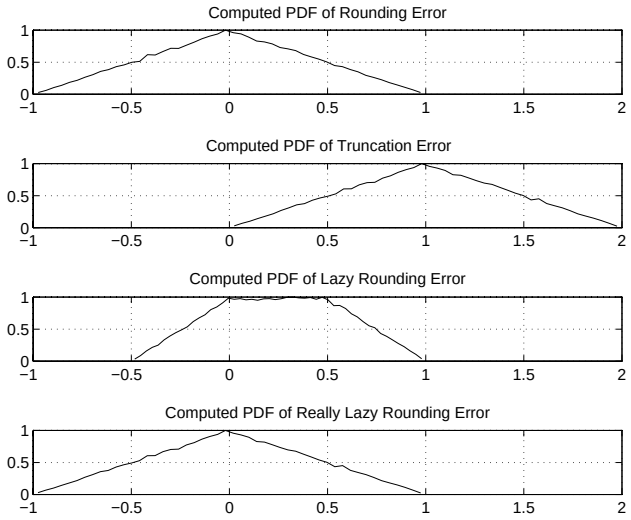

Figure 3-22: Error PDFs for wordlength reduction schemes.

successively reduce the number of rows. Dadda chooses the target number of rows after each iteration via a simple pattern. Figure 3-24 illustrates this method for the case of 12×12 unsigned multiplication. The number of full adders (FAs) and half adders (HAs) are tabulated for each stage, totaling 99 FAs and 11 HAs. After this reduction to two rows, a conventional fast-carry adder is used to produce the final product.

Considerable savings in hardware results if we ignore some of the low order bits in the initial tableaux. Of course, this approximation introduces some error, which we propose to reduce via LR. Figure 3-25 shows the initial tableaux and subsequent reduction steps that result from the truncation of all bits beyond the twelfth column. Now only 55 FAs and 7 HAs are required for this reduced Dadda scheme. This approach can also be applied to the more modern Compressor multiplication schemes [195].

There has been previous work with these truncated multipliers [116], [161]. Both of these previous approaches truncate lower order bit columns. These methods correct for the bias that is introduced by precalculating and adding in a correction constant. The LR method presented here implicitly introduces a correction constant, without adding an extra row to the tableaux.

The simple truncated multiplier approach of Figure 3-25 was compared against the LR approach. Figure 3-26 shows a comparison of the error for 1000 uniform random multiplication pairs. To generate this truncation error histogram, all product bits below the twelfth MSB were dropped. Recall that we are motivated to seek methods that use the optimized RCA of [207]. To this end, an LR approach that examined all the product bits in the thirteenth column and combined them pairwise was also tested, as shown in Figure 3-26. Clearly, both the error mean and variance are reduced by the LR method.

The performance of the two approaches as more bits are retained is shown in Figure 3-27. Figure 3-27a compares the error means of the two methods as the number of bits increases. Figure 3-27b compares the variances of the rounding noise of the two schemes. The figure shows that the LR approach introduces much less noise when relatively few multiplier product bits are used. The benefit decreases as a larger percentage of the bits are used.

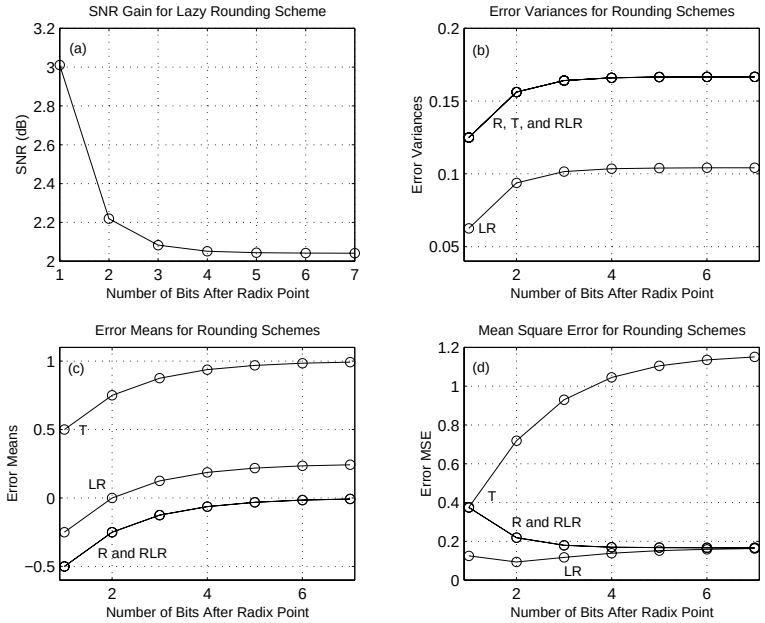

Figure 3-23: Performance of wordlength reduction schemes.

3.4.4 Summary

In this section, two lazy rounding schemes were presented that allow the designer to reduce noise variance or bias. The methods can be applied widely in situations where one must prune the wordlength of internal computations to maintain reasonable hardware complexity. Random combinations of the two methods lead to a continuum of methods with differing bias/variance properties. For truncated multiplication, further work comparing the lazy rounding approaches to the correction constant approaches has been performed by Fiore [58].

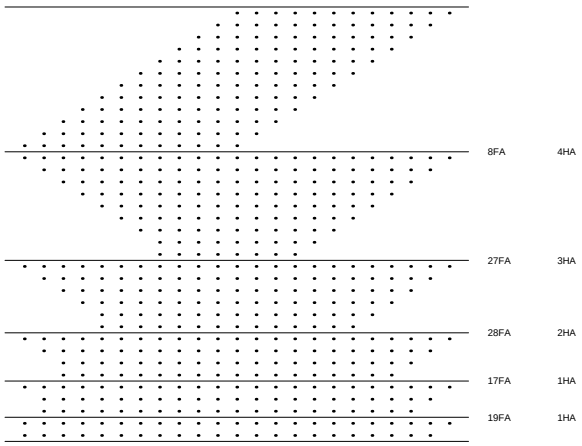

Figure 3-24: Dadda's scheme for partial product reduction (12×12 multiplication).

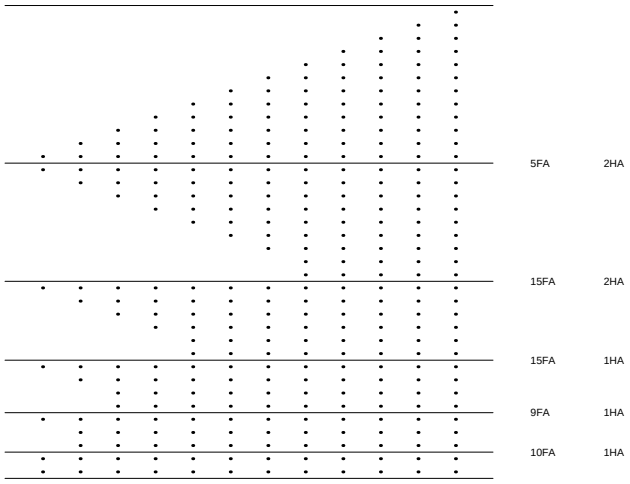

Figure 3-25: 12×12 multiplication with lower bits truncated.

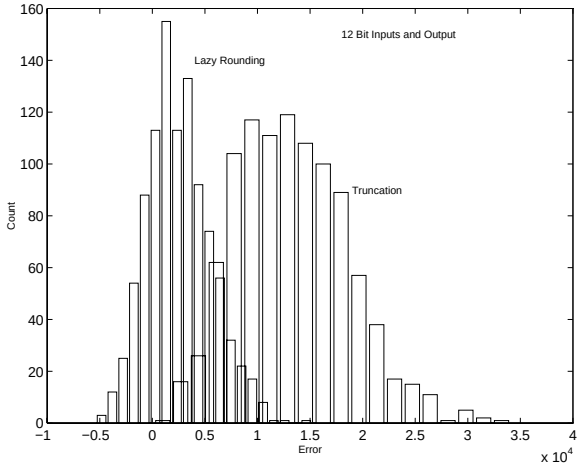

Figure 3-26: Output error of lazy rounding vs. truncation for parallel multiplier.

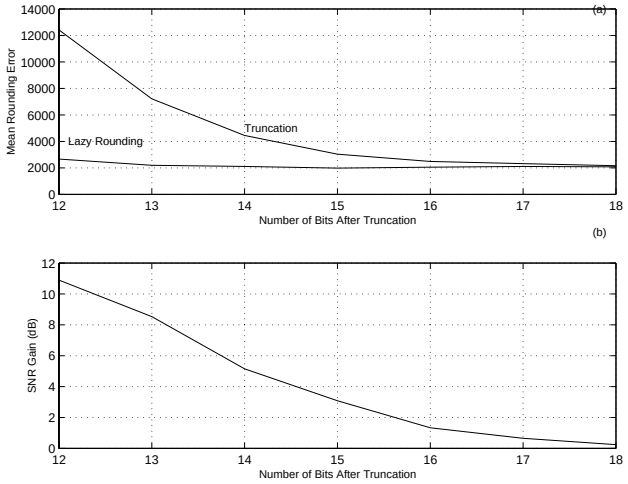

Figure 3-27: Performance for varying number of truncating bits and lazy rounding bits (12-bit inputs).

3.5 Reducing Multiplications: Efficient Computation of Multiple Bilinear Forms

3.5.1 Introduction

For many computational tasks, we seek to reduce the number of multiplications. This is especially important in custom hardware implementations, because multiplications can be costly in terms of gate count. Much work has been done in the past on reducing the number of multiplications for convolutions, fast Fourier transforms (FFT), and matrix multiplication. All of these problems are examples of efficiently evaluating multiple bilinear forms of indeterminants $\mathbf{q}$ and $\mathbf{p}$:

$$\begin{aligned} &\mathbf{q}^T\mathbf{A}_1\mathbf{p}, \\ &\vdots \\ &\mathbf{q}^T\mathbf{A}_m\mathbf{p}, \end{aligned} \tag{3.69}$$

where $\mathbf{q}$ and $\mathbf{p}$ are column vectors, and the $\mathbf{A}_i$ are appropriately sized, known, constant matrices. In this section, we will review what little is known about this problem, offer a new, heuristic algorithm to design low multiplicative-complexity algorithms, and then apply it to some common computer vision examples.

Winograd [203] observed that the canonical decomposition of the trilinear tensor A_{ijk} formed from the matrices $\mathbf{A}_i$ is in fact a minimal multiplication algorithm to evaluate the bilinear forms (3.69):

$$T = \sum_i \sum_j \sum_k A_{ijk} q_i p_j z_k = \sum_{m=1}^{r} l_q^m(\mathbf{q}) l_p^m(\mathbf{p}) l_z^m(\mathbf{z}), \tag{3.70}$$

where l_q^m, l_p^m, and l_z^m are linear operators. Here, r is the rank of the tensor, and is equal to the minimum number of multiplications possible. Unfortunately, we do not know a method to determine this canonical form. If such an approach can be found, then this will provide a clear, constructive answer to the minimal multiplication problem. (Comon and Mourrain [33] provide related information about decomposing totally symmetric tensors. Even this related problem is difficult.)

When there is only one bilinear form to calculate, we shall show that the optimal decomposition is easy to compute. With more than one form, there are only two special cases where closed form solutions are known. The first special case is when there are two symmetric bilinear forms, and one of them is positive definite; an algorithm to perform this is given in [68]. The second special case is found in the field of blind system identification, which has produced an algorithm known as *joint diagonalization* [40], [114], [190]. This can produce the desired decomposition (3.70), but only under very special circumstances; the rank r must be no larger than the minimum dimension of the $\mathbf{A}_i$'s.

We note in passing that we can easily derive a constructive algorithm that gives an upper bound on the number of multiplications required. Let $\mathbf{q}$ be $q \times 1$, and $\mathbf{p}$ be $p \times 1$. We have

$$\begin{aligned} \mathbf{q}^T\mathbf{A}_i\mathbf{p} &= \mathrm{vec}(\mathbf{q}^T\mathbf{A}_i\mathbf{p}) \\ &= (\mathbf{p} \otimes \mathbf{q})^T \mathrm{vec}(\mathbf{A}_i) \\ &= \mathrm{vec}^T(\mathbf{A}_i)(\mathbf{p} \otimes \mathbf{q}), \end{aligned} \tag{3.71}$$

where "$\otimes$" is the Kronecker product (the identity used in this equation is given in [26], [210]). By stacking all the row vectors $\mathrm{vec}^T(\mathbf{A}_i)$ into a single matrix $\mathbf{A}$, we have constructed

an algorithm that uses pq multiplications of indeterminants (i.e. form $\mathbf{p} \otimes \mathbf{q}$ using pq multiplications, then premultiply this vector by $\mathbf{A}$). For $m >> pq$, could be a reasonable approach; for much smaller m, this is generally inefficient.

In this section, we show how to use the more familiar singular value decomposition (SVD) algorithm to find good implementations. We treat the case of a single bilinear form separately, because it is much simpler than the general case. For the multiple bilinear form problem, we introduce a set of matrices which can be represented by a small set of outer products. The size of the set is in fact the number of multiplications needed by the resulting efficient algorithm.

A word is in order about how we count multiplications. Multiplication of an indeterminant by a constant is not counted. The motivation for this is that multiplying by, say, a power of 2, consumes no hardware resources. Multiplication by many other numbers may also be efficiently implemented by additions. Since additions generally require much fewer resources than multiplications, we can consider additions to be "free." Of course, care must be used if the precision of the constants are excessive. Then, the additive complexity may be quite large. Unfortunately, our algorithm, and indeed none of the algorithms for convolutions, FFTs, and matrix multiplications to date, take the complexity of additions into account. The only recourse we have then is to examine the final efficient algorithm for high precision constants, and make a determination if the additive complexity is low enough.

We also note that all of the resulting efficient algorithms described in this paper will work for matrices in place of the scalar indeterminant components of $\mathbf{q}$ and $\mathbf{p}$. The matrices of course must be compatibly sized, and the ordering of the indeterminants must not be swapped (because matrix multiplication is not commutative).

3.5.2 Single Bilinear Form

The procedure to efficiently evaluate a single bilinear form is sufficiently less complicated than for multiple forms that we treat it separately. It also provides some insight for the multiple form problem.

For a single bilinear form $\mathbf{q}^T\mathbf{A}\mathbf{p}$, the minimal number of multiplications may be found simply from $\mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V},^T$ the SVD of $\mathbf{A}$. Let the rank of $\mathbf{A}$ be r, so that

$$\mathbf{q}^T\mathbf{A}\mathbf{p} = \sum_{j=1}^{r} \sigma_j \left(\mathbf{q}^T\mathbf{u}_j\right)\left(\mathbf{v}_j^T\mathbf{p}\right). \tag{3.72}$$

This is in the correct "add-multiply-add" form for minimal complexity bilinear algorithms [22], [203], and r is the minimal number of multiplications. For example, if we have the bilinear form

$$F = 3q_1p_1 + q_1p_2 + q_2p_1 + 3q_2p_2, \tag{3.73}$$

4 multiplications would be required for a direct implementation. We write this as

$$F = \mathbf{q}^T\mathbf{A}\mathbf{p}, \tag{3.74}$$

with

$$\mathbf{A} = \begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \tag{3.75}$$

We thus write

$$F = 2(q_1 + q_2)(p_1 + p_2) + (q_1 - q_2)(p_1 - p_2), \tag{3.76}$$

which only requires $r = 2$ multiplications.

If we directly use the SVD, we have the decomposition

$$\begin{aligned} F &= 4(0.7071q_1 + 0.7071q_2)(0.7071p_1 + 0.7071p_2) \\ &\quad +2(-0.7071q_1 + 0.7071q_2)(-0.7071p_1 + 0.7071p_2), \end{aligned} \tag{3.77}$$

which requires only 2 multiplications of indeterminants, but challenges our notion of not counting multiplications by known constants. For this particular example, it is easy to transform from (3.77) to (3.76) by visual inspection. For larger problems, however, the transformation may not be so obvious. Fortunately, programs such as MATLAB have the ability to produce a "rational" basis for a subspace. For example, with

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{bmatrix}, \tag{3.78}$$

we have

$$\mathbf{q}^T\mathbf{A}\mathbf{p} = 2(q_1 + q_2)(p_1 + p_2 + p_3) - (q_1 + q_2)(p_1 - p_3) \tag{3.79}$$

using a rational basis for the various subspaces.

3.5.3 Multiple Bilinear Forms

We will work with a standard form $\mathbf{Q}\mathbf{p}$, where the elements of the matrix $\mathbf{Q}$ are linear combinations of the components of $\mathbf{q} = [q_1, \ldots, q_k]^T$. We need not consider linear combinations of the components of $\mathbf{p}$, because it is easy to reduce such a problem to the standard form. For example,

$$\begin{aligned} \begin{bmatrix} q_1 & q_2 \\ q_1 + q_2 & q_3 \end{bmatrix} \begin{bmatrix} p_1 + p_2 \\ p_1 - p_2 \end{bmatrix} &= \begin{bmatrix} q_1 & q_2 \\ q_1 + q_2 & q_3 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} \\ &= \begin{bmatrix} q_1 + q_2 & q_1 - q_2 \\ q_1 + q_2 + q_3 & q_1 + q_2 - q_3 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \end{bmatrix}. \end{aligned} \tag{3.80}$$

We may also transform more general problems $\mathbf{Q}\mathbf{P}$ into the standard form. Here, $\mathbf{P}$ is a matrix of linear combinations of the elements of $\mathbf{p}$. The required transformation in this case is to work with the matrix-vector product:

$$\begin{bmatrix} \mathbf{Q} & & \\ & \ddots & \\ & & \mathbf{Q} \end{bmatrix} \begin{bmatrix} \mathbf{P}_1 \\ \vdots \\ \mathbf{P}_n \end{bmatrix}, \tag{3.81}$$

where $\mathbf{P}_i$ is the i^{th} column of the matrix $\mathbf{P}$. If necessary, we then perform a transformation as in (3.80).

We thus desire an efficient method to compute $\mathbf{Q}\mathbf{p}$. Consider premultiplying by a series of vectors $\mathbf{r}^{(i)}$,

$$\mathbf{r}^{(i)T}\mathbf{Q}\mathbf{p} = \mathbf{q}^T\mathbf{R}^{(i)}\mathbf{p}, \quad i = 1, \ldots, m, \tag{3.82}$$

where the matrix $\mathbf{R}^{(i)}$ is found from the elements of $\mathbf{r}^{(i)}$. We desire to represent the matrices $\mathbf{R}^{(i)}$ as weighted sums of common outer products

$$\mathbf{R}^{(i)} = \sum_{j=1}^{J} \sigma_j^{(i)} \mathbf{u}_j \mathbf{v}_j^T, \quad i = 1, \ldots, m, \tag{3.83}$$

where the size J of the sets of vectors $\{\mathbf{u}_j\}$ and $\{\mathbf{v}_j\}$ is to be minimized (thus minimizing the number of multiplications). We therefore have

$$\mathbf{q}^T\mathbf{R}^{(i)}\mathbf{p} = \sum_{j=1}^{k_i} \sigma_j^{(i)} \left(\mathbf{q}^T\mathbf{u}_j\right)\left(\mathbf{v}_j^T\mathbf{p}\right). \tag{3.84}$$

Stacking the quadratic forms in (3.84) into a vector and factoring, we have

$$\begin{bmatrix} \mathbf{q}^T\mathbf{R}^{(1)}\mathbf{p} \\ \vdots \\ \mathbf{q}^T\mathbf{R}^{(m)}\mathbf{p} \end{bmatrix} = \begin{bmatrix} \sigma_1^{(1)} & \cdots & \sigma_n^{(1)} \\ \vdots & & \vdots \\ \sigma_1^{(m)} & \cdots & \sigma_n^{(m)} \end{bmatrix} \begin{bmatrix} \left(\mathbf{q}^T\mathbf{u}_1\right)\left(\mathbf{v}_1^T\mathbf{p}\right) \\ \vdots \\ \left(\mathbf{q}^T\mathbf{u}_n\right)\left(\mathbf{v}_n^T\mathbf{p}\right) \end{bmatrix}. \tag{3.85}$$

The vector of bilinear forms may be rewritten as

$$\begin{bmatrix} \mathbf{q}^T\mathbf{R}^{(1)}\mathbf{p} \\ \vdots \\ \mathbf{q}^T\mathbf{R}^{(m)}\mathbf{p} \end{bmatrix} = \begin{bmatrix} \mathbf{r}^{(1)T}\mathbf{Q}\mathbf{p} \\ \vdots \\ \mathbf{r}^{(m)T}\mathbf{Q}\mathbf{p} \end{bmatrix} = \begin{bmatrix} \mathbf{r}^{(1)T} \\ \vdots \\ \mathbf{r}^{(m)T} \end{bmatrix} \mathbf{Q}\mathbf{p} \stackrel{\triangle}{=} \tilde{\mathbf{R}}\mathbf{Q}\mathbf{p}. \tag{3.86}$$

As long as we choose the $\mathbf{r}^{(i)}$ so that $\tilde{\mathbf{R}}$ is invertible, we can combine (3.86) with (3.85) to obtain

$$\mathbf{Q}\mathbf{p} = \tilde{\mathbf{R}}^{-1} \begin{bmatrix} \sigma_1^{(1)} & \cdots & \sigma_n^{(1)} \\ \vdots & & \vdots \\ \sigma_1^{(m)} & \cdots & \sigma_n^{(m)} \end{bmatrix} \begin{bmatrix} \left(\mathbf{q}^T\mathbf{u}_1\right)\left(\mathbf{v}_1^T\mathbf{p}\right) \\ \vdots \\ \left(\mathbf{q}^T\mathbf{u}_n\right)\left(\mathbf{v}_n^T\mathbf{p}\right) \end{bmatrix}, \tag{3.87}$$

which is in the required "add-multiply-add" form, with the minimal number of multiplications.

We now address the central problem of finding the set of vectors $\{\mathbf{u}_j\}$ and $\{\mathbf{v}_j\}$. Our approach is to look for outer products that reduce the rank of one or more of the $\mathbf{R}^{(i)}$. Preferably, if a single outer product can reduce the rank of many of the $\mathbf{R}^{(i)}$, then large savings in multiplications may be achieved. Clearly, greedy approaches that try to find the largest subset, or perhaps approaches that randomly choose subsets, may be applied. Once a suitable outer product is found, we subtract scaled versions of it from the appropriate subset of the $\mathbf{R}^{(i)}$, thus reducing their rank. The procedure is repeated, resulting in a net reduction of the total rank of the entire set of modified $\mathbf{R}^{(i)}$. When all the modified $\mathbf{R}^{(i)}$ are zero matrices, the algorithm terminates.

The final multiplication count of this method depends on the ordering that we combine outputs. It also depends on the starting matrix $\tilde{\mathbf{R}}$. Thus, several repeated trials of the algorithm may be needed to find good results. Unfortunately, there is no guarantee that a given result will be optimal. Also, the method is plagued by roundoff error, so that one must set to zero any small numbers that arise during the calculations. Care must be used if the bilinear forms contain anything other than reasonably small integers.

Hand-Calculation Examples

We now consider a small example. Let

$$\begin{bmatrix} e \\ f \end{bmatrix} = \begin{bmatrix} q_1 & q_2 \\ q_1 + q_2 & q_1 - q_2 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \end{bmatrix}. \tag{3.88}$$

We have

$$\begin{bmatrix} r_1 & r_2 \end{bmatrix} \begin{bmatrix} q_1 & q_2 \\ q_1+q_2 & q_1-q_2 \end{bmatrix} = \begin{bmatrix} q_1 & q_2 \end{bmatrix} \begin{bmatrix} r_1+r_2 & r_2 \\ r_2 & r_1-r_2 \end{bmatrix}. \tag{3.89}$$

Setting $\mathbf{r}^{(1)} = [1\ 0]^T$ leads to

$$\mathbf{R}^{(1)} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}^T = \mathbf{U}^{(1)}\mathbf{S}^{(1)}\mathbf{V}^{(1)T}. \tag{3.90}$$

Similarly, setting $\mathbf{r}^{(2)} = [0\ 1]^T$ leads to

$$\mathbf{R}^{(2)} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}^T = \mathbf{U}^{(2)}\mathbf{S}^{(2)}\mathbf{V}^{(2)T}. \tag{3.91}$$

Since both $\mathbf{R}^{(1)}$ and $\mathbf{R}^{(2)}$ are rank 2, we look for a common rank-reducing outer product. Our inspired choice is

$$\mathbf{u}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad \mathbf{v}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \tag{3.92}$$

so that

$$\mathbf{u}_1\mathbf{v}_1^T = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \tag{3.93}$$

We subtract off appropriate multiples of this from the $\mathbf{R}^{(i)}$:

$$\mathbf{R}^{(1)} - \mathbf{u}_1\mathbf{v}_1^T = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \end{bmatrix}, \tag{3.94}$$

and

$$\mathbf{R}^{(2)} - 2\mathbf{u}_1\mathbf{v}_1^T = \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \begin{bmatrix} -1 & 1 \end{bmatrix}. \tag{3.95}$$

We therefore have

$$\mathbf{e} = \left(\begin{bmatrix} q_1 & q_2 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right) \left(\begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} \right) + \left(\begin{bmatrix} q_1 & q_2 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right) \left(\begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} \right), \tag{3.96}$$

and

$$\mathbf{f} = \left(2 \begin{bmatrix} q_1 & q_2 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right) \left(\begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} \right) + \left(\begin{bmatrix} q_1 & q_2 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \end{bmatrix} \right) \left(\begin{bmatrix} -1 & 1 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} \right). \tag{3.97}$$

This can be written more simply as

$$\begin{array}{lll} e & = q_1p_1 & +q_2p_2 \\ f & = 2q_1p_1 & +(q_1-q_2)(p_1-p_2), \end{array} \tag{3.98}$$

which has only 3 unique multiplication.

For another small example, consider

$$\begin{bmatrix} e \\ f \\ g \end{bmatrix} = \begin{bmatrix} q_1+q_2 & q_1-q_2 \\ q_1 & 2q_1 \\ q_2 & q_1-2q_2 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \end{bmatrix}. \tag{3.99}$$

We have

$$\begin{bmatrix} r_1 & r_2 & r_3 \end{bmatrix} \begin{bmatrix} q_1+q_2 & q_1-q_2 \\ q_1 & 2q_1 \\ q_2 & q_1-2q_2 \end{bmatrix} = \begin{bmatrix} q_1 & q_2 \end{bmatrix} \begin{bmatrix} r_1+r_2 & r_1+2r_2+r_3 \\ r_1+r_3 & -r_1-2r_3 \end{bmatrix}. \quad (3.100)$$

Setting $\mathbf{r}^{(1)} = [1\ 0\ 0]^T$ leads to

$$\mathbf{R}^{(1)} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \quad (3.101)$$

Similarly, setting $\mathbf{r}^{(2)} = [0\ 1\ 0]^T$ leads to

$$\mathbf{R}^{(2)} = \begin{bmatrix} 1 & 2 \\ 0 & 0 \end{bmatrix}. \quad (3.102)$$

Finally, setting $\mathbf{r}^{(3)} = [0\ 0\ 1]^T$ leads to

$$\mathbf{R}^{(3)} = \begin{bmatrix} 0 & 1 \\ 1 & -2 \end{bmatrix}. \quad (3.103)$$

Since $\mathbf{R}^{(2)}$ is rank 1, we choose its singular vectors as the covering vectors. We need to determine multiples of the outer product to subtract from $\mathbf{R}^{(1)}$ and $\mathbf{R}^{(2)}$, so that their rank is reduced from 2 to 1. We therefore seek

$$\mathbf{R}^{(1)} - \alpha \mathbf{u}_1 \mathbf{v}_1^T = \begin{bmatrix} 1-\alpha & 1-2\alpha \\ 1 & -1 \end{bmatrix} \quad (3.104)$$

to have low rank. Equivalently, we desire the determinant to equal zero, or

$$0 = (1-\alpha)(-1) - (1-2\alpha)(1), \quad (3.105)$$

which is solved by $\alpha = 2/3$. We solve a similar equation to reduce the rank of $\mathbf{R}^{(3)}$:

$$\mathbf{R}^{(3)} - \beta \mathbf{u}_1 \mathbf{v}_1^T = \begin{bmatrix} -\beta & 1-2\beta \\ 1 & -2. \end{bmatrix}. \quad (3.106)$$

Setting $\beta = 1/4$ drops the rank. With these settings of α and β, we have

$$\mathbf{R}^{(1)} - \frac{2}{3}\mathbf{u}_1 \mathbf{v}_1^T = \begin{bmatrix} 1/3 & -1/3 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 1/3 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & -1 \end{bmatrix} \quad (3.107)$$

and

$$\mathbf{R}^{(3)} - \frac{1}{4}\mathbf{u}_1 \mathbf{v}_1^T = \begin{bmatrix} -1/4 & 1/2 \\ 1 & -2 \end{bmatrix} = \begin{bmatrix} -1/4 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & -2 \end{bmatrix}. \quad (3.108)$$

We can therefore write the equations for e, f, and g as

$$\begin{array}{lllll} e & = \mathbf{q}^T \mathbf{R}^{(1)} \mathbf{p} & = & \frac{2}{3} q_1 (p_1 + 2p_2) & + (\frac{1}{3} q_1 + q_2)(p_1 - p_2) \\ f & = \mathbf{q}^T \mathbf{R}^{(2)} \mathbf{p} & = & q_1 (p_1 + 2p_2) & \\ g & = \mathbf{q}^T \mathbf{R}^{(3)} \mathbf{p} & = & \frac{1}{4} q_1 (p_1 + 2p_2) & + (-\frac{1}{4} q_1 + q_2)(p_1 - 2p_2), \end{array} \quad (3.109)$$

which has only 3 unique multiplications.

3.5.4 Recovery of the Common Rank-Reducing Outer Product

We are given a subset of the matrices $\mathbf{R}^{(1)}, \ldots, \mathbf{R}^{(m)}$, and in this section we relabel the subset as $\mathbf{R}^{(1)}, \ldots, \mathbf{R}^{(l)}$ for convenience. We seek vectors $\mathbf{u}$ and $\mathbf{v}$ such that each of

$$\mathbf{R}^{(i)} - \alpha_i \mathbf{u}\mathbf{v}^T, \quad i = 1, \ldots, l \tag{3.110}$$

is of lower rank than their corresponding $\mathbf{R}^{(i)}$. Let $\mathbf{U}^{(i)}\mathbf{S}^{(i)}\mathbf{V}^{(i)T} = \mathbf{R}^{(i)}$ be the SVDs, where we assume that columns of $\mathbf{U}^{(i)}$, $\mathbf{S}^{(i)}$, and $\mathbf{V}^{(i)}$ corresponding to zero singular values have been removed. That is, the $\mathbf{S}^{(i)}$ are square, and $\mathbf{U}^{(i)}$ and $\mathbf{V}^{(i)}$ are at least as tall as they are wide.

We have

$$\begin{aligned} \mathbf{R}^{(i)} - \alpha_i \mathbf{u}\mathbf{v}^T &= \mathbf{U}^{(i)}\mathbf{S}^{(i)}\mathbf{V}^{(i)T} - \alpha_i \mathbf{U}^{(i)}\delta_1^{(i)}\delta_2^{(i)T}\mathbf{V}^{(i)T} \\ &= \mathbf{U}^{(i)}\left(\mathbf{S}^{(i)} - \alpha_i \delta_1^{(i)}\delta_2^{(i)T}\right)\mathbf{V}^{(i)T}, \end{aligned} \tag{3.111}$$

where

$$\begin{aligned} \mathbf{u} &= \mathbf{U}^{(i)}\delta_1^{(i)}, \\ \mathbf{v} &= \mathbf{V}^{(i)}\delta_2^{(i)}, \end{aligned} \tag{3.112}$$

for all $i = 1, \ldots, l$. This last equation states that $\mathbf{u}$ is in the span of all the $\mathbf{U}^{(i)}$ and $\mathbf{v}$ is in the span of all the $\mathbf{V}^{(i)}$, which is a necessary condition. We thus require each of

$$\mathbf{S}^{(i)} - \alpha_i \delta_1^{(i)}\delta_2^{(i)T} \tag{3.113}$$

to be lower rank than their corresponding (full rank, diagonal) matrix $\mathbf{S}^{(i)}$. For simplicity, we choose $\alpha_1 = 1$, $\delta_1^{(i)} = [1\ 0\ \ldots\ 0]^T$ and $\delta_2^{(1)}(1) = 1$. Clearly, the rank of $\mathbf{S}^{(1)}$ is lowered for arbitrary choice of the remaining elements of $\delta_2^{(1)}$.

We must find all the $\delta_2^{(i)}$ for $i = 2, \ldots, l$. These can not be chosen arbitrarily, due to the relationship (3.112). Using (3.112) with $i = 1$, we modify (3.113) as

$$\begin{aligned} &\mathbf{S}^{(i)} - \alpha_i \delta_1^{(i)}\delta_2^{(i)T} \\ &= \mathbf{S}^{(i)} - \alpha_i \left(\mathbf{U}^{(i)T}\mathbf{U}^{(1)}\delta_1^{(1)}\right)\left(\mathbf{V}^{(i)T}\mathbf{V}^{(1)}\delta_2^{(1)T}\right). \end{aligned} \tag{3.114}$$

Reducing the rank of (3.114) is equivalent to reducing the rank of

$$\left(\mathbf{U}^{(i)T}\mathbf{U}^{(1)}\right)^{-1}\mathbf{S}^{(i)}\left(\mathbf{V}^{(1)T}\mathbf{V}^{(i)}\right)^{-1} - \alpha_i \delta_1^{(1)}\delta_2^{(1)T}, \tag{3.115}$$

assuming the inverses exist. The inverses may fail to exist if, for example, $\mathbf{U}^{(1)} \perp \mathbf{U}^{(i)}$. In this case, there can be no vector $\mathbf{u}$ common to all linear spans of the $\mathbf{U}^{(i)}$, so the algorithm terminates.

We therefore now assume that the inverses in (3.115) exist. Let

$$\mathbf{W}^{(i)} \triangleq \left(\mathbf{U}^{(i)T}\mathbf{U}^{(1)}\right)^{-1}\mathbf{S}^{(i)}\left(\mathbf{V}^{(1)T}\mathbf{V}^{(i)}\right)^{-1}, \tag{3.116}$$

which must be full rank by our previous comments. By construction, the rank one matrix $\delta_1^{(1)}\delta_2^{(1)T}$ has nonzero elements only in the first row. Thus, the determinant of $\mathbf{W}^{(i)} - \alpha_i \delta_1^{(1)}\delta_2^{(1)T}$ is an affine combination of the unknown elements in $\delta_2^{(1)T}$, times α_i. The coefficients multiplying the elements of $\delta_2^{(1)T}$ are simply the determinant cofactors of the $\mathbf{W}^{(i)}$.

More explicitly, we require the determinant $|\mathbf{W}^{(i)} - \alpha_i \delta_1^{(1)} \delta_2^{(1)T}| = 0$. We have

$$\begin{aligned} |\mathbf{W}^{(i)} - \alpha_i \delta_1^{(1)} \delta_2^{(1)T}| &= \begin{vmatrix} W_{1,1} - \alpha_i & W_{1,2} - \alpha_i \delta_2^{(1)}(2) & \cdots \\ W_{2,1} & W_{2,2} & \cdots \\ \vdots & \vdots & \end{vmatrix} \\ &= \textstyle\sum_k (-1)^{k+1} c_k^{(i)} \left(W_{1,k} - \alpha_i \delta_2^{(1)}(k) \right) = 0, \end{aligned} \tag{3.117}$$

where $c_k^{(i)}$ is the cofactor of $W_{1,k}^{(i)}$ (we merely calculate and use $c_k^{(i)}$ in what follows). Rearrange (3.117) as

$$\alpha_i \sum_k (-1)^{k+1} c_k^{(i)} \delta_2^{(1)}(k) = \sum_k (-1)^{k+1} c_k^{(i)} W_{1,k}, \tag{3.118}$$

or

$$\alpha_i \left(c_1^{(i)} + \sum_{k>1} (-1)^{k+1} c_k^{(i)} \delta_2^{(1)}(k) \right) = \sum_k (-1)^{k+1} c_k^{(i)} W_{1,k}, \tag{3.119}$$

because we set $\delta_2^{(1)}(1) = 1$. If the right-hand side $\sum_k (-1)^{k+1} c_k^{(i)} W_{1,k}$ equals zero, then either $\alpha_i = 0$, or $c_1^{(i)} + \sum_{k>1} (-1)^{k+1} c_k^{(i)} \delta_2^{(1)}(k) = 0$ (or both). The solution $\alpha_i = 0$ is unacceptable, because $\mathbf{R}^{(i)}$ will not be altered. We thus assume $\alpha_i \neq 0$ in what follows, but we need to check that this is true at the very end of the algorithm. Thus, if $\sum_k (-1)^{k+1} c_k^{(i)} W_{1,k} = 0$, we contribute the linear equation

$$c_1^{(i)} + \sum_{k>1} (-1)^{k+1} c_k^{(i)} \delta_2^{(1)}(k) = 0, \tag{3.120}$$

to a set of linear equations. If $\sum_k (-1)^{k+1} c_k^{(i)} W_{1,k}$ is not zero, then we rearrange (3.119) as

$$c_1^{(i)} + \sum_{k>1} (-1)^{k+1} c_k^{(i)} \delta_2^{(1)}(k) = \left(\sum_k (-1)^{k+1} c_k^{(i)} W_{1,k} \right) \frac{1}{\alpha_i}, \tag{3.121}$$

or finally,

$$\sum_{k>1} (-1)^{k+1} c_k^{(i)} \delta_2^{(1)}(k) - \left(\sum_k (-1)^{k+1} c_k^{(i)} W_{1,k} \right) \frac{1}{\alpha_i} = -c_1^{(i)}, \tag{3.122}$$

which is linear in $\delta_2^{(1)}(k)$ and $1/\alpha_i$. Therefore, when $\sum_k (-1)^{k+1} c_k^{(i)} W_{1,k}$ is not zero, we contribute one linear equation of the form (3.122).

Due to the presence of the α_i, the set of linear equations will generally be underdetermined. If they are consistent, then there are a multitude of possible solutions for $\delta_2^{(1)}$ and the α_i. Usually, a pseudoinverse approach is used to solve for the minimum norm solution. In this case, however, the minimum norm solution is not appropriate, because it could lead to extremely large values of the α_i. Since we really just want *a* solution, our *ad hoc* approach is to set the α_i in (3.122) equal to one. Clearly, other approaches are possible that improve the chances that the equations will be consistent.

Once we have $\delta_2^{(1)T}$ chosen, the proper determination of the α_i is a simple generalized eigenvalue problem. We use $\delta_1^{(1)T}$ and the calculated $\delta_2^{(1)T}$ to obtain $\mathbf{u}$ and $\mathbf{v}$, then seek all α_i such that

$$\mathbf{R}^{(i)} - \alpha_i \mathbf{u}\mathbf{v}^T, \quad i = 1, \ldots, l \tag{3.123}$$

is reduced rank. In other words, we seek α_i such that the equations

$$\left(\mathbf{R}^{(i)} - \alpha_i \mathbf{u}\mathbf{v}^T\right)\mathbf{x}^{(i)} = \mathbf{0} \tag{3.124}$$

each have solutions. Rearranging,

$$\mathbf{R}^{(i)}\mathbf{x}^{(i)} = \alpha_i \left(\mathbf{u}\mathbf{v}^T\right)\mathbf{x}^{(i)}, \tag{3.125}$$

which is simply a generalized eigenvalue calculation. Therefore, given $\mathbf{u}$ and $\mathbf{v}$, to determine the correct α_i, solve each of the generalized eigenvalue problems (3.125) for useful α_i. Note that we must reject any $\alpha_i = 0$, and perhaps complex α_i. We also must reject any extremely small or large α_i because these most likely result from roundoff errors. If we can not find usable α_i for all equations, then the method concludes that there is no common rank-reducing outer product for the set $\mathbf{R}^{(i)}$.

3.5.5 Complex Multiplication Experiment

Recall that calculating $(e + jf) = (a + jb) \cdot (c + jd)$ the straightforward way,

$$e = ac - bd, \quad f = ad + bc, \tag{3.126}$$

four real multiplications and two real additions are required. The algorithm of the previous section was run, producing the 3 multiplication code below:

$$\begin{array}{rl} t_1 & = a(c+d) \\ t_2 & = (a+b)d \\ t_3 & = (a-b)c \\ e & = t_1 - t_2 \\ f & = t_1 - t_3 \end{array} \tag{3.127}$$

3.5.6 Quaternion Multiplication Experiment

Recall from Section 2.2 that quaternions can be effectively used in orientation problems; cascaded rotations of points can be carried out by multiplying the corresponding unit quaternions (see Appendix B). Here we search for an efficient implementation of quaternion multiplication $\overset{\circ}{\mathbf{r}}=\overset{\circ}{\mathbf{q}}\overset{\circ}{\mathbf{p}}$, where

$$\overset{\circ}{\mathbf{r}}\overset{\triangle}{=}\begin{bmatrix} r_0 \\ r_x \\ r_y \\ r_z \end{bmatrix}, \quad \overset{\circ}{\mathbf{q}}\overset{\triangle}{=}\begin{bmatrix} q_0 \\ q_x \\ q_y \\ q_z \end{bmatrix}, \quad \overset{\circ}{\mathbf{p}}\overset{\triangle}{=}\begin{bmatrix} p_0 \\ p_x \\ p_y \\ p_z \end{bmatrix}. \tag{3.128}$$

Recall that the left quaternion matrix $\mathcal{Q}$ associated with a quaternion $\overset{\circ}{\mathbf{q}}$ is given by (see Appendix B):

$$\mathcal{Q} \overset{\triangle}{=} \begin{bmatrix} q_0 & -q_x & -q_y & -q_z \\ q_x & q_0 & -q_z & q_y \\ q_y & q_z & q_0 & -q_x \\ q_z & -q_y & q_x & q_0 \end{bmatrix}. \tag{3.129}$$

Using the left quaternion matrix representation, the quaternion product can be written as

$$\overset{\circ}{\mathbf{q}}\overset{\circ}{\mathbf{p}} = \mathcal{Q}\,\overset{\circ}{\mathbf{p}}\,. \tag{3.130}$$

The composition of rotations may be represented as the product of unit quaternions. Using the matrix representation of a quaternion, we see that only 16 multiplications and 12 additions are needed to compose two quaternions, whereas 27 multiplications and 18 additions are needed to form the product of two rotation matrices.

The algorithm of the previous section was run, producing the 10 multiply straightline code below:

$$\begin{aligned}
t_1 &= q_0(p_0 + p_x + p_y + p_z) \\
t_2 &= (q_0 + q_x + q_y - q_z)p_x \\
t_3 &= (q_0 + q_z)(p_y + p_z) \\
t_4 &= (q_0 - q_y)(2p_0 + p_x + p_z) \\
t_5 &= (q_0 + 2q_x + q_y)(p_x - p_z) \\
t_6 &= (q_0 - q_z)(2p_0 - p_x + p_y) \\
t_7 &= (q_y + q_z)p_z \\
t_8 &= (q_0 - 2q_x + q_z)(p_x + p_y) \\
t_9 &= (q_y - q_z)(p_x - p_y) \\
t_{10} &= (q_0 - q_x)p_0 \\
r_0 &= t_1 - t_2 - t_3 + t_9 \\
r_x &= t_1 - t_3 + t_7 - t_{10} \\
r_y &= t_1 - t_2 - t_4/2 + t_5/2 \\
r_z &= t_1 - t_2 - t_6/2 - t_8/2
\end{aligned} \tag{3.131}$$

3.6 Efficient SVD FPGA Implementation for Orientation Problems

In this section, we apply some of the hardware optimization principles developed earlier in the chapter to the problem of efficiently calculating the singular value decomposition (SVD). We will make use of the approach given here to accelerate algorithms for multiple view orientation in Chapter 5.

The SVD is one of the most elegant and satisfying algorithms in linear algebra. It provides not only quantitative information about the structure of matrices and linear equations, but also has stable, efficient, and extremely accurate numerical computational procedures. The solution to many optimization problems become simple and obvious with the correct application of the SVD.

It should perhaps be no surprise then that the SVD has found very widespread use in signal processing and adaptive filtering [81], beamforming and direction finding, optimization and control theory [27], and of course, computer vision [10], [31], [149], [189]. While the SVD exists for complex matrices, we focus here on purely real matrices.

The SVD theorem is simple to state. Given an $M \times N$ matrix $\mathbf{A}$, there exist matrices $\mathbf{U}_{M\times M}$, $\mathbf{S}_{M\times N}$, and $\mathbf{V}_{N\times N}$ such that

$$\mathbf{A} = \mathbf{USV}^T, \tag{3.132}$$

where $\mathbf{U}$ and $\mathbf{V}$ are orthonormal and $\mathbf{S}$ is diagonal with

$$\mathbf{S} = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_r), \tag{3.133}$$

and $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r > 0$. In (3.133), r is called the "rank" of $\mathbf{A}$, and we have that $r \leq \min(M, N)$. The theorem was first proved by Beltrami [15], (translated in [16]),

rediscovered several times, and perhaps finally becoming well recognized through the work of Eckart and Young [49].

3.6.1 Jacobi Method

In this section, we focus on the efficient FPGA computation of the SVD of a symmetric, square matrix. This is in fact the same as an eigenvalue decomposition, and in this case we have $\mathbf{U} = \mathbf{V}$. For this situation, we will develop the method known variously as the "Jacobi" method (sometimes called the "Givens" method). The Jacobi method employs a sequence of 2×2 plane rotations, each of which is designed to zero out a pair of off-diagonal elements of a matrix. Gradually, the data matrix is reduced to diagonal form, although the zeros created by a rotation are generally filled in by subsequent rotations. Nevertheless, the algorithm does converge quite rapidly. (Recently, Mackey [122] proposed a quaternion-based method that zeroes the 2×2 off-diagonal blocks of a 4×4 matrix; the algorithm requires fewer iterations to converge, but more work at each iteration.)

The utility of this approach is that the calculation and application of the plane rotations is simple and modular, and is particularly well suited to custom computing implementation. Also, a systolic array approach is known that allows for especially high speed processing if desired.

The algorithm proceeds by a sequence of basic transformations of the data matrix $\mathbf{A}$. In each transformation, the matrix is pre- and post-multiplied by a particular orthogonal matrix. The $(k+1)^{st}$ transformation is given as

$$T_{k+1}: \ \mathbf{A}^{(k+1)} = \mathbf{J}^{(k)T}\mathbf{A}^{(k)}\mathbf{J}^{(k)}. \tag{3.134}$$

Here, the superscripts indicate iteration number. The matrices $\mathbf{J}^{(k)}$ are orthonormal and are in fact of a special form that we shall see shortly. If we write out the result of p applications of the transformation (3.134), we have

$$\mathbf{A}^{(p)} = \mathbf{J}^{(p-1)T} \cdots \mathbf{J}^{(1)T}\mathbf{J}^{(0)T}\mathbf{A}^{(0)}\mathbf{J}^{(0)}\mathbf{J}^{(1)} \cdots \mathbf{J}^{(p-1)}, \tag{3.135}$$

with $\mathbf{A}^{(0)} = \mathbf{A}$. If after p iterations, the matrix $\mathbf{A}^{(p)}$ is diagonal (or as close to diagonal as we desire), we then extract the main diagonal of $\mathbf{A}^{(p)}$ and call this the matrix $\mathbf{S}$. Since the product of orthornomal matrices in (3.135) is also orthonormal, by setting $\mathbf{V} = \mathbf{J}^{(0)}\mathbf{J}^{(1)} \cdots \mathbf{J}^{(p-1)}$, we have the desired decomposition.

The orthonormal matrices $\mathbf{J}^{(k)}$ are of the special form

$$\mathbf{J}^{(k)} = \begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & c_k & \cdots & s_k & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & -s_k & \cdots & c_k & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix}, \tag{3.136}$$

where the matrix is the same as the identity matrix except for four elements, located at the four intersections of the i^{th} and j^{th} rows and columns. The special elements are $c_k = \cos\theta_k$ and $s_k = \sin\theta_k$ for some angle θ_k.

The rotations by $\mathbf{J}^{(k)}$ are designed to zero the (i,j) and (j,i) elements, moving their energy to the main diagonal. This leaves the matrix $\mathbf{A}^{(k+1)}$ with less off-diagonal energy than $\mathbf{A}^{(k)}$. We could seek the greatest energy reduction at each stage by searching for the off-diagonal elements with the maximum energy. However, this greatly increases the amount of work at each stage, and precludes parallel implementations.

What is typically done is to use a sequence of "sweeps," and each sweep is "cyclic" (by rows or columns). This means that we apply the transformation (3.134) for the $M(M-1)/2$ distinct pairs (i,j) in a cyclic order. We repeat for as many sweeps as necessary until the convergence criteria is met. This algorithm is known to converge (but this is definitely not obvious).

With the overall strategy thus defined, we must now address the computation of $\mathbf{J}^{(k)}$, the efficient formation of the running product that calculates $\mathbf{V}$, and of course, the calculation of $\mathbf{A}^{(k+1)}$.

Referring to (3.134) and (3.136), we see that the left-most $\mathbf{J}^{(k)}$ only alters columns i and j of $\mathbf{A}^{(k)}$, while the right-most $\mathbf{J}^{(k)}$ only alters rows i and j. All other elements of $\mathbf{A}^{(k)}$ are unchanged, and in fact, we do not access them for this iteration. The four elements of $\mathbf{A}^{(k)}$ at the intersection of the rows and columns i and j determine the elements c_k and s_k of $\mathbf{J}^{(k)}$. We can extract the relevant 2×2 submatrices from (3.134) and (3.136) and write the diagonalization requirement as

$$\begin{bmatrix} a_{ii}^{(k+1)} & 0 \\ 0 & a_{jj}^{(k+1)} \end{bmatrix} = \begin{bmatrix} c_k & -s_k \\ s_k & c_k \end{bmatrix} \begin{bmatrix} a_{ii}^{(k)} & a_{ij}^{(k)} \\ a_{ji}^{(k)} & a_{jj}^{(k)} \end{bmatrix} \begin{bmatrix} c_k & s_k \\ -s_k & c_k \end{bmatrix}. \tag{3.137}$$

By expanding the above matrix product, equating the off diagonal terms and dividing by $c_k s_k$, we have the requirement

$$0 = a_{ii}^{(k)} - a_{jj}^{(k)} - \frac{s_k}{c_k} a_{ji}^{(k)} + \frac{c_k}{s_k} a_{ij}^{(k)}. \tag{3.138}$$

Since $a_{ji}^{(k)} = a_{ij}^{(k)}$, if we make the definitions

$$t_k = \frac{s_k}{c_k}, \tag{3.139}$$

and

$$b = \frac{a_{jj}^{(k)} - a_{ii}^{(k)}}{2a_{ij}^{(k)}}, \tag{3.140}$$

we have that (3.138) becomes

$$t_k^2 + 2bt_k - 1 = 0. \tag{3.141}$$

It is a simple matter then to solve the above equation for t_k (either solution is acceptable). Since $t_k = \tan\theta_k$, we may therefore obtain the elements c_k and s_k. We have therefore determined the matrix $\mathbf{J}^{(k)}$.

3.6.2 FPGA Implementation

We now turn to the question of the efficient implementation of the update transformation (3.134). We first examine the post-multiplication of $\mathbf{A}^{(k)}$ by $\mathbf{J}^{(k)}$. Only columns i and j of $\mathbf{A}^{(k)}$ are altered. If we notionally extract these two columns of $\mathbf{A}^{(k)}$, and also extract the

2×2 rotation submatrix of $\mathbf{J}^{(k)}$, we see that we have exactly a sequence of M operations isomorphic to a complex multiplication. We know from Section 3.5.5 that this operation can be implemented using three real multiplications and three real additions per operation (with 2 preadditions performed before the M rotations). Thus, for a total of $3M$ multiplications and $3M + 2$ additions the post multiplication in (3.134) can be performed. Similar comments apply to the premultiplication in (3.134). We must also update the estimate of $\mathbf{V}$ by postmultiplying the current estimate by $\mathbf{J}^{(k)}$. Again, this is exactly the same type of operation as the post-multipication of $\mathbf{A}^{(k)}$. The operation count is therefore $9M$ multiplications and $9M + 2$ additions.

For FPGA implementation, an efficient procedure then is to implement a high speed complex multiplication algorithm in hardware, and stream the various rows and columns of $\mathbf{A}^{(k)}$ and the current estimate of $\mathbf{V}$ through in a pipelined fashion. Only three parallel multipliers and three adders are needed in this approach, so a large wordlength could be accommodated. With a proper implementation, we can therefore achieve a 3-to-1 speedup over a conventional processing implementation. Given the ubiquitous nature of the SVD, this is a valuable efficiency gain. Of course, the speedup can increase linearly with the addition of multiple FPGAs.

FPGA technology allows for an alternative implementation of this rotation operation: the so called "CORDIC" algorithm [22], [95], [196], [198]. This algorithm is especially suited to the rotation of a two-dimensional vector by a given angle. Generally, rotation of a vector by an angle θ requires a matrix-vector multiplication of the form

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}. \tag{3.142}$$

There is a special set of angles for which a more computationally efficient method exists to perform this rotation. These angles are of the form

$$\theta_l = \tan^{-1} 2^{-l} \quad \text{for } l = 0, 1, \ldots \ . \tag{3.143}$$

The angles are typically stored in a table for subsequent processing.

Using trigonometric identities, it can be shown that

$$\sin(\theta_l) = \frac{2^{-l}}{\sqrt{1 + 2^{-2l}}}, \tag{3.144}$$

and

$$\cos(\theta_l) = \frac{1}{\sqrt{1 + 2^{-2l}}}. \tag{3.145}$$

Therefore, a rotation of a vector (x,y) by θ_l can be evaluated as

$$x' = \frac{1}{\sqrt{1 + 2^{-2l}}}(x + 2^{-l}y), \tag{3.146}$$

$$y' = \frac{1}{\sqrt{1 + 2^{-2l}}}(y - 2^{-l}x). \tag{3.147}$$

To rotate an angle by $-\theta_l$, the same equations are used with the sign of 2^{-l} changed.

The above special set of angles can be used to rotate a vector by an arbitrary angle θ. Any angle can be decomposed as

$$\theta = \eta_1 \frac{\pi}{4} + \sum_{l=0}^{\infty} \eta_l \tan^{-1} 2^{-l}, \tag{3.148}$$

or,

$$\theta = \sum_{l=-1}^{\infty} \eta_l \theta_l, \tag{3.149}$$

where η_l is either 1 or -1. To rotate by the angle θ, rotate the vector sequentially by the amount θ_l in the directions specified by the η_l. The iterations are truncated after a suitable number have been performed. The error in the angle using n iterations is bounded by $\tan^{-1} 2^{1-n} \approx 2^{1-n}$.

The η_l are computed each iteration by comparing the current angle with the desired angle. If the current angle is larger, then the angle increment $\tan^{-1} 2^{-l}$ is subtracted from the current angle to form the next estimate. If the current angle is too small, the angle increment is added to the current angle. When the CORDIC algorithm is implemented, the scaling factors are generally ignored during the iterations. That is, the operations

$$x' = (x + 2^{-l} y) \tag{3.150}$$

and

$$y' = (y - 2^{-l} x) \tag{3.151}$$

are performed for each θ_l. This greatly simplifies the iterations, but introduces a magnitude gain. After n iterations, the magnification is

$$\prod_{l=1}^{n} \sqrt{1 + 2^{-2l}}, \tag{3.152}$$

which, fortunately, is independent of the signs of the individual rotations. This gain can be removed after the iterations are completed if necessary. For the SVD, we can remove it at the very end of the algorithm, if we use a cyclic sweep method. Table 3.3 tabulates this gain as a function of the number of stages; note that gain quickly converges to a constant.

Hardware implementations of the CORDIC algorithm typically use two accumulators and barrel shifters to perform the iterations (and possibly another accumulator for angle accumulation). The throughput of this accumulator-based operation is thus dependent on the number of iterations performed, which determines the accuracy of the method. The wordlength of the accumulators must accommodate the longest results, which are typically at the end of the calculation.

For greater throughput, the iterations can be pipelined along multiple computational units. Now rotations can be performed at a rate equal to one pipeline stage delay. To reduce hardware, the size of the adders at each stage can now be made different.

Table 3.3: CORDIC gain as function of the number of stages.

Num. Stages	Magnification
1	1.1180
2	1.1524
3	1.1614
4	1.1637
5	1.1642
6	1.1644
7	1.1644

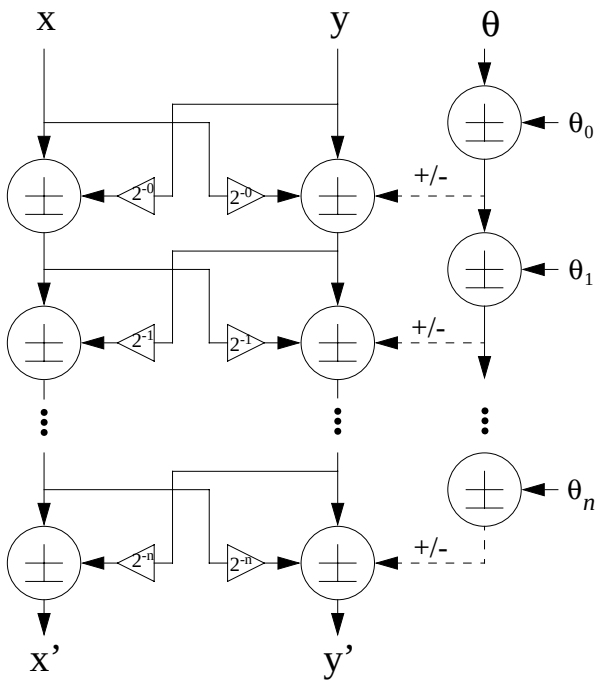

Figure 3-28: Parallel-pipelined implementation of the CORDIC algorithm (pipeline delay registers not shown). The θ_l are hardwired. Alternatively, the "+/−" control signals could be calculated in software for a given pass.

If we lay each stage out in a parallel-pipelined manner, the shift that implements the multiplications by 2^{-l} for stage l becomes hardwired, and we do not need to devote any hardware resources to this operation. Therefore, each stage consists of two adder/subtractors (one to update the x and y coordinates each), and an adder to calculate the angle rotation direction, as shown in Figure 3-28. Alternatively, if the number of bits of precision on the angle calculation is small enough, the sequence of rotation directions may simply be stored in a ROM.

Note that the angle increment magnitudes θ_l are fixed, and only their sign direction at each stage must be determined. This means that in a parallel-pipelined design, one input of each adder is hardwired, so that we do not need to supply a stream of angle increment values from some memory source.

A word about representing angles in two's complement integer format is in order. Both the two's complement system and angles in radian measure exhibit a "wrap-around" effect. An effective representation of angles is to store $2^{m-1}/\pi$ times the radian value of the angle, where m is the number of bits in the two's complement word. Thus, the wrap-around points coincide. Modulo arithmetic on angles is performed simply by ignoring carry bits beyond the wordlength.

We may assign a custom wordlength to the inputs of each adder in each stage, in an effort to save on hardware cost. The MCMC design algorithm of Section 3.3.5 may then be used to find good wordlength combinations. Lazy rounding (Section 3.4) may also be extensively used, as there is likely to be many occurrences of truncated bits, especially in stages with the largest l's.

Assume that the inputs x and y are limited to the range $-1 \le x, y \le 1$, which we represent as two's complement fractions. If the CORDIC algorithm is implemented according to (3.146) and (3.147), the outputs are limited to the range $-\sqrt{2} \le x', y' \le \sqrt{2}$. If we implement it without correcting for the magnitude gain from Table 3.3, then the output range is $-1.1644\sqrt{2} \le x', y' \le 1.1644\sqrt{2}$, which we round up to $[-2, 2]$ in order to determine the output format (as in (3.5)). Moreover, due to the actions of the control lines, this range is valid at the output of every stage of the algorithm in Figure 3-28.

Recall from Section 3.3.1 that to calculate contribution due to noise from stage i, we need to find the max absolute slope s_i. The calculation of the noise variance due to lazy rounding is straightforward but tedious, due to the numerous paths from any particular rounding point to the outputs. Using (3.3), the gain s_i due to these numerous paths, from stage i to the output as function of the total number of stages n is given in Table 3.4. This is used in (3.4), along with the formula for the variance of lazy rounding (from Table 3.2 or Figure 3-23 for higher accuracy). The lazy rounding variance must be appropriately scaled, by $4 \cdot 2^{-2b_{i+1}}$, because of the range $[-2, 2]$ and because the development in Section 3.4 assumed that the final output format was integer, rather than the fractional format which we use here.

Table 3.4: Gain s_i from stage i to final stage n.

i	$n=2$	$n=3$	$n=4$	$n=5$	$n=6$	$n=7$
1	1.5000	1.8750	2.1094	2.2412	2.3112	2.3474
2		1.1250	1.2188	1.2744	1.3046	1.3204
3			1.0312	1.0547	1.0684	1.0758
4				1.0078	1.0137	1.0171
5					1.0020	1.0034
6						1.0005

Another complicating factor is that because of the increasing shift amounts of half of the adder inputs in Figure 3-28, we must really take into account that the model used in lazy rounding is not exactly accurate for this case. We are led to consider three cases for the wordlength b_{i+1} at the output of stage i: b_{i+1} less than b_i, b_{i+1} greater than the length of the shifted input (which is $b_i + i - 1$), and finally, b_{i+1} between these two boundaries. After some tedious arithmetic, we arrive at the formula

$$\sigma_i^2 = \begin{cases} \frac{5}{12}2^{-2b_{i+1}} - \frac{2}{3}2^{-2b_i}, & b_{i+1} < b_i, \\ \frac{1}{3}2^{-2b_{i+1}} - \frac{1}{3}2^{-4b_{i+1}}, & b_i \le b_{i+1} < b_i + i - 1, \\ 0, & b_i + i - 1 \le b_{i+1}, \end{cases} \tag{3.153}$$

for $i = 1, \ldots, n-1$. The total variance is

$$V(\mathbf{b}) = \sum_{i=2}^{n} s_i^2 \sigma_i^2 \tag{3.154}$$

and the cost is

$$C(\mathbf{b}) = \sum_{i=2}^{n} b_i. \tag{3.155}$$

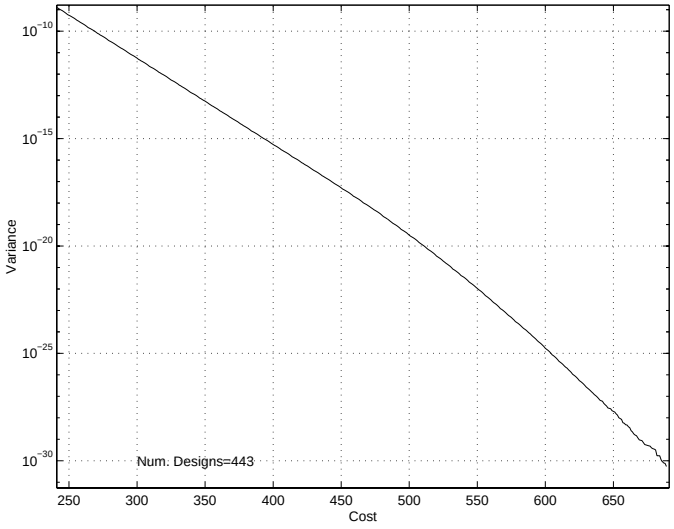

Figure 3-29: CORDIC cost/variance combinations using lazy rounding.

Using this information, the MCMC algorithm of Section 3.3 was used to design good wordlength combinations for the CORDIC algorithm. The number of stages was set at $n = 16$, and a 32-bit input wordlength was assumed. Figure 3-29 shows the Pareto-optimal cost/variance combinations, and Figure 3-30 shows the points visited by the MCMC random walk during a particular iteration. As an example, of the 443 Pareto-optimal designs found, the 300^{th} one had a cost of 539, and the wordlengths were

$$\mathbf{b} = [32, 32, 33, 35, 37, 37, 37, 36, 37, 36, 37, 36, 37, 36, 37, 36]^T \tag{3.156}$$

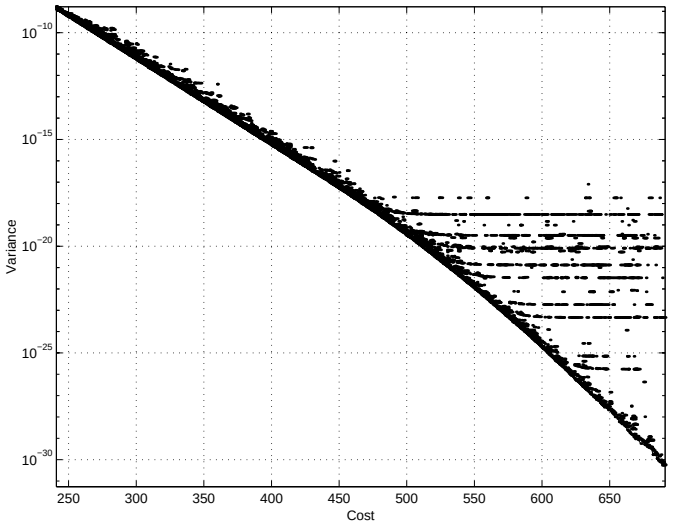

Figure 3-30: Design space exploration of CORDIC using lazy rounding. Points plotted are the design alternatives visited by the MCMC random walk for a particular iteration.

3.7 FPGA Implementation of Absolute Orientation

Recall from Section 2.5 the absolute orientation problem: Align two sets of three-dimensional feature points by a rotation and translation. In Section 2.5 we saw the SVD-based method for performing absolute orientation. We form the $3 \times N$ matrix $\mathbf{L}$ from the centroid-removed left coordinate-system data points $\tilde{\mathbf{l}}_i$, and similarly for $\mathbf{R}$, the matrix formed from the right coordinate-system data points $\tilde{\mathbf{r}}_i$. We then form the 3×3 matrix $\mathbf{RL}^T$ and take the SVD. In this section, we address this problem using a straightforward FPGA design. By tailoring the wordlengths, we shall be able to simultaneously speed up the computation and also deliver improved accuracy.

For large N, the major computational load is the formation of $\mathbf{RL}^T$. This requires $9N$ multiplications. Perhaps more importantly, the outputs will have approximately double the dynamic range of the inputs, so that roundoff errors are more likely using conventional DSP microprocessors. As an aside, one may wonder if the efficient bilinear form algorithm of Section 3.5 can be applied here. However, since each of the nine outputs consists of a single product, it is known from the work of Winograd ([203], Chap. 3, Thm. 2) that we cannot implement the outer product $\tilde{\mathbf{r}}_i \tilde{\mathbf{l}}_i^T$ in less than nine multiplications.

For the design, we make use of the outer product decomposition of the matrix product:

$$\mathbf{RL}^T = \sum_{i=1}^{N} \tilde{\mathbf{r}}_i \tilde{\mathbf{l}}_i^T. \tag{3.157}$$

Each pairing i will be processed sequentially in consecutive clock cycles for $i = 1, \ldots, N$. For each i, we must calculate the nine multiplications corresponding to the rank-1 outer product $\tilde{\mathbf{r}}_i \tilde{\mathbf{l}}_i^T$ and then add the multiplication outputs to nine accumulators. We assume without loss of generality that the inputs are scaled so that they are 2's complement fixed-point fractions, with range from -1 to 1. The multiplier outputs are then also confined to this range, and the accumulator outputs are confined to the range $\pm\lceil \log_2 N \rceil$.

Let the number of bits of used at the multiplier j inputs be b_{j1}. If a full precision multiplier is built, the number of bits produced at its output is b_{j1}^2. To save on hardware cost, we will consider sending (and calculating) a fewer number of bits, b_{j2}, to the accumulator. Let the number of bits input to accumulator j be b_{j2}. The accumulators will be constructed so that all $b_{j2} + \lceil \log_2 N \rceil$ output bits are produced, but a smaller number, b_{j3} might be send to the SVD computation.

For simplicity, we assume that the 3×3 SVD is performed in software. Thus, we are not concerned with its computational cost, but only its noise variance propagation properties. To assess the quality function $V(\mathbf{b})$ of a proposed wordlength combination $\mathbf{b}$, we could use several metrics. The simplest is of course the sum of the noise variances at the SVD inputs. Another possibility is the sum of the noise variances in the calculated rotation matrix $\mathbf{Q}$ components. Still another possibility is the noise variance of the components of the associated unit quaternion $\mathbf{q}$. Finally, we might want to take into account the errors in the translation direction which is calculated from the rotation matrix. The correct choice must therefore be dependent on the user's application. For simplicity, we will use the total variance at the input to the SVD algorithm as our metric.

The MCMC design method, described in detail in Section 3.3.5, was used to identify actual wordlength combinations of interest. Figure 3-31 shows a dataflowgraph of the portion implemented in FPGA. Figure 3-32 shows the resulting Pareto-optimal combinations. Figure 3-33 shows the design space exploration performed in the last iteration of the algorithm,

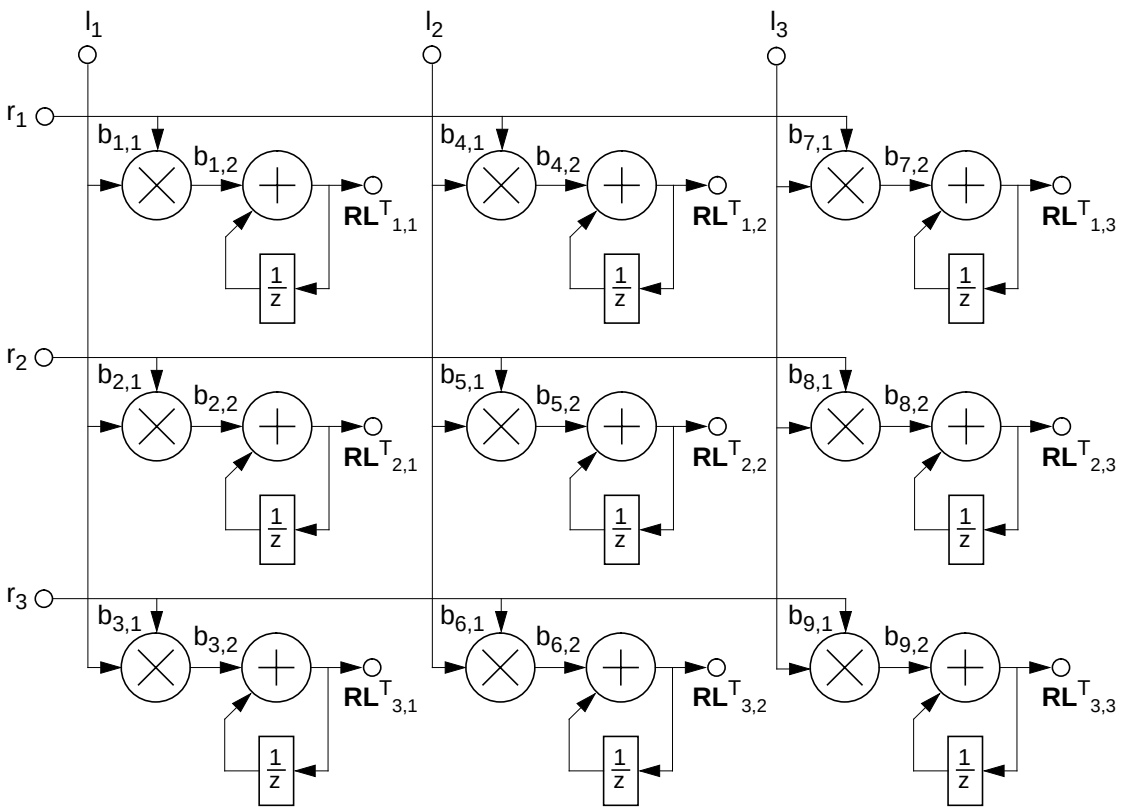

Figure 3-31: Flowgraph of absolute orientation preprocessing in FPGA.

with the cost/variance bound in place to avoid exploring the upper right-hand corner of the design space.

Figure 3-34 shows the resulting FPGA floorplan using the Xilinx floorplanner tool. One can clearly see the nine multiply-accumulate structures (in addition to some interface and control logic). The truncated multiplier structures were designed using Xilinx's "Core" series of VHDL libraries [207], rather than the lazy rounding approach of Section 3.4.

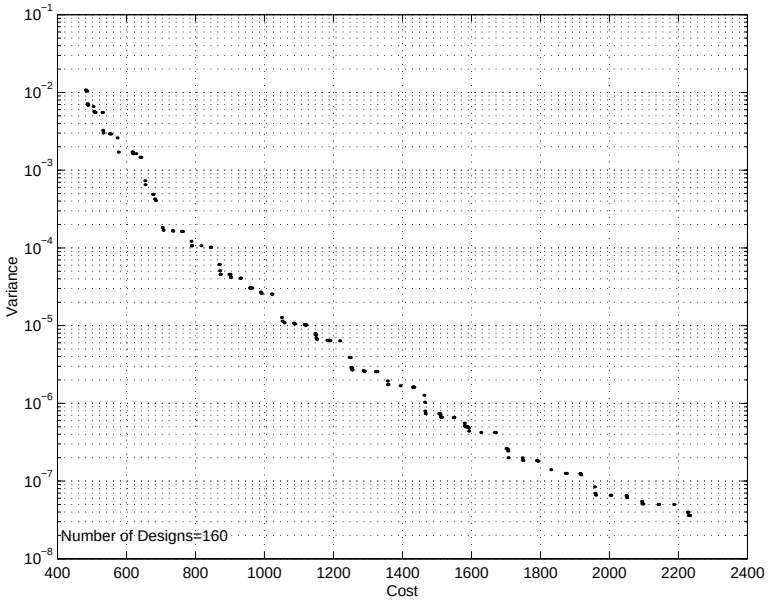

Figure 3-32: Optimal cost/variance designs for FPGA implementation of absolute orientation.

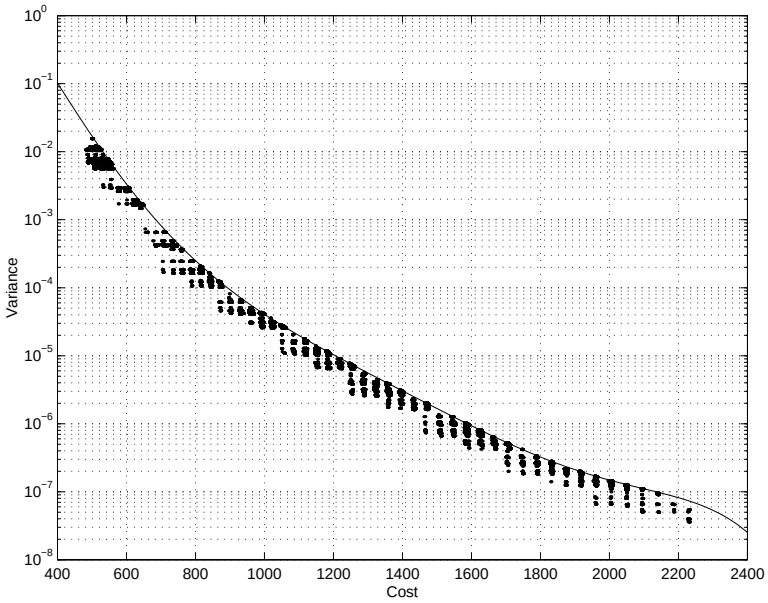

Figure 3-33: Design space exploration with cost/variance upper bound to limit excursions.

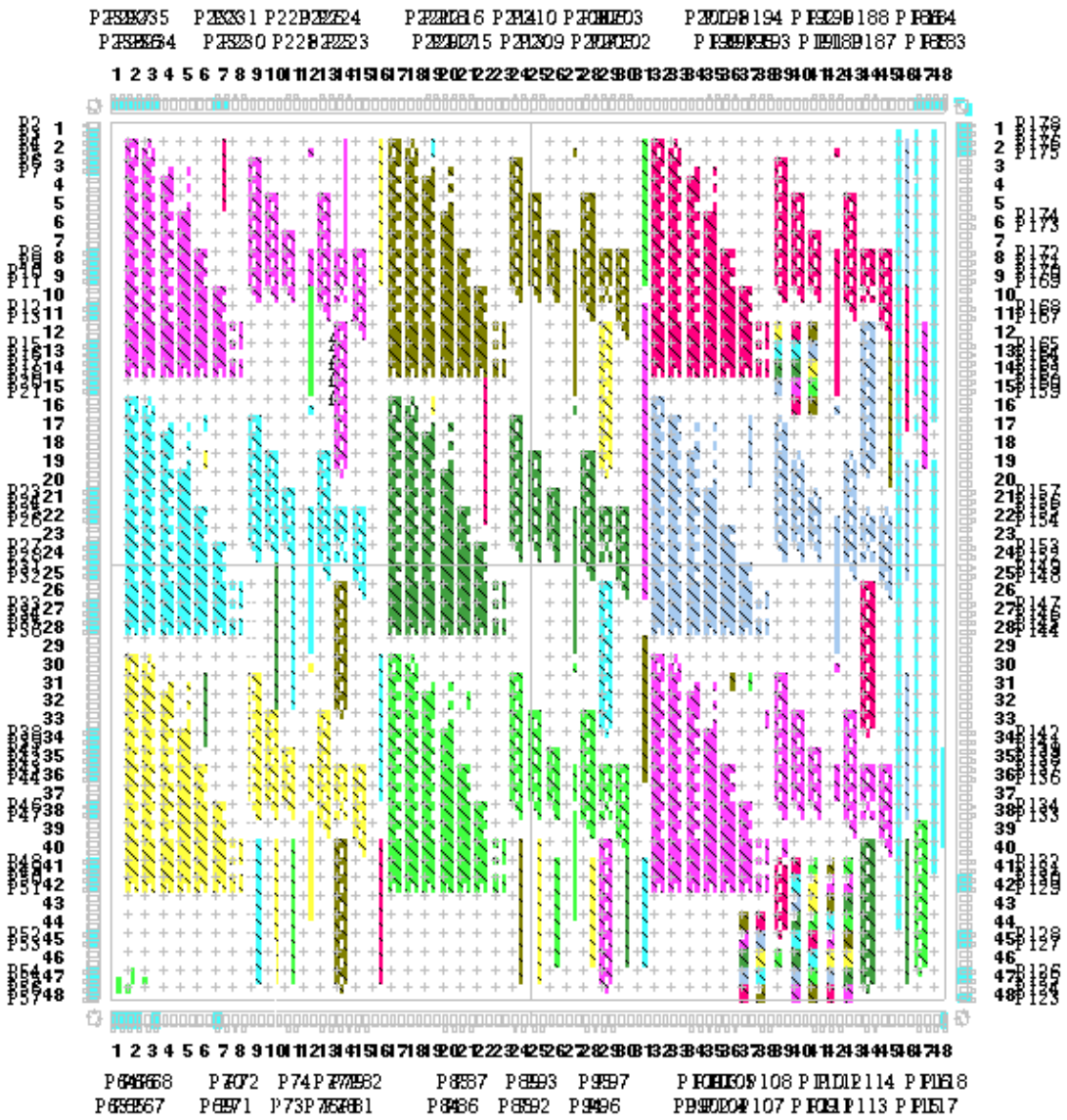

Figure 3-34: FPGA floorplan for custom wordlength absolute orientation.

3.8 Summary

This chapter was concerned with the various aspects of efficient hardware implementation that are needed for real-time computer vision systems. The methods and techniques presented nicely illustrate the approach to efficient implementations through simultaneous consideration of the algorithm and technology.

The techniques presented in this chapter appear to be different in many respects; however, they share the common goal of *reducing operations complexity, while meeting the algorithm performance goals.*

Section 3.2 provided a detailed understanding of the FPGA device family. Like software or ASICS, FPGAs are here to stay; it therefore behooves real-time computer vision system designers to learn about and to use these devices.

Section 3.3 presented the method for reducing system complexity by wordlength reduction. The approach taken was to first model the noise due to wordlength reduction by equations (3.1)-(3.7), which allowed us to seek designs with optimal cost/variance characteristics. A useful lower bound on these designs and an algorithm to compute this bound was given in Section 3.3.3.

Finding actual designs is a combinatorial optimization problem, which is difficult in general. Section 3.3.4 investigated the MCMC method to perform this optimization. This technique has proven useful in many other scientific fields, as it does for solving the wordlength optimization problem. The key idea that makes the MCMC useful is that the deviation from the stationary distribution of the Markov chain as a function of time can be made arbitrarily small, as given by (3.45) and (3.46).

Section 3.4 presented the lazy rounding method for reducing system complexity by focusing on the operations of addition and multiplication, which typically represent the major portion of any orientation algorithm implementation and custom computing design.

Section 3.5 presented the method for reducing system complexity by the more conventional approach of eliminating costly multiplications in the evaluation of bilinear forms. Section 3.5.3 presented an algorithm for multiple bilinear forms. While not optimal, it seems to work well in practice.

Finally, Sections 3.6 and 3.7 applied some of these ideas to the design of custom processors for the SVD and absolute orientation operation.

Although all of the problems presented here are well studied, the chapter featured several new and exciting design approaches:

- the lower bound approach and calculation for optimal wordlength designs (Section 3.3.3),
- applying the well known MCMC approach to finding actual designs (Section 3.3.5),
- the simple and efficient lazy rounding techniques of Section 3.4,
- the algorithm to search for common rank-reducing outer products (Section 3.5.3).

The development of these techniques required concepts from graph theory, statistical mechanics, computer arithmetic, dynamic systems, and signal processing. This illustrates the fact that device technology and algorithms should be considered together; not only can we use efficient devices to implement algorithms in real-time, but we can also use efficient algorithms to help optimize hardware designs. This close interplay between algorithm and implementation is at the heart of a successfully engineered system.

Chapter 4

Constrained Estimation and Motion Vision

4.1 Introduction and Chapter Overview

As we saw in Chapter 2, there are a great variety of orientation problem formulations and solution methods. For example, the linear approaches are favored by many computer vision researchers because of their speed, whereas photogrammetric work requires high accuracy. In this chapter we more closely examine general solution methods which enforce (perhaps inconvenient) constraints. We will take the viewpoint that one should maximize the likelihood function of the data, subject to parametric constraints supplied by the orientation problem at hand.

Since we are taking the maximum likelihood viewpoint, it would be helpful perhaps to review this idea. We assume that there is a signal $\mathbf{s}(n;\boldsymbol{\theta})$, where $\mathbf{s}$ is a vector of signal values, $\boldsymbol{\theta}$ is a vector of unknown, nonrandom parameters, and n is the time index. Due to noise, we cannot directly observe $\mathbf{s}$, but rather a corrupted version $\mathbf{y}$:

$$\mathbf{y}(n) = \mathbf{s}(n;\boldsymbol{\theta}) + \mathbf{v}(n). \tag{4.1}$$

For simplicity, assume that $\mathbf{v}(n)$ is $N(0,\ \mathbf{I})$. Under this assumption, the observed data is also distributed normally, with

$$\mathbf{y}(n) \sim N(\mathbf{s}(n;\boldsymbol{\theta}),\ \mathbf{I}). \tag{4.2}$$

When viewed as a function of the unknown parameters $\boldsymbol{\theta}$, the distribution $L(\boldsymbol{\theta};\mathbf{y}(n)) \stackrel{\triangle}{=} N(\mathbf{y}(n) - \mathbf{s}(n;\boldsymbol{\theta}),\ \mathbf{I})$ is known as the *likelihood* function [103], [200]. Maximum likelihood estimation (MLE) seeks the value of the parameter vector $\boldsymbol{\theta}$ that best explains the observed data $\mathbf{y}$. That is, we seek

$$\boldsymbol{\theta}_{ml} = \arg\max_{\boldsymbol{\theta}} L(\boldsymbol{\theta};\mathbf{y}(n)) = \arg\max_{\boldsymbol{\theta}} \ln L(\boldsymbol{\theta};\mathbf{y}(n)). \tag{4.3}$$

For normal distributions, this is easily done symbolically; for other distributions, numerical iterations are often employed.

Maximum likelihood estimation is therefore well established for the signal and data model of (4.1). In some cases, however, it may be very unclear what the signal $\mathbf{s}(n;\boldsymbol{\theta})$ is. This arises when $\mathbf{s}(n;\boldsymbol{\theta})$ is determined *implicitly* as a function of $\boldsymbol{\theta}$, rather than explicitly. In many situations, while $\mathbf{s}(n;\boldsymbol{\theta})$ is well-defined in that a given value of $\boldsymbol{\theta}$ implies a unique value of $\mathbf{s}(n;\boldsymbol{\theta})$, it may be difficult or impossible to explicitly write an algebraic formula for $\mathbf{s}(n;\boldsymbol{\theta})$. We may in fact be given only a set of constraint equations that define the

relationship between $\boldsymbol{\theta}$ and $\mathbf{s}(n;\boldsymbol{\theta})$. This is typical for orientation problems. Nevertheless, we may still seek to estimate $\boldsymbol{\theta}$ by maximizing the likelihood function $L(\boldsymbol{\theta};\mathbf{y}(n))$; we merely must ensure that none of the constraints are violated at the purported optimum. This optimum could be termed the *constrained maximum likelihood* estimate. However, we will still denote it by $\boldsymbol{\theta}_{mle}$, to recognize the fact that notionally there is no difference between this estimate and the more conventional MLE (although there will certainly be a *computational* difference).

A small example will illustrate these concepts. Let $\boldsymbol{\theta} = [\theta_1,\ \theta_2]^T$, and let $\mathbf{s}(n;\boldsymbol{\theta})$ be implicitly defined by the constraints

$$\mathbf{s}(n;\boldsymbol{\theta}) = \boldsymbol{\theta},\ n = 1,\ldots,N, \tag{4.4}$$

and

$$\theta_1 + \theta_2 = 1. \tag{4.5}$$

In this case, it is easy to express $\mathbf{s}(n;\boldsymbol{\theta})$ explicitly as a function of θ_1, and to then solve for the MLE via (4.1). Following this procedure, one obtains

$$\theta_{1,ml} = \frac{1 + \overline{Y}_1 + \overline{Y}_2}{2}, \tag{4.6}$$

where $\overline{Y}_1$ is the mean of the first component of the 2×1 vector of observed data $\mathbf{Y}_1,\ldots,\mathbf{Y}_N$ (and similarly for $\overline{Y}_2$. Since $\theta_2 = 1 - \theta_1$, we may use the invariance property of the MLE to write

$$\theta_{2,ml} = 1 - \frac{1 + \overline{Y}_1 + \overline{Y}_2}{2}. \tag{4.7}$$

If we were to ignore the constraint (4.5), we would obtain the estimate

$$\boldsymbol{\theta}_{est} = [\overline{Y}_1,\ \overline{Y}_2]^T, \tag{4.8}$$

which is incorrect, because in general $\overline{Y}_1 + \overline{Y}_2 \neq 1$. Thus, to obtain the MLE, it is not acceptable to ignore constraints. This is precisely the chief complaint about projective-geometry approaches to motion recovery; orthogonality constraints are ignored.

The above procedure relied on removing the redundant variable θ_2 from the problem, thereby removing the inconvenient constraint. For more complicated problems, this approach may not be possible. A way to avoid this problem is to use a Lagrange multiplier approach. We therefore seek

$$\boldsymbol{\theta}_{ml} = \arg\max_{\boldsymbol{\theta},\lambda} \ln L(\boldsymbol{\theta};\mathbf{y}(n)) + \lambda(\theta_1 + \theta_2 - 1). \tag{4.9}$$

Straightforward calculations show that (4.6) and (4.7) are again obtained.

Another simple demonstration is to obtain the total least squares (TLS) equations from an MLE problem. Let the true signal $\mathbf{s}(n;\boldsymbol{\theta})$ consist of a true matrix $\mathbf{A}_t$, and a true vector $\mathbf{b}_t$. Elements of $\mathbf{A}_t$ and $\mathbf{b}_t$ are independently corrupted with additive noise perturbations, each distributed as $N(0,1)$, resulting in the observed signal consisting of $\mathbf{A}_o$ and $\mathbf{b}_o$. We relate the signal to some unobserved parameters by the constraint $\mathbf{A}_t\mathbf{x} = \mathbf{b}_t$. Thus, the parameters $\boldsymbol{\theta}$ to be estimated are $\mathbf{A}_t$, $\mathbf{x}$, and $\mathbf{b}_t$. The constrained MLE equation is

$$\boldsymbol{\theta}_{ml} = \arg\max_{\mathbf{A}_t,\mathbf{x},\mathbf{b}_t,\boldsymbol{\lambda}} \ln L(\boldsymbol{\theta};\mathbf{A}_o,\mathbf{b}_o) + \boldsymbol{\lambda}^T(\mathbf{A}_t\mathbf{x} - \mathbf{b}_t). \tag{4.10}$$

We make the definitions

$$\begin{aligned} \widehat{\mathbf{A}}_o &\triangleq [\mathbf{A}_o \mid -\mathbf{b}_o], \\ \widehat{\mathbf{A}}_t &\triangleq [\mathbf{A}_t \mid -\mathbf{b}_t], \\ \widehat{\mathbf{x}} &\triangleq [\mathbf{x}^T \mid 1]^T, \end{aligned} \tag{4.11}$$

so that

$$\widehat{\mathbf{A}}_t \widehat{\mathbf{x}} = \mathbf{0} \tag{4.12}$$

is obtained. By the Gaussian assumption, (4.10) is equivalent to

$$\boldsymbol{\theta}_{ml} = \arg \min_{\mathbf{A}_t, \mathbf{x}, \mathbf{b}_t, \boldsymbol{\lambda}} J = \|\widehat{\mathbf{A}}_o - \widehat{\mathbf{A}}_t\|_F^2 + \boldsymbol{\lambda}^T \widehat{\mathbf{A}}_t \widehat{\mathbf{x}}. \tag{4.13}$$

Taking the derivative and setting equal to zero gives

$$\frac{\partial J}{\partial \widehat{\mathbf{A}}_t} = \mathbf{0} = -2\left(\widehat{\mathbf{A}}_o - \widehat{\mathbf{A}}_{ml}\right) + \boldsymbol{\lambda} \widehat{\mathbf{x}}_{ml}^T, \tag{4.14}$$

which clearly indicates that $\widehat{\mathbf{A}}_o - \widehat{\mathbf{A}}_{ml}$ must be rank-1. We can find $\widehat{\mathbf{A}}_{ml}$ easily by taking the SVD of $\widehat{\mathbf{A}}_o$ and setting the minimal singular value equal to zero. This minimizes $\|\widehat{\mathbf{A}}_o - \widehat{\mathbf{A}}_{ml}\|_F^2$. The matrix $\widehat{\mathbf{A}}_{ml}$ has a null space, and the estimate $\widehat{\mathbf{x}}_{ml}$ is the vector in this null space (defined up to a scale factor), so that the constraint $\widehat{\mathbf{A}}_t \widehat{\mathbf{x}} = \mathbf{0}$ is satisfied. We then rescale $\widehat{\mathbf{x}}_{ml}$ so that its last component is equal to one, and recover $\mathbf{A}_{ml}$, $\mathbf{b}_{ml}$, and $\mathbf{x}_{ml}$ from (4.11). Thus, TLS may be regarded as a constrained MLE problem.

The remainder of this chapter is organized as follows. Because orientation problems do not exactly fall into the "observation = signal + noise" category, more powerful techniques are required for high accuracy. Section 4.2 addresses this issue with a new approach: the Constrained Maximum Likelihood (CML) solution. The development of this CML approach was partly driven by the need to address several deficiencies present in current approaches.

In Section 4.3 we apply the CML technique to the estimation of motion parameters for a planar scene. This technique is compared with current techniques, and a also a simpler heuristic bias correction approach. An FPGA implementation is then presented; we make extensive use of the approaches derived in Chapter 3 in performing this design.

4.2 Constrained Maximum Likelihood Solutions

Given the desirability of correctly enforcing all constraints in an estimation problem, it seems surprising that many practitioners do not do this. This can be understood in cases where the computational complexity of an exact solution is too high. In other cases, however, it appears that researchers resort to *ad hoc* procedures that have no optimality properties. The purpose of this section is to address this problem by developing a general purpose constrained maximum likelihood solution to a set of linear equations.

4.2.1 Introduction

TLS is used to solve a set of inconsistent linear equations, $\mathbf{Ax} \approx \mathbf{y}$, when there are errors not only in the observations $\mathbf{y}$, but in the modeling matrix $\mathbf{A}$ as well. TLS seeks the least-squares perturbation of both $\mathbf{y}$ and $\mathbf{A}$ that leads to a consistent set of equations. When $\mathbf{y}$ and $\mathbf{A}$ have a defined structure, we desire the perturbations to also have this structure. Unfortunately, standard TLS does not generally preserve perturbation structure, so other

methods are required. This problem is of particular importance in solving for camera orientation.

In this section, we examine this problem using a new probabilistic framework. We derive an approach to determine the most probable set of perturbations, given an *a priori* perturbation probability density function. While this approach is applicable to both Gaussian and non-Gaussian distributions, we will specialize to the uncorrelated Gaussian case; a more detailed treatment, along with a comparison to existing methods, is given by Fiore and Verghese in [63]. (In [63] is given example applications in system identification, frequency estimation, Bayesian estimation, and the specialization to standard TLS.)

Solving a set of inconsistent linear equations $\mathbf{A}\mathbf{x} \approx \mathbf{y}$ is a ubiquitous problem in signal processing, computer vision, and control systems applications. For example, linear prediction seeks to model the next value of a sequence as a fixed linear combination of several past values of the sequence [187]. Linear prediction and more general estimation problems lead to a set of inconsistent linear equations.

When the inconsistency is due to $\mathbf{y}$ being a noise-corrupted version of a consistent right-hand side to the set of equations, an optimal solution is frequently determined by ordinary least squares (OLS). In this approach, the solution $\mathbf{x}$ that minimizes the (squared) 2-norm of the residual between $\mathbf{A}\mathbf{x}$ and $\mathbf{y}$ is sought. This solution is $\mathbf{x}_{ols} = \mathbf{A}^{+}\mathbf{y}$, where $\mathbf{A}^{+}$ is the pseudoinverse of $\mathbf{A}$ [68]; the corresponding estimate of the noise corrupting the right-hand side is $\Delta_{\mathbf{y}} = \mathbf{A}\mathbf{x}_{ols} - \mathbf{y}$.

Often, however, one would like to consider perturbing the modeling matrix $\mathbf{A}$ in addition to $\mathbf{y}$. This situation may arise when the model is uncertain, or was formed from noisy data, or was deliberately approximated, a case that arises during the iterative linearization of a nonlinear least squares fitting problem. The solution that minimizes the total sum of the squares of the perturbations in $\mathbf{A}$ and $\mathbf{y}$ is termed the *total least squares* (TLS) solution [68], [192]. This can be found simply via the singular value decomposition (SVD) algorithm [68].

TLS thus considers perturbing the modeling matrix $\mathbf{A}$ in addition to $\mathbf{y}$ by seeking $\Delta_{\mathbf{A}}$ and $\Delta_{\mathbf{y}}$ such that

$$(\mathbf{A} + \Delta_{\mathbf{A}})\mathbf{x} = \mathbf{y} + \Delta_{\mathbf{y}}. \tag{4.15}$$

Defining

$$\begin{array}{ll} \widehat{\mathbf{A}} & = [\mathbf{A} \mid -\mathbf{y}], \\ \widehat{\Delta} & = [\Delta_{\mathbf{A}} \mid -\Delta_{\mathbf{y}}], \end{array} \tag{4.16}$$

and

$$\widehat{\mathbf{x}} = \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}, \tag{4.17}$$

(4.15) becomes

$$(\widehat{\mathbf{A}} + \widehat{\Delta})\widehat{\mathbf{x}} = \mathbf{0}. \tag{4.18}$$

The total perturbation to be minimized is $\|\widehat{\Delta}\|_F^2$, where $\|\cdot\|_F$ is the Frobenius norm. The solution to (4.18) is (see [68]) to set $\widehat{\mathbf{x}}$ equal to the right singular vector of $\widehat{\mathbf{A}}$ associated with the minimum singular value of $\widehat{\mathbf{A}}$. The optimal $\mathbf{x}$ is then obtained by scaling $\widehat{\mathbf{x}}$ and using (4.17). In practice, to obtain this one singular vector, one may perform a complete SVD or perform a Rayleigh quotient iteration [68].

Frequently, the improvement given by the TLS solution over the OLS solution is minimal and perhaps not worth the added computational effort. This is largely because the TLS approach destroys any structure that may be inherent in the problem. For example, if $\mathbf{A}$ has

Toeplitz structure, we may desire the perturbations to also have Toeplitz structure, to reflect an assumption of Toeplitz structure on the underlying matrix $\mathbf{A} + \Delta_{\mathbf{A}}$. Unfortunately, TLS does not enforce this desire. Indeed, this is the chief complaint about the "linear methods" for relative orientation [30], [44], [78], [117], [146].

This limitation has led numerical analysts to investigate structured TLS (STLS) algorithms. Abatzoglou and Mendel [1] proposed the *constrained total least squares* (CTLS) method, in which elements of the perturbation matrix are represented as linear combinations of a smaller perturbation vector. De Moor [42] proposed an STLS method in which an affine structure is imposed on the final perturbed matrices. He solves this problem via iterative application of the generalized SVD algorithm. Lemmerling, De Moor, and Van Huffel [113] showed that these two methods were essentially equivalent. Van Huffel *et al.* [191] describe the *structured total least norms* (STLN) algorithm, which was derived from earlier work by Rosen *et al.* [157]. The STLN approach minimizes an induced p-norm of the perturbations.

The method derived in Section 4.2.2 is related to the methods of [1] and [191]. We propose a framework that incorporates perturbation models more directly than existing methods, which will provide insight into the nature of the STLS problem.

The remainder of this section is organized as follows. In Section 4.2.2, we develop the probabilistic approach. We derive the Gaussian specialization in Section 4.2.4. Finally, Section 4.2.5 provides a summary and conclusions.

4.2.2 Derivation of New Method

All of the existing methods seek to minimize a norm of the perturbation vector. In the absence of better criteria, this is of course a reasonable approach. However, in signal processing applications, we often have better metrics available to us. Specifically, we may know the prior statistics of the noise that contaminates the data measurements, or perhaps the prior statistics of the parameters to be estimated. Utilizing this prior information will lead to improved performance.

It is not obvious how one would express prior information in the CTLS or STLN approaches, although one can interpret CTLS as assuming that the noise is zero-mean AWGN. The solution given here starts directly from the given prior density function of the noise perturbations, thus more directly incorporating prior information into the model. Note that we can construct a prior PDF to simulate any induced p-norm metric of the STLN approach, so the method given here encompasses all STLN formulations.

We assume that $\mathbf{x}$ is an unknown, nonrandom parameter vector. We will derive the general formula for the estimator, which will be applicable for many model and data perturbation densities. We will then specialize to the Gaussian case, so that comparisons to the previous approaches can easily be made.

As do [1], [2], [42], [157], and [191], we model the elements of $\Delta_{\mathbf{A}}$ and $\Delta_{\mathbf{y}}$ as known, linear combinations of the elements of a random vector $\mathbf{a}$. The linear combinations are usually chosen so that the structure of the perturbations $\Delta_{\mathbf{A}}$ and $\Delta_{\mathbf{y}}$ mimic the structure of the data $\mathbf{A}$ and $\mathbf{y}$. In this situation, the structure of $\Delta_{\mathbf{A}}$ and $\Delta_{\mathbf{y}}$ in the presence of perturbations is maintained. We note in passing that this method and those in the previously cited papers allow for perturbation structure that is not identical to the structure in the data; in this case, the structure in the data is not maintained.

Let the probability that a particular noise vector realization $\mathbf{a}$ occurs be given by the prior joint PDF $p(\mathbf{a})$. We allow quite general $p(\mathbf{a})$, which represents a significant general-

ization of the STLN and CTLS approaches. Without loss of generality, we can assume that $p(\mathbf{a})$ has its mean at the origin. If the mean were nonzero, then we could simply bias the corresponding elements of $\mathbf{A}$ and $\mathbf{y}$ by values determined from $\mathbf{a}$.

We may view the observed data $\mathbf{A}$ and $\mathbf{y}$ as noise-corrupted versions of underlying or "true" nonrandom $\mathbf{A}_t$ and $\mathbf{y}_t$, where $\mathbf{A}_t\mathbf{x} = \mathbf{y}_t$, $\mathbf{A} = \mathbf{A}_t - \Delta_{\mathbf{A}}$, and $\mathbf{y} = \mathbf{y}_t - \Delta_{\mathbf{y}}$. The maximum likelihood estimates of the unknown, nonrandom parameters $\mathbf{x}$, $\mathbf{A}_t$, and $\mathbf{y}_t$ are found as the values that maximize the likelihood function $f(\mathbf{A}, \mathbf{y}\ ;\ \mathbf{x}, \mathbf{A}_t, \mathbf{y}_t)$. This is precisely equivalent to maximizing $p(\mathbf{a})$ subject to (4.15) and the structural constraints. We will call the resulting $\mathbf{x}$ the constrained maximum likelihood (CML) estimate, denoted by $\mathbf{x}_{cml}$. We have added the qualifier "constrained" to signify that this is an ML estimate obtained by considering the (nonlinear) constraints among the unknown nonrandom parameters, as well as the (linear) structured constraints in the noise model.

Care must be used here because there is an implicit assumption that there exist $\mathbf{a}$ and $\mathbf{x}$ such that equality (4.15) holds. Generally, if there are not enough degrees of freedom in the perturbations, then for some data $\mathbf{A}$ and $\mathbf{y}$ a solution may not exist. One method to guarantee a solution is to devote a new uncorrelated additive perturbation element to each element of $\mathbf{y}$, in addition to any structured perturbations already contained in $\mathbf{y}$. Of course, the values that these additional perturbations assume at the solution may be quite large, if the data does not fit the assumed structure model.

The approach developed below can be viewed as an iterative hill-climbing scheme, illustrated in Figure 4-1. We shall show that the current estimate for $\mathbf{x}$ constrains the perturbations $\mathbf{a}$ to lie on a hyperplane in the space defined by the elements of $\mathbf{a}$. At each iteration, the $\mathbf{a}$ that maximizes $p(\mathbf{a})$ on the current hyperplane is found; using this value of $\mathbf{a}$, the next estimate for $\mathbf{x}$ is found. This defines a new hyperplane, for which the process is repeated. The iterations terminate when there is no change in the hyperplane. The unconstrained maximum of $p(\mathbf{a})$ will generally not be on this terminal constraint hyperplane. In the specialized Gaussian case, we will show that the optimal constrained $\mathbf{a}$ can in fact be derived from $\mathbf{x}$ in closed-form, thus simplifying the calculations.

To develop the method, the key transformation (similar to one performed in [1], [2], and [191]) is to represent the vector $\widehat{\Delta}\widehat{\mathbf{x}}$ in (4.18) in terms of $\mathbf{a}$. This is accomplished by defining a matrix $\mathbf{X}$ such that

$$\mathbf{X}\mathbf{a} = \widehat{\Delta}\widehat{\mathbf{x}}. \tag{4.19}$$

The matrix $\mathbf{X}$ consists of the elements of $\mathbf{x}$, suitably repeated. Thus, $\mathbf{X}$ captures the structure of the perturbations in $\widehat{\Delta}$.

Note that we can have arbitrary linear combinations of the perturbation elements for each entry of $\widehat{\Delta}$. For example, we could have

$$\begin{aligned} \widehat{\Delta}\widehat{\mathbf{x}} &= \begin{bmatrix} 2a_1 + a_2 & a_3 \\ -a_3 + a_2 & a_1 \\ a_1 & 2a_2 + a_3 \end{bmatrix} \begin{bmatrix} x_1 \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} 2x_1 & x_1 & 1 \\ 1 & x_1 & -x_1 \\ x_1 & 2 & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \mathbf{X}\mathbf{a}. \end{aligned} \tag{4.20}$$

The general rule for constructing $\mathbf{X}$ is: if b_k is the coefficient of a_k in the $(i,j)^{\text{th}}$ element of $\widehat{\Delta}$, then b_k is also the coefficient of x_j in the $(i,k)^{\text{th}}$ element of $\mathbf{X}$. This rule originated in [157] and [191]. It is unique, up to a trivial reordering of the elements of $\mathbf{a}$. For small problems, such as (4.20), it easier to do the manipulation by inspection.

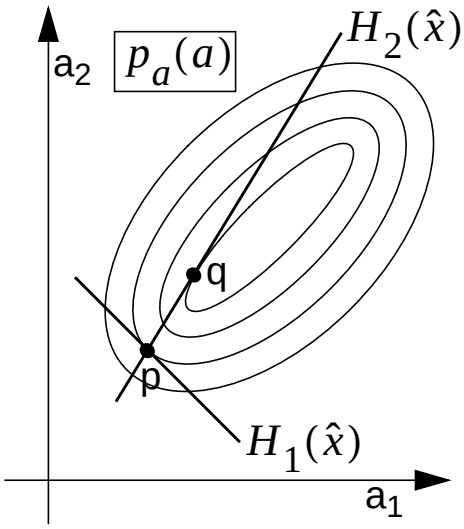

Figure 4-1: Conceptual view of the CML optimization method. Initially, $\mathbf{a}$ is constrained to hyperplane $H(\widehat{\mathbf{x}}^0)$, corresponding to the initial estimate $\widehat{\mathbf{x}}^0$. The optimal constrained $\mathbf{a}$ and a new estimate $\widehat{\mathbf{x}}^1$ are found at point $\mathbf{p}$. A new hyperplane $H(\widehat{\mathbf{x}}^1)$ is derived at $\mathbf{p}$, and the constrained optimization is repeated to arrive at $\mathbf{q}$.

This approach will be valid as long as $\mathbf{X}$ has full row rank. Otherwise, we are not guaranteed that a solution $\mathbf{a}$ can be found to exactly satisfy the constraints at values of $\widehat{\mathbf{x}}$ where $\mathbf{X}$ loses rank. This situation was considered in greater detail in [63]; for simplicity, in this thesis we assume that $\mathbf{X}$ has full row rank.

We now state and prove the main result of this section.

Proposition: The constrained maximum likelihood estimate $\mathbf{x}_{cml}$ is given by

$$\mathbf{x}_{cml} = \arg\max_{\mathbf{x}} \left(\max_{\mathbf{b}} \; p(-\mathbf{X}^+ \widehat{\mathbf{A}} \widehat{\mathbf{x}} + \mathbf{B}_{\mathbf{x}} \mathbf{b}) \right), \tag{4.21}$$

where $\mathbf{X}$ is the matrix constructed from $\mathbf{x}$ as in (4.19), and $\mathbf{B}_{\mathbf{x}}$ is a basis of the null space of $\mathbf{X}$.

Proof. Multiplying out (4.18) and substituting in (4.19) gives

$$\mathbf{X}\mathbf{a} = -\widehat{\mathbf{A}}\widehat{\mathbf{x}}. \tag{4.22}$$

This is the equation that constrains $\mathbf{a}$ to a hyperplane specified by $\mathbf{x}$. Since we have assumed that there is at least one $\mathbf{a}$ that satisfies (4.22), the most general solution is

$$\mathbf{a} = -\mathbf{X}^+ \widehat{\mathbf{A}} \widehat{\mathbf{x}} + \mathbf{B}_{\mathbf{x}} \mathbf{b}, \tag{4.23}$$

where $\mathbf{B}_{\mathbf{x}}$ is a basis for the null space of $\mathbf{X}$, $\mathbf{b}$ is an arbitrary vector, and $\mathbf{X}$ has full row rank. The basis $\mathbf{B}_{\mathbf{x}}$ will in general depend on the particular value of $\mathbf{x}$, because $\mathbf{X}$ depends on $\mathbf{x}$. With the $\mathbf{B}_{\mathbf{x}}$ corresponding to the particular guess $\mathbf{x}$, we can implement the likelihood maximization in the form shown in (4.21) by substituting (4.23) into the likelihood function $p(\mathbf{a})$ and maximizing over $\mathbf{x}$ and $\mathbf{b}$. Thus, through our choice of $\mathbf{x}$ and $\mathbf{a}$, we have effectively picked the optimal $\mathbf{x}$, $\mathbf{A}_t$, and $\mathbf{y}_t$ estimates according to the CML criteria (which completes the proof).

If the structure of $\mathbf{X}$ is simple enough, $\mathbf{B_x}$ can be expressed symbolically as a function of $\mathbf{x}$. If, due to the complexity of $\mathbf{X}$, it is not desirable or feasible to calculate $\mathbf{B_x}$ symbolically, we must resort to a numerical scheme, where we have a current guess $\mathbf{x}$, and then find the basis $\mathbf{B_x}$ numerically. Given the availability of general-purpose multivariable optimization software (i.e. MATLAB's "fmins" function, which implements a Nelder-Mead simplex search method [151]), there is no notional difference between the cases where $\mathbf{B_x}$ is determined symbolically or numerically (although the former case may be less computationally demanding).

4.2.3 Gradient Method

If we desire a better optimization technique (than, say, the Nelder-Mead method), we must provide gradient information. Assume that $p(\mathbf{a})$ is differentiable and that for a given guess $\mathbf{x}$ we have determined the optimal $\mathbf{a}$ by varying $\mathbf{b}$ in (4.21). To further increase $p(\mathbf{a})$, we must vary our guess $\mathbf{x}$ so that the change in $\mathbf{a}$ is as aligned as possible with $dp(\mathbf{a})/d\mathbf{a}$. Straightforward calculus allows us to determine the change in $\mathbf{x}$ that will achieve this.

For a particular guess $\mathbf{x}$, we start with (4.22), where we assume that at least one $\mathbf{a}$ exists that exactly solves this equation. Taking differentials,

$$\begin{aligned} \mathbf{X}\, d\mathbf{a} + d\mathbf{X}\, \mathbf{a} &= -\widehat{\mathbf{A}}\, d\widehat{\mathbf{x}} \\ \mathbf{X}\, d\mathbf{a} + \left(\textstyle\sum_{i=1}^{N} \frac{\partial \mathbf{X}}{\partial x_i}\, dx_i\right) \mathbf{a} &= -\mathbf{A}\, d\mathbf{x} \\ \mathbf{X}\, d\mathbf{a} + \left[\frac{\partial \mathbf{X}}{\partial x_1}\mathbf{a} \mid \ldots \mid \frac{\partial \mathbf{X}}{\partial x_N}\mathbf{a}\right] d\mathbf{x} &= -\mathbf{A}\, d\mathbf{x}, \end{aligned} \tag{4.24}$$

or finally

$$[\mathbf{X} \mid \mathbf{M}] \begin{bmatrix} d\mathbf{a} \\ d\mathbf{x} \end{bmatrix} = \mathbf{0} \tag{4.25}$$

with

$$\mathbf{M} = \mathbf{A} + \left[\frac{\partial \mathbf{X}}{\partial x_1}\mathbf{a} \mid \ldots \mid \frac{\partial \mathbf{X}}{\partial x_N}\mathbf{a}\right]. \tag{4.26}$$

By (4.25), the vector of differentials $[d\mathbf{a}^T\ d\mathbf{x}^T]^T$ must be confined to the null space of $[\mathbf{X} \mid \mathbf{M}]$. Let the matrix $\mathbf{B}$ represent a basis for this null space, and let $\mathbf{v}$ be an arbitrary vector. Conformally partitioning $\mathbf{B}$ into upper part $\mathbf{B}_1$ and lower part $\mathbf{B}_2$, we have

$$\begin{bmatrix} d\mathbf{a} \\ d\mathbf{x} \end{bmatrix} = \mathbf{B}\mathbf{v} = \begin{bmatrix} \mathbf{B}_1 \\ \mathbf{B}_2 \end{bmatrix} \mathbf{v}. \tag{4.27}$$

To move uphill, we would choose $d\mathbf{a} = s\, dp(\mathbf{a})/d\mathbf{a}$ if possible, where s is a positive step size. However, due to the constraint (4.27), it may not be possible to go exactly uphill. For a small enough step size, the best approach is to move in a direction as close as possible to the $dp(\mathbf{a})/d\mathbf{a}$ direction. The step direction that achieves this is found by projecting $dp(\mathbf{a})/d\mathbf{a}$ onto the subspace in which $d\mathbf{a}$ lies. We thus have

$$\mathbf{v} = s\mathbf{B}_1^{+} \frac{dp(\mathbf{a})}{d\mathbf{a}}, \tag{4.28}$$

which, combined with (4.27), gives

$$d\mathbf{x} = s\mathbf{B}_2\mathbf{B}_1^{+} \frac{dp(\mathbf{a})}{d\mathbf{a}}. \tag{4.29}$$

In a competent implementation, the step size s must of course be adaptively updated during the algorithm by taking trial steps, or perhaps by implementing the well-known Armijo rule [20].

In summary, a local maximum of the constrained maximum likelihood problem (4.21) is found by the iteration:

Gradient Iteration

- Given a current estimate $\mathbf{x}$, find the optimal $\mathbf{a}$ in the hyperplane by maximizing $p(\mathbf{a})$ over $\mathbf{b}$ using (4.23).
- Evaluate $\mathbf{M}$ using (4.26), calculate the null space basis $\mathbf{B}$, and evaluate $\frac{dp(\mathbf{a})}{d\mathbf{a}}$.
- Set $\mathbf{x} = \mathbf{x} + d\mathbf{x}$, where $d\mathbf{x}$ is obtained from (4.29), and repeat until convergence.

For notational simplicity, we do not distinguish between infinitesimal and incremental perturbations.

We note that in this method the $\mathbf{b}$ are the only independent variables in the iteration, whereas in the STLN approach, the variables $\mathbf{a}$ and $\mathbf{x}$ are treated as independent. If $\mathbf{a}$ is large, the latter approach results in a considerably larger search space than in the method given here.

To assess the computational cost of each iteration, let the sizes of the various vectors and matrices be as follows: $\mathbf{A}$ and $\Delta_{\mathbf{A}}$ are $M \times N$, $\mathbf{x}$ is $N \times 1$, $\mathbf{a}$ is $Q \times 1$, $\mathbf{X}$ is $M \times Q$, $\mathbf{B}_{\mathbf{x}}$ is $Q \times (Q-M)$, and $\mathbf{b}$ is $(Q-M) \times 1$. The sizes of $\mathbf{B}_{\mathbf{x}}$ and $\mathbf{b}$ assume that $\mathbf{X}$ has full row rank, and that $Q \geq M \geq N$. The maximization over $\mathbf{b}$ in the first step of the iteration requires a $(Q-M)$-dimensional search. We assume that $\mathbf{a}$ and $dp(\mathbf{a})/d\mathbf{a}$ are calculated as byproducts of this optimization. The number of operations to form $\mathbf{M}$ is $O(MNQ)$, while calculation of $\mathbf{B}$ and $\mathbf{B}_1^+$ require $O((Q+N)^2(Q+N-M))$, $d\mathbf{x}$ requires $O((Q+N-M)(Q+N))$, and $\mathbf{B}_{\mathbf{x}}$ requires $O(Q^2(Q-M))$ operations. At worst, when $Q = M = N$, this is $O(Q^3)$ operations plus a $(Q-M)$-dimensional search per iteration.

4.2.4 Specialization to the Gaussian and Similar PDFs

If $p(\mathbf{a})$ is purely a nonincreasing function of the scalar $J = \mathbf{a}^T\mathbf{R}^{-1}\mathbf{a}$ for some positive definite symmetric matrix $\mathbf{R}$, as in the zero-mean Gaussian case, then we can considerably simplify (4.21). For the remainder of this section we will assume the PDF is zero-mean Gaussian. However, all comments in this section do in fact apply to the more general class of PDFs.

Minimizing the quadratic form $\mathbf{a}^T\mathbf{R}^{-1}\mathbf{a}$ subject to the linear constraints (4.22) gives the standard result

$$\mathbf{a} = -\mathbf{R}\mathbf{X}^T(\mathbf{X}\mathbf{R}\mathbf{X}^T)^{-1}\widehat{\mathbf{A}}\widehat{\mathbf{x}}. \tag{4.30}$$

The minimum J is therefore

$$J = \widehat{\mathbf{x}}^T\widehat{\mathbf{A}}^T(\mathbf{X}\mathbf{R}\mathbf{X}^T)^{-1}\widehat{\mathbf{A}}\widehat{\mathbf{x}}, \tag{4.31}$$

which is the function of $\widehat{\mathbf{x}}$ that has to be minimized for the Gaussian case. We note that this objective function (4.31) is effectively the one obtained in [1], [2]. In fact, this derivation shows that Abatzoglou and Mendel's approach can be interpreted as solving for the optimal parameters $\mathbf{x}$ while assuming a Gaussian prior density for the perturbations $\mathbf{a}$. Therefore, for the Gaussian case, we have eliminated all "nuisance" parameters ($\mathbf{a}$ and $\mathbf{b}$) from the

estimation problem. This should be contrasted to the STLN approach, where the nuisance parameters $\mathbf{a}$ must be updated at every iteration step along with the solution vector $\mathbf{x}$.

Examining (4.31), we see that J is entirely a function of $\mathbf{x}$. We can consider forming the gradient vector $dJ/d\mathbf{x}$, and performing a steepest descent minimization. We show in [63] that the elements of the gradient vector are given by

$$\begin{aligned}\frac{\partial J}{\partial x_i} = \Big(2\widehat{\mathbf{A}}_i^T - \widehat{\mathbf{x}}^T\widehat{\mathbf{A}}^T(\mathbf{X}\mathbf{R}\mathbf{X}^T)^{-1} \\ \cdot (\tfrac{\partial \mathbf{X}}{\partial x_i}\mathbf{R}\mathbf{X}^T + \mathbf{X}\mathbf{R}\tfrac{\partial \mathbf{X}^T}{\partial x_i})\Big)(\mathbf{X}\mathbf{R}\mathbf{X}^T)^{-1}\widehat{\mathbf{A}}\widehat{\mathbf{x}},\end{aligned} \tag{4.32}$$

where $\widehat{\mathbf{A}}_i$ is the i^{th} column of $\widehat{\mathbf{A}}$. To evaluate the gradient, all that is needed are the $\partial\mathbf{X}/\partial x_i$, and these are derived easily from $\mathbf{X}$.

4.2.5 Summary and Conclusions

This section was concerned with the problem of obtaining maximum likelihood solutions to inconsistent linear equations, subject to a given structure of the required perturbations, and a prior perturbation PDF. By formulating the optimality criterion in a probabilistic framework, we were able to propose simple cost metrics which could then be optimized by general purpose optimization routines.

We gave several formulas for use by general purpose optimization routines, including gradient formulas for the case of differentiable PDFs. If the PDF is not differentiable at all points, optimization can still be performed by also considering the behavior at corners of the PDF.

We also showed here and in [63] that the existing TLS and CTLS methods were special cases of the CML approach; for CTLS, the derivation appears simpler than in [1], [2]. Furthermore, with the CML approach, it is apparent what happens when no feasible solution exists or if there are infeasible manifolds.

In the next section, we apply the CML technique to the problem of estimating the motion of a planar scene using brightness gradients.

4.3 CML Estimate of Motion Parameters for Planar Scene

4.3.1 CML Approach

Recall from Section 2.3.3 that for gradient-based relative orientation for a planar scene, the brightness change constraint equation (BCCE) (2.30) led to the equation

$$c + \mathbf{v}\cdot\boldsymbol{\omega} + (\mathbf{r}\cdot\mathbf{n})(\mathbf{s}\cdot\mathbf{t}) = 0. \tag{4.33}$$

for every pixel in the image. In (4.33), c is the brightess time gradient, $\boldsymbol{\omega}$ is the instantaneous rotation, $\mathbf{t}$ is the translation, $\mathbf{n}$ is the planar normal vector, and $\mathbf{r}$, $\mathbf{s}$, and $\mathbf{v}$ functions of the pixel coordinates. Negahdaripour and Horn [136] find a least-squares estimate of $\boldsymbol{\omega}$, $\mathbf{n}$, and $\mathbf{t}$ by minimizing

$$\arg\min_{\boldsymbol{\omega},\mathbf{t},\mathbf{n}} \; J = \int\int (c + \mathbf{v}\cdot\boldsymbol{\omega} + (\mathbf{r}\cdot\mathbf{n})(\mathbf{s}\cdot\mathbf{t}))^2 \,\mathrm{dx}\,\mathrm{dy}. \tag{4.34}$$

By considering all the image points, robustness against noise is obtained. However, (4.34) is not a maximum likelihood estimator, so the quality of the estimates obtained via (4.34) is

called into question. In what follows, we will derive the exact MLE for the motion parameters, and compare to an efficient approximate method suitable for FPGA implementation.

It will prove advantageous to work with an alternate form of (4.33); this is given by

$$c_t + \mathbf{r}^T \mathbf{P}_t \mathbf{s}_t = 0, \tag{4.35}$$

where

$$\mathbf{P}_t = \begin{bmatrix} p_1 & p_2 & p_3 \\ p_4 & p_5 & p_6 \\ p_7 & p_8 & p_9 \end{bmatrix}, \tag{4.36}$$

and where the t subscript indicates the "true" parameters. In this equation, c_t is the true time-gradient of brightness $E_{t,t}$, $\mathbf{P}_t$ is the true essential matrix, $\mathbf{r} = [x,\ y,\ 1]^T$, and

$$\mathbf{s}_t = \begin{bmatrix} -E_{t,x} \\ -E_{t,y} \\ xE_{t,x} + yE_{t,y} \end{bmatrix}. \tag{4.37}$$

(The notation $E_{t,t}$, $E_{t,x}$, and $E_{t,y}$ stands for the true first partial derivatives of the brightness E, not the second partial derivatives.) Also note that in Section 2.3.3, the essential matrix was denoted by "$\mathbf{E}$." We have relabeled it to "$\mathbf{P}$" here in this section to avoid confusion.

Negahdaripour and Horn showed how to recover $\boldsymbol{\omega}$, $\mathbf{t}$, and $\mathbf{n}$ uniquely from $\mathbf{P}_t$; thus if we find a maximum likelihood estimator of $\mathbf{P}_t$, we may use the MLE invariance property to claim that we have an MLE for $\boldsymbol{\omega}$, $\mathbf{t}$, and $\mathbf{n}$.

Rearrange (4.35) into the form

$$\mathbf{a}_t^T \mathbf{p} = -c_t, \tag{4.38}$$

where,

$$\mathbf{p} = [p_1\ p_2\ p_3\ p_4\ p_5\ p_6\ p_7\ p_8]^T, \tag{4.39}$$

$$\mathbf{a}_t = \begin{bmatrix} -xE_{t,x} \\ -xE_{t,y} \\ x^2E_{t,x} + xyE_{t,y} \\ -yE_{t,x} \\ -yE_{t,y} \\ xyE_{t,x} + y^2E_{t,y} \\ -E_{t,x} \\ -E_{t,y} \end{bmatrix}. \tag{4.40}$$

Note that we have used (4.37) and have set $p_9 = 0$.

For each pixel, we observe noisy versions of $E_{t,t}$, $E_{t,x}$, and $E_{t,y}$, denoted by $E_{o,t}$, $E_{o,x}$, and $E_{o,y}$, and desire the maximum likelihood estimates $\mathbf{p}_{ml}$. (We could also obtain $E_{ml,t}$, $E_{ml,x}$, and $E_{ml,y}$ if desired.) Assume the noisy observations are obtained from the true values by

$$\begin{aligned} E_{o,x} &= E_{t,x} - \Delta E_x \\ E_{o,y} &= E_{t,y} - \Delta E_y \\ E_{o,t} &= E_{t,t} - \Delta E_t, \end{aligned} \tag{4.41}$$

where, for simplicity, we assume that ΔE_x, ΔE_y, and ΔE_t are all distributed as $N(0, \sigma^2)$. For each pixel, take all the vectors $\mathbf{a}_t^T$ obtained from (4.40) and all the scalars c_t and stack them vertically, obtaining

$$\mathbf{A}_t \mathbf{p}_t = -\mathbf{c}_t, \tag{4.42}$$

with

$$\mathbf{A}_t = \begin{bmatrix} \mathbf{a}_{t1}^T \\ \vdots \\ \mathbf{a}_{tn}^T \end{bmatrix}, \quad \mathbf{c}_t = \begin{bmatrix} c_{t1} \\ \vdots \\ c_{tn} \end{bmatrix}. \tag{4.43}$$

We perform similar operations on the observed data:

$$\mathbf{A}_o = \begin{bmatrix} \mathbf{a}_{o1}^T \\ \vdots \\ \mathbf{a}_{on}^T \end{bmatrix}, \quad \mathbf{c}_o = \begin{bmatrix} c_{o1} \\ \vdots \\ c_{on} \end{bmatrix}, \tag{4.44}$$

with

$$\mathbf{a}_o = \begin{bmatrix} -xE_{o,x} \\ -xE_{o,y} \\ x^2E_{o,x} + xyE_{o,y} \\ -yE_{o,x} \\ -yE_{o,y} \\ xyE_{o,x} + y^2E_{o,y} \\ -E_{o,x} \\ -E_{o,y} \end{bmatrix}. \tag{4.45}$$

From (4.41), we have

$$\begin{aligned} \mathbf{A}_o &= \mathbf{A}_t - \Delta\mathbf{A} \\ \mathbf{c}_o &= \mathbf{c}_t - \Delta\mathbf{c}. \end{aligned} \tag{4.46}$$

Making the definitions,

$$\begin{aligned} \widehat{\mathbf{A}}_t &\triangleq [\mathbf{A}_t \mid \mathbf{c}_t], \\ \widehat{\mathbf{A}}_o &\triangleq [\mathbf{A}_o \mid \mathbf{c}_o], \\ \widehat{\Delta} &\triangleq [\Delta\mathbf{A} \mid \Delta\mathbf{c}], \\ \widehat{\mathbf{p}}^T &\triangleq [\mathbf{p}^T \mid 1]^T, \end{aligned} \tag{4.47}$$

we have finally

$$\left(\widehat{\mathbf{A}}_o + \widehat{\Delta}\right)\widehat{\mathbf{p}} = \mathbf{0}. \tag{4.48}$$

If $\Delta\mathbf{A}$ were not structured, we could simply use TLS to solve for the optimal $\mathbf{p}$. Since it is structured, we must use a CML formulation, as detailed in Section 4.2. To use CML, we make the rearrangement

$$\widehat{\Delta}\widehat{\mathbf{p}} = \mathbf{X}\boldsymbol{\alpha}, \tag{4.49}$$

where

$$\boldsymbol{\alpha} = \begin{bmatrix} \Delta E_{x1} \\ \Delta E_{y1} \\ \Delta E_{t1} \\ \vdots \\ \Delta E_{xn} \\ \Delta E_{yn} \\ \Delta E_{tn} \end{bmatrix}, \tag{4.50}$$

and

$$\mathbf{X} = \begin{bmatrix} \mathbf{b}_1^T & & \\ & \ddots & \\ & & \mathbf{b}_n^T \end{bmatrix} \tag{4.51}$$

with

$$\mathbf{b}_i = \begin{bmatrix} -xp_1 + x^2p_3 - yp_4 + xyp_6 - p_7 \\ -xp_2 + xyp_3 - yp_5 + y^2p_6 - p_8 \\ 1 \end{bmatrix}. \tag{4.52}$$

That is, $\mathbf{X}$ is block diagonal, with each block $\mathbf{b}_i$ depending only on pixel i's coordinates and the (global) value of $\mathbf{p}$.

The CML estimate of $\mathbf{p}$ is found via

$$\widehat{\mathbf{p}}_{cml} = \arg\min_{\mathbf{p}} \widehat{\mathbf{p}}^T \widehat{\mathbf{A}}_o^T \left(\mathbf{X}\mathbf{X}^T\right)^{-1} \widehat{\mathbf{A}}_o \widehat{\mathbf{p}}. \tag{4.53}$$

Using (4.44) and (4.51), we simplify (4.53) as

$$\widehat{\mathbf{p}}_{cml} = \arg\min_{\mathbf{p}} \widehat{\mathbf{p}}^T \left(\sum_{i=1}^{n} \frac{\widehat{\mathbf{a}}_{oi}\widehat{\mathbf{a}}_{oi}^T}{\|\mathbf{b}_i\|^2} \right) \widehat{\mathbf{p}}, \tag{4.54}$$

which may also be expressed as

$$\widehat{\mathbf{p}}_{cml} = \arg\min_{\mathbf{p}} \sum_{i=1}^{n} \frac{\|\widehat{\mathbf{a}}_{oi}^T \widehat{\mathbf{p}}\|^2}{\|\tilde{\mathbf{B}}_i \widehat{\mathbf{p}}^T\|^2} \tag{4.55}$$

with

$$\widehat{\mathbf{a}}_o = \begin{bmatrix} -xE_{o,x} \\ -xE_{o,y} \\ x^2E_{o,x} + xyE_{o,y} \\ -yE_{o,x} \\ -yE_{o,y} \\ xyE_{o,x} + y^2E_{o,y} \\ -E_{o,x} \\ -E_{o,y} \\ E_{o,t} \end{bmatrix}, \tag{4.56}$$

and

$$\tilde{\mathbf{B}}_i = \begin{bmatrix} -x & 0 & x^2 & -y & 0 & xy & -1 & 0 & 0 \\ 0 & -x & xy & 0 & -y & y^2 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{4.57}$$

Numerical procedures similar to the ones developed in Section 4.2 can now be applied to calculated the estimate $\widehat{\mathbf{p}}_{cml}$.

4.3.2 Bias Removal Approach

We saw in the previous section the CML approach for obtaining the motion parameters of a planar scene. While the solution obtained this way can be considered optimal, one must resort to iterative techniques for the solution. At the other extreme of course are the "linear" techniques, which are computationally efficient because they ignore inconvenient nonlinear constraints.

It would appear that most researchers tend to fall into one of these two camps. However, through the use of straightforward estimation concepts, we can explore an intermediate approach.

Recall that in Section 2.3.3, we showed how to solve for the motion parameters given the matrix $\mathbf{A}$ and the vector $\mathbf{c}$. These quantities are derived from the $\mathbf{r}$, $\mathbf{s}$, and c quantities from at least eight pixels (usually much more), to form the overdetermined set of linear equations (4.42). This equation is solved in the least squares sense, yielding estimates of the $\{e_i'\}$. These values were then used to construct the essential matrix, from which the motion parameters could be derived in a closed-form, using the approach given in [136].

Thus, the approach taken in [136] and given in Section 2.3.3 is to solve for the $\{e_i'\}$ via $\mathbf{e}' = -\mathbf{A}^{+}\mathbf{c}$. Unfortunately, in this situation, the pseudoinverse solution is not the optimal choice because the perturbations are not restricted to the right hand side vector of (4.42). That is, the elements of $\mathbf{A}$ have significant errors in them because they were formed from image brightness data. Actually, the elements of $\mathbf{A}$ are derived from the spatial gradients of the image, which are usually more noisy than the image brightness data. A TLS approach, which allows changes in the elements of $\mathbf{A}$ is also generally not appropriate, because TLS ignores the perturbation structure constraints.

Instead of exactly modeling the errors in the data, as was done for the CML approach, let us instead form $\mathbf{A}$ using the linear method, and then try to "repair" it, in a statistical sense. To do this, we derive the expected error of the elements of the normal equation matrix, and then merely subtract this bias off. This approach has been used before for orientation problems by Kanatani and his cohorts [101], [139].

Let us consider the effect of gradient errors on the estimates of the motion parameters. The matrix $\mathbf{A}^T\mathbf{A}$ which appears in the pseudoinverse solution is composed of a sum of outer products

$$\mathbf{A}^t\mathbf{A} = \sum_i^N \mathbf{a}\mathbf{a}^T, \tag{4.58}$$

where the summation is taken over N independent pixels. Since we set $p_9' = 0$, we can redefine $\mathbf{a}$ as

$$\mathbf{a} = (\mathbf{r} \otimes \mathbf{s})^T = [r_1 s_1 \; r_1 s_2 \; r_1 s_3 \; r_2 s_1 \; r_2 s_2 \; r_2 s_3 \; r_3 s_1 \; r_3 s_2]^T, \tag{4.59}$$

where we recall that $\mathbf{r} = [x \; y \; 1]^T$ and $\mathbf{s}$ is given in (2.33).

Assume that the spatial gradients E_x and E_y are perturbed by additive white Gaussian noise, w_x and w_y, respectively. We assume for simplicity that w_x and w_y are uncorrelated with zero mean and known variance σ^2. Because of the noise, we work with

$$\tilde{\mathbf{s}} = \mathbf{s} + \Delta\mathbf{s} = \mathbf{s} + \begin{bmatrix} -w_x \\ -w_y \\ xw_x + yw_y \end{bmatrix}. \tag{4.60}$$

The noise perturbs $\mathbf{a}$, so we are forced to work with

$$\tilde{\mathbf{a}} = \mathbf{a} + \Delta\mathbf{a}, \tag{4.61}$$

where $\Delta\mathbf{a}$ is defined as

$$\Delta\mathbf{a} = \mathbf{r} \otimes \Delta\mathbf{s} = \begin{bmatrix} r_1\Delta s_1 \\ r_1\Delta s_2 \\ r_1\Delta s_3 \\ r_2\Delta s_1 \\ r_2\Delta s_2 \\ r_2\Delta s_3 \\ r_3\Delta s_1 \\ r_3\Delta s_2 \end{bmatrix} = \begin{bmatrix} -xw_x \\ -xw_y \\ x^2w_x + xyw_y \\ -yw_x \\ -yw_y \\ xyw_x + y^2w_y \\ -w_x \\ -w_y \\ xw_x + yw_y \end{bmatrix}. \tag{4.62}$$

We wish to form $\mathbf{a}\mathbf{a}^T$ but instead we must form

$$\tilde{\mathbf{a}}\tilde{\mathbf{a}}^T = (\mathbf{a} + \Delta\mathbf{a})(\mathbf{a} + \Delta\mathbf{a})^T = \mathbf{a}\mathbf{a}^T + \mathbf{a}\Delta\mathbf{a}^T + \Delta\mathbf{a}\mathbf{a}^T + \Delta\mathbf{a}\Delta\mathbf{a}^T \tag{4.63}$$

When we sum the outer products $\tilde{\mathbf{a}}\tilde{\mathbf{a}}^T$ over many pixels to form an estimate $\tilde{\mathbf{A}}^T\tilde{\mathbf{A}}$ of $\mathbf{A}^T\mathbf{A}$ in (4.58), we get an averaging effect, so we can use expected values. Taking the expectation with respect to w_x and w_y of both sides of (4.63) gives

$$E\left\{\tilde{\mathbf{a}}\tilde{\mathbf{a}}^T\right\} = \mathbf{a}\mathbf{a}^T + \mathbf{a}E\left\{\Delta\mathbf{a}^T\right\} + E\left\{\Delta\mathbf{a}\right\}\mathbf{a}^T + E\left\{\Delta\mathbf{a}\Delta\mathbf{a}^T\right\}, \tag{4.64}$$

where $E\{\cdot\}$ denotes the expectation or ensemble average. Since $E\{\Delta\mathbf{a}\} = \mathbf{0}$, we have

$$E\left\{\tilde{\mathbf{a}}\tilde{\mathbf{a}}^T\right\} = \mathbf{a}\mathbf{a}^T + E\left\{\Delta\mathbf{a}\Delta\mathbf{a}^T\right\}. \tag{4.65}$$

Thus, on average, our estimate of $\mathbf{a}\mathbf{a}^T$ is biased by the covariance matrix of $\Delta\mathbf{a}$. Define this covariance matrix as

$$\mathbf{\Delta} \triangleq E\left\{\Delta\mathbf{a}\Delta\mathbf{a}^T\right\}. \tag{4.66}$$

We can determine the expected values of the entries of $\mathbf{\Delta}$ with the simplifying assumption that the x and y coordinates of the pixels are symmetric about the origin of the image plane. In this case,

$$\sum_i^N x^k = \sum_i^N y^k = 0, \tag{4.67}$$

for odd k. Since we assumed that w_x and w_y were zero mean Gaussian random variables, we have also

$$E\left\{\sum_i^N w_x^k\right\} = E\left\{\sum_i^N w_y^k\right\} = 0. \tag{4.68}$$

for odd k. The elements of the matrix $\mathbf{\Delta}$ are composed of sums of terms of the form

$$x^{k_1}y^{k_2}w_x^{k_3}w_y^{k_4}, \tag{4.69}$$

and thus on average, only those elements of $\mathbf{\Delta}$ with even exponents k_1, k_2, k_3, and k_4 are nonzero. Making the definitions

$$\alpha = \sum_i^N x^2 = \sum_i^N y^2 \tag{4.70}$$

and

$$\beta = \sum_i^N x^4 + \sum_i^N x^2 y^2 = \sum_i^N y^4 + \sum_i^N x^2 y^2, \tag{4.71}$$

we can determine the expected value of the elements of $\mathbf{\Delta}$. For example,

$$E\{\Delta_{11}\} = \alpha\sigma^2. \tag{4.72}$$

Similarly,

$$E\{\Delta_{22}\} = E\{\Delta_{44}\} = E\{\Delta_{55}\} = \alpha\sigma^2, \tag{4.73}$$

$$E\{\Delta_{33}\} = E\{\Delta_{66}\} = \beta\sigma^2, \tag{4.74}$$

$$E\{\Delta_{77}\} = E\{\Delta_{88}\} = N\sigma^2, \tag{4.75}$$

$$E\{\Delta_{37}\} = E\{\Delta_{73}\} = E\{\Delta_{68}\} = E\{\Delta_{86}\} = -\alpha\sigma^2. \tag{4.76}$$

All other $E\{\Delta_{ij}\} = 0$. We thus have obtained the average bias in the formation of $\mathbf{A}^T\mathbf{A}$. A similar analysis of the expected bias can be performed on $\mathbf{A}^T\mathbf{c}$, the right hand side vector formed during the pseudoinverse solution. If we assume that the error in the time derivatives $\mathbf{c}$ are uncorrelated with the errors in the spatial derivatives, then it can be shown that the expected bias in $\mathbf{A}^T\mathbf{c}$ is zero.

A straightforward procedure thus presents itself for removing the bias in solving the normal equations. With a known noise variance σ^2, we simply subtract off the average $\mathbf{\Delta}$ determined from (4.72) through (4.76) from the matrix $\tilde{\mathbf{A}}^T\tilde{\mathbf{A}}$ formed from the gradient data. We then solve for the motion parameters from

$$\mathbf{e}' = -\left(\tilde{\mathbf{A}}^T\tilde{\mathbf{A}} - \mathbf{\Delta}\right)^{-1}\tilde{\mathbf{A}}^T\mathbf{c}. \tag{4.77}$$

We note that any performance improvement over the conventional linear approach is available at the additional cost of only twelve subtractions, which is negligible when compared to the formation of the matrix $\tilde{\mathbf{A}}^T\tilde{\mathbf{A}}$ and the solving of the ensuing eigenvalue/vector problem.

4.3.3 Experimental Comparison

In this section, using a series of Monte Carlo runs, we compare the performance of the various methods for planar scene motion recovery. The four methods were the OLS linear method of Section 2.3.3, the CML method of Section 4.2, the bias removal (OLSB) method of Section 4.3.2, and the simple TLS method mentioned in Section 4.3.2.

For each run, a synthetic scene consisting of Gaussian shaped blobs was generated. The scene was transformed via the perspective equation to generate the image plane data. Analytic spatial and time derivatives were also calculated, based on the true motion parameters. Noise was then added to the true gradients. The motion parameter estimates were then solved for using the various approaches

This procedure was repeated for 10 different noise realizations at each signal-to-noise ratio (SNR). To measure the performance of each approach, we were interested in the angular errors of our estimates of $\boldsymbol{\omega}$, $\hat{\mathbf{n}}$, and $\hat{\mathbf{t}}$, as well as the relative error in the estimates of the magnitudes $|\boldsymbol{\omega}|$ and $|\mathbf{n}||\mathbf{t}|$. A very wide field of view (80 degrees) was assumed. Figures 4-2 - 4-9 show sample images and derivatives, as well as the various motion fields and processing results for different motions. We note from these figures that generally, the maximum likelihood and bias removal methods attain about the same level of performance, and that both perform better than the OLS and TLS methods.

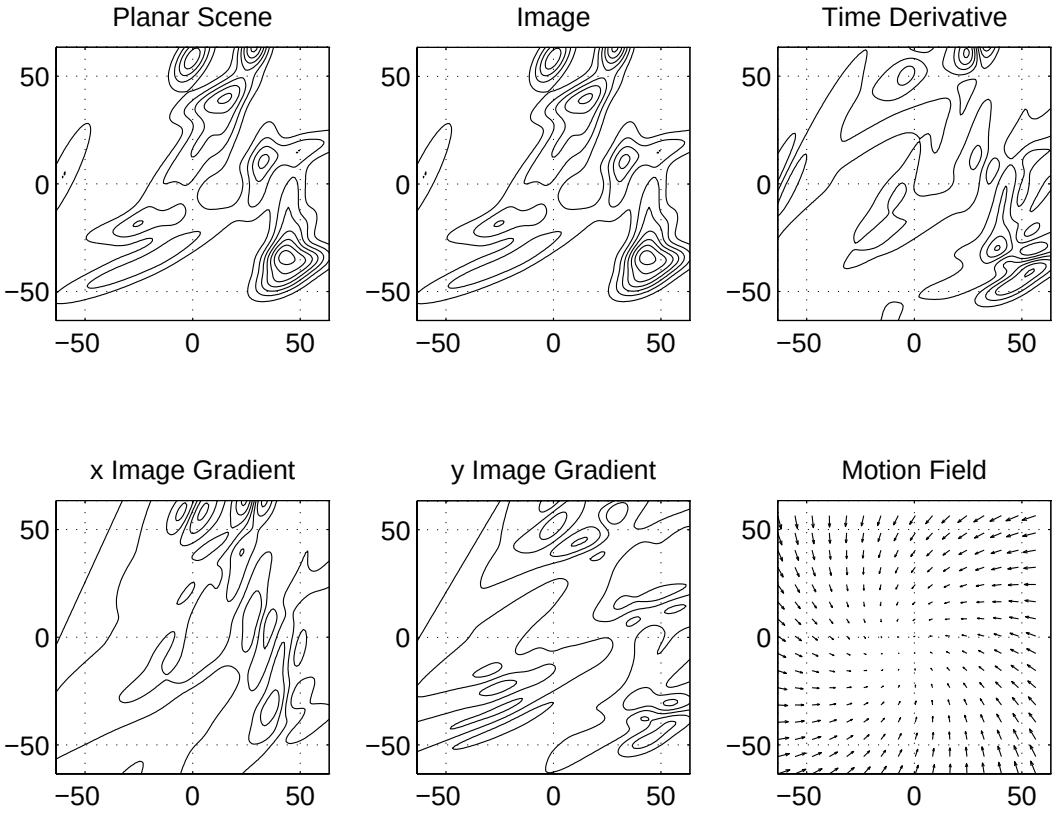

Figure 4-2: Contour plot of planar scene, spatial and time derivatives, and motion field for first test image. Scene plane is parallel to image plane. $\boldsymbol{\omega} = [0\ 0\ 0.1]^T$, $\mathbf{t} = [0\ 0\ -2]^T$, and $\mathbf{n} = [0\ 0\ -0.1]^T$.

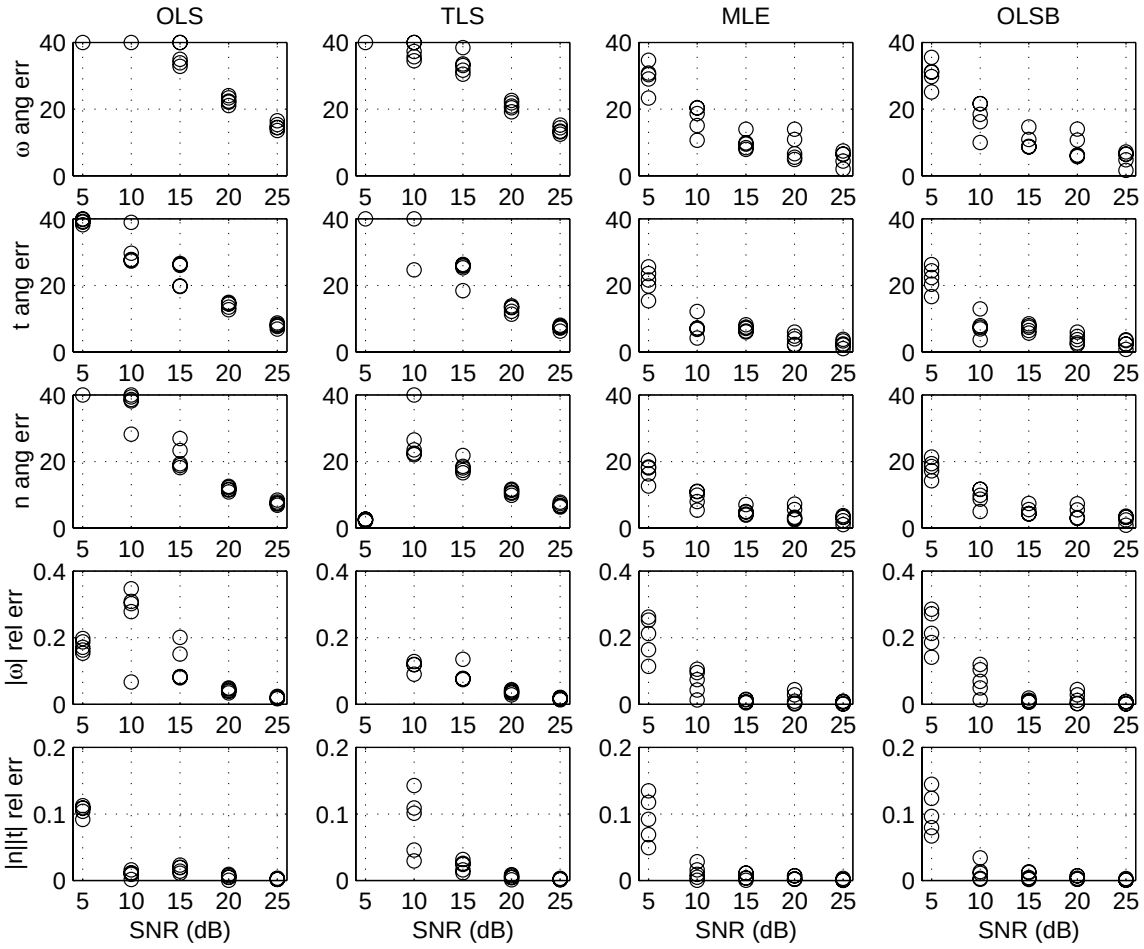

Figure 4-3: Comparison of angle and magnitude errors for motion field of Figure 4-2.

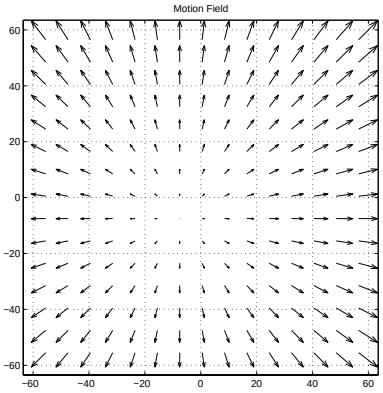

Figure 4-4: Motion field for $\boldsymbol{\omega} = [0\ 0\ 0]^T$, $\mathbf{t} = [0\ 0\ 2]^T$, and $\mathbf{n} = [0\ 0\ -0.1]^T$.

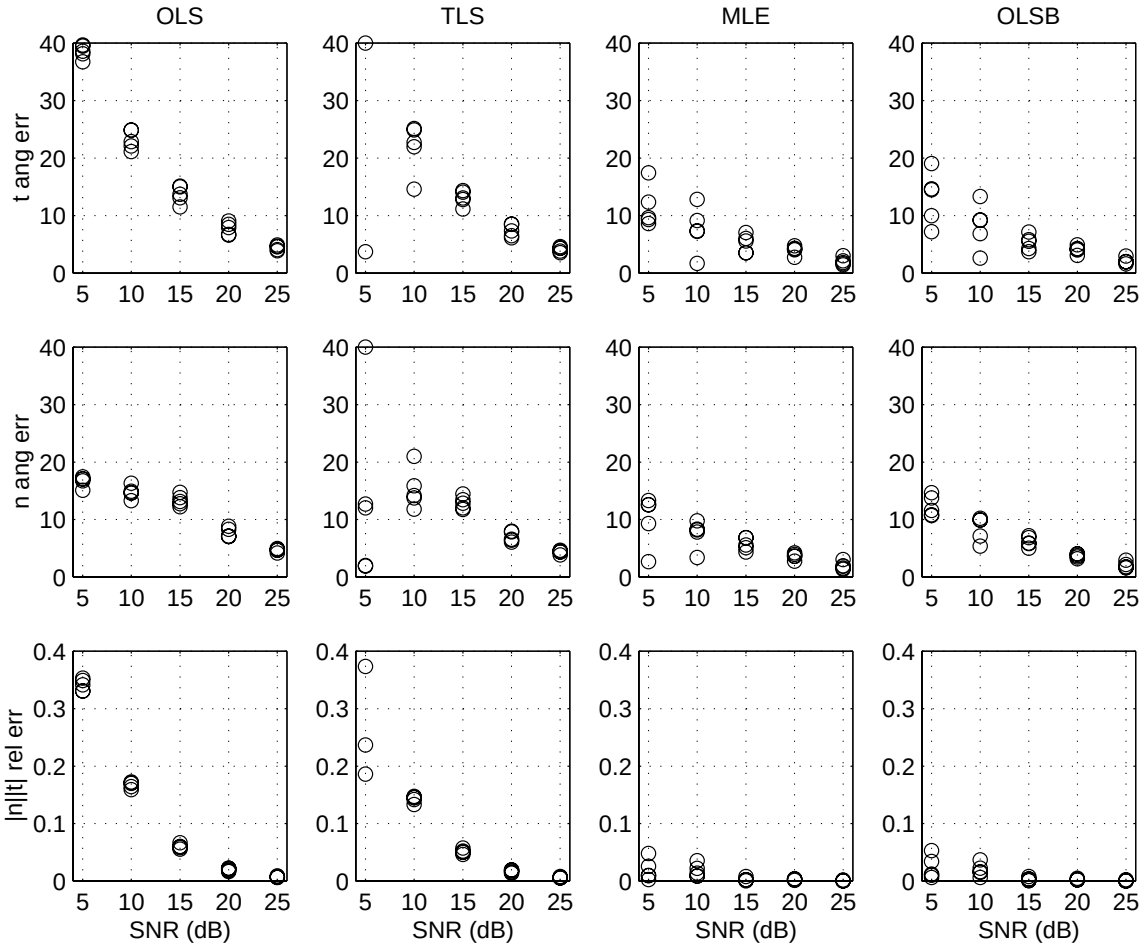

Figure 4-5: Comparison of angle and magnitude errors for motion field of Figure 4-4.

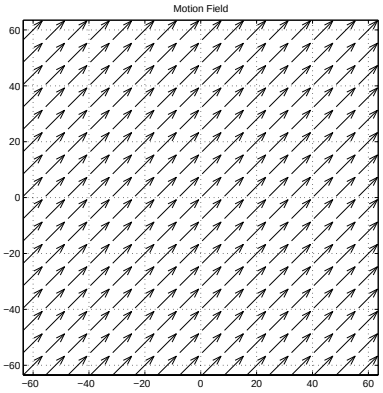

Figure 4-6: Motion field for $\boldsymbol{\omega} = [0\ 0\ 0]^T_,$ $\mathbf{t} = [1\ 1\ 0]^T_,$ and $\mathbf{n} = [0\ 0\ -0.1]^T$.

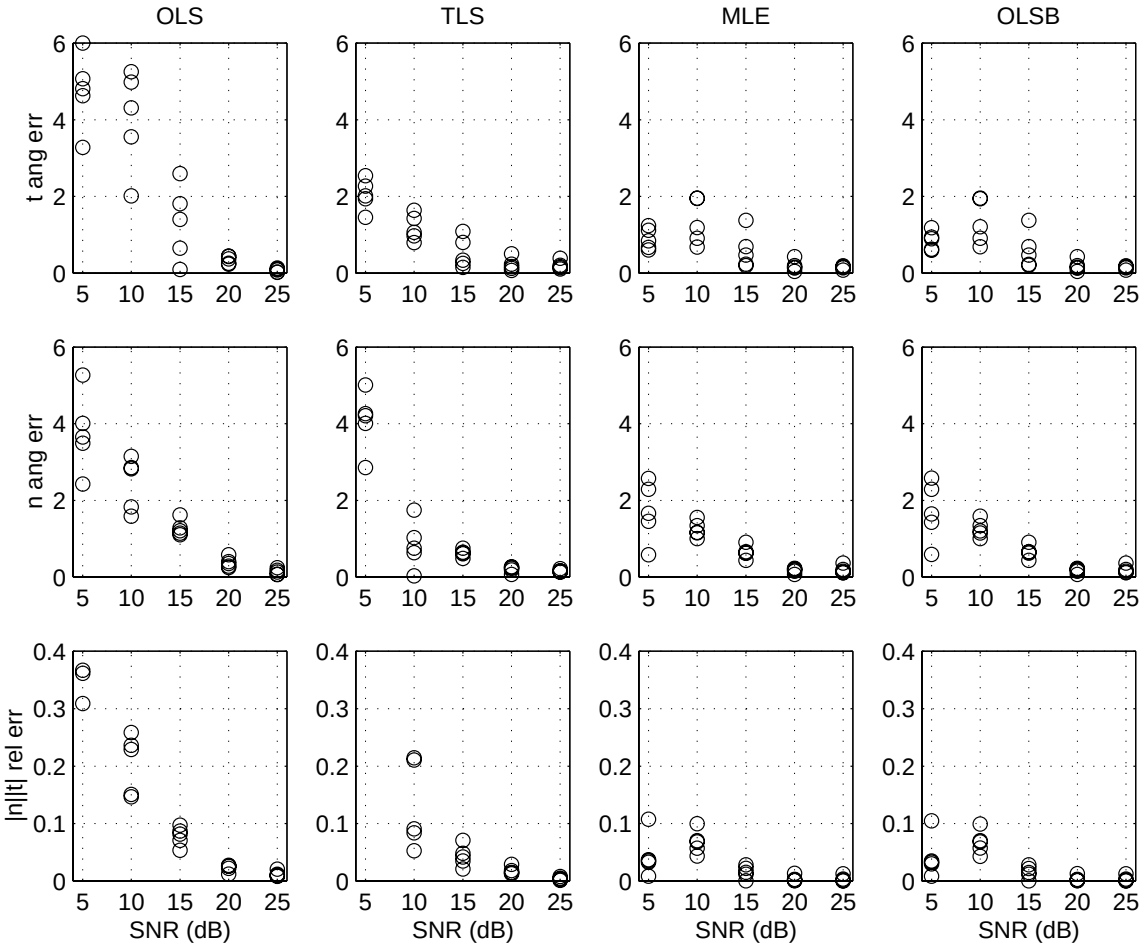

Figure 4-7: Comparison of angle and magnitude errors for motion field of Figure 4-6.

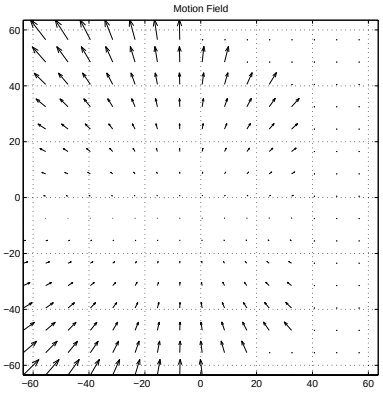

Figure 4-8: Motion field for $\boldsymbol{\omega} = [1\ 0\ 0.1]^T_,$ $\mathbf{t} = [0\ 0\ 1]^T_,$ and $\mathbf{n} = [0.0079\ 0\ \ -0.1]^T$.

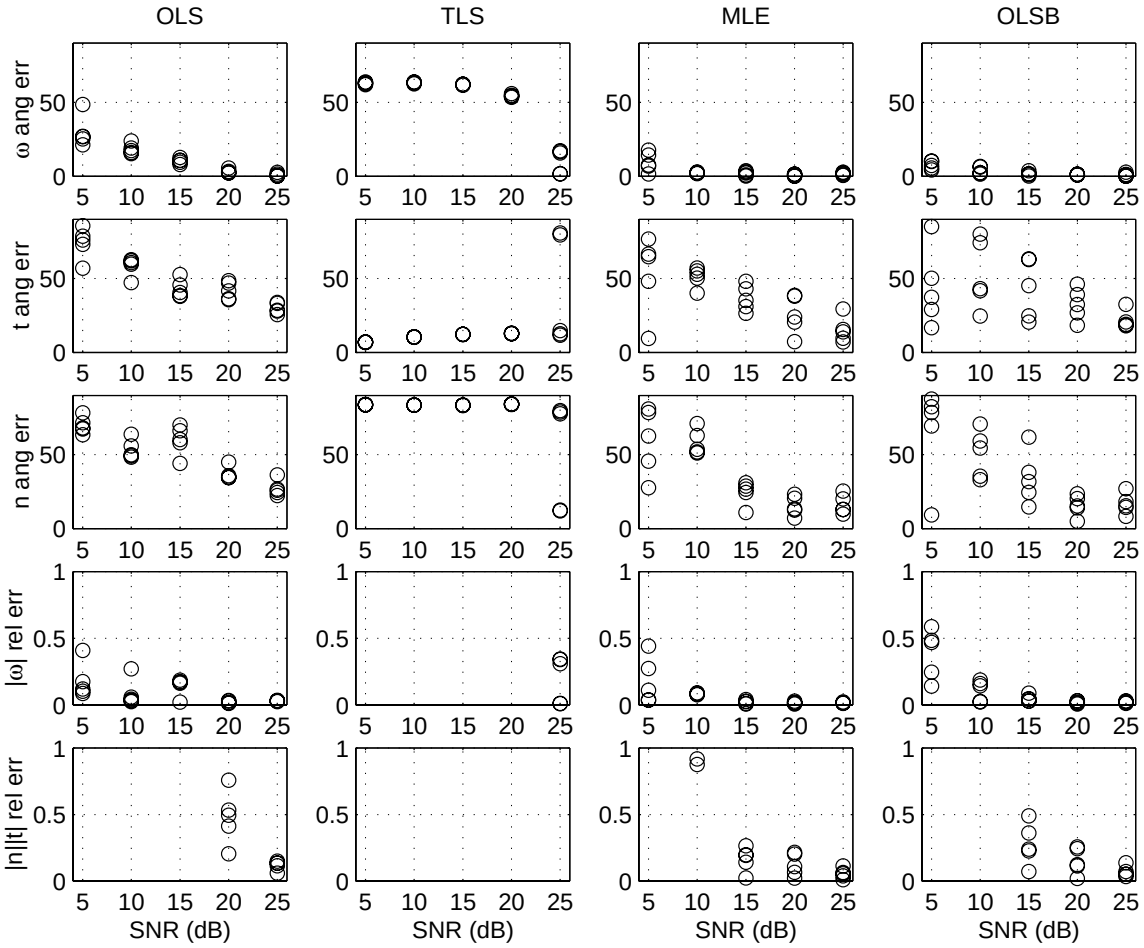

Figure 4-9: Comparison of angle and magnitude errors for motion field of Figure 4-8.

4.3.4 FPGA Implementation

In Section 4.3.3, we saw how the bias removal approach rivaled the CML approach in terms of accuracy of motion recovery parameters. Since this method is also reasonably computationally efficient, we examine an implementation of it in FPGA technology.

When we form the normal equations, we must calculate the summation

$$\mathbf{A}^T\mathbf{A} = \sum \mathbf{a}\mathbf{a}^T, \tag{4.78}$$

where the summation is over all the image pixels. Recall that for each pixel,

$$\mathbf{a} = \begin{bmatrix} -xE_x \\ -xE_y \\ x^2E_x + xyE_y \\ -yE_x \\ -yE_y \\ xyE_x + y^2E_y \\ -E_x \\ -E_y \end{bmatrix}, \tag{4.79}$$

where x and y are the image plane coordinates of the pixel. In order to visit and process each pixel only once, we form the outer product $\mathbf{a}\mathbf{a}^T$ for each pixel, and maintain a running accumulation. Since $\mathbf{a}\mathbf{a}^T$ is symmetric, there are $1+2+\cdots+8 = 36$ unique accumulations that must be formed.

In a DSP microprocessor implementation (where multiplies cost the same as additions), we would first form $\mathbf{a}$, requiring at best 8 clock cycles. Thus, at best $8+36 = 44$ clocks per pixel are needed by a DSP microprocessor to form $\mathbf{A}^T\mathbf{A}$. This is a rather high computational load, which limits the achievable frame rate for large images. Table 4.1 shows the maximum achievable frame rate as a function of image size for a DSP microprocessor with a 50MHz clock rate. Also shown in the table is the frame rate for an FPGA implementation which we describe below. Clearly, for larger image sizes, the FPGA approach can support real-time operation, whereas the DSP approach cannot.

Table 4.1: Frame rate in frames/sec. as a function of image size for DSP microprocessor and FPGA implementation.

Image Size	Number of Ops. Per Frame	Frame Rate (DSP)	Frame Rate (FPGA)
128×128	721×10^3	69.4	3051.8
256×256	2.9×10^6	17.3	762.9
512×512	11.5×10^6	4.3	190.7
1024×1024	46.1×10^6	1.1	47.7

An FPGA implementation has the potential to greatly speed up the calculations. Indeed, if we could fit 44 multiply/accumulations onto an FPGA, then we could process each pixel in a single clock cycle, for a factor of 44 speedup. While FPGA densities are currently increasing, this brute force approach is still beyond the capabilities of a single device. We could potentially perform some sharing of intermediate products, but this will not be sufficient.

Summation-By-Parts

What is needed then is a clever method to reduce the computational cost. Fortunately, there is some structure, not within a pixel's computations, but *across* pixel computations, that we can use to our advantage.

Assume that we are processing the pixels in raster order. Along a scan line, pixels have a constant y coordinate, and the x coordinates increment from one pixel to the next. We therefore have to accumulate products of powers of the x coordinate and functions of the image gradients. There is actually a technique, known as "summation-by-parts" (SBP) [23], [60], which may used in this situation. SBP is a discrete-time version of the continuous-time calculus operation "integration by parts."

Assume we must calculate the summation

$$S = \sum_{k=1}^{N} f_k x_k, \tag{4.80}$$

where f_k are known, fixed coefficients, and x_k is data (not x coordinates). Generally, SBP is useful when f_k is a polynomial in k (or can be well approximated by a low order polynomial in k). Recall from integration by parts, that if we are integrating a product of functions, then we can get the same result by differentiating one of the functions, and integrating the other function. There is another term as well, which will correspond to some end effects in the SBP formula.

Guided by the integration by parts methods, in order to reduce the degree of the polynomial f_k, we integrate x_k. In other words, we desire to evaluate the sum according to the following decomposition:

$$\begin{aligned} \textstyle\sum_{k=1}^{N} f_k x_k \quad &= g_N \cdot (x_N) \\ &\quad + g_{N-1} \cdot (x_N + x_{N-1}) \\ &\quad + g_{N-2} \cdot (x_N + x_{N-1} + x_{N-2}) \\ &\qquad \vdots \\ &\quad + g_1 \cdot (x_N + x_{N-1} + x_{N-2} + \ldots + x_1). \end{aligned} \tag{4.81}$$

The terms in the parenthesis are simply a running accumulation of the data x_k, from high indices to low indices. We find the values of the g's in terms of the f's by equating coefficients of the unknowns x_k and solving the set of linear equations:

$$\begin{bmatrix} f_N \\ \vdots \\ f_1 \end{bmatrix} = \begin{bmatrix} 1 & \cdots & 1 \\ & \ddots & \vdots \\ & & 1 \end{bmatrix} \begin{bmatrix} g_N \\ \vdots \\ g_1 \end{bmatrix}. \tag{4.82}$$

Inverting the matrix gives,

$$\begin{bmatrix} g_N \\ \vdots \\ g_1 \end{bmatrix} = \begin{bmatrix} 1 & -1 & & \\ & \ddots & \ddots & \vdots \\ & & 1 & -1 \\ & & & 1 \end{bmatrix} \begin{bmatrix} f_N \\ \vdots \\ f_1 \end{bmatrix}. \tag{4.83}$$

Or, in other words,

$$g_k = \begin{cases} f_k - f_{k-1} & k \neq 1, \\ f_k & k = 1. \end{cases} \tag{4.84}$$

With exception of the endpoint, the sequence g_k is thus the difference of the f_k's. For f_k a polynomial of degree d, g_k will be a polynomial of degree $d-1$. By iterating this procedure, we can reduce the degree of the polynomial to 0. In this case, the coefficients are constant, and thus may be calculated by a single multiplication, rather than N different multiplications.

Consider, for example, $f_k = k$. From (4.84), $g_k = 1$, for all k. Figure 4-10 shows the structure for $f_k = k$. The data is input in the reverse order $x_N, x_{N-1}, \ldots, x_1$. The first accumulator produces the running accumulation $x_N + x_{N-1} + \ldots + x_{N-l+1}$ at time l. This sequence goes into the second accumulator, which produces the required value $\sum_k kx_k$ at time N. The commutator samples the second accumulator output at time N to produce the desired output. Notice from the figure that no multiplications are required, only two additions per input sample.

POLYACC Structure

To calculate $\mathbf{a}\mathbf{a}^T$ in (4.78), we shall need to evaluate sums of products of several different polynomials with the image gradient data. By iterating the SBP procedure, we generate polynomials of all degrees from d down to 0. Figure 4-11 shows a structure that takes advantage of this observation. In the figure, all commutators are sampled at time N. The network of scalings and adders after the commutators are required to cancel the end effects that arise due to the compound formula in (4.84). Since the constants are simple combinations of powers of two, we count them as additions, rather than multiplications. We note that similar structures are known in the literature, although derived differently from here; see Wong and Siu [206] and the references therein.

For an $M \times M$ image, we have as coordinate values $\pm f_k = k - \frac{1}{2}$, for $k = 1, \ldots, M/2$, in one dimension. If we treat the positive and negative coordinates separately, then the structure in Figure 4-12 evaluates the required sums of products along a scanline. At the pixel data rate, only 5 additions per pixel are required by this structure. This should be contrasted with the four multiplications per pixel that would be required if the multiplications were actually performed (and the various powers of $k - 1/2$ were precalculated). One should also observe that there is a wordlength growth associated with the accumulators. Custom wordlength sizes are not a problem for FPGA implementations, but are highly problematic for DSP microprocessors. Thus, the structure in Figure 4-12, which we term a "polynomial accumulator" (POLYACC), is favored for FPGA implementation, but not for DSP microprocessor implementation.

For sums of products of two-dimensional functions, we can make use of the POLYACC structure if the coefficient polynomials of x and y are separable. That is, the required summation is

$$\begin{aligned} S &= \sum_{k=1}^{N} \sum_{l=1}^{N} f_k g_l x_{k,l} \\ &= \sum_{k=1}^{N} f_k \left(\sum_{l=1}^{N} g_l x_{k,l} \right). \end{aligned} \tag{4.85}$$

The second line of (4.85) shows the preferred ordering of the calculations. The summation in the parenthesis are evaluated for every x-direction scanline using the POLYACC structure. The result is a single number for each scanline. These numbers are then fed into another POLYACC structure (operating at a much lower data rate), corresponding to the y-direction. In both the x and y cases, positive and negative indices are handled separately, and the indices start at the out edge of the image and are decremented. Figure 4-13 shows the pixel input scanning pattern required to use the nested POLYACC approach.

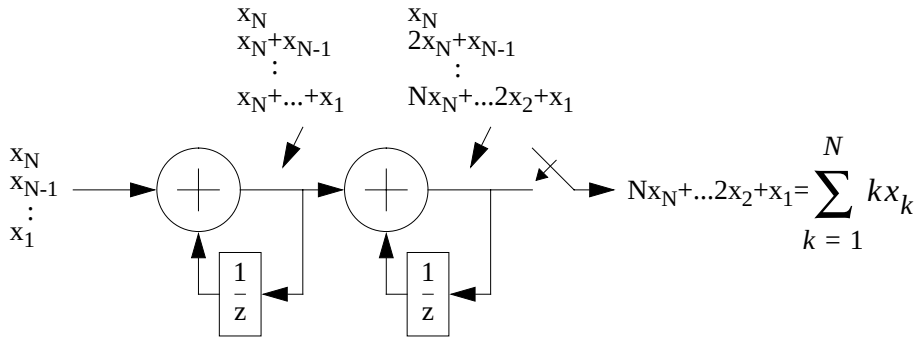

Figure 4-10: Multiplierless architecture to evaluate $\sum_{k=1}^{N} kx_k$.

With the basic POLYACC structure now understood, we return to the computation of $\mathbf{B} = \mathbf{a}\mathbf{a}^T$. The matrix $\mathbf{B}$ is symmetric, so we need only to calculate the elements B_{ij} for $j \geq i$. Using (4.79), these coefficients, as functions of the image coordinates x and y, and the image gradients E_x and E_y, are given in (4.86) - (4.97) below:

$$B_{1,1} = x^2E_x^2,\ B_{1,2} = x^2E_xE_y,\ B_{1,3} = -x^3E_x^2 - x^2yE_xE_y,\ B_{1,4} = xyE_x^2, \tag{4.86}$$

$$B_{1,5} = xyE_xE_y,\ B_{1,6} = -x^2yE_x^2 - xy^2E_xE_y,\ B_{1,7} = xE_x^2, B_{1,8} = xE_xE_y, \tag{4.87}$$

$$B_{2,2} = x^2E_y^2,\ B_{2,3} = -x^3E_xE_y - x^2yE_y^2,\ B_{2,4} = xyE_xE_y,\ B_{2,5} = xyE_y^2, \tag{4.88}$$

$$B_{2,6} = -x^2yE_xE_y - xy^2E_y^2,\ B_{2,7} = xE_xE_y,\ B_{2,8} = xE_y^2, \tag{4.89}$$

$$B_{3,3} = x^4E_x^2 + 2x^3yE_xE_y + x^2y^2E_y^2,\ B_{3,4} = -x^2yE_x^2 - xy^2E_xE_y, \tag{4.90}$$

$$B_{3,5} = -x^2yE_xE_y - xy^2E_y^2,\ B_{3,6} = x^3yE_x^2 + 2x^2y^2E_xE_y + xy^3E_y^2, \tag{4.91}$$

$$B_{3,7} = -x^2E_x^2 - xyE_xE_y,\ B_{3,8} = -x^2E_xE_y - xyE_y^2, \tag{4.92}$$

$$B_{4,4} = y^2E_x^2,\ B_{4,5} = y^2E_xE_y,\ B_{4,6} = -xy^2E_x^2 - y^3E_xE_y, \tag{4.93}$$

$$B_{4,7} = yE_x^2,\ B_{4,8} = yE_xE_y, \tag{4.94}$$

$$B_{5,5} = y^2E_y^2,\ B_{5,6} = -xy^2E_xE_y - y^3E_y^2,\ B_{5,7} = yE_xE_y,\ B_{5,8} = yE_y^2, \tag{4.95}$$

$$B_{6,6} = x^2y^2E_x^2 + 2xy^3E_xE_y + y^4E_y^2,\ B_{6,7} = -xyE_x^2 - y^2E_xE_y, \tag{4.96}$$

$$B_{6,8} = -xyE_xE_y - y^2E_y^2,\ B_{7,7} = E_x^2,\ B_{7,8} = E_xE_y,\ B_{8,8} = E_y^2. \tag{4.97}$$

These equations show that we need to process the functions E_x^2, E_y^2, and E_xE_y through nested POLYACC structures. Figures 4-14 and 4-15 show an architecture for performing this calculation. In Figure 4-14, the image gradients E_x and E_y are multiplied and squared to form E_x^2, E_y^2, and E_xE_y. These functions are then each passed through separate x-direction POLYACCs to form the various sums of products of powers of x and the image gradient functions. The outputs of this first bank of three POLYACCs, can now be fed into a second bank of y-direction POLYACCs, which operate at a much slower data rate. The outputs of the second bank of POLYACCs are then combined as necessary in Figure 4-15 to form the elements of $\mathbf{B}$. Since processing is done on image quadrants, we must perform four passes, and simply add the correspond $\mathbf{B}$ from each pass.

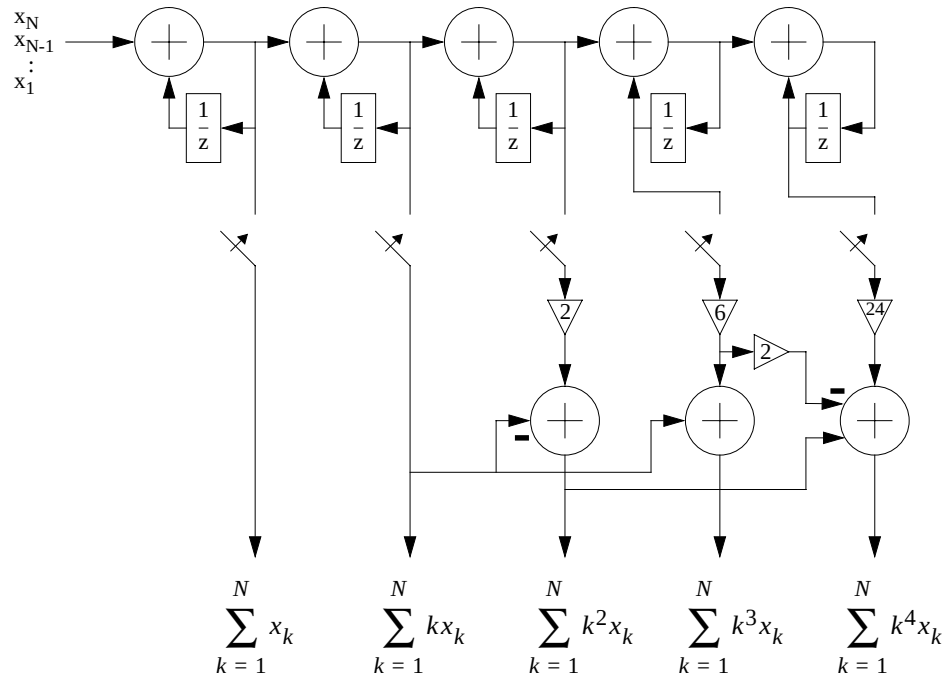

Figure 4-11: Multiplierless architecture to evaluate multiple sum of products of data with weights 1, k, k^2, k^3, and k^4.

For simplicity, we could also only implement the first bank of POLYACCs from Figure 4-14 in an FPGA. There would only be a total of $2M$ pieces of data for M^2 pixels to post-process in software, which is negligible in terms of the total computation time. In any case, we predict approximately a 44× speedup over a pure DSP software implementation. Table 4.1 shows the approximate frame rate as a function of image size.

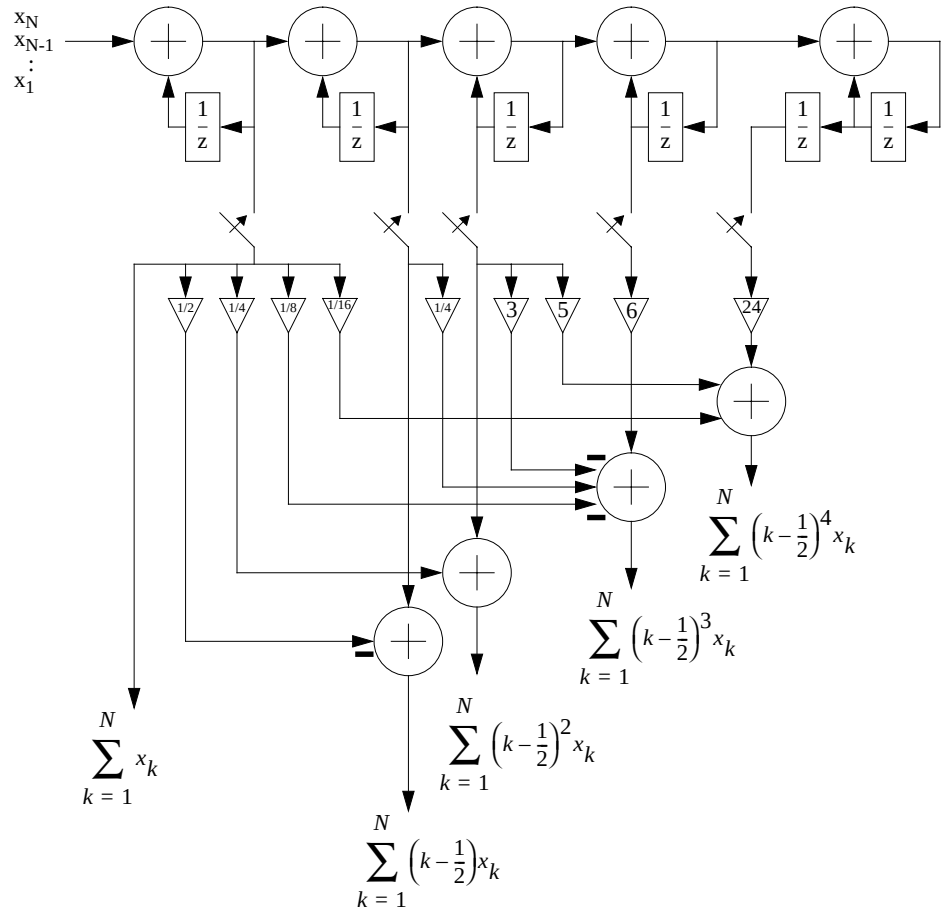

Figure 4-12: POLYACC structure. Multiplierless architecture to evaluate multiple sum of products of data with weights 1, $k-\frac{1}{2}$, $(k-\frac{1}{2})^2$, $(k-\frac{1}{2})^3$ and $(k-\frac{1}{2})^4$.

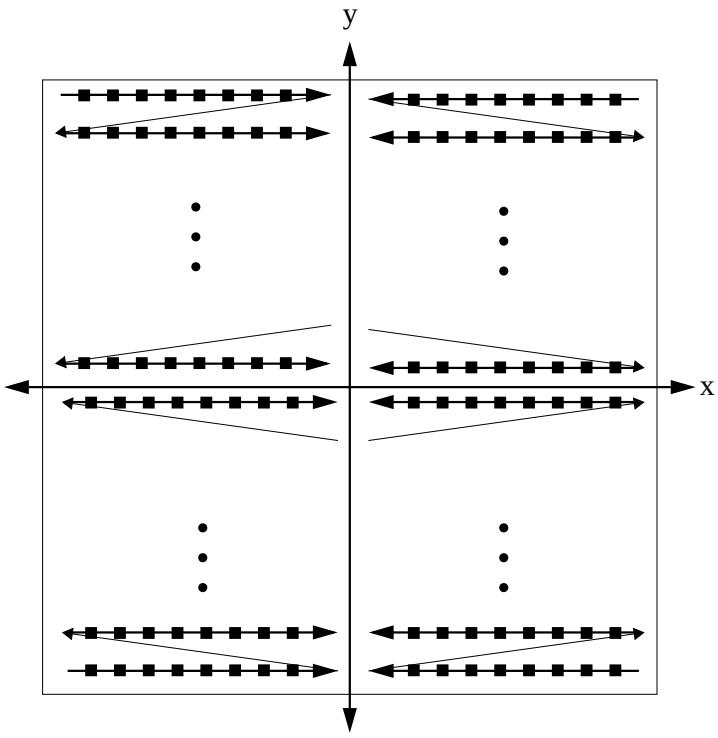

Figure 4-13: Scanning pattern for nested POLYACC approach.

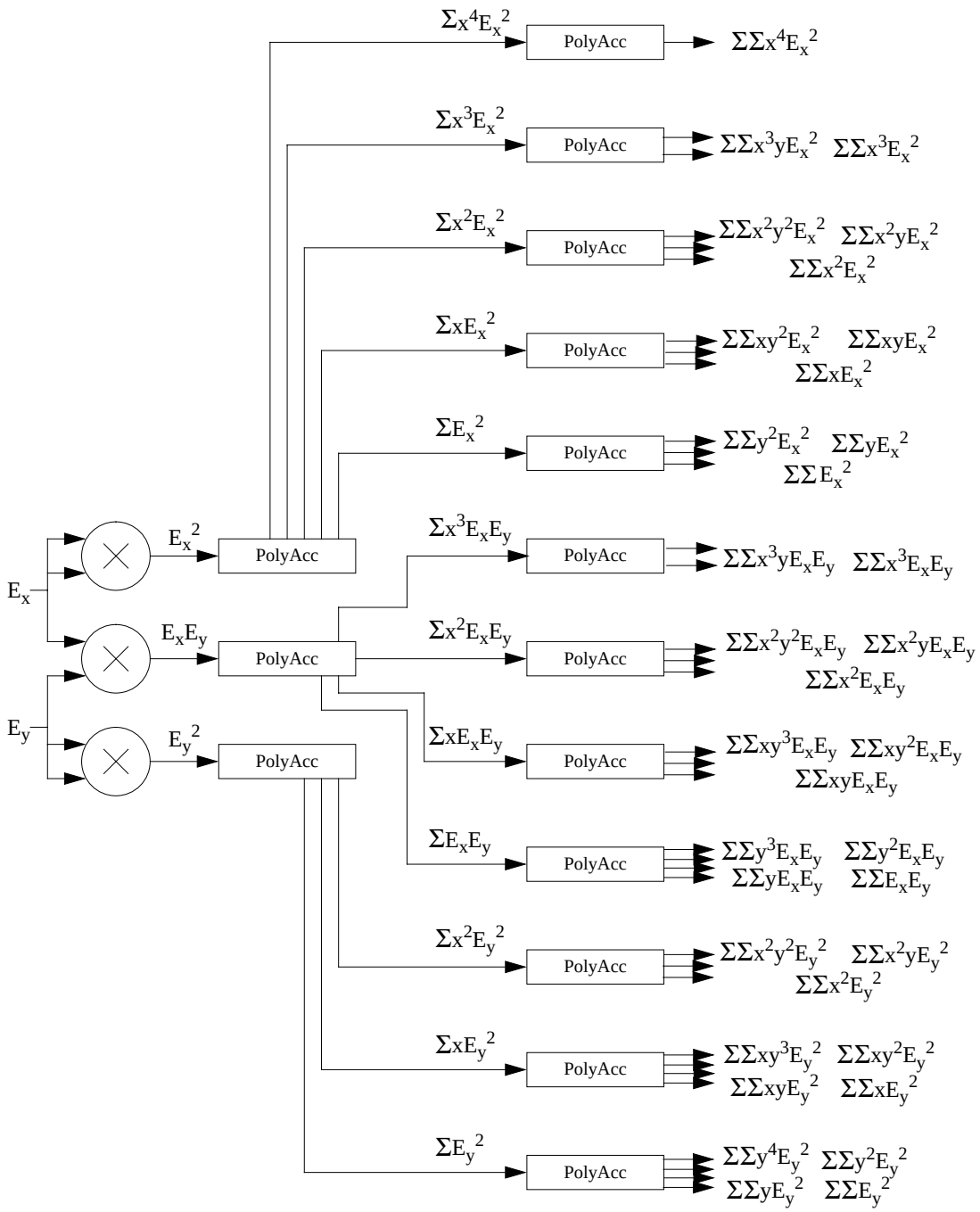

Figure 4-14: Using polynomial accumulator structure (Figure 4-12) to calculate Normal equations matrix **B** (Part I).

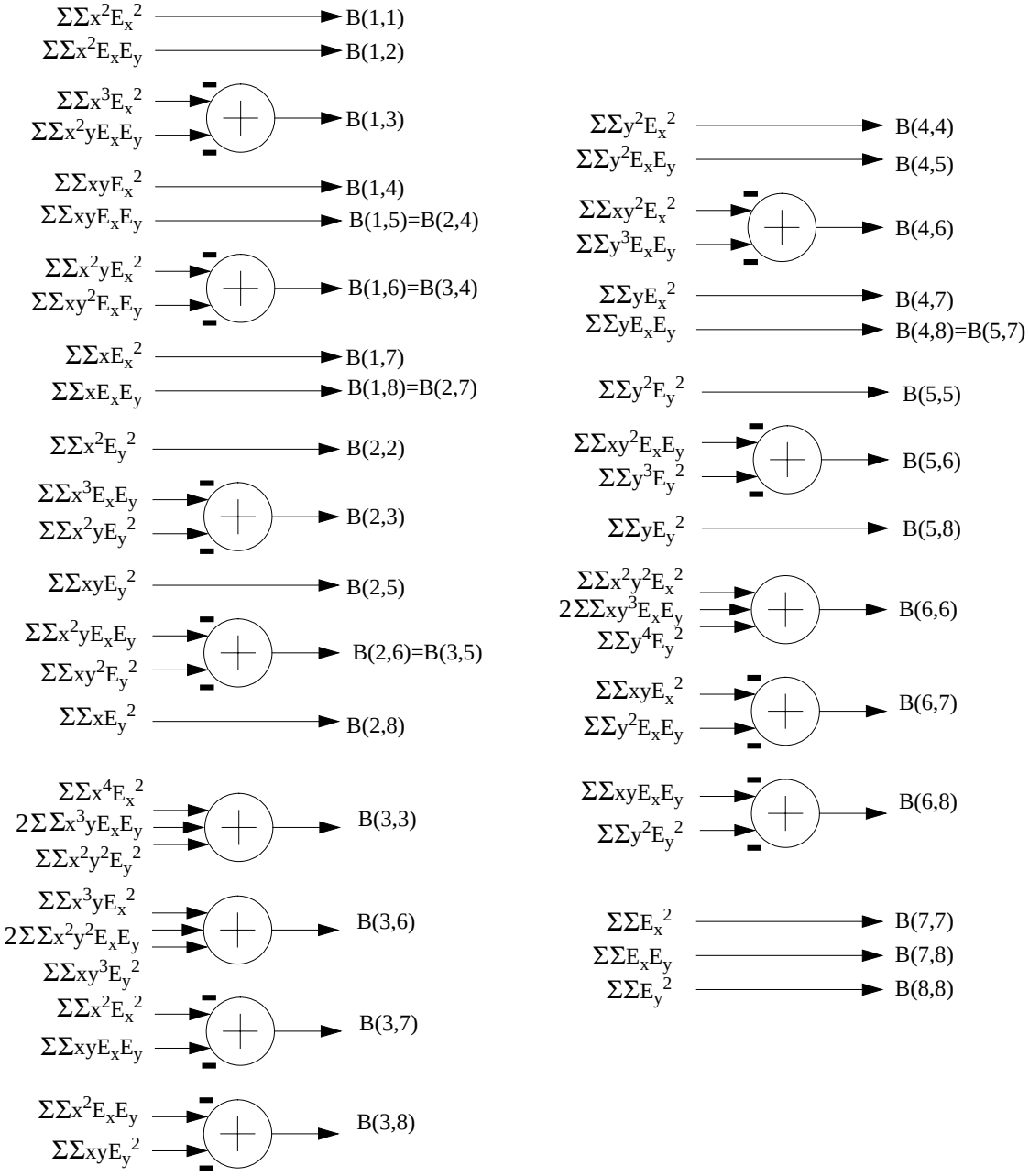

Figure 4-15: Using polynomial accumulator structure (Figure 4-12) to calculate Normal equations matrix **B** (Part II).

Wordlength Analysis

We now analyze the wordlength requirements of the accumulator chain structure (Figure 4-10). Of course, the methods of Chapter 3 could be applied to the entire circuit of Figure 4-12 or Figure 4-14. However, limiting our analysis to the chain of accumulators will allow us to derive approximate analytical expressions for the wordlengths. These will in turn provide insight into the nature of the optimal wordlengths, and provide starting states for the MCMC search.

Assume we have a chain of L accumulators as in Figure 4-10. Let node i denote the arc in the flowgraph after the stage i accumulator, and node 0 denotes the input arc. We will consider truncating at nodes $i = 0, \ldots, L$. Note that no truncation is allowed within the feedback portion of a stage, since the accumulator will generally become unstable. Let the number bits after truncation at node i be given by b_i.

From Section 3.3.1, we must first calculate the numerical range at each node. Assume that the node 0 range is $[-1, 1]$ (i.e. $M_0 = 1$). The range at node 1 is of course $[-N, N]$, and we can easily derive using combinatorial formulas ([210], page 37) that the range at node i is represented by

$$M_i = \begin{pmatrix} N-1+i \\ i \end{pmatrix} \tag{4.98}$$

for $i = 1, \ldots, L$. We make the approximation that $\tilde{M}_i = M_i$ (i.e. in (3.5), we assume the range is a power of 2). The slope s_i of the transfer function from node i to the output is easily found to be $s_i = M_L/M_i, i = 0, \ldots, L$. From the variance formula (3.7) we have that the variance at the output of the k^{th} stage is (after simplification)

$$V_k(\mathbf{b}) = \frac{1}{12} \begin{pmatrix} N-1+k \\ k \end{pmatrix}^2 \sum_{i=0}^{k} 2^{-2b_i}, \quad k = 0, \ldots, L. \tag{4.99}$$

Since $V_k(\mathbf{b})$ is monotonically increasing as k increases, the worst case noise variance occurs at $k = L$, so we concentrate just on this output.

We now calculate the cost $C(\mathbf{b})$ to implement the L stages. The cost of stage i is proportional to $log_2 N + b_{i-1}$, because this is the width of the accumulator in full adders (and is also the width of the storage elements). The total cost is thus proportional to

$$C(\mathbf{b}) = L \cdot log_2 N + \sum_{i=0}^{L-1} b_i. \tag{4.100}$$

With $V(\mathbf{b})$ and $C(\mathbf{b})$ now approximately given, we could perform the MCMC algorithm of Section 3.3.5, but in this case this is unnecessary. Because of the symmetry in the cost and variance formulas, we see that at the optimum all the wordlengths $b_0, b_1, \ldots, b_{L-1}$ are equal to each other. The final wordlength b_L has no cost impact, so we just set it as large as possible. All the Pareto-optimal points are therefore given by $b_0 = b_1 = \cdots = b_{L-1} \stackrel{\triangle}{=} b$, and we can plot the cost versus variance, parameterized by b, as shown in Figure 4-16.

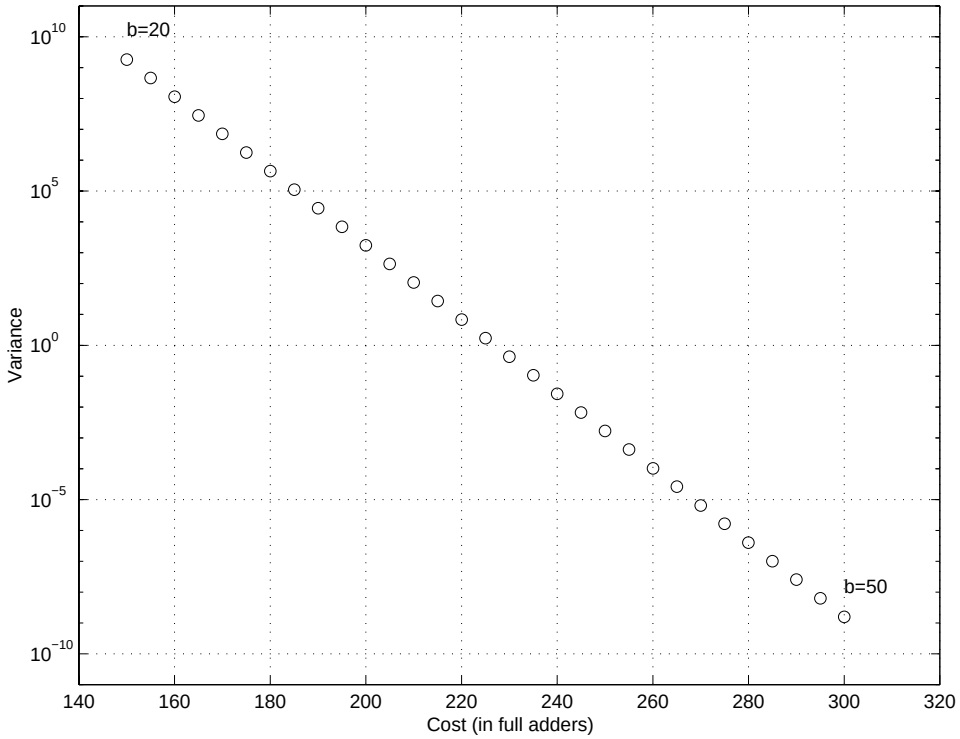

Figure 4-16: Cost vs. variance for simple model of accumulator chain. Cost is measured in number of full adders and storage elements. The image size is 1024×1024.

4.4 Summary

In this chapter, we were concerned with accurately solving complex motion equations. We saw that enforcing all (nonlinear) constraints led to an accurate, albeit complicated, solution approach. However, by selectively choosing what types of constraints to enforce, an efficient FPGA implementation to enable real-time operation was possible. By knowing the optimal approach that enforces all constraints, we are in a position to make a system engineering judgement as to whether or not the approximate solution is "good enough". Thus, in the chapter we have given a concrete demonstration of the main theme of this thesis; by considering both the algorithm and implementation we can achieve nearly optimal results in an extremely efficient FPGA device.

We reviewed the maximum likelihood concept in Section 4.1, and provided motivation for why poor estimates result when one ignores parametric constraints. The methods presented in that section relied on removing parameters from the estimation problem. Because this is often inconvenient or impossible, Section 4.2 developed the constrained maximum likelihood approach. The key result was the optimization criterion given in (4.21). Also of great interest is the Gaussian specialization given in (4.31), which is often sufficient for motion estimation.

We then applied this idea to the direct-method estimation of motion for a planar scene in Section 4.3; the CML cost function for this case is given by (4.54)–(4.57). While the solution obtained this way can be considered optimal, one must resort to iterative techniques for the solution. To speed up this iteration, could apply the techiques from Chapter 3 to design a custom computing datapath to calculate (4.54)–(4.57) for each iteration. However, in Section 4.3.2, we explored an alternative approach. There, we did not strictly enforced all constraints, but rather, we enforced them in a statistical sense. Enforcing the constraints in this manner was extremely simple (i.e. (4.77)). We also saw through extensive simulation in Section 4.3.3 that this bias removal approach performed nearly as well as the optimal CML approach. Moreover, the simplicity of the approach lends itself well to an efficient FPGA implementation; in Section 4.3.4 we used the summation-by-parts technique in addition to the techniques from Chapter 3 to perform the design optimization.

In this chapter there were several novel ideas put forth:

- the CML optimization criterion (Section 4.2),
- the application of the bias removal approach to the planar scene motion problem (Section 4.3.2),
- the FPGA implementation of the direct method for a planar scene (Section 4.3.4).

The development of these items required use of signal processing concepts to understand the effects of noise. The second item also required a substantial use of the efficient FPGA design techniques from Chapter 3. In the next chapter, we extend the idea of more accurately modeling the perturbations of the data to the problem of more than two data sets.

Chapter 5

Multiple Type and Multiple View Orientation

5.1 Introduction and Chapter Overview

In this chapter, we explore the topic of orientation and photogrammetry using multiple (more than two) data sets. This topic is currently receiving a great deal of interest due to the possibility of obtaining more accurate scene reconstructions than are possible with only two data sets. This chapter explores the unification of multiple view orientation data using error ellipses into a common processing framework, so that a suitable custom computing architecture can be formulated.

Researchers have sought to use multiple views to further improve the accuracy of the estimated camera motion, and for scene reconstruction (as we saw in Section 2.4). Accurate scene reconstruction can also be used to aid on obstacle avoidance for moving platforms, and for recognition of known objects in the environment.

Historically, scene reconstruction was first performed by photogrammetrists, with the emphasis placed on accuracy. A typical application was to build maps from aerial photographs. As shown in Appendix C, a Newton-Raphson iteration is the main computational algorithm for solving this problem. Since data collection was (and still is) rather expensive, the emphasis on highly accurate methods is understandable.

More recently, the computer vision community has taken great interest in this problem, but from a different viewpoint. The research emphasis is now placed on fast and robust methods, in order to construct systems that can act in real-time. In this chapter, we will explore both of these aspects.

The remainder of this chapter is organized as follows. Section 5.2 briefly summarizes some of the previous literature addressing the multiple view orientation problem. Section 5.3 reviews the optimal combination of noisy estimates using a statistical approach based on error ellipses. These results are then used in Section 5.4 to cast all interesting orientation problems into a unified framework. Conventional orientation problems such as absolute, exterior, and relative orientation fit naturally into this framework. Error ellipses are used in a slightly different manner in Section 5.5, where an efficient, approximate algorithm for multiple frame orientation using pairwise orientation estimates is considered. While not exact, the linear approach covered in this section appears to work well in practice. In Section 5.5.8, we use the FPGA-based SVD implementation derived in Section 3.6 to accelerate the linear epipolar algorithm. Finally, Section 5.6 summarizes the main results and contributions of the chapter.

5.2 Previous Work on Multiple View Orientation

The problem of obtaining orientation from multiple views has received attention in the literature. Okutomi and Kanade [141] use a linear array of cameras with known position and orientation to not only form a dense depth map, but also to find feature correspondences across the images. They minimize a combined cost function, which is essentially a measure of the similarity of a base image with the other images within small windows about the purported associated feature points. To avoid a costly multidimensional search, the authors parameterize the window locations by the inverse depth of the point. By scanning through a range of inverse depths, the various disparities in the multiple images are easily calculated, and the cost function are then easily calculated. Thus, for each point in the base image, a one-dimensional search is performed. An update to this approach using a moving stereo rig was recently reported by Yi and Oh [208].

Many researchers have noted that poor estimation performance often results when feature point observations are noisy, and a line of research attempting to characterize the uncertainty in motion estimates has been followed.

Zhang [209] gives a detailed account of robust methods to estimate the orientation parameters and the resulting uncertainty for two views. Using the implicit function theorem from the book by Faugeras [51], Zhang derives expressions for the covariance matrix of a solution to a general minimization problem, a form into which the calculation of the fundamental matrix may be cast. Related work by Czurka *et al.* [37] uses error ellipses to characterize the uncertainty of the fundamental matrix for uncalibrated stereo pairs. Luong and Faugeras [120] also briefly consider the fundamental matrix error ellipses.

Thomas and Oliensis [177] take an incremental approach to the structure from motion problem. They use the conventional two-frame algorithm to estimate the structure and motion for the most recent image pair, along with the various error covariances of the estimates. This most recent estimate is then fused with previous structure estimate, using a standard Kalman filter approach. Their approach shares many features with the error ellipse approach presented in this chapter.

Because of the difficulty in solving motion equations resulting from the pinhole camera model, many researchers attempt to relax this model. Marugame, Katto, and Ohta [124] attempt structure recovery using several perspective cameras and one scaled orthographic camera. They take a projective geometric approach, and solve for the projection matrices using the SVD. Nonlinear constraints among the parameters are not explicitly enforced.

Recent work has attempted to cast photogrammetry and other orientation problems as an absolute orientation problem, subject to non-isotropic noise. Hill, Cootes, and Taylor [87] consider the absolute orientation problem with nonscalar weights. They provide an iterative algorithm, which is the standard update of an incremental rotation matrix, given a current rotation matrix. Hel-Or and Werman [84], [85] make a similar modeling assumption. Both of these approaches offer high accuracy, at the cost of a slow, iterative solution method. The method proposed in this chapter can be seen in some ways as an extension of these two approaches.

It has been recently recognized that multiple views lead to tensor constraints on the orientation parameters. Work on three views by Avidan and Shashua [11], Stein [167] Hartley [79] is well studied. It is known that for four or more views, quadlinear constraints are sufficient. Any view beyond the fourth only adds redundant observations, rather than adding more constraints [159]. The work in this line of research is mainly focused on characterizing solutions to tensor equations, using concepts from algebraic geometry [36].

In contrast to these algebraic geometry approaches, this chapter follows a more "signal processing" approach, where we are more concerned about accurately combining information by taking into account the relative information content in the different observations.

5.3 Optimal Combination of Estimates

In estimation problems, one must not only be concerned with obtaining the actual estimate, but also the stability of that estimate in the presence of noise. For example, in classical parameter estimation, the quantities to be estimated are considered to be unknown, non-random parameters. For any unbiased estimator, the Fisher Information Matrix (FIM) provides information about the sensitivity of the solution. Specifically, this matrix can be thought of as a "curvature" matrix about the true solution. If one were to perform an eigendecomposition of the information matrix, then the eigenvectors indicate the direction of the principal axes of the *error ellipse* about the true parameter values, and the eigenvalues indicate the length of the principal axes.

The lengths of the principal axes of the error ellipse are quite important. The information matrix is the inverse of the estimator covariance matrix. Thus, both matrices have the same set of eigenvectors, which form an orthonormal set due to symmetry of the matrices. In the case where the covariance matrix is positive definite, the information matrix is also positive definite, with its eigenvalues being simply the reciprocal of those of the covariance matrix. Thus, for directions in the parameter space with large covariance, the curvature of the information matrix will be small. This indicates that our parameter estimate will exhibit sensitivity to noise in the direction of low curvature. Another way to view this case is that we have high uncertainty in the parameter estimates in directions of low curvature. Conversely, for directions in the parameter space with small covariance, the curvature of the information matrix will be large. In these directions, noise will have very little effect on the parameter estimates.

For the limiting case where the covariance matrix has zero eigenvalues, the curvature in these directions is infinite, which implies that there is no uncertainty in the parameter estimates in these directions.

Using the FIM for curvature estimates has one drawback: the matrix often depends on the values of the true parameters, and not just the data. We thus must seek another form for the curvature matrix. A method that in principle will always allow the determination of a covariance and hence curvature matrix, is to linearize the estimator about the operating point of the estimator output vector. Once the linearization is performed, we then determine the covariance matrix of the estimator output vector from the knowledge of the data covariance matrix. Indeed, consider an estimator

$$\widehat{\mathbf{x}} = \mathbf{f}(\mathbf{y}), \tag{5.1}$$

where $\widehat{\mathbf{x}}$ is the vector estimate of the unknown parameters, $\mathbf{y}$ is the data vector, and $\mathbf{f}()$ is the (perhaps complicated) function representing the estimation procedure. By performing a Taylor series expansion about the nominal estimate $\widehat{\mathbf{x}}$, and discarding high order terms, we see that small perturbations $\mathbf{y}'$ in the data lead to small perturbations $\widehat{\mathbf{x}}'$ in the estimate. These perturbations are connected by the linear equation

$$\widehat{\mathbf{x}}' = \mathbf{H}\mathbf{y}', \tag{5.2}$$

where $\mathbf{H}$ is the Jacobian matrix of $\mathbf{f}()$, evaluated at the nominal estimate $\widehat{\mathbf{x}}$. Postmultiplying both sides of (5.2) by their transposes gives

$$\widehat{\mathbf{x}}'\widehat{\mathbf{x}}'^T = \mathbf{H}\mathbf{y}'\mathbf{y}'^T\mathbf{H}^T. \tag{5.3}$$

Taking expected values,

$$E\left\{\widehat{\mathbf{x}}'\widehat{\mathbf{x}}'^T\right\} = \mathbf{H}E\left\{\mathbf{y}'\mathbf{y}'^T\right\}\mathbf{H}^T, \tag{5.4}$$

or,

$$\mathbf{\Lambda}_{\widehat{\mathbf{x}}'} = \mathbf{H}\mathbf{\Lambda}_{\mathbf{y}'}\mathbf{H}^T, \tag{5.5}$$

where $\mathbf{\Lambda}_{\widehat{\mathbf{x}}'}$ is the covariance matrix of the estimate perturbation $\widehat{\mathbf{x}}'$ and $\mathbf{\Lambda}_{\mathbf{y}'}$ is the covariance matrix of the data perturbation $\mathbf{y}'$. Generally, we are given $\mathbf{\Lambda}_{\mathbf{y}'}$, and often it is assumed to be diagonal or even a scaled identity matrix (i.e. $\mathbf{\Lambda}_{\mathbf{y}'} = \sigma^2\mathbf{I}$). Thus, given the data perturbation covariance matrix, we can use (5.5) to calculate the estimator covariance matrix. The estimator curvature matrix is the inverse of $\mathbf{\Lambda}_{\widehat{\mathbf{x}}'}$. For any zero eigenvalues of $\mathbf{\Lambda}_{\widehat{\mathbf{x}}'}$, we increase them by some small amount ϵ so that the inversion can be carried out (or better, use the pseudoinverse). Note that because $\mathbf{\Lambda}_{\widehat{\mathbf{y}}'}$ is symmetric and positive semidefinite, $\mathbf{\Lambda}_{\widehat{\mathbf{x}}'}$ is itself symmetric, and positive semidefinite.

A simple example will illustrate this idea. Let an estimator for the unknown parameters $\widehat{\mathbf{x}}$ satisfy

$$\begin{aligned} \widehat{x}_1^2 &= 2y_1 + 4y_2 \\ \widehat{x}_2 &= 2y_1^2 - y_2. \end{aligned} \tag{5.6}$$

For this estimator, the value of $\widehat{x}_1$ is given implicitly as a function of the data, rather than explicitly. Taking the derivative with respect to a small perturbation in the data,

$$\begin{aligned} 2\widehat{x}_1\widehat{x}_1' &= 2y_1' + 4y_2' \\ \widehat{x}_2' &= 4y_1y_1' - y_2'. \end{aligned} \tag{5.7}$$

Rearranging to isolate the output perturbations gives

$$\begin{aligned} \widehat{x}_1' &= \tfrac{y_1'}{\widehat{x}_1} + \tfrac{2y_2'}{\widehat{x}_1} \\ \widehat{x}_2' &= 4y_1y_1' - y_2', \end{aligned} \tag{5.8}$$

which in matrix form is

$$\begin{bmatrix} \widehat{x}_1' \\ \widehat{x}_2' \end{bmatrix} = \begin{bmatrix} \frac{1}{\widehat{x}_1} & \frac{2}{\widehat{x}_1} \\ 4y_1 & -1 \end{bmatrix} \begin{bmatrix} y_1' \\ y_2' \end{bmatrix}. \tag{5.9}$$

Using (5.5) and assuming $\mathbf{\Lambda}_{\mathbf{y}'} = \sigma^2\mathbf{I}$ gives

$$\mathbf{\Lambda}_{\widehat{\mathbf{x}}'} = \sigma^2 \begin{bmatrix} \frac{1}{\widehat{x}_1} & \frac{2}{\widehat{x}_1} \\ 4y_1 & -1 \end{bmatrix} \begin{bmatrix} \frac{1}{\widehat{x}_1} & 4y_1 \\ \frac{2}{\widehat{x}_1} & -1 \end{bmatrix} = \sigma^2 \begin{bmatrix} \frac{5}{\widehat{x}_1^2} & \frac{4y_1-2}{\widehat{x}_1} \\ \frac{4y_1-2}{\widehat{x}_1} & 16y_1^2+1 \end{bmatrix}. \tag{5.10}$$

We see that the covariance matrix only depends on the data and the estimated output values, which are also functions of the data. The true parameter values were not needed.

If one is not comfortable with the derivative approach given above, an equivalent approach that explicitly shows the perturbations can be used. Our problem (5.6) becomes

$$\begin{aligned} (\widehat{x}_1 + f_1)^2 &= 2(y_1 + e_1) + 4(y_2 + e_2) \\ (\widehat{x}_2 + f_2) &= 2(y_1 + e_1)^2 - (y_2 + e_2), \end{aligned} \tag{5.11}$$

where $\mathbf{e} = [e_1 \ e_2]^T$ is the data perturbation vector and $\mathbf{f} = [f_1 \ f_2]^T$ is the estimator perturbation vector. Expanding (5.11) gives

$$\begin{aligned} \widehat{x}_1^2 + 2\widehat{x}_1 f_1 + f_1^2 &= 2(y_1 + e_1) + 4(y_2 + e_2) \\ (\widehat{x}_2 + f_2) &= 2(y_1^2 + 2y_1 e_1 + e_1^2) - (y_2 + e_2). \end{aligned} \tag{5.12}$$

Linearizing (5.12) gives

$$\begin{aligned} 2\widehat{x}_1 f_1 &= 2e_1 + 4e_2 \\ f_2 &= 4y_1 e_1 - e_2. \end{aligned} \tag{5.13}$$

In performing the linearization, we only keep terms that are exactly linear in the components of either $\mathbf{e}$ or $\mathbf{f}$. Constant terms, quadratic terms, and higher order terms are removed. Writing (5.13) in matrix form gives

$$\begin{bmatrix} f_1 \\ f_2 \end{bmatrix} = \begin{bmatrix} \frac{1}{\widehat{x}_1} & \frac{2}{\widehat{x}_1} \\ 4y_1 & -1 \end{bmatrix} \begin{bmatrix} e_1 \\ e_2 \end{bmatrix}, \tag{5.14}$$

which is exactly the same form as (5.9).

If the data for this problem happened to be $\mathbf{y} = [0, \ 1]^T$ then by (5.6), $\widehat{\mathbf{x}} = [2, \ -1]^T$ and by (5.10),

$$\mathbf{\Lambda}_{\widehat{\mathbf{x}}'} = \sigma^2 \begin{bmatrix} 5/4 & -1 \\ -1 & 1 \end{bmatrix}, \text{ and } \mathbf{\Lambda}_{\widehat{\mathbf{x}}'}^{-1} = \frac{1}{\sigma^2} \begin{bmatrix} 4 & 4 \\ 4 & 5 \end{bmatrix}. \tag{5.15}$$

The relatively large negative off-diagonal terms in the covariance matrix $\mathbf{\Lambda}_{\widehat{\mathbf{x}}'}$ tell us to expect sensitivity to noise, approximately parallel to the line $\widehat{x}_1 = -\widehat{x}_2$ and through $\widehat{\mathbf{x}} = [2 \ -1]^T$. Indeed, the eigendecomposition of $\mathbf{\Lambda}_{\widehat{\mathbf{x}}'}^{-1}$ is

$$\mathbf{\Lambda}_{\widehat{\mathbf{x}}'}^{-1} = \frac{1}{\sigma^2} \begin{bmatrix} 0.7497 & 0.6618 \\ -0.6618 & 0.7497 \end{bmatrix} \begin{bmatrix} 0.4689 & 0 \\ 0 & 8.5311 \end{bmatrix} \begin{bmatrix} 0.7497 & -0.6618 \\ 0.6618 & 0.7497 \end{bmatrix}. \tag{5.16}$$

From (5.16), we see that the curvature is relatively low along the direction parallel to $[0.7497 \ -0.6618]^T$ thus indicating relatively large uncertainty in that direction.

If we had another estimator for $\widehat{\mathbf{x}}$, it would generally have a different error ellipse, oriented in a different direction. If the angle between the major axes of the two different error ellipses is large enough, we expect that combining the estimates will produce a much more reliable estimate. This is illustrated in Figure 5-1.

If we have several different individual estimators $\widehat{\mathbf{x}}_i$ for the same quantities, as well as the estimator covariance matrices $\mathbf{\Lambda}_{\widehat{\mathbf{x}}_i'}$, we might be interested in combining their estimates in some optimal manner. For small enough perturbations where the linearization holds, the probability density function (PDF) of each estimate is Gaussian, and is denoted $N_i = N(\widehat{\mathbf{x}}_i, \mathbf{\Lambda}_{\widehat{\mathbf{x}}_i'})$. The global estimate also has a Gaussian density, and we therefore desire the point where this global density is maximized. If we assume that the local estimate perturbations are uncorrelated from one i to the next, the optimal global density is the product of the local densities, and the maximum is found via

$$\begin{aligned} \widehat{\mathbf{x}}_{opt} &= \arg\max_{\mathbf{x}} \textstyle\prod_i N_i, \\ &= \arg\max_{\mathbf{x}} \textstyle\sum_i \ln(N_i), \\ &= \arg\min_{\mathbf{x}} \textstyle\sum_i (\mathbf{x} - \widehat{\mathbf{x}}_i)^T \mathbf{\Lambda}_{\widehat{\mathbf{x}}_i'}^{-1} (\mathbf{x} - \widehat{\mathbf{x}}_i). \end{aligned} \tag{5.17}$$

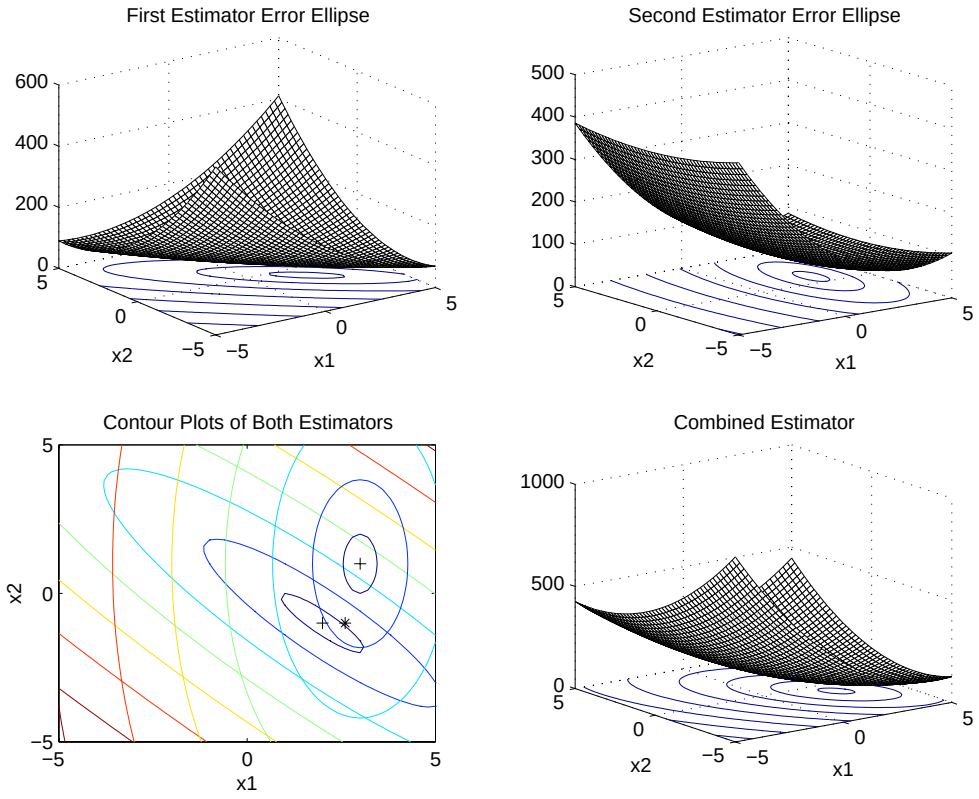

Figure 5-1: Combining estimates using error ellipses. The "+" symbols indicate the individual estimates and the "∗" indicates the combined estimate.

As an aside, an approximation to this approach which may at times be useful is to use only the major axis of the covariance matrix from each estimate. Each estimate contributes a line in the parameter space, and the problem is to the find the point closest to all the lines in the least square sense.

We can find the solution to (5.17) in closed-form. Expanding and collecting like terms in the last line of (5.17) gives

$$\widehat{\mathbf{x}}_{opt} = \arg\min_{\mathbf{x}} \sum_i \widehat{\mathbf{x}}_i^T \mathbf{\Lambda}_{\widehat{\mathbf{x}}_i'}^{-1} \widehat{\mathbf{x}}_i - 2\widehat{\mathbf{x}}_i^T \mathbf{\Lambda}_{\widehat{\mathbf{x}}_i'}^{-1} \mathbf{x} + \mathbf{x}^T \mathbf{\Lambda}_{\widehat{\mathbf{x}}_i'}^{-1} \mathbf{x}, \tag{5.18}$$

since $\mathbf{\Lambda}_{\widehat{\mathbf{x}}_i'}^{-1}$ is symmetric. This gives the same answer as

$$\begin{aligned} \widehat{\mathbf{x}}_{opt} &= \arg\min_{\mathbf{x}} \textstyle\sum_i \mathbf{x}^T \mathbf{\Lambda}_{\widehat{\mathbf{x}}_i'}^{-1} \mathbf{x} - 2\widehat{\mathbf{x}}_i^T \mathbf{\Lambda}_{\widehat{\mathbf{x}}_i'}^{-1} \mathbf{x}, \\ &= \arg\min_{\mathbf{x}} \mathbf{x}^T \left(\textstyle\sum_i \mathbf{\Lambda}_{\widehat{\mathbf{x}}_i'}^{-1} \right) \mathbf{x} - 2 \left(\textstyle\sum_i \widehat{\mathbf{x}}_i^T \mathbf{\Lambda}_{\widehat{\mathbf{x}}_i'}^{-1} \right) \mathbf{x}. \end{aligned} \tag{5.19}$$

Taking the derivative and setting equal to zero gives (again using symmetry)

$$\mathbf{0} = 2 \left(\sum_i \mathbf{\Lambda}_{\widehat{\mathbf{x}}_i'}^{-1} \right) \widehat{\mathbf{x}}_{opt} - 2 \sum_i \mathbf{\Lambda}_{\widehat{\mathbf{x}}_i'}^{-1} \widehat{\mathbf{x}}_i. \tag{5.20}$$

Finally, we obtain

$$\widehat{\mathbf{x}}_{opt} = \left(\sum_i \mathbf{\Lambda}_{\widehat{\mathbf{x}}_i'}^{-1} \right)^{-1} \sum_i \mathbf{\Lambda}_{\widehat{\mathbf{x}}_i'}^{-1} \widehat{\mathbf{x}}_i. \tag{5.21}$$

This is the key equation that gives a compact formula for the optimal combinations of estimates, and we will make extensive use of it throughout the chapter. It actually has a nice interpretation, which is: to optimally combine estimates, weight them by their (Fisher) information matrices, then add. This quantity is then equal to the optimal combined estimate, weighted by its (Fisher) information matrix.

5.4 Unification of Orientation using Error Ellipses

We saw in Section 5.3 how to combine multiple noisy estimates of a vector quantity through the use of error ellipses. In this section, we apply this idea to the problem of recovering motion parameters from noisy observations. Using this formulation, it will not matter if the observations are image feature points, 3-dimensional measured feature points, or perhaps even 3-dimensional stereo reconstructed points; the error ellipse approach handles all these cases in the same fashion.

Many authors have previously been concerned with more accurate modeling of errors in motion recovery algorithms. Ohta and Kanatani [139] discuss the absolute orientation problem in the presence of anisotropic and inhomogeneous noise. Those authors note that the conventional least squares solution [10], [89], [189] is inappropriate in many situations, namely if the 3-dimensional points are reconstructed from stereo pairs or range sensors. To solve for motion, Ohta and Kanatani apply a method they call "renormalization," which is exactly the same as the statistical bias removal approach we saw in Section 4.3.2.

Czurka *et al.* [37] use error ellipses to characterize the uncertainty of the fundamental matrix for uncalibrated stereo pairs. Hill, Cootes, and Taylor [87] consider the absolute orientation problem with nonscalar weights. They provide an iterative algorithm, which is the standard update of an incremental rotation matrix, given a current rotation matrix. Hel-Or and Werman [84], [85] make a similar modeling assumption. They start with the image plane perturbations, translated these to spherical coordinates, and then perform another change of variables to cartesian coordinates using Jacobian matrices. Since Hel-Or and Werman are working with covariance matrices, some of the elements of these matrices are necessarily infinite, a modeling approach that I would rather avoid. Simply working with FIMs solves this problem rather nicely, without the uneccessary change of variables. Although some of our FIMs are not invertible, from Section 5.3 we know that this merely corresponds to an extremely elongated error ellipse, so the lack of invertibility will not be a problem.

Assume we have two sets of corresponding noisy features, $\{\mathbf{l}_i\}$ and $\{\mathbf{r}_i\}$, for $i = 1, \ldots, n$, where the set elements are 3×1 vectors representing world coordinates in different coordinate systems. Associated with each $\mathbf{l}_i$ is an information matrix $\mathbf{\Lambda}_{\mathbf{l}_i}^{-1}$, and associated with each $\mathbf{r}_i$ is an information matrix $\mathbf{\Lambda}_{\mathbf{r}_i}^{-1}$. As in Section 5.3, we will need to model the corresponding feature point data as noisy observations of an unknown quantity. The noise-free version of the points $\mathbf{r}_i$ are therefore rotated by the rotation matrix $\mathbf{Q}$ and then translated by $\mathbf{t}$ into the $\mathbf{l}_i$ coordinate reference frame. If we let $\{\mathbf{x}_i\}$ be the set of 3×1 vectors of true locations in the $\mathbf{l}_i$ coordinate frame, then both $\mathbf{l}_i$ and $\mathbf{Q}\mathbf{r}_i + \mathbf{t}$ may be considered "estimates" of $\mathbf{x}_i$. If we assume that the estimates are unbiased, then using (5.21), these estimates are optimally

combined as

$$\begin{aligned}\widehat{\mathbf{x}}_i &= \left(\mathbf{\Lambda}_{\mathbf{l}_i}^{-1} + \mathbf{Q}\mathbf{\Lambda}_{\mathbf{r}_i}^{-1}\mathbf{Q}^T\right)^{-1}\left(\mathbf{\Lambda}_{\mathbf{l}_i}^{-1}\mathbf{l}_i + \mathbf{Q}\mathbf{\Lambda}_{\mathbf{r}_i}^{-1}\mathbf{Q}^T(\mathbf{Q}\mathbf{r}_i + \mathbf{t})\right) \\ &= \left(\mathbf{\Lambda}_{\mathbf{l}_i}^{-1} + \mathbf{Q}\mathbf{\Lambda}_{\mathbf{r}_i}^{-1}\mathbf{Q}^T\right)^{-1}\left(\mathbf{\Lambda}_{\mathbf{l}_i}^{-1}\mathbf{l}_i + \mathbf{Q}\mathbf{\Lambda}_{\mathbf{r}_i}^{-1}\mathbf{r}_i + \mathbf{Q}\mathbf{\Lambda}_{\mathbf{r}_i}^{-1}\mathbf{Q}^T\mathbf{t}\right).\end{aligned} \tag{5.22}$$

Some observations about (5.22) are in order. If in reality, one set of features (say, the $\mathbf{l}_i$) are image points, then we have $\mathbf{l}_i = l_i\mathbf{p}_i$, where l_i is the unknown depth of image point $\mathbf{p}_i$. As we shall see, the beauty of the error ellipse approach is that we will never have to carry around unknown depths in the equations for motion recovery.

The FIM of the estimate $\widehat{\mathbf{x}}_i$ is

$$\mathbf{\Lambda}_{\widehat{\mathbf{x}}_i}^{-1} = \mathbf{\Lambda}_{\mathbf{l}_i}^{-1} + \mathbf{Q}\mathbf{\Lambda}_{\mathbf{r}_i}^{-1}\mathbf{Q}^T. \tag{5.23}$$

The main diagonal elements of this FIM have the interesting property that their sum does not depend on the unknown $\mathbf{Q}$. This can be seen by taking the trace of the FIM:

$$\begin{aligned}\mathrm{tr}(\mathbf{\Lambda}_{\widehat{\mathbf{x}}_i}^{-1}) &= \mathrm{tr}(\mathbf{\Lambda}_{\mathbf{l}_i}^{-1} + \mathbf{Q}\mathbf{\Lambda}_{\mathbf{r}_i}^{-1}\mathbf{Q}^T) \\ &= \mathrm{tr}(\mathbf{\Lambda}_{\mathbf{l}_i}^{-1}) + \mathrm{tr}(\mathbf{Q}\mathbf{\Lambda}_{\mathbf{r}_i}^{-1}\mathbf{Q}^T) \\ &= \mathrm{tr}(\mathbf{\Lambda}_{\mathbf{l}_i}^{-1}) + \mathrm{tr}(\mathbf{\Lambda}_{\mathbf{r}_i}^{-1}).\end{aligned} \tag{5.24}$$

This is of interest because of the known inequality (see [103], page 65)

$$[\mathbf{\Lambda}_{\widehat{\mathbf{x}}_i}^{-1}]_{jj} \geq \frac{1}{[\mathbf{\Lambda}_{\widehat{\mathbf{x}}_i}]_{jj}}, \tag{5.25}$$

where the term in the denominator is the variance of the j^{th} component of the estimate $\widehat{\mathbf{x}}$. Combining this with (5.24), we have

$$\mathrm{tr}(\mathbf{\Lambda}_{\mathbf{l}_i}^{-1}) + \mathrm{tr}(\mathbf{\Lambda}_{\mathbf{r}_i}^{-1}) \geq \sum_j \frac{1}{[\mathbf{\Lambda}_{\widehat{\mathbf{x}}_i}]_{jj}}. \tag{5.26}$$

From (5.26) we see that definite, easily calculable lower bounds exist on the achievable variances in the components of the estimate.

5.4.1 FIM Catalogue

We now catalogue the various FIMs for observations according to the nature of the observation.

(*i*) In the absolute orientation problem it is generally assumed that the location errors are spherical, so that

$$\mathbf{\Lambda}_{Abs}^{-1} = \sigma^{-2}\mathbf{I}. \tag{5.27}$$

This situation is illustrated in Figure 5-2.

(*ii*) Under orthographic projection, we have no information on the feature point's z-coordinate, and therefore

$$\mathbf{\Lambda}_{Ortho}^{-1} = \sigma^{-2}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \tag{5.28}$$

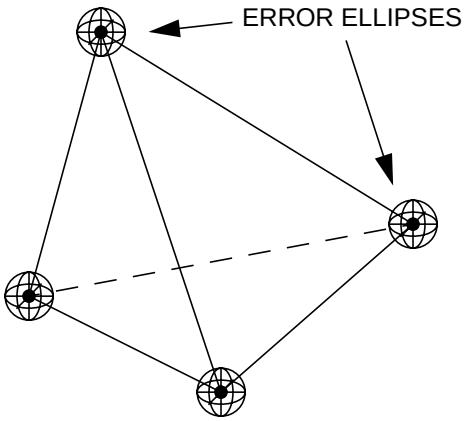

Figure 5-2: Error Ellipses for absolute orientation feature points.

(which points out the abuse of notation that can develop, because the matrix $\mathbf{\Lambda}_{Ortho}^{-1}$ is not invertible). With this model, the error ellipse is essentially stretched to infinity in the z-direction, as illustrated in Figure 5-3.

(*iii*) Under perspective projection, we generally model the uncertainty in the observation as occuring only in the image plane. With this model we have a circular error ellipse in the image plane, and no information in the direction of the point $\mathbf{p} = [u,\ v,\ 1]^T$. In order to obtain a symmetric FIM, we construct a perpendicular triad for the principal axes of the error ellipse. One direction will be in the $\mathbf{p}$ direction, the second direction will be in the image plane and is given by $\mathbf{p}_1 = [-v,\ u,\ 0]^T$, and the third direction is found from the cross product of $\mathbf{p}$ and $\mathbf{p}_1$ to be $\mathbf{p}_2 = [-u,\ -v,\ u^2+v^2]^T$. This is shown in Figure 5-4. We require

$$\begin{aligned} \tfrac{\mathbf{p}^T}{\|\mathbf{p}\|}\mathbf{\Lambda}_{Persp1}^{-1}\tfrac{\mathbf{p}}{\|\mathbf{p}\|} &= 0, \\ \tfrac{\mathbf{p}_1^T}{\|\mathbf{p}_1\|}\mathbf{\Lambda}_{Persp1}^{-1}\tfrac{\mathbf{p}_1}{\|\mathbf{p}_1\|} &= \sigma^{-2}, \\ \tfrac{\mathbf{p}_2^T}{\|\mathbf{p}_2\|}\mathbf{\Lambda}_{Persp1}^{-1}\tfrac{\mathbf{p}_2}{\|\mathbf{p}_2\|} &= \lambda\sigma^{-2}, \end{aligned} \tag{5.29}$$

for some λ, chosen so that the slice of the error ellipse in the image plane is circular. By choosing

$$\begin{aligned} \mathbf{\Lambda}_{Persp1}^{-1} &= \tfrac{\sigma^{-2}}{u^2+v^2}\begin{bmatrix} \mathbf{p} & \mathbf{p}_1 & \mathbf{p}_2 \end{bmatrix}\begin{bmatrix} 0 & & \\ & 1 & \\ & & 1 \end{bmatrix}\begin{bmatrix} \mathbf{p}^T \\ \mathbf{p}_1^T \\ \mathbf{p}_2^T \end{bmatrix} \\ &= \tfrac{\sigma^{-2}}{u^2+v^2}\begin{bmatrix} -v & -u \\ u & -v \\ 0 & u^2+v^2 \end{bmatrix}\begin{bmatrix} -v & u & 0 \\ -u & -v & u^2+v^2 \end{bmatrix}, \end{aligned} \tag{5.30}$$

we obtain the proper FIM. If both $u = 0$ and $v = 0$, we simply use the orthographic projection FIM given in (5.28), since $\mathbf{\Lambda}_{Persp1}^{-1} \to \mathbf{\Lambda}_{Ortho}^{-1}$ as $u^2+v^2 \to 0$.

(*iv*) An approximation that may be helpful is to assume we have no information in the direction of the image feature point, and that the error ellipse is circular in the plane normal

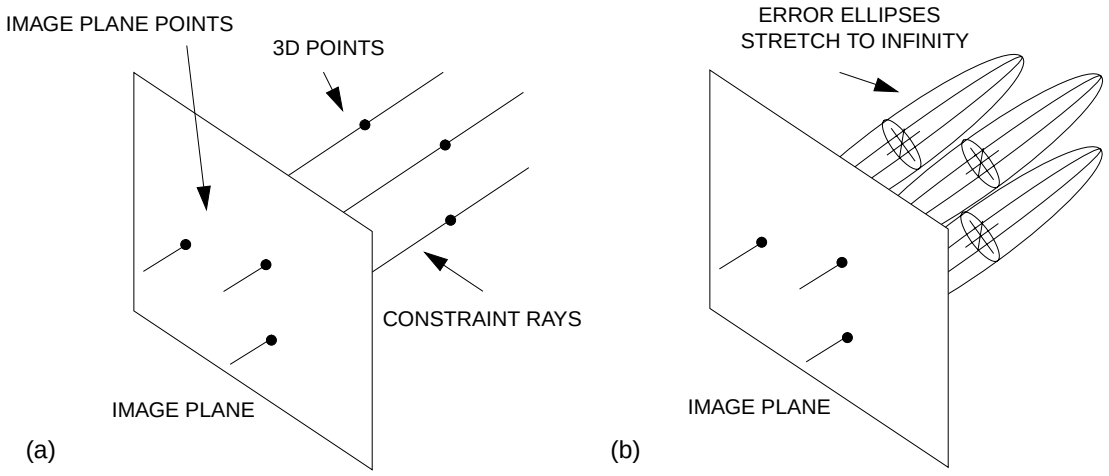

Figure 5-3: Unified Orthographic Projection. (a) True orthographic projection. (b) Ellipses resulting from image feature point location uncertainties.

to the direction of the image feature point. This is a mild generalization of the orthographic projection, which is appropriate for a narrow FOV, and is illustrated in Figure 5-5. (It would be exact if we had a "hemispherical focal plane array.") The FIM in this case is

$$\mathbf{\Lambda}_{Persp2}^{-1} = \sigma^{-2}\left(\mathbf{I} - \frac{\mathbf{p}\mathbf{p}^T}{\mathbf{p}^T\mathbf{p}}\right). \tag{5.31}$$

The beauty of using the FIMs in either (5.28), (5.30), or (5.31) is that the inconvenient homogeneous scale factors corresponding to unknown depth can never appear in the reconstruction (5.22) because we have from either (5.28), (5.30), or (5.31)

$$\mathbf{\Lambda}_i^{-1} l_i \mathbf{p}_i = \mathbf{0}. \tag{5.32}$$

(*v*) The FIM (5.31) is related to an error model for "range images". That is, we assume that we have a device (a laser, perhaps) that provides an estimate of the distance in a measured direction. In this case, the FIM is

$$\begin{aligned} \mathbf{\Lambda}_{Range}^{-1} &= \sigma^{-2}\left(\mathbf{I} - \tfrac{\mathbf{p}\mathbf{p}^T}{\mathbf{p}^T\mathbf{p}}\right) + \sigma_d^{-2}\tfrac{\mathbf{p}\mathbf{p}^T}{\mathbf{p}^T\mathbf{p}} \\ &= \sigma^{-2}\mathbf{I} + (\sigma_d^{-2} - \sigma^{-2})\tfrac{\mathbf{p}\mathbf{p}^T}{\mathbf{p}^T\mathbf{p}}. \end{aligned} \tag{5.33}$$

(*vi*) For a calibrated stereo rig (i.e. $\mathbf{Q}$ and $\mathbf{t}$ known), we may use (5.22) to reconstruct the points in space, and thereby consider the reconstructed points as "observations" (see Figure 5-7). The FIM of these observations then is simply

$$\mathbf{\Lambda}_{Stereo}^{-1} = \mathbf{\Lambda}_{\mathbf{l}}^{-1} + \mathbf{Q}\mathbf{\Lambda}_{\mathbf{r}}^{-1}\mathbf{Q}^T, \tag{5.34}$$

where $\mathbf{\Lambda}_{\mathbf{r}}^{-1}$ is obtained from either (5.30) or (5.31).

(*vii*) For a sequential updating approach to camera motion, where we have an updated estimate of the 3-dimensional locations for every newly acquired data frame, (5.22) and

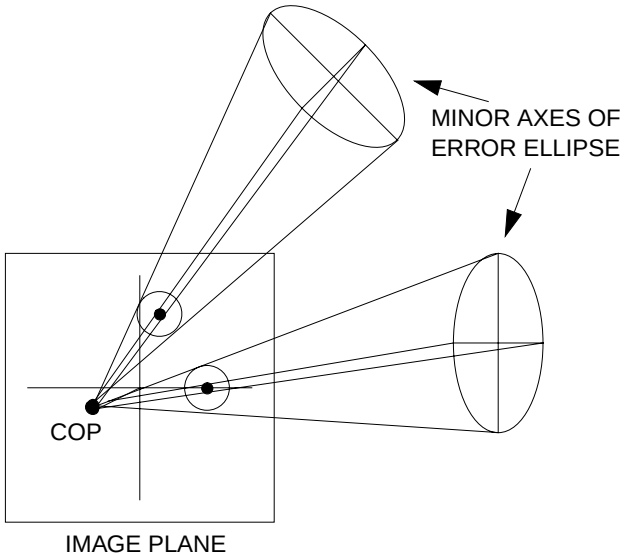

Figure 5-4: Unified Perspective Projection. Ellipses resulting from circular image feature point location uncertainties. The minor principal axes of the ellipse are all different lengths.

(5.23) yields a particularly simply recursive updating procedure. Let $\mathbf{Q}_k$ and $\mathbf{t}_k$ be the k^{th} motion relative to a fixed world frame, and $\mathbf{p}_k$ a particular point in image k. Then

$$\begin{aligned} \mathbf{\Lambda}_{\widehat{\mathbf{x}}_k}^{-1} &= \mathbf{\Lambda}_{\widehat{\mathbf{x}}_{k-1}}^{-1} + \mathbf{Q}_k \mathbf{\Lambda}_{\mathbf{p}_k}^{-1} \mathbf{Q}_k^T \\ &= \textstyle\sum_{m=1}^{k} \mathbf{Q}_m \mathbf{\Lambda}_{\mathbf{p}_m}^{-1} \mathbf{Q}_m^T \end{aligned} \tag{5.35}$$

and also

$$\begin{aligned} \mathbf{\Lambda}_{\widehat{\mathbf{x}}_k}^{-1} \widehat{\mathbf{x}}_k &= \mathbf{\Lambda}_{\widehat{\mathbf{x}}_{k-1}}^{-1} \widehat{\mathbf{x}}_{k-1} + \mathbf{Q}_k \mathbf{\Lambda}_{\mathbf{p}_k}^{-1} \mathbf{p}_k + \mathbf{Q}_k \mathbf{\Lambda}_{\mathbf{p}_k}^{-1} \mathbf{Q}_k^T \mathbf{t}_k, \\ &= \textstyle\sum_{m=1}^{k} \mathbf{Q}_m \mathbf{\Lambda}_{\mathbf{p}_m}^{-1} (\mathbf{p}_m + \mathbf{Q}_m^T \mathbf{t}_m). \end{aligned} \tag{5.36}$$

The top lines of (5.35) and (5.36) would be used in the sequential updating approach, and the bottom lines for a batch approach. The updating is very inexpensive, due to sharing of calculations between the two equations. For the sequential approach, we do not actually calculate $\widehat{\mathbf{x}}_k$ in (5.36), but instead perform the iteration

$$\mathbf{y}_k = \mathbf{y}_{k-1} + \mathbf{Q}_k \mathbf{\Lambda}_{\mathbf{p}_k}^{-1} \mathbf{p}_k + \mathbf{Q}_k \mathbf{\Lambda}_{\mathbf{p}_k}^{-1} \mathbf{Q}_k^T \mathbf{t}_k. \tag{5.37}$$

At any time k, if the actual estimate $\widehat{\mathbf{x}}_k$ is required, we calculate this using

$$\widehat{\mathbf{x}}_k = \mathbf{\Lambda}_{\widehat{\mathbf{x}}_k} \mathbf{y}_k. \tag{5.38}$$

This sequential formulation is similiar to an approach taken recently by Thomas and Oliensis [177].

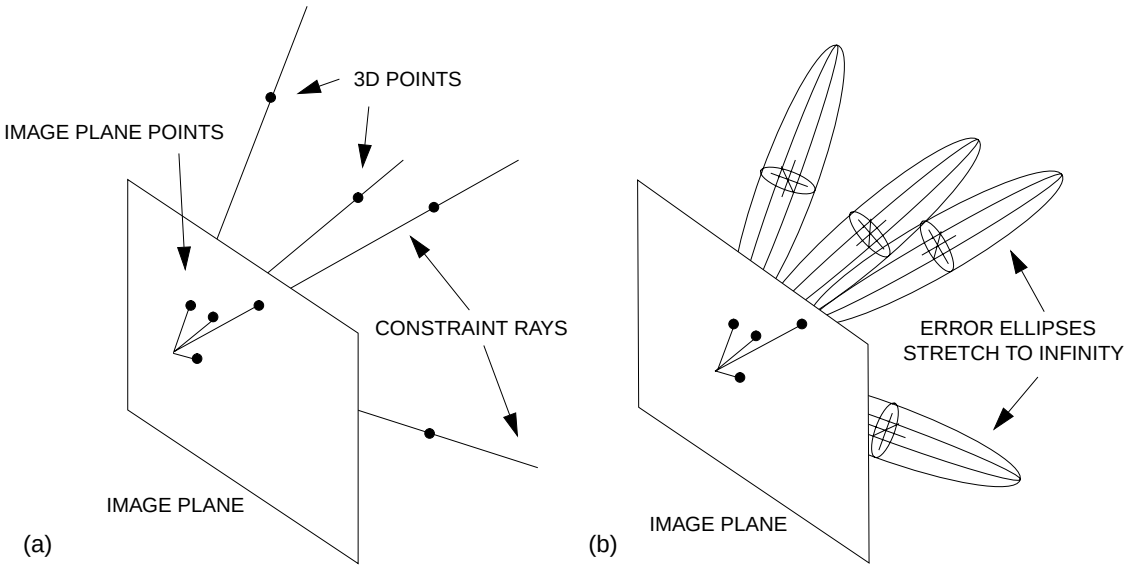

Figure 5-5: Unified Perspective Projection Approximation. (a) True perspective projection. (b) Ellipses resulting from image feature point location uncertainties. The principal axes of the ellipse are not in the image plane.

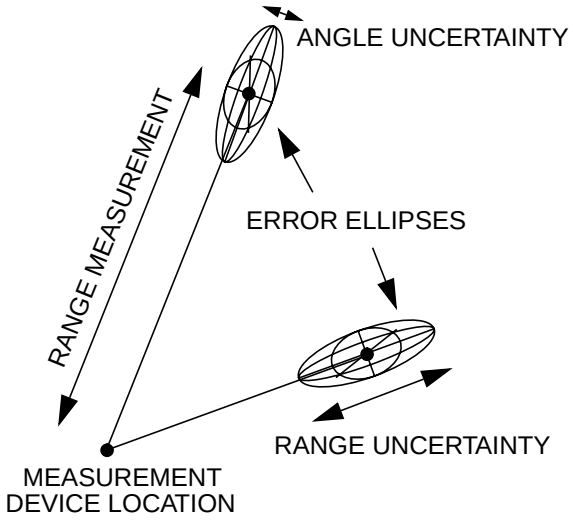

Figure 5-6: Unified Range Image. Ellipses resulting from angular and range uncertainties.

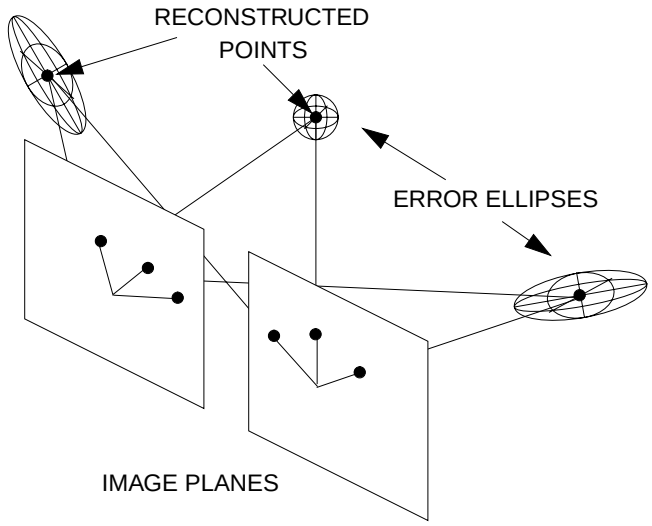

Figure 5-7: Unified Stereo Projection. Ellipses resulting from image feature point location uncertainties in reconstruction from stereo.

5.4.2 Reduction to Standard Models

Next, we show how the reconstructions given by (5.22) reduce to reasonable and satisfying results for absolute, exterior, and relative orientation. For absolute orientation, let the variances in (5.27) of the left and right data sets be denoted by σ_l^2 and σ_r^2. Substituting (5.27) into (5.22) gives after simplification

$$\widehat{\mathbf{x}}_i = \frac{\sigma_l^{-2}}{\sigma_l^{-2} + \sigma_r^{-2}}\mathbf{l}_i + \frac{\sigma_r^{-2}}{\sigma_l^{-2} + \sigma_r^{-2}}(\mathbf{Q}\mathbf{r}_i + \mathbf{t}). \tag{5.39}$$

From (5.39), we see that the optimal estimate is a convex combination of the left and right data sets (with $\mathbf{r}_i$ brought into the left coordinate frame, of course). As the relative sizes of σ_l^2 and σ_r^2 change, (5.39) will accordingly change the weighting of the two data sets in the optimal estimate. In the extreme, when the variance of one data set is infinite, the algorithm will totally ignore that data set.

For exterior orientation, we have $\mathbf{\Lambda}_{\mathbf{l}_i}^{-1}$ from either (5.30) or (5.31) (we use the latter here for illustration) and $\mathbf{\Lambda}_{\mathbf{r}_i}^{-1}$ from (5.27). Let $\mathbf{p}$ be the image point of $\mathbf{l}_i$. Defining $\alpha = \sigma_r^2/\sigma_l^2$, and noting that $\mathbf{\Lambda}_{\mathbf{l}_i}^{-1}\mathbf{l}_i = \mathbf{0}$, we have from (5.22)

$$\begin{aligned} \widehat{\mathbf{x}}_i &= \sigma_r^{-2}\left(\sigma_l^{-2}\left(\mathbf{I} - \frac{\mathbf{p}\mathbf{p}^T}{\mathbf{p}^T\mathbf{p}}\right) + \sigma_r^{-2}\mathbf{I}\right)^{-1}(\mathbf{Q}\mathbf{r}_i + \mathbf{t}) \\ &= \left((1+\alpha)\mathbf{I} - \alpha\frac{\mathbf{p}\mathbf{p}^T}{\mathbf{p}^T\mathbf{p}}\right)^{-1}(\mathbf{Q}\mathbf{r}_i + \mathbf{t}). \end{aligned} \tag{5.40}$$

At $\alpha = 0$, we see that the image data is ignored. Bringing the matrix over to the left hand side, and letting $\alpha \to \infty$, the 3-dimensional data $\mathbf{r}_i$ becomes unreliable. The equation thus approaches

$$\left(\mathbf{I} - \frac{\mathbf{p}\mathbf{p}^T}{\mathbf{p}^T\mathbf{p}}\right)\widehat{\mathbf{x}}_i = \frac{1}{\alpha}(\mathbf{Q}\mathbf{r}_i + \mathbf{t}) \to \mathbf{0}, \tag{5.41}$$

and the only solution is the homogeneous solution $\widehat{\mathbf{x}}_i = z\mathbf{p}$, for some unknown depth z. This is the ray emanating from the camera center of projection (COP) in the direction of the observed image feature point.

For relative orientation, we obtain both $\mathbf{\Lambda}_{\mathbf{l}_i}^{-1}$ and $\mathbf{\Lambda}_{\mathbf{r}_i}^{-1}$ from either (5.30) or (5.31). Noting now that both $\mathbf{\Lambda}_{\mathbf{l}_i}^{-1}\mathbf{l}_i = \mathbf{0}$ and $\mathbf{\Lambda}_{\mathbf{r}_i}^{-1}\mathbf{r}_i = \mathbf{0}$, we have from (5.22)

$$\widehat{\mathbf{x}}_i = \left(\mathbf{\Lambda}_{\mathbf{l}_i}^{-1} + \mathbf{Q}\mathbf{\Lambda}_{\mathbf{r}_i}^{-1}\mathbf{Q}^T\right)^{-1}\left(\mathbf{Q}\mathbf{\Lambda}_{\mathbf{r}_i}^{-1}\mathbf{Q}^T\right)\mathbf{t}. \tag{5.42}$$

As $\alpha = \sigma_r^2/\sigma_l^2 \to \infty$, the right image is increasingly unreliable, and we get the same behavior as in the exterior orientation situation (i.e. the estimate is along the ray in the left coordinate system). For $\alpha \to 0$, we rearrange (5.42), set $\mathbf{\Lambda}_{\mathbf{l}_i}^{-1} = \mathbf{0}$, and premultiply by $\mathbf{Q}^T$ giving

$$\mathbf{\Lambda}_{\mathbf{r}_i}^{-1}\mathbf{Q}^T\widehat{\mathbf{x}}_i = \mathbf{\Lambda}_{\mathbf{r}_i}^{-1}\mathbf{Q}^T\mathbf{t}. \tag{5.43}$$

Bringing everything over to the left hand side and factoring,

$$\mathbf{\Lambda}_{\mathbf{r}_i}^{-1}\mathbf{Q}^T(\widehat{\mathbf{x}}_i - \mathbf{t}) = \mathbf{0}. \tag{5.44}$$

Again, only the homogeneous solution is possible, which is $\mathbf{Q}^T(\widehat{\mathbf{x}}_i - \mathbf{t}) = z\mathbf{p}$ (the z and $\mathbf{p}$ are the right-hand depth and right-hand image coordinates). Thus, the optimal estimate is the ray emanating from the right-hand camera COP, in the direction of the right-hand image feature point, measured here in the left hand coordinate system.

In summary, the optimal estimate (5.22) does indeed give reasonable results for absolute, exterior, and relative orientation reconstructions.

5.4.3 Estimating Motion with Error Ellipses

Recall that from the maximum likelihood estimate given by (5.17), we were able to obtain the best combination of estimates in closed-form in (5.21) and apply it to motion estimation in (5.22). If we were to substitute (5.22) into the ML cost function (5.17) and add the cost functions for all observed points, we would obtain the total cost as a function of the rotations, translations, and the observed data. This ML cost could then be minimized through choice of the rotations and translations, yielding optimal estimates. We do this now for the various orientation problems. The optimization criterion is

$$\mathbf{Q}_{opt}, \mathbf{t}_{opt} = \arg\min_{\mathbf{Q},\mathbf{t}} \; J, \tag{5.45}$$

with

$$\begin{aligned} J = \textstyle\sum_i \quad & \left(\widehat{\mathbf{x}}_i - \mathbf{l}_i\right)^T \mathbf{\Lambda}_{\mathbf{l}_i}^{-1} \left(\widehat{\mathbf{x}}_i - \mathbf{l}_i\right) \\ & + \left(\widehat{\mathbf{x}}_i - (\mathbf{Q}\mathbf{r}_i + \mathbf{t})\right)^T \mathbf{Q}\mathbf{\Lambda}_{\mathbf{r}_i}^{-1}\mathbf{Q}^T \left(\widehat{\mathbf{x}}_i - (\mathbf{Q}\mathbf{r}_i + \mathbf{t})\right), \end{aligned} \tag{5.46}$$

where $\widehat{\mathbf{x}}_i$ is the reconstructed 3-dimensional point using the appropriate formula from Section 5.4.2 and where i is the index of matched feature points in the data sets.

(i) For absolute orientation, we take $\mathbf{\Lambda}_{\mathbf{l}_i}^{-1} = \mathbf{\Lambda}_{\mathbf{r}_i}^{-1} = \mathbf{I}$, and (5.46) becomes

$$J = \sum_i \left(\widehat{\mathbf{x}}_i - \mathbf{l}_i\right)^T \left(\widehat{\mathbf{x}}_i - \mathbf{l}_i\right) + \left(\widehat{\mathbf{x}}_i - (\mathbf{Q}\mathbf{r}_i + \mathbf{t})\right)^T \left(\widehat{\mathbf{x}}_i - (\mathbf{Q}\mathbf{r}_i + \mathbf{t})\right). \tag{5.47}$$

Substituting the optimal estimate $\widehat{\mathbf{x}}_i = (\mathbf{l}_i + \mathbf{Q}\mathbf{r}_i + \mathbf{t})/2$ from (5.39) into (5.47) gives after simplification

$$J = \sum_i \|\mathbf{Q}\mathbf{r}_i + \mathbf{t} - \mathbf{l}_i\|^2. \tag{5.48}$$

This is precisely the minimization problem solved by the standard absolute orientation approach using the Orthogonal Procrustes method. This development also serves as a convincing proof that the standard absolute orientation method is a maximum likelihood method (for Gaussian noise perturbations of feature point locations).

(ii) For relative orientation, we have $\mathbf{\Lambda}_{\mathbf{l}_i}^{-1}$ and $\mathbf{\Lambda}_{\mathbf{r}_i}^{-1}$ from (5.30). Since $\mathbf{\Lambda}_{\mathbf{l}_i}^{-1}\mathbf{l}_i = \mathbf{\Lambda}_{\mathbf{r}_i}^{-1}\mathbf{r}_i = \mathbf{0}$, (5.46) becomes

$$\begin{aligned} J &= \textstyle\sum_i \widehat{\mathbf{x}}_i^T \mathbf{\Lambda}_{\mathbf{l}_i}^{-1} \widehat{\mathbf{x}}_i + \left(\widehat{\mathbf{x}}_i - \mathbf{t}\right)^T \mathbf{Q}\mathbf{\Lambda}_{\mathbf{r}_i}^{-1}\mathbf{Q}^T \left(\widehat{\mathbf{x}}_i - \mathbf{t}\right), \\ &= \textstyle\sum_i \widehat{\mathbf{x}}_i^T \left(\mathbf{\Lambda}_{\mathbf{l}_i}^{-1} + \mathbf{Q}\mathbf{\Lambda}_{\mathbf{r}_i}^{-1}\mathbf{Q}^T\right) \widehat{\mathbf{x}}_i - 2\widehat{\mathbf{x}}_i^T \mathbf{Q}\mathbf{\Lambda}_{\mathbf{r}_i}^{-1}\mathbf{Q}^T\mathbf{t} + \mathbf{t}^T\mathbf{Q}\mathbf{\Lambda}_{\mathbf{r}_i}^{-1}\mathbf{Q}^T\mathbf{t}. \end{aligned} \tag{5.49}$$

From (5.22) we have $\left(\mathbf{\Lambda}_{\mathbf{l}_i}^{-1} + \mathbf{Q}\mathbf{\Lambda}_{\mathbf{r}_i}^{-1}\mathbf{Q}^T\right) \widehat{\mathbf{x}}_i = \mathbf{Q}\mathbf{\Lambda}_{\mathbf{r}_i}^{-1}\mathbf{Q}^T\mathbf{t}$. Substituting this into (5.49), we have

$$\begin{aligned} J &= \textstyle\sum_i \widehat{\mathbf{x}}_i^T \mathbf{Q}\mathbf{\Lambda}_{\mathbf{r}_i}^{-1}\mathbf{Q}^T\mathbf{t} - 2\widehat{\mathbf{x}}_i^T \mathbf{Q}\mathbf{\Lambda}_{\mathbf{r}_i}^{-1}\mathbf{Q}^T\mathbf{t} + \mathbf{t}^T\mathbf{Q}\mathbf{\Lambda}_{\mathbf{r}_i}^{-1}\mathbf{Q}^T\mathbf{t}, \\ &= \textstyle\sum_i \mathbf{t}^T\mathbf{Q}\mathbf{\Lambda}_{\mathbf{r}_i}^{-1}\mathbf{Q}^T\mathbf{t} - \widehat{\mathbf{x}}_i^T \mathbf{Q}\mathbf{\Lambda}_{\mathbf{r}_i}^{-1}\mathbf{Q}^T\mathbf{t}, \\ &= \textstyle\sum_i \mathbf{t}^T\mathbf{Q}\mathbf{\Lambda}_{\mathbf{r}_i}^{-1}\mathbf{Q}^T\mathbf{t} - \mathbf{t}^T\mathbf{Q}\mathbf{\Lambda}_{\mathbf{r}_i}^{-1}\mathbf{Q}^T \left(\mathbf{\Lambda}_{\mathbf{l}_i}^{-1} + \mathbf{Q}\mathbf{\Lambda}_{\mathbf{r}_i}^{-1}\mathbf{Q}^T\right)^{-1} \mathbf{Q}\mathbf{\Lambda}_{\mathbf{r}_i}^{-1}\mathbf{Q}^T\mathbf{t}, \\ &= \mathbf{t}^T \textstyle\sum_i \left(\mathbf{Q}\mathbf{\Lambda}_{\mathbf{r}_i}^{-1}\mathbf{Q}^T - \mathbf{Q}\mathbf{\Lambda}_{\mathbf{r}_i}^{-1}\mathbf{Q}^T \left(\mathbf{\Lambda}_{\mathbf{l}_i}^{-1} + \mathbf{Q}\mathbf{\Lambda}_{\mathbf{r}_i}^{-1}\mathbf{Q}^T\right)^{-1} \mathbf{Q}\mathbf{\Lambda}_{\mathbf{r}_i}^{-1}\mathbf{Q}^T\right)\mathbf{t}, \\ &= \mathbf{t}^T \textstyle\sum_i \left(\mathbf{Q}\mathbf{\Lambda}_{\mathbf{r}_i}^{-1}\mathbf{Q}^T \left(\mathbf{\Lambda}_{\mathbf{l}_i}^{-1} + \mathbf{Q}\mathbf{\Lambda}_{\mathbf{r}_i}^{-1}\mathbf{Q}^T\right)^{-1} \mathbf{\Lambda}_{\mathbf{l}_i}^{-1}\right)\mathbf{t}, \end{aligned} \tag{5.50}$$

where, in the last line, we have used the matrix identity $\mathbf{A}-\mathbf{A}(\mathbf{B}+\mathbf{A})^{-1}\mathbf{A} = \mathbf{A}(\mathbf{B}+\mathbf{A})^{-1}\mathbf{B}$. We see then that $\mathbf{t}_{opt}$ is simply determined in closed-form from $\mathbf{Q} = \mathbf{Q}_{opt}$ as the eigenvector corresponding to the minimum eigenvalue of

$$\sum_i \left(\mathbf{Q}\mathbf{\Lambda}_{\mathbf{r}_i}^{-1}\mathbf{Q}^T \left(\mathbf{\Lambda}_{\mathbf{l}_i}^{-1} + \mathbf{Q}\mathbf{\Lambda}_{\mathbf{r}_i}^{-1}\mathbf{Q}^T \right)^{-1} \mathbf{\Lambda}_{\mathbf{l}_i}^{-1} \right), \tag{5.51}$$

and $\mathbf{Q}_{opt}$ is chosen so that the minimum eigenvalue of the matrix in (5.51) is minimized. Note that we can only determine $\mathbf{t}$ up to a scale factor, which is well known for the relative orientation problem. As long as there is some motion, the matrix inverse required in (5.51) exists, and we have therefore decoupled the estimation of the rotation and translation.

To find $\mathbf{Q}_{opt}$ then requires a search with three degrees of freedom (i.e. use the unit quaternion representation). Although the minimum eigenvalue can be written in closed-form as a function of the unit quaternion elements, this does not appear to be fruitful.

(*iii*) For exterior orientation, we have $\mathbf{\Lambda}_{\mathbf{l}_i}^{-1}\mathbf{l}_i = \mathbf{0}$ and $\mathbf{\Lambda}_{\mathbf{r}_i}^{-1} = \mathbf{I}$, and (5.46) becomes

$$J = \textstyle\sum_i \widehat{\mathbf{x}}_i^T \mathbf{\Lambda}_{\mathbf{l}_i}^{-1} \widehat{\mathbf{x}}_i + (\widehat{\mathbf{x}}_i - (\mathbf{Q}\mathbf{r}_i + \mathbf{t}))^T (\widehat{\mathbf{x}}_i - (\mathbf{Q}\mathbf{r}_i + \mathbf{t})). \tag{5.52}$$

From (5.22) we have $\widehat{\mathbf{x}}_i = \left(\mathbf{\Lambda}_{\mathbf{l}_i}^{-1} + \mathbf{I} \right)^{-1} (\mathbf{Q}\mathbf{r}_i + \mathbf{t})$ and therefore

$$\begin{aligned} \widehat{\mathbf{x}}_i - (\mathbf{Q}\mathbf{r}_i + \mathbf{t}) &= \left(\left(\mathbf{\Lambda}_{\mathbf{l}_i}^{-1} + \mathbf{I} \right)^{-1} - \mathbf{I} \right) (\mathbf{Q}\mathbf{r}_i + \mathbf{t}) \\ &= -\left(\mathbf{\Lambda}_{\mathbf{l}_i}^{-1} + \mathbf{I} \right)^{-1} \mathbf{\Lambda}_{\mathbf{l}_i}^{-1} (\mathbf{Q}\mathbf{r}_i + \mathbf{t}), \end{aligned} \tag{5.53}$$

(again using the matrix identity). We also have

$$\widehat{\mathbf{x}}_i^T \mathbf{\Lambda}_{\mathbf{l}_i}^{-1} \widehat{\mathbf{x}}_i = (\mathbf{Q}\mathbf{r}_i + \mathbf{t})^T \mathbf{\Lambda}_{\mathbf{l}_i}^{-1} \left(\mathbf{\Lambda}_{\mathbf{l}_i}^{-1} + \mathbf{I} \right)^{-2} (\mathbf{Q}\mathbf{r}_i + \mathbf{t}), \tag{5.54}$$

where we have used the fact that $\mathbf{\Lambda}_{\mathbf{l}_i}^{-1}$ and $\left(\mathbf{\Lambda}_{\mathbf{l}_i}^{-1} + \mathbf{I} \right)^{-1}$ are commutative. Substituting (5.53) and (5.54) into (5.52), we obtain after simplification

$$J = \sum_i (\mathbf{Q}\mathbf{r}_i + \mathbf{t})^T \mathbf{\Lambda}_{\mathbf{l}_i}^{-1} \left(\mathbf{\Lambda}_{\mathbf{l}_i}^{-1} + \mathbf{I} \right)^{-1} (\mathbf{Q}\mathbf{r}_i + \mathbf{t}). \tag{5.55}$$

We can solve for $\mathbf{t}_{opt}$ as a function of $\mathbf{Q}_{opt}$ in closed-form by differentating J with respect to $\mathbf{t}$ and setting the result equal to zero. We obtain

$$\mathbf{t}_{opt} = -\left(\sum_i \mathbf{A}_i \right)^{-1} \left(\sum_i \mathbf{A}_i \mathbf{Q}_{opt} \mathbf{r}_i \right), \tag{5.56}$$

where $\mathbf{A}_i \triangleq \mathbf{\Lambda}_{\mathbf{l}_i}^{-1} \left(\mathbf{\Lambda}_{\mathbf{l}_i}^{-1} + \mathbf{I} \right)^{-1}$. To obtain an equation for $\mathbf{Q}_{opt}$, we substitute (5.56) into (5.55). After some tedious algebra, we obtain

$$J = \sum_i (\mathbf{Q}\mathbf{r}_i)^T \mathbf{A}_i (\mathbf{Q}\mathbf{r}_i) - \left(\sum_i \mathbf{A}_i \mathbf{Q}\mathbf{r}_i \right)^T \left(\sum_i \mathbf{A}_i \right)^{-1} \left(\sum_i \mathbf{A}_i \mathbf{Q}\mathbf{r}_i \right) \tag{5.57}$$

which is a quadratic form in the elements of $\mathbf{Q}$. To find $\mathbf{Q}_{opt}$ then, one would minimize the quadratic form subject to $\mathbf{Q}$ being a rotation matrix (see Appendix E for an interesting duality approach to this sort of optimization).

(*iv*) In the most general two-dataset case, we have the estimate $\widehat{\mathbf{x}}_i$ from (5.22), and defining $\mathbf{A}_i = \mathbf{\Lambda}_{\mathbf{l}_i}^{-1}$ and $\mathbf{B}_i = \mathbf{Q}\mathbf{\Lambda}_{\mathbf{r}_i}^{-1}\mathbf{Q}^T$ we rewrite the estimate as

$$\widehat{\mathbf{x}}_i = (\mathbf{A}_i + \mathbf{B}_i)^{-1}\mathbf{A}_i\mathbf{l}_i + (\mathbf{A}_i + \mathbf{B}_i)^{-1}\mathbf{B}_i(\mathbf{Q}\mathbf{r}_i + \mathbf{t}). \tag{5.58}$$

We will rewrite this equation in several ways, using the matrix identities

$$\begin{aligned}
(\mathbf{A}+\mathbf{B})^{-1}\mathbf{A} &= \mathbf{I} - (\mathbf{A}+\mathbf{B})^{-1}\mathbf{B}, \\
(\mathbf{A}+\mathbf{B})^{-1}\mathbf{B} &= \mathbf{I} - (\mathbf{A}+\mathbf{B})^{-1}\mathbf{A}, \\
\mathbf{A}(\mathbf{A}+\mathbf{B})^{-1} &= \mathbf{I} - \mathbf{B}(\mathbf{A}+\mathbf{B})^{-1}, \\
\mathbf{B}(\mathbf{A}+\mathbf{B})^{-1} &= \mathbf{I} - \mathbf{A}(\mathbf{A}+\mathbf{B})^{-1}, \\
\mathbf{A}(\mathbf{A}+\mathbf{B})^{-1}\mathbf{B} &= \mathbf{B}(\mathbf{A}+\mathbf{B})^{-1}\mathbf{A},
\end{aligned} \tag{5.59}$$

where the last identity is proved using the second and fourth identities:

$$\begin{aligned}
\mathbf{A}(\mathbf{A}+\mathbf{B})^{-1}\mathbf{B} &= \mathbf{A}(\mathbf{I} - (\mathbf{A}+\mathbf{B})^{-1}\mathbf{A}), \\
&= \mathbf{A} - \mathbf{A}(\mathbf{A}+\mathbf{B})^{-1}\mathbf{A}, \\
&= (\mathbf{I} - \mathbf{A}(\mathbf{A}+\mathbf{B})^{-1})\mathbf{A}, \\
&= \mathbf{B}(\mathbf{A}+\mathbf{B})^{-1}\mathbf{A}.
\end{aligned} \tag{5.60}$$

Using the first identity in (5.59), (5.58) becomes

$$\begin{aligned}
\widehat{\mathbf{x}}_i &= \mathbf{l}_i - (\mathbf{A}_i + \mathbf{B}_i)^{-1}\mathbf{B}_i\mathbf{l}_i + (\mathbf{A}_i + \mathbf{B}_i)^{-1}\mathbf{B}_i(\mathbf{Q}\mathbf{r}_i + \mathbf{t}), \\
\widehat{\mathbf{x}}_i - \mathbf{l}_i &= (\mathbf{A}_i + \mathbf{B}_i)^{-1}\mathbf{B}_i(\mathbf{Q}\mathbf{r}_i + \mathbf{t} - \mathbf{l}_i).
\end{aligned} \tag{5.61}$$

Using the second identity in (5.59), (5.58) becomes

$$\begin{aligned}
\widehat{\mathbf{x}}_i &= (\mathbf{A}_i + \mathbf{B}_i)^{-1}\mathbf{A}_i\mathbf{l}_i(\mathbf{Q}\mathbf{r}_i + \mathbf{t}) - (\mathbf{A}_i + \mathbf{B}_i)^{-1}\mathbf{A}_i(\mathbf{Q}\mathbf{r}_i + \mathbf{t}), \\
\widehat{\mathbf{x}}_i - (\mathbf{Q}\mathbf{r}_i + \mathbf{t}) &= -(\mathbf{A}_i + \mathbf{B}_i)^{-1}\mathbf{A}_i(\mathbf{Q}\mathbf{r}_i + \mathbf{t} - \mathbf{l}_i).
\end{aligned} \tag{5.62}$$

We substitute the last lines of these last two equations into (5.46), obtaining

$$\begin{aligned}
J = \textstyle\sum_i \quad & (\mathbf{Q}\mathbf{r}_i + \mathbf{t} - \mathbf{l}_i)^T \left(\mathbf{B}_i(\mathbf{A}_i + \mathbf{B}_i)^{-1}\mathbf{A}_i(\mathbf{A}_i + \mathbf{B}_i)^{-1}\mathbf{B}_i \right. \\
& \left. + \mathbf{A}_i(\mathbf{A}_i + \mathbf{B}_i)^{-1}\mathbf{B}_i(\mathbf{A}_i + \mathbf{B}_i)^{-1}\mathbf{A}_i\right)(\mathbf{Q}\mathbf{r}_i + \mathbf{t} - \mathbf{l}_i),
\end{aligned} \tag{5.63}$$

where we have made use of the fact that all the matrices are symmetric. We may greatly simplify the matrix expression in (5.63) using the last identity of (5.59)

$$\begin{aligned}
& \mathbf{B}_i(\mathbf{A}_i + \mathbf{B}_i)^{-1}\mathbf{A}_i(\mathbf{A}_i + \mathbf{B}_i)^{-1}\mathbf{B}_i \\
+ \quad & \mathbf{A}_i(\mathbf{A}_i + \mathbf{B}_i)^{-1}\mathbf{B}_i(\mathbf{A}_i + \mathbf{B}_i)^{-1}\mathbf{A}_i, \\
= \quad & \mathbf{A}_i(\mathbf{A}_i + \mathbf{B}_i)^{-1}\mathbf{B}_i(\mathbf{A}_i + \mathbf{B}_i)^{-1}\mathbf{B}_i \\
+ \quad & \mathbf{A}_i(\mathbf{A}_i + \mathbf{B}_i)^{-1}\mathbf{B}_i(\mathbf{A}_i + \mathbf{B}_i)^{-1}\mathbf{A}_i, \\
= \quad & \mathbf{A}_i(\mathbf{A}_i + \mathbf{B}_i)^{-1}\mathbf{B}_i(\mathbf{A}_i + \mathbf{B}_i)^{-1}(\mathbf{A}_i + \mathbf{B}_i), \\
= \quad & \mathbf{A}_i(\mathbf{A}_i + \mathbf{B}_i)^{-1}\mathbf{B}_i.
\end{aligned} \tag{5.64}$$

We thus obtain

$$J = \sum_i (\mathbf{Q}\mathbf{r}_i + \mathbf{t} - \mathbf{l}_i)^T \mathbf{A}_i (\mathbf{A}_i + \mathbf{B}_i)^{-1} \mathbf{B}_i (\mathbf{Q}\mathbf{r}_i + \mathbf{t} - \mathbf{l}_i). \tag{5.65}$$

This shows that the optimal rotation and translation can be obtained as the minimizers of the sum of the weighted fitting errors, $\mathbf{Q}\mathbf{r}_i + \mathbf{t} - \mathbf{l}_i$. If the $\mathbf{A}_i$ and $\mathbf{B}_i$ are invertible, then we may rewrite (5.65) in a more familiar form using the identity

$$\Big(\mathbf{A}(\mathbf{A}+\mathbf{B})^{-1}\mathbf{B}\Big)^{-1} = \mathbf{B}^{-1}(\mathbf{A}+\mathbf{B})\mathbf{A}^{-1} = \mathbf{B}^{-1} + \mathbf{A}^{-1}, \tag{5.66}$$

so

$$\mathbf{A}(\mathbf{A}+\mathbf{B})^{-1}\mathbf{B} = (\mathbf{A}^{-1} + \mathbf{B}^{-1})^{-1}. \tag{5.67}$$

We thus have

$$\begin{aligned} J &= \textstyle\sum_i (\mathbf{Q}\mathbf{r}_i + \mathbf{t} - \mathbf{l}_i)^T (\mathbf{A}_i^{-1} + \mathbf{B}_i^{-1})^{-1} (\mathbf{Q}\mathbf{r}_i + \mathbf{t} - \mathbf{l}_i), \\ &= \textstyle\sum_i (\mathbf{Q}\mathbf{r}_i + \mathbf{t} - \mathbf{l}_i)^T (\mathbf{\Lambda}_{\mathbf{l}_i} + \mathbf{Q}\mathbf{\Lambda}_{\mathbf{r}_i}\mathbf{Q}^T)^{-1} (\mathbf{Q}\mathbf{r}_i + \mathbf{t} - \mathbf{l}_i). \end{aligned} \tag{5.68}$$

We see then that the optimal weighting matrix for the fitting error $\mathbf{Q}\mathbf{r}_i + \mathbf{t} - \mathbf{l}_i$ is the inverse covariance matrix of that fitting error, a rather satisfying result. Hill, Cootes, and Taylor [87] allude to this result, but mainly concentrate on using a fixed weighting matrix for their nonscalar absolute orientation problem. We note that this expression is only valid for invertible $\mathbf{\Lambda}_{\mathbf{r}_i}$ and $\mathbf{\Lambda}_{\mathbf{l}_i}$, and therefore is not suitable for image data (which limits the applicability of the work of Hill *et al.*). In contrast, our more general form in (5.65) works well with image data.

We can of course obtain the translation estimate in closed-form from the rotation estimate by differentiating (5.65) to obtain

$$\mathbf{t}_{opt} = -\left(\sum_i \mathbf{A}_i (\mathbf{A}_i + \mathbf{B}_i)^{-1} \mathbf{B}_i\right)^{-1} \left(\sum_i \mathbf{A}_i (\mathbf{A}_i + \mathbf{B}_i)^{-1} \mathbf{B}_i (\mathbf{Q}_{opt}\mathbf{r}_i - \mathbf{l}_i)\right), \tag{5.69}$$

with $\mathbf{A}_i = \mathbf{\Lambda}_{\mathbf{l}_i}^{-1}$ and $\mathbf{B}_i = \mathbf{Q}_{opt}\mathbf{\Lambda}_{\mathbf{r}_i}^{-1}\mathbf{Q}_{opt}^T$

(v) If we have more than two datasets, it is clear that the ML criterion (5.45) generalizes to handle this situation. Now we have to optimize over several rotations and translations; in practice, this can be a difficult computational problem, with the attendant initialization and convergence issues that accompany any multidimensional nonlinear optimization problem. Critical to the success of this method then is the availability of a good starting approximation for the motion parameters; this issue is addressed by a new, efficient algorithm in the next section for the case where the datasets are all camera images.

5.5 A Linear Epipolar Algorithm for Multiframe Orientation

In this section, we attempt recovery of the orientation of M calibrated cameras frames by first determining the epipoles. This will allow us to decouple the camera rotations and translations, allowing for an efficient, linear solution. Moreover, the use of the SVD (or perhaps EVD) will allow a noniterative solution. This is in contrast to various iterative approaches, which must estimate and reorthogonalize a rotation matrix at each iteration. The general strategy for this approach is as follows:

- For each image pair, calculate the Essential matrix and the corresponding epipoles.
- For each image triple, calculate the angle between the baselines, and use the law of sines to determine the relative baseline lengths.
- Use the baseline lengths and the baseline angles to constrain the dot products of the baseline vectors. Solve the dot-product problem for the centers of projection (COP).
- Solve for the camera rotations using the calculated baselines.

For each of the above steps, we use the SVD as the main computational algorithm. The following sections provide more detail on the required computations.

5.5.1 Calculation of Epipoles

Recall from Section 2.2 that an epipole is the image in one camera of the COP of the other camera, as shown in Figure 2-2. With the exception of the epipole, only one epipolar line goes through any image point. Also, all epipolar lines go through the camera's epipole.

Also recall from Section 2.2 that without measurement noise, each pair of corresponding image points $\mathbf{p}_l = [x_l \ y_l \ 1]^T$ and $\mathbf{p}_r = [x_r \ y_r \ 1]^T$ satisfy the constraint

$$\mathbf{p}_l^T \mathbf{E} \mathbf{p}_r = 0, \tag{5.70}$$

where $\mathbf{E}$ is the essential matrix. If we substitute the right epipole $\mathbf{e}_r$ for $\mathbf{p}_r$ into (5.70) we have

$$\mathbf{p}_l^T \mathbf{E} \mathbf{e}_r = 0, \tag{5.71}$$

for *any* $\mathbf{p}_l$. This is because all epipolar lines goes through the epipole. For (5.71) to hold for any left image point, it must be true that $\mathbf{e}_r$ lies in the right null space of $\mathbf{E}$. Similar reasoning shows that the left epipole $\mathbf{e}_l$ lies in the left null space of $\mathbf{E}$. Recall that without noise, the true essential matrix is determined by $\mathbf{E} = \mathbf{BR}$, where $\mathbf{B}$ is the skew-symmetric representation of the cross product with the baseline translation $\mathbf{t}$, and $\mathbf{R}$ is the rotation matrix. From this, we see that ideally $\mathbf{E}$ is rank-2. Thus, the left and right epipoles are determined from the left and right singular vectors corresponding to the only singular value of $\mathbf{E}$ that equals zero. In practice, since we form $\mathbf{E}$ numerically, we chose the singular vectors corresponding to the minimum singular value.

To find $\mathbf{E}$, we use the linear normalized 8-point algorithm [78] for each pair of images. The entire calculation for the epipoles is summarized as:

1. Normalize the feature points by forming $\tilde{\mathbf{p}}_{il} = \mathbf{p}_{il}/\|\mathbf{p}_{il}\|$ and similarly $\tilde{\mathbf{p}}_{ir} = \mathbf{p}_{ir}/\|\mathbf{p}_{ir}\|$.

2. Form the matrix

$$\mathbf{A} = \begin{bmatrix} \mathbf{a}_1^T \\ \vdots \\ \mathbf{a}_N^T \end{bmatrix}, \tag{5.72}$$

with $\mathbf{a}_i = (\tilde{\mathbf{p}}_{il} \otimes \tilde{\mathbf{p}}_{ir})$ and "$\otimes$" is the Kronecker product.

3. Calculate the SVD $\mathbf{A} = \mathbf{USV}^T$, and let $\mathbf{v} = [v_1, \ \ldots, \ v_9]^T$ be the left singular vector corresponding to the minimum singular value.

4. Form $\mathbf{E}$ as

$$\mathbf{E} = \begin{bmatrix} v_1 & v_2 & v_3 \\ v_4 & v_5 & v_6 \\ v_7 & v_8 & v_9 \end{bmatrix}. \tag{5.73}$$

5. Calculate the SVD $\mathbf{E} = \mathbf{USV}^T$, set the left epipole $\mathbf{e}_l$ equal to $\mathbf{U}(:, 3)$, and set the right epipole $\mathbf{e}_r$ equal to $\mathbf{V}(:, 3)$. The epipoles are each determined only up a sign ambiguity, which we correctly resolve in Section 5.5.2.

We also note that other options are available for epipole calculation. For example, we could use image triplets and form the trilinear tensor. In this case, Hartley [79] shows that the epipoles lie in the left and right null spaces of the various two-dimensional slices of the the trilinear tensor. Of course, all the various nonlinear two-frame approaches [90], [91] may be used.

Error Ellipse Analysis

Error ellipses can be used to understand the behavior of the 8-point and normalized 8-point algorithm in estimating the epipoles. When we estimate the essential parameters using TLS on $\mathbf{A}$ in (5.72), we are assuming the noise in each element of the matrix is zero-mean IID white Gaussian noise (i.e. see (4.10)–(4.14) and the associated text for this proof). Unfortunately, because of the way $\mathbf{A}$ is formed, the perturbations are not independent at all – there is some correlation. This skews the results obtained by the 8-point algorithm.

To determine the deviation from the TLS assumptions, we assume that perturbations in different rows of $\mathbf{A}$ are uncorrelated, and that the covariance matrix for perturbations in row i are equal to 9×9 covariance matrix $\mathbf{\Lambda}_i$. Let the true normalized image feature point locations be given by $\tilde{\mathbf{p}}_{ilt}$ and $\tilde{\mathbf{p}}_{irt}$, and let the observations be given by

$$\begin{aligned} \tilde{\mathbf{p}}_{il} &= \tilde{\mathbf{p}}_{ilt} + \mathbf{c}_i \\ \tilde{\mathbf{p}}_{ir} &= \tilde{\mathbf{p}}_{irt} + \mathbf{d}_i, \end{aligned} \tag{5.74}$$

where $\mathbf{c}_i$ and $\mathbf{d}_i$ are independent zero-mean Gaussian perturbations with covariance matrices $\mathbf{\Lambda}_{ci}$ and $\mathbf{\Lambda}_{di}$. We develop our result for general $\mathbf{\Lambda}_{ci}$ and $\mathbf{\Lambda}_{di}$, and then specialize them for both the unnormalized and normalized 8-point algorithm.

We have that

$$\begin{aligned} E\{\mathbf{a}_i\} &= E\{\tilde{\mathbf{p}}_{il} \otimes \tilde{\mathbf{p}}_{ir}\} = E\{(\tilde{\mathbf{p}}_{ilt} + \mathbf{c}_i) \otimes (\tilde{\mathbf{p}}_{irt} + \mathbf{d}_i)\} \\ &= E\{\tilde{\mathbf{p}}_{ilt} \otimes \tilde{\mathbf{p}}_{irt}\} + E\{\tilde{\mathbf{p}}_{ilt} \otimes \mathbf{d}_i\} + E\{\mathbf{c}_i \otimes \tilde{\mathbf{p}}_{irt}\} + E\{\mathbf{c}_i \otimes \mathbf{d}_i\} \\ &= \tilde{\mathbf{p}}_{ilt} \otimes \tilde{\mathbf{p}}_{irt}, \end{aligned} \tag{5.75}$$

because the last three expectations are all zero. Thus (5.75) shows that each row of $\mathbf{A}$ is unbiased. Now we calculate the covariance of each row:

$$\mathbf{\Lambda}_i = E\{(\tilde{\mathbf{p}}_{il} \otimes \tilde{\mathbf{p}}_{ir})(\tilde{\mathbf{p}}_{il} \otimes \tilde{\mathbf{p}}_{ir})^T\} - (\tilde{\mathbf{p}}_{ilt} \otimes \tilde{\mathbf{p}}_{irt})(\tilde{\mathbf{p}}_{ilt} \otimes \tilde{\mathbf{p}}_{irt})^T, \tag{5.76}$$

and

$$\begin{aligned}
&E\{(\tilde{\mathbf{p}}_{il} \otimes \tilde{\mathbf{p}}_{ir})(\tilde{\mathbf{p}}_{il} \otimes \tilde{\mathbf{p}}_{ir})^T\} \\
&\quad = E\{\tilde{\mathbf{p}}_{il}\tilde{\mathbf{p}}_{il}^T \otimes \tilde{\mathbf{p}}_{ir}\tilde{\mathbf{p}}_{ir}^T\} \\
&\quad = E\{(\tilde{\mathbf{p}}_{ilt} + \mathbf{c}_i)(\tilde{\mathbf{p}}_{ilt} + \mathbf{c}_i)^T \otimes (\tilde{\mathbf{p}}_{irt} + \mathbf{d}_i)(\tilde{\mathbf{p}}_{irt} + \mathbf{d}_i)^T\} \\
&\quad = E\{(\tilde{\mathbf{p}}_{ilt}\tilde{\mathbf{p}}_{ilt}^T + \mathbf{c}_i\tilde{\mathbf{p}}_{ilt}^T + \tilde{\mathbf{p}}_{ilt}\mathbf{c}_i^T + \mathbf{c}_i\mathbf{c}_i^T) \\
&\qquad \otimes(\tilde{\mathbf{p}}_{irt}\tilde{\mathbf{p}}_{irt}^T + \mathbf{d}_i\tilde{\mathbf{p}}_{irt}^T + \tilde{\mathbf{p}}_{irt}\mathbf{d}_i^T + \mathbf{d}_i\mathbf{d}_i^T)\} \\
&\quad = E\{\tilde{\mathbf{p}}_{ilt}\tilde{\mathbf{p}}_{ilt}^T \otimes \tilde{\mathbf{p}}_{irt}\tilde{\mathbf{p}}_{irt}^T + \tilde{\mathbf{p}}_{ilt}\tilde{\mathbf{p}}_{ilt}^T \otimes \mathbf{d}_i\mathbf{d}_i^T \\
&\qquad +\mathbf{c}_i\mathbf{c}_i^T \otimes \tilde{\mathbf{p}}_{irt}\tilde{\mathbf{p}}_{irt}^T + \mathbf{c}_i\mathbf{c}_i^T \otimes \mathbf{d}_i\mathbf{d}_i^T\} \\
&\quad = \left(\tilde{\mathbf{p}}_{ilt}\tilde{\mathbf{p}}_{ilt}^T + \mathbf{\Lambda}_{ci}\right) \otimes \left(\tilde{\mathbf{p}}_{irt}\tilde{\mathbf{p}}_{irt}^T + \mathbf{\Lambda}_{di}\right)
\end{aligned} \tag{5.77}$$

We thus have that

$$\mathbf{\Lambda}_i = \tilde{\mathbf{p}}_{ilt}\tilde{\mathbf{p}}_{ilt}^T \otimes \mathbf{\Lambda}_{di} + \mathbf{\Lambda}_{ci} \otimes \tilde{\mathbf{p}}_{irt}\tilde{\mathbf{p}}_{irt}^T + \mathbf{\Lambda}_{ci} \otimes \mathbf{\Lambda}_{di}, \tag{5.78}$$

which shows that the covariance for each row depends on the actual values of the data and the matrices $\mathbf{\Lambda}_{ci}$ and $\mathbf{\Lambda}_{di}$. We again note that TLS would assume that $\mathbf{\Lambda}_i = \sigma^2\mathbf{I}$; the amount that $\mathbf{\Lambda}_i$ differs from $\sigma^2\mathbf{I}$ is an indication of the error that will result from the 8-point algorithm.

We now discuss the usual values of $\mathbf{\Lambda}_{ci}$ and $\mathbf{\Lambda}_{di}$. For the unnormalized 8-point algorithm it is easy to see that we replace the various normalized $\tilde{\mathbf{p}}_{(\cdot)}$ quantities in the above development with the unnormalized $\mathbf{p}_{(\cdot)}$. In this situation the only error is in the first two coordinates of these three-element vectors, so we have

$$\mathbf{\Lambda}_{ci} = \mathbf{\Lambda}_{di} = \sigma^2 \begin{bmatrix} 1 & & \\ & 1 & \\ & & 0 \end{bmatrix}, \; i = 1, \ldots, M. \tag{5.79}$$

For the normalized 8-point algorithm, where we calculate the various $\tilde{\mathbf{p}}_{(\cdot)}$ quantities by scaling the $\mathbf{p}_{(\cdot)}$ quantities so that they have unit length, the covariance matrices $\mathbf{\Lambda}_{ci}$ and $\mathbf{\Lambda}_{di}$ are data-dependant and are thus a little more complicated. It turns out that the form is exactly identical to the information form shown in (5.30). For example, if $\mathbf{p}_{li} = [u_i \; v_i \; 1]^T$ then

$$\mathbf{\Lambda}_{ci} = \frac{\sigma^2}{u_i^2 + v_i^2} \begin{bmatrix} -v_i & -u_i \\ u_i & -v_i \\ 0 & u_i^2 + v_i^2 \end{bmatrix} \begin{bmatrix} -v_i & u_i & 0 \\ -u_i & -v_i & u_i^2 + v_i^2 \end{bmatrix}, \tag{5.80}$$

This can be seen by considering Figure 5-4; this is the same figure we would draw to determine the covariance matrix of projecting an image plane point (u_i, v_i) onto a sphere. An analogous situation is the correspondence between the unnormalized covariance matrices in (5.79) and the orthographic projection information matrix (5.28).

With the formulas (5.79) and (5.80) for the covariance matrices $\mathbf{\Lambda}_i$, we are in a position to illustrate the deviation of these matrices from a scaled identity matrix. Essentially, for each formula, $\mathbf{\Lambda}_i$ is low-rank, and the remaining nonzero eigenvalues are not all equal to each other. This is easy to see because $\mathbf{\Lambda}_{ci}$ and $\mathbf{\Lambda}_{di}$ are rank-2 for both the unnormalized and normalized 8-point algorithm. Using the fact that the rank of a Kronecker product of two matrices is the product of the ranks, in (5.78) we see that $\mathbf{\Lambda}_i$ is the sum of two rank-2

matrices and one rank-4 matrix. Thus, $\mathbf{\Lambda}_i$ is at most rank-8, a rather extreme deviation from a scaled identity matrix. Compounding the situation, the null axes of the $\mathbf{\Lambda}_i$ error ellipses depend on the data; there is no way that a simple transformation or change of variables can align all of them (thereby precluding the use of a weighted TLS approach to removing the null axes and reduce the problem's dimensionality).

5.5.2 Resolving the Epipole Sign Ambiguity

The essential matrix yields the direction of the epipoles, but not whether they point to the front or the back of the cameras. Knowing the signs of the epipoles is important in correctly formulating the dot-product constraints of Section 5.5.5.

In some situations the epipole signs are trivially determined. For example, when images are acquired from a single moving camera, which is generally facing in the forward direction of motion, then the epipoles from earlier image frames to later image frames have positive z-element, and the epipoles from later image frames to earlier image frames have negative z-element. Another situation in which the epipole signs may be easily determined occurs when we are tracking multiple cameras over time. Assuming we have correctly calculated the epipole signs at one time instant, then epipole signs at the next time instant should be similar.

If we cannot assume that one of the above situations holds true, then we must explicitly solve for the correct epipole signs. Recall from Section 2.2.1 that when we solve the relative orientation problem,

$$\mathbf{P}_r = \mathbf{R}\mathbf{P}_l + \mathbf{t} \tag{5.81}$$

we obtain two possible rotation matrices $\mathbf{R}$ (i.e. (2.16) and (2.17)), and two possible translations $\mathbf{t}$ (ignoring the unrecoverable scale factor, we have a sign ambiguity for $\mathbf{t}$). Only one of the four possible combinations of $\mathbf{R}$ and $\mathbf{t}$ will yield positive depth values for all the reconstructed feature points in three-dimensions (assuming reasonable SNR). We therefore carry out this reconstruction, and find the correct rotation and translation sign.

We can save a little bit of work on this step by determining only the depth z_r and z_l of the points $\mathbf{P}_r$ and $\mathbf{P}_l$, for a postulated $\mathbf{R}$, $\mathbf{t}$ pairing. Factoring out the depths from (5.81),

$$z_r\mathbf{p}_r = z_l\mathbf{R}\mathbf{p}_l + \mathbf{t}, \tag{5.82}$$

or

$$\begin{bmatrix} -\mathbf{R}\mathbf{p}_l & \mathbf{p}_r \end{bmatrix} \begin{bmatrix} z_l \\ z_r \end{bmatrix} = \mathbf{t}. \tag{5.83}$$

Since this is overdetermined and only approximate, we solve for the depths using OLS:

$$\begin{bmatrix} z_l \\ z_r \end{bmatrix} = \frac{\begin{bmatrix} \mathbf{p}_r^T\mathbf{p}_r & \mathbf{p}_r^T\mathbf{R}\mathbf{p}_l \\ \mathbf{p}_r^T\mathbf{R}\mathbf{p}_l & \mathbf{p}_l^T\mathbf{p}_l \end{bmatrix} \begin{bmatrix} -\mathbf{t}^T\mathbf{R}\mathbf{p}_l \\ \mathbf{t}^T\mathbf{p}_r \end{bmatrix}}{(\mathbf{p}_r^T\mathbf{p}_r)(\mathbf{p}_l^T\mathbf{p}_l) - (\mathbf{p}_r^T\mathbf{R}\mathbf{p}_l)^2}. \tag{5.84}$$

By the Cauchy-Schwartz inequality the denominator of (5.84) is nonnegative, so we may simply ignore it for the purposes of determining the depth signs.

The translation sign is the critical component for determining the epipole signs. If the third component t_3 of $\mathbf{t}$ is positive, then the COP for the left camera is in front of the right camera, and the corresponding right-to-left camera epipole sign is positive; if $t_3 < 0$, the epipole sign is negative. By rearranging (5.81), we have that similar comments apply to the sign of the third component of $-\mathbf{R}^T\mathbf{t}$ and the left-to-right camera epipole sign.

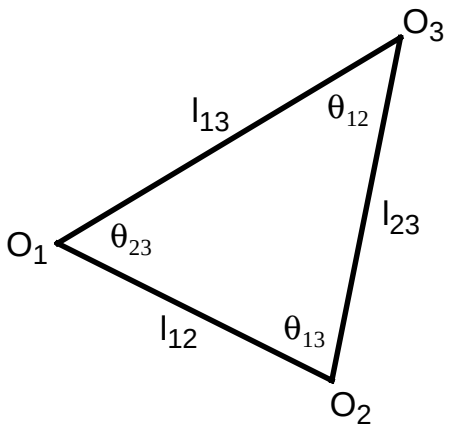

Figure 5-8: Geometry for law of sines.

5.5.3 Determining Baseline Angles and Lengths

From the previous section, we now have the epipole locations for each pair of images. If we consider three images, then their COPs form a triangle. We can recover the shape of this triangle from the corresponding epipole locations.

Let the COPs of the images be denoted by the points O_1, O_2, and O_3. Let the length of the baseline translation connecting O_1 and O_2 be denoted as l_{12}, and similarly define l_{13} and l_{23}. Also, let the angle opposite l_{12} be be denoted by θ_{12}, with θ_{13}, θ_{23} similarly defined. Figure 5-8 shows this geometry. With these definitions, the *law of sines* [88] states that

$$\frac{\sin\theta_{12}}{l_{12}} = \frac{\sin\theta_{13}}{l_{13}} = \frac{\sin\theta_{23}}{l_{23}}. \tag{5.85}$$

This is easily rearranged into matrix-vector form as

$$\begin{bmatrix} 0 & \sin\theta_{23} & -\sin\theta_{13} \\ -\sin\theta_{23} & 0 & \sin\theta_{12} \\ \sin\theta_{13} & -\sin\theta_{12} & 0 \end{bmatrix} \begin{bmatrix} l_{12} \\ l_{13} \\ l_{23} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{5.86}$$

The sines in (5.86) are easily determined from the epipoles. For example, if $\mathbf{e}_2$ and $\mathbf{e}_3$ denote the (unit-length) epipoles of O_2 and O_3 respectively in image 1, then $\cos\theta_{23} = \mathbf{e}_2^T\mathbf{e}_3$, and therefore $\sin\theta_{23} = \sqrt{1-(\mathbf{e}_2^T\mathbf{e}_3)^2}$. (Note that since $0 \le \theta_{ij} \le \pi$, the signs of the sine functions must all be positive.)

Because of noise, we do not solve (5.86) for each image triple individually. Rather, we solve for all the lengths l_{ij} simultaneously, by forming the l_{ij} into a large $M(M-1)/2 \times 1$ vector $\mathbf{l}$ and filling in a sparse $3M(M-1)(M-2)/2 \times M(M-1)/2$ matrix $\mathbf{A}$ with the appropriate sine information from each matrix of the form (5.86). We then get the best estimate of $\mathbf{l}$ from the SVD $\mathbf{A} = \mathbf{USV}^T$ by setting $\mathbf{l}$ equal to the column of $\mathbf{V}$ corresponding to the minimum singular value.

5.5.4 Forming the Dot-Product Constraints

We now have the baseline lengths, determined up to an unknown scale factor. With these baseline lengths, and the cosines of the angles between the baselines, we can write equations

for the COPs (the O_i's). Let each COP O_i be represented algebraically by a 3-dimensional vector $\mathbf{o}_i$. For the first image triple, we have

$$\frac{\mathbf{o}_2 - \mathbf{o}_1}{l_{12}} \cdot \frac{\mathbf{o}_3 - \mathbf{o}_1}{l_{13}} = \cos\theta_{23} = \mathbf{e}_2^T \mathbf{e}_3, \tag{5.87}$$

or

$$(\mathbf{o}_2 - \mathbf{o}_1)^T (\mathbf{o}_3 - \mathbf{o}_1) = l_{12} l_{13} \mathbf{e}_2^T \mathbf{e}_3, \tag{5.88}$$

or finally

$$\begin{bmatrix} 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{o}_1^T \mathbf{o}_1 \\ \mathbf{o}_1^T \mathbf{o}_2 \\ \mathbf{o}_1^T \mathbf{o}_3 \\ \mathbf{o}_2^T \mathbf{o}_3 \end{bmatrix} = l_{12} l_{13} \mathbf{e}_2^T \mathbf{e}_3. \tag{5.89}$$

By considering all other image triples, we obtain a set of similar equations, with the right hand sides known from our previous calculations, and the unknown $\mathbf{o}_i's$ combined in dot products on the left hand sides. If we stack all these equations together into a large $M(M-1)(M-2)/2 \times M(M+1)/2$ matrix $\mathbf{A}$, and attempt to solve for all the dot products using ordinary least squares, we find that the matrix is rank-deficient. This is because there are some unrecoverable degrees of freedom in the coordinate frame of the $\mathbf{o}_i$'s relative to the world coordinate frame. We are therefore free to assign one of the COPs, say $\mathbf{o}_0$, to the origin $[0\ 0\ 0]^T$ of the new coordinate frame. This assignment means that $\mathbf{o}_0^T \mathbf{o}_i = 0$, for all i. By removing the corresponding columns in $\mathbf{A}$, we will obtain a sparse, full-rank $M(M-1)(M-2)/2 \times M(M-1)/2$ matrix, allowing us to solve for all the remaining dot products using ordinary least squares. The reduced matrix (or, perhaps better, its pseudoinverse) may be precalculated, allowing us to greatly reduce the real-time computational load.

5.5.5 Solving the Dot-Product Constraints

We now have a set of constraints $\mathbf{o}_i^T \mathbf{o}_j = a_{ij}$ with a_{ij} known for all possible combinations of $1 \le i, j \le N = M - 1$. (Actually, we have $a_{0j} = a_{i0} = 0$, but we remove the zeroeth camera from the problem since it was mapped to the origin.)

We are thus given all $N(N-1)/2$ possible dot-products from N 3-dimensional points, and our goal is to find the N points. Define the $3 \times N$ matrix of unknowns

$$\mathbf{X} = \begin{bmatrix} \mathbf{o}_1 & \cdots & \mathbf{o}_N \end{bmatrix}, \tag{5.90}$$

and the matrix of dot-products (which is also called a "Gram matrix")

$$\mathbf{A} = \begin{bmatrix} a_{1,1} & \cdots & a_{1,N} \\ \vdots & & \vdots \\ a_{N,1} & \cdots & a_{N,N} \end{bmatrix}. \tag{5.91}$$

Note that because $\mathbf{A}$ is symmetric, we are guaranteed that it has real eigenvalues and real, orthonormal eigenvectors [174].

We are thus attempting to find the rank-3 matrix $\mathbf{X}$ such that $\mathbf{X}^T \mathbf{X} \approx \mathbf{A}$. A straightforward procedure is to choose the best low-rank approximation to the matrix square root of $\mathbf{A}$. That is, first decompose $\mathbf{A}$ into its eigenvalue/vector expansion,

$$\mathbf{A} = \mathbf{V}\mathbf{D}\mathbf{V}^T, \tag{5.92}$$

where $\mathbf{V}$ is orthonormal and $\mathbf{D}$ is diagonal and is partitioned as

$$\mathbf{D} = \begin{bmatrix} \mathbf{D}_3 & \mathbf{0} \\ \mathbf{0} & \mathbf{D}_{N-3} \end{bmatrix}, \tag{5.93}$$

and $\mathbf{D}_3$ is of size 3×3. Then, set the estimate $\widehat{\mathbf{X}}$ of the true $\mathbf{X}$ to be

$$\widehat{\mathbf{X}} = \begin{bmatrix} \sqrt{\mathbf{D}_3} & \mathbf{0} \end{bmatrix} \mathbf{V}^T. \tag{5.94}$$

The solution (5.94) is not unique. In fact, pre-multiplication of $\widehat{\mathbf{X}}$ by any 3×3 orthonormal matrix $\mathbf{Q}$ results in a valid solution because

$$(\mathbf{Q}\widehat{\mathbf{X}})^T(\mathbf{Q}\widehat{\mathbf{X}}) = \widehat{\mathbf{X}}^T\mathbf{Q}^T\mathbf{Q}\widehat{\mathbf{X}} = \widehat{\mathbf{X}}^T\widehat{\mathbf{X}}. \tag{5.95}$$

Therefore, we can determine $\widehat{\mathbf{X}}$ only up to an arbitrary rotation and flip.

This solution minimizes

$$\|\widehat{\mathbf{X}}^T\widehat{\mathbf{X}} - \mathbf{A}\|_F^2, \tag{5.96}$$

where $\|\cdot\|_F^2$ is the Frobenius norm. The error norm in (5.96) may be rewritten in terms of the error in approximating the data $a_{i,j}$. Let $\widehat{\mathbf{A}} = \widehat{\mathbf{X}}^T\widehat{\mathbf{X}}$ be the matrix of reconstructed dot-products, with $\widehat{a}_{i,j} = [\widehat{\mathbf{A}}]_{i,j}$. We have

$$\begin{aligned} \|\widehat{\mathbf{X}}^T\widehat{\mathbf{X}} - \mathbf{A}\|_F^2 &= \|\widehat{\mathbf{A}} - \mathbf{A}\|_F^2 \\ &= \textstyle\sum_{i=1}^{N}\sum_{j=1}^{N}(\widehat{a}_{i,j} - a_{i,j})^2 \\ &= 2\textstyle\sum_{i=1}^{N-1}\sum_{j=i+1}^{N}(\widehat{a}_{i,j} - a_{i,j})^2 + \sum_{i=1}^{N}(\widehat{a}_{i,i} - a_{i,i})^2. \end{aligned} \tag{5.97}$$

Clearly, this solution minimizes a weighted sum of the squared reconstruction errors. The squared errors in the dot-product reconstructions, $(\widehat{a}_{i,j} - a_{i,j})^2$, for $i \neq j$, are weighted twice as much as the squared error in the reconstruction of the magnitude-squared of the points. This means that the method will not try as hard to maintain the measured magnitude-squared ($a_{i,i}$'s) of the points as it would for a least-squares solution.

5.5.6 Solving for Camera Attitudes

From (5.94) we thus have the solution for the COPs $\mathbf{o}_i$, up to an arbitrary scaling, rotation and possibly a flip. In many applications, this would perhaps be sufficient. However, if we desire the camera attitudes in the new coordinate frame, we can obtain this information from the $\mathbf{o}_i$.

We of course could perform the relative orientation operation between all pairs of images, and somehow enforce consistency among all the pairwise rotations. However, since we already have calculated good camera COPs, we desire to use this global information to our advantage.

By performing the subtraction $\mathbf{t} = \mathbf{o}_i - \mathbf{o}_j$, we obtain the translation from camera j and camera i. If we properly determine the camera attitudes in the new coordinate frame, then the epipolar direction from camera j to camera i will align with $\mathbf{t}$. For each camera j, we therefore seek to align all epipoles to camera $i \neq j$ with the difference $\mathbf{o}_i - \mathbf{o}_j$ (normalized to unit length). This is exactly the Orthogonal Procrustes problem, which we solve efficiently using the SVD. This yields the attitude of camera j in the new coordinate

frame. Incidently, the determinant of the orthogonal matrix produced by the Orthogonal Procrustes procedure indicates if we have chosen the flip reconstruction in Section 5.5.5. In the noiseless case, the determinants of all the camera attitude matrices would be equal to -1 if the flip reconstruction had been chosen. If this situation occurs, a simply remedy is to negate the third components of the $\mathbf{o}_i$'s before recovering the camera attitude matrices.

5.5.7 Experiment

A MATLAB simulation was conducted to check the veracity of the approach given in the previous sections, and to get an indication of the required accuracy for the image feature point locations. Figure 5-9 shows a typical setup. There were 10 cameras and 50 feature points, all of which were visible to all of the cameras (this is not a requirement, however). The points were projected into the cameras' image planes, and a small amount of noise was added to the image feature locations. In this example, the noise corresponded to a little over 3 pixels of error, assuming 1000×1000 pixel images. Figure 5-10 shows a typical image.

The lengths between the camera centers were recovered (up to an unknown scale factor) using the approach given in Section 5.5.3. Also note that the pseudo "condition" number (the ratio of the maximum singular value to the second smallest singular value) of the sine matrix used to determine the lengths for this scene was approximately 2. (This was generally true for many different trials). As with the linear exterior orientation algorithm of Section 2.6.2, the extremely low condition number indicates that this approach will not be sensitive (at least at high SNR) to feature location noise. Figure 5-11 shows the true length versus the computed length for all l_{ij} pairs. Note that even with the large amount of image location noise, the scatter plot is nearly linear, indicating good length recovery.

Next, the camera COPs were recovered using the dot-product solution of Section 5.5.5, and the camera attitudes recovered using the algorithm in Section 5.5.6. Figure 5-12 shows the results of this operation. To generate the figure, the arbitrary rotation and scaling between the original scene and our reconstruction was removed using the absolute orientation algorithm. This was done only for convenience in comparing to the true scene; in reality, this transformation cannot be accomplished without auxillary information.

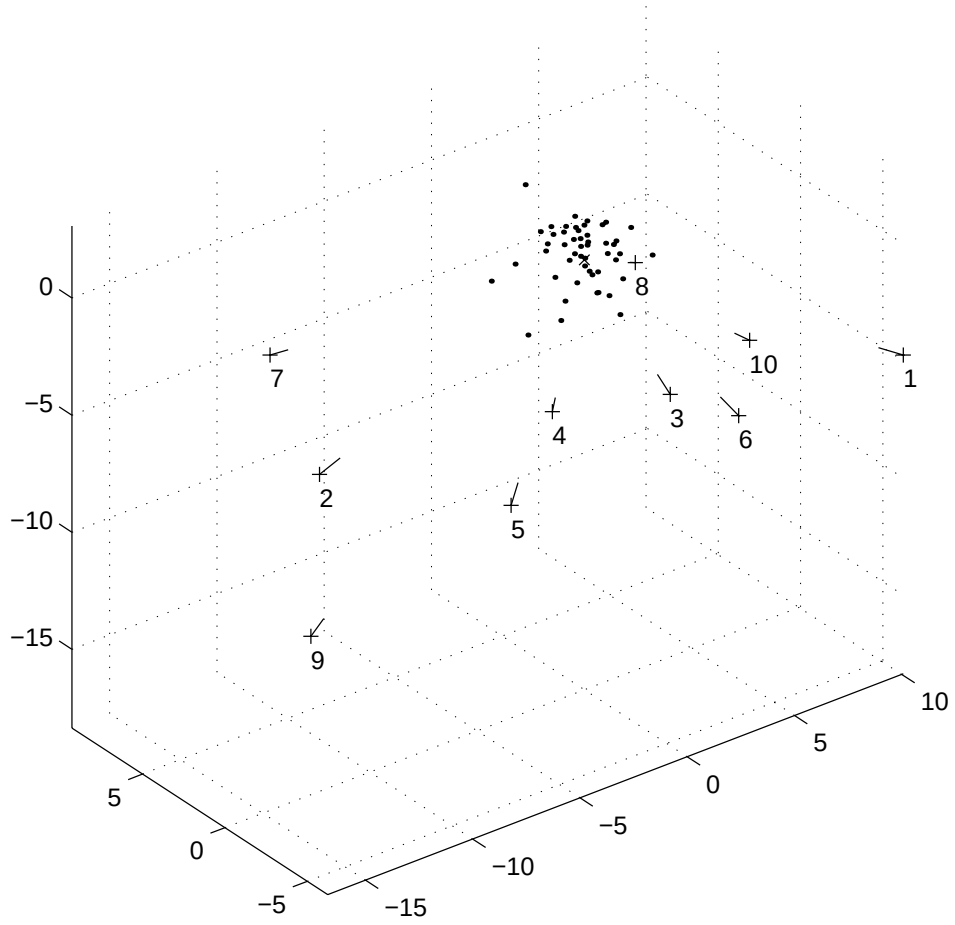

Figure 5-9: Multicamera scene. The "+" indicate the camera center of projection. The lines emanating from the +'s is the principal image axis. The dots are the feature points (50 in this example). The × is the origin of the space.

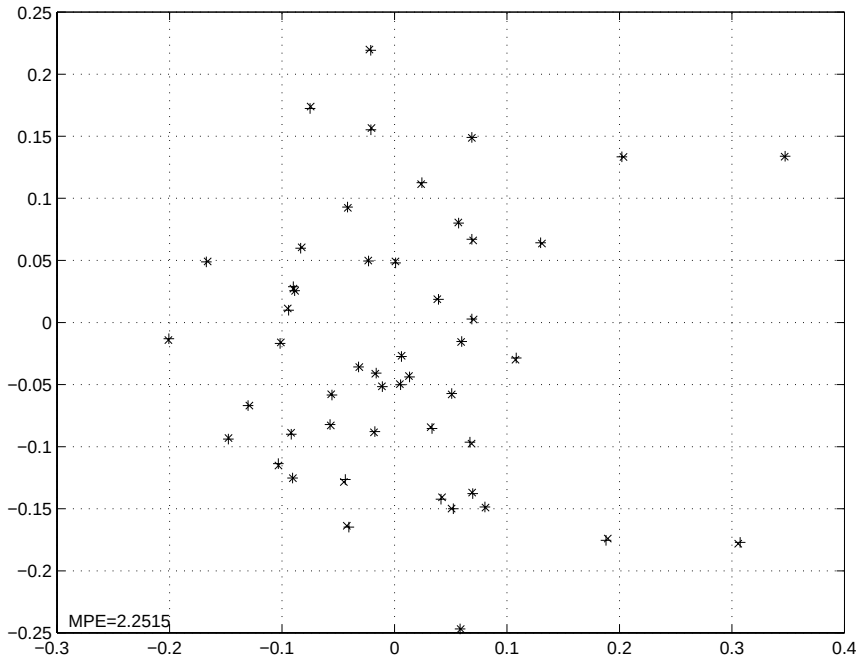

Figure 5-10: A typical image (for camera 10). The "+" denote the true image point location, the "×" denote the noise-perturbed location. Based on a 1000×1000 pixel image, the mean pixel location error is 3.24 pixels.

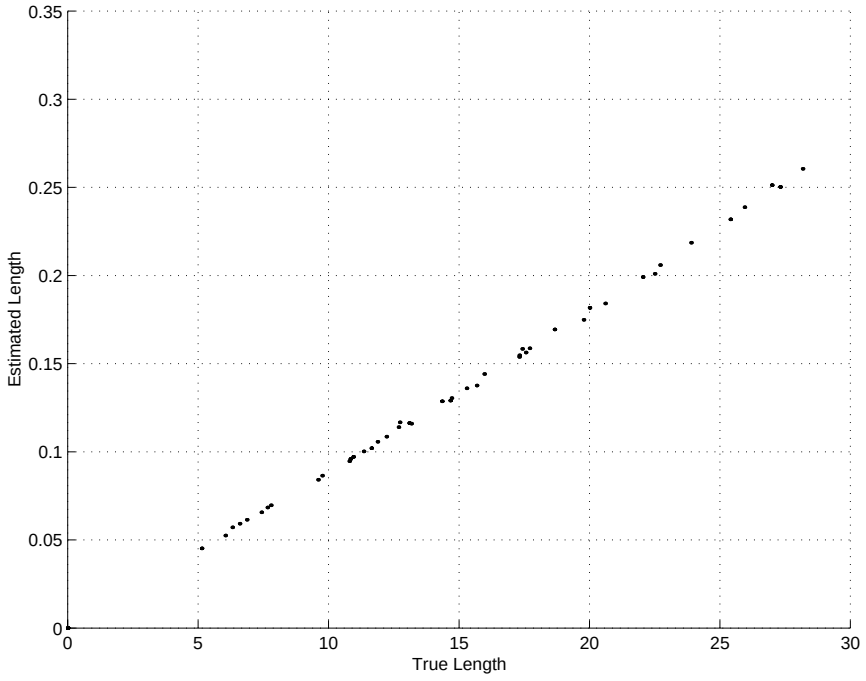

Figure 5-11: Scatter plot of true lengths between pairs of COPs and estimated lengths.

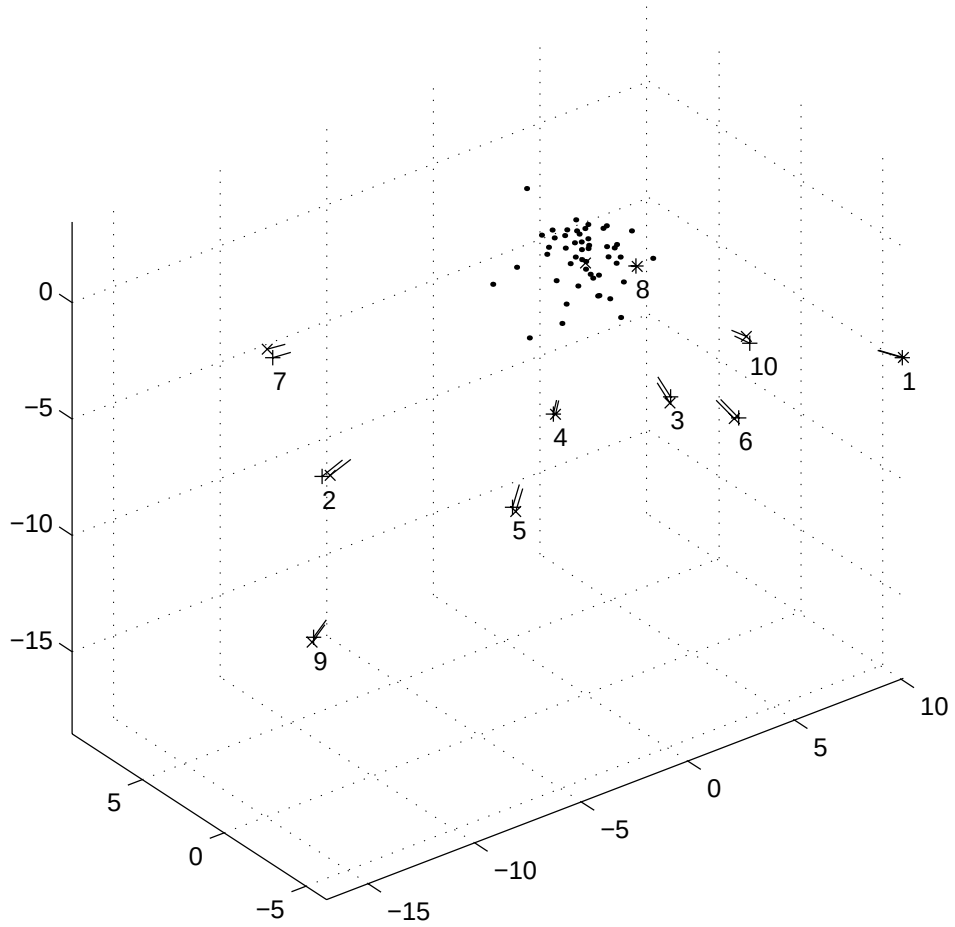

Figure 5-12: Results of camera placement estimation. The "+" denote the true COPs, the "×" denote the estimated COPs. Also shown are the true and estimated principal axis of the cameras.

5.5.8 Operations Count and FPGA Implementation

The linear epipolar algorithm gives good results for camera motion and point location reconstruction. However, the amount of operations required can become large as the number M of images grows. Table 5.1 shows the number of operations required (as determined by MATLAB's "FLOPS" command). From the table, we see that calculating the lengths between all pairs of COPs begins to dominate the number of computations as M increases (i.e. at $M = 15$, 73.8% of the calculations are devoted to length calculation).

Table 5.1: Operations Count for Linear Epipolar Algorithm. M is the number of images. The number of points was set at $N = 50$.

M	Epipoles	Lengths	Form Dot-Prods.	Solve Dot-Prods.	Cam. Att.
5	1,259,824	133,053	38,985	14,021	7,559
7	2,647,490	1,877,459	482,066	183,791	12,434
10	5,584,447	28,520,776	6,715,686	2,836,679	22,026
15	13,296,890	568,963,721	129,242,216	58,888,294	41,041

The operation of determining of the lengths thus becomes a candidate for FPGA acceleration. Recall that Section 3.6 developed the hardware efficient approach for calculating the SVD of a symmetric matrix using the CORDIC algorithm. Also recall that the CORDIC rotations are done in "cyclic sweeps," for each distinct pair of rows and columns.

To apply this efficient SVD design to the length calculation in Section 5.5.3, we first premultiply the large $3M(M-1)(M-2)/2 \times M(M-1)/2$ matrix $\mathbf{A}$ of sine values by its transpose $\mathbf{A}^T$. The resulting matrix $\mathbf{A}^T\mathbf{A}$ is symmetric and of size $M(M-1)/2 \times M(M-1)/2$. The required number of CORDIC rotations for a given column pair is thus $M(M-1)$, and the number of column pairs per cyclic sweep is (after simplification) $M(M+1)(M-1)(M-2)/8$.

Table 5.2: Number of CORDIC operations per sweep and execution times for length calculation. M is the number of images. A single CORDIC unit is assumed.

M	#CORDICs (per sweep)	Exec. time (1 sweep)	Exec. time (10 sweeps)
5	900	$18\mu s$	$180\mu s$
10	89,100	1.8ms	18ms
15	1,146,600	22.9ms	229ms
20	6,822,900	136.5ms	1.37sec
30	82,123,650	1.64sec	16.4sec
40	473,943,600	9.48sec	94.8sec

This means we have to perform a total of $M^2(M+1)(M-1)^2(M-2)/8$ CORDIC rotations per sweep. Haykin [81] states that for matrices of the sizes we consider here, on the order of 4 to 10 sweeps are required. With this information, Table 5.2 gives the total execution time for various M for a single pipelined CORDIC processor of Figure 3-28, operating at a 50MHz clock rate. We can in fact do better than this because the particular

CORDIC design wordlengths given by (3.156) required only 539 CLBs. Modern devices have available more than 20 times this number of CLBs. We could thus perform many CORDIC rotations in parallel, allowing us to speed up the calculations shown in Table 5.2 by an order of magnitude, while still requiring only a single FPGA device (of course, multiple devices may be used to further increase the processing speed).

5.6 Summary

This chapter was concerned with determining camera orientation and the three-dimensional structure of a scene using multiple data sets. The goal was to show the use of the optimal combination of estimates formula (5.21) in solving for the motion and structure parameters, and to compare to perhaps simple, less accurate methods. We saw the power of the approach through its ability to unify many different types of orientation problems. We also saw its limitations – the equations for the optimal rotation are generally very complex.

This chapter relates to the main theme of the thesis by detailing the statistically optimal method of dealing with multiple data sets; knowing the best method allows one to assess the value of suboptimal, but computationally efficient custom computing approaches.

Section 5.2 introduced the standard concepts of error ellipses, covariance matrices, and Fisher information matrices. The critical equation there was the optimal combination formula (5.21).

To be able to apply this formula to orientation problems, we needed to be able to determine the FIM of the observations for the various orientation problems. Section 5.4.1 did just that; the FIMs for a wide variety of models, from three-dimensional data to calibrated stereo, are given by (5.27)–(5.34). Section 5.4.2 showed how this formulation reduced to reasonable results for the reconstructed three-dimensional feature point locations.

Section 5.4.3 then discussed how to actually compute motion estimates from the various data sets. The maximum likelihood cost function (5.46) was optimized through the choice of rotation and translation estimates. An interesting point that does not seem to be mentioned in the literature is that for two datasets, the translation estimate can always be determined from the rotation estimate (i.e. by using (5.69)) regardless of the form of the data, thereby reducing the search space dimensionality from five to three.

To optimally estimate the multiple rotations and translations for images using the methods in Section 5.4.3 we need good starting values for the ensuing iteration. These initial motion estimates should be easy to compute and have some sort of global, approximate enforcement of motion consistency; this was neatly provided by the linear epipolar algorithm of Section 5.5. Section 5.5 was primarily concerned with an efficient method for orientation of multiple calibrated cameras from feature point matches. Pairwise epipole locations were first extracted, which then allowed us to reconstruct the locations of the camera center of projections (up to an unrecoverable global rotation, scale, and possibly a flip). With the centers of projections calculated, pairwise epipoles are once more used to calculate the camera attitudes.

The approach presented in Section 5.5 made extensive use of the SVD, for calculating the essential matrix and epipoles, determining the relative lengths between camera centers of projections, solving the dot-product constraints, and finally, for solving the Orthogonal Procrustes problem. Since it is known that the various subproblems in Sections 5.5.1-5.5.5 are not exactly solved by linear methods such as the SVD, one must view this method presented in this section as being appropriate for low feature point location noise situations.

Nevertheless, the simulation performed in Section 5.5.7 shows reasonable performance for a realistic situation. Finally, in Section 5.5.8 we applied the FPGA-based CORDIC approach from Chapter 3 to accelerate the length calculation of the linear epipolar approach.

Other opportunities to accelerate the computations of Section 5.5 are easily envisioned. For example, we could calculate the epipoles in Section 5.5.1 by forming the matrix $\mathbf{A}^T\mathbf{A}$ and then solving the ensuing eigenvector problem. A custom FPGA design would perform the multiplications and additions required to form the matrix $\mathbf{A}^T\mathbf{A}$. Complexity could be reduced slightly by taking advantage of the structure in the outer products $\mathbf{a}_i\mathbf{a}_i^T$ although there will still be many multiplications required. In this situation, the ability to implement many multiplier units in parallel in an FPGA will lead to a dramatic speedup. Reduced wordlength processing would also be employed, using the MCMC method of Chapter 3.

In solving the least squares problem of Section 5.5.4, we multiply a known vector by the pseudoinverse of a large, sparse matrix $\mathbf{A}$. Because the nonzero elements of $\mathbf{A}$ are constrained to be ± 1, the pseudoinverse matrix elements are generally of low complexity. For example, for $M = 5$, the elements of $(66 \cdot \mathbf{A}^+)$ are elements of the set $\{-10, -5, 1, 2, 4, 5, 7, 10, 17, 22\}$. The required matrix-vector product can be efficiently implemented using only shifts and additions, without the need for multiplication operations.

Still other opportunities exist to speed up the processing of Section 5.5. The CORDIC SVD custom FPGA design of Section 3.6 could be used to accelerate the eigendecomposition required in Section 5.5.5 to solve the dot-product constraints for the camera COPs. The custom FPGA implementation of absolute orientation, designed Section 3.7 could be used to accelerate the camera attitude calculation required in Section 5.5.6.

This chapter contained several contributions to orientation and photogrammetry field:

- Using perhaps non-invertible FIMs to describe data in a unified format, and showing the resulting relationship between this general representation and known special cases (Section 5.4),
- the fast, approximate, SVD-based epipolar algorithm for multiple frame orientation (Section 5.5).

The approaches to solving the multiple view and data set orientation problems given in this chapter again demonstrate the interplay between algorithms and efficient implementations.

Chapter 6

Summary and Future Directions

Computer vision processing has increased in importance in numerous applications in recent years; this has been largely due to new developments in microelectronic technology, but has also relied on corresponding advances in algorithm-specific design techniques. Some of the newest and most exciting applications on the horizon are highly dependent on cost-effective, reliable, and low-power implementations of high-performance systems. It is natural to expect that the methods and techniques put forth in this thesis can be applied to other orientation and computer vision problems; we mention here a few possibilities.

For consumer applications, custom computing will allow more wearable (or pocketable) devices to be manufactured compactly and inexpensively. As an example, consider portable biometric devices. Biometrics seeks to measure biological features (i.e. fingerprints) in order to verify a person's identity. Currently, this operation is usually confined to entry points of buildings or controlled access areas, so efficiency is generally not a concern. We can envision that using the techniques of this thesis, portable biometric verification, perhaps incorporated into a credit-card, could become feasible. This would allow more widespread usage of biometrics for identity verification.

Robotic and vehicle navigation are obvious applications which can benefit from low-cost and robust vision processing. Being able to detect obstacles, estimate time to collision, and calculate deviations from the planned route, are all requirements of any mobile system.

Surveillance applications easily benefit from custom computing. For example, the ability to automatically distinguish motion as human vs. animal or machine, or perhaps friendly vs. unfriendly would relieve security personnel of the tedious task of staring at television monitors for hours on end. For automobile traffic monitoring, the ability to measure traffic flow and to detect accidents automatically in real-time, would allow for the widespread distribution of sensors, so that a more complete picture of the traffic situation could be known to all travelers.

For military applications, efficient vision processing will be essential to the operation of mobile sensor networks. For example, one sensor might recognize an object or target. Other sensors would then reprogram so that they might efficiently recognize and locate the object as well. To make this possible, the ability to implement a variety of algorithms efficiently is essential.

In support of these possible applications, research into the basic techniques will continue to receive attention, and some areas where we can expect continued activity are mentioned below.

Direct methods (i.e. calculation of motion from image brightness and temporal gradients) provide many opportunities for custom computing acceleration. We saw in Chapter 4 the design of a circuit for estimating the motion of a planar scene. A similar custom computing design good be obtained for accelerating Stein's direct method based on the trifocal tensor

(covered briefly in Section 2.4).

Optical flow methods were not covered in depth in this thesis. The calculation of optical flow requires the calculation and processing of first and second derivatives of image brightness data; there are many similarities between optical flow based orientation and direct methods in this regard. Optical flow calculation may be more complicated because of the need for regularization, often done interactively under user control. Custom computing acceleration of the basic optical flow calculation then would greatly improve the usability of interactive optimization.

Gradient-based methods for complex scenes encounter problems at occlusion boundaries because various smoothness assumptions no longer apply, and there is a tremendous amount of literature on methods proposed to overcome this problem. Of current interest are Markov random fields approaches. These are iterative and computationally intensive approaches, which are closely related to the MCMC technique and simulated annealing. We can easily imagine a custom processor which accelerates the computation at each image pixel. Thus, custom computing may offer a way to perform joint segmentation and motion estimation in real-time. This type of architecture could also allow extension of real-time motion estimation to non-rigid scenes.

In this thesis, we mainly focused on calibrated cameras; indeed Section 2.7 showed how a known object could be used to efficiently calibrate a camera. Not covered in this thesis is the large and extremely active area of research concerning the use of uncalibrated cameras to recover scene structure and camera orientation. Clearly, custom implementation of these projective geometry approaches is a possibility. Similar comments apply to the various orthographic, scaled orthographic, and paraperspective image formation models that are receiving attention in the literature.

This thesis has focused on the design and implementation aspect of orientation and photogrammetry, encompassing theory, algorithms, architectures and circuits. Although many of the core orientation algorithms have been known for quite sometime, it is only recently that the emergence of implementation technology has allowed their economical and widespread application.

While the field of custom computing has been active for the last decade, the specific focus on the codesign of algorithm and implementation has taken on an increased importance due to the rapid expansion of consumer demand for high-performance products. In contrast to the past, custom computing algorithm implementations are not used in a standalone system, but are typically placed within a larger overall application, often performing real-time high-performance tasks.

Many challenges remain for designers of portable, cost-effective, real-time systems. One of the most interesting questions in the custom computing field is to what extent the current FPGA architectures should be altered to better support high-performance numerical computations. For example, ongoing research is investigating architectures based on 4 or 8-bit data paths, as opposed to the one-bit datapaths of today. Other research is aimed at incorporating general-purpose microprocessors and FPGA resources on the same die, in an attempt to get the best features of both technologies. Many of the applications studied by researchers in their attempt to refine FPGA architectures have been in high-profile areas, such as wireless communications, automatic target recognition, and image compression. It is hoped that this thesis has provided the background and motivation for similar treated of high-performance orientation architectures.

Orientation and model-based target recognition have been applied to traditional pinhole camera images and image sequences, where no computational effort is required to form

the images. Of growing importance is the area of *computed imagery*, where we must perform significant amount of digital signal processing to obtain an image in the first place. Examples include synthetic aperture radar (both strip-map and spotlight mode), x-ray tomography, acoustic tomography for both medical ultrasound imaging and ocean acoustic imaging, optical tomography for non-line-of-site propagation media (i.e. turbulent atmospheric and oceanographic imagery), and hyperspectral processing for remote sensing of environmental changes.

While the scientific communities for these specialties are enamored of the beautifully rendered images obtained from these sensing modalities, it is the engineer who must become increasingly concerned with the formation of these images. From a detection and estimation viewpoint, it is not the computed images themselves that are important; rather, it is the ability to extract and match features to models that must become of paramount interest.

We therefore expect to see custom computing as a way to integrate image formation, feature extraction, precision platform orientation, photogrammetry for environmental remote sensing, and model-based detection, in a way that not only improves the total system performance, but at a drastically reduced system cost and computation time. This will enable the high-speed prescreening of voluminous, high-dimensional data sets, which are already well beyond the two-dimensional realm of pin-hole camera images.

There are of course many issues that were not covered in this thesis. In the hardware realm, for example, systolic arrays that implement orthogonal decompositions and matrix multiplications are well known. In the next few years, FPGA densities will increase enough to allow these systolic algorithms to be efficiently implemented. Also not covered are power-dissipation issues. The algorithms of Chapter 3 could easily be adapted to include power consumption as an optimization criterion.

In the last few years, the field of algebraic geometry has found increasing application to engineering problems. Many of the orientation and photogrammetry problems considered in this thesis can be cast into the form of solving high-dimensional multivariate polynomial equations. System designers are generally put off by the requirement to solve these equations in real-time. With the increasing understanding that methods better than a generic constrained optimization exist, we can expect to see more application of algebraic geometry to computer vision problems in the future.

Closely related to algebraic geometry is tensor analysis. Long the realm of physicists, tensor concepts have been steadily invading the computer vision literature. Current work is focused on the nature of constraints given by multiple images. In the signal processing field, tensors are being used to solve for unknowns that appear as products in the relevant equations; since this also characterizes most orientation and photogrammetry problems, we can expect an influx of ideas from this direction into the computer vision field.

Clearly, no thesis could cover in depth all these relevant topics on algorithms and implementation. What this thesis did was to find design approaches that had immediate applicability to most orientation and photogrammetric problems. It is hoped that the methods and results developed in this thesis have established a foundation that will serve as a basis for further research.

Appendix A

Acronyms

ALU: Arithmetic and Logic Unit
ASIC: Application Specific Integrated Circuit
ATM: Automated Teller Machine
ATR: Automatic Target Recognition
AWGN: Additive White Gaussian Noise
BCCE: Brightness Change Constraint Equation
CAD: Computer Aided Design
CAT: Computer Aided Tomography
CCD: Charge Coupled Device
CLA: Carry Lookahead Adder
CLB: Configurable Logic Block
CML: Constrained Maximum Likelihood
COP: Center of Projection
CORDIC: Coordinate Rotation, Digital Computer
CRLB: Cramer-Rao Lower Bound
CSA: Carry Save Adder
CTLS: Constrained Total Least Squares
DARPA: Defense Research Project Agency
DFT: Discrete Fourier Transform
DSP: Digital Signal Processing (or Processor)
EKF: Extended Kalman Filter
EVD: Eigenvalue Decomposition
FA: Full Adder cell
FFT: Fast Fourier Transform
FIM: Fisher Information Matrix
FIR: Finite Impulse Response
FOE: Focus of Expansion
FOV: Field of View
FPGA: Field Programmable Gate Array
GPS: Global Positioning System
IEKF: Iterated, Extended Kalman Filter
IIR: Infinite Impulse Response
IOB: Input/Output Logic Block
IR: Infrared

LMS: Least Mean-Squares
LR: Lazy Rounding
LSB: Least Significant Bit
LUT: Lookup Table
MAP: Maximum *a posteriori* (estimation)
MAV: Micro Air-Vehicle
MCMC: Markov Chain Monte Carlo Method
MHz: MegaHertz
MLE: Maximum Likelihood Estimation
MRI: Magnetic Resonance Imaging
MSB: Most Significant Bit
MSE: Mean Square Error
MUX: Multiplexer
NRE: Non-recurring Engineering
OLS: Ordinary Least Squares
PDF: Probability Density Function
PLD: Programmable Logic Device
POLYACC: Polynomial Accumulator
RAM: Read/Write Memory
RCA: Ripple Carry Adder
RLR: Really Lazy Rounding
ROM: Read-Only Memory
RPD: Relative Pointwise Distance
SAR: Synthetic Aperture Radar
SBP: Summation By Parts
STLN: Structured Total Least Norms
STLS: Structured Total Least Squares
SVD: Singular Value Decomposition
TLS: Total Least Squares
UAV: Unmanned Aerial Vehicle
VHDL: VHSIC Hardware Design Language
VHSIC: Very High Speed Integrated Circuit
VLSI: Very Large Scale Integration
XC4000: Xilinx 4000 family of FPGAs

Appendix B

Quaternion Algebra

Quaternions are frequently used to represent rotations. This material is taken mainly from Horn [88], [89], [91], and Dron [48].

Quaternions are elements of a 4D vector space and can be represented as a vector,

$$\mathring{\mathbf{q}} \stackrel{\triangle}{=} \begin{bmatrix} q_0 \\ q_x \\ q_y \\ q_z \end{bmatrix}, \tag{B.1}$$

or a combination of a scalar component and a 3D vector component,

$$\mathring{\mathbf{q}} \stackrel{\triangle}{=} (q_0, \mathbf{q}), \tag{B.2}$$

where

$$\mathbf{q} \stackrel{\triangle}{=} \begin{bmatrix} q_x \\ q_y \\ q_z \end{bmatrix}. \tag{B.3}$$

Because quaternions may be thought of as a vector, common vector operations such as the inner product of two quaternions, transpose, and multiplication of a matrix by a quaternion are defined normally.

An element of 3D Euclidean space may be represented by a quaternion with zero scalar part. A scalar may be represented as a quaternion with a zero vector part. The conjugate $\mathring{\mathbf{q}}^*$ of a quaternion $\mathring{\mathbf{q}}$ is defined by

$$\mathring{\mathbf{q}}^* \stackrel{\triangle}{=} (q_0, -\mathbf{q}). \tag{B.4}$$

The addition $\mathring{\mathbf{c}}=\mathring{\mathbf{a}} + \mathring{\mathbf{b}}$ of quaternions is defined by

$$(c_0, \mathbf{c}) \stackrel{\triangle}{=} (a_0 + b_0, \mathbf{a} + \mathbf{b}). \tag{B.5}$$

Addition of quaternions is associative and commutative, with the additive identity being $(0, \mathbf{0})$.

The multiplication of two quaternions is given by

$$\mathring{\mathbf{a}}\mathring{\mathbf{b}} \stackrel{\triangle}{=} (a_0 b_0 - \mathbf{a} \cdot \mathbf{b}, a_0\mathbf{b} + b_0\mathbf{a} + \mathbf{a} \times \mathbf{b}). \tag{B.6}$$

Quaternion multiplication is associative (but not commutative),

$$\mathring{\mathbf{a}}\,(\mathring{\mathbf{b}}\mathring{\mathbf{c}}) = (\mathring{\mathbf{a}}\mathring{\mathbf{b}})\,\mathring{\mathbf{c}}\,. \tag{B.7}$$

The multiplicative identity is $\overset{\circ}{\mathbf{e}} \overset{\triangle}{=} (1, \mathbf{0})$. The squared magnitude of a quaternion is given by

$$\| \overset{\circ}{\mathbf{q}} \|^2 \overset{\triangle}{=} \overset{\circ}{\mathbf{q}} \overset{\circ}{\mathbf{q}}^* = \overset{\circ}{\mathbf{q}} \cdot \overset{\circ}{\mathbf{q}} = q_0^2 + q_x^2 + q_y^2 + q_z^2 \tag{B.8}$$

This shows that a nonzero quaternion has an inverse

$$\overset{\circ}{\mathbf{q}}^{-1} = \frac{\overset{\circ}{\mathbf{q}}^*}{\| \overset{\circ}{\mathbf{q}} \|^2} \tag{B.9}$$

Some useful quaternion identities are:

$$\begin{aligned}
\left(\overset{\circ}{\mathbf{a}} \overset{\circ}{\mathbf{a}}^* \right) &= \left(\overset{\circ}{\mathbf{a}} \cdot \overset{\circ}{\mathbf{a}} \right) \overset{\circ}{\mathbf{e}} \\
\left(\overset{\circ}{\mathbf{a}} \overset{\circ}{\mathbf{b}} \right)^* &= \overset{\circ}{\mathbf{b}}^* \overset{\circ}{\mathbf{a}}^* \\
\left(\overset{\circ}{\mathbf{a}} \overset{\circ}{\mathbf{c}} \right) \cdot \left(\overset{\circ}{\mathbf{b}} \overset{\circ}{\mathbf{c}} \right) &= \left(\overset{\circ}{\mathbf{a}} \cdot \overset{\circ}{\mathbf{b}} \right) \left(\overset{\circ}{\mathbf{c}} \cdot \overset{\circ}{\mathbf{c}} \right) \\
\left(\overset{\circ}{\mathbf{a}} \overset{\circ}{\mathbf{b}} \right) \cdot \left(\overset{\circ}{\mathbf{a}} \overset{\circ}{\mathbf{b}} \right) &= \left(\overset{\circ}{\mathbf{a}} \cdot \overset{\circ}{\mathbf{a}} \right) \left(\overset{\circ}{\mathbf{b}} \cdot \overset{\circ}{\mathbf{b}} \right) \\
\left(\overset{\circ}{\mathbf{a}} \overset{\circ}{\mathbf{c}} \right) \cdot \overset{\circ}{\mathbf{b}} &= \overset{\circ}{\mathbf{a}} \cdot \left(\overset{\circ}{\mathbf{b}} \overset{\circ}{\mathbf{c}} \right).
\end{aligned} \tag{B.10}$$

A 3D vector can be represented by a quaternion with zero scalar part. If $\overset{\circ}{\mathbf{a}}$, $\overset{\circ}{\mathbf{b}}$, and $\overset{\circ}{\mathbf{c}}$ are quaternions with zero scalar part then

$$\begin{aligned}
\overset{\circ}{\mathbf{a}}^* &= - \overset{\circ}{\mathbf{a}}^* \\
\overset{\circ}{\mathbf{a}} \cdot \overset{\circ}{\mathbf{b}} &= \mathbf{a} \cdot \mathbf{b} \\
\overset{\circ}{\mathbf{a}} \overset{\circ}{\mathbf{b}} &= (-\mathbf{a} \cdot \mathbf{b}, \mathbf{a} \times \mathbf{b}) \\
\overset{\circ}{\mathbf{a}} \cdot \left(\overset{\circ}{\mathbf{b}} \overset{\circ}{\mathbf{c}} \right) &= \overset{\circ}{\mathbf{a}} \cdot (\mathbf{b} \times \mathbf{c}) = [\mathbf{abc}].
\end{aligned} \tag{B.11}$$

The last identity represents the triple product.

It is extremely useful on occasion to represent quaternion multiplication in a matrix-vector product format. We define a *left quaternion matrix* $\mathcal{Q}$ associated with a quaternion $\overset{\circ}{\mathbf{q}}$ by

$$\mathcal{Q} \overset{\triangle}{=} \begin{bmatrix} q_0 & -q_x & -q_y & -q_z \\ q_x & q_0 & -q_z & q_y \\ q_y & q_z & q_0 & -q_x \\ q_z & -q_y & q_x & q_0 \end{bmatrix}. \tag{B.12}$$

Using the left quaternion matrix representation, the quaternion product can be written as

$$\overset{\circ}{\mathbf{a}} \overset{\circ}{\mathbf{b}} = \mathcal{A} \overset{\circ}{\mathbf{b}}. \tag{B.13}$$

Similarly, we define a *right quaternion matrix* $\mathcal{Q}$ associated with a quaternion $\mathring{\mathbf{q}}$ by

$$\overline{\mathcal{Q}} \triangleq \begin{bmatrix} q_0 & -q_x & -q_y & -q_z \\ q_x & q_0 & q_z & -q_y \\ q_y & -q_z & q_0 & q_x \\ q_z & q_y & -q_x & q_0 \end{bmatrix}. \tag{B.14}$$

Using the right quaternion matrix representation, the quaternion product can be written as

$$\mathring{\mathbf{a}}\mathring{\mathbf{b}} = \overline{\mathcal{B}}\,\mathring{\mathbf{a}}\,. \tag{B.15}$$

A unit quaternion corresponds to a rotation in 3D; a unit quaternion is one whose magnitude is unity, so that

$$\| \mathring{\mathbf{q}} \| = 1. \tag{B.16}$$

The inverse of a unit quaternion is just its conjugate.

The unit quaternion $\mathring{\mathbf{q}}$ that represents a rotation by an angle θ about an axis $\hat{\boldsymbol{\omega}}$ is given by

$$\mathring{\mathbf{q}} = \left(\cos\frac{\theta}{2}, \hat{\boldsymbol{\omega}} \sin\frac{\theta}{2}\right), \tag{B.17}$$

where $\|\hat{\boldsymbol{\omega}}\| = 1$. A vector $\mathbf{a}$ may be rotated by a unit quaternion to a new vector $\mathbf{b}$ using

$$\mathring{\mathbf{b}} = \mathring{\mathbf{q}}\mathring{\mathbf{a}}\mathring{\mathbf{q}}^*, \tag{B.18}$$

where $a_0 = b_0 = 0$. This may be written using quaternion matrices as

$$\mathring{\mathbf{q}}\mathring{\mathbf{a}}\mathring{\mathbf{q}}^* = (\mathcal{Q}\,\mathring{\mathbf{a}})\,\mathring{\mathbf{q}}^* = \overline{\mathcal{Q}^*}(\mathcal{Q}\,\mathring{\mathbf{a}}) = \overline{\mathcal{Q}}^{\mathcal{T}}\mathcal{Q}\,\mathring{\mathbf{a}} \tag{B.19}$$

So

$$\begin{aligned} \mathring{\mathbf{b}} &= \overline{\mathcal{Q}}^T \mathcal{Q}\,\mathring{\mathbf{a}} \\ &= \begin{bmatrix} 1 & \mathbf{0} \\ \mathbf{0} & \mathbf{R} \end{bmatrix} \mathring{\mathbf{a}}, \end{aligned}, \tag{B.20}$$

where $\mathbf{R}$ is an orthonormal rotation matrix. Expanding terms allows us to find $\mathbf{R}$ in terms of the unit quaternion:

$$\mathbf{R} = \begin{bmatrix} q_0^2 + q_x^2 - q_y^2 - q_z^2 & 2(q_x q_y - q_0 q_z) & 2(q_x q_z + q_0 q_y) \\ 2(q_x q_y + q_0 q_z) & q_0^2 - q_x^2 + q_y^2 - q_z^2 & 2(q_y q_z - q_0 q_x) \\ 2(q_x q_z - q_0 q_y) & 2(q_y q_z + q_0 q_x) & q_0^2 - q_x^2 - q_y^2 + q_z^2 \end{bmatrix}. \tag{B.21}$$

Noting that

$$(-\mathring{\mathbf{q}})\,\mathring{\mathbf{a}}\,(-\mathring{\mathbf{q}}^*) = \mathring{\mathbf{b}} = \mathring{\mathbf{q}}\mathring{\mathbf{a}}\mathring{\mathbf{q}}^*, \tag{B.22}$$

we see that $-\mathring{\mathbf{q}}$ and $\mathring{\mathbf{q}}$ represent the same rotation. Thus, the space of rotations is isomorphic to one hemisphere of a unit 3D sphere.

Rotations may be composed using unit quaternions. Let $\mathbf{a}$ be rotated by $\mathring{\mathbf{q}}$ into $\mathbf{b}$, then $\mathbf{b}$ be rotated by $\mathring{\mathbf{p}}$ into $\mathbf{c}$:

$$\begin{aligned} \mathring{\mathbf{b}} &= \mathring{\mathbf{q}}\mathring{\mathbf{a}}\mathring{\mathbf{q}}^*, \\ \mathring{\mathbf{c}} &= \mathring{\mathbf{p}}\mathring{\mathbf{b}}\mathring{\mathbf{p}}^*. \end{aligned} \tag{B.23}$$

We have

$$\begin{aligned}\overset{\circ}{\mathbf{c}} &= \overset{\circ}{\mathbf{p}}\,(\overset{\circ}{\mathbf{q}}\overset{\circ}{\mathbf{a}}\overset{\circ}{\mathbf{q}}^{*})\,\overset{\circ}{\mathbf{p}}^{*}, \\ &= (\overset{\circ}{\mathbf{p}}\overset{\circ}{\mathbf{q}})\,\overset{\circ}{\mathbf{a}}\,(\overset{\circ}{\mathbf{q}}^{*}\overset{\circ}{\mathbf{p}}^{*}), \\ &= (\overset{\circ}{\mathbf{p}}\overset{\circ}{\mathbf{q}})\,\overset{\circ}{\mathbf{a}}\,(\overset{\circ}{\mathbf{p}}\overset{\circ}{\mathbf{q}})^{*},\end{aligned} \tag{B.24}$$

where $\overset{\circ}{\mathbf{p}}\overset{\circ}{\mathbf{q}}$ is a unit quaternion. Thus, composition of rotations corresponds to multiplication of unit quaternions.

We note that it takes fewer operations to multiply two quaternions than it does to multiply two rotation matrices. Also, due to finite wordlength effects, a sequence of rotations may no longer be orthonormal. For quaternions, simply rounding to the nearest orthonormal quaternion is easy, whereas finding the nearest orthonormal matrix to a given matrix is hard. For this reason, quaternions find extensive use in computer graphics

When converting a unit quaternion to a rotation matrix, (B.21) may be used directly, but a more efficient method is to use the following "straight line code:"

$$\begin{aligned}t_1 &= q_x^2, \\ t_2 &= q_y^2, \\ t_3 &= q_z^2, \\ t_4 &= q_0 q_x, \\ t_5 &= q_0 q_y, \\ t_6 &= q_0 q_z, \\ t_7 &= q_x q_y, \\ t_8 &= q_x q_z, \\ t_9 &= q_y q_z, \\ R_{1,1} &= 1 - 2(t_2 + t_3), \\ R_{2,2} &= 1 - 2(t_1 + t_3), \\ R_{3,3} &= 1 - 2(t_1 + t_2), \\ R_{1,2} &= 2(t_7 - t_6), \\ R_{2,1} &= 2(t_7 + t_6), \\ R_{3,1} &= 2(t_8 - t_5), \\ R_{1,3} &= 2(t_8 + t_5), \\ R_{2,3} &= 2(t_9 - t_4), \\ R_{3,2} &= 2(t_9 + t_4),\end{aligned} \tag{B.25}$$

for a total of 9 multiplications and 9 additions. This assumes that the quaternion is indeed normalized, so that $q_0^2 = 1 - q_x^2 - q_y^2 - q_z^2$.

Converting from a perfectly orthonormal rotation matrix $\mathbf{R}$ to a unit quaternion $\overset{\circ}{\mathbf{q}}$ would appear to be a simple matter. We note that

$$q_0^2 = \cos^2 \frac{\theta}{2} = \frac{\mathrm{tr}(\mathbf{R}) + 1}{4}, \tag{B.26}$$

and that $\hat{\boldsymbol{\omega}}$ may be obtained by solving the eigenvector equation

$$\mathbf{R}\hat{\boldsymbol{\omega}} = \hat{\boldsymbol{\omega}}. \tag{B.27}$$

We then use (B.17) to recover $\overset{\circ}{\mathbf{q}}$.

Perhaps more simply, we note that the diagonal of $\mathbf{R}$ in (B.21) and the unit magnitude constraint give four linear equations, which may be summarized as:

$$\begin{bmatrix} 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} q_0^2 \\ q_x^2 \\ q_y^2 \\ q_z^2 \end{bmatrix} = \begin{bmatrix} R_{1,1} \\ R_{2,2} \\ R_{3,3} \\ 1 \end{bmatrix}. \tag{B.28}$$

Inverting the matrix gives

$$\begin{bmatrix} q_0^2 \\ q_x^2 \\ q_y^2 \\ q_z^2 \end{bmatrix} = \frac{1}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & -1 & 1 \\ -1 & 1 & -1 & 1 \\ -1 & -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} R_{1,1} \\ R_{2,2} \\ R_{3,3} \\ 1 \end{bmatrix}. \tag{B.29}$$

To resolve the sign ambiguities, we choose $q_0 \geq 0$ and note that the off-diagonal components of $\mathbf{R}$ give the following three equations:

$$\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} q_y q_z \\ q_0 q_x \end{bmatrix} = \tfrac{1}{2} \begin{bmatrix} R_{3,2} \\ R_{2,3} \end{bmatrix},$$

$$\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} q_x q_z \\ q_0 q_y \end{bmatrix} = \tfrac{1}{2} \begin{bmatrix} R_{1,3} \\ R_{3,1} \end{bmatrix}, \tag{B.30}$$

$$\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} q_x q_y \\ q_0 q_z \end{bmatrix} = \tfrac{1}{2} \begin{bmatrix} R_{2,1} \\ R_{1,2} \end{bmatrix}.$$

From these equations we extract the products $q_0 q_x$, $q_0 q_y$, $q_0 q_z$ as

$$\begin{aligned} q_0 q_x &= (R_{3,2} - R_{2,3})/4, \\ q_0 q_y &= (R_{1,3} - R_{3,1})/4, \\ q_0 q_z &= (R_{2,1} - R_{1,2})/4. \end{aligned} \tag{B.31}$$

Because we chose $q_0 \geq 0$, the signs of q_x, q_y, and q_z may therefore be determined simply as

$$\begin{aligned} \text{sign}(q_x) &= \text{sign}(R_{3,2} - R_{2,3}), \\ \text{sign}(q_y) &= \text{sign}(R_{1,3} - R_{3,1}), \\ \text{sign}(q_z) &= \text{sign}(R_{2,1} - R_{1,2}). \end{aligned} \tag{B.32}$$

We can summarize this procedure using the straight line code:

$$\begin{aligned}
t_1 &= R_{1,1} + R_{2,2}, \\
t_2 &= R_{1,1} - R_{2,2}, \\
t_3 &= 1 + R_{3,3}, \\
t_4 &= 1 - R_{3,3}, \\
t_5 &= t_3 + t_1, \\
t_6 &= t_3 - t_1, \\
t_7 &= t_4 + t_2, \\
t_8 &= t_4 - t_2, \\
q_0 &= \sqrt{t_5}/2, \\
t_9 &= \sqrt{t_7}/2, \\
t_{10} &= \sqrt{t_8}/2, \\
t_{11} &= \sqrt{t_6}/2, \\
q_x &= \text{sign}(R_{3,2} - R_{2,3})t_9, \\
q_y &= \text{sign}(R_{1,3} - R_{3,1})t_{10}, \\
q_z &= \text{sign}(R_{2,1} - R_{1,2})t_{11},
\end{aligned} \tag{B.33}$$

for a total of 4 square roots and 11 additions.

We noted earlier that the composition of rotations may be represented as the product of unit quaternions. Using the matrix representation of a quaternion, we see that only 16 multiplications and 12 additions are needed to compose two quaternions, whereas 27 multiplications and 18 additions are needed to form the product of two rotation matrices.

Note the structure of the left quaternion matrix representation in (B.12); it is suggestive of a trick used to calculate a complex multiplication using only three real multiplications [22]. Recall that calculating $(e + jf) = (a + jb) \cdot (c + jd)$ the straightforward way,

$$e = ac - bd, \quad f = ad + bc, \tag{B.34}$$

four real multiplications and two real additions are required. The more subtle method [22] requires only three real multiplications (but five additions):

$$e = (a - b)d + a(c - d), \quad f = (a - b)d + b(c + d). \tag{B.35}$$

Rewriting (B.34) in a matrix-vector format,

$$\begin{bmatrix} e \\ f \end{bmatrix} = \begin{bmatrix} a & -b \\ b & a \end{bmatrix} \begin{bmatrix} c \\ d \end{bmatrix}, \tag{B.36}$$

we notice that the structure of the matrix in (B.36) is identical to the structure of the left quaternion matrix $\overline{\mathcal{Q}}$, if we view $\overline{\mathcal{Q}}$ as a 2×2 block matrix with 2×2 sub-blocks. Therefore, to apply this idea to quaternion composition, we make the associations

$$\mathring{\mathbf{r}} = \mathring{\mathbf{q}}\mathring{\mathbf{p}} = \mathcal{Q}\,\mathring{\mathbf{p}} = \left[\begin{array}{c|c} \mathbf{A} & \text{- } \mathbf{B} \\ \hline \mathbf{B} & \mathbf{A} \end{array}\right] \left[\begin{array}{c} \mathbf{c} \\ \hline \mathbf{d} \end{array}\right] \tag{B.37}$$

where

$$\mathbf{A} = \begin{bmatrix} q_0 & -q_x \\ q_x & q_0 \end{bmatrix}, \; \mathbf{B} = \begin{bmatrix} q_y & q_z \\ q_z & -q_y \end{bmatrix}, \tag{B.38}$$

$$\mathbf{c} = \begin{bmatrix} p_0 \\ p_x \end{bmatrix}, \ \mathbf{d} = \begin{bmatrix} p_y \\ p_z \end{bmatrix}, \tag{B.39}$$

and

$$\mathbf{e} = \begin{bmatrix} r_0 \\ r_x \end{bmatrix}, \ \mathbf{f} = \begin{bmatrix} r_y \\ r_z \end{bmatrix}. \tag{B.40}$$

We thus have

$$\mathbf{A} - \mathbf{B} = \begin{bmatrix} q_0 - q_y & -q_x - q_z \\ q_x - q_z & q_0 + q_y \end{bmatrix}, \tag{B.41}$$

and

$$\mathbf{c} - \mathbf{d} = \begin{bmatrix} p_0 - p_y \\ p_x - p_z \end{bmatrix}, \ \mathbf{c} + \mathbf{d} = \begin{bmatrix} p_0 + p_y \\ p_x + p_z \end{bmatrix}. \tag{B.42}$$

The term $(\mathbf{A} - \mathbf{B})\mathbf{d}$ is calculated in the straightforward way using

$$(\mathbf{A} - \mathbf{B})\mathbf{d} = \begin{bmatrix} q_0 - q_y & -q_x - q_z \\ q_x - q_z & q_0 + q_y \end{bmatrix} \begin{bmatrix} p_y \\ p_z \end{bmatrix}, \tag{B.43}$$

which requires four multiplications To calculate the terms $\mathbf{A}(\mathbf{c} - \mathbf{d})$ and $\mathbf{B}(\mathbf{c} + \mathbf{d})$, we use the complex multiplication trick again, to further reduce the number of multiplications to three for each of these terms. The total number of multiplications for the entire algorithm is thus 10. The total procedure can be summarized by the straightline code:

$$\begin{aligned}
t_1 &= p_0 - p_y, \\
t_2 &= p_0 + p_y, \\
t_3 &= p_x - p_z, \\
t_4 &= p_x + p_z, \\
t_5 &= (q_0 - q_x)t_3, \\
t_6 &= q_0(t_1 - t_3), \\
t_7 &= q_x(t_1 + t_3), \\
t_8 &= t_5 + t_6, \\
t_9 &= t_5 + t_7, \\
t_{10} &= (q_z - q_y)t_4, \\
t_{11} &= q_y(t_2 + t_4), \\
t_{12} &= q_z(t_2 - t_4), \\
t_{13} &= t_{10} + t_{11}, \\
t_{14} &= t_{10} + t_{12}, \\
t_{15} &= q_0 - q_y, \\
t_{16} &= q_0 + q_y, \\
t_{17} &= q_x - q_z, \\
t_{18} &= q_x + q_z, \\
t_{19} &= t_{15}p_y - t_{18}p_z, \\
t_{20} &= t_{17}p_y + t_{16}p_z, \\
r_0 &= t_{19} + t_8, \\
r_x &= t_{20} + t_9, \\
r_y &= t_{19} + t_{13}, \\
r_z &= t_{20} + t_{14}.
\end{aligned} \tag{B.44}$$

The procedure requires only 10 multiplications and 24 additions.

There are, however, more efficient implementations of quaternion multiplication. The following straight line code computes the quaternion product $\mathring{\mathbf{r}}=\mathring{\mathbf{q}}\mathring{\mathbf{p}}$ using only eight multiplications and 27 additions:

$$
\begin{aligned}
t_1 &= (q_z - q_y)(p_y - p_z), \\
t_2 &= (q_0 + q_x)(p_0 + p_x), \\
t_3 &= (q_0 - q_x)(p_y + p_z), \\
t_4 &= (q_y + q_z)(p_0 - p_x), \\
t_5 &= (q_z - q_x)(p_x - p_y), \\
t_6 &= (q_z + q_x)(p_x + p_y), \\
t_7 &= (q_0 + q_y)(p_0 - p_z), \\
t_8 &= (q_0 - q_y)(p_0 + p_z), \\
t_9 &= t_6 + t_7 + t_8, \\
t_{10} &= (t_5 + t_9)/2, \\
r_0 &= t_1 + t_{10} - t_6, \\
r_x &= t_2 + t_{10} - t_9, \\
r_y &= t_3 + t_{10} - t_8, \\
r_z &= t_4 + t_{10} - t_9.
\end{aligned}
\tag{B.45}
$$

Appendix C

Multiple Photographs - Bundle Adjustment and Photogrammetry

Photogrammetrists typically use iterative Newton-Raphson approaches for aerotriangulation and photographic surveys. It usually the case that multiple, overlapping photographs are taken, arranged in a grid-like fashion. With availability of known, surveyed ground control point, the problem thus becomes one of a simultaneous adjustment of the image coordinates, camera location and attitude, and control points. In this section we will review this method, following the approach outlined in [163].

We now proceed to cast the solution of sets of nonlinear equations, which result from the general perspective projection camera model, into the above framework. In general, a problem model consisting of m equations, involving n unknowns and r measurements can be expressed as

$$\begin{aligned} f_1(\boldsymbol{\theta}; \mathbf{y}) &= 0, \\ &\vdots \\ f_m(\boldsymbol{\theta}; \mathbf{y}) &= 0, \end{aligned} \tag{C.1}$$

where $\boldsymbol{\theta}$ is an $n \times 1$ vector of unknown, nonrandom parameters, and $\mathbf{y}$ is an $r \times 1$ vector of observations or known values. These nonlinear equations must be linearized before they can be used in an iterative adjustment solution. Generally, Newton's first order approximation is used. Let the measured values be denoted as $\mathbf{y}^{00}$, and let these be related to the true values $\mathbf{y}$, via

$$\mathbf{y} = \mathbf{y}^{00} + \mathbf{v}, \tag{C.2}$$

where $\mathbf{v}$ is a vector of Gaussian noise variables. Furthermore, let $\boldsymbol{\theta}^0$ be an estimate of $\boldsymbol{\theta}$ such that

$$\boldsymbol{\theta} = \boldsymbol{\theta}^0 + \delta\boldsymbol{\theta}, \tag{C.3}$$

where $\delta\boldsymbol{\theta}$ is the needed correction to the approximation. Linearizing the f_i about the current point given by $\boldsymbol{\theta}^0$ and $\mathbf{y}^{00}$,

$$f_i(\boldsymbol{\theta}; \mathbf{y}) \approx f_i(\boldsymbol{\theta}^0; \mathbf{y}^{00}) + \left[\frac{\partial f_i}{\partial \mathbf{y}}\right]^0 \cdot \mathbf{v} + \left[\frac{\partial f_i}{\partial \boldsymbol{\theta}}\right]^0 \cdot \delta\boldsymbol{\theta}, \tag{C.4}$$

where the vector of partial derivatives evaluated at current values is denoted by $\left[\frac{\partial f_i}{\partial \cdot}\right]^0$. We may thus replace the set of nonlinear equations with an approximate set of linear equations as

$$\mathbf{A}\mathbf{v} + \mathbf{B}\delta\boldsymbol{\theta} = \mathbf{c}, \tag{C.5}$$

where $\mathbf{A}$ is $m \times r$, and $\mathbf{B}$ is $m \times n$. Both of these matrices are composed of partial derivatives. The vector $\mathbf{c}$ is equal to the negative of the current value of the set of function in (C.1):

$$\mathbf{c} = - \begin{bmatrix} f_1(\boldsymbol{\theta}^0; \mathbf{y}^{00}) \\ \vdots \\ f_m(\boldsymbol{\theta}^0; \mathbf{y}^{00}) \end{bmatrix}. \tag{C.6}$$

A least squares solution to the set of equations (C.5) yields the most probable set of residuals $\mathbf{v}$ and a set of corrections $\delta\boldsymbol{\theta}$. It is easily shown that solution to this problem is given by

$$\mathbf{B}^T(\mathbf{A}\mathbf{W}^{-1}\mathbf{A}^T)^{-1}\mathbf{B}\delta\boldsymbol{\theta} = \mathbf{B}^T(\mathbf{A}\mathbf{W}^{-1}\mathbf{A}^T)^{-1}\mathbf{c}, \tag{C.7}$$

where $\mathbf{W}$ is an error weighting matrix. In fact, it is the Fisher information matrix (FIM) of the residuals $\mathbf{v}$.

We now apply this technique to the multiple exterior orientation problem, given in (2.99). We now assume that there are m photographs covering an area in which there are n feature points of interest. Also, let r of these points be control points, whose position have been somehow previously determined. For ease of development, we also assume that all n points appear in all m photographs. This assumption is easily relaxed by properly applying a weight matrix, as we shall see shortly.

Our goal is to determine all exterior orientation parameters $\mathbf{R}_i$ and $\mathbf{t}_i$ for $i = 1, \ldots, m$ from noisy measurements. We now have

$$l_{ij} \begin{bmatrix} x_{ij} \\ y_{ij} \\ f \end{bmatrix} = \mathbf{R}_i(\mathbf{a}_j + \mathbf{t}_i), \quad j = 1, \ldots, n, \quad i = 1, \ldots, m. \tag{C.8}$$

To eliminate the unknown projective parameters l_{ij}, for each image i we divide the first two rows by the third and rearrange, giving

$$\begin{aligned} f_x &\stackrel{\triangle}{=} x_{ij} - \tfrac{f[r_{11i}\ r_{12i}\ r_{13i}](\mathbf{a}_j+\mathbf{t}_i)}{[r_{31i}\ r_{32i}\ r_{33i}](\mathbf{a}_j+\mathbf{t}_i)} \approx 0, \\ f_y &\stackrel{\triangle}{=} y_{ij} - \tfrac{f[r_{21i}\ r_{22i}\ r_{23i}](\mathbf{a}_j+\mathbf{t}_i)}{[r_{31i}\ r_{32i}\ r_{33i}](\mathbf{a}_j+\mathbf{t}_i)} \approx 0, \end{aligned} \tag{C.9}$$

where

$$\begin{aligned} \mathbf{R}_i &= \begin{bmatrix} r_{11i} & r_{12i} & r_{13i} \\ r_{21i} & r_{22i} & r_{23i} \\ r_{31i} & r_{32i} & r_{33i} \end{bmatrix} \\ &= \begin{bmatrix} \cos\omega_i & \sin\omega_i & 0 \\ -\sin\omega_i & \cos\omega_i & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\phi_i & 0 & \sin\phi_i \\ 0 & 1 & 0 \\ -\sin\phi_i & 0 & \cos\phi_i \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\kappa_i & \sin\kappa_i \\ 0 & -\sin\kappa_i & \cos\kappa_i \end{bmatrix}. \end{aligned} \tag{C.10}$$

In (C.10), ω_i, ϕ_i, and κ_i are the Euler angles of the rotation of image i. Let $x_{ij} = x_{ij}^{00} + v_{xij}$ and $y_{ij} = y_{ij}^{00} + v_{yij}$, where x_{ij}^{00} and y_{ij}^{00} are the measured image coordinates and v_{xij} and v_{yij} are the least squares corrections needed to account for random errors in the measurements. Similarly, let

$$\begin{aligned} \omega_i &= \omega_i^0 + \delta\omega_i \\ \phi_i &= \phi_i^0 + \delta\phi_i \\ \kappa_i &= \kappa_i^0 + \delta\kappa_i \\ \mathbf{t}_i &= \mathbf{t}_i^0 + \delta\mathbf{t}_i \\ \mathbf{a}_j &= \mathbf{a}_j^0 + \delta\mathbf{a}_j, \end{aligned} \tag{C.11}$$

where the superscript "0" denotes a current estimate and the δ's are corrections to the estimate. With these definitions, we linearize (C.9):

$$\mathbf{v}_{ij} + \dot{\mathbf{B}}_{ij}\dot{\boldsymbol{\delta}}_i + \ddot{\mathbf{B}}_{ij}\ddot{\boldsymbol{\delta}}_j = \mathbf{c}_{ij}. \tag{C.12}$$

In (C.12), the superscript "one-dot" is used to denote corrections to the exterior orientation parameters, and "two-dots" is used to denote corrections to the 3-dimensional coordinates. We have

$$\dot{\boldsymbol{\delta}}_i = \begin{bmatrix} \delta\omega_i \\ \delta\phi_i \\ \delta\kappa_i \\ \delta\mathbf{t}_i \end{bmatrix}, \quad \ddot{\boldsymbol{\delta}}_j = \begin{bmatrix} \delta\mathbf{a}_j \end{bmatrix}, \tag{C.13}$$

$$\dot{\mathbf{B}}_{ij} = \begin{bmatrix} \frac{\partial f_x^0}{\partial\omega_i} & \frac{\partial f_x^0}{\partial\phi_i} & \frac{\partial f_x^0}{\partial\kappa_i} & \frac{\partial f_x^0}{\partial\mathbf{t}_i^T} \\ \\ \frac{\partial f_y^0}{\partial\omega_i} & \frac{\partial f_y^0}{\partial\phi_i} & \frac{\partial f_y^0}{\partial\kappa_i} & \frac{\partial f_y^0}{\partial\mathbf{t}_i^T} \end{bmatrix}, \quad \ddot{\mathbf{B}}_{ij} = \begin{bmatrix} \frac{\partial f_x^0}{\partial\mathbf{a}_i^T} \\ \\ \frac{\partial f_y^0}{\partial\mathbf{a}_i^T} \end{bmatrix}, \tag{C.14}$$

and finally,

$$\mathbf{v}_{ij} = \begin{bmatrix} v_{xij} \\ v_{yij} \end{bmatrix}, \quad \mathbf{c}_{ij} = -\begin{bmatrix} f_x^0 \\ f_y^0 \end{bmatrix}. \tag{C.15}$$

We now form the ground control equations. Let $\mathbf{a}_j = \mathbf{a}_j^{00} + \mathbf{v}_{aj}$, where $\mathbf{a}_j^{00}$ are the measured 3-dimensional locations, and $\mathbf{v}_{aj}$ are the least squares corrections needed to account for random errors in the measurements. From (C.11), we obtain

$$\mathbf{a}_j^{00} + \mathbf{v}_{aj} = \mathbf{a}_j^0 + \delta\mathbf{a}_j, \tag{C.16}$$

or

$$\mathbf{v}_{aj} - \delta\mathbf{a}_j = \mathbf{a}_j^0 - \mathbf{a}_j^{00}, \tag{C.17}$$

which is represented as

$$\ddot{\mathbf{v}}_j - \ddot{\boldsymbol{\delta}}_j = \ddot{\mathbf{c}}_j. \tag{C.18}$$

We now form the equations for the exterior orientation parameter updates. These can be derived in a manner similar to (C.18):

$$\begin{bmatrix} v_{\omega i} \\ v_{\phi i} \\ v_{\kappa i} \\ v_{\mathbf{t}i} \end{bmatrix} - \begin{bmatrix} \delta\omega_i \\ \delta\phi_i \\ \delta\kappa_i \\ \delta\mathbf{t}_i \end{bmatrix} = \begin{bmatrix} \omega_i^0 - \omega_i^{00} \\ \phi_i^0 - \phi_i^{00} \\ \kappa_i^0 - \kappa_i^{00} \\ \mathbf{t}_i^0 - \mathbf{t}_i^{00} \end{bmatrix}, \tag{C.19}$$

or simply

$$\dot{\mathbf{v}}_i - \dot{\boldsymbol{\delta}}_i = \dot{\mathbf{c}}_i. \tag{C.20}$$

We are now in a position to collect all the equations for all photographs and points. The complete set of collinearity equations (C.12) for point j on the m photographs is

$$\begin{bmatrix} \mathbf{v}_{1j} \\ \vdots \\ \mathbf{v}_{mj} \end{bmatrix} + \begin{bmatrix} \dot{\mathbf{B}}_{1j} & & \\ & \ddots & \\ & & \dot{\mathbf{B}}_{mj} \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{\delta}}_1 \\ \vdots \\ \dot{\boldsymbol{\delta}}_m \end{bmatrix} + \begin{bmatrix} \ddot{\mathbf{B}}_{1j} \\ \vdots \\ \ddot{\mathbf{B}}_{mj} \end{bmatrix} \ddot{\boldsymbol{\delta}}_j = \begin{bmatrix} \mathbf{c}_{1j} \\ \vdots \\ \mathbf{c}_{mj}, \end{bmatrix}, \tag{C.21}$$

which may be expressed simply as

$$\mathbf{v}_j + \dot{\mathbf{B}}_j\dot{\boldsymbol{\delta}} + \ddot{\mathbf{B}}_j\ddot{\boldsymbol{\delta}}_j = \mathbf{c}_j. \tag{C.22}$$

Since it is assumed that each point j is imaged in all m photographs, each point gives rise to a set of equations as above. Collecting all these for all n points:

$$\begin{bmatrix} \mathbf{v}_1 \\ \vdots \\ \mathbf{v}_n \end{bmatrix} + \begin{bmatrix} \dot{\mathbf{B}}_1 & & \\ & \ddots & \\ & & \dot{\mathbf{B}}_n \end{bmatrix} \dot{\boldsymbol{\delta}} + \begin{bmatrix} \ddot{\mathbf{B}}_1 \\ \vdots \\ \ddot{\mathbf{B}}_n \end{bmatrix} \begin{bmatrix} \ddot{\boldsymbol{\delta}}_1 \\ \vdots \\ \ddot{\boldsymbol{\delta}}_n \end{bmatrix} = \begin{bmatrix} \mathbf{c}_1 \\ \vdots \\ \mathbf{c}_n, \end{bmatrix}, \tag{C.23}$$

or finally

$$\mathbf{v} + \dot{\mathbf{B}}\dot{\boldsymbol{\delta}} + \ddot{\mathbf{B}}\ddot{\boldsymbol{\delta}} = \mathbf{c}. \tag{C.24}$$

Next, the observation equations for the exterior orientation parameters and for the known ground control points are collected. The complete set of exterior orientation equations can be stacked together as:

$$\begin{bmatrix} \dot{\mathbf{v}}_1 \\ \vdots \\ \dot{\mathbf{v}}_m \end{bmatrix} - \begin{bmatrix} \dot{\boldsymbol{\delta}}_1 \\ \vdots \\ \dot{\boldsymbol{\delta}}_m \end{bmatrix} = \begin{bmatrix} \dot{\mathbf{c}}_1 \\ \vdots \\ \dot{\mathbf{c}}_m \end{bmatrix}, \tag{C.25}$$

which is written as

$$\dot{\mathbf{v}} - \dot{\boldsymbol{\delta}} = \dot{\mathbf{c}}. \tag{C.26}$$

Similarly, the complete set of observation equations for the ground control points is

$$\begin{bmatrix} \ddot{\mathbf{v}}_1 \\ \vdots \\ \ddot{\mathbf{v}}_n \end{bmatrix} - \begin{bmatrix} \ddot{\boldsymbol{\delta}}_1 \\ \vdots \\ \ddot{\boldsymbol{\delta}}_n \end{bmatrix} = \begin{bmatrix} \ddot{\mathbf{c}}_1 \\ \vdots \\ \ddot{\mathbf{c}}_n, \end{bmatrix}, \tag{C.27}$$

which is written as

$$\ddot{\mathbf{v}} - \ddot{\boldsymbol{\delta}} = \ddot{\mathbf{c}}. \tag{C.28}$$

Combining (C.24), (C.26), and (C.28) gives a complete mathematical model of the update problem:

$$\begin{bmatrix} \mathbf{v} \\ \dot{\mathbf{v}} \\ \ddot{\mathbf{v}} \end{bmatrix} + \begin{bmatrix} \dot{\mathbf{B}} & \ddot{\mathbf{B}} \\ -\mathbf{I} & \mathbf{0} \\ \mathbf{0} & -\mathbf{I} \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{\delta}} \\ \ddot{\boldsymbol{\delta}} \end{bmatrix} = \begin{bmatrix} \mathbf{c} \\ \dot{\mathbf{c}} \\ \ddot{\mathbf{c}} \end{bmatrix}, \tag{C.29}$$

or

$$\bar{\mathbf{v}} + \bar{\mathbf{B}}\boldsymbol{\delta} = \bar{\mathbf{c}}. \tag{C.30}$$

It is a simple matter to show that the optimal least squares update $\boldsymbol{\delta}$ can be found from the normal equations

$$(\bar{\mathbf{B}}^T\bar{\mathbf{W}}\bar{\mathbf{B}})\boldsymbol{\delta} = \bar{\mathbf{B}}^T\bar{\mathbf{W}}\bar{\mathbf{c}}, \tag{C.31}$$

where $\bar{\mathbf{W}}$ is the FIM of the perturbations $\bar{\mathbf{v}}$. We construct $\bar{\mathbf{W}}$ as follows. Let $\mathbf{W}_{ij}$ be the FIM of the measurement (x_{ij}, y_{ij}). Stacking the weight matrices for all the image coordinates for point j gives

$$\mathbf{W}_j = \begin{bmatrix} \mathbf{W}_{1j} & & \\ & \ddots & \\ & & \mathbf{W}_{mj} \end{bmatrix}. \tag{C.32}$$

Next, stacking these equations for all n points gives

$$\mathbf{W} = \begin{bmatrix} \mathbf{W}_1 & & \\ & \ddots & \\ & & \mathbf{W}_n \end{bmatrix}. \tag{C.33}$$

Similarly, let the FIM of the exterior orientation parameter measurements be given by $\dot{\mathbf{W}}_i$. Assuming no correlation among the orientation parameters of different photographs, the complete set of weight matrices for all m photographs is

$$\dot{\mathbf{W}} = \begin{bmatrix} \dot{\mathbf{W}}_1 & & \\ & \ddots & \\ & & \dot{\mathbf{W}}_m \end{bmatrix}. \tag{C.34}$$

Similarly, let the FIM of the ground control point j measurements be given by $\ddot{\mathbf{W}}_j$. Again assuming no correlation among the different control points, the complete set of weight matrices for all ground coordinates is

$$\ddot{\mathbf{W}} = \begin{bmatrix} \ddot{\mathbf{W}}_1 & & \\ & \ddots & \\ & & \ddot{\mathbf{W}}_n \end{bmatrix}. \tag{C.35}$$

Finally, we have

$$\bar{\mathbf{W}} = \begin{bmatrix} \mathbf{W} & & \\ & \dot{\mathbf{W}} & \\ & & \ddot{\mathbf{W}} \end{bmatrix}. \tag{C.36}$$

The matrix $\mathbf{W}$ allows us to relax the assumption that every point is observed in every image. By setting the corresponding element of $\mathbf{W}$ to zero, we are informing the algorithm that we have no information about the point. Consequently, the "residual" in fitting to the point is not penalized. From an information theoretic viewpoint, this is exactly the correct procedure.

We mention finally that photogrammetrists use refined methods of solving (C.31), utilizing the sparsity and structure of the matrices. The ability to do this is partially a result of the pattern in which images are typically taken (generally a rectangular grid). By numbering the images in a raster order, banded, sparse matrices result, allowing for efficient computational procedures.

From an estimation viewpoint, this procedure is optimal in that least squares methods are applied to exactly the quantities that are subject to measurement errors. We therefore have a constrained maximum likelihood procedure.

Appendix D

Polynomial Coefficients for Resultant-Based Exterior Orientation

The expressions for the even b_i coefficients in Section 2.6.1, equation (2.109), are as follows:

$$b_0 = d_{12}^4 a_3^2 + 2a_7 d_{12}^2 a_3 + d_{12}^8 + 2d_{12}^6 a_3 + a_7^2 + 2a_7 d_{12}^4, \tag{D.1}$$

$$\begin{aligned} b_2 = \; & -d_{12}^6 a_1^2 + 2a_6 d_{12}^4 - 4a_7 d_{12}^2 - d_{12}^2 a_5^2 \\ & +2d_{12}^4 a_2 a_3 + 2a_6 a_7 - 4d_{12}^6 - 6d_{12}^4 a_5 c_{12} \\ & +2a_7 d_{12}^2 a_2 - 2a_7 a_3 - 2d_{12}^6 c_{12} a_1 \\ & +16a_7 c_{12}^2 d_{12}^2 - 2d_{12}^4 a_5 a_1 - 2d_{12}^2 a_5 a_3 c_{12} \\ & +2a_6 d_{12}^2 a_3 - 2a_3^2 d_{12}^2 + 4a_7 c_{12}^2 a_3 + 2a_7 c_{12} a_5 \\ & +4d_{12}^4 c_{12}^2 a_3 + 6a_7 c_{12} d_{12}^2 a_1 + 2d_{12}^6 a_2 \\ & -6d_{12}^4 a_3 + 2d_{12}^4 a_3 a_1 c_{12}, \end{aligned} \tag{D.2}$$

$$\begin{aligned} b_4 = \; & 4a_6 c_{12}^2 a_3 + 2a_6 c_{12} a_5 + 16a_6 c_{12}^2 d_{12}^2 \\ & +8d_{12}^4 + 4a_7 + 8d_{12}^2 a_3 - 2a_6 a_3 - 4a_6 d_{12}^2 + 16a_7 c_{12}^4 \\ & -6d_{12}^4 a_2 + 3d_{12}^4 a_1^2 + d_{12}^4 a_2^2 - 2a_7 a_2 \\ & +6a_6 c_{12} d_{12}^2 a_1 - 4a_3 a_1 d_{12}^2 c_{12} \\ & -2d_{12}^2 a_4 a_3 c_{12} - 2d_{12}^2 a_5 a_2 c_{12} \\ & -4d_{12}^2 a_5 c_{12}^2 a_1 + 2d_{12}^4 a_2 a_1 c_{12} \\ & +2a_7 c_{12} a_4 - 6a_7 c_{12} a_1 + 4a_7 c_{12}^2 a_2 + 8a_7 c_{12}^3 a_1 \\ & +12a_5 d_{12}^2 c_{12} + 4a_5 d_{12}^2 a_1 + 2a_5 a_3 c_{12} \\ & +6d_{12}^4 c_{12} a_1 - 8d_{12}^2 c_{12}^2 a_3 - 2d_{12}^2 a_4 a_5 \\ & -6d_{12}^4 a_4 c_{12} - 2d_{12}^4 a_4 a_1 - 8d_{12}^2 a_5 c_{12}^3 \\ & +4d_{12}^4 c_{12}^2 a_2 + 2a_6 d_{12}^2 a_2 - 4a_2 d_{12}^2 a_3 \\ & +a_5^2 + a_6^2 + a_3^2 - 16a_7 c_{12}^2, \end{aligned} \tag{D.3}$$

$$\begin{aligned} b_6 = \; & 8a_6 c_{12}^3 a_1 - 6a_6 c_{12} a_1 + 4a_6 c_{12}^2 a_2 \\ & +2a_1 a_3 c_{12} + 2a_6 c_{12} a_4 - 4a_3 - 8d_{12}^2 + 16d_{12}^2 c_{12}^2 \\ & +4a_6 - 2a_1 a_5 + 8d_{12}^2 a_2 - 2a_6 a_2 - 3a_1^2 d_{12}^2 - 16a_6 c_{12}^2 \\ & +2a_4 a_5 - 4a_5 c_{12} + 8a_5 c_{12}^3 - d_{12}^2 a_4^2 - 2a_2^2 d_{12}^2 \\ & +2a_2 a_3 + 8a_3 c_{12}^2 - 4a_2 a_1 d_{12}^2 c_{12} \\ & -2d_{12}^2 a_4 a_2 c_{12} - 4d_{12}^2 a_4 c_{12}^2 a_1 \\ & +12a_4 d_{12}^2 c_{12} + 4a_4 d_{12}^2 a_1 + 2a_4 a_3 c_{12} \\ & +2a_5 a_2 c_{12} + 4a_5 c_{12}^2 a_1 - 8d_{12}^2 c_{12}^2 a_2 \\ & -8d_{12}^2 a_4 c_{12}^3 + 16a_6 c_{12}^4, \end{aligned} \tag{D.4}$$

$$
\begin{aligned}
b_8 = \quad & 2a_4a_2c_{12} + 16c_{12}^4 + 4a_4c_{12}^2a_1 + a_1^2 \\
& + a_4^2 - 2a_1a_4 + 8a_4c_{12}^3 + 4 - 4a_2 + 2a_1a_2c_{12} \\
& - 4a_1c_{12} + a_2^2 + 8a_1c_{12}^3 - 4a_4c_{12} + 8a_2c_{12}^2 - 16c_{12}^2.
\end{aligned} \tag{D.5}
$$

Appendix E

A Duality Approach to Minimizing a Quadratic Form with Quadratic Constraints

Here we consider a duality approach to the minimization of a quadratic form, subject to quadratic constraints. This problem arises is several machine vision tasks, including relative orientation and scene reconstruction.

Given symmetric matrices $\mathbf{A}$, $\mathbf{B}_1, \ldots, \mathbf{B}_n$, we desire the solution vector $\mathbf{x}$ such that

$$\min_{\mathbf{x}} \ \mathbf{x}^T \mathbf{A} \mathbf{x}, \tag{E.1}$$

subject to the constraints

$$\mathbf{x}^T \mathbf{B}_i \mathbf{x} = c_i, \quad i = 1, \ldots, n. \tag{E.2}$$

Let $\mathbf{x}^*$ denote the optimal feasible value of $\mathbf{x}$. We will assume that $\mathbf{A}$ is positive definite, and that at least one of the $\mathbf{B}_i$'s (say, $\mathbf{B}_1$) is positive definite. Without loss of generality, we can therefore assume that $\mathbf{B}_1$ is the identity matrix (by performing a change of variables, if necessary). We further assume that each c_i is a nonnegative scalar. We assume that at least one feasible point exists. This problem has the same solution as:

$$\min_{\mathbf{x}} \ \mathbf{x}^T \mathbf{A} \mathbf{x} + \sum_{i=1}^{n} \lambda_i \left(\mathbf{x}^T \mathbf{B}_i \mathbf{x} - c_i \right), \tag{E.3}$$

subject to

$$\mathbf{x}^T \mathbf{B}_i \mathbf{x} = c_i, \quad i = 1, \ldots, n. \tag{E.4}$$

Here, the scalars λ_i are Lagrange multipliers, which we collect into a vector $\boldsymbol{\lambda}$. If we relax some constraints in (E.4), then we lower bound the optimal value of (E.3), because we are enlarging the feasible region. To get the tightest lower bound, we thus choose the λ_i's to maximize the lower bound:

$$\max_{\boldsymbol{\lambda}} \min_{\mathbf{x}} \ \mathbf{x}^T \mathbf{A} \mathbf{x} + \sum_{i=1}^{n} \lambda_i \left(\mathbf{x}^T \mathbf{B}_i \mathbf{x} - c_i \right), \tag{E.5}$$

subject to

$$\mathbf{x}^T \mathbf{B}_i \mathbf{x} = c_i, \quad i = 1, \ldots, m, \tag{E.6}$$

with $m < n$. For our purposes, it will be convenient to take $m = 1$, giving the restriction

$$\|\mathbf{x}\|^2 = \mathbf{x}^T \mathbf{x} = c_1. \tag{E.7}$$

In order to use (E.5), we first choose a fixed setting $\boldsymbol{\lambda}$, and then find $\mathbf{x}$ that minimizes (E.3). The goal then is to find the $\boldsymbol{\lambda}$ that makes the ensuing minimization over $\mathbf{x}$ as "poor" as possible.

There are several properties of this dual formulation that make it particularly attractive. First, the dual objective function $J(\boldsymbol{\lambda})$ is bounded from above by all the feasible solutions. Next, for a given $\boldsymbol{\lambda}$, the optimal $\mathbf{x}$ is easily found. Rearranging (E.3) gives

$$J(\boldsymbol{\lambda}) = \mathbf{x}^T \left(\mathbf{A} + \sum_{i=1}^{n} \lambda_i \mathbf{B}_i \right) \mathbf{x} - \sum_{i=1}^{n} \lambda_i c_i. \tag{E.8}$$

To minimize $J(\boldsymbol{\lambda})$ in (E.8) with respect to $\mathbf{x}$, subject to (E.7), we simply find the eigenvector $\mathbf{v}$ corresponding to the minimum eigenvalue of

$$\mathbf{A} + \sum_{i=1}^{n} \lambda_i \mathbf{B}_i. \tag{E.9}$$

We then set $\mathbf{x}$ equal to $\sqrt{c_1}\mathbf{v}$, in order to satisfy (E.7).

The next useful property of the dual formulation is that the dual objective function is concave. This is implies that any local maximum is also the global maximum. To show concavity, assume we have evaluated (E.3) at two locations, $\boldsymbol{\lambda}$ and $\boldsymbol{\lambda}'$. For any intermediate point given by $\theta\boldsymbol{\lambda} + (1-\theta)\boldsymbol{\lambda}'$, with $0 \leq \theta \leq 1$,

$$\begin{array}{rl} J(\theta\boldsymbol{\lambda} + (1-\theta)\boldsymbol{\lambda}') & = \min_{\mathbf{x}} \mathbf{x}^T\mathbf{A}\mathbf{x} + \sum_{i=1}^{n} \left(\theta\lambda_i + (1-\theta)\lambda_i'\right) \left(\mathbf{x}^T\mathbf{B}_i\mathbf{x} - c_i\right), \\ & = \min_{\mathbf{x}} \theta \left(\mathbf{x}^T\mathbf{A}\mathbf{x} + \sum_{i=1}^{n} \lambda_i \left(\mathbf{x}^T\mathbf{B}_i\mathbf{x} - c_i\right)\right) \\ & \quad + (1-\theta) \left(\mathbf{x}^T\mathbf{A}\mathbf{x} + \sum_{i=1}^{n} \lambda_i' \left(\mathbf{x}^T\mathbf{B}_i\mathbf{x} - c_i\right)\right), \\ & \geq \theta J(\boldsymbol{\lambda}) + (1-\theta) J(\boldsymbol{\lambda}') \end{array} . \tag{E.10}$$

The inequality in the last line follows because the minimizing $\mathbf{x}$'s for $J(\boldsymbol{\lambda})$ and $J(\boldsymbol{\lambda}')$ are free to be chosen separately. Thus, (E.10) demonstrates that the dual objective function is concave.

Another useful property of the dual approach is that the set of λ_i's that we must consider is convex, and that we can iteratively shrink the search region for the optimal λ_i's while keeping the set convex. To show convexity, assume we have two fixed settings of $\boldsymbol{\lambda}'$ and $\boldsymbol{\lambda}''$ such that

$$\begin{array}{rcccl} \min_{\mathbf{x}'} \theta \mathbf{x}'^T \left(\mathbf{A} + \sum_{i=1}^{n} \lambda_i' \mathbf{B}_i\right) \mathbf{x}' & \geq & \theta c & \geq & 0 \text{ and} \\ \min_{\mathbf{x}''} (1-\theta) \mathbf{x}''^T \left(\mathbf{A} + \sum_{i=1}^{n} \lambda_i'' \mathbf{B}_i\right) \mathbf{x}'' & \geq & (1-\theta)c & \geq & 0, \end{array} \tag{E.11}$$

with $0 \leq \theta \leq 1$. Then, for any other $\mathbf{x}$, we have

$$\begin{array}{rcl} \theta \mathbf{x}^T \left(\mathbf{A} + \sum_{i=1}^{n} \lambda_i' \mathbf{B}_i\right) \mathbf{x} & \geq & \min_{\mathbf{x}'} \theta \mathbf{x}'^T \left(\mathbf{A} + \sum_{i=1}^{n} \lambda_i' \mathbf{B}_i\right) \mathbf{x}' \text{ and} \\ (1-\theta) \mathbf{x}^T \left(\mathbf{A} + \sum_{i=1}^{n} \lambda_i'' \mathbf{B}_i\right) \mathbf{x} & \geq & \min_{\mathbf{x}''} (1-\theta) \mathbf{x}''^T \left(\mathbf{A} + \sum_{i=1}^{n} \lambda_i'' \mathbf{B}_i\right) \mathbf{x}''. \end{array} \tag{E.12}$$

Adding these equations and using (E.11), we obtain

$$\mathbf{x}^T \left(\mathbf{A} + \sum_{i=1}^{n} (\theta\lambda_i' + (1-\theta)\lambda_i'') \mathbf{B}_i \right) \mathbf{x} \geq c. \tag{E.13}$$

Since this is true for any $\mathbf{x}$, it is true in particular for the $\mathbf{x}$ that minimizes the left-hand side of (E.13), so that

$$\min_{\mathbf{x}} \mathbf{x}^T \left(\mathbf{A} + \sum_{i=1}^{n} (\theta \lambda_i' + (1-\theta)\lambda_i'') \mathbf{B}_i \right) \mathbf{x} \geq c, \tag{E.14}$$

which shows that the set is convex. If we are at some $\boldsymbol{\lambda}$, with objective value c, then we can simply perform a line search in the direction of increasing c. At the maximum objective value on this line, we pick a new search direction and repeat the search.

Bibliography

[1] T. J. Abatzoglou and J. M. Mendel. Constrained total least squares. In *Proc. IEEE Intl. Conf. Acoust., Speech, Signal Processing (ICASSP)*, pages 1485–1488, 1987.

[2] T. J. Abatzoglou, J. M. Mendel, and G. A. Harada. The constrained total least squares technique and its applications to harmonic superresolution. *IEEE Trans. Signal Processing*, 39(5):1070–1087, May 1991.

[3] R. Ablamowicz, P. Lounesto, and J. M. Parra, editors. *Clifford Algebras with Numeric and Symbolic Computations*. Birkhauser Boston, 1996.

[4] J. K. Aggarwal and N. Nandhakumar. On the computation of motion from sequences of images - a review. *Proceedings of the IEEE*, 76(8), August 1988.

[5] P. M. Q. Aguiar and J. M. F. Moura. Robust 3D structure from motion under ortography. In *Proc. IEEE Image and Multidimensional Digital Signal Processing (IMDSP)*, pages 171–174, 1998.

[6] P. M. Q. Aguiar and J. M. F. Moura. Factorization as a rank 1 problem. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 178–184, 1999.

[7] I. A. Ajwa, Z. Liu, and P. S. Wang. Grobner bases algorithm. Technical Report ICM-9502-00, Inst. for Comp. Math., Kent State University, February 1995. http://symbolicnet.mcs.kent.edu/icm/reports/index.html.

[8] N. Ancona and T. Poggio. Optical flow from 1D correlation: Application to a simple time-to-crash detector. Technical Report AIM-1375, MIT Artificial Intelligence Laboratory, October 1993.

[9] M. Arjmand and R. A. Roberts. On comparing hardware implementations of fixed point digital filters. *IEEE Circuits and Systems Magazine*, 3(2), 1981.

[10] K. S. Arun, T. S. Huang, and S. D. Blostein. Least-squares fitting of two 3-D point sets. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 9(5), September 1987.

[11] S. Avidan and A. Shashua. Novel view synthesis by cascading trilinear tensors. *IEEE Trans. on Visualization and Computer Graphics*, 4(4):293–306, 1999.

[12] A. Azarbayejani and A. Pentland. Recursive estimation of motion, structure, and focal length. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 17(6), June 1995.

[13] H. S. Baird. *Model-Based Image Matching Using Location*. MIT Press, 1984.

[14] C. W. Barnes, B. N. Tran, and S. H. Leung. On the statistics of fixed point roundoff error. *IEEE Trans. Acoustics, Speech, and Signal Proc.*, 33(3), June 1985.

[15] E. Beltrami. Sulle funzioni bilineari. *Giornale di Mathematiche*, 11:98–106, 1873.

[16] E. Beltrami. On bilinear functions. In M. Moonen and B. De Moor, editors, *SVD and Signal Processing III*, pages 9–18. Elsevier Science, 1995.

[17] A. Benedetti and P. Perona. Real-time 2-D feature detection on a reconfigurable computer. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 1998.

[18] J. R. Bergendahl. A computationally efficient stereo vision algorithm for adaptive cruise control. Master's thesis, Massachusetts Institute of Technology, June 1997.

[19] M. Bertozzi and A. Broggi. Vision-based vehicle guidance. *IEEE Computer*, pages 49–55, July 1997.

[20] D. Bertsekas. *Nonlinear Programming*. Athena Scientific, 1995.

[21] D. Bertsimas and J. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, 1997.

[22] R. E. Blahut. *Fast Algorithms for Digital Signal Processing*. Addison-Wesley, 1985.

[23] G. F. Boudreaux-Bartels and T. W. Parks. Discrete fourier transform using summation by parts. In *Proc. IEEE Intl. Conf. Acoust., Speech, Signal Processing (ICASSP)*, pages 1827–1830, 1987.

[24] B. Box. Field programmable gate array based reconfigurable preprocessor. In *IEEE Workshop on FPGAs for Custom Computing Machines (FCCM)*, April 1994.

[25] R. Braham. Technology 1999: Analysis and forecast. *IEEE Spectrum*, pages 73–78, January 1999.

[26] J. W. Brewer. Kronecker products and matrix calculus in system theory. *IEEE Trans. Circuits and Systems*, 25(9), 1978.

[27] W. L. Brogan. *Modern Control Theory*. Prentice-Hall, 3rd edition, 1991.

[28] T. J. Broida, S. Chandrashekhar, and R. Chellappa. Recursive 3-D motion estimation from a monocular sequence. *IEEE Trans. Aerospace and Electronic Systems*, 26(4), July 1990.

[29] L. G. Brown. A survey of image registration techniques. *ACM Computing Surveys*, 24(4), December 1992.

[30] S. Chaudhuri and S. Chatterjee. Performance analysis of total least squares methods in three-dimensional motion estimation. *IEEE Trans. on Robotics and Automation*, 7(5), October 1991.

[31] S. Chaudhuri, S. Sharma, and S. Chatterjee. Recursive estimation of motion parameters. *Computer Vision and Image Understanding*, 64(3), November 1996.

[32] H. Choi and W. P. Burleson. Search-based wordlength optimization for VLSI/DSP synthesis. In *VLSI Signal Processing*, 1994.

[33] P. Comon and B. Mourrain. Decomposition of quantics in sums of powers of linear forms. *Signal Processing*, 53:93–107, 1997.

[34] S. R. Coorg. *Pose Imagery and Automated Three-Dimensional Modeling of Urban Environments*. PhD thesis, Massachusetts Institute of Technology, August 1998.

[35] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. McGraw-Hill, 1990.

[36] D. Cox, J. Little, and D. O'Shea. *Using Algebraic Geometry*. Number 185 in Graduate Texts in Mathematics. Springer-Verlag, 1998.

[37] G. Csurka et al. Characterizing the uncertainty of the fundamental matrix. *Computer Vision and Image Understanding*, 68(1), October 1997.

[38] L. Dadda. Some schemes for parallel multipliers. *Alta Frequenza*, 34:349–356, May 1965.

[39] W. R. Davis et al. Micro air vehicles for optical surveillance. *The Lincoln Laboratory Journal*, 9(2), 1996.

[40] L. De Lathauwer. *Signal Processing Based on Multilinear Algebra*. PhD thesis, K. U. Leuven, 1997.

[41] H. De Man et al. Architecture-driven synthesis techniques for VLSI implementation of DSP algorithms. *Proceedings of the IEEE*, pages 319–334, February 1990.

[42] B. De Moor. Total least squares for affinely structured matrices and the noisy realization problem. *IEEE Trans. Signal Processing*, 42(11):3104–3113, November 1994.

[43] R. De Mori, S. Rivoira, and A. Serra. A special purpose computer for digital signal processing. *IEEE Trans. on Computers*, pages 1202–1211, December 1975.

[44] K. I. Diamantaras et al. Total least squares 3-D motion estimation. In *Proc. IEEE Intl. Conf. on Image Processing (ICIP)*, 1998.

[45] B. Dipert. Motion capture systems for animation. *EDN*, pages 59–66, December 1999. http://www.ednmag.com.

[46] I. J. Dowman. Automating image registration and absolute orientation: Solutions and problems. *The Photogrammetric Record*, 16(91), April 1998.

[47] L. G. Dron. The multi-scale veto model: A two-stage analog network for edge detection and image reconstruction. Technical Report AIM-1320, MIT Artificial Intelligence Laboratory, March 1992.

[48] L. G. Dron. *Computing 3-D Motion in Custom Analog and Digital VLSI*. PhD thesis, Massachusetts Institute of Technology, August 1994.

[49] G. Eckart and G. Young. The approximation of one matrix by another of lower rank. *Pyschometrika*, 1:211–218, 1936.

[50] M. Elad, P. Teo, and Y. Hel-Or. Optimal filters for gradient-based motion estimation. In *Proc. Intl. Conf. on Computer Vision (ICCV)*, pages 559–565, 1999.

[51] O. D. Faugeras. *Three-Dimensional Computer Vision*. The MIT Press, 1993.

[52] O. D. Faugeras et al. 3-D reconstruction of urban scenes from image sequences. *Computer Vision and Image Understanding*, 69(3), March 1998.

[53] P. D. Fiore. Image registration using both distance and angle information. In *Proc. IEEE Intl. Conf. on Image Processing (ICIP)*, volume 3, pages 220–223, 1995.

[54] P. D. Fiore. Lazy rounding. In *Proc. IEEE Workshop on Signal Processing Systems (SiPS)*, October 1998.

[55] P. D. Fiore. Low complexity implementation of a polyphase filter bank. *Digital Signal Processing*, 8(2):126–135, April 1998.

[56] P. D. Fiore. Optimal wordlength DSP designs via Markov chains. *MIT 18.416 Term Paper*, December 1998.

[57] P. D. Fiore. Parallel multiplication using fast sorting networks. *IEEE Trans. on Computers*, 48(6), June 1999.

[58] P. D. Fiore. Efficient wordlength reduction for DSP applications. *The Journal of VLSI Signal Processing*, 24(1):9–18, February 2000. Invited Paper.

[59] P. D. Fiore et al. Efficient feature tracking with application to camera motion estimation. In *Proc. IEEE Asilomar Conf. on Signals, Systems, and Computers*, November 1998.

[60] P. D. Fiore and S. W. Lang. Efficient phase-only frequency estimation. In *Proc. IEEE Intl. Conf. Acoust., Speech, Signal Processing (ICASSP)*, volume 5, pages 2809–2812, 1996.

[61] P. D. Fiore and L. Lee. Closed-form and real-time wordlength adaptation. In *Proc. IEEE Intl. Conf. Acoust., Speech, Signal Processing (ICASSP)*, 1999.

[62] P. D. Fiore and P. N. Topiwala. Bit-ordered tree classifiers for SAR target classification. In *Proc. IEEE Asilomar Conf. on Signals, Systems, and Computers*, 1997.

[63] P. D. Fiore and G. C. Verghese. Constrained maximum likelihood solution of linear equations. *IEEE Trans. Signal Processing*, 48(3):671–679, March 2000.

[64] IEEE Standard for Binary Floating-Point Arithmetic (IEEE STD 754-1985). Institute of Electrical and Electronic Engineers, Inc. 1985.

[65] S. E. Frumkin. Time to collision algorithm: Properties and real time implementation. Master's thesis, Massachusetts Institute of Technology, May 1997.

[66] T. Fukushige, P. Hut, and J. Makino. High-performance special-purpose computers in science. *Computing in Science and Engineering*, 1(2), March 1999.

[67] S. K. Ghosh. *Analytic Photogrammetry*. Pergamon Press, 2nd edition, 1988.

[68] G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns-Hopkins University Press, 3rd edition, 1996.

[69] D. Goryn and S. Hein. On the estimation of rigid body rotation from noisy data. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 17(12), December 1995.

[70] G. R. Goslin. A guide to using field programmable gate arrays (FPGAs) for application-specific digital signal processing performance. *http://www.xilinx.com*, 1995.

[71] N. Gracias and J. Santos-Victor. Robust estimation of the fundamental matrix and stereo correspondence. In *Proc. 5^{th} Intl. Symp. on Intell. Robotic Systs.*, pages 295–304, 1997.

[72] Alta Group. *Fixed-Point Optimizer User's Guide*. Cadence Design Systems, Inc., 1994.

[73] J. J. Guerrero and C. Sagues. Direct method to obtain straight edge depth from motion. *Optical Engineering*, 37(7):2124–2132, July 1998.

[74] N. Gupta and L. Kanal. Gradient based image motion estimation without computing gradients. *International Journal of Computer Vision*, 122(1):81–101, 1998.

[75] R. M. Haralick et al. Pose estimation from corresponding point data. *IEEE Trans. Systems, Man, and Cybernetics*, 19(6), November 1989.

[76] R. M. Haralick et al. Analysis and solutions of the three point perspective pose estimation problem. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 592–598, 1991.

[77] R. M. Haralick and L. G. Shapiro. *Computer and Robot Vision*. Addison-Wesley, 1993.

[78] R. I. Hartley. In defense of the eight-point algorithm. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 19(6), June 1997.

[79] R. I. Hartley. Minimizing algebraic error in geometric estimation problems. In *Proc. Intl. Conf. on Computer Vision (ICCV)*, pages 469–476, 1998.

[80] S. Hauck. The roles of FPGA's in reprogrammable systems. *Proceedings of the IEEE*, 86(4), April 1998.

[81] S. Haykin. *Adaptive Filter Theory*. Prentice-Hall, 2nd edition, 1991.

[82] J. Heel. Dynamical systems and motion vision. Technical Report AIM-1037, MIT Artificial Intelligence Laboratory, April 1988.

[83] J. Heel. Direct estimation of structure and motion from multiple frames. Technical Report AIM-1190, MIT Artificial Intelligence Laboratory, March 1990.

[84] Y. Hel-Or and M. Werman. Absolute orientation from uncertain point data: A unified approach. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 77–82, 1992.

[85] Y. Hel-Or and M. Werman. Pose estimation by fusing noisy data of different dimensions. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 17(2), February 1995.

[86] A. Heyden. *Geometry and Algebra of Multiple Projective Transformations*. PhD thesis, Lund Institute of Technology, Sweden, December 1995.

[87] A. Hill, T. F. Cootes, and C. J. Taylor. Least-squares solution of absolute orientation with non-scalar weights. In *Proc. Intl. Conf. on Pattern Recognition (ICPR)*, pages 461–465, 1996.

[88] B. K. P. Horn. *Robot Vision*. The MIT Press, 1986.

[89] B. K. P. Horn. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America, A*, 4:629–642, April 1987.

[90] B. K. P. Horn. Recovering baseline and orientation from essential matrix. *Journal of the Optical Society of America*, January 1990.

[91] B. K. P. Horn. Relative orientation revisited. *Journal of the Optical Society of America*, 8:1630–1638, October 1991.

[92] B. K. P. Horn, H. M. Hilden, and S. Negahdaripour. Closed-form solution of absolute orientation using orthonormal matrices. *Journal of the Optical Society of America*, 5:1127–1135, July 1988.

[93] B. K. P. Horn and B. G. Schunck. Determining optical flow. *Artificial Intelligence*, 16(1):185–203, August 1981.

[94] B. K. P. Horn and E. J. Weldon Jr. Direct methods for recovering motion. *International Journal of Computer Vision*, 2:51–76, February 1988.

[95] Y. H. Hu. CORDIC-based VLSI architectures for digital signal processing. *IEEE Signal Processing Magazine*, July 1992.

[96] K. Hwang. *Computer Arithmetic: Principles, Architecture, and Design*. Wiley, 1979.

[97] T. Jebara, A. Azarbayejani, and A. Pentland. 3D structure from 2D motion. *IEEE ASSP Magazine*, 16(3), May 1999.

[98] C. Jerian and R. Jain. Polynomial methods for structure from motion. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 12(12), December 1990.

[99] Q. Ji et al. An integrated linear technique for pose estimation from different geometric features. *International Journal of Pattern Recognition and Artificial Intelligence*, 13(5), 1999.

[100] K. Kanatani. *Group-Theoretical Methods in Image Understanding*. Springer-Verlag, 1990.

[101] K. Kanatani. *Introduction to Statistical Optimization for Geometric Computation*. http://www.ail.cs.gunma-u.ac.jp/ kanatani/e, 1997.

[102] A. Kavalade and E. A. Lee. The extended partitioning problem: Hardware/software mapping and implementation-bin selection. In *Proc. IEEE Intl. Workshop on Rapid System Prototyping (RSP)*, pages 12–18, 1995.

[103] S. M. Kay. *Fundamentals of Statistical Signal Processing: Estimation Theory.* Prentice-Hall, 1993.

[104] S. Kim, K.-I. Kum, and W. Sung. Fixed-point optimization utility for C and C++ based digital signal processing programs. *IEEE Trans. Circuits and Systems - II: Analog and Digital Signal Processing*, 45(11), November 1998.

[105] E. J. King and E. E. Swartzlander. Data-dependent truncation scheme for parallel multipliers. In *Proc. IEEE Asilomar Conf. on Signals, Systems, and Computers*, 1997.

[106] S. K. Knapp. Using programmable logic to accelerate DSP functions. *http://www.xilinx.com*, 1995.

[107] D. Kottke and P. D. Fiore. Systolic array for acceleration of template based ATR. In *Proc. IEEE Intl. Conf. on Image Processing (ICIP)*, 1997.

[108] K.-I. Kum and W. Sung. Word-length optimization for high-level synthesis of digital signal processing systems. In *Proc. IEEE Workshop on Signal Processing Systems (SiPS)*, pages 569–578, 1998.

[109] F. J. Kurdahi, N. Bagherzadeh, P. Athanas, and J. L. Munoz. Guest editors' introduction: Configurable computing. *IEEE Design and Test of Computers*, pages 17–19, January 2000.

[110] J. Lasenby et al. A new framework for the formation of invariants and multiple-view constraints in computer vision. In *Proc. IEEE Intl. Conf. on Image Processing (ICIP)*, 1996.

[111] J. Lasenby et al. New geometric methods for computer vision. *International Journal of Computer Vision*, 36(3), 1998.

[112] S. Lee and Y. Kay. A Kalman filter approach for accurate 3-D motion estimation from a sequence of stereo images. *Computer Vision, Graphics, and Image Processing: Image Understanding*, 54(2), September 1991.

[113] P. Lemmerling, B. De Moor, and S. Van Huffel. On the equivalence of constrained total least squares and structured total least squares. *IEEE Trans. Signal Processing*, 44(11):2908–2911, November 1996.

[114] S. E. Leurgans, R. T. Ross, and R. B. Abel. A decomposition for three-way arrays. *SIAM Journal on Matrix Analysis and Applications*, 14(4):1064–1083, October 1993.

[115] R. Lidl and G. Pilz. *Applied Abstract Algebra.* Springer-Verlag, 1998.

[116] Y. C. Lim. Single-precision multiplier with reduced circuit complexity for signal processing applications. *IEEE Trans. on Computers*, 41(10), October 1992.

[117] H. C. Longuet-Higgins. A computer algorithm for reconstructing a scene from two projections. *Nature*, 293:133–135, September 1981.

[118] P. Lounesto, editor. *Clifford Algebras and Spinors.* Cambridge University Press, 1997.

[119] M. R. Luettgen, W. C. Karl, and A. S. Willsky. Efficient multiscale regularization with applications to the computation of optical flow. *IEEE Trans. Image Processing*, 3(1), January 1994.

[120] Q.-T. Luong and O. D. Faugeras. The fundamental matrix: Theory, algorithms, and stability analysis. *International Journal of Computer Vision*, 17:43–75, 1996.

[121] G.-K. Ma and F. J. Taylor. Multiplier policies for digital signal processing. *IEEE ASSP Magazine*, January 1990.

[122] N. Mackey. Hamilton and Jacobi meet again: Quaternions and the eigenvalue problem. *SIAM J. Matrix Anal. Appl.*, 16(2):421–435, April 1995.

[123] W. H. Mangione-Smith et al. Seeking solutions in configurable computing. *Computer*, December 1997.

[124] A. Marugame, J. Katto, and M. Ohta. Structure recovery with multiple cameras from scaled orthographic and perspective views. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 21(7), July 1999.

[125] L. G. McIlrath and P. M. Zavracky. An architecture for low-power real time image analysis using 3D silicon technology. In *SPIE*, volume 3362-19, pages 184–195, April 1999.

[126] I. Mcquirk. An analog VLSI chip for estimating the focus of expansion. Technical Report AIM-1577, MIT Artificial Intelligence Laboratory, June 1996.

[127] I. Mcquirk, B. K. P. Horn, H.-S. Lee, and J. L. Wyatt Jr. Estimating the focus of expansion in analog VLSI. *International Journal of Computer Vision*, 28(3):261–277, July 1998.

[128] P. Meer, C. V. Stewart, and D. E. Tyler. Robust computer vision: An interdisciplinary challenge. *Computer Vision and Image Understanding*, 78(1):1–7, 2000.

[129] J. P. Mellor, S. Teller, and T. Lozano-Perez. Dense depth maps from epipolar images. Technical Report AIM-1593, MIT Artificial Intelligence Laboratory, November 1996.

[130] L. Mintzer. Large FFTs in a single FPGA. In *Intl. Conf. on Signal Proc. Applications and Technology (ICSPAT)*, pages 895–899, October 1996.

[131] E. Mirsky and A. DeHon. Matrix: A reconfigurable computing architecture with configurable instruction distribution and deployable resources. In *IEEE Workshop on FPGAs for Custom Computing Machines (FCCM)*, April 1996.

[132] A. Morgado and I. Dowman. A procedure for automatic absolute orientation using aerial photographs and a map. *ISPRS Journal of Photogrammetry and Remote Sensing*, 52(4):169–182, August 1997.

[133] A. Morgan. *Solving Polynomial Systems Using Continuation for Engineering and Scientific Problems*. Prentice-Hall, 1987.

[134] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1997.

[135] S. Negahdaripour. *Direct Methods for Structure from Motion*. PhD thesis, Massachusetts Institute of Technology, September 1986.

[136] S. Negahdaripour and B. K. P. Horn. Direct passive navigation. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 9(1), January 1987.

[137] L. Ng and V. Solo. A data-driven method for choosing smoothing parameters in optical flow problems. In *Proc. IEEE Intl. Conf. on Image Processing (ICIP)*, 1997.

[138] DARPA Electronics Technology Office. Smart Modules Program. *http://www.darpa.mil /eto/smartmod/index.html*, 1999.

[139] N. Ohta and K. Kanatani. Optimal estimation of three-dimensional rotation and reliability evaluation. *IEICE Trans. Information and Systems*, E82-D(11), November 1998.

[140] V. G. Oklobdzija, D. Villeger, and S. S. Liu. A method for speed optimized partial product reduction and generation of fast parallel multipliers using an algorithmic approach. *IEEE Trans. on Computers*, 45(3), March 1996.

[141] M. Okutomi and T. Kanade. A multiple-baseline stereo. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 15(4), April 1993.

[142] A. V. Oppenheim and R. W. Schafer. *Digital Signal Processing*. Prentice-Hall, 1975.

[143] J. M. Ortolf. Roving eye - research for imaging surveillance. In *Proc. SPIE Conf. on Sensors, C3I, Infor. and Training Technologies for Law Enforcement*, volume 3577-01, pages 2–13, November 1998.

[144] H. Pan. A direct closed-form solution to general relative orientation of two stereo views. *Digital Signal Processing*, 9:195–221, 1999.

[145] X. Pan and D. M. Lane. Pose determination from angles and relative line lengths using spherical trigonometry. *Image and Vision Computing*, 17:937–953, 1999.

[146] T. Papadimitriou et al. Robust estimation of rigid body 3-D motion parameters from point correspondences. In *Proc. IEEE Intl. Conf. Acoust., Speech, Signal Processing (ICASSP)*, 1999.

[147] A. Papoulis. *Probability, Random Variables, and Stochastic Processes*. McGraw-Hill, 1984.

[148] J. Philip. A non-iterative algorithm for determining all essential matrices corresponding to five point pairs. *The Photogrammetric Record*, 15(88), October 1996.

[149] M. Pilu. A direct method for stereo correspondence based on singular value decomposition. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 261–266, 1997.

[150] V. V. Prasolov. *Problems and Theorems in Linear Algebra*. American Mathematical Society, 1994.

[151] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, 1986.

[152] L. Quan and Z. Lan. Linear N-point camera pose determination. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 21(8):774–780, August 1999.

[153] N. K. Ratha and Anil K. Jain. Computer vision algorithms on reconfigurable logic arrays. *IEEE Trans. Parallel and Distributed Systems*, 10(1):29–43, January 1999.

[154] J. Rhea. Proposed micro-reconnaissance helicopter uses GPS. *Military and Aerospace Electronics*, April 1999.

[155] W. Roelandts. IC technology enables new programmable DSP solutions. In *Proc. IEEE Intl. Conf. Acoust., Speech, Signal Processing (ICASSP)*, 1999. Keynote Speech.

[156] S. Roman. *Advanced Linear Algebra*. Springer-Verlag, 1992.

[157] J. B. Rosen, H. Park, and J. Glick. Total least norm formulation and solution for structured problems. *SIAM J. Matrix Anal. Appl.*, 17(1):110–126, January 1996.

[158] S. Roy and I. J. Cox. A maximum-flow formulation of the n-camera stereo correspondence problem. In *Proc. Intl. Conf. on Computer Vision (ICCV)*, pages 492–499, 1998.

[159] A. Shashua. Algebraic functions for recognition. Technical Report AIM-1452, MIT Artificial Intelligence Laboratory, January 1994.

[160] A. Shashua and S. Avidan. The rank 4 constraint in multiple view geometry. In *Proc. European Conf. Computer Vision*, 1996.

[161] M. J. Shulte and E. E. Swartzlander Jr. Truncated multiplication with correction constant. In *VLSI Signal Processing*, volume 4. IEEE Press, 1993.

[162] A. Sinclair and M. Jerrum. Approximate counting, uniform generation and rapidly mixing Markov chains. *Information and Computation*, 82:93–133, 1989.

[163] C. C. Slama, editor. *Manual of Photogrammetry*. American Society of Photogrammetry, 4th edition, 1980.

[164] M. J. Smith and D. W. G. Park. Towards a new approach for absolute orientation and exterior orientation. *PhotoRec*, 16(94), October 1999.

[165] S. Soatta and P. Perona. Reducing "structure from motion". In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 1996.

[166] P. Stefanovic. Relative orientation - a new approach. *I.T.C. Journal*, 3:417–448, 1973.

[167] G. P. Stein. *Geometric and Photometric Constraints: Motion and Structure from Three Views*. PhD thesis, Massachusetts Institute of Technology, June 1998.

[168] G. P. Stein. Tracking from multiple view points: Self-calibration of space and time. In *DARPA Image Understanding Workshop*, November 1999.

[169] G. P. Stein and A. Shashua. Direct methods for estimation of structure and motion from three views. Technical Report AIM-1594, MIT Artificial Intelligence Laboratory, November 1996.

[170] C. V. Stewart. Robust parameter estimation in computer vision. *SIAM Review*, 41(3):513–537, 1999.

[171] G. W. Stewart and J.-G. Sun. *Matrix Perturbation Theory*. Academic Press, 1990.

[172] J. Stoer and R. Bulirsch. *Introduction to Numerical Analysis*. Springer, 2nd edition, 1993.

[173] G. Strang. *Linear Algebra*. Academic Press, 1980.

[174] G. Strang. *Introduction to Applied Mathematics*. Wellesley-Cambridge Press, 1986.

[175] W. Sung and K.-I. Kum. Simulation-based word-length optimization method for fixed-point digital signal processing systems. *IEEE Trans. Signal Processing*, pages 3087–3090, December 1995.

[176] E. E. Swartzlander Jr. Merged arithmetic. *IEEE Trans. on Computers*, 29:946–950, 1980.

[177] J. I. Thomas and J. Oliensis. Dealing with noise in multiframe structure from motion. *Computer Vision and Image Understanding*, 72(2):109–124, November 1999.

[178] C. M. Thompson. *Robust Photo-Topography by Fusing Shape-from-Shading and Stereo*. PhD thesis, Massachusetts Institute of Technology, February 1993.

[179] C. Tomasi and T. Kanade. Detection and tracking of point features. Technical Report CMU-CS-91-132, Carnegie Mellon University, April 1991.

[180] C. Tomasi and T. Kanade. Shape and motion from image streams under ortography: a factorization approach. *International Journal of Computer Vision*, 9(2), 1992.

[181] T. Tommasini, A. Fusiello, E. Trucco, and V. Roberto. Making good features track better. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 178–183, 1998.

[182] J. Ton and A. K. Jain. Registering Landsat images by point matching. *IEEE Trans. Geoscience and Remote Sensing*, 27(5), September 1989.

[183] B. Triggs. Camera pose and calibration from 4 or 5 known 3D points. In *Proc. Intl. Conf. on Computer Vision (ICCV)*, pages 278–284, 1999.

[184] E. Trucco and A. Verri. *Introductory Techniques for 3D Computer Vision*. Prentice Hall, 1998.

[185] R. Y. Tsai and T. S. Huang. Uniqueness and estimation of three-dimensional motion parameters of rigid objects with curved surfaces. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 6(1), January 1984.

[186] D. W. Tufts and P. D. Fiore. Simple, effective estimation of frequency based on Prony's method. In *Proc. IEEE Intl. Conf. Acoust., Speech, Signal Processing (ICASSP)*, volume 5, pages 2801–2804, 1996.

[187] D. W. Tufts and R. Kumaresan. Estimation of frequencies of multiple sinusoids: Making linear prediction perform like maximum likelihood. *Proceedings of the IEEE*, 70(9), September 1982.

[188] D. W. Tufts, E. C. Real, and J. W. Cooley. Fast approximate subspace tracking (FAST). In *Proc. IEEE Intl. Conf. Acoust., Speech, Signal Processing (ICASSP)*, 1997.

[189] S. Umeyama. Least-squares estimation of transformation parameters between two point patterns. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 13(4), April 1991.

[190] A.-J. Van Der Veen. Algebraic methods for deterministic blind beamforming. *Proceedings of the IEEE*, 86(10):1987–2008, October 1998.

[191] S. Van Huffel, H. Park, and J. B. Rosen. Formulation and solution of structured total least norm problems for parameter estimation. *IEEE Trans. Signal Processing*, 44(10):2464–2474, October 1996.

[192] S. Van Huffel and J. Vandewalle. *The Total Least Squares Problem: Computational Aspects and Analysis*. SIAM, 1991.

[193] M. Velez-Reyes and G. C. Verghese. Decomposed algorithms for parameter estimation with fast convergence rates. *MIT Laboratory for Electromagnetic and Electronic Systems*, 1997.

[194] J. Villasenor et al. Configurable computing solutions for automatic target recognition. In *IEEE Workshop on FPGAs for Custom Computing Machines (FCCM)*, 1996.

[195] D. Villeger and V. Oklobdzija. Analysis of Booth encoding efficiency in parallel multipliers using compressors for reduction of partial products. In *Proc. IEEE Asilomar Conf. on Signals, Systems, and Computers*, volume 1, pages 781–784, 1993.

[196] J. Volder. The CORDIC trigonometric computing technique. *IRE Trans. Electron. Computing*, 8:330–334, September 1959.

[197] B. Von Herzen. Signal processing at 250MHz using high-performance FPGAs. In *Proc. 1997 ACM Fifth Annual Symposium on Field-Programmable Gate Arrays (FPGA97)*, February 1997.

[198] J. S. Walther. A unified algorithm for elementary functions. In *Spring Joint Computer Conf.*, pages 379–385, 1971.

[199] Z. Wang and A. Jepson. A new closed-form solution for absolute orientation. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 129–134, 1994.

[200] A. D. Whalen. *Detection of Signals in Noise*. Academic Press, 1971.

[201] A. Wilson. Gate-array-based adaptive computing speeds image processing. *Vision Systems Design*, September 1998.

[202] A. Wilson. Programmable logic promotes image-processing ICs. *Vision Systems Design*, April 1999.

[203] S. Winograd. *Arithmetic Complexity of Computations*. Regional Conference Series in Applied Math, CBMS-NSF No. 33. SIAM, 1980.

[204] R. D. Wittig and P. Chow. OneChip: An FPGA processor with reconfigurable logic. In *IEEE Workshop on FPGAs for Custom Computing Machines (FCCM)*, April 1996.

[205] P. W. Wong. Quantization and roundoff noises in fixed-point FIR digital filters. *IEEE Trans. Signal Processing*, 39(7), July 1991.

[206] W.-H. Wong and W.-C. Siu. Improved digital filter structure for fast moments computation. *IEE Proc.-Vis Image Signal Process.*, 146(2), April 1999.

[207] Xilinx Corporation. The programmable logic data book. *http://www.xilinx.com*, 1998.

[208] J.-W. Yi and J.-H. Oh. Estimation of depth and 3D motion parameter of moving object with multiple stereo images. *Image and Vision Computing*, 14:501–516, 1996.

[209] Z. Zhang. Determining the epipolar geometry and its uncertainty: A review. *International Journal of Computer Vision*, 27(2):161–195, 1998.

[210] D. Zwillinger, editor. *CRC Standard Mathematical Tables and Formulae*. CRC Press, 30th edition, 1996.