

**Reducing Noise in 3D Models Recovered
From a Sequence of 2D Images**

J. Inigo Thomas

CMPSCI TR93-74

September 1993

This work was supported by the Advanced Research Projects Agency (via TACOM) under contract number DAAE07-91-C-RO35, and by the National Science Foundation under grant number CDA-8922572.

**REDUCING NOISE IN 3D MODELS RECOVERED FROM A
SEQUENCE OF 2D IMAGES**

A Dissertation Presented

by

J. INIGO THOMAS

Submitted to the Graduate School of the
University of Massachusetts in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

September 1993

Department of Computer Science

© Copyright by J. INIGO THOMAS 1993

All Rights Reserved

Dedicated to my wife Anne

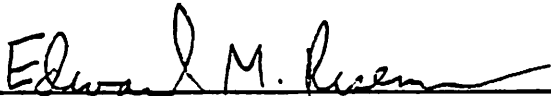
**REDUCING NOISE IN 3D MODELS RECOVERED FROM A
SEQUENCE OF 2D IMAGES**

A Dissertation Presented


by

J. INIGO THOMAS

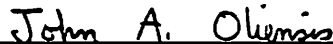
Approved as to style and content by:



Edward M. Riseman, Chair




Allen R. Hanson, Member




John A. Oliensis, Member



Roderic A. Grupen, Member



Haluk Derin, Member



W. Richards Adrion, Department Chair
Department of Computer Science

ACKNOWLEDGMENTS

I thank Ed Riseman for reading several drafts, painstakingly commenting on them, advising me and challenging me. Many thanks to Al Hanson for his comments, motivation, support, and for setting high standards. I also thank Ed and Al for making it possible for me to work in their excellently equipped, high-quality research laboratory. I thank John Oliensis very much for working closely with me, commenting on innumerable drafts, and always having the time for me. I also thank Rod Grupen for his valuable suggestions and ideas, and Prof. Derin for his comments and theoretical insights.

I wish to specially thank Anne Vainikka, who I consider as an unofficial member of my committee. Anne read almost every draft of the dissertation, with constructive criticisms on style, language and logic of presentation. I also thank Sandy Pollatsek, Brian Burns, Harpreet Sawhney, R. Manmatha, Lance Williams, Jan Koenderink and Volker Aurich, for the many interesting discussions about Vision. Thanks to the researchers at the UMass computer vision laboratory, especially Rabi Dutta, Bruce Draper, Mark Snyder, Poornima Balasubramanyam, Chris Connolly, Teddy Kumar, Rich Weiss, Zhongfei Zhang, Bob Collins, John Dolan, Ross Beveridge, Benny Rochweg, Gökhan Kutlu, Sumit Badal, Sashi Buluswar, Jonathan Lim, Yong-Qing Cheng, Runsheng Wang, and Brian Pinette, for helping me enhance my knowledge of various topics in computer vision.

I thank Harpreet Sawhney and Teddy Kumar for the image sequences and related data used in Experiments I and II (Chapter 4). Thanks also to R. Manmatha and R. Dutta for providing me with clarifying information on the image sequence used in Experiment III. Special thanks to Teddy Kumar for making available his program for

pose estimation which was modified and used in Experiments IV and V. Thanks to Jonathan Lim, who helped me with videos, displays, and system software. Thanks also to Bob Heller for helping me with several computers, and answering many questions. Thanks are due to Bruce Draper for providing me with the code for Absolute Orientation, to Harpreet Sawhney for providing me with the code for Relative Orientation, to Ross Beveridge and Bob Collins for the matrix inversion routine, and to Benny Rochweg for the optical flow code. Special thanks to Kwan Liu, who is converting the original LISP implementation to a C implementation. Many thanks to Prof. Aurich who made vision-related computing hardware available to me during my eight-month stay at the University of Düsseldorf, and thanks to J. Weule for helping me with the hardware. Special thanks to Laurie, Janet, and Val, for making things go smoothly at UMass.

I thank my wife Anne for bearing with me during the many long, stressful periods, and always being a source of support and love. Many thanks to my parents, especially my mother, for teaching me the value of education and helping me all along, in spite of the hardships that we faced. Thanks to my brother and sisters, and my friends, especially those from my church, who have made this work possible.

ABSTRACT

REDUCING NOISE IN 3D MODELS RECOVERED FROM A SEQUENCE OF 2D IMAGES

SEPTEMBER 1993

J. INIGO THOMAS

B.E., COLLEGE OF ENGINEERING, MADRAS

M.S., UNIVERSITY OF MASSACHUSETTS AMHERST

PH.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Edward M. Riseman

The goal of this dissertation is to develop a technique for constructing models of a scene using camera images obtained by a moving robot. Such models are useful for a navigating robot, especially for positioning itself in the world, following paths and avoiding obstacles. In order to construct a 3D model from images, the information about the camera's motion between viewpoints is needed. However, estimating the camera's motion turns out to be a major source of error in the constructed 3D model; in all previous work on motion this motion error has been neglected. The main contribution of this dissertation is isolating the motion error, estimating its effect, and correcting for it using an incremental algorithm.

Motion error manifests itself in cross-correlations of errors between points in the 3D model. The algorithm developed in this dissertation weights an individual 3D model by the inverse of its covariance matrix (which contains the cross-correlations), reflecting the accuracy of the model. Such weighted 3D models - obtained as the robot moves - are then combined.

The performance of the algorithm was compared against three algorithms which

neglect the motion error: Horn's two-frame algorithm, a multi-frame blind averaging algorithm, and a standard multi-frame Kalman Filtering algorithm. In three experiments considered (involving a robot workcell sequence, an indoor lobby sequence, and an outdoor rocket-field sequence), the algorithm consistently outperformed (by a factor of 2-3) the other three algorithms. In further experimentation, the constructed 3D model was used to determine the position of a robot with a accuracy of 2-3%.

The computational complexity of the algorithm is $O(n^3)$ (for n points in the model). In preliminary experiments, it was determined that reducing computational time by ignoring parts of the covariance matrix does not appear promising, whereas dividing larger 3D models into smaller subsets of points (while maintaining the full covariance matrix for each subset) may turn out to speed up the algorithm without sacrificing accuracy. Furthermore, it is estimated that constructing and updating a model made up of 22 points takes only 1.8 seconds on a fast Silicon Graphics machine (SGI) every time the robot moves.

TABLE OF CONTENTS

	<u>Page</u>
ACKNOWLEDGMENTS	v
ABSTRACT	vii
LIST OF TABLES	xiii
LIST OF FIGURES	xvi
Chapter	
1. INTRODUCTION	1
1.1 Relevance of Motion	2
1.2 Applying Motion to Computer Vision	3
1.2.1 General Framework	4
1.2.2 General Assumptions	6
1.2.3 Defining SFM	7
1.3 General Problems with Two-Frame SFM	12
1.4 Multi-Frame Structure From Motion	14
1.5 Goals of the Dissertation	16
1.6 Outline of the Dissertation	18
2. REVIEW OF ALGORITHMS	20
2.1 Introduction	20
2.2 Options in SFM	20
2.2.1 Representational Primitives	21
2.2.2 Dense vs. Sparse Models	22
2.2.3 Shape vs. Structure	22
2.2.4 General vs. Constrained Camera Motion	23
2.2.5 Visual vs. Non-Visual Information	23
2.2.6 Computational Techniques	24
2.2.7 Input Data	24

2.3	Two-Frame SFM Algorithms	24
2.3.1	Optical Flow	25
2.3.2	Direct Methods	29
2.3.3	Tracked Features	31
2.3.3.1	Advantages and Disadvantages	31
2.3.3.2	Essential Matrix Algorithms	33
2.3.3.3	Solution Techniques	33
2.4	Problems in Two-Frame SFM	35
2.4.1	Inherent Theoretical Restrictions	35
2.4.2	Practical Problems	36
2.5	Multi-Frame SFM Algorithms	39
2.6	Batch Methods	39
2.6.1	Uniform Rotation	40
2.6.2	Uniform Translation and Rotation	41
2.6.3	Planar Motion	43
2.6.4	General Motion	44
2.7	Incremental MFSFM	45
2.7.1	General Assumptions of Incremental MSFSM	46
2.7.2	Options in Incremental MFSFM	47
2.7.2.1	Representing the 3D Model Error	47
2.7.2.2	Transforming the Error Across Coordinate Systems	48
2.7.3	Incremental Algorithms	49
2.7.3.1	Camera Restricted to Pure Translational Motion	50
2.7.3.2	Camera Restricted to Planar Motion	50
2.7.3.3	General Motion with Restricted Experiments	51
2.7.3.4	General Motion with Unrestricted Experiments	52
3.	A FRAMEWORK FOR MULTIFRAME STRUCTURE FROM MOTION	54
3.1	The Cross-Correlation-based Incremental Algorithm	55
3.1.1	The Steps of the Algorithm	55
3.1.2	The Error Modules	60
3.2	The Indirect Error	62
3.2.1	Defining the Problem	62
3.2.2	$\frac{\partial M}{\partial I}$: Motion error (∂M) with respect to Image Error (∂I)	62
3.2.2.1	The Components of the Motion M	65
3.2.2.2	Determining the A Matrix	67
3.2.2.2.1	The Two Translation Parameters.	67
3.2.2.2.2	The Three Rotation Parameters.	68
3.2.2.2.3	The Components of the A Matrix.	69
3.2.2.2.4	The Four Translation Derivatives of A	69
3.2.2.2.5	The Nine Rotational Derivatives of A	70
3.2.2.2.6	The Twelve Mixed Derivatives of A	71

3.2.2.3	Determining the B matrix	71
3.2.2.3.1	The Components of the Image Coordinates I.	72
3.2.2.3.2	The Components of the B Matrix.	72
3.2.2.3.3	The Two Coordinates of the First Image.	72
3.2.2.3.4	The Two Coordinates of the Second Image	74
3.2.3	$\frac{\partial W}{\partial M}$: 3D Model Error (dW) with respect to Motion Error (dM)	75
3.3	The Direct Error	76
3.4	The Combined Effect of Direct and Indirect Errors	77
3.5	The Iterative Step: Fusing the New Two-frame 3D model with the Old 3D model	79
3.6	Theoretical Motivation for Using Cross-Correlations	81
3.6.1	The Meaning of Cross-correlations: The Two-Point Case	81
3.6.2	The Effect of Cross-correlations in Kalman Filtering	83
4.	EXPERIMENTAL RESULTS	86
4.1	Introduction	86
4.2	The Four Algorithms	86
4.2.1	The Two-Frame Algorithm	87
4.2.2	Blind Averaging	87
4.2.3	Standard Kalman Filtering vs. the CC-based Algorithm	88
4.3	Experiment I: Robot Workcell Sequence	90
4.3.1	The Image Sequence and Ground Truth	90
4.3.2	Input to the Algorithms	93
4.3.2.1	Tracked Image Coordinates	93
4.3.2.2	The Covariance of the Error	95
4.3.2.3	Guess of Interframe Camera Motion	95
4.3.2.4	Scale of the Model	96
4.3.3	Results of the Four Algorithms	98
4.3.3.1	Representation of the Results	98
4.3.3.2	Discussion of the Results	99
4.4	Experiment II: Indoor Robot Sequence	104
4.4.1	The Image Sequence and Ground Truth	104
4.4.2	Input to the Algorithms	107
4.4.3	Results of the Four Algorithms	108
4.4.3.1	Representation of the Results	108
4.4.3.2	Discussion of the Results	109
4.5	Experiment III: Outdoor Mobile Vehicle Sequence	113
4.5.1	The Image Sequence and Ground Truth	113
4.5.2	Input to the Algorithms	115
4.5.2.1	Tracked Image Coordinates	115
4.5.2.2	The Covariance of the Image Error	118
4.5.2.3	Guess of Interframe Camera Motion	118
4.5.2.4	Scale of the Model	118

4.5.3	Results of the Four Algorithms	120
4.5.3.1	Representation of the Results	120
4.5.3.2	Discussion of the Results	120
4.6	Application to Robot Navigation	129
4.7	Experiment IV: Simulated Model Acquisition and Model-Based Navigation	131
4.7.1	Simulating the Image Sequences	131
4.7.2	Acquiring the 3D Model	133
4.7.3	Determining the Position of the Robot	135
4.8	Experiment V: Real Model Acquisition and Model-Based Navigation	135
4.8.1	The Image Sequence	135
4.8.2	Acquiring the 3D Model	138
4.8.3	Determining the Position of the Robot	138
4.9	Conclusion	139
5.	COMPUTATIONAL ISSUES	142
5.1	Introduction	142
5.2	Time Complexity of the Components	143
5.2.1	Theoretical Complexity	143
5.2.2	Actual Running Times	144
5.3	Reducing Running Time	146
5.3.1	Effect of Reducing Cross-correlations on Running Time	147
5.3.2	Effect of Reducing Cross-correlations on Accuracy	153
5.3.2.1	Experiment A: Dropping Cross-correlations of Points	154
5.3.2.2	Experiment B: Dropping Cross-correlations of Points in Multi-Frame Models	156
5.3.2.3	Experiment C: Dropping Cross-correlations of Coordinates	157
5.4	Reducing the Size of the 3D Model	160
5.5	Conclusion	164
6.	CONCLUSION	166
6.1	Contributions of the Dissertation	166
6.2	Future Research Directions	168
A.	KALMAN FILTERING TERMINOLOGY	172
	APPENDICES	
	BIBLIOGRAPHY	174

LIST OF TABLES

Table	Page
2.1 Batch MFSFM algorithms showing the number of motion variables (general case: $6m - 1$) and average accuracy computed from the reported results (m denotes the number of camera movements).	40
2.2 Recursive MFSFM algorithms showing the number of elements used to represent the 3D model error (general case: $9n^2$).	50
4.1 Box Sequence: The image coordinates of the 35 points in the first image	92
4.2 Box Sequence: Ground truth of tracked points	94
4.3 Camera Parameters	94
4.4 Box Sequence: The initial estimate and recovered two-frame values of the interframe camera motion	97
4.5 Box Sequence: Mean and standard deviation σ of the 3D model error in mm at each frame for the four algorithms.	99
4.6 Lobby Sequence: The image coordinates of the 29 points in the first image.	106
4.7 Lobby Sequence: Ground truth of tracked points with respect to the first camera position, in a camera-centered coordinate system.	107

4.8	Lobby Sequence: The camera parameters of the Sony AVC-D1 camera that is mounted on the mobile robot.	108
4.9	Lobby Sequence: Mean percentage 3D model errors at each frame for the four algorithms with standard deviations σ	112
4.10	Rocket-Field: The parameters of the camera mounted on the ALV. .	113
4.11	Rocket-Field Sequence: Ground truth for the tracked image points of the sequence in the camera-coordinate system at the first camera position	116
4.12	Rocket-Field Sequence: The image coordinates of the 22 points in the eleventh image of the sequence	117
4.13	Rocket-Field Sequence: The ground truth, initial guesses and recovered values of the interframe camera motion	119
4.14	Rocket-Field Sequence: Mean percentage 3D model errors at each frame for the four algorithms with standard deviations σ	123
4.15	Rocket-Field Sequence Frame 8: Ground Truth, Blind Average 3D Model, and Blind Average Error	124
4.16	Rocket-Field Frame 9: Ground Truth, Two-Frame 3D Model, and Two-Frame Error	124
4.17	Rocket-Field: Obtaining Blind Average (Frame 9) results from Transformed Blind Average (from Frame 8)	125

4.18 Rocket-Field Final Frame: Ground Truth, CC-based 3D Model, and CC-based 3D Model Error	128
4.19 Simulated Model Acquisition and Model-Based Navigation: Ground truth (first row) and recovered pose (second row) using the CC-based model at each frame of the second stage of the experiment	136
4.20 Lobby Sequence: Ground truth (first row) and recovered pose (second row) using the CC-based model	139
5.1 Running time for the four main components of the CC-based algorithm in the case of the Rocket-Field Sequence. The theoretically time com- plexity is indicated for convenience. The last column is the sum of the times for the four components plus a small overhead. The high value of 3D Error time for frames 10-11 is clearly an outlier, and possibly due to an unusual overhead from memory paging.	145
5.2 Effect of Reducing the number of 3D points on the accuracy of the resulting 3D model: mean percentage error for 3D models. This table lists the values used to plot Figure 5.6.	163

LIST OF FIGURES

Figure		Page
1.1	The Coplanarity Constraint in SFM	5
1.2	Problems with SFM	13
1.3	An Example of Incremental MFSFM	15
1.4	The Paths of Error in Constructing a 3D model using SFM	17
3.1	The first iteration of the cross-correlation-based algorithm	56
3.2	An i th iteration of the cross-correlation-based algorithm	57
4.1	Rotating Box Image Sequence	91
4.2	Box Sequence: Reconstruction Error (in mm) for the four algorithms compared in this experiment.	100
4.3	Box Sequence: Standard deviation of the error in the 3D models for four algorithms.	101
4.4	Image Sequence	105
4.5	Lobby Sequence: Mean error in the 3D models for the four algorithms	110
4.6	Lobby Sequence: Standard Deviation of the error in the 3D models for the four algorithms	111

4.7	Rocket-Field Image Sequence	114
4.8	Rocket-Field Sequence: Mean percentage error in the 3D models for the four algorithms	121
4.9	Rocket-Field Sequence: Standard deviation of the percentage error the of 3D models for the four algorithms	122
4.10	Rocket-Field Sequence: Resulting Z coordinate of point 9 after blind averaging	127
4.11	Simulated Model Acquisition and Model-Based Navigation: True 3D Model of Building consisting of walls, stairwells, and a cooling tank (at the top right-hand corner of the building)	132
4.12	Simulated Model Acquisition Experiment: Results of the acquired model.	134
4.13	Simulated Model-Based Navigation: Results showing top view of the true path and the recovered path	137
4.14	Model-Based Navigation in the Lobby: Results showing top view of the true path and the recovered path.	140
5.1	Partitioning the Covariance Matrix into submatrices for fast matrix inversion	149
5.2	Running times for inverting covariance matrices of size 30×30 , 48×48 , 66×66 and 84×84	152
5.3	Experiment A: Effect of dropping cross-correlations of points on the accuracy of the 3D models	155

5.4	Experiment B: Effect of dropping cross-correlations of points on the accuracy of the 3D models	158
5.5	Experiment C: Effect of dropping cross-correlations of coordinates on the accuracy of the 3D models	159
5.6	Effect of Reducing the number of 3D points on the accuracy of the resulting 3D models	162

CHAPTER 1

INTRODUCTION

Developing intelligent autonomous vehicles that navigate in arbitrary environments is a major application of computer vision research. The primary focus of such research is to achieve navigational capabilities for robots - namely, the ability to determine their location, to follow paths and to avoid obstacles - based on visual sensory information. In order to navigate in its environment, an intelligent vehicle often has an internal 3D model of its environment. By matching landmarks between the internal model and the camera images, the vehicle keeps continuous track of its position in the world. For such a task to be effective, an accurate 3D model of the world is required. Developing a technique for constructing accurate 3D models based on visual information is the primary goal of this dissertation. The 3D model that is constructed could also provide information for identifying and avoiding potential obstacles in the path of the robot.

Apart from the task of robot navigation, the model of the environment obtained by a moving camera can potentially be used in many other ways. A 3D model of objects facilitates visually guided manipulation and grasping by dextrous robots. In addition, deriving 3D models represents the final goal in tasks such as surveying, cartography or photogrammetry. In the rapidly growing field of computer graphics and virtual reality, accurate 3D models of objects and environments are important for displaying objects easily from any position or orientation. Providing 3D models for

such purposes may turn out to be an important application of the approach developed in this dissertation.

In a general situation, a single camera image does not contain enough information to construct a 3D model of the environment. However, two or more images from different locations typically provide enough information. Several approaches to constructing 3D models are being pursued at the University of Massachusetts (UMass) in the Unmanned Ground Vehicle (UGV) research project. One approach involves using stereo information from two cameras mounted at a fixed distance from each other on the UGV. A second approach involves information obtained from a single moving camera; the use of motion information is the focus of the work in this dissertation as well as several previous dissertations at the University of Massachusetts [2] [5] [55] and [78]. A third approach is model-based extension of a partial model using pose determination and feature tracking [52].

1.1 Relevance of Motion

Humans perceive the world three-dimensionally through stereopsis – by combining information from both eyes – as well as by combining the different views of the world obtained by moving. Psychophysical experiments (such as the so-called random dot experiments [61]) provide evidence that even when all the surfaces in the world are painted with random black and white dots, monocularly viewing the world from two or more different positions – as in a movie sequence – produces a powerful perception of depth. That is, we have a strong perception of the spatial relationships in the world even in the random dot experiments which are devoid of all cues except those due to motion. The different views falling on even just a single retina, for example due to the movement of the observer, provide a very good source of 3D information about

the environment.¹ Furthermore, based on neurophysiological studies it is known that motion cues and depth perception are closely linked. The same region of the visual cortex is involved both in sensing objects in motion as well as in perceiving distances to objects in the world [32].

In human vision, Gibson [31] was the first to analyze the characteristic changes in the observed image due to relative motion between the observer and the 3D scene, and to suggest how these characteristic changes might be used to infer information about the environment. He showed that the change in the image is related both to the distance between the observer and the object, and to the direction in which the observer moves. For example, when the observer moves sideways (as in looking out a train window) far away objects seem stationary whereas nearby objects seem to move faster.

1.2 Applying Motion to Computer Vision

In computer vision, the basic observation that image changes are related to distances has been adapted to a computational procedure for determining a 3D model of the environment using two or more images; this procedure infers 3D *structure*. When the images from different viewpoints are acquired by *moving* the camera, the approach is called *Structure From Motion* (SFM). Note that a difference between human vision and computer vision is that humans typically process images falling continually on the retina, whereas in computer vision the images are discretely sampled.

An approach related to SFM employs a stereo camera pair. In this case a 3D model of the environment is constructed based on images obtained from two separate cameras at a known distance apart. Constructing the model is similar to SFM

¹Other visual cues such as information from shading and the effects of perspective are also used by humans to perceive distances to objects. However, the perception of depth is in general strongest when two or more views are available.

except that the relative position of the two camera views is assumed to be known via calibration. An advantage of knowing the relative position of the cameras is that this relative position need not be calculated, which in SFM introduces additional error (presumably more than the calibration error). However, an inherent problem with stereo is the fixed distance (or baseline) between the cameras; the smaller the baseline the less accurate the reconstruction of the environment (especially for distant points). The problem of a fixed, short distance between the cameras can be solved if instead of a fixed arrangement, one camera is moved from point to point. This is the case of Structure from Motion, sometimes loosely referred to as *extended stereo*. Apart from the fact that almost any pair of views of the scene can be obtained - in effect by manipulating the distance between the cameras - SFM and stereo are logically equivalent. Due to its flexibility we have chosen the SFM approach to reconstruct the environment.²

1.2.1 General Framework

In order to understand the general framework of SFM we will first consider the minimal case of deriving a 3D model based on just two views, obtained by moving the camera once. This approach is referred to as *Two-Frame SFM* and was the earliest instance of SFM to be considered in computer vision. Later in this chapter we will discuss general problems with Two-Frame SFM.

Figure 1.1 depicts a typical scenario where an image from two different viewpoints is obtained. The main point of the figure is to represent the crucial information used in SFM (the symbols in this figure will be explained in more detail in Section 1.2.3). Whenever the camera rotates and moves, the vector representing the movement (translation T) and the two rays from each camera position to a point in the

²A recent direction of research is binocular motion, which is the integration of stereo and motion in an attempt to combine the advantages of both [8].

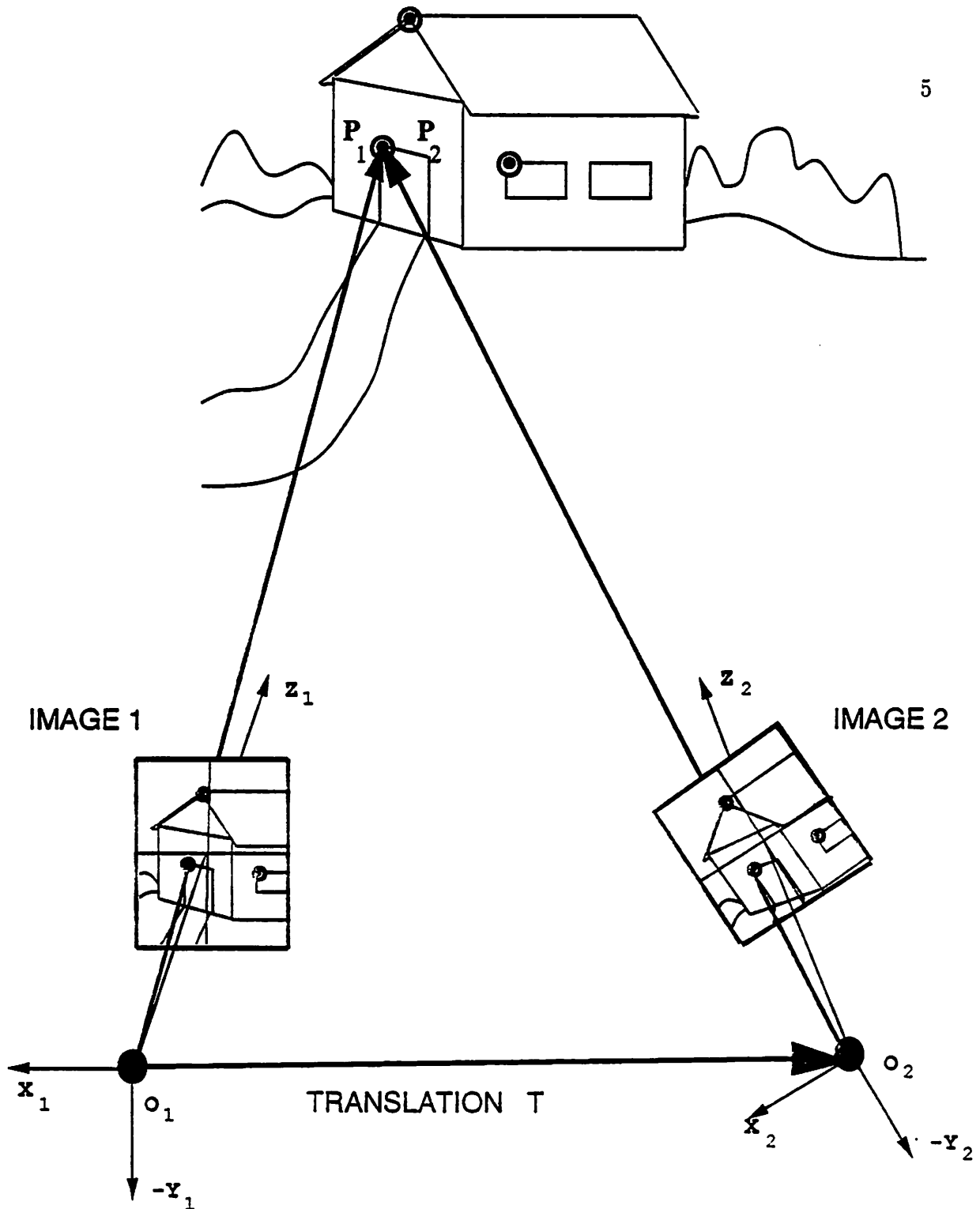


Figure 1.1: The Coplanarity Constraint in SFM. The darkened rays from the two camera locations to the corner of the door are coplanar with the translation vector T . The coordinate system associated with the first camera position has an origin O_1 and the three axes X_1, Y_1 and Z_1 ; the corresponding coordinate system in the second camera position is X_2, Y_2, Z_2 with origin O_2 . The point in the world has 3D coordinates P_1 with respect to the first camera position, and P_2 with respect to the second camera position.

world are on the same plane, i.e., the three rays are coplanar. This is referred to as the *coplanarity constraint* and it can be used in the deduction of the 3D coordinates of a point in the environment, given the two images.

1.2.2 General Assumptions

The following are standard assumptions in Two-Frame SFM:

1. The same 3D environment is observed in both images.
2. The two images are taken from different locations, by translating and rotating the camera.
3. The image in the camera is formed through perspective projection and the focal length of the camera is known.

The first assumption is often referred to as the *rigidity* assumption. It reflects the fact that two images are adequate to determine a model of the environment only if the environment does not change significantly from one image to the next. If the two images depict different scenes, their information cannot be combined and a 3D model cannot be constructed. The second assumption follows from the fact that 3D reconstruction is possible only when more than one view is involved. In a variant of assumption 2, the camera could be stationary while the visible environment moves rigidly. The third assumption is a good approximation of how a typical camera obtains the image; we will return to perspective projection in the next section.

The first assumption of a rigid environment is frequently violated in real-world situations. There are two types of violations of rigidity, one of which can be accommodated by SFM with additional preprocessing, while the other may be impossible to adequately deal with in SFM. The first type of violation involves the presence of moving objects in the environment to be modelled. In such a case, the objects can be

isolated (i.e., segmented) and standard SFM can, in turn, be applied to each object independently [3]. The second type of violation involves non-rigid deformation of the environment such as dilation/contraction, shearing and warping (e.g. clouds, water, a pulsating heart). Modifications of standard SFM have tried to account for small violations of rigidity [6] by including a single rigid/non-rigid parameter (cf. Chapter 2), but techniques that model various kinds of non-rigid deformations [65] fall outside of the scope of standard SFM. Furthermore, the complicated type of deformations (such as warping) occur only with some objects and hence will not be viewed as a serious deficiency of standard SFM.

1.2.3 Defining SFM

In this section, the mathematical basis for reconstructing a 3D model of the environment from a pair of 2D images is established. For the sake of exposition, let us consider a model consisting of selected 3D points (e.g. corners of buildings). Possibilities for using features other than points are discussed in Chapter 2.

Since the camera moves from one image to the next, two possible camera coordinate systems arise (as shown in Figure 1.1) in which the 3D model of the environment could be represented. We shall assume that the frame of reference is the second camera position; if necessary, it is straightforward to convert to the original camera position. Specifically, the model will be represented in the 3D coordinate system formed by the three orthogonal axes (X, Y, Z) with an origin O at the focal point of the camera. The Z -axis is the direction of gaze and is perpendicular to the image, while the other two axes (X, Y) lie on a plane parallel to the image. The goal of SFM is to obtain the 3D coordinates of each point in the model based on the 2D coordinates of the point in the two images. This goal is achieved by utilizing the coplanarity constraint (cf. Section 1.2.1).

Consider the problem of computing the 3D coordinates of an interesting feature in the environment, such as the corner of an object. Let us refer to the coordinates of the corner in the first image with respect to the axes (X, Y) as (x_1, y_1) and the corresponding coordinates in the second image as (x_2, y_2) ; matching corresponding points from one image to the next is termed the *correspondence problem*.³ The image coordinates are the result of projecting/mapping⁴ the point in the world onto the 2D image plane. Since there is a precise relationship between the position of the 3D point and its location in the image, we can use this fact to reconstruct the position of the corner in the real world. The coordinates of this corner with respect to the first camera position (i.e., image 1 in Figure 1.1) are referred to as (X_1, Y_1, Z_1) and with respect to the second camera position as (X_2, Y_2, Z_2) ; the 3D model is a set of such (X_2, Y_2, Z_2) coordinates for each point. The relationship between the 2D coordinates and any corresponding 3D point is given in the following equations:

$$x_1 = f \frac{X_1}{Z_1} \quad (1.1)$$

$$y_1 = f \frac{Y_1}{Z_1} \quad (1.2)$$

$$x_2 = f \frac{X_2}{Z_2} \quad (1.3)$$

$$y_2 = f \frac{Y_2}{Z_2} \quad (1.4)$$

³A matching procedure such as tracking is necessary in order to determine corresponding points in the two images; the set of points constitutes the input to SFM (see Chapter 2 for more details).

⁴The underlying geometry involved in creating a camera's image is typically assumed to be perspective projection. Although this projection exactly represents an idealized pinhole camera, the projection does not reflect minor distortions that arise in a camera due to combining several lenses, or due to lens defects. We will ignore these effects, as is standard.

These equations result from assuming perspective projection (cf. footnote 4), where f denotes the known focal length of the camera. For a single point, these equations by themselves do not provide enough information to enable us to compute the unknown variables (X_2, Y_2, Z_2) since (X_1, Y_1, Z_1) are also unknown, resulting in six unknown variables in four independent equations. However, if (X_1, Y_1, Z_1) is known in terms of (X_2, Y_2, Z_2) , then a solution is possible. This will be the case if the motion of the camera is known or estimated.

Before we proceed any further we need to establish the representation for the camera's movement. Any camera motion between two images can be represented by a unique combination of a rotation R followed by a translation (or displacement) T . The rotation R - which can be thought of as the rotation about three orthonormal axes - requires at least three parameters for its representation. Similarly, the translation T involves three parameters corresponding to the displacement along the three axes. A general problem with SFM is that the three translation parameters cannot be uniquely established due to an inherent scale ambiguity [95]. This ambiguity arises since simultaneously scaling both the environment and the camera's displacement (translation) by the same factor does not change the image. Thus, the overall scale of the 3D model and the magnitude of the camera's translation cannot be determined absolutely from visual information. The validity of this scale ambiguity is independent of the number of images used. For the purposes of this discussion we will assume that the magnitude of the translation is set to unity; this fixes the scale of the model as well.

Using the camera's motion, i.e., its rotation (R) and translation (T), the pair of 3D coordinates of a point in the two camera positions can now be related as follows:

$$\begin{pmatrix} X_2 \\ Y_2 \\ Z_2 \end{pmatrix} = R \begin{pmatrix} X_1 \\ Y_1 \\ Z_1 \end{pmatrix} - T \quad (1.5)$$

Provided the camera motion is known, we can now solve for the value of (X_2, Y_2, Z_2) . In practice, however, the precise camera movement is usually not known due to mechanical issues and other problems. Nevertheless, the above equations can be used if the camera motion is first estimated. A theoretical variant of this idea is to simultaneously solve for the value of the camera motion as well as the 3D coordinates. In this discussion on SFM we will consider only one of the two possible alternatives. The first alternative, which decomposes the goal of SFM into two simpler subgoals (solving for camera motion and then calculating the 3D coordinates), will be considered instead of the second - i.e., simultaneously solving for the camera motion and the 3D coordinates. Note that both alternatives involve non-linear searches to find a solution. However, the first alternative involves a small, fixed-sized search (for the five parameters of the camera motion) whereas the second alternative involves a large search for $5 + 3n$ elements, where n is the number of 3D points in the model.

If we represent the 3D coordinates of the point in the first camera coordinate system as P_1 , and the 3D coordinates in the second camera coordinate system as P_2 , then Equation 1.5 - which relates the 3D coordinates from the first camera coordinate system to the second - can be rewritten as:

$$P_2 = R P_1 - T \quad (1.6)$$

Let us first determine the motion of the camera, namely, its rotation (R) and translation (T). In order to simplify Equation 1.6, we take the scalar (dot) product of both sides of the above equation with the term $(R P_1 \times P_2)$. This term represents a vector normal to the plane spanned by P_1 and P_2 . This normal will be perpendicular to T if the three rays (P_1, P_2, T) are indeed coplanar. Taking the dot product makes this condition apparent:

$$\mathbf{T} \cdot (R \mathbf{P}_1 \times \mathbf{P}_2) = 0 \quad (1.7)$$

Equation 1.7 holds even if the vectors \mathbf{P}_1 and \mathbf{P}_2 are replaced by their unit direction vectors $\hat{\mathbf{P}}_1$ and $\hat{\mathbf{P}}_2$, where $\hat{\mathbf{P}}_1 = \mathbf{P}_1/|\mathbf{P}_1|$ and $\hat{\mathbf{P}}_2 = \mathbf{P}_2/|\mathbf{P}_2|$. After the replacement we get:

$$\mathbf{T} \cdot (R \hat{\mathbf{P}}_1 \times \hat{\mathbf{P}}_2) = 0 \quad (1.8)$$

This equation is commonly referred to as the Coplanarity Constraint [43] (cf. Section 1.2.1), the Essential Matrix Constraint [95], or the Epipolar Constraint [104]. The equation captures the fact that the camera's translation (\mathbf{T}) is coplanar with the pair of rays ($\hat{\mathbf{P}}_1, \hat{\mathbf{P}}_2$) from the two locations of the camera to the 3D point in the environment.⁵

Once $\hat{\mathbf{P}}_1$ and $\hat{\mathbf{P}}_2$ (the unit *directions*) are written in terms of the known image coordinates of the points in the two images, then only R and \mathbf{T} will remain unknown in the equation. The replacement is as follows:

$$\hat{\mathbf{P}}_1 = \frac{(x_1, y_1, f)}{\sqrt{x_1^2 + y_1^2 + f^2}} \quad (1.9)$$

$$\hat{\mathbf{P}}_2 = \frac{(x_2, y_2, f)}{\sqrt{x_2^2 + y_2^2 + f^2}} \quad (1.10)$$

The values of $\hat{\mathbf{P}}_1$ and $\hat{\mathbf{P}}_2$ from equations 1.9 and 1.10 can now be substituted in equation 1.8; the only unknowns in the resulting equation are the rotation (R) and translation (\mathbf{T}) of the camera, which was our original goal.

⁵This equation also reflects the scale ambiguity in SFM discussed earlier which dictates that the magnitude of the translation cannot be established with just image information. If we replace T by, say, $\frac{1}{2}T$ (i.e. scale T by half) in equation 1.8, this scale change ($\frac{1}{2}$) gets factored out.

Although we have arrived at an equation relating the unknown camera motion to the known image coordinates, it turns out that a single equation is insufficient because R and T involve a total of five parameters. However, using five points gives rise to five equations which in general are enough to compute the five parameters of the camera motion, except for some finite ambiguities discussed in Chapter 2. Even the ambiguities can in general be resolved by using a larger number of points. Different solutions have been used to solve the system of equations for the camera motion that arise from tracking five or more points; various techniques will be discussed in Chapter 2.

Once the camera motion is computed, the 3D coordinates of the point can be determined. Equation 1.6 can now be rewritten as follows (using vector algebra; cf. Horn [43]):

$$\mathbf{P}_2 = \frac{(\mathbf{T} \times R \hat{\mathbf{P}}_1) \cdot (R \hat{\mathbf{P}}_1 \times \hat{\mathbf{P}}_2)}{|R \hat{\mathbf{P}}_1 \times \hat{\mathbf{P}}_2|^2} \hat{\mathbf{P}}_2 \quad (1.11)$$

This equation provides the 3D coordinates of a point in the environment completely in terms of the known image coordinates and in terms of the estimated camera motion. Recall that calculating the 3D coordinates of points in the environment is the final goal of SFM.

1.3 General Problems with Two-Frame SFM

We have considered an idealized situation until now in order to discuss the general framework of Two-Frame SFM. However, SFM is faced with many theoretical limitations as well as practical problems. One problem with SFM is that the input does not always lead to an unambiguous estimation of camera motion. For example, Adiv [3] depicts the image displacement of points (or the change in images) under certain

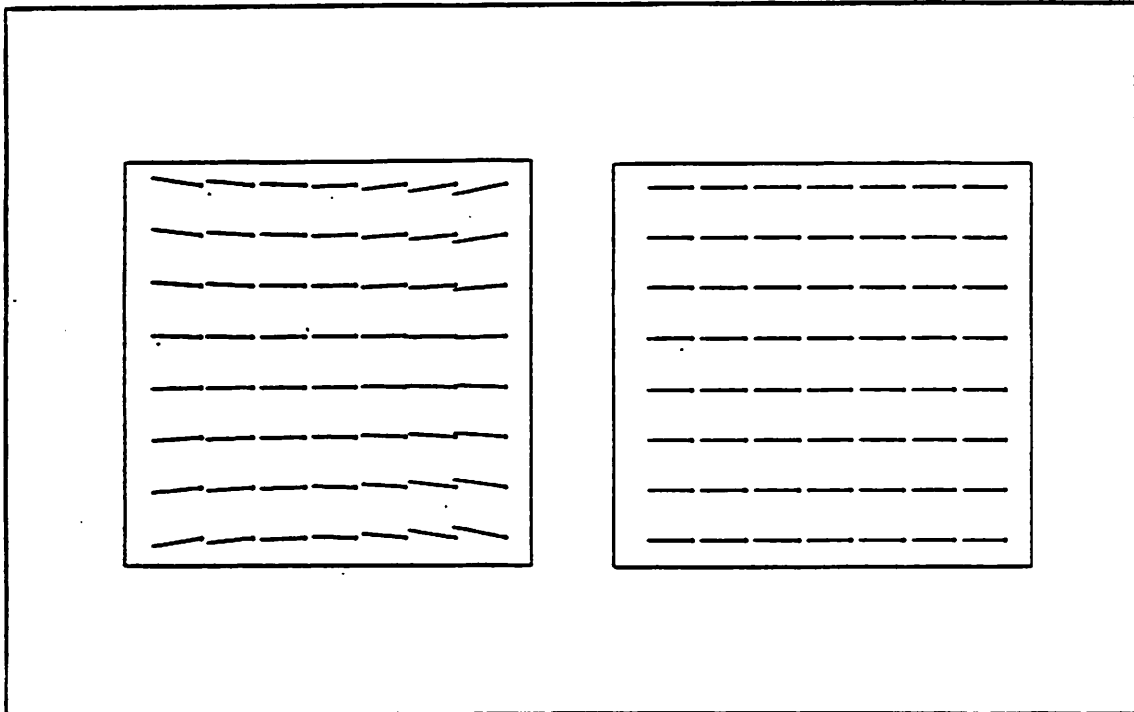


Figure 1.2: Problems with SFM. The changes in the image represented by arrows are almost the same all over the image even though the camera rotated in one case and translated in the other case. This figure is adapted from Adiv [3].

types of rotational and translational motion, and shows that the image displacement is similar – for most parts of the image – regardless of the type of motion (cf. Figure 1.2). Furthermore, problems arise due to low image resolution and various kinds of noise that corrupts the image measurements. Two-Frame SFM is therefore inaccurate due to these and other reasons which are discussed in detail in the following chapter.

1.4 Multi-Frame Structure From Motion

It makes intuitive sense that use of additional images should yield more accurate 3D models of the environment than using just two images, since information from different views can be used to reduce the effect of noise. We shall see in Chapter 2 how using only two images produces inaccurate 3D models. We will then discuss how to use more than two image frames to construct the 3D model; such methods are called *Multi-Frame Structure from Motion* (MFSFM).

If information from more than two images – say, ten images – is available, the information from all ten images can be processed at once (so-called batch methods) or incrementally, one image at a time (incremental methods). Batch methods suffer from the difficulty of a very large number of variables based on the large amount of information processed at one time. Consequently, batch methods are not suitable for real time processing; they also typically require more storage/memory in order to store all the previous information until processing is begun. For example, they are not suited for tasks involving a mobile robot, where the results are needed on-line, as opposed to later when the scene may have changed. In this dissertation the incremental approach will be adopted.

A technique for processing information from images incrementally to update a 3D model of the environment is depicted in Figure 1.3. Details of this particular incremental MFSFM algorithm will be developed in the following chapters and it will

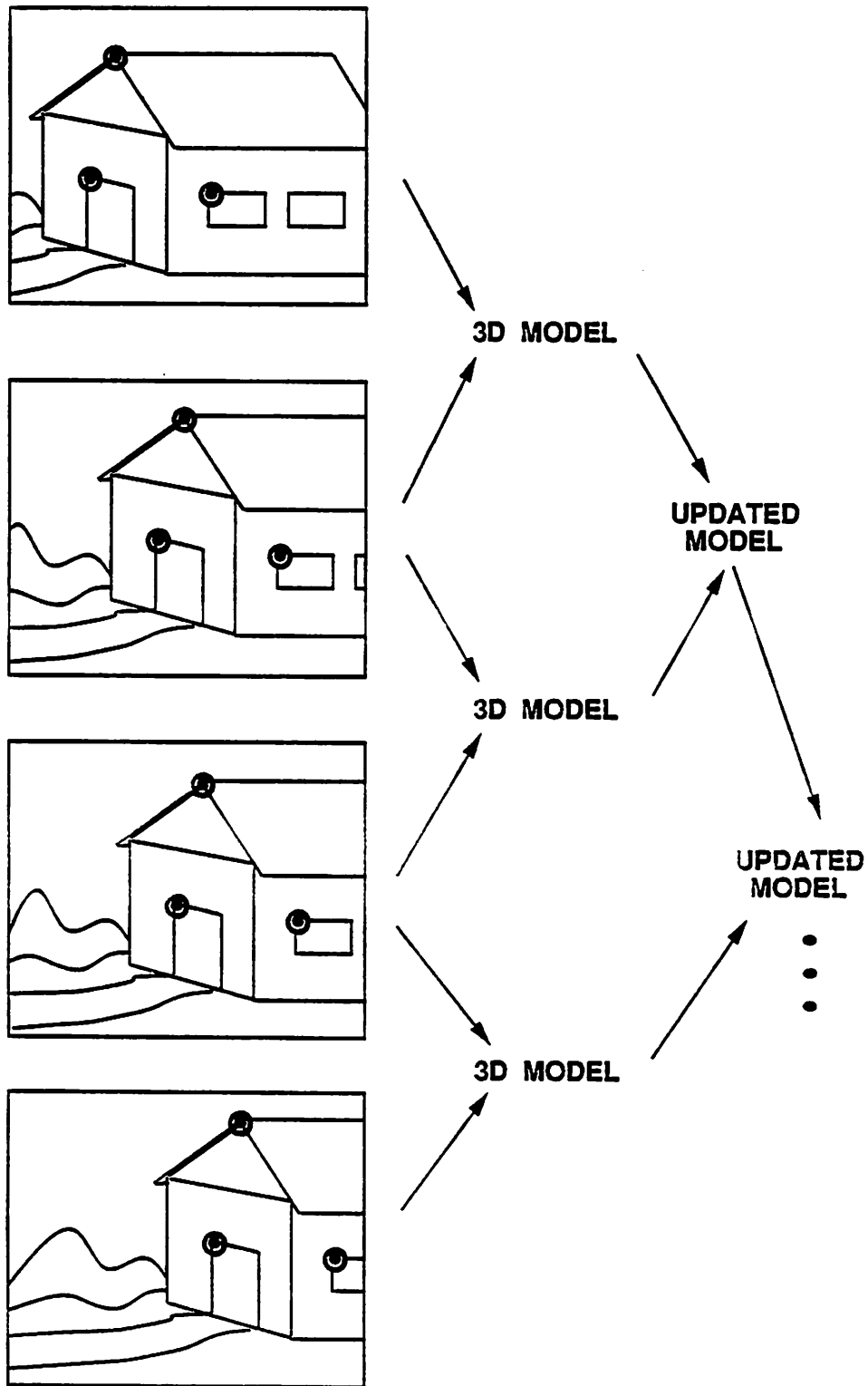


Figure 1.3: An Example of Incremental MFSFM. A model obtained from every pair of consecutive images is used to update the original model. See Chapter 3 for details of the particular algorithm developed here under this incremental approach.

be compared to other algorithms. In evaluating MFSFM approaches, it is important to note that intelligent updating or refining of the 3D model will occur only if a reasonable idea of the error in the 3D model is known. Otherwise, the 3D model will be blindly modified with new information, giving equal importance to both its accurate and inaccurate components. Much of the work in incremental MFSFM revolves around being able to compute a reliable estimate of the error in the 3D model so that it can be refined over time.

The origin of error in SFM is the error in the 2D image coordinates of tracked points. Since we are interested in a 3D model constructed from on such 2D points, we need to convert the errors in the 2D points (*image error*) into error in the 3D coordinates (*3D model error*). It is therefore necessary to consider the process by which the 3D model is obtained from the pair of 2D images. Since the 2D coordinates are used to construct a 3D model every time a new image is obtained (as shown in Figure 1.3), any error in the tracked points affects the 3D model directly. We will refer to this effect of the image error on the 3D model as *direct error*. In addition to direct error, the inaccurate 2D coordinates give rise to error in the 3D model in a second way: through the error in the estimated camera motion. We shall refer to the error in the camera motion as the *motion error*; the effect of this motion error in the 3D model will be referred to as *indirect error*. Note that the indirect error could be large even for small motion errors (cf. Chapter 2 for details). The types of errors and their mutual relationship are depicted in Figure 1.4; estimating and correcting the errors is the main goal of this dissertation.

1.5 Goals of the Dissertation

The primary goal of this dissertation is to show that taking the effect of noise in the input into account improves the accuracy of the models of the environment obtained

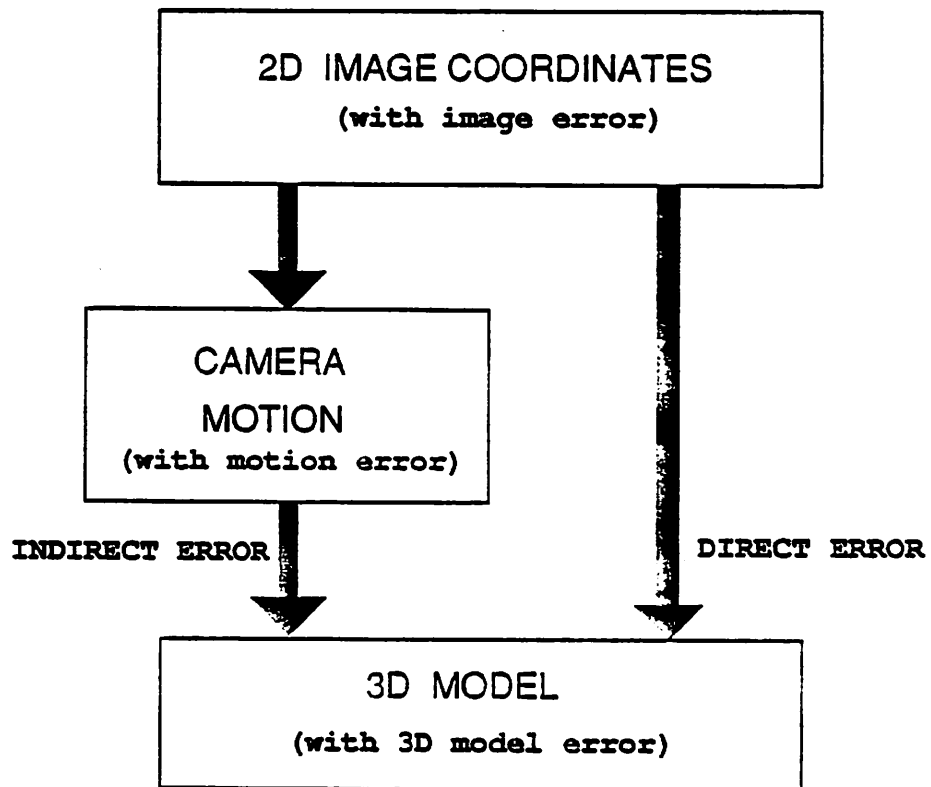


Figure 1.4: The Paths of Error in Constructing a 3D Model using SFM.

by incremental MFSFM. More specifically, this dissertation identifies an important source of error – the motion error – that has been neglected in all prior work. The mechanism for capturing the effect of this error (and other less important types of error) is developed here and is incorporated into a new incremental MFSFM algorithm which is capable of constructing fairly accurate 3D models. However, accuracy is obtained at the cost of computational complexity. A further goal of this work is to attempt to make the algorithm computationally feasible. In addition it will be shown that the acquired 3D models are sufficiently accurate for a specific task (position estimation) in robot navigation.

1.6 Outline of the Dissertation

In Chapter 2 we discuss the dominant techniques in Two-Frame SFM, analyzing the algorithms and approaches with respect to their assumptions and constraints. We then turn to the reasons for the failure of two-frame algorithms. Multi-Frame SFM is then presented as an obvious improvement over Two-Frame SFM. However, using many images introduces a new set of problems due to the large amount of information. In Chapter 2 current multi-frame approaches are analyzed based on how they constrain the input, or the camera motion, in order to deal with these problems. In contrast, the algorithm developed in this dissertation does not pose any such constraints.

Chapter 3 provides details of the algorithm – including the error analysis component – and motivates the approach theoretically. Experimental results are provided in Chapter 4. The ultimate test of any MFSFM algorithm lies in its application. The models obtained by our MFSFM algorithm are not only useful in recovering the shape of a room, or the topology of a terrain, but are also useful in determining the

position of a robot. In Chapter 4 the models constructed by the present algorithm are applied to the problem of determining the position of a moving robot.

After demonstrating the effectiveness of the new MFSFM approach both theoretically and experimentally, the computational aspects of the algorithm are analyzed in Chapter 5. We attempt to make the algorithm feasible for practical applications by considering ways of reducing the computational requirements. Future applications and extensions to this work are addressed in Chapter 6.

C H A P T E R 2

REVIEW OF ALGORITHMS

2.1 Introduction

Chapter 1 described how a 3D model of a robot's environment could be acquired using information from 2D images. However, the discussion was focussed primarily on standard approaches. In contrast, in this chapter we will first lay out various options and issues in Structure from Motion. Based on these issues, salient approaches to two-frame SFM algorithms will be analyzed. We will then posit reasons for the failure of two-frame techniques and, in doing so, motivate the need for using more than just two images (multi-frame SFM). Using many images introduces even more options; for example, as mentioned in Chapter 1, a multi-frame approach can either process all the images in one batch, or incrementally process one image at a time. Specific batch algorithms will be analyzed in this chapter followed by an analysis of incremental algorithms.

2.2 Options in SFM

Any SFM technique is faced with a variety of options at both the input as well as the output stage, although typically the input and output options are not independent. Different options may require different assumptions beyond the general assumptions outlined in Section 1.2.2. Whether any particular option is superior

under all conditions is an open research question; most likely a particular choice depends on the type of application at hand, on the computational complexity involved, or on the desired accuracy of the resulting 3D model. Let us now turn to the options and issues that are relevant to SFM algorithms; this outline will be relevant for the analysis of the SFM algorithms later in the chapter (Sections 2.3 and 2.5).

2.2.1 Representational Primitives

A 3D model of the environment can be represented in several ways. The simplest representation involves the most basic three-dimensional entity: 3D points. The most abstract representations are made up of volumetric primitives, such as generalized cylinders, voxels, or parametric volumes [41]. Lines and surfaces are primitives of intermediate complexity.

The simplest type of model – made up of points – may not be directly useful in certain applications, such as obstacle avoidance. In such an application, the set of 3D points in the model would have to be processed further to identify the volumes of the obstacles or at least their extents in space. It is likely that a volumetric representation (or even its approximation by planar surfaces) would be preferable for such a task. For other applications, such as determining the robot's position in a hallway, the most useful model is made up of lines, due to the predominance of edges (e.g. doors and baseboards).

Although the applicability of 3D models made up of points is somewhat restricted, such models are straightforward to construct using MFSFM, and are adequate for various tasks. For example, we show in Chapter 4 that the model constructed by our algorithm – consisting of points – performs well when applied to the task of robot position estimation.

2.2.2 Dense vs. Sparse Models

Regardless of the choice of the geometric primitives for representing the world, SFM algorithms can vary considerably in the denseness of the constructed models. Algorithms using points as primitives may either have been designed to obtain a dense model (a dense cloud of points) or a sparser model, the density of which depends on the number of points tracked from one image to the other. Similarly, the total number of surfaces and volumes in the model depends on the input, although surfaces and volumes by definition provide dense information locally. In general, dense models are superior to sparse models, except for the usual overhead of computational cost.

2.2.3 Shape vs. Structure

Certain techniques derive only the *shape* of the environment, i.e., all aspects of the environment except for its position relative to the camera. Such a model is generally represented in an object-centered coordinate system, without its location with respect to the camera. In contrast, there are techniques that either construct the model in a camera-centered coordinate system, or are able to transform the model from the object-centered coordinate system to a camera-centered coordinate system. A model derived by such techniques is referred to as *structure*. Consider the following example to bring out the difference between shape and structure. If the 3D model of, say, a desk consists of 3D coordinates of points on the desk with respect to a coordinate system fixed to the center of the desk and if the location of the desk with respect to the camera is not known, then the 3D model represents shape. On the other hand, if the 3D coordinates are represented with respect to the camera, such a model represents structure. Note that shape can be inferred from structure but not vice versa. Some techniques (e.g. those that involve orthographic projection as in section 2.6.4) can only compute shape and not the structure. The advantage of knowing the

structure of the environment over knowing just its shape is that the structure can be directly employed in robot navigation.

2.2.4 General vs. Constrained Camera Motion

By restricting the motion of the camera (perhaps in a straight line, with constant velocity, etc.) it is possible to simplify the process of extracting structure from an image sequence. Constraining the camera's movement, however, limits the applicability of the algorithms in realistic situations; furthermore, it is difficult in practice to precisely move the camera in the desired way due to e.g. mechanical limitations. All other things being equal, a general algorithm that does not constrain the camera motion (such as the one developed in this dissertation) is superior to one involving constraints.

2.2.5 Visual vs. Non-Visual Information

This dissertation concentrates on algorithms that use visual information for estimating the camera's movement (required to construct a 3D model of the environment). There are techniques that assume, on the other hand, that the motion of the camera is available based on information other than purely visual data. For example, Jezouin and Ayache [49] assume that the motion of the camera is given, while Szeliski [87] determines the relative motion between the camera and an object rotating on a turntable using grey codes painted on the turntable. Krotkov and Kories [51] obtain the motion of the camera using mechanical encoders. However, algorithms relying on mechanical and other similar sensors in order to obtain the precise camera movement tend to be inaccurate¹ or are limited to highly constrained situations (e.g. a turntable).

¹Although mechanical arms could measure their movements accurately, what is in question here is the accuracy in measuring the precise motion of the camera's image plane.

2.2.6 Computational Techniques

Algorithms vary in the computational techniques they employ in order to find the optimal solution (i.e. the optimal 3D model) from the image measurements. The computational techniques used in two-frame SFM find the maximum-likelihood estimate [90] for the given measurements. We refer the reader to Gelb [90] for a discussion of the equivalence of the various computational techniques including maximum-likelihood estimation, Kalman filtering, Bayesian estimation, and least-mean-square estimation.

2.2.7 Input Data

The most significant difference between SFM algorithms involves their choice of the type of input. Owing to their importance, the types of input typically assumed or computed as a first step are discussed in detail in the next section, and are employed as the main criterion in categorizing Two-Frame SFM algorithms. Within each category, the other options discussed above are also considered whenever relevant.

2.3 Two-Frame SFM Algorithms

In Two-Frame SFM the information is, by definition, obtained from a pair of images. A major difference between the various Two-Frame algorithms is the extent of pre-processing of the images that is needed to extract the required form of input. The most straightforward type of input needs no pre-processing; instead, the images from the camera constitute the input to the algorithm (these are called Direct Methods). The two other classes of algorithms involve a pre-processing stage in which optical flow information or feature correspondences are extracted; this information is the actual input to the Two-Frame algorithms.

Ullman [98] categorizes human visual perception of motion into two main types, continuous and discrete motion. He argues for the existence of two different mechanisms in the human visual system for dealing with the two kinds of motion. He proposes that the mechanism related to continuous motion (as in motion pictures) is directly based on the local changes of intensity information in images. On the other hand, the mechanism dealing with discrete (or abrupt) motion analyzes changes in the location of features (e.g. corners, lines, or regions). In fact, optical flow approaches and direct methods fall into the first category (continuous motion) whereas feature tracking methods fall into the second category (discrete motion).

In the following sections we will provide an analysis of SFM algorithms based on their choice of input.

2.3.1 Optical Flow

Every visible point in the environment projects to a point in both the first and the second image, except when the point is occluded in one image or the other. Hence for any point that is unoccluded in both images, a 2D vector in the coordinate frame of either image can be assigned to denote the displacement of the point from one image to the next. Note that the information provided by the 2D vector is effectively equivalent to that provided by knowing the image coordinate of the point in the second image; in both cases, the information specifies how a point has moved from one image to the other. Given this, the information available from a 2D vector along with the image coordinate of a point in the first image is sufficient for the standard SFM algorithm discussed in Section 1.2.3.

A standard technique for obtaining the 2D vector (the so-called correspondence problem, cf. Section 1.2.3) for a particular point in the first image involves searching for a point in the second image that is most similar to it with respect to its neighborhood, i.e., the surrounding intensity or brightness information. When such intensity

information is directly used to compute the 2D vector, the vector is referred to as Optical Flow.

Early work in optical flow techniques concentrated on studying the properties of optical flow from a theoretical perspective [50] [57] [15] [74]. The main focus of the work was to ascertain theoretically that optical flow combined with known camera motion is sufficient for determining distances from the camera to points in the environment. For example, Longuet-Higgins and Prazdny [57] established that the location of points in the environment can be fully determined based on the component of optical flow which is due to the camera's *translation* (as opposed to its rotation). To understand this intuitively, consider the hypothetical situation where the camera only rotates and does not translate. In such a situation, any point in the environment located along any given direction has the same optical flow regardless of how far it actually is from the camera. In a typical situation, however, a camera not only rotates but also translates. In such a situation the component of the optical flow that arises due to rotation provides no information for estimating distances, while the component of the optical flow that is induced by the translation contains all the information necessary for determining distances.

Since the early work was mainly theoretical, it was simply assumed that the precise optical flow as well as the actual camera motion were known. However, algorithms developed later – such as the seminal algorithms of Bruss & Horn [15] and Adiv [1] – do not make these assumptions. Instead of assuming that optical flow is known precisely, these algorithms deal with noise in optical flow. This is done by using information from as many flow vectors as possible, in order to decrease the distortion in the final result arising from the error in any individual flow vector. Furthermore, these algorithms do not assume that the camera motion is known. In fact, Bruss & Horn's key idea involves finding the best camera motion by minimizing the discrepancy

between the *measured* optical flow and the *predicted* optical flow. This basic idea was also employed by Adiv [1]. However, his algorithm differs from that of Bruss and Horn's in the way the camera's motion is estimated. Instead of solving for the camera's motion in one step, Adiv's technique has two stages for computing the two components of the camera's motion (i.e., translation and rotation). The first stage involves sampling the entire range of possible camera translations; at the second stage rotation is solved for at each sampled translation value. Finally, the best translation-rotation pair is selected based on how closely their prediction matches the measured optical flow. This technique has an advantage over comparable techniques [73] that first sample rotation instead of translation, since rotation involves one extra dimension.

Most two-frame techniques using optical flow (e.g. [33] [86] [47] [7] and [69]) differ primarily in how they estimate the camera's motion, since this is the area where the algorithm can potentially be improved. Once the camera motion has been estimated, the construction of a 3D model from two images (based on the measured optical flow and the estimated camera motion) is straightforward. For further details we refer the reader to Franzen's [30] comprehensive review of two-frame optical flow algorithms.

The model provided by optical flow algorithms consists of 3D points in a camera-centered coordinate system. The density of the model depends solely on the density of the optical flow. Although optical flow algorithms, in principle, need not constrain the type of camera motion, the actual formulations require that the camera does not move substantially between the two images due to the assumption of continuous (or instantaneous) motion. Recall also that the optical flow is typically computed by searching for the most similar intensity neighborhood; the larger the camera motion, the larger the region in the second image over which the search must be conducted, resulting in a substantial increase in computational cost.

A fundamental problem of optical flow is that it does not always reflect the relative motion of the object and the camera. Horn [41] illustrates this problem as follows. If a stationary camera views a featureless ball that is rotating about a fixed axis, then in spite of the relative motion between the ball and the camera, there will be *no* optical flow. Conversely, there *will* be optical flow in a situation where neither the ball nor the camera moves but the light source moves. Verri and Poggio [99] go so far as to claim that optical flow is equal to the actual displacement of a point from one image to the next only when "very special conditions are satisfied" (p. 171).

A further problem with optical flow is that it is difficult to determine the flow accurately. In fact, typically it is only possible to determine the 2D optical flow vector along one of two directions [41]. Consider a contour map based on intensity where a line has been drawn through points of equal brightness; such lines are referred to as iso-brightness contours. In extracting optical flow, iso-brightness contours from the first image are matched with the corresponding contours in the second image. The final goal is to match individual points along the two contours. However, unless the shape of the iso-brightness contour has some specific signature (such as a corner), it is not possible to exactly localize particular points along the contour. That is, optical flow can accurately capture how far a point has moved away from its original contour, but not how far along the contour the point has moved. In practice, obtaining the complete flow crucially depends on being able to first match corners or other significant points in the two images, and then propagating from these anchor points.

Research has revealed that precise determination of optical flow is impossible due to ambiguous matches, uniform texture (resulting in uniform intensity), and complex illumination. Recently new approaches have been developed to circumvent the need for obtaining optical flow; these are discussed in the next section.

2.3.2 Direct Methods

In algorithms called *direct methods* a pair of images taken from different camera locations is directly used as input. These algorithms share some characteristics with optical flow algorithms. Both types of algorithms belong to Ullman's [98] category of algorithms for continuous motion, i.e., they require small camera movement between the two images and they use the changes in brightness for computing the 3D model. As with most optical flow algorithms, direct methods have the advantage of producing dense models.

In spite of the similarities between direct methods and optical flow approaches, the two approaches differ in a significant way: direct methods collapse the two steps of optical flow algorithms into one. In the first step of optical flow algorithms, flow is computed based on the changes in brightness from one image to the next. In the second step, the optical flow is used to determine the camera's motion and to construct the 3D model. Since computing optical flow is inaccurate (cf. Section 2.3.1), direct methods have done away with the separate step for the computation of flow. This approach was first introduced by Negahdaripour and Horn [67]. They proposed that changes in brightness from one image to the next (corresponding to the first step of optical flow algorithms) can be directly related to the camera's motion and to the 3D model of the environment. However, owing to the lack of sufficient information from just the brightness changes (stemming from the problem with the iso-brightness contours discussed earlier for optical flow) they were able to solve only for a case where the environment consists of a single plane.

In a later direct method algorithm, Negahdaripour and Horn [68] did not constrain the shape of the environment to a plane, but assumed instead that the rotation of the camera is *known*. Given the known rotation, this algorithm is able to estimate the translation of the camera and to construct the 3D model. However, in practice it

is unrealistic to assume that the camera rotation is known. As an alternative which does not assume known rotation, Negahdaripour and Horn suggest discrete sampling of the set of all possible rotations in order to locate the best value. However, in this type of sampling scheme the correct rotation may be far from the closest sampled value (i.e. it may in effect be missed) because of the discrete samples. In yet another alternative technique, Horn and Weldon [44] provide a way to determine the rotation in the hypothetical case of no camera translation (cf. also earlier work by Aloimonos and Brown [4]).

Instead of constraining the camera motion, Taalebinezhad [88] develops an algorithm for general camera motion by assuming that a point in the world remains at the same location from one image to the next (i.e., the point is fixated on). Note that fixating on a point is equivalent to tracking the point over two frames; because of the tracking, additional information is gained. Taalebinezhad describes a simple software technique to create fixated images from non-fixated ones by image rotation. The problem with this approach is that Taalebinezhad has to make an additional assumption in order to solve for the camera movement. He arbitrarily assumes that the best camera motion involves the smallest possible value of $1/(\frac{1}{Z_0} - \frac{1}{Z})$, where Z_0 refers to the distance from the camera to the the fixated point and Z refers to the distance from the camera to any other point in the environment. In effect, this assumption considers the world to be a plane in front of the camera, which is seldom the case in reality. It is unclear how robust the algorithm is when the assumption does not hold.

The most general direct method to date is that of Hanna [34]. This algorithm decomposes the image into clusters of small square patches, each of which corresponds to a frontal plane in the 3D model. The algorithm starts with an initial guess of a set of 3D planes and of the camera motion, which are then incrementally adjusted based

on the changes in brightness from one image to the next. The algorithm alternately adjusts a) the estimate of the camera's motion and b) the position of the planes in the model, for a fixed number of adjustments. Furthermore, the entire process of adjustments progresses from low resolution images to higher resolution images. The results from this algorithm reported for outdoor environments look promising; however, since comparisons against ground truth are not reported, the results can be only evaluated qualitatively. Furthermore, this algorithm requires an initial set of guesses; it is unclear how well the algorithm will perform for arbitrary initial guesses.

The above discussion covers the most promising direct method algorithms reported. Since this approach to SFM is a recent one the full extent of the problems associated with this approach - or its potential - may not yet have been realized. There is reason to anticipate complications because direct methods essentially utilize the same basic brightness information as optical flow approaches do; recall that the change along the iso-brightness contour cannot be unambiguously identified (cf. Section 2.3.1).

2.3.3 Tracked Features

2.3.3.1 Advantages and Disadvantages

While optical flow approaches and direct methods rely on the information provided by changes in brightness from one image to the next, an alternative technique uses information obtained by tracking distinctive features across images. The chosen features may include corners of obstacles (points), edges of doorways (lines) and walls and posters (surfaces). For a set of features, their location in the two images constitutes the input to a SFM algorithm. For example, in a case where corners are tracked, their 2D coordinates in the two images are precisely what is required in order to solve Equations 1.1-1.5, and to construct a 3D model of the environment.

A drawback of SFM algorithms using feature tracking is the required pre-processing: first to identify features in the images and then to match features between images. This pre-processing step is not only computationally costly (on serial computers) but is also prone to inaccuracies and false matches. A further practical limitation of feature tracking algorithms is that the density of the constructed 3D models is limited by the number of distinguishable features in the environment; however, note that there are cases where many such features exist, such as in an office.

In spite of the limitations of feature tracking, this technique is more accurate than using the changes in brightness from one image to the next. In fact, 3D features such as corners typically correspond to distinct signatures in an iso-brightness contour (cf. Sections 2.3.1 and 2.3.2). Using tracked features is thus similar to selectively using only the most reliable information available to optical flow or direct method algorithms. A further advantage of feature tracking approaches over optical flow or direct methods is that feature tracking algorithms can deal with larger camera movements, since tracking distinct features (such as corners) is easier than tracking featureless points (such as a point on a uniform wall) across a large displacement in the image. All other things being equal, in practice a large camera translation results in a more accurate 3D model than a small translation. Since optical flow algorithms and direct methods involve continuous motion and instantaneous image changes, they are not designed to deal with large camera movement. Furthermore, small camera movements are required in order to satisfy the underlying assumption that changes of brightness are due to motion rather than to overriding changes in illumination.

Before turning to specific algorithms and solution techniques, we close this discussion on the advantages of feature tracking with a quote from Verri and Poggio [99]: *"Hence feature-based matching algorithms [...] are more appropriate than point-*

to-point ones to solve problems that rely on accurate recovery of the 2-D motion field, such as structure from motion."

2.3.3.2 Essential Matrix Algorithms

Numerous Two-Frame SFM algorithms that use tracked features as input have been reported in the literature. An early, seminal algorithm of this type is that of Tsai and Huang [95] which introduced a novel representation for recording the camera's motion, using a single matrix (referred to as an essential matrix, cf. Equation 1.8). They compute the elements of the matrix based on information from tracked points. The estimated matrix is decomposed into the rotation and translation components of the camera's motion, which are then used to construct the model of the environment. A major problem associated with the essential matrix approach is that estimating the elements of the matrix is not straightforward, since the elements are non-linearly related to each other. Note that this problem is not unique to the essential matrix approaches, but holds for other representations of motion as well. In order to be able to calculate the elements of the matrix in closed form, Tsai and Huang make the simplifying assumption that the elements are linearly related; this forces them to track a minimum of eight points instead of the usual five. Weng, Huang and Ahuja [106] attempt to improve the essential matrix approach by adding a second corrective stage. They use the estimated camera motion from a closed-form solution [105] as an initial guess for a later stage, where errors introduced by the first stage are corrected. In a further extension, the same authors (with Liu) adapt the algorithm to be used with lines instead of points [108].

2.3.3.3 Solution Techniques

Two-frame algorithms involving tracked features solve equations 1.1-1.5 (or their variants) in order to construct the best possible model of the environment based on

noisy measurements. The algorithms primarily differ in the techniques that they actually employ to find the best (or optimal) solution to the equations. The most commonly used solution techniques are singular value decomposition (SVD) techniques and gradient descent techniques [64]. SVD techniques (used, for example, in the algorithm of Tsai and Huang [95]) reformulate the system of equations into matrix form involving a single essential matrix which represents the camera motion; these techniques then determine the elements of the essential matrix by singular value decomposition.

The second category - gradient descent techniques - is exemplified by the Relative Orientation algorithm of Horn [43] which will be used in this dissertation. In the Relative Orientation algorithm, the camera's motion (i.e., the translation and rotation) is adjusted until the best fit to the set of equations is found; this is done by decreasing/minimizing (i.e. "descent") the deviation from the perfect fit.²

Apart from the above techniques, there exist algorithms that compute the best possible 3D model that is maximally likely, in a probabilistic sense, for a given input. This can be done either using a batch method [103], or a recursive method such as Kalman Filtering [90] as employed by Faugeras, Lustman and Toscani [26]. They linearize Equations 1.1-1.5 and cast them as a Kalman Filter to determine the optimal estimate of the camera motion. In this algorithm the Kalman Filter is used with two images only, rather than the usual case involving multiple images. Weng, Ahuja and Huang [104] point out that Kalman Filtering does not perform as well as a descent technique for Two-Frame SFM, since it requires linearizing around an arbitrary initial guess.³

²The algorithms discussed here involve a solution that minimizes the error in 3D. Another class of algorithms [83] recasts the equations in terms of the 2D image and seek to minimize the error in 2D.

³However, this performance does not apply to the multi-frame algorithm developed in this dissertation. See Chapter 3 and Appendix A for details.

We have discussed various two-frame approaches, of which feature tracking appears to be the most reliable one given the current level of knowledge. Let us now turn to the general problems of two-frame SFM and to the reasons for the inaccuracy of the 3D models obtained using only two images.

2.4 Problems in Two-Frame SFM

There are two types of problems in Two-Frame SFM. The first type involves inherent deficiencies arising from theoretical limitations, whereas the second type are problems due to practical considerations. We will consider both kinds of problems here.

2.4.1 Inherent Theoretical Restrictions

Even under idealized conditions (i.e., when there is no noise in the measurements obtained from the images) it is impossible in some cases to construct a unique 3D model of the environment. This is primarily due to the ambiguities involved in determining the camera's motion. For example, Faugeras and Maybank [27] have shown that using just five points results in ten possible solutions; recall that Two-Frame SFM algorithms in theory require only five tracked points (cf. Section 1.2.3). A further problem in determining the camera motion was pointed out by Horn [43] and Maybank [64]; this problem is independent of the *number* of tracked points, but rather depends on the *location* of the points. These authors show that if all the tracked points lie on certain quadratic surfaces (referred to as Critical Surfaces), then the camera's motion cannot be unambiguously determined based on image information. However, in a general case with a large number of points, there are at most three possible ways the camera could have moved [64].

Given the ambiguities discussed above, distinguishing the correct camera motions from incorrect motions may be difficult. However, in practice these theoretical limitations turn out not to be very serious. An unambiguous model can typically be constructed because more than five points are usually tracked (about 30 points in this dissertation), and because all these points seldom lie exactly on a critical surface. When the camera motion is approximately known, which is often the case in robot navigation, then this additional information can sometimes be used to disambiguate between the solutions.

2.4.2 Practical Problems

Practical considerations give rise to serious problems in Two-Frame SFM algorithms. In practice, inaccuracies in the final 3D model primarily stem from noise in the input to the SFM algorithm. When the input involves tracked features, errors due to the feature tracking algorithm corrupt the input.⁴ The noise from feature tracking is often small; however, in some instances when gross mismatches of features occur, the resulting error is considerable. For example, a corner of a door in one image may be incorrectly matched with a corner of a different door in the next image. Such gross errors (or outliers) can severely distort the estimated camera motion and the 3D model.

Errors in tracking can arise from poor image resolution, since the feature can be localized only up to the image resolution if no sub-pixel precision localization methods are used. Moreover, cameras produce noise (from electronic components or lens distortions) that blurs the images, making it difficult to precisely locate features. For example, it is difficult to localize a corner of an obstacle as a point even in a

⁴We will not consider optical flow approaches in this discussion since optical flow is typically more erroneous than tracked features. Nevertheless, the problems discussed in this section have counterparts in optical flow.

conventionally high resolution image (say one of 512×512 pixels). Although sub-pixel interpolation is possible, it is constrained by the quality of the image and the variation in intensity near the tracked feature. That is, if the image intensity varies sharply near a corner, then it may be possible to fit an intensity surface and to locate the corner point to sub-pixel accuracy. On the other hand, if the intensity surface is either noisy or uniform, sub-pixel interpolation is bound to fail.

Apart from problems due to incorrect tracking there are several factors that affect the accuracy of the 3D model obtained from a Two-Frame SFM algorithm. For example, Adiv [3] claims that in practice a change in the image could have resulted from a large number of camera motions (see Figure 1.2 for an example), only one of which is the correct one. In addition, he describes the typical factors that limit the accuracy of the 3D model, including a small field of view of the camera, small camera movement (i.e., small magnitude of translation), far away objects in the environment and a small number of tracked features. Furthermore, erroneous camera calibration results in inaccuracies in the 3D model; for example, if the camera's center is erroneously estimated by the calibration process, the 3D model will be distorted. In a general analysis, Weng, Huang and Ahuja [107] show that Two-Frame SFM tends to be inaccurate if the changes in the image are small and also show that accuracy improves if the changes from one image to the next are large. However, note that large image changes may give rise to gross tracking errors.

The practical problems and the inherent ambiguities discussed above reduce the accuracy with which the camera's motion can be estimated, which is a serious limitation of Two-Frame SFM. In fact, it appears that the limits of the two-frame approach in estimating the camera's motion have been reached. Using theoretical statistics (Cramer-Rao [20] [76] lower bounds of optimal estimation), it is possible to establish the maximum accuracy of an estimate of the camera's motion for a given noise level in

the input. Weng, Huang and Ahuja [107] argue based on simulations that the performance of their two-frame algorithm has reached the theoretically possible maximum accuracy. However, even this near-optimal algorithm (as well as all other two-frame algorithms) suffers from errors in estimating the camera motion which significantly distort the 3D model. They state that for a 512×512 image with Gaussian noise (variance of 1 pixel) "*under a small motion with 2-pixel maximum disparity (average disparity is roughly equal to 1 pixel), the errors in translation are bounded below by 60%*" (p.364). They conclude that recovering motion parameters from small motion (disparity of a few pixels) is unreliable.

Furthermore, it has been shown that even a small error in the estimated rotation of the camera's movement can result in large errors in the 3D model (Dutta, Manmatha, Riseman and Snyder [23]). Dutta and Snyder [25] simulate a realistic situation,⁵ and statistically show that the distance from the camera to most points in the environment has an error of 10% (when the input contained a 1-pixel error). In another simulation they conclude, "*when depths are in the range of 5 to 20 times the total translation we still get a 45% relative error in depth when the rotational parameters are changed by 0.1° .*" (p. 109). This means that even algorithms that are shown to be optimal in estimating the camera's motion are still not able to produce accurate 3D models.⁶ Given this situation, a reasonable approach is to move from using just two images to using more than two images with the aim of acquiring more accurate 3D models.

⁵In the simulation no particular Two-frame SFM algorithm was employed to determine the camera motion; rather, the correct value of the camera motion was corrupted by a small error in order to simulate errors that occur in Two-Frame SFM algorithms. The simulation involves a camera with a FOV of 45° , focal length of 309 pixels, and image size of 256×256 pixels.

⁶The above error analyses apply both to feature-based and flow-based approaches. There is hardly any examination of errors in direct methods apart from work by Weldon and Lui [102]. For the restricted case of a planar patch and *accurately known* camera motion, they show that the error in estimating the distance from the camera to the patch is as small as 3 % for a patch that contains a few hundred pixels.

2.5 Multi-Frame SFM Algorithms

Multi-frame Structure from Motion (MFSFM) approaches derive a 3D model of the environment using more than two images. The key idea of a multi-frame approach is to exploit redundancy to reduce error arising from noisy measurements. Many different views of a scene are used to construct better 3D models than what can be obtained from just two noisy images. The most significant difference between MFSFM algorithms is the manner in which the images are processed. The algorithms of one type process all the images at once in a single batch (Batch Methods) and the algorithms of the second type process images incrementally, one at a time (Incremental Methods).

2.6 Batch Methods

Batch methods attempt to construct the 3D model by assuming that *all* images – from every camera position in the image sequence – are available before processing is begun. However, processing all the images at once involves a large number of variables which makes the computation unwieldy and can lead to inaccuracies, due to finding locally optimal results (as opposed to the global optimum). Each camera motion involves 6 variables (3 for translation and 3 for rotation) and each 3D point involves 3 variables (x, y, z). If the 3D model is constructed based on $m + 1$ pictures (from m camera movements) and represented using n features (say, n 3D points), then the most general batch method involves $6m - 1 + 3n$ variables (the scale ambiguity in SFM, cf. Chapter 1, decreases the total number of unknown variables by 1). Therefore for, say, 10 camera movements and for 30 points, the number of unknown variables is 149.

Batch algorithms have typically reduced the number of variables by restricting the way the camera moves, assuming that any deviation from these constrained motions

Table 2.1: Batch MFSFM algorithms showing the number of motion variables (general case: $6m - 1$) and average accuracy computed from the reported results (m denotes the number of camera movements).

ALGORITHM	MOTION VBLS	RESULTS ON IMAGE SEQUENCES	
		ACCURACY	COMMENTS
Sawhney et. al. [80]	3	0.9% (25 images)	Motion as expected
Broida & Chellappa [14]	7	2.5% (12 images)	Motion as expected
		40.3% (16 images)	Unexpected motion
Kumar et. al [54]	7	-	Only qualitative results
Franzen [29]	14	7.4% (8 images)	Unexpected motion
Taylor et. al. [89]	$3m - 1$	-	Only synthetic results
Tomasi et. al. [94]	$3m - 1$	2.4% (150 images)	Object far away

can be dealt with as noise. In practice, however, the deviations may be large or biased and therefore cannot be accounted for simply as random noise. Table 2.1 lists the batch MFSFM algorithms reported in the literature based on the number of variables used to approximate the underlying robot motion; since all algorithms share the $3n$ variables (i.e. the 3D coordinates of the points), these are not included in the table. In the following section these algorithms are discussed in more detail, in ascending order of generality of allowable motions.⁷

2.6.1 Uniform Rotation

A special multi-frame approach has been developed by Sawhney, Oliensis and Hanson [80] in which the camera motion is assumed to be rotational about a fixed axis.⁸ In this simple case of restricting the camera motion to rotation, three variables

⁷There is an important similarity among many batch algorithms, namely constructing 3D models in object-centered coordinate systems. For certain algorithms (e.g. [13]) it is claimed that using an object-centered coordinate system is preferable over using a camera-centered coordinate system.

⁸Note that this is equivalent to the camera being stationary while the objects in the environment rotate instead, which is how the experiments for this algorithm were actually conducted.

are sufficient to completely represent the camera movement. Assuming purely rotational motion permits a closed-form solution for the three variables representing the rotation, as well as for the 3D coordinates of a given point. The solution is based on the shape of the trajectory traced out by a point while the camera rotates. Sawhney et. al. show that extracting the shape of the trajectory of an individual point from image information is inaccurate, and then demonstrate a robust way of combining information from several points located on the same object.

In an image sequence of 25 frames, where the camera's motion was made to adhere to the motion restrictions, the average error of the distances from the camera to points in the 3D model was 0.9%. Although this is a very good result, the applicability of this work is severely constrained by the requirement of purely rotational motion. It is clear from the nature of the formulation of this technique that it cannot be generalized to arbitrary camera motion. However, this approach could be used for acquiring models of objects, similar to the work of Szeliski [87] where objects rotate on a turntable (cf. Section 2.2.5).

2.6.2 Uniform Translation and Rotation

An algorithm that allows for a more general motion than the previous technique was formulated by Broida and Chellapa [13] [14]. They explicitly assume that the objects in the environment - as opposed to the camera - move and turn, and that the motion occurs at a constant rate. They represent the actual motion of the object by a uniform translation velocity and by a constant rotation, resulting in approximation of general motion; these terms correspond only up to the first order terms in the Taylor's series expansion of the general motion equations. Such a restricted representation reduces the number of motion variables from $6m - 1$ (general case) to 7. The authors solve for the motion and for the 3D coordinates of the points by applying a gradient descent technique (conjugate gradient) on the entire batch of image

measurements. Broida & Chellappa [13] construct a 3D model from 12 images with an average accuracy of 2.5% when the object moves and turns at a constant rate as expected. In a second image sequence of 16 frames, in which the object violates the expected motion, the average error in the 3D model is 40.3%.

Kumar, Tirumalai and Jain [54] use the same formulation as Broida and Chellappa but employ the Levenberg–Marquardt method [60]⁹ (instead of conjugate gradient descent) to find a solution, claiming faster convergence. They also present a quasi-incremental version of their algorithm that only operates on a fixed number of recent images, discarding older images. In this version, the object motion and the 3D model for the current batch of images are not computed from scratch, but by starting with the solution of the previous batch of images (which typically permits quick convergence to the optimal answer). The advantage of this technique is that it can track variations in object motion, unlike Broida and Chellappa’s algorithm which assumes a fixed type of motion over the entire sequence. Kumar et. al. do not report quantitative results for real image sequences.

Rather than assuming uniform velocity, Franzen [29] goes a step further in allowing for uniform 3D acceleration. He solves for the motion and the 3D model using a closed-form solution by removing the non-linear term, $\frac{1}{z^2}$ (where z is the distance to a given point from the camera) from his variants of equations 1.1–1.5. These variant equations represent what the algorithm tries to minimize in order to find a solution, i.e., the difference between the measured image coordinates of points and the predicted image coordinates. The effect of removing the non-linearity is that the constructed 3D model ends up being “more compact than it should be” as observed by Franzen [29] (p. 60). In an attempt to correct (i.e. inflate) the model, heuristics are later applied.

⁹The Levenberg-Marquardt [60] technique combines steepest gradient descent (when far away from the minimum) with inverse Hessian methods (near the minimum).

The kind of motion that Franzen allows (also referred to as chronogenous motion) requires 14 variables for its representation. Reasonably good 3D models are reported for real-world image sequences. For example, based on 8 images of the UMass Rocket-Field Sequence, a 3D model was constructed with an accuracy of 7.4%, although the motion of the vehicle that obtained the sequence was not entirely chronogenous (see [24] and Chapter 4 for details regarding the UMass sequence).

The batch MFSFM algorithms discussed so far share the property that a fixed number of motion variables are used to represent the movement of the camera (or the object) regardless of the actual movement of the camera across the entire image sequence. On one hand, such an absolute limit on the number of variables results in robustness of the algorithms when the actual motion is similar to the allowable motion; but if the actual motion is considerably different from the allowable motion, it is reasonable to expect the algorithms to fail.

2.6.3 Planar Motion

The two remaining batch algorithms reported in the literature represent the camera's motion using a set of variables, where the size of the set is proportional to the number of times the camera actually moves.

In the simplest algorithm involving a varying number of motion variables, Taylor, Kriegman and Anandan [89] constrain the motion of their camera to a *plane*. By imposing this constraint, they reduce the number of variables representing the camera motion from $6m-1$ (the unconstrained case) to $3m-1$. To solve for motion and the 3D model, Taylor et. al. minimize the mean square difference between the actual image measurements and the 2D image projection of the currently estimated 3D model. They split this minimization into a number of smaller problems that are computed in parallel using the Levenberg-Marquardt technique. Inverting the Hessian matrix in the Levenberg-Marquardt method is usually expensive, but Taylor et. al. avoid this

by optimizing over small subspaces and inverting only 2×2 matrices. The parameters of interest are divided into three mutually exclusive sets: the coordinates of each tracked point, the camera positions, and the camera orientations. Optimization over these sets of parameters is decomposed into subspaces made up of only the point's coordinates, the camera's position, or the camera's orientation. Whether the fast parallel scheme of Taylor et. al. is applicable also to non-planar cases is an open research question. No real image experiments have been reported for this algorithm in the literature.

2.6.4 General Motion

Of all the batch methods, the work by Tomasi and Kanade [94] is the only unrestricted one in terms of camera motion. However, their algorithm crucially depends on a different assumption, namely that the camera image is formed by orthographic projection.¹⁰ By exploiting this assumption, Tomasi and Kanade separate the *shape* of the environment from the camera's motion in an ingenious fashion. They use the linearity of the orthographic projection in order to decompose the image measurements (over a sequence of images) into the camera's rotation and into the object's shape. The camera's translation does not need to be calculated because of the assumption of orthographic projection; i.e., only $3m - 1$ variables need to be utilized. This approach works very well in determining the shape of objects which are far away, or very small, or shallow in depth (i.e., the effect of perspective is negligible).

A disadvantage of this approach is that the constructed 3D model is not available in a camera (or robot) coordinate system, which is necessary for tasks such as obstacle

¹⁰Recent related work by Poelman and Kanade [72] involves *paraperspective projection* which is a development over strict orthographic projection. This work takes into account the variation in the size of an object depending on its distance and direction from the camera, which is ignored in orthographic projection. Although this technique may be applicable to a larger range of real-world scenarios than algorithms that assume orthographic projection, experiments involving real-world imagery are yet to be reported [72].

avoidance. However, this is not a serious problem since there exist ways of determining the location of objects based on their shape and based on an image of the object from the current position of the robot. For example, pose determination algorithms such as that of Kumar [52] could be used. A more serious difficulty with this approach is that it cannot in general be employed for the task of obstacle avoidance, since the objects to be avoided occur close to the camera and have high perspective distortion.

The shape of the 3D model constructed by Tomasi and Kanade's algorithm has an accuracy of 2.4% for a reported 150-image sequence [94]. Based on the results, this approach appears to be promising. However, it is important that future research establish the role of the assumption of orthographic projection, by applying this approach to quantitative experiments on real-world image sequences involving nearby obstacles.

In conclusion, batch methods cannot simultaneously solve for the large number of variables required to represent general motion in unconstrained environments, due to the computational complexity. We have seen how various batch MFSFM algorithms have either restricted the camera motion or the camera model in order to get around the difficulties of batch methods. On the other hand, these restrictions and the fact that all images are processed together limit the applicability of batch methods to general robotic applications where real-time decision-making and control are crucial.

2.7 Incremental MFSFM

Unlike batch MFSFM methods, incremental methods do not assume that all past images are available; rather, they use only the most recent image(s)¹¹ together with an estimate of the 3D model derived from the previous images. Incremental methods

¹¹Typically the current image or the current and previous images are used.

attempt to update the estimated 3D model by incorporating information from the most recent image(s).

In this section we will consider incremental MFSFM algorithms in detail. However, before we analyze the various algorithms, we will discuss the general assumptions made by incremental algorithms as well as present the options particular to incremental algorithms.

2.7.1 General Assumptions of Incremental MSFSM

The three basic assumptions of Two-frame SFM (Section 1.2.2) also hold for standard MFSFM. In addition, the following assumptions are typically made in incremental MFSFM:

1. The current model is updated every time a new image is obtained.
2. An estimate of the error in the current model is available (3D model error in Figure 1.4).
3. The estimated covariance of the 3D model error reflects the actual error.

An initial model can be derived by applying Two-Frame SFM to the first pair of images; this is the approach used in this dissertation. Alternatively, the initial model can be arbitrarily set as a plane (comparable to a wall directly ahead) [37].

The first assumption represents the fundamental basis of incremental MFSFM according to which the current model is refined incrementally. Concerning assumption 2, consider the situation where no estimate of the error is available; i.e., we do not know how reliable different portions of the model are. In this case incrementally updating the model would be equivalent to blindly averaging the model with the information obtained from the new image. All incremental MFSFM approaches assume

that an error estimate is available; they vary in how much of the error information is taken into consideration.

The actual 3D model error arises from noise due to inaccurate tracking of points from one image to the next (image error in Figure 1.4). This actual 3D model error is assumed to be best approximated by a Gaussian distribution. The mean of such a distribution corresponds to the 3D coordinates of the model, while the covariance of the distribution represents the reliability of the coordinates (assumption 3). Although in practice the error distribution is not strictly Gaussian[62], experimental work [91] suggests that the covariance might adequately capture the error for the purposes of incremental MFSFM.

2.7.2 Options in Incremental MFSFM

The input of an incremental MFSFM algorithm is identical to the input of a Two-Frame SFM algorithm, i.e., 2D image coordinates of tracked points; the only difference is that due to the incremental step the points end up being tracked over many frames as opposed to only two. An incremental algorithm typically works as follows: initialize, move, transform, update and iterate (cf. Chapter 3 for details).

There are two important issues that an incremental MFSFM algorithm faces: (i) how to represent the error in the 3D model, and (ii) how to transform the error in the model to the new camera coordinate system. Let us consider these in turn before using them in the analysis of incremental algorithms.

2.7.2.1 Representing the 3D Model Error

An important distinction between incremental algorithms lies in how the algorithm represents and corrects for the 3D model error. In order to refine the 3D model, using information from the new image, it is crucial that an estimate of the error in the

model is available. If this error reflects – in a probabilistic sense – the actual error in the model, then incremental refinement is possible.

One representation of the 3D model error is a complete covariance matrix. If the model consists of n 3D points, then the error in the model is represented by a covariance matrix of size $9n^2$ (i.e. $3n \times 3n$). In previously reported MFSFM work, the $9n^2$ elements of the covariance matrix are approximated by much fewer than $9n^2$ elements. The simplest class of approximations involve using only n elements to represent the 3D model error. Each of these n elements approximates the expected error in distance from the camera to one 3D point. Incremental algorithms vary in the number of elements used to represent the error (from n to less than $9n^2$); this will be further discussed in Section 2.7.3.

One of the main points of this dissertation is that elements of the covariance matrix that have typically been neglected (the off-diagonal terms) turn out to contain important information. These off-diagonal terms capture a major source of the 3D model error due to the error in the estimated camera motion (motion error in Figure 1.4). The motion error affects all the 3D coordinates of the model in a systematic way; this is reflected as correlations between the errors of each point. We return to this point in Chapter 3. Recording the correlations in the covariance matrix provides a means of estimating and compensating for the effect of the erroneous camera motion. Let us now consider how such a covariance matrix can be transformed across coordinate systems.

2.7.2.2 Transforming the Error Across Coordinate Systems

When the camera moves to a new position, the image taken from that position involves a new coordinate system. Since the existing model (based on previous images) and the new image information have different coordinate systems, one of them has to be transformed to the coordinate system of the other. An important reason for

preferring the new coordinate system at the current location of the sensor is that the 3D model is then readily available for immediate use.

Transforming to the new coordinate system requires that both the model and its estimated 3D model error be transformed to the new coordinate system. Since the transformation is done based on the estimated camera motion (between the previous image and the new image), the motion error ends up further corrupting the transformed model during the process of transformation. Let us refer to this error as the *transformation error* (cf. Appendix A for the equivalent Kalman Filtering terminology).

Various algorithms have approximated the transformation error in different ways, including a simple heuristic which inflates the estimated error by a fixed value. The heuristic - *constant age-weighting* - used in these algorithms accounts for the transformation error by decreasing the importance of the past estimate by a constant; i.e., the older an estimate is, the less important it gets. Using constant age-weighting is preferable to completely ignoring the transformation error. However, it cannot sufficiently take into account large errors, especially those due to rotation. Furthermore, constant age-weighting is not effective, since it does not differentiate between more erroneous and less erroneous transformations. More advanced recent error analyses involve first-order error analysis.

2.7.3 Incremental Algorithms

The algorithms to be discussed here are classified according to their restrictions on the motion of the camera. Furthermore, each algorithm will be analyzed based on how they capture the transformation error and how many elements of the covariance matrix they use for capturing the 3D model error (as summarized in Table 2.2).

Table 2.2: Recursive MFSFM algorithms showing the number of elements used to represent the 3D model error (general case: $9n^2$).

ALGORITHM	ERROR ELEM.	RESULTS ON IMAGE SEQUENCES	
		ACCURACY	COMMENTS
Matthies et. al [62]	n	0.5% (11 images)	Only pure translation
Shigang et. al. [82]	n	15% (40 images)	Only planar motion
Heel [37] [38] [40]	n	-	Only qualitative results
Ando [6]	n	-	Only simulations
Stephens & Pike [85]	$9n$	1% (50 images)	Results for only 1 point
Sawhney & Hanson [79]	$18n$	3.0% (6 images)	Only frontal planes of shallow objects
		3.4% (10 images)	
Cui et. al. [21]	$9n$	fluctuates	No ground truth

2.7.3.1 Camera Restricted to Pure Translational Motion

As an instance of the most constrained type of camera motion, Matthies, Szeliski and Kanade [62] [63] obtain dense 3D models from a sequence of images by assuming that the camera moves (translates) parallel to a fixed line which is the horizontal scanning direction of the image. Using the resulting one-dimensional flow and *known* camera motion, the model is reconstructed by triangulation. Matthies et. al. use constant age-weighting to take care of the transformation error, and use only n elements to represent the 3D model error. By applying the algorithm in a situation where the camera's movement was precisely controlled (optical bench) a model was obtained using 11 images with an error of 0.5%.

2.7.3.2 Camera Restricted to Planar Motion

In a slightly more general case, Shigang, Tsuji and Imai [82] constrain the motion of their camera to a plane. Their algorithm involves a novel technique for determining camera *rotation* based on the vanishing point of horizontal lines in an image. The

camera's translation, on the other hand, is obtained using shaft encoders on the robot's wheels. Using the complete camera motion, they construct a model made up of vertical 3D lines and refine the model over time. Shigang et. al. use only n terms to approximate the 3D model error, even though they consider more general camera motion than Matthies et. al. When they allow the camera to move freely on a plane, the error in the final model is 15% after 40 images.

2.7.3.3 General Motion with Restricted Experiments

The remainder of the algorithms to be discussed do not explicitly constrain the camera's movement. However, for most of these algorithms only simulations or restricted camera motions have been considered in the reported experiments. After discussing these algorithms we will turn to the algorithms which provide experimental results for general camera motion.

Heel [37] [38] uses optical flow as input and obtains a dense 3D model using Kalman Filtering. All experiments reported are for the case of pure translation. Heel does not correct for transformation error, and he approximates the 3D model error with just n elements. Reasonably good qualitative results are reported for two experiments where the camera moves precisely in a straight line. In these experiments the approximation of the covariance matrix by n elements seems to be adequate to obtain qualitatively reasonable 3D models, but this performance may well be due to the highly constrained motions (pure translation on an optical bench).

In Heel's later work [40] (with Ray) the noisy optical flow is eliminated and motion is computed using a direct method. Even this improved method does not consider the fact that the transformation is erroneous; again only n elements are used to represent the 3D model error. All experimental results are qualitative and involve pure translational camera motion.

Ando [6] attempts to develop dense 3D models which incorporate completely arbitrary motion, without any constraints. He gives a detailed computational scheme for straightforward Kalman filtering using optical flow. Unlike other MF/SFM algorithms, his does not require strict rigidity (cf. Section 1.2.2). This is achieved by using a term that describes the deviation from rigidity. An interesting side effect is that the output of Ando's algorithm behaves as predicted by psychophysical experiments involving two rotating transparent cylinders of dots.¹² Ando uses constant age-weighting and uses n elements to capture the motion error. Since his experiments involve only simulations, it remains to be seen whether this approach works in real-world situations.

Stephens and Pike [85] use a straightforward Kalman filter to refine the 3D coordinates of each tracked point. Unlike Heel and Ando, Stephens et. al. use $9n$ elements to approximate the $9n^2$ covariance matrix. The experiments reported involve the camera moving almost directly ahead. Quantitative results are reported only for *one* point located about 100 m from the camera; the error in distance is about 10% after 10 m of forward motion (over 25 images). After 50 images, the error is reduced to 1%.

Although the results from some of the recursive algorithms discussed so far look promising, given the lack of generality of camera motions it is unclear whether these algorithms are useful for general applications.

2.7.3.4 General Motion with Unrestricted Experiments

Unlike all the algorithms discussed so far – which represent the scene with points and/or lines – Sawhney and Hanson [79] attempt to identify flat objects with negli-

¹²When two cylinders of the same size rotate at different speeds but at the same depth, humans perceive them as occurring at different depth. Since Ando's SFM algorithm maximizes overall rigidity – i.e. it does not require strict rigidity – it computes the most rigid interpretation of all the points taken together, which turns out to be similar to the human percept.

gible depth (“shallow structures”), which then constitute the 3D model.¹³ In this algorithm, the transformation error is taken into account, and $18n$ elements are used to represent the model error, where n denotes the number of shallow objects. Each shallow object consists of three lines, the error of which is represented by six elements in the covariance matrix (cf. Equation 6, 9, and 11 in [79]). Although this algorithm can deal with general motion, it assumes that the objects modelled have surfaces sufficiently distant in the environment so that the models will be frontal planar surfaces. Using this assumption, reasonably good 3D models have been constructed (with errors of 3.0% and 3.4% in the two sequences reported in [79]).

The first to report results on real images for an unconstrained environment and arbitrary motion are Cui, Weng and Cohen [21]. Their algorithm is also the first to take into account the transformation error as well as to use $9n$ elements of the covariance matrix. However, despite the careful error analysis, the experimental results for general motion do not look very promising. In the only reported real-world experiment, the error in the 3D model fluctuates randomly between $6mm$ and $32mm$. Since the true distances corresponding to this error are not reported, it is impossible to evaluate these figures apart from noting that the error does not decrease over time, but rather fluctuates randomly.

The above results suggest that there is much room for improvement in MFSFM for real-world applications, where neither the motion, the camera, nor the environment are constrained. The following chapter describes in detail our MFSFM framework which does not impose any such constraints. This algorithm uses the full $(9n^2)$ covariance matrix, i.e. all the cross-correlations. Highly accurate 3D models have already been reported for this cross-correlation-based algorithm by Thomas and Oliensis [93]. Detailed experimental results are presented in Chapter 4.

¹³Such a sparse model has been accurately extended using Kumar’s model-based approach of pose recovery, while tracking and triangulating on new points [52].

CHAPTER 3

A FRAMEWORK FOR MULTIFRAME STRUCTURE FROM MOTION

In the MFSFM algorithm developed in this dissertation, the goal is to incrementally update the 3D model of the world using information from new images obtained as the camera moves. The algorithm is developed for unconstrained environments, and allows arbitrary camera motion. This chapter discusses the details of this incremental algorithm. Most of the discussion focusses on how to capture the error in the 3D model and in the new image information, such that the error analysis supports intelligent updating of the 3D model based on the new image.

Recall from Chapter 1 that a 3D model of the environment obtained from SFM is entirely based on the 2D image coordinates of tracked points or features (assuming that the camera calibration is known). The main goal of this chapter is to provide a mechanism to represent and estimate the error in the 3D model stemming from the error in the image coordinates of tracked points (see Section 2.4.2 for a discussion of errors in tracking). In order to refine the 3D model over time, it is desirable that the estimate of the error in the model accurately reflects the actual error. The most important difference between this work and previous incremental MFSFM research is that the present work provides a framework within which *all* the elements of the 3D model error (i.e. all $9n^2$ elements, cf. Section 2.7.2.1) are represented. Recall from Chapter 2 that the most advanced incremental algorithm in the literature used only $9n$ elements; the algorithm ignored all the terms in the covariance matrix that represented the cross-correlations between any two points in the 3D model. In this work

these cross-correlations are explicitly computed and used in incrementally updating the initial 3D model.

3.1 The Cross-Correlation-based Incremental Algorithm

It is assumed that the camera moves from one location to the next, and obtains a new image at every location (recall Figure 1.3). The goal of the algorithm is to construct a 3D model consisting of points (such as corners of walls, doors, and obstacles) tracked across the images.

3.1.1 The Steps of the Algorithm

The cross-correlation-based algorithm works incrementally as follows. Let us denote the i th multi-frame model by H_i , the i th two-frame model by W_i , and the i th image by I_i (Figure 3.1 represents the first iteration of this incremental algorithm):

1. **Initialize.** Construct an initial model W_1 from the first 2 images I_0 and I_1 . Estimate the error, $Cov(W_1)$, in this model. At this initial step, H_1 is equivalent to W_1 .
2. **Move.** Move the camera to a different position; construct a two-frame model W_2 from the current image I_2 and previous image I_1 . In Figure 3.1, constructing the two-frame model corresponds to the dotted box.
3. **Find Error.** Estimate the 3D model error in the new two-frame model, $Cov(W_2)$. In Figure 3.1, this step corresponds to the three darkened boxes. This stage involves the crucial difference between this work and previously reported work.

FIRST ITERATION

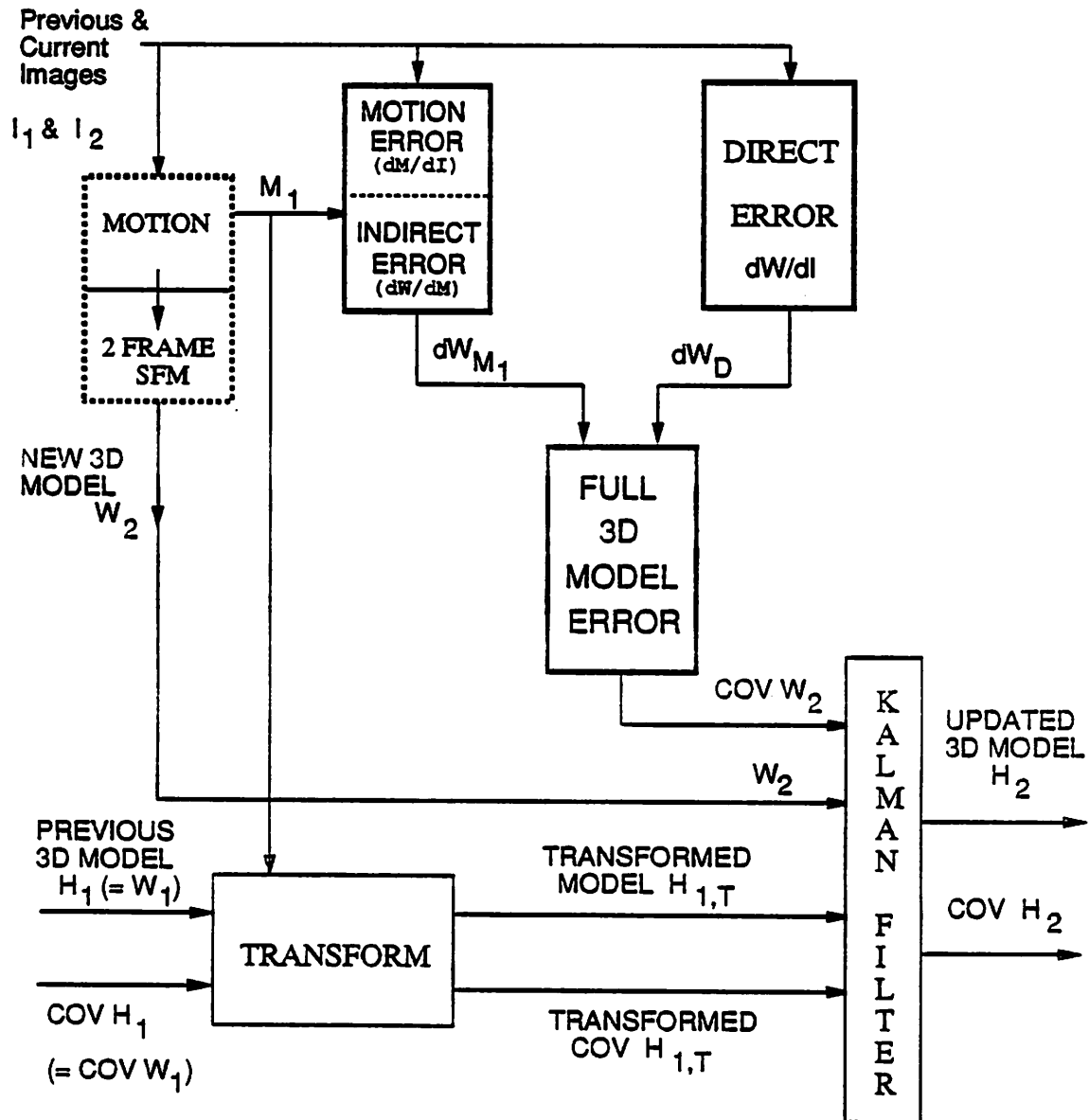


Figure 3.1: The first iteration of the cross-correlation-based algorithm. The dark boxes indicate the bulk of what is done in this chapter. The dotted box denotes a Two-Frame SFM algorithm due to Horn that we have adopted. The remaining boxes correspond to steps of a Kalman Filter.

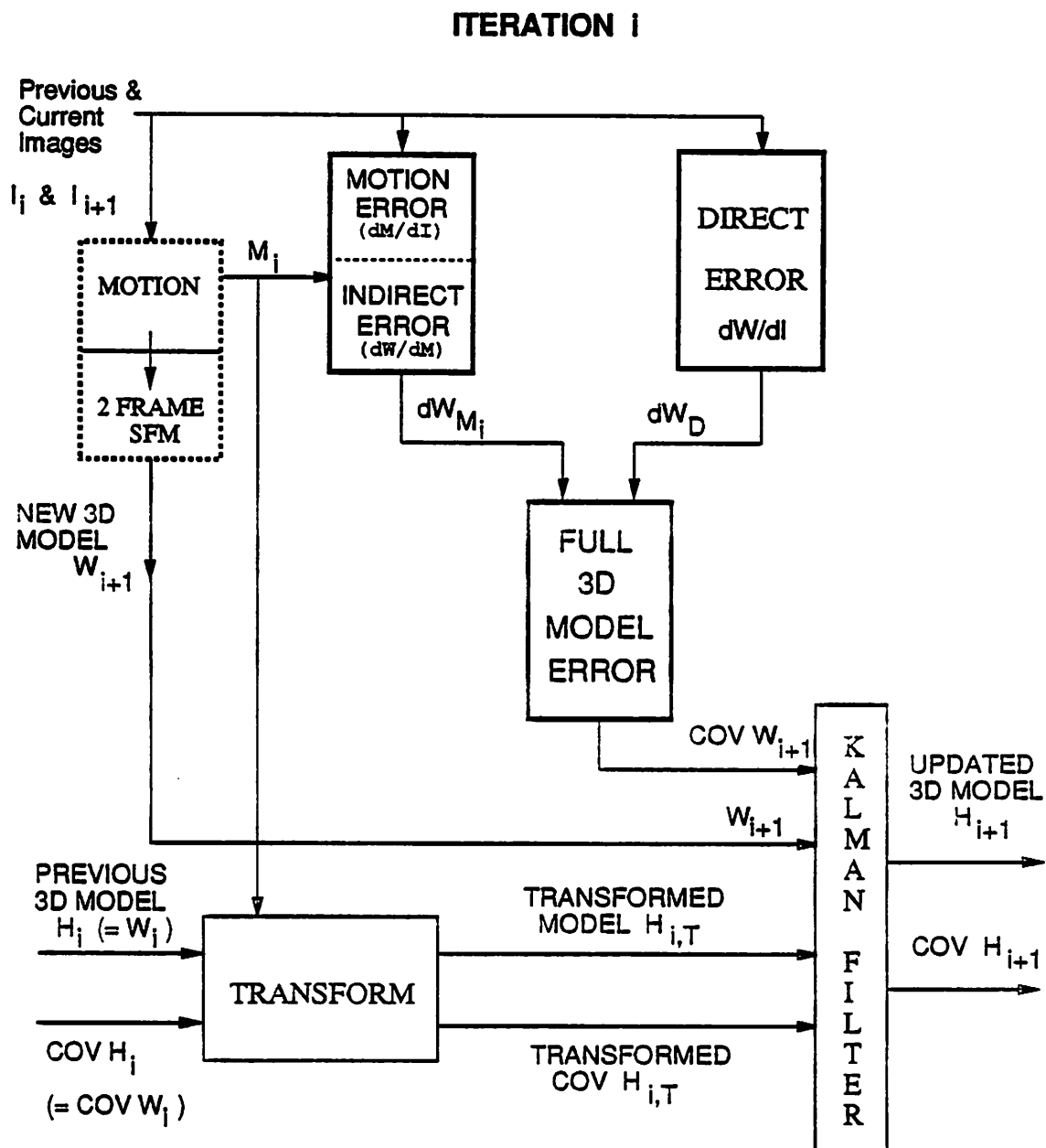


Figure 3.2: An i th iteration of the cross-correlation-based algorithm. The dark boxes indicate the bulk of what is done in this chapter. The dotted box denotes a Two-Frame SFM algorithm due to Horn that we have adopted. The remaining boxes correspond to steps of a Kalman Filter.

4. **Transform.** Transfer the previous model (H_1) and its error $Cov(H_1)$ to the new camera coordinate system of image I_2 . The result is $H_{1,T}$ and $Cov(H_{1,T})$.
5. **Update.** Update the transformed model using the newly acquired two-frame model W_2 . This step involves Kalman Filtering. Compute the error $Cov(H_2)$ of the updated model (H_2).
6. **Iterate.** Go to Step 2 (increment subscripts by 1). See Figure 3.2 for the steps of the algorithm at iteration i .

The initial model W_1 is a set of 3D coordinates of the tracked points in the environment obtained by the two-frame technique (that uses the steps outlined in Section 1.2.3), using the first and second image.¹ The error in this model is then estimated; a reliable estimate of the error is crucial for an accurate MFSSFM algorithm. Determining the error of the two-frame model is the major contribution of this chapter (Sections 3.1.2-3.4). The techniques developed here are not only applicable at the initial step, but are also useful every time Steps 2-3 are executed, i.e., whenever a new image is obtained. Every time the camera moves, the most recent pair of images is used to obtain a new two-frame model of the environment, the error of which is again estimated.

The goal of the algorithm is to fuse the initial model H_1 with the newly acquired two-frame model W_2 . Before this goal can be realized, the previous model and the new model have to share the same coordinate system. Note that the previous 3D model H_1 is in the camera coordinate system of the previous image I_1 , whereas the new model W_2 is in the camera coordinate system of the current image I_2 . Either model could, in principle, be transformed to the other coordinate system. However,

¹Note that reasonably accurate tracked points are assumed as the input to the algorithm in the experiments considered in Chapter 4. It is assumed that tracking is at least accurate to 1 pixel (in a 256×256 pixel image).

it is best to have the updated model in the current camera coordinate system, since it is useful for the robot to know the environment from its current position. The coordinate transform can be done as follows:

$$\mathbf{H}_{1,T} = R_1 \mathbf{H}_1 - T_1 \quad (3.1)$$

where R_1 represents the rotation of the camera and T_1 represents its translation between images I_1 and I_2 . (Recall that the estimates for R_1 and T_1 are computed by the Two-Frame SFM algorithm; cf. Section 1.2.3.)

Transforming the old model to the new coordinate system necessitates that the error estimate of the old model is also transformed. This is done as follows:

$$Cov(\mathbf{H}_{1,T}) = R_1 Cov(\mathbf{H}_1) R_1^T + N_1 \quad (3.2)$$

where N_1 denotes the additional noise that corrupts the model due to the process of transferring coordinate systems. How well the noise N_1 is estimated affects the accuracy of any incremental MFSFM algorithm. Exactly how the transformation is done is addressed in Section 3.5.

Finally, the old model (\mathbf{H}_1) that has been transformed to $\mathbf{H}_{1,T}$ can be updated with the new two-frame model \mathbf{W}_2 , resulting in the new multi-frame model as follows:

$$\mathbf{H}_2 = Cov(\mathbf{H}_2) [Cov(\mathbf{H}_{1,T})^{-1} \mathbf{H}_{1,T} + Cov(\mathbf{W}_2)^{-1} \mathbf{W}_2] \quad (3.3)$$

where

$$Cov(\mathbf{H}_2) = [Cov(\mathbf{H}_{1,T})^{-1} + Cov(\mathbf{W}_2)^{-1}]^{-1} \quad (3.4)$$

These equations define the crucial fusion component of a Kalman Filter [90].

Both 3D models in Equation 3.3 ($\mathbf{H}_{1,T}$ and \mathbf{W}_2) are multiplied by the inverse of their covariance. If the covariance is large - indicating a large error, and thus an

unreliable model – then the inverse is small, giving the relevant model less weight in the final result. That is, fusion using a Kalman Filter is a kind of weighted averaging, where the previous model and the new two-frame model have equal status apart from the value of the associated covariances ("weights"). The precise definition of the Kalman Filter and its equations are given in Appendix A, and adapted for this algorithm in Section 3.5. The numerical issues relevant to the equations are considered in Chapter 5.

The new multi-frame model (H_2) is iteratively updated by the next two-frame model obtained by moving the camera to a new position. For example, experiments 1 and 2 in Chapter 4 involve 8–9 camera movements.

3.1.2 The Error Modules

The process of obtaining 3D models involves (either explicitly or implicitly²) a step that computes the camera motion between the two images, and a second step which involves computing the 3D coordinates based on the camera motion and the 2D image coordinates. In the cross-correlation-based algorithm the two steps are explicitly separated.

In order to formalize the two types of error affecting any two-frame model (direct and indirect error, as in Figure 1.4), let us represent the 3D model by a vector W , made up of n 3D points (w_i , $i = 1 \dots n$). Recall that a new 3D model is obtained based on image coordinates (from a consecutive pair of images) (I) and the camera motion (represented by a vector M). Thus the we can write:

$$W = W(M, I) \quad (3.5)$$

²The implicit case involves the situation where the camera's motion as well as the 3D model are computed simultaneously. Any errors in the explicit case have counterparts in the implicit approach.

However, in SFM the camera motion is in turn calculated based on image information, and hence is a function of the image coordinates; i.e., M can be written as $M(I)$ which can be substituted in the above equation to obtain:

$$W = W(M(I), I) \quad (3.6)$$

This equation explicitly represents our method of calculating W by first calculating the camera's motion $M(I)$. This calculation corresponds to the dotted box in Figures 3.1 and 3.2. The model W is then calculated in a second stage using the original image coordinates I and the computed camera motion. Note that M and W can also be calculated in one step; equation 3.6 will hold implicitly in such a case.

In the second stage of calculation (which computes the 3D model W), the 3D model error in W can be computed in terms of the motion error in M and the image error in I . Using a standard first-order approximation, the total 3D model error in W can thus be written as:³

$$dW = \frac{\partial W}{\partial M} dM + \frac{\partial W}{\partial I} dI \quad (3.7)$$

where dW , dM and dI represent the respective errors in W , M , and I . The first term denotes the indirect error (cf. Figure 1.4) and the second term denotes the direct error. In using this first-order approximation to calculate the error in dW , we need to estimate the value of dI . We assume that the error in dI has a variance of 1 pixel, which is reasonable given the current status of point-tracking algorithms. The remaining three terms in Equation 3.7 are $\frac{\partial W}{\partial M}$, $\frac{\partial W}{\partial I}$ and dM . Let us now turn to a detailed discussion of how to calculate these terms.

³ M is a function of I and is not an independent variable; hence dM needs to be expanded further as done later in Equation 3.8.

3.2 The Indirect Error

3.2.1 Defining the Problem

The amount of error in the 3D model partially depends on how accurately the camera's motion is estimated; this is discussed below in Section 3.2.3. Before the exact *effect* of motion error on the 3D model – i.e. the indirect error – can be determined, we will first discuss how the error in the motion is calculated.

The computed camera motion M depends on the image coordinates I . The error in M can therefore be expressed in terms of the error in I as:

$$dM = \frac{\partial M}{\partial I} dI \quad (3.8)$$

Let us denote the first term on the right-hand side of Equation 3.7 by dW_M , the indirect error arising due to the motion error. The indirect error dW_M can be rewritten using Equation 3.8 as follows:

$$dW_M = \frac{\partial W}{\partial M} dM = \frac{\partial W}{\partial M} \frac{\partial M}{\partial I} dI \quad (3.9)$$

Thus to calculate dW_M it is necessary to determine the two partial derivatives $\frac{\partial W}{\partial M}$ and $\frac{\partial M}{\partial I}$. The first partial derivative denotes the indirect error that is introduced in the 3D coordinates due to an error in the motion. The second partial derivative captures the motion error that results from noise in the 2D image coordinates.

3.2.2 $\frac{\partial M}{\partial I}$: Motion error (∂M) with respect to Image Error (∂I)

In order to determine the partial derivative $\frac{\partial M}{\partial I}$ the process that is used to obtain the camera motion M from the image coordinates I needs to be analyzed.

The two-frame SFM algorithm (due to Horn [43]) that we use in the cross-correlation-based algorithm employs the Coplanarity Constraint (cf. Figure 1.1) to

find the best camera motion. This algorithm was chosen because of the simplicity of the function which it tries to minimize in order to arrive at the optimal solution. Horn's algorithm minimizes the deviation E from the coplanarity constraint which is a function of camera motion M and the image coordinates I , represented by the following equation involving a scalar triple product:⁴

$$E = \sum_{i=1}^n [\mathbf{T}, \mathbf{R}(l_i), r_i]^2 \quad (3.10)$$

where \mathbf{T} is the camera's translation between the two images, \mathbf{R} the camera's rotation, l_i the unit ray from the first camera position to the image coordinate in the first image, and r_i the corresponding ray for the second image (cf. Figure 1.1).⁵ The first and second image can also be referred to as the left and right image [43]; hence the usage l_i and r_i . \mathbf{T} and \mathbf{R} are related to motion M , and l_i and r_i are related to I , i.e. to the image coordinates of the world point; the exact relationships will be explored later. The summation over i denotes the sum of the contributions of each of the n reconstructed 3D points. The scalar triple product in equation 3.10 corresponds to the coplanarity error because it is directly proportional to the volume formed by the three rays (\mathbf{T}, l_i, r_i) ; if the three rays are coplanar, then the volume is zero. Thus, in the two-frame algorithm used here, the recovered motion M corresponds to a minimum of the coplanarity error E . This means that

$$\frac{\partial E(M, I)}{\partial M} = 0 \quad (3.11)$$

at the computed value of M .

In order to determine the exact effect of image error on the camera motion, we will analyze the effects of a small perturbation. If the image coordinates are perturbed (by

⁴We will use the notation $[, ,]$ to indicate a scalar triple product. i.e. $[a, b, c] = a \cdot (b \times c)$

⁵ l_i is in the coordinate system of the first camera whereas r_i and \mathbf{T} are in the coordinate system of the second camera.

$d\mathbf{I}$), the recovered motion will also usually change (by $d\mathbf{M}$). However, the coplanarity error is still minimized and Equation 3.11 holds for the perturbed values of \mathbf{M} and \mathbf{I} . Thus, the partial derivative $\frac{\partial E}{\partial \mathbf{M}}$ is unchanged under the perturbation of \mathbf{M} and \mathbf{I} . Considering $\frac{\partial E}{\partial \mathbf{M}}$ as a function of \mathbf{M} and \mathbf{I} , this implies:

$$\frac{\partial}{\partial \mathbf{M}} \left(\frac{\partial E}{\partial \mathbf{M}} \right) d\mathbf{M} + \frac{\partial}{\partial \mathbf{I}} \left(\frac{\partial E}{\partial \mathbf{M}} \right) d\mathbf{I} = 0 \quad (3.12)$$

or:

$$\frac{\partial^2 E}{\partial \mathbf{M}^2} d\mathbf{M} + \frac{\partial^2 E}{\partial \mathbf{M} \partial \mathbf{I}} d\mathbf{I} = 0 \quad (3.13)$$

Let

$$A \equiv \frac{\partial^2 E}{\partial \mathbf{M}^2} \quad (3.14)$$

and

$$B \equiv \frac{\partial^2 E}{\partial \mathbf{M} \partial \mathbf{I}} \quad (3.15)$$

By rearranging equation 3.13, we can obtain (assuming that A has an inverse⁶):

$$\frac{\partial \mathbf{M}}{\partial \mathbf{I}} = -A^{-1} B \quad (3.16)$$

Equation 3.16 provides the exact relationship between noise or perturbation in the images (i.e. image error) and the camera motion error. In order to calculate this quantity we need to make an explicit choice for our representation of the camera motion \mathbf{M} .

⁶In all the experiments conducted using the algorithm developed here, A has had an inverse. If A does not have an inverse, this indicates that the error analysis is very unstable at the estimated value of the camera motion. In such a case, a straightforward solution is to discard the current image, and acquire a new image.

3.2.2.1 The Components of the Motion M

Any 3D motion of the camera can be described in terms of a translation and rotation. Due to the previously discussed scale ambiguity in SFM, the scale of the translation cannot be determined, but only its direction. Recall from Section 1.2.3. that the total number of parameters describing the motion (that can be estimated based on solely image information) is five: two for the translation and three for the rotation. For our purposes, it is best to have a Cartesian representation where the two parameters representing translation are relatively unconstrained.⁷ A straightforward representation of the two unconstrained parameters for practically all translation directions⁸ involves a special coordinate system that depends on the estimated value of T . The three orthonormal axes are T_e , a and b , where T_e is the estimated direction of camera translation, and a and b are arbitrary orthonormal axes. In this system, any translation direction that is acceptable for our perturbation analysis is represented by its coordinates along a and b .

Formally, any translation (T) can be expressed in this coordinate system as a vector with three coordinates:

$$T \equiv (T \cdot a, T \cdot b, T \cdot T_e) \quad (3.17)$$

Every valid T (cf. footnote 8) gives rise to a unique combination of the first two coordinates. In fact, the third coordinate can be computed based on the first two coordinates if (as we assume) that T has *unit* length. Under these conditions, the third coordinate is:

$$T \cdot T_e = \sqrt{1 - |T_P|^2} \quad (3.18)$$

⁷Constrained coordinates (e.g. polar coordinates involving azimuth and elevation [41]) are bounded and such bounds would have to be additionally taken care of in the analysis. For example, elevation is in the range $(-\frac{\pi}{2}, \frac{\pi}{2})$.

⁸The valid translation directions are those directions which subtend an angle of $\frac{\pi}{2}$ or less with T_e . These vector directions are sufficient for the purposes of this error analysis since we consider only small perturbations about T_e to analyze the error in T_e .

where $\mathbf{T}_P \equiv (\mathbf{T} \cdot \mathbf{a}, \mathbf{T} \cdot \mathbf{b}, 0)$. In the following analysis the translation direction (the unit vector \mathbf{T}) will be represented by its corresponding \mathbf{T}_P vector, that is by the two coordinates $(\mathbf{T}_P \cdot \mathbf{a}, \mathbf{T}_P \cdot \mathbf{b})$.

There are several ways to represent rotation, viz., by Euler angles, quaternions, axis and angle, and matrices. Although all of these representations have the same representative power, quaternions have been preferred in SFM algorithms such as Horn's [43]. A variant of the quaternion representation will be used in the present work.

If the motion involves a rotation of an angle θ about an axis \mathfrak{R} then the unit quaternion (\mathbf{Q}) representing the rotation is :

$$\mathbf{Q} = \left(\cos \frac{\theta}{2}, \mathfrak{R} \sin \frac{\theta}{2} \right) \quad (3.19)$$

The crucial information about rotation is contained in the second element $(\mathfrak{R} \sin \frac{\theta}{2})$ of the quaternion which will be referred to as \mathbf{R} (for Rotation). The first element of the unit quaternion can be written in terms of the second element, since the following constraint holds:

$$|\mathbf{Q}| = 1 \quad (3.20)$$

that allows us to substitute the first element in terms of the second element as:

$$\cos \frac{\theta}{2} = \pm \sqrt{1 - |\mathbf{R}|^2} \quad (3.21)$$

If we consider only small perturbations (i.e. those which satisfy the bounds $-\frac{\pi}{2} \leq \frac{\theta}{2} \leq \frac{\pi}{2}$), the sign ambiguity disappears. Therefore, any allowed perturbation involving the first element of the unit quaternion can be rewritten in terms of \mathbf{R} (i.e. the second element).

Once the first element of the quaternion has been effectively eliminated, the rotation can be represented by three parameters in a Cartesian coordinate system

(x, y, z) , which we will refer to as $(R \cdot x, R \cdot y, \text{ and } R \cdot z)$. These do not have a straightforward relationship to representations in terms of angular variables, such as Euler angles, but contain the same information.

The combination of the three rotation parameters $(R \cdot x, R \cdot y, R \cdot z)$ and the two translation parameters $(T_P \cdot a, T_P \cdot b)$ constitute M .

3.2.2.2 Determining the A Matrix

We now return to determining the motion error (dM) with respect to (w.r.t.) the error in the image coordinates (dI). In equation 3.16 this derivative was represented as $-A^{-1}B$. The matrix A (in equation 3.14) was shown to be the second partial derivative of the coplanarity error (E) w.r.t. to the motion (M). First we need to compute the first partial derivative as an intermediate step. This step involves computing the partial derivative of E w.r.t. each of the five motion parameters.

3.2.2.2.1 The Two Translation Parameters. The partial derivative of E (Equation 3.10) w.r.t. one of the two translation parameters is :

$$\frac{\partial E}{\partial(T_P \cdot a)} = 2 \sum_{i=1}^n [T, R(l_i), r_i] \left[\frac{\partial T}{\partial(T_P \cdot a)}, R(l_i), r_i \right] \quad (3.22)$$

where T is determined as a function of $T_P \cdot a$ by the equation

$$T = T_P + \sqrt{1 - |T_P|^2} T_e \quad (3.23)$$

The derivative on the righthand side of equation 3.22 can be obtained by differentiating the above equation w.r.t. $(T_P \cdot a)$.

Applying the following

$$\frac{\partial c}{\partial(c \cdot \hat{b})} = \hat{b} \quad (3.24)$$

(where \hat{b} is a unit vector) to equation 3.22 we obtain:

$$\frac{\partial \mathbf{T}}{\partial (\mathbf{T}_P \cdot \mathbf{a})} = \mathbf{a} - \left(\frac{\mathbf{T}_P \cdot \mathbf{a}}{\sqrt{1 - |\mathbf{T}_P|^2}} \right) \mathbf{T}_e \quad (3.25)$$

Finally, by substituting the above expansion in equation 3.22 we end up with the fully expanded partial derivative of E w.r.t. the translation parameter $(\mathbf{T}_P \cdot \mathbf{a})$:

$$\frac{\partial E}{\partial (\mathbf{T}_P \cdot \mathbf{a})} = 2 \sum_{i=1}^n [\mathbf{T}, R(\mathbf{l}_i), \mathbf{r}_i] \left[\mathbf{a} - \left(\frac{\mathbf{T}_P \cdot \mathbf{a}}{\sqrt{1 - |\mathbf{T}_P|^2}} \right) \mathbf{T}_e, R(\mathbf{l}_i), \mathbf{r}_i \right] \quad (3.26)$$

The derivative of E w.r.t. the second translation parameter $(\mathbf{T}_P \cdot \mathbf{b})$ is identical to equation 3.26, except for the replacement of \mathbf{a} by \mathbf{b} .

3.2.2.2.2 The Three Rotation Parameters. The remaining three of the five first partial derivatives are with respect to the rotation parameters. Let us determine the derivative of E w.r.t. the first parameter $(\mathbf{R} \cdot \mathbf{x})$. This partial derivative is:

$$\frac{\partial E}{\partial (\mathbf{R} \cdot \mathbf{x})} = 2 \sum_{i=1}^n [\mathbf{T}, R(\mathbf{l}_i), \mathbf{r}_i] \left[\mathbf{T}, \frac{\partial R(\mathbf{l}_i)}{\partial (\mathbf{R} \cdot \mathbf{x})}, \mathbf{r}_i \right] \quad (3.27)$$

In order to expand the partial derivative on the righthand side of the above equation, the term $R(\mathbf{l}_i)$ must first be expressed in terms of \mathbf{R} . By using Rodrigues' formula [41] and vector algebra the expanded expression turns out to be:

$$R(\mathbf{l}_i) = (1 - 2|\mathbf{R}|^2) \mathbf{l}_i + 2 \sqrt{1 - |\mathbf{R}|^2} (\mathbf{R} \times \mathbf{l}_i) + 2 (\mathbf{R} \cdot \mathbf{l}_i) \mathbf{R} \quad (3.28)$$

The partial derivative on the righthand side of equation 3.27 obtained by applying rules of vector differentiation (c.f. Horn [41]) is:

$$\begin{aligned} \frac{\partial R(\mathbf{l}_i)}{\partial (\mathbf{R} \cdot \mathbf{x})} = & -4 (\mathbf{R} \cdot \mathbf{x}) \mathbf{l}_i - 2 \frac{(\mathbf{R} \cdot \mathbf{x})}{\sqrt{1 - |\mathbf{R}|^2}} (\mathbf{R} \times \mathbf{l}_i) \\ & + 2 \left\{ \sqrt{1 - |\mathbf{R}|^2} (\mathbf{x} \times \mathbf{l}_i) + (\mathbf{x} \cdot \mathbf{l}_i) \mathbf{R} + (\mathbf{R} \cdot \mathbf{l}_i) \mathbf{x} \right\} \end{aligned} \quad (3.29)$$

The fully expanded partial derivative of E w.r.t. $(\mathbf{R} \cdot \mathbf{x})$ is obtained by a direct substitution of the above equation (3.29) into equation 3.27. The derivatives of E w.r.t. the two remaining rotation parameters $((\mathbf{R} \cdot \mathbf{y})$ and $(\mathbf{R} \cdot \mathbf{z}))$ will be identical to the derivative w.r.t. $(\mathbf{R} \cdot \mathbf{x})$ except for the replacement of \mathbf{x} with \mathbf{y} and \mathbf{z} , respectively.

3.2.2.2.3 The Components of the A Matrix. Thus far we have discussed the first partial derivatives with respect to the five motion parameters. The second partial derivatives (matrix A) can now be computed. This computation involves differentiating each one of the five first partial derivatives w.r.t. every one of the five motion parameters. The result of such a computation involves 25 terms which constitute the matrix A . The elements of A are of three types: four derivatives w.r.t. only the translation (\mathbf{T}_P) , nine derivatives w.r.t. only the rotation (\mathbf{R}) , and 12 derivatives w.r.t. both the translation and the rotation.

3.2.2.2.4 The Four Translation Derivatives of A . One of the derivatives of E w.r.t. translation is $\frac{\partial^2 E}{\partial(\mathbf{T}_P \cdot \mathbf{a})\partial(\mathbf{T}_P \cdot \mathbf{b})}$. In order to calculate the expansion of this second derivative, $\frac{\partial E}{\partial(\mathbf{T}_P \cdot \mathbf{a})}$ (in equation 3.26) should be differentiated w.r.t. $(\mathbf{T}_P \cdot \mathbf{b})$. The result of the differentiation is :

$$\begin{aligned} \frac{\partial^2 E}{\partial(\mathbf{T}_P \cdot \mathbf{a})\partial(\mathbf{T}_P \cdot \mathbf{b})} &= 2 \sum_{i=1}^n \left[\mathbf{a} - \left(\frac{\mathbf{T}_P \cdot \mathbf{a}}{\sqrt{1 - |\mathbf{T}_P|^2}} \right) \mathbf{T}_e, R(l_i), r_i \right] \\ &\quad \left[\left(\mathbf{b} - \left(\frac{\mathbf{T}_P \cdot \mathbf{b}}{\sqrt{1 - |\mathbf{T}_P|^2}} \right) \mathbf{T}_e \right), R(l_i), r_i \right] \\ &\quad - [\mathbf{T}, R(l_i), r_i] \\ &\quad \left[\left(\frac{\mathbf{a} \cdot \mathbf{b}}{\sqrt{1 - |\mathbf{T}_P|^2}} + \frac{(\mathbf{a} \cdot \mathbf{T}_P)(\mathbf{b} \cdot \mathbf{T}_P)}{(1 - |\mathbf{T}_P|^2)^{\frac{3}{2}}} \right) \mathbf{T}_e, R(l_i), r_i \right] \end{aligned} \quad (3.30)$$

We have now arrived at the value of one element of the matrix A . However, the elements of the matrix A must be evaluated at the best estimate (\mathbf{T}_e) of the robot's movement, i.e., as $\mathbf{T} \rightarrow \mathbf{T}_e$. In effect this substitution of \mathbf{T} implies that $\mathbf{T}_P \rightarrow 0$

(from equations 3.17 and 3.18) resulting in:

$$\frac{\partial^2 E}{\partial(\mathbf{T}_P \cdot \mathbf{a})\partial(\mathbf{T}_P \cdot \mathbf{b})} = 2 \sum_{i=1}^n [\mathbf{a}, R(\mathbf{l}_i), \mathbf{r}_i] [\mathbf{b}, R(\mathbf{l}_i), \mathbf{r}_i] - [\mathbf{T}, R(\mathbf{l}_i), \mathbf{r}_i]^2 (\mathbf{a} \cdot \mathbf{b}) \quad (3.31)$$

The remaining three elements w.r.t. the translation are identical to the above (equation 3.31) except for the replacement of either \mathbf{a} by \mathbf{b} or \mathbf{b} by \mathbf{a} or both.

3.2.2.2.5 The Nine Rotational Derivatives of A . We will first consider one element involving rotation derivatives: $\frac{\partial^2 E}{\partial(\mathbf{R} \cdot \mathbf{x})\partial(\mathbf{R} \cdot \mathbf{y})}$. In order to expand this element, we differentiate equation 3.27 w.r.t. $(\mathbf{R} \cdot \mathbf{y})$ and obtain:

$$\begin{aligned} \frac{\partial^2 E}{\partial(\mathbf{R} \cdot \mathbf{x})\partial(\mathbf{R} \cdot \mathbf{y})} &= 2 \sum_{i=1}^n [\mathbf{T}, \mathbf{r}_i, \frac{\partial R(\mathbf{l}_i)}{\partial(\mathbf{R} \cdot \mathbf{x})}] [\mathbf{T}, \mathbf{r}_i, \frac{\partial R(\mathbf{l}_i)}{\partial(\mathbf{R} \cdot \mathbf{y})}] \\ &\quad + [\mathbf{T}, \mathbf{r}_i, R(\mathbf{l}_i)] [\mathbf{T}, \mathbf{r}_i, \frac{\partial^2 R(\mathbf{l}_i)}{\partial(\mathbf{R} \cdot \mathbf{x})\partial(\mathbf{R} \cdot \mathbf{y})}] \end{aligned} \quad (3.32)$$

Of the three partial derivatives on the right-hand side of equation 3.32, the terms $\frac{\partial R(\mathbf{l}_i)}{\partial(\mathbf{R} \cdot \mathbf{x})}$ and $\frac{\partial R(\mathbf{l}_i)}{\partial(\mathbf{R} \cdot \mathbf{y})}$ have already been obtained (Equation 3.29). The remaining derivative is obtained by differentiating equation 3.29:

$$\begin{aligned} \frac{\partial^2 R(\mathbf{l}_i)}{\partial(\mathbf{R} \cdot \mathbf{x})\partial(\mathbf{R} \cdot \mathbf{y})} &= 2 \{ (\mathbf{x} \cdot \mathbf{l}_i) \mathbf{y} + (\mathbf{l}_i \cdot \mathbf{y}) \mathbf{x} - 2(\mathbf{y} \cdot \mathbf{x}) \mathbf{l}_i \} \\ &\quad - \frac{2}{\sqrt{1 - |\mathbf{R}|^2}} \{ (\mathbf{x} \cdot \mathbf{R})(\mathbf{y} \times \mathbf{l}_i) + (\mathbf{y} \cdot \mathbf{R})(\mathbf{x} \times \mathbf{l}_i) \} \\ &\quad - \frac{2}{\sqrt{1 - |\mathbf{R}|^2}} \left\{ \frac{(\mathbf{x} \cdot \mathbf{R})(\mathbf{y} \cdot \mathbf{R})}{1 - |\mathbf{R}|^2} + \mathbf{y} \cdot \mathbf{x} \right\} (\mathbf{R} \times \mathbf{l}_i) \end{aligned} \quad (3.33)$$

The above is only one of nine rotational derivatives. The remaining eight derivatives can be obtained using the remaining nine possible two-tuples from the set $(\mathbf{R} \cdot \mathbf{x}, \mathbf{R} \cdot \mathbf{y}, \mathbf{R} \cdot \mathbf{z})$.

3.2.2.2.6 The Twelve Mixed Derivatives of A . What remains to be expanded now are the twelve remaining elements of the matrix A , also called the mixed derivatives since they are derivatives w.r.t. translation as well as rotation. Of the twelve elements only six are distinct. Consider the first of these six derivatives, $(\frac{\partial^2 E}{\partial(\mathbf{T}_P \cdot \mathbf{a})\partial(\mathbf{R} \cdot \mathbf{x})})$. This derivative is obtained by first differentiating equation 3.26 w.r.t. $(\mathbf{R} \cdot \mathbf{x})$ and then replacing \mathbf{T}_P by $\mathbf{0}$ (just as in the case of translation in equation 3.31). The final result is:

$$\begin{aligned} \frac{\partial^2 E}{\partial(\mathbf{T}_P \cdot \mathbf{a})\partial(\mathbf{R} \cdot \mathbf{x})} &= -2 \sum_{i=1}^n [\mathbf{a}, R(l_i), \mathbf{r}_i] [\mathbf{T}, \mathbf{r}_i, \frac{\partial R(l_i)}{\partial(\mathbf{R} \cdot \mathbf{x})}] \\ &+ [\mathbf{T}, R(l_i), \mathbf{r}_i] [\mathbf{a}, \mathbf{r}_i, \frac{\partial R(l_i)}{\partial(\mathbf{R} \cdot \mathbf{x})}] \end{aligned} \quad (3.34)$$

Recall that the value of $\frac{\partial R(l_i)}{\partial(\mathbf{R} \cdot \mathbf{x})}$ has already been determined in equation 3.29.

The five remaining distinct derivatives can be obtained by a straightforward replacement of $(\mathbf{R} \cdot \mathbf{x})$ by $(\mathbf{R} \cdot \mathbf{y})$ or $(\mathbf{R} \cdot \mathbf{z})$, or the replacement of $(\mathbf{T}_P \cdot \mathbf{a})$ by $(\mathbf{T}_P \cdot \mathbf{b})$, or both, in equation 3.34.

We have now enumerated the complete matrix A which is necessary to compute the motion error (dM) due to the image error (i.e., the noise in the image coordinates, dI). The remaining term (B) in this computation (equation 3.16) will be addressed in the following section.

3.2.2.3 Determining the B matrix

We now turn to the calculation of the matrix

$$B = \frac{\partial^2 E}{\partial M \partial I} \quad (3.35)$$

Before proceeding to determine the exact values of the matrix B , we need to establish an exact representation for I .

3.2.2.3.1 The Components of the Image Coordinates I. In order to determine the motion of a moving camera, points (such as corners) are tracked from one image to the next. The 2D image coordinates of these points – in the first (left) image and in the second (right) image – provide information necessary to compute the camera motion (M) and to reconstruct the world (W). The collection of the left and right image coordinates of all the tracked points constitute I .

Let us denote the 2D coordinates of the i th point in the first image as F_i and the corresponding coordinates in the second image as S_i . For an image coordinate system (u, v) , with its origin at the center of the image, we end up with the four image coordinates corresponding to each point being $(F_i \cdot u, F_i \cdot v, S_i \cdot u, S_i \cdot v)$. The set of these four image coordinates of all the tracked points constitutes I .

3.2.2.3.2 The Components of the B Matrix. Given the representation of the image coordinates (I) we are now in a position to assemble the B matrix, which involves the calculation of the derivatives of $(\frac{\partial E}{\partial M})$ w.r.t. the image coordinates. Recall that the five terms corresponding to $\frac{\partial E}{\partial M}$ are enumerated in section 3.2.2.2.1 (two translation terms) and 3.2.2.2.2 (three rotation terms). The elements of B involve the derivatives of these five terms w.r.t. the $4n$ parameters of I . This results in a $5 \times 4n$ matrix that constitutes the matrix B .

The elements of B can be subdivided into two kinds of derivatives. The first kind are the derivatives w.r.t. the image coordinates in the first image ($F_i \cdot u$ and $F_i \cdot v$, $i = 1, \dots, n$). The second kind of derivatives of B are those that involve the second image, i.e., $S_i \cdot u$ and $S_i \cdot v$, $i = 1, \dots, n$.

3.2.2.3.3 The Two Coordinates of the First Image. Let us first consider one of the two image coordinates of one point, say, $(F_1 \cdot u)$. By a straightforward process of differentiation of equation 3.26 we arrive at the derivative of the first translation term $(T_P \cdot a)$ w.r.t. the first image coordinate $(F_1 \cdot u)$:

$$\begin{aligned} \frac{\partial^2 E}{\partial(\mathbf{T}_P \cdot \mathbf{a})\partial(\mathbf{F}_1 \cdot \mathbf{u})} &= 2[\mathbf{a}, \mathbf{r}_1, R(l_1)][\mathbf{T}, \mathbf{r}_1, R\left(\frac{\partial l_1}{\partial \mathbf{F}_1 \cdot \mathbf{u}}\right)] \\ &\quad + 2[\mathbf{T}, \mathbf{r}_1, R(l_1)][\mathbf{a}, \mathbf{r}_1, R\left(\frac{\partial l_1}{\partial \mathbf{F}_1 \cdot \mathbf{u}}\right)] \end{aligned} \quad (3.36)$$

where

$$\frac{\partial l_1}{\partial(\mathbf{F}_1 \cdot \mathbf{u})} = \frac{\mathbf{u}}{\sqrt{(\mathbf{F}_1 \cdot \mathbf{u})^2 + (\mathbf{F}_1 \cdot \mathbf{v})^2 + f^2}} - \frac{(\mathbf{F}_1 \cdot \mathbf{u})l_1}{(\mathbf{F}_1 \cdot \mathbf{u})^2 + (\mathbf{F}_1 \cdot \mathbf{v})^2 + f^2} \quad (3.37)$$

The above derivative (equation 3.37) is obtained by differentiating the value of the unit ray l_1 :

$$l_1 = \frac{(\mathbf{F}_1 \cdot \mathbf{u}, \mathbf{F}_1 \cdot \mathbf{v}, f)}{\sqrt{(\mathbf{F}_1 \cdot \mathbf{u})^2 + (\mathbf{F}_1 \cdot \mathbf{v})^2 + f^2}} \quad (3.38)$$

where f denotes the camera's focal length. The value of l_1 directly follows from its definition that it is the unit ray from the center of the camera to the (first) tracked point in the camera's image.

The derivative w.r.t. the second translation term ($\mathbf{T}_P \cdot \mathbf{b}$) can be obtained by the direct substitution of \mathbf{a} by \mathbf{b} in equation 3.36.

The derivative of one rotational term w.r.t. the first image coordinate is:

$$\begin{aligned} \frac{\partial^2 E}{\partial(\mathbf{R} \cdot \mathbf{x})\partial(\mathbf{F}_1 \cdot \mathbf{u})} &= 2[\mathbf{T}, \mathbf{r}_1, R\left(\frac{\partial l_1}{\partial \mathbf{F}_1 \cdot \mathbf{u}}\right)][\mathbf{T}, \mathbf{r}_1, \frac{\partial R(l_1)}{\partial(\mathbf{R} \cdot \mathbf{x})}] \\ &\quad + 2[\mathbf{T}, \mathbf{r}_1, R(l_1)][\mathbf{T}, \mathbf{r}_1, \frac{\partial^2 R(l_1)}{\partial(\mathbf{R} \cdot \mathbf{x})\partial(\mathbf{F}_1 \cdot \mathbf{u})}] \end{aligned} \quad (3.39)$$

The value of $\frac{\partial^2 R(l_1)}{\partial(\mathbf{R} \cdot \mathbf{x})\partial(\mathbf{F}_1 \cdot \mathbf{u})}$ is computable by combining equations 3.29 and 3.37.

The two related rotational terms in matrix B are identical to the expansion provided by the above equation except for the replacement of $\mathbf{R} \cdot \mathbf{x}$ by $\mathbf{R} \cdot \mathbf{y}$ and $\mathbf{R} \cdot \mathbf{z}$. The remaining terms in matrix B that involve the first image coordinates (\mathbf{F}_i) are straightforwardly derivable from the above two equations.

3.2.2.3.4 The Two Coordinates of the Second Image The partial derivatives in the B matrix that involve the image coordinates of the second image are obtained by a similar sequence of steps as in the previous section.

Let us consider the partial derivatives involving the first tracked point. The first term involves the translation term $(T_P \cdot a)$ and its value (obtained by differentiating equation 3.26) is:

$$\begin{aligned} \frac{\partial^2 E}{\partial(T_P \cdot a)\partial(S_1 \cdot u)} &= 2[a, R(l_1), \frac{\partial r_1}{\partial S_1 \cdot u}][T, R(l_1), r_1] \quad (3.40) \\ &+ 2[a, R(l_1), r_1][T, R(l_1), \frac{\partial r_1}{\partial S_1 \cdot u}] \end{aligned}$$

$\frac{\partial r_1}{\partial S_1 \cdot u}$ is identical to $\frac{\partial l_1}{\partial F_1 \cdot u}$ except for the replacement of F by S and l by r in equation 3.37.

The second term that involves the other translation term $T_P \cdot b$ is identical to the above equation except for the replacement of a by b .

The remaining three related rotational terms are derived by differentiating equation 3.29 to obtain:

$$\begin{aligned} \frac{\partial^2 E}{\partial(R \cdot x)\partial(S_1 \cdot u)} &= 2[T, R(l_1), \frac{\partial r_1}{\partial S_1 \cdot u}][T, \frac{\partial R(l_1)}{\partial R \cdot x}, r_1] \quad (3.41) \\ &+ 2[T, R(l_1), r_1][T, \frac{\partial R(l_1)}{\partial R \cdot x}, \frac{\partial r_1}{\partial S_1 \cdot u}] \end{aligned}$$

The other two rotational terms are identical to the above equation except for the appropriate replacement of $R \cdot x$ by $R \cdot y$ and $R \cdot z$.

The remaining elements of matrix B that involve the second image coordinates (S_i) are all obtainable from the above two equations by the replacement of the subscript 1 by any $i, i = 2, \dots, n$.

With the determination of the two matrices A and B we have completed the derivations of all the terms involved in the computation of $\frac{\partial M}{\partial I}$ - the motion error

due to the image error. Given any noise (dI) in the image coordinates, we are now in a position to calculate the effect of this noise on the estimated camera motion; i.e., we can isolate the effect of tracking inaccuracies in the process of estimating how the camera has moved. We will now turn to a consideration of how the error in the camera motion affects the accuracy of the 3D model.

3.2.3 $\frac{\partial W}{\partial M}$: 3D Model Error (dW) with respect to Motion Error (dM)

Equation 3.9 expressed the error in the 3D model due to an error in the camera motion. For convenience we repeat that equation here:

$$dW(M) = \left(\frac{\partial W}{\partial M} \right) \left(\frac{\partial M}{\partial I} \right) dI \quad (3.42)$$

The rest of this section discusses the term $\frac{\partial W}{\partial M}$, the effect of the incorrect camera motion on the 3D model. In order to determine $\frac{\partial W}{\partial M}$, we need to first write W as a function of the motion parameters. The 3D coordinate of any point (w_i) can be shown to be a vector function (adapted from Horn [43]):

$$w_i = \frac{(\mathbf{T} \times R(l_i)) \cdot (R(l_i) \times \mathbf{r}_i)}{|R(l_i) \times \mathbf{r}_i|^2} \mathbf{r}_i \quad (3.43)$$

The first term (i.e. the fraction) evaluates the distance of a 3D point from the second camera; the second term (\mathbf{r}_i) is the direction of the point from the second camera. A potential improvement over just using the above equation is to average the 3D coordinate obtained with the corresponding estimate for the first camera, followed by an analysis similar to what is proposed here. However, since such an approach would further complicate the error analysis we do not pursue it here.

By differentiating the above equation w.r.t. the translation parameter (of the camera motion) $\mathbf{T}_P \cdot \mathbf{a}$ we obtain:

$$\frac{\partial \mathbf{w}_i}{\partial (\mathbf{T}_P \cdot \mathbf{a})} = \frac{(\mathbf{a} \times R(\mathbf{l}_i)) \cdot (R(\mathbf{l}_i) \times \mathbf{r}_i)}{|R(\mathbf{l}_i) \times \mathbf{r}_i|^2} \mathbf{r}_i \quad (3.44)$$

The second translation derivative can be obtained by a replacement of \mathbf{a} by \mathbf{b} .

One of the three rotational derivatives is obtained by differentiating equation 3.43 w.r.t. $\mathbf{R} \cdot \mathbf{x}$ to yield:

$$\begin{aligned} \frac{\partial \mathbf{w}_i}{\partial (\mathbf{R} \cdot \mathbf{x})} = & \frac{1}{|R(\mathbf{l}_i) \times \mathbf{r}_i|^2} \left\{ (\mathbf{T} \times \frac{\partial R(\mathbf{l}_i)}{\partial (\mathbf{R} \cdot \mathbf{x})}) \cdot (R(\mathbf{l}_i) \times \mathbf{r}_i) \right. \\ & + (\mathbf{T} \times R(\mathbf{l}_i)) \cdot \left(\frac{\partial R(\mathbf{l}_i)}{\partial (\mathbf{R} \cdot \mathbf{x})} \times \mathbf{r}_i \right) \\ & - 2 \frac{(\mathbf{T} \times R(\mathbf{l}_i)) \cdot (R(\mathbf{l}_i) \times \mathbf{r}_i)}{|R(\mathbf{l}_i) \times \mathbf{r}_i|^2} \\ & \left. (R(\mathbf{l}_i) \times \mathbf{r}_i) \cdot \left(\frac{\partial R(\mathbf{l}_i)}{\partial (\mathbf{R} \cdot \mathbf{x})} \times \mathbf{r}_i \right) \right\} \mathbf{r}_i \quad (3.45) \end{aligned}$$

where the derivative $(\frac{\partial R(\mathbf{l}_i)}{\partial (\mathbf{R} \cdot \mathbf{x})})$ has already been determined in equation 3.29.

The two remaining rotational derivatives are obtainable from the above equation by the replacement of $(\mathbf{R} \cdot \mathbf{x})$ by $(\mathbf{R} \cdot \mathbf{y})$ or $(\mathbf{R} \cdot \mathbf{z})$.

The term $\frac{\partial \mathbf{W}}{\partial \mathbf{M}}$ corresponds to a $3n \times 5$ matrix, due to the n 3D coordinates and the 5 motion parameters. This matrix can now be filled in for every value of i ($i = 1, \dots, n$) using Equations 3.44 and 3.45.

3.3 The Direct Error

So far we have discussed how to evaluate the indirect error in the 3D model due to motion error. In this section we will study the second type of error in the 3D model, i.e. the direct error. Even if we knew the camera motion accurately, error would enter into the 3D model due to the image error (cf. Fig. 1.4).⁹

⁹In experiments with simulated images, Thomas and Oliensis [91] studied such a case.

In order to determine the direct error, we consider again equation 3.43, where w_i represents the coordinates of a point in the model W :

$$w_i = \frac{(\mathbf{T} \times R(\mathbf{l}_i)) \cdot (R(\mathbf{l}_i) \times \mathbf{r}_i)}{|R(\mathbf{l}_i) \times \mathbf{r}_i|^2} r_i \quad (3.46)$$

The derivatives of w_i w.r.t. image coordinates constitute the elements of the direct error (dW_D in Figure 3.1).

The partial derivative w.r.t. one of the left image coordinates ($\mathbf{F}_1 \cdot \mathbf{u}$) is:

$$\begin{aligned} \frac{\partial w_i}{\partial(\mathbf{F}_1 \cdot \mathbf{u})} &= \frac{1}{|R(\mathbf{l}_i) \times \mathbf{r}_i|^2} \{(\mathbf{T} \times R(\mathbf{u})) \cdot (R(\mathbf{l}_i) \times \mathbf{r}_i) \\ &\quad + (\mathbf{T} \times R(\mathbf{l}_i)) \cdot (R(\mathbf{u}) \times \mathbf{r}_i) \\ &\quad - 2 \frac{(\mathbf{T} \times R(\mathbf{l}_i)) \cdot (R(\mathbf{l}_i) \times \mathbf{r}_i)}{|R(\mathbf{l}_i) \times \mathbf{r}_i|^2} \\ &\quad (R(\mathbf{l}_i) \times \mathbf{r}_i) \cdot (R(\mathbf{u}) \times \mathbf{r}_i)\} r_i \end{aligned} \quad (3.47)$$

Similarly, the partial derivative w.r.t. one of the right image coordinates is:

$$\begin{aligned} \frac{\partial w_i}{\partial(\mathbf{S}_1 \cdot \mathbf{u})} &= \frac{1}{|R(\mathbf{l}_i) \times \mathbf{r}_i|^2} \{(\mathbf{T} \times R(\mathbf{l}_i)) \cdot (R(\mathbf{l}_i) \times \mathbf{u}) \\ &\quad - 2 \frac{(\mathbf{T} \times R(\mathbf{l}_i)) \cdot (R(\mathbf{l}_i) \times \mathbf{r}_i)}{|R(\mathbf{l}_i) \times \mathbf{r}_i|^2} \\ &\quad (R(\mathbf{l}_i) \times \mathbf{r}_i) \cdot (R(\mathbf{l}_i) \times \mathbf{u})\} r_i \\ &\quad + \frac{(\mathbf{T} \times R(\mathbf{l}_i)) \cdot (R(\mathbf{l}_i) \times \mathbf{r}_i)}{|R(\mathbf{l}_i) \times \mathbf{r}_i|^2} \mathbf{u} \end{aligned} \quad (3.48)$$

The direct effect of the noise in the image on the 3D model can now be combined with the previously discussed indirect error.

3.4 The Combined Effect of Direct and Indirect Errors

All the elements required to determine the value of the 3D model error dW (from Equation 3.7) are now available. Since the dependence of the motion error on the image error has been calculated, Equation 3.7 can be rewritten as:

$$\begin{aligned}
d\mathbf{W} &= \frac{\partial \mathbf{W}}{\partial \mathbf{M}} \frac{\partial \mathbf{M}}{\partial \mathbf{I}} d\mathbf{I} + \frac{\partial \mathbf{W}}{\partial \mathbf{I}} d\mathbf{I} \\
&= \left(\frac{\partial \mathbf{W}}{\partial \mathbf{M}} \frac{\partial \mathbf{M}}{\partial \mathbf{I}} + \frac{\partial \mathbf{W}}{\partial \mathbf{I}} \right) d\mathbf{I}
\end{aligned} \tag{3.49}$$

In the previous discussion we have seen how to compute all of the terms in equation 3.49. Since it is not possible to know the exact value of the image error $d\mathbf{I}$, the value of $d\mathbf{W}$ cannot be calculated exactly. However, we make the standard assumption that the image noise is Gaussian (with zero mean) and hence its variance determines the probabilistic extent of error. This Gaussian image noise gets transformed into an error in the 3D model \mathbf{W} , the covariance of which is thus crucial. This covariance is the expected value of the variance of $d\mathbf{W}$, i.e.

$$Cov(d\mathbf{W}) = E(d\mathbf{W} d\mathbf{W}^T) \tag{3.50}$$

which can be expanded (using 3.49) as:

$$Cov(d\mathbf{W}) = G Cov(d\mathbf{I}) G^T \tag{3.51}$$

where

$$G = \frac{\partial \mathbf{W}}{\partial \mathbf{M}} \frac{\partial \mathbf{M}}{\partial \mathbf{I}} + \frac{\partial \mathbf{W}}{\partial \mathbf{I}} \tag{3.52}$$

and

$$Cov(d\mathbf{I}) = \sigma^2 J \tag{3.53}$$

In the above equation σ^2 denotes the variance of the error in the image coordinates of any point, and J denotes an identity matrix of size $4n \times 4n$. The value of σ is specified using the estimated accuracy of the point tracking algorithm; e.g. in the experiments reported in Chapter 4, σ is set to 1 pixel. Note that this choice of

the covariance of the image noise assumes that the noise in each image coordinate is independent. Such an assumption is typically valid, since individual points are independently tracked, and tracking algorithms will generally not introduce correlated errors in the 2D image coordinates of any two image points, except in special cases where the entire image is corrupted in some correlated fashion, such as by a rotational blur.

Thus far we have established how to calculate the full 3D error in any *two-frame model* (i.e., the darkened boxes in Figures 3.1 and 3.2). We are now in a position to update the previous model using the new two-frame model.

3.5 The Iterative Step: Fusing the New Two-frame 3D model with the Old 3D model

At any location of the camera, our ultimate aim is to update the previous 3D model (\mathbf{H}_P)¹⁰ with the new two-frame model \mathbf{W} , to obtain an improved 3D model (cf. Figure 1.3). This updating step requires that the previous 3D model be *transformed* to the current coordinate system using the calculated motion parameters. Also, the 3D model error of the previous model should be transformed to this coordinate system.

The transformed 3D model (i.e. the vector of n 3D coordinates) can be written as:

$$\mathbf{H}_T = R_{mat}(\mathbf{H}_P) - T_{mat} \quad (3.54)$$

where R_{mat} and T_{mat} represent matrix notations for rotation and translation, respectively, and are constructed from the camera motion parameters (\mathbf{R} , \mathbf{T}). Since the transformation matrices (R_{mat} and T_{mat}) are themselves noisy, they will further corrupt the previous model. Note that while this transformation error corrupts the

¹⁰In Figure 3.1 the previous 3D model is denoted by \mathbf{H}_1 .

previous model, the motion error (and other errors discussed in the previous sections) corrupts the new two-frame model. As a first approximation, the effect of the noisy transformation can be taken into account by linearizing as follows:

$$d\mathbf{H}_T = \frac{\partial \mathbf{H}_T}{\partial \mathbf{H}_P} d\mathbf{H}_P + \frac{\partial \mathbf{H}_T}{\partial \mathbf{M}} \frac{\partial \mathbf{M}}{\partial \mathbf{I}} d\mathbf{I}. \quad (3.55)$$

If K is defined as:

$$K = \frac{\partial \mathbf{H}_T}{\partial \mathbf{M}} \frac{\partial \mathbf{M}}{\partial \mathbf{I}} \quad (3.56)$$

then the covariance (of the transformed 3D model) is:

$$Cov(d\mathbf{H}_T) = R_{mat} Cov(d\mathbf{H}_P) R_{mat}^T + K Cov(d\mathbf{I}) K^T \quad (3.57)$$

Here it has been assumed that $(d\mathbf{I})$ and $(d\mathbf{H}_P)$ are statistically independent; after a few frames, the correlation between the error in the left image (half the terms in \mathbf{I}) and the error in the previous combined estimate should be small, making the independence assumption valid.

Finally, to determine the new multi-frame model \mathbf{H}_N (or \mathbf{H}_2 in Figure 3.1) the result of the current two-frame model (\mathbf{W}) and the transformed previous multi-frame model (\mathbf{H}_T) must be combined. This is straightforward except for the overall scale ambiguity in SFM. Currently, the scale of every 3D model is maintained at a fixed value; this is valid since the actual environment that is being modeled is rigid with unchanging scale (this is discussed further in Chapter 4). Then, the standard Kalman filter result [90] for the combined 3D model is:

$$\mathbf{H}_N = Cov(\mathbf{H}_N) [Cov(\mathbf{H}_T)^{-1} \mathbf{H}_T + Cov(\mathbf{W})^{-1} \mathbf{W}], \quad (3.58)$$

where $Cov(\mathbf{H}_N)$ is the covariance of the updated 3D model at the updated camera position:

$$Cov(\mathbf{H}_N) = [Cov(\mathbf{H}_T)^{-1} + Cov(\mathbf{W})^{-1}]^{-1} \quad (3.59)$$

$Cov(W)$, the estimated error in the two-frame 3D model (which is identical to $Cov(dW)$) has already been calculated in Equation 3.51. Thus, all the ingredients have been assembled for computing the updated 3D model every time the camera moves.

3.6 Theoretical Motivation for Using Cross-Correlations

Previous incremental MFSFM algorithms have ignored the off-diagonal terms of the covariance matrix in Equation 3.59 for reasons of simplicity and computational complexity. These terms represent the cross-correlations between the errors of different 3D points in the model.

We have assumed that W is made up of n 3D points ($w_i, i = 1 \dots n$) and that it represents the entire 3D model; the error in each w_i has been modeled in terms of direct error and indirect error. When the error in W is represented as a covariance matrix the elements of this matrix are given by the following equation :

$$Cov(dW) = E(dw_i dw_j), \quad 1 \leq i, j \leq n \quad (3.60)$$

where $E(x)$ denotes the expected value of x .

In this theoretical analysis, in order to bring out the meaning and role of the cross-correlation terms clearly, it will be assumed that a 3D model consists of just two points. The following analysis is purely theoretical; in practice, two points are insufficient for SFM, which requires a minimum of at least five points.

3.6.1 The Meaning of Cross-correlations: The Two-Point Case

In the two-point case, the covariance matrix is reduced to

$$Cov(dW) = \begin{pmatrix} E(dw_1 dw_1) & E(dw_1 dw_2) \\ E(dw_2 dw_1) & E(dw_2 dw_2) \end{pmatrix} \quad (3.61)$$

This covariance matrix has four correlation terms, two of which ($E(dw_1 dw_2)$ and $E(dw_2 dw_1)$) are equivalent, since they both represent the cross-correlation between the error in w_1 and the error in w_2 . The other two, ($E(dw_1 dw_1)$ and $E(dw_2 dw_2)$), are the covariance of the error in w_1 and w_2 ; in previous work (e.g. Cui, Weng and Cohen [21]) these have been the basis of the refinement, although they have been assumed to represent the *complete* error. Therefore, we will concentrate in this discussion on the role of the *cross-correlation term*, $E(dw_1 dw_2)$.

Using the direct and indirect components of the error in each 3D point (cf. Equation 3.49), we can expand the cross-correlation term in Equation 3.61 as:

$$E(dw_1 dw_2) = E\left[\left(\frac{\delta w_1}{\delta M} dM + \frac{\delta w_1}{\delta I_1} dI_1\right) \left(\frac{\delta w_2}{\delta M} dM + \frac{\delta w_2}{\delta I_2} dI_2\right)^T\right] \quad (3.62)$$

Since it is realistic to assume that any two arbitrary image coordinates (of chosen points) are corrupted by independent noise (cf. Equation 3.53 and its justification),

$$E(dI_1 dI_2) = 0 \quad (3.63)$$

and therefore one of the terms in the expansion of Equation 3.62 will vanish. The resultant expansion is:

$$E(dw_1 dw_2) = \frac{\delta w_1}{\delta M} E(dM dM^T) \frac{\delta w_2}{\delta M} + \frac{\delta w_1}{\delta M} E(dM dI_2^T) \frac{\delta w_2}{\delta I_2} + \frac{\delta w_1}{\delta I_1} E(dI_1^T dM) \frac{\delta w_2}{\delta M} \quad (3.64)$$

From equation 3.64 it can be seen that the cross-correlation term will be zero only when the three terms fortuitously cancel, or when they are all zero. This is probably a rare occurrence. In all other cases the cross-correlation term does make a contribution to the performance of an incremental MFSFM algorithm.

Let us assume that the coordinates of the two points have changed considerably between the two images, resulting in large optical flow. Therefore, a small error in

the optical flow (which corresponds to a small error in \mathbf{I}) can be assumed to have little effect on the error in the motion, $d\mathbf{M}$; i.e.,

$$E(d\mathbf{M}d\mathbf{I}_i^T) \approx 0 \quad i = 1, 2 \quad (3.65)$$

For this particular case, the expansion of Equation 3.64 is :

$$E(d\mathbf{w}_1 d\mathbf{w}_2) = \frac{\delta\mathbf{w}_1}{\delta\mathbf{M}} \text{Cov}(d\mathbf{M}) \frac{\delta\mathbf{w}_2}{\delta\mathbf{M}} \quad (3.66)$$

Equation 3.66 shows that the cross-correlation is directly proportional to the motion error, represented as the covariance of the error in the motion ($d\mathbf{M}$). If Equation 3.65 does not hold the situation is more complicated: the cross-correlation is influenced not only by the motion error but also (indirectly) by the image error. In either case, the cross-correlation term is closely related to the motion error.

3.6.2 The Effect of Cross-correlations in Kalman Filtering

In this section the previous analysis is extended to study the effect of cross-correlations on the process of refining 3D models using the Kalman filter.

The goal of the Kalman filter (cf. Appendix A) is to iteratively fuse the 3D models over time and obtain the optimal 3D model in the current sensor coordinate system utilizing all the information. The optimal fused 3D model is in effect the sum of the individual 3D models ($\mathbf{W}(t)$ at time t) weighted by the inverse of their covariances; this makes intuitive sense since if a particular covariance is large – suggesting a large error in the 3D model – then this 3D model should be given less weight. Consequently, the optimal fused 3D model (\mathbf{H}) at time t is as follows (i.e. the value obtained from standard Kalman filtering [90]):¹¹

¹¹Note that the Kalman Filter uses a recursive formulation to estimate the value of $\mathbf{H}(t)$; in Equation 3.67 we have expanded out the recursion. Furthermore, the value N (in Equation 3.67) is a normalizing term which is irrelevant for this analysis; cf. Equation 3.58 for a specific instantiation of N when $t = 2$.

$$\mathbf{H}(t) = N \sum^t \text{Cov}(\mathbf{W}(t))^{-1} \mathbf{W}(t) \quad (3.67)$$

$\bar{\mathbf{W}}$ will denote a *weighted W*, defined as $\text{Cov}(\mathbf{W})^{-1} \mathbf{W}$. In order to determine the exact contribution of a single 3D model $\bar{\mathbf{W}}$ at any time (t) the covariance can be expanded using Equation 3.61 (assuming Equation 3.65 is valid) in the following way:

$$\text{Cov}(\mathbf{W}) = \begin{pmatrix} S_1 + M_{11} & M_{12} \\ M_{21} & S_2 + M_{22} \end{pmatrix} \quad (3.68)$$

where

$$S_i = \frac{\delta \mathbf{w}_i}{\delta \mathbf{I}_i} \text{Cov}(d\mathbf{I}_i) \frac{\delta \mathbf{w}_i}{\delta \mathbf{I}_i} \quad (3.69)$$

and

$$M_{ij} = \frac{\delta \mathbf{w}_i}{\delta \mathbf{M}} \text{Cov}(d\mathbf{M}) \frac{\delta \mathbf{w}_j}{\delta \mathbf{M}} \quad (3.70)$$

S_i represents the error in the 3D coordinates due to the error in the image coordinates ($d\mathbf{I}$) assuming that the motion is perfectly known; M_{ij} represents the error in the 3D coordinates due to the error in the camera motion ($d\mathbf{M}$) assuming that the image coordinates are perfectly known.

Weighted \mathbf{W} can now be written as

$$\bar{\mathbf{W}} = \begin{pmatrix} S_1 + M_{11} & M_{12} \\ M_{21} & S_2 + M_{22} \end{pmatrix}^{-1} \begin{pmatrix} \mathbf{w}_1 \\ \mathbf{w}_2 \end{pmatrix} \quad (3.71)$$

Equation 3.71 can be expanded (after Bar-Shalom and Fortmann [9]) to obtain:

$$\bar{\mathbf{W}} = \begin{pmatrix} C_1^{-1} \bar{\mathbf{w}}_1 \\ C_2^{-1} \bar{\mathbf{w}}_2 \end{pmatrix} \quad (3.72)$$

where

$$C_1 = S_{11} + M_{11} - M_{12}(S_{22} + M_{22})^{-1} M_{12}^T \quad (3.73)$$

$$\bar{\mathbf{w}}_1 = \mathbf{w}_1 - M_{12}(S_{22} + M_{22})^{-1} \mathbf{w}_2 \quad (3.74)$$

$$C_2 = S_{22} + M_{22} - M_{21}(S_{11} + M_{11})^{-1} M_{21}^T \quad (3.75)$$

and

$$\bar{\mathbf{w}}_2 = \mathbf{w}_2 - \mathbf{M}_{21}(\mathbf{S}_{11} + \mathbf{M}_{11})^{-1}\mathbf{w}_1 \quad (3.76)$$

Let us now concentrate on the effect of the cross-correlation on a single optimally fused 3D coordinate ($\bar{\mathbf{w}}_1$ in Equation 3.74). If there is no motion error – i.e. M_{12} is zero – $\bar{\mathbf{w}}_1$ is identical to \mathbf{w}_1 . However, since this is generally not true in practice, the value of \mathbf{w}_2 has a corrective effect on \mathbf{w}_1 . The magnitude of the correction depends on the size of the cross-correlation M_{12} . Since we have shown that the cross-correlation captures the motion error (Section 3.6.1), the magnitude of the correction depends on the (shared) motion error that corrupts both \mathbf{w}_1 and \mathbf{w}_2 .

The covariance of $\bar{\mathbf{w}}_1$ is

$$\text{Cov}(\bar{\mathbf{w}}_1) = E([\mathbf{w}_1 - \mathbf{M}_{12}(\mathbf{S}_{22} + \mathbf{M}_{22})^{-1}\mathbf{w}_2][\mathbf{w}_1 - \mathbf{M}_{12}(\mathbf{S}_{22} + \mathbf{M}_{22})^{-1}\mathbf{w}_2]^T) \quad (3.77)$$

Again, this can be simplified to obtain:

$$\text{Cov}(\bar{\mathbf{w}}_1) = \mathbf{S}_{11} + \mathbf{M}_{11} - \mathbf{M}_{12}(\mathbf{S}_{22} + \mathbf{M}_{22})^{-1}\mathbf{M}_{12}^T \quad (3.78)$$

As stipulated by Kalman filtering, any contribution (towards the fused optimal model) is weighted by the inverse of its covariance. Thus we expect that the weight (C_1^{-1}) in Equation 3.72 should be the inverse of the covariance of $\bar{\mathbf{w}}_1$. Since the right-hand side of Equation 3.78 turns out to be equal to C_1 , this is *exactly* the case.

This analysis reveals that the cross-correlation terms are important. If the motion error is large (which includes the ambiguity involved in decoupling the camera's rotation from its translation), then the cross-correlation terms become significant and play a crucial role. Since the motion error in SFM is typically large [25], we predict that without cross-correlations the benefits of Kalman filtering are being lost, and the accuracy of the updated 3D model is affected. In the next chapter we study the importance of correlations experimentally.

CHAPTER 4

EXPERIMENTAL RESULTS

4.1 Introduction

The aim of this chapter is to provide experimental evidence for the effectiveness of the cross-correlation-based algorithm developed in this dissertation; henceforth, this algorithm will be referred to as the CC-based algorithm. In particular the goal is to show that the CC-based algorithm is superior to multi-frame algorithms which neglect the covariance information utilized by the CC-based algorithm. In the first part of this chapter (Sections 4.2-4.5) the CC-based algorithm is tested on three real-world image sequences involving a robot workcell, an indoor mobile robot, and an autonomous outdoor vehicle. For each image sequence the results of the CC-based algorithm are compared against three algorithms: Two-Frame, Blind Averaging, and Standard Kalman Filtering (described below). In the second part of the chapter (Sections 4.6-4.8), the results of the CC-based algorithm applied to the task of position estimation of a mobile robot are presented.

4.2 The Four Algorithms

Since the results from the first three sequences will be reported in terms of the same four algorithms, the algorithms are described here. The input and output parameters specific to each experimental scenario will be discussed later in conjunction with the experiments.

4.2.1 The Two-Frame Algorithm

Recall that the input to the CC-based algorithm at every new position of the robot is the two-frame model obtained by Horn's Relative Orientation [43] algorithm. Based on information from the previous image and the current image, this algorithm constructs a 3D model of tracked points (W in Figure 3.1) in the coordinate system of the current robot position. The details of this two-frame algorithm have been provided in Section 1.2.3. The 3D models obtained by this two-frame algorithm at each incremental step (i.e. the input to the CC-based algorithm) will be compared with the output of the CC-based algorithm as well as the blind averaging algorithm and standard KF algorithm. Such a comparison brings out the improvement that is due entirely to the use of multiple images.

4.2.2 Blind Averaging

A possible straightforward improvement over the two-frame approach is to obtain 3D models from consecutive pairs of images using the two-frame algorithm and then to maintain a running blind average of the 3D models. This corresponds to a rudimentary multi-frame algorithm which does not compute an error estimate in the 3D model, and therefore gives identical weight (or importance) to each of the 3D models. Blind averaging and the CC-based algorithm represent the two logical extremes of incremental multi-frame algorithms with respect to the use of covariance to capture 3D model error; the former uses no 3D model error while the latter incorporates the complete covariance of the 3D model error.

The multi-frame blind average is calculated as follows. Let us denote a vector representing the 3D coordinates (x, y, z) of a point (such as a corner) in a blind average model by b_t (where t denotes the iteration number). Then:

$$b_t = s \frac{1}{t} [w_t + (t-1)b'_{t-1}] \quad (4.1)$$

where s denotes a scale parameter (which will be discussed for each experiment), and w_t is a vector from the current two-frame model corresponding to the same 3D point as b_t . Similarly, b'_{t-1} is a 3D vector for the same point from the previous blind average that has been transformed to the current coordinate system:

$$b'_{t-1} = R (b_{t-1}) + T \quad (4.2)$$

The terms R and T denote the coordinate transform (rotation and translation; cf. Section 3.5) required to bring the previous blind average (b_{t-1}) to the current camera coordinate system. Finally, the actual blind average model is made up of a set of n vectors (b_t) corresponding to each tracked point.

Since the blind average model is constructed without an estimate of the 3D model error, the presence of unreliable measurements can skew the final result. However, note that blind averaging (with no relative weights) could result in a better model than using incorrect weights.

4.2.3 Standard Kalman Filtering vs. the CC-based Algorithm

There is a class of incremental multi-frame algorithms that use a subset of the elements of the covariance matrix to represent the 3D model error (cf. Table 2.2 for an overview). The most sophisticated algorithm in this class previously reported in the literature involves using a 3×3 covariance matrix to capture the error in each tracked point (e.g. Cui, Weng and Cohen [21]). In such an algorithm the total number of elements representing the 3D model error is $9n$ (for n points), as opposed to no such elements in the blind averaging algorithm and $9n^2$ elements in our CC-based algorithm.

In order to isolate the exact effect of using the full covariance matrix, we need to compare algorithms that differ only in the number of elements of the covariance matrix used in the error analysis. For the comparison, we consider a multi-frame algorithm

that employs 3×3 weights/covariances (unlike equal weights in blind averaging) for each point at each iteration. Such an incremental algorithm is an instantiation of a Kalman Filter (cf. Gelb [90] and Appendix A). Since the algorithm uses n 3×3 covariance matrices (for n points), as is currently the standard in the field, it is referred to as Standard Kalman Filtering (standard KF). The CC-based algorithm differs from standard KF in the number of terms in the covariance matrix.

The 3×3 matrix (c_i) associated with a point i ($i = 1, \dots, n$) is a subset of the complete covariance matrix (C) (for the entire set of points) obtained by applying the error analysis of Chapter 3. The subset is defined as follows:

$$c_i = C_{j,k} \quad \text{for } 3(i-1) < j, k \leq 3i \quad (4.3)$$

where $C_{j,k}$ denotes the element at the j th row and k th column of matrix C . As shown by the equation, the submatrices defined by c_i lie along the diagonal band of the complete covariance matrix.

The standard KF algorithm used here is identical to the CC-based algorithm (Chapter 3) apart from the values of the covariance matrices. In the standard KF algorithm the terms of the covariance matrix which do not correspond to any c_i in Equation 4.3 are zero. Assuming that the value of these terms is zero is equivalent to computing a partial covariance matrix; the ignored (zero) terms record the correlation of error between pairs of points in the 3D model. Previously reported algorithms have not only ignored the cross-correlations (i.e. the terms lying off the diagonal band defined by Equation 4.3), but they inherently lack a mechanism for computing these terms.

A full comparison of the results of the four algorithms - Two-frame, Blind Averaging with equal weights, standard KF with a $9n$ -element matrix, and the CC-based algorithm with a $9n^2$ -element matrix - will be provided for the first three experiments. Other possible algorithms involving a covariance matrix with more than $9n$ but fewer

than $9n^2$ elements will not be considered here, although ignoring certain subsets of the cross-correlations for computational reasons will be considered in Chapter 5.

4.3 Experiment I: Robot Workcell Sequence

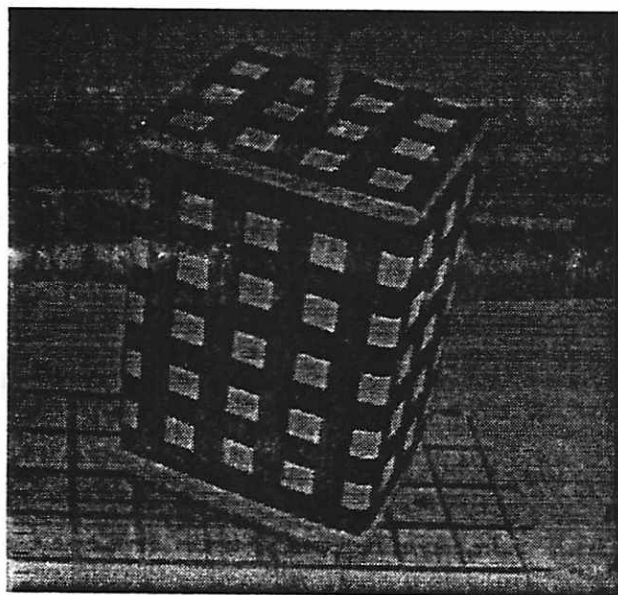
The following experiment involves an image sequence obtained while an object was rotated by a robotic arm in its workcell.

4.3.1 The Image Sequence and Ground Truth

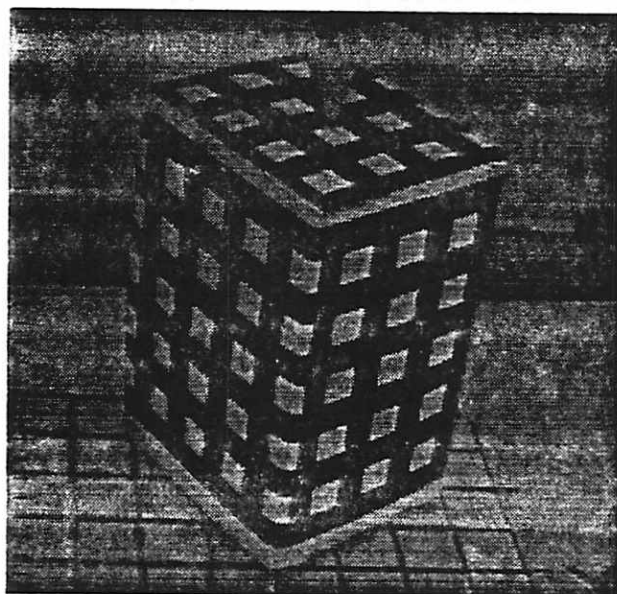
A sequence of pictures was taken by a stationary camera of a box rotated by a robot arm. An image was obtained after every 4-degree rotation about the vertical axis of the box. Figure 4.1 shows the first and the last image of the nine image sequence.

A set of 35 points have been selected in the first image and tracked across the nine images in previous work by Sawhney [78]. All 35 points are corners of the small white squares on the face of the box. Tracking of corners was done using the tracking algorithm of Williams and Hanson [109]; corners were chosen since they can be tracked robustly by their algorithm. The image coordinates of the 35 points in the first image are listed in Table 4.1.

The ground truth measurements of the 3D coordinates for the 35 selected points were obtained and made available by Sawhney [78]; these are listed in Table 4.2. The origin of a Cartesian (object-centered) coordinate system was fixed to a corner of the box, and the three edges at that corner correspond to the three axes (X, Y, Z) of the coordinate system. The dimensions of the box along the three axes are $133mm \times 157mm \times 70mm$. The coordinates of each tracked point were measured in the box-centered coordinate system with a ruler; the accuracy of measuring the position of



(a) First image



(b) Last (ninth) image

Figure 4.1: Rotating Box Image Sequence.

Table 4.1: Box Sequence: The image coordinates of the 35 points in the first image. This information was provided by Sawhney [78].

Point	x	y	Point	x	y
1	29.71	37.55	19	-51.73	-1.92
2	15.08	33.51	20	-49.44	-14.97
3	-54.56	37.48	21	62.00	-37.57
4	-60.04	44.94	22	12.29	-55.24
5	-63.41	49.66	23	-1.57	-59.43
6	1.49	29.29	24	-39.48	-46.79
7	-13.49	25.06	25	-45.42	-36.21
8	-45.11	24.85	26	-42.38	-32.26
9	-51.25	33.41	27	74.08	-54.36
10	-35.88	45.59	28	-26.40	56.38
11	-27.58	21.17	29	23.81	77.83
12	38.38	95.05	30	-33.31	-38.19
13	43.36	28.61	31	-45.12	78.60
14	60.36	-15.47	32	42.27	76.12
15	9.50	-32.79	33	-17.69	66.54
16	-4.61	-37.18	34	-13.40	80.92
17	-43.61	-22.80	35	58.39	8.36
18	-46.00	-10.34			

each point was estimated to be $\pm 1.5mm$.¹ In general, obtaining ground truth for SFM experiments has proven to be extremely difficult and considerable effort is currently spent in the field to obtain reliable ground truth (cf. Dutta, Manmatha, Williams and Riseman [24]). However, in this restricted environment of a robot's workcell, the measurements of the box have an accuracy that is on a par with the best experimental situations reported in the literature. Note that the measurements capture only the *shape* of the box, rather than its structure (cf. Section 2.2.3). This means that the location and orientation of the box with respect to the camera coordinate system are not accurately known; hence, the results of the algorithms can only be evaluated in terms of the shape of the box.

The camera used to obtain the images was a Sony black and white CCD stationary camera which was located about 600mm from the box. The parameters of this camera are listed in Table 4.3.

4.3.2 Input to the Algorithms

The following describes all the information given to the algorithms in addition to the camera parameters in Table 4.3.

4.3.2.1 Tracked Image Coordinates

All four algorithms tested here use a set of 2D image coordinates of tracked points as input. The image coordinates of the 35 points selected and tracked across 9 images by Sawhney [78] were employed in this experiment. For the purposes of this experiment, each image is assumed to have been obtained by a moving camera, which is logically equivalent to what was physically done.

¹Sawhney (personal communication).

Table 4.2: Box Sequence: Ground truth of tracked points. The coordinates were made available by Sawhney [78] with respect to the final camera position. The coordinates retain the shape of the box exactly as measured by hand since they have only been *rigidly* transformed from the box-centered coordinate system to a camera-centered coordinate system.

Point	X mm	Y mm	Z mm	Point	X mm	Y mm	Z mm
1	47.0	48.6	610.0	19	-39.6	-18.7	632.7
2	37.5	41.2	600.7	20	-38.2	-31.5	640.6
3	-33.2	26.9	592.2	21	76.1	-24.8	684.4
4	-45.4	31.0	600.9	22	40.0	-52.9	647.0
5	-56.2	34.7	609.5	23	30.0	-60.3	637.6
6	28.3	34.5	591.3	24	-23.2	-60.5	647.9
7	18.7	27.4	582.0	25	-35.3	-56.5	656.5
8	-10.3	19.5	575.0	26	-24.6	-47.7	639.9
9	-22.4	23.1	583.6	27	86.5	-43.5	709.6
10	-2.0	41.6	566.3	28	-4.5	53.0	584.3
11	9.4	21.1	572.6	29	22.2	85.1	630.3
12	18.4	106.6	666.3	30	-2.4	-42.8	614.7
13	57.0	42.1	627.3	31	-49.7	68.2	618.8
14	72.8	-0.3	668.5	32	41.3	87.6	631.1
15	36.8	-28.2	631.1	33	-6.5	63.7	602.3
16	26.5	-35.5	621.7	34	-18.8	78.8	628.9
17	-25.9	-35.3	632.0	35	69.6	24.9	652.6
18	-27.3	-22.5	624.0				

Table 4.3: Box Sequence: The camera parameters of the Sony black and white camera used in the rotating box sequence.

focal length	fov X	fov Y	Size
6 cm	23.4°	22.4°	256 × 242

4.3.2.2 The Covariance of the Error

Tracking of points across images is error-prone, and the actual error is not known. Therefore, given the typical sub-pixel accuracy of the tracking algorithm used by Sawhney [109], a conservative estimate for the variance of error in each image coordinate was chosen to be one pixel. Since all of the tracked points are corners in this experiment, there is no reason to expect that any point is tracked more accurately than the others - hence a fixed variance is a reasonable choice for the estimated error in tracking.

4.3.2.3 Guess of Interframe Camera Motion

Horn's two-frame algorithm requires an initial guess of the camera's movement between the two images to avoid local minima (or non-optimal solutions). In general this initial guess might be the command issued to the robot moving the camera. However, in this sequence the robot did not move the camera, but instead the robot was commanded to rotate the box and the stationary camera was pointed at the box at a slant from above. Although this scenario is equivalent to the case that we assume for the purpose of generating a model using Two-Frame SFM - that the camera moved and the box was stationary - transforming the command of 3.6° rotation of the box into an equivalent command for moving the camera involves knowing the coordinate transform between the box and the camera, i.e. the position and orientation (pose) of the box with respect to the camera.

For this experiment, the initial estimate was obtained using pose estimation techniques developed by Kumar [52], whereby the pose of the camera at the first image was computed with respect to the box, and similarly for the second image. The difference in the two camera poses constitutes the initial estimate for the camera motion. The initial estimate for the camera motion are provided in Table 4.4. The computed

results of Horn's algorithm, which differ from the initial estimate, are shown for comparison; we will return to a discussion of these results. The initial estimate localizes the operating region of Horn's algorithm, but the final result which the algorithm converges to is entirely dependent on the image coordinates. Ideally, to fully automate this algorithm, either the robot's command or a discrete sampling scheme as in Adiv [1] should provide the initial guess of motion.

4.3.2.4 Scale of the Model

The scale of any 3D model can be defined by a single number S as follows:

$$S = \frac{1}{n} \sum_1^n |w_i - \bar{w}| \quad (4.4)$$

where w_i is the i th point in the model and \bar{w} is the centroid of the model:

$$\bar{w} = \frac{1}{n} \sum_1^n w_i \quad (4.5)$$

Intuitively, the value of S represents the spread of the model around its center, which is a measure of scale. The scale of the two-frame model at every iteration is arbitrary owing to the scale ambiguity in SFM (cf. Section 1.2.3). The problem is further compounded by the fact that the 3D models obtained by the two-frame algorithm (within an iteration of MFSFM) vary in actual scale from iteration to iteration, making fusion impossible.² As a solution, we fix the scale of the model to a constant value (i.e. S is maintained as a constant, S_f) corresponding to the constant scale of the environment. A model A with an arbitrary scale S_A can be converted to a model C with a constant scale S_f as follows:

²The models vary in scale because of the unrecoverable translation (\mathbf{T}) magnitude; SFM algorithms arbitrarily set $|\mathbf{T}| = 1$ and return a model for this value of $|\mathbf{T}|$. It is then left to the user to scale the model up or down.

Table 4.4: Box Sequence: The initial estimate and recovered two-frame values of the interframe camera motion. (T_x, T_y, T_z) denotes the unit translation direction, (R_x, R_y, R_z) denotes the rotation axis and θ denotes the rotation angle in degrees. The magnitude of translation is not computed by SFM; instead, the 3D model is scaled as described in Section 4.3.2.4. Of the two rows of motion parameters associated with each interframe camera motion, the first row lists the initial estimate and the second row lists the recovered values. The command issued to the robot could not be used in this experiment as the initial guess of motion (see discussion in text).

Frame	T_x	T_y	T_z	R_x	R_y	R_z	θ
1-2	0.999	0.044	-0.020	-0.05	0.84	-0.55	3.57
	0.911	0.411	0.027	-0.05	0.53	-0.85	2.21
2-3	0.985	0.163	-0.053	-0.17	0.82	-0.54	3.70
	0.993	0.086	-0.083	-0.10	0.86	-0.51	4.07
3-4	0.990	0.126	-0.062	-0.13	0.85	-0.51	3.48
	0.988	0.156	-0.007	-0.15	0.81	-0.57	3.16
4-5	0.992	0.119	-0.038	-0.12	0.82	-0.56	3.21
	0.987	0.148	-0.060	-0.15	0.79	-0.60	3.12
5-6	0.985	0.164	-0.050	-0.16	0.82	-0.55	3.67
	0.987	0.154	-0.044	-0.16	0.85	-0.50	3.88
6-7	0.986	0.164	-0.039	-0.17	0.83	-0.53	3.83
	0.989	0.144	-0.040	-0.13	0.71	-0.69	2.84
7-8	0.988	0.140	-0.061	-0.14	0.82	-0.55	3.54
	0.987	0.148	-0.060	-0.15	0.85	-0.50	3.87
8-9	0.989	0.143	-0.047	-0.15	0.83	-0.53	3.63
	0.991	0.126	-0.047	-0.13	0.85	-0.51	3.82

$$C = \frac{S_f}{S_A} A \quad (4.6)$$

The term $\frac{S_f}{S_A}$ is referred to as the scale parameter s in Equation 4.1 above.

In order to reliably compare any model against the ground truth, the scale of the model and the scale of the ground truth must be the same. Therefore, the constant S_f is set at the value of the scale of the ground truth, which is part of the input given to the algorithm. Note that under the assumption of fixed scale, fusing models across iterations is no longer a problem.³

4.3.3 Results of the Four Algorithms

4.3.3.1 Representation of the Results

Since the ground truth for the box sequence is only given for the shape of the box, the 3D model produced by any of the four algorithms has to be rotated and translated (rigidly) to align with the ground truth (given in Table 4.2) in order to determine the performance of the algorithm. The mismatch between the aligned 3D model and the ground truth is the error in the shape. The alignment that minimizes the mismatch error can be determined exactly in closed form using Horn's Absolute Orientation Algorithm [42].

The error in the shape after alignment will be reported for the models acquired by all four algorithms. The contribution of a single point p in the extracted 3D model to the shape error is defined as $|p - t|$ where p is the position of a point (after alignment) and t is the position of the same point according to the ground truth. That is, $|p - t|$ represents the distance between the aligned 3D point and actual point; if this distance is zero over the entire model, the alignment is perfect and the shape has been accurately recovered. The overall error of the 3D model will be

³When any model is scaled by a term s , the corresponding covariance associated with the 3D model error is scaled by the term s^2 .

Table 4.5: Box Sequence: Mean and standard deviation σ of the 3D model error in mm at each frame for the four algorithms.

FRAME	Two-Frame		Blind-Avg		Std KF		CC-based	
	mean	σ	mean	σ	mean	σ	mean	σ
2	23.5	9.2	23.5	9.2	23.5	9.2	23.5	9.2
3	5.1	2.9	9.8	4.0	11.3	11.3	11.4	6.2
4	6.8	3.5	8.3	3.5	13.1	9.8	7.1	3.6
5	5.1	3.9	5.7	2.6	12.7	7.8	2.9	1.5
6	4.1	2.5	5.0	2.4	11.7	8.0	1.5	0.8
7	15.3	6.2	6.3	2.5	12.2	7.5	1.5	0.9
8	5.3	3.2	4.9	2.0	10.3	6.6	1.4	0.8
9	5.4	3.3	4.7	2.0	9.6	6.2	1.5	0.9

reported as the mean and standard deviation of the shape errors for the set of points in the model. Note that since the mean accuracy of the ground truth is $\pm 1.5mm$, any shape error less than $1.5mm$ may be due to error in ground truth.

4.3.3.2 Discussion of the Results

Figure 4.2 shows a graph of the mean shape error at each iteration of the algorithm for the four algorithms, and Figure 4.3 provides the corresponding standard deviations. The actual data from which the graphs were derived is shown in Table 4.5.

Consider the iteration associated with Frame 5 in Figure 4.2. The average error at this iteration for the Two-Frame algorithm denotes the error in the 3D model obtained using images 4 and 5. For the three multi-frame algorithms, the error at

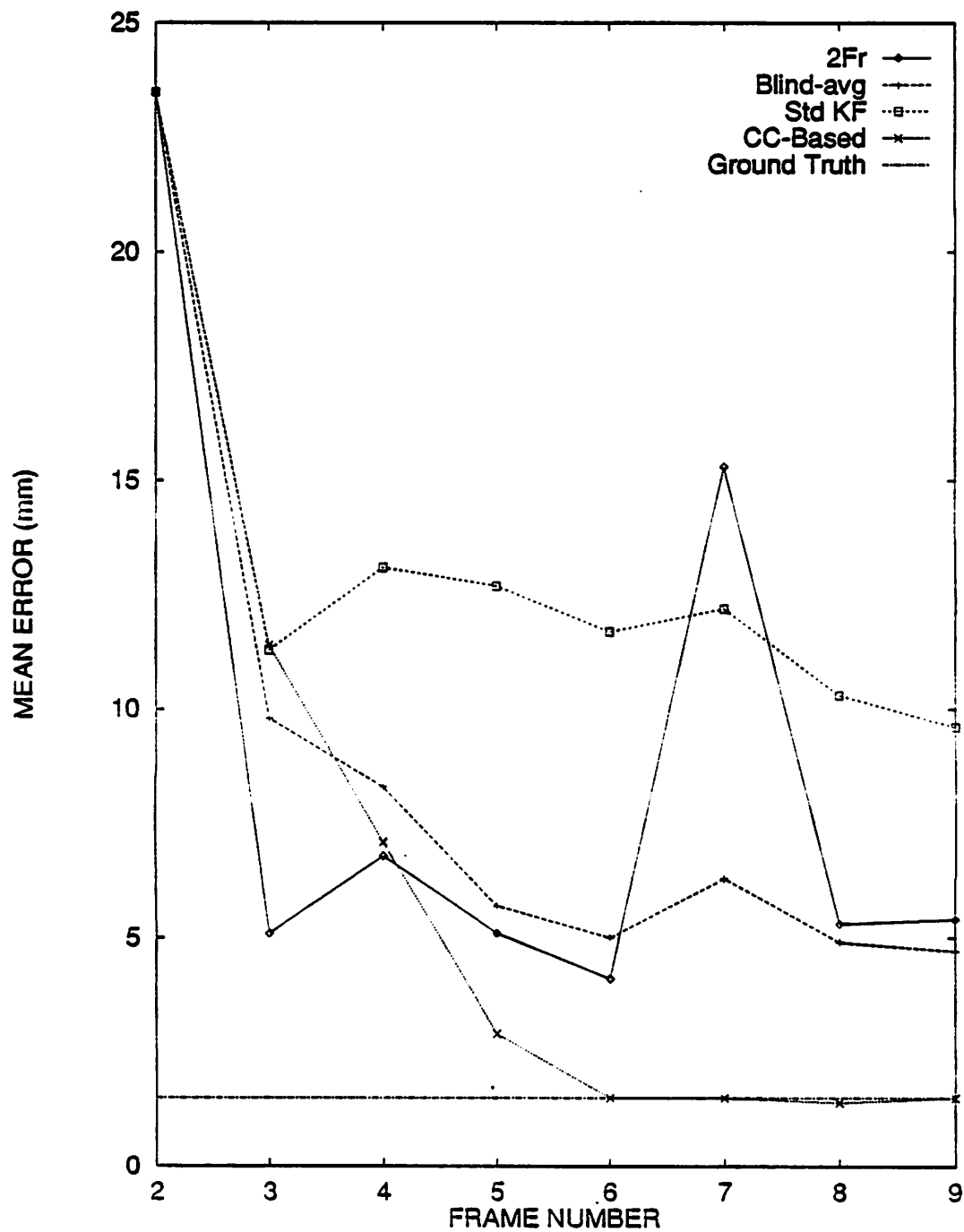


Figure 4.2: Box Sequence: Reconstruction Error (in *mm*) for the four algorithms compared in this experiment.

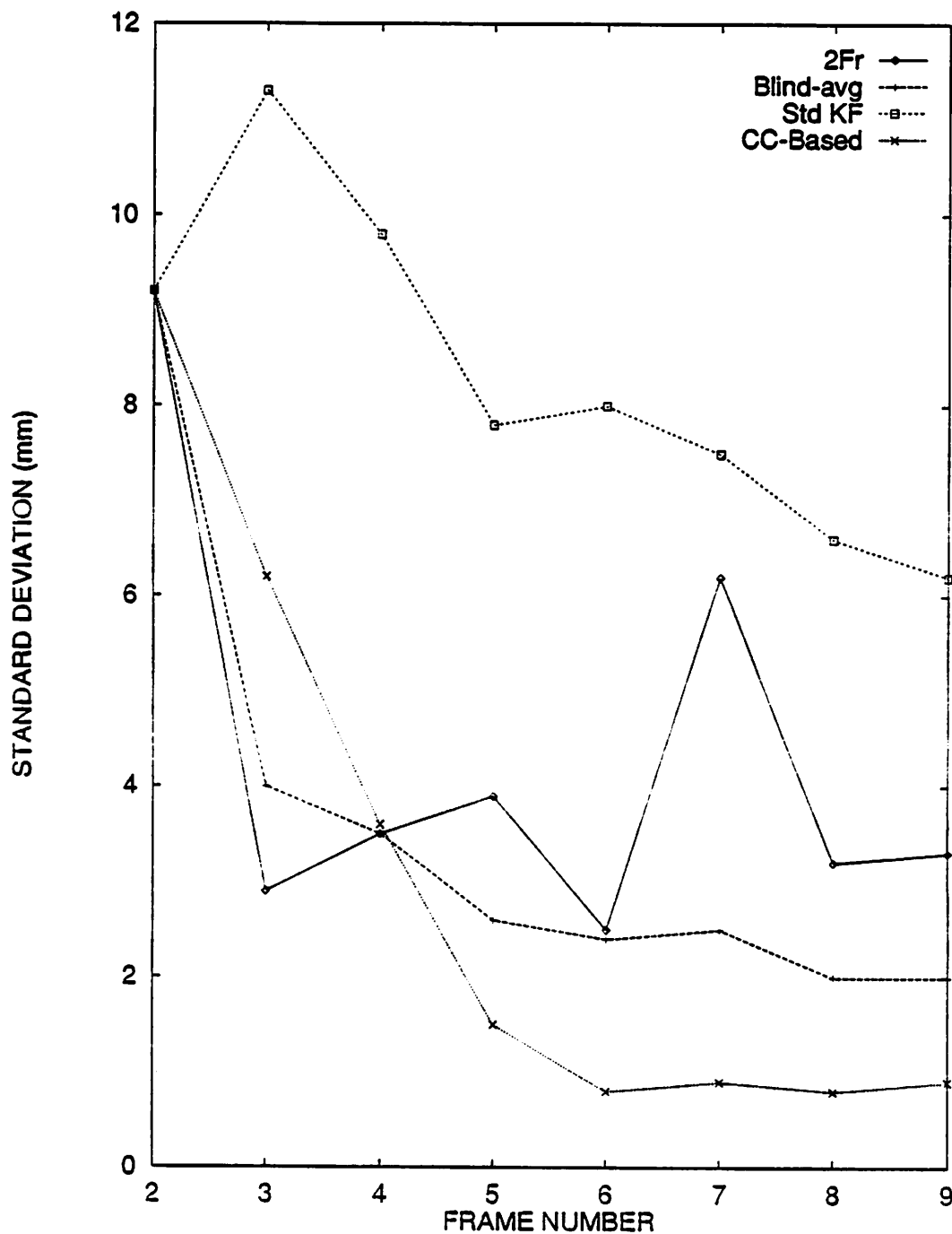


Figure 4.3: Box Sequence: Standard deviation of the error in the 3D models for four algorithms.

this iteration denotes the error in the 3D model which is the result of fusing the two-frame models up to this point (i.e. the two-frame models available at frames 2, 3, 4, and 5). For example, for the blind averaging algorithm, this involves averaging the two-frame models of frames 2, 3, 4, and 5.

From the graph in Figure 4.2 it can be observed that the error in the 3D model of the Two-Frame algorithm fluctuates; there is no clear trend. Since at each iteration the two-frame model is constructed "from scratch" with no information from the previous models, the fluctuation is expected. For example, there is no connection between the model at frame 4 and at frame 6. In general, the unpredictability (in mean and standard deviation) of the results of the Two-Frame algorithm make this algorithm unreliable. Table 4.4 reveals that when the camera motion was estimated least accurately, the resulting two-frame model was the least accurate as well. The error in camera motion can be clearly noticed in the difference in the rotation angle θ between the original and the recovered values in Table 4.4, such as at frames 1-2 and 6-7. Figure 4.2. shows that the two-frame model has the highest errors at frames 2 and 7. This result corroborates a similar observation by Dutta and Snyder [25] regarding the large effect of relatively small rotation errors on the 3D model.

Blind averaging, the simplest multi-frame improvement over the two-frame algorithm, shows considerable improvement in this experiment. The graph in Figure 4.2 shows that the error in 3D models obtained from blind averaging decreases except in the 7th frame, where the two-frame model is very inaccurate. Since the blind averaging scheme gives the same weight to this two-frame model as all the others, the error in the averaged 3D model increases. Although the blind average appears to follow a decreasing trend, this is not expected in a situation where the two-frame results fluctuate more dramatically. The final blind average model has an average error of $4.7mm$ with a standard deviation of $2.0mm$.

Figure 4.2 shows that the error of the standard KF algorithm converges slowly, apart from frames 4 and 7 where the individual two-frame models are more erroneous than the previous ones. What is striking is that the KF approach is worse than blind averaging in each frame. This situation could be anomalous in that blind averaging may be doing unexpectedly well. Alternatively, this result may also indicate that the weights employed in standard KF are grossly incorrect, which might give rise both to a high mean error and a high standard deviation. Such a situation would indicate that it is better to use equal weights (as in blind averaging) than to use incorrect weights. In fact, the previously reported standard KF algorithm of Cui, Weng and Cohen [21] produces fluctuating results with no trend in their real image experiment. Based on their results, these authors conclude that, "*our method gives accurate results from the first two frames*" (p. 228). This statement reveals that Cui, Weng and Cohen's version of the standard KF algorithm performed no better than the two-frame algorithm (since the model based on the first two frames is considered as good as any later model!). Thus, a likely explanation for the poor performance in the rotating box sequence is the use of incorrect weights. In this sequence, the final mean error for the standard KF algorithm is $9.6mm$ with a standard deviation of $6.2mm$.

On the other hand, the superior performance of the CC-based algorithm suggests that the weights used in this algorithm accurately reflect the 3D model errors. The high accuracy of this algorithm (as shown in Figure 4.2) is entirely due to the inclusion of the motion error, which is recorded in the cross-correlations. Figure 4.2 shows that the average 3D model error falls monotonically until frame 6 and then remains as low as the error in the ground truth ($1.5mm$) for the last four frames. Note that when the error falls *below* the error in the ground truth, any fluctuations in the error are meaningless and can be considered noise. Figure 4.3 shows that the standard

deviation also monotonically decreases until the mean 3D model error reaches the level of the error in the ground truth, as predicted by a properly functioning Kalman Filter with reliable covariances. The final model of the CC-based algorithm has a mean error of $1.5mm$ (which is at the level of the ground truth) and a standard deviation of $0.9mm$.

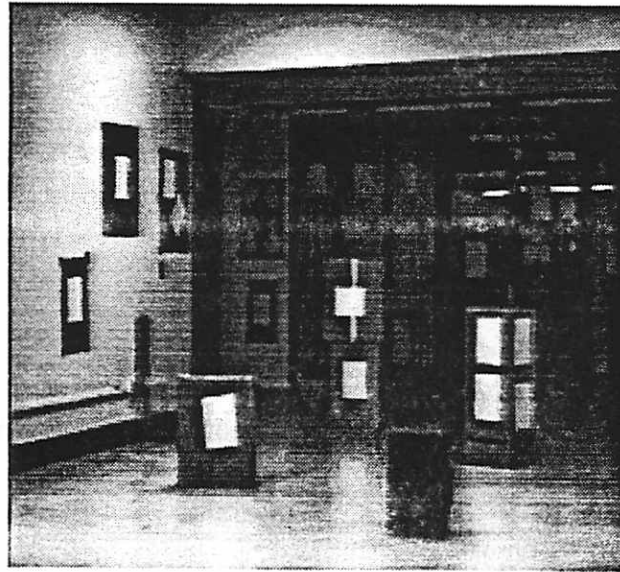
4.4 Experiment II: Indoor Robot Sequence

This experiment compares the effectiveness of the four algorithms (described in Section 4.2) in recovering a model of a lobby in the Computer Science Department at UMass, using a sequence of images obtained by an indoor mobile robot.

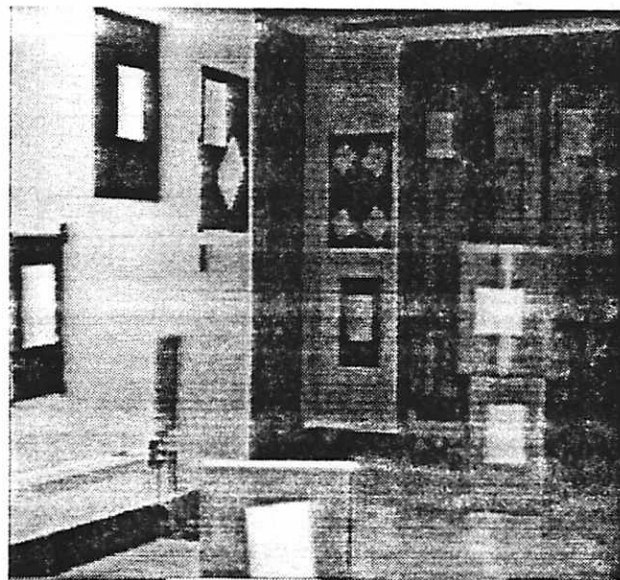
4.4.1 The Image Sequence and Ground Truth

A sequence of images was taken of the lobby by a camera mounted on a moving Denning mobile robot, as described by Sawhney [78]. The robot was commanded to move directly ahead in steps of 1.4 feet, but there were small rotations and drift. Figure 4.4 depicts the first and the last image of the ten-image sequence. For this experiment 29 corners of posters and obstacles were hand-selected from the first image and tracked across the ten images (cf. Table 4.6). Tracking was done using the tracking algorithm of Williams and Hanson [109].

The ground truth for 23 of the 29 points was measured by hand by Sawhney [78]. The ground truth involves 3D coordinates for each point, in a camera-centered coordinate system at the first location of the robot. Measurements were obtained using a steel tape measure; the accuracy of the measurements was estimated to be on the order of 0.1 feet. The remaining 6 points were corners of posters that were not directly measured by hand. For the purposes of this experiment the ground truth for these 6 points was interpolated based on the measured ground truth of surrounding



(a) First image



(b) Final (tenth) image

Figure 4.4: Lobby Image Sequence.

Table 4.6: Lobby Sequence: The image coordinates of the 29 points in the first image.

Point	x	y	Point	x	y
1	90.0	76.1	16	-6.2	9.9
2	75.9	73.7	17	-6.1	-2.3
3	75.6	29.5	18	-9.7	-36.3
4	90.0	29.9	19	-9.7	-20.8
5	85.1	62.8	20	-19.1	-36.4
6	79.4	61.9	21	95.8	-17.2
7	79.7	44.0	22	27.2	-29.0
8	85.1	45.0	23	35.7	-34.6
9	66.5	65.7	24	40.0	80.4
10	54.7	63.1	25	33.4	53.2
11	54.7	23.0	26	15.7	47.4
12	66.3	22.8	27	-2.5	59.8
13	12.8	79.3	28	-2.6	46.4
14	49.2	-36.4	29	-14.3	46.4
15	-2.2	21.2			

Table 4.7: Lobby Sequence: Ground truth of tracked points with respect to the first camera position, in a camera-centered coordinate system.

Point	X ft	Y ft	Z ft	Point	X ft	Y ft	Z ft
1	5.6	3.9	30.7	16	-0.5	0.6	35.8
2	4.9	3.9	31.9	17	-0.4	-0.1	35.8
3	4.9	1.6	31.9	18	-0.7	-2.2	35.8
4	5.6	1.5	30.7	19	-0.7	-1.2	35.8
5	5.4	3.3	31.1	20	-1.4	-2.2	35.8
6	5.1	3.3	31.5	21	5.9	-0.9	29.9
7	5.1	2.3	31.5	22	1.4	-1.2	25.7
8	5.4	2.3	31.1	23	1.9	-1.5	25.7
9	4.5	3.7	33.3	24	3.5	5.7	42.7
10	3.9	3.7	35.1	25	2.9	3.9	43.4
11	3.9	1.3	35.1	26	1.4	3.5	42.8
12	4.5	1.3	33.3	27	-0.3	4.4	42.2
13	1.1	5.7	42.7	28	-0.3	3.5	42.2
14	2.6	-1.6	25.7	29	-1.2	3.4	41.8
15	-0.2	1.3	35.8				

points.⁴ The ground truth for all 29 tracked points in the first position of the camera are reported in Table 4.7. The points were at a distance of 25–44 feet from the camera. The camera parameters for this experiment are given in Table 4.8.

4.4.2 Input to the Algorithms

The 2D image coordinates of the 29 tracked points across the ten images (provided in Table 4.6) constitute the main input to the algorithms along with the camera parameters. As in the previous experiment the error covariance was fixed at 1 pixel. The initial estimate of camera motion was taken to be the command to the robot

⁴Interpolation was carried out with the help of an accurate algorithm due to Collins [19].

Table 4.8: Lobby Sequence: The camera parameters of the Sony AVC-D1 camera that is mounted on the mobile robot.

focal length	fov X	fov Y	Size
16 mm	29.3°	22.9°	256 × 242

(i.e. zero rotation and translation of 1.4 feet along Z) and the scale of the model was input as a single number (cf. Section 4.3.2 for further details).

4.4.3 Results of the Four Algorithms

4.4.3.1 Representation of the Results

The available ground truth for the lobby sequence consists of 3D coordinates of points in a camera-centered coordinate system (as opposed to the object-centered coordinate system, of the rotating box sequence). That is, the *structure* of the lobby was measured, which provides more information than just having the shape measurements as in the case of the box. This enables a comparison of the models recovered by each of the four algorithms in terms of both the shape and position of the objects in the environment relative to the camera. Furthermore, since the recovered model and the ground truth are both in a camera-centered coordinate system, the accuracy of the models can be directly compared without alignment (Section 4.3.3.1).

For any 3D point in a recovered model the difference in 3D location between the recovered point and the actual point (ground truth) is calculated as a percentage of the true distance from the origin of the camera to the actual point. This percentage is the error in recovering the particular 3D point. The error in the 3D model at any iteration will be reported as the mean of the percentage errors over all the points in the 3D model. Note that for the same absolute deviation from ground truth, a point

closer to the camera results in a larger percentage error than a point further away. Since the accuracy of nearby points is critical in tasks such as obstacle avoidance, an error measure which gives more importance to nearby points is reasonable.

The ground truth was measured at the first camera location. In order to compare the models obtained at the subsequent camera locations, the measured ground truth was transferred to each position of the robot using the coordinate transform obtained from the apparently highly accurate position estimation algorithm of Kumar [52]. However, it is possible that the position estimation algorithm introduces error into the transformed ground truth. The information on the exact bound of this error is not directly available, although it may be possible to determine the bound based on an analysis of Kumar's algorithm. Given the prior experimental accuracy of the algorithm it is assumed that such an error is negligible. The pose estimation results have also been used to transform the ground truth for this sequence by Sawhney [78]. In any case our primary Let us now turn to the results of the four algorithms for the indoor lobby sequence.

4.4.3.2 Discussion of the Results

As with the box sequence, the graph in Figure 4.5 shows that the error in the 3D model obtained by the Two-Frame algorithm fluctuates randomly, although the two-frame models at frames 7-10 are quite good. The standard deviation shown in Figure 4.6 also fluctuates for the two-frame models. The last two-frame model has an error of 3.4%, with a standard deviation of 3.6%.

Figures 4.5 and 4.6 show that the errors in the 3D models obtained from blind averaging are fairly stable. The error in the model decreases monotonically, because the two-frame models improve across the later frames. The final model has an error of 9.1% (with standard deviation of 3.3%). As in the box sequence, the result of the standard KF algorithm is worse than blind averaging, except for the final frame,

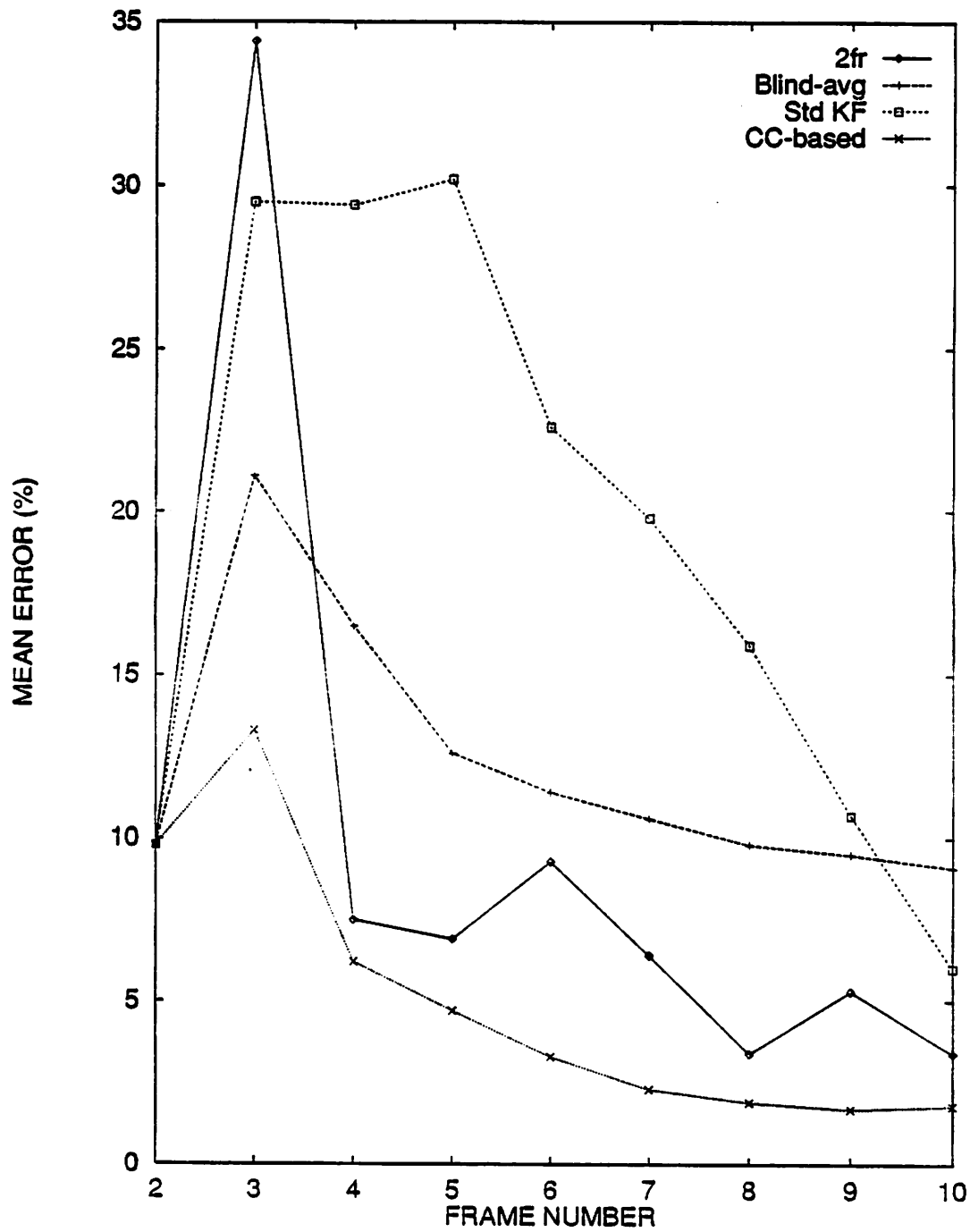


Figure 4.5: Lobby Sequence: Mean error in the 3D models for the four algorithms. The mean errors are also listed in Table 4.9.

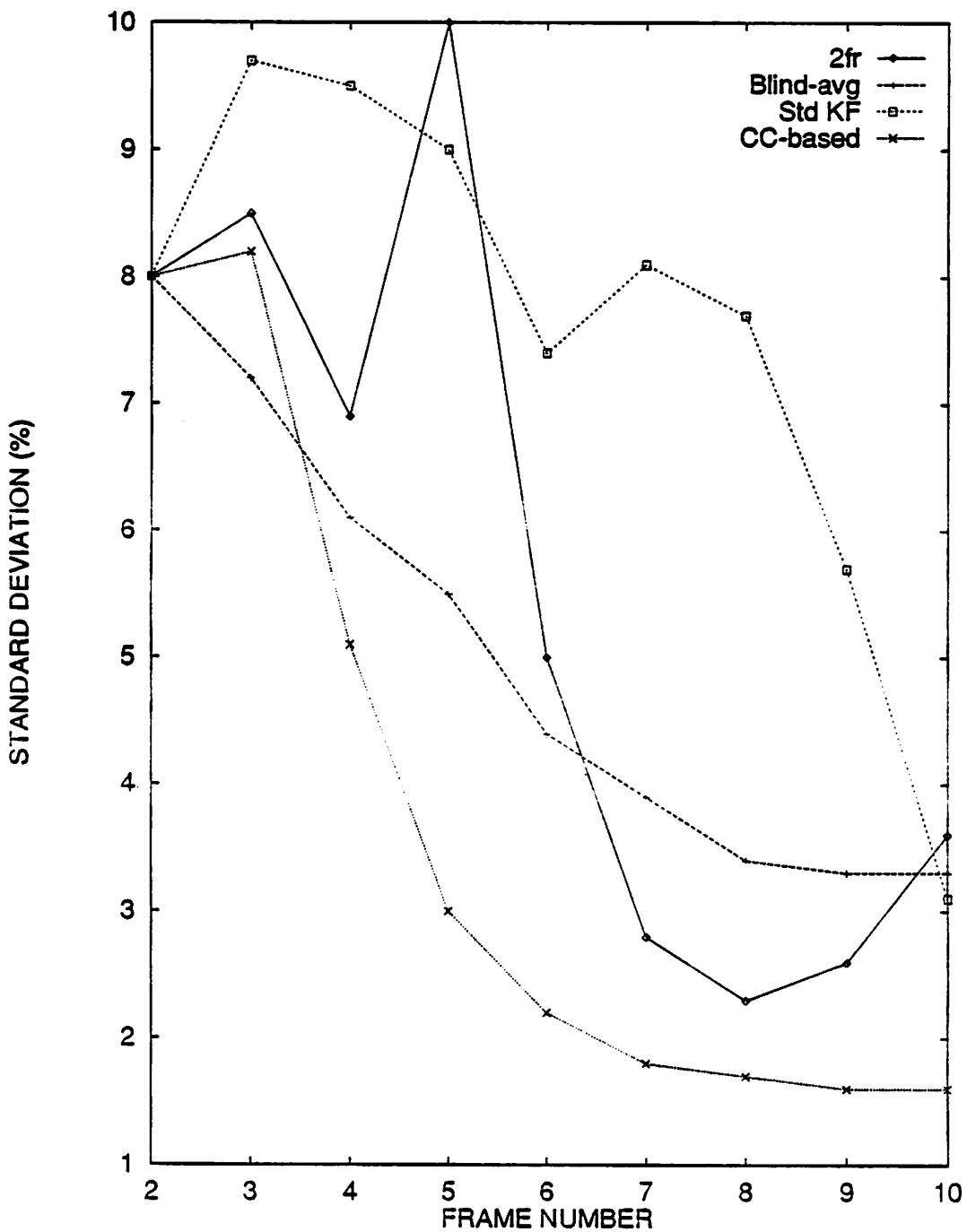


Figure 4.6: Lobby Sequence: Standard Deviation of the error in the 3D models for the four algorithms. The values of the standard deviation are also listed in Table 4.9.

Table 4.9: Lobby Sequence: Mean percentage 3D model errors at each frame for the four algorithms with standard deviations σ .

FRAME	Two-Frame		Blind-Avg		Std KF		CC-based	
	mean	σ	mean	σ	mean	σ	mean	σ
2	9.8	8.0	9.8	8.0	9.8	8.0	9.8	8.0
3	34.4	8.5	21.1	7.2	29.5	9.7	13.3	8.2
4	7.5	6.9	16.5	6.1	29.4	9.5	6.2	5.1
5	6.9	10.0	12.6	5.5	30.2	9.0	4.7	3.0
6	9.3	5.0	11.4	4.4	22.6	7.4	3.3	2.2
7	6.4	2.8	10.6	3.9	19.8	8.1	2.3	1.8
8	3.4	2.3	9.8	3.4	15.9	7.7	1.9	1.7
9	5.3	2.6	9.5	3.3	10.7	5.7	1.7	1.6
10	3.4	3.6	9.1	3.3	6.0	3.1	1.8	1.6

which has an error of 6.0% (standard deviation 3.1%); however the errors are rapidly decreasing after frame 5). Again, using partial weights (as in standard KF) is worse than using equal weights (Blind Averaging) especially at frames 2-5. Once again, the CC-based algorithm yields the best accuracy of the four algorithms compared here. Figure 4.5 shows that the average 3D model error has a decreasing trend after the third frame, with a final error of 1.8% after ten frames with a standard deviation of 1.6%.

There is a striking difference in the behavior of the three multi-frame algorithms at frame 3, where the two-frame model is very erroneous. The CC-based algorithm, although showing a modelst increase in error, preforms quite well regardless of the fact that it had only one prior model to compare to (and combine with) this erroneous model. On the other hand, the standard KF algorithm, which lacks the complete error covariance, is unable to deal with this situation; blind averaging performs just as expected.

4.5 Experiment III: Outdoor Mobile Vehicle Sequence

This experiment uses an outdoor image sequence (referred to as the Umass Rocket-field sequence [24]) obtained by the Autonomous Land Vehicle (ALV) at Martin Marietta in Denver, Colorado. The performance of the four algorithms described in Section 4.2 will be compared.

4.5.1 The Image Sequence and Ground Truth

To obtain the images in this sequence, the ALV (with its forward-pointing camera) was commanded to move directly ahead in steps of about 2.5 *ft*. The reported camera parameters are given in Table 4.10 (cf. Dutta, Manmatha, Williams and Riseman [24]). Due to the unevenness of the ground and the ubiquitous vehicular drift, the ALV rotated and deviated from its expected course. An inertial navigation system (INS) on the ALV was used to monitor the vehicle's actual motion and that will be treated as the ground truth for the motion. The measurements of the INS have an estimated translation error of 0.4% of the distance travelled and an estimated rotation error of σ 0.06° (for elevation and roll) and 0.03° (for azimuth).

Table 4.10: Rocket-Field: The parameters of the camera mounted on the ALV.

focal length	fov X	fov Y	Size
6 mm	72°	57°	255 × 246

The scene viewed by the ALV consisted of several obstacles, the closest ones being traffic cones and trash cans. Further away there were low buildings and a truck, and in the background there were electric poles and mountains. Figure 4.7 provides the



(a) First image



(b) Final (eleventh) image

Figure 4.7: Rocket-Field Image Sequence.

first and the eleventh images of this sequence. In this experiment only the first eleven frames were used because several of the obstacles with known ground truth are no longer visible in the later frames.

Obtaining an outdoor sequence with ground truth involves careful and time-consuming measurements with sophisticated equipment. Due to the lack of well-defined geometric structures and due to the large distances involved, measuring the ground truth in outdoor environments is a formidable task. The ground truth was measured using surveying equipment for corners of 13 landmarks appearing in the first eleven images. In this experiment the ground truth measurements [24] consist of not only the 3D coordinates of the selected points in a camera-centered coordinate system, but also the movement of the camera provided by the Inertial Navigation System.

4.5.2 Input to the Algorithms

As in the previous experiments, the input to the four algorithms consisted of the camera parameters, the tracked image coordinates, the covariance of the image error, the estimate of interframe camera motion, and the scale of the model; details are provided below.

4.5.2.1 Tracked Image Coordinates

Of the 13 points for which ground truth was measured, two points furthest away from the vehicle were discarded in this experiment. One of them is an unclear point on a bush on a mountain on the left-hand side of the image, while the other point is the top of an electric pole near the Focus of Expansion (FOE). Owing to the distance of the second point (635 feet) and its proximity to the FOE throughout the sequence, there is not enough information to recover the top of the electric pole using a SFM

Table 4.11: Rocket-Field Sequence: Ground truth for the tracked image points of the sequence in the camera-coordinate system at the first camera position. The ground truth was available only for the first 11 tracked points used in this sequence. The last two columns denote the distance of the point from the camera i.e., $\sqrt{X^2 + Y^2 + Z^2}$ in meters and feet, respectively.

Point	X m	Y m	Z m	Distance m	Distance ft
1	8.85	-8.37	24.62	27.47	91.56
2	10.50	-10.49	33.79	36.91	123.03
3	4.00	-8.26	24.25	25.93	86.42
4	-2.61	-6.75	17.79	19.20	64.00
5	-6.66	-8.97	27.89	30.04	100.14
6	0.48	-8.47	24.77	26.18	87.28
7	-2.80	-8.01	60.54	61.13	203.78
8	-2.20	-10.08	32.37	33.97	113.23
9	10.30	-10.43	43.87	46.26	154.18
10	0.74	-10.60	40.11	41.50	138.32
11	19.35	-14.17	47.78	53.46	178.20

algorithm; for example, Franzen [30] reconstructs the point as being at 10^{10} feet (p.131). All the remaining 11 points with known ground truth were tracked across the eleven images. The distance from the vehicle to the objects ranged from 64 ft to 204 ft. The ground truth is reported in Table 4.11.

In addition to the 11 points with ground truth, 11 additional points located on the obstacles in the scene were chosen for this experiment. The image coordinates of all 22 points are provided in Table 4.12. In general, Two-Frame SFM does poorly with few points. Therefore, it is better to track more points – even without ground truth – since this is beneficial for estimating the camera motion more accurately. A more accurate 3D model (including the 3D points with ground truth) can thus be obtained from Two-Frame SFM.

Table 4.12: Rocket-Field Sequence: The image coordinates of the 22 points in the eleventh image of the sequence. The points were hand-selected in the eleventh image, since the scene had the best clarity of all the images in the sequence. See Figure 4.7 for a comparison of the first and eleventh image.

Point	x	y	Point	x	y
1	120.5	-88.0	12	43.5	-52.0
2	92.5	-74.0	13	120.5	-98.0
3	64.5	-91.0	14	92.5	-80.0
4	-28.5	-121.0	15	63.5	-101.0
5	-46.5	-83.0	16	-46.5	-90.0
6	18.5	-93.0	17	18.5	-101.0
7	2.5	-24.0	18	23.5	-101.0
8	-2.5	-79.0	19	23.5	-93.0
9	66.5	-52.0	20	19.5	-74.0
10	19.5	-51.0	21	32.5	-51.0
11	105.5	-67.0	22	98.5	-69.0

Tracking of this image sequence was made difficult due to the blurring of small objects (such as the cones and trash cans; cf. Figure 4.7). The 22 selected points were tracked using Anandan's algorithm [5]. Due to the blurring in the images in this sequence, resulting in gross errors (i.e. tracking the wrong points), tracking was checked by hand and gross errors were reset to within one pixel. On average, two thirds of the points were tracked automatically between any two images. Note that the performance of the CC-based algorithm in outdoor environments with unsupervised tracking remains to be tested. It is, however, important to remember that the main purpose of this experiment is to compare the performance of the four algorithms, all of which use the same tracking information as input.

4.5.2.2 The Covariance of the Image Error

The covariance of the image error in the tracked points was set to 1 pixel, just as in the previous experiments. This choice is valid since the manual checking/resetting ensured that tracking was correct up to a pixel.

4.5.2.3 Guess of Interframe Camera Motion

Unlike in the previous experiments, an accurate value of the vehicle's motion was determined by the Inertial Navigation System (INS) on the vehicle. For the initial guess of motion required by the Two-Frame algorithm, the INS value (rounded off to the first decimal place) was given. Since the command issued to the vehicle was not reported along with the sequence in [24], the rounded INS value was taken to represent the original command. Table 4.13 lists the original INS value, the rounded value, and the final camera motion from the Two-Frame algorithm.

4.5.2.4 Scale of the Model

Recall that in SFM the scale of the model cannot be recovered from the images because the exact distance the camera actually moves (magnitude of camera translation) is unknown. In Experiments I and II, since the precise camera translation was unknown, the scale of the model was set to a constant value (cf. Section 4.3.2.4). However, in this experiment the distance of the vehicle's movement is known (up to the accuracy of the INS). This distance information was provided as input to the Two-Frame SFM algorithm at each position of the vehicle resulting in a 3D model with the correct scale. Note that this information corresponds to the one unrecoverable motion parameter; the Two-Frame algorithm still needed to calculate the remaining five camera motion parameters. Since the two-frame models were of the correct scale, they could be fused by the multi-frame algorithms without further adjustment.

Table 4.13: Rocket-Field Sequence: The ground truth, initial guesses and recovered values of the interframe camera motion. (T_x, T_y, T_z) denotes the translation in meters, (R_x, R_y, R_z) denotes the rotation axis and θ denotes the rotation angle in degrees. Of the three rows of motion parameters associated with each interframe camera motion, the first row lists the ground truth (INS value), the second row lists the initial guess (rounded INS) and the third row lists the value recovered by the Two-Frame algorithm. Since the command issued to the vehicle is not reported [24], the INS value was rounded off to the first decimal place and used instead as the initial guess.

Frame	T_x	T_y	T_z	R_x	R_y	R_z	θ
1-2	-0.01	-0.18	0.78	-0.08	0.98	0.16	0.39
	0.00	-0.20	0.80	-0.10	1.00	0.20	0.40
	0.05	-0.19	0.78	0.19	0.97	0.14	0.50
2-3	-0.02	-0.21	0.93	-0.06	0.97	0.22	0.61
	0.00	-0.20	0.90	-0.10	1.00	0.20	0.60
	0.02	-0.13	0.94	-0.09	0.98	0.20	0.61
3-4	-0.02	-0.19	0.84	0.42	0.90	-0.08	0.64
	0.00	-0.20	0.80	0.40	0.90	-0.10	0.60
	0.01	-0.27	0.81	-0.25	0.84	0.49	0.60
4-5	-0.02	-0.21	0.91	-0.40	0.91	-0.08	0.46
	0.00	-0.20	0.90	-0.40	0.90	-0.10	0.50
	0.05	-0.23	0.91	0.22	0.86	0.47	0.61
5-6	-0.02	-0.23	1.04	-0.25	0.95	-0.15	0.57
	0.00	-0.20	1.00	-0.30	1.00	-0.20	0.60
	-0.01	-0.34	1.01	0.60	0.34	0.72	0.75
6-7	-0.02	-0.18	0.81	0.16	0.99	0.04	0.50
	0.00	-0.20	0.80	0.20	1.00	0.00	0.50
	0.06	-0.22	0.80	0.16	0.97	0.18	0.67
7-8	-0.01	-0.21	0.95	-0.92	0.36	-0.17	0.46
	0.00	-0.20	1.00	-0.90	0.40	-0.20	0.50
	0.02	-0.21	0.95	0.91	0.12	0.40	0.53
8-9	-0.02	-0.19	0.91	0.51	0.86	-0.03	0.52
	0.00	-0.20	0.90	0.50	0.90	0.00	0.50
	0.13	-0.22	0.89	-0.25	0.92	0.30	0.74
9-10	0.00	-0.19	0.85	0.24	-0.84	-0.49	0.56
	0.00	-0.20	0.90	0.20	-0.80	-0.50	0.60
	0.05	-0.30	0.82	0.27	-0.96	0.04	0.59
10-11	-0.01	-0.20	0.87	0.06	0.82	0.56	0.61
	0.00	-0.20	0.90	0.10	0.80	0.60	0.60
	0.03	-0.24	0.86	0.10	0.98	0.15	0.68

4.5.3 Results of the Four Algorithms

4.5.3.1 Representation of the Results

As in the lobby sequence (Experiment II), the ground truth was available in a camera-centered coordinate system, facilitating comparison of the recovered structures. The results of the four algorithms are reported as a mean percentage error exactly as in Experiment II (cf. Section 4.4.3.1). The percentage error is an average over the 11 tracked points with known ground truth.

4.5.3.2 Discussion of the Results

Figure 4.8 provides the mean percentage error as a graph after every movement of the vehicle, and Table 4.14 lists the individual error values. The standard deviations are plotted in Figure 4.9. Two-frame error is typically around 30%, which is much higher than in the indoor robot sequence. The error in the two-frame model fluctuates without any clear trend. The two-frame model is especially erroneous at frame 9, which also has the highest error in rotation angle θ of 0.22° (cf. Table 4.13, frame 8-9, difference between first and third row). Recall that a similar relationship between high rotation error and high two-frame error was observed in Experiment I. The high error in the two-frame results in this outdoor sequence is expected because of the low image resolution, compounded by the fact that most of the tracked points lie near the FOE in this sequence. Furthermore, the distance the vehicle moved between images (about $1m$) is very small compared to the distance of the tracked points from the camera (cf. Table 4.11 for distances to the points); in such a case the two-frame error is expected to be high (cf. Dutta [22]).

The errors in the first five 3D models (frames 2-6) obtained by the Blind Averaging algorithm (Figure 4.8) reflect the average of the errors in the individual two-frame models. From frame 7 onwards the blind average result is better than expected, if

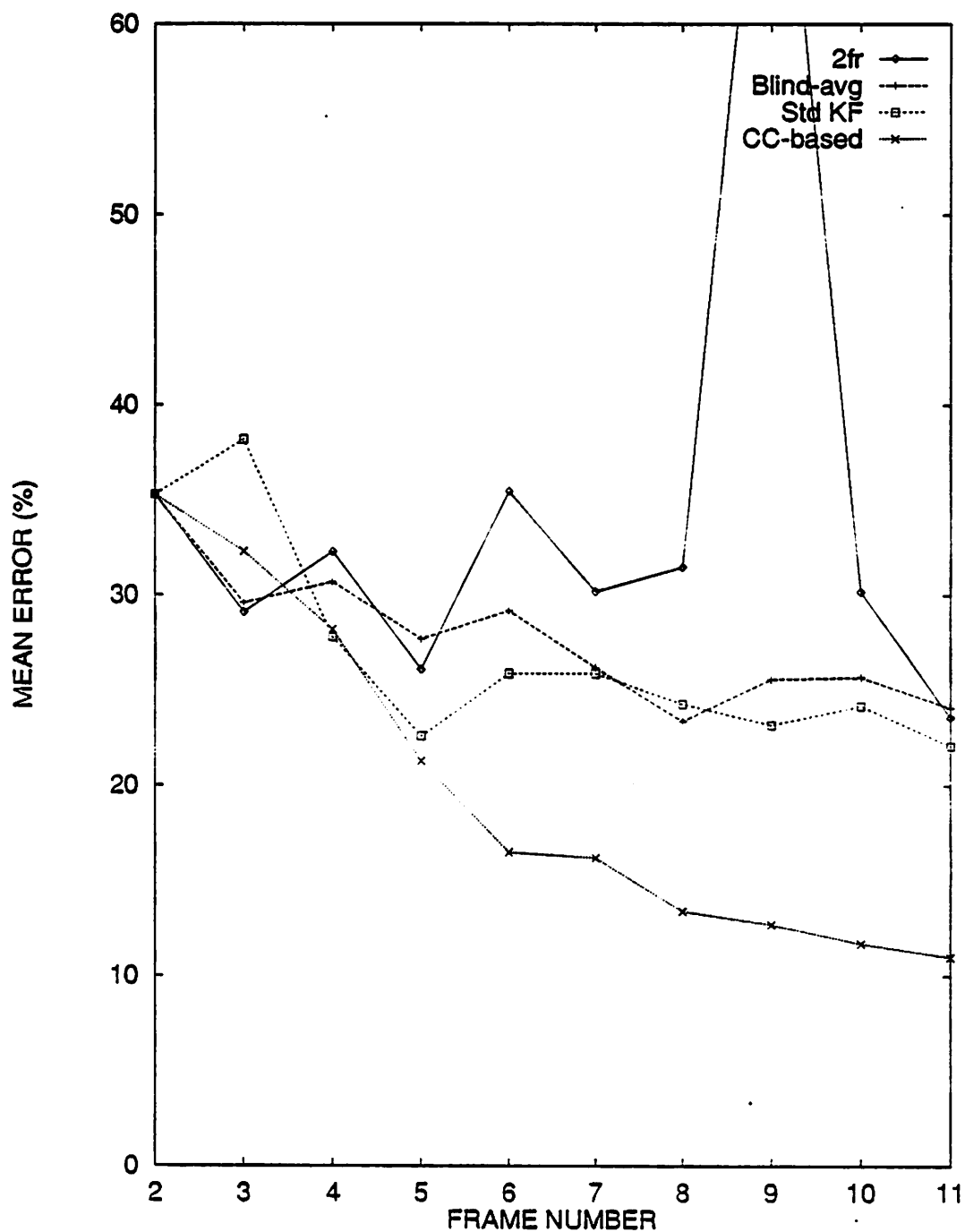


Figure 4.8: Rocket-Field Sequence: Mean percentage error in the 3D models for the four algorithms. The mean errors are also listed in Table 4.14. At Frame 9 the error in the two-frame model is 77.3%.

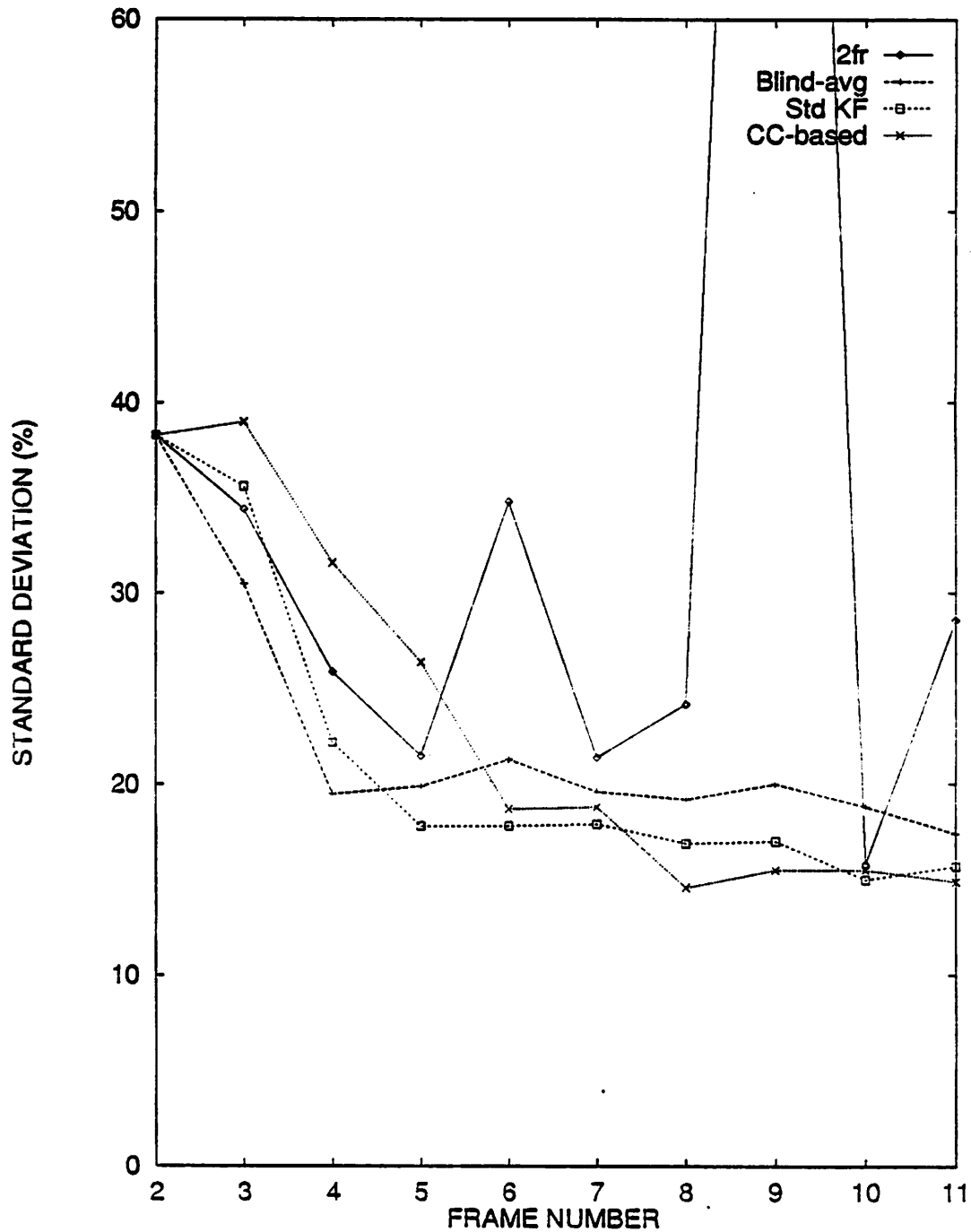


Figure 4.9: Rocket-Field Sequence: Standard deviation of the percentage error the of 3D models for the four algorithms. The standard deviations are also listed in Table 4.14. At Frame 9 the standard deviation in the two-frame model is 134.5%.

Table 4.14: Rocket-Field Sequence: Mean percentage 3D model errors at each frame for the four algorithms with standard deviations σ .

FRAME	Two-Frame		Blind-Avg		Std KF		CC-based	
	mean	σ	mean	σ	mean	σ	mean	σ
2	35.3	38.3	35.3	38.3	35.3	38.3	35.3	38.3
3	29.1	34.4	29.6	30.5	38.2	35.6	32.3	39.0
4	32.3	25.9	30.7	19.5	27.8	22.2	28.2	31.6
5	26.1	21.5	27.7	19.9	22.6	17.8	21.3	26.4
6	35.5	34.8	29.2	21.3	25.9	17.8	16.5	18.7
7	30.2	21.4	26.2	19.6	25.9	17.9	16.2	18.8
8	31.5	24.2	23.4	19.2	24.3	16.9	13.4	14.6
9	77.3	134.5	25.6	20.0	23.2	17.0	12.7	15.5
10	30.2	15.8	25.7	18.8	24.2	15.0	11.7	15.5
11	23.6	28.6	24.1	17.4	22.1	15.7	11.0	14.9

the two-frame absolute values were averaged. The final 3D model obtained by blind averaging has an error of 24.1%, with a standard deviation of 17.4%. The performance of the standard KF algorithm is slightly better but comparable to the blind averaging algorithm, with a final result of 22.1% (standard deviation 15.7%). Note that neither blind averaging nor standard KF result in monotonically decreasing errors.

The apparently unexpected behavior of blind averaging at frame 9 is a straightforward result of the computation, the results of which are recorded in Table 4.15–4.17. Table 4.15 lists the ground truth points at frame 8, and the blind average model corresponding to these 11 3D points. The absolute error and the percentage error are calculated as described in Section 4.4.3.1. Note that the mean error of the blind average model is 23.4%. Similarly, Table 4.16 lists the ground truth and corresponding Two-Frame model at frame 9. The mean error of 77.3% in the Two-Frame model is the highest in the sequence.

Table 4.15: Rocket-Field Sequence Frame 8: Ground Truth, Blind Average 3D Model, and Blind Average Error. All values are in meters. Dist represents the distance to the point from the camera.

Point no.	Truth (Frame 8)				Blind Average (Frame 8)				
	X	Y	Z	Dist	X	Y	Z	Error	% Error
1	10.2	-6.8	17.9	21.7	10.4	-6.6	17.9	0.3	1.2
2	12.3	-8.9	27.0	31.0	12.2	-8.5	26.0	1.1	3.4
3	5.3	-6.7	17.8	19.7	4.1	-5.0	12.8	5.4	27.2
4	-1.6	-5.2	11.7	12.9	-1.7	-5.8	12.3	0.9	6.6
5	-5.1	-7.4	22.0	23.8	-6.3	-9.4	26.3	4.9	20.7
6	1.8	-6.9	18.5	19.8	1.0	-5.1	12.8	6.0	30.3
7	0.5	-6.1	54.4	54.7	0.0	-3.2	28.3	26.2	47.9
8	-0.4	-8.4	26.2	27.6	-0.5	-5.7	16.5	10.1	36.6
9	12.7	-8.7	37.0	40.1	10.8	-7.3	30.8	6.7	16.8
10	2.9	-8.9	33.8	35.1	1.3	-3.1	13.6	21.1	60.0
11	22.0	-12.4	40.5	47.7	20.8	-11.2	37.7	3.2	6.8
								MEAN:	23.4

Table 4.16: Rocket-Field Sequence Frame 9: Ground Truth, Two-Frame 3D Model, and Two-Frame Error.

Point no.	Truth (Frame 9)				Two-Frame (Frame 9)				
	X	Y	Z	Dist	X	Y	Z	Error	% Error
1	10.3	-6.7	16.9	20.9	7.5	-4.6	12.1	5.9	28.2
2	12.6	-8.8	25.9	30.1	10.3	-6.7	21.0	5.8	19.3
3	5.5	-6.6	16.8	18.9	6.8	-8.0	20.9	4.6	24.2
4	-1.5	-5.1	10.8	12.0	-2.3	-8.0	16.3	6.3	52.3
5	-4.9	-7.3	21.1	22.9	-5.6	-8.3	23.3	2.5	11.0
6	2.0	-6.8	17.5	18.9	1.7	-7.5	19.0	1.6	8.7
7	1.0	-6.2	53.5	53.8	0.3	-2.7	24.3	29.3	54.5
8	-0.2	-8.4	25.3	26.6	-0.4	-11.4	33.2	8.5	31.9
9	13.0	-8.7	36.0	39.2	74.0	-48.8	208.4	187.2	477.1
10	3.2	-8.9	32.8	34.2	0.7	-1.5	6.4	27.6	80.7
11	22.3	-12.4	39.3	46.9	36.6	-18.6	64.2	29.3	62.6
								MEAN:	77.3

Table 4.17: Rocket-Field: Obtaining Blind Average (Frame 9) results from Transformed Blind Average (from Frame 8). All values are in meters.

Point no.	Transformed Blind Av.			Result Blind Av. (Frame 9)				
	X	Y	Z	X	Y	Z	Error	% Error
1	10.5	-6.3	16.9	10.1	-6.1	16.3	0.8	4.0
2	12.5	-8.1	25.0	12.2	-7.9	24.5	1.7	5.7
3	4.2	-4.7	11.9	4.5	-5.1	13.0	4.2	22.0
4	-1.7	-5.5	11.5	-1.8	-5.8	12.1	1.5	12.7
5	-6.1	-9.1	25.5	-6.0	-9.0	25.3	4.6	20.3
6	1.1	-4.8	11.9	1.2	-5.2	12.8	5.1	26.8
7	0.2	-2.9	27.4	0.2	-2.9	27.1	26.6	49.5
8	-0.4	-5.5	15.7	-0.4	-6.2	17.9	7.8	29.1
9	11.0	-6.9	29.8	18.9	-12.1	52.1	17.5	44.6
10	1.4	-2.9	12.7	1.3	-2.7	11.9	21.9	64.0
11	21.2	-10.8	36.6	23.1	-11.7	40.1	1.3	2.7
							MEAN:	25.6

In order to obtain the blind average at frame 9, the blind average 3D model from frame 8 is first transformed to the camera coordinate system in frame 9 (cf. Equation 4.2). The result of this transformation is provided in Table 4.17. The transformed blind average model is then averaged with the two-frame model at frame 9 (cf. Equation 4.1). The final result is shown in Table 4.17. Although the two-frame model has large errors in several points, the total impact of these points on the blind average made up of 11 points is diffused. Consider Point 9 which has the highest error in the two-frame model (Table 4.16), especially its Z coordinate (208.4 m as opposed to the ground truth 36.0 m). However, in the transformed blind average, the Z coordinate at frame 9 (Table 4.17) is 29.8 m, which means that the point is estimated to be closer to the camera than it actually is (36.0 m). This indicates that the previous two-frame models have had a low value of the Z coordinate, on an

average. That is, seven previous models “vote” (on an average) for the value 29.8 m whereas one two-frame value “votes” for 208.4 m. The resulting Z value is $\frac{7 \times 29.8 + 208.4}{8}$ m, or 52.1 m (cf. Table 4.17, Point number 9, the second Z column) is a result; the computed error for point 9 is 44.6%, which is much smaller than might be expected since the resulting average Z value is closer to the ground truth due to the alignment of points as shown in Figure 4.10. The effect of this value on the mean percentage error of the model is further diluted by the more accurate points (e.g. 1, 2, and 11). Note that the example we considered is for the Z coordinate, but it holds also for all coordinates across all points.

The results of the CC-based algorithm are clearly the best in this experiment, although its standard deviation is comparable to that of the other two multi-frame algorithms as shown in Figure 4.8. The mean error monotonically decreases and has a final value of 11% (standard deviation 14.9%), which is about half the final error of the other algorithms. Note that there is a decreasing trend in the mean error even at the eleventh frame, although the rate of decrease is slowing down.

Although the CC-based algorithm does considerably better than the other algorithms, the 11% error is high compared to the two indoor sequences reported. The crucial difference between this sequence and the previous ones is that the input to the CC-based algorithm (i.e., the two-frame models) is much more corrupt. In addition, it should be kept in mind in evaluating these results that the accuracy of the ground truth measurements is not known for this sequence, due to the difficulties in collecting data from outdoor environments. Another problem specific to this sequence is that the location of the center of the camera was not provided in [24]. Nevertheless, although the average error in the final 3D model obtained by the CC-based algorithm is high, Table 4.18 reveals that most of the error is due to two points (7 and 10). These points are the two points closest to the FOE, the top of the telephone pole in

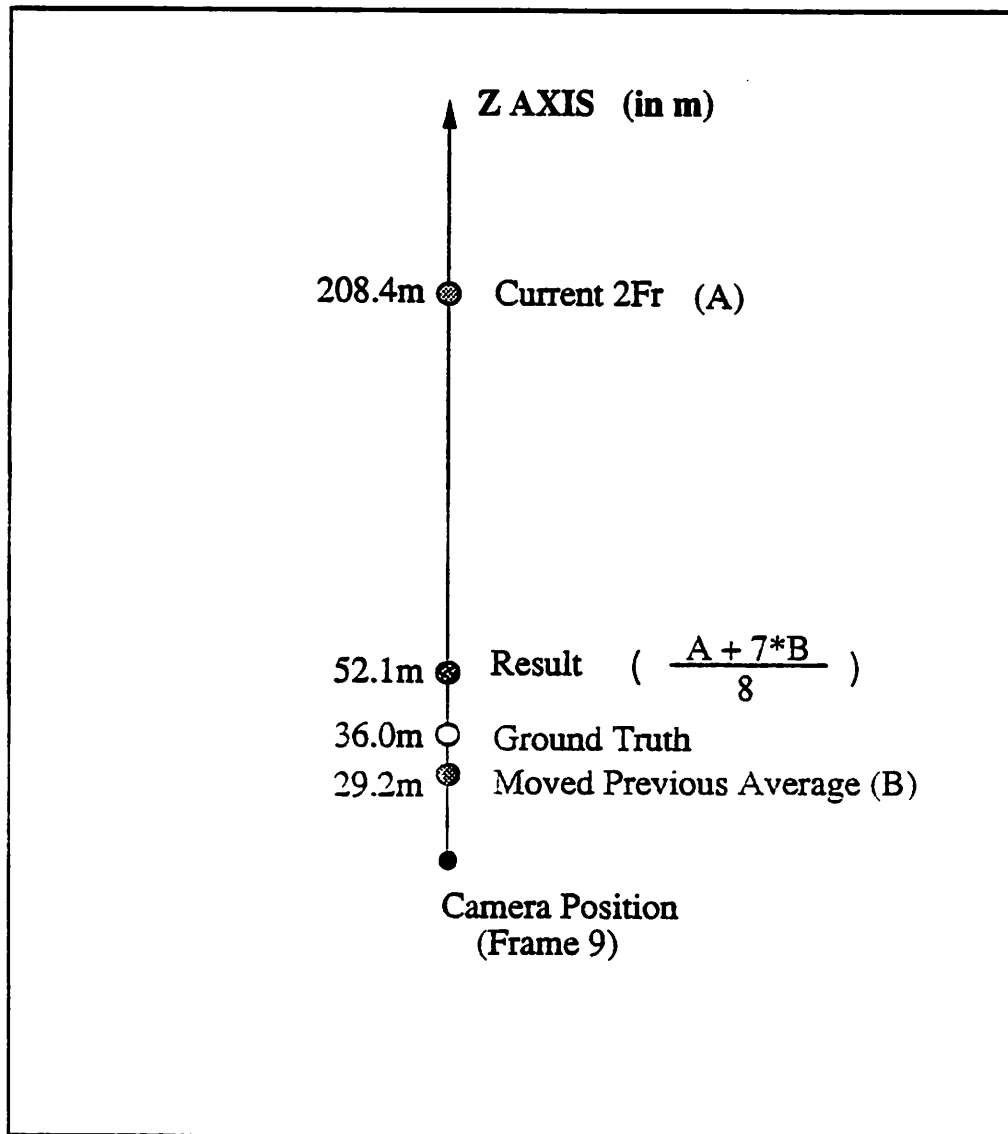


Figure 4.10: Rocket-Field Sequence: Resulting Z coordinate of point 9 after blind averaging. Note that the average Z coordinate of the first seven 3D models is 29.2 m which is closer to the camera than where the point actually is (36.0 m). Although the Z coordinate in the next 3D model is 208.4 m, the final averaged result is only 52.1 m.

Table 4.18: Rocket-Field Sequence Final Frame: Ground Truth, CC-based 3D Model, and CC-based 3D Model Error. All values are in meters. Error represents the distance from a point in the CC-based model to where it should have been (i.e. ground truth). The dashes indicate unavailable data. Note that although the CC-based algorithm has a mean error of 11.0, the two highly erroneous points (7 and 10) skew this result.

Point no.	Truth (Frame 11)			CC-based (Frame 11)			Error (meter)	True Dist	Error (%)
	X	Y	Z	X	Y	Z			
1	10.4	-6.4	15.1	10.5	-6.0	15.4	0.5	19.4	2.6
2	12.6	-8.5	24.2	12.3	-7.7	23.4	1.2	28.5	4.1
3	5.5	-6.2	15.0	5.5	-6.0	14.9	0.3	17.2	1.8
4	-1.5	-4.7	9.0	-1.5	-4.9	9.3	0.4	10.3	3.6
5	-4.9	-7.0	19.4	-5.5	-7.6	20.9	1.7	21.2	8.1
6	2.0	-6.5	15.8	1.6	-6.4	15.6	0.4	17.2	2.5
7	1.0	-6.0	51.7	0.4	-2.7	27.8	24.2	52.1	46.5
8	-0.2	-8.1	23.5	-0.4	-8.4	24.3	0.8	24.9	3.2
9	13.0	-8.4	34.2	12.1	-7.4	32.1	2.5	37.6	6.7
10	3.3	-8.6	31.1	2.4	-4.7	20.9	11.0	32.4	33.9
11	22.3	-12.1	37.6	20.8	-10.1	34.6	3.9	45.3	8.5
12	-	-	-	7.8	-7.2	31.6	-	-	-
13	-	-	-	10.6	-6.6	15.5	-	-	-
14	-	-	-	12.6	-8.4	23.9	-	-	-
15	-	-	-	5.2	-6.4	14.5	-	-	-
16	-	-	-	-5.2	-7.8	19.7	-	-	-
17	-	-	-	1.5	-6.4	14.3	-	-	-
18	-	-	-	1.9	-6.3	14.2	-	-	-
19	-	-	-	2.0	-6.0	14.6	-	-	-
20	-	-	-	2.8	-8.2	25.1	-	-	-
21	-	-	-	5.3	-6.5	28.9	-	-	-
22	-	-	-	20.6	-11.0	36.6	-	-	-
								MEAN	11.0

the middle of the picture (point 7) and the corner of the low building in front of the pole (cf. Figure 4.7). Furthermore, the top of the pole is the furthest tracked point. Without these two points the average error is 4.6% as opposed to 11.0% for all the 11 points with ground truth.

The three experiments discussed indicate that the CC-based algorithm which takes into account the cross-correlation of the error between pairs of 3D points is consistently superior to the algorithms that ignore these cross-correlation terms.

4.6 Application to Robot Navigation

The aim of the two experiments in the remainder of this chapter is to demonstrate the usefulness of the CC-based algorithm for the problem of position estimation in the context of autonomous navigation. One of the goals of the UMass Unmanned Ground Vehicle (cf. Chapter 1) is to keep track of its position with respect to a set of known and modeled landmarks in the world. Estimating a robot's position using vision-based techniques has traditionally employed a 3D model of the scene and a 2D view (image) of the environment from the current position of the robot. Although several techniques have been developed that determine the robot's position (or so-called *exterior orientation* or *pose estimation* [41], [56] [58] [52]), these techniques require either a fairly accurate 3D model or a reliable estimate of the model noise in case the 3D model is noisy. Since it is difficult to automatically obtain accurate 3D models of unconstrained environments, or to reliably estimate the error in noisy models, in previous work the 3D models have been constructed manually.

In recent work, Kumar and Hanson [53] have combined Sawhney's automatic determination of shallow structure [78] (frontal-planar surface representation) to serve as an initial model acquisition algorithm, followed by the pose estimation algorithm developed by Kumar [52] for model extension of newly tracked points. Although

the resulting algorithm is applicable to many environments, it requires frontal-planar objects for the initial model acquisition step, and therefore non-shallow surfaces must be distant. The CC-based algorithm is proposed here as an alternative for the model acquisition step, since it does not impose any such constraints on the environment and can work for arbitrary camera motions. Moreover, the 3D model obtained by the CC-based algorithm provides a covariance matrix that reliably represents the noise in the model, which is directly useful for Kumar's position estimation algorithm - the only reported position estimation algorithm that can utilize noise estimates in the 3D model.

The following two experiments are reported to provide empirical evidence for the plausibility of coupling the CC-based algorithm with Kumar's position estimation algorithm. The first experiment is a simulation and the second experiment involves real imagery obtained from a moving robot.

The experiments consist of two stages. In the initial bootstrapping phase (the model acquisition stage) the robot is moved to various positions and it obtains an image from each position. Using the sequence of images, a 3D model of the environment is constructed by the CC-based algorithm. In the second phase the robot continues to move, while keeping track of its position with respect to the acquired 3D model.

Kumar's algorithm determines the robot's rotation and translation with respect to the origin of the coordinate system of the 3D model. In order to determine the robot motion, the 3D model and the 2D image measurements are related through an unknown coordinate transform (of rotation and translation) and a known perspective projection transform (of the camera). The problem of determining the robot pose is then cast as an optimization problem in terms of the 3D model coordinates and the corresponding 2D image coordinates; solutions typically involve robust non-linear optimization techniques. For further details see Kumar [52].

The CC-based algorithm generates the entire covariance matrix consisting of $9n^2$ elements to represent the error in the 3D model. However, Kumar's algorithm uses only $9n$ terms. This discrepancy does not affect the coupling of the two algorithms, however, for the following reasons. The effect of the cross-correlations has already been taken into account in deriving the 3D model. Assuming that the 3D model is accurate, Kumar's algorithm does not require the cross-correlations. In fact, Kumar (cf. Chapter 4 of [52]) points out that he has ignored the cross-correlations because the noise in the 3D model is assumed to be small, which holds in this case due to accuracy of the CC-based algorithm (at least for the indoor sequences). Kumar also states that if the noise in the 3D model were large, it may not be possible to ignore the effects of the cross-correlations even in the position estimation algorithm.

4.7 Experiment IV: Simulated Model Acquisition and Model-Based Navigation

The first experiment is a synthetic experiment that involves determining the position of the robot outside the Computer Science building at UMass.

4.7.1 Simulating the Image Sequences

In order to obtain the image sequence for the CC-based algorithm, a simulated robot was moved along the perimeter of a circle (6 feet in radius) with the robot's simulated camera aimed approximately at the building about 130 feet away. Figure 4.11 shows a wire-frame model of the building. This model consists of 30 points which are corners of walls, stairwells, and a cooling tank (at the top right-hand corner of the building). The points have been connected into faces in order to facilitate visualizing the model. An image was synthesized for the 30 points at each position of the robot using the simulated parameters of the camera. The camera parameters were: 70°

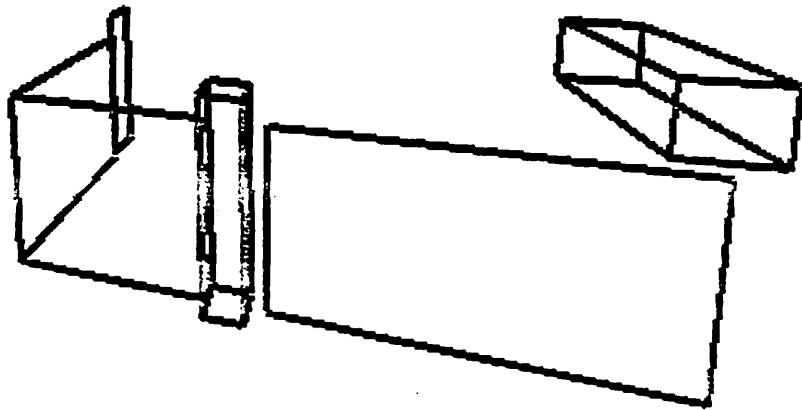


Figure 4.11: Simulated Model Acquisition and Model-Based Navigation: True 3D Model of Building consisting of walls, stairwells, and a cooling tank (at the top right-hand corner of the building). This wire-frame model of the building is based on ground truth. The 30 corner points have been connected into faces in order to facilitate visualizing the model.

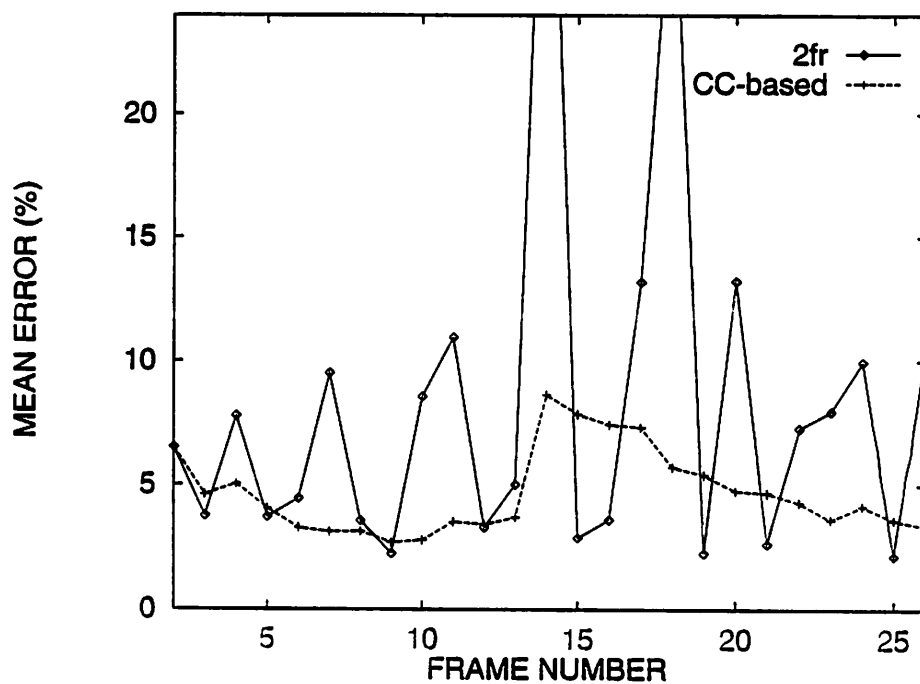
FOV, 512×512 pixel image size, and focal length $15.24mm$. In order to simulate a realistic image sequence, random noise was added to each image. The noise was Gaussian, with zero mean and a standard deviation σ of 0.32 pixels, and bounded to a maximum value of 2σ .⁵ Such noise with its bound is comparable to the tracking errors in real image sequences.

For the second phase of this experiment – position estimation – a second simulated image sequence was synthesized. In this phase the simulated robot was moved 27 ft towards the building in steps of 3 ft. The robot was rotated about its vertical axis with alternating rotations of -1° , 0° , and 1.0° . Again, at each position a synthetic image was obtained and the same amount of Gaussian noise was added as in the first sequence.

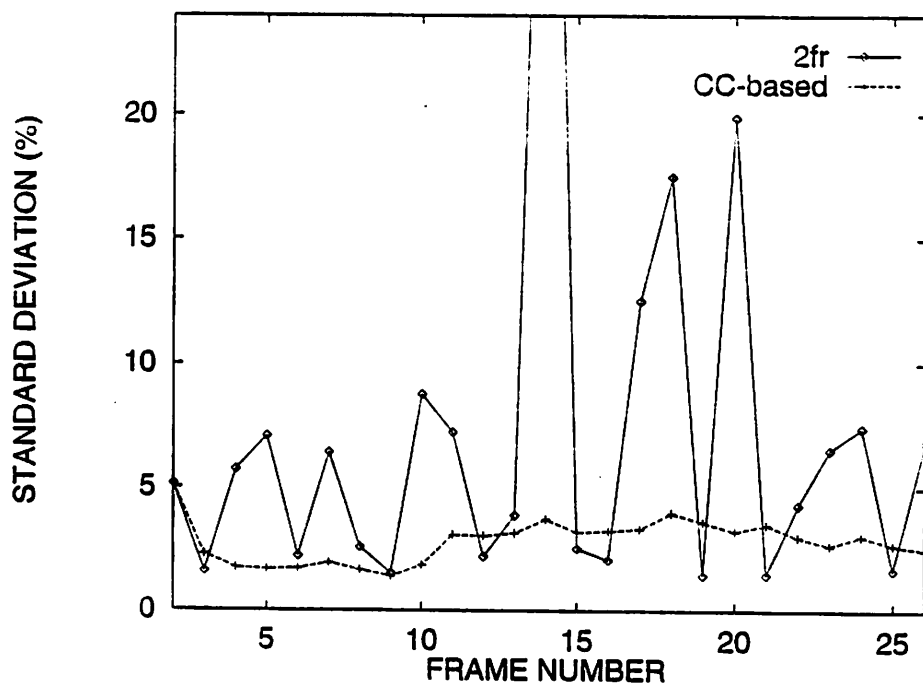
4.7.2 Acquiring the 3D Model

At the bootstrapping model acquisition stage of the experiment, the CC-based algorithm was applied to the first simulated image sequence. The error in the 3D model was calculated as a percentage of the actual distance, as in Experiments II and III. Figure 4.12 shows the mean error and standard deviation of the 3D model acquired by the bootstrapping stage at each position. The final model is accurate to 3.35% (standard deviation 2.52%). That is, in the final model the average error is 6.1 ft for 30 points between 149 and 290 ft from the robot. The graphs in Figure 4.12 show how drastically Two-Frame SFM can fluctuate; even the CC-based algorithm is affected by the very erroneous (39% error) two-frame model at frame 14. However, the CC-based algorithm is able to recover from the effect of this frame. The graphs also bring out the output characteristic of the CC-based algorithm over a longer sequence of images than those considered in the real-world sequences of Experiments

⁵The Gaussian noise was bounded to prevent the occurrence of outliers (theoretically involving infinitely large values of noise) which otherwise arise from the shape of the Gaussian distribution.



(a) Mean percentage error. Values not shown: 39% (fr 14) & 30%



(b) Standard deviation. Value not shown: 49% (fr 14)

Figure 4.12: Simulated Model Acquisition Experiment: Results of the acquired model.

I-III. Over long sequences, the behaviour of the CC-based algorithm is akin to a properly functioning filter, i.e. the algorithm takes as input information with high fluctuations of error and produces as output *filtered* information with more uniform (lower) error.

4.7.3 Determining the Position of the Robot

Recall that the purpose of this part of the experiment is to demonstrate that the 3D models acquired by the CC-based algorithm are sufficiently accurate to use for pose recovery. In this stage of the experiment, the 3D model and its error covariance matrix were employed by the pose determination algorithm to compare the current image with the 3D model. This enabled the algorithm to determine the robot's position along the path in the second simulated sequence.

Table 4.19 provides the results of the pose determination algorithm using the model acquired by the CC-based algorithm. These results are also depicted as the top view of the path that the robot "thinks" it took in Figure 4.13. This recovered path is compared against the actual path used to generate the second simulated image sequence. The comparison reveals that the robot's movement was recovered within an accuracy of 1.7 feet throughout the robot's path. After 27 feet of motion the robot's recovered final position had an error of 10.7 inches; this corresponds to an error of 3.3% of the total distance travelled, given a CC-based 3D model with 3.4% error.

4.8 Experiment V: Real Model Acquisition and Model-Based Navigation

4.8.1 The Image Sequence

This experiment involves the same real image sequence as used in Experiment II, consisting of 10 images obtained by a robot moving in the lobby of the Computer

Table 4.19: Simulated Model Acquisition and Model-Based Navigation: Ground truth (first row) and recovered pose (second row) using the CC-based model at each frame of the second stage of the experiment. (T_x, T_y, T_z) denotes the translation (in feet), (R_x, R_y, R_z) denotes the rotation axis and θ denotes the rotation angle in degrees.

Frame	R_x	R_y	R_z	θ	T_x	T_y	T_z
1	0.0	1.0	0.0	0.0	0.0	0.0	0.0
	-0.385	0.89	-0.253	-0.11	1.48	-0.074	-0.7062
2	0.0	1.0	0.0	-1.0	0.0	0.0	3.0
	-0.013	1.0	-0.033	-1.18	1.19	-0.067	2.174
3	0.0	1.0	0.0	1.0	0.0	0.0	6.0
	-0.07	0.997	0.034	0.79	1.06	0.18	5.328
4	0.0	1.0	0.0	0.0	0.0	0.0	9.0
	-0.013	1.0	-0.019	-0.31	0.88	0.01	8.453
5	0.0	1.0	0.0	1.0	0.0	0.0	12.0
	0.089	0.996	0.002	0.65	0.76	-0.133	11.56
6	0.0	1.0	0.0	-1.0	0.0	0.0	15.0
	-0.001	1.0	-0.004	-1.49	0.29	0.03	14.56
7	0.0	1.0	0.0	0.0	0.0	0.0	18.0
	-0.153	0.99	-0.031	-0.36	0.54	-0.235	17.20
8	0.0	1.0	0.0	-1.0	0.0	0.0	21.0
	-0.025	1.0	-0.027	-1.44	0.38	-0.098	20.39
9	0.0	1.0	0.0	1.0	0.0	0.0	24.0
	-0.75	0.658	0.075	0.64	-0.160	1.34	22.96
10	0.0	1.0	0.0	0.0	0.0	0.0	27.0
	-0.229	0.97	-0.099	-0.4	0.60	-0.255	26.63

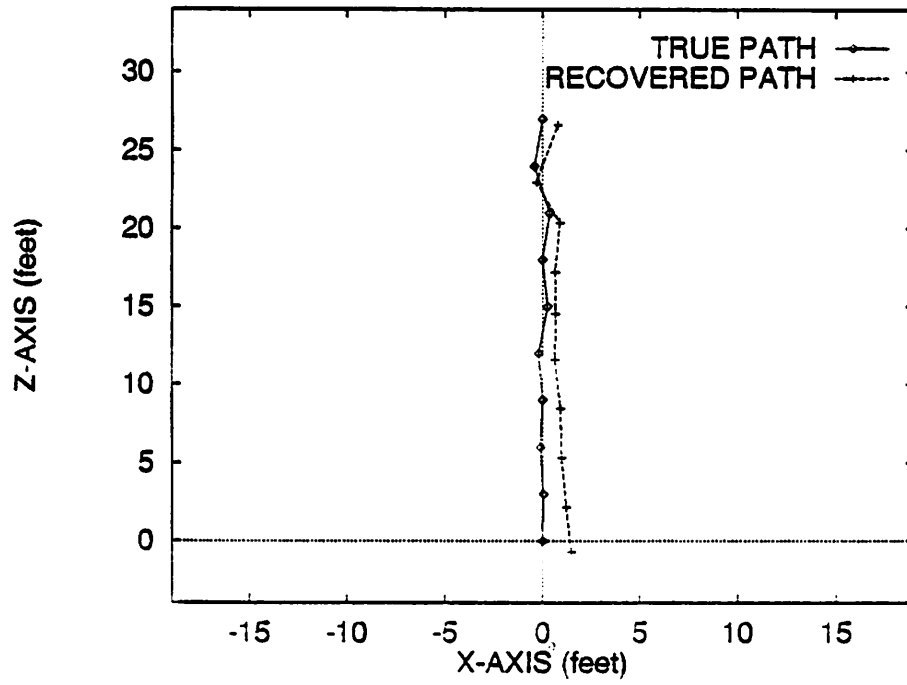


Figure 4.13: Simulated Model-Based Navigation: Results showing top view of the true path and the recovered path

Science Department at UMass (cf. Figure 4.4). The goal of this experiment was to determine the position of the robot as it moved across the lobby, using a 3D model acquired by the CC-based algorithm. As in the previous experiment, there are two parts to the experiment: (i) acquiring the 3D model (bootstrapping stage) and (ii) position estimation (Kumar's algorithm [52]).

Since the experiment consists of two parts, two different image sequences would ideally be required. In the previous experiment simulating two different sequences was straightforward. However, in this real image experiment, we only had access to one image sequence consisting of ten images. Due to this constraint half the sequence was used for each stage of the experiment. This situation is comparable to a typical scenario with a vehicle moving directly ahead on a road. In a such a situation the vehicle could acquire a model as it moves, and then monitor its position with respect to the acquired model while continuing to move along the same direction.

4.8.2 Acquiring the 3D Model

The 3D model obtained by the CC-based algorithm in Experiment II at the sixth frame constitutes the final model of the model acquisition stage. This model has an accuracy of 3.3% (standard deviation 2.2%), as was shown in Table 4.9. This model consists of 29 points which are corners of walls, doors, posters and obstacles in the lobby.

4.8.3 Determining the Position of the Robot

In the second stage of the experiment, the acquired 3D model is applied to the task of position estimation. With respect to this 3D model, the position of the robot as it moves forward from the sixth frame is determined using the image from each remaining location of the sequence. The resulting positions and orientations of the

Table 4.20: Lobby Sequence: Ground truth (first row) and recovered pose (second row) using the CC-based model. (T_x, T_y, T_z) denotes the translation (in feet), (R_x, R_y, R_z) denotes the rotation axis and θ denotes the rotation angle in degrees.

Frame	R_x	R_y	R_z	θ	T_x	T_y	T_z
6	0.83	0.42	0.36	0.35	0.07	-0.12	0.08
	0.38	0.8	-0.46	0.04	0.02	-0.01	0.06
7	0.64	-0.76	-0.12	0.48	0.07	-0.10	1.50
	-0.08	-0.93	-0.36	0.44	0.05	0.04	1.46
8	-0.13	-0.98	-0.17	0.88	0.08	-0.04	2.93
	-0.33	-0.90	-0.28	1.05	0.04	0.05	2.92
9	0.01	-1.00	0.09	1.32	0.09	-0.02	4.35
	-0.24	-0.97	0.01	1.42	0.05	0.13	4.31
10	0.29	-0.93	0.22	1.94	0.09	-0.03	5.75
	0.11	-0.98	0.17	1.93	0.05	0.12	5.73

robot as determined based on the acquired 3D model and Kumar's pose estimation algorithm are given in Table 4.20. Figure 4.14 depicts the top view of the actual path of the robot (i.e. the ground truth) as well as the recovered path. Along the 5.8-foot path of the robot (from frame 6 until frame 10), the error in each recovered robot position lies between 1.1 inches and 1.7 inches. This error corresponds to a deviation of 1.6% - 2.4% of the total distance travelled; at the final position the error is 1.4 inches (2.0%).

4.9 Conclusion

The first three experiments involving different real world situations clearly establish that the Two-Frame Structure from Motion algorithm suffers from erratic

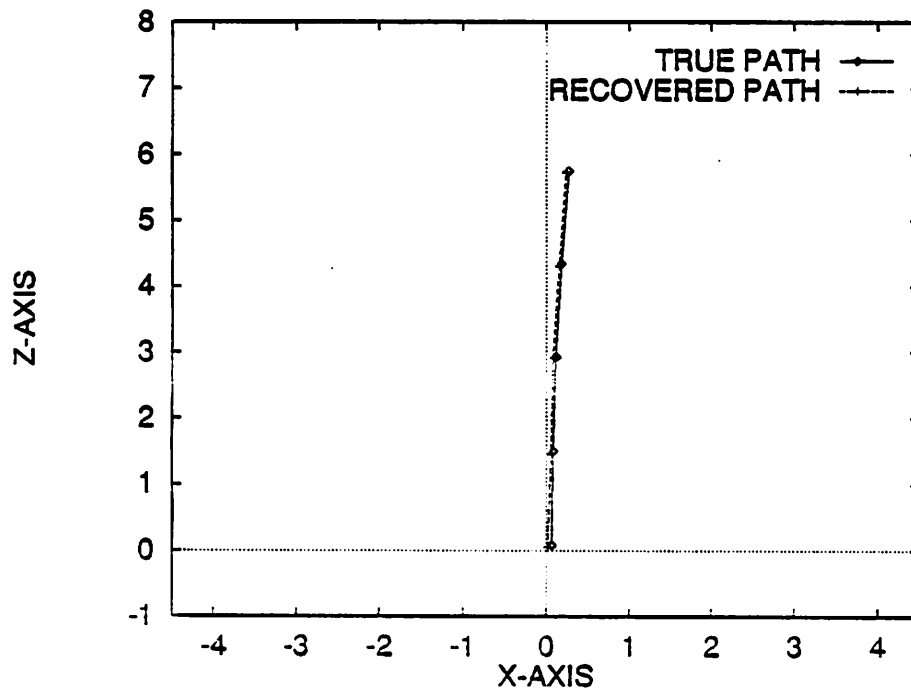


Figure 4.14: Model-Based Navigation in the Lobby: Results showing top view of the true path and the recovered path.

behavior. In contrast, the other three multi-frame algorithms considered behave in a more consistent manner, as is expected when a fusing several two-frame models.

Of the three multi-frame algorithms, the Blind Averaging algorithm maintains a non-weighted average of the two-frame models, while the other two algorithms involve weighted averaging. The standard KF algorithm corresponds to the most advanced previously reported incremental MFSFM algorithm (cf. Chapter 2). Experiments I-III demonstrate that the incomplete weights employed in the standard Kalman Filtering algorithm do not lead to significantly better results than using equal weights: in Experiments I and II, Blind Averaging does better than Standard KF, while in Experiment III their results are comparable. The set of weights employed in the cross-correlation-based algorithm, on the other hand, consistently produced improved performance both in terms of overall accuracy and in terms of the incremental increase in accuracy as new sensory data is incorporated. The final error obtained by the most accurate of the other two multi-frame algorithms is 2-3 times the final error in the 3D model acquired by the CC-based algorithm: the CC-based algorithm consistently outperforms the standard Kalman Filtering algorithm which only uses the diagonal band of the covariance matrix. Since the cross-correlations record the error in the 3D model arising from incorrectly estimated camera motion, the main conclusion to be drawn from these experiments is that taking this error into account is critical.

The last two experiments indicate that the 3D model acquired by the CC-based algorithm can be successfully applied in a practical task of determining the robot's location in its environment. As the results of experiments IV and V show, the robot's position all along its path was estimated quite accurately. The practical applicability of the algorithm will be further considered in Chapter 5 with respect to computational complexity.

CHAPTER 5

COMPUTATIONAL ISSUES

5.1 Introduction

In Chapters 3 and 4, the role of cross-correlations has been illustrated both theoretically and experimentally. In particular, the superiority of the cross-correlation-based algorithm over a standard Kalman Filtering algorithm has been established in the experiments reported in Chapter 4. However, in using the full covariance matrix, the CC-based algorithm deals with significantly more information than the standard KF algorithm, which uses only a subset of the covariance matrix. Therefore an expected practical consequence of using cross-correlations to improve accuracy is an increase in computational expense.

In this chapter the components of the CC-based algorithm will be analyzed in terms of their required run time, and methods for reducing computational complexity will be considered, along with preliminary experiments. The full running time has been recorded for a LISP implementation, while a partial record has been obtained for the ongoing C implementation (from which the total running time will be estimated). As might be expected, dropping cross-correlations to reduce running time has an adverse effect on accuracy. An alternative approach to reducing running time without compromising accuracy would be to divide the model into subparts and apply the CC-based algorithm on each part.

5.2 Time Complexity of the Components

A complexity analysis of the main components of the CC-based algorithm will be provided in this section. Such an analysis predicts theoretically the relationship between the expected running time of each component as a function of the number of points in the 3D model. Following the complexity analysis, the actual running times are reported for a particular experiment (consisting of 22 points) using the current implementation of the algorithm in LISP on a TI Explorer;¹ the times conform to the predictions made by the complexity analysis.

Figure 3.1 depicts the various components of the CC-based algorithm and the steps of the algorithm are given in Section 3.1.1. At each iteration a two-frame model of the environment is reconstructed (Step 2 in Section 3.1.1), the full 3D error (direct plus indirect error) of the model is computed (Step 3), and the two-frame model is fused with the previous multi-frame 3D model (transformed to the current camera position; Steps 4 and 5). Let us now turn to the computational complexity analysis of each of these main components.

5.2.1 Theoretical Complexity

For a 3D model consisting of n points, the two-frame algorithm is of $O(cn)$ time complexity, where c is the number of iterations that the two-frame algorithm requires to converge. The linear term n arises from the fact that the 2D (image) coordinates of each point independently contribute to calculating both the camera motion (cf. Equation 3.10) and the 3D location of the point.

Computing the full 3D error of the two-frame model - i.e., the full covariance matrix - is of complexity $O(n^3)$, reflecting the number of elements in the covariance

¹A faster C implementation is underway; initial tests indicate that an order of magnitude speedup can be obtained in C over LISP for at least a major time consuming part of the algorithm. See Section 5.3.1 for details.

matrix, $O(n^2)$, times the computation involved in each element of the matrix, $O(n)$. Every element of the matrix is again calculated based on the 2D image coordinates of all points; if any image coordinate changes, this affects the camera motion, which in turn affects every element of the matrix. The $O(n^3)$ time complexity is directly derivable from the matrix formulation. The full covariance matrix involves computing $(\frac{dW}{dI} + \frac{dW}{dM} \frac{dM}{dI})(\frac{dW}{dI} + \frac{dW}{dM} \frac{dM}{dI})^T$ (cf. Equations 3.51–3.52). The term $(\frac{dW}{dI} + \frac{dW}{dM} \frac{dM}{dI})$ is of size $(3n \times 4n)$; multiplication of this term and its transpose is of time complexity $O(n^3)$.

In order for fusion to take place, the previously obtained multi-frame model and its error estimate are transformed to the current camera position. Transforming the previous model is of $O(n)$ complexity, since each point is independently rotated and translated to the new coordinate system. Transforming the covariance matrix is of $O(n^2)$ complexity, because a finite operation (cf. Equation 3.57) is performed on every one of the n^2 elements of the matrix. Finally, the fusion component is of complexity $O(n^3)$. Recall that fusion in Kalman Filtering involves weighting each 3D model that is fused by the inverse of its covariance matrix (cf. Equations 3.58–3.59). Computing the inverse of a matrix using standard methods [75] is of complexity $O(n^3)$.

Thus, apart from the unpredictable convergence time of the two-frame algorithm (the term c above), the complexity analysis predicts that determining the covariance matrix and fusing the models take the longest time to run, while the transformation component should be faster.

5.2.2 Actual Running Times

The LISP implementation allowed rapid development and modifications while the algorithm was being explored, and it now is being ported to C for an efficient run-time implementation. Let us consider the actual running time of each of these components using a LISP version of the CC-based algorithm running on a TI Explorer. As a test

Table 5.1: Running time for the four main components of the CC-based algorithm in the case of the Rocket-Field Sequence. The theoretical time complexity is indicated for convenience. The last column is the sum of the times for the four components plus a small overhead. The high value of 3D Error time for frames 10-11 is clearly an outlier, and possibly due to an unusual overhead from memory paging.

Frame No.	Two-Frame	3D Error	Transformation	Fusion	Total
	$O(cn)$	$O(n^3)$	$O(n^2)$	$O(n^3)$	$O(n^3 + cn)$
	sec	sec	sec	sec	sec
1-2	5.6	34.1	3.6	39.4	84.6
2-3	7.5	35.8	5.4	40.0	90.2
3-4	7.6	37.9	6.5	39.1	92.6
4-5	11.7	36.6	4.0	40.2	94.0
5-6	7.3	36.3	4.9	41.2	91.5
6-7	6.8	37.0	7.4	40.9	95.3
7-8	29.3	39.9	4.3	41.8	119.1
8-9	12.0	30.5	4.3	36.7	87.0
9-10	5.3	28.8	3.8	37.3	76.4
10-11	3.1	68.6	4.3	38.5	120.3
AVG	9.6	38.6	4.8	39.5	95.1

case the Rocket-Field Sequence (Experiment III in Chapter 4) was employed. Recall that the experiment consists of 22 tracked points across 11 images. The running times were recorded for each of the 10 iterations and are shown in Table 5.1.

In accordance with the complexity analysis, the average time required to compute the 3D error is comparable to the average time taken by the fusion component (38.6 and 39.5 seconds respectively); both of these components are theoretically $O(n^3)$ time complexity. For this case involving 22 points, the transformation time ($O(n^2)$) corresponds to an average time of only 4.8 seconds. The large fluctuations in the two-frame component (3.1 seconds - 29.3 seconds) are a result of the variability in

the search space involved in solving for the interframe camera motion, corresponding to the unpredictable term c discussed earlier (cf. Section 5.2.1).

In this section we have considered the computational complexity of the main components of the CC-based algorithm both theoretically and in terms of a specific implementation. The total time for one iteration of the CC-based algorithm is 95.1 seconds in this LISP implementation. Although a much faster implementation in C on a Silicon Graphics machine is underway (between one and two orders of magnitude speedup) it is worth considering whether the running time per iteration could be reduced without sacrificing accuracy; this is the topic of the following sections.

5.3 Reducing Running Time

The two most time-consuming components of the CC-based algorithm are the estimation of the 3D error covariance and the fusion (Kalman Filtering) module, as was shown in Table 5.1. Recall that these two components are of time complexity $O(n^3)$. Such high complexity is a direct function of the number of cross-correlations used in the CC-based algorithm. Without cross-correlations (as in the standard KF algorithm), both components would be of time complexity $O(n)$. Since the majority of the terms in the covariance matrix representing the 3D error are cross-correlations, ignoring cross-correlations would directly reduce the time to compute the elements of the covariance matrix. Similarly, dropping cross-correlations may speed up fusion; this is because most of the time taken by the fusion component involves inverting covariance matrices, and matrix inversion is typically faster with sparser matrices.

In this section we will consider the effect of dropping cross-correlations on the running time of the algorithm and on the accuracy of the 3D models. The results discussed here indicate that reducing the number of cross-correlations to a point

where a significant speedup is achieved has an adverse effect on the performance of the algorithm.

5.3.1 Effect of Reducing Cross-correlations on Running Time

In order to study the effect of dropping cross-correlations on running time, we consider one of the two most time-consuming components of the CC-based algorithm, the fusion component. As shown in Table 5.1 fusion takes an average of 39.5 seconds/iteration. However, it turns out that most of this time is taken up by matrix inversion. In fact, under the current LISP implementation it takes an average of 12.7 seconds to invert the covariance matrix for the Rocket-Field Sequence; 3 such inversions are required for each instance of fusion (cf. Equations 3.58-3.59), therefore approximately 96% of the 39.5 seconds required by the fusion component. Since matrix inversion takes up almost all the time of the fusion component, we will concentrate on the time taken for matrix inversion when cross-correlations are dropped. In this section we report an experiment on matrix inversion times which is implemented in C on a Silicon Graphics machine (SGI).² Although running the experiment with a LISP implementation would enable a direct comparison with Table 5.1, the C implementation was chosen because it provides a first approximation to the computational performance of the CC-based algorithm on a fast machine such as the SGI. The results will be extrapolated to the total running time of the algorithm.

Although matrix inversion time may be reduced by dropping cross-correlations, for any particular matrix inversion algorithm a reduction of time can only be achieved if cross-correlations are omitted systematically such that the matrix can be partitioned in a meaningful way. Partitioning the matrix into submatrices makes it possible to determine the inverse of the matrix by combining the inverses of smaller

²The specifications of this particular SGI are: Silicon Graphics Power Series 340 GTX with four R 3000, 25 MIPS processors.

submatrices. If any submatrix is empty or only filled along the diagonal, it directly reduces the computational time of combining the submatrices [75]. However, unless the dropped cross-correlations give rise to such a submatrix, omitting cross-correlations does not – at least in any obvious way – result in reduced computational time.

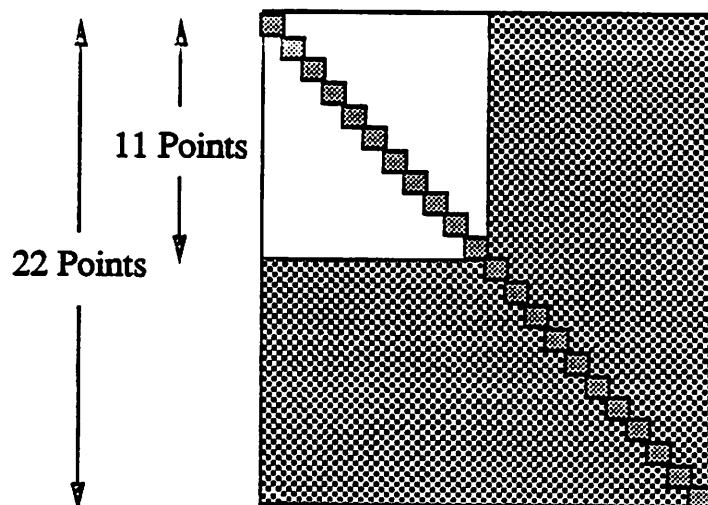
One systematic way to drop cross-correlations in the CC-based algorithm is shown in Figure 5.1, for 22 3D points (a 66×66 matrix). In Figure 5.1(a) the small squares along the diagonal represent the 3×3 matrices of the 22 points; these are the only terms used in standard Kalman Filtering. The white area represents the dropped cross-correlations corresponding to 11 of the 22 points; the remaining cross-correlations are depicted by the dark area. Henceforth, we will refer to the proportion of rows/columns³ with dropped cross-correlations to the total number of rows/columns in the covariance matrix as the *degree of dropped cross-correlations*. For example, in Figure 5.1(a) the degree of dropped cross-correlations is 50%. Note that this is not equivalent to a situation where 50% of the total number of cross-correlation entries have been dropped. The degree of dropped cross-correlations is directly related to the number of points whose cross-correlations have been dropped; in Figure 5.1 (a) the mutual cross-correlations of 50% of the points (i.e. 11 out of 22 points) have been dropped.⁴

Figure 5.1(b) provides a partitioning of this matrix into four submatrices (P, Q, R, S). Equations 5.1–5.3 (adapted from [75]) show how the original covariance matrix (C) can be inverted in terms of these smaller submatrices.

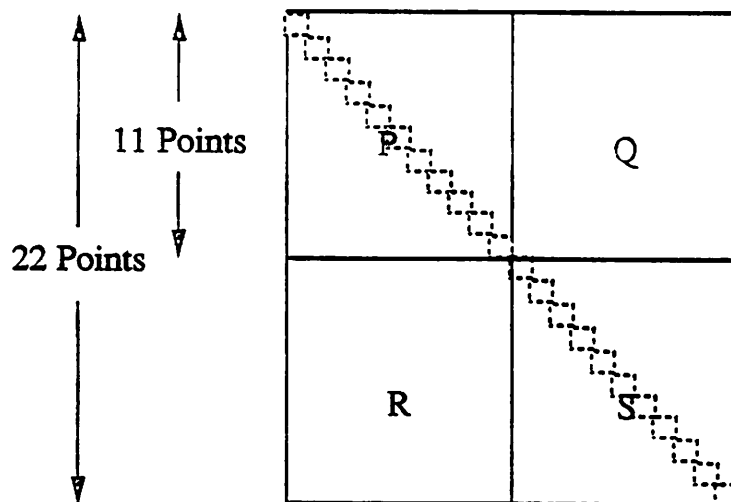
$$C = \begin{pmatrix} P & Q \\ R & S \end{pmatrix} \quad (5.1)$$

³Since the covariance matrix is by definition symmetric, the rows and columns are equivalent.

⁴A different way to drop cross-correlations would be to drop all cross-correlations between a subset of points and all the points in the covariance matrix. This turns out to be equivalent to dividing the 3D model into two separate parts, one part having no cross-correlations (as in the standard KF algorithm) and the other part maintaining a full covariance matrix (as in the CC-based algorithm). This possibility will not be pursued here.



(a) The dropped and used cross-correlations



(b) The four submatrices P, Q, R and S.

Figure 5.1: Partitioning the Covariance Matrix into submatrices for fast matrix inversion. In (a) the white area represents the dropped cross-correlations corresponding to 11 of the 22 points; the remaining cross-correlations are depicted by the dark area. (b) shows how the covariance matrix is partitioned into the submatrices P, Q, R, and S (cf. Equations 5.1–5.3).

$$C^{-1} = \begin{pmatrix} P' & Q' \\ R' & S' \end{pmatrix} \quad (5.2)$$

where

$$\begin{aligned} S' &= (S - RP^{-1}Q)^{-1} \\ R' &= -S'RP^{-1} \\ Q' &= -P^{-1}QS' \\ P' &= P^{-1} - Q'RP^{-1} \end{aligned} \quad (5.3)$$

Of the four submatrices, P is the only one that is inverted directly. Owing to its structure, inverting P is very straightforward and involves simply inverting each of the small submatrices along the diagonal of P . In this case, ignoring cross-correlations gives rise to a situation with a banded-diagonal matrix, the inverse of which is of time complexity $O(n)$; this reflects the fixed inversion time of each of the n 3×3 matrices along the diagonal band. The remaining calculations in Equation 5.3, which involve the three other submatrices in Figure 5.1(b), are all computationally less complex than inverting the original matrix.

It is relatively straightforward to determine the computational complexity of inverting the entire covariance matrix using Equation 5.3, for a sample case where the degree of dropped cross-correlations is 50% (as in Fig 5.1(a)). There are only two unique matrix inversions, P^{-1} and $(S - RP^{-1}Q)^{-1}$, both of which occur in the computation of S' . P^{-1} takes negligible time, with time complexity $O(n/2)$,⁵ since n has been reduced by 50%. The second inversion takes $O((n/2)^3)$, which is one eighth of the time required to invert the full covariance matrix $O(n^3)$.

In addition to the two inversions, Equation 5.3 involves matrix multiplication and summation. The computational complexity of matrix summation is $O(n/2)$, requiring

⁵Owing to a lack of better terminology we use $O(n/2)$ (and similar terms) to explain the achieved speedup, although *theoretically* $O(n/2)$ means the same as $O(n)$.

negligible time. In general, matrix multiplication is of time complexity ($O(n^3)$). Each of the operations in Equation 5.3 consists of multiplying three matrices, one of which is P^{-1} . Multiplying with P^{-1} is $O(n/2)$ complexity, since P^{-1} is a matrix with $n/2$ submatrices of size 3×3 along the diagonal. Finally, multiplying the remaining two matrices with each other (as in $RP^{-1}Q$) is of time complexity $O((n/2)^3)$. There are four such multiplications in Equation 5.3, resulting in a total time complexity for multiplication of $O(n^3/2)$. Combining the time complexity of the various operations in Equation 5.3 results in a total time complexity of $O(5n^3/8)$, for the case of 50% degree of dropped cross-correlations. This is a combination of the complexity of inversion, $O((n/2)^3)$, and of multiplication, $O(n^3/2)$.⁶ This complexity analysis reveals that when the degree of dropped cross-correlations is 50%, the computational time is expected to be reduced by 37.5% (i.e. $3/8$). This means that an entire iteration of the CC-based algorithm may be speed up by at most 10-25%.

Figure 5.2 plots the time (in seconds) taken to invert matrices of the type shown in Figure 5.1(a), using Equation 5.3; the average over 10 runs is reported. Matrices of four different sizes were tested: 30×30 , 48×48 , 66×66 and 84×84 . The inversion routine was written in C and run on the SGI. The X-axis encodes the degree of dropped cross-correlations. The 0% case corresponds to the CC-based algorithm and the 100% case to the standard KF algorithm.

As can be seen in Figure 5.2, a significant reduction in time for matrix inversion, is obtained only if the majority of the cross-correlations are dropped. For example, for the case of the 66×66 matrix, if the degree of dropped cross-correlations is 50% or less, this has little effect on computational time (for inversion). In fact, for the 50% case the reduction is smaller than what is predicted by the theoretical complexity analysis. Based on the 37.5% predicted speedup we would expect a decrease from

⁶Since $O(n^3 + n) = O(n^3)$ [75], the time taken by the $O(n)$ operations in Equation 5.3 does not affect the final computational complexity.

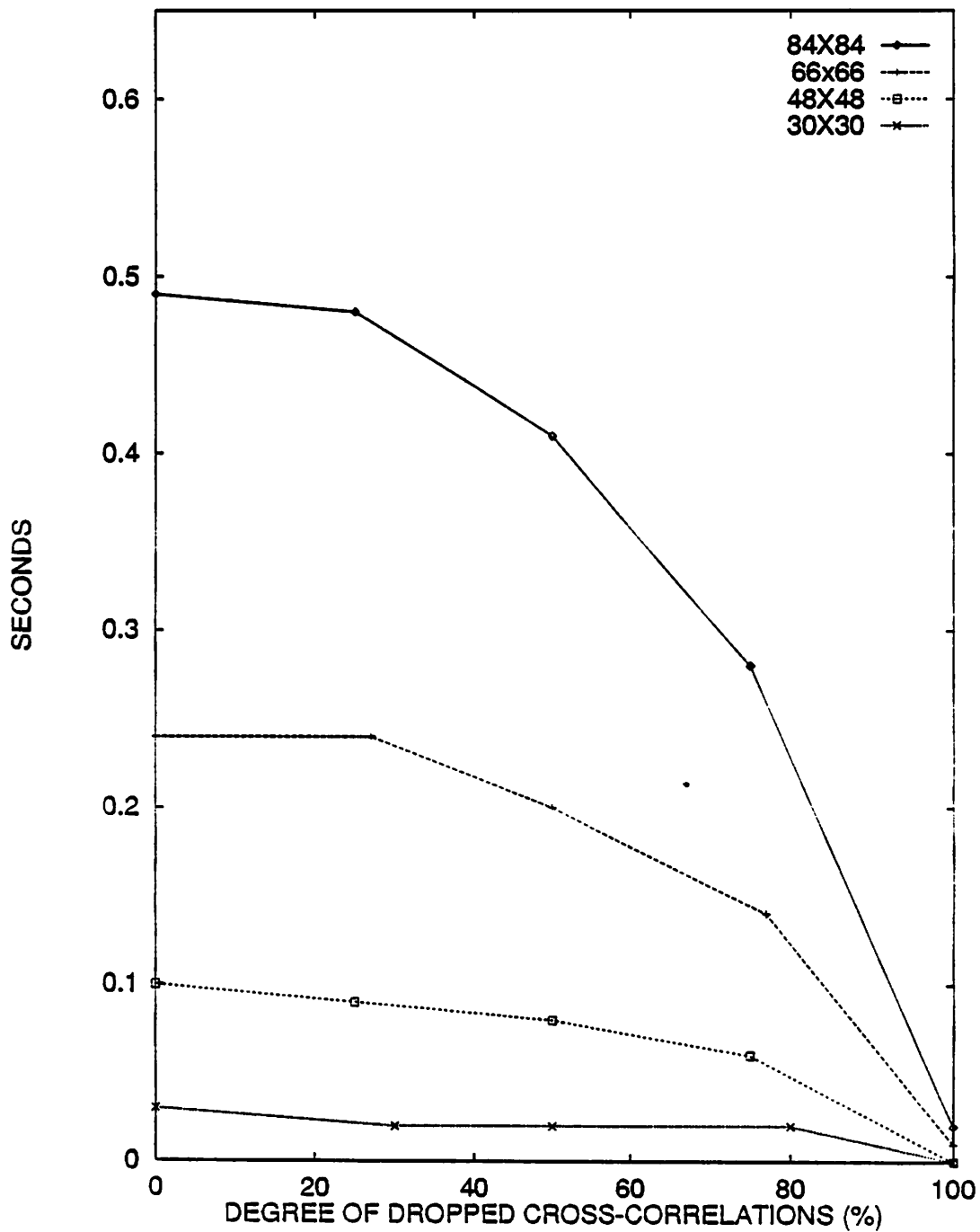


Figure 5.2: Running times for inverting covariance matrices of size 30×30 , 48×48 , 66×66 and 84×84 . The CC-based algorithm corresponds to 0% and the standard KF corresponds to 100%. The difference in running time for the 100% is due to the difference in the size of the matrices.

0.24 seconds to 0.15 seconds rather than 0.2 seconds. This points to overhead – such as time for storage operations (e.g. memory paging in the case of these large matrices) – which is not part of the theoretical analysis. The inversion time is cut in half only when the degree of dropped cross-correlations is over 80%. Thus, to obtain a significant reduction in computational time for the fusion component, the majority of the cross-correlations would have to be dropped. However, in the following section it will be shown that in such a case the 3D models obtained are highly inaccurate.

Before turning to the issue of accuracy, let us consider the question of total running time for one iteration of the CC-based algorithm on the faster SGI machine. Recall that inverting a 66×66 matrix (22 points in the Rocket-Field sequence) takes 12.7 seconds in LISP on the TI Explorer, and the total running time of 95.1 seconds (cf. Table 5.1) is 7.5 times the inversion time. Figure 5.2 shows that inverting a 66×66 matrix takes 0.24 seconds in a C implementation on the SGI. Assuming a corresponding speedup for the whole algorithm, the total running time of the C implementation is predicted to be roughly 7.5×0.24 seconds, or 1.8 seconds (per iteration).

5.3.2 Effect of Reducing Cross-correlations on Accuracy

Using the Rocket-Field sequence (from Experiment III) as a test case we will consider the effect of varying the degree of dropped cross-correlations on the accuracy of the 3D models.⁷ In a pilot study we observed that dropping cross-correlations arbitrarily led to instability and highly fluctuating results. In this section preliminary results from dropping cross-correlations in more systematic ways will be reported. As in Experiment III, errors will be plotted as the mean error in the 3D location of the points, represented as a percent of the actual depth of the points. Based on

⁷The initial conditions were identical to Experiment III.

these results it can be concluded that speeding up the algorithm by dropping cross-correlations (at least in the ways considered here) has a serious detrimental effect on accuracy.

In the experiments reported on here, cross-correlations associated with the *lowest* values along the diagonal of the covariance matrix were dropped (details are provided below). In the first two experiments the cross-correlations between points were dropped (each point consisting of three coordinates, and the cross-correlations consisting of 3×3 submatrices). In the third experiment cross-correlations between coordinates were dropped; the three coordinates of a point were treated separately. In all the experiments the cross-correlations were dropped in a particular covariance matrix just before inverting that matrix.

5.3.2.1 Experiment A: Dropping Cross-correlations of Points

In this experiment, the degree of dropped cross-correlations was varied as shown in Figure 5.3, for the degrees 0%, 9%, 14%, and 23%; recall that 0% corresponds to the CC-based algorithm. In each case, the cross-correlations of the points with the lowest value for the Z coordinate along the diagonal were dropped, since the Z coordinate is typically the most erroneous coordinate in SFM.

Figure 5.3 shows that when the degree of dropped cross-correlations in this experiment was set at 9%, the 3D model is comparable to the CC-based algorithm (with no dropped cross-correlations) in terms of accuracy. However, for the 14% case the error begins to increase, and in the 23% case the algorithm appears to break down (we will return to the discussion of the 23% case shortly).

These results suggest that the degree of dropped cross-correlations should be lower than 23% for the Rocket-Field sequence, given the way in which the points (for which cross-correlations were dropped) were selected in this experiment. However, as was shown in Figure 5.2, there is no speedup in computational time for the 66×66

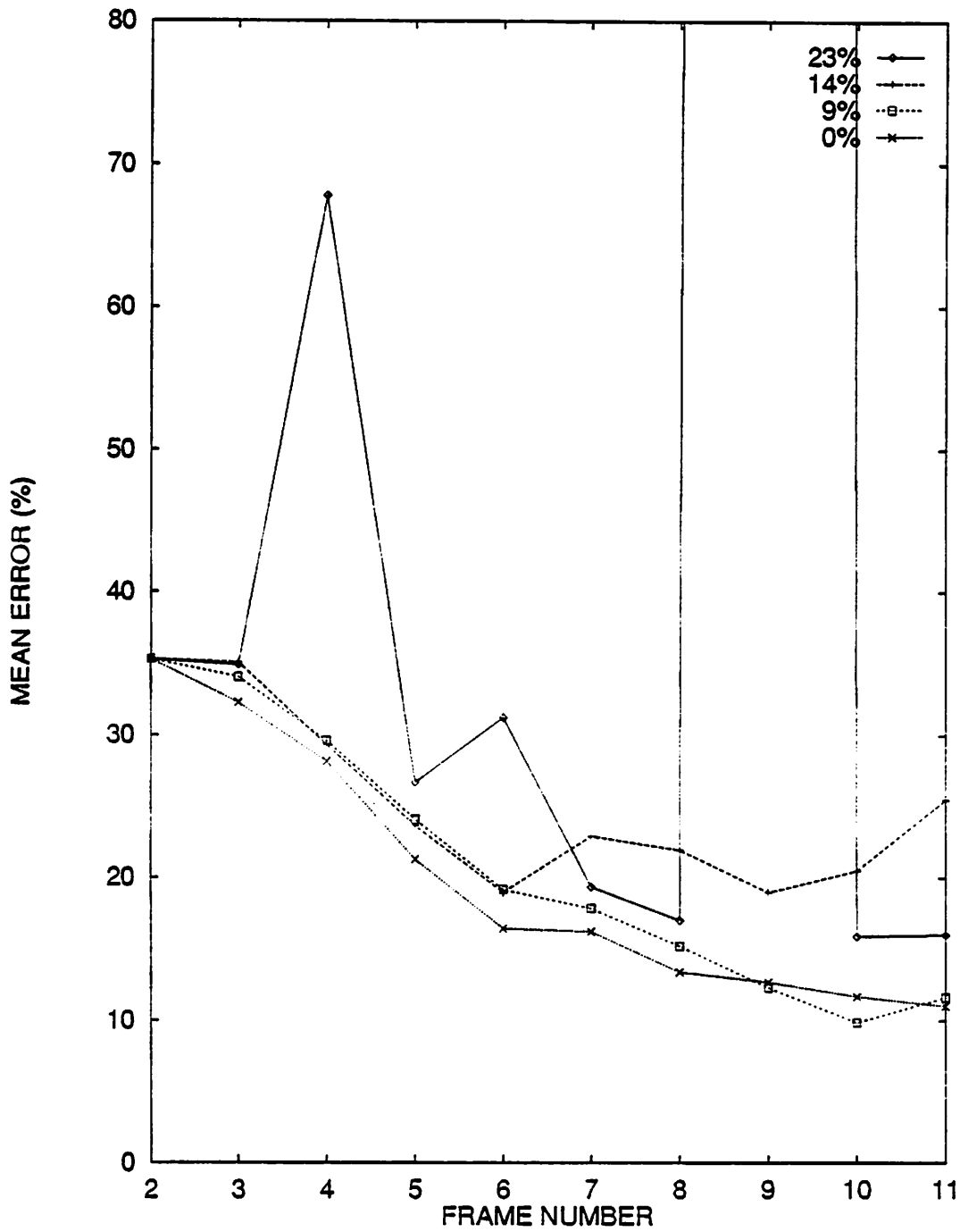


Figure 5.3: Experiment A: Effect of dropping cross-correlations of points on the accuracy of the 3D models. Each plot corresponds to a different value of the degree of dropped cross-correlations. The value of the plot for the 23% case at Frame 9 is 2288% (cf. Section 5.3.2.2 for discussion).

full covariance matrix (corresponding to the Rocket-Field sequence) if the degree of dropped cross-correlations is lower than 23%; in fact, there is no significant speedup until the degree of dropped cross-correlations reaches 80%.

5.3.2.2 Experiment B: Dropping Cross-correlations of Points in Multi-Frame Models

As was shown in Figure 5.3, the performance of the algorithm in the 23% case was highly volatile. Owing to the large amount of information involved in the covariance matrices, it is especially difficult to straightforwardly explain the source of the extremely high error in the 3D model of the 23% case at frame 9. The present experiment addresses the question of where such an error might have come from. In the process an alternative way of dropping cross-correlations is presented.

Recall from Figure 4.8 that the Two-Frame error at frame 9 of the Rocket-Field sequence was very high (77.3%). This frame corresponds to the highest error in the 23% case (Figure 5.3). Furthermore, the two remaining spikes of the Two-Frame error plot (Figure 4.8) also correlate with the only other spikes of the error plot for the 23% case. The behavior of the 23% case and its correlation with the two-frame results suggests that the error in the two-frame model may not have been properly accounted for once the relatively small subset of cross-correlations were dropped.

This idea was tested by *not* dropping cross-correlations from the covariance matrix associated with the two-frame model. However, cross-correlations from the covariance matrix of the previous multi-frame model were dropped, as in Experiment A. Recall that at the fusion step, the current two-frame model is fused with the previous multi-frame model, each being weighted by the inverse of its covariance matrix. Apart from keeping the cross-correlations of the two-frame models, the setup of this experiment and Experiment A were identical.

Figure 5.4 plots the results for the 23% case and the 27% case of dropped cross-correlations. The 23% case is well-behaved, unlike in Experiment A, corroborating the idea that the very high error in Figure 5.3 is due to incorrectly modeled error in the two-frame model of frame 9. Since the only difference between the 23% case in Figures 5.3 and 5.4 is the fact that cross-correlations in the two-frame model were dropped in Figure 5.3, but not in Figure 5.4, the fluctuating nature of the 23% case in Figure 5.3 must derive from this difference. In such a case, the inverse of the covariance matrix of the two-frame model at frame 9 - after cross-correlations have been dropped - has the effect of incorrectly weighting the points in the two-frame model, resulting in the grossly incorrect value for the 23% case in Figure 5.3.

The idea of dropping cross-correlations only from the multi-frame 3D model was tested for the next possible case: a degree of cross-correlations of 27% (or dropping cross-correlations for 6 of the 22 points, as opposed to 5 points in the 23% case). Figure 5.4 shows that the algorithm begins to degenerate at this point. For degrees higher than 27%, the results again fluctuate randomly (not shown). Thus, although maintaining the cross-correlations for the two-frame model results in somewhat better performance, this approach does not appear to be promising in terms of reducing computational time. However, a variant of this approach will be pursued in the following experiment.

5.3.2.3 Experiment C: Dropping Cross-correlations of Coordinates

In this experiment, instead of treating 3D points as single entities and dropping cross-correlations between points, coordinates of points were treated as individual entities. As in Experiment B, cross-correlations were dropped only for the previous updated model, and retained for the two-frame model. The cross-correlations associated with the coordinates with the lowest variance (i.e. value along the diagonal) were dropped.

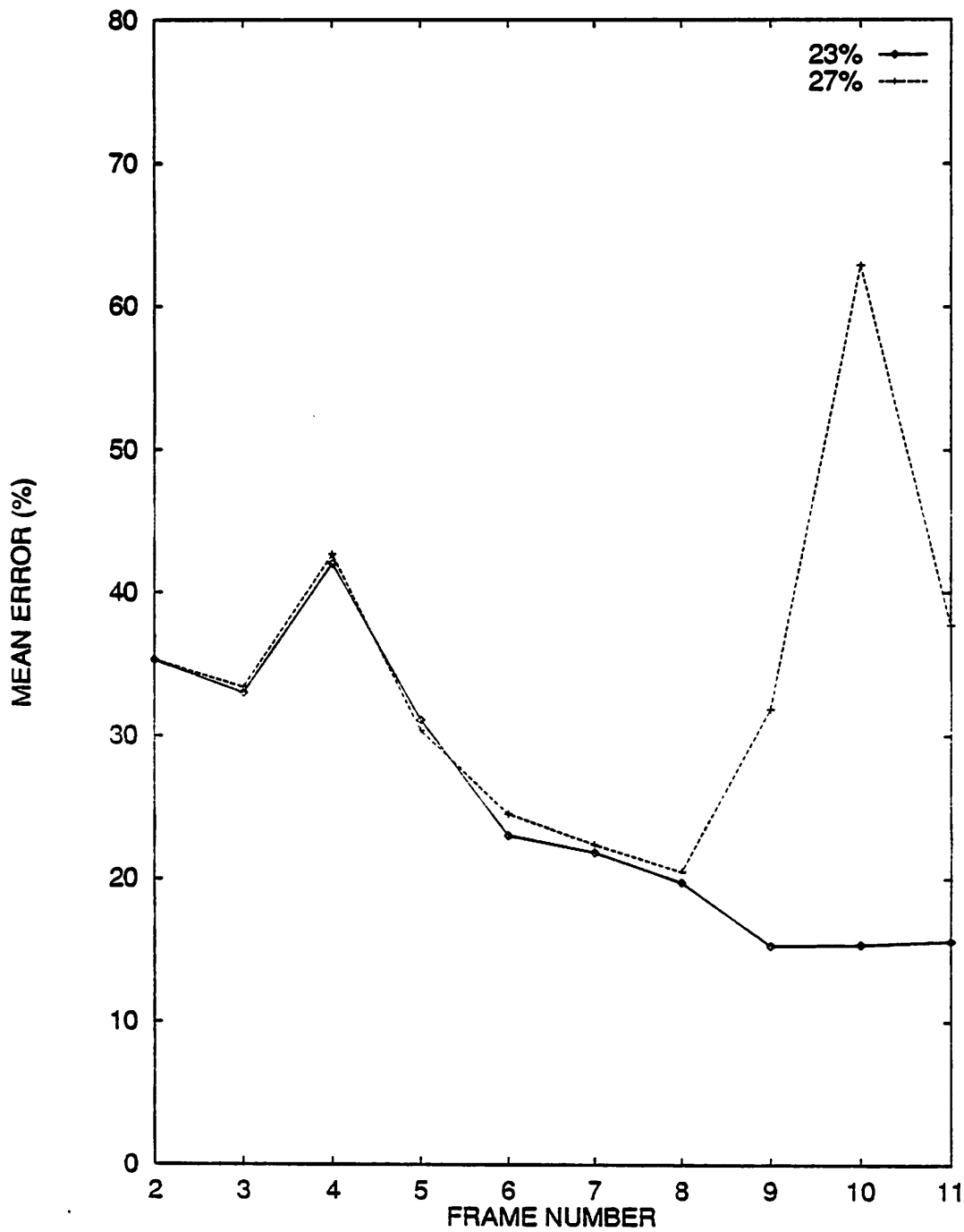


Figure 5.4: Experiment B: Effect of dropping cross-correlations of points on the accuracy of the 3D models. Each plot corresponds to a different value of the degree of dropped cross-correlations.

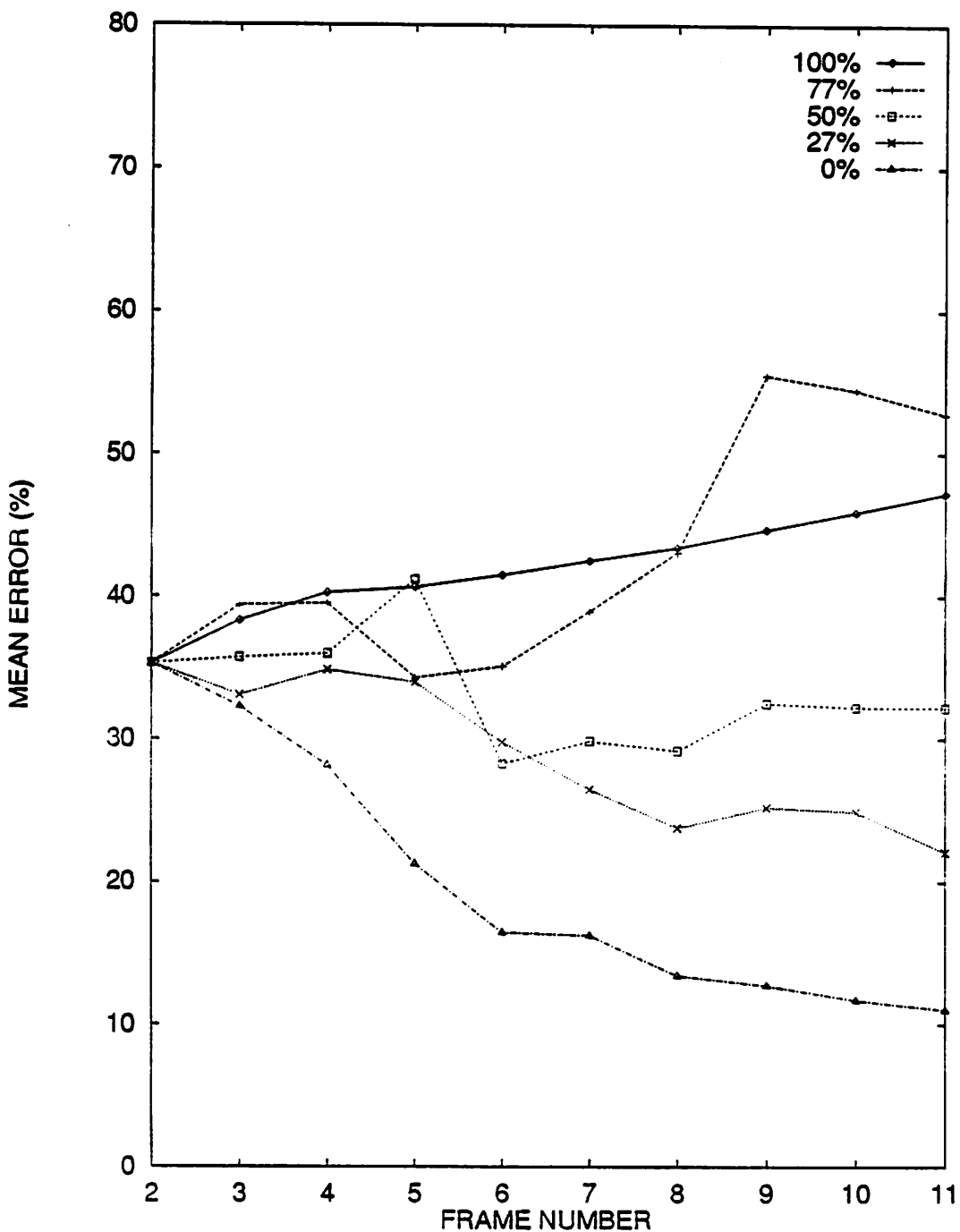


Figure 5.5: Experiment C: Effect of dropping cross-correlations of coordinates on the accuracy of the 3D models. Each plot corresponds to a different value of the degree of dropped cross-correlations.

Figure 5.5 plots the results for the cases where the degree of dropped cross-correlations is 100%, 77%, 50%, 27% and 0%.⁸ Although these results are fairly stable, only the 27% and 50% cases have a decreasing trend. However, the final error of the 27% case is twice the final error of the CC-based algorithm.

In Figure 5.5 the error seems to have a gradually increasing trend for the 100% case; in contrast the standard KF error (Experiment III, Figure 4.8) slowly decreases. Both the 100% case and standard KF use only the 3×3 matrices along the diagonal of the covariance matrix for the previous multi-frame model; however, the 100% case uses the full covariance matrix for the two-frame models, whereas standard KF uses only the elements along the diagonal band for the two-frame models. Since the 100% case and standard KF form a minimal pair their difference in output behavior can be attributed to the single difference in their operation, which will be the subject of future research.

A significant outcome of these test cases is that even systematically chosen subsets of cross-correlations – based on the values along the diagonal of the covariance matrix – cannot be arbitrarily emphasized over other subsets in accounting for the motion error. In order to reduce running time by dropping cross-correlations, a firmer theoretical basis would need to be developed for determining which cross-correlations can be ignored.

5.4 Reducing the Size of the 3D Model

The results of the previous section suggest that attempting to reduce computational complexity by ignoring cross-correlations may not be a promising approach. An alternative way to speed up the algorithm is to divide the entire 3D model into

⁸Again, the 0% case corresponds to the CC-based algorithm. However, the 100% case does not correspond to standard KF, since the full covariance matrix of the two-frame models is used.

subsets of points corresponding to different parts of the model, and apply the CC-based algorithm on each individual part (using all the cross-correlations within the subset). However, using subsets implies using fewer points, with an effect on the accuracy of the 3D model. In order to study this effect, varying sizes of 3D models will be compared in terms of accuracy. The Lobby Sequence from Experiment II (Chapter 4), involving 10 images and 29 tracked points, was used as a test case. The input in the present study was identical to that of Experiment II.

In the experiment reported here, we will see that a subset of 11 points performs well, and that having more than 11 points in the model does not considerably improve the accuracy of the CC-based algorithm. Thus, reducing the number of points in the model may turn out to be a promising way to reduce computational time, if these results generalize. For example, a set of 28 points can be divided into two subsets of 14 points, where each subset contributes half of the full 3D model. Recall that the time per iteration is 7.5 times the matrix inversion time. Given Figure 5.2, this means that a full iteration for a 28 point case (84×84 matrix) takes 3.6 seconds per iteration (on the SGI). In contrast, building two 14 point models takes a total of under 1.45 seconds per iteration (i.e. less than the time taken by the 48×48 matrix shown in Figure 5.2), which is a significant speedup over using the full matrix.⁹

In the actual experiment, the first seven points (cf. Table 4.6) were selected from the 29 points of the Lobby Sequence to constitute the first two-frame model. The CC-based algorithm was then applied to these seven points across the 10-image sequence. This involved computing the full covariance matrix for the selected subset at each iteration. As shown in Figure 5.6, the performance of the CC-based algorithm in the 7-point case fluctuates and has a relatively high final error of 16%.

⁹Since the CC-based algorithm has overall time complexity $O(n^3)$, reducing n by 50% is predicted to lead to an eightfold speedup. Since we have 2 sets of size $n/2$, we build two models each with a speedup of eightfold, but on a total giving only a four-fold speedup.

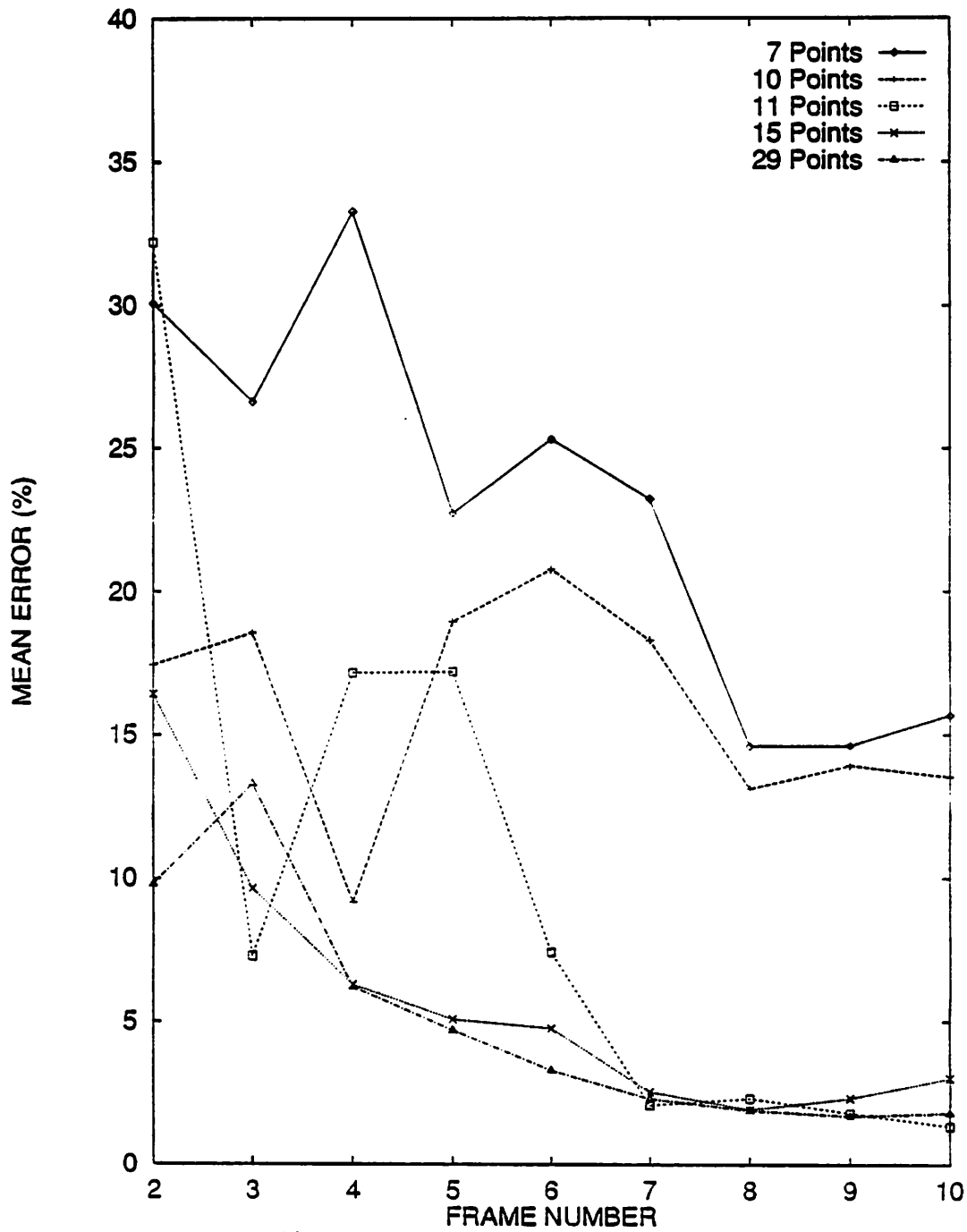


Figure 5.6: Effect of Reducing the number of 3D points on the accuracy of the resulting 3D models. The values of error are listed in Table 5.2.

Table 5.2: Effect of Reducing the number of 3D points on the accuracy of the resulting 3D model: mean percentage error for 3D models. This table lists the values used to plot Figure 5.6.

Frame No.	7 Points %	10 Points %	11 Points %	15 Points %	29 Points %
2	30.0	17.4	32.2	16.4	9.8
3	26.6	18.6	7.3	9.7	13.3
4	33.3	9.2	17.2	6.3	6.2
5	22.8	19.0	17.2	5.1	4.7
6	25.3	20.8	7.4	4.8	3.3
7	23.2	18.3	2.1	2.6	2.3
8	14.6	13.1	2.3	1.9	1.9
9	14.6	13.9	1.8	2.3	1.7
10	15.7	13.5	1.3	3.0	1.8

Subsequently, the size of the subset was increased to locate the point at which the performance of the algorithm improves. As Figure 5.6 (and Table 5.2) shows, there is a dramatic difference between using 10 points as opposed to using 11 points. Note that points 10 and 11 are corners of the same poster on the left wall (cf. Figure 4.4 and Table 4.7 for the ground truth), and are not geometrically distinguished. Hence, the breakpoint in the behaviour cannot be explained by the inclusion of some type of high leverage point. On the other hand, the fact that this breakpoint occurs points to a possibly deeper phenomenon; studying this behaviour will be the subject of future research.

Using 10 or fewer points results in fluctuating performance and high final error, while using 11 (or more) points results in high accuracy from frame 7 on. In fact, the final accuracy of the 11-point case is comparable to using all 29 tracked points. Of the four subsets considered, the 15-point case performs best overall. The results of an intermediate case (22 points) between 15 points and the maximum possible 29

points was found to be similar to the 15 and 29-point cases (results not shown since it would just clutter the graph). Using 15 points – with a full covariance matrix of size 45×45 – requires under 0.1 seconds for matrix inversion (on the SGI) as shown in Figure 5.2. This corresponds to a running time of 0.75 seconds/iteration and to a total time of 1.5 seconds/iteration for two 15-point models. In contrast, building the 29-model – without dividing it into subparts – would take over 3.6 seconds/iteration; hence dividing the model, in this case provides a 2.4-fold speedup.

In the experiment considered here, using 11–15 points leads to accurate results, while resulting in considerable speedup over constructing a single 3D model consisting of 29 points. However, it is not claimed that the optimal size of subsets in general is 11–15 points; determining the optimal size based on more experimental data over a wide variety of experiments will be left as a subject of future research. In general, dividing a set of n points into m subsets of equal size is expected to result in an m^2 -fold speedup, since the time complexity of the CC-based algorithm is $O(n^3)$. Constructing m models, each of size n/m , is of time complexity $m O(n^3/m^3)$, i.e. $\frac{1}{m^2}O(n^3)$.

5.5 Conclusion

The results of the experiments reported in this chapter show that determining the elements of the covariance matrix and fusing the 3D models take the longest time in the execution of the CC-based algorithm. A major time-consuming component is matrix inversion. However, matrix inversion can be speeded up if large amounts of information contained in the covariance matrix is ignored. On the other hand, drop-

ping even systematically selected clusters of the covariance matrix leads to unreliable results. A promising alternative approach is to use the full covariance matrix, but reduce the number of points in the model by dividing the original model into subsets of points.

CHAPTER 6

CONCLUSION

6.1 Contributions of the Dissertation

Constructing a 3D model of the environment using images obtained by a moving camera was discussed in Chapters 1 and 2. Various Two-Frame Structure from Motion (SFM) algorithms were analyzed, and different contributing components of error arising from using two images for reconstruction were studied. Incorrectly estimated camera motion was identified as the major source of error: even small errors in camera motion may give rise to large errors in the 3D model. In Two-Frame SFM such errors are responsible for apparently random and unpredictable depth recovery. In order to improve the performance of SFM algorithms, the use of more information from image sequences was pursued. Previous multi-frame algorithms were analyzed and compared in Chapter 2. An incremental approach was chosen over batch techniques since processing of information online, as images are obtained, is useful for real-world applications such as robot navigation and obstacle avoidance.

Incremental Multi-Frame SFM involves weighting information from different images according to their reliability, followed by combining the weighted information. However, for incremental MFSFM to be effective, the estimated weights for each image in the sequence should reflect the actual underlying error in the image. The MFSFM algorithm developed in this dissertation weights the individual 3D models (constructed every time a new image is obtained) by the inverse of their error covariance. The error covariance used here captures the expected error in the 3D model.

The main contribution of this dissertation is the identification and estimation of a previously neglected major source of error – the error in the camera motion. This motion error manifests itself in cross-correlations of errors between points in the 3D model. For example, an error in estimating the movement of the camera in a particular direction shifts the entire 3D model along that direction. In Chapter 3 the details of estimating the motion error and other less important types of error were provided, along with the steps of the cross-correlation-based (CC-based) algorithm. Furthermore, the exact role of the cross-correlations – how they capture and correct for the motion error – was illustrated theoretically for a simplified 3D model consisting of two points.

In Chapter 4, the accuracy of the CC-based algorithm was addressed by comparing the effectiveness of this novel approach against the standard approaches in the field. The performance of the CC-based algorithm was compared in realistic scenarios against Horn's Two-Frame Algorithm (which provides the input to the CC-based algorithm), against a blind averaging algorithm, and against a standard Kalman Filtering algorithm. The standard KF algorithm is identical to the CC-based algorithm apart from the fact that it ignores the cross-correlations between 3D points in the covariance matrix; in other words, the effect of the motion error is neglected. In the three experimental scenarios considered (involving a robot workcell sequence, an indoor lobby sequence, and an outdoor rocket-field sequence), the CC-based algorithm consistently outperformed the standard KF algorithm, as well as the other two algorithms. The results also corroborated the expected random fluctuations in the results of Two-Frame SFM. Two further experimental scenarios were presented wherein the 3D model acquired by the CC-based algorithm was effectively applied to the task of determining the position of the robot.

Since the CC-based algorithm deals with significantly more information associated with the cross-correlation errors than the standard KF algorithm, it obviously requires more computation. The current implementation in LISP (running on the TI Explorer) takes an average of 95 seconds per iteration for a model consisting of 22 points; the predicted running time for a C implementation on a Silicon Graphics machine (SGI) is 1.8 seconds/iteration. Matrix inversion is a major bottleneck in the computation, taking up 40% of the time per iteration. However, ignoring subsets of cross-correlations in order to speed up matrix inversion results in poor performance, as was shown in Chapter 5. For a model consisting of 22 points, ignoring cross-correlations related to more than 6 points resulted in unstable performance, whereas cross-correlations related to at least 18 points would have to be ignored in order to speed up the computation time per iteration by a third. On the other hand, the results from a further experiment reported in Chapter 5 suggest that computing the model for smaller sets of points – without ignoring any cross-correlations – is a more promising approach to reducing computational time, while preserving accuracy. In this experiment we found that using a subset of 15 points may give rise to accurate 3D models comparable to using a larger set (even about twice its size).

6.2 Future Research Directions

One important goal of future research is to obtain a deeper conceptual understanding of how the CC-based algorithm works. The sensitivity of the performance of the CC-based algorithm to the values computed by the individual equations (in Chapter 3) is yet to be completely understood. Apart from a theoretical study, conducting controlled experiments with various types and amounts of error in the estimated camera motion should be revealing.

Given an understanding of the distribution of information across the cross-correlations, it might be possible to ignore selective subsets of the covariance matrix for the purposes of speeding up the algorithm. More dramatic than reducing matrix inversion on a serial machine, a parallel implementation of the CC-based algorithm could provide significant speedup, given that most of the computation in the CC-based algorithm can be conducted in parallel.

The CC-based algorithm uses the output of Horn's two-frame algorithm as its input. However, the framework outlined in Chapter 3 is not restricted to any particular two-frame algorithm. A possible variant of the present algorithm involves replacing Horn's algorithm with another two-frame algorithm such as Adiv's [1]. Since Adiv's algorithm also uses a version of the coplanarity constraint, the replacement should be straightforward. Nevertheless, the theoretical ramifications arising from certain assumptions; in particular, the small rotation assumption used by Adiv must be examined.

A further theoretical issue is the representation of scale of the 3D model, which is problematic due to the scale ambiguity of SFM. In general, apart from the discussion in Cui et. al. [21], the question of scaling in SFM has been ignored in previous literature. Rather than attempting to derive an optimal technique for setting the scale of a 3D model, SFM algorithms typically use the magnitude of camera movement made available by odometers or similar sensors. This method is simplistic and works well if the camera movement is estimated accurately; this method was used in Experiment III (via the INS measurement of the magnitude of vehicle translation). Another possibility is to use the known distance between two points to scale the 3D model. This technique is problematic if these two points in the 3D model turn out to be incorrect. An improvement of this technique is to use the average distance between the centroid of the model and every point in the model. Intuitively, this average, or

scale S , represents the spread of the model around its center. Note that the value of S is a constant for a fixed set of points in a rigid environment. Although SFM algorithms are unable to recover the absolute value of S , they can ensure that this scale is at least maintained as a constant; such a method was used in Experiment I and II. However, none of these techniques address the question of error in scaling. Modeling the error introduced by incorrect scaling would result in an important extension of the CC-based algorithm.

Currently, the CC-based algorithm does not deal with the problem of how to modify the covariance matrices to deal with appearing or disappearing points. The ramifications of the most straightforward solution of adding or eliminating the corresponding rows and columns from the covariance matrix need to be considered in future work.

An improvement of the CC-based algorithm involving a significant amount of research would be modifying the algorithm to deal with features other than points, such as lines and surfaces. It would also be useful to consider the kind of information used by humans to navigate when there are few features in the environment (such as in a hallway). It is likely that humans fill in abstractions (e.g. surfaces) from representations of a few basic features in some systematic way, although it is yet unclear as to what the strategies of this system are. An extension of the CC-based algorithm from points to more abstract 3D features (such as lines and planes) would give rise to representations of the environment that could dramatically influence tasks in autonomous navigation, especially path planning and obstacle avoidance.

A possible alternative to using cross-correlations to capture the effect of camera motion error is to correct for the motion error before the 3D model is constructed. In such a case, a subset of tracked points would be used to calculate the motion and an estimate of its probable error. Using this estimate the calculated motion could be

refined and then applied to obtaining a dense 3D model, without needing to transfer the error information in large covariance matrices.

The idea of cross-correlations to capture error is not restricted to the domain of SFM. It is applicable in any domain where a single major source of error corrupts all portions of the final result in some systematic way. For example, in vision research based on projective geometry (such as [19]), incrementally building models of the world in a projective space should have a comparable counterpart to the motion error that corrupts models in 3D space; the error analysis of such work could be improved by incorporating cross-correlations.

A P P E N D I X A
KALMAN FILTERING TERMINOLOGY

The equations of the CC-based algorithm are provided here in terms of a traditional Kalman Filter (adapted from [90]); k denotes the k th iteration of the CC-based algorithm.

H	Multi-frame 3D Model
R	Camera Rotation
T	Camera Translation
M	Camera motion (R,T)
N	Noise due to coordinate transformation
$N(a, b)$	Random variable with mean a and covariance b
C^M	Covariance of noise in motion M
W	Two-frame 3D Model
v	Noise in Two-Frame model
C^W	Covariance of noise in Two-Frame model W
C	Covariance of noise in Multi-Frame Model W

System Model	$\mathbf{H}_k = (R_{k-1}\mathbf{H}_{k-1} - \mathbf{T}_{k-1}) + \mathbf{N}_{k-1}, \quad \mathbf{N}_k \sim N(0, C_k^M)$
Measurement Model	$\mathbf{W}_k = \mathbf{H}_k + \mathbf{v}_k, \quad \mathbf{v}_k \sim N(0, C_k^W)$
Initial Conditions	$E[\mathbf{H}_1] = \mathbf{W}_1, C_1 = C_1^W$
Other Assumptions	$E[d\mathbf{N}_j \mathbf{v}_k^T] = 0$ for all j, k
State Estimate Extrapolation	$\mathbf{H}_k(-) = R_{k-1}\mathbf{H}_{k-1} - \mathbf{T}_{k-1}$
Error Covariance Extrapolation	$C_k(-) = R_{k-1}C_{k-1}(\div)R_{k-1}^T + D_{k-1}C_{k-1}^M D_{k-1}^T$ where $D = \frac{d\mathbf{H}_k(-)}{d\mathbf{M}_{k-1}}$
State Estimate Update	$\mathbf{H}_k(+) = \mathbf{H}_k(-) + K_k[\mathbf{W}_k - \mathbf{H}_k(-)]$
Error Covariance Update	$C_k(+) = [I - K_k]C_k(-)$
Kalman Gain Matrix	$K_k = C_k(-)[C_k(-) + C_k^W]^{-1}$

BIBLIOGRAPHY

- [1] Adiv, G. "Generating three-dimensional motion and structure from optic flow generated by several moving objects," *IEEE Trans. PAMI*, 7(4), 1985, pp. 384-401.
- [2] Adiv, G. *Interpreting optical flow*, Ph.D. dissertation, Department of Computer Science, University of Massachusetts, Amherst, 1985.
- [3] Adiv, G. "Inherent ambiguities in recovering 3d information from a noisy flow field," *IEEE Trans. PAMI*, 11(5), 1989.
- [4] Aloimonos, Y. and Brown, C. "Direct processing of curvilinear motion from a sequence of perspective images," *Proc. Workshop on Computer Vision, Representation and Control*, Annapolis, Maryland, 1984.
- [5] Anandan, P. *Measuring Visual Motion from Image Sequences*, Ph.D. dissertation, Department of Computer Science, University of Massachusetts, Amherst, 1987.
- [6] Ando, H. "Dynamic Reconstruction of 3D Surfaces and 3D Motion," *Proc. IEEE workshop on visual motion*, Princeton, NJ, pp. 101-110, 1991.
- [7] Arnspang, J. "Optic Acceleration," *Proceedings 2nd IEEE International Conference on Computer Vision*, Tampa, Florida, Dec. 1988, pp. 364-373.
- [8] Balasubramanyam, P. Ph.D. dissertation, in progress, Department of Computer Science, University of Massachusetts.
- [9] Bar-Shalom, Y. and Fortmann, T.E. *Tracking and Data Association*, Academic Press, Orlando, Fl, 1991, pp. 277.
- [10] Beveridge, J.R., Weiss, R. and Riseman, E. "Optimization of 2-Dimensional Model Matching," *Proceedings IEEE International Conference on Pattern Recognition*, Atlantic City, N.J., June 1990.
- [11] Bharwani, S., Riseman, E. and Hanson, A. "Refinement of environment depth maps over multiple frames," *Proceedings of the Workshop on Motion: Representation and Analysis*, Charleston, S.C., May 1986.

- [12] Bolles, R.C. and Baker, H.H. "Epipolar-plane Image Analysis: a technique for analyzing motion sequences," *Proc. Workshop on Computer Vision: Representation and Control*, Bellaire, Michigan, 1985, pp. 168-178.
- [13] Broida, T.J. and Chellappa, R. "Performance bounds for estimating three-dimensional motion parameters from a sequence of noisy images," *Journal of the Optical Society of America A*, 1989, pp. 879-889.
- [14] Broida, T.J. and Chellappa, R. "Estimating the Kinematics and Structure of a Rigid Object from a Sequence of Monocular Images", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 6, pp. 497-513, 1991.
- [15] Bruss, A.R. and Horn, B.K.P. "Passive Navigation," *Computer Vision, Graphics and Image Processing*, Vol. 21, No.1, 1983, pp. 3-20.
- [16] Chandrashekar, S. and Chellappa, R. "Passive Navigation in a Partially Known Environment," *Proc. IEEE workshop on visual motion*, Princeton, NJ, pp. 2-7, 1991.
- [17] Chang, Y.L. and Aggarwal, J.K. "Reconstructing 3D Lines from a Sequence of 2D Projections: Representation and Estimation," *Proceedings 3rd IEEE International Conference on Computer Vision*, Osaka, Japan, 1990, pp. 101-105.
- [18] Chang, Y.L. and Aggarwal, J.K. "3D Structure Reconstruction from an Ego Motion Sequence Using Statistical Estimation and Detection Theory," *Proc. IEEE workshop on visual motion*, Princeton, NJ, pp. 268-273, 1991.
- [19] Collins, R.T. "Single Plane Model Extension using Projective Transformations," *Proc. Darpa I.U. Workshop*, San Diego, CA., Jan 1992, pp. 917-923.
- [20] Cramer, H. *Mathematical Methods of Statistics*. Princeton Univ. Princeton, New Jersey, 1946.
- [21] Cui, N., J. Weng and Cohen, P. "Extended Structure and Motion Analysis from Monocular Image Sequences," *Proceedings 3rd IEEE International Conference on Computer Vision*, Osaka, Japan, 1990, pp. 222-229.
- [22] Dutta, R. Ph.D. dissertation, in progress, Department of Computer Science, University of Massachusetts.
- [23] Dutta, R., Manmatha, R., Riseman, E. and Snyder, M. "Issues in extracting motion parameters and depth from approximate translational motion," *Proc. Darpa Image Understanding Workshop*, Volume 2, pp 945-960, April 1988.

- [24] Dutta, R., Manmatha, R., Williams, L.R. and Riseman, E.M. "A Data Set for Quantitative Motion Analysis," *CVPR*, San Diego, California, pp. 159-164, 1989.
- [25] Dutta, R. and Snyder, M. "Robustness of Correspondence-Based Structure from Motion," *Proceedings 3rd IEEE International Conference on Computer Vision*, Osaka, Japan, Dec. 1990.
- [26] Faugeras, O.D., Lustman, F. and Toscani, G. "Motion and Structure from Motion from Point and Line Matches," 1987, pp. 25-34.
- [27] Faugeras, O.D. and Maybank, S. "Motion from point matches: multiplicity of solutions," *Proc. IEEE Workshop on Motion*, Irvine, CA, March, 1989. pp. 248-255.
- [28] Fischler, M.A. and Bolles, R.C. "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography," *Communications of the the ACM*, Vol. 24, pp. 381-395, 1981.
- [29] Franzen, W.O. "Structure and Motion from Uniform 3-D Acceleration," *Proc. IEEE workshop on visual motion*, Princeton, NJ, pp. 14-20, 1991.
- [30] Franzen, W.O. *Structure from chronogenous motion*, Ph.D. dissertation, University of Southern California, May 1991.
- [31] Gibson, J.J. *The perception of the visual world*, Cambridge, Mass, Riverside, 1950.
- [32] Grady, D, "The Vision Thing: Mainly in the Brain," *Discover*, Volume 14, Number 6, pp. 57-66. June 1993.
- [33] Guissin, R. and Ullman, S. "Direct Computation of the Focus of Expansion from Velocity field measurements," *Proc. IEEE workshop on visual motion*, Princeton, NJ, 1991, pp. 146-155.
- [34] Hanna, K.J. "Direct Multi-Resolution Estimation of Ego-Motion and Structure from Motion," *Proc. IEEE workshop on visual motion*, Princeton, NJ, 1991, pp. 156-162.
- [35] Haralick, R.M. and Joo, H. "2D-3D Pose Estimation," *CVPR*, Rome, Italy, pp. 385-391, 1988.
- [36] Harris, C.G. and Pike, J.M. "3D Positional integration from image sequences," *Image and Vision Computing*, Vol 6, no. 2, May 1988, pp. 87-90.
- [37] Heel, J. "Dynamic Motion Vision," *Image Understanding Workshop*, Palo Alto, CA, pp. 702-713, 1989.

- [38] Heel, J. "Temporal Surface Reconstruction," *CVPR*, Hawaii, June, 1991, pp. 607-612.
- [39] Heel, J. and Negahdaripour, S. "Time-sequential Structure and Motion Estimation without Optical Flow," *SPIE Symposium on Electronic Imaging: Sensing and Reconstruction of 3D Objects and Scenes*, 1990.
- [40] Heel, J. and Rao, S. "Temporal Integration of Visual Surface Reconstruction," *Proc. IUW*, Pittsburgh, PA, pp. 376-382, 1990.
- [41] Horn, B.K.P. *Robot Vision*, MIT Press, Cambridge MA, 1986, pp.279-281.
- [42] Horn, B.K.P. "Closed Form Solution of Absolute Orientation Using Unit Quaternions," *J. Opt. Soc. Am. A*, vol. 4, pp. 629-642, 1987.
- [43] Horn, B.K.P. "Relative Orientation," *International Journal of Computer Vision*, Vol. 4, pp. 59-78, 1990.
- [44] Horn, B.K.P. and Weldon, E.J.Jr. "Direct methods for recovering motion," *International Journal Computer Vision*,2, 1988.
- [45] Huber, P.J. *Robust Statistics*, John Wiley and Sons, NY, 1981.
- [46] Iu, S.L. and Wohn, K. "Estimation of 3-D Motion and Structure based on a Temporally-Oriented approach with the method of Regression," *Proc. IEEE Workshop on Motion*, Irvine, CA, March, 1989, pp. 273-281.
- [47] Jepson, A.D. and Heeger, D.J. "A fast Subspace Algorithm for Recovering Rigid Motion," *Proc. IEEE workshop on visual motion*, Princeton, NJ, 1991, pp. 124-132.
- [48] Jerian, C. and Jain, R. "Polynomial methods for Structure from Motion," *Proceedings 2nd IEEE International Conference on Computer Vision*, Tampa, Florida, Dec. 1988, pp. 197-206.
- [49] Jezouin, J.L. and Ayache, N. "3D Structure from a Monocular Sequence of Images," *Proc. Third ICCV*,, Osaka, Japan, pp. 441-445, 1990.
- [50] Koenderink, J.J. and van Doorn, A.J. "Local structure of movement parallax of the plane," *Journal of the Optical Society of America*, Vol. 66, No. 7, 1976, pp. 717-723.
- [51] Krotkov, E. and Kories, R. "Integrating Multiple Uncertain Views of a Static Scene Acquired by and Agile Camera System," MS-CIS-88-11, GRASP Lab, University of Pennsylvania.

- [52] Kumar, R. *Model Dependent Inference of 3D Information from a Sequence of 2D Images*, Ph.D. dissertation, Department of Computer Science, University of Massachusetts, Amherst, 1992.
- [53] Kumar, R. and Hanson, A. "The Application of Pose Determination Techniques to Model Extension and Refinement," *Proceedings DARPA Image Understanding Workshop*, San Diego, California, January 1992.
- [54] Kumar, R.V.R., Tirmalai, A. and Jain, R.C. "A Nonlinear Optimization Algorithm for the Estimation of Structure and Motion Parameters," *CVPR*, San Diego, CA, pp. 136-143, 1989.
- [55] Lawton, D.T. *Processing dynamic image sequences from a moving sensor*, Ph.D. dissertation, Department of Computer Science, University of Massachusetts, Amherst, 1984.
- [56] Liu, Y. Huang, T.S. and Faugeras, O.D. "Determination of Camera Location from 2D to 3D line and point correspondences," *Proceedings IEEE Conference on Computer Vision and Pattern Recognition*, pp 82-88, 1988.
- [57] Longuet-Higgins, H.C. and Prazdny, K. "The interpretation of a moving retinal image," pp. 179-193.
- [58] Lowe, D.G. "Fitting Parameterized Three-dimensional Models to Images," *Proceedings IEEE 3rd International Conference on Computer Vision*, Osaka, Japan, Dec 1990.
- [59] Mahalanobis, P.C. "On the generalized distance in Statistics," *Proc. National Inst. of Science*, 12, 1936, pp. 49-55.
- [60] Marquardt, D.W. *J.Soc. Ind. Appl. Math.*, vol. 11, 1963, pp. 431-441.
- [61] Marr, D., *Vision : A computational investigation into the human representation and processing of visual information*, San Francisco: W.H. Freeman, 1982.
- [62] Matthies, L., Kanade, T. and Szeliski, R. "Kalman Filter-Based Algorithms for Estimating Depth from Image Sequences," *International Journal of Computer Vision*, vol 3, pp. 209-236, 1989.
- [63] Matthies, L., Szeliski, R. and Kanade, T. "Incremental Estimation of Dense Depth Maps from Image Sequences," *CVPR*, Ann Arbor, Michigan, pp. 366-374, 1988.
- [64] Maybank, S. *Theory of Reconstruction from image motion*, Springer-Verlag, Berlin, 1993.

- [65] Metaxas, D. and Terzopoulos, D. "Recursive Estimation of Shape and Nonrigid Motion," *Proceedings IEEE Workshop on Visual Motion*, Princeton, pp 306-311, 1991.
- [66] Mukawa, N. "Estimation of Shape, Reflection coefficients and Illuminant Direction from Image Sequences," *Proceedings 3rd IEEE International Conference on Computer Vision*, Osaka, Japan, 1990, pp 507-512.
- [67] Negahdaripour, S. and Horn, B.K.P. "Direct passive navigation," *IEEE PAMI*, Vol. 9, No.1, 1987.
- [68] Negahdaripour, S. and Horn, B.K.P. "A direct method for locating the focus of expansion," *Computer Vision, Graphics and Image Processing*, Vol.46(3), June 1989.
- [69] Negahdaripour, S. and Lee, S. "Motion Recovery from Image Sequences Using First-Order Optical Flow Information," *Proc. IEEE workshop on visual motion*, Princeton, NJ, pp. 132-139, 1991.
- [70] Oliensis, J. and Thomas, J.I. "Incorporating Motion Error in Multi-frame Structure from Motion," *Proceedings IEEE Workshop on Visual Motion*, Princeton, pp 8-13, 1991.
- [71] Pentland, A.P. "Spatial and Temporal Surface Interpolation using Wavelet Bases," MIT Media Lab Vision and Modeling Group, Technical Report No. 142, June 1990.
- [72] Poelman, C.J. and Kanade, T. "A paraperspective Factorization Method for Shape and Motion Recovery," *Proc. Image Understanding Workshop*, Washington D.C., pp 683-690, April 1993.
- [73] Prazdny, K. "Determining the instantaneous direction of motion from optical flow generated by a curvilinearly moving observer," *CVPR*, Dallas, Texas, 1981, pp. 109-114.
- [74] Prazdny, K. "On the information in Optical Flow," *Computer Vision, Graphics and Image Processing*, Vol. 22, 1983, pp.239-259.
- [75] Press, W.H., Flannery, B.P., Teukolsky, S.A. and Vetterling, W.T. *Numerical Recipes in C*, Cambridge University Press, Cambridge, U.K., 1988.
- [76] Rao, C.R. *Linear Statistical Inference and Its Applications*, 2nd Ed., Wiley, New York, 1973.
- [77] Rousseeuw P.J. and Leroy, A.M. *Robust Regression and Outlier Detection*, John Wiley and Sons, NY, 1987.

- [78] Sawhney, H.S. *Spatial and Temporal Grouping in the Interpretation of Image Motion*, Ph.D. dissertation, Dept. of Computer Science, University of Massachusetts, Amherst, 1992.
- [79] Sawhney, H.S. and Hanson, A.R. "Identification and 3-D Description of "Shallow" Environmental Structure in a Sequence of Images," *CVPR*, Hawaii, June, 1991, pp. 179-185.
- [80] Sawhney, H.S., Oliensis, J. and Hanson, A.R. "Description and Reconstruction from Image Trajectories of Rotational Motion", in *ICCV*, Osaka, Japan, December, 1990, pp. 494-498.
- [81] Shahraray, B. and Brown, M.K. "Robust Depth Estimation from Optical Flow," *Proceedings 2nd IEEE International Conference on Computer Vision*, Tampa, Florida, Dec. 1988, pp. 641-650.
- [82] Shigang, L., Tsuji, S. and Imai, M. "Determining of Camera Rotation from Vanishing Points of Lines on Horizontal Planes," *Proceedings 3rd IEEE International Conference on Computer Vision*, Osaka, Japan, 1990, pp. 499-502.
- [83] Spetsakis, M.E. and Aloimonos, J. "Optimal Computing of Structure from Motion Using Point Correspondences in Two Frames," *Proceedings 2nd IEEE International Conference on Computer Vision*, Tampa, Florida, Dec. 1988, pp. 449-538.
- [84] Spetsakis, M. and Aloimonos, J. "A Multi-frame Approach to Visual Motion Perception," *Proc. IEEE Workshop on Motion*, Irvine, CA, March, 1989.
- [85] Stephens, M.J., Blissett, R.J., Charnley, D., Sparks, E.P. and Pike, J.M. "Outdoor Vehicle Navigation Using Passive 3D Vision," *CVPR*, San Diego, CA, pp. 556-562, 1989.
- [86] Sundereswaran, V. "Egomotion from Global Flow Field Data", *Proc. IEEE workshop on visual motion*, Princeton, NJ, 1991, pp.140-145.
- [87] Szeliski, R. "Shape from Rotation," *CVPR*, Hawaii, June, 1991, pp. 625-631.
- [88] Taalebinezhaad, M.A. "Direct Recovery of Motion and Shape in the General Case by Fixation," *PAMI*, 1992.
- [89] Taylor, C.J., Kreigman, D.J., and Anandan, P. "Structure and Motion in Two Dimensions from Multiple Images: A Least Squares Approach," *Proc. IEEE workshop on visual motion*, Princeton, NJ, pp. 242-248, 1991.
- [90] Technical Staff, The Analytical Sciences Corp., and Gelb, A. ed., *Applied Optimal Estimation*, MIT Press, 1986.

- [91] Thomas, J.I. and Oliensis, J. "Fusing Structure by Kalman Filtering," TR 90-93, COINS, UMASS, May 1990.
- [92] Thomas, J.I. and Oliensis, J. "Incorporating Motion Error in Multiframe Structure from Motion," *7th Scandinavian Conference on Image Analysis*, Denmark, pp. 950-957, 1991.
- [93] Thomas, J.I. and Oliensis, J. "Recursive Structure from Multi-frame Motion," *Proc. Darpa Image Understanding workshop*, Los Angeles, CA, 1992.
- [94] Tomasi, C. and Kanade, T. "Factoring Image Sequences into Shape and Motion," *Proc. IEEE workshop on visual motion*, Princeton, NJ, pp. 21-28, 1991.
- [95] Tsai, R.Y. and Huang, T.S. "Uniqueness and estimation of 3-D motion parameters and surface structures of rigid objects," pp. 135-171.
- [96] Tziritas, G. "Estimation of Motion and Structure of 3-D Objects from a Sequence of Images," *Proceedings 1st IEEE International Conference on Computer Vision*, London, England, 1987, pp. 693-697.
- [97] Ullman, S. *The interpretation of visual motion*, MIT Press, Cambridge, Massachusetts, 1979.
- [98] Ullman, S. "Analysis of visual motion by biological and computer systems," *Computer*, 1981, pp. 57-69.
- [99] Verri, A. and Poggio, T. "Against quantitative optical flow," *Proceedings 1st IEEE International Conference on Computer Vision*, London, England, 1987, pp. 171-180.
- [100] Vieville, T. and Faugeras, O. "Feed-Forward Recovery of Motion and Structure from a sequence of 2D-Lines Matches," *Proceedings 3rd IEEE International Conference on Computer Vision*, Osaka, Japan, 1990, pp. 517-520.
- [101] Weems, C.C., Levitan, S.P., Hanson, A.R., Riseman, E.R., Shu, D.B. and Nash, J.G. "The Image Understanding Architecture," *IJCV*, 2, pp. 251-282.
- [102] Weldon, E.J. Jr. and Lui, H. "How Accurately Can Direct Motion Vision Determine Depth," *CVPR*, Hawaii, June, 1991, pp. 613-618.
- [103] Weng, J., Ahuja, N., Huang, T. "Closed-Form Solution + Maximum Likelihood: A Robust Approach to Motion and Structure Estimation," *CVPR*, Ann Arbor, Michigan, 1988, pp. 381-386.
- [104] Weng, J., Ahuja, N., Huang, T. "Optimal Motion and Structure Estimation," *CVPR*, San Diego, CA, pp. 144-152, 1989.

- [105] Weng, J., Huang, T. and Ahuja, N. "Error Analysis of Motion Parameter Estimation from Image Sequences," *Proceedings 1st IEEE International Conference on Computer Vision*, London, England, 1987, pp. 703-707.
- [106] Weng, J., Huang, T. and Ahuja, N. "A Two-Step Approach to Optimal Motion and Structure Estimation," *Proc. IEEE Computer Society Workshop on Computer Vision*, 1987, Miami Beach, Fl, pp.355-357.
- [107] Weng, J., Huang, T. and Ahuja, N. "Motion from Images: Image Matching, Parameter Estimation and Intrinsic Stability," *Proc. IEEE Workshop on Motion*, Irvine, CA, March, 1989, pp. 359-366.
- [108] Weng, J., Lui, Y., Huang, T. and Ahuja, N. "Estimating Motion/Structure from Line Correspondences: A Robust Linear Algorithm and Uniqueness Theorems," *CVPR*, Ann Arbor, Michigan, 1988, pp. 387-392.
- [109] Williams, L.R. and Hanson, A.R. "Translating Optical Flow into Token Matches and Depth from Looming," *Proc. 2nd Intl. Conf. on Computer Vision*, pp. 441-448, 1988.
- [110] Wolf, P.R. *Elements of Photogrammetry*, II edition, McGraw-Hill, New York, NY, 1983.