Object tracking with a pan, tilt and zoom camera

Mark Macdonald Sinclair



A thesis submitted for the degree of Doctor of Philosophy.
The University of Edinburgh.
November 2005

UNIV. EDINBURGH

Abstract

Tracking moving objects has always been of great interest in computer vision with applications ranging widely from CCTV surveillance to smart interactive rooms to motion picture special effects. Many of the tracking methods common in robotics and computer vision use static camera systems with a fixed focal length lens. However in other applications, such as person tracking in security systems, the camera is able to pan, tilt and zoom. A secondary task could be the identification of people and objects which is often difficult without high resolution images. Such images can be easily obtained with a pan/tilt/zoom camera. As the object of interest is often moving there is a need to be able to track moving objects while the camera is rotating and zooming.

This thesis reviews the existing work done on tracking with pan/tilt/zoom cameras and proposes a novel method of multiscale block tracking with perspective cameras using principles and methods from both computer vision and computer graphics.

The theory of perspective transforms in 3D graphics is reviewed as well as relating real world cameras to synthetic cameras through camera calibration. Conventional 2D block tracking methods are reviewed and then expanded to track under pan, tilt and zoom conditions.

Two methods of pan/tilt/zoom tracking are developed and then applied to blocks and blobs (connected regions of pixels). A novel offline technique of fitting contours to these track points is then explored with and without the presence of pan, tilt and zoom. A method of updating the multiscale reference block is presented and the tracker is evaluated without block updating (feed forward) and with block updating and position filtering.

Experimental results for people following are presented using synthetic ray-traced data with absolute ground truth and real sequences taken with a commercial pan/tilt/zoom camera with hand estimated ground truth. All sequences were processed offline, however, with optimisation it appears to be feasible to implement a real-time solution. Applying position filtering and prediction, using a Kalman filter, improves results and robustness.

The application of this method to existing vision techniques is discussed as well as the possible methods for real-time implementation of these algorithms.

Declaration of originality

I hereby declare that the research recorded in this thesis and the thesis itself was composed and originated entirely by myself in the School of Engineering and Electronics at The University of Edinburgh.

The initial work on contour tracking for fixed view cameras was done in collaboration with Peter Hillman, however the further extensions of this work to tracking under pan, tilt and zoom conditions is entirely my own work.

MARK MACDONALD SINCLAIR

Acknowledgements

Firstly, I would like to thank Tom Bishop for patiently walking around in front of the cameras while capturing the test sequences. My thanks also go to Chris Haworth and Peter Hillman for helping capture initial test sequences.

A big thanks goes to Sudipta Sinha and Marc Pollefeys for kindly providing portions of their sparse Levenberg-Marquadt source code which greatly sped up the development of the camera calibration method. A special thanks goes to Sudipta for answering a long stream of e-mails.

Thank you to Bernie Mulgrew, James Hopgood and Bob Fisher for various discussions on numerical methods – all of your insights were invaluable.

I would also like to thank the various people from IDCOM and IMNS for the many hours of conversation over lunch on a variety of odd subjects! In no particular order they are: Peter Hillman, Andrew Peacock, Chris Haworth, Damon Thomson, James Hopgood, Christoph Burger-Scheidlin and Michael Bennett. A special thanks goes to David Stewart and members of IT team who do a fantastic job that often goes unrecognised. Many thanks to Pippa, Hattie and Tom for proofreading this document.

As with anything in life, there needs to be a balance. This work would not have been possible without the bad influence of the members of the Edinburgh University Windsurfing and Surfing Club. Thank you all for spending many hours enjoying the windswept parts of Scotland together.

Thanks also to the Tytler Court crew (Hattie, Keir, Rob and Sandeep) for taking me in when no-one else would and all the great chat.

My greatest thanks goes to my supervisors, John Hannah and David Renshaw. Thank you both for your solid guidance and support throughout the last several years. I hope you have both benefitted as much from this as I have!

I would like to thank my parents for their unconditional love and support. None of this would be possible without you both and I am eternally grateful. Thank you also to Pippa and my surrogate parents, Allan and Pat, for making me always feel at home in Scotland – you have all been wonderful.

Last but not least, I would like to thank Mona for all her love and support in the final dark days. Without you, it would have been a very long and lonely road to the end.

Contents

Declaration of originality . . . iii
Acknowledgements . . . iv
Contents . . . v
List of figures . . . xi
List of tables . . . xvii
Terms, acronyms and abbreviations . . . xviii
Notation . . . xix

1 Introduction **1**
1.1 Motivation . . . 3
1.2 Contributions . . . 5
1.3 Thesis Structure . . . 6
1.4 Summary . . . 7

2 Background **9**
2.1 Introduction . . . 9
2.2 Object tracking under pan, tilt and zoom conditions . . . 11
2.3 Tracking People and Surveillance . . . 14
2.4 Camera Calibration . . . 17
2.5 A Review of Zoom Tracking Methods . . . 22
2.6 Applications of Pan/Tilt/Zoom tracking . . . 29
2.7 Hardware Overview . . . 30
2.8 Summary . . . 32

3 Camera Models **33**
3.1 Introduction . . . 33
3.2 Co-ordinate Systems . . . 34
3.2.1 World Space . . . 35
3.2.2 Camera View Space . . . 35
3.2.3 Viewing Frustum . . . 35
3.2.4 Image Space . . . 37
3.3 Viewing in 2D . . . 38
3.4 Viewing in 3D . . . 39
3.4.1 Transformation Matrices . . . 40
3.5 OpenGL-style Rendering Pipeline . . . 42
3.5.1 Describing a Camera . . . 43
3.5.2 Camera View Matrix . . . 45
3.6 Perspective Camera Model . . . 47
3.6.1 Perspective Projection . . . 47
3.6.2 Viewport Transform . . . 48
3.6.3 Inverse Perspective Projection . . . 49
3.6.4 Povray Perspective Camera . . . 50

3.7 Texture Mapping . . . 51
3.8 Environment Mapping and Front Cube Face Projections . . . 55
3.9 Pan/Tilt/Zoom Camera Model . . . 57
3.9.1 Zoom . . . 57
3.9.2 Rotation . . . 58
3.9.3 Translation . . . 59
3.9.4 Rotation and Zoom . . . 59
3.10 Synthetic vs Real Cameras . . . 60
3.10.1 Radial Distortion . . . 60
3.10.2 Depth of Field . . . 61
3.10.3 Frame Grabbers and Interlacing . . . 61
3.11 Camera Calibration . . . 62
3.11.1 Intrinsic Parameters . . . 62
3.11.2 Extrinsic Parameters . . . 63
3.11.3 Camera Transform . . . 63
3.12 Convergence of Computer Graphics and Image Processing . . . 64
3.12.1 Projecting points using a camera calibration and a perspective transform 64
3.13 Summary . . . 65

4 Pan/Tilt/Zoom Tracking **67**
4.1 Introduction . . . 67
4.2 Block Tracking Without Pan/Tilt/Zoom . . . 67
4.2.1 Definition – Reference Block . . . 68
4.2.2 Definition – Search Region . . . 69
4.2.3 Definition – Match Position and Match Value . . . 70
4.2.4 Definition – Reference Block Updating . . . 71
4.2.5 Block Matching Methods . . . 72
4.2.6 Match Position Prediction and Filtering . . . 73
4.2.7 Reference Block Updating . . . 76
4.2.8 FIR block updating . . . 76
4.2.9 Kalman block updating . . . 77
4.2.10 Robustness and Occlusions . . . 79
4.3 Summary of Current Zoom Tracking Methods . . . 80
4.4 Transforming Between Views . . . 81
4.4.1 Forward and Reverse Camera Transforms . . . 81
4.4.2 Warping Images Between Views . . . 82
4.4.3 Texture Mapping . . . 84
4.5 Zoom Tracking Method 1 – Warp Search Area . . . 86
4.5.1 Reference Block Updating . . . 89
4.6 Zoom Tracking Method 2 – Warp Reference Block . . . 90
4.6.1 Reference Block Updating . . . 90
4.7 Discussion . . . 93
4.8 Region Tracking . . . 94
4.8.1 Average Scene Background . . . 94
4.8.2 Warping the average scene background . . . 95
4.8.3 Segmentation and Target Identification . . . 96
4.8.4 Dynamic background updating . . . 96

4.8.5 Comparison to Block Tracking 97
4.9 Summary 97

5 Contour Tracking **99**
5.1 Introduction 99
5.2 Snakes fit to object boundaries 100
5.3 Contour tracking with a fixed view camera 101
5.3.1 Adapting snakes to tracking contours 101
5.3.2 Control Points 102
5.3.3 External Forces 106
5.3.4 Internal Forces 108
5.3.5 Iterative Algorithm 110
5.3.6 Using the contour for interframe position interpolation 113
5.3.7 Alternative kinematic models 113
5.3.8 Limitations 114
5.4 Contour tracking with a pan/tilt/zoom camera 116
5.4.1 Contour fit in world space 116
5.4.2 Least squares spline fit 117
5.4.3 Tracking with the contour 119
5.4.4 Reference block updating 121
5.4.5 Limitations 121
5.5 Applications 122
5.6 Summary 122

6 Camera Calibration **123**
6.1 Introduction 123
6.2 Overview of Camera Calibration Techniques 123
6.3 Bundle Adjustment 124
6.4 Self-calibration 127
6.4.1 Implementation Overview 127
6.4.2 Image Grid 129
6.4.3 Points Correspondences 130
6.4.4 Pairwise Homographies 130
6.4.5 Chain Homographies 136
6.4.6 Global Feature List 137
6.4.7 Radial Distortion Model 142
6.4.8 Bundle I – Radial distortion and homographies 142
6.4.9 Hartley's Linear Method for finding Intrinsic Parameters 143
6.4.10 Rotation matrices 145
6.4.11 Bundle II – Refining the intrinsic matrix and view rotations 146
6.4.12 Zoom Calibration 147
6.5 Camera Model 149
6.5.1 Camera pan and tilt model (Extrinsic Parameters) 149
6.5.2 Zoom Model (Intrinsic Parameters and Radial Distortion) 151
6.5.3 Integration in tracking framework 152
6.6 Calibration and Results 153
6.6.1 Sequences 154

6.6.2 Implementation . . . 155
6.6.3 Results – Camera Model . . . 160
6.7 Model Verification . . . 165
6.8 Summary . . . 168

7 Testing and Results **169**
7.1 Introduction . . . 169
7.2 Definitions and interpreting tables . . . 170
7.3 Test Sequences . . . 171
7.3.1 Overview . . . 171
7.3.2 Synthetic Sequences . . . 171
7.3.3 Real World Sequences . . . 172
7.3.4 Sequence – `image1` . . . 174
7.3.5 Sequence – `man1` . . . 176
7.3.6 Sequence – `bottle1` . . . 178
7.3.7 Sequence – `tom1` . . . 180
7.3.8 Sequence – `tom3` . . . 182
7.3.9 Sequence – `tom4` . . . 184
7.3.10 Sequence – `cyclist` . . . 186
7.3.11 Sequence – `carpark` . . . 186
7.4 Block Tracking . . . 189
7.4.1 NCC Block Trackers . . . 190
7.4.2 Multiscale Trackers . . . 195
7.4.3 Discussion . . . 204
7.4.4 Runtime Performance . . . 205
7.5 Contour Tracking . . . 206
7.5.1 Fixed view . . . 206
7.5.2 With pan, tilt and zoom . . . 208
7.5.3 Discussion . . . 216
7.6 Region Tracking . . . 220
7.6.1 Implementation . . . 220
7.6.2 Results . . . 224
7.6.3 Discussion . . . 224
7.7 Comparison of methods and discussion . . . 226
7.8 Summary . . . 228

8 Conclusions and Future Work **231**
8.1 Introduction . . . 231
8.2 Discussion . . . 231
8.3 Future Work . . . 235
8.3.1 Multiscale tracking . . . 236
8.3.2 Hardware acceleration . . . 237
8.3.3 Synthetic Sequences . . . 237
8.3.4 Contour Tracking . . . 238
8.4 Final Remarks . . . 240

References **241**

A Results **253**
A.1 Block Tracking . . . 253
A.1.1 Sequence – `image1` . . . 253
A.1.2 Sequence – `man1` . . . 258
A.1.3 Sequence – `bottle1` . . . 263
A.1.4 Sequence – `tom1` . . . 268
A.1.5 Sequence – `tom3` . . . 273
A.1.6 Sequence – `tom4` . . . 278
A.2 Contour Tracking . . . 283
A.2.1 Sequence – `bottle1` . . . 283
A.2.2 Sequence – `tom1` . . . 286
A.2.3 Sequence – `tom3` . . . 289
A.2.4 Sequence – `tom4` . . . 292

B Numerical Optimisation **295**
B.1 Introduction . . . 295
B.2 General optimisation problem . . . 295
B.3 Newton's Method . . . 295
B.3.1 Gauss-Newton Method . . . 296
B.4 Levenberg-Marquardt minimisation . . . 298
B.5 Block Matrices . . . 298
B.6 Numerical Jacobian and Hessian Matrices . . . 299
B.6.1 Jacobian Matrix . . . 299
B.6.2 Hessian Matrix . . . 300
B.7 Bundle adjustment . . . 301
B.7.1 Partitioning parameters . . . 302
B.7.2 Implementation . . . 304
B.7.3 Bundle Adjustment with two partitions . . . 306
B.7.4 Bundle Adjustment with three partitions . . . 308

C Pan/Tilt/Zoom Camera Calibration **315**
C.1 Jacobians – Pairwise homographies . . . 316
C.1.1 Partition A . . . 317
C.1.2 Partition B . . . 317
C.2 Jacobians – Bundle I . . . 318
C.2.1 Partition A . . . 318
C.2.2 Partition B . . . 319
C.2.3 Partition C . . . 320
C.3 Jacobians – Bundle II . . . 322
C.3.1 Partition A . . . 323
C.3.2 Partition B . . . 323
C.3.3 Partition C . . . 324
C.4 Jacobians – Zoom Final Bundle . . . 326
C.4.1 Partition A . . . 326
C.4.2 Partition B . . . 327
C.5 Matrix normalisation . . . 328

D Software **329**
D.1 Povray 329
D.2 Sony EVI-D31 Serial Driver 330
D.2.1 VISCA Protocol 330
D.2.2 Implementation 331
D.3 Vision C/C++ Library 332
D.3.1 Organisation 333
D.4 Applications 333
D.4.1 Ground Truth 333
D.4.2 Grid Capture 334

List of figures

2.1 Applications of pan, tilt and zoom tracking . . . 11
2.2 Applications for zoom tracking . . . 12
2.3 Some methods of object tracking organised by complexity . . . 13
2.4 Object tracking for surveillance and security applications . . . 13
2.5 Sony EVI-D31 . . . 30

3.1 Camera view space . . . 36
3.2 Viewing frustum . . . 37
3.3 Rendering pipeline . . . 42
3.4 Describing a perspective camera . . . 43
3.5 Povray Camera . . . 51
3.6 Texture Mapping . . . 52
3.7 Texture mapping an image onto a plane . . . 53
3.8 Virtual reality panoramic image . . . 54
3.9 Environment Map . . . 55
3.10 Front cube face projection . . . 56
3.11 Scene viewed at two zoom factors . . . 57
3.12 Scene viewed at several camera rotations . . . 58
3.13 Scene viewed at several camera rotations and increasing zoom . . . 59
3.14 Radial distortion effects . . . 60

4.1 Reference block . . . 69
4.2 Search region . . . 70
4.3 Match position . . . 71
4.4 Reference block updating . . . 71
4.5 Constant velocity Kalman position filter . . . 73
4.6 Example FIR block updating . . . 77
4.7 Kalman block updating – reshaping the reference block into a column vector . . . 78
4.8 Example Kalman block updating . . . 79
4.9 Forward and reverse camera transforms . . . 82
4.10 Transform between two quads . . . 83
4.11 Warping a quad to a rectangular quad . . . 84
4.12 Texture mapping a triangle . . . 84
4.13 Sub-pixel alignment when texture mapping . . . 85
4.14 Zoom tracking – Method 1 (Part 1) . . . 87
4.15 Zoom tracking – Method 1 (Part 2) . . . 88
4.16 Zoom tracking – Method 2 (Part 1) . . . 91
4.17 Zoom tracking – Method 2 (Part 2) . . . 92
4.18 Warping the average scene background . . . 95

5.1 Forward and backward tracking . . . 99
5.2 Tracking contour . . . 102

5.3 Contour and areas of false high matches . . . 110
5.4 Contour with match forces settling . . . 111
5.5 Algorithm for contour tracking in a fixed view scene . . . 112
5.6 Alternative contour kinematic spring model . . . 114
5.7 Alternative contour kinematic elastic model . . . 114
5.8 Limitations of contour tracker . . . 115
5.9 Contour fit in world space of sequence `tom3` . . . 117
5.10 Algorithm for contour tracking under pan, tilt and zoom conditions . . . 120

6.1 Jacobians and the Hessian matrix for three partitions . . . 125
6.2 Missing data and Jacobians for Bundle Adjustment . . . 125
6.3 Grid for pan and tilt calibration and zooming in one direction . . . 129
6.4 Finding the pairwise homographies for pan and tilt . . . 131
6.5 Finding the pairwise homographies for zoom . . . 132
6.6 Chain homographies between views . . . 137
6.7 Global features for `lab1` . . . 138
6.8 An example global feature list . . . 138
6.9 A subset of the global features for `lab1` . . . 140
6.10 A subset of the global features for `zoom1` . . . 141
6.11 Pan and tilt calibration sequence `lab1` . . . 156
6.12 Pan and tilt calibration sequence `lab2` . . . 156
6.13 Pan and tilt calibration sequence `lab3` . . . 157
6.14 Pan and tilt calibration sequence `jcmb1` . . . 157
6.15 Pan and tilt calibration sequence `outside1` . . . 158
6.16 Zoom calibration sequence `lab1` . . . 158
6.17 Zoom calibration sequence `lab2` . . . 159
6.18 Pan and tilt angles (degrees) vs camera pan and tilt indexes . . . 163
6.19 Focal length (pixels) vs camera zoom parameter . . . 163
6.20 Principal point (pixels) vs camera zoom index . . . 164
6.21 Radial distortion parameters vs camera zoom index . . . 164
6.22 `lab1` rendered to the front cube face from camera parameters using the camera model . . . 165
6.23 `outside1` rendered to the front cube face from camera parameters using the camera model . . . 166
6.24 Zoom sequence rendered to the front cube face from camera parameters using the camera model . . . 166
6.25 A random series of camera pans, tilts and zooms rendered to the front cube face using the camera model . . . 167
6.26 A random series of camera pans, tilts and zooms rendered to the front cube face using the camera model . . . 167

7.1 Sequence `image1` . . . 174
7.2 Ground truth for sequence `image1` on composite image . . . 175
7.3 Camera parameters for sequence `image1` . . . 175
7.4 Sequence `man1` . . . 176
7.5 Ground truth for sequence `man1` on composite image . . . 177
7.6 Camera parameters for sequence `man1` . . . 177

7.7 Sequence **bottle1** . . . 178
7.8 Ground truth for sequence **bottle1** on composite image . . . 179
7.9 Camera parameters for sequence **bottle1** . . . 179
7.10 Sequence **tom1** . . . 180
7.11 Ground truth for sequence **tom1** on composite image . . . 181
7.12 Camera parameters for sequence **tom1** . . . 181
7.13 Sequence **tom3** . . . 182
7.14 Ground truth for sequence **tom3** on composite image . . . 183
7.15 Camera parameters for sequence **tom3** . . . 183
7.16 Sequence **tom4** . . . 184
7.17 Ground truth for sequence **tom4** on composite image . . . 185
7.18 Camera parameters for sequence **tom4** . . . 185
7.19 Four frame composite of sequence **cyclist** . . . 186
7.20 Four frame composite of sequence **carpark** . . . 186
7.21 Sequence **cyclist** . . . 187
7.22 Sequence **carpark** . . . 188
7.23 Tracks for NCC trackers . . . 194
7.24 Composite tracks for multiscale tracker . . . 197
7.25 Composite tracks for multiscale tracker . . . 198
7.26 Composite tracks for multiscale tracker . . . 199
7.27 Tracks for multiscale tracker . . . 200
7.28 Errors for multiscale tracker . . . 201
7.29 Reference blocks for multiscale trackers . . . 202
7.30 Reference blocks for multiscale trackers . . . 203
7.31 Composite **cyclist** – contour position shown in yellow . . . 208
7.32 Sequence **cyclist** – start and end tracking positions . . . 209
7.33 Composite **carpark** – contour position shown in yellow . . . 209
7.34 Sequence **carpark** – start and end tracking positions . . . 210
7.35 RMS pixel errors for contour trackers with fixed camera views . . . 211
7.36 Contour Tracks for multiscale tracker . . . 213
7.37 Tracks for contour tracker . . . 214
7.38 Errors for contour tracker . . . 215
7.39 Contour evolution **tom1** . . . 217
7.40 Contour evolution **tom1** . . . 218
7.41 Contour evolution **tom3** . . . 219
7.42 Region Tracking for sequence **tom3** . . . 221
7.43 Region tracking for **tom3** . . . 223
7.44 Region track for sequence **tom3** in world space co-ordinates overlayed on a composite image of the sequence . . . 225
7.45 Corner features separating while zooming . . . 227

A.1 Object tracks using multiscale trackers for sequence **image1** in world space co-ordinates (projected onto the front cube face) overlayed on a composite image of the sequence . . . 254
A.2 Reference block tracked in multiscale Method 1 and Method 2 with FIR updating for sequence **image1** . . . 254

A.3 Object tracks using multiscale trackers for sequence **image1** in image pixel co-ordinates . . . 255
A.4 Errors **image1** . . . 255
A.5 Match confidence for multiscale trackers for sequence **image1** . . . 256
A.6 Object tracks using NCC trackers for sequence **image1** in world space co-ordinates (projected onto the front cube face) overlayed on a composite image of the sequence . . . 256
A.7 Object tracks using NCC trackers for sequence **image1** in image pixel co-ordinates . . . 257
A.8 Match confidence for NCC trackers for sequence **image1** . . . 257
A.9 Object tracks using multiscale trackers for sequence **man1** in image pixel co-ordinates . . . 258
A.10 Object tracks using multiscale trackers for sequence **man1** in world space co-ordinates (projected onto the front cube face) overlayed on a composite image of the sequence . . . 259
A.11 Errors **man1** . . . 260
A.12 Object tracks using NCC trackers for sequence **man1** in world space co-ordinates (projected onto the front cube face) overlayed on a composite image of the sequence . . . 260
A.13 Match confidence for multiscale trackers for sequence **man1** . . . 261
A.14 Object tracks using NCC trackers for sequence **man1** in image pixel co-ordinates 261
A.15 Match confidence for NCC trackers for sequence **man1** . . . 262
A.16 Reference block tracked in multiscale Method 1 and Method 2 with FIR updating for sequence **man1** . . . 262
A.17 Object tracks using multiscale trackers for sequence **bottle1** in image pixel co-ordinates . . . 263
A.18 Object tracks using multiscale trackers for sequence **bottle1** in world space co-ordinates (projected onto the front cube face) overlayed on a composite image of the sequence . . . 264
A.19 Errors **bottle1** . . . 265
A.20 Object tracks using NCC trackers for sequence **bottle1** in world space co-ordinates (projected onto the front cube face) overlayed on a composite image of the sequence . . . 265
A.21 Match confidence for multiscale trackers for sequence **bottle1** . . . 266
A.22 Object tracks using NCC trackers for sequence **bottle1** in image pixel co-ordinates . . . 266
A.23 Match confidence for NCC trackers for sequence **bottle1** . . . 267
A.24 Reference block tracked in multiscale Method 1 and Method 2 with FIR updating for sequence **bottle1** . . . 267
A.25 Object tracks using multiscale trackers for sequence **tom1** in image pixel co-ordinates . . . 268
A.26 Object tracks using multiscale trackers for sequence **tom1** in world space co-ordinates (projected onto the front cube face) overlayed on a composite image of the sequence . . . 269
A.27 Errors **tom1** . . . 270

A.28 Object tracks using NCC trackers for sequence `tom1` in world space co-ordinates (projected onto the front cube face) overlayed on a composite image of the sequence 270
A.29 Match confidence for multiscale trackers for sequence `tom1` 271
A.30 Object tracks using NCC trackers for sequence `tom1` in image pixel co-ordinates 271
A.31 Match confidence for NCC trackers for sequence `tom1` 272
A.32 Reference block tracked in multiscale Method 1 and Method 2 with FIR updating for sequence `tom1` 272
A.33 Object tracks using multiscale trackers for sequence `tom3` in image pixel co-ordinates 273
A.34 Object tracks using multiscale trackers for sequence `tom3` in world space co-ordinates (projected onto the front cube face) overlayed on a composite image of the sequence 274
A.35 Errors `tom3` 275
A.36 Object tracks using NCC trackers for sequence `tom3` in world space co-ordinates (projected onto the front cube face) overlayed on a composite image of the sequence 275
A.37 Match confidence for multiscale trackers for sequence `tom3` 276
A.38 Object tracks using NCC trackers for sequence `tom3` in image pixel co-ordinates 276
A.39 Match confidence for NCC trackers for sequence `tom3` 277
A.40 Reference block tracked in multiscale Method 1 and Method 2 with FIR updating for sequence `tom3` 277
A.41 Object tracks using multiscale trackers for sequence `tom4` in image pixel co-ordinates 278
A.42 Object tracks using multiscale trackers for sequence `tom4` in world space co-ordinates (projected onto the front cube face) overlayed on a composite image of the sequence 279
A.43 Errors `tom4` 280
A.44 Object tracks using NCC trackers for sequence `tom4` in world space co-ordinates (projected onto the front cube face) overlayed on a composite image of the sequence 280
A.45 Match confidence for multiscale trackers for sequence `tom4` 281
A.46 Object tracks using NCC trackers for sequence `tom4` in image pixel co-ordinates 281
A.47 Match confidence for NCC trackers for sequence `tom4` 282
A.48 Reference block tracked in multiscale Method 1 and Method 2 with FIR updating for sequence `tom4` 282
A.49 Contour Tracking – Composite `bottle1` 284
A.50 Contour Tracking – Track `bottle1` 285
A.51 Contour Tracking – Errors `bottle1` 285
A.52 Contour Tracking – Composite `tom1` 287
A.53 Contour Tracking – Track `tom1` 288
A.54 Contour Tracking – Errors `tom1` 288
A.55 Contour Tracking – Composite `tom3` 290
A.56 Contour Tracking – Track `tom3` 291
A.57 Contour Tracking – Errors `tom3` 291
A.58 Contour Tracking – Composite `tom4` 293
A.59 Contour Tracking – Track `tom4` 294

A.60 Contour Tracking – Errors `tom4` 294

B.1 Dependence of each parameter on other parameters in the parameter vector . . 303
B.2 Dependence of each parameter on other parameters in the parameter vector . . 304
B.3 Outline of a bundle adjustment algorithm 305
B.4 Bundle adjustment algorithm for two partitions 307
B.5 Bundle adjustment algorithm for three partitions 310
B.6 Structure of the Jacobians and Hessian matrix for three partitions 313

D.1 Types of VISCA packets and command sequences 330
D.2 Example VISCA timing diagram 331
D.3 VISCA driver layer diagram 331
D.4 VISCA driver transmit and receive queues 332
D.5 Vision library class diagram 334
D.6 Ground Truth Application 335
D.7 Grid Capture Application 335

List of tables

6.1 Summary of calibration sequences 154
6.2 Residual reprojection error for camera calibration sequences 161
6.3 Reprojection error for camera calibration sequences 161

7.1 Overview of sequences and methods 171
7.2 RMS pixel errors for all conventional NCC trackers 191
7.3 Parameters for multiscale trackers 195
7.4 RMS pixel errors for all multiscale trackers sequences 196
7.5 RMS pixel errors for contour tracker – sequences without pan, tilt and zoom . . 207
7.6 Parameters for contour tracker per sequence 210
7.7 RMS pixel errors for contour tracker for sequences with pan, tilt and zoom . . . 210
7.8 Summarised RMS pixel errors all trackers for real world sequences 226

Terms, acronyms and abbreviations

BMA	Block Matching Algorithm	*Sec 4.2 pg 67*
Blob	Connected group of pixels (also called a *region*)	*Sec 4.8 pg 94*
Block	Rectangular area of pixels used with Block Matching Algorithms.	*Sec 4.2 pg 67*
DCT	Discrete Cosine Transform	
FIR	Finite Impulse Response	
Match Position	Point in the image where the reference block matches the current image with the highest match value.	*Sec 4.2.3 pg 70*
Match Value	The confidence of a match normalised to lie in the range 0 to 1.	*Sec 4.2.3 pg 70*
MPEG	Motion Picture Experts Group	
NCC	Normalised cross-correlation is a method of block matching.	*Sec 4.2.5 pg 72*
PC	Personal Computer	
PCA	Principal Component Analysis	
PTZ	Pan, tilt and zoom	
Quad	Any quadrilateral defined by four points in 3D space (homogenous co-ordinates). A quad on a 2D image can be treated as a 3D quad on a plane in the viewing frustum.	
RANSAC	RANdom SAmpling Consensus algorithm	*Sec 6.4.4.2 pg 134*
Reference Block	Template block of the target being tracked.	*Sec 4.2.1 pg 68*
RGB	Red Green Blue	
Search Region	Area of the image to search for the target in; often located around the previous match position.	*Sec 4.2.2 pg 69*
Texture Map	The mapping of pixels from a rectangular image or *texture* onto the surface of a 3D mesh.	*Sec 3.7 pg 51*
Viewing Frustum	The volume resulting from the intersection of two parallel planes with a pyramid. Defines the view volume of a perspective camera.	*Sec 3.2.3 pg 35*

Notation

Vectors and Matrices

a	Scalar
$\mathbf{a} = (a_1, a_2, \ldots, a_n)^\mathsf{T}$	Column vector
$\mathbf{A}$	Matrix
$\mathtt{H}$	Homography
$\mathtt{P}$	Projective transform
$(\mathbf{x}_1^\mathsf{T}, \mathbf{x}_2^\mathsf{T}, \ldots, \mathbf{x}_n^\mathsf{T})^\mathsf{T}$	Column vectors stacked one on top of each other to produce another column vector
$\widehat{\mathbf{a}}, \widehat{\mathbf{A}}, \widehat{\mathtt{H}}$	Estimates of various vectors and matrices

Points and Transforms

$\mathbf{x}$	A point in either 2D or 3D
$\mathbf{x}' = \mathtt{A}\mathbf{x}$	A transformed point projected by some transform $\mathtt{A}$
$\tilde{\mathbf{x}} = (x, y)^\mathsf{T}$	2D inhomogeneous point
$\mathbf{x} = (x, y, w)^\mathsf{T}$	2D homogenous point (normally $w = 1$)
$\mathbf{x} = (x, y, z, w)^\mathsf{T}$	3D homogenous point (normally $w = 1$)

Notation for self-calibration

$\mathbf{x}_i = (x_i, y_i)^\mathsf{T}$	2D inhomogeneous points in image 1
$\mathbf{x}'_i = (x'_i, y'_i)^\mathsf{T}$	2D inhomogeneous points in image 2
$\mathbf{X}_i = (x_i, y_i, x'_i, y'_i)^\mathsf{T}$	Point correspondence between image 1 and image 2
$\mathbf{X} = (\mathbf{X}_1^\mathsf{T}, \mathbf{X}_2^\mathsf{T}, \ldots, \mathbf{X}_n^\mathsf{T})^\mathsf{T}$	Stacked measurement vector of n point correspondences
$\widehat{\mathbf{x}}_i = (\widehat{x}_i, \widehat{y}_i)^\mathsf{T}$	Estimated 2D inhomogeneous points in image 1
$\widehat{\mathbf{x}}'_i = (\widehat{x}'_i, \widehat{y}'_i)^\mathsf{T}$	Estimated 2D inhomogeneous points in image 2
$\widehat{\mathbf{X}}_i = (\widehat{x}_i, \widehat{y}_i, \widehat{x}'_i, \widehat{y}'_i)^\mathsf{T}$	Estimated point correspondence between image 1 and image 2
$\widehat{\mathbf{X}} = (\widehat{\mathbf{X}}_1^\mathsf{T}, \widehat{\mathbf{X}}_2^\mathsf{T}, \ldots, \widehat{\mathbf{X}}_n^\mathsf{T})^\mathsf{T}$	Stacked estimated measurement vector of n point correspondences
$\mathtt{H}$	Homography that maps 2D points from image 1 to 2D points in image 2 such that $\mathbf{x}' = \mathtt{H}\mathbf{x}$
$\widehat{\mathtt{H}}$	Estimated homography that maps 2D points from image 1 to 2D points in image 2 such that $\widehat{\mathbf{x}}' = \widehat{\mathtt{H}}\widehat{\mathbf{x}}$
$\mathbf{H}$	Hessian matrix
$\mathbf{J}$	Jacobian matrix (also called the design matrix)
ϵ	Error vector
δ	Parameter partition increment vector
$\mathtt{U}^*$	Augmented matrix for variable step size. Diagonal of $\mathtt{U}$ entries have been multiplied by $(1 + \lambda)$
$d(\cdot)$	Euclidian distance between two 2D points (in other texts this notation is often used to indicate a generic distance measure operator)

Chapter 1
Introduction

Following and monitoring the movements of other people has been an obsession of humankind since the dawn of time. Whether it was to drive them away from valuables and property, or simply to admire them, people have been watching, tracking and following other people with all sorts of technology. This technology has changed from tracking footprints on the ground to following people with a modern security camera from a control room located many miles away. Luckily, humanity is occasionally interested in non-aggressive uses for technology, so following moving objects has found its way into innocent areas such as robots playing football or into movie special effects.

The advent of modern computing systems has allowed much of this 'following people' work to be conducted by digital systems rather than human operators. In the past twenty years there has been a rise in the number of CCTV and surveillance installations, hence there has been an increase in the amount of research being done in these areas. The high amounts of computing power available and storage capacity has opened up areas of tracking research that were previously impossible to compute at the birth of computer vision and image processing.

Recently, the huge demand for computer graphics hardware in the gaming industry has meant that the advances in graphics hardware have been immense. All desktop computers are now supplied with some of the highest complexity integrated circuits humans have developed, solely for the purpose of rendering 3D graphics. In many computers the amount of graphics processing available exceeds the base processing power. Machine vision and image processing have often drawn on principles of geometry that are common with computer graphics. With this proliferation of graphics hardware, researchers will have access to massively optimised computer graphics hardware that will outperform current software implementations by several times. The result will be the implementation of hybrid computer vision and computer graphics algorithms.

Many examples of this convergence exist already: structure from motion and structured lighting both are techniques that combine methods from computer vision, image processing and computer graphics. This has lead to several commercial products such as 2D3's Boujou[1],

which is used in many post production workflows. The Hawk-eye tennis system[2] is another example which should be familiar to anyone watching television coverage of tennis events such as Wimbledon or any international cricket match.

Methods of tracking objects using computer vision techniques have existed for several decades. Many of these tracking methods have been used for applications from navigation of rovers on Mars to tracking missiles and fighter jets. Eye tracking software has been useful to quadriplegic patients to control a computer mouse cursor by moving their eyes. The obvious applications of object tracking in surveillance scenarios has resulted in intelligent security systems that can identify people and analyze their behaviour only from visual data.

Besides the tasks of identifying the presence of people in a scene or intruder detection, there is a large need to identify people in CCTV footage such as the atrocities of September 11 in the USA or the London tube bombings of 2005. Due to the volume of data in most CCTV systems it is impossible to keep high resolution video sequences, therefore most systems use low resolution video stored on magnetic media. This makes the identification of people difficult due to effects such as noise, interlacing and poor resolution. An expensive solution is to use high resolution digital video cameras, however given the number of cameras necessary to monitor an underground train system, this approach is not viable. Therefore, an alternative is to use an active camera system that can pan, tilt and zoom. By identifying targets of interest using conventional fixed view cameras, it is possible to use the pan, tilt and zoom (PTZ) camera system to obtain high resolution images of the targets by filling the image frame with the target. In other applications, such as television, it may be necessary to track players in a football match with a PTZ camera with the possibility of adding 3D graphics for analyzing the game at a production stage. As a final example, most modern cinema has expansive special effects to satisfy audiences and tracking under pan/tilt/zoom conditions is a useful tool in producing exciting visual effects. These types of PTZ tracking have been investigated in the past by several researchers [2–4]. This thesis adds to that body of knowledge by reviewing past work and presenting novel methods of tracking objects with a PTZ camera.

1.1 Motivation

In recent work there has been much success tracking objects with a PTZ camera using corner feature based methods[4–9]. There has been a large amount of work in tracking objects (such as people and vehicles) with PTZ cameras in surveillance system projects in the USA[3, 10–18] with the intention of analyzing human behaviour and detecting intruders. These methods tend to use motion and colour segmentation. Some of the last block-based tracking methods with a zooming camera was work done by Fayman[19] and Cahn von Seelen[20], but since then no more work into block-only based methods appears to have been published.

With the rapid developments in computer graphics hardware, it is now possible to warp blocks of pixels in real-time using 3D accelerated graphics cards common in all desktop computers. This is leading towards a convergence between computer vision and computer graphics methods, where accelerated graphics hardware technology will lead to improved computer vision algorithms that could possibly run in real-time. Before, this may have required a specialised DSP or VLSI implementations. However, this technology is now accessible to many researchers. Algorithms can now be developed using open source computer vision libraries[21, 22] and standard graphics libraries[23, 24] to produce methods which can be run on a desktop computer. Convergence examples such as structure from motion work by Pollefeys[25–27], self-calibration techniques[28–31] and structured lighting/image based rendering methods by Devebec[32, 33] have inspired the work contained in this thesis.

Hence, a new area of research has become apparent – the modelling of the deformation of blocks under PTZ conditions using projective geometry and using computer graphics methods to warp the block to allow standard block matching methods to be used to track objects while panning, tilting and zooming. A sound basis in projective geometry[31, 34] and computer graphics techniques[35, 36] can be used to implement these algorithms on accelerated graphics hardware using libraries such as OpenGL[23] or DirectX[24]. Cheap consumer electronics PTZ cameras also avoid the need to use complicated robotic pan/tilt heads such as those used in [4, 5, 37, 38]. The PTZ cameras manufactured by Sony and Canon all have serial interfaces that can be used to control the camera position and query the camera for parameters such as the zoom position. These types of cameras have been used in the past to track people [3, 15, 18], however these methods have always used high level models of humans and motion segmentation, rather than block based methods.

Block methods have an advantage over other methods as there is a measure of certainty that the target being tracked is still the same target when the tracker was initialised. For methods using clusters of corner features or motion segmentation, it is not guaranteed that the centroid of the tracked region will remain in the same place on the target. For certain applications, this type of accuracy is critical. However, block tracking is not the 'holy grail' of tracking solutions and has many associated problems, such as robustness to occlusions and temporal drift. Therefore, the methods developed in this thesis are yet another tracking tool which works well for certain applications and scenarios, but are by no means the ultimate tracking solution.

While investigating various zoom tracking methods and using offline sequences for testing, the idea of an offline *tracking contour* evolved. If the start and end points of a track are known *a priori*, a fair idea of the target's trajectory can be inferred from only a few match points. This tracking contour idea was initially inspired by Kass's work on fitting snakes[39] to object edge boundaries. Some modifications were needed to use the tracking contour with sequences where the camera was panning, tilting and zooming. The tracking contour method has many advantages such as making tracking robust when the target experiences long periods of occlusion. As the contour is a continuous function the interframe target positions can be interpolated from the tracking contour.

One of the challenges facing computer vision researchers is obtaining good test sequences. The use of ray-traced[40] artificial sequences provided a good method of generating early test sequences with very controlled conditions, such as constant lighting and little noise. As the tracking methods use principles from computer graphics, the use of a ray-tracer added to the understanding of the geometry of the camera PTZ movement. When using a real PTZ camera it was decided to use the camera telemetry to determine the position, hence it was necessary to model and calibrate the camera. Although much work has been done on the self-calibration of PTZ cameras[28–31] the published work often simplifies the explanation of methods or omits steps, therefore an attempt was made at bringing together this work into a single coherent cookbook method. Although calibration toolboxes exist for fixed intrinsic parameter cameras[22, 41, 42], no tutorial, textbook or comprehensive guide to calibrating a PTZ camera exists. Most techniques use numerical methods such as bundle adjustment[43], which often cloud the process of calibration as the published literature covers too many problems from structure from motion to self-calibration. Textbooks such as [34] and [31] omit steps or leave them up to the reader to calculate as exercises. A large effort has been made to explain these numerical methods,

such as bundle adjustment, in the context of self-calibration of a PTZ camera. All necessary expressions, such as the Jacobians, are defined explicitly in order to demystify the subject and increase the accessibility of the material.

The material contained in this thesis will provide a perspective on how to create a tracking system that could be implemented to work in real-time on a desktop computer using a consumer electronics PTZ camera.

1.2 Contributions

One of the general aims of this thesis is the attempt to bring together the fields of computer vision and computer graphics for the purpose of object tracking. Although this convergence is apparent in other areas of computer vision, the convergence in tracking has not been previously highlighted.

Two novel methods of multiscale block tracking for use with PTZ cameras are presented with the application of surveillance in mind. Concepts such as a 'reference view' for tracking and warping reference blocks using texture mapping are introduced. After warping the reference block is not likely to be rectangular, therefore an approach of comparing these quadrilateral shaped reference blocks to an area of an image is investigated and analyzed.

Under pan, tilt and zoom conditions the best match block is often a different scale to the reference block. A method of updating of a high-resolution reference block by warping areas of good match from an image into the reference view by texture mapping is presented.

Methods of maintaining a PTZ tracking framework and the choices in the size of the reference block are analyzed. As a comparison the extension of these specific methods to region or blob tracking are briefly discussed in the context of surveillance.

A novel method of offline *contour tracking* is introduced. Algorithms for tracking with a fixed view sequence using a kinematic model for match forces are presented. This is then extended to an algorithm for *contour tracking* in sequences where the camera is free to pan, tilt and zoom. The method draws on the novel multiscale block trackers discussed above.

Also contained in this thesis is a cookbook style method for the self-calibration of a PTZ camera. This method collates and combines previous work by several other authors [25, 28–30, 34].

The method is mainly based on a conference paper by Sinha[28]. Often in various sources a few steps are either omitted or superficially discussed. A large effort has been made to explicitly define all expressions needed to implement the calibration method, even though this may be verbose and laborious at times.

1.3 Thesis Structure

An introduction and background material to the general field of object tracking as well as a review of previous PTZ tracking techniques can be found in Chapter 2. Applications of PTZ tracking methods are discussed briefly in the context of surveillance, media post production and robotics.

Chapter 3 reviews concepts from computer vision, projective geometry and computer graphics and attempts to unify the fields into a common view of modelling PTZ cameras. These models are necessary to understand the proposed algorithms in Chapter 4 and Chapter 5. The implementation of these models with graphics libraries such as OpenGL are also discussed.

Chapter 4 begins with a review of conventional block tracking methods. This is then extended to use the models from Chapter 3 to track objects with PTZ cameras. Two novel multiscale block tracking algorithms are introduced.

The offline technique of using a *tracking contour* is explained in Chapter 5. Algorithms for using the tracking contour on sequences with a fixed view camera and a PTZ camera are presented.

Chapter 6 explores various methods of calibrating a consumer electronics PTZ camera such as the Sony EVI-D31 . Previous work on self-calibration for PTZ cameras is collected into a single cookbook style method with references to the full details of the method, which can be found in Appendix C.

Test sequences and the results of testing the various methods of Chapters 4 and 5 can be found in Chapter 7. A comparison of the methods is also presented.

In conclusion, Chapter 8 contains a summary of the work presented in this thesis as well as a critical discussion. Suggestions of future areas of research are also presented.

The full set of results and graphs discussed in Chapter 7 can be found in Appendix A. Ap-

pendix B provides detail on various numerical methods such as Levenberg-Marquardt minimisation and bundle adjustment. Appendix C is dedicated entirely to the calibration of a PTZ camera and expands details of the approach discussed in Chapter 6. A summary of the software developed during the course of this work is presented in Appendix D – a list of classes and some example screenshots are shown.

1.4 Summary

A brief introduction to the applications and existing methods of tracking objects with a PTZ camera have been presented. A more detailed background discussion can be found in Chapter 2 with references to previous work and a full analysis of the applications of PTZ tracking.

The motivation for having done the work presented in this thesis has been examined along with an outline of how that work will be laid out in this document. A summary of the contributions made by this research has also been discussed.

Chapter 2
Background

2.1 Introduction

Recently with the large developments in computer graphics and the expansion of the computer gaming and gaming console markets, huge amounts of research have been done into optimising computer graphics rendering algorithms and hardware. The high volumes of gaming consoles produced has driven down the prices of extremely complicated dedicated graphics hardware to the point where all personal computers have accelerated graphics hardware.

Along with this explosion of graphics hardware, has been the steady increase of processing power in normal computer hardware and an even larger increase in the available memory in the average PC. Therefore, the ability to do real-time image processing on desktop PCs is now possible. There is no longer the need for dedicated image processing hardware, DSP and VLSI implementations, making advanced image processing methods more accessible to researchers. Consequently there has been the development of high performance image processing libraries, such as the Intel Image Processing Library[44] which are fast hand-coded assembler implementations of common image processing algorithms. Open source software development principles have lead to popular image processing libraries such as OpenCV[22] and VXL[21].

Given the availability of cheap high performance computers and advanced graphics hardware, the convergence between computer graphics, computer vision and image processing is inevitable. The previously slow matrix manipulations common in projective geometry can be calculated extremely quickly using accelerated video hardware and matrix extensions such as MMX on Intel processors. Although often overlooked, the visualisation aspects of image processing have benefitted largely from advances in graphics hardware too. The new generation of graphics hardware allows access to the video buffer memory – the result will be hybrid graphics/vision algorithms where graphics hardware is used to improve performance. Manufacturers such as nVidia and ATI allow hardware accelerated rendering to video memory, which can then be transferred back to the system memory. Therefore, all image processing algorithms that texture map 2D images onto 3D shapes (such as those found in structure from motion problems)

can take advantage of this accelerated hardware to perform the graphics operations, previously done slowly in software, extremely quickly in hardware. Existing robust graphics libraries such as OpenGL[23] and DirectX[24] naturally take advantage of this hardware acceleration and provide an easy framework in which to develop computer vision algorithms.

Below are some examples of this convergence that are already present in image processing: Pollefeys[25–27, 45, 46] has produced large amounts of work on *structure from motion* where the Euclidean structure of a scene is determined from the correspondences between various images in a sequence taken by a free-moving camera. Methods of identifying the lighting of a scene from omni-direction images (also known as *structured lighting*) have been developed by Debevec[32, 33] using ideas from computer graphics, such as the reflectance of objects and environment maps[23, 47–49]. The Hawk-eye[2] tennis system is the closest current example of a real-time capable image-processing/graphics convergent system: the trajectory of a tennis ball is predicted to provide a 3D rendered analysis of the match and to verify line calls.

This chapter covers the background material necessary to understand the motivations for the work done in this thesis as well as introducing certain key concepts. A review of zoom tracking methods is presented in Section 2.5. Several methods ranging from block methods to feature-based methods are analyzed and discussed.

An application driven approach will be taken to introduce methods of object tracking in general and under pan/tilt/zoom conditions. Three broad areas of the application of pan, tilt and zoom tracking are shown in Figure 2.1 and will be discussed throughout this chapter. Emphasis will be on the surveillance and robotics applications, as the majority of the previous work in this area has been concerned with following people. The overlap with media post production will be obvious and often examples will be drawn from applications for television and movie special effects.

Most tracking systems can be divided into three basic tasks: target acquisition, target tracking and post tracking tasks. Figure 2.2 shows examples of these three tasks for the areas of application described above. The novel tracking methods presented in this thesis are only concerned with the second task of tracking. This chapter will cover some of the aspects of target acquisition and post tracking tasks to provide a context for the methods and results presented in later chapters.

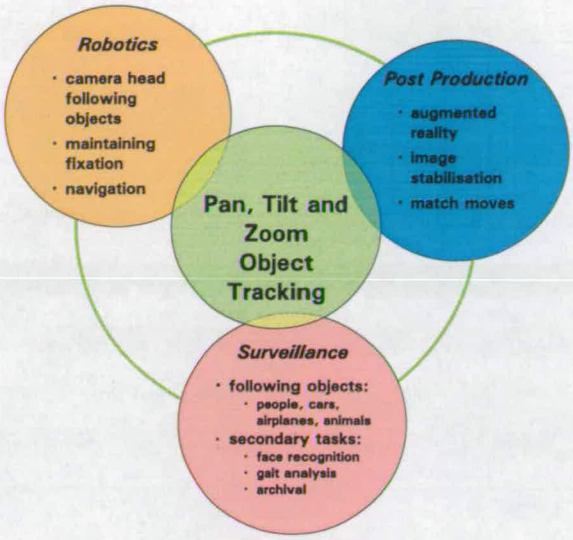


Figure 2.1: *Applications of pan, tilt and zoom tracking*

2.2 Object tracking under pan, tilt and zoom conditions

Tracking objects with a PTZ camera forms a subset of general tracking methods. A selection of tracking methods will be discussed in the context of pan, tilt and zoom tracking throughout this chapter. Figure 2.3 shows the division of object tracking methods into a few groups and organised according to the 'level' of the algorithm.

Low level algorithms use simple image processing techniques such as block matching and convolution kernels to detect edges. They are highly dependent on the content of the image and often use signal processing techniques by treating the image as a temporally varying 2D signal.

Methods such as optical flow and colour distributions still rely on the image data, but tend to work on a level of abstraction not entirely connected to the image. An example of this is segmented regions that can be used without the image, however they still contain pixels from the image that contribute valuable information, such as colour distributions.

The highest level of algorithm uses information that has been abstracted away from the image completely. For example, eigenshapes represent areas of the image as contours that are no longer connected to any image pixels – they can be treated independently of the image. Human stick models are another example where an image has been segmented and classified into an abstract model of a person where the image is now represented as abstract objects such as a head or an arm.

	Robotics	Post Production	Surveillance
Target Acquisition	• user input • good features to track • moving objects	• user input • corner features • algorithms/plug-ins	• other camera views and high level cues • background subtraction and segmentation • occasional user input
Tracking	• maintain fixation on target which is moving • maintain size of target	• accurate object tracking • track background for image stabilisation	• track multiple moving objects • maintain fixation on moving objects
Post Tracking Tasks	• object identification • navigation	• augmented reality • scene stabilisation • sports TV analysis	• face recognition • gait analysis • high resolution photos of people • intruder detection

Figure 2.2: *Applications for zoom tracking*

The novel tracking methods presented in this thesis fall between the categories of low level and medium level algorithms as they are highly dependent on the image data yet they use medium level camera calibration information. This is further illustrated using the example of tracking for surveillance and security applications in Figure 2.4 where the output of tracking is passed on to higher level algorithms, such as face recognition.

An important consideration when choosing a tracking method is the type of information the method outputs. For example, an accurate position of one target in a scene requires a different method to a system which identifies whether or not there are people or cars in a scene. The higher level methods tend to produce results that consist of a list of object types found in a scene and their approximate position. Low level methods produce the exact positions of a group of pixels in the image and some certainty of the match. Real-time systems tend to favour the more high level methods as the amount of data they produce is small and processing loads can be smaller – there is an obvious tradeoff between performance and accuracy.

Often accuracy is more important than performance and offline processing is ideally suited to produce extremely accurate tracks. Methods such as the contour tracker discussed in Chapter 5 can be used to ensure an object is accurately tracked between two exact points in an image

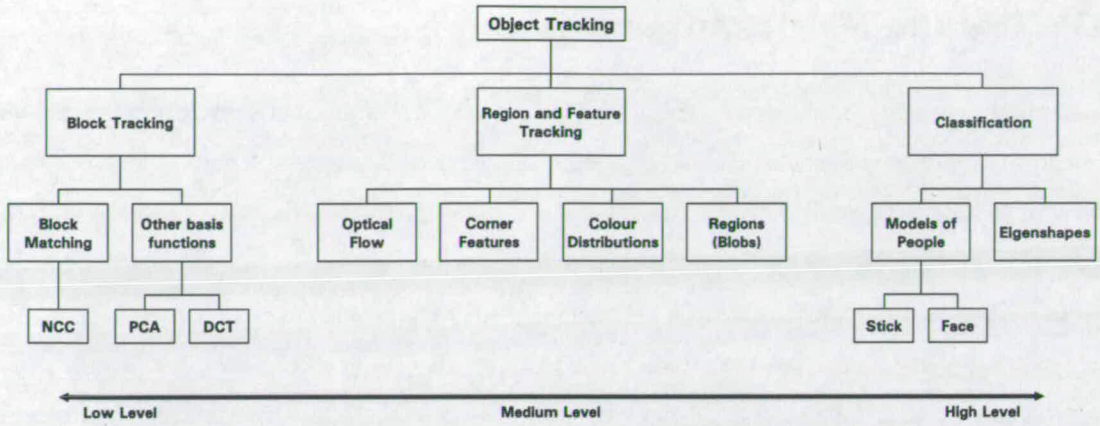


Figure 2.3: *Some methods of object tracking organised by complexity*

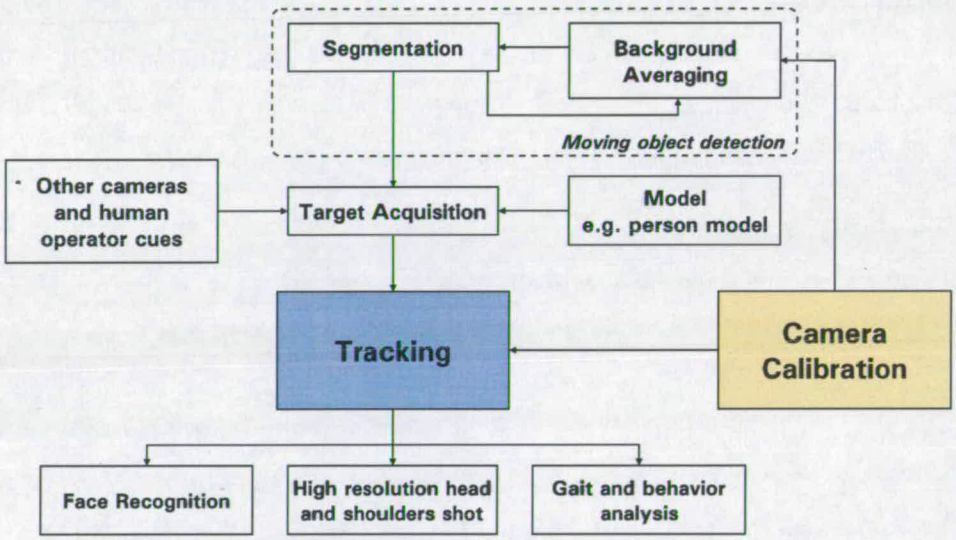


Figure 2.4: *Object tracking for surveillance and security applications*

sequence. This type of tracking has wide applications in media post production where artificially generated image data needs to interact seamlessly with real image data. For example, an artificial character such as Gollum in *Lord of the Rings* who interacts with real actors needs to be rendered and lit in a realistic fashion to create a believable character. This requires accurate tracking of feature points in the image. In another example, fixed PTZ cameras are used to record football matches. Using telemetry from the camera it is possible to produce simple video effects such as artificial viewpoints and adding 3D rendered annotations. However these additions to the sequence often need to interact with the content and some sort of tracking will be necessary.

2.3 Tracking People and Surveillance

Often many tracking applications are aimed at specific scenarios, such as following people walking in a public space with CCTV surveillance systems. These systems tend to use a mixture of fixed cameras and PTZ cameras. Therefore, many methods exist which work with both types of cameras. In order to achieve this, it is necessary to restrict the scenario and track specific types objects such as people or cars. It has been possible to achieve real-time object tracking using high level algorithms[3, 13, 15, 16, 50, 51]. Tracking human movements in 'smart' interactive rooms[11, 52] has been of interest to creating interactive spaces that respond to human movement. If a tracking system needs to identify and follow humans, the obvious solution is to create a model of human geometry and motion. Hogg[53] was among the earliest to suggest tracking by modelling walking movements. Essa[54] reviews many methods that could be used to track people and parts of people, such as hands for gesture recognition. Sometimes the identification of people is sufficient for the application, such as work by Casas[55] in which face detection is used to count the number of people in a crowd at a protest.

Humans also tend to come in many different shapes, sizes and colours, therefore several tracking approaches have been used to deal with the wide variation in the appearances of humans and tracking environments. The most commonly used method of tracking people is *blob* or *region* tracking[3, 11, 18, 56–65]. Blobs are connected groups of related pixels. They are often created by finding pixels which have moved in a scene by subtracting frames from an average view of the scene. Wren[11], Raja[63], Haritaoglu[3], McKenna[64, 65] and Stillman[18] all use motion segmentation to group moving pixel regions together. Jones and Giaccone[66] take this further to group moving regions of interest into moving volumes of interest when there are several views of the same scene. Blobs can also be created by combining a segmentation along edges in the image and motion segmentation[56]. Raja[63] and more recently Kuo[67] use colour segmentation to find moving regions in real-time. Hidalgo[68] combines both spatio-temporal segmentation and colour segmentation to identify moving objects in a scene and maintain a stable background. Other methods, such as work by Bichsel[69], use Bayesian methods to classify pixels as foreground or background pixels. Hager[57] uses an estimate of the change in geometry in a scene to warp regions and blobs to improve region tracking. In other approaches, Xu[70] uses a Support Vector Machine (SVM) to identify pedestrians as blobs in thermal imagery. In order to robustly track blobs it is necessary to maintain a stable background so that moving foreground objects can be removed from the background. Most approaches use frame-

background differences to selectively update the background. Moscheni[71], Dufaux[72] and Kompatsiaris[73] implement methods to do this using spatio-temporal segmentation to distinguish foreground and background objects. Only static objects contribute to the background. These methods also allow objects to be incorporated into the background over time.

Extensive work has been done to track skin coloured regions, although the applications for this have mostly been in video coding and for gesture recognition[3, 52, 54]. Chang[51] and Xu[10] use skin colour models to identify people in a scene. The obvious limitation of these methods is when there is an object in the scene that has a colour distribution similar to that of skin or under a certain type of illumination can appear to be skin-like. The 'apparent colour' of an object is often not only the result of the physical make-up of an object, but dependent on the colour of the lighting in a scene. Therefore the colour of an object will change depending on the how it is illuminated. Early work by Forsyth[74] and Kurt[75] introduce the idea of *colour constancy* which is concerned with the modelling of changes in colour of an object under various sources of illumination. Matas[76], Soriano[77] and Stern[78] use adaptive colour models to compensate for changes in illumination and changes in the physical colour of object (due to rotation for example). Wu[79] proposes a method of classifying the colour of an object using neural networks and self-ordering maps. This allows the identification of specific objects in a scene based on colour. An important application of this is in the handoff of tracking between cameras: Bowden[80] uses colour region tracking to identify moving objects in various camera views and track between them.

Many systems for surveillance have been suggested. Latzel[61] recently describes a system which uses background subtraction and motion bandpass filtering to detect moving objects and their boundaries. In other systems, such as that proposed by Maurin[59], all objects in a scene (such as cars and people) need to be tracked. Blob tracking identifies moving objects and Kalman filters are used to track individual objects. Cai[13], Khan[81] and Stillman[18] also suggest systems for tracking and identifying multiple people with multiple cameras in structured environments. Tracking the same object in multiple views addresses the problem of occlusion in single views to a certain degree[17, 51] but does not solve it entirely as situations occur where there is occlusion in all views. Rosales and Sclaroff[62] use a combined 2D/3D approach to predict when people or other moving targets will occlude each other. An extended Kalman filter is used to estimate every moving objects' position in the scene. Collisions are predicted from the intersection of trajectories predicted by the Kalman filter. After an occlu-

sion or 'collision' each target can be re-identified once the group separates based on predicted positions from the Kalman filter.

A system aimed at home video surveillance is presented by Cucchiara[60, 82] which identifies human behaviour such as walking and lying down. This information can be used to monitor the houses of elderly or infirm people to automatically determine if they require emergency medical assistance. Such a system also uses multiple cameras. Micheloni[83] presents another blob tracking system for use with a camera network. The work is concerned with fusing data between views and designing a surveillance network.

Template matching, such as block tracking and eigenshape matching, can also be used to track people. In order to track a person, who is relatively large in relation to the blocks, several blocks are tracked simultaneously by assuming that the shape of a person does not change quickly, so the tracked blocks will retain their relative positions. Hardware vision systems can perform specific vision tasks extremely quickly, however their versatility is limited: Komuro[84] presents a high speed VLSI implementation of a tracking algorithm which can track multiple objects at speeds of up to 1000 frames/s. The only drawback of this hardware is that the image sensor and tracking hardware are integrated, i.e. processing is done in logic contained inside each pixel. Kim[85] proposes a method which changes the block size in conventional block matching algorithms depending on size of object. This improves results when compared with normal block matching algorithms. Eigenshapes[86] and silhouettes[50] are also common techniques to identifying people in a large scene where they are small. Computationally it is not currently possible to use eigenshapes for large objects, however small areas (roughly 30×40 pixels) can be tracked in real-time.

Most surveillance systems contain PTZ cameras which are used to follow moving targets in a scene. Lim[87] builds on work at the University of Maryland[3, 88] to control the position of a PTZ camera to obtain high resolution images of a moving target. The method uses a master camera with a wide-angle view of the scene to direct the calibrated PTZ camera until the object is viewed at a high zoom factor. Senior *et al*[89] have recently done work in using a PTZ camera and a fixed camera to follow objects of interest in a security system. The homographies between cameras are estimated by tracking features between views and using RANSAC. Their final method attempts to find objects of interest (normally people), observe them at least once and attempt to find the best head view to obtain good face imagery. Details of their tracking method are not given and generally they seem more concerned with producing a commercial

surveillance system.

For the most part surveillance systems for tracking people make the assumption that targets of interest are moving. Therefore, motion segmentation is the obvious choice for a tracking method. Other methods of identifying humans by higher level methods such as stick models have been discussed. However, for any of these algorithms to be implemented in the real world using PTZ cameras, it is necessary to perform some sort of camera calibration.

2.4 Camera Calibration

Camera calibration forms an integral part of pan, tilt and zoom tracking: either it is the product of a method or a means to an end. In order to track objects when the camera parameters are variable, a calibration over the full range of parameters should be done *before* tracking. However, sometimes, camera calibration is the product of finding the Euclidean structure of a scene in a *structure from motion* type problem – in this case the scene structure was the goal and the camera calibration is a by-product. Both types of method are reviewed with more emphasis being placed on conventional camera calibration rather than *structure from motion* methods.

Early work into camera calibration was done by Brown[90] in the early 1970's however the beginnings of camera calibration using calibration objects started with Tsai's paper[41] in 1987. Tsai used a checkerboard pattern on a planar surface and extracted the positions of the corners which he used to estimate the intrinsic and extrinsic parameters of the camera as well as the radial distortion. The radial distortion model used by Tsai has been used by almost all subsequent calibration techniques. A more comprehensive model which incorporate radial as well as tangential distortion is presented by Weng *et al*[91].

The next significant method of camera calibration was proposed by Faugeras *et al*[92]. The method, known as *self-calibration*, does not require a known 3D shape or calibration object, but rather uses the correspondence of corner features between views. Faugeras takes a rigourous projective geometry approach to relate the absolute conic and the plane at infinity. The *Kruppa equations* were coined by Faugeras after work by Kruppa[93] in 1913. The method has since been found to be sensitive to noise and can suffer from convergence problems[94]. Faugeras also insists on using a method by Sturm[95] to find the epipoles published in German in 1869. This insistence of referring to very old papers which are not written in English and the extremely formal 'proposition and proof' style unfortunately makes Faugeras's work inaccessible.

Although his treatment of the material is very elegant, it is shrouded under the veil of formal mathematics. This approach continues in a book by Faugeras and Luong[31] on multiple view geometry. As a reference for projective geometry, it is a fantastic book and provides a comprehensive treatment of the subject matter. However, this also makes it difficult to approach, whereas Hartley and Zisserman's book [34] is much more useful for implementing practical projective geometry algorithms.

Later work by Maybank and Faugeras investigates the self-calibration of moving cameras[96] using methods similar to [92]. Zeller and Faugeras [97] produced a technical report which discusses all of the camera self-calibration methods used at INRIA. It deals with the entire method from finding point correspondences between views, to using the Kruppa equations for determining the camera parameters. Methods of determining the fundamental matrix and the epipolar geometry are also presented. A slightly later piece of work by Hartley [98] provides an alternative derivation of the Kruppa equations.

A new method of camera calibration was proposed by Zhang [42, 99] which uses a maximum likelihood approach to solving for the camera parameters rather than Faugeras's closed form approach. Again, a fixed planar calibration object, such as a checkerboard, is observed in several different views and corner features are matched against a 3D model of the calibration object. The following camera parameters are then estimated by Levenberg-Marquardt minimisation: intrinsic parameters, extrinsic parameters and radial distortion parameters. The method is less susceptible to noise than closed form methods. Camera parameters and radial distortion parameters are simultaneously estimated. The only drawback of the method is that the entire calibration needs to be visible in all views to obtain reliable results and the intrinsic parameters must remain constant.

The OpenCV image processing library [22] contains a C implementation of a MATLABtoolbox by Jean-Yves Bouguet. A planar checkerboard pattern is used to calibrate a fixed intrinsic parameter camera. A human operator specifies the orientation of the checkerboard by clicking on its position in several images. The camera parameters are then estimated using methods based on work by Zhang[42], Heikkila and Silven[100], Sturm[101] and Tsai[41]. A confidence in the parameters is also calculated and can be used to determine whether the calibration was sufficiently accurate. In order to calibrate a PTZ camera, a calibration for several zoom levels would need to be found and a piecewise function used to model the calibration at any intermediate zoom level. One major disadvantage of this method is that the consistency between

calibrations is not maintained. The bundle adjustment methods described later do maintain this consistency.

All of the approaches discussed so far have used either features from a fixed calibration object or corner features from a real-world scene. Willson [102, 103] does not take this approach and tracks the position of a laser on the camera imaging CCD. He is interested in all aspects of calibrating a zoom lens system from the optics behind the image formation to hysteresis in the lens position control motors. Willson's analysis of a zoom lens system is extremely rigourous and certain of his tests cannot be applied to a consumer electronics camera. In general, although the hysteresis effects could be quantified and modelled they are generally ignored as the variation is only a few pixels. The incorporation of a hysteresis model into a tracking system would be fairly complex and it would be debatable as to whether it would improve the accuracy much.

Willson produced a technical report [104] and a paper [105] on various methods of estimating the centre of an image. He analyzes various formulations such as using the centre of projection as the centre of the image and how radial distortion affects the centre of the image. The conclusion of his work is that it is fairly difficult to accurately estimate the centre of the image in many cases and careful choice should be made in selecting a model.

Sturm[106, 107] extends the work by Faugeras[92] to calibrate a moving zoom-lens camera. The Kruppa equations are parameterised in terms of the focal length. A calibration object is used to pre-calibrate the fixed intrinsic parameters of the camera and these are used to initialise the Kruppa equations. The Kruppa equations for each pair of views is found and a minimised solution is found using Levenberg-Marquardt minimisation. The best solution to the Kruppa equations is retained and a piece-wise function is used to create a full zoom model for the camera. The uncertainty and convergence problems of Faugeras's methods are reduced by fixing some of the parameters. Although the best solution from the Kruppa equations is chosen, there is no measure of the consistency of the calibration with increasing zoom. Again, the bundle adjustment structure from motion type approach discussed later addresses this.

Pollefeys has produced a vast amount of extremely good work on structure from motion problems [25–27, 45, 108]. Generally his methods use feature correspondence between views from a free-moving camera to estimate the 3D structure of a scene. In order to achieve this, a calibration step needs to be included so that the structure of the scene has some physical significance. Dense depth maps are built up from stereo feature correspondences and are used to create wire-

frame models of a scene, onto which the original image data is texture mapped. From these, artificial views can be created. This has applications in augmented reality and post production. [46] discusses a method of using a hand-held camera to obtain the structure of an archeological site. [25] is a very useful tutorial presented at several conferences which introduces all concepts in camera calibration and determining structure from motion and has been built on work done by Pollefeys in his PhD thesis[26]. All are extremely useful references and provide good starting points for any work on structure from motion or multiple view calibration problems.

Hartley proposed a method[30] of self-calibration for cameras that only rotates but has fixed intrinsic parameters. The method finds the correspondence between corner features between pairs of views. The pairwise homographies can be determined for four point correspondences using a RANSAC[109] algorithm. The intrinsic parameters and homographies can be related eliminating the camera rotation. Many point correspondences and homographies can then be used to estimate the intrinsic parameters using a SVD and Cholesky decomposition. Results are refined using a non-linear Levenberg-Marquardt minimiser. This method and the associated methods are explained comprehensively in Hartley and Zisserman's book[34].

Independent work by Heyden [110] and Triggs [111] builds on the methods discussed before and add a *bundle adjustment*[43] to globally optimise camera parameters. Bundle adjustment can optimise many parameters, such as camera parameters and point correspondences, so that the globally optimal solution is found. A later paper by Triggs[43] is a comprehensive review of bundle adjustment methods – although it is a good reference, it is possibly not a good starting point and texts such as [34] or [31] should rather be used first.

The first work in specifically calibrating a PTZ camera using bundle adjustment was presented by de Agapito[112]. This work is later expanded in [29] to include radial distortion. The method builds on work by Hartley[30] and uses his *infinite homography constraint* to relate the dual image of the absolute conic and the homographies in each view. A Levenberg-Marquardt iterative algorithm is used to estimate the intrinsic parameters in each view. The results obtained sometimes overfit to the data as the estimation of the principal point is not accurate. The errors in estimating the position of the principal point are a common problem [105].

Later work by de Agapito *et al*[113] looks at linear self-calibration of a PTZ camera using the image of absolute conic rather than its dual. This method is not robust to all scenarios and fails if certain matrices are not positive definite. It does have the advantage of being significantly

faster than iterative methods, however it is less general. As with previous work by Faugeras *et al*[92] linear methods are sensitive to noise and often do not solve, hence iterative methods are preferred as they will normally solve given reasonable initial conditions.

Collins [114] take a slightly different approach to calibrating a PTZ camera system by estimating the dense optical flow between views and minimising the total *sum-of-squared distances* (SSD) between warped images using Powell's method[115]. Powell's method is used as it does not require the calculation of derivatives, as in Levenberg-Marquardt algorithms, however it suffers from slow convergence time. Corner features were discarded as their density in an outdoor scene was not uniform. Outliers due to moving vehicles caused problems due to the sparseness of the problem. Much of the rest of the technical report deals with relating the position of several outdoor landmarks between various cameras for an outdoor surveillance system. In future work they propose to use bundle adjustment to refine the camera calibrations and landmark locations.

Davis and Chen[116] use a non-feature based method of calibrating a network of fixed and PTZ cameras in a surveillance network. Their method identifies the position of an LED in a dark scene in every camera view. The position of the LED is identified in each view and is used to calibrate each camera. The relative positions of the cameras to each other can also be determined.

Sinha and Pollefeys[117] use much of the work described in this section to calibrate a PTZ camera for creating high-resolution mosaics of outdoor scenes. Their method uses a feature based registration method based on work by Hartley and Zisserman[30, 34] and bundle adjustment[43] is used to globally optimise the intrinsic parameters for varying zoom factors. This work is then extended in [28] to calibrate a network of PTZ cameras by calibrating each camera and finding the epipolar geometry between them. Much of the calibration methods used in this thesis are based on this work by Sinha and Pollefeys – a more detailed discussion of their method can be found in Chapter 6.

Recently Lu[118] has produced a survey of methods for reconstructing 3D structure and techniques of camera calibration. It draws together all the work discussed in this chapter and includes other work on stereo calibration.

With a fully calibrated camera an attempt can be made to follow targets with a PTZ camera. Although not all methods require a calibrated camera, self-calibration methods such as fea-

ture correspondence are often used to relate various views of the same scene. Often and not surprisingly, zoom tracking methods have a close resemblance to self-calibration techniques.

2.5 A Review of Zoom Tracking Methods

The following section is a review of existing tracking methods under pan, tilt and zoom conditions in chronological order. A cross-section of methods are discussed from block methods, on which the novel methods of this thesis are based, to higher level model based methods for tracking people with PTZ cameras.

The early days of tracking with pan, tilt and zoom heads are characterised by work attempting to reproduce human vision or biological vision systems. *Active vision* systems became a popular term to describe the control of parameters in a vision system for some advantage, for example controlling the *gaze* of a robot head[1] to follow targets. Work by Brown[119], Coombs[38] and Bradshaw[37] are examples of early active vision work on following targets with pan and tilt robot heads. They are primarily concerned with the control aspects of moving the pan/tilt motors and controlling the vergence[2] of the binocular cameras. They do deal with some basic image processing tasks to detect moving objects or find feature correspondence between views using spatial Fourier transforms, but for the most part are interested in all aspects of creating a robotic device to follow targets, from developing hardware to software. Later work by Rougeaux[120] approaches the same problems from a purely image processing point of view. Optical flow and segmentation are used to detect moving objects from a foveated active vision head and Kalman filters are used to control servo motors, providing smooth target tracking. By the end of the decade the control of robot heads had become a fairly well understood problem.

In the early 1990's, large consumer electronics such as Sony and Canon begin to produce cheap PTZ cameras which can be controlled by a serial interface. This prompted a new branch of zoom tracking which diverges from the stereo vision and vergence problems of robot heads and is concerned with tracking targets using a single PTZ camera. Primarily, these types of cameras have applications in the surveillance and security industries, although there is some scope for automated video conferencing systems in which PTZ cameras rotate to face the speaker [121].

[1] These robot heads consisted of two cameras, also known as binocular cameras, on a pan and tilt head. Sometimes these were zoom cameras, however most often they were fitted with foveated lenses which simulate human vision i.e. they have high detail towards the centre of the image and large low detail peripheral vision.

[2] Vergence is the angle between two binocular cameras mounted a fixed distance apart.

Early work on block tracking while zooming was done by Bajcsy and Cahn von Seelen[20] using a zoom camera mounted on a robot head platform. Fayman *et al*[19] expand this work by tracking using a type of affine block matching. Zoom tracking then shifts to using clusters of corners prompted by Reid and Murray's[122] method of tracking corners by affine transfer – Hayman[5–7] and Tordoff[4, 8, 9] both use corner features to track targets while panning, tilting and zooming. Alternative methods, such as those by Xu[10] and Raja[123] use colour profiles of targets to track them while zooming. It can be assumed that an object will remain roughly the same colour while zooming. You[124] uses another approach and performs a Radon transform which produces a 1D signal which is invariant to scale, rotation and translation. Matching is performed by correlation in one dimension and the image position is found by finding the inverse transform. Yoshimura[125] uses several *eigenimages*, templates at various scales, and compares each with normalised correlation to match them when the tracked object is changing size either naturally or due to zooming. This pyramid approach can be slow for large zoom ranges as many eigenimages need to be compared for each possible zoom.

Bajcsy and Cahn von Seelen[20] produced a paper in 1996 which is the most similar example of previous work to this thesis. In their method, the 2D affine translation and scale change between two frames are estimated by minimising the variance between the pixel intensities in the reference block and a test block in the new frame. A match is produced when the affine parameter estimation is accurate and the test block is the correct match. The reference block is never updated, but rather replaced once the scale change exceeds a threshold. Their motivation for this is that warping the reference block causes blurring and drift due to scale changes and requires accurate estimates of the affine parameters. Fairly large blocks are used and results are shown for 16×16 and 32×32 pixel blocks. Sequences are captured from a zoom camera which is mounted on a pan and tilt head with motion that is either only zooming or slow panning. The target remains stationary, as does the position of the camera. Targets are successfully tracked with scale changes of up to 20% and there is small temporal drift in the template due to the slow panning motion.

There are some clear limitations to the work of Bajcsy and Cahn von Seelen: the sequences they have chosen only have slow, smooth pan/tilt motion and they are happy to accept that tracking fails after a 20% change in scale. Real world applications of zoom tracking will have much faster pan/tilt motion and the cameras have large zoom ranges (in the region of $10\times$ to $15\times$). By using the intensity values of pixels from the reference block to estimate the affine parameters,

they are assuming that the illumination of the target remains constant or that the frame rate is high enough to eliminate changes in the target – in reality neither of these scenarios are valid and block tracking methods such as normalised cross-correlation and block updating have been introduced to compensate for these effects. More robust of methods of estimating scale change (or affine parameters in general) exist now[5, 7, 9] which do not rely on intensity constancy and use more intensity invariant features such as corners with normalised cross-correlation for block matching. Also, the affine transfer method used only takes scale change and translation in two dimensions into account. In a real-world pan, tilt and zoom tracking scenario, the depth of an object will change causing apparent motion which can only be accurately modelled in three dimensions.

Another limitation of Bajcsy and Cahn von Seelen's work is that the target lies near the camera's optical axis during zooming, therefore, there is no prediction for the position of the object. This means apparent motion due to zooming could place the object outside of the search area. Also, by having no position prediction, this method is not robust to occlusions. However, in the authors' defense, robustness to occlusions and position prediction were not common place at the time of publication, nor was the computing power available to perform this except in DSP and VLSI implementations.

Fayman *et al*[19] expand on the work of Bajcsy and Cahn von Seelen, to produce two methods of zoom tracking: the first uses optical flow to maintain the size of a target by zooming and the second, uses block matching to track moving objects while zooming. The optical flow method uses depth estimation from the autofocus sensor to calculate the flow between frames during zooming. Fayman found that noise created unstable flows which produced results that were not accurate enough to remove the scale change by zooming the camera. It was then decided to use a SUSAN[126] operator to find corners in pairs of images during zooming. The corners were then mapped by matching the features in each image using a sum of squared distance (SSD) block matcher. Results were obtained from a photograph of light-bulb moving on a robotic arm while zooming and on a rendered cube produced in OpenGL[23]. The purpose of zoom tracking was to find a dense depth map. This method is a slightly different approach to [20] and is the beginnings of later work by Tordoff[9] and Hayman[7] which track corner features. Fayman *et al* have no need of a calibrated camera as they assume that zooming causes the features to move away from the centre of the image, however this assumes the camera remains fixed. If the camera moves or rotates then this assumption fails. The use of the autofocus sensor

is ingenious but relies on tracking only one object in an empty scene and that the camera has correctly focused on the target. Most commercial PTZ cameras have slow reacting autofocus systems that use interference techniques to focus the image, therefore, for a short period of time, the image is blurred and the depth estimation of the target is uncertain. Therefore, this method is unsuitable for fast moving objects with large amounts of pan, tilt and zoom.

Zoom tracking work up to this point[19, 20] has mainly been block based with Fayman[19] introducing corner features as blocks to track. Hayman[5] extends work by Reid and Murray[122] to track corner features while panning, tilting and zooming using the affine transfer between views. Reid and Murray's method track clusters of corners, as they remain constant over time whereas individual corners are transient, appearing in some views but not others – making them unsuitable for tracking over large periods of time. Hayman uses a corner detector to find features in each view. The corner correspondence is then found by block matching. Corners are constrained to lie within a window of the point of interest to avoid needing to segment foreground and background corners. Hayman identifies this as a limitation, but makes this assumption in order to track in real time. The detection window is moved by transferring its 3D position into the 2D view using data from the camera – this maintains the correct point of interest even if the camera moves. The affine transfer between the images is determined and the target is identified by matching points between views. Results are shown for offline and real-time tracking of a model car using a zoom camera mounted on a pan/tilt head. Hayman notes that when tracking a target with constant velocity, there is a large increase in relative velocity with increased zoom. When panning quickly while zooming the image became blurred due to the motion and errors in the focus point, which caused corners to be detected incorrectly or disappear. Also, as the zoom increases more detail appears and more corners become visible. In a later publication on the same work, Hayman[7] shows sequences where a person is tracked successfully while zooming. He also highlights the problem of maintaining fixation on interesting areas of people (such as the head) while zooming, however no solution for this is offered.

Hayman's work again estimates the affine transfer between views on the fly rather than using camera telemetry and *a priori* calibration information to calculate the affine transfer. If corners are unsuccessfully identified then the estimation of the affine transfer is unreliable and target could be lost. As Hayman points out, maintaining fixation on interesting areas of a human has not been addressed. In a wide angle view of a person, corner features on the entire body

will be visible. Once zooming begins, there is large uncertainty in which group of corners will be tracked, i.e. there could be more corners in the feet and legs of a person rather than the head (which is of more interest in surveillance). Once the zoom factor is large, many of the original corners in the cluster will be outside the image. This method would be unsuitable to a post-production application where artificially rendered image data needs to be overlayed onto an existing image sequence, as the exact position of portions of the image are unknown – all that is known is that the same target is being tracked. However, this type of tracking is very useful in robotics for maintaining fixation on targets. Although this method is fairly resilient to occlusions, it may fail when targets merge and part.

Tordoff[4, 8, 9] addresses some of the problems in Hayman's work by segmenting motion in the image into foreground and background motion. Tordoff again uses the corner feature approach with correlation matching to find feature correspondence between views from an uncalibrated camera. The camera parameters (including radial distortion) and a homography between views are estimated. The foreground and background features are segmented using a Maximum Likelihood Estimation by Sampling Consensus (MLESAC)[127] which incorporates the random sampling of a RANSAC algorithm[109] by measuring the likelihood of a solution. Targets are then tracked to preserve their size by controlling the zoom of a camera. This method is then extended to panning, tilting and zooming to maintain fixation on the target and preserve its size. Tordoff does not address the temporal stability of features between frames while zooming and suggests using multiscale feature detection as a possible area of future work. Problems in deciding the level of zoom as well as the point of fixation are not addressed.

By segmenting motion into foreground and background motion, Tordoff eliminates Hayman's need for restricting the search window to an area around the point of fixation. This results in a superior and more robust zoom tracking method. Tordoff's method is still limited to tracking a single target, although it is robust to partial occlusions. As with Hayman's work, the exact position of the target is uncertain as the exact corner features that are matched are not of importance, but rather the general foreground and background motion of the scene. The applications of this method are clearly for use in robotics to maintain fixation on a single target.

Xu[10] developed a system called Rits Eye that detects human faces based on skin colour and face geometry. A Gaussian model of skin in RGB space is used to detect and track faces. Once a face has been detected it is tracked with a PTZ camera so that the face remains towards the centre of the view. A simple constant velocity and constant acceleration model are used

to predict the position of the face from previous frames and adjust the camera pan and tilt to follow the target. Results show the tracker working in real time and tracking multiple faces or a single face.

In another approach, features of interest can be matched using scale invariant comparison methods. Comaniciu[128] suggests a kernel-based method which compares the probability distribution functions (pdfs) between target models and candidates in the image. The pdfs are generated by creating histograms of the colour spaces of an ellipsoid region of the image. A sample estimate of the Bhattacharyya Coefficient is calculated to find the distance between the model and candidate pdfs. The distance function gradient and the mean shift vector[129] are used to estimate the position of the target in subsequent frames and constrain the search region. Results are very good and allow the tracking of a target at 150fps on a 1GHz PC. Extensions to the algorithm are also presented were an extended Kalman filter is used to predict the structure of the kernel as well as the position of the target. Comaniciu's method shows good robustness to scale change and rotation as the intensity distribution of a region should remain fairly constant under these transformations. This method is a good alternative to conventional block matching techniques.

Mikolajcyk *et al*[130] provide a good review paper of several methods of *affine region detection*. Regions of the image around feature points of interest are transformed into features which are invariant to affine transformations. Most viewpoint changes results in a transformation called an *affinity*. An affinity is sufficient to locally model viewpoint changes when a scene surface can be locally approximated by a plane (or for a rotating camera) and perspective effects can be ignored as they are small for local changes. Five different detectors are presented: Harris-Affine and Hessian Affine detectors, edge based region detectors, intensity extrema-based region detectors, maximally stable extremal region detectors (MSER) and salient region detectors. Each of these detectors creates a region around either a feature point (such as a corner or edge) or an area of high intensity. These regions are then transformed such that they are invariant to affine transformations. It is then possible to match features between images where the scale or viewpoint has changed. Results are shown for changes in the viewpoint, scale, blur, varying JPEG artifacts and lighting changes. Generally, the Hessian-Affine or MSER detectors performed with the highest accuracy and repeatability. Although this paper addresses many methods of matching features between views, it does not address a method of tracking a specific target - however, these methods could be incorporated into a tracking framework as a

replacement for an algorithm such as normalised cross-correlation.

In some slightly less related work, Clarkson[131] investigates the registration of medical image data with 3D models, for example, registering imagery of a head with a 3D model of a skull (proprietary software is used to create the 3D models using surface reconstruction). A calibrated camera is used along with an estimation of the camera pose to texture map the imagery onto a plane that matches the 3D model. Features are then tracked in the texture mapped view using block matching. Results indicate that the use of texture mapping improves tracking accuracy and efficiency. Although this work is not directly comparable to the tracking work in this thesis, it does indicate the use of texture mapping and warping of images to assist and improve tracking algorithms.

Most tracking algorithms compare the tracking results to some sort of *ground truth*. Determining the ground truth is normally fairly tricky and is mostly done by a human operator hand-tracking the position of targets in a sequence of frames. Stolkin[132] proposes an interesting method of obtaining ground truth for a PTZ sequence. A camera is mounted on a robot arm, which moves the camera while viewing a calibration object. A calibration is then done using a method similar to Zhang[42, 99]. The calibration object is removed and the sequence is filmed with the same camera movement. From the calibration data and markers on objects it is possible to recreate the ground truth for the sequence. Unfortunately, this method is only useful when the scene can be filmed several times and requires large amounts of specialised robotics hardware. Vacchetti[133] uses a slight variation on this to improve tracking by estimating the pose of the camera using commercial software[1] and warping portions of a keyframe in a video stream to improve normal cross-correlation block matching methods.

Many of the tracking methods discussed have been the means to an end, such as maintaining the size of a target in the frame while zooming. Although there is no 'general' zoom tracking method, there have been many attempts and often they are designed using existing methods, such as region tracking or motion segmentation, and have been tailored for a specific application.

2.6 Applications of Pan/Tilt/Zoom tracking

As discussed before, the three main areas of application for PTZ tracking which will be examined in this thesis are robotics, media post production and surveillance. This section will outline some selected examples where objects have been tracked using PTZ tracking methods. Other examples have been chosen as they relate closely to PTZ tracking methods and there is a high crossover between the techniques used.

Many researchers are interested in tracking the players in a football match. The tracking data could be used to provide viewers with artificial views and sophisticated analysis of the game. Xu[134] tracks players on a football pitch by region or blob tracking. Players are classified according to the colour and type of their uniform. Up to eight cameras are used to track all the players and the resulting tracks are combined using data fusion. In a slightly more bizarre form of football, Borsato[135] describes a system used in *Robot Soccer*. Teams of robots play football against each other. This presents many interesting problems from the identification of other 'players' to motion segmentation and target tracking. Many of the robots used in Robot Soccer have PTZ heads or PTZ cameras which are used to follow the game and provide navigational cues.

Hawk-eye [2] is a system many viewers of tennis at Wimbledon will be familiar with: the system tracks the trajectory of the ball and is used to analyze shots or to determine if a ball was in or out of play. Tennis courts, like football pitches, are covered by a variety of cameras, some of which are fixed and others can pan, tilt and zoom. Hawk-eye slowly builds up a calibration between cameras by estimating the position of the lines of the court in each view and finding correspondences. During the match, the ball is tracked in multiple views and a 3D model of the trajectory is built up by fusing all the tracking information or 'tracklets'. The Hawk-eye system is a good example of the fusion between graphics and image processing: targets are tracked in several views with moving cameras and the data is correlated and presented to the viewer as rendered 3D models. Hawk-eye is also used in cricket matches to analyze bowling and to determine if batsman are out LBW by estimating the trajectory of the ball past the batsman's legs. This type of technology will become increasingly popular in television sports broadcasting as it adds a valuable extra dimension to watching a sport on television.

In surveillance applications several PTZ tracking systems have been investigated [3, 11, 13, 15, 50]. These systems mainly use motion segmentation and high level human stick models to

track people using PTZ cameras and multiple views from distributed camera systems. A recent example of this is work by Scotti[136] where a 360° camera is used in conjunction with a PTZ camera. The 360° camera detects motion and positions the PTZ camera to obtain high resolution views of the target. In a different surveillance scenario Canals[137] tracks targets using a block matching method from a flying surveillance drone using a pan/tilt camera.

2.7 Hardware Overview

All of the hardware used to obtain the results presented in this thesis is commonly available consumer electronics hardware. A fairly powerful dual processor 1.8GHz Intel Xeon desktop computer was used to develop and test all of the algorithms presented. Sequences were captured using the Sony EVI-D31 (shown in Figure 2.5). The analogue composite video signal was digitised by an Imagenation PXC200A capture card. The Sony EVI-D31 can be controlled using an RS-232 serial interface and Sony's VISCA protocol. The details of the software drivers developed and used can be found in Appendix D.



Figure 2.5: *Sony EVI-D31*

A consumer electronics PTZ camera was used mainly as it was available and didn't require the development of sophisticated robotics hardware and control. The hardware used is relatively inexpensive compared to the active vision heads used by Tordoff[4, 8, 9] and Hayman[5–7] for example.

The Sony EVI-D31 does however suffer from some serious limitations: the pan/tilt/zoom position returned by the camera is transmitted at 9600 bps on a half-duplex RS-232 link. There

is significant lag between receiving the position data packet and the issuing the information request packet. This results in position data lagging behind the true position of the camera. Position data cannot be streamed from the camera and needs to be polled. It is also not possible to obtain the pan, tilt and zoom positions simultaneously – two requests need to be made (one for pan/tilt and the other for the zoom position). In an active vision robotics head, the position output of the camera will be a signal which is digitised by the position controller. It is possible to obtain an extremely fast position sampling rate which the EVI-D31 would never be capable of. The documentation for the VISCA protocol (see Appendix D) and accurate timing diagrams for the EVI-D31 are very limited. This shortfall in the documentation and accessibility to technical details of the camera were a tradeoff against the cost of the hardware.

The EVI-D31 was never designed with real-time robotics in mind but rather for a human operator to control the position of the camera using a joystick unit. Therefore, the response time is not critical. In order to be certain of the accuracy of the pan/tilt/zoom position information a sacrifice to speed needed to be made. Before a frame was captured, the camera was allowed to settle and the pan/tilt/zoom positions were queried from the camera. This took just under two seconds on average, resulting in an extremely poor frame rate. Given that the algorithms presented in this thesis were tested offline, this is not critical. However, for a real-time implementation, the Sony EVI-D31 would not be suitable. Other PTZ cameras used in CCTV surveillance appear to have RS-232 and RS-422 serial control, however the specifications of the control protocols and timings are not freely available. It is possible that other hardware would suffer the same lag as the Sony EVI-D31.

A positive aspect of the Sony EVI-D31 is that is it (or similar Sony hardware) has been used in several previous publications on PTZ tracking and calibration[3, 15, 28, 50, 114, 116]. There are also several open source drivers for controlling the camera via the serial interface. This is due to the EVI-D31 being a popular choice as a webcam where users can control the direction of view and the zoom level.

2.8 Summary

As discussed, many methods exist to track people. The most common method is blob tracking by interframe differencing and maintaining a stable background or colour segmentation to remove objects with certain colour profiles (for example skin tones) from the background. Colour segmentation handles slow object changes and suffers failure when static objects have similar colour profiles to the object of interest, for example skin coloured walls when tracking people. More sophisticated methods such as eigenshape matching and silhouettes can be used. These are normally computationally expensive and can only be used to track one or two people or when the template consists of few pixels. With the increased number of CCTV installations multicamera (or multiview) tracking is becoming more popular for security applications. Large bodies of work exist to handle the handoff between cameras and tracking the same object from several different view points. Multiview problems can be extended to include a pan-tilt-zoom camera to obtain high resolution images of people that could be used with face recognition. Zoom tracking is then necessary to keep the target in the frame as they move about. Currently most scale invariant tracking methods use corner feature matching, adaptive colour profiles to track or simplistic template update schemes with block tracking. Most of the methods discussed above have been developed and tested in controlled conditions (i.e. constant illumination and uncomplicated scenes). There is an opportunity to investigate the performance of these methods under real-world conditions and offer a more robust method of person tracking in real-world scenarios.

Camera calibration is essential to any PTZ tracking system either as *a priori* knowledge or as on-the-fly homography estimation between camera views. Many methods of camera calibration with a specific focus on calibrating PTZ cameras were presented and analyzed.

A review and discussion of existing zoom tracking methods was presented. Applications of PTZ tracking systems were outlined in the areas of robotics, media post production and surveillance.

The PTZ camera hardware used to obtain the results presented in this thesis were discussed. Limitations of the hardware are also presented. Detailed information on the hardware and software used for this work can be found in Appendix D.

The background material developed in this chapter will be built on in the following chapter, which brings together concepts from computer graphics, projective geometry and image processing necessary to understand the tracking algorithms presented in Chapter 4.

Chapter 3
Camera Models

3.1 Introduction

One of the aims of this thesis is to bring together concepts from projective geometry, image processing and computer graphics. Each of these areas deals with the relation between 2D and 3D geometry in a specific way. This results in a large amount of notation and terminology. The purpose of this chapter is to review the principles of viewing a 3D scene with a pin-hole camera in the context of implementing the system on a 3D graphics system such as OpenGL[23].

Many good texts on computer graphics exist, such as books by Foley[35, 36], which introduce 2D and 3D computer graphics concepts. Both of these texts were written before the days of accelerated 3D graphics hardware and the acceptance of OpenGL as the *de facto* standard for rendering on desktop computers. Although other books[138–140] on modern computer graphics exist, they tend to approach the problem for the point of view of coding a computer graphics application. The projective geometry involved is explained for computer programmers and is not particularly rigourous or detailed.

The approach taken by Hartley and Zisserman[34], or Faugeras[31] in particular, is to establish a theoretical basis for projective geometry from first principles. The goals of these texts is vastly different from the graphics texts. They are primarily concerned with establishing a sound mathematical basis for projective geometry. This makes them fairly unaccessible to readers from a computer graphics background and the connections between projective geometry and computer graphics are very unclear.

In order to fully understand the technical aspects and design choices of the pan/tilt/zoom tracking algorithms presented in Chapter 4 it is first necessary to understand concepts from both disciplines and their relation to each other. This chapter introduces both subjects in the context of tracking under rotation and scale change, as one would expect with a PTZ camera.

This chapter deals with the various co-ordinate systems and the transformation of points from one co-ordinates system or view into another. The effect of panning, tilting and zooming (or

general rotation and scale change in 3D) on 2D projections of scenes is analyzed visually to give the reader a sense of the deformation of blocks of pixels under pan/tilt/zoom conditions. A model for a pin-hole camera that is common to both computer graphics and computer vision is developed and reviewed. Texture mapping and its application to computer vision are also briefly discussed. Finally, all of these concepts are woven together to provide a model of a PTZ camera that is consistent with principles of projective geometry which can be implemented in a computer graphics system such as OpenGL.

3.2 Co-ordinate Systems

In 2D image processing any point on an image can be described as an (x, y) pixel co-ordinate. A point in this form is known as an *inhomogeneous* point. Also, any (x, y) pixel point can be represented by the *homogenous* point $(x, y, w)^\mathsf{T}$, where w is the relative depth of the pixel co-ordinate to the camera. As all 2D image processing occurs in the same plane, w is normally set to 1. By using the homogenous form of representing points, the resulting points from 2D transformations such as scaling and rotation can be guaranteed to lie in the same image plane by dividing through by w to make it equal to one. The representation of homogenous 3D points is much the same as for 2D points. Any point in 3D space can be represented by a four-vector, $(x, y, z, w)^\mathsf{T}$. Again, by fixing w to 1, all 3D points are guaranteed to lie within the same volume in the same way that 2D points are constrained to a plane. A point of this type is often said to be represented in homogeneous co-ordinates. Homogenous co-ordinates also allow the projection process to be split into a series of linear matrix operations and a non-linear projection, which is effected by dividing through by w. Throughout this thesis several co-ordinate systems will be used to specify points depending on the context of their use:

World space (3D)	Reference co-ordinates for all other systems. Analogous to the real world.
Camera view space (3D)	World space co-ordinates viewed for the camera position in the direction of gaze.
Viewing frustum (3D)	Normalised projected co-ordinates on the image plane that still contains depth information.
Image space (2D)	Standard 2D image co-ordinates with no depth information.

3.2.1 World Space

Any real object can be represented as a set of points in 3 dimensional space. These points are represented in 4D homogenous co-ordinates as $(x, y, z, 1)^{\mathsf{T}}$. The unit vectors $\hat{x}$, $\hat{y}$ and $\hat{z}$ define the x, y and z axes respectively. The physical significance of world space co-ordinates is fairly arbitrary. Often in the literature it is said that any point in 3D space can be determined up to a scale factor. This means that any point, if measured in pixels, can be converted to metres with the appropriate scaling transform, which takes the form of a transform matrix, as the scaling in each dimension could be different.

Throughout this thesis, points are referred to as lying in world space co-ordinates. All world space co-ordinates from real data will be measured in pixels. None of the applications in this thesis require object positions to have significant physical parameters. However, in some situations such as adding 3D models that interact with areas of the image using a full kinematics model, true positions in metres would be required. The conversion of these pixel values into physical measurements such as metres would require a further calibration step that relates the length of an object in metres to its pixel length.

3.2.2 Camera View Space

Camera view space is defined as a view of world space from the *eye* point, in the direction of the *look at* point. The relationship between world space and camera view space is defined by a 4×4 transformation matrix, which consists of a rotation and translation.

The image plane defines the unit vectors $\hat{u}$, $\hat{v}$ and $\hat{n}$ as shown in Figure 3.1 with the origin of the co-ordinate system being at the camera point. Orthogonal vectors $\hat{u}$ and $\hat{v}$ lie in the image plane and $\hat{n}$ is normal to the image plane.

3.2.3 Viewing Frustum

Most real world cameras have a finite viewing angle which allows only a conical section of the world to be imaged at a time. This conic view of the world can be represented by a *viewing frustum*. Eric Weisstein[141] defines a frustum as the portion of a solid which lies between two parallel planes cutting the solid. The viewing frustum is defined as the volume inside the intersection of a pyramid with two parallel planes, as shown in Figure 3.2. The plane nearest

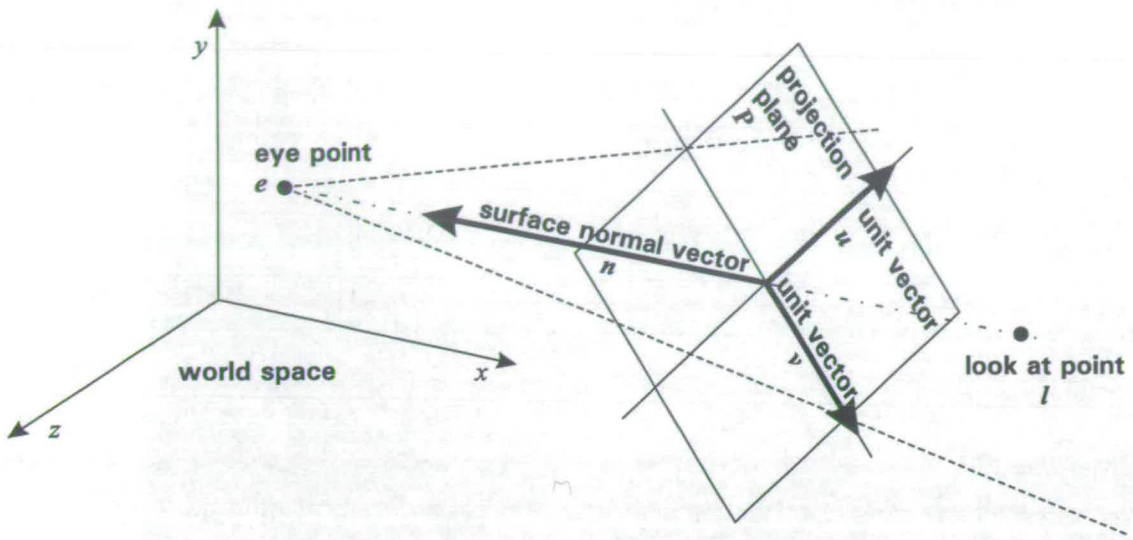


Figure 3.1: *Camera view space*

the camera is used to model the image plane and the furthest plane represents the distance away from the camera that will be treated as infinity. This simplistic model does not allow for effects such as depth-of-field and assumes a perfect pinhole camera.

The view volume defines six planes: the front and back clipping planes and the four side clipping planes. The front clipping plane and the image plane are equivalent. The shape of the viewing volume is determined by the type of camera projection used. For example, an orthogonal projection would produce two pairs of parallel side clipping planes, whereas the perspective transform produces planes that form a pyramid. Throughout this thesis only perspective projections will be considered as real-world cameras are perspective cameras.

The clipping planes in the viewing frustum are defined such that:

$$u_{min} \leq u \leq u_{max}$$

$$v_{min} \leq v \leq v_{max}$$

$$front \leq n \leq back$$

It is convention to normalise the viewing frustum, such that the image plane is located at the origin and anything inside the viewing volume lies within one unit of the camera. The viewable

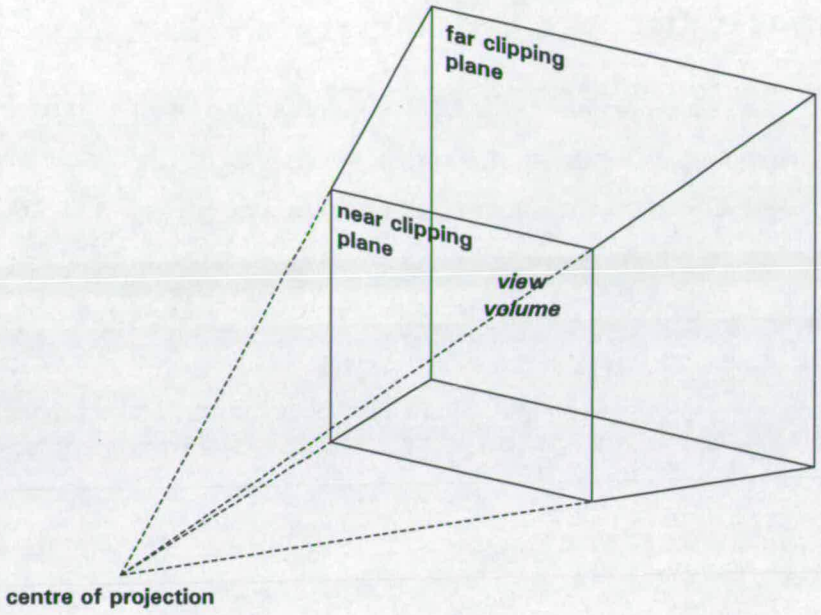


Figure 3.2: *Viewing frustum*

limits of the image plane are one unit in each direction:

$$-1 \leq u \leq 1$$
$$-1 \leq v \leq 1$$
$$0 \leq n \leq 1$$

3.2.4 Image Space

Normally, points in images are defined as the integer distance in pixels from the top left corner of the image with increasing x values to the right and increasing y values down. As discussed in Section 3.2, points in an image can also be represented in homogeneous form as $(x, y, w)^\mathsf{T}$. This idea can be taken further and the image can be considered to lie in 3D world space. Therefore, any pixel position on the image can be represented by the homogenous point $(x, y, z, w)^\mathsf{T}$. The extra dimension, z can be interpreted as the depth of the pixel relative to the camera position. When the camera view is defined as looking directly down the z axis and the image plane is aligned with the x-y plane one unit from the origin, image pixel co-ordinates can be represented as $(x, y, 1, 1)^\mathsf{T}$.

3.3 Viewing in 2D

Given a pair of images it is possible, under certain conditions, to relate points from one image to the other image by a projective transform or *homography*. Hartley and Zisserman[34] discuss the various conditions under which this projective transform exists, but for the most part, a projective transform exists if each set of points from two images are coplanar in projective space. From this it is possible to relate pairs of images to other pairs by epipolar geometry and resolve the Euclidean structure of a scene. It also leads to the self-calibration of cameras and allows points to be related between multiple images. A homography is a projective transform that relates points from one image to another:

$$\mathbf{x}' = \mathtt{H}\mathbf{x}$$
$$\begin{pmatrix} x' \\ y' \\ w' \end{pmatrix} = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix} \begin{pmatrix} x \\ y \\ w \end{pmatrix} \tag{3.1}$$

where $\mathbf{x}$ homogenous point in image 1
$\mathbf{x}'$ homogenous point in image 2
$\mathtt{H}$ 3×3 homography matrix with elements $h_1, h_2, \ldots, h_9$

Homographies are generally used to represent various 2D transformations, such as scale changes, translation, rotation and skews. Other perspective effects are created by combining these various transforms. A method of finding the homography between two images is discussed in Section 6.4.4 on page 130. Further material on projective geometry and 2D transforms can be found in the following sources: Foley[35, 36] discusses the 2D transform of points in the context of computer graphics and the rendering of 2D shapes, before expanding this to 3D graphics. Faugeras[31] and Hartley and Zisserman[34] both take the projective geometry approach to relating images from multiple camera views and include discussions on techniques from camera calibration to structure from motion. Both texts develop a good theoretical basis for projective geometry with many examples and applications.

3.4 Viewing in 3D

With 3D scenes, the transforms are much the same as for 2D viewing as discussed in Section 3.3. A projective transform relates 3D points from one view to another as follows:

$$\mathbf{x}' = \mathtt{P}\mathbf{x}$$

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{bmatrix} p_1 & p_2 & p_3 & p_4 \\ p_5 & p_6 & p_7 & p_8 \\ p_9 & p_{10} & p_{11} & p_{12} \\ p_{13} & p_{14} & p_{15} & p_{16} \end{bmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} \tag{3.2}$$

where $\mathbf{x}$ homogenous point in view 1

$\mathbf{x}'$ homogenous point in view 2

$\mathtt{P}$ 4×4 projection matrix with elements $p_1, p_2, \ldots, p_{16}$

The projective transform can be constrained to have less degrees of freedom by forcing it to have affine properties. Under affine transformations, multiple views can be related to each other via the *plane at infinity* and the *absolute conic*. An affine projective transform has the form:

$$\mathbf{x}' = \mathtt{P}\mathbf{x}$$

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{bmatrix} a_1 & a_2 & a_3 & t_1 \\ a_4 & a_5 & a_6 & t_2 \\ a_7 & a_8 & a_9 & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} \tag{3.3}$$

where $\mathbf{x}$ homogenous 3D point 1

$\mathbf{x}'$ homogenous 3D point 2

$\mathtt{P}$ 4×4 affine projection matrix

$\mathbf{A}$ affine transform matrix, such as rotation or scale change, with elements $a_1, a_2, \ldots, a_9$

$\mathbf{t}$ translation vector with components t_1, t_2, t_3

A detailed discussion of projective geometry is beyond the scope of this thesis, although some concepts have been discussed briefly. For further reading, consult Faugeras's *The Geometry of Multiple Images*[31] or Hartley and Zisserman's *Multiple View Geometry in Computer Vision*[34].

The plane at infinity is used to ensure that a projective transform exists between two camera views. The Kruppa equations were rediscovered[1] by Faugeras and Luong[92] for use in camera calibration and auto-calibration. The absolute conic and *dual image of the absolute conic* are useful in finding the *intrinsic parameters* of a camera from point correspondences in multiple views. For the most part, a computer graphics approach to perspective transforms will be used rather than the projective geometry approach, in an attempt to demystify the methods that get lost in a quagmire of notation. Some assumptions are made that reduce the generality of the perspective transforms: only affine viewing conditions are considered and all perspective transformation matrices are non-singular and invertible. This implies that all cameras are calibrated (and can be calibrated) and points can be related to other points between all views, without the need to triangulate them using stereo vision methods or epipolar geometry.

3.4.1 Transformation Matrices

Homogenous points in world space can be transformed using 4×4 transformation matrices, also known as *projectivities*. Some standard transform matrices for translation, rotating and scaling are listed below:

Translation by vector $\mathbf{v} = (v_x, v_y, v_z, 0)^\mathsf{T}$:

$$\mathbf{T}(\mathbf{v}) = \begin{bmatrix} 1 & 0 & 0 & v_x \\ 0 & 1 & 0 & v_y \\ 0 & 0 & 1 & v_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

[1] Faugeras named the Kruppa equations after work done in projective geometry by Kruppa[93] in 1913. The Kruppa equations relate the image of the absolute conic to the epipolar transformation. Faugeras insists on using a method from 1869 by Rudolf Sturm[95] to find the epipoles. A more rigorous discussion of the absolute conic, epipolar geometry and the Kruppa equations can be found in Faugeras's book *The Geometry of Multiple Images* [31]. There are also several later derivations of the Kruppa equations by Xu[142] and Hartley[98].

A rotation of θ around the x-axis:

$$\mathbf{R}(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Scaling by vector $\mathbf{s} = (s_x, s_y, s_z, 0)^\mathsf{T}$:

$$\mathbf{S}(\mathbf{s}) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Transformation matrices can be multiplied together to produce a single matrix that performs the entire transform. The order of multiplication is important as transformation operations are not always commutable. The right-hand most transformation matrix is applied first and the left-hand most transformation matrix last. An example of a transformation matrix $\mathbf{M}$ that translates points by a and then scales them by b is given by:

$$\mathbf{M} = \mathbf{S}(\mathbf{b})\mathbf{T}(\mathbf{a}) = \begin{bmatrix} b_x & 0 & 0 & 0 \\ 0 & b_y & 0 & 0 \\ 0 & 0 & b_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_x \\ 0 & 1 & 0 & a_y \\ 0 & 0 & 1 & a_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} b_x & 0 & 0 & a_x b_x \\ 0 & b_y & 0 & a_y b_y \\ 0 & 0 & b_z & a_z b_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Given a point $\mathbf{p} = (p_x, p_y, p_z, 1)^\mathsf{T}$ the transformed point $\mathbf{p}'$ is given by:

$$\mathbf{p}' = \mathbf{M}\mathbf{p} = \begin{bmatrix} b_x & 0 & 0 & 0 \\ 0 & b_y & 0 & 0 \\ 0 & 0 & b_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_x \\ 0 & 1 & 0 & a_y \\ 0 & 0 & 1 & a_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix} = \begin{pmatrix} p'_x \\ p'_y \\ p'_z \\ 1 \end{pmatrix}$$

The inverse transformation can be found by inverting[2] the matrix, giving $\mathbf{M}^{-1}$. Multiplying transformed points by the inverse matrix gives the original points: $\mathbf{M}^{-1}\mathbf{p}' = \mathbf{p}$.

[2] If the inverse of the matrix cannot be found, then the projective transform matrix is not linear. This means that information has been lost in the transformation by combining dimensions that cannot be separated. Certain forms of the perspective projection matrix are *not* invertible.

3.5 OpenGL-style Rendering Pipeline

The process of transforming a 3D point into a 2D point on an image can be divided into three transformation matrices as shown in Figure 3.3. This formulation is used in OpenGL and DirectX. Each portion of the transformation chain will be discussed in the sections below.

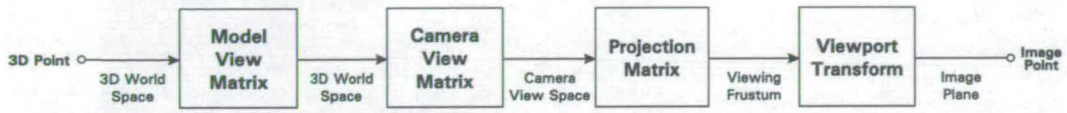


Figure 3.3: *Rendering pipeline*

The model view matrix modifies the orientation and scale of 3D object models. The camera view matrix rotates and translates points from world space into the camera view space. The projection matrix projects the transformed points into the viewing frustum. The viewport transform, transfers points from the viewing frustum onto the display device, such as a computer monitor.

If the transform matrices are invertible then it is possible to reverse the projection and rendering process, however this is not always possible. Some formulations of the perspective transform are not linear, as depth information is transformed into shifts in the x and y directions. It is not possible to recover the depth information from perspective projected points. With more than one view of an object it is possible to triangulate the depth as is done in many stereo vision problems[31, 34]. All transformation matrices used in this thesis are invertible, and under pan, tilt and zoom conditions it is possible to transform points between views without ambiguity. These transformation matrices and concepts will be developed throughout this chapter.

The model view matrix represents the transformation of an object relative to the camera view and is intended to be used in OpenGL to animate 3D models of objects. For example, a ballet dancer spinning on a point would be animated by applying successive rotation matrices to the model view matrix. Whereas, a camera panning over a static object should modify the projection matrix.

The projection matrix projects 3D points into the viewing frustum (described in Section 3.2.3)

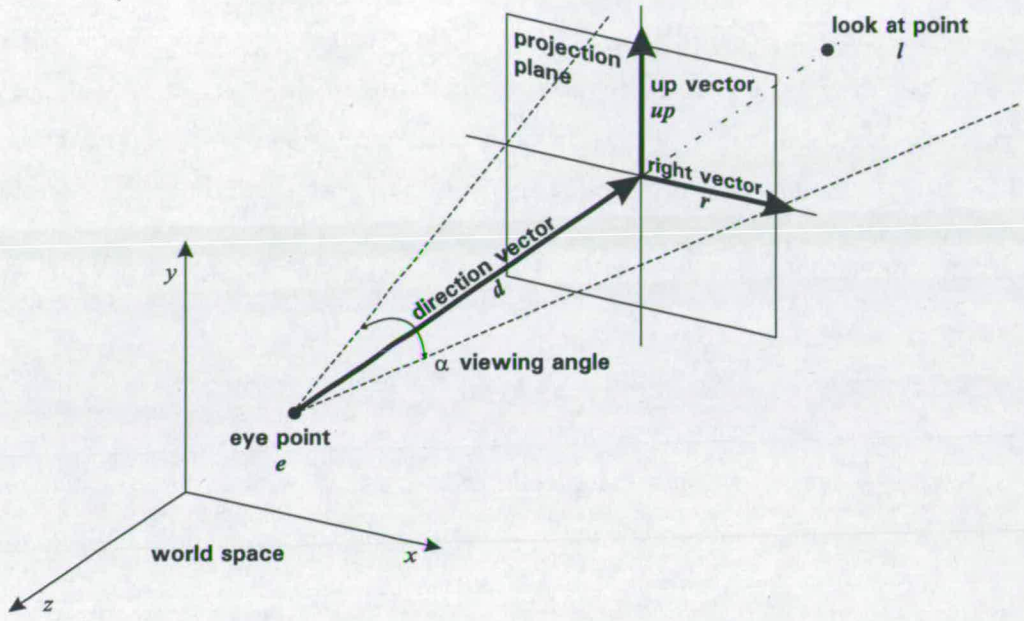


Figure 3.4: *Describing a perspective camera*

and implicitly onto the image plane. Sometimes the projection matrix is non-singular, such the perspective projection defined by Foley[35, 36]. This results in transformed points having a w component that is not equal to one. By dividing the other vector components through by w, the perspective projection is effected. However, by using a perspective transformation of this form (i.e. a matrix which is not invertible) the projection process is not linear and points cannot be easily transformed between world co-ordinates and image co-ordinates using matrix algebra. Therefore, the projection matrix should always be chosen so that it is invertible.

Most projection matrices are defined so that any points are transformed into the canonical viewing frustum, i.e. they lie within one unit of the camera in all directions. This allows for an easy transformation into the viewport. The *viewport* in a graphics system could be anything from a computer monitor to a printer. Therefore, a linear transform is needed that changes world points into physical display co-ordinates and vice versa. This transform is defined as the *viewport transform*.

3.5.1 Describing a Camera

As there are several methods for describing cameras, a variety of parameter specifications and nomenclature exists: the graphics system PHIGS, described by Foley et al[35, 36], uses a pro-

jection referenced system whereas other newer systems such as OpenGL[23], DirectX[24] and Povray[40], use a camera-centric system. It is possible to create a 4×4 transform matrix that equates the two systems. It is confusing to have two fairly different formulations of the same projection process, therefore the OpenGL style of camera description will be used throughout this thesis as it is currently used by all computer graphics hardware and software manufacturers.

Five vectors describe the position, orientation and zoom of the camera (as shown in Figure 3.4):

Eye position	The position vector of the camera or the position from which the scene is viewed.
Look at point	Defines the direction of view that starts at the *eye* point and ends at the *look at* point (sometimes called the point of fixation).
Direction vector	Specifies the orientation of the projection plane. Usually in the same direction as the *look vector*.
Up vector	A unit vector which controls the roll of the camera. Normally this vector points in the direction of the sky.
Right vector	Controls the handedness of the co-ordinate system and is used in the calculation of the aspect ratio.

The *eye position* is defined by the point $\mathbf{e}$ and specifies the camera's location. It is the start point of the *look* vector and determines the point from which the scene will be viewed. Translation is achieved by moving the *eye position*. Zooming effects are created by moving the *eye position* closer to the subject (however camera zoom should not be implemented in this way but rather using the *direction* vector).

The *look* vector $\mathbf{l}$ starts at the *eye position* point and ends at the *look at* point. This vector describes the orientation of the view or the direction of gaze. Moving the *look at* point creates effects such as pan and tilt. It does not affect the apparent zoom as its length is irrelevant (it becomes a unit vector in all calculations).

The *direction* vector $\mathbf{d}$ is the vector between the *eye position* and the near clipping plane (or projection plane). The zoom factor is proportional to the length of the *direction* vector. The *direction* vector is usually parallel to the projection plane normal. Note that by shortening the length of the direction vector the viewing angle increases producing a more wide angle view.

The *up* vector **up** defines the rotation of the camera around the z-axis. The *up* vector is defined by default as $(0, 1, 0, 0)^{\mathsf{T}}$ which gives a camera level with the horizon pointing down the z-axis. The *right* vector **right** defines whether the co-ordinate system is right-handed or left-handed. In a left-handed co-ordinate system the z-axis points out of the viewport in the direction of $\mathbf{d}$. A negative *right* vector defines a left-handed co-ordinate system. Throughout this thesis all 3D co-ordinates are specified in a left-handed co-ordinate system.

3.5.2 Camera View Matrix

The purpose of the camera view matrix is to transform world space points into camera view space so that the points appear as if they were viewed from the camera position. This is done by translating the eye position to the origin and rotating all the points so that the view axes (u, v and n) correspond with the world space axes (x, y and z). The camera view matrix is determined from the parameters described in Section 3.5.1. This method of describing the camera is useful as the camera view direction can be specified to look at a specific point, rather than choosing rotation matrices until the camera view is correct. This formulation of the camera view matrix is not useful for real cameras – a series of rotation and translation matrices should be used instead. The derivation of this camera transform matrix is shown below.

Using the *eye* position $\mathbf{e}$ the following matrix translates the centre of projection to the origin:

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & -e_x \\ 0 & 1 & 0 & -e_y \\ 0 & 0 & 1 & -e_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.4}$$

The projection plane can be defined by the *look* vector and the *up* vector. Three unit vectors ($\hat{u}$, $\hat{v}$ and $\hat{n}$) can be determined from these that define the projection view. The unit projection plane normal is given by:

$$\hat{n} = -\hat{l} = \frac{\mathbf{e} - \mathbf{c}}{\|\mathbf{e} - \mathbf{c}\|} \tag{3.5}$$

where $\hat{l}$ unit *look* vector
$\mathbf{e}$ eye position
$\mathbf{c}$ centre of projection

Using the unit *up* vector ($\hat{up} = \frac{up}{\|up\|}$) expressions for $\hat{u}$ and $\hat{v}$ can be found:

$$\hat{u} = \frac{\hat{up} - \hat{l}(\hat{l} \cdot \hat{up})}{\|\hat{up} - \hat{l}(\hat{l} \cdot \hat{up})\|} \tag{3.6}$$

$$\hat{v} = \frac{\hat{l} \times \hat{u}}{\|\hat{l} \times \hat{u}\|} \tag{3.7}$$

Once at the origin, world space is rotated so that the u, v and n axes correspond with the x, y and z axes.

$$\mathbf{R} = \begin{bmatrix} \hat{u_x} & \hat{u_y} & \hat{u_z} & 0 \\ \hat{v_x} & \hat{v_y} & \hat{v_z} & 0 \\ \hat{n_x} & \hat{n_y} & \hat{n_z} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.8}$$

The complete camera view matrix is given by:

$$\mathbf{M} = \mathbf{RT} = \begin{bmatrix} \hat{u_x} & \hat{u_y} & \hat{u_z} & 0 \\ \hat{v_x} & \hat{v_y} & \hat{v_z} & 0 \\ \hat{n_x} & \hat{n_y} & \hat{n_z} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -e_x \\ 0 & 1 & 0 & -e_y \\ 0 & 0 & 1 & -e_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.9}$$

3.6 Perspective Camera Model

The following section describes the transformation matrices and viewport transforms needed to model a perspective camera. The camera is assumed to be a perfect pin-hole camera with radial distortion (radial distortion effects are discussed in Section 3.10.1).

3.6.1 Perspective Projection

An invertible form of the perspective transform P can be specified by the following parameters that create the six planes of the viewing frustum shown in Figure 3.2:

$$\mathtt{P} = \begin{bmatrix} \frac{2near}{right-left} & 0 & \frac{right+left}{right-left} & 0 \\ 0 & \frac{2near}{top-bottom} & \frac{top+bottom}{top-bottom} & 0 \\ 0 & 0 & -\frac{far+near}{far-near} & -\frac{2(far)(near)}{far-near} \\ 0 & 0 & -1 & 0 \end{bmatrix} \tag{3.10}$$

where

$near$	z co-ordinate of the near clipping plane (equivalent to the magnitude of the *direction* vector)
far	z co-ordinate of the far clipping plane (a large value $\approx 10\,000$)
$left$	x co-ordinate of the left clipping plane
$right$	x co-ordinate of the right clipping plane
top	y co-ordinate of the top clipping plane
$bottom$	y co-ordinate of the bottom clipping plane

The w component of the transformed point is sometimes not equal to one after multiplication by matrix P. Dividing the components of the vector by w forces w to be equal to one and effects the transform.

An alternative formulation of the perspective transform is given in terms of the vertical field of view (α) and the aspect ratio (a) of the horizontal and vertical viewing angles:

$$\mathtt{P} = \begin{bmatrix} \frac{1}{a}\cot\frac{\alpha}{2} & 0 & 0 & 0 \\ 0 & \cot\frac{\alpha}{2} & 0 & 0 \\ 0 & 0 & \frac{far+near}{near-far} & \frac{2(far)(near)}{near-far} \\ 0 & 0 & -1 & 0 \end{bmatrix} \tag{3.11}$$

3.6.2 Viewport Transform

The projection matrix described above projects 3D points into the viewing frustum so that they lie in the following range:

$$u_{min} \leq u \leq u_{max}$$
$$v_{min} \leq v \leq v_{max}$$
$$front \leq n \leq back$$

If the viewing frustum is canonical then $-1 \leq u \leq 1$, $-1 \leq v \leq 1$ and $0 \leq n \leq 1$. If the centre of projection is the centre of viewing volume, then the viewport transform can be defined as:

$$\begin{aligned} x' &= x_{win} + width(\frac{x+1}{2}) \\ y' &= y_{win} + height(\frac{y+1}{2}) \\ z' &= \frac{z+1}{2} \end{aligned} \tag{3.12}$$

where	$\mathbf{x}$	point in view frustum with components $(x, y, z)^\mathsf{T}$
	$\mathbf{x}'$	transformed point on viewport plane with components $(x', y', z')^\mathsf{T}$
	x_{win}	viewport x position in pixels (normally 0)
	y_{win}	viewport y position in pixels (normally 0)
	$width$	viewport width in pixels
	$height$	viewport height in pixels

This can also be represented in matrix form as a series of translations $\mathbf{T}(\cdot)$ and scale changes $\mathbf{S}(\cdot)$:

$$\mathbf{x}' = \mathbf{T}(x_{win}, y_{win}, 0) \quad \mathbf{S}(\frac{width}{2}, \frac{height}{2}, \frac{1}{2}) \quad \mathbf{T}(1, 1, 1)\mathbf{x} \tag{3.13}$$

If the centre of the viewport is not the centre of the projection, Foley[35, 36] contains the generalised form of the viewport transform above. However, for most computer graphics packages and the methods contained in the thesis, the camera will be assumed to be aligned with the centre of the viewport.

After applying this transform visible points will lie in the range $0 < x' < width$ and $0 < y' < height$. Any points outside of this range are not visible and are said to have been 'clipped against the viewport rectangle'. By default, most graphics packages also clip objects according to their depth, therefore anything in front of the $near$ plane or behind the far plane will not be visible. By dividing through by w', the perspective projection is effected. The true depth of all of the vertices in the scene are transformed into the depth relative to the image plane as the z co-ordinate. It is no longer possible to extract the true depth information from the image plane. However, when the camera is only able to pan, tilt and zoom, it is possible to place the points at the depth of the image plane and treat them as if the true depth was known. This does not hold if the camera translates.

3.6.3 Inverse Perspective Projection

The inverse perspective projection projects a 2D image point into 3D world space. As the depth information is lost in the perspective transform a depth needs to be specified as a parameter as any value between the near and far clipping plane. As the viewing frustum is normalised this is value between 0 and 1, corresponding to the near and far clipping planes respectively. If the *eye* position has not moved then the selection of the depth is arbitrary, however if the camera has translated then careful consideration needs to be taken to avoid parallax errors. With *a priori* knowledge of the translation the inverse projection becomes a stereo vision problem. However, in real PTZ cameras there is negligible translation, so these effects can be ignored. Therefore the depth will always be selected as 0 to place the inverse projected points on the front clipping plane. When transforming points between views, the depth information is known and should not be discarded. Only when points are extracted from images, such as 2D tracking start positions, should the depth be set to zero.

The inverse viewport transform relates a point from the viewport into the canonical viewing frustum by:

$$\begin{aligned} x &= \frac{-2x'}{width} - 1 \\ y &= \frac{-2y'}{height} - 1 \\ z &= 2z' - 1 \end{aligned} \tag{3.14}$$

Again, this can be represented in matrix form as a series of translations $\mathbf{T}(\cdot)$ and scale changes $\mathbf{S}(\cdot)$:

$$\mathbf{x} = \mathbf{T}(-1,-1,-1) \quad \mathbf{S}(\frac{-2}{width}, \frac{-2}{height}, 2)\mathbf{x}' \tag{3.15}$$

In order to find the co-ordinate of the point in 3D world space the projection and camera transform need to be reversed, so the entire inverse perspective transform from 2D images point to 3D world co-ordinates is given by:

$$\begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = [\text{(projection matrix)(camera transform)}]^{-1} \begin{bmatrix} \frac{-2x'}{width} - 1 \\ \frac{-2y'}{height} - 1 \\ 2z' - 1 \\ 1 \end{bmatrix}$$

And using the symbols for the projection and camera transform matrices defined previously:

$$\begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = [\mathbf{PRT}]^{-1} \begin{bmatrix} \frac{-2x'}{width} - 1 \\ \frac{-2y'}{height} - 1 \\ 2z' - 1 \\ 1 \end{bmatrix}$$

Again the w component of the resultant point may not be equal to one and hence the vector needs to be divided by w to effect the transform and ensure co-ordinate consistency.

3.6.4 Povray Perspective Camera

The Povray[40] camera model is very similar to the perspective camera model described so far. The only difference is in the bounds of the viewing frustum. Instead of $-1 \leq u \leq 1$ and $-1 \leq v \leq 1$, the bounds are $-\frac{1}{2} \leq u \leq \frac{1}{2}$ and $-\frac{1}{2} \leq v \leq \frac{1}{2}$ as shown in Figure 3.5. This difference in the viewing frustum can be accommodated by setting the clipping planes in Equation 3.10 to $\frac{1}{2}$, with the front clipping plane located at the magnitude of the direction vector. After this, the viewport transform of Equation 3.12 will behave as before, as the perspective transform will project points into the canonical viewing frustum.

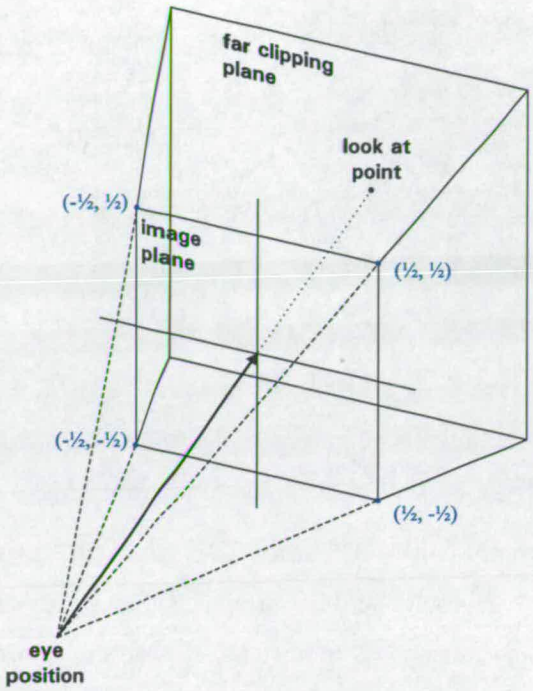


Figure 3.5: *Povray Camera*

3.7 Texture Mapping

Texture mapping is a term from computer graphics that describes the process of taking an image and 'stretching' it across the 3D model of an object, as is shown in Figure 3.6. This technique is used in computer games and special effects to make simple 3D models look lifelike, by taking real images of objects and skinning them onto 3D skeleton models of the object in a 3D graphics package. Transformation matrices are used to manipulate the size and orientation of the texture on the 3D model. The methods of texture mapping images onto objects are discussed thoroughly in both Foley[35, 36] and the OpenGL Red Book[23].

Given a scene in world space, an image is the projection of the scene onto a plane. Conversely, given an image, the scene can be visualised as the image data texture mapped onto a plane (at the position of the projection plane the image was originally capture on). This process is shown in Figure 3.7. It is then possible to move the camera and create an artificial view of the scene, based only on the image data. An example of this is a virtual reality viewer for panoramic scenes such as QuickTimeVR or VRML, where there is an interactive 360° view of a scene. This is achieved by taking a spherical image[3] and projecting it onto a plane to match the user's

[3]More correctly, the 'spherical image' is a spherical projection of many images that have been mosaiced together into a single panoramic image.



Figure 3.6: *Texture Mapping*

desired viewpoint as shown in Figure 3.8. The artificial view will only be accurate under certain conditions. Mathematically, these conditions are when the projection matrix, or projectivity, is actually a homography. Under other conditions, strange depth artifacts will appear in the image, that a human viewer will immediately see as incorrect. Reconstruction of views from free camera movement requires that the image has a depth map. Marc Pollefeys's work[25–27] is a good example of using structure from motion on images from a free camera to create artificial views of a scene by estimating the 3D structure of the scene and texture mapping the images onto the 3D model. Another example is work by Debevec[32, 33] which uses structured lighting and structure from motion to create realistic 3D models of historical sites. Both of these methods use substantially more complicated camera motion models and require rendering time that makes them unsuitable for any kind of real-time tracking. Therefore, only the simpler case of pan, tilt and zoom camera motion will be considered as it is feasible to perform the texture mapping and 3D transforms in real-time with accelerated hardware, making it possible to track targets under these conditions.

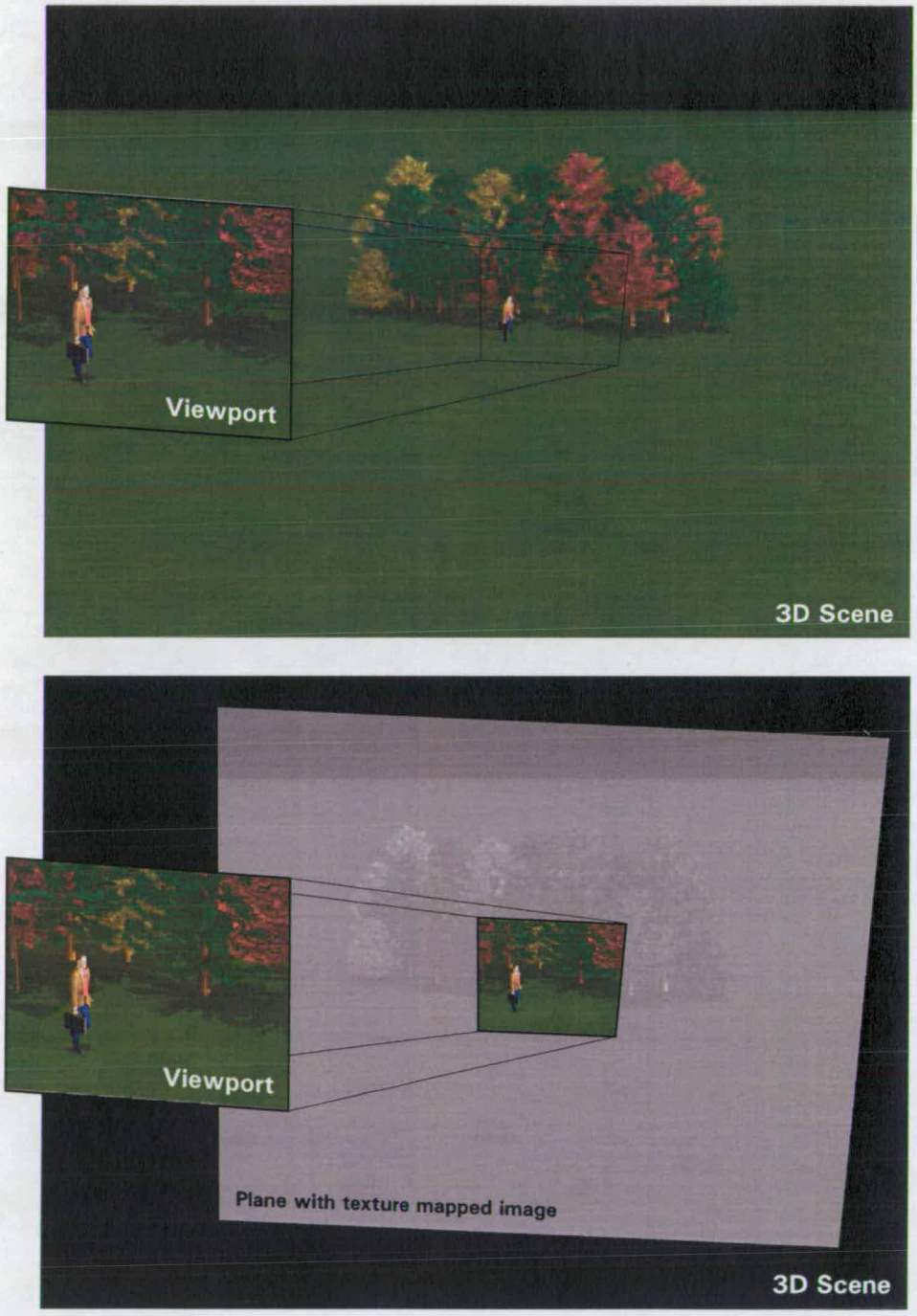


Figure 3.7: *Texture mapping an image onto a plane*



Figure 3.8: *Virtual reality panoramic image created from a series of photographs taken by the author at the Nyhaven in Copenhagen and stitched using Hugin with Panotools*

3.8 Environment Mapping and Front Cube Face Projections

Environment mapping[23, 47–49], as shown in Figure 3.9, is a common method of storing a 360° view of scene and creating reflective objects. A unit cube is placed around the scene and it is projected onto each face of the cube to produce an environment map. This method is advantageous, as it uses six images to store a high resolution representation of a scene that can later be texture mapped back onto a cube to produce a high quality reconstruction of the scene viewed from an arbitrary angle with little distortion.

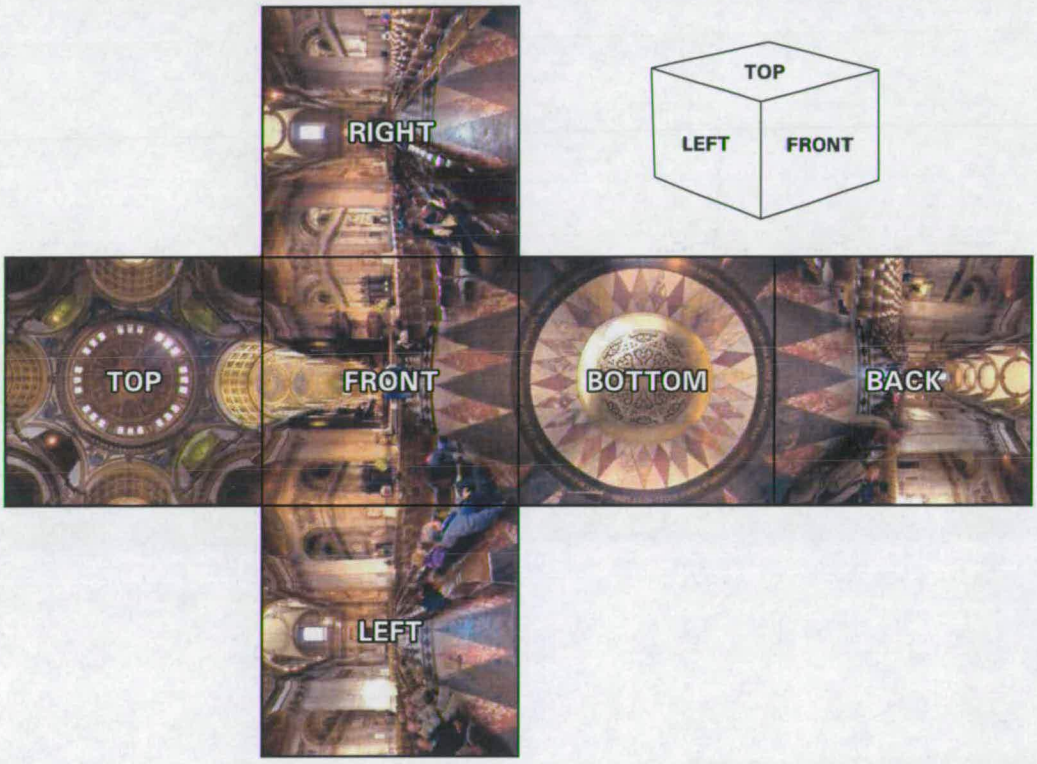


Figure 3.9: *An environment map of the interior of St Paul's Cathedral* (base image courtesy of Paul Debevec)

Another common method of visualising large scenes is a front cube face projection shown in Figure 3.10, where a scene is projected onto a plane located on the front face of the unit cube, i.e. the plane $z = 1$. An associated back cube face is used to store the scene directly behind the camera viewpoint. A front cube face projection has the trademark bow-tie shape and suffers large distortions at pan and tilt angles greater than 45° – this distortion can be removed by using a full environment map.

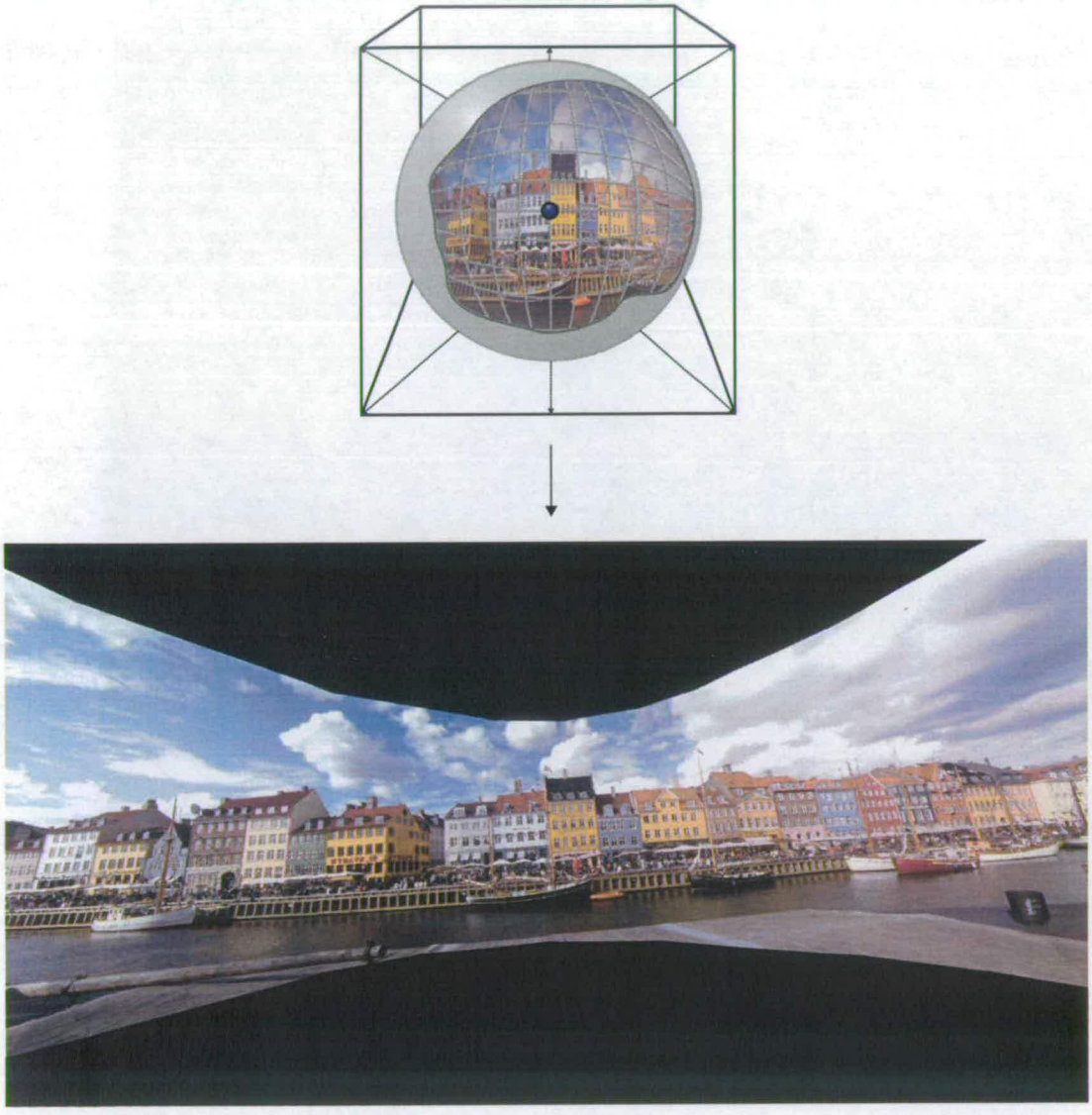


Figure 3.10: *Front cube face projection*

3.9 Pan/Tilt/Zoom Camera Model

The following section briefly explores the heuristic effects of pan, tilt and zoom on images captured by a PTZ camera. Previously in this chapter, the mathematical transforms of points have been discussed and this section attempts to reconcile these transforms with their visual effects.

3.9.1 Zoom

The magnification or zoom factor is controlled by varying the length of the *direction* vector. By doubling the length of the *direction* vector the zoom factor is doubled, as the projection plane moves closer towards the scene. Therefore the amount of relative zoom between two different views is given by the ratio of the length of the *direction* vectors in each view. Figure 3.11 shows the effect of changing the zoom factor. The left column shows an decrease in zoom factor (close angle to wide angle) and the right column shows an increase in zoom factor (wide angle to close angle).

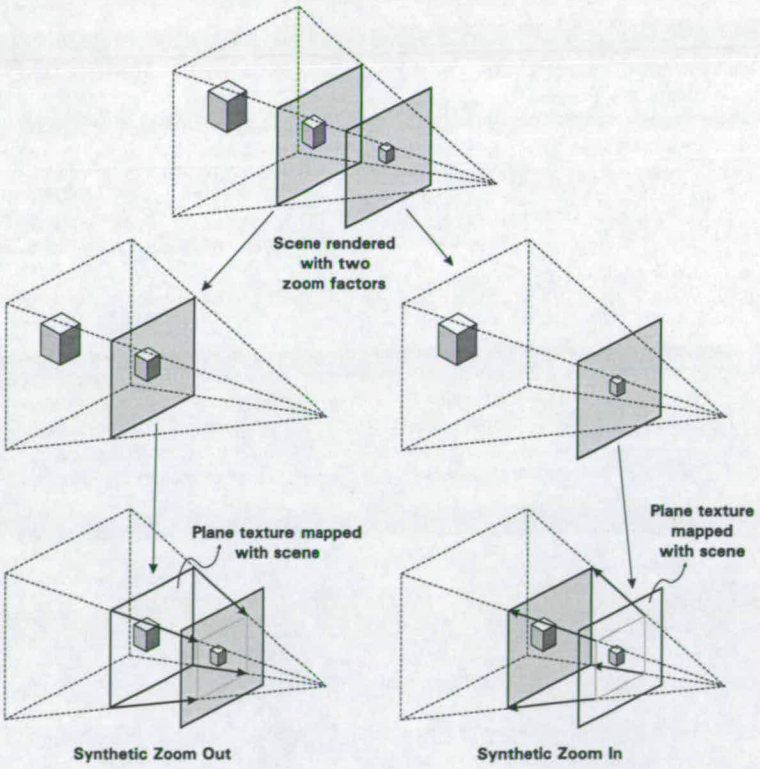


Figure 3.11: *Scene viewed at two zoom factors*

A change in only the zoom factor can be simulated by texture mapping (Section 3.7) the 2D image of the scene onto a plane at the same location as the projection plane and then viewing the scene with a different *direction* vector. Figure 3.11 also illustrates this and shows that the scene can be reasonably reproduced at different zoom factors by texture mapping only if the camera position remains fixed.

3.9.2 Rotation

Figure 3.12 shows a rectangular block viewed with increasing rotation. Note how the rectangular block becomes more quadrilateral-like with increasing rotation. Figure 3.12 shows that as the projection plane rotates the block is 'squashed' horizontally. Therefore with increasing rotation the ability to recover image information from the block decreases (as the spatial sampling rate remains constant, yet the projection causes the object width to decrease). The amount of block degeneration can be determined from its aspect ratio. A block with no degeneration will have an aspect ratio of near 1. However a degenerate or 'squashed' block will have asymmetrical height and width, producing an aspect ratio that is either very small or very large, but not 1. Another degeneration measure for quadrilateral blocks in general could be the pixel count comparisons.

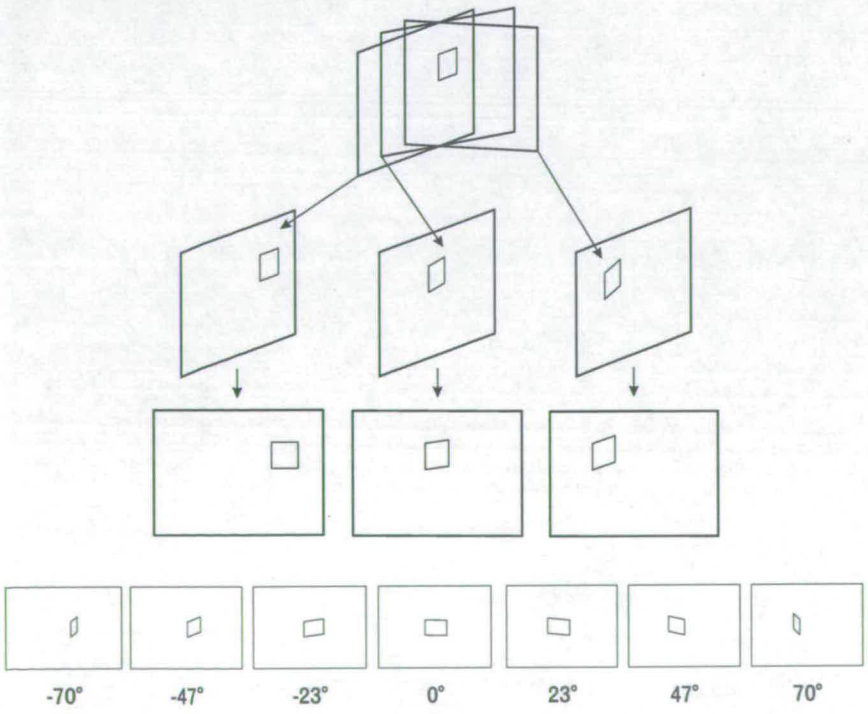


Figure 3.12: *Scene viewed at several camera rotations*

3.9.3 Translation

Translation results in different *eye positions* between two camera views and produces a stereo vision problem. Without knowledge of the object depth there is uncertainty as to its position. For most PTZ cameras there is negligible translation around the lens centre, so the effects of translation can be ignored. If translation does occur, then the epipolar geometry between the two views need to be calculated as described in [31, 34] and the position of the object triangulated. If the camera is free to move anywhere, then relating the views becomes a structure from motion problem. This has been explored by Pollefeys and van Gool[27, 108] in determining the Euclidean structure of scenes from hand-held video footage. Under pan, tilt and zoom conditions there is no translation, so these effects are beyond the scope of this work.

3.9.4 Rotation and Zoom

Figure 3.13 shows the effect of rotation and zooming at the same time. As the transforms are linear they can be applied separately to simulate the full camera parameter change.

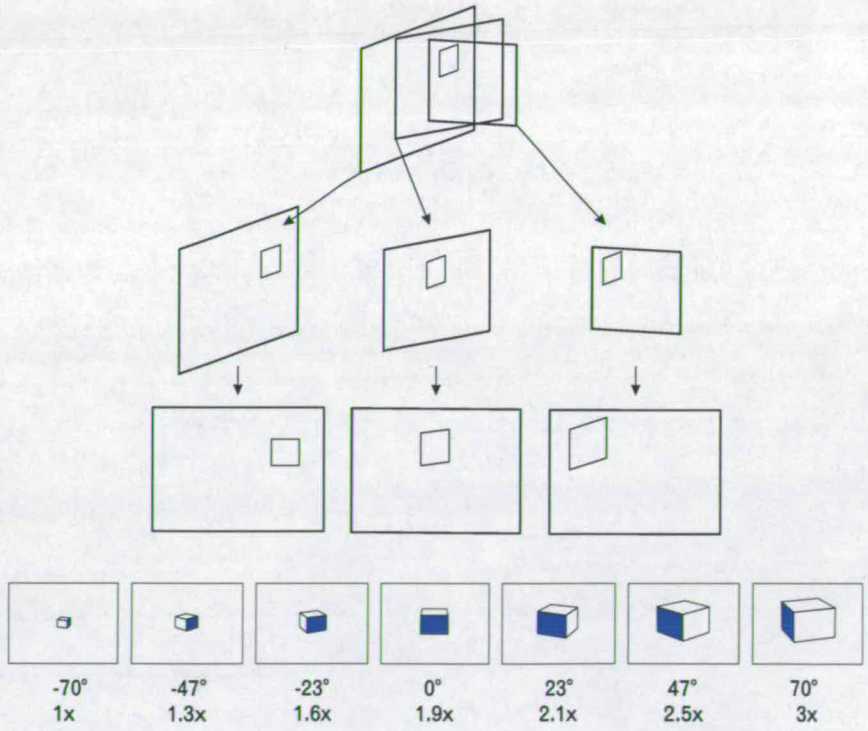


Figure 3.13: *Scene viewed at several camera rotations and increasing zoom*

3.10 Synthetic vs Real Cameras

All of the camera models discussed so far have been for ideal pin-hole cameras that do not suffer from real world effects such as radial distortion and depth of field. The process of digitising video sequences also adds noise and unwanted effects due to scanline synchronisation errors and interlacing. All of these effects are discussed in the following section in relation to the sequences captured for this thesis.

3.10.1 Radial Distortion

As a PTZ camera could have a large zoom range from wide angle to around 10x zoom, allowance needs to be made for the barrel and pincushion effects in the lens[102, 103, 105] due to limitations in lens manufacturing. As this distortion is radially symmetric it is possible to remove this distortion by transforming the image with a radial polynomial expression.



Figure 3.14: *Radial distortion effects*

Tsai's model of radial distortion[41] is widely used throughout computer vision and is briefly discussed below. The expressions for radial distortion in a lens, models the amount of distortion of the length of a ray extending radially outwards from the centre of distortion (x_c, y_c) (also known as the *principal point*). The amount of distortion is modelled as an even polynomial function as follows:

$$r_d = rF(r) = r(1 + \kappa_1 r^2 + \kappa_2 r^4 + ...) \tag{3.16}$$

where
- r_d distorted radius
- r undistorted radius ($r = \sqrt{(x - x_c)^2 + (y - y_c)^2}$)
- $\kappa_1 ... \kappa_n$ co-efficients of n^{th} order polynomial $F(r)$

Therefore, the distorted position of any pixel in an image due to a lens with radial distortion is given by:

$$x_d = (x - x_c)(1 + \kappa_1 r^2 + \kappa_2 r^4 + ...) + x_c \tag{3.17}$$

$$y_d = (y - y_c)(1 + \kappa_1 r^2 + \kappa_2 r^4 + ...) + y_c \tag{3.18}$$

where	(x, y)	undistorted pixel co-ordinates
	(x_d, y_d)	distorted pixel co-ordinates
	(x_c, y_c)	principal point or centre of distortion in pixel co-ordinates
	$\kappa_1 ... \kappa_n$	co-efficients of n^{th} order polynomial $F(r)$

3.10.2 Depth of Field

All the camera models described so far have assumed that the camera system is a perfect pin-hole camera. In reality, lens systems have finite thicknesses and the lens aperture is finite or can vary. This results in the focal point of a lens becoming a locus and the effects of this are generally attributed to 'depth of field' effects. The most obvious visual effect is where objects at a certain depth are in focus, while other objects in front and behind them are not in focus.

Generally, most commercial PTZ cameras have short focal lengths and small apertures which tend to remove most depth of field effects. The point of focus in most security applications is also generally at the lens 'infinity' point, therefore it is justifiable to use a pin-hole model of the camera.

3.10.3 Frame Grabbers and Interlacing

The Sony EVI-D31 PTZ camera produces a composite analogue PAL video signal that needs to be digitised before any image processing can be done on the video frames. As with most consumer video equipment, the Sony EVI-D31 is interlaced, which means that every alternate scanline is updated in every field. Therefore, if an object is moving, every alternate scanline in an interlaced image will be one time-step behind the other scanlines. Although, to a human viewer this effect is not distracting it does create some unwanted artifacts for machine vision applications. Hence, all of the sequences in this thesis are captured at half the frame rate and half the vertical resolution to remove the interlacing effects. Other techniques to remove

interlacing in software are discussed in [86]. Frame grabbers can also introduce some unwanted artifacts, such as small mismatches in scanline synchronisation which causes slight wobble in an image. With good quality frame grabbers it is possible to adjust the synchronisation thresholds and timing to remove these effects.

3.11 Camera Calibration

Camera calibration is a set of methods that relates points in the real world to points in an image by modelling the position of the camera relative to the world axes and the type of imaging projection it uses. The *intrinsic parameters* for a lens are dependent on the focal length of the lens, therefore for a fixed focal length lens the intrinsic parameters remain constant, but for a PTZ camera lens, they are represented as a function of the camera zoom parameter. The *extrinsic parameters* represent a geometric transform of the camera's position relative to the world co-ordinate axes. The extrinsic parameters are modelled as a translation, followed by a rotation.

3.11.1 Intrinsic Parameters

The camera intrinsic parameter matrix is defined as follows:

$$\mathbf{K} = \begin{bmatrix} \alpha f_y & s & x_c \\ 0 & f_y & y_c \\ 0 & 0 & 1 \end{bmatrix} \tag{3.19}$$

where
- α — aspect ratio between the vertical and horizontal focal lengths
- f_y — focal length of the lens in the vertical direction in pixels
- (x_c, y_c) — principal point or centre of the lens in pixels (often considered to be the centre of radial distortion too)
- s — skew factor (most cameras have square pixels therefore $s = 0$)

If real world physical dimensions are required then the units of the focal length and principal point need to be calculated in metres rather than pixels. Normally, this requires a simple scale change, but for more complicated lens systems it may be simpler to work only in metres. For most good lenses the principal point is near image centre. As the optical properties of the pan/tilt/zoom lens change the principal point will not remain fixed and will move as the lens zooms in or out. Therefore, the intrinsic parameters for a zoom lens are as before, but need to be described as a functions of some zoom parameter z:

$$\mathbf{K}(z) = \begin{bmatrix} \alpha f_y(z) & 0 & x_c(z) \\ 0 & f_y(z) & y_c(z) \\ 0 & 0 & 1 \end{bmatrix} \tag{3.20}$$

3.11.2 Extrinsic Parameters

The extrinsic camera parameters determine the geometric orientation of the camera with respect to the world and take the form of a rotation $\mathbf{R}$ followed by a translation $\mathbf{T}$. The result is a 3×4 matrix (which incorporates the reduction to a 2D point) as follows:

$$\mathbf{M} = \mathbf{RT} \tag{3.21}$$

$$= \begin{bmatrix} r_1 & r_2 & r_3 & t_1 \\ r_4 & r_5 & r_6 & t_2 \\ r_7 & r_8 & r_9 & t_3 \end{bmatrix} \tag{3.22}$$

3.11.3 Camera Transform

The camera transform is a projection matrix that transforms homogenous 3D points into homogenous 2D points, as follows (using symbols from above):

$$\mathbf{x}' = \mathbf{KRTx}$$

$$\begin{pmatrix} x' \\ y' \\ w' \end{pmatrix} = \begin{bmatrix} \alpha f_y(z) & 0 & x_c(z) \\ 0 & f_y(z) & y_c(z) \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_1 & r_2 & r_3 & t_1 \\ r_4 & r_5 & r_6 & t_2 \\ r_7 & r_8 & r_9 & t_3 \end{bmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} \tag{3.23}$$

Often w' is not equal to one and to effect the transform $\mathbf{x}'$ is divided through by w'. This ensures that the point is projected onto the image plane.

3.12 Convergence of Computer Graphics and Image Processing

There is an obvious convergence between image processing and computer graphics, as techniques from both streams of research are being used together often in applications such as the creation of panoramic images[48], environment mapping[47, 49] and structure from motion[25, 27]. As the computer graphics aspects of research tends to deal with graphics and rendering hardware, its notation and terminology is the most logical choice for any implementation of a convergent vision algorithm. The projective geometry aspects add a good theoretical background and can be adapted for use with computer graphics systems such as OpenGL[23]. This section deals with the crossover between these two disciplines and how established concepts from each can complement each other and be used together.

3.12.1 Projecting points using a camera calibration and a perspective transform

By performing a camera calibration using methods described in Chapter 6 there will be numerical values for all of the parameters in the intrinsic and extrinsic camera matrices. These can be used to create projection matrices for use with a graphics system such as OpenGL to transform images from one camera to another. World co-ordinate points projected onto the image plane using Equation 3.23 lie within the image boundaries, i.e. $0 \leq x \leq width$, $0 \leq y \leq height$ and $z \geq 1$. The boundaries for the associated viewing frustum are the same as the image rectangle boundaries. By using the perspective transform of Equation 3.10, it is possible to project the points from the viewing frustum into the canonical viewing frustum. Once in the canonical viewing frustum, the points can be transformed using the viewport transform (Equation 3.12) to a destination image that is not necessarily the same size as the source image or could contain a sub-region of the total viewable region. This allows a large amount of freedom in the size of the output image, which has benefits such as reduced computation time for temporary results that are rendered as small output images or for increased detail, by rendering a small section of the scene at high resolution.

By combining Equation 3.23 above with the perspective transform of Equation 3.10, a homogenous 3D point $\mathbf{x}$ is projected into the canonical viewing frustum by:

$$\mathbf{x}' = \mathtt{P}\mathbf{KRT}\mathbf{x}$$

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{bmatrix} \frac{2near}{right-left} & 0 & \frac{right+left}{right-left} & 0 \\ 0 & \frac{2near}{top-bottom} & \frac{top+bottom}{top-bottom} & 0 \\ 0 & 0 & -\frac{far+near}{far-near} & -\frac{2(far)(near)}{far-near} \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} \alpha f_y(z) & 0 & x_c(z) & 0 \\ 0 & f_y(z) & y_c(z) & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_1 & r_2 & r_3 & t_1 \\ r_4 & r_5 & r_6 & t_2 \\ r_7 & r_8 & r_9 & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} \tag{3.24}$$

Finally, to get the position of the point in image pixel co-ordinates, the viewport transform of Equation 3.12 is applied to give:

$$\begin{aligned} x'_{image} &= x_{win} + width(\frac{x'+1}{2}) \\ y'_{image} &= y_{win} + height(\frac{y'+1}{2}) \\ z'_{image} &= \frac{z'+1}{2} \end{aligned} \tag{3.25}$$

3.13 Summary

This chapter has introduced the need for a common approach to modelling a camera viewing a 3D scene drawing on concepts from computer graphics, computer vision and projective geometry. The process of projecting points from various camera views has been investigated. A visual analysis of the effect of panning, tilting and zooming on 2D projections of 3D scenes has been presented. Concepts such as texture mapping and image warping have been defined and discussed. All of these concepts together have enabled the creation of a model of a PTZ camera that can be used in Chapter 4 to track various targets using a real-world PTZ camera.

Chapter 4
Pan/Tilt/Zoom Tracking

4.1 Introduction

Before introducing the two novel methods of tracking under pan, tilt and zoom conditions, conventional block tracking with static cameras is discussed in Section 4.2. A review of existing zoom tracking methods follows in Section 4.3.

An new approach to block tracking with a PTZ camera is presented as two methods. The first method warps the search area to a reference view and returns a sub-pixel match of the block. The second, warps the reference block into the current view and matches the target using normalised cross-correlation. The first method is more accurate than the second, however it is significantly slower. A method of maintaining and updating a reference block from images of varying scale is also presented.

Methods and techniques common to both novel zoom tracking methods are discussed in Section 4.4. The descriptions of the novel zoom tracking algorithms are presented in Section 4.5 and Section 4.6. Both methods are critically analyzed in Section 4.7. A brief section on region tracking follows in Section 4.8 as a comparison method for applications in security and surveillance.

4.2 Block Tracking Without Pan/Tilt/Zoom

Conventional block tracking[86] involves taking a rectangular region of pixels (known as a *block*) and attempting to match it with another rectangular region of the same size in another image. Normally, the sample block (or *reference block*) is chosen to be in a region of the image containing the target to be tracked and it normally is chosen surrounding a distinctive feature such as a corner or edge[143]. A detailed discussion of the reference block can be found below in Section 4.2.1.

Searching the entire image is not computationally feasible in real-time, so a reduced portion

known as the *search region* is searched for the target. The choice of size of the search region and its effect on the performance of the tracker can be found in Section 4.2.2 below.

Once a *match position* (defined in Section 4.2.3) has been found along with the *match value* or match confidence, it can be determined whether the target has been found in a frame. Using methods discussed later in Section 4.2.10 the tracker can be made robust under conditions with false matches and occlusions.

As most targets that need to be tracked for surveillance and media post production are not rigid (e.g. people and faces) the reference block will need to be updated so that it accurately represents the target being tracked. Section 4.2.4 below defines reference block updating and Section 4.2.7 discusses various methods of implementing block updating.

Many different methods of matching blocks exist, such as absolute difference, sum of squared difference and normalised cross-correlation (NCC), to search for targets in subsequent frames. These various methods are discussed in Section 4.2.5, however normalised cross-correlation is the only tracking method used throughout this thesis.

In order to make practical block trackers robust to mismatches and improve performance, a model of the target's movement is created and position filtering is performed to estimate the position of the target in subsequent frames based on previous observations. Implementation of position filtering and making trackers robust are discussed in Section 4.2.6 and Section 4.2.10 respectively.

4.2.1 Definition – Reference Block

The reference block, also sometimes called the *template*, is a rectangular group of pixels sampled from the tracking target that will be identified in frames of an image sequence. An example reference block is shown in Figure 4.1. Normally, reference blocks are square and are typically between 8×8 pixels and 16×16 pixels in size. The size of the reference block affects the tracking performance in many ways: a larger block increases the amount of computational time needed to match the block to an area of the image and if the block is too big it will contain more of the background than the tracking target (which results in poor tracking results).

In situations where there is scale change and rotation, reference blocks should be an odd number of pixels in width and height. This allows the centre of the block to be used as the block position

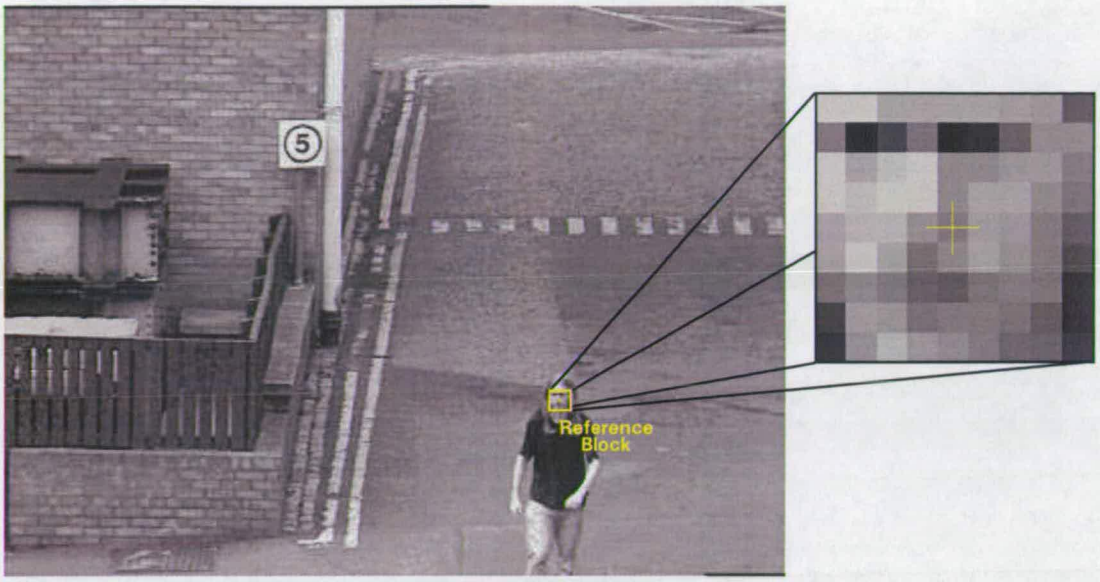


Figure 4.1: *Reference block*

and scale changes or rotations do not move the block position (which would be the case if the top-left corner was used as the block position).

All reference blocks used in this work were square and were either 9×9 pixels or 17×17 pixels in size.

4.2.2 Definition – Search Region

When searching for a target in an image it is computationally prohibitive to do an exhaustive search of the image, so only a small region of the image is searched for the target. The position of this small region, or *search region*, is normally set from some external source such as a human operator or the last known position of the object. The search region is sometimes referred to as the search aperture or search area and its 'size' is often referred to as the search radius. The search radius is normally half the width of the square search region (see Figure 4.2).

The size of the search region affects the performance of the tracker, as a large search region means that the reference block needs to be tested against a large number of positions in the image, which is computationally expensive.

The search region also needs to be large enough that when an object moves between frames it will still be inside the search region (relative to its last known position). If the average speed of the object and the frame rate of the video sequence are known it is possible to estimate

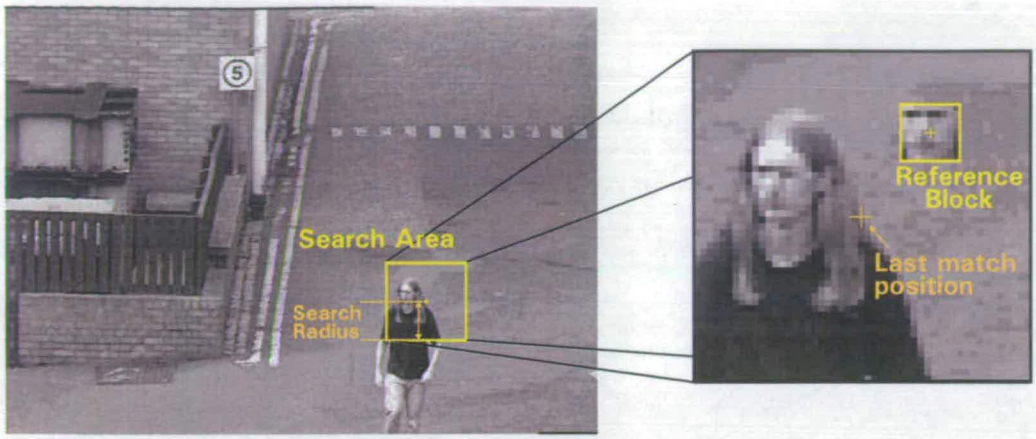


Figure 4.2: *Search region*

the minimum size of the search region. Many algorithms adaptively vary the size of the search region[86] depending on the certainty of the estimate of the target's position i.e. a high certainty of the position will mean the search area will be small. There is however, the risk of not finding the target as the search area is too small.

Typically, the size of the search region used in this work was 15 to 35 pixels on either side of the last known position of the target. For a 35 pixel search radius a square region 70 pixels by 70 pixels is search, i.e. 4900 matches are calculated. Compared to searching the 110 592 positions for the entire image[1], it is feasible to track in real-time with this small search area.

4.2.3 Definition – Match Position and Match Value

When a good match for the target is found this is called the *match position* and in this work will be defined as the pixel position of the centre of the matched block in image pixel co-ordinates (Figure 4.3).

The match value is an indication of the confidence of the match. If every pixel in the reference block matches every other pixel in a region of the image there is a 100% match. All match values should be normalised so that they lie in the range [0; 1], where 0 is a complete mismatch and 1 is a perfect match. Good matches are considered to be match values with a confidence of at least 85% with the optimal range being 90% - 100%.

[1] Calculated for a 328 × 288 pixel frame.

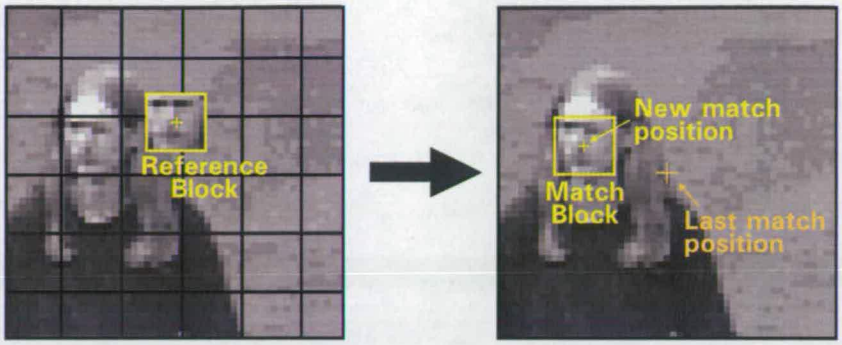


Figure 4.3: *Match position*

4.2.4 Definition – Reference Block Updating

As most targets that will be tracked are not rigid objects they change over time and hence the reference block (or sample of the object) needs to be updated to reflect the changes in the object. Again, there is a tradeoff between updating the block too quickly and allowing the reference block to 'drift' off the target onto something else that is similar and not incorporating changes quickly enough.



Figure 4.4: *Reference block updating*

Many approaches for updating the reference block exist with the simplest being block replacement, where the entire block is replaced with a new block from the best match position. More sophisticated spatio-temporal statistical methods, such as Kalman updating[144], can be used to reject spurious changes in the object while still updating changes within expected bounds and estimating the confidence of the update.

Three modes of block updating are used throughout this work: the first is no update at all and the target is tracked using the reference block generated in the first frame of the sequence. The second is a naive finite impulse response (FIR) filter that blends blocks temporally and the last is a sophisticated Kalman filter based block updater. These methods are discussed further in Section 4.2.7.

4.2.5 Block Matching Methods

Many types of block matching methods exist and most methods compare the intensity of pixels in rectangular regions of the image with a rectangular reference block of the same size. The list below summarises the various methods of measuring the difference between a reference block and a test block:

Absolute difference $1 - \frac{\sum_{ij} \frac{|R_{ij} - T_{ij}|}{\text{maximum intensity}}}{n^2}$

This method finds the absolute intensity difference between each pixel in the reference block and a test block in the image. The difference is normalised by the maximum intensity and the number of pixels so that the match value lies between 0 and 1 and is subtracted from one, so that a perfect match is signified by 11

Sum of squared difference $1 - \frac{\sum_{ij} \frac{(R_{ij} - T_{ij})^2}{(\text{maximum intensity})^2}}{n^2}$

Similar to the absolute difference method and without normalisation, measures the variance between the two blocks.

Normalised Cross Correlation (NCC) $\frac{1}{2} - \frac{\sum_{ij} \left[(R_{ij} - \bar{R}) \cdot (T_{ij} - \bar{T}) \right]}{2\sqrt{\sum_{ij} (R_{ij} - \bar{R})^2 \cdot \sum_{ij} (T_{ij} - \bar{T})^2}}$

Phase correlation is used to find the similarity of two blocks. Subtracting the mean of both blocks makes this match method less susceptible to illumination changes. $\bar{R}$ and $\bar{T}$ are the mean intensities of the reference and test blocks respectively.

where
- R_{ij} Pixel intensity in the reference block at row i and column j
- T_{ij} Pixel intensity in the test block indexed by pixel ij
- n Total number of pixels in the reference block

The methods described above match blocks in greyscale, however matching can be done with colour frames. A colour space transform could be used to transform an RGB image into CIE

L*A*B space[2] for example and pixel intensities are now compared as vectors, i.e. the intensity difference from before is now the magnitude of the difference of two colour vectors.

Block matching does not necessarily need to be done with pixel intensities and matching can be done in the DCT domain as is common in MPEG block based video compression. Other block matchers use PCA space or wavelets[86] and the method can be extended to any set of basis functions. Now the block matching process is essentially the same where the transform is performed and a greyscale image is created (from one or more of the basis vectors) and block matching is performed in the same way as before, such as using cross-correlation for example.

Other methods exist that do not use blocks at all, such as Tomasi and Kanade[146] use a colour histogram skin tracker, which tracks objects in colour space comparing only the histograms of regions or blobs. This is useful in following hands[22] or faces.

4.2.6 Match Position Prediction and Filtering

Predicting the position of the target in subsequent frames is a good way to improve the robustness of a tracker. If the position of the target can be accurately predicted then a small search region can be used to find the target without exhaustively searching the entire frame. Kalman filters[147] are used throughout this thesis to estimate the position of the target. If the reader is unfamiliar with Kalman filters it is recommended that [148, 149] be read first, as a knowledge of Kalman filters will be assumed for the remainder of this section.

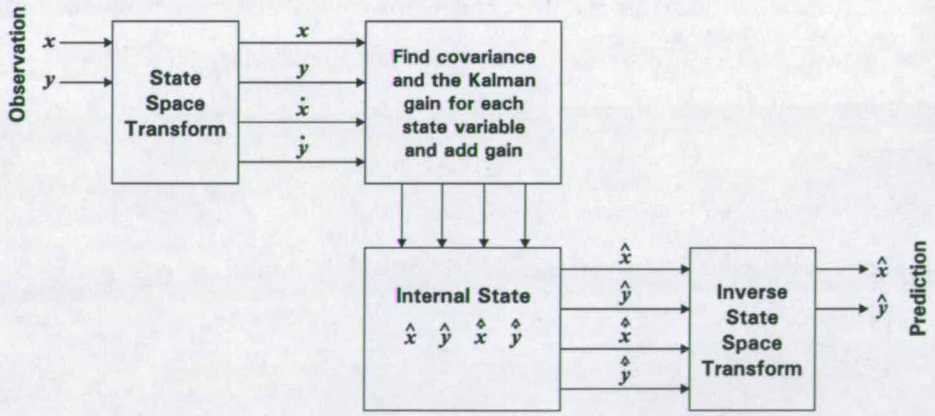


Figure 4.5: *Constant velocity Kalman position filter*

[2]The CIE L*A*B colour model is designed to arrange colours in a similar way to human perception of colour difference. See Hunt[145] for more details or the *Commission Internationale de l'Eclairage* online at `http://www.cie.co.at`.

Recently particle filters[150, 151] have begun to supersede Kalman filters due to the larger range of scenarios they can model very accurately. In simple situations such as when the object is modelled as having constant velocity, a particle filter will reduce to the same predictor/corrector equations as in a Kalman filter. Given the simple nature of the position prediction particle filters were not considered as their added complexity produces no extra benefits since they reduce to the same expressions as for the Kalman filter. Other naive approaches such as moving averages and FIR type position filters could be used, however these lack the advantage of statistical filters, which reject outlier observations.

4.2.6.1 Kalman position filtering

A simple constant velocity model was used to estimate the position of targets. The model is based on the assumption that for short periods of the sequence a target will have constant velocity. From simple control theory, a second order velocity filter will be able to track the position of a target without any lead or lag. This is ideal for the types of moving targets that will be tracked in this work. If the target has varying velocity (as should be the case for real world targets) there will be a lag in the velocity estimation, however this is advantageous as it will smooth any erratic movement of the target and prevent the tracker from matching to an object that is not the target (and producing a false match). As a Kalman filter is not a simple linear filter it can reject false observations, which a simple linear filter would treat as a step change in position resulting in unstable dynamics for a short period.

The observation vector for the Kalman filter consists only of the x and y positions of the target as matched in each frame. The target's position can be calculated by the following state space transform:

$$\mathbf{s}(t + \Delta t) = \mathbf{s}(t) + \mathbf{v}(t)\Delta t \tag{4.1}$$

where	$\mathbf{s}(t) = (x(t), y(t))^{\mathsf{T}}$	target position vector
	$\mathbf{v}(t) = \dot{\mathbf{s}}(t) = (\dot{x}(t), \dot{y}(t))^{\mathsf{T}}$	target velocity vector
	Δt	finite time step, which is set to 1 if time is defined as the frame number

This can be represented in matrix form as follows:

$$\mathbf{s}(t + \Delta t) = \mathbf{T}\mathbf{x}(t)$$

$$\begin{pmatrix} x(t+\Delta t) \\ y(t+\Delta t) \end{pmatrix} = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \end{bmatrix} \begin{pmatrix} x(t) \\ y(t) \\ \dot{x}(t) \\ \dot{y}(t) \end{pmatrix} \tag{4.2}$$

If the time parameter t is defined as the frame number then the timestep Δt is set to 1 and the state space transform matrix becomes:

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} \tag{4.3}$$

The state space transform matrix transforms the observation of the target position into the internal Kalman state vector, which contains both the target's position and velocity. The model noise is set to allow small variations in position and smaller changes in velocity and the cross terms are zero which indicates that the position and velocity are independent, giving the model noise matrix:

$$\mathbf{M} = \begin{bmatrix} \text{x variation} & 0 & 0 & 0 \\ 0 & \text{y variation} & 0 & 0 \\ 0 & 0 & \text{x velocity variation} & 0 \\ 0 & 0 & 0 & \text{y velocity variation} \end{bmatrix} \tag{4.4}$$

The process noise reflects the errors in observing the target's position, which in the case of tracking is the estimated uncertainty in the match position. It is reasonable to assume that all matches will be within a few pixels of the object's true position. This results in a process noise matrix as follows:

$$\mathbf{P} = \begin{bmatrix} \text{x position variation} & 0 \\ 0 & \text{y position variation} \end{bmatrix} \tag{4.5}$$

The estimated position of the target is found by calculating the Kalman gain from the model and process noise covariance matrices (Equation 4.4 and 4.5). The Kalman gain updates the internal state vector to give the corrected estimate of the internal state. The inverse state-space transform is then applied to give the output in terms of the position only. The confidence of the position estimate can be found from the predicted error covariance matrix.

4.2.7 Reference Block Updating

When tracking objects that are not rigid, they will change from frame to frame due to many factors such as rotation, illumination changes and self-occlusion. It is therefore necessary to update the reference block so that the target can still be tracked even if it has changed significantly. Unfortunately, updating the reference block comes at the cost of increasing the possibility of changing the reference block so significantly that is no longer matches the target – for example, this could be due to a false match block being incorporated into the reference block.

Several update strategies exist and they range from naive block replacement to a model based estimate of the reference block based on past observations. Block replacement is the simplest update strategy where after each successful match the reference block is replaced with a block from the current frame. This method works well unless there is a false match, in which case the target being tracked will no longer be the original object. Bajcsy and Cahn von Seelen[20] used block replacement to track objects while zooming in. This unfortunately is a naive approach and is not very robust.

4.2.8 FIR block updating

A more reliable method is to blend the current reference block with the block at the match position. If a percentage of the match block is added to the reference block it will slowly change over time. An example of FIR block updating is shown in Figure 4.6. This finite impulse response (FIR) type filter is easy to implement as a recurrence equation where each pixel in the reference block is added to ratio of the match block as follows:

$$R_{ij}(t+1) = (1-\alpha)R_{ij}(t) + \alpha M_{ij}(t) \tag{4.6}$$

where
- R_{ij} reference block
- M_{ij} match block
- α a value in the range $[0;1]$ which controls the percentage of pixel intensities of each pixel that gets added to the reference block from the match block
- t time index

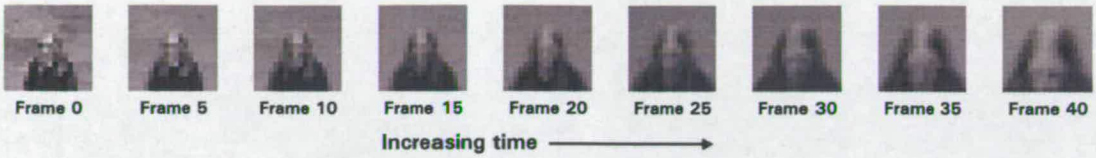


Figure 4.6: *Example FIR block updating*

If α is set to 1 then the reference block will be replaced at each time step. The choice of α controls the amount of updating that will occur at each time step. If the value of α is too small the reference block may not update quick enough if the object changes quickly.

A disadvantage of using FIR updating is that the reference block becomes blurred or smudged as it updates through time. This can sometimes result in low match values, as although the target in the current frame matches the reference block, it does not match exactly as it is an 'average' of the previous match blocks. An FIR updated reference block will always lag the target as it depends on past inputs only. In order to actively estimate the new reference block a more sophisticated filter needs to be used.

4.2.9 Kalman block updating

Peacock and Haworth[144] suggest a method of updating a reference block using a Kalman filter. The reference block is unwrapped into a single column vector of pixel intensities as the Kalman state vector. Each pixel is treated as a temporally varying signal with values between 0 and 255. The state space transform matrix is the identity matrix as each pixel is independent and does not depend on any other pixel in the block. Although this is not strictly true, it is a reasonable simplification which is compensated for in the estimated covariance matrix or Kalman gain matrix.

The model noise matrix is set to allow a pixel intensity variation of n down the diagonal as follows:

$$\mathbf{M} = \begin{bmatrix} n & 0 & \dots & 0 \\ 0 & n & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & n \end{bmatrix} \tag{4.7}$$

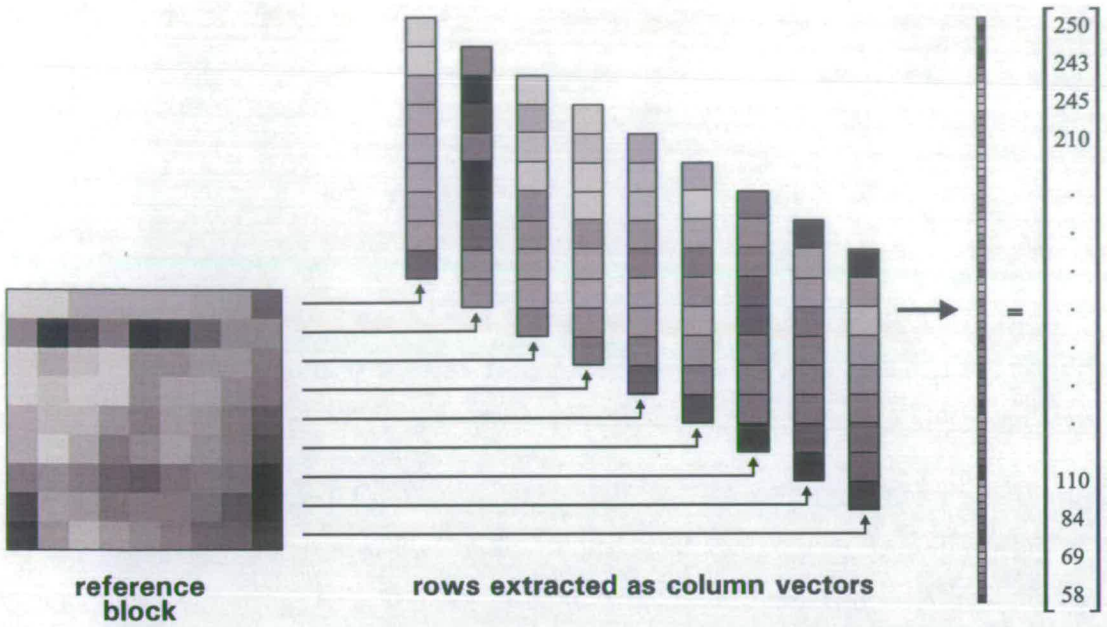


Figure 4.7: *Each row of the reference block* (left) *is extracted into a column vector* (centre) *and then stacked one on top of the other to form a large column vector* (right) *of intensity values which form the Kalman state vector*

The process noise is similar with m typically set to 0.01 as the pixel intensity is known fairly accurately, with small variations due to defects in the CCD, thermal noise or frame-grabber artifacts for example:

$$\mathbf{P} = \begin{bmatrix} m & 0 & \dots & 0 \\ 0 & m & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & m \end{bmatrix} \tag{4.8}$$

The current reference block is obtained by finding the corrected Kalman state vector and reshaping it from a column vector into a block of pixels. The Kalman update method has two significant draw backs: speed and memory usage. For blocks larger than 32×32 pixels the memory usage becomes immense (several hundred megabytes) – see Section 7.4.2 for a further discussion on implementation of Kalman filters for updating reference blocks. However, the method has many advantages as the reference block is an estimate based on past observations with outliers being rejected. This makes it a very attractive and robust updating method.



Figure 4.8: *Example Kalman block updating*

4.2.10 Robustness and Occlusions

A major cause of block trackers losing the target during tracking are full or partial occlusion of the target and regions of the image that are similar to the target which result in false matches. The simplest remedy to this problem is to add position filtering and prediction to the tracker. [59, 62, 83] track through occlusions by using the position prediction as the object prediction if the match confidence drops below a threshold, in the hope that the object will only be occluded for a few frames and will become visible again. If the position prediction has been good then the target should be inside the search area after the occlusion and hence matched.

Position filtering also provides a good indication as to whether the target matched is the true match. If the target matched is too far from the estimated position the outlier rejection can be performed and another good match close to the estimated position can be chosen as the match position.

If a statistical position filter, such as a Kalman filter, is used, then it is often possible to obtain a measure of the confidence of the position prediction. This can be used to control the size of the search window: when the position prediction confidence is high the search region can be small; when it is low a larger search area can be used. By reducing the size of the search window, it is possible to exclude areas of high false match, thereby making the tracker more robust. This does however add extra uncertainty if the position prediction is inaccurate. Other methods such as one proposed by Loutas[152] use mutual information from several feature tracks to deal with partial occlusions.

4.3 Summary of Current Zoom Tracking Methods

This section is a brief review of some previous work on methods of pan/tilt/zoom tracking – a full critical can be found in Section 2.5 on page 22. Early work in zoom tracking was performed with active vision heads[37, 38, 119, 120] which were more concerned with vergence and focal length control. Researchers attempted to emulate biological vision systems and reproduce tracking functions in computer vision systems. Bajcsy and Cahn von Seelen[20] produced an early method of block tracking while zooming. Their method estimated the scale change by performing affine block matching. Their results were successful for scale changes of up to 20%. In another method, Fayman *et al*[19] estimate the scale change by optical flow to maintain the size of an object while zooming. A second method by Fayman *et al* uses block matching to track corner features while zooming and moving. Reid and Murray[122] produced a much more mature method of tracking corners by affine transfer upon which Hayman[5–7] and Tordoff[4, 8, 9] both build to track targets while panning, tilting and zooming. These methods use a mixture of tracking only corners and by finding corner correspondences by block matching.

There are many methods of tracking people while panning, tilting and zooming which do not use block methods: Haritaoglu *et al*[3] produce a system called W4, which models human behaviour by stick models from a motion segmentation of video. This method has similarities to the Ritseye system[10] which tracks human faces based on skin colour models and face geometry. Colour tracking and model based methods are fairly robust to scale change and rotation. These methods do however suffer from some uncertainty in the true position of an object. This is due to variations in the segmentation and fitting of the models – block tracking does not suffer from this and there is a measure of the quality of the match (which is not always available in motion segmentation). Other methods attempt to transform the image where scale and rotation invariant comparisons can be done[124]. Eigenimages[125] can be used to compare templates to an image of the target at various scales, however this method is mostly applied to objects that change scale rather than by the camera zooming.

The choice of tracking method depends on the application: for security and surveillance applications it makes sense to use blob or region based methods as some sort of motion segmentation will be available. In order to maintain the size of one object, the corner methods suggested by Tordoff[4] would be more useful. However, for accurate tracks of specific points on an object, a block matching method is more suitable as it does not depend on finding a corner feature or require that the target is moving.

4.4 Transforming Between Views

A perspective camera model for a PTZ camera, as described in Chapter 3, can be found using the methods in Chapter 6 which allows the transformation of points, geometric shapes and images between camera views. In object tracking it is common to transform points between several views – this process is explained in Section 4.4.1. Images and blocks also need to be warped (or transformed) from one camera view into another to allow object tracking or reference block updating. Section 4.4.2 and Section 4.4.3 explain how to warp images between views as well as the texture mapping used to implement this with reference to computer graphics rendering environments such as OpenGL[23] and DirectX[24].

In the tracking algorithms presented in Section 4.5 and Section 4.6 the following terms will be used to describe the camera view and co-ordinate systems used: the *current view* and the *reference view*. The *current view* is the camera view of the frame in which the target will be matched. This view contains the target block and the current frame in image pixel co-ordinates. The current view can also be described in world co-ordinates i.e. the image can be considered to be an image plane orientated in the same way as the projection plane of the camera. The *reference view* is a virtual camera view which contains the reference block. This view is entirely synthetic and is chosen to maintain the best representation of the reference block for the pan, tilt and zoom range. Ideally the reference view should be chosen such that it has the average pan and tilt position of the area of interest and the highest zoom factor. Points in the reference view can be described as 2D image pixel co-ordinates or in world co-ordinates, as for the *current view*.

4.4.1 Forward and Reverse Camera Transforms

It is useful to design a process that transforms points, polygons and images from view to view using the same interface. This is illustrated in Figure 4.9. Polygons can be considered to be a collection of points and images are 2D arrays of points. Therefore, transforming polygons and images is actually repeatedly transforming many image points.

The transforms shown in Figure 4.9 are the product of the projective transform matrix (Equation 3.24) and the viewport transform (Equation 3.25). As these transforms are invertible, the inverse transform can be found by inverting the forward transform matrix. All points to be transformed must be represented by 4-vectors as 3D homogenous co-ordinates. In order to

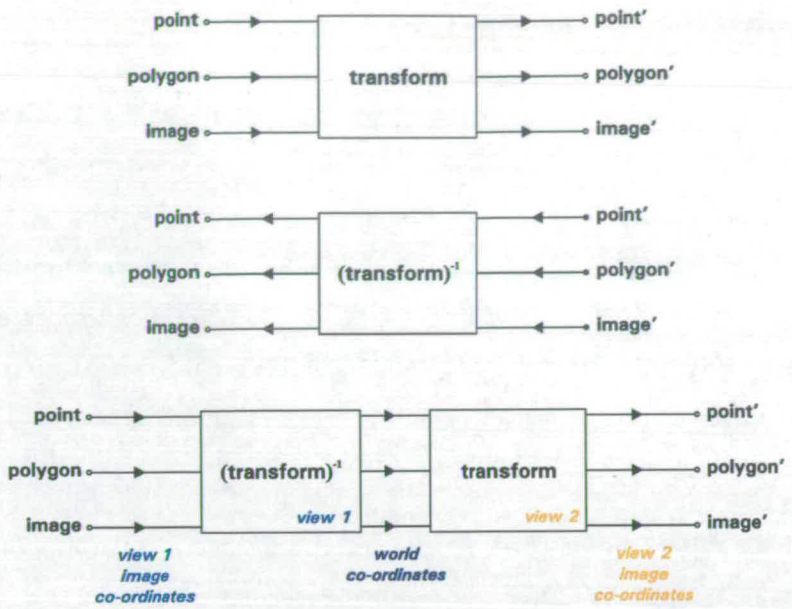


Figure 4.9: *Forward and reverse camera transforms*

transform a point from one view to another, the point needs to be untransformed into world co-ordinates and then transformed into the target camera view image co-ordinates.

It has been assumed that the radial distortion has been removed from all images. However, if this has not been done in a preprocessing step, then it would be necessary to include radial distortion into the transform.

4.4.2 Warping Images Between Views

In order to warp images between two views, two quadrilaterals or *quads* (one in each view) bounding the regions to be warped are needed. The ordering of the vertices in the quads is important to avoid the degenerate case of a concave quadrilateral (i.e. where two sides of the quadrilateral cross). If a quad is known in the source image it can be projected into the destination image using the camera transforms associated with each camera view.

Normally, texture mapping will take place between a rectangular quad (the reference block) and a non-rectangular quad (the image block) as shown in Figure 4.11. The reference block is stored as an image, therefore a mapping needs to be found between the reference block image and the quad on the image. This is done by dividing both quads into two triangles as in Figure 4.10 and a transform matrix **T** can be found which transforms points from *triangle A* into *triangle B* using the equation below:

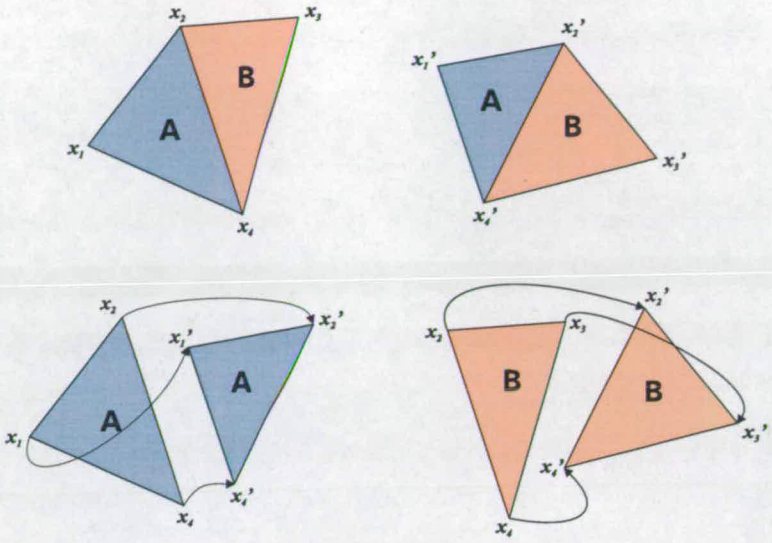


Figure 4.10: *Transform between two quads*

$$\mathbf{X_1} = \mathbf{T}\mathbf{X_2} \tag{4.9}$$

$$\begin{bmatrix} \vdots & \vdots & \vdots \\ \mathbf{x_1} & \mathbf{x_2} & \mathbf{x_3} \\ \vdots & \vdots & \vdots \end{bmatrix} = \mathbf{T} \begin{bmatrix} \vdots & \vdots & \vdots \\ \mathbf{x'_1} & \mathbf{x'_2} & \mathbf{x'_3} \\ \vdots & \vdots & \vdots \end{bmatrix}$$

$$\begin{bmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} T_1 & T_2 & T_3 \\ T_4 & T_5 & T_6 \\ T_7 & T_8 & T_9 \end{bmatrix} \begin{bmatrix} x'_1 & x'_2 & x'_3 \\ y'_1 & y'_2 & y'_3 \\ 1 & 1 & 1 \end{bmatrix} \tag{4.10}$$

where	$\mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3}$	2D vertices of first triangle
	$\mathbf{x'_1}, \mathbf{x'_2}, \mathbf{x'_3}$	2D vertices of second triangle
	$\mathbf{T}$	transform matrix (homography) between triangles

Essentially, $\mathbf{T}$ is a homography, as the transform between the two triangles is a rotation, translation and scale change. Using the transform $\mathbf{T}$ it is now possible to scan every pixel in the destination triangle and find the corresponding pixel in the source triangle, hence warping the image from one view into another. This process is known as texture mapping, which is discussed further below.

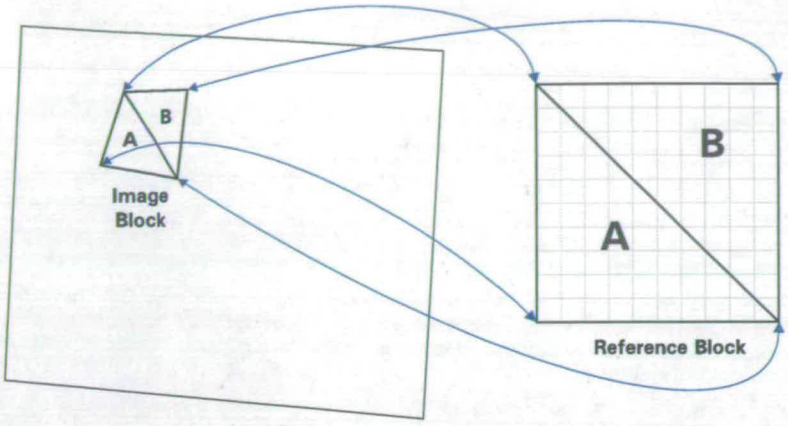


Figure 4.11: *Warping a quad to a rectangular quad*

4.4.3 Texture Mapping

Any arbitrary closed shape can be tesselated into a triangular mesh by methods such as Delaunay Triangulation[153] which are commonly used in computer graphics and finite element analysis. But in the case of texture mapping images from one view to another, the quadrilateral bounding the image is known. Any quadrilateral can be divided into two triangles. Therefore, it is possible to texture map pixels from any quadrilateral shaped area into another quadrilateral shaped area of an image. In order to implement this it is necessary to be able to scan every pixel in an arbitrary triangle.

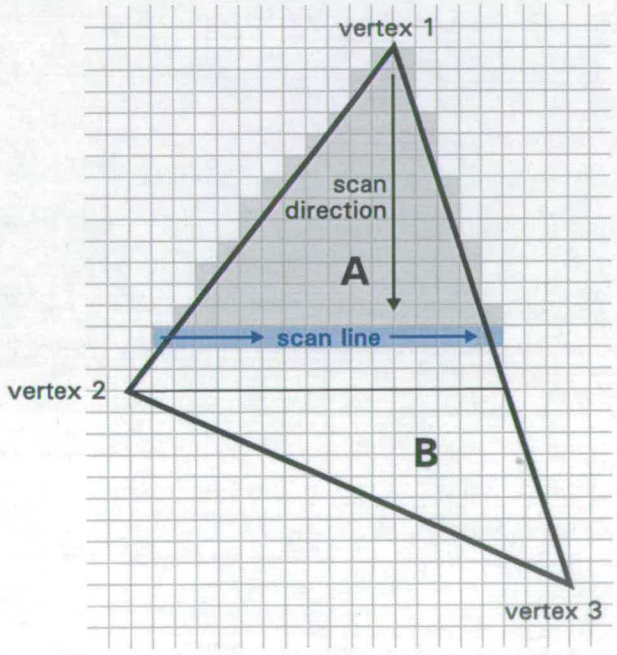


Figure 4.12: *Texture mapping a triangle by scanning through all the pixels*

Firstly, the triangle must not be a line or a point, i.e. the vertices may not be collinear. Secondly, the total area must be larger than a pixel. Once these checks have been performed, the vertices of the triangle are sorted in ascending y co-ordinate order. The triangle is then split into two triangles as in Figure 4.12. Each pixel in every scan-line is then visited first in the upper triangle (A) and then in the lower triangle (B). For each pixel position it is possible, using the transform $\mathbf{T}$ found in Equation 4.10 above, to find the corresponding pixel in the other triangle.

Separating the triangle into two triangles allows for efficient implementation by looping through the y co-ordinates (or scan-lines) and incrementing the left and right x bounds by the gradient of the left and right edges. At the mid-point (vertex 2 in Figure 4.12) the gradients need to be changed to those of the edges of the lower triangle. These gradients can be precalculated before scanning each pixel. This improves efficiency and removes the need to check the bounds of every scan-line.

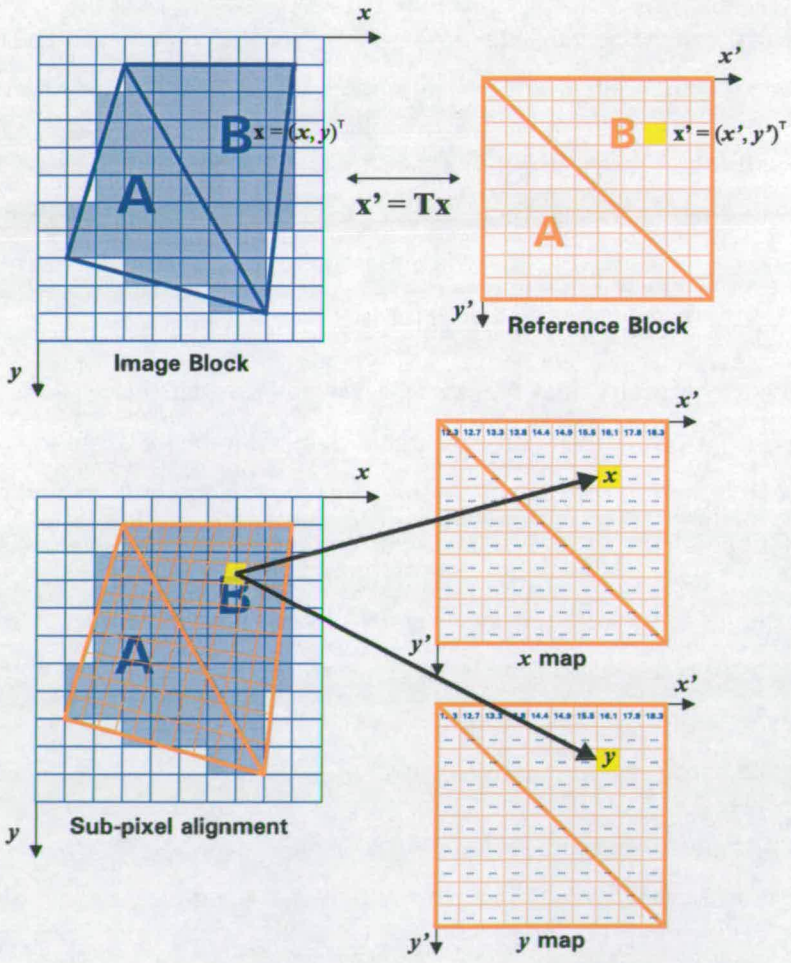


Figure 4.13: *Sub-pixel alignment when texture mapping*

It is then possible to build up a map of sub-pixel x and y co-ordinates of the source pixels from each pixel in the target image as in Figure 4.13. Using these maps, texture mapping can be performed by copying pixels from the source image to the target image. Pixel intensities can be found using bilinear interpolation or the nearest neighbour pixel intensity. For colour images, this process is repeated for each channel independently.

Generally in computer graphics, texture mapping is used to place textures on 3D wireframe models of objects to make them appear realistic. When the camera motion in a 3D scene consists only of pans, tilts and zooms it is possible to texture map these images of the scene onto planes, orientated to match the image planes they were created from.

4.5 Zoom Tracking Method 1 – Warp Search Area

Figure 4.14 and Figure 4.15 show a combined pictorial and flowchart representation of the algorithm. The first column contains the flowchart, the second column contains the co-ordinate system used and the third column is the pictorial representation of the operation.

The last known match position, in 3D world co-ordinates, is used as the starting point in the search for the block in the current frame. If the tracker is being initialised then the start position will be externally determined, for example a user clicking on an object of interest or an automated start position from a motion detector.

The start position is projected into the *current view* and a rectangular search area is created around it. The dimensions of this search rectangle should nominally be set to 15 – 35 pixels to constrain the search and improve speed. Although the search area is rectangular it is specified as a quad i.e. four points.

The search area quad is projected into the *reference view* at which point it may no longer be rectangular. As the search will be done in the *reference view* it is necessary to make the search area rectangular again for good performance. A bounding rectangle is fitted around the quad and this will be known as the *bounding rectangular quad.*

The *bounding rectangular quad* is projected into the *current view* – again note that it will not be rectangular in general. The resulting quad is known as the *projected bounding rectangular quad.* Pixels from the area inside the *projected bounding rectangular quad* need to be mapped into the area inside the *bounding rectangular quad.* This is done by finding the 3×3 transform

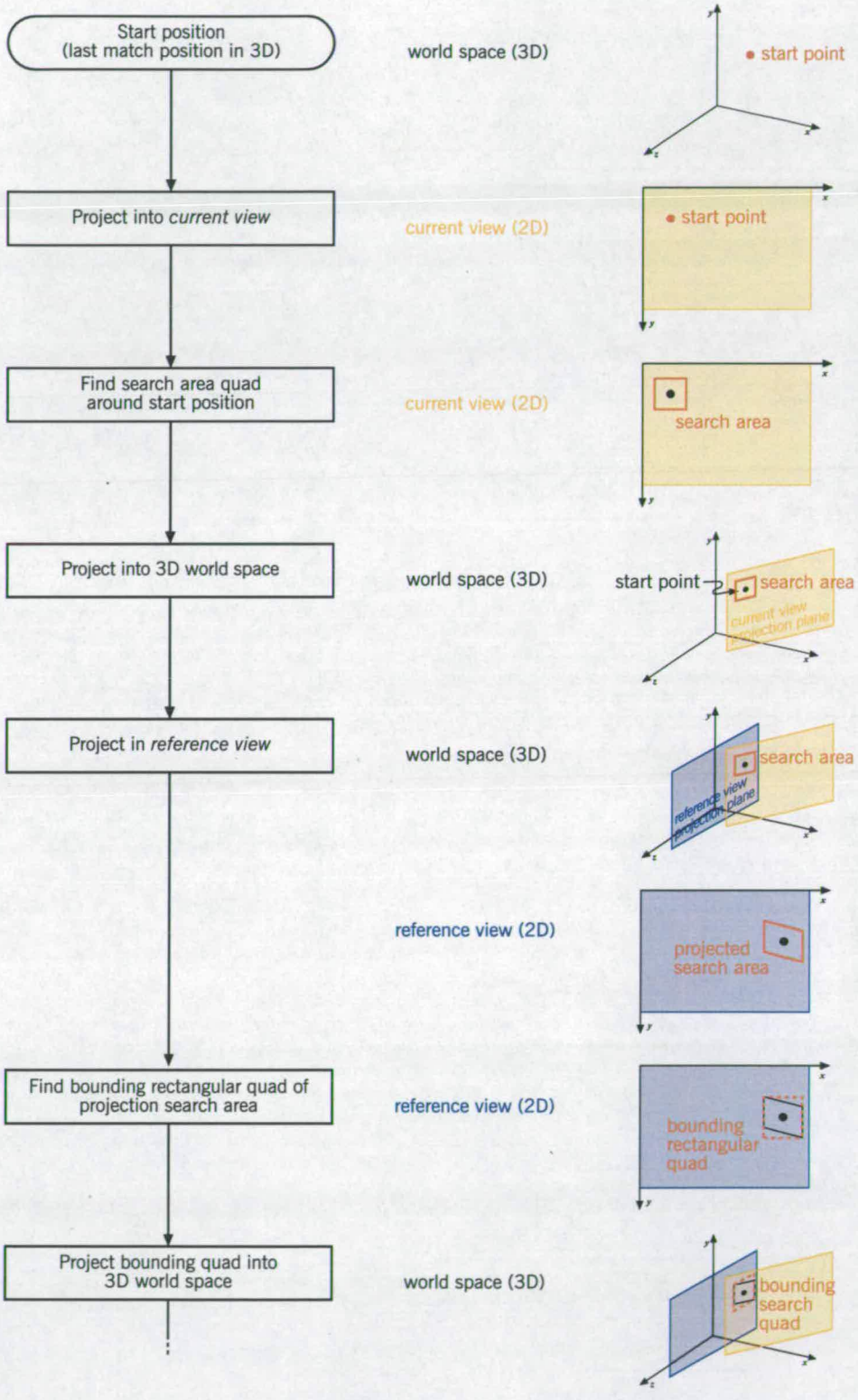


Figure 4.14: *Zoom tracking – Method 1 (Part 1)*

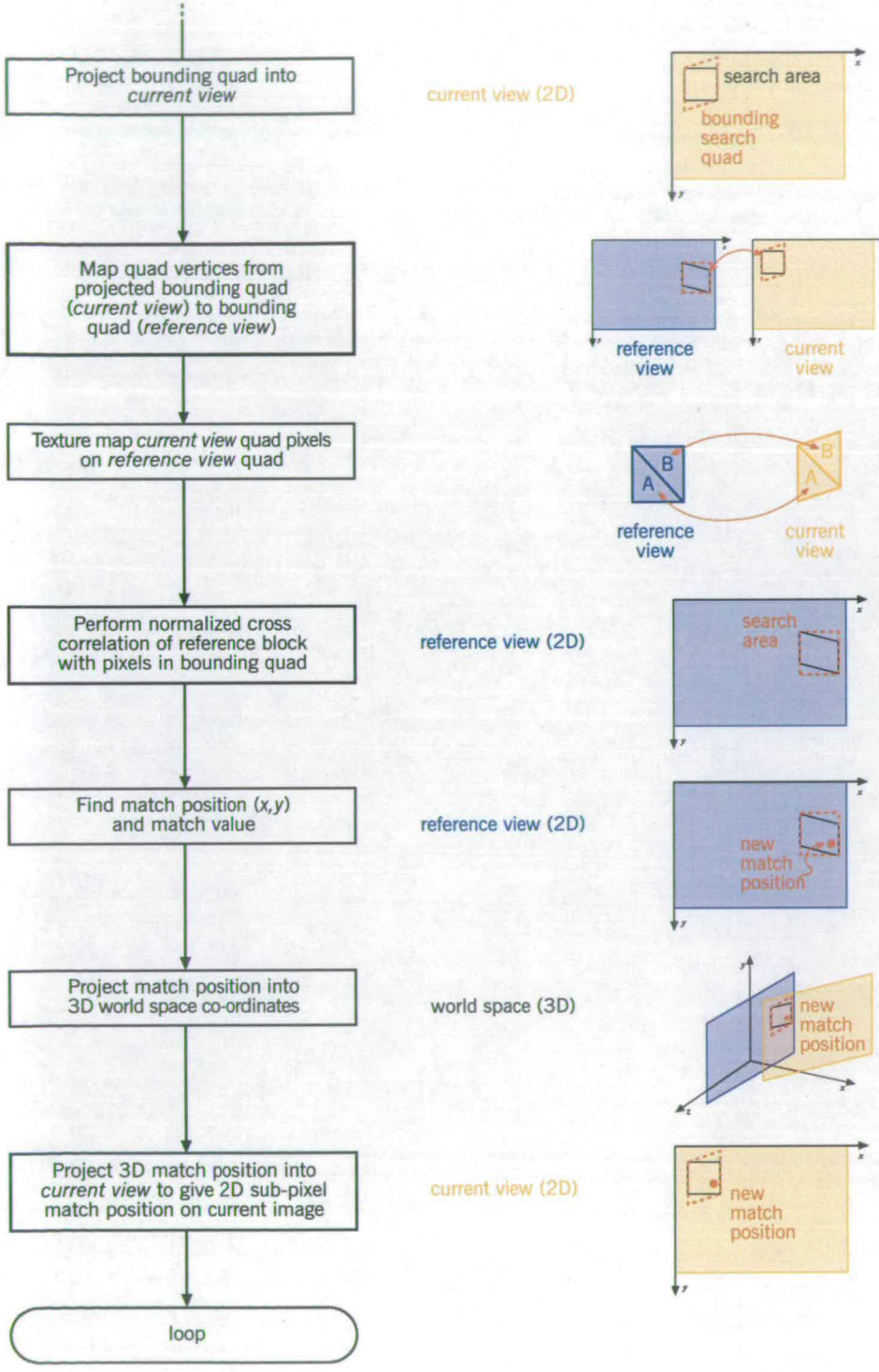


Figure 4.15: *Zoom tracking – Method 1 (Part 2)*

matrix between the points defining the two triangles contained in the each quad as discussed in Section 4.4.2. The quads are divided into two triangles and if the consistency of the edges has been maintained (i.e. the order of the vertices is left unchanged) the relative orientation of the triangles can be arbitrary, for example one triangle could map to a mirrored version of itself.

After mapping the pixels from the *current view* into the *reference view* it is now possible to perform a standard block match to find the best match position of the reference block. The match position found is located in the *reference view*, so to find the match position on the *current image* it is necessary to project the match position into the *current view*. The match position is also projected into world space co-ordinates and is used as the starting point in the next frame.

Outlier rejection is then performed to ensure that the match found is the true match based on the target position prediction. If the match is further than a distance threshold and has a match value lower than a threshold, another match candidate is chosen. The alternative match candidate is selected by finding the next best match according to match value. If the distance of the match position is within a second distance threshold of the predicted target position it is accepted as the correct match. This process is repeated until all match candidates have been exhausted, when the predicted target position is chosen as the match position. The new match position is then projected in the *current view* and world co-ordinates.

The match position in the *current view* is sub-pixel because the registration between the *current view* and the *reference view* will not be integral. Also, the *reference view* should be chosen so that it is at the highest zoom factor, resulting in interpolation when mapping pixels into the *current view*. If there is no interpolation then a loss of accuracy will occur because the match position will be quantised to the pixel pitch of the *reference view*.

4.5.1 Reference Block Updating

Updating the reference block is relatively simple, as during the matching process the search area was warped into the *reference view*. This means that conventional block updating methods can be used – the only subtle difference is that the size of the reference block will be large, for example if there is a $10\times$ scale change, the reference block for an 8×8 minimum block size, will be 80×80 pixels. This makes Kalman filtered block updates impossible to compute in real time, but other approaches such as FIR filters can be used – these are discussed in Section 4.2.7.

4.6 Zoom Tracking Method 2 – Warp Reference Block

As in Section 4.5 a combined pictorial and flowchart representation of the algorithm is shown in Figure 4.16 and Figure 4.17. The tracker is initialised with a starting point in world co-ordinates which is either the last match position or an initial object point provided externally. The reference block quad is projected from the *reference view* into the *current view*. The projected reference block quad in the *current view* is not guaranteed to be rectangular, so a bounding rectangle needs to be found before texture mapping.

As in Method 1, a 3×3 transform matrix is found that maps pixels from the reference block (in the *reference view*) to the transformed reference block in the *current view*. Again, the quad is divided into two triangles and each triangle is texture mapped from the *reference view* into the *current view*. During the texture mapping process, a mask is created that indicates which pixels in the rectangular transformed reference block are valid. This mask will be used to perform normalised cross correlation of the non-rectangular block with the image.

The transformed reference block (and its associated mask) can be used to perform standard block matching. Once a good match position has been found, a sub-pixel match is done around the match point.

Outlier rejection is then performed by requiring the match confidence to be above a threshold and that the match position lies within a certain distance of the predicted target position. If this is not the case, the next best match candidate is chosen if it is within a second threshold of the predicted target position. If no suitable match candidate can be found, the predicted position is used as the match position. The match position is then transformed into world co-ordinates and returned along with the match value.

4.6.1 Reference Block Updating

In Method 2, updating the reference block is slightly more involved than Method 1 as the matched block is in the original image and needs to be warped into the *reference view*. The match position is known in image co-ordinates, so the match position can be found in world co-ordinates and in the *reference view*. As the size of the transformed reference block changes depending on the camera view, it is necessary to project the reference block quad from the *reference view* into the *current view*. An assumption is made that the reference block lies in a

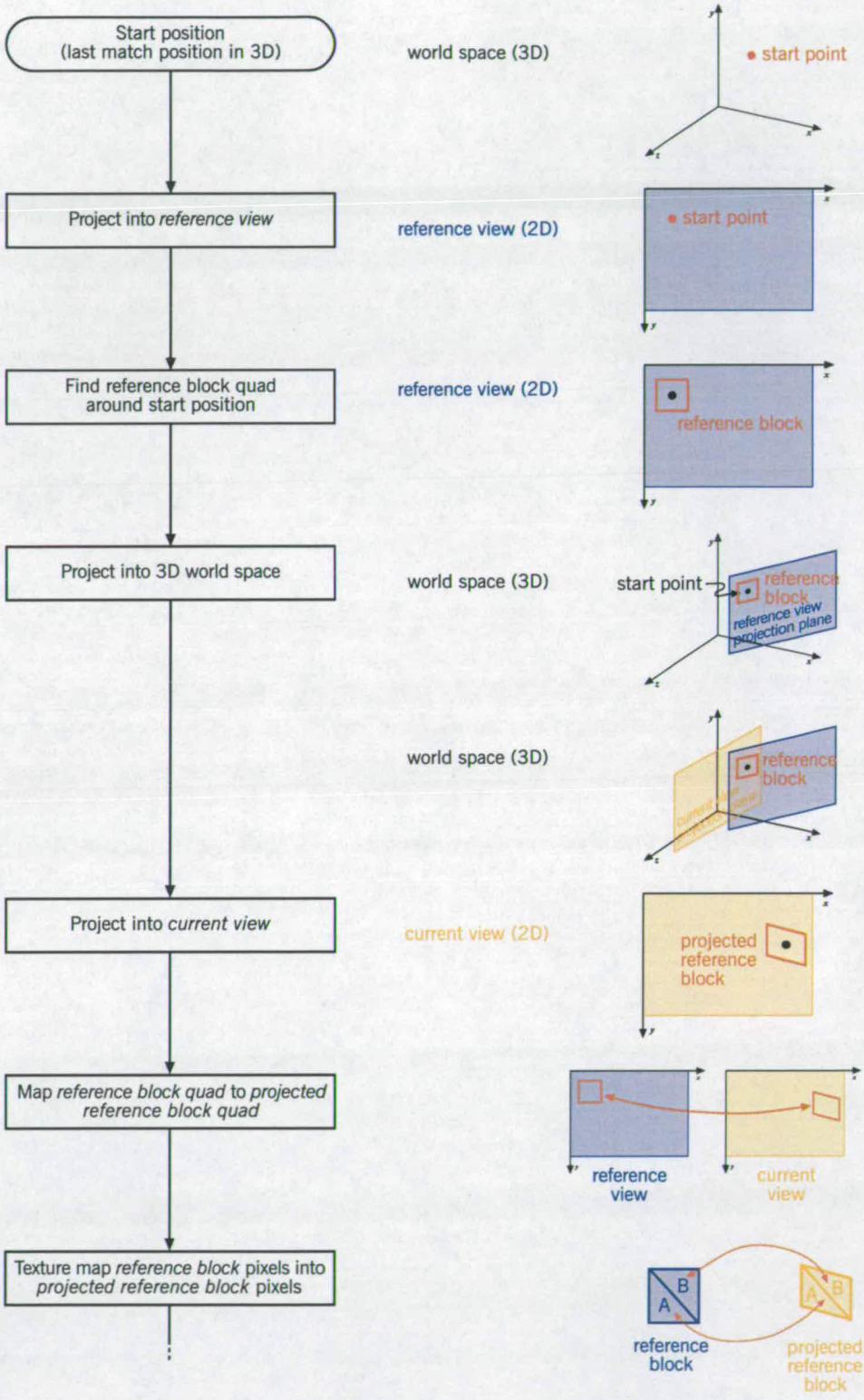


Figure 4.16: *Zoom tracking – Method 2 (Part 1)*

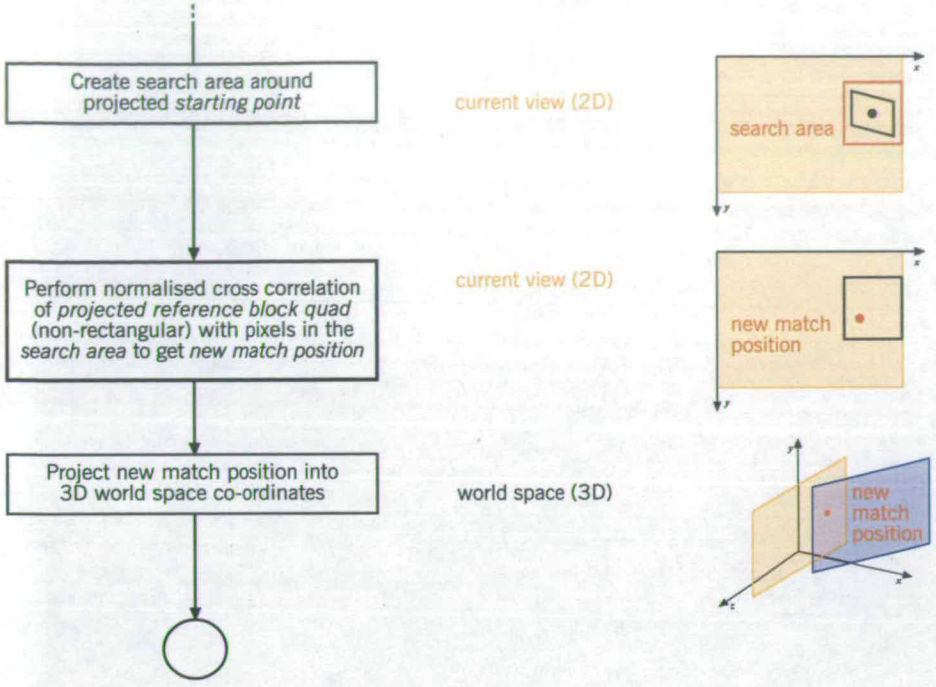


Figure 4.17: *Zoom tracking – Method 2 (Part 2)*

plane at the depth (or z co-ordinate) of the match position. This however is not strictly true, but given the size of the reference block the assumption is fair. It does introduce slight registration errors that can result in the reference block 'drifting' or becoming distorted after updates. This slight inaccuracy is a downside in the tradeoff for the vastly improved speed of Method 2. Once the reference block quad has been transformed into the *current view*, it can be divided into triangles as before and pixels from the current frame are texture mapped into the *reference view*. At this stage conventional block updating methods can be used to incorporate the pixels from the current view into the reference block.

4.7 Discussion

There is a clear distinction between Method 1 and Method 2: speed and accuracy. Method 1 is much slower than Method 2 as the entire search area needs to be texture mapped in to the reference view, whereas Method 2 only texture maps the reference block. However, Method 1 is more accurate as the match performed is implicitly sub-pixel accurate and it does not suffer from the blurring of the reference block due to warping as in Method 2.

As is clear from the results in Section 7.4 small amounts of drift can be introduced into the reference block by updating. These are due to two factors: slight mismatches due to interpolation and assuming the reference block lies in a plane at the depth of the match position's z co-ordinate. The mismatches due to interpolation are caused when the reference block is texture mapped into the *current view*. In most cases the reference block will be scaled down which results in defined edges becoming blurred. Even with sub-pixel matching, the match position can have errors of up to a pixel. When it comes to updating the block, all the pixels from the current view are offset by up to one pixel, causing the reference block to appear to drift after updating. As less smoothing occurs in texture mapping the reference block (when the *current view* approaches the same zoom factor as the *reference view*) this effect disappears and the drift decreases, as is visible in most results in Section 7.4.

The assumption that the reference block lies in a plane at a constant depth is also incorrect and this results in anisotropic distortion in the reference block which also causes drifting effects and smudging (or blurring). Suggestions to remove these effects are discussed in Section 8.3.

The performance of the multiscale methods presented depends largely on the accurate prediction of the target's trajectory in world space. Kalman filters have been used for the position

prediction/filtering, however more heuristic and rule based trajectory prediction could be used to make tracking more robust to occlusion for example. For the sequences presented in Chapter 7 it is not necessary to use another method more complicated than a Kalman filter. However, when tracking multiple targets or when there are multiple occlusions it would be necessary to use a more advanced position filtering and behaviour prediction system[150, 154].

4.8 Region Tracking

In security and surveillance applications it is common that there is some sort of background averaging and moving object segmentation that occurs[15, 56, 63, 65, 69, 72, 73, 154–156] to detect moving targets in a scene. For the most part, these systems use fixed view cameras, but in a practical CCTV surveillance system many cameras will be able to pan, tilt and zoom. Therefore, a basic region tracker will be investigated as a comparison to the novel block trackers as an alternative tracking method in a surveillance scenario. A rigourous treatment is beyond the scope of this thesis and a large body of work already exists on this subject, so only the basics will be discussed and a simple comparison explored.

The region tracking method discussed in this work was designed so that is could be implemented in such a way that it could potentially use OpenGL accelerated 3D rendering hardware to run in real-time. The current implementation only uses software rendering, as 3D video cards that can render to video memory and transfer this back to the operating system have only been available more recently. This feature is also only available from a small set of manufacturers and there is no agreed standard, meaning that any use of hardware to provide fast off-screen rendering needs to be implemented for a particular set of hardware. The methods described in this thesis are intended to be fairly general and not require specific hardware in the hope that in the future this type of off-screen rendering will be available in all hardware accelerated 3D graphics cards.

4.8.1 Average Scene Background

A simple region tracker subtracts an average of the scene (also known as the *average scene background*) from each frame to segment only the moving objects. This average scene should also be updated temporally, to allow for changes in illumination and to allow objects that have entered the scene and stopped moving, to become part of the 'background'. When the camera

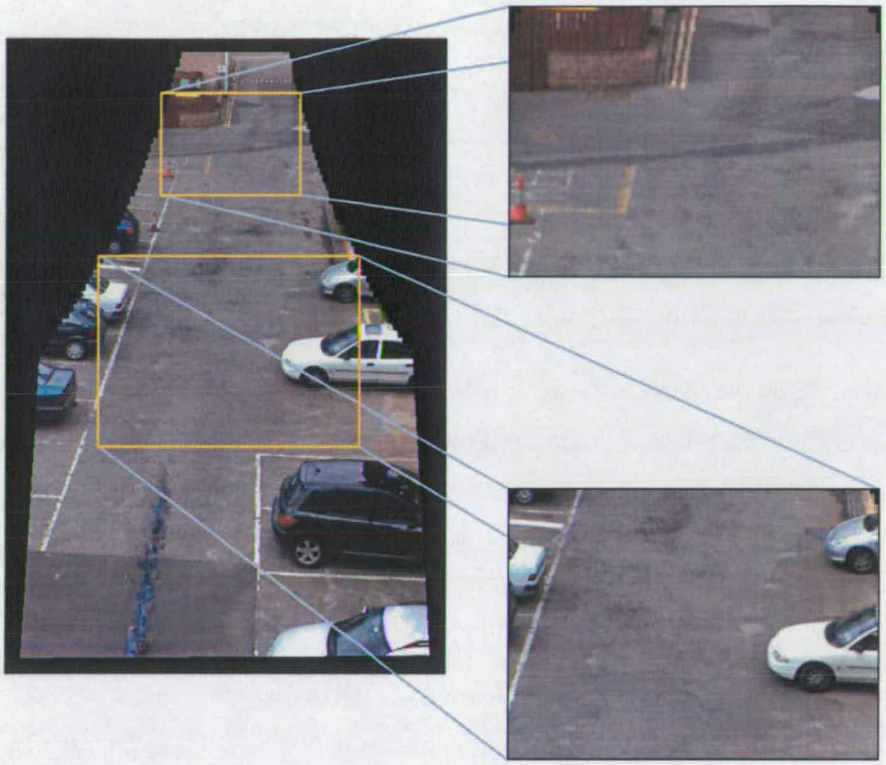


Figure 4.18: *Warping the average scene background*

view is static, creating an average scene is achieved by maintaining a frame in memory to which each live frame is added and averaged. The averaging can be done in an intelligent way[3, 72, 73] such that moving objects are not incorporated into the average.

In a situation where the camera is a PTZ camera it is necessary to maintain an environment map or a front cube face projection of the average[3]. Given that the camera can zoom, it may also be necessary to maintain a pyramid of environment maps for various zooms, however this depends on the application and level of detail needed in the segmentation of small objects. As mentioned in the Chapter 6 a by-product of calibrating the PTZ camera is a high resolution panorama (or mosaic), which can be used to create the average scene background.

4.8.2 Warping the average scene background

When using a PTZ camera, only a portion of the average scene background will be visible in a frame and it is necessary to warp the background so that it matches the orientation of the

[3]For a 360° view, a full environment map should be kept rather than just the front cube face as distortion occurs when the pan or tilt angle are too large.

current frame (as shown in Figure 4.18). This is done by texture mapping the average scene background onto the image plane using the current camera parameters, as is done in the block tracking of Section 4.4. This is identical to the method used in virtual reality viewers for panoramic scenes[48] and in environment mapping.

4.8.3 Segmentation and Target Identification

Finding moving objects in each frame is done by subtracting the current frame from the average. The difference frame is then thresholded and morphological operators are used to remove noisy pixels and to consolidate regions. A floodfill operation is then performed to segment each object in a region or connected group of pixels. Due to the errors in the segmentation, objects often fragment or split into several parts. [56, 63, 69] suggests ways of combining fragmented objects. Fragmentation is very common when a moving target moves in front of something of similar colour and texture. The segmented objects often contain regions of pixels that have large bounding boxes due to edge noise that has become connected into a large object. These objects are not of interest and can be easily removed as they are too large. Other annoying artifacts such as shadows caused by clouds or moving objects outside the scene are much more difficult the remove. In other cases, the scene can contain naturally moving objects such as trees or doors. Other segmentation methods based on object colour distributions are also common and have been explored by Raja[63], Xu[10] and McKenna[64, 123].

As with block tracking it is necessary to have some sort of target position prediction method to identify the object. Other constraints, such as the size of the object or number of pixels in the region, should also be included in a position filter such as a Kalman filter[148] or particle filter[151]. Based on some estimate of the target position and size, the list of segmented moving objects can be searched and the best match selected. Outlier rejection can then be performed if necessary and another match selected in much the same way as for a block tracker.

4.8.4 Dynamic background updating

In the practical running of a region tracking system the scene will change over time with variations in lighting throughout the day (i.e. changes from day to night) and local changes due to weather, such as rain and clouds. It is necessary to have a background averaging method that can run continuously. When using a PTZ camera it is necessary to warp the current frame into

the same reference view as the average scene background. This can be easily done using texture mapping after which normal averaging methods can be performed to merge the warped frame with the average. The OpenGL framework provides an easy way of doing this and the method is identical to that mentioned before in the block tracking methods of Section 4.4.2.

4.8.5 Comparison to Block Tracking

In many security applications the exact location of a portion of the target is not necessary, however in media post production high accuracy tracks are required. Accuracy can be reduced for an improvement in speed and the ability to follow multiple moving targets. Other benefits such as colour tracking[10, 64, 123] and human stick models[3] can be used to deal with difficult situations as multiple occlusions and groups of moving objects merging and separating.

In conclusion, the choice of when to use region tracking or block tracking depends largely on the application and the type of input images used. If a segmentation exists, then it makes sense to use region tracking. However, if the exact position of an object needs to be found accurate to a pixel or two, then block tracking is clearly the method that needs to be used.

4.9 Summary

In this chapter a summary of the terms and definitions used in conventional block tracking were presented and then expanded to include tracking scenarios with pan, tilt and zoom conditions. A review of previous work on zoom tracking was present before introducing and discussion two novel methods of tracking with a PTZ camera. A short discussion of region tracking methods and implementation was presented as a comparison method to the block tracking methods with application to security and surveillance systems.

The tracking methods described so far will use a method of calibrating a PTZ camera described in the following chapter to implement a tracking system which will be tested and discussed in Chapter 7.

Chapter 5
Contour Tracking

5.1 Introduction

In many sequences conventional block matching algorithms are able to track a target through a fair percentage of the sequence without very sophisticated algorithms. However, very few algorithms will be guaranteed to track a target from the beginning to the end of the sequence. Figure 5.1 shows that two separate block trackers, one starting from the first frame moving forwards and the other starting from the last frame and moving backwards – both fail to track the target for all the frames in the sequence. By combining or blending the results from these two trackers it would be possible to track the target over the entire sequence.

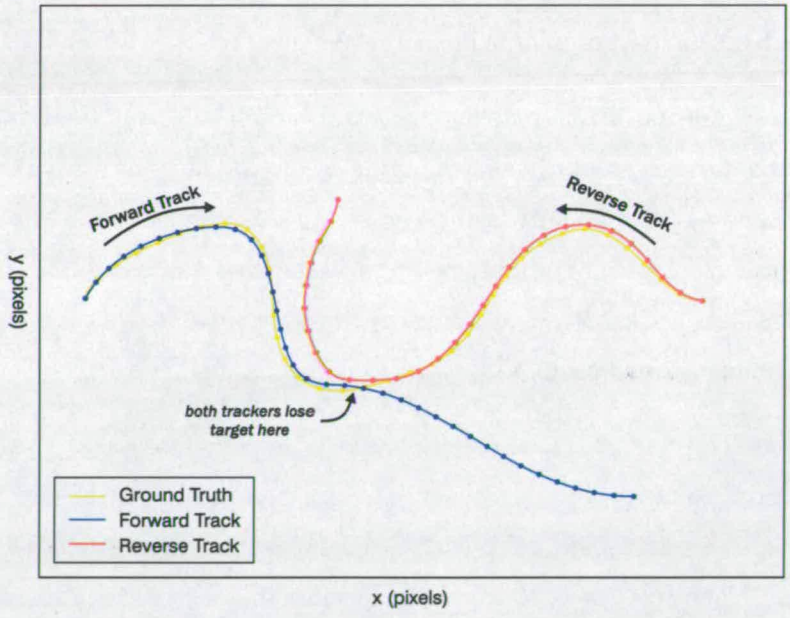


Figure 5.1: *Forward and backward tracking*

This concept can be taken further to use the information from both trackers simultaneously to guide the trackers to search for the target in regions of the image where it is likely to be, based on the fact that it must lie on a contour somewhere between the start and end points. With more

and more estimates, the shape of this contour will begin to approximate the true trajectory of the object very closely. As the estimation technique requires past and future frames it can only be used with offline sequences – real-time implementations are impossible.

In this section, the concept of a *tracking contour* will be presented for sequences where the camera view is fixed (Section 5.3) and under pan, tilt and zoom conditions (Section 5.4). The tracking contour is introduced in the context of using snakes[39] to fit contours to object boundaries. Snakes are discussed in Section 5.2 and their adaptation to tracking contours is presented in Section 5.3.1. Applications of the tracking contour for static cameras and PTZ cameras are present in Section 5.5. An implementation and results for the performance of a contour tracker on real sequences can be found in a later chapter on page 206 in Section 7.5.

5.2 Snakes fit to object boundaries

Kass[39] developed a method of fitting snakes[1] to object boundaries. Edge detectors often produce spurious edges and the boundary of an object is often fragmented due to the edge not being detected properly. A snake is resilient to missing portions of the object boundary and false edges as it finds the path of minimum energy along edges.

The movement of a snake is driven by two opposing forces: the *internal force* and the *external force*. The *internal force* is proportional and normal to the curvature of the snake and prevents the snake from bending too tightly by trying to push the snake into a circle. The *external force* is produced from the edge map and pushes the snake towards any edges. The *external force* is usually proportional to the derivative of the edge image so that the snake is pushed towards peaks in the edge map (which correspond to edges in the original image).

These two forces act on control points (which are treated as point masses) using Newtonian mechanics. The control points can be connected by straight lines, as implemented by Lobregt[157], or more commonly by a smooth function such as a cubic spline[115, 158, 159]. The position of the snake at any point can be interpolated by the function that joins the control points.

An iterative algorithm is used to fit the snake to edges. The snake is initialised to lie near the object boundaries in the image either by hand or by some other alternative technique. An

[1] Snakes are also referred to as *active contours*, however, to avoid confusing with active tracking contours, they will be referred to as only *snakes* throughout this thesis. Any reference made to *active contours* refers to an active *tracking contour*.

edge image is generated using an edge detector (for example, a Sobel operator). The *internal* and *external forces* acting on each control point are found using the edge image and from the local curvature of the snake through each control point respectively. A Newtonian mechanics model is used to allow the resultant force to act on the snake and allow it to accelerate to a new position. The snake is allowed to change size and contour points are added or removed to keep the length of sections between them within set limits. This process is repeated until the snake is at rest, which can be measured by calculating its kinetic energy.

The *external force* can be generated from information other than edges. Malpica[160] used optical flow as an external energy field to cause the snake to settle along the boundary of differently moving objects. Colour and texture information was used by Zhou[161] to generate the external energy field.

5.3 Contour tracking with a fixed view camera

The following section will introduce a novel tracking method using the tracking contour for a camera that remains fixed. The central ideas of snakes are extended to define the tracking contour. A description of the algorithm can be found in Section 5.3.5. The implementation of this algorithm and results can be found in a later chapter in Section 7.5. Using a tracking contour under pan, tilt and zoom conditions is discussed later in Section 5.4.

5.3.1 Adapting snakes to tracking contours

The tracking contour described in this thesis is slightly different to a snake: a tracking contour has a start point and an end point (whereas most snakes are closed contours) and the *external forces* are generated from a map of match values obtained from a block matching algorithm. The contour is defined by control points which are fixed in time but allowed to move in space (x and y image co-ordinates). For example, if there are 20 frames between the start and end point and there are 8 control points, there will be a control point every 2.5 time steps.

Figure 5.2 shows a tracking contour passing through a few frames with the match value field visible. The start and end points are set by a human operator. These positions are used to create two reference blocks for each end of the contour for use in block matching. Areas of good matches (high correlation) create forces which pull the tracking contour towards them in each

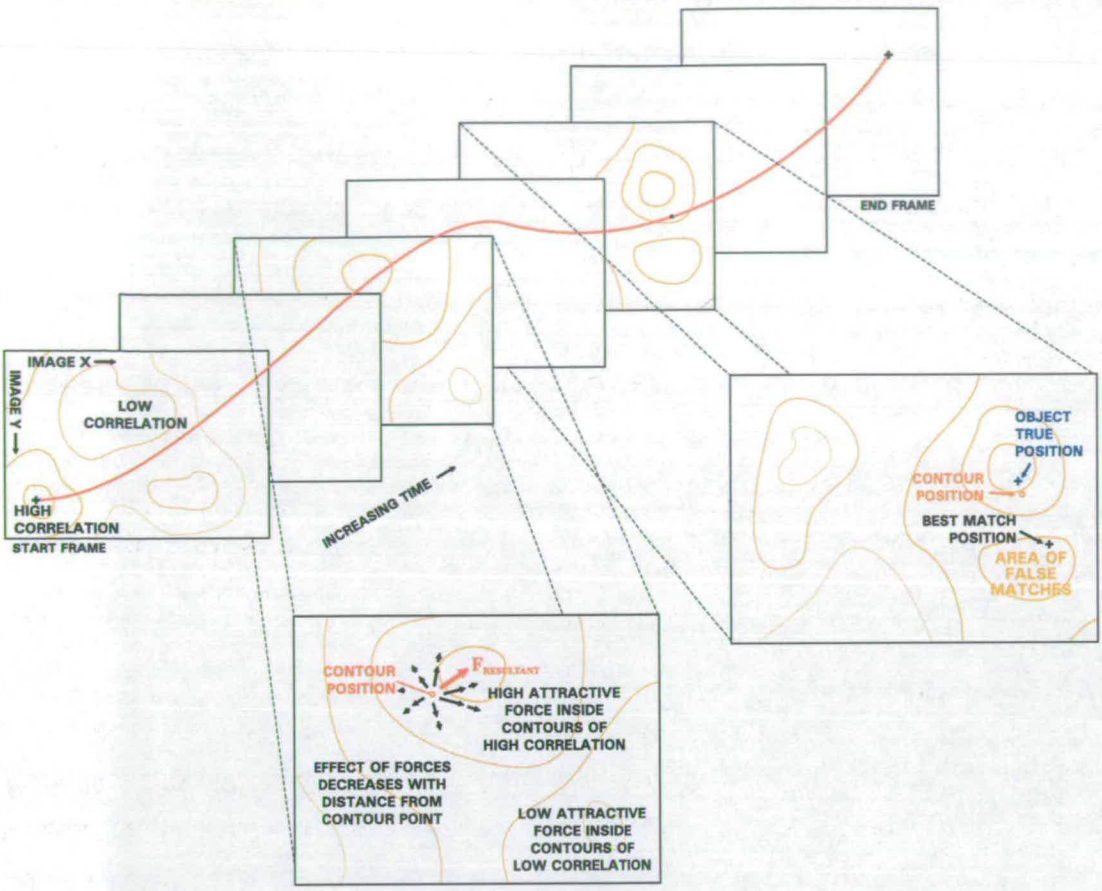


Figure 5.2: *Tracking contour*

frame. The magnitude of the force is also proportional to the distance between the contour and an area of good matches. Areas of poor match will not produce a large force and the contour will therefore not be attracted to this area as much as to areas of good match. Due mainly to the *internal forces* the tracking contour is not always pulled towards the global maximum correlation in each frame. It will instead by pulled towards the local maximum. This unique property allows the contour to be resilient to areas of false match in a small number of frames – these mismatches would normally cause a conventional block tracker to match to the wrong target.

5.3.2 Control Points

The shape of the contour is controlled by a small number of control points, which could be used to hand-edit the track if necessary. Each control point is modelled as a rigid point mass with

acceleration, velocity and position. A resultant force, calculated from the internal and external force, acts on each point mass to move the contour to a new position. In summary, each control point has the following attributes:

1. **Time** – temporal position in the sequence (it is allowed to be located between frames, i.e. non-integer values are allowed)

2. **Position** – x and y pixel co-ordinates in the image frame

3. **Velocity** – the velocity is calculated in each time step and is constrained to be only in the $x - y$ plane

4. **Acceleration** – the acceleration in each time step is computed, but constrained to be in the $x - y$ plane only

5. **Mass** – the control point is considered to be a rigid point mass

The number of control points chosen affects the smoothness of the contour. Many control points will allow the contour to overfit to the match data and too few control points will produce an overly smooth curve. The optimal number of control points is entirely dependent on the object motion – if the motion is simple, few control points should be used and if the motion is complicated, many control points should be used. A large number of control points also makes the contour difficult to hand-edit by a human operator.

The following kinematic laws govern the motion of the control point[2]:

$$\mathbf{a}(t) = \frac{\mathbf{F}_{\text{resultant}}(t)}{M} \tag{5.1}$$

$$\mathbf{v}(t + \Delta t) = \mathbf{v}(t) + \mathbf{a}(t)\Delta t \tag{5.2}$$

$$\mathbf{s}(t + \Delta t) = \mathbf{s}(t) + \mathbf{v}(t + \Delta t) \tag{5.3}$$

[2]For the tracking contour $\Delta t = 1$ as the time base used is the frame number and the 'time' between frames is one frame. Therefore, the time t has no physical significance or dimension such as seconds as it has been discretized into equal time steps. This makes calculations simpler as all the dt terms disappear. However, if the frame rate of the sequence is not constant, then the true time in seconds of each frame would need to be used and Δt included in all kinematic expressions.

where	Δt	time step
	$\mathbf{a}$	acceleration vector
	$\mathbf{v}$	velocity vector
	$\mathbf{s}$	displacement vector
	$\mathbf{F}_{\text{resultant}}$	resultant force acting on the object
	M	mass of the object

The kinematic model for the control point is calculated at each step in the iterative process of fitting the contour, allowing the contour to move realistically.

5.3.2.1 Cubic Spline Fit To Points

The following section contains the method described by Eric Weisstein[158] and Bartels[159] of fitting a cubic spline interpolating function to data. Given a set of 1-dimensional data, it is possible to fit a piece-wise interpolating cubic spline function $Y_i(x)$ with n knots (or $n-1$ splines). A natural cubic spline is fit to the data as the start and end gradients are unknown and will be assumed to be zero.

The number of knots in the interpolating function is equal to the number of data points, therefore, the interpolating function passes through all the data points. The interpolating function is made of up of several cubic splines, as defined in Equation 5.4, with the gradient remaining constant across the knots to create a continuous smooth function.

In order to find the interpolated function y value for a given x value, a binary search needs to be performed to find the i-th cubic spline which is used to interpolate that portion of the function. The i-th cubic spline is then given by:

$$Y_i(t) = a_i + b_i t + c_i t^2 + d_i t^3 \tag{5.4}$$

$$t(x) = \frac{x - x_i}{x_{i+1} - x_i} \tag{5.5}$$

$$a_i = y_i \qquad c_i = 3(y_{i+1} - y_i) - 2M_i - M_{i+1}$$

$$b_i = M_i \qquad d_i = 2(y_i - y_{i+1}) + M_i + M_{i+1}$$

where	x	value of x from which to interpolate function y value
	i	i-th piece of the cubic spline where $i = 0, \ldots, n-1$ found by a binary search of x values
	t	a parameter from $[0; 1]$ indicating the distance along the i-th piece of the cubic spline
	x_n	n-th data point x value
	y_n	n-th data point y value
	a_i, b_i, c_i, d_i	spline fit parameters
	$\mathbf{M}$	weighting vector as calculated by Equation 5.6

Boundary conditions on the gradient of each cubic spline enforces smoothness over the interpolating function. Re-arranging the expressions above with boundary conditions results in the tridiagonal system of Equation 5.6. The weighting vector $\mathbf{M}$ can then be found by finding the QR-decomposition of matrix $\mathbf{A}$ and back-substituting with the vector $\mathbf{Y}$ created from the y values of the data.

$$\mathbf{AM} = \mathbf{Y} \tag{5.6}$$

$$\begin{bmatrix} 2 & 1 & 0 & \ldots & 0 & 0 & 0 \\ 1 & 4 & 1 & \ldots & 0 & 0 & 0 \\ 0 & 1 & 4 & \ldots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & 4 & 1 & 0 \\ 0 & 0 & 0 & \ldots & 1 & 4 & 1 \\ 0 & 0 & 0 & \ldots & 0 & 1 & 2 \end{bmatrix} \begin{bmatrix} M_0 \\ M_1 \\ M_2 \\ \vdots \\ M_{n-2} \\ M_{n-1} \\ M_n \end{bmatrix} = \begin{bmatrix} 3(y_1 - y_0) \\ 3(y_2 - y_0) \\ 3(y_3 - y_1) \\ \vdots \\ 3(y_{n-1} - y_{n-3}) \\ 3(y_n - y_{n-2}) \\ 3(y_n - y_{n-1}) \end{bmatrix}$$

The tracking contour, however, is a 2-dimensional function. Therefore, to find the 2D interpolating function, a cubic spline is fitted to the data for each dimension independently. Although this is a solution, the more correct solution would be to use the method described in Numerical Recipes[115] for interpolating multidimensional functions.

For a multidimensional function Equation 5.4 becomes:

$$\mathbf{Y}_i = \mathbf{a}_i + \mathbf{b}_i t + \mathbf{c}_i t^2 + \mathbf{d}_i t^3 \tag{5.7}$$

$$\begin{bmatrix} Y_{i_x} \\ Y_{i_y} \\ Y_{i_z} \\ \vdots \end{bmatrix} = \begin{bmatrix} a_{i_x} \\ a_{i_y} \\ a_{i_z} \\ \vdots \end{bmatrix} + \begin{bmatrix} b_{i_x} \\ b_{i_y} \\ b_{i_z} \\ \vdots \end{bmatrix} t + \begin{bmatrix} c_{i_x} \\ c_{i_y} \\ c_{i_z} \\ \vdots \end{bmatrix} t^2 + \begin{bmatrix} d_{i_x} \\ d_{i_y} \\ d_{i_z} \\ \vdots \end{bmatrix} t^3$$

where t as defined above in Equation 5.5

$\mathbf{a}_i, \mathbf{b}_i, \mathbf{c}_i, \mathbf{d}_i$ vectors of fit parameters

The cubic spline fit is performed using Equation 5.6 for each dimension independently.

5.3.3 External Forces

Tracking contours are pulled towards areas of high correlation with the reference blocks (whereas snakes are pulled towards edges). Normalised cross-correlation (NCC), as described in Section 4.2.5, is used to match the reference block to blocks in each frame. For each pixel in the match force image, the normalised cross-correlation of the image block with the reference block is given by:

$$c(x,y) = \tfrac{1}{2} - \frac{\sum_{ij}\left[(R_{ij}-\bar{R})\cdot(T_{ij}-\bar{T})\right]}{2\sqrt{\sum_{ij}(R_{ij}-\bar{R})^2 \cdot \sum_{ij}(T_{ij}-\bar{T})^2}} \tag{5.8}$$

where c normalised cross-correlation (NCC) in the range $[0;1]$ for a block at position (x,y)

R_{ij} Reference block indexed by pixel in row i and column j

T_{ij} Test block in the image also indexed by pixel ij

n Total number of pixels in the reference block

The external force $\mathbf{f}$ acting at each point $\mathbf{p}$ in the image frame is calculated as follows, each pixel at position $\mathbf{t}$ within a set distance of $\mathbf{p}$ exerts a force $\mathbf{f}$ on $\mathbf{p}$ given by:

$$\mathbf{f}(\mathbf{p},\mathbf{t}) = \frac{(\mathbf{p}-\mathbf{t})\cdot c(\mathbf{t})}{\|\mathbf{p}-\mathbf{t}\|^{\alpha}} \tag{5.9}$$

where $\mathbf{f}$ external force acting at point $\mathbf{p}$ in the image in the direction $\overrightarrow{tp}$

$\mathbf{p}$ position vector of the point in the image where the external force is acting

$\mathbf{t}$ position vector of a block with match value c

c NCC match value of a block at position $\mathbf{t}$ in the range $[0; 1]$ at position $\mathbf{t}$

α a constant with a value greater than 1

i, j pixel indexes used for iteration in summing all the forces $\mathbf{f}$ over a range of pixels (explained below)

The above expression generates a force $\mathbf{f}$ that acts in the direction $\overrightarrow{tp}$ and is attenuated by the distance to the match and its match value. Therefore, the contour is affected by two factors:

1. Matches nearer the contour will have more influence than those further away
2. Better match values (high values of c) will produce greater forces

The total force acting at any point $\mathbf{p} = (x, y)$ is found by summing the forces $\mathbf{f}$ for all nearby pixels:

$$\mathbf{F}(\mathbf{p}) = \sum_{\mathbf{t}} \mathbf{f}(\mathbf{p}, \mathbf{t}) \tag{5.10}$$

where $\mathbf{F(p)}$ total external force acting at point $\mathbf{p} = (x, y)$ in each frame

$\mathbf{t}$ position vector of all pixels in the image

$\mathbf{f(p,t)}$ match force at $\mathbf{p}$ due to a match at $\mathbf{t}$ as defined above in Equation 5.9

Since finding the cross correlation values is computationally expensive it is recommended that these are cached for reuse in subsequent iterations.

Two minor modifications were made to ensure better results. Firstly, it was found that large areas of medium correlation tend to dominate over small areas of high correlation causing

the match force to act away from the maxima and towards more average values. This can be prevented by ignoring all pixels which have a correlation lower than a threshold (a value of 0.7 was found to be good). Secondly, pixels are ignored where the correlation value is lower than that given for a block based around point $\mathbf{p}$. Thus, the force cannot act to pull $\mathbf{p}$ towards a worse match than its current position.

5.3.4 Internal Forces

The position of the control points needs to be influenced by every frame in the sequence and not just by nearby control points. A method of distributing the match forces in each frame to the control points is needed. This is achieved by implicity implementing the internal forces on the contour. By distributing the force from the frame to every control point, smoothness is maintained. The distribution is weighted so that control points closer in the sequence to the frame contribute more match force than those further away in time.

The expression below is used to calculate the distributed match force $\mathbf{f}_{k_n}$ on each control point c_k for frame n:

$$\mathbf{f}_{k_n} = u \cdot \mathbf{f}_n \cdot (1 + m - |t_n - t_{c_k}|)^{\beta} \tag{5.11}$$

$$u = \Big(\sum_h (1 + m - |t_n - t_{c_h}|)^{\beta} \Big)^{-1} \tag{5.12}$$

where

h or k	index of each contour control point
t_n	time index for the n-th frame in the sequence
t_{c_k}	time index for control point k
$\mathbf{f}_{k_n}$	match force distributed to control point k from frame n
$\mathbf{f}_n$	match force in frame n
m	maximum difference between the time index of control point c and t_n
β	factor to control the rate of fall-off
u	fall-off factor

The total force acting on each control point k is given by summing all the forces in the above expression:

$$\mathbf{f}_k = \sum_n \mathbf{f}_{kn} \tag{5.13}$$

It was found that adding a retarding force proportional to the contour velocity was useful in speeding up convergence and improving stability by damping the contour motion. This force is analogous to an air resistance force which is proportional to the contour velocity and acts in the opposite direction. Therefore, the total force acting on each control point is given by:

$$\mathbf{F}_k = \mathbf{f}_k - \rho \mathbf{v}_k \tag{5.14}$$

where ρ air resistance coefficient
$\mathbf{f}_k$ distributed match force acting on the control point k
$\mathbf{v}_k$ the current velocity of the control point i.e. before applying the force

A typical value for the air resistance ρ is 0.15. If this value is made too large the contour will never settle and a negative value will result in massive underdamping and chaotic behaviour, therefore ρ should always be positive.

5.3.4.1 Kinetic energy to measure convergence

The kinetic energy of the contour can be used to calculated whether it has converged to a solution or not. When the contour has settled on a solution it will no longer be in motion, therefore it will have no kinetic energy. The kinetic energy of the contour can be calculated by:

$$K_{\text{contour}} = \sum_{n=1}^{N} \tfrac{1}{2} M_n \mathbf{v}_n \cdot \mathbf{v}_n \tag{5.15}$$

where n index of each control point
N total number of control points
M_n mass of the n-th control point
$\mathbf{v}_n$ velocity vector of the n-th control point

If the point masses are all assumed to have a mass of one, then the above expression simplifies to the sum of the velocity magnitudes squared for each control point. This simplification should

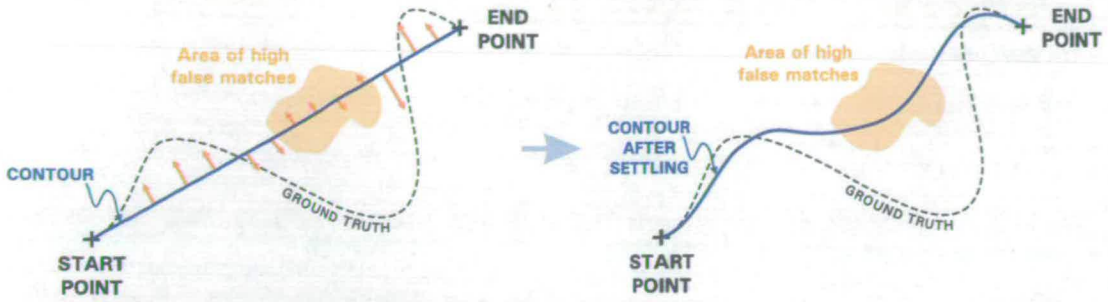


Figure 5.3: *Where the entire contour is settled simultaneously, the contour may be attracted towards areas of false high matches in the centre of the contour (the time axis is not shown in this figure)*

only be used when all other kinematic constants are dimensionless and have no bearing on physical reality.

5.3.5 Iterative Algorithm

The algorithm for implementing the tracking contour is shown in Figure 5.5. Note that the forces are not calculated in one step but in multiple passes. The reason for this is illustrated in Figure 5.3. In this case there is an area of false matches close to the initial position of the contour, shown as a dotted line. If forces from the entire contour are considered from the beginning, the contour will be attracted towards this area even though it is some distance from the true position of the contour; the correct matches will be too far from the position of the contour for it to be attracted towards them. Since the position of the contour is known at the end points, frames close to these points are used first. Subsequently more points are used to refine the contour position. This ensures that the contour is always close to points being considered. By the time the position of the contour in the middle of the sequence is considered, it will have moved away from the area of false matches.

Figure 5.4 shows this process in operation. Initially, the contour forms a straight line between the end points. In the first iteration, only the first and last frames are in the active set and so only these two frames act on the contour. As time progresses, more frames are included in the active set until all frames are acting on the contour.

In the case that the final path of the contour is close to the initial linear path, this process is not necessary and the contour can be resolved in a single step for speed.

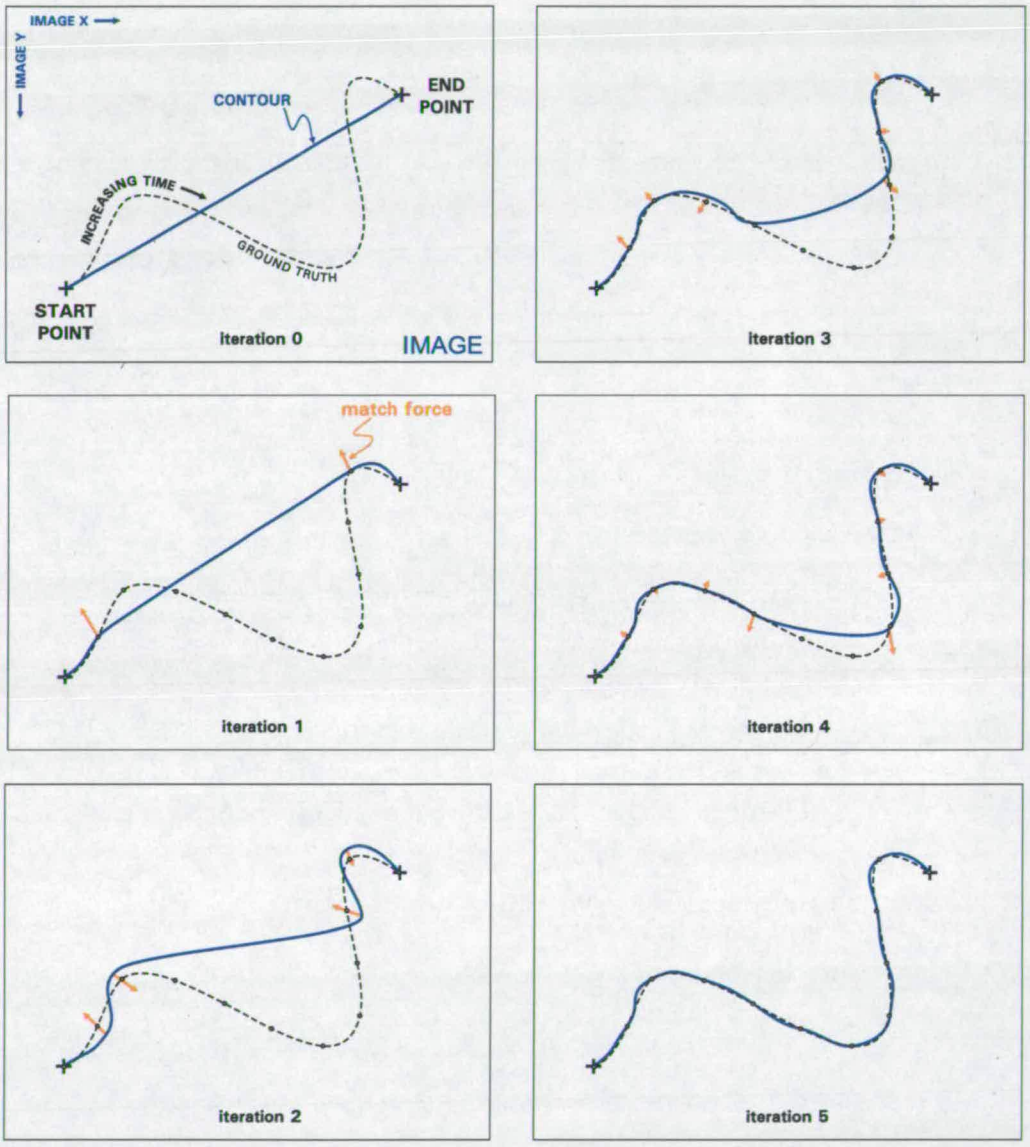


Figure 5.4: *Progressively more and more frames provide forces (shown as arrows) which cause the contour to move*

1. User indicates start S_{ij} and end E_{ij} positions of the contour in the start frame s and end frame e
2. Extract reference blocks around points S_{ij} and E_{ij} from the start and end frames
3. Initialise the contour as a straight line between S_{ij} and E_{ij} with n evenly spaced control points
4. Set of active frames = $\emptyset$
5. Set of inactive frames = all intermediate frames
6. While there are frames in inactive set
   (a) Remove first and last frames from inactive set and add to active set
   (b) For several hundred iterations or until contour settles:
      i. For each frame in active set
         A. Let $\mathbf{P}_n$ be the position of contour in frame n
         B. Find $\mathbf{f}_n$, the force acting on point $\mathbf{P}_n$ (see Section 5.3.3)
         C. Distribute force $\mathbf{f}_n$ across all control points (see Section 5.3.4)
      ii. For each control point k
         A. Let $\mathbf{T}_k$ be $\mathbf{f}_k - \rho \mathbf{v}_k$ where $\mathbf{v}_k$ is the velocity of control point k and ρ is a constant
         B. New velocity of contour point $\mathbf{v}_k \leftarrow \mathbf{v}_k + \mathbf{T}_k / m$ where m is a constant
         C. New position of contour point $\mathbf{c}_k \leftarrow \mathbf{c}_k + \mathbf{v}_k$

Figure 5.5: *Algorithm for contour tracking in a fixed view scene*

5.3.6 Using the contour for interframe position interpolation

Often in post production applications it is necessary to know what a target's trajectory is between frames, for example in sequences where there is slow motion. The interframe position of the target can be used to place rendered synthetic objects in such a way that they look realistic. The contour tracker allows this to be done easily, as the contour is a continuous smooth function over the entire range of the targets movement. The interframe position is obtained by evaluating the contour position at a frame position that is not an integer.

Due to the smoothness constraints of the cubic spline fit, the interframe movement of the target will be consistent with the rest of its trajectory in other frames i.e. the acceleration and velocity of the object will be correct. One caveat is that the object must have a smooth trajectory. If it does not have smooth motion, then the contour will over-smooth the motion and could produce odd visual artifacts. Also, if the contour has too many control points it can be overfitted to the data and the interframe motion will be inaccurate as the contour is attempting to pass through too many control points rather than accurately reflecting the target's smooth trajectory.

5.3.7 Alternative kinematic models

The current model of the contour uses implicit smoothness constraints by distributing the forces along the contour using Equation 5.11. An alternative is to use an explicit model of point masses joined by springs, as shown in Figure 5.6. The contour could also be view as a continuous elastic string as shown in Figure 5.7. Each of these alternative models is briefly discussed further in Section 8.3 as future areas of research.

The spring contour model would be treated in the same way as the contour tracker described above, where the match force in each frame is distributed and applied to the point masses of each control point in turn. This implicitly models the contour as a string of constant density that has been transformed into a series of connected point masses.

An elastic contour explicitly models the contour as a continuous elastic string. The match forces in each frame would be applied to points along the contour and an integral would need to be performed to sum the effect of the point forces along the contour. The contour would be assumed to have constant elasticity and density along its length and would be constrained to move only in the x and y directions, i.e. there would be no temporal motion, which is easily modelled as zero elasticity in the time direction.

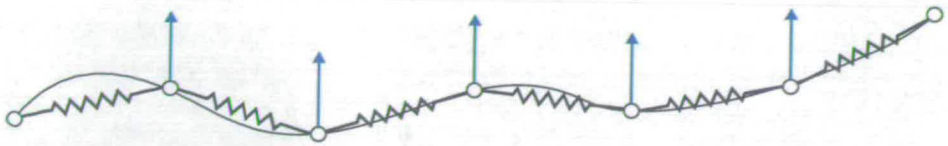


Figure 5.6: *Alternative contour kinematic spring model*

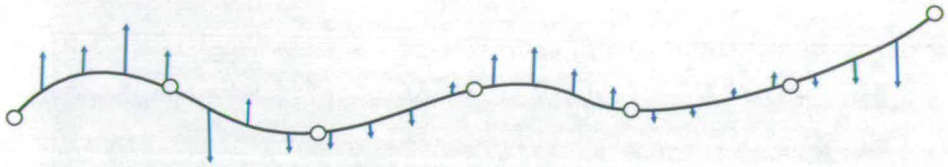


Figure 5.7: *Alternative contour kinematic elastic model*

5.3.8 Limitations

As with any tracking system, the contour tracker will not perform perfectly under all conditions. There are some limitations that will be discussed briefly below with reference to some possible solutions to these limitations presented in Section 8.3. The contour tracking method can only work with offline sequences, as all frames are needed simultaneously to find the contour. This makes it impossible to use in real-time sequences as future frames would be required.

Another constraint is that the target must have a smooth trajectory for the algorithm to be accurate. If there are step changes in the trajectory, due to erratic camera motion from a hand-held camera for example, the cubic spline contour will attempt to fit to the step changes resulting in *looping* effects as shown in Figure 5.8. If the camera motion can be modelled, as with the PTZ camera of Section 5.4.3, the contour tracker could be used as the global motion of the object would be smooth.

If the frame rate is low, the same motion step changes described above would be present. This could be the result of two scenarios: the first is poor quality hardware that can only achieve low frames rates (e.g. a cheap CMOS camera from a mobile phone) or secondly, where the object is moving extremely quickly relative to the frame sampling rate. In both cases, the contour tracker is likely to fail to track the target accurately.

If the sequence contains many large areas of false matches, the contour could possibly not settle on a single position. Areas of false matches could be the result of many similar objects in a

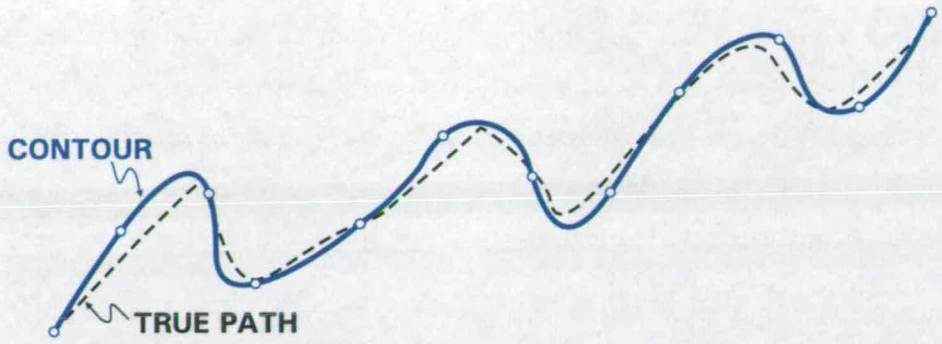


Figure 5.8: *Limitations of contour tracker*

the scene, such as tracking a human head in a crowed street scene. By initialising the contour carefully and reducing the area of effect (or search area) of the match values, the contour performance could increase in these situations, however it would be largely dependent on the content of the sequence.

Unfortunately there is no easy way to estimate what the choice of the physical constants for the control point masses, air resistance and fall-off factors should be. These parameters were intended to be used as interface *sliders* in a post production package, such as Apple's Shake[162], where a human operator would manually adjust the parameters until a good track was achieved. The sensitivities of the contour to these parameters and methods of estimating them based on the content of the sequence was not investigated but could be an area of future research, as discussed in Section 8.3.

Computational costs are very high in this method as it is necessary to calculate a much larger number of cross correlation values than in normal block matching algorithms. Even though a caching system can be used to store match values, the computational time runs into several hours for a 100 frame sequence on a quiet dual processor Intel Xeon 1.8GHz Linux-based PC to track the entire sequence. Performance increases could be achieved by restricting the number of matches calculated around the contour or by parallelising the algorithm for use on multi-machine computing farms that most post production houses now use for 3D rendering and video processing.

5.4 Contour tracking with a pan/tilt/zoom camera

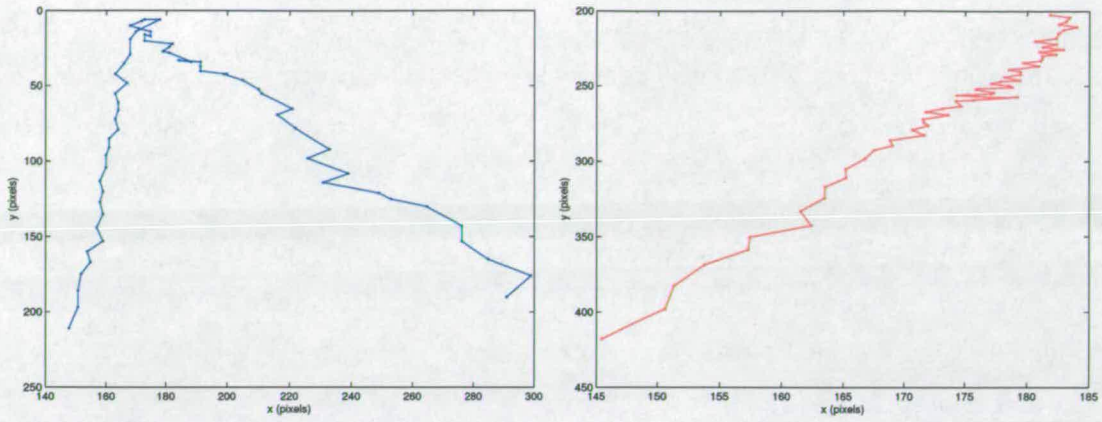
When tracking a sequence under pan, tilt and zoom conditions the contour tracker described above could fail as the target's trajectory may become extremely complex due to apparent movement caused by the camera panning, tilting and zooming. It can be seen from the examples in Section 5.4.1 and Figure 5.9 that by using a model of the camera to fit a contour in world co-ordinates, a complex target trajectory becomes very simple.

This section outlines an extension to the method described in Section 5.3 to allow the contour tracker to work successfully under pan, tilt and zoom conditions. In Section 5.3.8 one of the limitations of the static view contour tracker is in determining the values of the physical constants, such as the mass of the control points and the fall-off factors. These were intended to be manipulated by a human operator in a post-production type workflow environment such as Apple's post production compositing package, Shake[162]. Given the computational time in matching under pan, tilt and zoom conditions with unoptimised code, it was infeasible to manually estimate the correct values of all the parameters. Therefore, it was decided to simplify the tracking contour by removing the Newtonian mechanics model and rather using a non-linear optimiser to minimise the residual error between the contour function and the target match position data points.

This simplified method resembles something of a maximum likelihood estimator, however, there are some difficulties in the modelling of the match probability distribution. The simplified contour fits a least squares cubic spline with a small number of control points, typically 5 to 10, to a cubic spline fit of all the active match points in the sequence. As in the static view contour tracker (Section 5.3.5) frames in the sequence are marked active, starting at the ends and moving towards the centre in time with each iteration. Section 5.4.3 discusses this algorithm fully. An implementation of the contour tracker under pan, tilt and zoom conditions is described in Section 7.5 as well as the results obtained from test sequences.

5.4.1 Contour fit in world space

Under pan, tilt and zoom conditions, the movement of a target can come from two sources: firstly, the target itself can move and secondly, it can have apparent motion due to a change in the camera parameters, for example panning away from the target. If the camera movement is fast, it is possible to create step changes in the target's position in the image between frames.



(a) Image co-ordinates

(b) World co-ordinates

Figure 5.9: *Contour fit in world space of sequence* `tom3`

By working with the target's position in world co-ordinates, however, this effect will disappear as the camera's movement will be removed. The difference between an object's trajectory in image co-ordinates and world co-ordinates can be clearly seen in Figure 5.9.

Generally, it can be assumed that any target moving normally will have a simple trajectory in world co-ordinates. For example, a person walking will tend to walk in a reasonably straight line or a moving vehicle will move in a fixed direction at roughly constant velocity. This adds more weight to the decision to fit the contour in world co-ordinates. Each point on the contour position in world co-ordinates is specified by a 3-vector as $(x, y, z)^{\mathsf{T}}$. The z co-ordinate is not truly the depth of the target, but rather the depth of the image plane relative to the reference frame. Using the camera parameters for each frame it is possible to project the contour position into image co-ordinates from world co-ordinates and vice versa.

5.4.2 Least squares spline fit

As with the static view contour tracker (Section 5.3) one of the goals is to produce a contour that is easily editable by a human operator, therefore, any smooth function that approximates the trajectory must have only a few control points. Two criteria need to be satisfied: the contour needs to pass through a small number of control points yet it needs to pass very close to or through a large number of match positions. By fitting a piecewise cubic spline to all the match

positions and then fitting a least squares cubic spline (i.e. the tracking contour) with a small number of knots to the match position spline, achieves both these criteria. The cubic spline fit to the match positions is fitted using conventional methods as described in Section 5.3.2.1. The tracking contour, which is a reduced knot cubic spline, is then fitted to the first cubic spline using a non-linear minimiser to estimate the least squares position of the contour control points.

Each control point in the contour is parameterised by a 4-vector $\mathbf{c}_n = (x, y, z, t)^\mathsf{T}$ where (x, y, z) are the 3D position of the contour in world co-ordinates and t is the frame number at which the control point is located temporally. All of the control point vectors can be stacked into a single $4n$ parameter vector $\mathbf{p}$ as follows:

$$\mathbf{p} = (x_1, y_1, z_1, t_1, x_2, y_2, z_2, t_2, \ldots, x_n, y_n, z_n, t_n)^\mathsf{T} \tag{5.16}$$

A standard cubic spline is fitted to the match positions in world co-ordinates. It is then possible to sum the residual error between the estimated least squares spline fit and the cubic split fit to the match points. This cost function (Equation 5.17) is used by a non-linear minimiser to estimate the value of $\mathbf{p}$ that produces the least residual error between the least squares cubic spline and the a standard cubic spline fit to the data.

$$\text{cost function} = C(\hat{\mathbf{p}}) = \sum_{u=0}^{1} \left\| \hat{\mathbf{S}}(\hat{\mathbf{p}}, x(u)) - \mathbf{Y}_i(x(u)) \right\|^2 \tag{5.17}$$

$$x(u) = (\max x - \min x) \cdot u + \min x \tag{5.18}$$

where
- u — parameter in the range $[0; 1]$ incremented in steps of $\Delta u = 0.1$ to sum values across the interpolating function
- $x(u)$ — maps parameter u to an x value
- $\hat{\mathbf{S}}$ — estimated least squares cubic spline
- $\hat{\mathbf{p}}$ — parameter vector containing unwrapped co-ordinates of control points of spline $\hat{\mathbf{S}}$
- $\mathbf{Y}_i$ — cubic spline fit to data

If the number of control points in the least squares fit cubic spline equals the number of data points, then the least squares fit cubic spline will be identical to a cubic spline fit of the data.

However, the advantage in using this method is that a cubic spline with only a few knots can be used to smoothly fit to the match position data. The number of control points to use depends on the smoothness of the target's trajectory.

VXL's implementation of Powell's method[3] for the non-linear estimation of $\mathbf{p}$ was used to fit a least squares cubic spline to the match position data used . Powell's method was chosen as the gradient of the cost function does not need to be calculated, as would be in other methods such as the conjugate gradient or simplex methods. The only disadvantage of the method is high computational complexity, however, for the small number of points involved the algorithm only took a few seconds to compute, making optimisation pointless.

5.4.3 Tracking with the contour

Figure 5.10 shows the algorithm for contour tracking under pan, tilt and zoom conditions. All match positions and contour positions are in world co-ordinates and *not* image pixel co-ordinates. The reasons for this are discussed in Section 5.4.1. The contour tracker has several parameters that can be adjusted to suit different sequences: the search area size, outlier rejection thresholds and block update thresholds.

Step 6d in the algorithm in Figure 5.10 uses the method described in Section 5.4.2 to refit the contour to the new set of match positions. This process is subtly different to the fixed view contour tracking, however, it achieves the same objective which is to bring the contour towards the local region of good matches where the target's true position lies. It is, however, still suspectable to becoming trapped in regions of high false match if initialised to lie in that region of the image. However, this is true of the static view contour tracker too. The careful choice of parameters can overcome this type of situation for both methods.

Step 7 is marked as optional, as it was found that only a single iteration was needed to find the tracking contour. In some circumstances, iterating reduced the accuracy of the tracking contour by moving it into regions of false matches – this was due to a reduction in the size of the search window. If the search window is not reduced, then the position of the contour should not move after one iteration, as the best matches for that area have already been found.

The position of the target can either be taken as the best match in each frame or, if the trajectory

[3] See Numerical Recipes in C[115] for a full discussion and VXL documentation[21] for implementation details.

1. User indicates start s and end e positions of the contour in the start frame $t = 0$ and end frame $t = N$ in world co-ordinates
2. Extract reference blocks around points s and e from the start and end frames, warping them into the reference view
3. Initialise the contour as a straight line between s and e with n evenly spaced control points
4. Set of active frames = $\emptyset$
5. Set of inactive frames = all intermediate frames
6. While there are frames in inactive set
   (a) Remove first and last frames from inactive set and add to active set
   (b) Match the respective start and end reference blocks to the frames newly made active and perform outlier rejection
   (c) [*optional*] Update the reference blocks
   (d) Move the contour
      i. Remove all least squares fit data points
      ii. Add match positions for all active frames as data points for the least squares cubic spline fit
      iii. Refit the least squares cubic spline to new data
      iv. Find the new positions for all the control points
7. [*optional*] Iterate from step 4 with a smaller search window until contour remains constant

Figure 5.10: *Algorithm for contour tracking under pan, tilt and zoom conditions*

is smooth enough and the frame rate is high, the contour position. This depends on the sequence and the application. For example, if the tracking contour is used to estimate the position of the target between frames, then the contour position should rather be used.

5.4.4 Reference block updating

Reference block updating for the contour tracker is identical to that described for the block trackers in Section 4.2.7. Instead of a single reference block, there are two – one created from the start position and one created from the end position. Each of two reference blocks is updated independently from frames starting on either side of the sequence and moving towards the middle. Ideally, if the updating has been very accurate, the two reference blocks should be nearly identical at the mid-point of the sequence. Due to drift and mismatches, this will be highly unlikely in real sequences.

The problems associated with reference block updating in the block trackers of Section 4.2.7, such as drift, smudging and blurring, occur in the contour tracker as well. However, the effects are not as noticeable due to the better estimation of the target's position, resulting in less false matches and hence a more accurate block. However, effects due to misregistration and small pixel mismatches can not be removed by the contour only. Section 8.3 discusses the possibility of blending the two reference blocks and using the *a priori* knowledge in both directions to estimate the structure of the reference blocks towards the mid-point of the sequence.

5.4.5 Limitations

The limitations of the contour tracker under pan, tilt and zoom conditions are much the same as those discussed in Section 5.3.8 for the static view camera. This method will only work with offline sequences as past and future frames are needed simultaneously. Again, the target's motion in world co-ordinates must be smooth for the contour tracker to be accurate. As with the static view contour tracker, poor trajectory smoothness can be the result of low frame rates or erratic target movement. Therefore, a fairly high frame rate is required, although, as was demonstrated in the results in Section 7.5 the contour tracker performs well matching the target in low frame rate sequences, however, the contour itself is too smooth to be accurate.

5.5 Applications

As has been mentioned throughout this chapter, the contour tracker was designed with the intention that it be used in a post production workflow environment. Packages such as Apple's video compositor Shake[162] have interfaces where external plug-ins can be written to work within its framework and provide a common user interface. These plug-ins normally allow the user to adjust parameters (for example, the number of control points and the match thresholds) graphically and watch the results calculate on the fly.

The method of working in the post production industry is fairly rigid and any new approach needs to be designed with this in mind. Although, the contour tracker is a prototype, it could eventually be implemented as a Shake plug-in. The ability to hand edit a contour track using the control points is an essential feature making the contour tracker easily usable by a human operator and to provide quick, easy and robust tracking.

By adding the flexibility to work under pan, tilt and zoom conditions, the contour tracker could be used in a far larger range of post production scenarios – such as special effects in sports television. It also has minor applications to the offline processing of video surveillance data, where the paths of people moving through a CCTV area could be monitored and analyzed for patterns.

5.6 Summary

A novel tracking contour was presented in this section and discussed in the context of previous work by Kass[39] for fitting snakes to object boundaries. Methods for contour tracking with a fixed camera view and under pan, tilt and zoom conditions were explored. Algorithms for each of these scenarios were presented in pseudocode.

The merits and limitations of the contour tracker were discussed and several applications in media post production were explored, such as the possibility of creating a plug-in for video post production software tools.

An implementation of a contour tracker and results from real video sequences can be found in Chapter 7. Ideas on how to expand and improve the contour tracker in the future are briefly discussed in Section 8.3.

Chapter 6
Camera Calibration

6.1 Introduction

This chapter discusses the calibration of the Sony EVI-D31 PTZ camera. An overview of camera calibration techniques is presented in Section 6.2 as well as a motivation for the method used. A brief outline of bundle adjustment is given in Section 6.3 and full details can be found in Appendix B.7.

A self-calibration technique based on work by Sinha[28] is described in Section 6.4 which is then implemented (Section 6.6.2). Several test sequences were captured (Section 6.6.1) and results of the calibration are presented in Section 6.6.3.

A model of the camera is developed in Section 6.5 and the integration of this model into the tracking framework is discussed in Section 6.5.3. The quality of the calibration is assessed in Section 6.7 with several illustrative examples shown to illustrate this.

6.2 Overview of Camera Calibration Techniques

This section contains a brief summary of the comprehensive review of camera calibration techniques which can be found in Section 2.4 on page 17. Early work on camera calibration was done by Brown[90] and Tsai[41]. Tsai's formulation of camera parameters and radial distortion are widely used throughout computer vision to calibrate cameras.

Calibration methods can be broken into two categories: calibration using a calibration object (such as a plane with a checkerboard pattern) and auto-calibration or self-calibration which matches good features in a scene. Zhang [42, 99], Heikkila and Silven[100], Sturm[101] and the calibration toolbox by Jean-Yves Bouguet[22] all require a fixed calibration object which is normally a checkerboard pattern on one or more planes. Faugeras *et al*[92] and Maybank[96] propose methods of self-calibration which relate corner features in many views via the absolute conic and the plane at infinity. The calibration object methods are useful if the calibration must

relate distances in the image to fixed distances in the real world, however this could also be achieved by specifying the size of the sensor elements (or pixels) in a self-calibration method.

Willson [102, 103] produced a comprehensive thesis on calibrating a zoom lens, however this requires shining a laser onto the image sensor. Work on calibrating PTZ cameras has been done by Sturm[106, 107], Hartley[30], de Agapito[29] and Sinha and Pollefeys[117]. All of these methods are self-calibration techniques and use corner features found in all of the various camera views. Davis and Chen[116] use an LED in a dark scene to calibrate a camera network with a mixture of PTZ and fixed cameras. Collins[114] uses dense optical flow and image differences to calibrate a network of cameras, some of which can pan, tilt and zoom.

The most enticing method of calibrating a PTZ camera is self-calibration. Sinha and Pollefeys' method[117] was chosen as it uses established self-calibration methods and requires no calibration object. The calibration it produces is globally optimised due to several bundle adjustment[43] steps. This means that the zoom model is consistent over the entire zoom range and a continuous camera model can be created from this. If the camera was calibrated independently at various steps, using Jean-Yves Bouguet's[22] calibration toolbox for instance, there would be no guarantee of a consistent model between calibration points. Another attractive feature is the absence of a calibration object, which is difficult to manage with a PTZ camera. The self-calibration method is automatic and only requires the basic setup of the camera so that an acceptable scene is imaged. The remainder of this chapter discusses this calibration method and its application to the modelling of the Sony EVI-D31 PTZ camera.

6.3 Bundle Adjustment

Bundle adjustment is a technique that has become popular in self-calibration in the last few years. It was developed by Brown[163] to align images for aerial cartography for the U.S. Air Force in the late 1950's. More recently Triggs[43] has provided a review of bundle adjustment methods and their implementation as well as a historical overview of their use. Where bundle adjustment has come into its own, is to globally optimise the parameters of several cameras and points in a structure from motion type problem. Most current implementations of bundle adjustment use some form of non-linear least squares minimisation such as Gauss-Seidel iteration or Levenberg-Marquardt minimisation. For many types of structure from motion problems, profile Cholesky decomposition has been used to improve performance by a factor of two. These

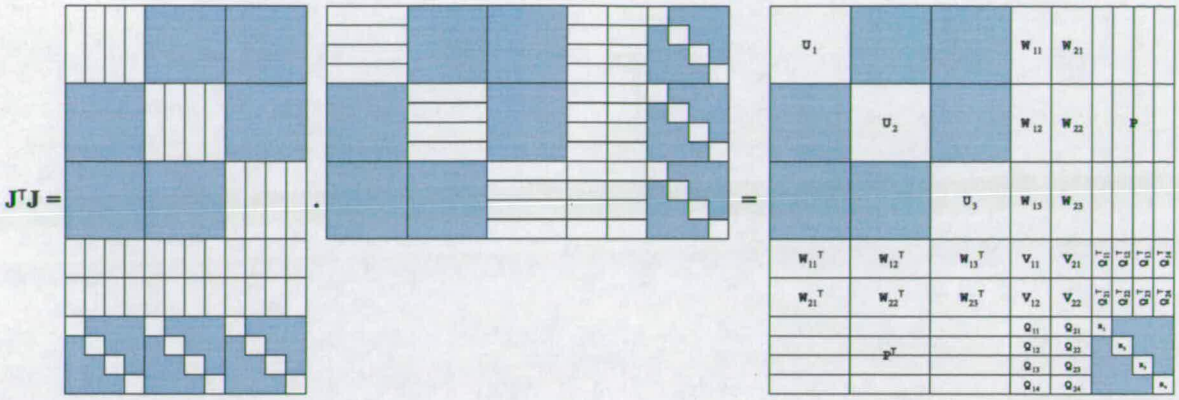


Figure 6.1: *Jacobians and the Hessian matrix for three partitions – a full page enlargement of this diagram can be found in an appendix on page 313*

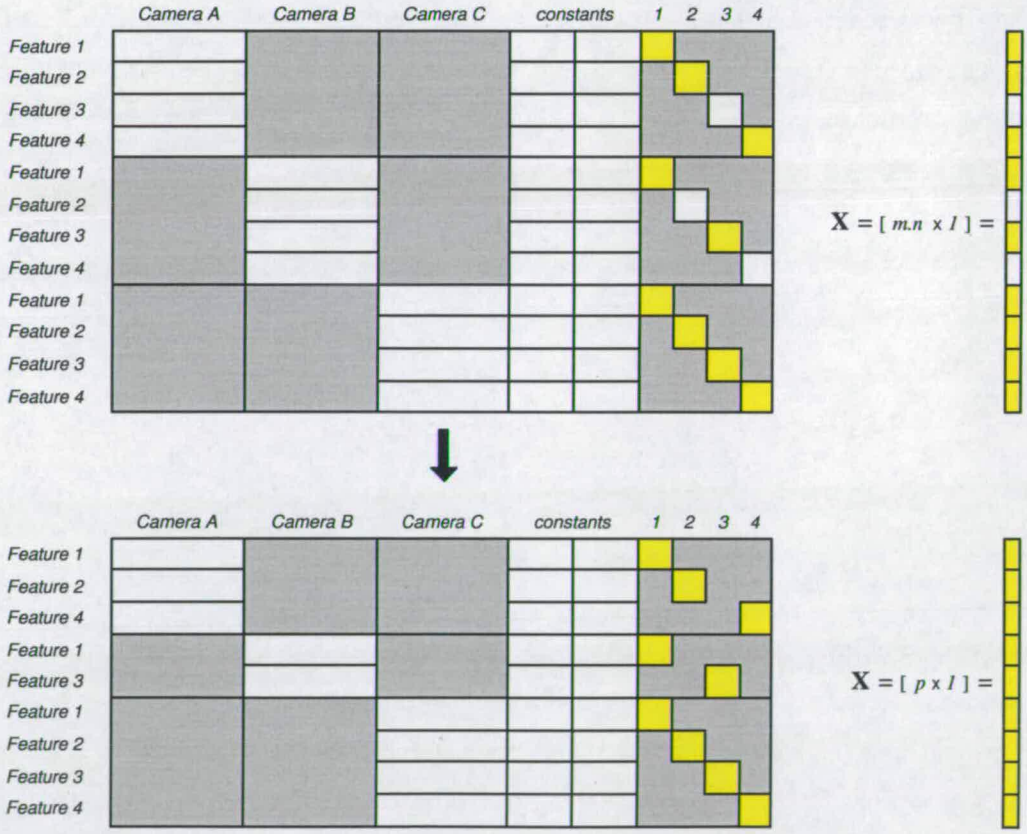


Figure 6.2: *Missing data and Jacobians for Bundle Adjustment – missing data is shown in white squares and yellow squares indicates a data point that exists*

methods and many others are discussed by Triggs[43]. Multiple View Geometry by Hartley and Zisserman[34] is a good text which explains bundle adjustment and its implementation simply and clearly. Pollefeys has a brief treatment of bundle adjustment with examples in a tutorial[25] given at various conferences. Faugeras[31] also gives a brief description of bundle adjustment.

Bundle adjustment is extremely useful in self-calibration problems as it allows the optimisation of many sets of camera parameters using points that are visible in small subsets of camera views. The result is a solution that is globally valid. The iterative non-linear algorithms converge on a solution within 20 iterations and sparse matrix implementations mean memory usage for several thousand points is very low. Bundle adjustment also allows missing data, such as features that are not visible in all views or parameters that only apply to a subset of the cameras. This flexibility makes bundle adjustment an extremely powerful tool. The details of bundle adjustment and its implementation are included in Appendix B.7, however a brief summary is given below.

Bundle adjustment estimates the value of a parameter vector $\mathbf{P}$ such that the error ϵ between an estimated measurement vector $\hat{\mathbf{X}} = \mathbf{f}(\mathbf{P})$ and the true measurement vector $\mathbf{X}$ is minimised. Levenberg-Marquardt minimisation is a method that can be used to find the value of $\mathbf{P}$ which minimises the error $\epsilon = \mathbf{X} - \mathbf{f}(\mathbf{P})$ by iteratively adding an increment vector $\mathbf{\Delta}$ to $\mathbf{P}$ until a solution is found. The increment vector $\mathbf{\Delta}$ is found by evaluating the Normal Equations:

$$(\mathbf{J}^\mathsf{T}\mathbf{J} + \lambda\mathtt{I})\mathbf{\Delta} = -\mathbf{J}^\mathsf{T}\epsilon \tag{6.1}$$

where
- $\mathbf{J}$ Jacobian matrix (see Section B.6.1)
- ϵ Residual error vector $\epsilon = \mathbf{X} - \mathbf{f}(\mathbf{P})$
- λ Step control size
- $\mathbf{\Delta}$ Parameter vector increment

The step size λ allows the method to alternate between Gradient Descent (fast, unstable convergence) and Newton's Method (slow, stable convergence). If the increment $\mathbf{\Delta}$ increases the total residual error $\sum_i \epsilon_i$, then the step size is made larger, as the parameter vector is not in the correct search area of the solution space. However, if the total residual error decreases, then the solution is nearby and the step size is decreased.

The Hessian matrix (Section B.6.2) is approximated by $(\mathbf{J}^\mathsf{T}\mathbf{J} + \lambda\mathtt{I})$ on the left-hand side of the Normal Equations. In order to solve for $\mathbf{\Delta}$, the Hessian needs to be inverted. The Hessian

matrix is fairly sparse, where, for example, a 2000 element sparse matrix can have 200 non-zero elements[25] for most self-calibration and structure from motion problems. This makes it feasible to invert the Hessian quickly using matrix factorisation. The structure of the Hessian (Figure 6.1), also allows it to be divided into block matrices, which can be evaluated independently by QR-decomposition or profile Cholesky factorisation. By partitioning the parameter vector correctly, this structure can be introduced into the Hessian. Bundle adjustments can also accommodate missing data, for example some features may only be visible from a few views. The missing data modifies the structure of the Jacobians and Hessian matrices as shown in Figure 6.2. Two and three partition bundle adjustments are discussed further in Appendix B.7.

6.4 Self-calibration

The following sections describe the implementation of a method calibrating a PTZ camera from a scene without a calibration object described by Sinha[28]. Sinha's work draws on earlier work by de Aggipito[29, 112, 113], Hartley[30, 34] and Hayman[7] on calibrating zoom cameras.

6.4.1 Implementation Overview

The camera parameters at the minimum zoom factor are determined by the following broad steps:

1. **Image grid** – capture images in a grid by panning and tilting the camera in fixed steps.

2. **Pairwise homographies** – a homography is found between each horizontal and vertical pair of images by finding corners in each image, matching them by block matching and using RANSAC to get a first estimate of the homographies. A maximum likelihood estimator is then used to find accurate vertical and horizontal pairwise homographies using only the inlier matched corners.

3. **Chain homographies** – a reference image is chosen and a homography between the reference image and every other image in the grid is found by multiplying together the pairwise homographies on the shortest path between them. There is a systematic error build-up in this process.

4. **Global feature list** – a global feature list can be created by using the pairwise homographies to estimate the position of corner points in every other image in the grid.

5. **Bundle Adjustment I** – the chain homographies are then refined by a bundle adjustment and the radial distortion parameters are estimated. The bundle adjustment also estimates the maximum likelihood 2D position of the global features in the reference image coordinates.

6. **Bundle Adjustment II** – the intrinsic parameters and rotation matrices for each image are estimated as well as the radial distortion parameters. The global feature positions are now estimated as 3D points in world space.

7. **Full calibration at minimum zoom** – the intrinsic parameters $\mathbf{K}$, the rotation matrices $\mathbf{R_1} \ldots \mathbf{R_n}$ for each view and the radial distortion parameters are known at the minimum zoom level. This provides a full calibration at the minimum zoom level.

The zoom calibration process is very similar and requires the initial estimates of the intrinsic parameters, rotation matrices and radial distortion at the minimum zoom:

1. **Zoom sequence** – a fixed direction is picked and the camera is zoomed in progressively from the minimum zoom. Sinha and Pollefeys argue that it isn't necessary to capture a zoom sequence for every pan and tilt position.

2. **Minimum zoom parameters** – the intrinsic parameters, rotation matrices, radial distortion parameters and global features from the above calibration are used to initialise the method.

3. **Pairwise Bundle Adjustment** – a bundle adjustment is done between parameters at successive zooms using parameters estimated in the previous step.

4. **Final Bundle Adjustment** – a final bundle adjustment is done over the whole range of zoom steps.

5. **Full calibration at all zoom levels** – now the intrinsic parameters, rotation matrices and radial distortion parameters are known for all zoom levels. The camera is fully calibrated.

Each of these steps is discussed in detail in the following sections.

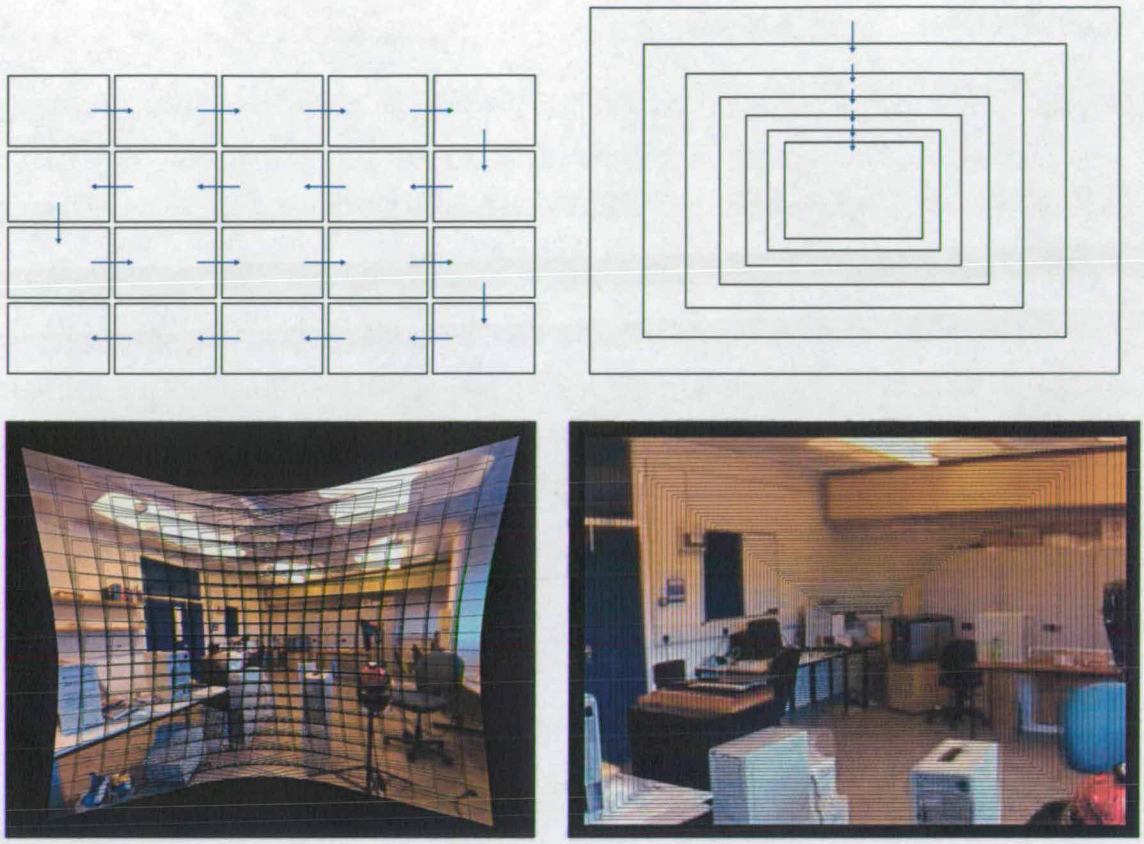


Figure 6.3: (left) *Grid for pan and tilt calibration and* (right) *zooming in one direction*

6.4.2 Image Grid

The images used in the calibration are captured in a grid by capturing images at various pan and tilt positions. The Sony EVI-D31 has a pan range of $\pm 110°$ and a tilt range of $\pm 21°$. Images were capturing at equal increments over a reduced pan/tilt range (approximately $\pm 50°$ pan angle) as shown in Figure 6.3. The reference image was chosen as the centre of the camera's pan and tilt range ($0°$ pan and $0°$ tilt).

The zoom sequences were created by panning and tilting the camera in a fixed direction (the reference direction is convenient) and incrementally zooming the camera over the full range of zoom (as shown in Figure 6.3). When frames were captured for either the pan/tilt or zoom sequences, the camera parameters for each frame were recorded. The pan, tilt and zoom index can be queried from the camera using the VISCA protocol discussed in Appendix D.2.1. These will be used later in determining the camera model (Section 6.6.3).

6.4.3 Points Correspondences

A Harris corner detector[164] was used to find corners in a pair of images. A one-to-one mapping was then found between each corner in image 1 and a corner in image 2. Each corner was mapped by taking a region of the image around the point in image 1 and searching for it in image 2 using Normalised Cross-correlation (NCC).

Some of the matches will be correct and others will be incorrect. In the next step of finding pairwise homographies, the matches will be divided into inlier pairs and outlier pairs that are consistent with a homography. As the pairwise homography method is robust, a rough match can be performed as the algorithm can deal with large amounts of outliers (50% outliers is usual).

SIFT[165, 166] features could be used instead of a Harris corner detector as they are more robust to scaling and rotation, which is common in this problem.

6.4.4 Pairwise Homographies

A method to find the homography between a pair of images is spread throughout several chapters of Hartley and Zisserman's *Multiple View Geometry in Computer Vision*[34], therefore the core concepts have been collected and summarised here in the context of camera calibration. In many cases, Hartley and Zisserman also omit the exact dimensions and structure of the stacked matrices for use with direct linear methods or leave portions of the algorithms as exercises for the reader. These omissions have been spelled out so that anyone wishing to implement this method does not have to suffer from the uncertainty of whether errors in the method arise from coding errors or mathematical errors. The complete method for finding pairwise homographies is outline below:

1. **Points of interest** – detect the corners in each image using a Harris[164] corner detector. Other methods could be used such as finding SIFT[165, 166] parameters which would be more suited to this type of problem.

2. **Point correspondences** – match the corners from the first image to corners in the second image using block matching (normalised cross-correlation).

3. **RANSAC homography estimation** – picking 4 points at random, a homography is calculated and the reprojection error is calculated. The number of inliers consistent with the

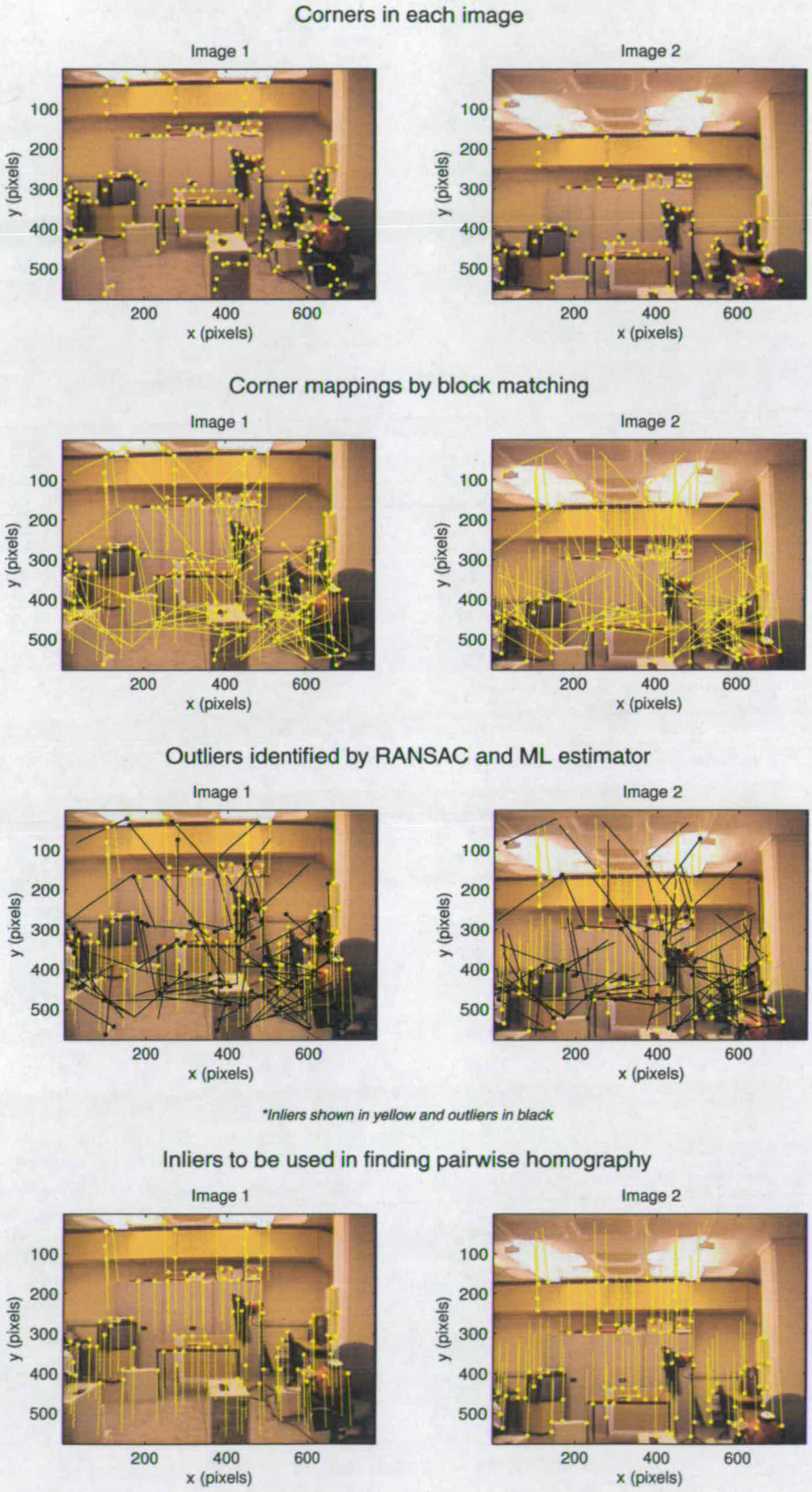


Figure 6.4: *Finding the pairwise homographies for pan and tilt*

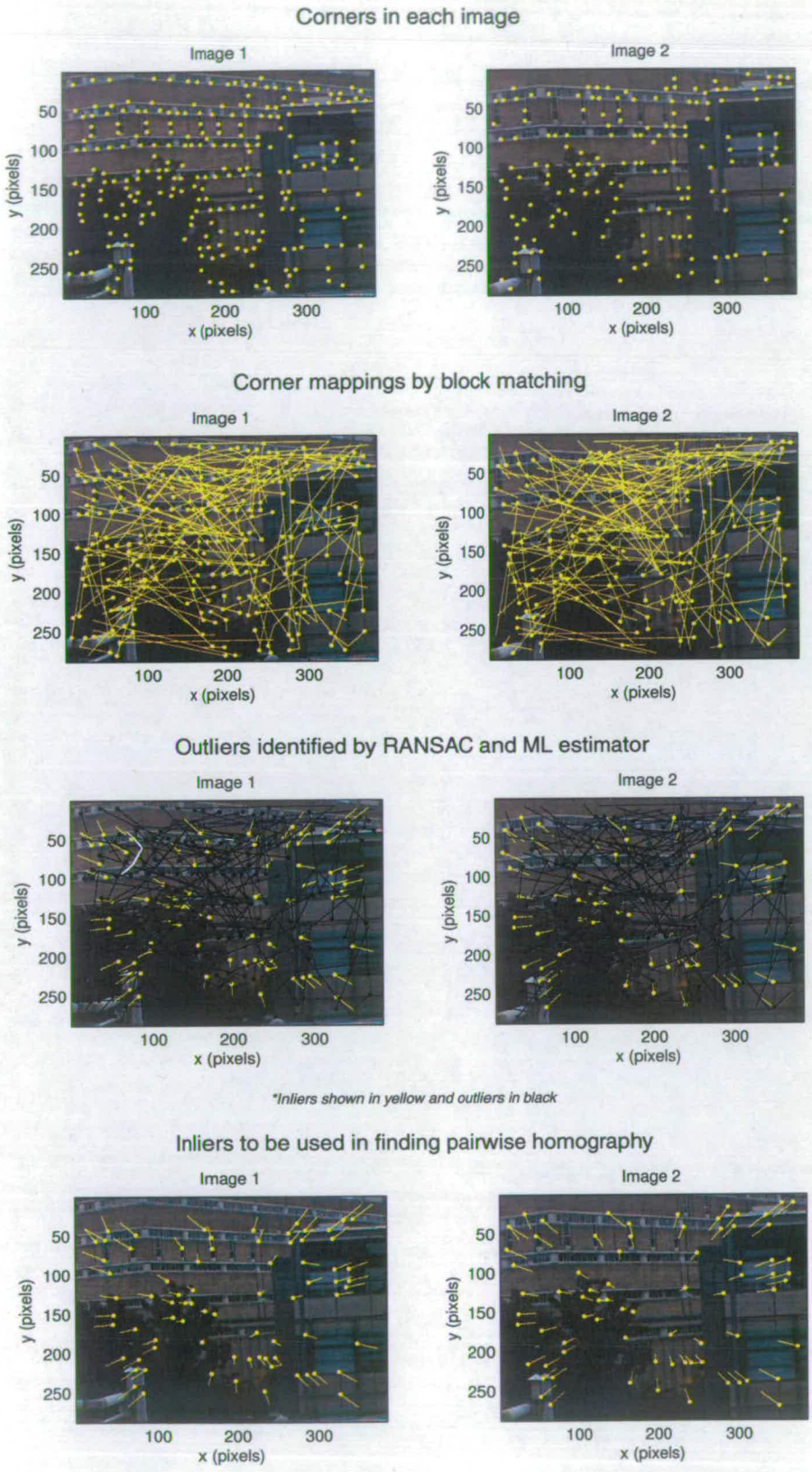


Figure 6.5: *Finding the pairwise homographies for zoom*

homography is calculated. This process is repeated and the homography with the largest number of inliers is chosen.

4. **Homography refinement** – the homography is then re-estimated with all the inliers using a maximum likelihood estimator (Levenberg-Marquardt algorithm).

5. **Guided matching** – the refined homography is then used to add more point correspondences and remove more outliers.

6. **Iterate** – the last two steps are iterated until the number of point correspondences becomes stable.

See Figure 6.4 and Figure 6.5 for a graphical representation of this process for pan/tilt and zoom sequences respectively.

6.4.4.1 Find a homography from 4 point correspondences

The Direct Linear Transform (DLT) algorithm needs a minimum of 4 point correspondences to determine a homography, so given $n \geq 4$ 2D to 2D point correspondences $\mathbf{x}_i \leftrightarrow \mathbf{x}'_i$ a homography $\mathtt{H}$ can be found such that $\mathbf{x}'_i = \mathtt{H}\mathbf{x}_i$.

The elements of $\mathtt{H}$ are arrange into a vector $\mathbf{h} = \begin{pmatrix} h_1 & h_2 & h_3 & h_4 & h_5 & h_6 & h_7 & h_8 & h_9 \end{pmatrix}^\mathsf{T}$. Arrange the point correspondences into a 2×9 matrix $\mathbf{A}_i$ such that $\mathbf{A}_i\mathbf{h} = \mathbf{0}$:

$$\mathbf{A}_i = \begin{bmatrix} 0 & 0 & 0 & -w'_i x_i & -w'_i y_i & -w'_i w_i & y'_i x_i & y'_i y_i & y'_i w_i \\ w'_i x_i & w'_i y_i & w'_i w_i & 0 & 0 & 0 & -x'_i x_i & -x'_i y_i & -x'_i w_i \end{bmatrix} \tag{6.2}$$

Create the $2n \times 9$ matrix $\mathbf{A}$ from the n point correspondences:

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 & -w'_1 x_1 & -w'_1 y_1 & -w'_1 w_1 & y'_1 x_1 & y'_1 y_1 & y'_1 w_1 \\ w'_1 x_1 & w'_1 y_1 & w'_1 w_1 & 0 & 0 & 0 & -x'_1 x_1 & -x'_1 y_1 & -x'_1 w_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & -w'_n x_n & -w'_n y_n & -w'_n w_n & y'_n x_n & y'_n y_n & y'_n w_n \\ w'_n x_n & w'_n y_n & w'_n w_n & 0 & 0 & 0 & -x'_n x_n & -x'_n y_n & -x'_n w_n \end{bmatrix} \tag{6.3}$$

Calculate the Singular Value Decomposition (SVD)[1] of $\mathbf{A}$ i.e. $\mathbf{A} = \mathbf{UDV}^{\mathsf{T}}$. The solution for $\mathbf{h}$ is the column of $\mathbf{V}$ corresponding to the smallest eigenvalue (diagonal entries of $\mathbf{D}$). The vector $\mathbf{h}$ can then be reshaped into the homography matrix $\mathtt{H}$.

6.4.4.2 RANSAC

The RANSAC algorithm was developed by Fischler and Bolles[109] to robustly estimate functions from data with many outliers. RANSAC estimates many functions from minimal subsets of the data and measures how much of the data agrees with each estimate. The estimated function with the largest number of inliers is chosen as the solution.

To estimate the homography with RANSAC, four points are selected at random and a homography is calculated using the DLT algorithm described in Section 6.4.4.1. The reprojection error for each point is calculated and it is marked as an inlier or an outlier as described below:

$$\begin{cases} \text{inlier} & d(\mathbf{x}_i, \mathtt{H}^{-1}\mathbf{x}'_i)^2 + d(\mathbf{x}'_i, \mathtt{H}\mathbf{x}_i)^2 < t \text{ pixels} \\ \text{outlier} & d(\mathbf{x}_i, \mathtt{H}^{-1}\mathbf{x}'_i)^2 + d(\mathbf{x}'_i, \mathtt{H}\mathbf{x}_i)^2 \geq t \text{ pixels} \end{cases}$$

where	$d(\cdot)$	Euclidean distance between two vectors
	t	Outlier threshold ($t = \sqrt{5.99}\sigma$ for 95% confidence interval)
	$d(\mathbf{x}_i, \mathtt{H}^{-1}\mathbf{x}'_i)^2 + d(\mathbf{x}'_i, \mathtt{H}\mathbf{x}_i)^2$	systematic transfer error

The set of inliers is known as the *consensus set*. An estimate of the size of an acceptable consensus set is determined by:

$$T = (1 - \epsilon)n \tag{6.4}$$

where	n	total number of points
	ϵ	estimated percentage of outliers

It is possible that the points chosen will produce a degenerate homography, so it is important that the 4 points chosen are not collinear and are spread fairly well (Hartley and Zisserman[34] suggest several ways of doing this). The number of inliers is counted for each random selection of four points and if the consensus set is big enough (i.e. the number of inliers is greater

[1] More can be found on Singular Value Decompositions in Golub's *Matrix Computations*[167] or in Appendix A 4.4 of *Multiple View Geometry in Computer Vision 2nd Edition*[34]. The SVD implementation in *Numerical Recipes in C*[115] contains many errors and doesn't produce reliable results – this implementation is **not** recommended.

than T) then the homography is kept aside. After sampling many times, the homography with the most inliers is taken as the solution. Every combination of 4 point samples can't feasibly be calculated, so an estimate is made as to the number of samples needed to be taken to produce the correct homography. The number of samples needed to find a solution is described in [34] (Algorithm 4.5 page 121). The resulting homography is produced from the minimal point set needed to calculate it so it is unaffected by the problems of over-fitting to data with many outliers, which would be the case as minimal effort is put into determining the point correspondences.

6.4.4.3 Maximum Likelihood Estimator - Levenberg-Marquardt

A homography can be estimated using all the inlier points from the RANSAC step described above as an input to a maximum likelihood estimator. The maximum likelihood estimator minimises a cost function with Levenberg-Marquardt minimisation, as described in Section B.7.3.

The entries of the homography matrix H are reshaped into the vector:

$$\mathbf{h} = \begin{pmatrix} h_1 & h_2 & h_3 & h_4 & h_5 & h_6 & h_7 & h_8 & h_9 \end{pmatrix}^\mathsf{T}$$

Each of the n point correspondences is then placed one after the other in the measurement vector $\mathbf{X} = (\mathbf{x}_1^\mathsf{T}, \mathbf{x'}_1^\mathsf{T}, \mathbf{x}_2^\mathsf{T}, \mathbf{x'}_2^\mathsf{T}, \ldots, \mathbf{x}_n^\mathsf{T}, \mathbf{x'}_n^\mathsf{T})^\mathsf{T}$ where $\mathbf{x}_i$ is the inhomogeneous 2D point $(x_i, y_i)^\mathsf{T}$. The measurement vector has a covariance matrix $\mathbf{\Sigma_X}$ which is assumed to be the block identity matrix, i.e. each 2D point is independent and the reprojection error variance is one ($\mathbf{\Sigma_{X}}_i = \mathtt{I}_{2\times 2}$). The parameter vector $\mathbf{P}$ is divided into two partitions $\mathbf{a}$ and $\mathbf{b}$ so that $\mathbf{P} = (\mathbf{a}^\mathsf{T}, \mathbf{b}^\mathsf{T})^\mathsf{T} = (\mathbf{h}^\mathsf{T}, \mathbf{X}^\mathsf{T})^\mathsf{T}$.

A function $\mathbf{f}$ is defined which takes the parameter vector to an estimate of the measurement vector $\hat{\mathbf{X}}$:

$$\mathbf{f} : \mathbf{P} \mapsto \hat{\mathbf{X}} \tag{6.5}$$

The error ϵ can be defined as the difference between the measurement vector and the estimated measurement vector:

$$\epsilon = \mathbf{X} - \hat{\mathbf{X}} \tag{6.6}$$

The cost function we wish to minimise is given by $\epsilon^\mathsf{T}\mathbf{\Sigma_X}\epsilon$ which is equivalent to the square

of the reprojection error. Since it is assumed that the 2D points are independent and have a variance of one the cost function becomes:

$$\sum_{i=1}^{n} \epsilon_i^{\mathsf{T}} \epsilon_i = (\mathbf{X}_i - \hat{\mathbf{X}}_i)^{\mathsf{T}} (\mathbf{X}_i - \hat{\mathbf{X}}_i) \tag{6.7}$$

The Jacobian matrices have a special form because only the point in the second image $\mathbf{x}_i'$ is affected by $\hat{\mathbf{H}}$. This gives:

$$\mathtt{A}_i = \frac{\partial \hat{\mathbf{X}}_i}{\partial \mathbf{h}} = \begin{bmatrix} \mathtt{0}_{2\times 9} \\ \frac{\partial \hat{\mathbf{x}}_i'}{\partial \mathbf{h}} \end{bmatrix} \tag{6.8}$$

$$\mathtt{B}_i = \frac{\partial \hat{\mathbf{X}}_i}{\partial \hat{\mathbf{x}}_i} = \begin{bmatrix} \mathtt{I}_{2\times 3} \\ \frac{\partial \hat{\mathbf{x}}_i'}{\partial \hat{\mathbf{x}}_i} \end{bmatrix} \tag{6.9}$$

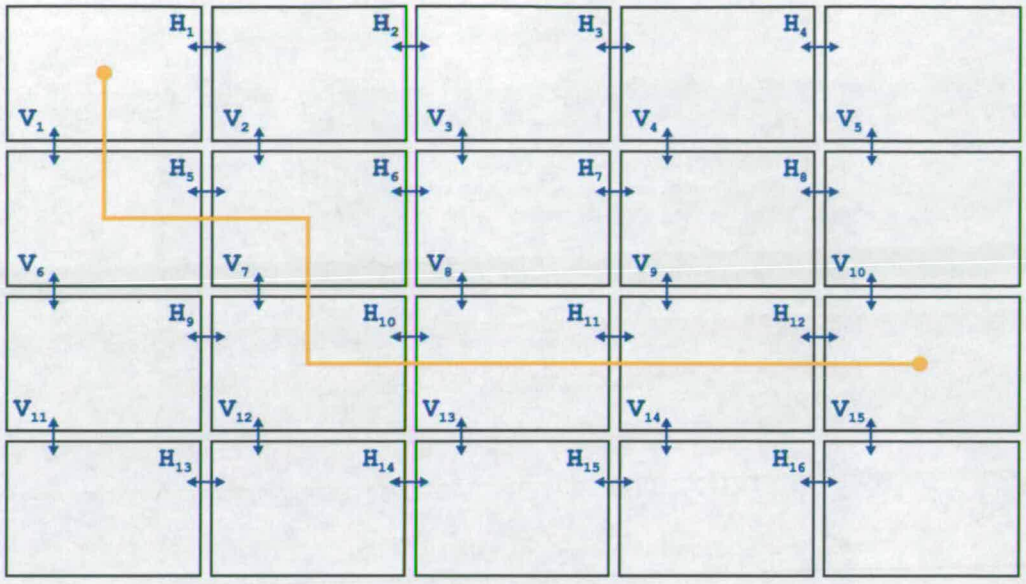
The Jacobians can be calculated by finite difference approximations in each of the dimensions of $\mathbf{h}$ avoiding the need to find an analytic solution, however analytic Jacobians are presented in Appendix C.2. Sometimes, it is necessary to use analytic Jacobians rather than finite difference approximations due to numerical instability caused by limited precision in floating point numbers.

6.4.4.4 Guided matching

Once a better estimation of the pairwise homography has been obtained from the maximum likelihood estimator it is possible to take each outlier point and re-search for the true match. This is done by projecting the point into the other view and searching the list of corners for another un-paired point that is nearby (within a few pixels – normally two). If the point cannot be found in the list of corners, block matching can be used to search for a feature that is similar around the reprojection point. A small search region should be used and the match value should be high to avoid introducing more outlier points. If a good match is found, a point match pair should be created and added to the inlier list.

6.4.5 Chain Homographies

At this point there are accurate homographies between each pair of images in the grid. However, homographies from the reference image mapping points to each image in the grid are necessary



$$H_{chain} = V_1\ H_5\ V_7\ H_{10}\ H_{11}\ H_{12}$$

Figure 6.6: *Chain homographies between views*

to calculate the camera intrinsic parameters and rotation matrices.

The 'chain homography' between the reference image and any image in the grid is found by multiplying the vertical and horizontal homographies along the shortest path between them, as shown in Figure 6.6. This process builds up a systematic error the further away from the reference image the destination image is, hence the need for bundle adjustment and global registration.

6.4.6 Global Feature List

The *global feature list* is used to track the transfer of a feature between camera views. Each point in every view is attached to another point in another view from the pairwise homography process described before. Therefore, it is possible to follow the chains formed by projecting one point from one view to another through multiple views. Figure 6.7 shows the position of global features for a grid of images. The more red the point is, the more views the global feature appears in. Good global features exist in many views and improve the accuracy of the calibration.

The global feature list is created by recursively following the links between pairs of points from each view. Once this is done, many duplicates will exist and these can be easily removed to

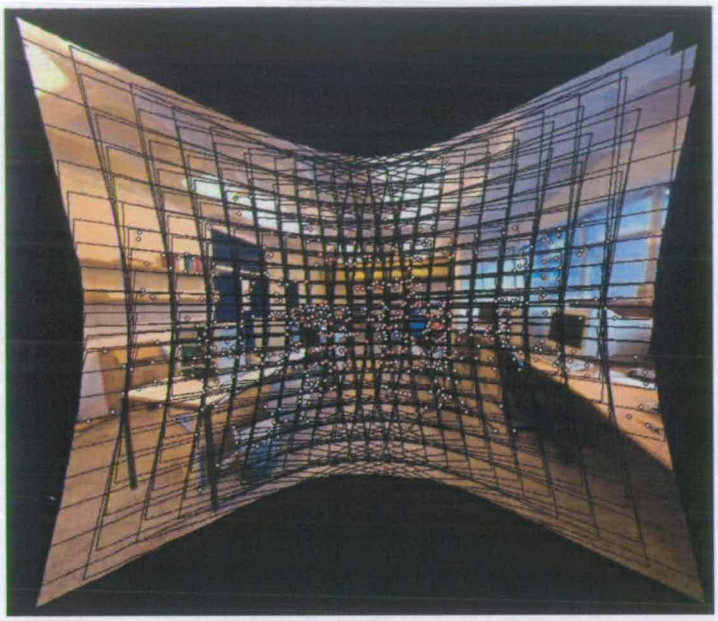


Figure 6.7: *Global features for* `lab1` *– each dot shows a global feature and the more red the dot is, the more images that feature has been matched in*

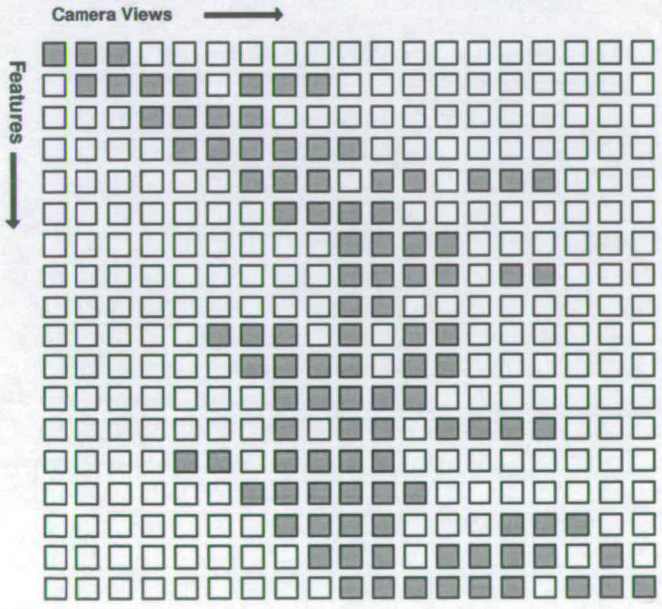


Figure 6.8: *An example global feature list with grey squares indicating the presence of a feature in a camera view*

produce a unique list. A feature must appear in at least three views to be included in the list. This ensures that the results of the bundle adjustment will be globally optimal, as the points contributing to the cost function of the minimiser exist in many views.

The global feature list can be viewed as an array (as shown in Figure 6.8), where each row is a feature and each column is a camera view. Where a feature is visible in a camera view, it will have an entry in the row specifying the co-ordinates of the feature as a 2D point. This results in the banded structure evident in the real global features lists shown in Figure 6.9 and Figure 6.10, as features are visible in certain views but not others. Each block in Figure 6.9 is a 32×32 pixel region around the feature point in each camera view. The grey squares show that the feature was not identified in that camera view, either because it is not visible or it was not correctly matched. This creates gaps in the 'feature chain'. It would be possible to fill these gaps in by rematching features, however, the added computational time of doing this does not increase the accuracy of the calibration and could introduce unwanted outliers due to mismatching. The absence of features in some images where they should exist is not critical as the bundle adjustment handles missing data easily. A good global optimisation will result from good underlying data, therefore, it is better to include only the high confidence matches rather than attempting to find the feature in every possible view and therefore risking adding outliers to the global feature list.

'Columns' of features appear to exist in Figure 6.9 – this is as a result of the grid arrangement of the frames where frames were captured in a 13×13 image grid, hence the repetitive nature every 13 rows. The features from the zoom sequence (shown in Figure 6.10) have no 'column' type pattern and contain gaps (or missing features) as when a feature is missed in the 'feature chain' a new chain will be created.

The banded structure of the global feature list has advantages as it can be represented by a sparse matrix. This sparseness allows fast computation as portions of the global feature list do not contribute anything to the optimisation and can be ignored. In a modest sized problem with fair amounts of overlap between camera views, this sparseness requires that only 3-4% of the entries of the total matrix will be used during the optimisation.

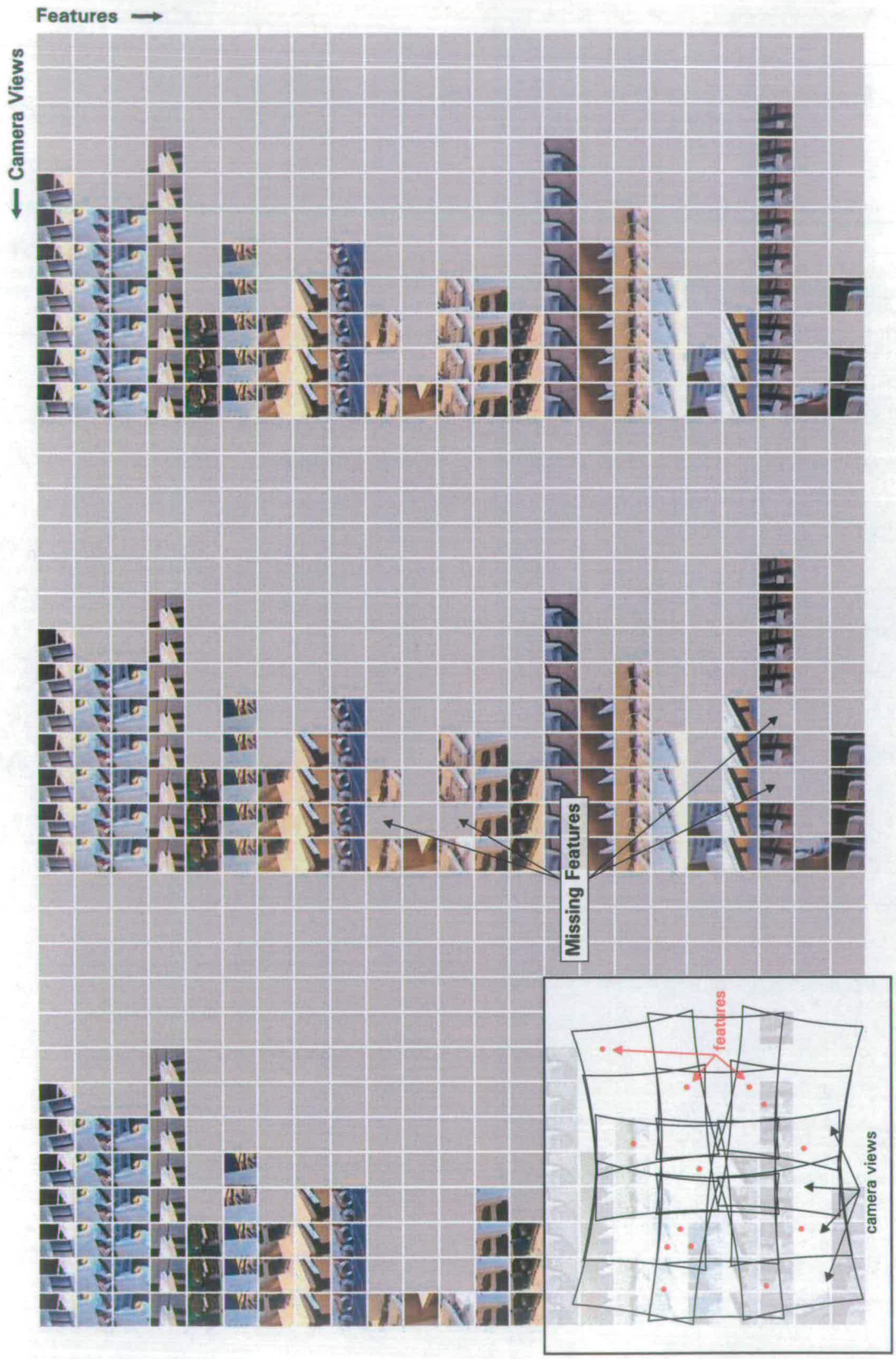


Figure 6.9: *A subset of the global features for* `lab1` *showing when each feature appears in each image (a grey box indicates the feature was not matched in that image)*

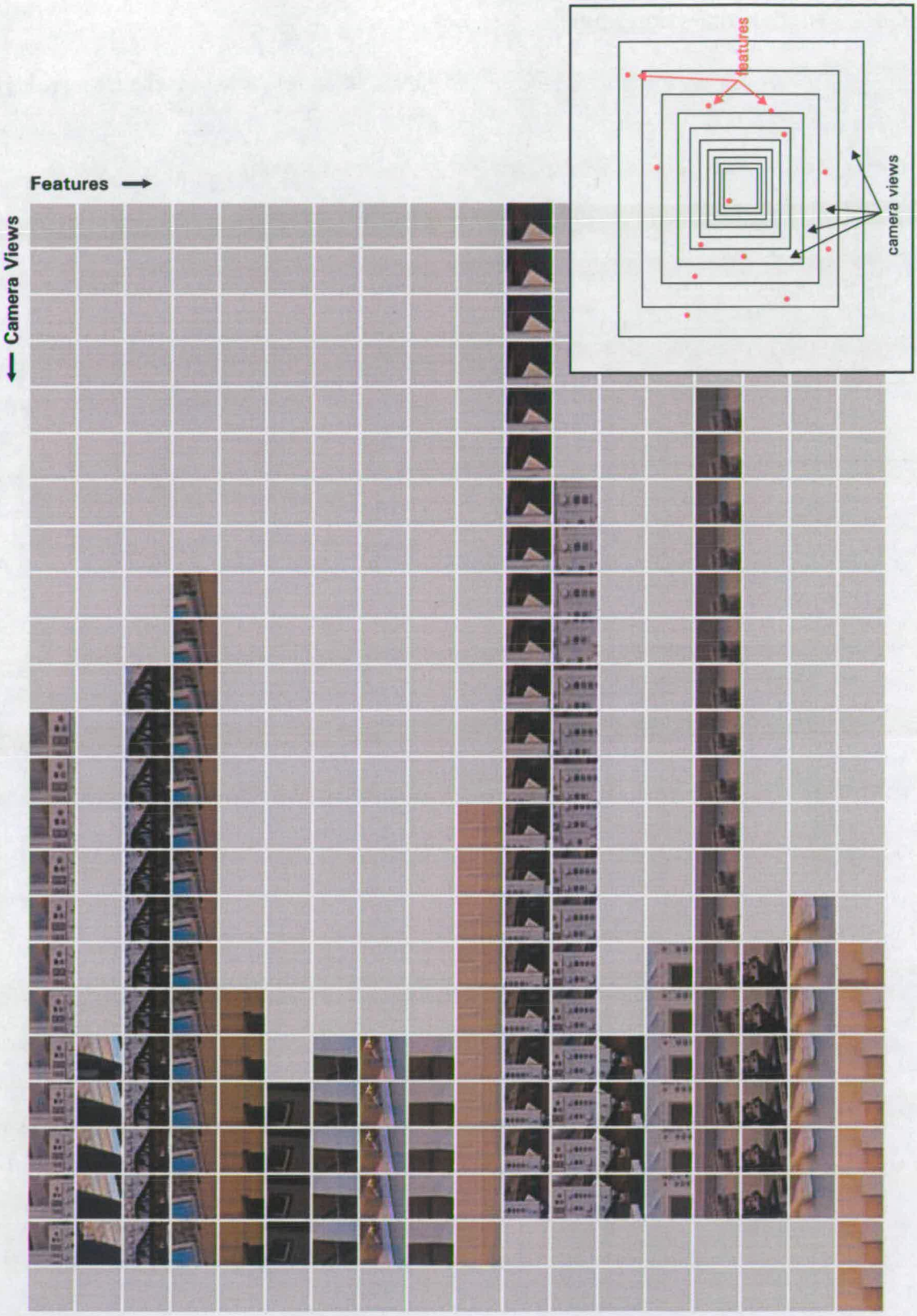


Figure 6.10: *A subset of the global features for* `zoom1` *showing when each feature appears in each image (a grey box indicates the feature was not matched in that image)*

6.4.7 Radial Distortion Model

The camera has varying intrinsic parameters and radial distortion at each zoom level. Tsai's[41] radial distortion model was used and it is outlined below. Take any undistorted point in the image $\mathbf{x} = (x, y)$ and find its distance from the centre of distortion (x_c, y_c):

$$r = \sqrt{(x - x_c)^2 - (y - y_c)^2} \tag{6.10}$$

The distortion function is given by a Taylor series in terms of the distance from the centre of distortion:

$$L(r) = 1 + \kappa_1 r^2 + \kappa_2 r^4 + \kappa_3 r^6 + \dots \tag{6.11}$$

The distorted point is given by $\mathbf{x} \times L(r)$:

$$x_d = (x - x_c) \times L(r) + x_c \tag{6.12}$$

$$y_d = (y - y_c) \times L(r) + y_c \tag{6.13}$$

Given a distorted point, removing the distortion is given by:

$$x = (x_d - x_c)/L(r) + x_c \tag{6.14}$$

$$y = (y_d - y_c)/L(r) + y_c \tag{6.15}$$

The tangential distortion is difficult to calculate due to uncertainty in the centre of distortion when distortion is low and is ignored[105]. The camera principal point and the centre of distortion are assumed to be the same point.

Throughout this chapter, three radial distortion coefficients are always used. The number of terms to use from the Taylor series is dependent on the implementation. All of the Jacobians and other expressions which are affected by the radial distortion can be easily modified for any number of radial distortion coefficients. This is done by removing the rows from affected matrices which correspond to the removed radial distortion parameter.

6.4.8 Bundle I – Radial distortion and homographies

The three partition bundle adjustment method described in Appendix B.7.4 is used to globally optimise the homographies and register all the images. It is then possible to use Hartley's linear

method to find the intrinsic parameters at the lowest zoom level as described in Section 6.4.9. A second bundle adjustment is performed to globally optimise the intrinsic parameters, rotation matrices and radial distortion parameters over all images in the grid.

Bundle I only estimates the homographies and radial distortions and globally optimises them using m camera views and the n global features from Section 6.4.6 with p observations. The following parameters need to be estimated:

Variable	*Size*	*Parameter*	*Initial Value*
Partition a	$\mathbf{a} = (h_{11}, h_{21}, h_{31}, \ldots, h_{1m}, h_{2m}, \ldots, h_{9m})^\mathsf{T}$		
	$9m$ element vector	Homographies for each camera view	Chain homographies from pairwise homographies
Partition b	$\mathbf{b} = (\kappa_1, \kappa_2, \kappa_3)^\mathsf{T}$		
	3-vector	Radial distortion parameters	$\kappa_1 = \kappa_2 = \kappa_3 = 0$ and (x_c, y_c) fixed to $(0, 0)$ and not optimised
Partition c	$\mathbf{c} = (x_1, y_1, w_1, x_2, y_2, w_2, \ldots, x_n, y_n, w_n)^\mathsf{T}$		
	$3n$ element vector	Global features in reference view	From the global feature list and initialised as points on the front cube face i.e. $(x_i, y_i, 1)$
Measurement vector $\mathbf{X}$	$2p$ element vector	Global feature positions in each view	Initialised from the global feature list as 2D points $(\tilde{x_i}, \tilde{y_i})$

The analytic solutions for the Jacobians are described in Appendix C.2. Nearly all texts omit the steps in finding the Jacobians which unfortunately adds to the mystery surrounding bundle adjustment as discussed by Triggs[43]. Although finite difference approximations can be used, the analytical Jacobians provided very good numerical stability and are more aesthetically pleasing. Figure 6.1 shows the Hessian matrix created by transpose multiplying the Jacobians: $\mathbf{H} = \mathbf{J}^\mathsf{T}\mathbf{J}$ (as in Equation B.7).

When creating the global feature list, not all the features are visible in all views. This means that portions of the measurement vector are missing and no longer form a simple $n \times m$ grid. Figure 6.2 shows the effect of missing data on the Jacobians and the measurement vector.

6.4.9 Hartley's Linear Method for finding Intrinsic Parameters

Hartley's linear method can be used to find the intrinsic parameters given n homographies $\mathtt{H}_i$ for several views where the intrinsic parameters remain constant. All homographies must be

normalised so that their determinant is equal to one. Appendix C.5 explains how this is done for any square positive definite matrix.

First, Equation 4 in Section 6 from Hartley[30] needs to be expressed in terms of the entries of $\mathtt{H}_i$ and $\mathbf{C}$ as follows:

$$\mathbf{C}\mathtt{H}_i^{-\mathsf{T}} = \mathtt{H}_i\mathbf{C} \tag{6.16}$$

$$\mathbf{C} - \mathtt{H}_i\mathbf{C}\mathtt{H}_i^{\mathsf{T}} = 0 \tag{6.17}$$

where:

$$\mathbf{C} = \begin{bmatrix} a & b & c \\ b & d & e \\ c & e & f \end{bmatrix} \quad \text{and} \quad \mathtt{H}_i = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix}$$

Expanding[2] the left-hand side of Equation 6.17 results in a vector in terms of the elements of $\mathtt{H}$ and $\mathbf{C}$:

$$\begin{bmatrix} -2h_3eh_2 - h_2^2d - 2h_3ch_1 - 2h_2h_1b + (-h_1^2+1)a - h_3^2f \\ (-h_3h_5 - h_2h_6)e - h_2h_5d + (-h_3h_4 - h_1h_6)c + (1 - h_2h_4 - h_1h_5)b - h_1h_4a - h_3fh_6 \\ (-h_9h_2 - h_8h_3)e - h_8h_2d + (1 - h_9h_1 - h_7h_3)c + (-h_8h_1 - h_7h_2)b - h_7h_1a - h_9fh_3 \\ (-h_3h_5 - h_2h_6)e - h_2h_5d + (-h_3h_4 - h_1h_6)c + (1 - h_2h_4 - h_1h_5)b - h_1h_4a - h_3fh_6 \\ -2h_6eh_5 + (1 - h_5^2)d - 2h_6ch_4 - 2h_5h_4b - h_4^2a - h_6^2f \\ (1 - h_6h_8 - h_5h_9)e - h_5h_8d + (-h_6h_7 - h_4h_9)c + (-h_5h_7 - h_4h_8)b - h_4h_7a - h_6fh_9 \\ (-h_9h_2 - h_8h_3)e - h_8h_2d + (1 - h_9h_1 - h_7h_3)c + (-h_8h_1 - h_7h_2)b - h_7h_1a - h_9fh_3 \\ (1 - h_6h_8 - h_5h_9)e - h_5h_8d + (-h_6h_7 - h_4h_9)c + (-h_5h_7 - h_4h_8)b - h_4h_7a - h_6fh_9 \\ (1 - h_9^2)f - 2h_9eh_8 - h_8^2d - 2h_9ch_7 - 2h_8h_7b - h_7^2a \end{bmatrix} \tag{6.18}$$

Collecting the terms for a, b, c, d, e, f for each row of the left-hand side gives the matrix equation below:

$$\mathbf{A}_i\mathbf{c} = 0$$

[2] Hartley[30] omits this step and it is presented here for completeness and easy following by readers not familiar with SVD's and direct linear methods.

$$\begin{bmatrix} -h_1^2+1 & -2h_2h_1 & -2h_3h_1 & -h_2^2 & -2h_3h_2 & -h_3^2 \\ -h_1h_4 & 1-h_2h_4-h_1h_5 & -h_3h_4-h_1h_6 & -h_2h_5 & -h_3h_5-h_2h_6 & -h_3h_6 \\ -h_7h_1 & -h_8h_1-h_7h_2 & 1-h_9h_1-h_7h_3 & -h_8h_2 & -h_9h_2-h_8h_3 & -h_9h_3 \\ -h_1h_4 & 1-h_2h_4-h_1h_5 & -h_3h_4-h_1h_6 & -h_2h_5 & -h_3h_5-h_2h_6 & -h_3h_6 \\ -h_4^2 & -2h_5h_4 & -2h_6h_4 & 1-h_5^2 & -2h_6h_5 & -h_6^2 \\ -h_4h_7 & -h_5h_7-h_4h_8 & -h_6h_7-h_4h_9 & -h_5h_8 & 1-h_6h_8-h_5h_9 & -h_6h_9 \\ -h_7h_1 & -h_8h_1-h_7h_2 & 1-h_9h_1-h_7h_3 & -h_8h_2 & -h_9h_2-h_8h_3 & -h_9h_3 \\ -h_4h_7 & -h_5h_7-h_4h_8 & -h_6h_7-h_4h_9 & -h_5h_8 & 1-h_6h_8-h_5h_9 & -h_6h_9 \\ -h_7^2 & -2h_8h_7 & -2h_9h_7 & -h_8^2 & -2h_9h_8 & 1-h_9^2 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \end{bmatrix} = 0 \tag{6.19}$$

For each of the n homographies $\mathtt{H}_i$ it is possible to create the corresponding matrix $\mathbf{A}_i$. A $9n \times 6$ matrix $\mathbf{X}$ is created by stacking the matrices $\mathbf{A}_i$ on top of each other. It is possible to solve for $\mathbf{c} = (a, b, c, d, e, f)^\mathsf{T}$ by finding the SVD of $\mathbf{X}^\mathsf{T}\mathbf{X}$.

The entries of $\mathbf{c}$ can be reshaped into the matrix $\mathbf{C}$ and the intrinsic matrix $\mathbf{K}$ can be found by taking the Cholesky decomposition[115] of $\mathbf{C} = \mathbf{K}\mathbf{K}^\mathsf{T}$. The matrix $\mathbf{C}$ must be positive definite[3] to find the Cholesky decomposition.

6.4.10 Rotation matrices

Once the intrinsic parameters are known, the rotation matrix $\mathbf{R}_i$ for each view can then be found by substituting $\mathbf{K}$ into:

$$\mathbf{R}_i = \mathbf{K}^{-1}\mathtt{H}_i\mathbf{K} \tag{6.20}$$

As the homographies are pairwise, the rotation is relative to each pair of homographies. Therefore to find the total rotation for any camera view from the reference view it is necessary to chain add the rotations to get the total rotation. The chain adding process is identical to that of chain multiplying the homographies in Section 6.4.5.

[3] Ironically, the fastest way computationally to find out if a matrix is positive definite or not, is to find the Cholesky decomposition.

6.4.10.1 Extracting the yaw, pitch and roll from a rotation matrix

One of the goals of the calibration is to find the pan and tilt angles (from which a pure rotation matrix can be generated). Therefore, it is necessary to extract the yaw, pitch and roll from the estimated rotation matrix. Once this is done, only the yaw and pitch angles are kept and the roll is assumed to be zero.

Using a method from LaValle[168], it is possible to extract the yaw α, pitch β and roll γ from an arbitrary 3×3 rotation matrix $\mathbf{R}$ as follows:

$$\mathbf{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

$$\alpha = \arctan\left(\frac{r_{11}}{r_{21}}\right) \quad \beta = \arctan\left(\frac{\sqrt{r_{32}^2 + r_{33}^2}}{-r_{31}}\right) \quad \gamma = \arctan\left(\frac{r_{32}}{r_{33}}\right) \tag{6.21}$$

It is sometimes necessary to add multiples of 90° to the angles to rotate them to coincide with the correct orientation of the world x, y and z axes. This can also be achieved by pre-multiplying $\mathbf{R}$ by the appropriate rotation matrix. If $\mathbf{R}$ is not a perfect rotation matrix, as is often the case from numerically estimated rotation matrices, results can sometimes be unpredictable. A simple test should be performed to validate the angles by reconstructing the rotation matrix from α, β and γ and comparing it to $\mathbf{R}$. When implementing this in C/C++ the `atan2` function should be used to ensure the angles are in the correct quadrants[4].

6.4.11 Bundle II – Refining the intrinsic matrix and view rotations

A bundle adjustment is set up to determine the rotation for each view and a parameter constant over all views, consisting of the radial distortion and camera intrinsic matrix at the minimum zoom. The global feature positions are again parameterised as 3-vectors $(x_i, y_i, 1)$ and each rotation is parameterised as a 2-vector of pan and tilt (θ, ϕ). The constant parameter for all views is given by $(\kappa_1, \kappa_2, \kappa_3, x_c, y_c, \alpha, f_y)^\mathsf{T}$.

[4]There is an error in LaValle[168] where it is assumed that `atan2` returns angles in the range 0 to 2π, however the ANSI standard function returns angles in the range $-\pi$ to π. As the online edition is a draft copy, the print edition may have this error corrected.

Variable	*Size*	*Parameter*	*Initial Value*
Partition $\mathbf{a}$	$\mathbf{a} = (\theta_1, \phi_1, \theta_2, \phi_2, \ldots, \theta_m, \phi_m)^\mathsf{T}$		
	$2m$ element vector	Pan and tilt angles (θ, ϕ) for each camera view	Hartley's linear algorithm and Section 6.4.10
Partition $\mathbf{b}$	$\mathbf{b} = (\kappa_1, \kappa_2, \kappa_3, x_c, y_c, \alpha, f_y)^\mathsf{T}$		
	7-vector	Intrinsic parameters and radial distortion parameters	$\kappa_1, \kappa_2, \kappa_3, x_c$ and y_c are taken from Bundle I and the intrinsic parameters (α,f_y) are found by Hartley's linear algorithm
Partition $\mathbf{c}$	$\mathbf{c} = (x_1, y_1, w_1, x_2, y_2, w_2, \ldots, x_n, y_n, w_n)^\mathsf{T}$		
	$3n$ element vector	Global features in reference view	From the global feature list and initialised as points on the front cube face i.e. $(x_i, y_i, 1)$
Measurement vector $\mathbf{X}$	$2p$ element vector	Global feature positions in each view	Initialised from the global feature list as 2D points $(\tilde{x}_i, \tilde{y}_i)$

6.4.12 Zoom Calibration

The methods used in the zoom calibration are almost identical to the calibration at the minimum zoom with some minor adjustments. Frames are captured from the camera with the pan and tilt set to a fixed direction. Corner features are extracted from each image using a Harris corner detector[164] (as in Section 6.4.3) and pairwise homographies are found. A pairwise bundle adjustment is performed to estimate radial distortion parameters as well as the intrinsic parameters in the second view (the intrinsic parameters at the minimum zoom are kept constant). Using the parameter uncertainty calculated from the Hessian matrix, the radial distortion parameters are clamped to zero if they become too uncertain. Finally, all the intrinsic parameters and radial distortion parameters are globally optimised by a final bundle adjustment.

6.4.12.1 Pairwise Homographies

As in the rotation calibration, corners are found in each image and a homography between images is found by the RANSAC method in Section 6.4.4.2. This is followed by a maximum likelihood estimation step to find the homography between the views using the inlier corner features as in Section 6.4.4.3.

6.4.12.2 Pairwise Bundle Adjustment

The intrinsic matrices of any two levels of zoom are related by the homography between the views as follows:

$$\mathbf{K}' = \mathtt{H}\mathbf{K} \tag{6.22}$$

Since the homography in the lower zoom level and the pairwise homography are known, $\mathbf{K}'$ can be found.

From the previous step, a list of pairwise point matches exist. These are used to create the *global feature list*. Each global point is the average point between the two zoom levels and the reprojection error between views is used as the maximum likelihood estimator cost function.

Only the radial distortion parameters in the second view are estimated. This maintains the consistency of the radial distortion and prevents a distortion in one view being corrected for in a second view. Radial distortion parameters are clamped if their uncertainty exceeds a threshold and are excluded from the next step of the optimisation. This is done by calculating the covariance matrix (see Equation B.28 in Appendix B). For example, when using two radial distortion coefficients, κ_2 is first clamped to zero once it becomes too uncertain. Then, when κ_1 drops below a threshold it is also clamped to zero and radial distortion is completely ignored. For each zoom step there is a variable number of radial distortion κ parameters, i.e. two, one or zero. The value of the aspect ratio α is kept constant throughout the bundle adjustment.

6.4.12.3 Final Bundle Adjustment

A final bundle adjustment is performed to estimate the full set of intrinsic parameter matrices and radial distortion parameters. Each zoom step is parameterised by at most a 5-vector: $(f_y, x_c, y_c, \kappa_1, \kappa_2)^\mathsf{T}$. If both radial distortion parameters have been clamped it will become the 3-vector: $(f_y, x_c, y_c)^\mathsf{T}$. The aspect ratio α becomes a constant, common to all views and is held constant. The global feature list is created in the same fashion as in the rotation calibration described in Section 6.4.6. Figure 6.10 shows an example global feature list for a zoom sequence – note the scale change of the features in each camera view. A two partition bundle adjustment was performed and the parameter vector was split as follows:

Variable	*Size*	*Parameter*	*Initial Value*
Partition a	$\mathbf{a} = (f_{y1}, x_{c1}, y_{c1}, \kappa_{11}, \kappa_{21}, \ldots, f_{y2}, x_{c2}, y_{c2}, \kappa_{12}, \ldots, f_{ym}, x_{cm}, y_{cm})^{\mathsf{T}}$		
	$5m$ element vector *(maximum)*	Intrinsic and radial distortion parameters for each camera view	From pairwise bundle adjustment
Partition b	$\mathbf{c} = (x_1, y_1, w_1, x_2, y_2, w_2, \ldots, x_n, y_n, w_n)^{\mathsf{T}}$		
	$3n$ element vector	Global features in reference view	From the global feature list and initialised as points on the front cube face i.e. $(x_i, y_i, 1)$
Measurement vector $\mathbf{X}$	$2p$ element vector	Global feature positions in each view	Initialised from the global feature list as 2D points $(\tilde{x}_i, \tilde{y}_i)$

The Jacobians are calculated per view, so the number of radial distortion parameters can be determined from the dimensions of $\mathbf{a}_j$. The measurement vector and the global features are identical to previous bundle adjustments.

6.5 Camera Model

The result of the camera calibration is a complete camera model that is parameterised by three parameters: the Sony EVI-D31 pan index (p), tilt index (t) and zoom index (z). These three parameters need to map to the camera intrinsic and extrinsic parameters and radial distortion parameters.

Section 6.5.1 outlines the fitting of the extrinsic parameters (i.e. the pan and tilt angles) from the camera pan and tilt indexes. The same treatment is given to the intrinsic parameters in Section 6.5.2 and a piecewise function is used to estimate the radial distortion at each zoom index. For the numerical results for this model refer to Section 6.6.3.

6.5.1 Camera pan and tilt model (Extrinsic Parameters)

Using the camera pan and tilt index from each view it is possible to create a mapping to the yaw and pitch angles using a linear least squares method. The yaw and pitch are not independent

and result in an equation of the form:

$$\theta = ap + bt + c \tag{6.23}$$

$$\phi = dp + et + f \tag{6.24}$$

where	p	camera pan index
	t	camera tilt index
	θ	pan angle in radians
	ϕ	tilt angle in radians
	a, b, c, d, e, f	fit parameters

In matrix form the above equations becomes:

$$\begin{pmatrix} \theta \\ \phi \end{pmatrix} = \begin{bmatrix} p & t & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & p & t & 1 \end{bmatrix} \begin{pmatrix} a \\ b \\ c \\ d \\ e \\ f \end{pmatrix} \tag{6.25}$$

It is possible to solve for the fit parameters by setting up the relation below by stacking many observations of the pan and tilt angles versus the camera pan and tilt indexes:

$$\mathbf{Ax} = \mathbf{b}$$

$$\begin{bmatrix} p_1 & t_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & p_1 & t_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ p_m & t_m & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & p_m & t_m & 1 \end{bmatrix} \begin{pmatrix} a \\ b \\ c \\ d \\ e \\ f \end{pmatrix} = \begin{pmatrix} \theta_1 \\ \phi_1 \\ \vdots \\ \theta_m \\ \phi_m \end{pmatrix} \tag{6.26}$$

Taking the pseudoinverse of $\mathbf{A}$ and multiplying by $\mathbf{b}$ solves for $\mathbf{x}$. Now the fit parameters are known and the following mapping can be used to translate the camera pan and tilt parameters

(p, t) to pan and tilt angles (θ, ϕ) by evaluating Equation 6.25:

$$(p, t) \mapsto (\theta, \phi) \tag{6.27}$$

6.5.2 Zoom Model (Intrinsic Parameters and Radial Distortion)

The focal length for each step of the zoom calibration is known for each camera zoom parameter. A least squares exponential model of the focal length can be found using the Curve Fitting Toolbox in MATLAB in the form:

$$f(z) = ae^{bz} + ce^{dz} \tag{6.28}$$

where	$f(z)$	focal length as a function of z
	z	camera zoom parameter
	a, b, c, d	fit parameters

A least squares exponential fit for the principal point was found using MATLAB to give two expressions for x_c and y_c as functions of the zoom parameter z:

$$x_c(z) = ae^{bz} + ce^{dz} \tag{6.29}$$

$$y_c(z) = fe^{gz} + he^{iz} \tag{6.30}$$

where	z	camera zoom parameter
	a, b, c, d	fit parameters
	f, g, h, i	fit parameters

The radial distortion is estimated by a linear piecewise function between each optimised radial distortion point for every camera zoom level. A simple lookup table is used to store the radial distortion parameters. The aspect ratio α is constant for all zoom levels. From these parameters an intrinsic camera matrix and radial distortion parameters can be estimated for every camera zoom level.

6.5.3 Integration in tracking framework

Once the calibration has been completed it is necessary to integrate it into the tracking framework described in Chapter 4. The transition between using homographies and the projection matrices used by OpenGL style rendering environments is fairly simple: homographies represent the transfer mechanism for 2D points between image planes and the projection matrices transform 3D points between viewing frustums. Hence, an extra dimension is added to all points for 3D viewing frustums which ensures that all points lie within the same volume of world space.

Consider an arbitrary homography that projects 2D points (represented as 3-vectors) from one image into another:

$$\mathbf{x}' = \mathtt{H}\mathbf{x}$$

$$\begin{pmatrix} x' \\ y' \\ w' \end{pmatrix} = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix} \begin{pmatrix} x \\ y \\ w \end{pmatrix}$$

At this point the normal procedure is to divide $\mathbf{x}'$ through by the w parameter, which effects the projection of the point onto the image plane (or the front cube face which is indicated by $w = 1$). In a 3D viewing volume, however, the depth information needs to be kept and each point $\mathbf{x}$ is now represented by a 4-vector:

$$\mathbf{x}' = \mathtt{P}_{\mathtt{H}}\mathbf{x}$$

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{bmatrix} h_1 & h_2 & h_3 & 0 \\ h_4 & h_5 & h_6 & 0 \\ h_7 & h_8 & h_9 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$

In OpenGL style rendering environments it is convention to separate the projection matrix into a model view matrix and a projection matrix. The model view matrix represents the transform applied to an object (or model) and the projection matrix is used to set the camera view of the scene. For example, a spinning top should be animated by modifying the model view matrix and a camera fly-by should be animated by modifying the projection matrix. Therefore, as

the homography between two views affects the 'position' of the camera, the projection matrix should be set to $\mathtt{P}_{\mathtt{H}}$.

When using the full camera model (represented as intrinsic and extrinsic parameters) the transform between two images is given by the 3×3 homography:

$$\mathtt{H} = \mathbf{N}'\mathbf{K}'\mathbf{R}'\mathbf{R}^{-1}\mathbf{K}^{-1}\mathbf{N}^{-1} \tag{6.31}$$

where $\mathbf{N}', \mathbf{N}$ normalising matrices (normally $\mathbf{N}' = \mathbf{N}$)
$\mathbf{K}, \mathbf{R}$ intrinsic and extrinsic (rotation) matrices for first view
$\mathbf{K}', \mathbf{R}'$ intrinsic and extrinsic (rotation) matrices for second view

OpenGL uses a left-handed co-ordinates system, where the z-axis is orientated into the screen, so often it is necessary to pre-multiply the projection matrix by a transform matrix that flips the z-axis.

If $\mathtt{P}$ is the viewport projection matrix from Equation 3.10 in Section 3.6.1 on page 47 then the OpenGL projection matrix is given by:

$$\text{projection matrix} = \mathtt{P}\mathbf{T}\mathtt{P}_{\mathtt{H}} \tag{6.32}$$

where $\mathtt{P}$ 4×4 perspective projection matrix
$\mathbf{T}$ transform matrix to flip z-axis and align world space with the viewport (if necessary)
$\mathtt{P}_{\mathtt{H}}$ homography between views as given in Equation 6.31 reshaped into a 4×4 matrix

Using the projection matrix above it is now possible to treat the images in a pan, tilt and zoom sequences as texture mapped planes in 3D world space and to transform points between them.

6.6 Calibration and Results

The self-calibration techniques discussed in the previous chapters were implemented and tested on several sequences. The results from the calibration on each sequence were put together into a single camera model for the Sony EVI-D31 PTZ camera.

Section 6.6.1 outlines the sequences used and the factors affecting the choice of each sequence. An implementation of the self-calibration methods discussed before is presented in Section 6.6.2 and the results produced can be found in Section 6.6.3. The verification and accuracy of the camera model developed is explored in Section 6.7.

6.6.1 Sequences

Several sequences were captured using the Sony EVI-D31 PTZ camera for calibration purposes. Two types of sequences were captured: the first for rotation, captures images on a grid with only changing the pan and tilt angles and the second, fixes the camera pan and tilt position while incrementally zooming in from a wide angle view. For all sequences the camera parameters were also recorded by querying the camera using Sony's VISCA protocol (Appendix D.2) before capturing each frame. Each 'frame' is an average of 5 frames captured of the same view. This was done to eliminate slight vibrations due to the camera settling and flickering in the scene from 50 Hz electric lighting for indoor scenes. Most sequences took a few minutes to capture. Table 6.1 below summarises the captured sequences:

Sequence	*Pan*	*Tilt*	*Zoom*	*Frames*	*Grid*
`lab1`	-300 – +300	-282 – +283	0	169	13×13
`lab2`	-200 – +200	-282 – +283	0	121	11×11
`lab3`	-300 – +300	-282 – +283	0	35	5×7
`jcmb1`	-400 – +400	-282 – +283	0	63	11×11
`outside1`	-400 – +400	-282 – +283	0	49	9×7
`lab1`	0	0	0 – 1023	69	–
`lab2`	0	0	0 – 1023	103	–

Table 6.1: *Summary of calibration sequences*

The first of the minimum zoom calibration sequences is `lab1` (shown in Figure 6.11) where a grid of 169 images was captured by incrementally capturing images on a grid with the camera pan and tilt index by 40. An indoor scene was used so that the scene could remain static without objects moving in the wind and to avoid swift illumination changes (both are very common in Scotland). `lab2` (Figure 6.12) is much the same as `lab1` except that less frames were captured and a smaller tilt range was used. The sequence was captured from a slightly different position and at a different time.

The last of the indoor sequences is `lab3`, shown in Figure 6.13, was one of the first sequences to be captured. It has relatively few frames (only 35) and was used for initial testing and fast

processing. The overlap in each image is much smaller than the previous two sequences. It was captured at night to avoid any illumination changes from natural lighting outside.

`jcmb1` (shown in Figure 6.14) is the first of the two outdoor scenes and was taken from the fourth floor of a building looking onto other buildings. The scene was chosen as there are lots of corners and very little sky is visible for a large pan and tilt range. As most of the objects in the scene are fixed there would be very little change in the scene during the capture process. Unfortunately the window is in the foreground, but it could not be opened as a strong wind was blowing which added large amounts of vibration to all the images.

The second outdoor scene, illustrated in Figure 6.15, is a view of trees and buildings with some sky visible. The scene was captured on a fairly still day, so there is little movement in the trees and the clouds in the sky do not move too much. Due to the grid-like nature of the sequence clouds visible in two rows of the grid could move up to 50 pixels on a windy day. This adds massive errors if the only objects to align the images from are clouds. Therefore it was necessary to ensure that all frames contained something other than clouds.

Two zoom sequences (Figure 6.16 and Figure 6.17) were taken from two slightly different positions of an indoor scene. The first has less frames than the second. Using outdoor scenes proved to be difficult due to objects moving in the wind or reflections of objects moving in the wind. Indoor scenes provide equally good calibrations as outdoor scenes as most objects in the scene are far enough away from the camera that there are no depth of field effects.

6.6.2 Implementation

The calibration framework was implemented as a library and C++ classes with a Windows GUI interface to view points and registration errors, and a console application to perform the calibration. As a by-product the console application also stitched high resolution panoramic images. An application (see Appendix D.4.2) was also developed to capture images either in a grid or along a random series of movements. The camera parameters associated with each frame are stored as well.

The C++ classes use libraries developed for this work as described in Appendix D.3. Wrapper classes were implemented from the matrix classes for dense and sparse matrices in the VXL[21] libraries. The Harris corner detector and subpixel corner routines from OpenCV[22] were used. Initially routines from Numerical Recipes in C[115] were used however they were found to

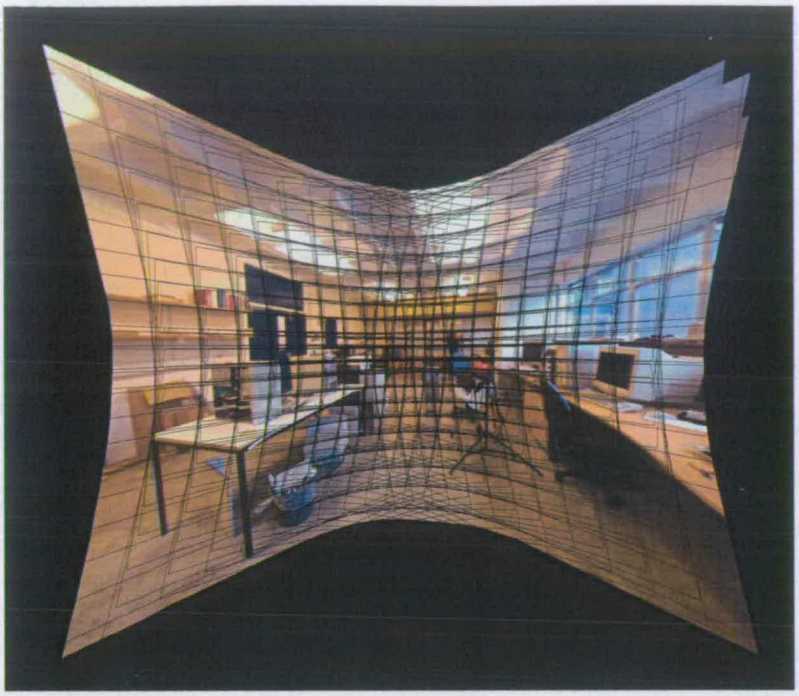


Figure 6.11: *Pan and tilt calibration sequence* `lab1`



Figure 6.12: *Pan and tilt calibration sequence* `lab2`



Figure 6.13: *Pan and tilt calibration sequence* `lab3`

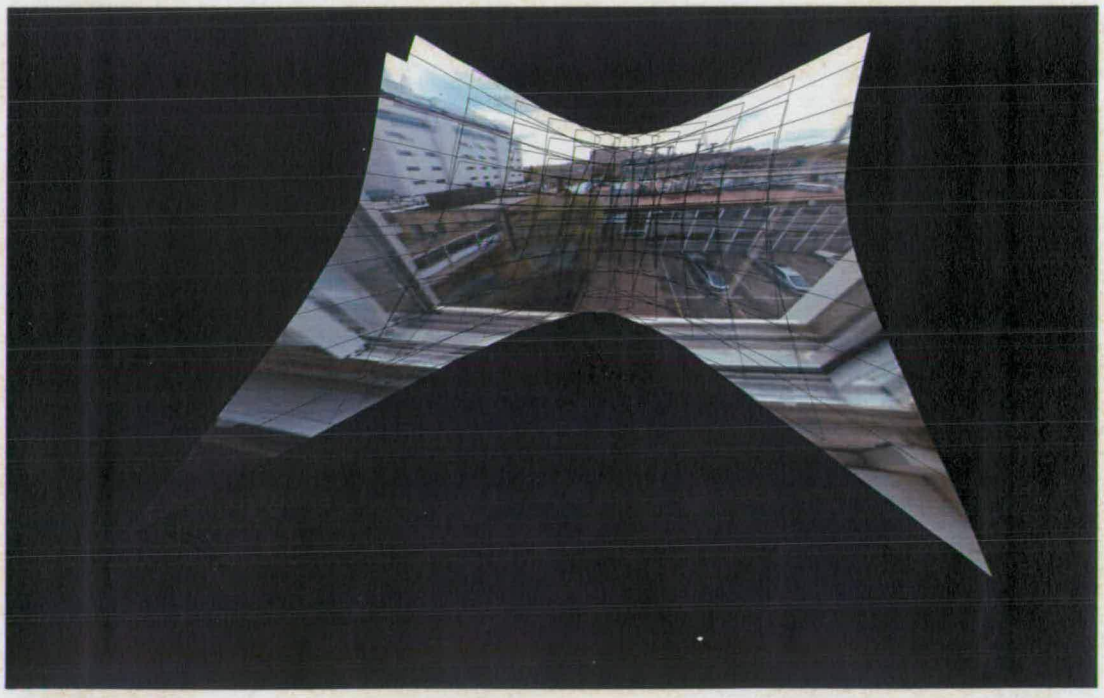


Figure 6.14: *Pan and tilt calibration sequence* `jcmb1`

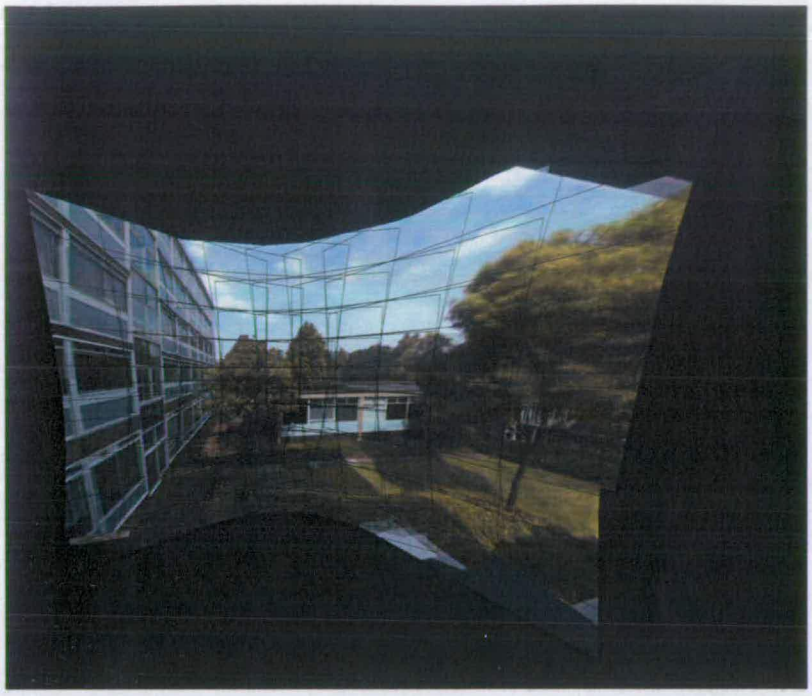


Figure 6.15: *Pan and tilt calibration sequence* `outside1`

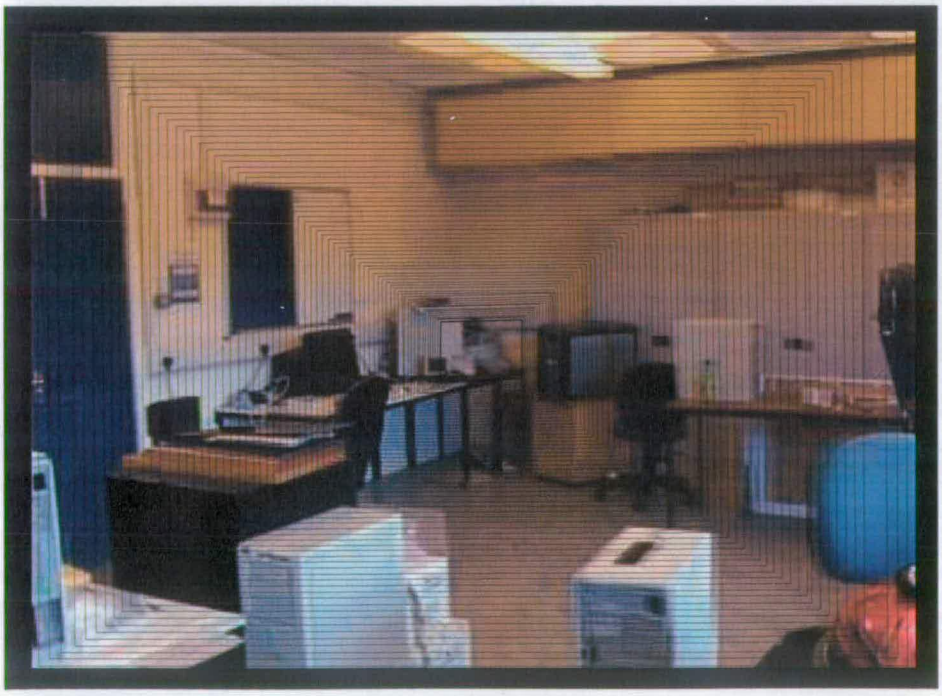


Figure 6.16: *Zoom calibration sequence* `lab1`



Figure 6.17: *Zoom calibration sequence* `lab2`

contain several bugs and produced incorrect and unreliable results. The VXL libraries wrap conversions of LINPACK[169] code and produce reliable results[5] with fast execution times.

Some of the classes are loosely based on source code kindly supplied by Sudipta Sinha[28, 117]. Sinha's code was used verbatim for doing the pairwise bundle adjustment in the zoom calibration as this method is poorly explained in his paper[28] and many subtle details of the code are not included in the paper. Otherwise, all other code was developed independently from other sources[29, 30, 34, 111] and Sinha's code was used merely as a comparison.

Runtime was fairly slow and depended largely on the number of corner matches used between images. The capture process took a few minutes (up to 10 minutes for large grids). Extracting corners and doing the minimum zoom registration took from 5 to 15 minutes, depending on the number of images. The zoom sequence was considerably quicker taking only a few minutes to compute completely. Large bundle adjustments (for upwards of 2000 points) run fairly slowly taking almost around 30 seconds per iteration and requiring anywhere from 15 to 50 iterations to converge. Many portions of the code could be optimised to run faster. Freely available implementations of panorama stitchers, such as PanoTools [6] and Hugin[7], execute faster than the stitcher implemented for this work. A complete calibration, rotation and zoom with image capture, could be completed in roughly 30 minutes for a good pan and tilt range. As the calibration is automatic it only requires the camera to be set up in the scene and then requires no more operator intervention. The results of this calibration can then be loaded into a class that implements the model described in Section 6.5. The model class will produce either 3×3 homography matrices or 4×4 projection matrices given the pan, tilt and zoom indices. The 4×4 projection matrix is useful to create the model view and camera matrices for an OpenGL style rendering environment.

6.6.3 Results – Camera Model

Each sequence produced an estimate of the camera's intrinsic and extrinsic parameters. The measure of the accuracy of the calibration is determined from the reprojection error. The table below outlines the reprojection error for each calibration sequence.

[5] Many of MATLAB's numerical routines call methods from LINPACK.

[6] PanoTools is a freely available set of panorama tools written by Helmet Dersch and are available online from `http://panotools.sourceforge.net/`.

[7] Hugin is a SourceForge panorama application and contains a freeware stitcher called Nona. It is available online from `http://hugin.sourceforge.net/`.

Sequence	Mean (pixels)	Variance (pixels)	No of points
`lab1`	0.6621	1.0038	13 967
`lab2`	0.5340	0.9350	21 787
`lab3`	3.3364	1.5485	4 405
`outside1`	1.1953	1.2602	5 573
`jcmb1`	1.4517	1.3308	8 012

Table 6.2: *Residual reprojection error for camera calibration sequences*

The camera's intrinsic, extrinsic and radial distortion parameters can be determined by three integer parameters: pan, tilt and zoom index. Therefore it was possible to create a mapping that determined the camera matrices and radial distortion parameters by fitting a model to the calibration data.

A least squares fit of the calibration data for each of the parameters produced the following model for the Sony EVI-D31 PTZ camera:

Symbol	Description	Fit Function Form	Values
α	Aspect Ratio	Constant	0.973795
$f(z)$	Focal Length	$ae^{bz} + ce^{dz}$	$1.067e^{0.00095z} + 0.2503e^{0.00383z}$
$x_c(z)$	Principal point x	$ae^{bz} + ce^{dz}$	$0.00033e^{-0.00033z} + 0.00045e^{0.00398z}$
$y_c(z)$	Principal point y	$ae^{bz} + ce^{dz}$	$-0.0012e^{0.0034z} + 0.00241e^{-0.00183z}$
$(p,t) \mapsto (\theta,\phi)$	θ pan angle ϕ tilt angle	$\theta = ap + bt + c$ $\phi = dp + et + f$	$\theta = -0.0020p + 1.032 \times 10^{-5}t - 0.004$ $\phi = 1.837 \times 10^{-5}p + 0.0014t - 0.011$

where p Pan index (integer from -880 to 880)
t Tilt index (integer from -330 to 330)
z Zoom index (integer from 0 to 1023)

Table 6.3: *Reprojection error for camera calibration sequences*

The radial distortion parameters for varying zoom indexes are estimated from a linear piecewise function. A single set of radial distortion parameters was chosen to avoid ambiguities caused by combining the different data from each calibration due to the uncertainty in the estimation method.

The results for the pan and tilt angles *vs* camera parameters, shown in Figure 6.18, were much as expected. It is clear from the results, that mechanically, the camera pan/tilt mechanism is not perfect as the pan and tilt angles are not independent. This also implies that the rotation is actu-

ally around a locus rather than a point, however this can be accurately modelled by dependency of the pan and tilt angles on each other. The focal length (Figure 6.20) was well estimated by both sequences and remains smooth over the range of camera zoom parameters. The results obtained also reflect the physical situation accurately and when converted to millimetres and magnification factors, match the camera specification.

Estimating the principal point is a fairly difficult problem and the two spikes visible in Figure 6.20 are the result of the radial distortion being clamped to zero and the principal point being used to compensate for the absence of radial distortion in later zoom steps. As the Sony EVI-D31 is a piece of consumer electronics, the quality of its optics are not going to be fantastic. Therefore a significant amount of drift in the principal point is to be expected. There are also large difficulties in estimating in the principal point which have been explored in [104].

The estimated radial distortion follows a similar trend to the results obtained by Sinha[28] for a different model of the Sony camera. The data used for the piecewise radial distortion function was chosen as that from `lab1`. Although this is not ideal, the results it produced appeared to be adequate and Section 8.3 discusses improvements to the calibration method that would produce more accurate results.

Table 6.2 shows that all calibrations were successful as the residual error is near one pixel for all sequences with a small variance over thousands of points. The results from the calibration are verified in the next section (Section 6.7) and appear to be accurate enough to produce high resolution panoramas (or mosaics) using only the camera parameters, meaning the views are well registered for a large range of independent sequences.

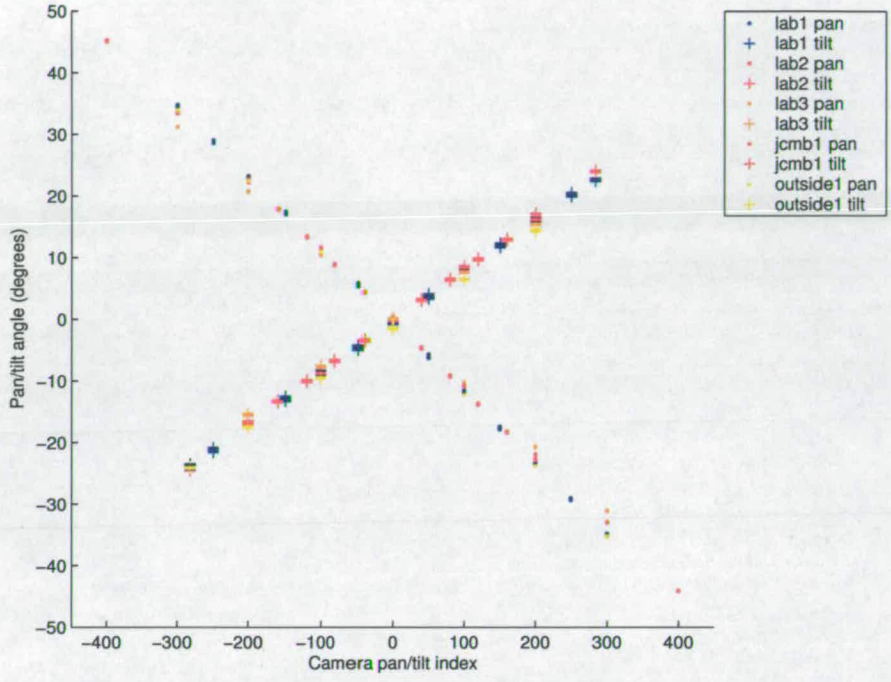


Figure 6.18: *Pan and tilt angles (degrees) vs camera pan and tilt indexes*

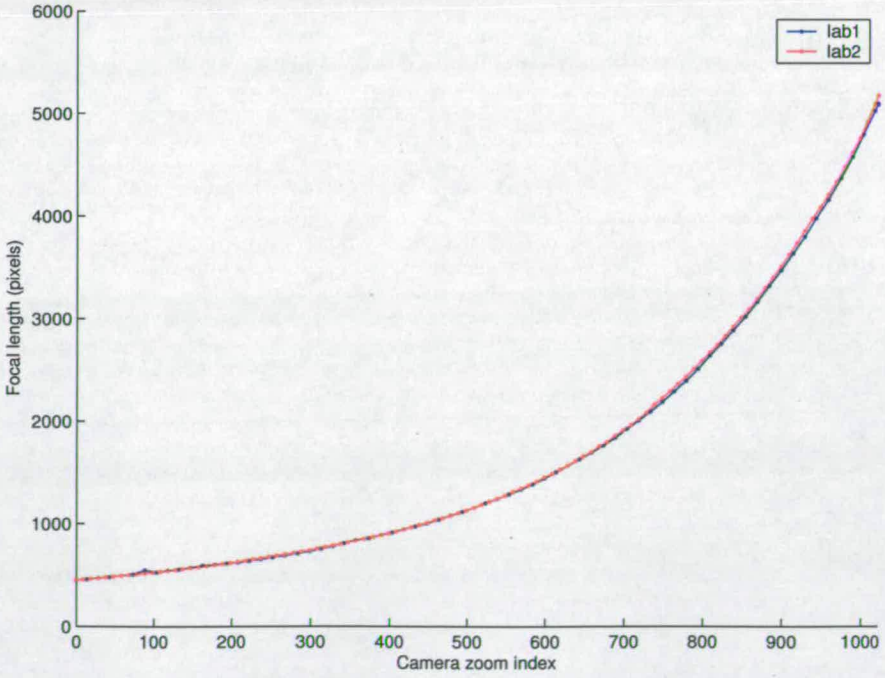


Figure 6.19: *Focal length (pixels) vs camera zoom parameter*

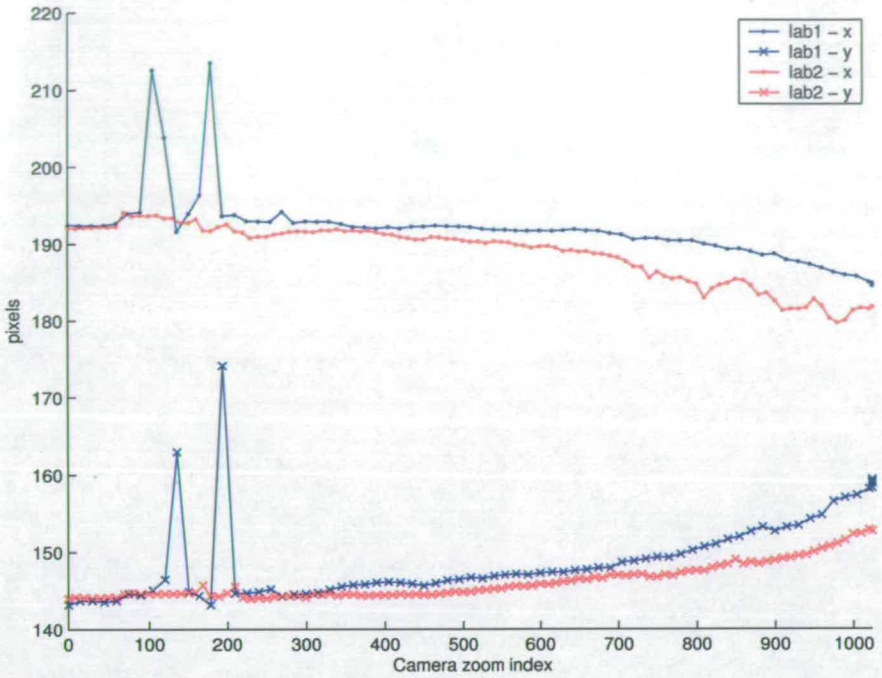


Figure 6.20: *Principal point (pixels) vs camera zoom index*

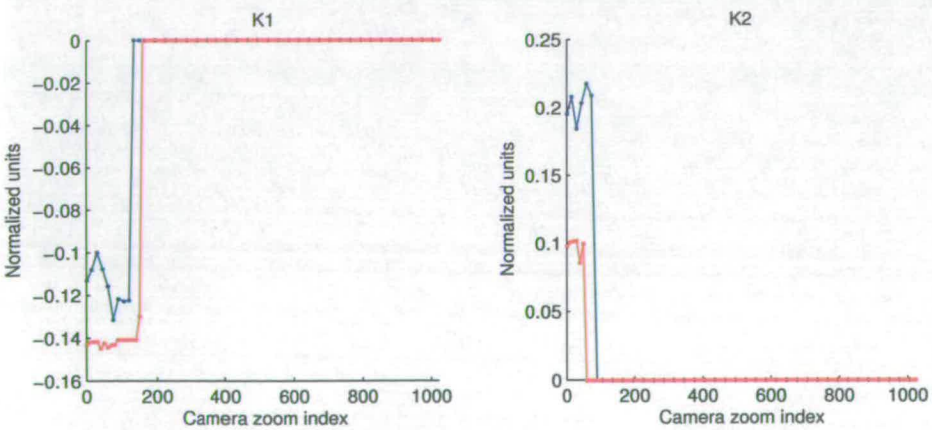


Figure 6.21: *Radial distortion parameters vs camera zoom index*

6.7 Model Verification

If the model of the camera is good enough, then it is possible to create seamless panoramic (or mosaiced) images using *only* the camera pan, tilt and zoom indexes. This feed-forward method also verifies the quality of the model. Two types of tests were performed, one where regular increments of pan and tilt were performed to produce a 'virtual wide angle lens' view of a scene and the second was to do several random camera movements with varying pan, tilt and zoom. The composites of each of the tests are shown in Figure 6.22 to Figure 6.26.



Figure 6.22: `lab1` *rendered to the front cube face from camera parameters using the camera model*

It is possible to see small errors in registration which are due to the clamping of the radial distortion parameters at low zoom factors and uncertainty in the principal point. This could be improved by adding an extra calibration step, as discussed in Section 8.3, where a random path with varying intrinsic parameters is bundle adjusted to improve the estimate of the principal point and radial distortion parameters.

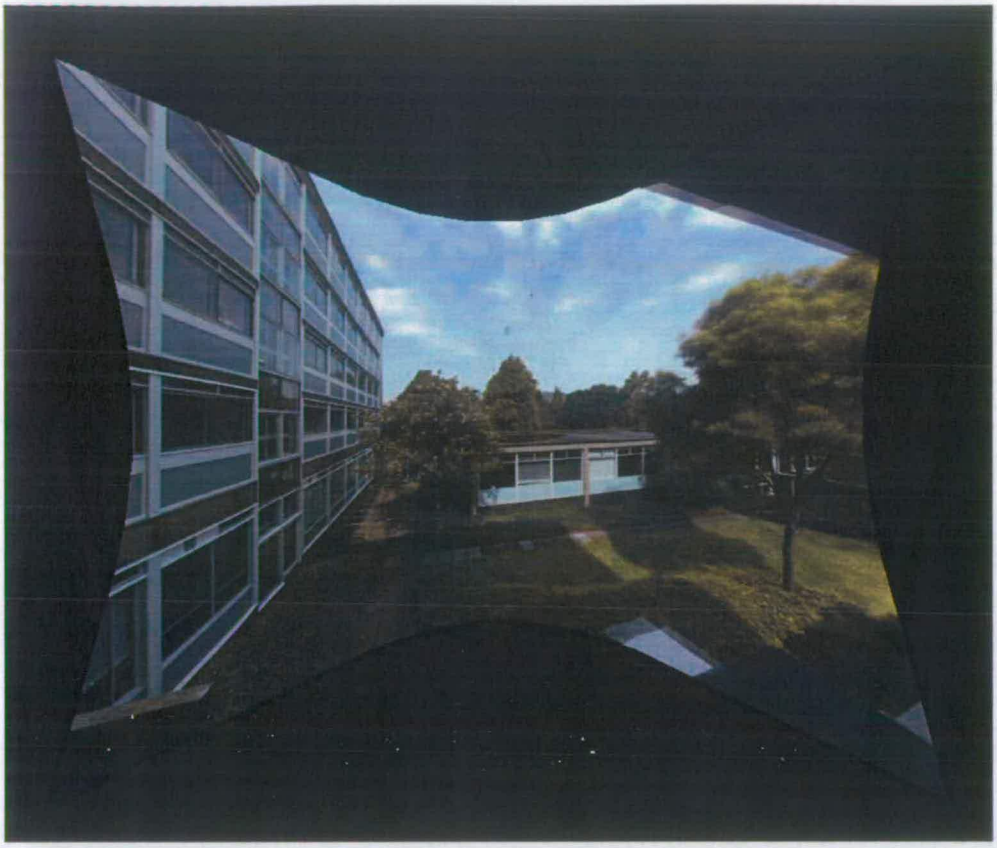


Figure 6.23: `outside1` *rendered to the front cube face from camera parameters using the camera model*

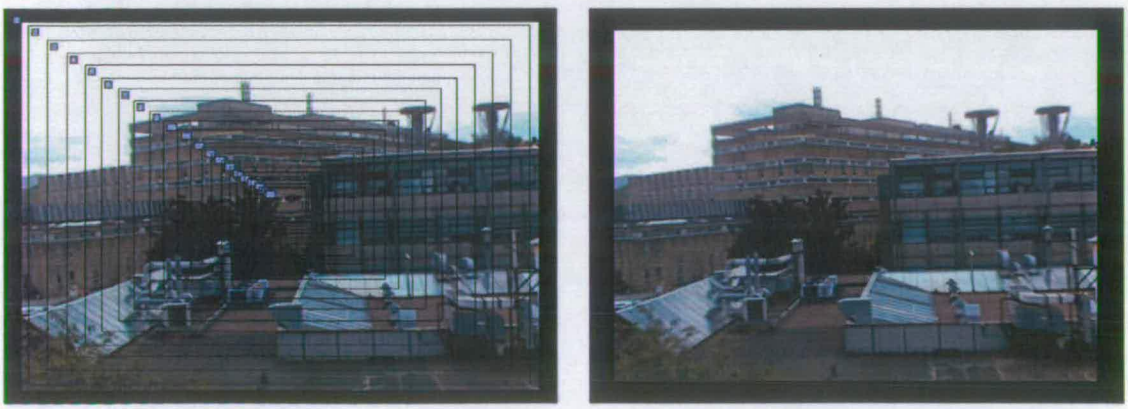


Figure 6.24: *Zoom sequence rendered to the front cube face from camera parameters using the camera model*

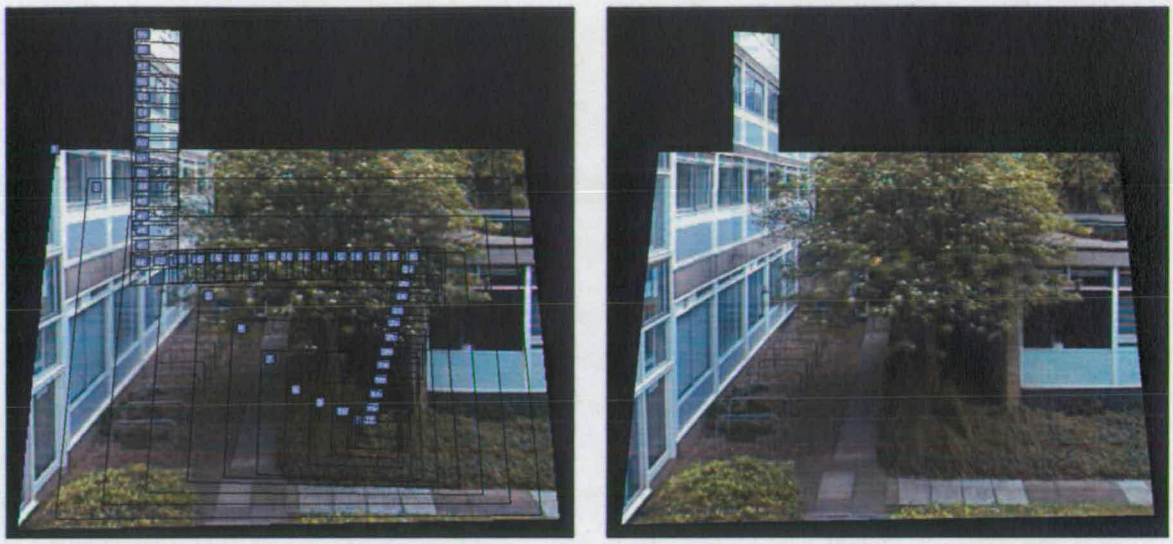


Figure 6.25: *A random series of camera pans, tilts and zooms rendered to the front cube face using the camera model*

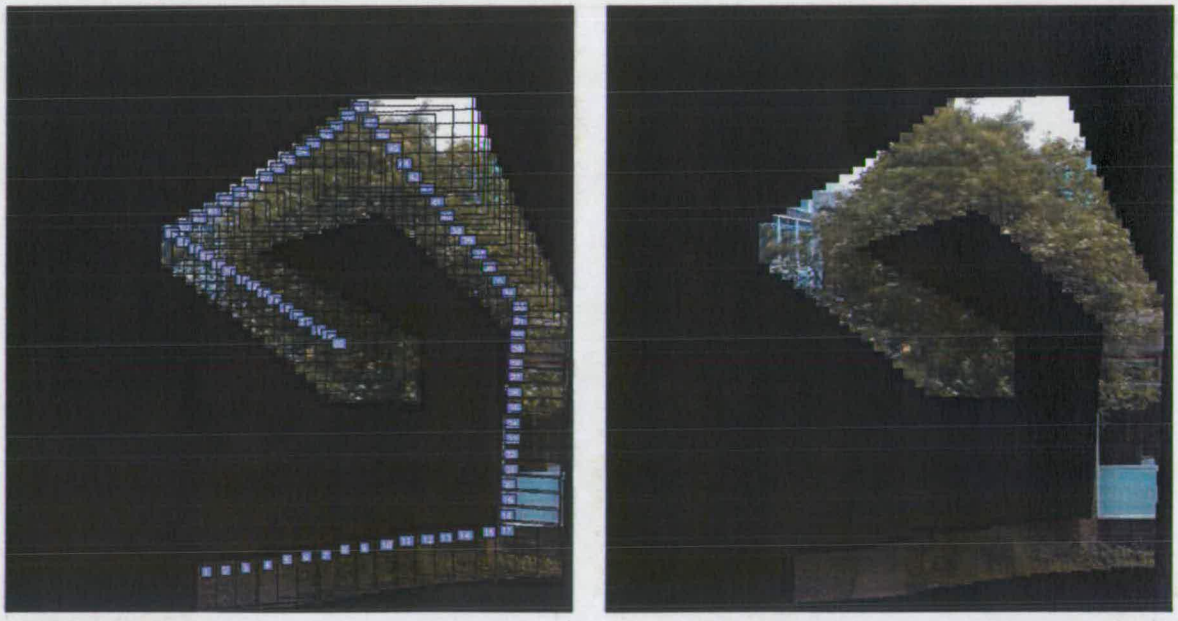


Figure 6.26: *A random series of camera pans, tilts and zooms rendered to the front cube face using the camera model*

6.8 Summary

This chapter has discussed the possible camera calibration techniques that exist and the reasons for choosing Sinha's[28] self-calibration method for PTZ cameras. A review of this method and the theory that accompanies it, such as bundle adjustment and linear equation solving methods, were presented with references to full details in appendices at the end of this thesis.

A model of the PTZ camera was developed and numerical results for the various parameters were obtained from a series of test sequences. The accuracy of this model was evaluated by reconstructing the various sequences of images into high resolution panoramas or mosaics. In the following chapter, this model and the block tracking methods presented in Chapter 4 will be tested on a variety of image sequences.

Chapter 7
Testing and Results

7.1 Introduction

This chapter presents the results obtained from implementations of the object tracking methods discussed in the previous chapters. It also draws on the camera calibration results discussed in Chapter 6.

A set of test sequences is introduced in Section 7.3 where they are divided into two categories: synthetic sequences rendered with a ray-tracer (Section 7.3.2) and real-world sequences captured with a Sony EVI-D31 PTZ camera (Section 7.3.3). Sample frames from each sequences, the hand-tracked ground truth and camera parameters are shown per sequence in Sections 7.3.4 to 7.3.11.

Three different techniques of tracking objects are presented: block tracking (Section 7.4), contour tracking (Section 7.5) and region tracking (Section 7.6). The implementation of these methods and results for each are discussed in the respective sections. All methods are implemented in C/C++ using the libraries discussed in Appendix D and run on a dual processor Intel based Windows XP system.

It is important to quickly refer to Section 7.2 which outlines several definitions of terms used throughout the chapter and more importantly the format of the tables in which tracking results are presented.

In summary and conclusion a comparison and discussion of the different tracking methods can be found in Section 7.7.

7.2 Definitions and interpreting tables

Throughout this chapter a standard table format will be used to present the tracking results from various methods. An example table is shown below:

	Position Filtered NCC		Best NCC		Method 1		Method 2	
No block updating	0.4*	(3)	4.4	(42)	3.5*	(21)	7.5	(42)
FIR block updating	2.0*	(15)	–		3.0†	(42)	3.2	(42)
Kalman updating	9.9	(42)	1.1	(42)	–		–	

† Track fails * Partial track

(See Section 7.2 pg 170 for explanation of table)

The values given are calculated by finding the average RMS error between the track position $\mathbf{t}(n)$ (in image pixel co-ordinates) and the ground truth $\mathbf{g}(n)$ (which is calculated for synthetic sequences or hand-tracked for real-world sequences) for N frames in the sequence:

$$\text{RMS pixel error} = \sqrt{\frac{1}{2N}\sum_{n=1}^{N}\|\mathbf{t}(n)-\mathbf{g}(n)\|^2} \tag{7.1}$$

The number of frames over which the RMS error was calculated is given in brackets. A small symbol next to the RMS error indicates if the track was partial (*) or unsuccessful (†). No symbol, indicates the track was successful and target was identified correctly in all frames.

In the case of a partial track e.g. 5.7* (35) should be interpreted as having an RMS pixel error of 5.7 for 35 frames of the sequence. After 35 frames, the algorithm fails to track the object and the RMS pixel error becomes large and meaningless. The purpose of quoting the error in this way is to show how well the algorithm was performing while it was tracking the object but still indicating that the target was not tracked for the entire sequence.

When the track fails the RMS value will be quoted as follows: 14.1† (60). This should be interpreted as having an RMS pixel error of 14.1 for 60 frames of the sequence. Although the tracking succeeded, the track has wandered to another portion of the target (for example, through accumulated mismatch errors) or tracked another object in the scene. By referring to the graphs that will be presented with these numerical results, it will be obvious where the algorithm began to track the incorrect target.

7.3 Test Sequences

Two sets of sequences were used to obtain results: a set of synthetic sequences where the absolute ground truth is known (by calculation) and a set of real world sequences taken with a PTZ camera with hand-tracked ground truth. The synthetic sequences were ray-traced in Povray at a resolution similar to that of commercially available PTZ cameras. Few illumination sources were used and most textures were matte to avoid ambiguities due to changes in illumination and reflection. Real world sequences were captured using a Sony EVI-D31 PTZ camera and an Imagenation PXC200A frame grabber. All the features of the Sony EVI-D31 can be controlled via an RS-232 serial link such as the pan and tilt motor positions. It is also possible to obtain the current pan, tilt and zoom positions from the camera stepper motors via the serial link. These were recorded with the image frames.

7.3.1 Overview

Table 7.1 below provides an overview of the various sequences used to test each of the tracking methods. Each of the sequences is described in sections 7.3.4 to 7.3.11 along with the reasons for presenting each sequence.

	NCC Block Tracker	Multiscale Method 1	Multiscale Method 2	Contour	Regions
Synthetic sequences					
`image1`	×	×	×		
`man1`	×	×	×		
Real world pan/tilt/zoom sequences					
`bottle1`	×	×	×	×	
`tom1`	×	×	×	×	
`tom3`	×	×	×	×	×
`tom4`	×	×	×	×	
Real world fixed view sequences					
`cyclist`	×			×	
`carpark`	×			×	

Table 7.1: *Overview of sequences and methods*

7.3.2 Synthetic Sequences

Synthetic sequences provide a good way of testing an algorithm under controlled conditions that are impossible to replicate in the real world without excessive cost and effort. Factors

such as extreme changes in illumination and poor frame rate can be eliminated. The movement of the camera can be controlled precisely and an analytical ground truth can be obtained by calculation (avoiding the uncertainty in a hand-tracked ground truth). It was decided to use the freeware ray-tracer, Povray (see Section D.1 for details), to render the synthetic sequences using a scene and camera model similar to that of the Sony EVI-D31 . Povray was chosen as it is freeware and open source allowing it to be rendered on several machines simultaneously to speed up the rendering process. It also uses a versatile scripting langauge which can perform matrix multiplications and point transforms outputting the results as text to produce ground truth and other information about camera position and zoom parameters. The model of the Sony EVI-D31 obtained from the camera calibration and the synthetic camera were matched to provide a consistent framework in which to perform tests. All synthetic sequences were rendered *without* radial distortion as this is removed in a pre-processing step in the real world sequences.

Two simple sequences were rendered as test sequences. The first, `image1`, is a high resolution image which has been texture mapped onto a plane and the camera zooms in while panning and tilting. The sequence provides a scene where more details is visible with increasing zoom while keeping all other factors such as illumination constant. The second sequence, `man1`, is intended to simulate the real sequences captured where the camera tracks a man walking while zooming in.

Thirty PC's running Linux were used to render the sequences and rendering times were typically 30 seconds to a few minutes per frame depending on the complexity. With varying machine loads, a complete sequence (with ground truth) could be rendered in a few minutes. It is also possible to use Povray in a closed loop tracking system where the camera is moved to maintain fixation on an object as discussed in Section 8.3. Each frame could be rendered before tracking to reflect changes in a virtual camera's pan, tilt and zoom position. Although this possibility exists it was never implemented.

7.3.3 Real World Sequences

A set of real world sequences was taken where a person was followed by the Sony EVI-D31 PTZ camera using the RS-232 serial link to return the current pan and tilt positions as well as the zoom of the camera. The sequences are of a person walking through an empty car park and are intended to represent a simple CCTV surveillance scenario. In a realistic system the

scene would contain many moving targets which could occlude each other and the illumination would vary. However, as a first investigation into the various tracking methods the sequences are adequate and the algorithms could be made more robust at a later stage to accommodate more real world problems than were considered.

The serial control of the camera (described in Appendix D.2) uses Sony's VISCA protocol running at 9600 bps to transfer commands to the camera and return information about the camera to the PC. The slow serial link speed and the protocol overhead means that a command to pan and tilt to a specific position is issued and reading back the camera position (once the movement stops) takes roughly 2 – 2.5 seconds. It was necessary to wait for the camera to come to a complete halt before reading the position back as the latency in the VISCA protocol results in the reported camera position and the true camera position being vastly different. Therefore, moving the camera along a path in a stepwise fashion is very slow. Hence it was only possible to achieve a frame rate of 0.4 fps where the position of the camera was accurate. Higher frame rates could be achieved, however the camera's position would be unpredictably uncertain. However, even at these extremely low frame rates the algorithm appears to track well and the synthetic sequences show the algorithm working under higher frame rate conditions that would be possible with alternative hardware.

Fixed view sequences were captured from a camera without the presence of pan, tilt or zoom for testing the contour tracker. The sequence `cyclist` was filmed with a hand-held Sony DV camcorder on a tripod and converted from MPEG video to individual image frames. `carpark` is footage from a CCTV camera recorded onto VHS tape and digitised using a Pinnacle DV-500 video editing card. As before, ground truth was produced by hand tracking each sequence using the application described in Appendix D.4.1.

7.3.4 Sequence – `image1`

The simplest of the synthetic scenes is a high resolution image of the V & A Waterfront in Cape Town that has been texture mapped onto a flat plane at the origin of the world co-ordinate system. A checkerboard pattern provides easy orientation for humans looking at the scene, but remains unused for tracking and could be removed without influencing results. Everything in the scene remains static except for the camera parameters. This allows for several sequences to be generated, such as zooming in and out along the camera axis or full pan, tilt and zoom movements. For example, a sequence can be created where there is only a change in scale by increasing the focal length while viewing an object on the camera axis. Other effects such as movement due to zooming can be achieved by moving the object off the camera axis. It is also possible to start the sequence from a high zoom factor and move to a lower zoom factor or vice versa. The high resolution image contains features common in real-world scenes and removes the artifacts created during zooming by an imperfect lens system, such as radial distortion and drift in the principal point.



Figure 7.1: *Sequence* `image1`

Only one of these scenarios (shown in Figures 7.1, 7.2 and 7.3) was chosen to present results with as it combines the most interesting camera parameter changes while still keeping the sequence simple. The sequence begins with an off-centre view of the plane from a wide angle. The camera pans and tilts towards the object while zooming until the object centre is on the camera axis. In the final frame the object is viewed at $10\times$ magnification.

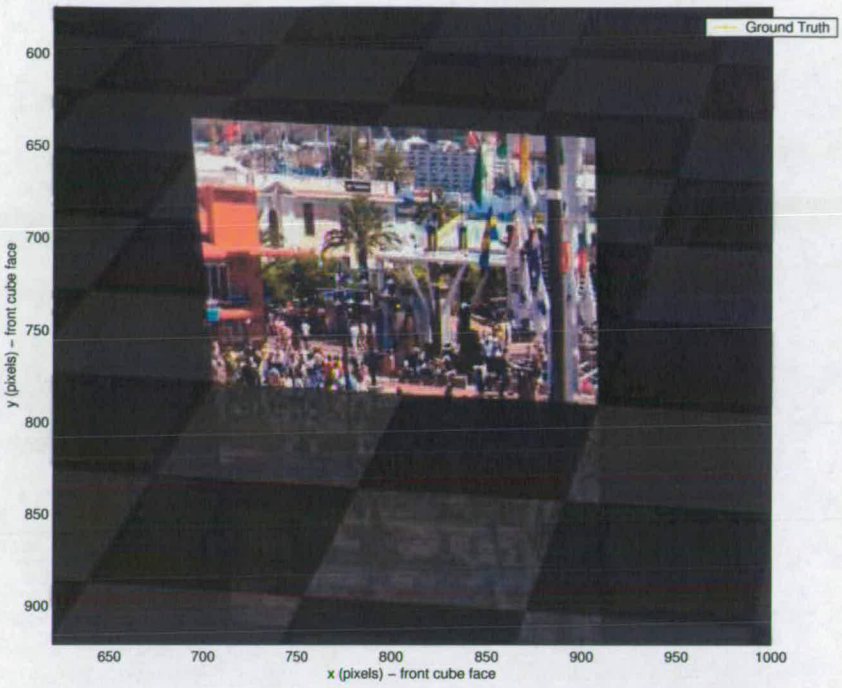


Figure 7.2: *Ground truth for sequence* **`image1`** *on composite image*

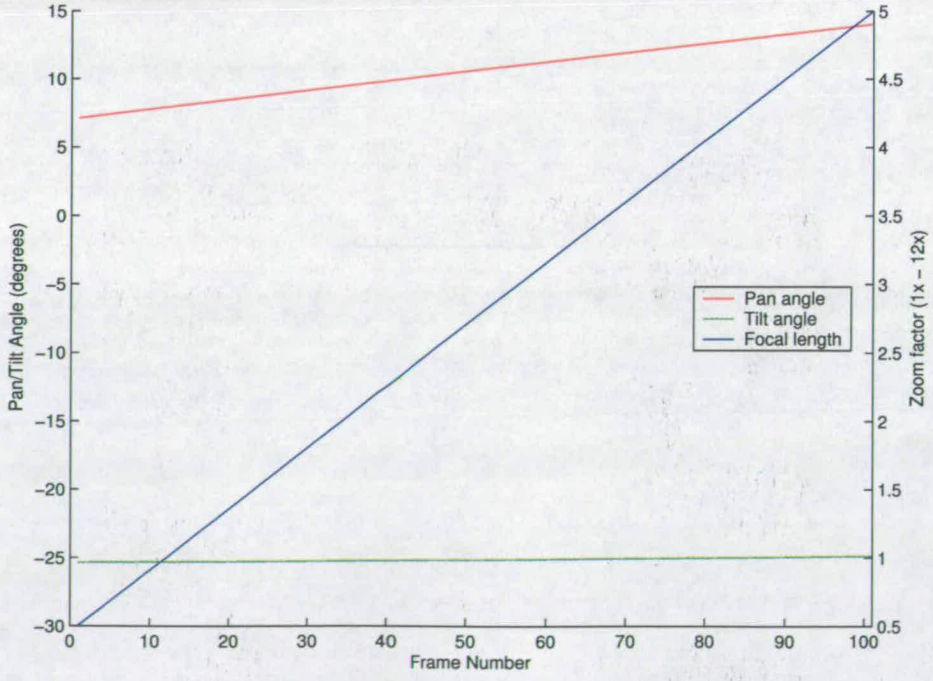


Figure 7.3: *Camera parameters for sequence* **`image1`**

7.3.5 Sequence – `man1`

A more complex scene was created where a man holding a briefcase moves in front of a complex background of trees. The man does not move along a straight path between the start and end points but rather moves along a curvy 'S' path so that he will change scale while moving. No occlusions were included as making block trackers robust to occlusions is a well explored problem and is beyond the scope of this work. The camera pans and tilts along a fixed path between the start and end points of the path the man follows. This results in the man not remaining in the centre of the view. The camera zooms in from a wide angle view of the entire scene to a close angle view with the man near the centre of the view. The scene is intended to be similar to the real-world carpark sequences captured with the Sony EVI-D31 camera and provide a scenario similar to that commonly found in CCTV surveillance applications. The sequence shown in Figures 7.4, 7.5 and 7.6 provides the following interesting effects: the target is moving, changes scale, does not stay in the centre of the frame and the camera pans, tilts and zooms. The target is never occluded and never leaves the frame.

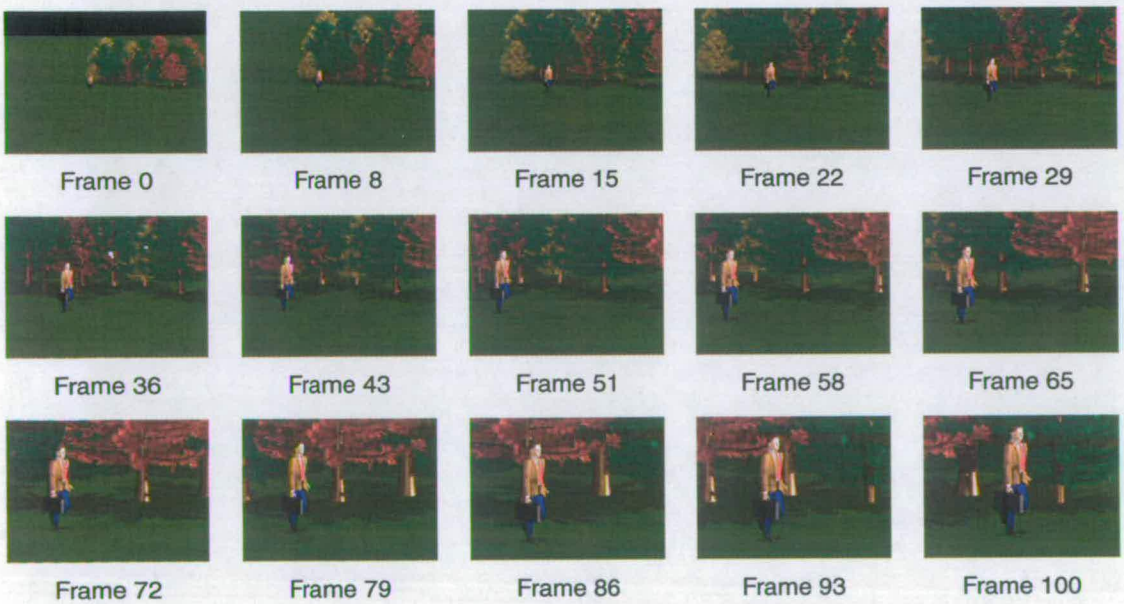


Figure 7.4: *Sequence* **`man1`**

As in the previous sequence (Section 7.3.4) many different scenarios can be created from this scene. This sequence was chosen as it provided similar conditions to the real-world sequences discussed later. Some obvious improvements, such as animating the man, could be made to make the scene more life-like – these are discussed in Section 8.3.

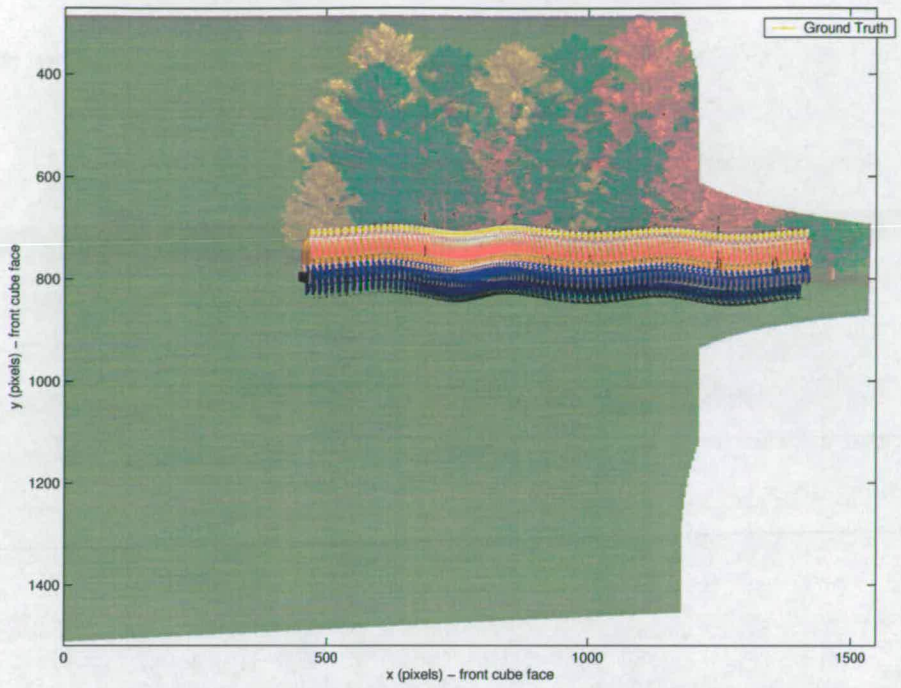


Figure 7.5: *Ground truth for sequence* **man1** *on composite image*

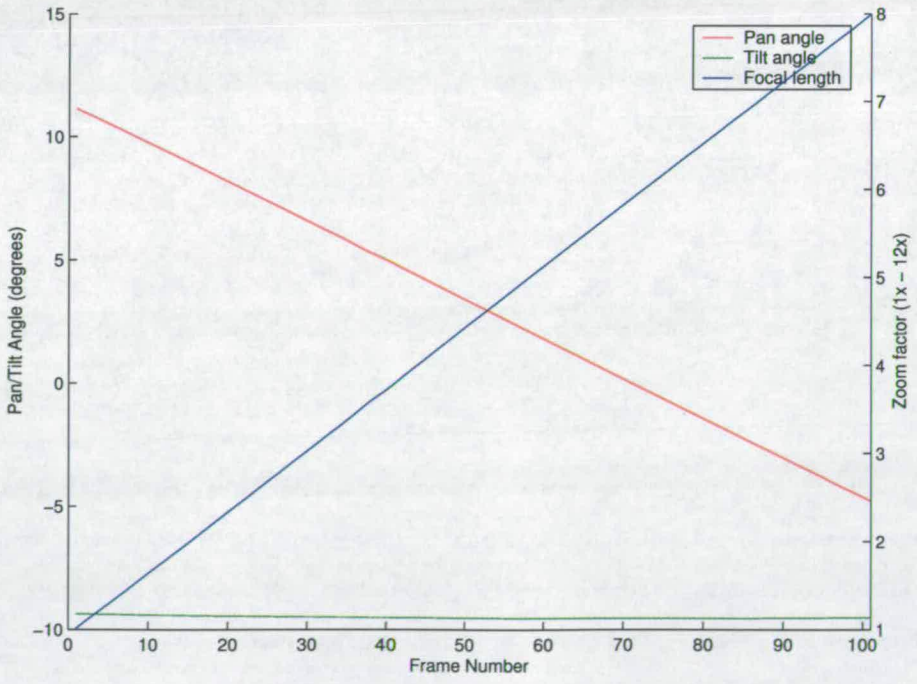


Figure 7.6: *Camera parameters for sequence* **man1**

7.3.6 Sequence – `bottle1`

In the first of the real-world sequences, a soft drink bottle was placed off centre in the most wide angle view (first frame) and the camera rotates and zooms so that it is in the centre of the frame in the most close angle view (last frame). This creates two interesting effects: firstly, the object increases in scale and more detail is visible with increasing zoom and secondly, there is apparent movement of the object as it is not located on the optical axis of the lens.

This sequence was created mainly as the object is stationary. This allows easy visual evaluation of the tracking algorithms such as the efficiency of the block updating algorithm with increasing detail. It also demonstrates the tracking possibilities of tracking static objects – many other zoom tracking algorithms require movement to segment objects. Selected frames from the sequence are shown below:

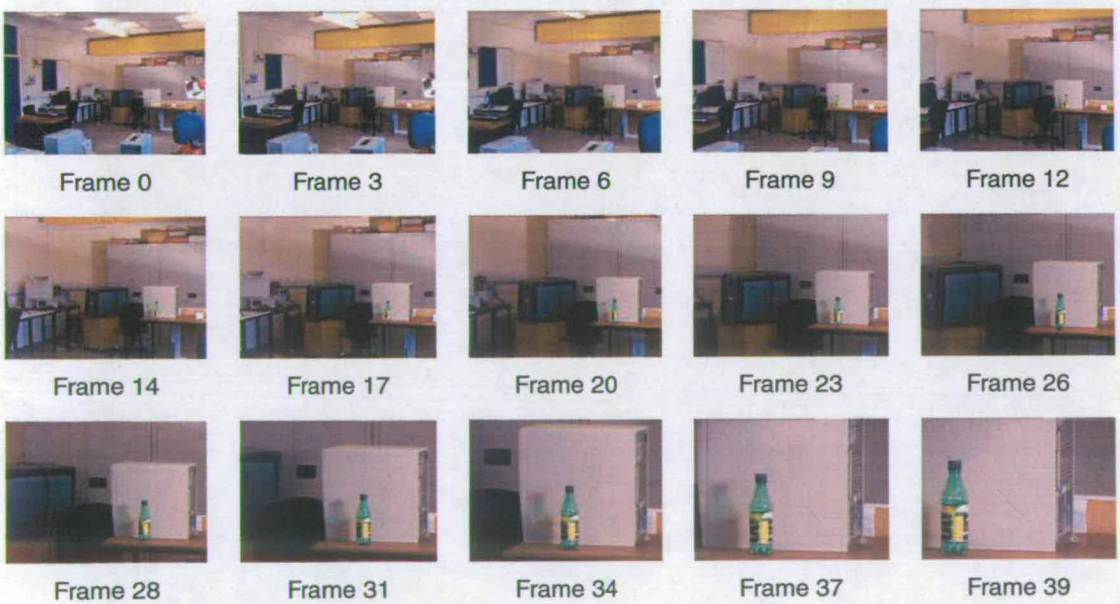


Figure 7.7: *Sequence* **`bottle1`**

A composite image of all the frames in the sequence and a hand-tracked ground truth are shown in Figure 7.8 below – all images in the sequence and the ground truth have been projected onto the front cube face. Figure 7.9 shows the PTZ camera parameters during the sequence with all angles in degrees. The zoom factor or magnification is calculated as the ratio of the focal length to the most wide angle focal length.

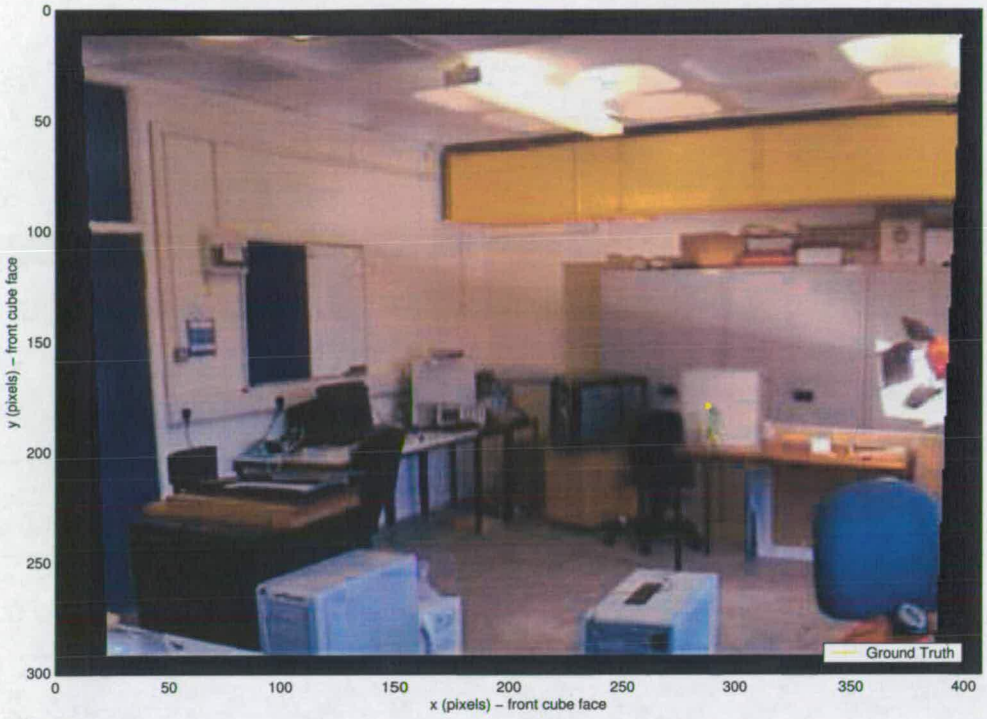


Figure 7.8: *Ground truth for sequence* **`bottle1`** *on composite image*

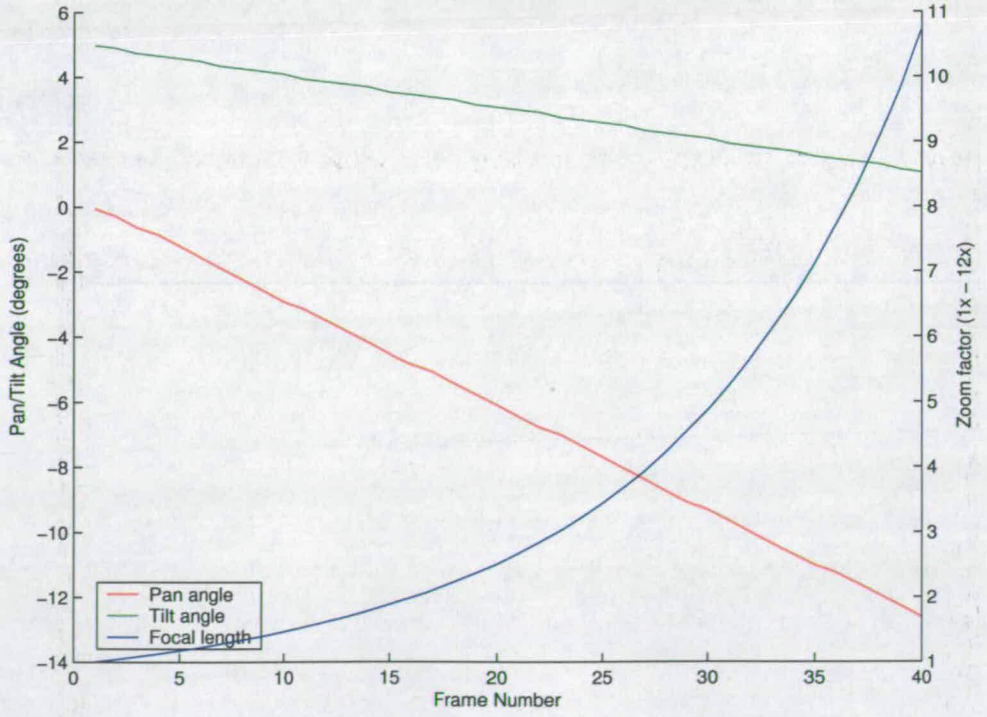


Figure 7.9: *Camera parameters for sequence* **`bottle1`**

7.3.7 Sequence – `tom1`

The sequence starts with a high angle view of a car park from the PTZ camera. A person is viewed (in a wide angle shot) walking across the car park and between some parked cars with increasing zoom, while keeping the person towards the centre of the frame. The zoom in the sequence varies from $1\times$ to approximately $10\times$.

This sequence would be a common scenario in CCTV surveillance where a person walks between cars in a public carpark. The person is partially occluded by the cars towards the middle of the sequence. Partial occlusion is often difficult to deal with in block tracking as a large portion of the block changes rapidly, however most of the block is still very similar to the target.

Some selected frames from the sequence are shown below:

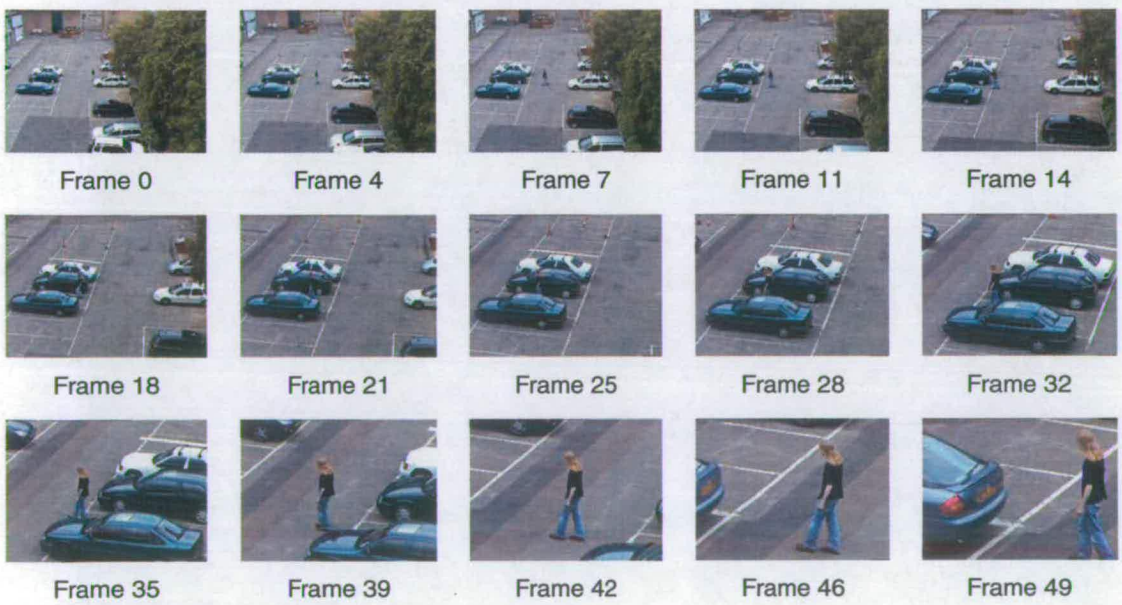


Figure 7.10: *Sequence* `tom1`

A composite image of all the frames in the sequence and a hand-tracked ground truth projected onto the front cube face are shown below in Figure 7.11. Figure 7.12 shows the PTZ camera parameters during the sequence with all angles in degrees. The zoom factor or magnification is calculated as the ratio of the focal length to the most wide angle focal length.



Figure 7.11: *Ground truth for sequence* `tom1` *on composite image*

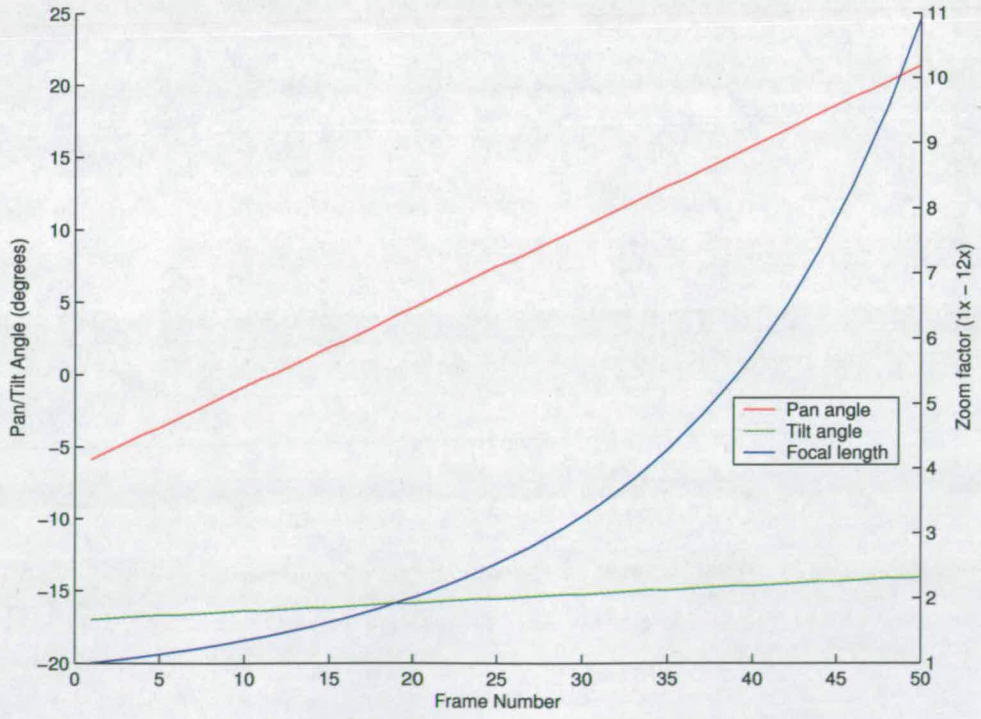


Figure 7.12: *Camera parameters for sequence* `tom1`

7.3.8 Sequence – tom3

In this sequence a person is viewed at a high zoom factor (approximately 9×) from a distance away and as the person moves towards the camera in a straight line the zoom level is decreased. Although, the full zoom range of the camera is not used, there is a large scale change. The camera only tilts to follow the target.

The most notable feature of this sequence is that the person remains at roughly the same scale throughout the sequence. The rate of movement in the scene is fairly constant although the target does not stay in the centre of the frame throughout the sequence. The person does not rotate much so the majority of the movement is in the arms and legs, while the head remains nearly stationary. There are no occlusions, the target remains in the frame at all times and the illumination in the scene remains approximately constant.

Selected frames from the sequence are shown below:

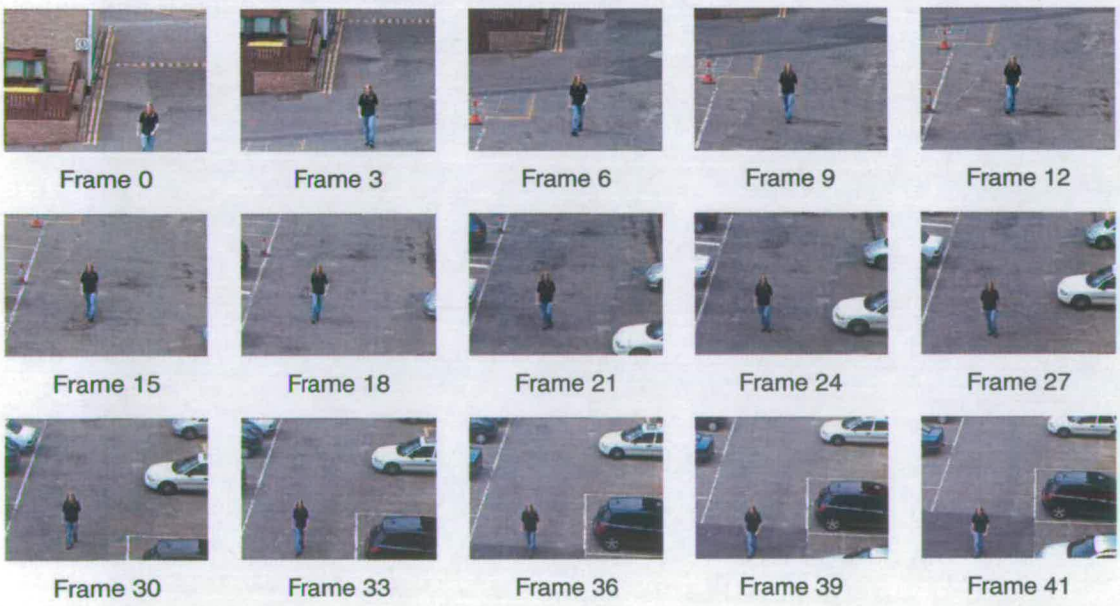


Figure 7.13: *Sequence* tom3

A composite image of all the frames in the sequence and a hand-tracked ground truth projected onto the front cube face are shown in Figure 7.14. Figure 7.15 shows the PTZ camera parameters during the sequence with all angles in degrees.

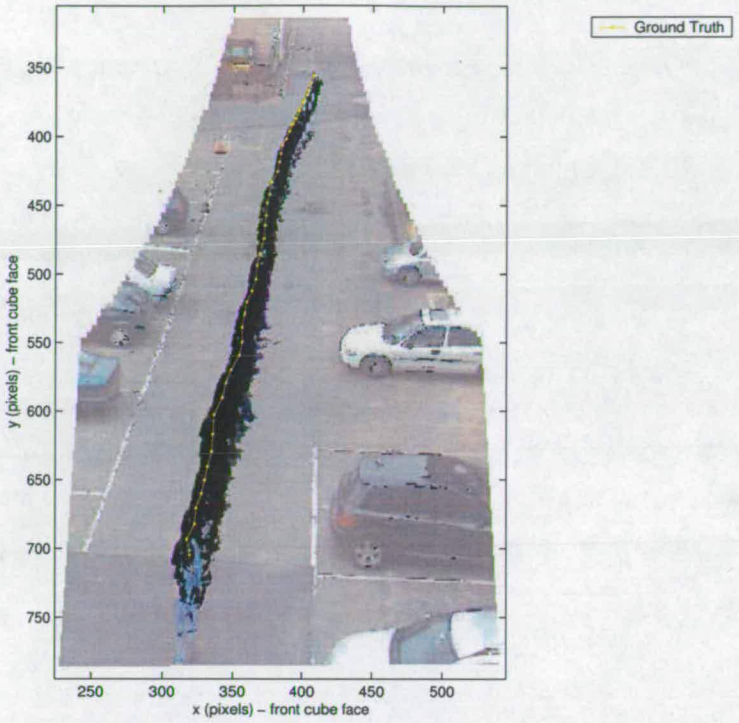


Figure 7.14: *Ground truth for sequence* `tom3` *on composite image*

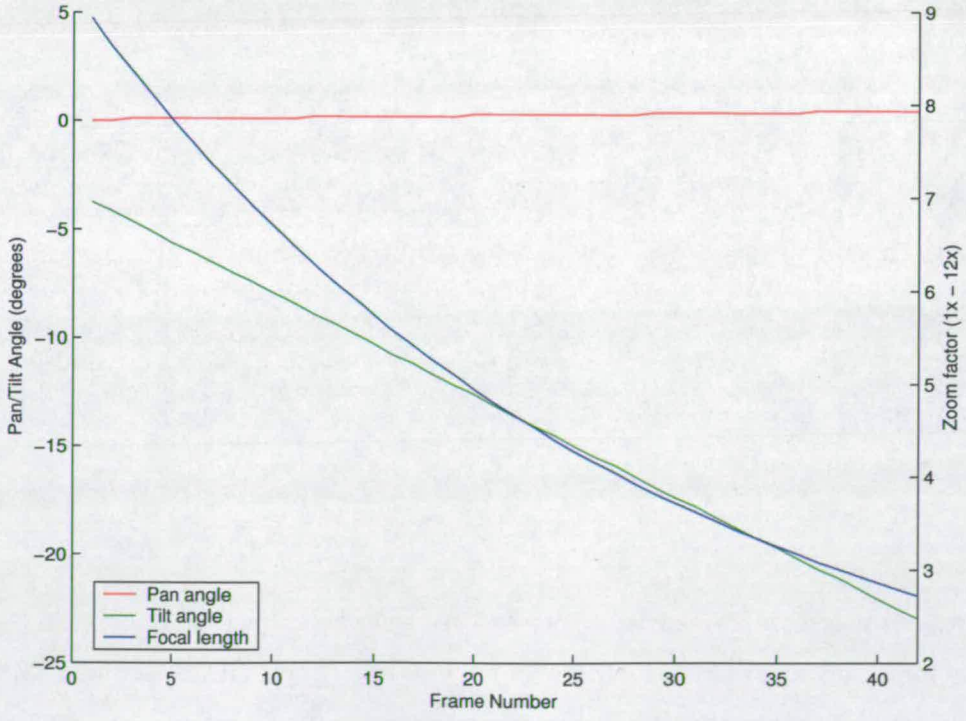


Figure 7.15: *Camera parameters for sequence* `tom3`

7.3.9 Sequence – `tom4`

This sequence is the reverse of the sequence `tom3`: a person is followed from a wide angle shot to a close angle shot while walking away from the camera. Almost the entire zoom range of the camera is used. The camera pans and tilts keeping the person in the camera view at all times. As the person moves at a different rate to the camera there is significant movement of the target around the frame. As before, there are no occlusions and the scene illumination is roughly constant. The global motion of the target is fairly erratic with the target almost coming to a complete halt towards the middle of the sequence.

One the most interesting features of this sequence is that the target changes scale significantly during the sequence as one might expect when zooming in on a target. This makes the sequence difficult to track by region tracking using interframe motion differences and by conventional block tracking, as the size of the object changes vastly.

Selected frames from the sequence are shown below:

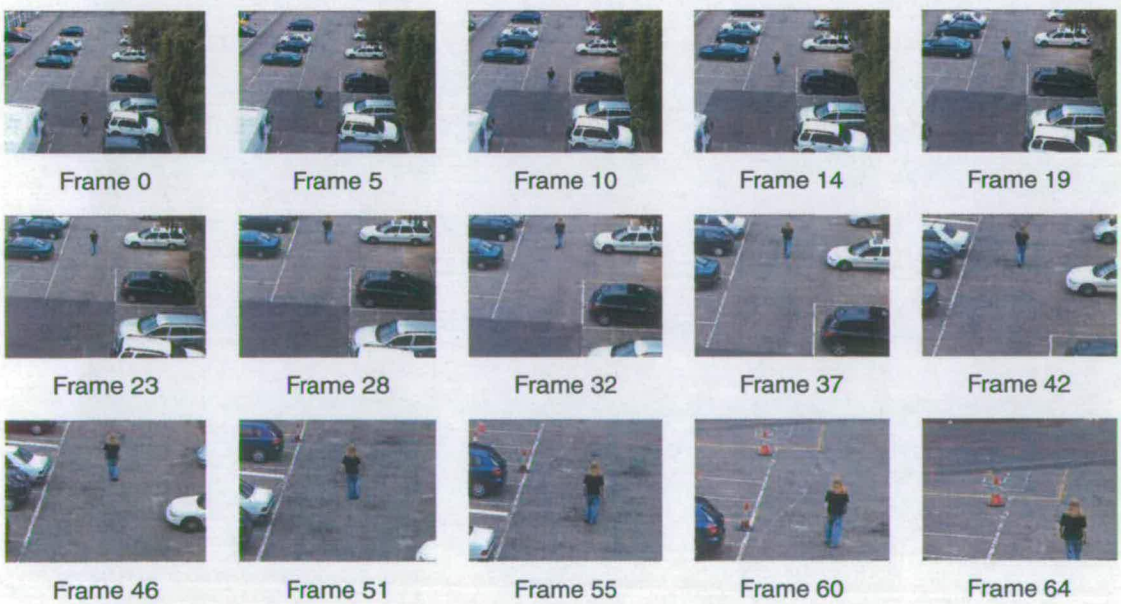


Figure 7.16: *Sequence* `tom4`

A composite image of all the frames in the sequence and a hand-tracked ground truth projected onto the front cube face are shown below in Figure 7.17. Figure 7.18 shows the PTZ camera parameters during the sequence with all angles in degrees.

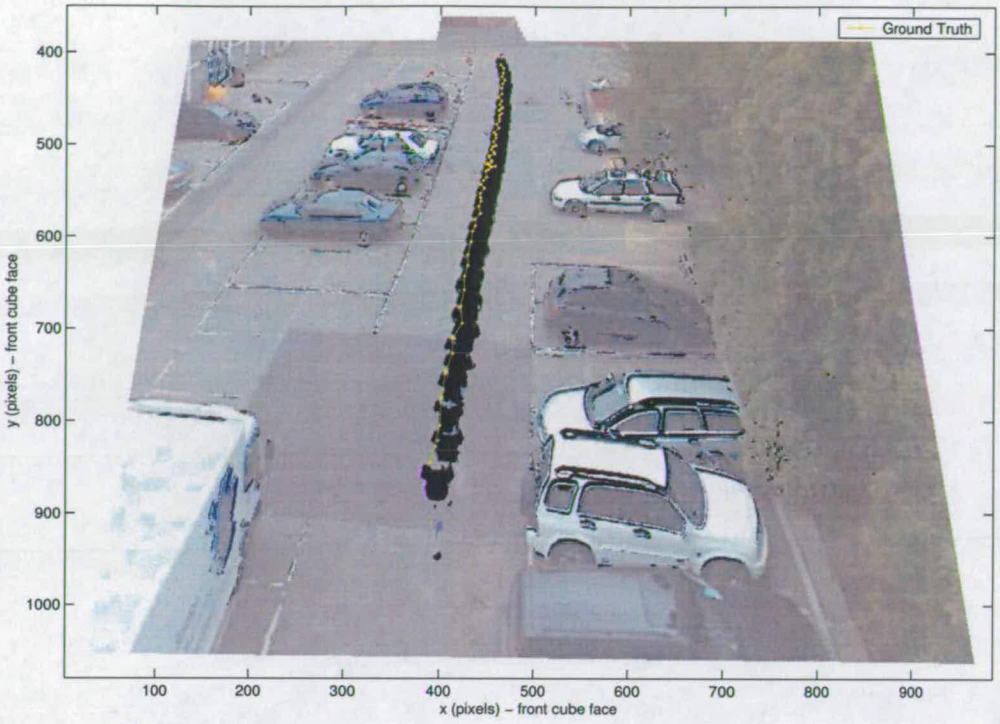


Figure 7.17: *Ground truth for sequence* `tom4` *on composite image*

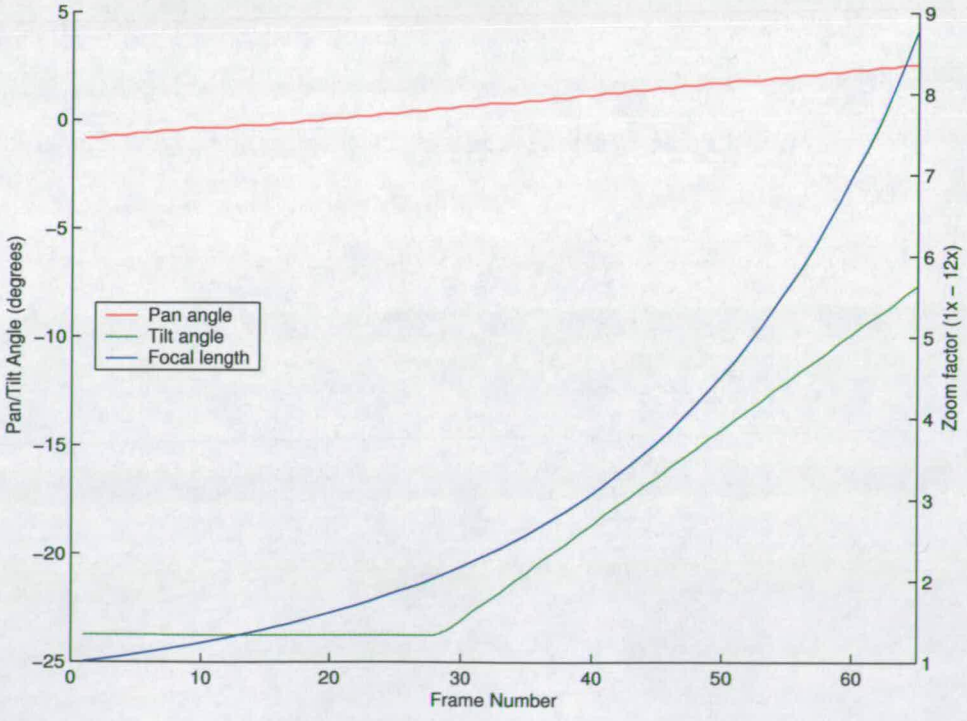


Figure 7.18: *Camera parameters for sequence* `tom4`

7.3.10 Sequence – `cyclist`

In this scene there is a fixed camera viewing a cyclist moving past parked cars in a carpark. Figure 7.21 shows selected frames from the sequence and a composite of four frames is shown in Figure 7.19.



Figure 7.19: *Four frame composite of sequence* **`cyclist`**

7.3.11 Sequence – `carpark`

CCTV footage of a carpark was digitised from VHS video and selected frames from this were used to create **`carpark`**. A car moves from the right side of the image to the left as shown in Figure 7.22 shows selected frames from the sequence and a composite of four frames is shown in Figure 7.20.



Figure 7.20: *Four frame composite of sequence* **`carpark`**

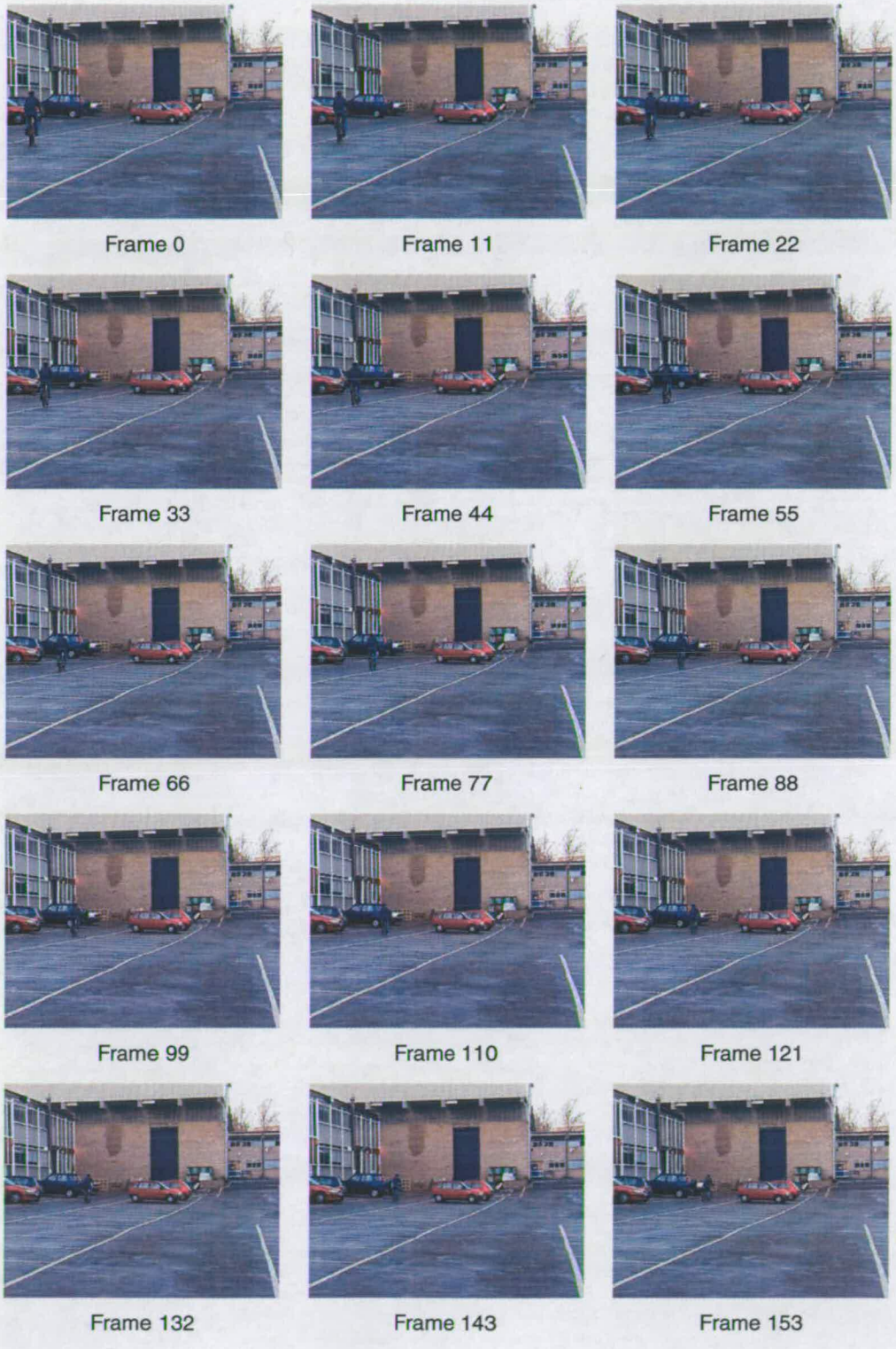


Figure 7.21: *Sequence* `cyclist`

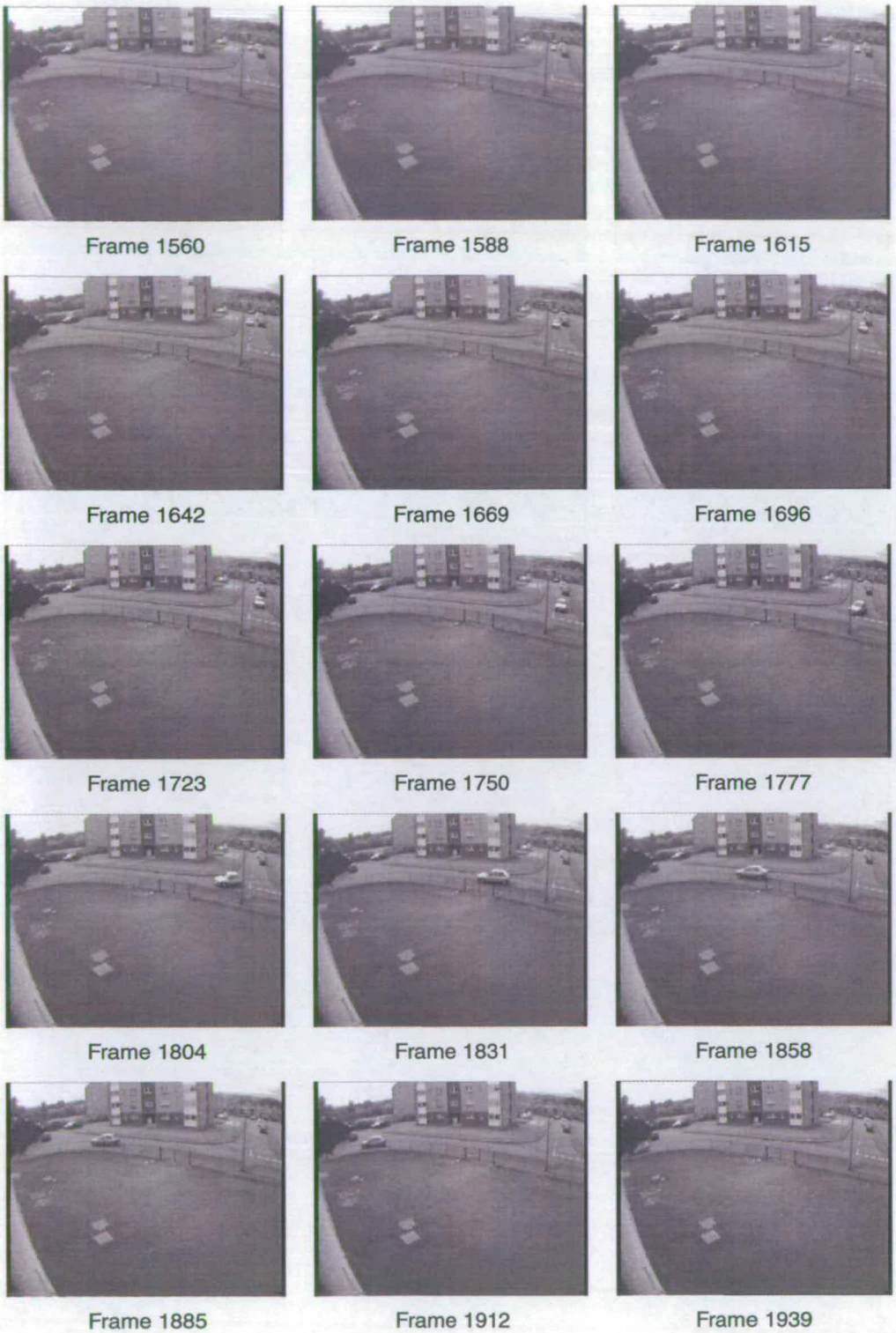


Figure 7.22: *Sequence* **carpark**

7.4 Block Tracking

Several types of block trackers were implemented for comparison to the multiscale block tracking methods discussed in Chapter 4. This section outlines the basic implementations and the following few sections contain exact details of the tracker parameters for each sequence. Four broad types of block trackers were implemented for tracking objects in synthetic and real-world sequences:

NCC Tracker	conventional block tracker with simple position filtering
NCC Best Tracker	conventional block tracker aware of camera pan, tilt and zoom position
Multiscale Method 1	multiscale block tracker where search area is warped into the reference view and tracked
Multiscale Method 2	multiscale block tracker where the reference block is warped into current view and sub-pixel tracked

The conventional block trackers (Section 4.2) use normalised cross-correlation block matching with a Kalman filter (Section 4.2.6) to estimate the position search region in image co-ordinates. Three different types of reference block updating (Section 4.2.7) are implemented: no updating, FIR updating and Kalman updating. Section 7.4.1 discusses the implementation in more detail. This implementation of a block tracker is referred to in the results as the *NCC Tracker*.

An improvement to the conventional block tracker is to use the camera pan, tilt and zoom information to predict the position of the search region in world co-ordinates. This is done by predicting the world co-ordinate position of the object, projecting this into the current view and searching around that point. The tracker was implemented with and without Kalman block updating. This implementation of a block tracker is referred to in the results as the *NCC Best Tracker*.

Two implementations, discussed in Section 7.4.2, of the multiscale block trackers are implemented with and without reference block updating. Both methods use a Kalman filter to predict the position of the target in world co-ordinates and update the block only if the match value is high. The implementations of the multiscale block trackers are referred to in the results as *Multiscale Method 1* and *Multiscale Method 2* (or sometimes just *Method 1* and *Method 2*).

All implementations of the block trackers reject matches that are too far from the predicted position and look for the next best match that is closer to the predicted position, only if the distance is too large and the match value is below a threshold.

7.4.1 NCC Block Trackers

All block trackers were initialised with a 9×9 pixel reference block created from the ground truth position in the first frame of the sequence. The reference block remains the same size throughout the track. All track positions refer to the centre of the reference block in image co-ordinates.

A constant velocity Kalman filter (as described in Section 4.2.6) was implemented to estimate the object position in each view. For the conventional block tracker the observation vector was the last match position in image co-ordinates. The position and velocity noise parameters for the Kalman filter were 0.01 pixels and 0.001 pixels/frame respectively. The measurement noise was set to 2 pixels. This allows for bad measurements, but forces the model to predict small variations in velocity, rejecting large variations as outliers.

For the improved or *Best NCC Tracker*, the Kalman filter predicts the position of the object in world co-ordinates and uses the camera parameters to transform the object position into image co-ordinates. The observation vector is the last match position in world co-ordinates. The position and velocity noise parameters were set to 1.0 pixels and 0.1 pixels/frame respectively. Again, the measurement noise was set to 2 pixels. The model parameters are larger to allow for errors in the camera model (and hence the transformations between world and image co-ordinates). Assuming the camera model to be accurate to less than one pixel would be unrealistic in general.

The effect of updating the reference block was investigated by allowing three modes of operation: no block updating, a simple FIR block update and Kalman block updating. The FIR update approach adds the current frame to the average after scaling it by a factor α as described in Section 4.2.7. For all FIR block updates α was set to 0.15 i.e. 15% of the current frame was added to the average.

The reference block was also updated using a Kalman filter by unwrapping the pixels in the block into a continuous vector and filtering each pixel intensity as a single time varying signal as described in Section 4.2.7. The process noise for each pixel was set to 20 and the measurement

noise to 5. This allows measurement errors due to aliasing effects and constrains changes in intensity to around 10%. The spatial dependence of the pixels on each other can be determined from the Kalman posteriori co-variance matrix. The match block is created from the current estimate of the pixel intensities. Kalman filtering on blocks larger then 20×20 pixels becomes computationally slow as the covariance matrix becomes a dense matrix with more than 160 000 elements, compared to the 6 561 elements for 9×9 pixel reference block.

For the conventional block tracker a search region of 30 pixels was used for all real-world sequences except **bottle1** where a value of 10 pixels was used. The improved or best conventional tracker used a search area of 60 pixels for all real-world sequences and 20 pixels for **bottle1**.

All of the parameters discussed above are dependent on the rates of pan, tilt and zoom. High rates of change result in large changes in target positions and appearance. The values above have been chosen to best suit the sequences tested. Given other sequence these values would need to be varied until good results are obtained. It is difficult to predict these values without *a priori* knowledge of the sequence. This is a weakness of all block trackers – creating a robust tracker in one circumstance doesn't guarantee robustness for all sequences.

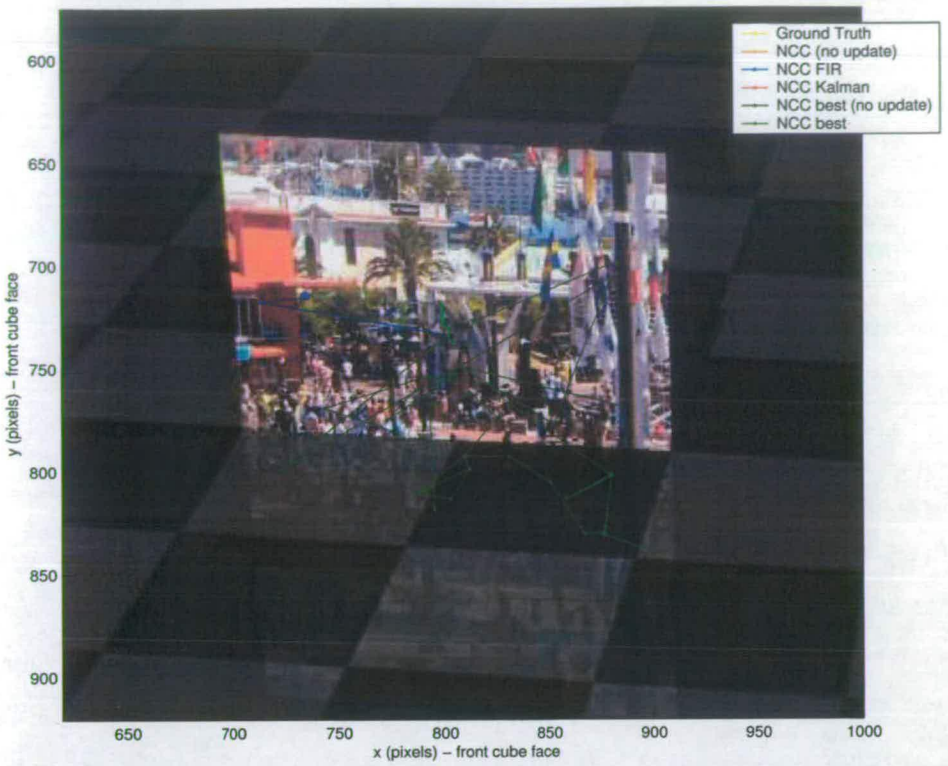
The results from tracking all of the sequences using the NCC block trackers is summarised in Table 7.2. Full results can be found in Appendix A where results are listed for each sequence with full size graphs. A discussion of these follows later in the chapter.

Sequence	NCC Block Trackers									
	No block update		*FIR block update*		*Kalman block update*		*No update & position filtering*		*Kalman block update & position filtering*	
image1	0.01*	(2)	132.9†	(101)	32.8†	(101)	0.01*	(2)	32.8†	(101)
man1	0.08*	(8)	11.1†	(101)	13.3†	(101)	0.1*	(8)	13.3†	(101)
bottle1	36.8*	(24)	46.9†	(40)	38.3†	(40)	57.8†	(40)	9.3†	(40)
tom1	2.6*	(20)	2.5*	(41)	1.6*	(18)	2.6*	(20)	1.6*	(16)
tom3	0.4*	(3)	2.0*	(15)	9.9	(42)	4.4	(42)	1.1	(42)
tom4	21.3†	(65)	20.6†	(65)	19.7†	(65)	8.5†	(65)	4.7	(65)

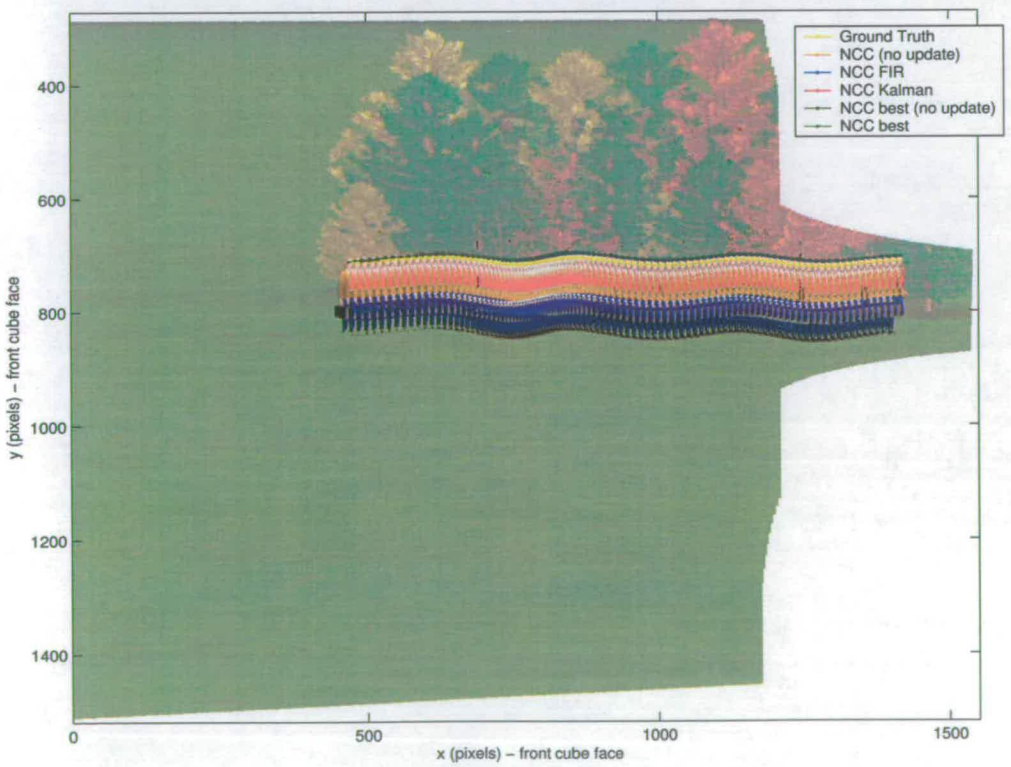
† Track fails * Partial track

(See Section 7.2 pg 170 for explanation of table)

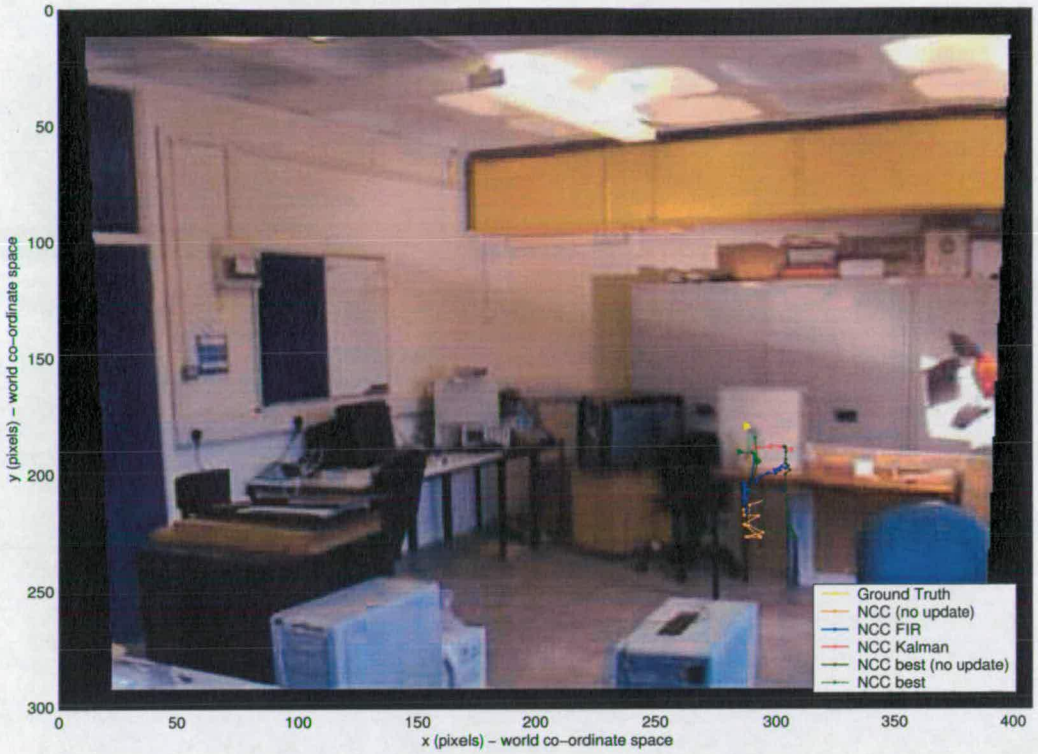
Table 7.2: *RMS pixel errors for all conventional NCC trackers (all units are image pixel co-ordinates)*



(a) Sequence **`image1`**



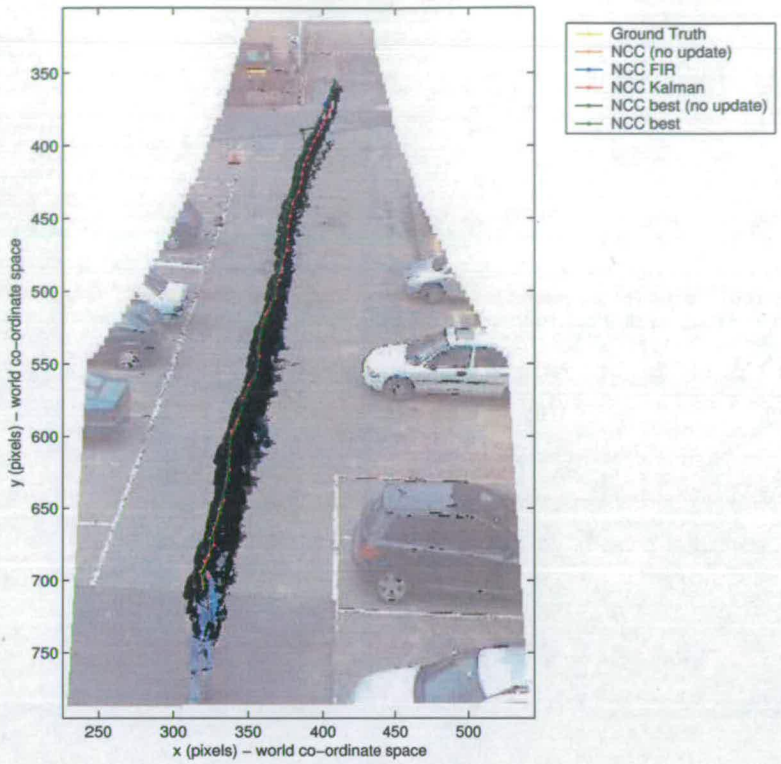
(b) Sequence **`man1`**



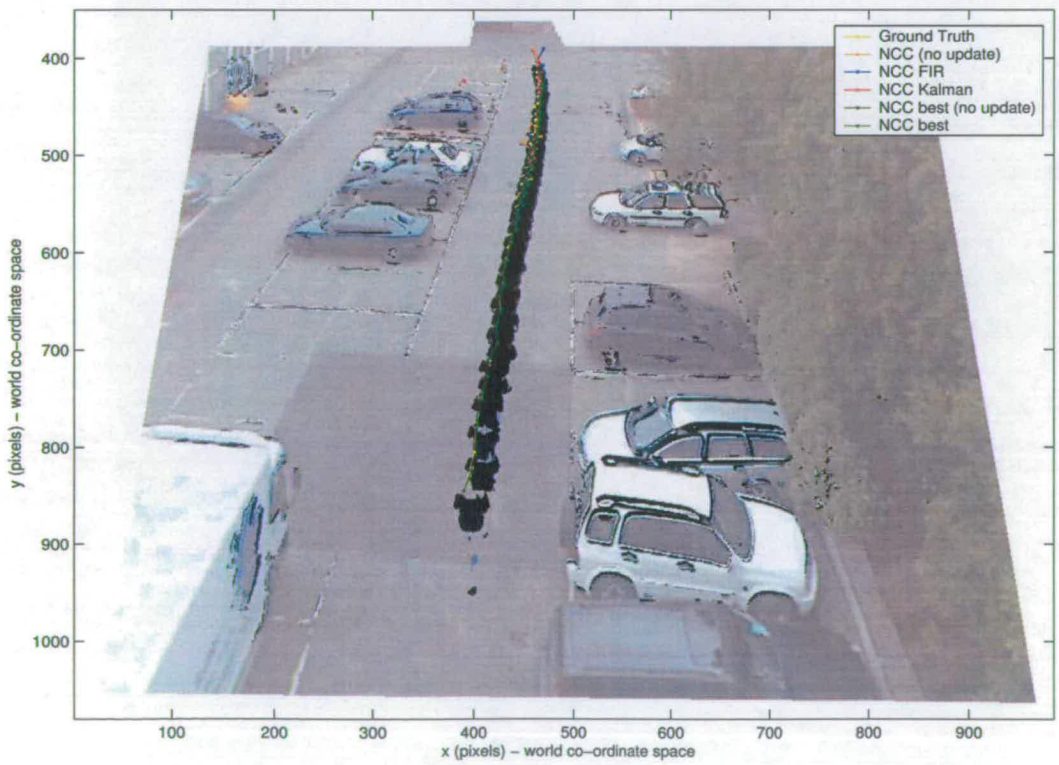
(c) Sequence **bottle1**



(d) Sequence **tom1**



(e) Sequence **tom3**



(f) Sequence **tom4**

Figure 7.23: *Tracks for NCC trackers*

7.4.2 Multiscale Trackers

As with the conventional block trackers, the start position was obtained from the ground truth. A reference block was created from the first frame at this position. Hence, the first frame tracked is the second frame in the sequence.

Two multiscale tracker methods were implemented: first, where the search area is warped into the reference view and secondly, the reference block is warped into the current view and matched normally. Method 1 is slower, but more accurate and uses more memory. Method 2 is faster and uses less memory, however tends to be less accurate and is more susceptible to accumulated mismatch error (the reference block slowly wanders off the target).

As with the conventional block trackers describe above, a Kalman filter was used to predict the position of the object in world co-ordinates in each frame assuming constant velocity. The observation vector is the last match position in world co-ordinates. The position and velocity noise parameters were set to 1.0 pixels and 0.1 pixels/frame respectively. The measurement noise was set to 2 pixels. The estimated position of the target in the current frame is returned by the Kalman filter in world co-ordinates.

False matches were rejected by limiting the correct match position to be within a certain distance of the predicted Kalman value if it was below as certain match threshold. These thresholds and the size of search area are listed for each sequence in Table 7.3.

Sequence	*Reference Block Size*		*Search Region*	*Match Threshold*	*Distance Threshold*	*Update Threshold*
	Method 1 (pixels)	*Method 2 (pixels)*	*(pixels)*		*(pixels)*	
`image1`	68	90	15	0.8	9.0	0.89
`man1`	34	72	15	0.8	9.0	0.89
`bottle1`	68	97	5	0.9	5.0	0.89
`tom1`	18	98	25	0.8	9.0	0.81
`tom3`	18	29	35	0.8	9.0	0.89
`tom4`	18	78	35	0.8	9.0	0.89

Table 7.3: *Parameters for multiscale trackers*

It is not possible to set a constant size of the reference block in the multiscale trackers for all sequences, as the reference view is dependent on the sequence. Reference block sizes are also

chosen so that they will be at least a certain size at the minimum zoom factor. If a reference block is created from a view where the zoom factor is say $2\times$, when warped to a view where the zoom factor is less, the block will be scaled down. Hence, to ensure the reference block is always at least a 9×9 pixel block it is necessary to create a 18×18 pixel reference block if the reference view has a zoom factor of $2\times$.

More often than not, the reference view was fixed to zero pan and tilt with the zoom factor set to $7\times$. This provided a good compromise for sequences with a large zoom range, so that the reference block at wide angle would not be overly interpolated and would contain good levels of detail at high zoom factors.

Two modes of block updating operation were implemented: no block updating and a simple FIR type update. Kalman block updating was not implemented, as often the reference block size is 70 - 90 pixels. A 90×90 pixel reference block would need a 65 million element covariance matrix, which would require 500 megabytes of memory to store. The Kalman filter uses the covariance matrix several times, so the computational time of estimating the reference block is too long to be useful. Therefore, only a simple FIR approach, as described in the conventional block trackers above, was implemented with a value of 0.15 used for α.

Table 7.4 summarises all of the results obtained from tracking the sequences using the various multiscale tracking methods. Full results can be found in Appendix A with full size graphs and tables of results for each sequence.

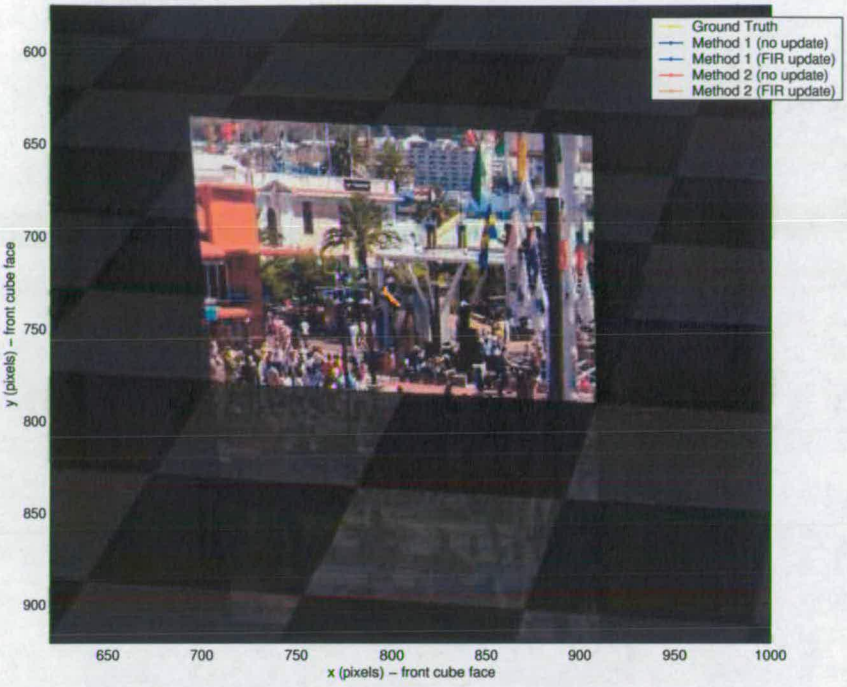
	Multiscale Method 1				Multiscale Method 2			
Sequence	*No update*		*FIR update*		*No update*		*FIR update*	
`image1`	2.9	(101)	3.6	(101)	3.5	(101)	18.1†	(101)
`man1`	0.1*	(15)	0.9	(101)	2.3	(101)	15.3†	(101)
`bottle1`	3.1	(40)	3.3	(40)	1.3	(40)	6.7	(40)
`tom1`	2.1*	(16)	1.7*	(16)	1.0*	(25)	2.3*	(27)
`tom3`	3.5*	(21)	3.0†	(42)	7.5	(42)	3.2	(42)
`tom4`	9.3*	(47)	8.3*	(65)	1.5	(39)	4.0	(65)

† Track fails * Partial track

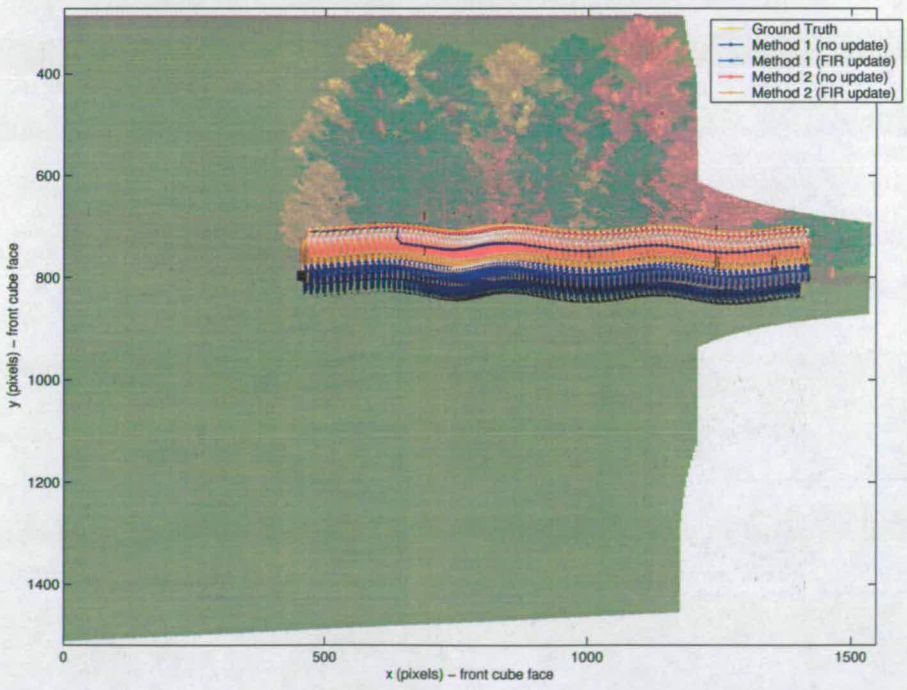
(See Section 7.2 pg 170 for explanation of table)

Table 7.4: *RMS pixel errors for all multiscale trackers sequences (all units are image pixel co-ordinates)*

A comparison of the various tracking methods and their suitability to the different sequences is discussed in Section 7.7. A discussion of these results can be found in Section 7.4.3 on page 204.

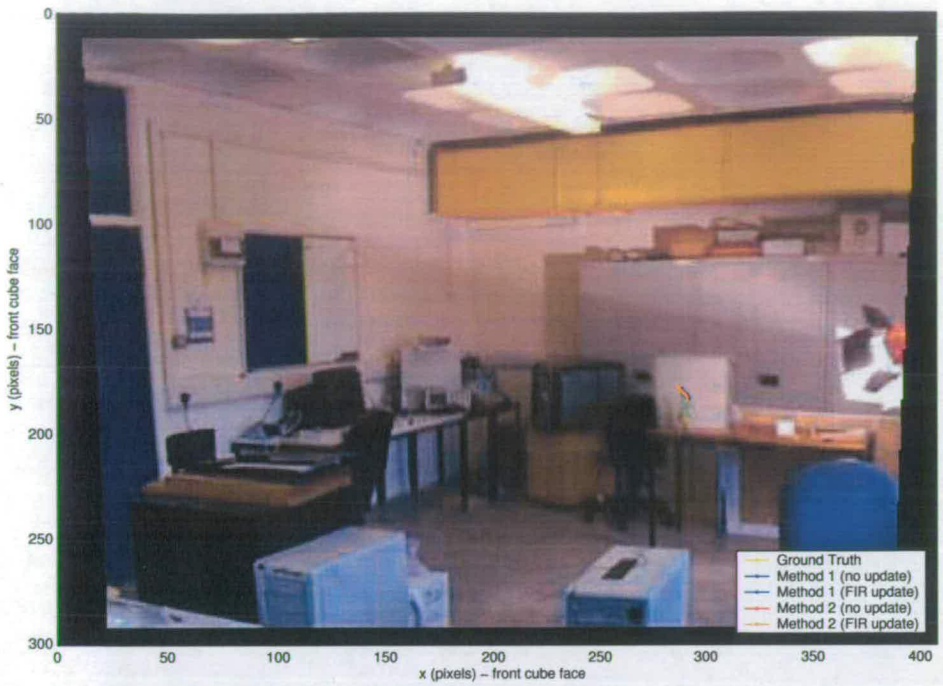


(a) Sequence **`image1`**



(b) Sequence **`man1`**

Figure 7.24: *Composite tracks for multiscale tracker*

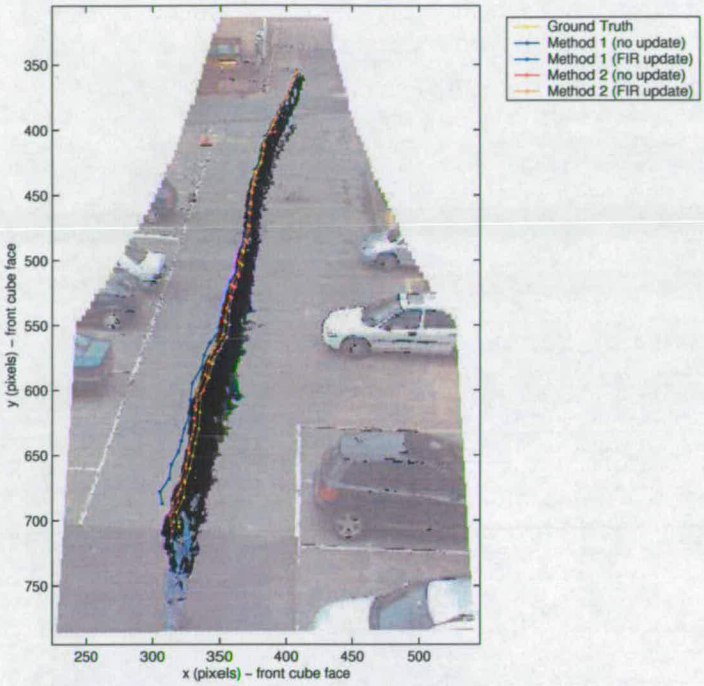


(a) Sequence `bottle1`



(b) Sequence `tom1`

Figure 7.25: *Composite tracks for multiscale tracker*

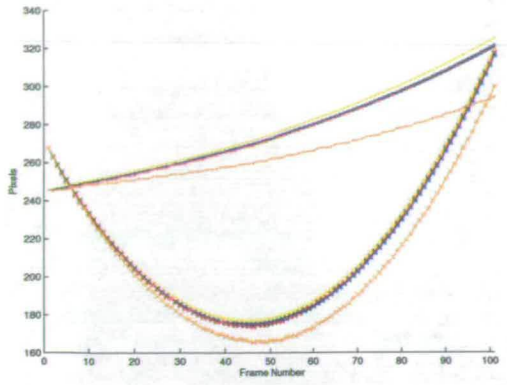


(a) Sequence **tom3**

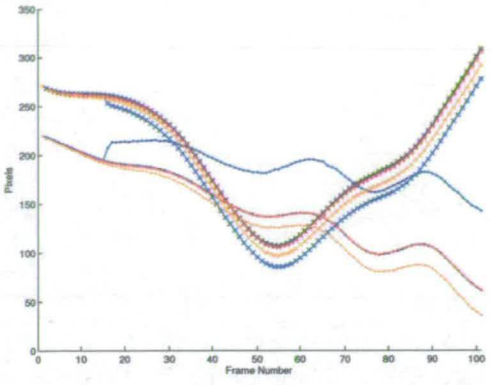


(b) Sequence **tom4**

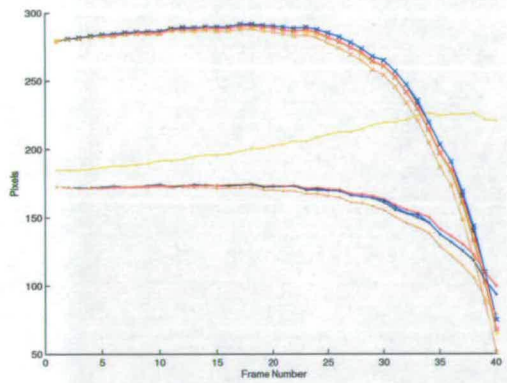
Figure 7.26: *Composite tracks for multiscale tracker*



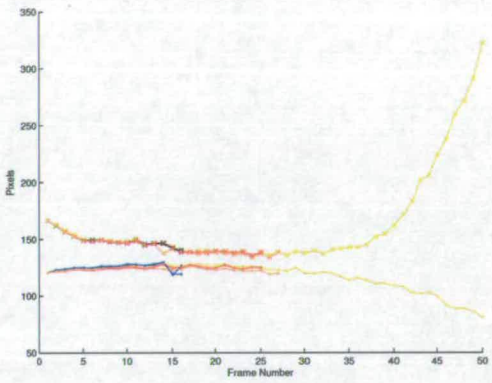
(a) Sequence **image1**



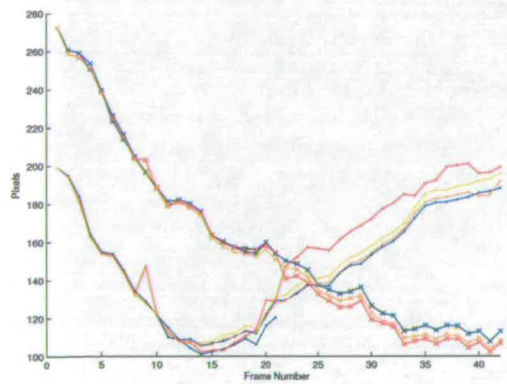
(b) Sequence **man1**



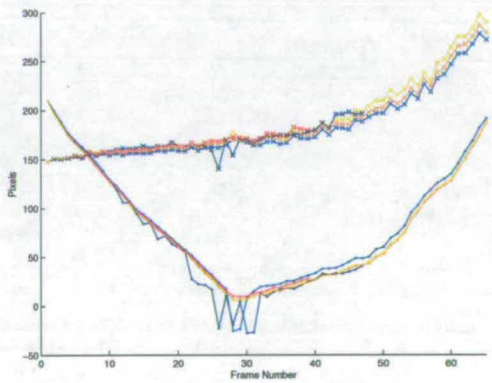
(c) Sequence **bottle1**



(d) Sequence **tom1**



(e) Sequence **tom3**



(f) Sequence **tom4**

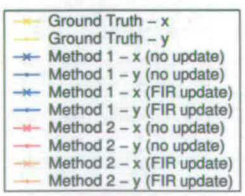
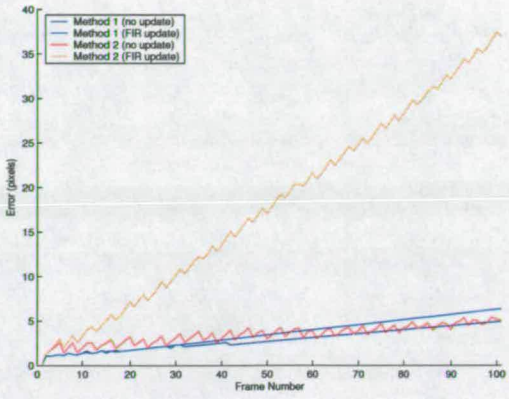
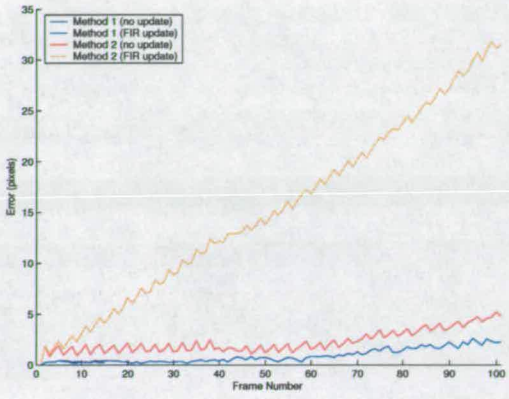


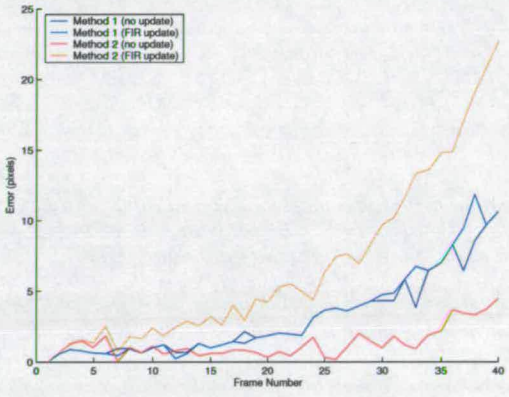
Figure 7.27: *Tracks for multiscale tracker*



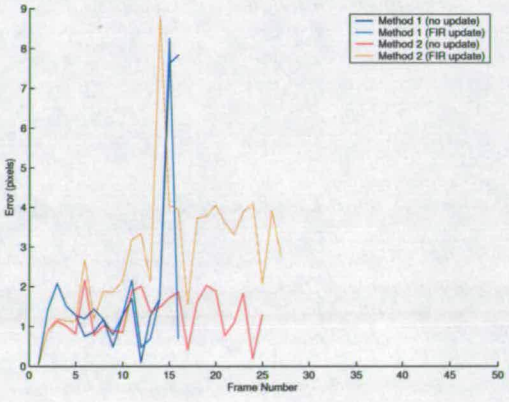
(a) Sequence **image1**



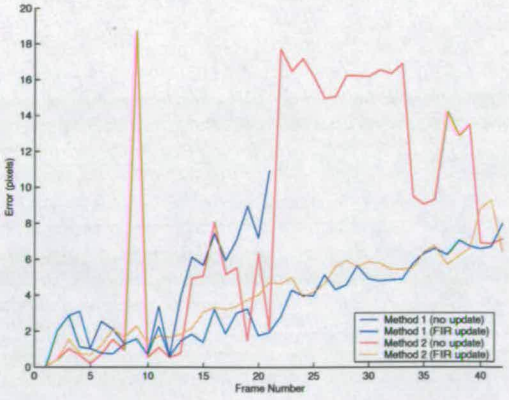
(b) Sequence **man1**



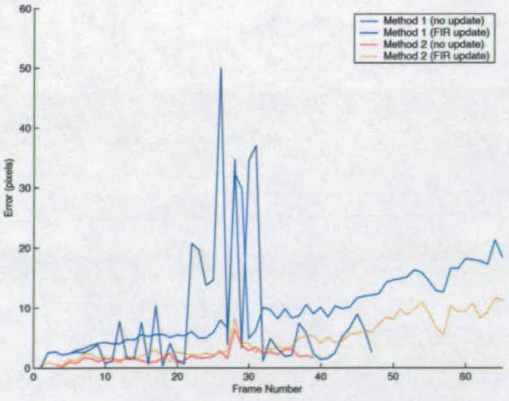
(c) Sequence **bottle1**



(d) Sequence **tom1**



(e) Sequence **tom3**

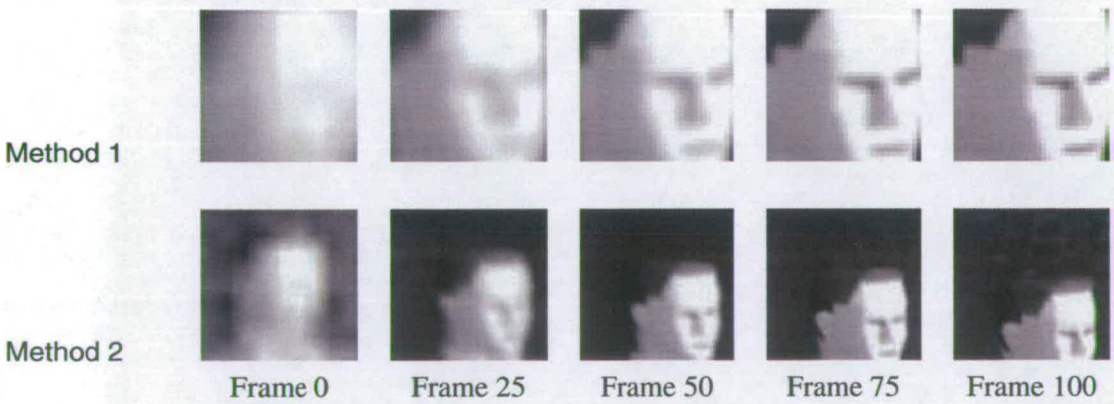


(f) Sequence **tom4**

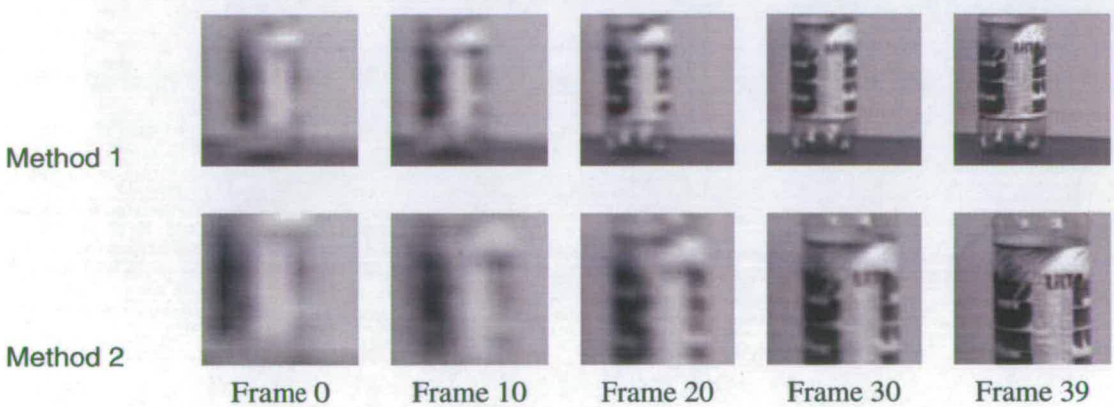
Figure 7.28: *Errors for multiscale tracker*



(a) Sequence **image1**

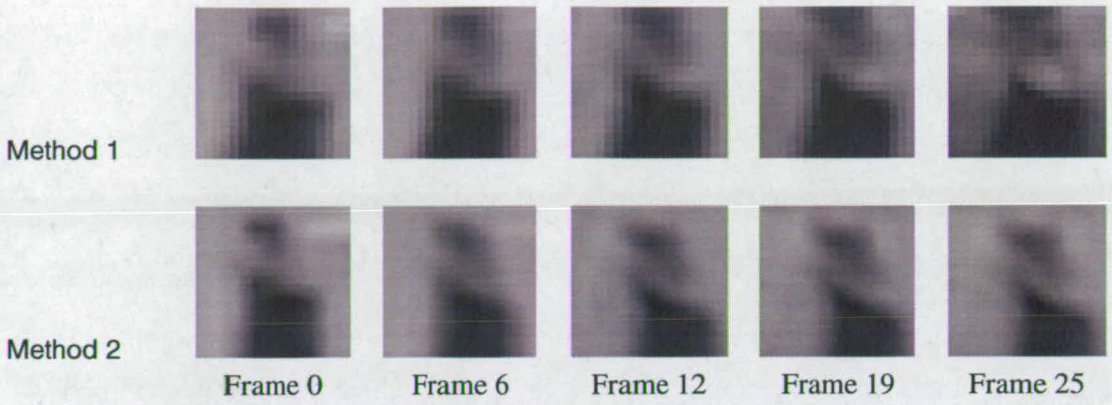


(b) Sequence **man1**

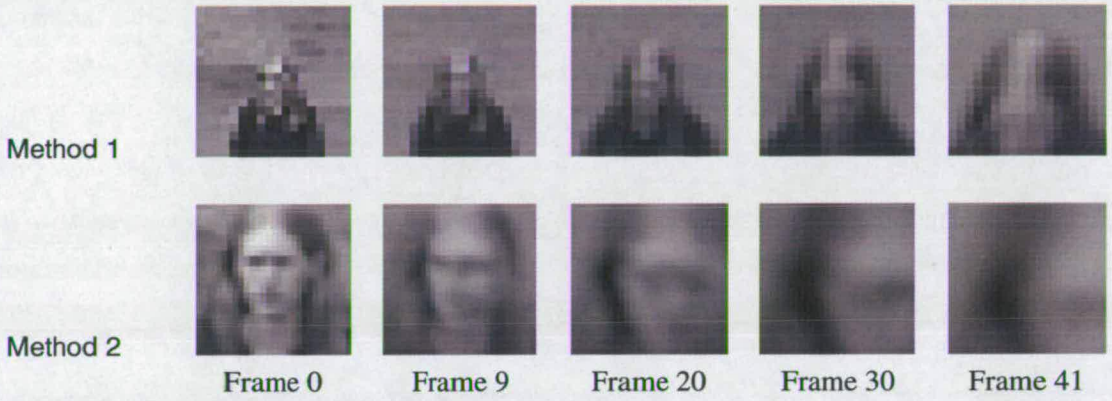


(c) Sequence **bottle1**

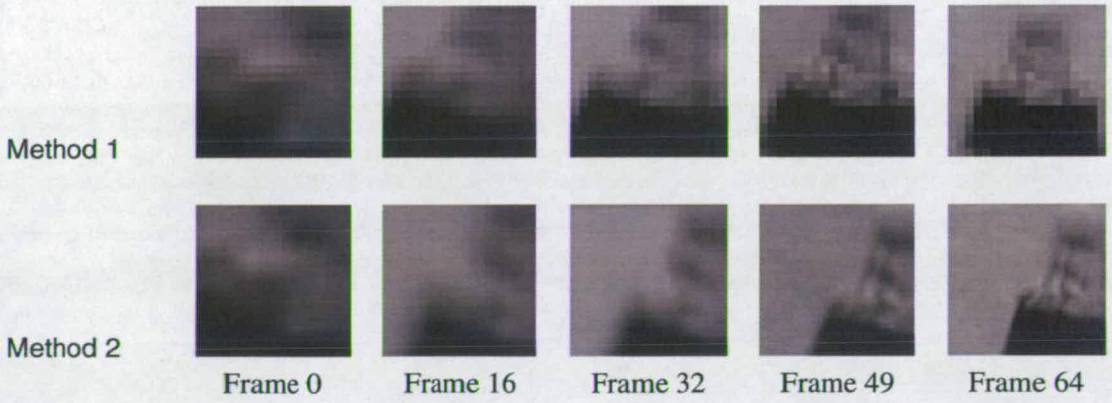
Figure 7.29: *Reference blocks for multiscale trackers*



(a) Sequence **`tom1`**



(b) Sequence **`tom3`**



(c) Sequence **`tom4`**

Figure 7.30: *Reference blocks for multiscale trackers*

7.4.3 Discussion

The conventional NCC block trackers failed on all sequences when the reference block was not updated. This is not surprising as there is a large scale change and the frame rates in the real-world sequences are not very high. FIR block updating appeared to improve the length of time the target was tracked for, however tracking results were still poor. The most improvement to robustness came from adding position filtering and prediction. This is largely due to the search area being correctly placed when the camera pans, tilts and zooms while only restricting the search area to a small portion of the image which is highly likely to contain the target. The best tracking results were obtained using Kalman block updating and position filtering/prediction. Interestingly, the NCC tracker with position filtering and Kalman block updating both failed on the sequence `bottle1`. Even though the reference block was updated, the target size became larger than the reference block and it was free to drift until it became fixed on a particular feature. This type of problem is very common in sequences where there is a large scale change – in the case of sequence `bottle1` a single pixel in the early frames becomes a 10×10 pixel block in later frames. This is larger than the reference block, therefore it can be expected that unless the feature tracked is a true corner, drift or mismatches will occur when the scale of the object increases.

Both novel multiscale tracking methods performed well, although not optimally. Method 2 appears to suffer from drift in the reference block which is most visible in the synthetic sequences. However, for real-world sequences this is not as obvious and the rapid updating appears to improve the tracking performance. The drift is largely caused by the smoothing effects when warping the reference block – sharp edges become blurred in the warped block and small mismatches slowly accumulate. At higher zoom factors this effect disappears as the size of the matched region approaches the size of the reference block, resulting in less blurring. As discussed in Section 8.3, a solution to this would be to use a more accurate interpolation method or attempt to detect and correct drift in the reference block. Method 1 is less susceptible to drift in the reference block, however this improved accuracy comes at the cost of performance – Method 2 is considerably faster than Method 1.

It is difficult to pick the 'best' overall tracking method as each of the tracking techniques performs differently over the range of sequences. For a real-time system, the NCC block tracker with Kalman block updating and position filtering appears to track most targets accurately with near real-time performance. However, as a good compromise technique, Multiscale Method 1

provides accurate tracking over a large PTZ range. The only fair conclusion that can be drawn is that the novel multiscale methods are yet another tracking tool and their performance depends largely on the sequence.

7.4.4 Runtime Performance

All of the algorithms implemented were intended to be run offline, therefore they contain a fair amount of debugging information which could be removed to improve performance. However, a vague indication of the runtime performance of the methods will be given as a rough guide. Method 1 generally took 2–4 seconds per frame depending on the size of the search aperture and the size of the reference block. The search area is typically a 600 pixel image which needs to be warped into the reference view. This requires a fair amount of processing time using unoptimised software texture mapping. This figure could be made significantly smaller with optimised assembler code and real-time by using rendering hardware.

Method 2 is considerably faster than Method 1 as only the reference block is warped into the current view. Typically, it takes less than a second per frame to match the target. Again, with optimised software or hardware acceleration, it could be feasible for this method to run in real-time.

The NCC trackers without block updating and using FIR block updating run in real-time up to 12.5 frames per second. When using Kalman block updating near real-time performance is approached. The size of the reference block affects the performance of the Kalman block updating significantly.

The contour tracker for fixed view sequences was very slow and took several hours for the contour position to settle. This was due to the large amount of cross-correlation values that needed to be calculated. Slow computation speed is a large disadvantage of this method, however with optimisation this time could be cut down considerably. The contour tracking method for PTZ sequences is considerably faster as the match forces are not calculated. Hence, the PTZ tracking contour settled in less than half a minute. This type of performance is more useful in a practical system and highlights the need for revising the calculation of the match forces or optimising them.

Improving the run-time performance of all of the methods discussed is briefly investigated in Section 8.3 as well as the possibility of using accelerated graphics hardware.

7.5 Contour Tracking

In this section the implementation details of the contour tracker, described in Chapter 5, are discussed below for use with a fixed view camera and under pan, tilt and zoom conditions. The results for both implementations are present in Section 7.5.1 and 7.5.2 and a discussion of these results follows in Section 7.5.3.

When the camera is fixed, the strategy of finding match forces[170] was used. However, in the presence of pan, tilt and zoom a simpler strategy was used that fits a least squares curve from the start point to the end point and adjusts this during the track (as described in detail in Section 5.4). The two strategies are compared in Section 7.5.3 where the merits of each approach are discussed.

7.5.1 Fixed view

Two tracking methods were implemented to compare with the contour track: a standard NCC block tracker and a NCC block tracker with Kalman filtered position prediction. Each tracker is run in two directions, one forwards and one backwards. Therefore, each half of the sequence is tracked with a separate reference block. A 16×16 pixel reference block is chosen from the start and end frame positions obtained from the hand-tracked ground truth. No block updating is performed in any of the tracker implementations. Strategies for updating the reference block and estimating the reference block are discussed in Section 8.3. The standard block tracker is initialised with a position from the start or end of the track (depending on whether tracking is done in the forward or reverse direction) and Normalised Cross Correlation is used to find a match position. In the next frame, the last match position is used as the centre of the search region.

As a second comparison, a tracker was implemented using a simple constant velocity Kalman filter (Section 4.2.6) to estimate the search position in each frame using observations of the best match position. The object track position was taken as the position component of the correct Kalman state vector. This results in the tracking output being smoothed and helps with outlier rejection due to false matches. Again, the tracker was run in the forward and reverse directions.

A curve fit was done to the best Kalman filtered tracker results as a further comparison to the contour tracker. The curve was initialised as a straight line between the start and end points

with the same number of control points as the contour. The curve is then fitted to the output of the Kalman fitted data by minimising the residual Euclidean distance between the curve and Kalman filter track points. The control points are moved such as to minimise this error. This process is repeated until none of the control points move.

The contour tracker was initialised with some fixed parameters for both sequences: the minimum distance any point must be from the contour was fixed to 0.5 pixels. Any match values less than 0.7 were clamped to 0, therefore contributing nothing to the match force. The temporal falloff was set to 3.0 and a maximum of 3000 iterations was set. As before, a 16×16 pixel reference block was used with normalised cross correlation to generate the match forces.

The following table compares the contour trackers to a conventional NCC tracker with and without Kalman position filtering to the contour tracker.

Sequence	NCC Tracker		Kalman Filter		*Curve fit to best Kalman*	*Contour Tracker*
	Forwards	*Backwards*	*Forwards*	*Backwards*		
`cyclist`	2.225	1.8294	2.6620	2.6409	2.6176	2.4639
`carpark`	1.5984	(track fails)	1.5411	(track fails)	1.5393	1.3251

Table 7.5: *RMS pixel errors for contour tracker – sequences without pan, tilt and zoom*

7.5.1.1 Sequence – `cyclist`

A 16×16 pixel block from the cyclists helmet was chosen as a tracking target point. The cyclist moves from left to right in front of buildings in the background with the camera remaining fixed. Near the end of the sequence the cyclist's helmet passes in front of a car of similar colour and becomes difficult to isolate from the background. Figure 7.31 shows a composite of four frames from the sequence with the contour position overlayed in yellow. The RMS pixel errors for each of the tracking methods is shown in Figure 7.35.

7.5.1.2 Sequence – `carpark`

In this scene, CCTV footage of a car passing around the perimeter of a carpark is shown. The front headlight of the car was chosen as the tracking target and a 16×16 pixel reference block was created from this point in the first and last frames. During the sequence, the headlight becomes partly occluded as it passes behind the fence. Figure 7.33 shows a composite of four



Figure 7.31: *Composite* `cyclist` *– contour position shown in yellow*

frames from the sequence with the contour position overlayed in yellow. The RMS pixel errors for each of the tracking methods are shown in Figure 7.35.

7.5.2 With pan, tilt and zoom

The tracker is initialised with the start and end points obtained from the ground truth (these could easily be set by an operator). Multiscale Method 2 (Section 4.6) was chosen as the block matching method as it produces good results for good performance. Method 1 should be used (Section 4.5) for a sequence where accuracy is more important than computational speed. All matching is done in greyscale and the reference block was updated using FIR block updating with α set to 0.15.

Ten control points were chosen for all sequences except `tom1` which used five control points. If a small number of control points is used, the contour position is more constrained and rejects more false matches. In `tom1` a large portion of the target's motion is partially occluded and the contour must be forced to pass through the correct region. Too many control points allow the contour to pass through areas of the image that produce high false match values. The selection



frame 000 (start) frame 154 (end)

Figure 7.32: *Sequence* **`cyclist`** *– start and end tracking positions*

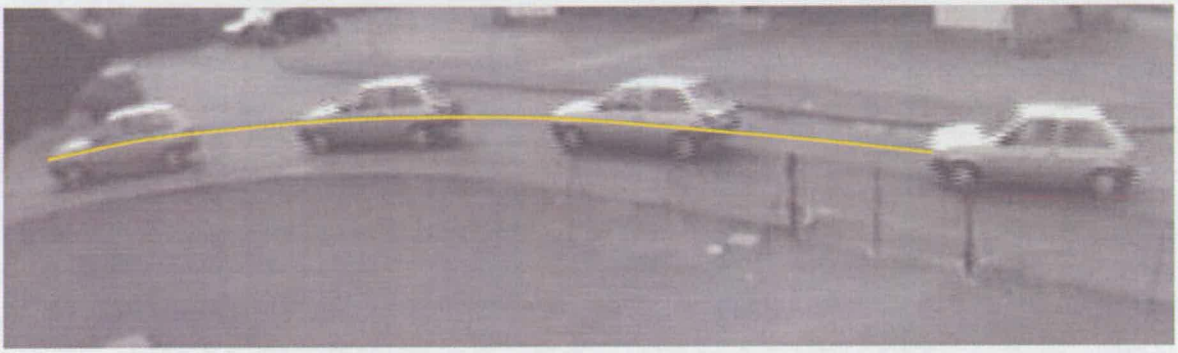


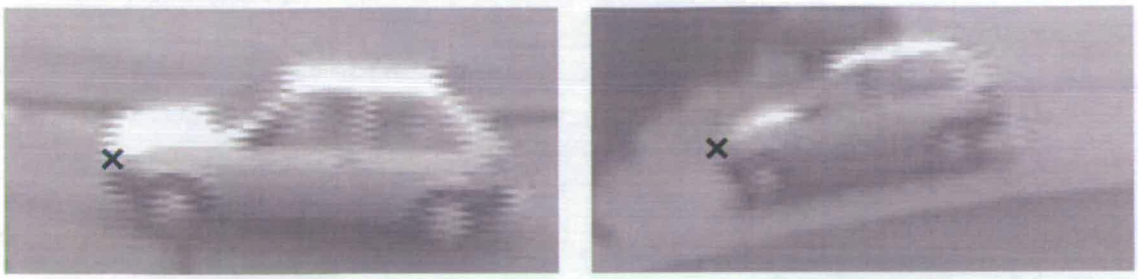
Figure 7.33: *Composite* **`carpark`** *– contour position shown in yellow*

of control points is a tradeoff between easy editability (by a human operator), fitting the true motion of the target and rejecting areas of false matches.

False matches are rejected by constraining the match to be within *distance 1* pixels of the estimated contour position if the match value is below the *match value threshold*. If the match does not satisfy these parameters, then it is constrained further to lie within *distance 2* pixels of the estimated contour position (*distance 2* must be less than *distance 1*) and must have a match value greater than 90% of the current match value. If none of these criteria is met, then the estimated position of the contour is used as the current match position.

Each sequence requires slightly different parameters to track effectively with the contour tracker. These parameters are summarised in Table 7.6. There is no automated way to estimated these parameters and they would need to be set by a human operator or tuned for a specific application. Automatic estimation of these parameters was not investigated however this could be a future area of research.

The match positions were taken as the true matches of the reference block with the image. It



frame 1825 (end) frame 1925 (end)

Figure 7.34: *Sequence* **carpark** *– start and end tracking positions*

Sequence	*Block size (pixels)*	*Control points*	*Search area (pixels)*	*Distance 1 (pixels)*	*Distance 2 (pixels)*	*Match Value Threshold*	*Update Threshold*
bottle1	9 × 9	10	10	20	10	0.90	0.82
tom1	9 × 9	5	5	15	7	0.94	0.82
tom3	9 × 9	10	20	20	10	0.90	0.82
tom4	12 × 12	10	15	20	10	0.90	0.82

Table 7.6: *Parameters for contour tracker per sequence*

is possible to use the contour position in each frame as the match position, however, with the low frame rates in the real-world sequences this would lead to gross errors. With higher frame rates it is possible to use the contour positions as shown in the fixed view sequences. It is also possible to interpolate the position of the target between frames using the contour.

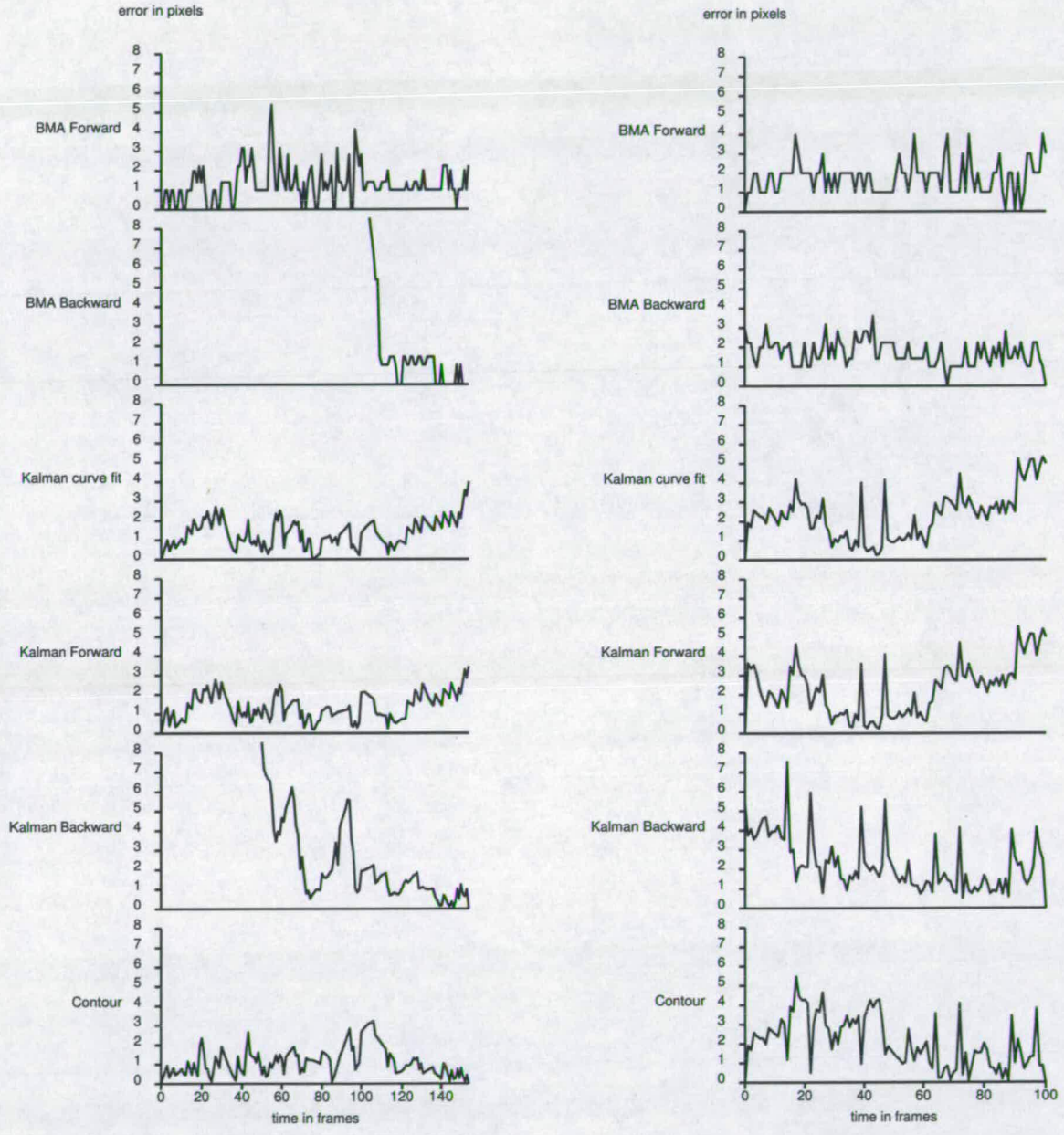
The following table summarises the results of the contour tracker for all the real world sequences:

	NCC Tracker				Contour Tracker			
Sequence	*No update*		*FIR update*		*No update*		*FIR update*	
bottle1	57.8†	(40)	9.3†	(40)	1.0	(40)	1.2	(40)
tom1	2.6*	(20)	1.6*	(16)	2.0	(50)	2.3	(50)
tom3	4.4	(42)	1.1	(42)	2.9	(42)	1.9	(42)
tom4	8.5†	(65)	4.7	(65)	1.4	(65)	2.0	(65)

† Track fails * Partial track

(See Section 7.2 pg 170 for explanation of table)

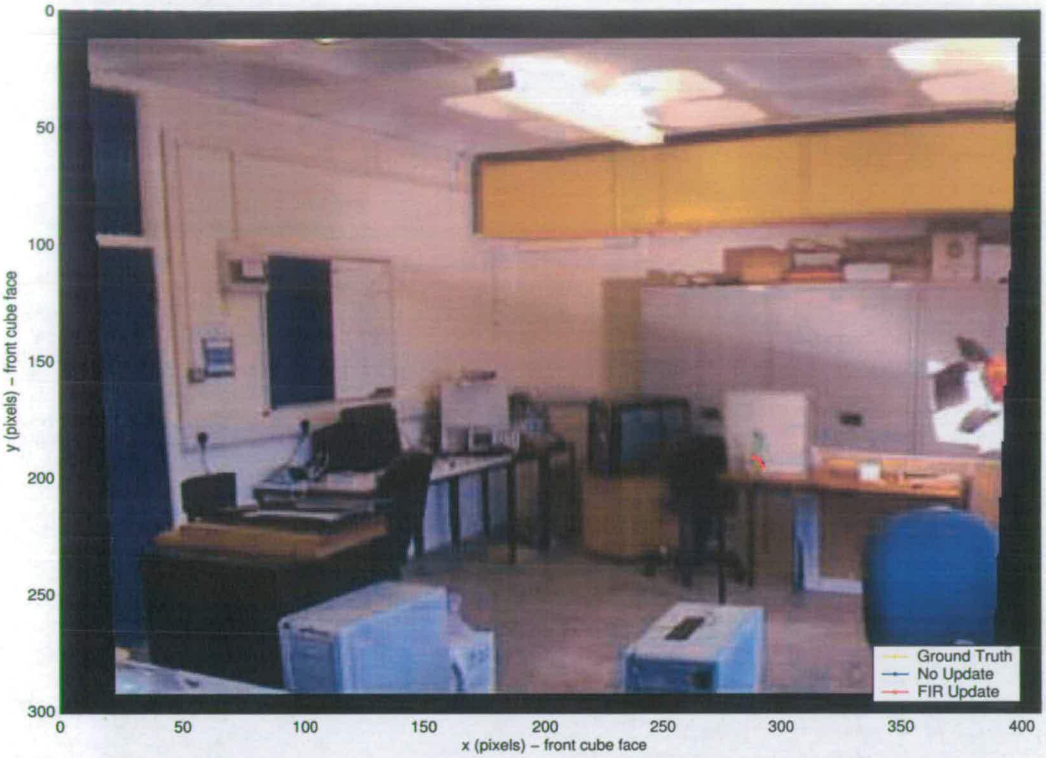
Table 7.7: *RMS pixel errors for contour tracker for sequences with pan, tilt and zoom (all units are image pixel co-ordinates)*



(a) RMS pixel errors for sequence `cyclist`

(b) RMS pixel errors for sequence `carpark`

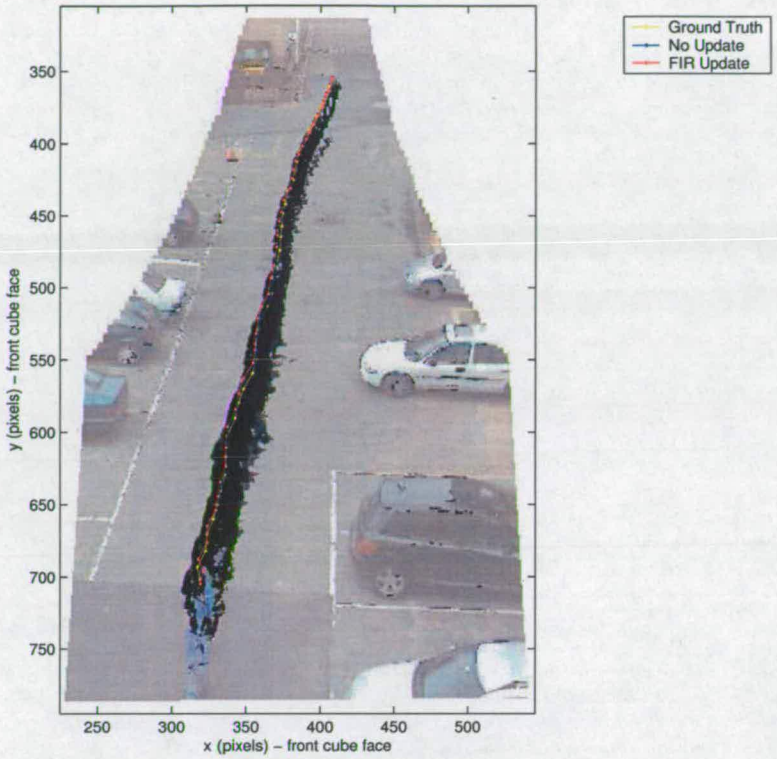
Figure 7.35: *RMS pixel errors for contour trackers with fixed camera views*



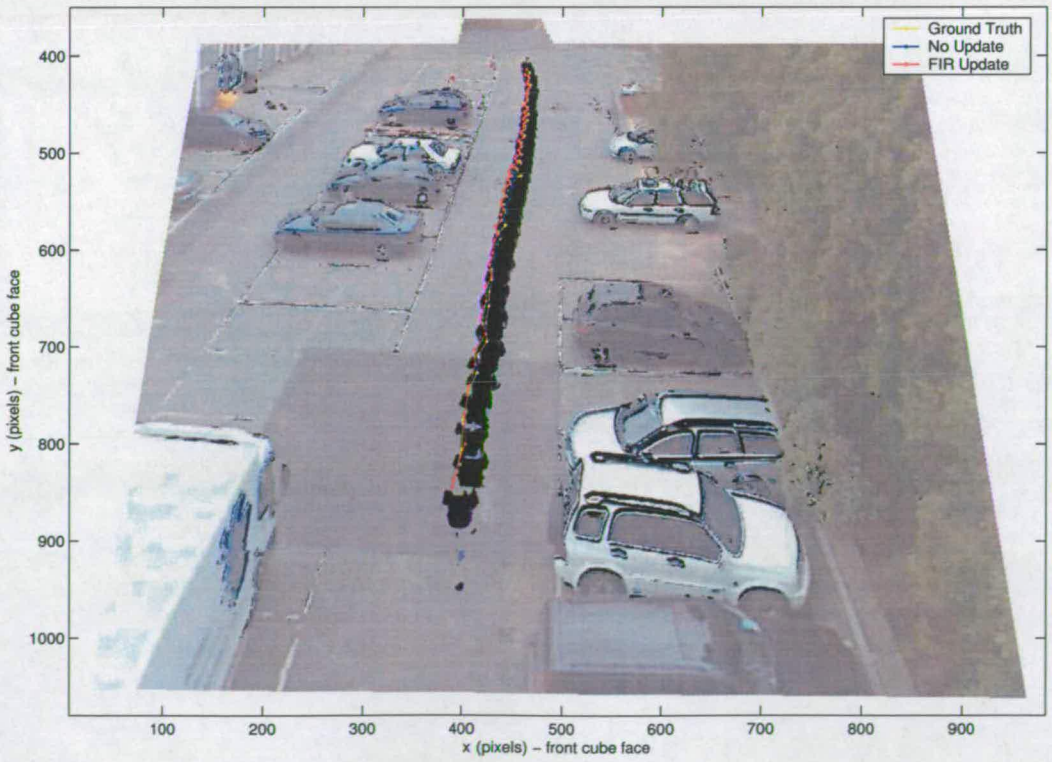
(a) Sequence `bottle1`



(b) Sequence `tom1`

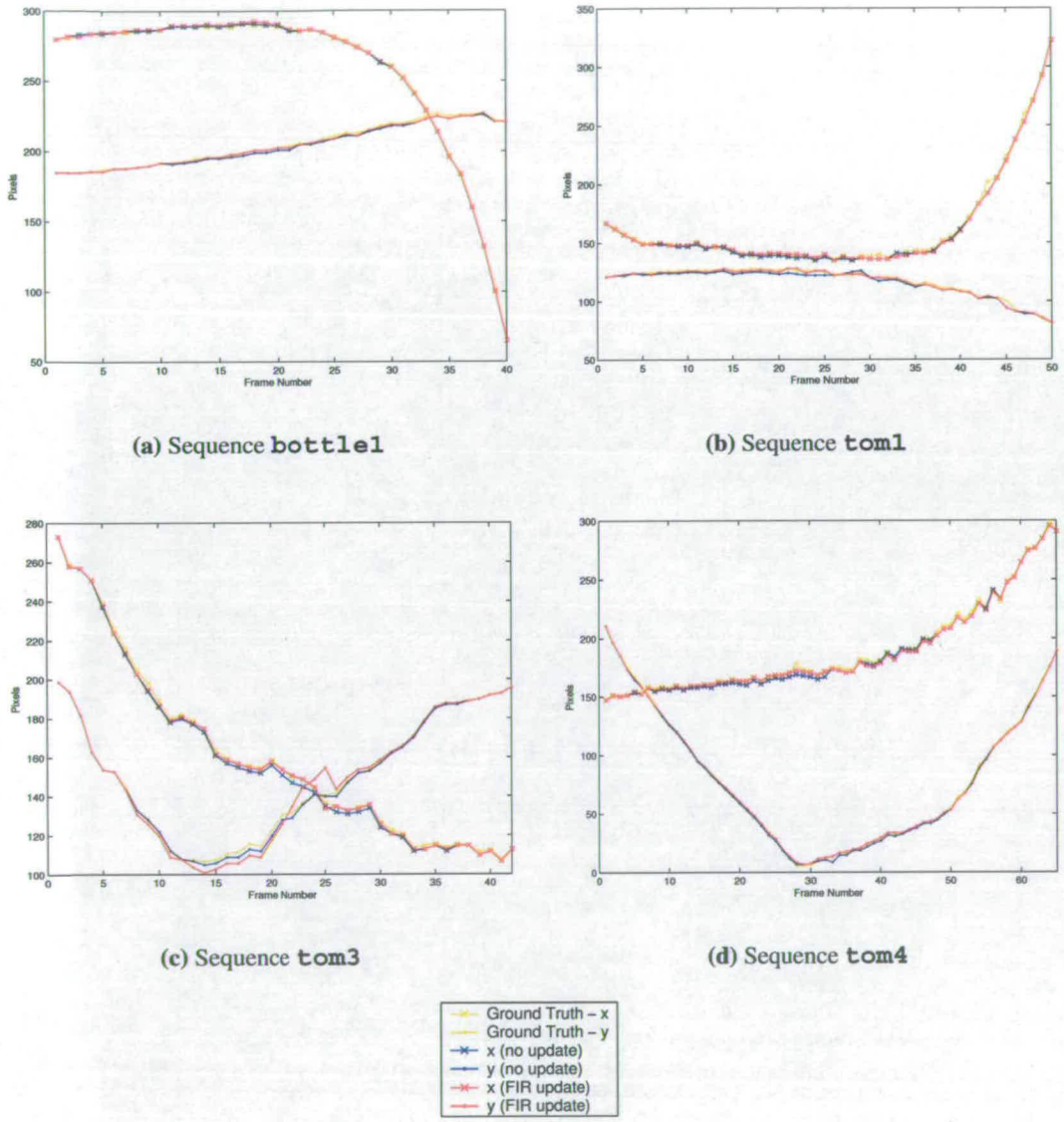


(c) Sequence **tom3**



(d) Sequence **tom4**

Figure 7.36: *Contour Tracks for multiscale tracker*



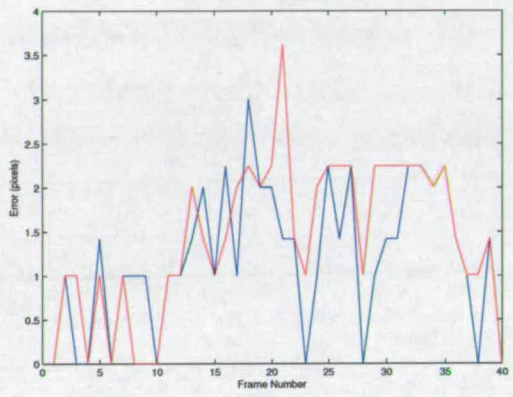
(a) Sequence `bottle1`

(b) Sequence `tom1`

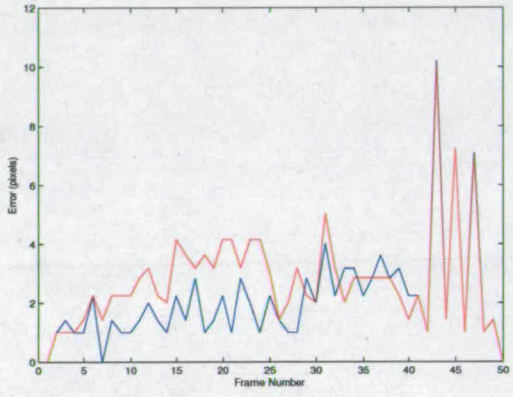
(c) Sequence `tom3`

(d) Sequence `tom4`

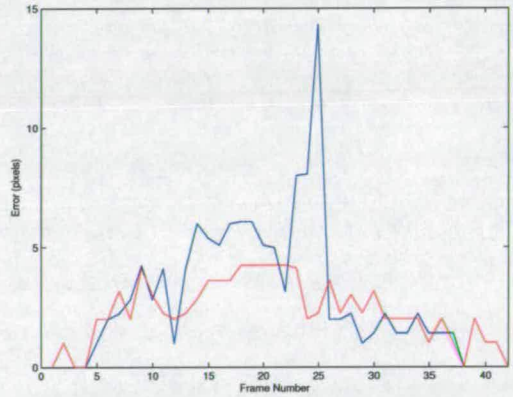
Figure 7.37: *Tracks for contour tracker*



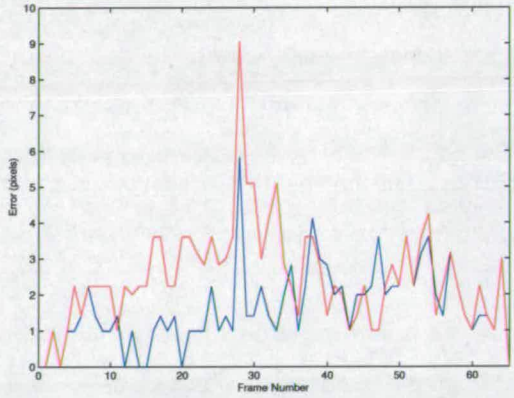
(a) Sequence **bottle1**



(b) Sequence **tom1**



(c) Sequence **tom3**



(d) Sequence **tom4**



Figure 7.38: *Errors for contour tracker*

7.5.3 Discussion

The backward NCC and Kalman trackers both lose track in the `cyclist` sequence, however the contour tracker appears to match the ground truth well. The contour tracker shows improved results over the Kalman filtered tracker and the curve fit to the Kalman filtered tracker. The forward NCC tracker appears to perform better numerically than the contour tracker, but this could be largely due to synchronisation errors and interlacing causing jitter which is smoothed by the contour. Depending on the application, the tracking of the jitter may be more important than a smooth track. The spikes in the error plots in the `carpark` are largely due to synchronisation errors which are due to the original video sequence being stored on VHS tape – both the contour tracker and the Kalman filter tend to over-smooth the track.

Generally, the contour tracker produced the most accurate tracks of the tracking methods implemented. It does, however, have one major drawback, which is that it can only operate offline. This makes it unsuitable for any sort of real-time tracking. The improvement it achieves comes from reducing the error at the end of the track which normally occurs to drift in the reference block or the target changing over time. This approach also has benefits, such as the start and end points of the track are guaranteed. For problematic tracks, it would be possible to subdivide the track into several portions, hence minimising the total error in the track, by ensuring that the track passes through fixed points chosen by a human operator.

Ensuring that the track ends at a certain point improves even simple tracks such as in the sequence `bottle1`. In the normal multiscale tracking methods the reference block is created from a wide angle view, so by the time the camera has zoomed in, the pixels in the original block are now represented by several pixels. This creates an ambiguity in the match – if the scale change over the duration of the sequence was $5\times$ for example, a pixel in the first frame is now represented by five pixels. If the feature being matched is an edge or a corner (which are normally considered good features to track), the edge transition that was one pixel is now five pixels. This creates an ambiguity of up to five pixels in the match position. By fixing the end point this ambiguity is removed by tracking towards the centre of the sequence therefore moving the ambiguity to the centre of the sequence. It could be argued that in a sequence where the camera zoomed in and then zoomed out again, this ambiguity would reappear – the solution is to divide the sequence into two sequences.

The sequence `tom1` is the most interesting sequence for the contour tracker as it is possible to

track through a large occlusion where the other tracker implementations failed. By knowing the start and end points of the track it is possible to estimate the target's trajectory with a large degree of certainty.

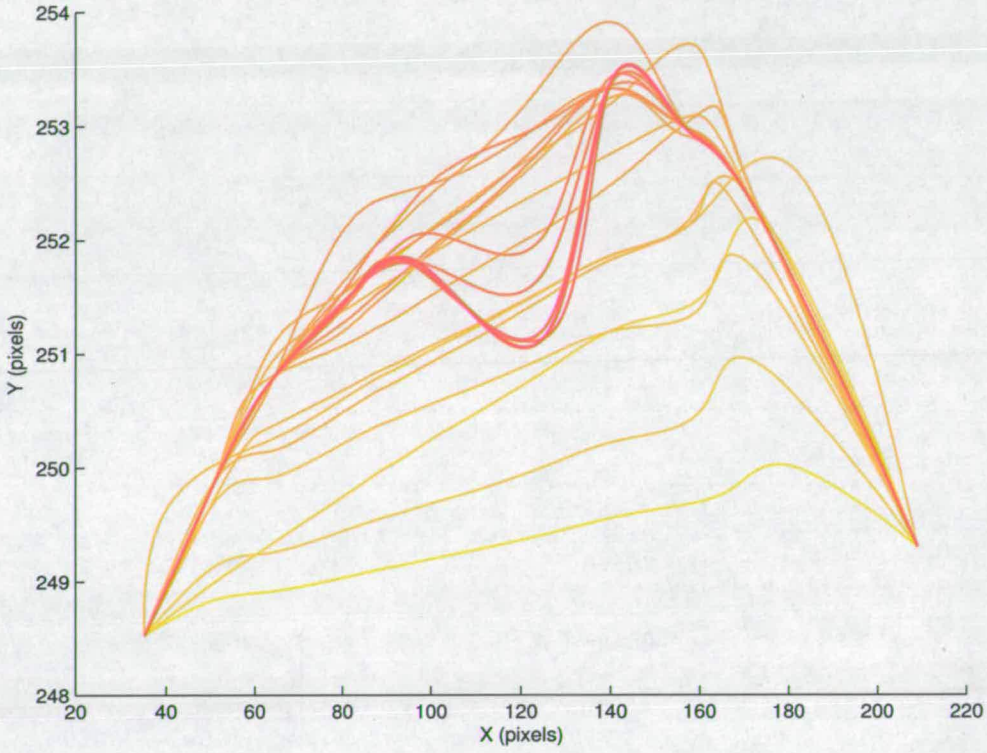


Figure 7.39: *Contour evolution* `tom1` *– the early positions of the contour are shown in yellow and the final positions are red*

Figure 7.39 and Figure 7.40 show the *evolution* of the contour over time. As more matches are added to the least squares fitted contour, the more the contour approximates the true path of the target (shown by the ground truth in yellow). During the partial occlusion there is still significant mismatching. This is largely due to the low frame rate, which prevents the reference block from updating robustly – large variations in the background of the block need to be incorporated slowly for good matching. Two possible solutions would be to segment the block into foreground and background based on the temporal variations of the pixels or to increase the update rate of the FIR filter. Both of these are beyond the scope of this work and could be investigated at a later stage.

It is also possible to see from Figure 7.40 that the contour position provides a better match to the target's true path than the actual match positions. In a case such as this, it would be better

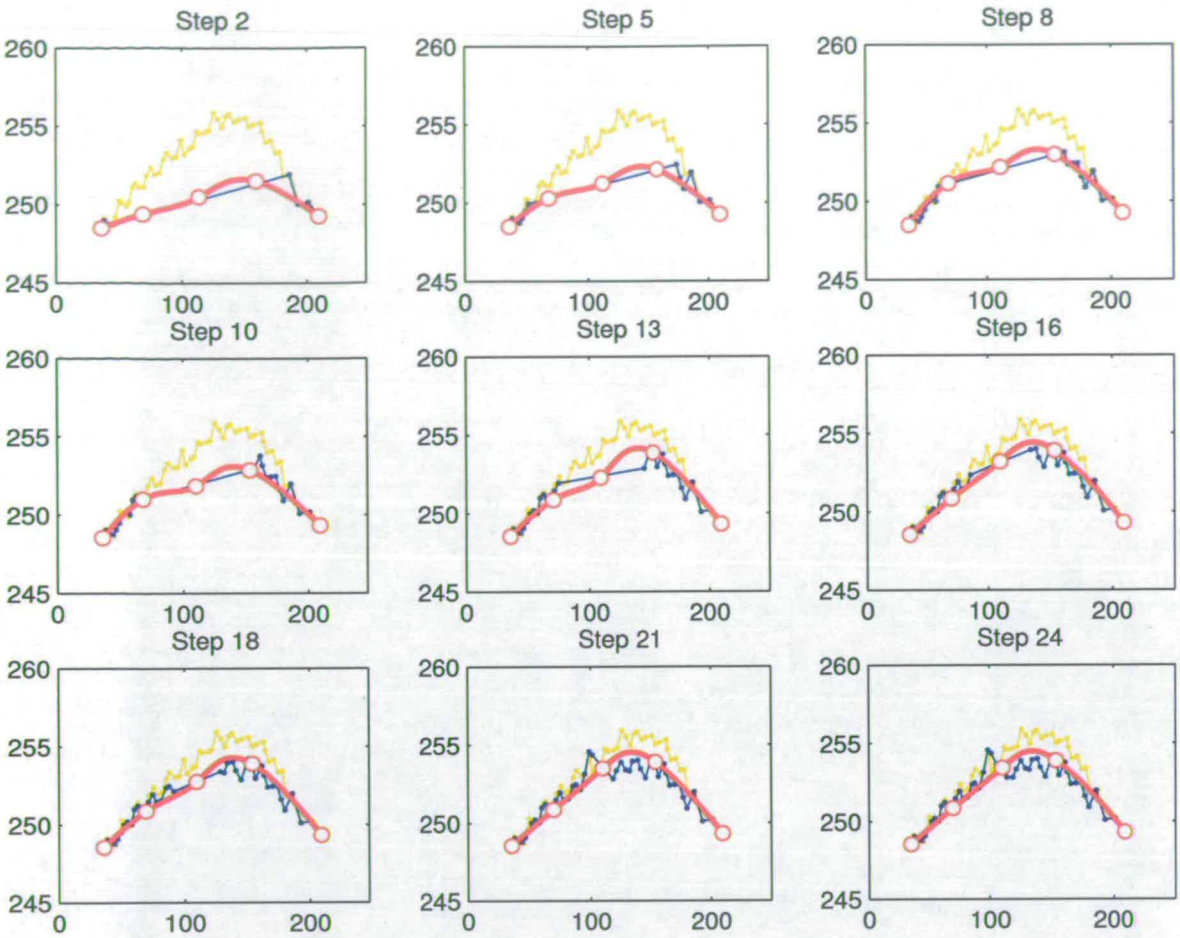


Figure 7.40: *Contour evolution* `tom1` *– the blue line shows the match positions in the image and red line shows the contour with control points marked as circles (ground truth shown in yellow)*

to use the contour position in each frame as the match position. In a higher frame rate sequence this would be possible; however, given the jagged nature of the ground truth, using the smooth contour positions could produce unwanted effects. Improved results could also be obtained by repeating the tracking process without re-initialising the contour and reducing the search area (and distance thresholds) until the contour position remains unchanged. This method is similar to the approach used in the fixed view contour tracking method (Section 5.3).

In a sequence without occlusion the contour tracker performs well, producing better results than the multiscale tracker and being comparable to the conventional block tracker with Kalman block updating. Figure 7.41 shows the *evolution* of the contour during matching. As the trajectory of the target in world space co-ordinates is nearly a straight line it is important to have

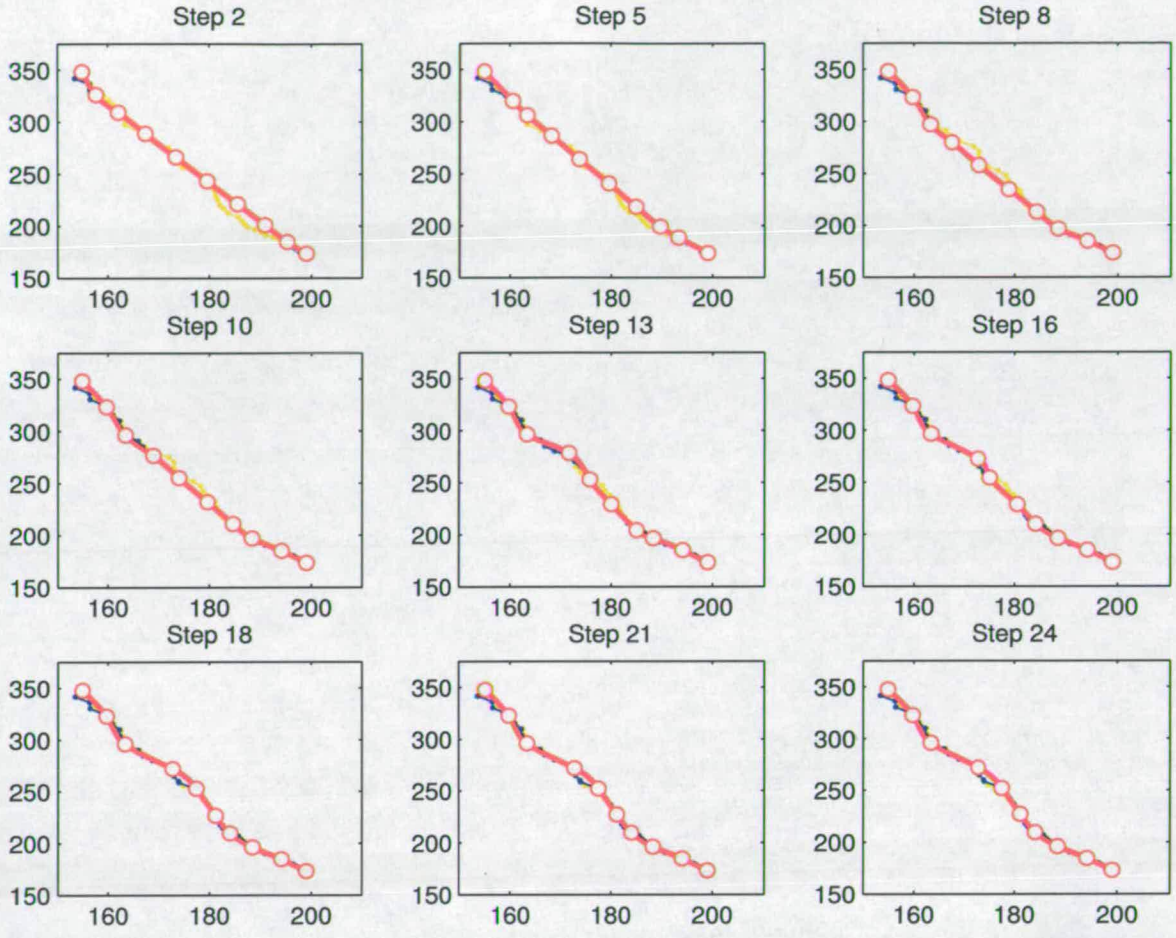


Figure 7.41: *Contour evolution* `tom3` *– the blue line shows the match positions in the image and red line shows the contour with control points marked as circles (ground truth shown in yellow)*

a larger number of control points to allow for the jagged motion caused by the low frame rate. Another interesting effect that is visible in Figure 7.41 is the velocity of the target. As the control points are spaced equally temporally, the closer they are spaced the slower the target is moving.

It is also interesting to note that only half of the frames in the sequence are needed to estimate the trajectory of the target. Given a sequence with high frame rate and a target that moves with constant velocity, only a few frames would be needed to estimate the position of the object.

Sequence `tom4` tracks well with the exception of the small region of mismatches near the centre of the sequence (visible in Figure 7.38 at frame 28). This is caused by the target nearly leaving

the edge of the frame. This effect could be compared to a partial occlusion. The performance of the tracker is adequate in this section of the track. If this type of result occurred in an application such as video post-production, a human operator could edit the contour easily and correct the mismatch if it was obviously incorrect visually.

Block updating appears to have a negative effect on the tracking performance in general. This is due to the low number of frames and small changes in the target. Given more frames or larger variations in the target it would be necessary to update the reference block. For example, in the sequence `tom3` the size of the object varies little, so it is possible to track the target from the original reference block – this will not always be the case.

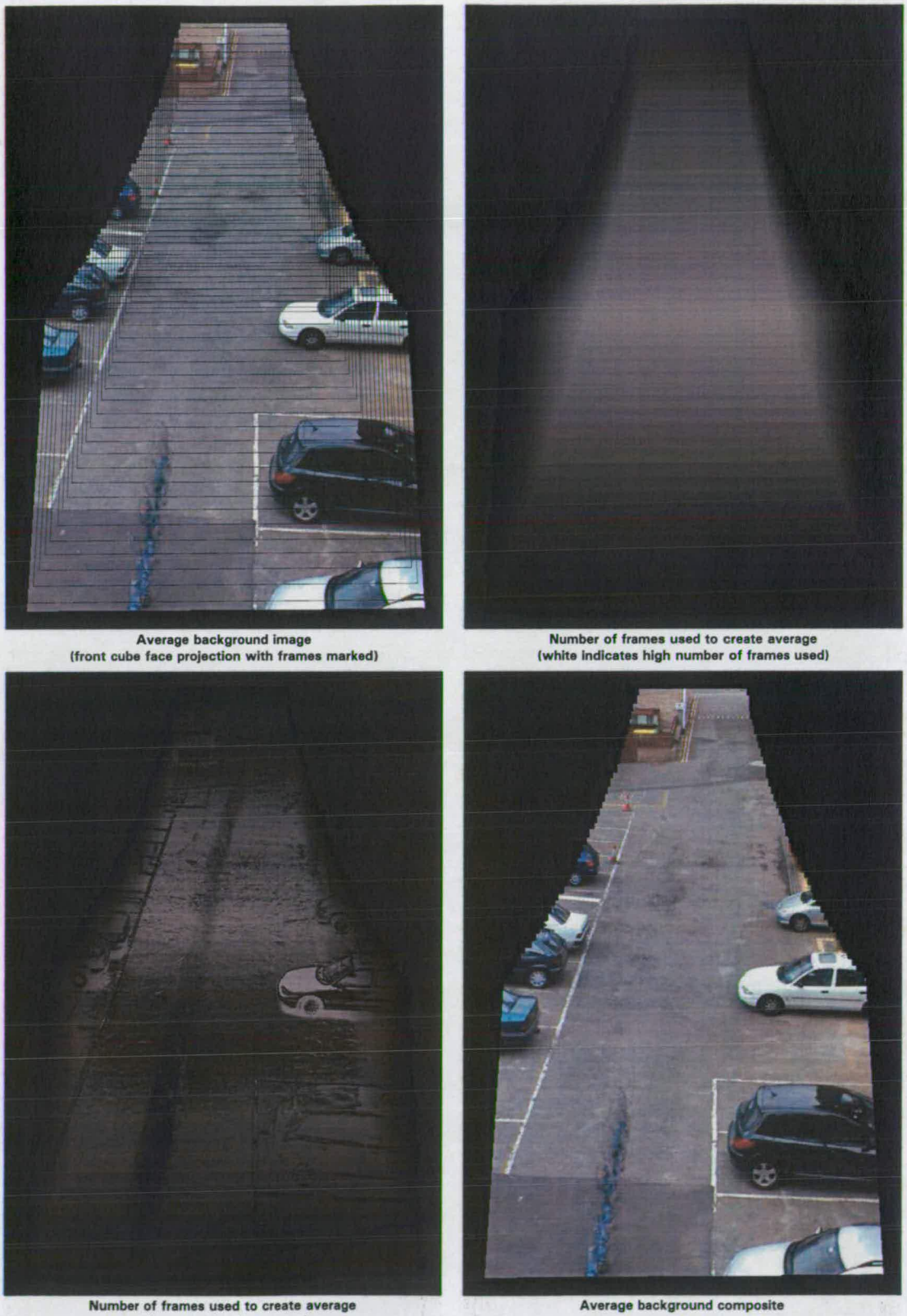
The tracking contour seems to provide consistently good results for all sequences with pan, tilt and zoom, outperforming all other trackers except in one sequence. It also guarantees the start and end positions, which for many post production type applications is fairly important.

7.6 Region Tracking

A simple region tracker was implemented as a comparison to the block matching methods. In many surveillance type applications a moving region segmentation may be available or the objects of interest are only those that are moving. Therefore, it is necessary to look briefly at applying the same methods used in block tracking to regions. Many better region trackers exist and a large body of work has been done on identifying moving objects, all of which could not be implemented to compare with the block trackers. Implementation details can be found in Section 7.6.1 and the results produced are presented in Section 7.6.2. A short discussion of the results can be found in Section 7.6.3.

7.6.1 Implementation

An average front cube face image was created (as described in Section 4.8) by averaging the pixels in each transformed frame and removing the changes (similar to outlier rejection). The temporal average of each pixel was calculated and changes outside of an intensity threshold were rejected. The top left and bottom right sections of Figure 7.42 show the number of frames used to create the average. In areas of high change (where objects are moving or around edges) only a few frames contribute to the average after thresholding. At the bottom of the average



Average background image
(front cube face projection with frames marked)

Number of frames used to create average
(white indicates high number of frames used)

Number of frames used to create average
after removing foreground

Average background composite

Figure 7.42: *Region Tracking for sequence* `tom3`

the moving person is visible. This is due to a low number of overlapping frames meaning it is impossible to separate the foreground from the background. In some areas the true background is never visible.

The average was created using less pixels than are available i.e. not every pixel in the high zoom factor views was projected to the reference view. This was done to show how the technique could be used in real time with an OpenGL type implementation (either using software or hardware rendering). The medium resolution average is smoothed when warped to wide angle views and interpolated when warped to close angle views. The resolution was chosen to provide a good balance between detail and memory usage (and hence rendering time).

The change in each frame is detected by transforming the average image into the current view and subtracting it from the frame. The difference image is then thresholded, dilated, eroded and region growing is performed to identify regions. Regions are then sorted by the number of pixels they contain and the centroid co-ordinate is used as the tracking point.

Figure 7.43 illustrates the process of warping the average background into the current view and segmenting the changed regions. The matched region is then used to create the segmented target in the most right-hand column.

A cubic spline tracking contour, as described in Section 5.4, was used to constrain the position of the target. A region was considered a match if the tracking contour lay within its bounding box and the centroid co-ordinate was within 10 pixels of the contour position. A size constraint was also placed on the region, to exclude fragmented regions that had mistakenly become joined together and lying close to the contour position.

A limit to the number of pixels is set so that 'large' regions are created from spurious connection of pixels during dilation and erosion. The number of pixels is also an indication of whether the object matched is the same object, although fragmentation (where a region splits into two regions) is common and can lead to mismatches.

The position of matches is constrained using a tracking contour, which avoids the unnecessary tweaking of Kalman position filters to predict the next position and size of the object. In real-time applications contour tracking is not viable and a more intelligent prediction system should be used to constrain the search region.



Figure 7.43: *Difference image* (left)*, after thresholding* (centre) *showing regions and segmentation* (right)

7.6.2 Results

Only a single sequence is presented to show results for region tracking and these are represented visually as the quality of the track is largely dependent on the segmentation method. Figure 7.43 shows the matching process for several frames and Figure 7.44 shows a composite of match regions. No hand-tracked ground truth was produced as it would only serve to measure the quality of the segmentation and not the quality of the tracking method. A poor segmentation, such as fragmenting the target into two regions, would produce a poor numerical tracking results as the region pixel count would drop and the centroid would move to the centroid of one of the regions. Many segmentation methods exist (see Section 4.8) and it was felt that exploring all of these was beyond the scope of this work.

7.6.3 Discussion

Visual inspection of the segmentation in each frame (summarised in Figure 7.44) shows good identification of the target throughout the sequence. The target was found in every frame and there were very few instances of fragmentation. Unfortunately, as was discussed above, it is not possible to create a numerical evaluation to the track and hence a comparison to the block trackers. However, from the visual results it shows that for the identification of the target, the method was adequate for applications such as surveillance. The segmentation could be used, for example, to create a composite view for surveillance of a large area (such as the car park in the test sequences) and track targets at high zoom factors with a PTZ camera, or even a network of cameras[3, 13, 87, 114]. A high quality segmentation is not necessary for security applications as the application would be only to identify moving objects and not to remove them perfectly from the background. If a high quality segmentation was needed, another averaging and segmentation method would need to be used. The algorithm took roughly one minute to execute on a quiet dual processor 1.8GHz Intel Xeon to track 64 frames. The rendering time to create the average image took ten to fifteen minutes, however with optimised rendering code (or hardware rendering) this could be cut down significantly.

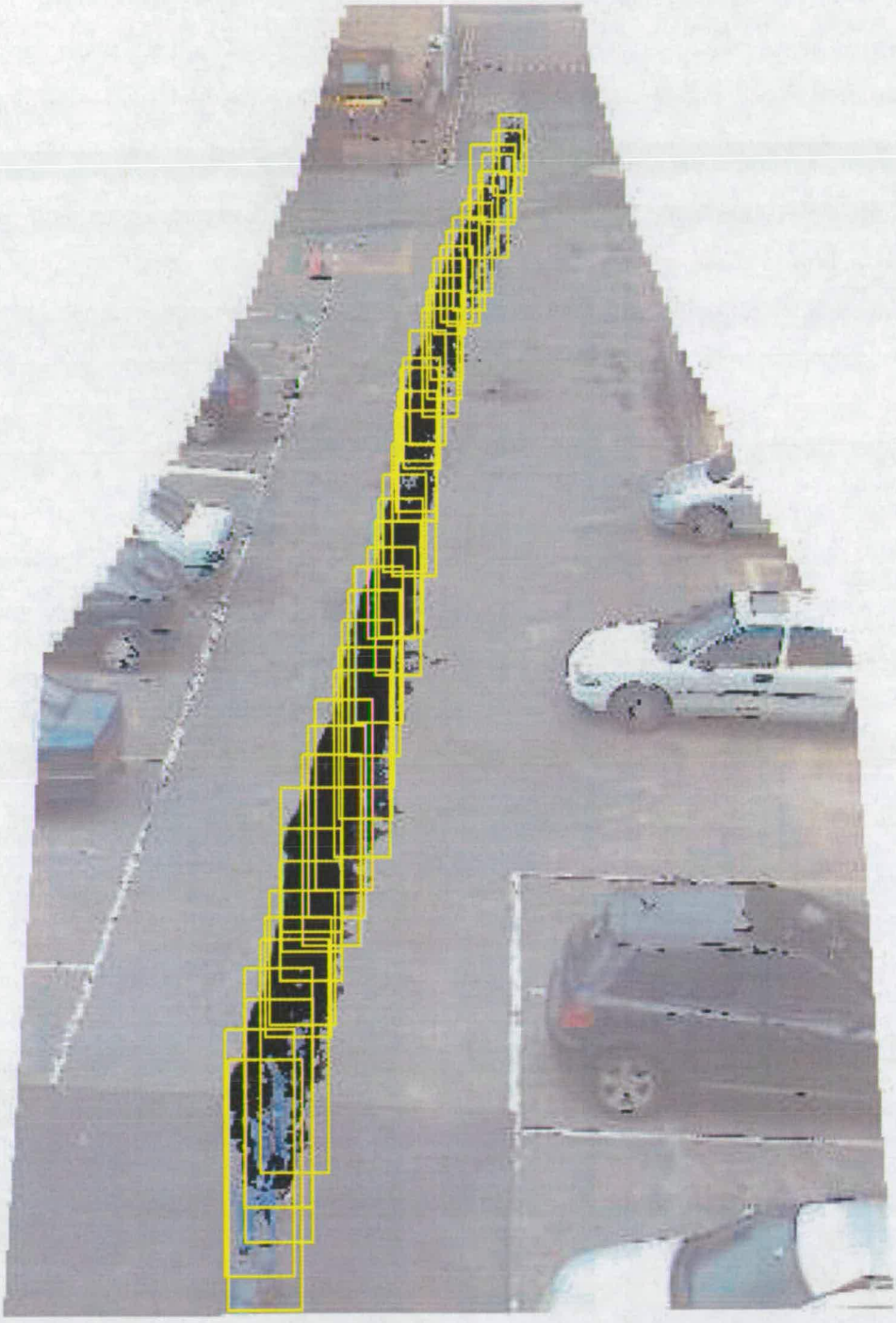


Figure 7.44: *Region track for sequence* `tom3` *in world space co-ordinates (projected onto the front cube face) overlayed on a composite image of the sequence*

7.7 Comparison of methods and discussion

As with many image processing techniques selecting the correct tracking method depends heavily on the sequence and target to be tracked. Other factors, such as accuracy and computational time also need to be taken into account. This section explores the suitability of each method to the various sequences and compares their effectiveness.

Table 7.8 below summarises the performance of all the multiscale tracking methods against a conventional NCC block tracker and an improved NCC block tracker with position filtering and Kalman block updating. All multiscale methods use block updating and position filtering is used to constrain the search region.

Sequence	NCC Block Trackers				Multiscale trackers				Contour tracker	
	No update		*Kalman update & position filtering*		*Method 1 (with update)*		*Method 2 (with update)*		*(with update)*	
`bottle1`	36.8*	(24)	9.3†	(40)	3.3	(40)	6.7	(40)	1.2	(40)
`tom1`	2.6*	(20)	1.6*	(16)	1.7*	(16)	2.3*	(27)	2.3	(50)
`tom3`	0.4*	(3)	1.1	(42)	3.0†	(42)	3.2	(42)	1.9	(42)
`tom4`	21.3†	(65)	4.7	(65)	8.3*	(65)	4.0	(65)	2.0	(65)

† Track fails * Partial track

(See Section 7.2 pg 170 for explanation of table)

Table 7.8: *Summarised RMS pixel errors all trackers for real world sequences (all units are image pixel co-ordinates)*

The most obvious conclusion that can be drawn from the above results is that a plain NCC block tracker with no block updating and position filtering (or camera model awareness) fails hopelessly for all sequences. Adding some sophisticated block updating and position estimation based on the camera parameters, provides a fast and fairly accurate tracking solution. The best NCC tracker could operate in real time for small block sizes.

The novel multiscale tracking methods perform well for all sequences, providing two approaches: one for accuracy and one for speed. Although neither method was implemented to run in real-time, with optimised coding and some hardware acceleration opportunities it could be possible to use these methods in real-time applications. Block updating appears to affect the accuracy of trackers negatively, however given a changeable or non-rigid object, accuracy may have to be sacrificed for maintaining the track.

For all the block tracks the unpredictable nature of the block updating and normal variations in

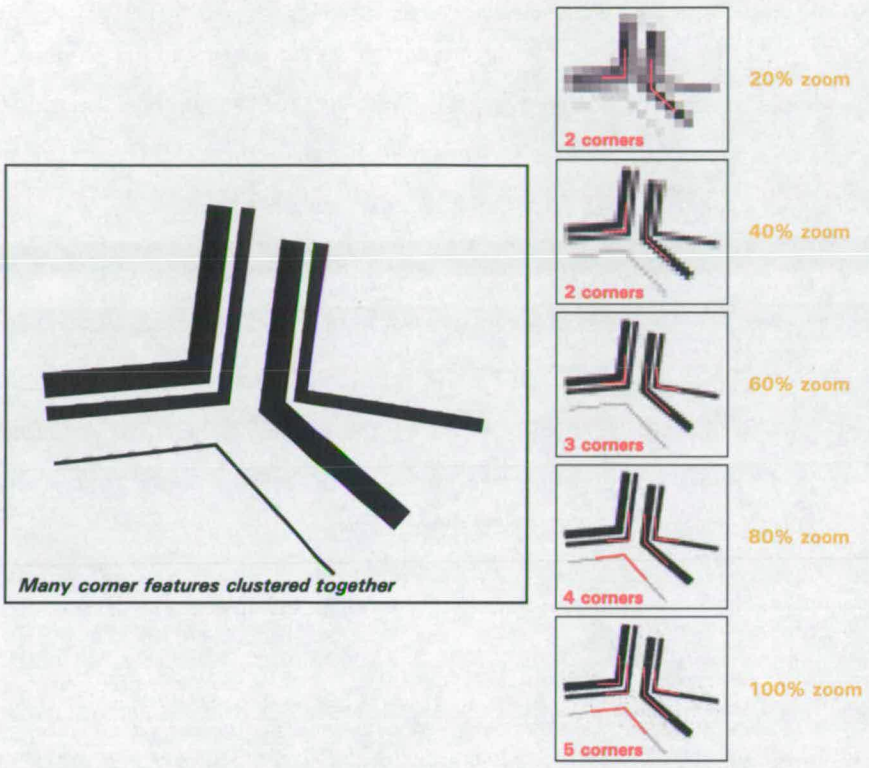


Figure 7.45: *Several clustered corner features appear as one corner at low zoom factors and become many corners at high zoom factors*

the target, could result in the tracker wandering off the target (or onto another portion of the target) over a long period of time. The solution to this is to use the contour tracker, which ensures fixed start and end points. Results produced are very accurate and interframe interpolation can be obtained from the estimated contour position between frames. The only drawback of the contour tracker method is that it can only run offline. Many position prediction methods exist such as Kalman filters and recently particle filters.

In other applications, such as surveillance and security, tracking moving objects is the primary task. The region tracking method provides good results using simple techniques. The accuracy and robustness of the tracker depends largely on the segmentation method, but is an attractive alternative to block matching techniques. Several region tracking systems have been successful for implementing surveillance systems to tracking people with PTZ cameras, such as [3, 13, 15, 18].

For a conventional block tracker to track well on a sequence where the camera zooms, the feature being tracked must be fairly invariant to scale and rotation such as a SIFT[165] feature

or a corner. Sometimes, a 'corner' feature is a corner at low zoom factors, however, as the camera zooms in it will split into several corner features. An example of this is shown in Figure 7.45. As the zoom factor increases, the block tracker will update the block and it will randomly lock onto the nearest feature. It is not possible to predict which corner it would prefer or if in fact this has happened without a more sophisticated method.

Unfortunately, no obvious conclusion can be drawn as to which tracking method suits the widest range of scenarios from the sequences collected, but more can be said about the suitability of each method to different situations. The novel multiscale methods will consistently produce more accurate results than any conventional block tracker, even if the camera model is incorporated. For a conventional block tracker to compete, the feature being tracked must be fairly invariant to scale and rotation (such as a SIFT[165] feature).

A second scenario where a conventional block tracker will fail is if there is rapid scale change or erratic scale changes. Even incorporating the camera model to accurately estimate the position of the target does not estimate the appearance of the block given the new camera view. The multiscale methods would only fail if the camera model is incorrect or if the target object has changed significantly between views. In this case, most tracking methods will fail.

7.8 Summary

In this chapter a set of test sequences have been introduced. They have then been used to evaluate the performance of three tracking methods (block tracking, contour tracking and region tracking) under pan, tilt and zoom conditions. The tracker software was developed to work in the context of a system that could use accelerated video hardware and OpenGL to achieve real-time tracking results, however only a software rendering solution was implemented. Section 8.3 discusses the possibilities of using graphics hardware to improve performance. A comparison of the various methods and results obtained from each was discussed in Section 7.7.

The novel multiscale block tracking methods performed consistently, however, a conventional NCC block tracker with position prediction and Kalman block updating proved to be faster and more accurate for most test cases, except when tracking a feature which changes significantly with increasing zoom (i.e. *not* a corner or edge feature). A discussion of these results can be found in Section 7.4.3.

An offline contour tracker produced the most accurate tracking results for all test sequences (with and without the presence of pan, tilt and zoom). The applications of the tracking contour are limited to offline applications as all frames in the sequence are required simultaneously.

A basic region tracker was implemented as an alternative tracking method and for brief comparison. In the application of tracking for surveillance, this would be the most likely choice of tracker as a motion segmentation is likely to exist already.

Chapter 8 follows with concluding comments on the work and results presented in this thesis. It also explores possible futures extensions to the methods discussed so far and suggests some strategies for implementing them.

Chapter 8
Conclusions and Future Work

8.1 Introduction

Concepts from projective geometry, computer graphics and computer vision have been brought together to produce a hybrid method of block tracking targets using a PTZ camera. The idea of a tracking contour for offline tracking was also introduced and analyzed.

Section 8.2 contains a critical discussion on the methods and analysis which have been presented in this thesis. Various avenues of future research and extensions to the proposed methods can be found in Section 8.3. In conclusion, Section 8.4 contains some final remarks.

8.2 Discussion

This thesis has introduced and brought together concepts from computer vision, projective geometry and computer graphics to produce a hybrid method of implementing tracking algorithms. Useful methods from each of these areas were brought together using a common notation and approach in the context of implementing algorithms in a rendering environment such as OpenGL. This approach can produce huge gains in performance by using accelerated graphics hardware to perform matrix transforms on images with real-time performance. Several other examples of this convergence between these fields were introduced, such as structure from motion, structured lighting and environment maps. Each of these fields has a weakness and a strength: projective geometry provides a rigourous mathematical treatment of the subject which is extremely powerful in its formal style, however this also makes it inaccessible. Computer graphics on the other hand glosses over the mathematical aspects of projective geometry and simply defines the projective transforms necessary to implement a 3D computer game or visualisation system. By bringing these two fields together the power of projective geometry can be used with the high performance rendering of computer graphics hardware to produce real-time 3D computer vision algorithms. Unfortunately the cross-over from the two fields is not even, as someone very familiar with projective geometry could more easily understand the

concepts in computer graphics than vice versa. Therefore there is an argument to say, why not treat everything in terms of projective geometry? A possible counter argument is that computer graphics hardware is necessary to implement real-time algorithms, therefore it is necessary to approach the solution to the problem with both disciplines in mind. Computer graphics hardware has been created for computer gamers or gaming consoles therefore it is unlikely that any mainstream graphics hardware will ever be developed for hardware computer vision processing. However, many realistic gaming effects, such as reflection off metallic surfaces, use environment maps and other filtering techniques common in computer vision. Therefore there will be some limited hardware implementations of these algorithms. If the functionality exists commonly in hardware it makes sense to use it to achieve real-time speeds in algorithm implementations.

The result of consolidating and analyzing this convergence between computer vision and computer graphics has lead to the development of a novel multiscale block tracking algorithm which tracks targets when there are large changes in scale while panning and tilting. The idea of using a quadrilateral block which is warped into the current view based on the camera parameters was introduced. Two methods of comparing this warped reference block to normal image data were presented and evaluated. As a comparison several conventional block trackers were implemented, which use a variety of block updating methods and position filtering/predication. The novel multiscale trackers consistently outperformed conventional block tracking methods in both synthetic and real-world sequences. The most accurate tracking method developed, called the *Best NCC Tracker*, uses conventional normalised cross-correlation to match blocks while predicting the target position with a Kalman filter and updating the reference block by Kalman filtering each of the pixels in the reference block. The target trajectory tracked by *Best NCC Tracker* was closest to the hand-tracked ground truth for most of the sequences. The success of the Kalman tracker appears to be highly dependent on tracking a feature such as a corner which is partially invariant to scale change and rotation. When tracking a feature that changes dramatically with increasing zoom (as in the `bottle1` sequence) it fails to track the correct target. However, *Multiscale Method 1* performed consistently and very accurately over a large variety of camera pans, tilts and zooms – tracking was successful even with a scale change of $10\times$ and with low frame rates. All the block tracking methods failed to track the sequence `tom1` where the target is partially occluded while moving between parked cars. A higher frame rate would possibly allow more of the background to be included in the reference block. The low frame rate of the sequence could also be seen to be irrelevant, as although the

occlusion is long in real-time, it is equivalent to a short occlusion at a higher frame rate. By using the *contour tracker* introduced in Chapter 5, it was possible to track the target through the occlusion.

In summary, the novel multiscale block tracking methods tracked most of the sequences successfully with large changes in scale and while panning and tilting. A successful method of updating the reference block was also presented to reflect changes in the target during tracking. All algorithms were implemented in such a way that they could take advantage of accelerated computer graphics hardware to texture map portions of the image between camera views and the reference view. The performance of the multiscale methods and the best conventional block matching are comparable. *Multiscale Method 1* performs well on all sequences but lacks fast performance. The *Best NCC Tracker* with Kalman block updating tracks well for most sequences, but then fails completely on others. However, it can be implemented to run in near real-time, which is attractive for online systems. These methods illustrate the variety in tracking strategies and the choice of when to use each method is largely dependent on the type of application, the accuracy of the track needed and the image sequence.

The idea of an offline tracking contour was developed while investigating many methods of multiscale tracking. When a sequence is tracked offline, most approaches do not take advantage of the fact that all frames are available. Hence, a user can specify a start and end position, and the tracker can constrain the search for the targets' trajectory to be between those two points. Initially, a system of calculating the match forces based on a force field created from match values was introduced. A tracking contour was fitted to two sequences taken with fixed view cameras with the intention of demonstrating applications of the tracking contour in media post-production. The contour fit could be used to obtain the interframe position of the target which is useful for re-timing a sequence or performing a match-move and adding rendered graphics to the sequence which interact correctly with existing objects in the scene. The *contour tracker* produced a track which is marginally better than a Kalman position filtered NCC block tracker. One of the disadvantages of the *contour tracker* for fixed view cameras is how to determine the values of the kinematic model parameters. Alternative kinematic models were discussed briefly, however, an area of future work is the automatic calculation of the contour parameters needed to give a good fit to the target trajectory.

The *contour tracker* was then extended to work under pan, tilt and zoom conditions. The method was simplified slightly and the complex method of calculating match forces was dis-

carded in favour of a method which fits a piecewise cubic spline to partial tracking data by minimising the distance between the contour position and the match data. This method is considerably faster than the fixed view contour tracker and settles in a minute rather than several hours. The tracking contour produces good results on sequences with pan, tilt and zoom. In the sequence `tom1`, the tracking contour successfully tracks when the target is partially occluded for more than half of the sequence – all of the other block tracking methods fail to track the target. As the frame rate of the sequences is low, the target's motion is jittery which is reflected in the RMS pixel error values, which are between one and two pixels. Given a higher frame rate sequence this figure would be much lower. However, the tracking contour is easily editable as it contains five control points. Therefore, in post production applications the contour could be quickly edited by a human operator until the result was visually acceptable.

The contour tracker has been developed in such a way that it could be integrated into a post production workflow and implemented as a plug-in for a compositing package such as Shake[162]. All of the parameters would be implemented as sliders in a GUI, however the algorithm would need to be optimised so that it would run in less than a minute.

One of the limitations of the contour tracker is that it can only operate on offline sequences as it requires all of the frames in the sequence to be available at once. Even though this excludes it from any sort of real-time tracking application, it makes it desirable for applications in post production and offline analysis of target trajectories in surveillance.

In contrast to previous zoom tracking work, the methods in this thesis use the pan, tilt and zooming parameters from the camera in a feed-forward manner. Other methods[4, 5, 19, 20] estimate the homographies between views while matching and combine these with calibration. The approach in this thesis has been to first calibrate the camera and then use the pan/tilt motor positions and the zoom lens position to create a model of the camera. A cookbook-type method of self-calibration for a PTZ camera was presented. The technique is based mainly on work by Sinha[28, 117], however it analyzes and consolidates work from many sources and explicitly brings together everything needed to calibrate and model a PTZ camera without omitting steps. Details of the analytical Jacobians for use in Levenberg-Marquardt minimisation, which are often left to the reader to determine themselves or simply omitted are included in Appendix C. Self-calibration was chosen as it is difficult to calibrate a PTZ camera with a calibration object and produce a calibration that is globally consistent over the entire zoom range. The most uncertain portion of the camera calibration is the estimation of the radial distortion and principal

point. Although clamping of the radial distortion parameters once they become highly uncertain is necessary, it results in unwanted spikes in other parameters such as the principal point to compensate. A solution to this may be to use only one radial distortion parameter, i.e. only κ_1 or possibly an alternative formulation of radial distortion. A simple improvement that would remove most of the error in the principal point estimation is to use a random path with various pans, tilts and zooms as a calibration sequence once the parameters had been determined using the methods in Chapter 6. It may then be possible to allow the radial distortion parameters to be un-clamped, however the effectiveness of this method is unknown and impossible to predict. A full camera zoom model was developed and evaluated by creating panoramic mosaics of several camera sequences. The mosaics are registered to within a pixel with variances of a pixel, which is sufficiently accurate for tracking purposes.

In conclusion, several previous methods of tracking under pan, tilt and zoom conditions have been investigated as well as the background to tracking in general. Two new multiscale block tracking methods have been presented as well as the idea of contour tracking. A complete method for calibrating a PTZ camera was analyzed and discussed. This was used to track several synthetic and real sequences. Most of the algorithms implemented tracked the sequences successfully with the most accurate results being obtained by the offline contour tracker. The possibilities of implementing the algorithms using accelerated video hardware are discussed later in Section 8.3 to produce a real-time pan/tilt/zoom block tracking system. This thesis has been successful in examining and investigating various methods of producing a pan/tilt/zoom tracking system which could be implemented on a desktop computer.

8.3 Future Work

One of the most significant areas where more research into the work presented in this thesis can be done is the capturing of sequences at full frame rate with cameras such as the Sony EVI-D31. This would mean that the camera parameters would significantly lag the actual camera parameters due to slow transfer rates across the RS-232 serial link. An estimate of the true camera parameters could be made using the self-calibration techniques discussed in Chapter 6. The actual camera parameters could be used to estimate the error in the camera model prediction using the lagging camera telemetry. For slow changes in the camera parameters, such as smooth zooming, the lag will have little effect. However, for rapid step changes there may be significant errors that cause the search window to be placed in the incorrect place or the reference block will

be transformed to the incorrect scale. The sensitivity of the tracking algorithm to this needs to be determined through a mixture of synthetic and real sequences. If the algorithms prove to be too sensitive to lagging camera parameters then it would be necessary to choose hardware that can provide full frame rate telemetry. Many of the new Sony cameras have Firewire interfaces to transfer digital video and it may become possible to control the camera with the high-speed Firewire connection rather than a slow RS-232 link.

8.3.1 Multiscale tracking

Both novel multiscale tracking methods presented in this thesis assume that the reference block is located on a plane at a fixed depth along the direction of gaze. This is not strictly correct and causes effects such as smudging and blurring. The result is unwanted drift in the reference block. A solution to this is to represent the corners of the reference block in world co-ordinates and predict the depth of each corner using a Kalman filter. It is not possible to determine the depth of the reference block as the camera is only panning, tilting and zooming. However, apparent changes in the position of the vertices of the reference block could be modelled as changes due to motion or changes due to depth.

Drift in the reference block in Method 2 could be detected by comparing reference blocks temporally or performing by optical flow. If motion in the reference block is detected this could be used to select another match candidate or remove the drift. The obvious sources of the drift in the reference block are the over-smoothing of the reference block when a small block, say 8×8 pixels is texture mapped to a large block of around 80×80 pixels. Bi-linear interpolation was used during texture mapping, however another method, such as bi-cubic interpolation could produce more accurate results. There is a tradeoff between interpolating the data to produce a smooth reference block and over-smoothing the data. The sensitivities to smoothing need to be investigated further.

Particle filters[150, 151] could be used instead of Kalman filters to model complex behaviour of targets. In a practical tracking system this may be of great importance. Other heuristic cues, such as whether the target is a person or a vehicle, could be used to predict more accurately how fast the target will move or how erratic its motion is likely to be. Work by Rosales[62] would be a good starting point for classifying the motion of humans, such as walking, running or rollerblading.

Most of the sequences presented were fairly short, however for long sequences it may be necessary to adjust the reference view so that the reference block is always stored with optimal resolution and rotation. For example, if a target is initially acquired at one extreme of the camera's pan range and moves on the other extreme of the pan range, the reference block will be highly distorted. However, if the reference view is switched to the other extreme pan position and the existing reference block is warping into this view, future block updates will improve the reference block and it will be more likely to accurately represent the target. A possible measure of the degradation of the reference block can be obtained by calculating the vertical/horizontal aspect ratio. Ideally a reference block should be square. If it is too narrow there will be excess distortion in certain views.

8.3.2 Hardware acceleration

The ATI Radeon 9500[1] supports hardware acceleration of image processing algorithms. An example given in the SDK is applying various convolution filters such as blurring, sharpening and edge detection. By using the OpenGL method `glReadPixels` it is possible to gain access to the frame buffer. Hence, any high quality rendering and texture mapping done in hardware can be accessed by copying it from the frame buffer into normal memory. It is necessary to first consult the graphics adapters' documentation to determine how this function has been implemented. For some graphics cards, it is possible to use an API provided by the manufacturer to transfer the video memory to conventional memory.

8.3.3 Synthetic Sequences

Povray[40] was used to ray-trace many of the synthetic sequences. It would be possible to use Povray in a closed loop system where frames are rendered depending on the required view from a synthetic camera. This could be used to simulate a tracking algorithm that pans and tilts the camera to keep a target in the centre of the frame. Povray would render the scene with the new position of the target, based on some animation model, given the new camera view. As each frame takes several seconds to render the system would not run in real-time, however a simulation of a real-world tracking system could be developed. This method would eliminate the dependence of algorithms on specific quirks of the hardware as the camera would

[1] More information is available online from `http://www.ati.com`

be simulated entirely in software as if it were running online.

All of the synthetic sequences were rendered using static objects which did not deform or move. The model of the man in the sequence `man1` could be animated to produce a more life-like model of a person. Many applications such as MilkShape3D[2] can be used to add 'skeletons' to 3D models and animate them to produce convincing and lifelike movement.

8.3.4 Contour Tracking

One of the major disadvantages of the offline tracking contour for fixed view cameras is speed. As the match values for a large portion of the image are needed it is necessary to perform cross-correlation of the reference block with most of the image. This is extremely computationally intensive. A method of improving this could be to only calculate the match values for areas of the image which are moving. A motion segmentation would limit the areas where the match forces need to be calculated. A bonus to this could be that areas of high false match could be removed, as although they are areas of the image which are similar to the target they are not moving and are hence excluded.

The main application of the offline tracking contour is media post production. There is a series of standard workflow packages which most post production houses use and the contour tracker could be implemented as a plug-in to one of these packages. It would need to be optimised to run faster and allow on-the-fly changes to parameters. Further work also needs to be done in estimating the values for the contour parameters – such as air resistance and the mass of each control point.

Alternative kinematic models for the contour tracker could be used such as a series of point masses connected by ideal springs. The spring constants of each spring would determine the ease with which the contour would move and how much damping the system would experience. As an addition to this, a 'shock-absorber' could also be added to control the amount of damping. The contour can also be considered to be a continuous elastic string with constant cross-sectional area and density. The match forces in each frame would act directly on the contour. The total force on the contour could be calculated by integrating the forces along the contour. The displacement of the contour in each frame could then be calculated and the current method of moving the contour could be used. This method is analogous to restricting the

[2] Available online from `http://www.swissquake.ch/chumbalum-soft`

curvature of the contour and the rate of change of curvature.

The complexity of the target's trajectory will determine the number of control points needed to represent the contour. Too few control points will result in an over-smoothed contour and too many points will cause the contour to over-fit to the position data. A sensitivity analysis of the balance of control points to the error in the reconstruction of the trajectory needs to be investigated. Also, when the target has many step changes in velocity, it may be necessary to break the contour up into several smaller contours, which are discontinuous at the step changes. This would allow the target to have right angle changes in its trajectory for example. Several fitting passes could be done while dividing the contour until the optimal number of contours needed to fit the trajectory well is obtained.

The contour tracker is implemented using an absolute measure of the match confidence to generate the match forces. Instead, the contour position could be moved to maximise the likelihood of the contour position representing the true object trajectory. This could reduce the amount of computation required as the match forces would not need to be calculated. The metric used in the maximum likelihood estimation would instead be a mixture of spatial and temporal match information that would have similar results to finding the match forces, i.e. areas of good match would reinforce the contour position and areas of poor match would drive the solution away. The spatio-temporal distribution of the match values would need to be analyzed and possibly preconditioned to allow the contour to solve to the local minimum by 'smoothing' the match value fields.

Strategies of updating the reference block while contour tracking were only briefly investigated. Since the reference block is known at the beginning and end of the sequence it should be possible to use this to estimate the change in the reference block at any point in the sequence. As more matches become available, the true values of the pixel intensities in the reference block can be incorporated to produce a more accurate estimate of the reference block for unmatched frames. It could be possible to use an interpolation method where temporal curves are fitted to each pixel in the reference block. Another possibility is to use a Kalman filter approach. The camera parameters could also be included to make the reference block independent of changes in the zoom factor. Therefore, the only changes being modelled would be changes in the target.

8.4 Final Remarks

Computer technology has become an accepted part of modern life and associated with this is the wider proliferation of electronic surveillance and digitally produced entertainment. The methods and analysis presented in this thesis will contribute to the greater understanding of tracking methods in a small set of situations. This will hopefully lead to creating safer environments for everyone to live and play in while being enthralled by the creative genius of others made concrete by advanced vision and graphics technology.

References

[1] "Boujou." 2D3's Boujou match-moving package – available online from `http://www.boujou.com` and `http://www.2d3.com`.

[2] N. Owens, C. Harris, and C. Stennett, "Hawk-eye tennis system," in *International Conference on Visual Information Engineering, VIE 2003*, pp. 182–185, 2003.

[3] I. Haritaoglu, D. Harwood, and L. Davis, "W4: Who? when? where? what? a real time system for detecting and tracking people," in *Proceedings of International Conference on Automatic Face and Gesture Recognition, FG'00*, pp. 222 – 227, 1998.

[4] B. Tordoff, *Active Control of Zoom for Computer Vision*. PhD thesis, Robotics Research Group, Department of Engineering, University of Oxford, 2002.

[5] E. Hayman, I. Reid, and D. Murray, "Zooming while tracking using affine transfer," in *Proceeding of the British Machine Vision Conference*, vol. 2, pp. 395 – 400, 1996.

[6] E. Hayman, D. Murray, and T. Thorhallsson, "Zoom-invariant tracking using points and lines in affine views: An application of the affine multifocal tensors," in *Proceedings of the Internation Conference on Computer Vision*, pp. 269 – 277, 1999.

[7] E. Hayman, T. Thorhallsson, and D. Murray, "Tracking while zooming using affine transfer and multifocal tensors," *International Journal of Computer Vision*, vol. 51, no. 1, pp. 1 – 31, 2003.

[8] B. Tordoff and D. Murray, "Reactive zoom control while tracking," Tech. Rep. OUEL2228/00, Department of Engineering Science, Oxford University, 2003.

[9] B. Tordoff and D. Murray, "Reactive zoom control while tracking using an affine camera," in *Proceedings of the British Machine Vision Conference*, 2001.

[10] G. Xu and T. Sugimoto, "Rits eye: A software-baed system for relatime face detection and tracking using pan-tilt-zoom controllable camera," in *Proceedings of the 14th International Conference on Pattern Recognition*, pp. 1194 – 1997, 1998.

[11] C. Wren, A. Azerbayejani, T. Darrell, and A. Pentland, "Pfinder: real-time tracking of the human body," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 7, pp. 780 – 785, 1997.

[12] J. K. Aggarwal and Q. Cai, "Human motion analysis: A review," *Computer Vision and Image Understanding*, vol. 73, no. 3, pp. 428–440, 1999.

[13] Q. Cai and J. Aggarwal, "Tracking human motion in structured environments using a distributed-camera system," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 21, no. 12, pp. 1241 – 1247, 1999.

[14] Q. Cai, *Tracking Human Motion in Indoor Environments Using a Distributed-Camera System*. PhD thesis, University of Texas at Austin, 1997.

[15] R. Collins, A. Lipton, T. Kanadeand, H. Fujiyoshi, D. Duggins, and Y. Tsin, "A system for video surveillance and monitoring: VSAM final report.," Tech. Rep. CMU-RI-TR-00-12, Carnegie Mellon University, 2003.

[16] K. Dawson-Howe, "Active surveillance using dynamic background subtraction," Tech. Rep. TCD-CS-96-06, Computer Science Department, Trinity College Dublin, 1996.

[17] S. Dockstader and A. Tekalp, "Multiple camera tracking of interacting and occluded human motion," *Proceedings of the IEEE*, vol. 89, pp. 1441–1455, October 2001.

[18] S. Stillman, R. Tanawongsuwan, and I. Essa, "A system for tracking and recognizing multiple people with multiple cameras," Tech. Rep. GIT-GVU-98-25, Graphics, Visualization and Usability Center, Georgia Institute of Technology, 2003.

[19] J. Fayman, O. Sudarsky, and E. Rivlin, "Zoom tracking and its applications," in *Proceedings of the IEEE Conference on Robotics and Automation*, 1998.

[20] R. Bajcsy and U. Cahn von Seelen, "Adaptive correlation tracking of targets with changing scale," Tech. Rep. MS-CIS-96-22/GRASP LAB 405, GRASP Laboratory of Robotics Research and Education, University of Pennsylvania, 1996.

[21] "VXL." C++ Computer Vision Libraries. Available online at `http://vxl.sourceforge.net/`.

[22] "OpenCV – Open Computer Vision Library." Available online at `http://sourceforge.net/projects/opencvlibrary/`.

[23] J. Neider, T. Davis, and M. Woo, *OpenGL Programming Guide – The Official Guide to Learning OpenGL, Release 1*. Reading, MA, USA: Addison-Wesley, 1993.

[24] K. Gray, *DirectX9 Programmable Graphics Pipeline*. Redmond, WA: Microsoft Press, 2003.

[25] M. Pollefeys, "3D modeling from images," in *Siggraph 2002/2001/2000, 3DIM 2001/2003, ECCV 2000*, 2000–2003. Also available online at `http://www.cs.unc.edu/marc/tutorial/`.

[26] M. Pollefeys, *Self-calibration and metric 3D reconstruction from uncalibrated image sequences*. PhD thesis, K.U.Leuven, 1999.

[27] M. Pollefeys, M. Vergauwen, K. Cornelis, J. Tops, F. Verbiest, and L. V. Gool, "Structure and motion from image sequences," in *Conference on Optical 3-D Measurement Techniques*, pp. 251–258, Oct 2001.

[28] S. N. Sinha and M. Pollefeys, "Towards calibration a pan-tilt-zoom camera network," in *OMNIVIS 2004, workshop on Omnidirectional Vision and Camera Networks held in conjunction with ECCV 2004*, 2004.

[29] L. de Agapito, E. Hayman, and I. Reid, "Self-calibration of rotating and zooming cameras," *International Journal of Computer Vision*, vol. 45, pp. 107–127, 2001.

[30] R. Hartley, "Self calibration of stationary cameras," *International Journal of Computer Vision*, vol. 22, no. 1, pp. 5 – 23, 1997.

[31] O. D. Faugeras and Q.-T. Luong, *The Geometry of Multiple Images*. Cambridge, Massachusetts: MIT Press, 2001.

[32] P. E. Debevec, *Modeling and Rendering Architecture from Photographs*. PhD thesis, University of California at Berkeley, 1996.

[33] P. E. Debevec, "Rendering synthetic objects into real scenes: Bridging traditional and image-based graphics with global illumination and high dynamic range photography," in *Proceedings of SIGGRAPH 98, Computer Graphics Proceedings, Annual Conference Series*, pp. 189–198, July 1998.

[34] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. Cambridge Univeristy Press, 2nd ed., 2003.

[35] J. Foley, A. van Dam, S. Feiner, and J. Hughes, *Computer Graphics: Principles and Practice, Second Edition*. Addison Wesley, 1990.

[36] J. Foley, A. van Dam, S. Feiner, J. Hughes, and R. Phillips, *Introduction to Computer Graphics*. Addison Wesley, 1994.

[37] K. Bradshaw, P. McLauchlan, and I. Reid, "Saccade and persuit on an active head/eye platform," *Image and Vision Computing*, vol. 12, no. 3, pp. 155 – 163, 1994.

[38] D. Coombs and C. Brown, "Real-time smooth pursuit tracking for a moving binocular head," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 23 – 28, 1992.

[39] M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: Active contour models," *International Journal of Computer Vision*, vol. 1, pp. 321–331, 1998.

[40] "Povray – Persistence of Vision Raytracer." Available online at `http://www.povray.org/`.

[41] R. Tsai, "A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses," *IEEE Journal of Robotics and Automation*, vol. 3, no. 4, pp. 323 – 344, 1987.

[42] Z. Zhang, "A flexible new technique for camera calibration," Tech. Rep. MSR-TR-98-71, Microsoft Research, December 1998.

[43] B. Triggs, P. McLauchlan, R. Hartley, and A. Fitzgibbon, "Bundle adjustment – A modern synthesis," in *Vision Algorithms: Theory and Practice* (W. Triggs, A. Zisserman, and R. Szeliski, eds.), vol. 1883 of *Lecture Notes In Computer Science*, pp. 298–375, Springer Verlag, 2000.

[44] Intel, "Intel Image Processing Libraries v2.5." `http://developer.intel.com/software/products/perflib/ipl/` – Intel no longer supports IPL and now produces the Intel Integrated Performance Primitives available on the Internet at `http://www.intel.com/cd/software/products/asmo-na/eng/perflib/ipp`.

[45] M. Pollefeys, R. Koch, M. Vergauwen, and L. V. Gool, "Hand-held acquisition of 3D models with a video camera," in *Second International Conference on 3-D Digital Imaging and Modeling*, pp. 14–23, 1999.

[46] M. Pollefeys, M. Vergauwen, K. Cornelis, F. Verbiest, J. Schouteden, J. Tops, and L. V. Gool, "3D acquisition of archaeological heritage from images," in *VAST2001 (Virtual Reality, Archaeology, and Cultural Heritage)* (A. Press, ed.), pp. 255–262, 2001.

[47] S. Vallance and P. Calder, "Inward looking projections," in *Proceedings of the 1st international conference on Computer graphics and interactive techniques in Australasia and South East Asia, GRAPHITE '03*, (New York, NY, USA), pp. 219–222, ACM Press, 2003.

[48] S. E. Chen, "QuickTime VR — an image-based approach to virtual environment navigation," *Computer Graphics*, vol. 29, no. Annual Conference Series, pp. 29–38, 1995.

[49] Y. Xiong and K. Turkowski, "Registration, calibration and blending in creating high quality panoramas," in *Proceedings of the Fourth IEEE Workshop on Applications of Computer Vision, WACV '98*, pp. 69–74, 1998.

[50] I. Haritaoglu, D. Harwood, and L. Davis, "Hydra: Multiple people detection and tracking using silhoettes," in *Proceedings of the Workshop on Visual Surveillance*, pp. 6 – 13, 1999.

[51] S. Gong and T. Chang, "Tracking multiple people with a multi-camera system," in *Proceedings of IEEE Workshop on Multi-Object Tracking*, pp. 19 – 26, 2001.

[52] A. Bobick, J. Davis, and S. Intille, "The KidsRoom: An example application using a deep perceptual interface," in *Workshop on Perceptual User Interfaces*, 1997.

[53] D. Hogg, "Model-based vision: a program to see a walking person," *Image and Vision Computing*, vol. 1, no. 1, pp. 5 – 20, 1983.

[54] I. Essa, "Computers seeing people," *AI Magazine*, vol. 20, no. 1, pp. 69 – 82, 1999.

[55] J. Casas, A. Sitjes, and P. Folch, "Mutual feedback scheme for face detection and tracking aimed at density estimation in demonstrations," *IEE Proceedings on Vision, Image and Signal Processing*, vol. 152, no. 3, pp. 334–346, 2005.

[56] Q. Huang, B. Dom, N. Megiddo, and W. Niblack, "Segmenting and representing background in color images," in *Proceedings of the 13th International Conference on Pattern Recognition*, vol. 3, pp. 13–17 vol.3, 1996.

[57] G. D. Hager and P. N. Belhumeur, "Efficient tegion tracking with parametric models of geometry and illumination," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, pp. 1025–1039, 1998.

[58] X. Desurmont, A. Bastide, C. Chaudy, C. Parisot, J. Delaigle, and B. Macq, "Image analysis architectures and techniques for intelligent surveillance systems," *IEE Proceedings on Vision, Image and Signal Processing*, vol. 152, no. 2, pp. 224–231, 2005.

[59] B. Maurin, O. Masoud, and N. Papanikolopoulos, "Tracking all traffic: computer vision algorithms for monitoring vehicles, individuals, and crowds," *IEEE Robotics & Automation Magazine*, vol. 12, no. 1, pp. 29–36, 2005.

[60] R. Cucchiara, C. Grana, A. Prati, and R. Vezzani, "Computer vision system for in-house video surveillance," *IEE Proceedings on Vision, Image and Signal Processing*, vol. 152, no. 2, pp. 242–249, 2005.

[61] M. Latzel, E. Darcourt, and J. Tsotsos, "People tracking using robust motion detection and estimation," in *Proceedings of the 2nd Canadian Conference on Computer and Robot Vision*, pp. 270–275, 2005.

[62] R. Rosales and S. Sclaroff, "Trajectory guided tracking and recognition of actions," Tech. Rep. 1999-002, Boston University, 9, 1999.

[63] Y. Raja, S. McKenna, and S. Gong, "Segmentation and tracking using colour mixture models," in *Proceeding of the Asian Conference on Computer Vision*, 1998.

[64] S. McKenna, Y. Raja, and S. Gong, "Tracking color objects using adaptive mixture models," *Image and Vision Computing*, vol. 17, no. 3, pp. 225 – 232, 1999.

[65] S. McKenna, Y. Raja, and S. Gong, "Object tracking using adaptive colour mixture models," in *Proceeding of the Asian Conference on Computer Vision*, 1998.

[66] G. Jones and P. Giaccone, "Hierarchical tracking of motion in multiple images," in *Proceeding of the IEE Colloquium on Multiresolution Modelling and Analysis in Image Processing and Computer Vision*, pp. 7/1–7/6, April 1995.

[67] C. Kuo, T. Wang, C. Kuo, and T. Wang, "A real-time segmentation scheme for continuous color images," in *Proceedings of the IEEE International Symposium on Circuits and Systems, ISCAS 2001*, vol. 2, pp. 385–388, May 2001.

[68] J. Hidalgo and P. Salembier, "Robust segmentation and representation of foreground key-regions in video sequences," in *Proceeding of the IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP'01*, 2001.

[69] M. Bichsel, "Segmenting simply connected moving-objects in a static scene," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, pp. 1138–1142, November 1994.

[70] F. Xu, X. Liu, and K. Fujimura, "Pedestrian detection and tracking with night vision," *IEEE Transactions on Intelligent Transportation Systems*, vol. 6, no. 1, pp. 63–71, 2005.

[71] F. Moscheni, F. Dufaux, and M. Kunt, "A new two-stage global/local motion estimation based on foreground/background segmentation," in *Proceeding of the IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP'95*, 1995.

[72] F. Dufaux, F. Moscheni, and A. Lippman, "Spatial-temporal segmentation based on motion and static segmentation," in *Proceeding of the IEEE Internation Conference on Image Processing, ICIP'95*, 1995.

[73] I. Kompatsiaris and M. Strintzis, "Spatiotemporal segmentation and tracking of objects in image sequences," in *Proceedings of International Conference on Image Processing, ICIP'99*, pp. 155 – 158, 1999.

[74] D. Forsyth, *Colour Constancy and its Applications in Machine Vision*. PhD thesis, University of Oxford, 1988.

[75] S. Kurt and J. Ramesh, "Illumination independent change detection for real world sequence," *Computer Vision, Graphics and Image Processing*, vol. 46, pp. 387 – 399, 1989.

[76] J. Kittler, R. Marik, and J. Matas, "Illumination invariant colour recognition," in *Proceeding of the British Machine Vision Conference*, pp. 469 – 479, 1994.

[77] M. Soriano, B. Martinkauppi, S. Huovinen, and M. Laaksonen, "Skin detection in video under changing illumination conditions," in *Proceedings of International Conference on Automatic Face and Gesture Recognition, FG'00*, pp. 839 – 842, 2000.

[78] H. Stern and B. Efros, "Adaptive color space switching for face tracking in multi-colored lighting environments," in *Proceeding of the 5th IEEE Internation Conference on Automatic Face and Gesture Recognition*, 2002.

[79] Y. Wu and T. Huang, "Nonstationary color tracking for vision-based human-computer interaction," *IEEE Transactions on Neural Networks*, vol. 13, no. 4, pp. 948–960, 2002.

[80] R. Bowden and P. KaewTraKulPong, "Towards automated wide area visual surveillance: tracking objects between spatially-separated, uncalibrated views," *IEE Proceedings on Vision, Image and Signal Processing*, vol. 152, no. 2, pp. 213–223, 2005.

[81] S. Khan, O. Javed, Z. Rasheed, and M. Shah, "Human tracking in multiple cameras," in *Proceedings of the Eighth IEEE International Conference on Computer Vision, ICCV 2001*, vol. 1, pp. 331–336, 2001.

[82] R. Cucchiara, C. Grana, A. Prati, and R. Vezzani, "Probabilistic posture classification for human-behavior analysis," *IEEE Transactions on Systems, Man and Cybernetics, Part A*, vol. 35, no. 1, pp. 42–54, 2005.

[83] C. Micheloni, G. Foresti, and L. Snidaro, "A network of co-operative cameras for visual surveillance," *IEE Proceedings on Vision, Image and Signal Processing*, vol. 152, no. 2, pp. 205–212, 2005.

[84] T. Komuro, I. Ishii, M. Ishikawa, and A. Yoshida, "A digital vision chip specialized for high-speed target tracking," *IEEE Transactions on Electron Devices*, vol. 50, no. 1, pp. 191–199, 2003.

[85] W. Kim, S. Kim, and J. Lee, "Efficient tracking of a moving object using optimal representative blocks," in *Proceedings of the IEEE/ASME International Conference on Advanced Intelligent Mechatronics, AIM 2003*, vol. 2, pp. 1417–1422 vol.2, 2003.

[86] R. Gonzales and R. Woods, *Digital Image Processing*. Addison-Wesley, 1993.

[87] S.-N. Lim, A. Elgammal, and L. Davis, "Image-based pan-tilt camera control in a multi-camera surveillance environment," in *Proceedings of the International Conference on Multimedia and Expo, ICME '03*, vol. 1, pp. 645–8, 2003.

[88] A. Elgammal, R. Duraiswami, D. Harwood, and L. S. Davis, "Background and foreground modeling using nonparametric kernel density for visual surveillance," *Proceedings of the IEEE*, vol. 90, pp. 1151–1163, July 2002.

[89] A. Senior, A. Hampapur, and M. Lu, "Acquiring multi-scale images by pan-tilt-zoom control and automatic multi-camera calibration," in *IEEE Workshop on Applications of Computer Vision*, pp. I: 433–438, 2005.

[90] D. Brown, "Close-range camera calibration," *Photogrammetric Engineering*, vol. 37, pp. 855–866, 1971.

[91] J. Weng, P. Cohen, and M. Herniou, "Camera calibration with distortion models and accuracy evaluation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 10, pp. 965–980, 1992.

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

[93] E. Kruppa, "Zur ermittlung eines objecktes aus zwei perspektiven mit innerer orientierung," *Sitzungsberichte der Mathematische Naturwissenschaftlich, Kaiser- licher Akademie der Wissenschaften (Wein), Abteilung IIa*, vol. 122, p. 19391948, 1913.

[94] P. Sturm, "A case against Kruppa's equations for camera self-calibration," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 10, pp. 1199–1204, 2000.

[95] R. Sturm, "Das problem der projektivitt und seine anwendung auf die flchen zweiten grades," *Math. Ann.*, vol. 1, pp. 533–574, 1869.

[96] S. J. Maybank and O. D. Faugeras, "A theory of self-calibration of a moving camera," *International Journal of Computer Vision*, vol. 8, no. 2, pp. 123–151, 1992.

[97] C. Zeller and O. Faugeras, "Camera self-calibration from video sequences: the Kruppa equations revisited," Tech. Rep. RR-2793, INRIA, France, 1996.

[98] R. Hartley, "Kruppa's equations derived from the fundamental matrix," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 2, pp. 133–135, 1997.

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

[100] J. Heikkila and O. Silven, "A four-step camera calibration procedure with implicit image correction," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR97*, (Washington, DC, USA), p. 1106, IEEE Computer Society, 1997.

[101] P. Sturm and S. Maybank, "On plane-based camera calibration: A general algorithm,singularities, applications," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR99*, vol. 1, pp. –437 Vol. 1, 1999.

[102] R. G. Willson, *Modeling and calibration of automated zoom lenses*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 1994.

[103] R. G. Willson and S. A. Shafer, "A perspective projection camera model for zoom lenses," Tech. Rep. CMU-RI-TR-94-03, Robotics Institute, Camegie Mellon University, Pittsburgh, PA, USA, January 1994.

[104] R. G. Willson and S. A. Shafer, "What is the Center of the Image?," Tech. Rep. CMU-CS-93-122, Carnegie Mellon University, April 1993.

[105] R. G. Willson and S. A. Shafer, "What is the center of the image," *Journal of Optical Society of America*, vol. 11, pp. 2946 – 2955, November 1994.

[106] P. Sturm, "Self-calibration of a moving camera by precalibration," in *British Machine Vision Conference*, pp. 675– 84, September 1996.

[107] P. Sturm, "Self-calibration of a moving zoom-lens camera by pre-calibration," *Image and Vision Computing*, vol. 15, pp. 583–589, August 1997.

[108] M. Pollefeys, R. Koch, and L. V. Gool, "Self-calibration and metric reconstruction in spite of varying and unknown internal camera parameters," *International Journal of Computer Vision*, vol. 32, no. 1, pp. 7–25, 1999.

[109] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *In Communications of the ACM*, vol. 24, pp. 381–395, 1981.

[110] A. Heyden and K. Astrom, "Euclidean reconstruction from image sequences with varying andunknown focal length and principal point," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR97*, pp. 438–443, 1997.

[111] B. Triggs, "Autocalibration and the absolute quadric," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR97*, pp. 609–614, 1997.

[112] L. de Agapito, E. Hayman, and I. Reid, "Self-calibration of a rotating camera with varying intrinsic parameters," in *British Machine Vision Conference*, 1998.

[113] L. de Agapito, R. Hartley, and E. Hayman, "Linear self-calibration of a rotating and zooming camera," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR99*, vol. 1, pp. –21 Vol. 1, 1999.

[114] R. Collins and Y. Tsi, "Calibration of an outdoor active camera system," Tech. Rep. CMU-RI-TR-98-36, The Robotics Institute, Carnegie Mellon University, 1998.

[115] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes: The Art of Scientific Computing*. Cambridge (UK) and New York: Cambridge University Press, 2nd ed., 1992.

[116] J. Davis and X. Chen, "Calibrating pan-tilt cameras in wide-area surveillance networks," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR03*, pp. 144–149 vol.1, 2003.

[117] S. N. Sinha, M. Pollefeys, and S. J. Kim, "High-resolution multiscale panoramic mosaics from pan-tilt-zoom cameras," in *Proceedings of the 4th Indian Conference on Computer Vision, Graphics and Image Processing*, 2004.

[118] Y. Lu, J. Zhang, Q. Wu, and Z.-N. Li, "A survey of motion-parallax-based 3-D reconstruction algorithms," *IEEE Transactions on Systems, Man and Cybernetics, Part C*, vol. 34, no. 4, pp. 532–548, 2004.

[119] C. Brown, "Gaze controls with interactions and decays," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 20, no. 2, pp. 518–527, 1990.

[120] S. Rougeaux and Y. Kuniyoshi, "Robust real-time tracking on an active vision head," in *Proceedings of IEEE/RAS International Conference on Intelligent Robots and Systems*, pp. 873–879, 1997.

[121] S. G. Goodridge and M. G. Kay, "Multimedia sensor fusion for intelligent camera control," in *Proceedings of the IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems*, pp. 655–662, 1996.

[122] I. Reid and D. Murray, "Active tracking of foveated feature clusters using affine structure," *International Journal of Computer Vision*, vol. 18, no. 1, pp. 41 – 60, 1996.

[123] Y. Raja, S. McKenna, and S. Gong, "Tracking and segmenting people in varying lighting conditions using colour," in *Proceedings of International Conference on Automatic Face and Gesture Recognition, FG98*, 1998.

[124] J. You, W. Lu, J. Li, G. Gindi, and Z. Liang, "Image matching for translation, rotation and uniform scaling by the Radon transform," in *Proceedings of the International Conference on Image Processing, ICIP'98*, vol. 1, pp. 847 – 851, October 1998.

[125] S. Yoshimura and T. Kanade, "Fast template matching based on the normalized correlation by using multiresolution eigenimages," in *Proceedings of International Conference on Intelligent Robots and Systems*, pp. 2086 – 2093, 1994.

[126] S. Smith and J. Brady, "SUSAN – a new approach to low level image processing," *International Journal of Computer Vision*, vol. 23, pp. 45–78, May 1997.

[127] P. Torr and A. Zisserman, "MLESAC: A new robust estimator with application to estimating image geometry," *Computer Vision and Understanding*, vol. 78, pp. 138–156, 2000.

[128] D. Comaniciu, V. Ramesh, and P. Meer, "Kernel-based object tracking," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 5, pp. 564–577, 2003.

[129] D. Comaniciu and P. Meer, "Mean shift: a robust approach toward feature space analysis," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 5, pp. 603–619, 2002.

[130] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L. V. Gool, "A comparison of affine region detectors," *International Journal of Computer Vision*, vol. 65, no. 1/2, pp. 43–72, 2005.

[131] M. J. Clarkson, *Registration of Optical Images to 3D Medical Images*. PhD thesis, Computational Imaging Sciences Group, Division of Radiological Sciences and Medical Engineering, University of London, 2000.

[132] R. Stolkin, A. Greig, and J. Gilby, "Video with ground-truth for validation of visual registration, tracking and navigation algorithms," in *Proceedings of the 2nd Canadian Conference on Computer and Robot Vision*, pp. 210–217, 2005.

[133] L. Vacchetti, V. Lepetit, and P. Fua, "Stable real-time 3D tracking using online and offline information," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 10, pp. 1385–1391, 2004.

[134] M. Xu, J. Orwell, L. Lowey, and D. Thirde, "Architecture and algorithms for tracking football players with multiple cameras," *IEE Proceedings on Vision, Image and Signal Processing*, vol. 152, no. 2, pp. 232–241, 2005.

[135] F. Borsato and F. Flores, "A real time method to object detection and tracking applied to robot-soccer," in *Proceedings of the IEEE Conference on Cybernetics and Intelligent Systems*, vol. 1, pp. 174–178 vol.1, 2004.

[136] G. Scotti, L. Marcenaro, C. Coelho, F. Selvaggi, and C. Regazzoni, "Dual camera intelligent sensor for high definition 360 degrees surveillance," *IEE Proceedings on Vision, Image and Signal Processing*, vol. 152, no. 2, pp. 250–257, 2005.

[137] R. Canals, A. Roussel, J.-L. Famechon, and S. Treuillet, "A biprocessor-oriented vision-based target tracking system," *IEEE Transactions on Industrial Electronics*, vol. 49, no. 2, pp. 500–506, 2002.

[138] A. H. Watt, *3D Computer Graphics*. Harlow: Addison-Wesley, 3rd ed., 2000.

[139] F. S. Hill, *Computer graphics : using OpenGL*. Upper Saddle River, NJ: Prentice Hall, 2nd ed., 2001.

[140] W. Schroeder, K. Martin, and B. Lorensen, *The visualization toolkit : an object-oriented approach to 3D graphics*. Upper Saddle River, N.J: Prentice Hall, 1996.

[141] E. W. Weisstein, *Frustum*. From MathWorld–A Wolfram Web Resource. `http://mathworld.wolfram.com/Frustum.html`, 1999-2005.

[142] G. Xu and N. Sugimoto, "Algebraic derivation of the Kruppa equations and a new algorithm for self-calibration of cameras," *Journal of the Optical Society of America A*, vol. 16, pp. 2419–2424, October 1999.

[143] J. Shi and C. Tomasi, "Good features to track," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR'94*, pp. 593–600, 1994.

[144] C. Haworth, A. Peacock, and D. Renshaw, "Performance of reference block updating techniques when tracking with the block matching algorithm," in *Proceedings of the International Conference on Image Processing*, vol. 1, pp. 365–368, 2001.

[145] R. Hunt, *The Reproduction of Colour*. England: Fountain Press, 5th ed., 1995.

[146] C. Tomasi and T. Kanade, "Detection and tracking of point features," Tech. Rep. CMU-CS-91-132, Carnegie Mellon University, April 1991.

[147] M. Efe, D. Atherton, and J. Bather, "Adaptive kalman filters for manoeuvering target tracking," in *IEE Colloquium on Target Tracking and Data Fusion*, pp. 4/1–4/7, 9 June 1998.

[148] E. Brookner, *Tracking and Kalman Filtering Made Easy*. New York: John Wiley and Sons, 1998.

[149] S. M. Bozic, *Digital and Kalman Filtering*. London: Edward Arnold, 1979.

[150] C. Hue, J. L. Cadre, and P. Perez, "Tracking multiple objects with particle filtering," tech. rep., IRISA, October 2000.

[151] S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, "A tutorial on particle filters for on-line non-linear/non-gaussian bayesian tracking," *IEEE Transactions on Signal Processing*, vol. 50, pp. 174–188, February 2002.

[152] E. Loutas, I. Pitas, and C. Nikou, "Entropy-based metrics for the analysis of partial and total occlusion in video object tracking," *IEE Proceedings on Vision, Image and Signal Processing*, vol. 151, no. 6, pp. 487–497, 2004.

[153] E. W. Weisstein, *Delaunay Triangulation*. From MathWorld–A Wolfram Web Resource. `http://mathworld.wolfram.com/DelaunayTriangulation.html`, 1999-2005.

[154] R. Rosales and S. Sclaroff, "Improved tracking of multiple humans with trajectory prediction and occlusion modeling.," in *Proceeding of the IEEE International Conference on Computer Vision and Pattern Recognition, ICVPR'98*, 2003.

[155] D. Gibbins, G. Newsam, and M. Brooks, "Detecting suspicious background changes in video surveillance of busy scenes," in *Proceeding of the 3rd IEEE Workshop on Applications of Computer Vision*, pp. 22 – 26, 1996.

[156] F. Moscheni, F. Dufaux, and M. Kunt, "Object tracking based on temporal and spatial information," in *Proceeding of the IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP'96*, vol. 4, pp. 1914 – 1917, 1996.

[157] S. Lobregt and M. A. Viergever, "A discrete dynamic contour model," *IEEE Transactions on Medical Images*, vol. 14, pp. 12–24, March 1995.

[158] E. W. Weisstein, *Cubic Spline*. From MathWorld–A Wolfram Web Resource. `http://mathworld.wolfram.com/CubicSpline.html`, 1999-2005.

[159] R. H. Bartels, J. C. Beatty, and B. A. Barsky, *An Introduction to Splines for Use in Computer Graphics and Geometric Modelling*. San Francisco, CA: Morgan Kaufmann, 1998.

[160] N. Malpica, A. Santos, E. Perez, M. Garcia-Fernandez, and M.Desco, "A snake model for anatomic M-mode tracking in echocardiograhy," in *Proceedings of the 3rd International Symposium on Image and Signal Processing and Analysis*, vol. 2, pp. 722–726, 18-20 September 2003.

[161] Q. Zhou, L. Ma, D. Chelberg, and M. Celerik, "Color and texture priors in active contours for model-based image segmentation," in *Proceedings of the 3rd International Symposium on Image and Signal Processing and Analysis*, vol. 2, pp. 690–695, 18-20 September 2003.

[162] "Shake." Apple Computers, Inc – `http://www.apple.com/shake/`.

[163] D. C. Brown, "A solution to the general problem of multiple station analytical stereotriangulation," Tech. Rep. 43 (or AFMTC TR 58-8), Patrick Airforce Base, Florida, 1958.

[164] C. Harris and M. Stephens, "A combined corner and edge detector," in *Alvey Vision Conference*, pp. 147–151, 1988.

[165] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[166] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the International Conference on Computer Vision ICCV'99, Corfu*, pp. 1150–1157, 1999.

[167] G. H. Golub and C. F. V. Loan, *Matrix Computations*. Baltimore, MD, USA: The Johns Hopkins University Press, second ed., 1989.

[168] S. M. LaValle, *Planning Algorithms*. Cambridge University Press (also available at http://msl.cs.uiuc.edu/planning/), 2006. To be published in 2006.

[169] J. J. Dongarra, C. B. Moler, J. R. Bunch, and G. W. Stewart, *LINPACK Users' Guide*. Philadelphia, USA: Society for Industrial and Applied Mathematics, 1979. Also available online at `http://www.netlib.org/linpack/`.

[170] M. Sinclair, P. Hillman, J. Hannah, and D. Renshaw, "Active contours for best fit interpolation of object tracks for motion picture post production," in *Proceedings of the 1st European Conference on Visual Media Production CVMP 2004*, pp. 39 – 48, 2004.

Appendix A
Results

This appendix contains the full results of the tracking methods described in Chapter 7. All of the graphs are presented in large format for ease of reading. Results for the block tracking methods can be found in Section A.1 below and results for the contour tracker can be found in Section A.2 on page 283. A full explanation of the methods and implementations can be found in the main body of the thesis in Chapter 7.

A.1 Block Tracking

The following section presents all of the tracking results from Section 7.4 on page 189. A discussion of these results can be found in Section 7.4.3 on page 204.

A.1.1 Sequence – `image1`

Figure A.6 and Figure A.1 show the tracking results on a composite of the sequence projected onto the front cube face in world co-ordinates. The position of the centre of the image is shown in Figure A.3 and the confidence of the match (or match value) as a percentage is shown in Figure A.5. The reference block for each multiscale tracker with block updating is shown in Figure A.2 at various stages during the track. A summary of the RMS pixel tracking errors is shown in the table below:

	Position Filtered NCC		Best NCC		Method 1		Method 2	
No block updating	0.01*	(2)	0.01*	(2)	2.9	(101)	3.5	(101)
FIR block updating	132.9†	(101)	–		3.6	(101)	18.1†	(101)
Kalman updating	32.8†	(101)	32.8†	(101)	–		–	

† Track fails * Partial track

(See Section 7.2 pg 170 for explanation of table)

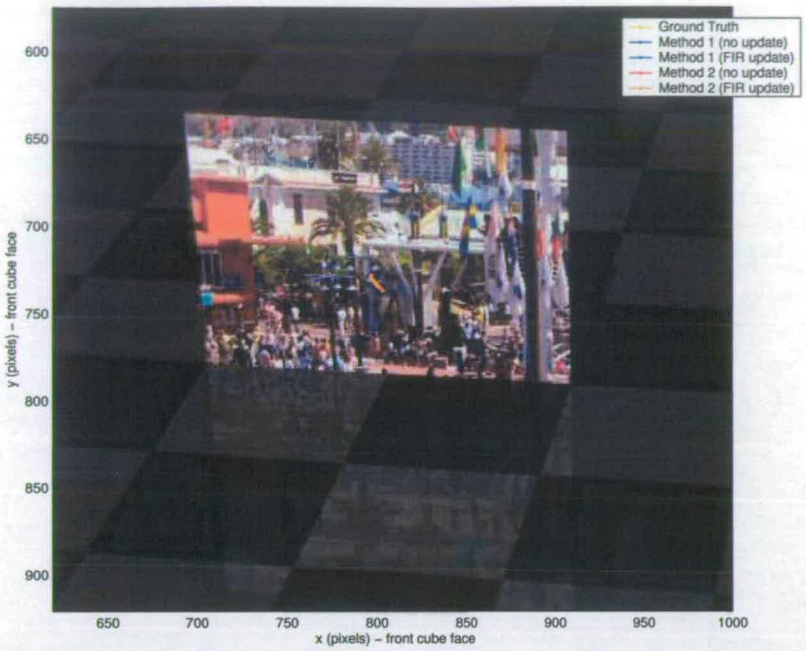


Figure A.1: *Object tracks using multiscale trackers for sequence* **`image1`** *in world space coordinates (projected onto the front cube face) overlayed on a composite image of the sequence*

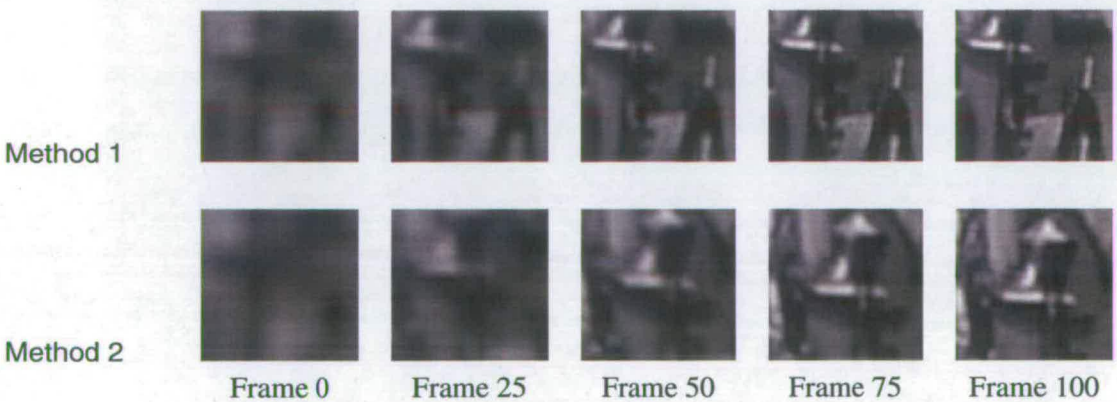


Figure A.2: *Reference block tracked in multiscale Method 1 and Method 2 with FIR updating for sequence* **`image1`**

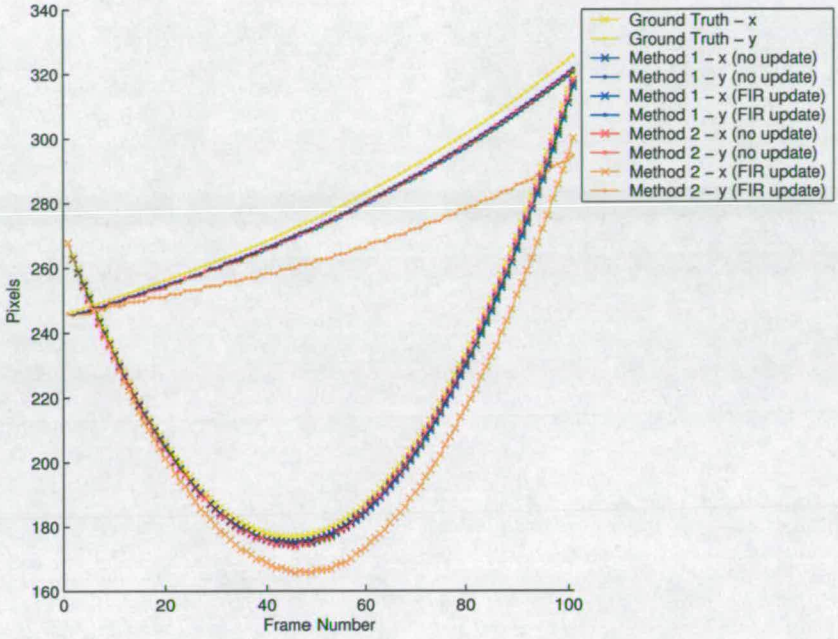


Figure A.3: *Object tracks using multiscale trackers for sequence* **`image1`** *in image pixel coordinates*

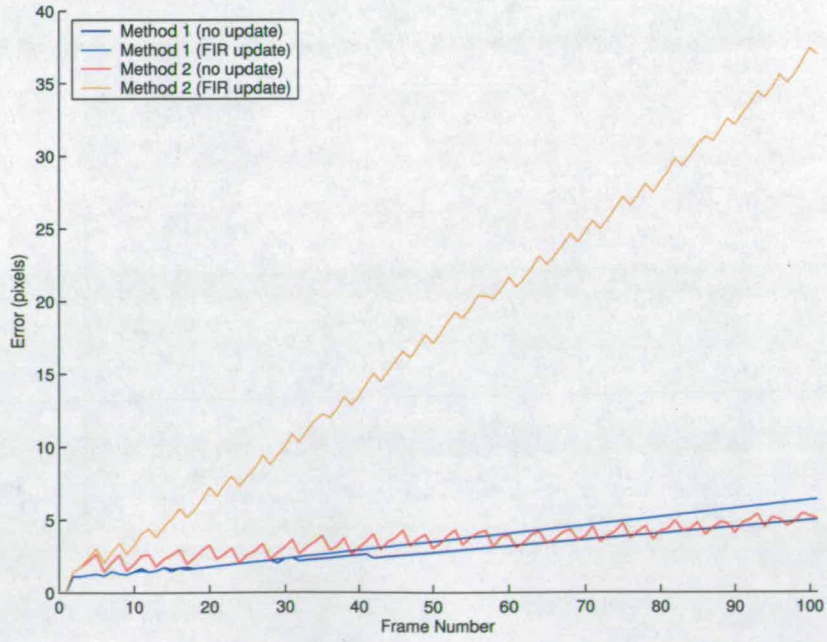


Figure A.4: *Errors* **`image1`**

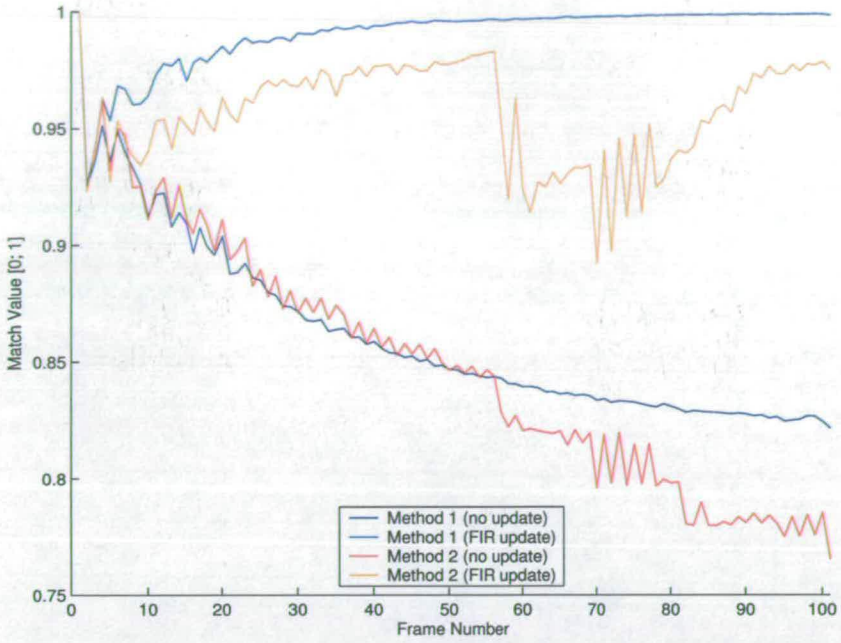


Figure A.5: *Match confidence for multiscale trackers for sequence* **`image1`**

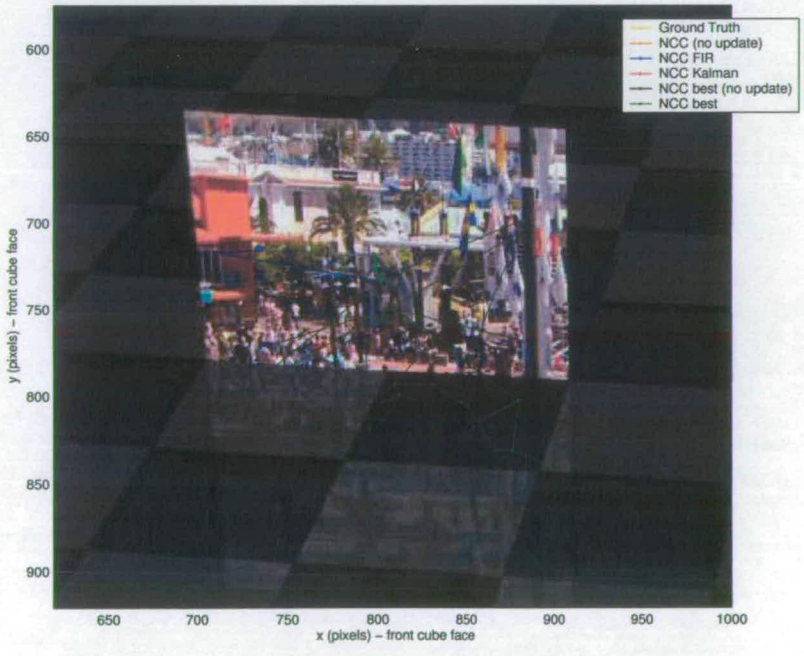


Figure A.6: *Object tracks using NCC trackers for sequence* **`image1`** *in world space co-ordinates (projected onto the front cube face) overlayed on a composite image of the sequence*

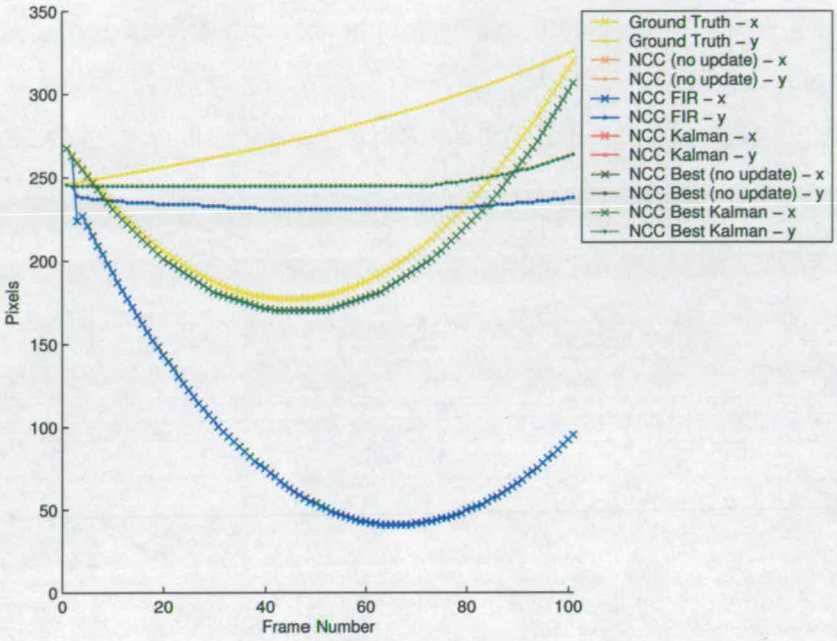


Figure A.7: *Object tracks using NCC trackers for sequence* **`image1`** *in image pixel co-ordinates*

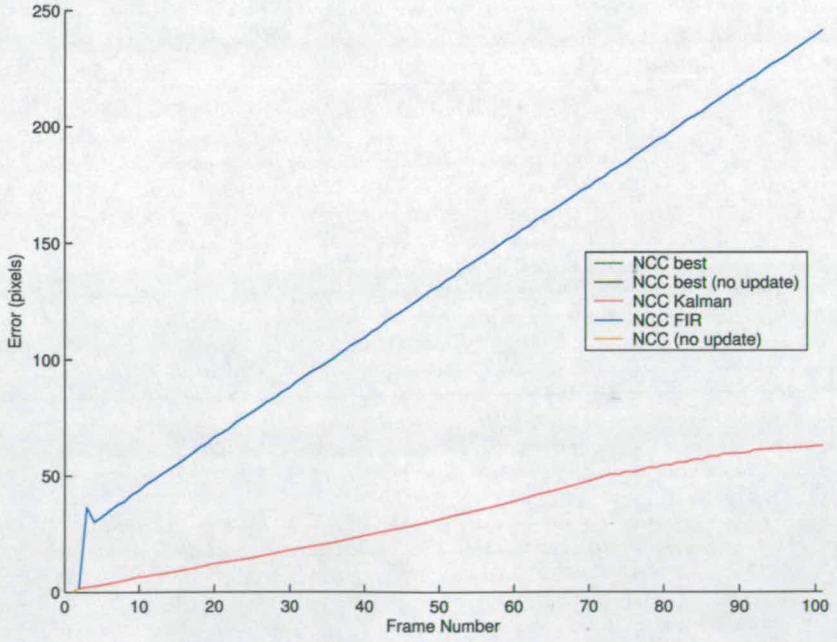


Figure A.8: *Match confidence for NCC trackers for sequence* **`image1`**

A.1.2 Sequence – `man1`

Figure A.12 and Figure A.10 show the tracking results on a composite of the sequence projected onto the front cube face in world co-ordinates. The track position in image co-ordinates is shown in Figure A.9 and the confidence of the match (or match value) as a percentage is shown in Figure A.13. The reference block is shown in Figure A.16 at various stages during the track for the multiscale methods. A summary of the RMS pixel tracking errors is shown in the table below:

	Position Filtered NCC		Best NCC		Method 1		Method 2	
No block updating	0.08*	(8)	0.1*	(8)	0.1*	(15)	2.3	(101)
FIR block updating	11.1†	(101)	–		0.9	(101)	15.3†	(101)
Kalman updating	13.3†	(101)	13.3†	(101)	–		–	

† Track fails * Partial track

(See Section 7.2 pg 170 for explanation of table)

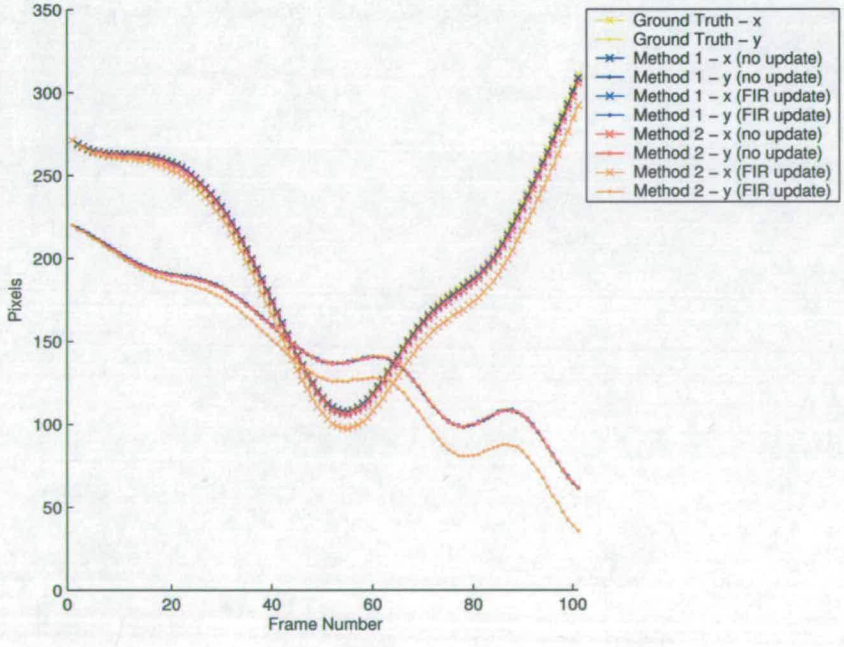


Figure A.9: *Object tracks using multiscale trackers for sequence* `man1` *in image pixel co-ordinates*

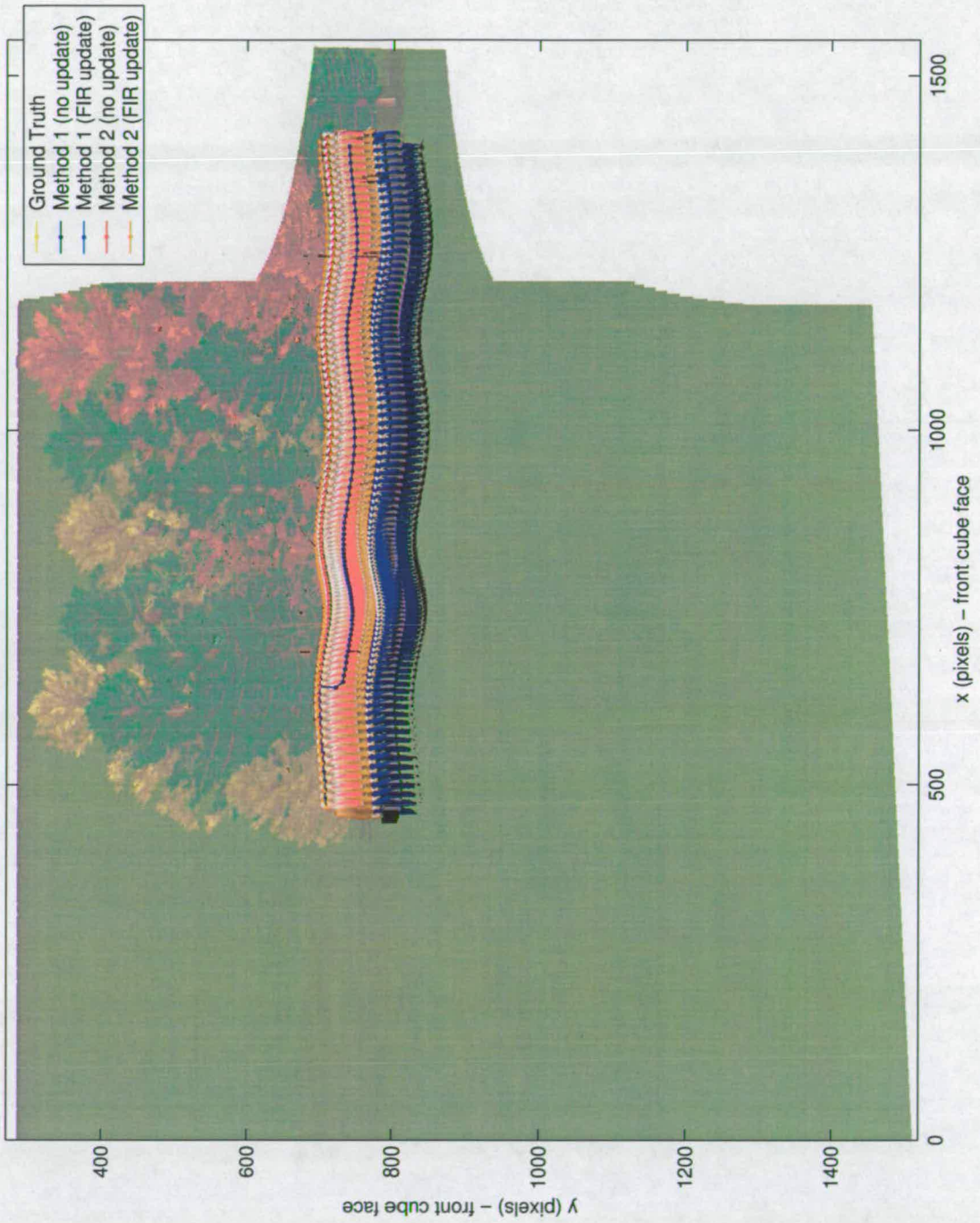


Figure A.10: *Object tracks using multiscale trackers for sequence* **man1** *in world space coordinates (projected onto the front cube face) overlayed on a composite image of the sequence*

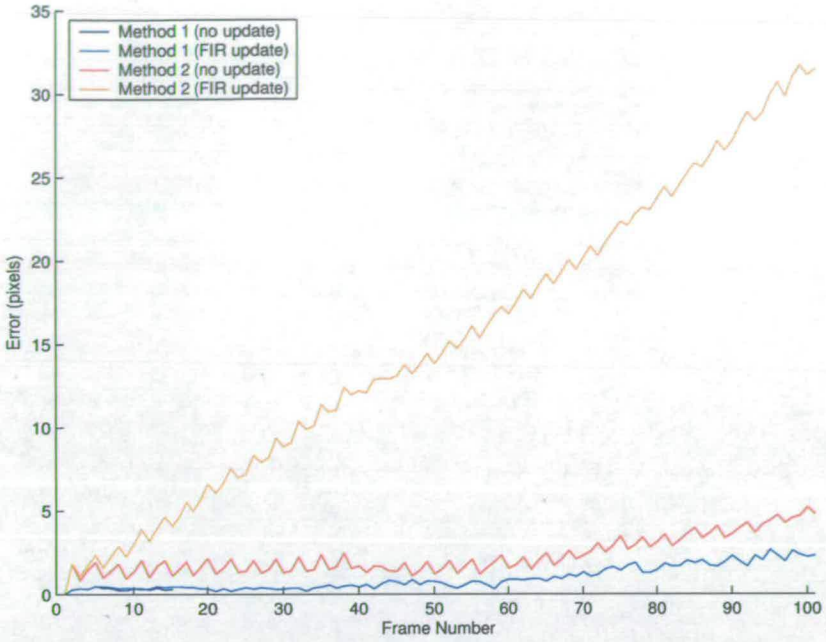


Figure A.11: *Errors* `man1`

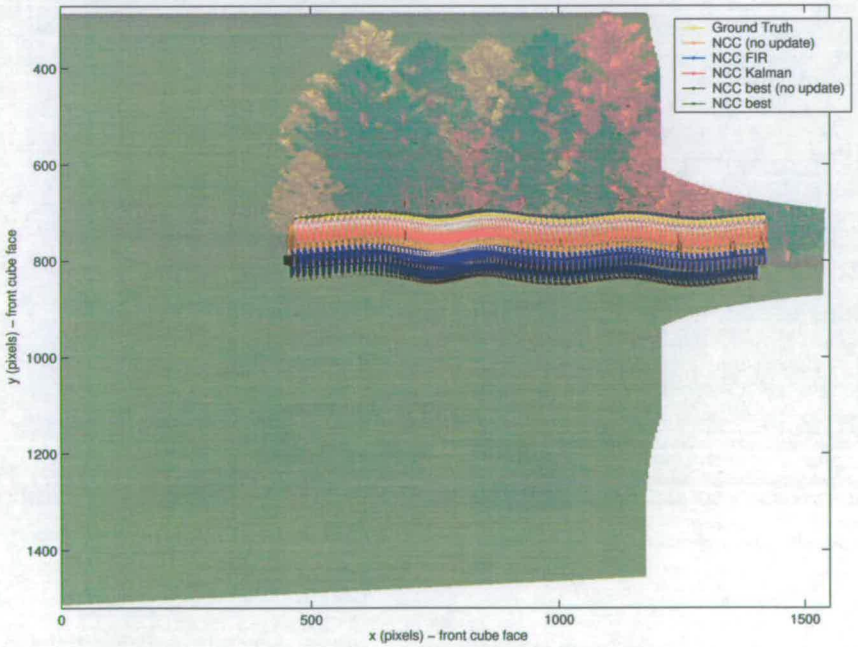


Figure A.12: *Object tracks using NCC trackers for sequence* `man1` *in world space co-ordinates (projected onto the front cube face) overlayed on a composite image of the sequence*

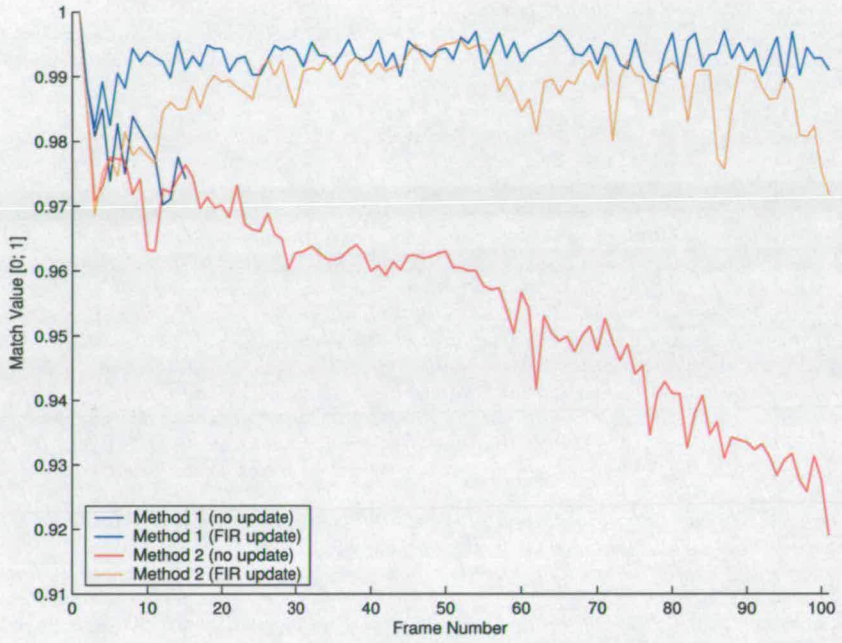


Figure A.13: *Match confidence for multiscale trackers for sequence* **man1**

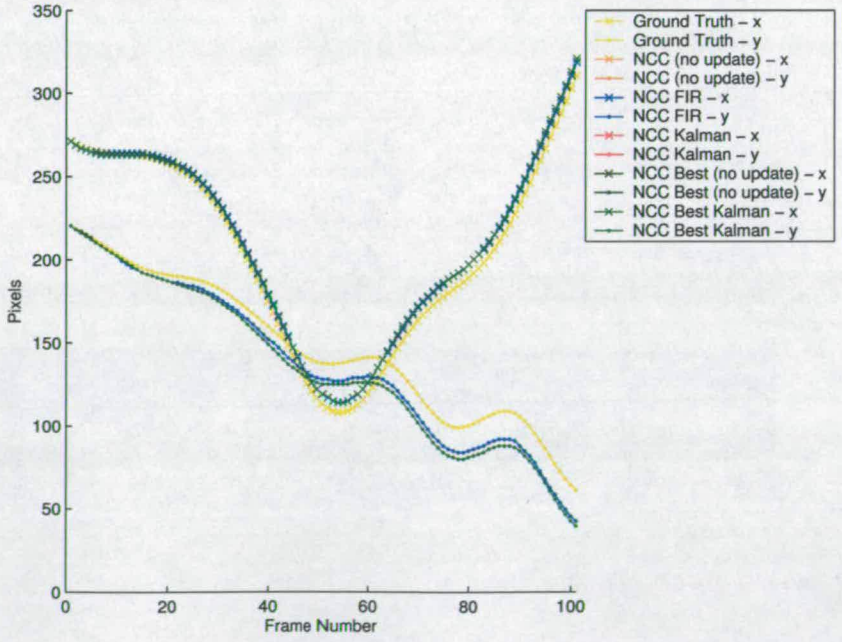


Figure A.14: *Object tracks using NCC trackers for sequence* **man1** *in image pixel co-ordinates*

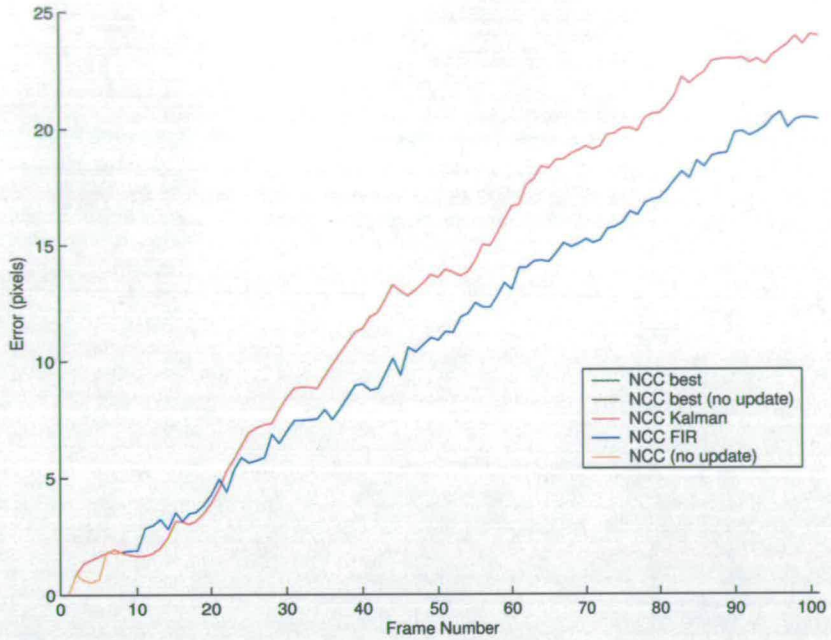


Figure A.15: *Match confidence for NCC trackers for sequence* **man1**

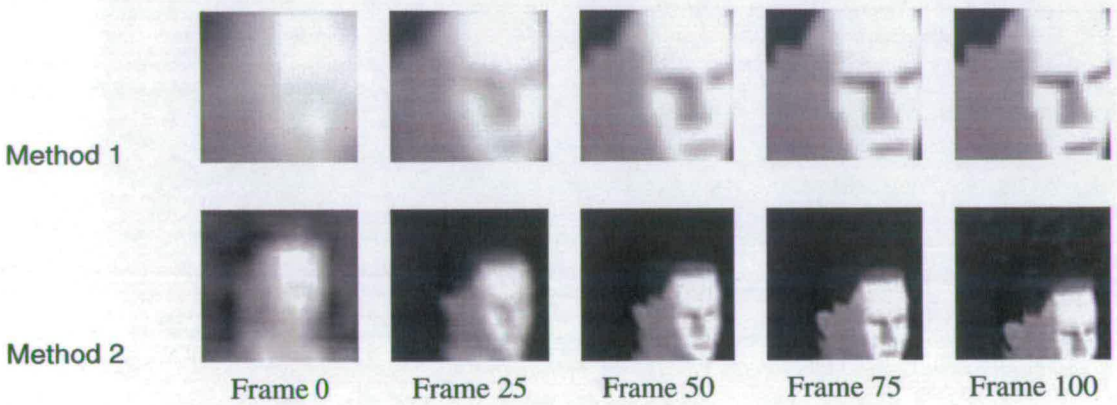


Figure A.16: *Reference block tracked in multiscale Method 1 and Method 2 with FIR updating for sequence* **man1**

A.1.3 Sequence – `bottle1`

Figure A.20 and Figure A.18 show the tracking results on a composite of the sequence projected onto the front cube face in world co-ordinates. The label co-ordinates on the image are shown in Figure A.17 and the confidence of the match (or match value) as a percentage is shown in Figure A.21. The reference block is shown in Figure A.21 at various stages during the track for the multiscale methods. A summary of the RMS pixel tracking errors is shown in the table below:

	Position Filtered NCC		Best NCC		Method 1		Method 2	
No block updating	36.8*	(24)	57.8†	(40)	3.1	(40)	1.3	(40)
FIR block updating	46.9†	(40)	–		3.3	(40)	6.7	(40)
Kalman updating	38.3†	(40)	9.3†	(40)	–		–	

† Track fails * Partial track

(See Section 7.2 pg 170 for explanation of table)

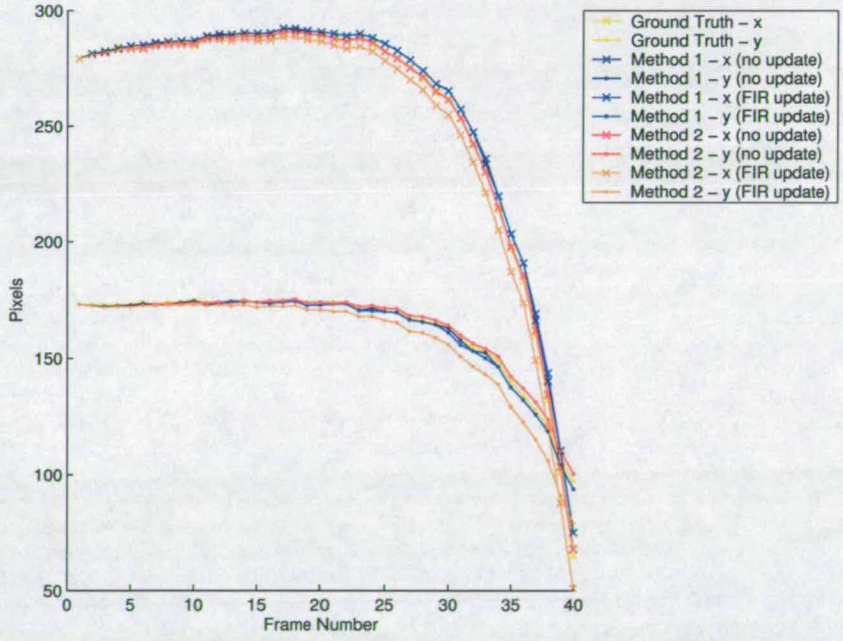


Figure A.17: *Object tracks using multiscale trackers for sequence* `bottle1` *in image pixel co-ordinates*

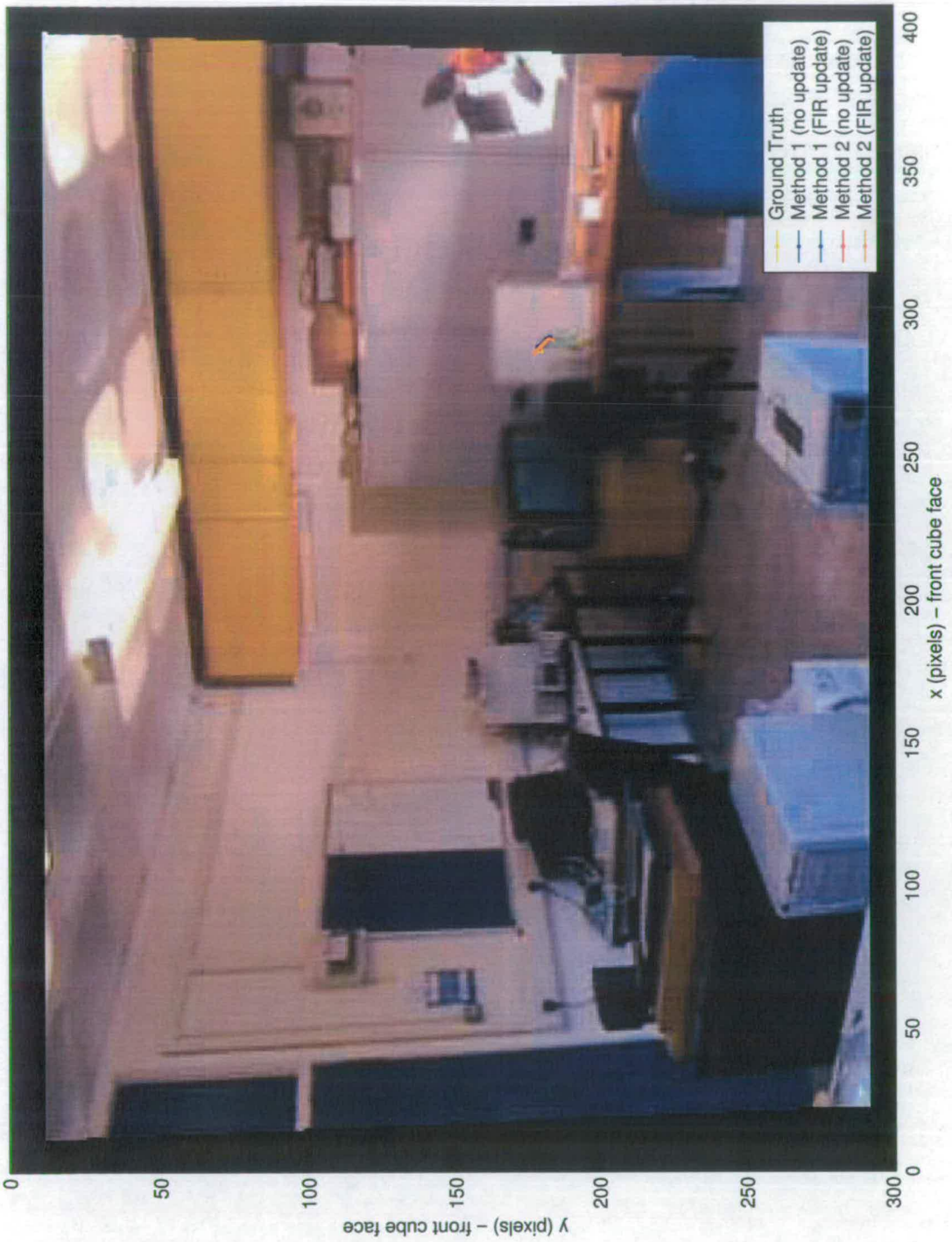


Figure A.18: *Object tracks using multiscale trackers for sequence* **`bottle1`** *in world space coordinates (projected onto the front cube face) overlayed on a composite image of the sequence*

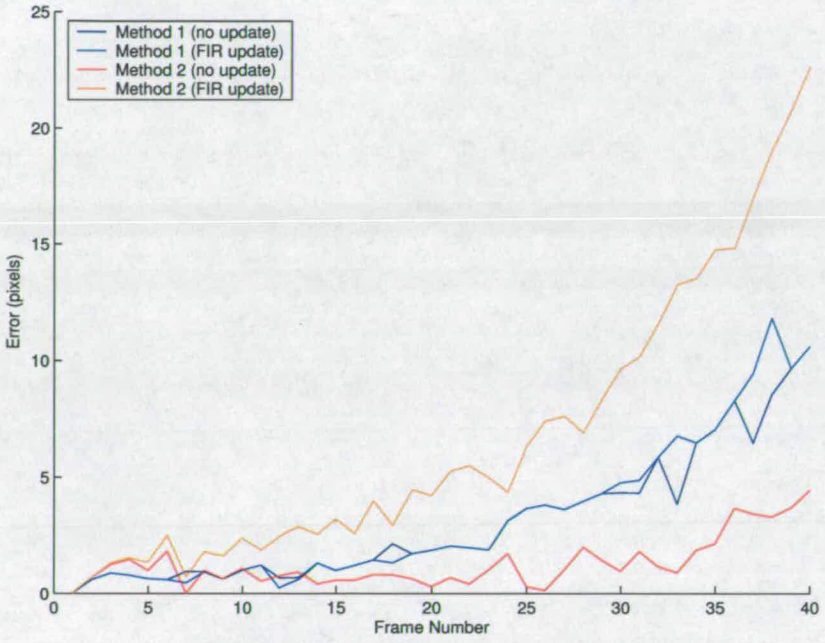


Figure A.19: *Errors* `bottle1`

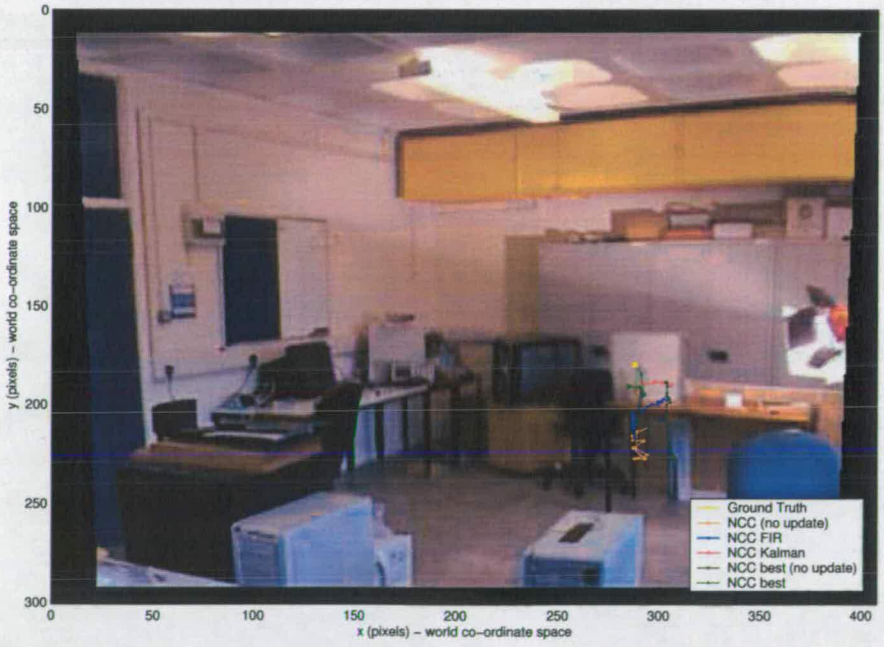


Figure A.20: *Object tracks using NCC trackers for sequence* `bottle1` *in world space co-ordinates (projected onto the front cube face) overlayed on a composite image of the sequence*

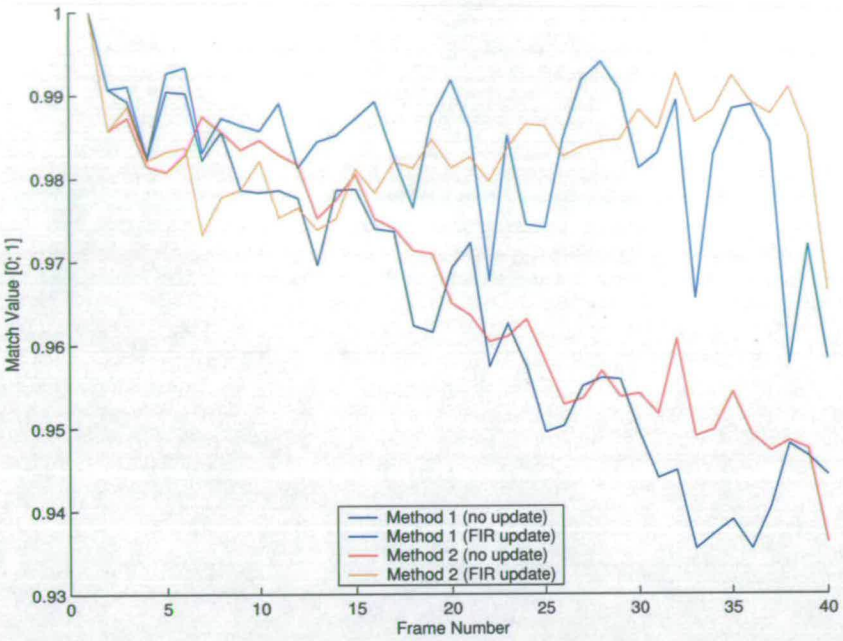


Figure A.21: *Match confidence for multiscale trackers for sequence* **`bottle1`**

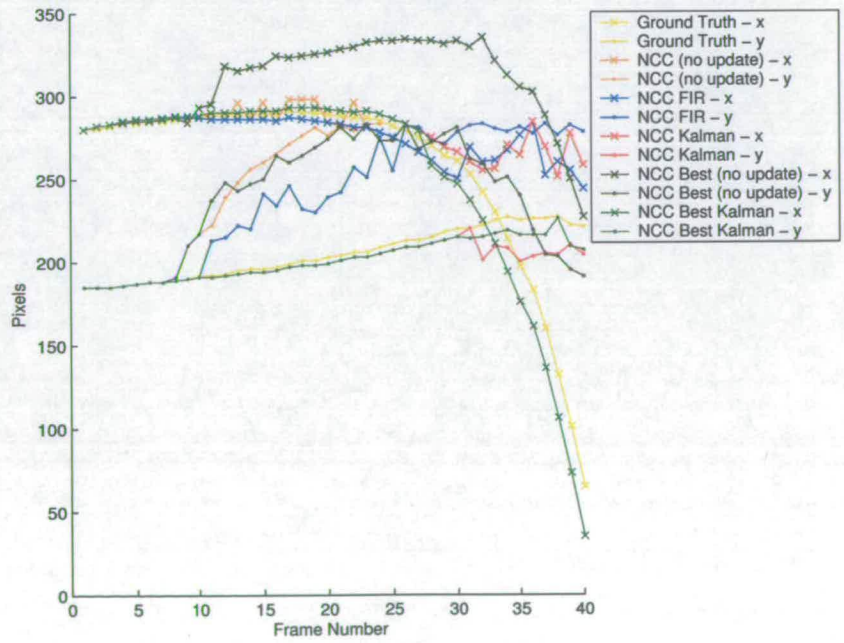


Figure A.22: *Object tracks using NCC trackers for sequence* **`bottle1`** *in image pixel coordinates*

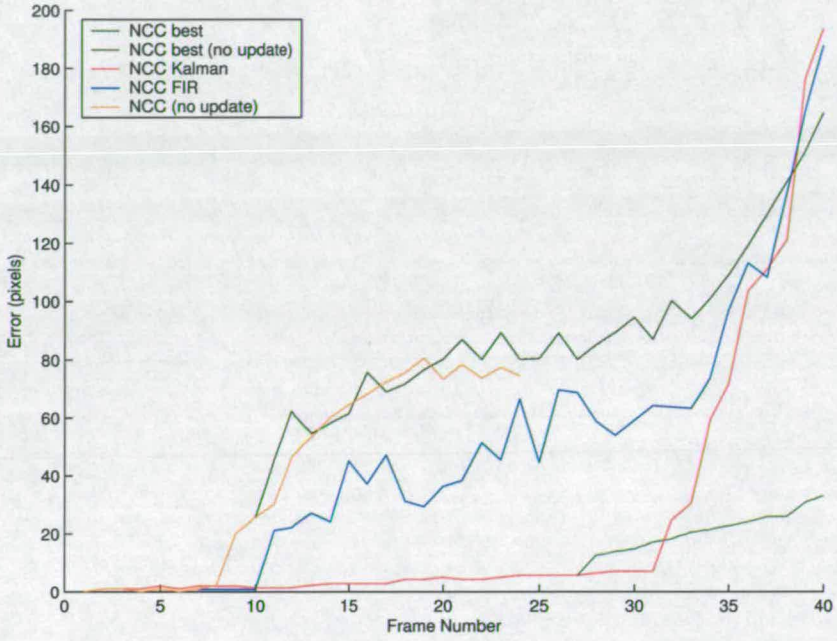


Figure A.23: *Match confidence for NCC trackers for sequence* **`bottle1`**

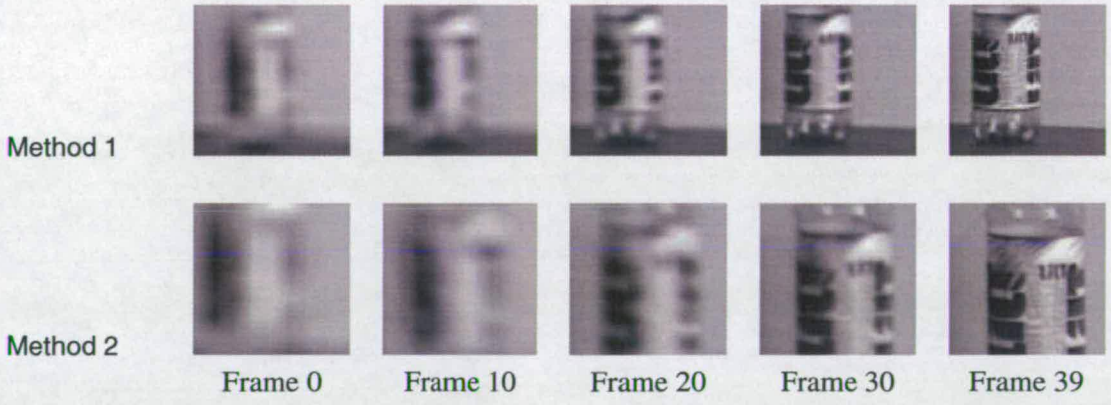


Figure A.24: *Reference block tracked in multiscale Method 1 and Method 2 with FIR updating for sequence* **`bottle1`**

A.1.4 Sequence – tom1

Figure A.28 and Figure A.26 show the tracking results on a composite of the sequence projected onto the front cube face in world co-ordinates. The co-ordinates of the track position in each frame is shown in Figure A.25 and the confidence of the match (or match value) as a percentage is shown in Figure A.29. The reference block is shown in Figure A.29 at various stages during the track for the multiscale methods. A summary of the RMS pixel tracking errors is shown in the table below:

	Position Filtered NCC		Best NCC		Method 1		Method 2	
No block updating	2.6*	(20)	2.6*	(20)	2.1*	(16)	1.0*	(25)
FIR block updating	2.5*	(41)	–		1.7*	(16)	2.3*	(27)
Kalman updating	1.6*	(18)	1.6*	(16)	–		–	

† Track fails * Partial track

(See Section 7.2 pg 170 for explanation of table)

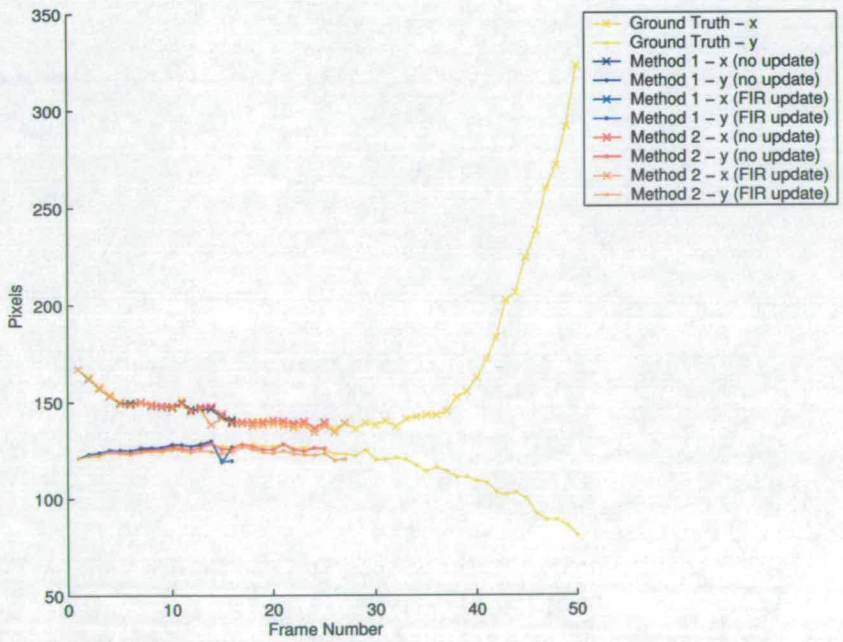


Figure A.25: *Object tracks using multiscale trackers for sequence* `tom1` *in image pixel co-ordinates*



Figure A.26: *Object tracks using multiscale trackers for sequence* `tom1` *in world space coordinates (projected onto the front cube face) overlayed on a composite image of the sequence*

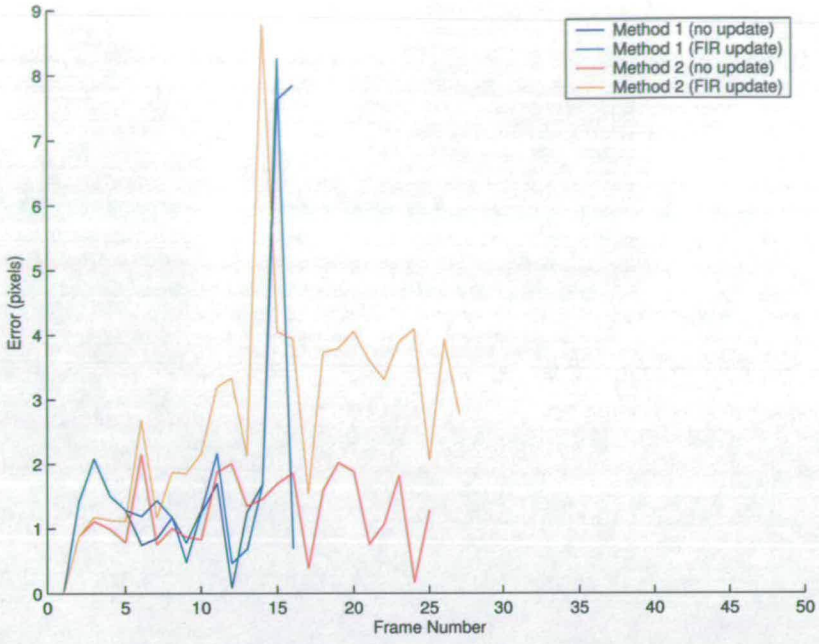


Figure A.27: *Errors* `tom1`



Figure A.28: *Object tracks using NCC trackers for sequence* `tom1` *in world space co-ordinates (projected onto the front cube face) overlayed on a composite image of the sequence*

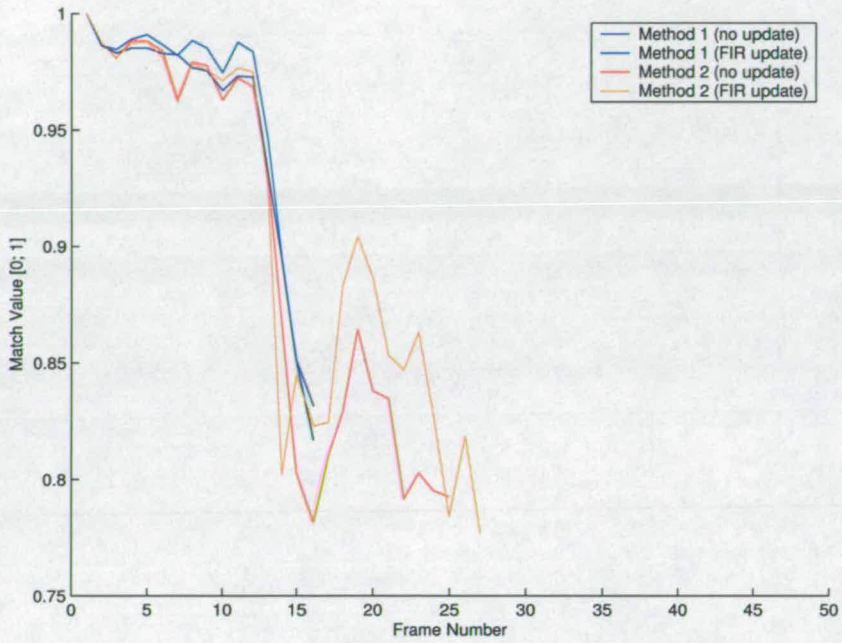


Figure A.29: *Match confidence for multiscale trackers for sequence* `tom1`

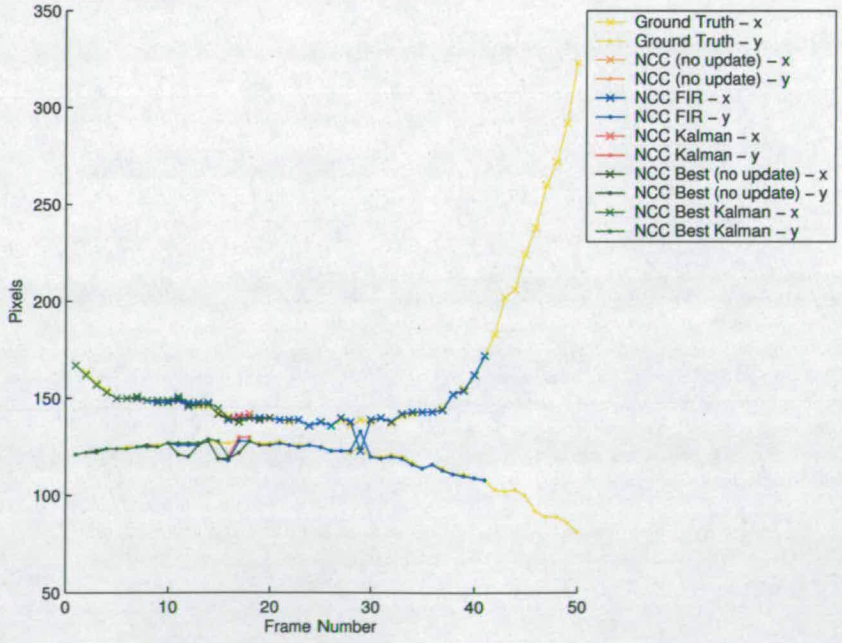


Figure A.30: *Object tracks using NCC trackers for sequence* `tom1` *in image pixel co-ordinates*

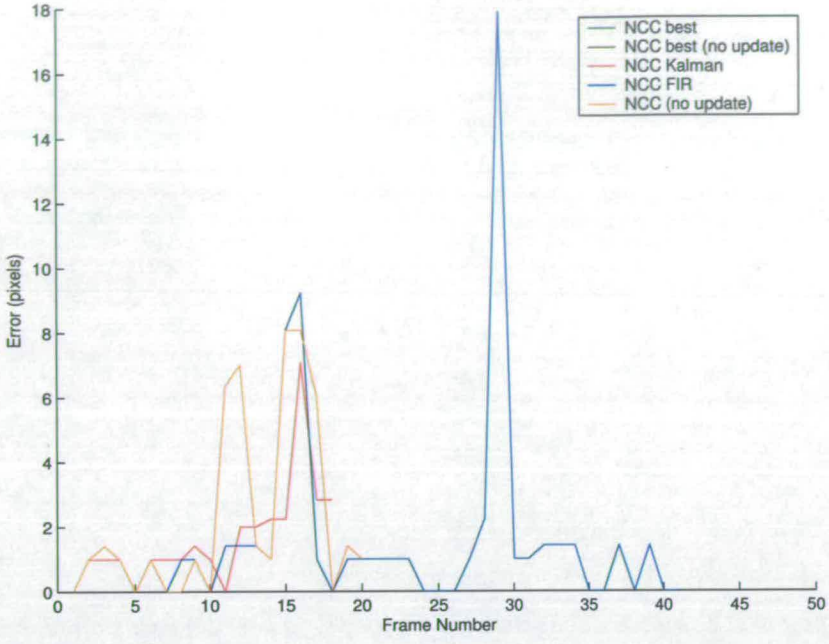


Figure A.31: *Match confidence for NCC trackers for sequence* `tom1`

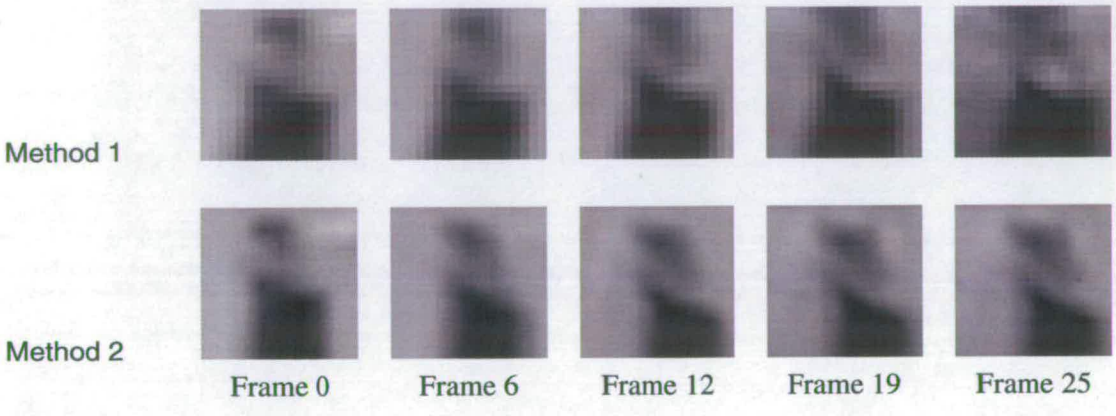


Figure A.32: *Reference block tracked in multiscale Method 1 and Method 2 with FIR updating for sequence* `tom1`

A.1.5 Sequence – `tom3`

Figure A.36 and Figure A.34 show the tracking results on a composite of the sequence projected onto the front cube face in world co-ordinates. Track co-ordinates on the image are shown in Figure A.33 and the confidence of the match (or match value) as a percentage is shown in Figure A.37. The reference block is shown in Figure A.40 at various stages during the track for the multiscale methods. A summary of the RMS pixel tracking errors is shown in the table below:

	Position Filtered NCC		Best NCC		Method 1		Method 2	
No block updating	0.4*	(3)	4.4	(42)	3.5*	(21)	7.5	(42)
FIR block updating	2.0*	(15)	–		3.0†	(42)	3.2	(42)
Kalman updating	9.9	(42)	1.1	(42)	–		–	

† Track fails * Partial track

(See Section 7.2 pg 170 for explanation of table)

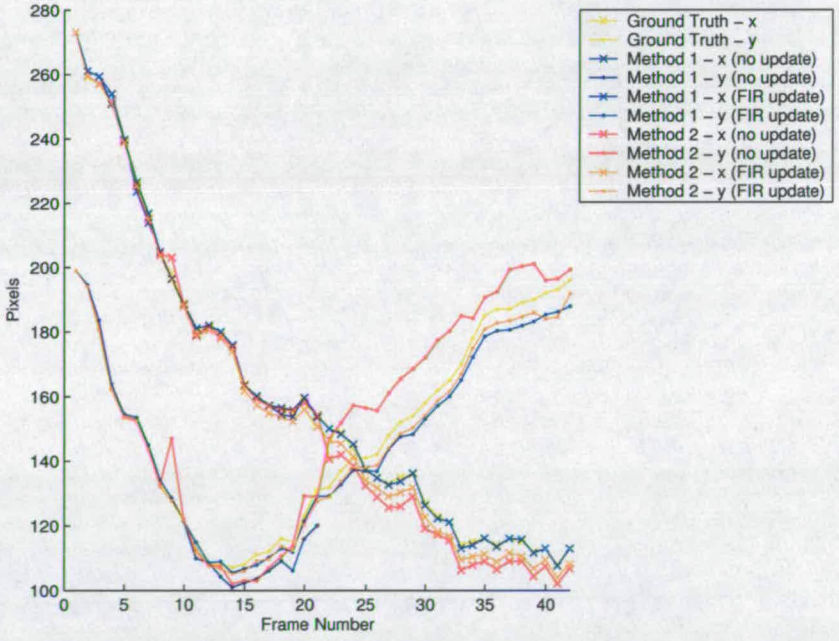


Figure A.33: *Object tracks using multiscale trackers for sequence* `tom3` *in image pixel co-ordinates*

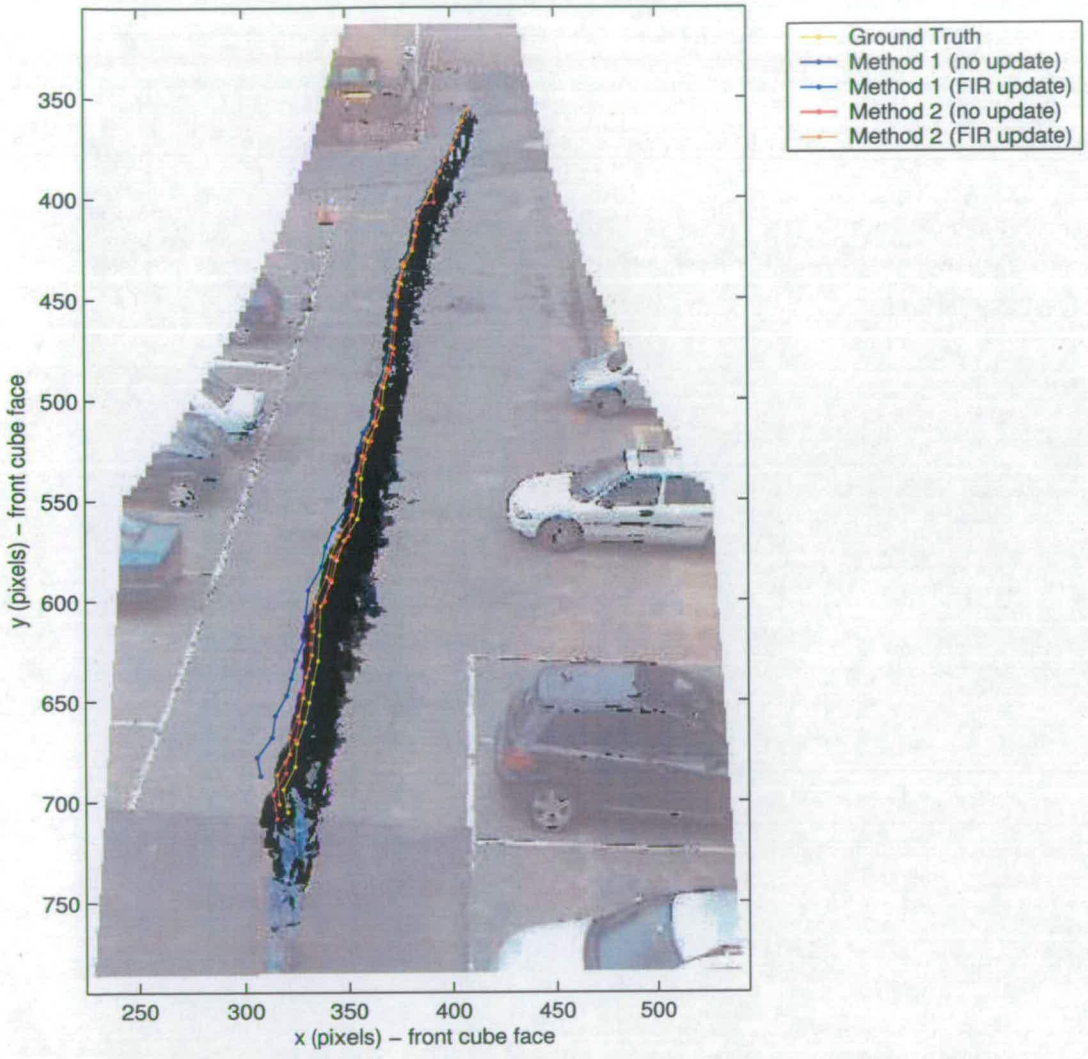


Figure A.34: *Object tracks using multiscale trackers for sequence* `tom3` *in world space co-ordinates (projected onto the front cube face) overlayed on a composite image of the sequence*

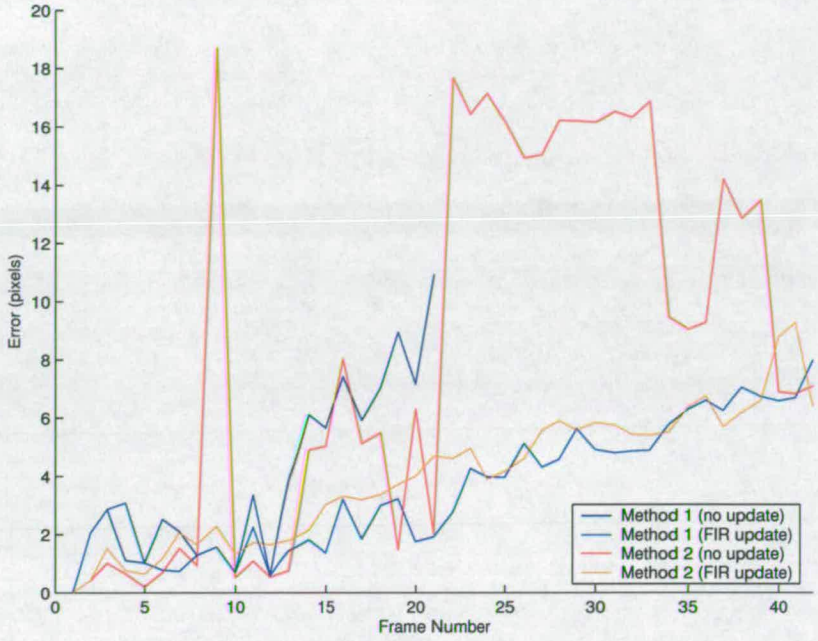


Figure A.35: *Errors* `tom3`

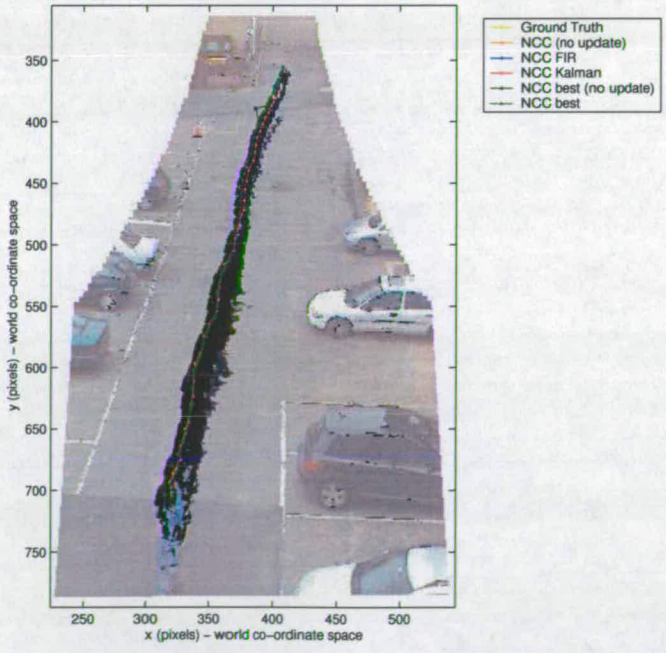


Figure A.36: *Object tracks using NCC trackers for sequence* `tom3` *in world space co-ordinates (projected onto the front cube face) overlayed on a composite image of the sequence*

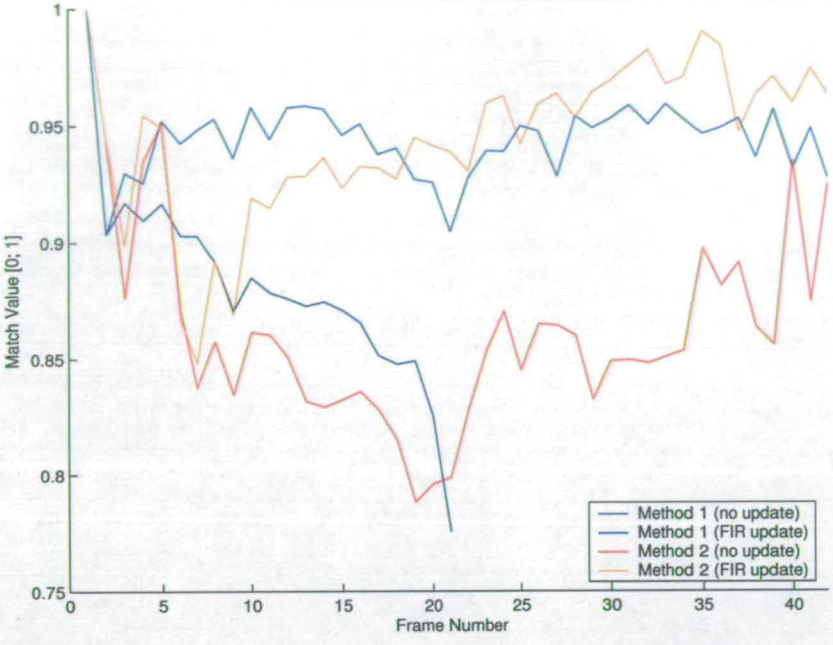


Figure A.37: *Match confidence for multiscale trackers for sequence* `tom3`

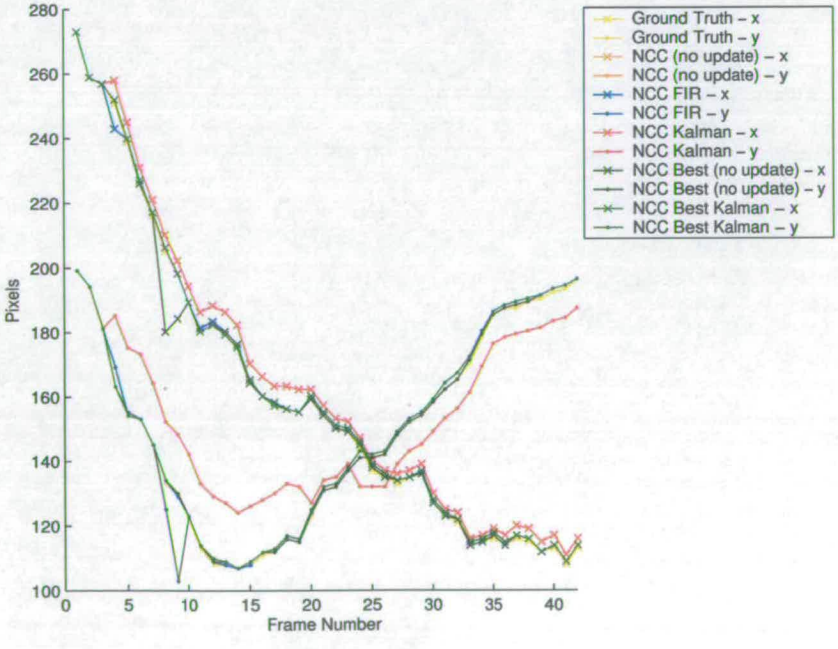


Figure A.38: *Object tracks using NCC trackers for sequence* `tom3` *in image pixel co-ordinates*

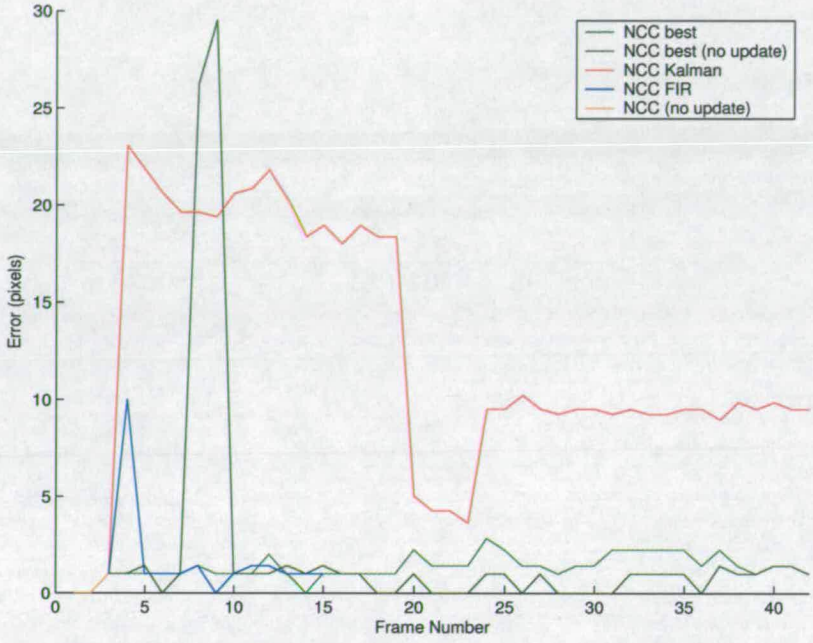


Figure A.39: *Match confidence for NCC trackers for sequence* `tom3`

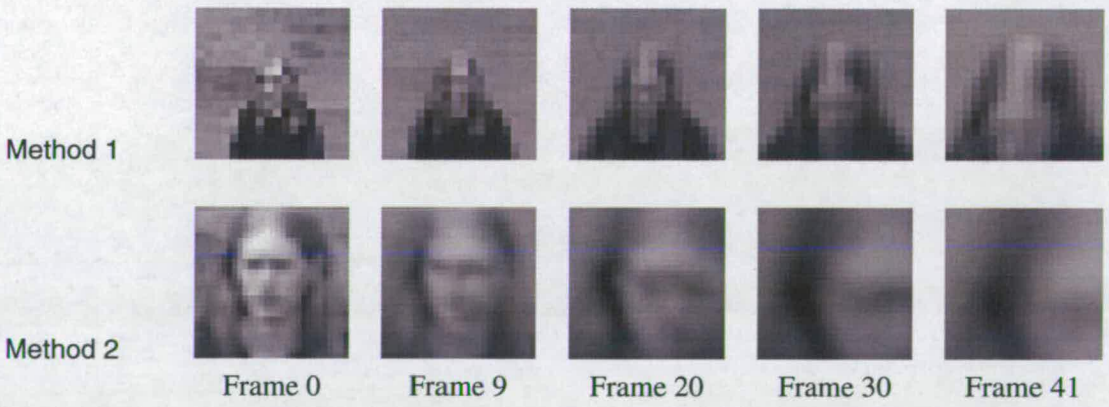


Figure A.40: *Reference block tracked in multiscale Method 1 and Method 2 with FIR updating for sequence* `tom3`

A.1.6 Sequence – `tom4`

Figure A.44 and Figure A.42 show the tracking results on a composite of the sequence projected onto the front cube face in world co-ordinates. The target co-ordinates on the image are shown in Figure A.41 and the confidence of the match (or match value) as a percentage is shown in Figure A.45. The reference block is shown in Figure A.48 at various stages during the track for the multiscale methods. A summary of the RMS pixel tracking errors is shown in the table below:

	Position Filtered NCC		Best NCC		Method 1		Method 2	
No block updating	21.3†	(65)	8.5†	(65)	9.3*	(47)	1.5	(39)
FIR block updating	20.6†	(65)	–		8.3*	(65)	4.0	(65)
Kalman updating	19.7†	(65)	4.7	(65)	–		–	

† Track fails * Partial track

(See Section 7.2 pg 170 for explanation of table)

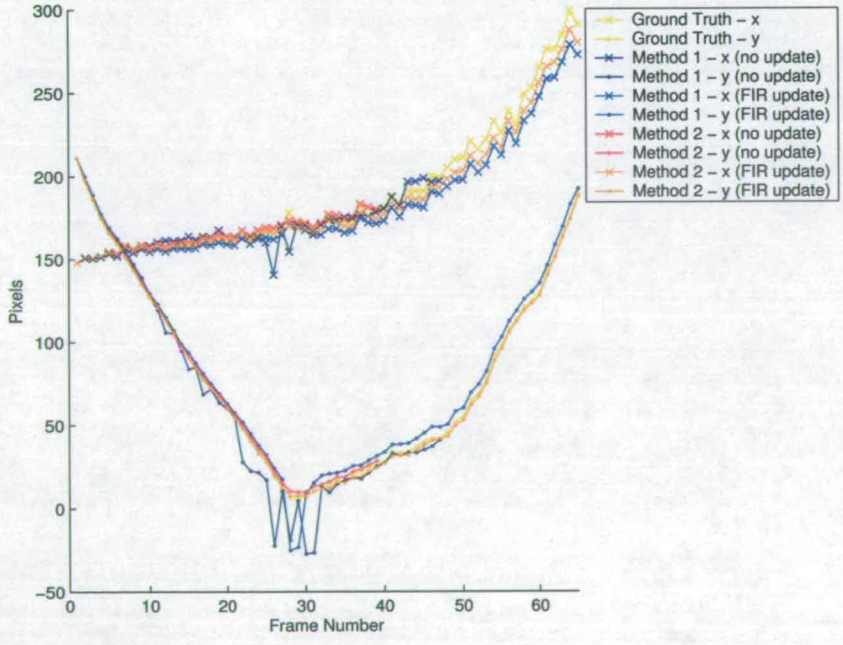


Figure A.41: *Object tracks using multiscale trackers for sequence* `tom4` *in image pixel co-ordinates*



Figure A.42: *Object tracks using multiscale trackers for sequence* `tom4` *in world space coordinates (projected onto the front cube face) overlayed on a composite image of the sequence*

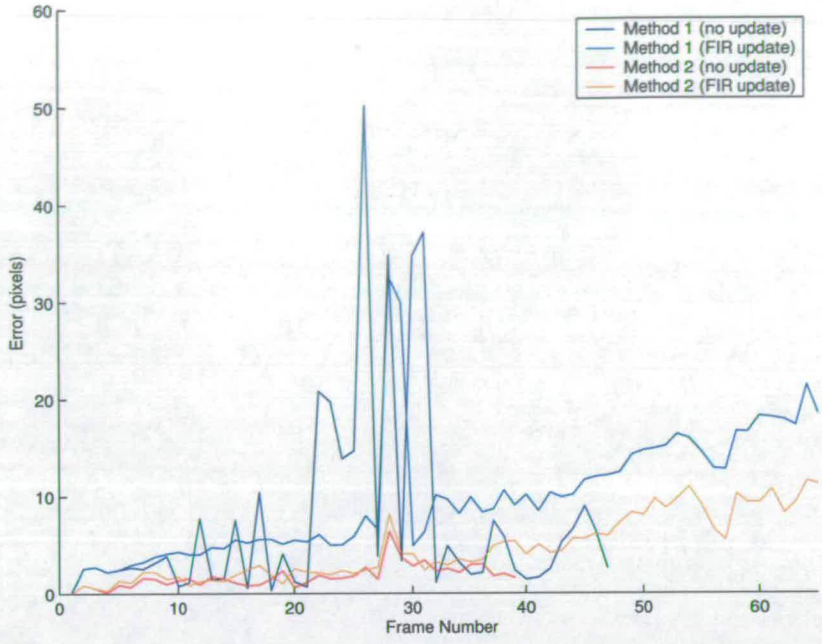


Figure A.43: *Errors* `tom4`

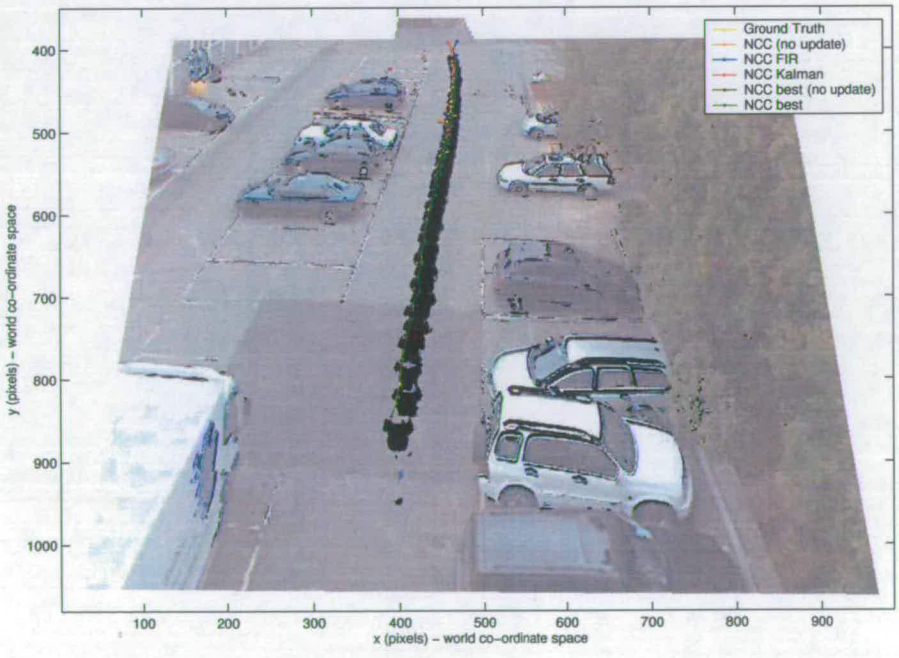


Figure A.44: *Object tracks using NCC trackers for sequence* `tom4` *in world space co-ordinates (projected onto the front cube face) overlayed on a composite image of the sequence*

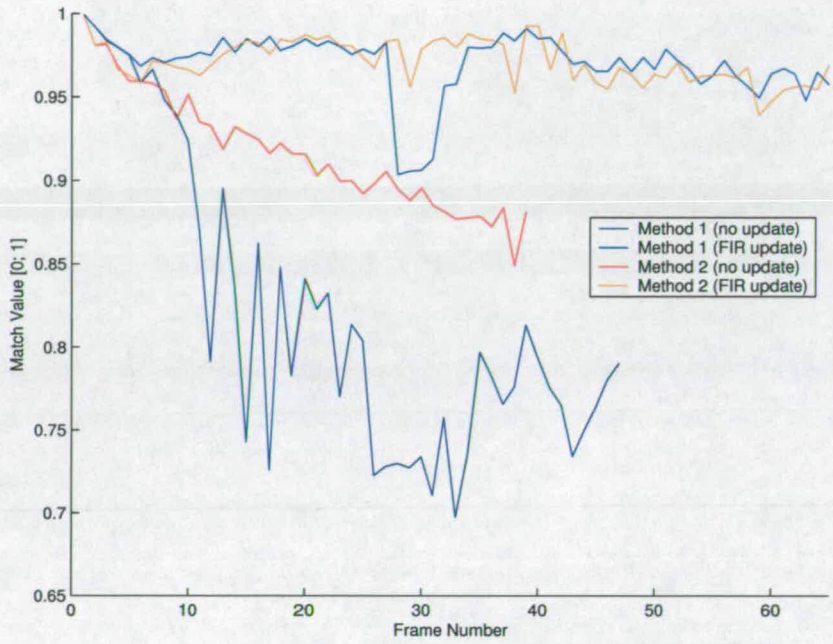


Figure A.45: *Match confidence for multiscale trackers for sequence* `tom4`

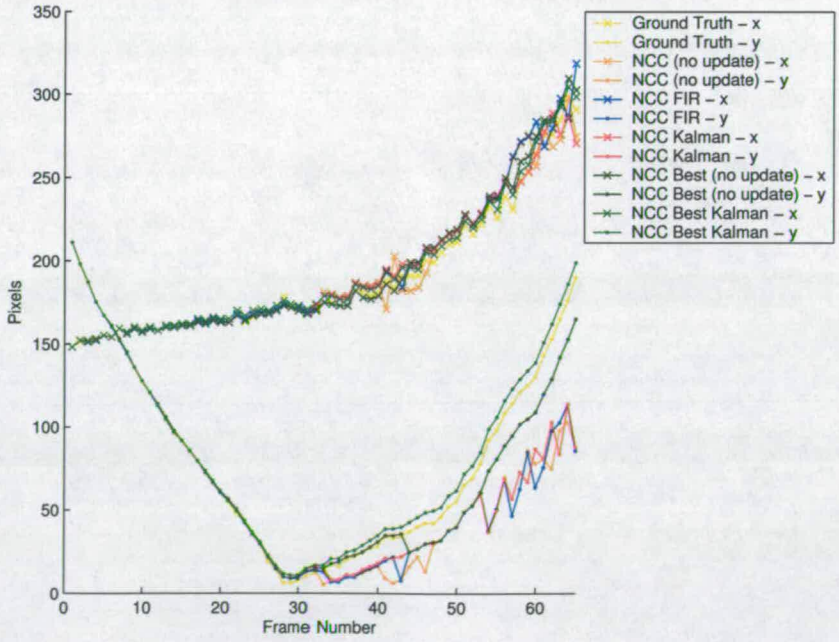


Figure A.46: *Object tracks using NCC trackers for sequence* `tom4` *in image pixel co-ordinates*

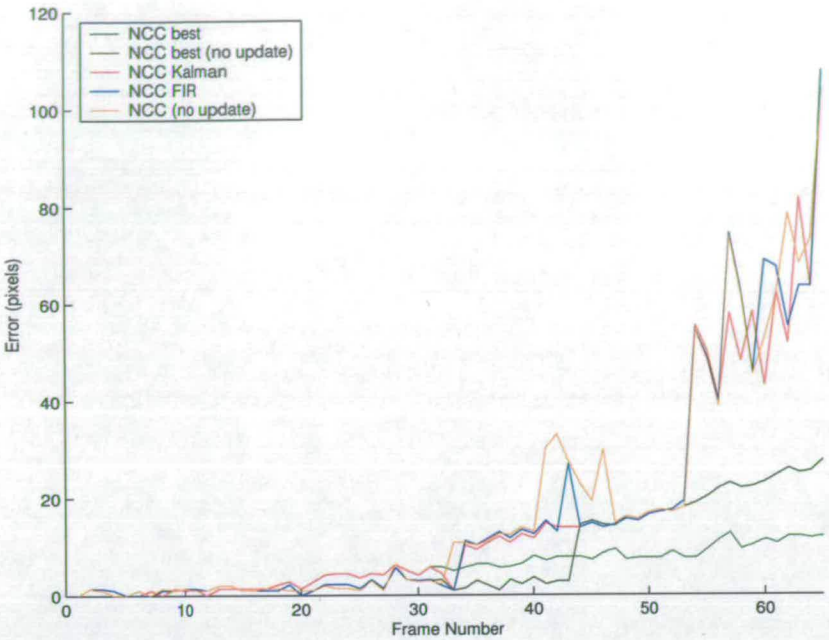


Figure A.47: *Match confidence for NCC trackers for sequence* `tom4`

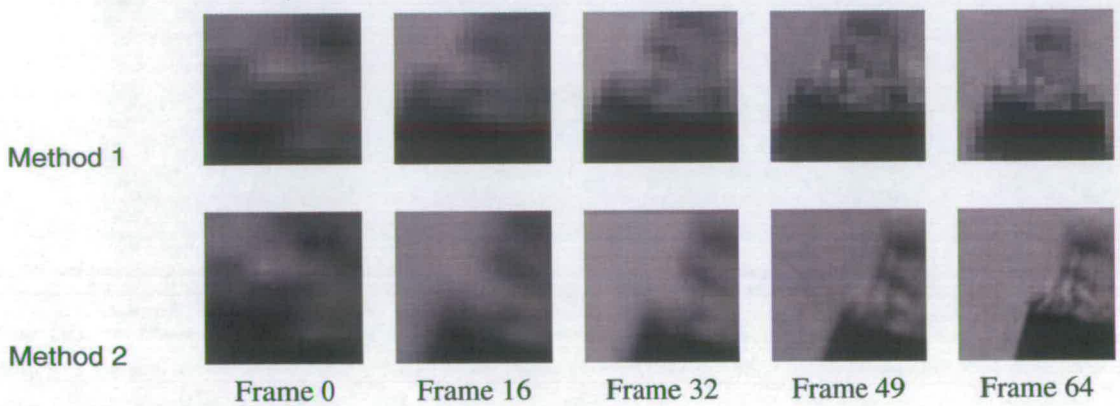


Figure A.48: *Reference block tracked in multiscale Method 1 and Method 2 with FIR updating for sequence* `tom4`

A.2 Contour Tracking

The following section lists the results from Chapter 7 for the contour tracker. A discussion of these results can be found in Section 7.5.3 on page 216.

A.2.1 Sequence – `bottle1`

The following table shows the RMS pixel tracking errors for the sequence **`bottle1`**. It compares the contour tracker to the multiscale tracker (using method 2) and the best conventional block tracker.

	Best NCC	Multiscale Method 2	Contour Tracker
No block updating	57.8†	1.3	1.0
FIR block updating	–	6.7	1.2
Kalman updating	9.3†	–	–

† Track fails * Partial track

(See Section 7.2 pg 170 for explanation of table)

Figure A.49 shows a composite of the sequence with the match position plotted with and without block updating. The hand-tracked ground truth is also shown. Figure A.50 shows the track positions in image co-ordinates for each frame against the ground truth.

The RMS tracking errors shown in Figure A.51 follow the trend where the error increases towards the middle of the sequence. This is due to mismatch errors accumulated from the beginning and the end of the sequences as the forward and backward trackers meet in the middle.



Figure A.49: *Contour Tracking – Composite* `bottle1`

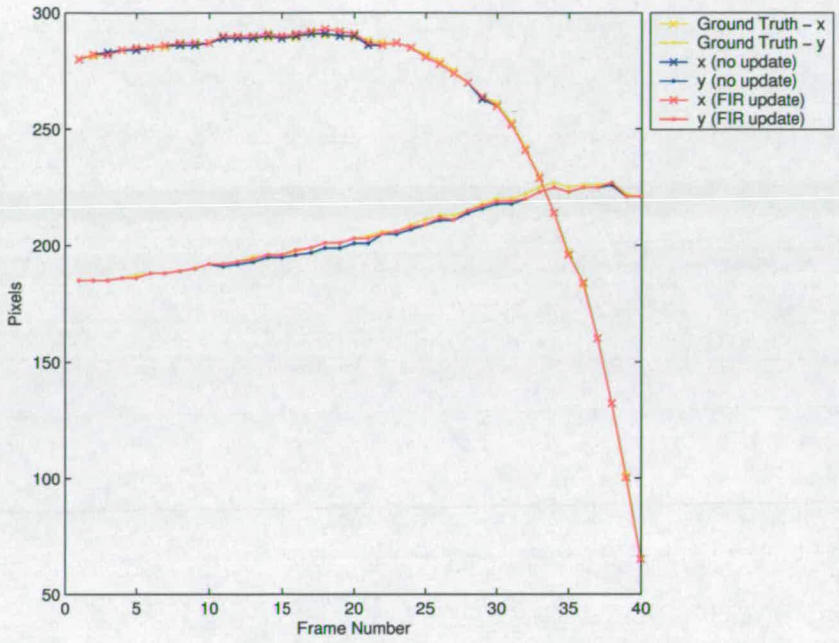


Figure A.50: *Contour Tracking – Track* `bottle1`

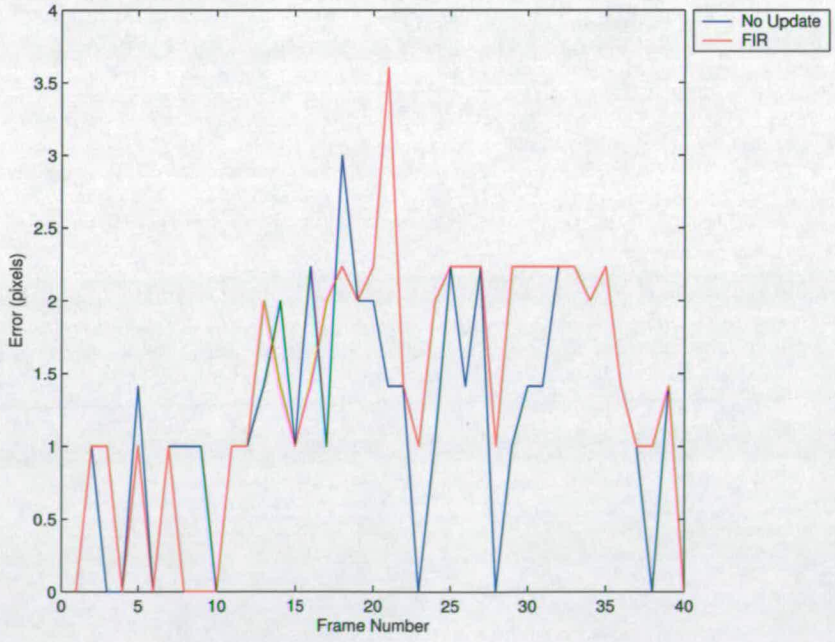


Figure A.51: *Contour Tracking – Errors* `bottle1`

A.2.2 Sequence – `tom1`

The following table shows the RMS pixel tracking errors for the sequence `tom1`. It compares the contour tracker to the multiscale tracker (using method 2) and the best conventional block tracker.

	Best NCC	Multiscale Method 2	Contour Tracker
No block updating	2.6*	1.0*	2.0
FIR block updating	–	2.3*	2.3
Kalman updating	1.6*	–	–

† Track fails * Partial track

(See Section 7.2 pg 170 for explanation of table)

Figure A.52 shows a composite of the sequence with the match position plotted with and without block updating. The hand-tracked ground truth is also shown. Figure A.53 shows the track positions in image co-ordinates for each frame against the ground truth.

There is a large variation in the RMS tracking errors (shown in Figure A.54) at the end of the sequence. This is caused by a combination of the choice of reference view and the low frame rate. There is a white line in the background of the reference block which only appears in the last frame. With a higher frame rate this line would disappear, but as it doesn't, the strong structure it adds to the reference causes mismatches.



Figure A.52: *Contour Tracking – Composite* `tom1`

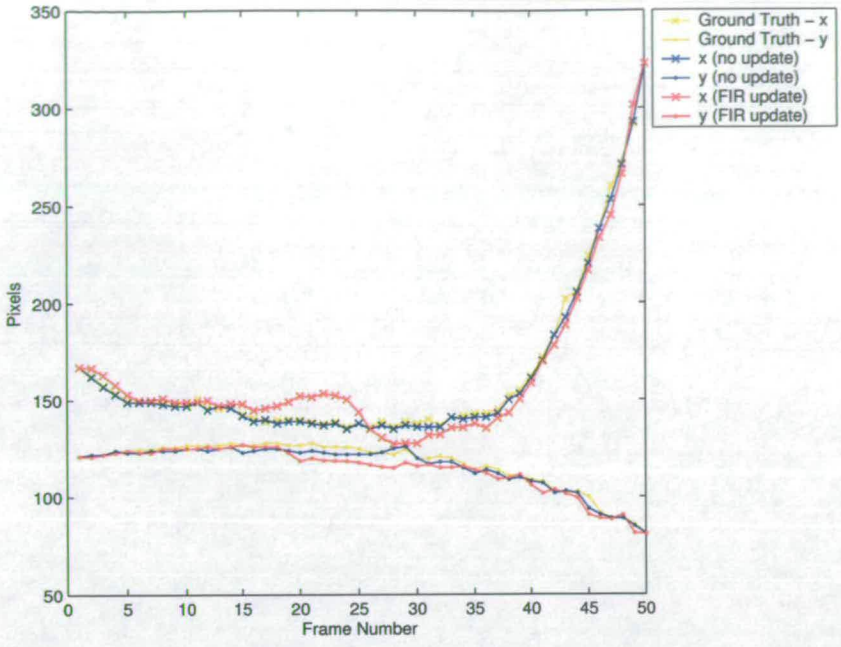


Figure A.53: *Contour Tracking – Track* `tom1`

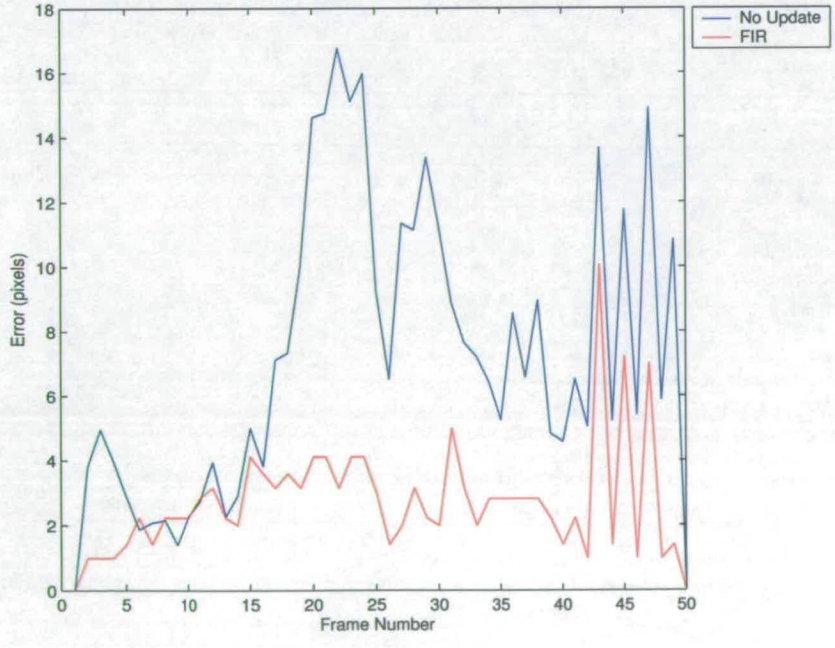


Figure A.54: *Contour Tracking – Errors* `tom1`

A.2.3 Sequence – `tom3`

The following table shows the RMS pixel tracking errors for the sequence `tom3`. It compares the contour tracker to the multiscale tracker (using method 2) and the best conventional block tracker.

	Best NCC	Multiscale Method 2	Contour Tracker
No block updating	4.4	7.5	2.9
FIR block updating	–	3.2	1.9
Kalman updating	1.1	–	–

† Track fails * Partial track

(See Section 7.2 pg 170 for explanation of table)

Figure A.55 shows a composite of the sequence with the match position plotted with and without block updating. The hand-tracked ground truth is also shown. Figure A.56 shows the track positions in image co-ordinates for each frame against the ground truth.

A clear mismatch is visible in the RMS tracking errors shown in Figure A.57 two thirds way through the sequence. The FIR block update appears to make the tracker more robust than without block updating which infers that the object has changed significantly by the middle of the sequence.

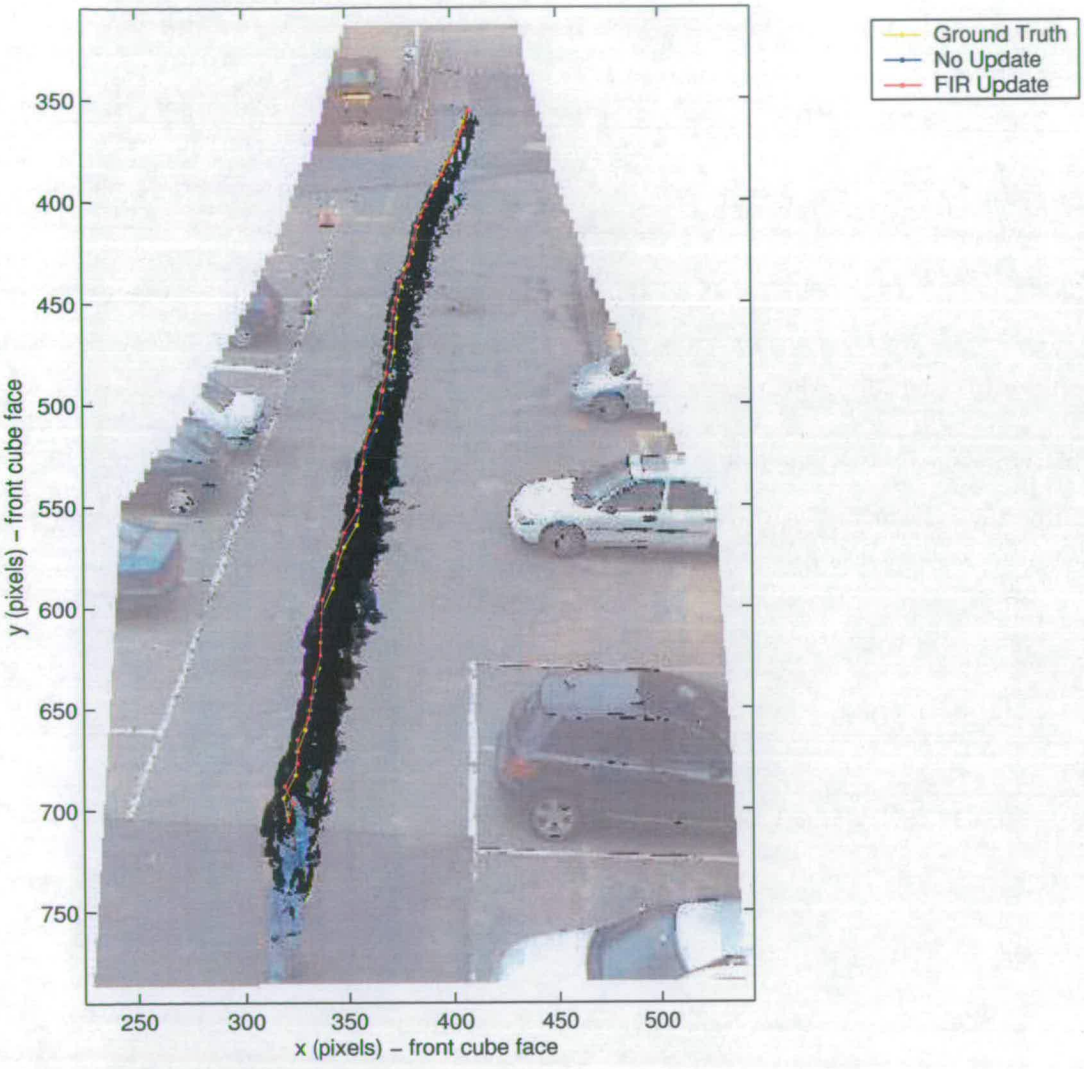


Figure A.55: *Contour Tracking – Composite* `tom3`

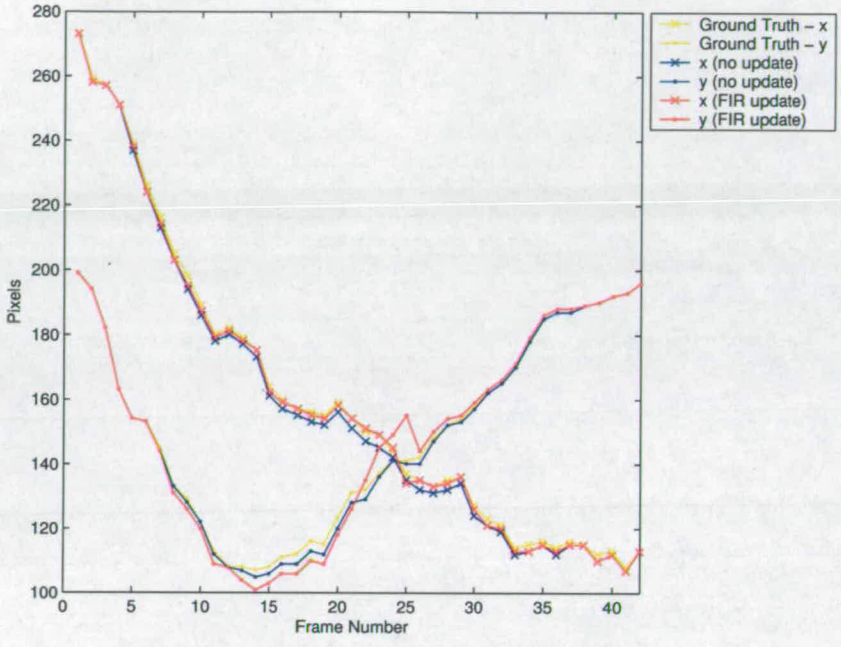


Figure A.56: *Contour Tracking – Track* `tom3`

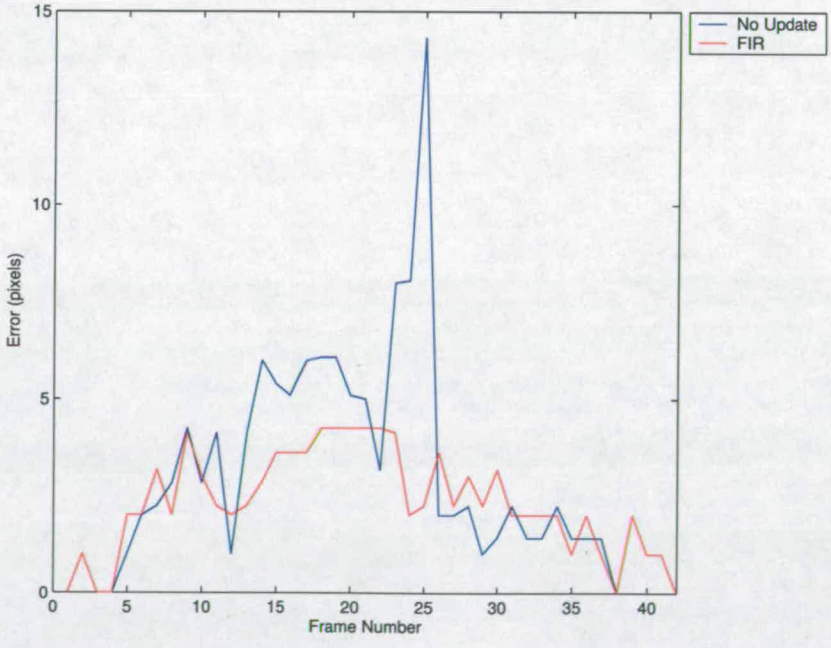


Figure A.57: *Contour Tracking – Errors* `tom3`

A.2.4 Sequence – tom4

The following table shows the RMS pixel tracking errors for the sequence **tom4**. It compares the contour tracker to the multiscale tracker (using method 2) and the best conventional block tracker.

	Best NCC	Multiscale Method 2	Contour Tracker
No block updating	8.5†	1.5	1.4
FIR block updating	–	4.0	2.0
Kalman updating	4.7	–	–

† Track fails * Partial track

(See Section 7.2 pg 170 for explanation of table)

Figure A.58 shows a composite of the sequence with the match position plotted with and without block updating. The hand-tracked ground truth is also shown. Figure A.59 shows the track positions in image co-ordinates for each frame against the ground truth.

There is a spike in RMS tracking errors shown in Figure A.60 caused by the target almost leaving the frame near the middle of the sequence. Due to drifting in the reference block during updating, the match position is not visible in the image anymore. The combination of reference block drift and the target being close to the edge of the frame is similar to a partial occlusion. Performance in this portion of the sequence, although not ideal, is adequate.



Figure A.58: *Contour Tracking – Composite* `tom4`

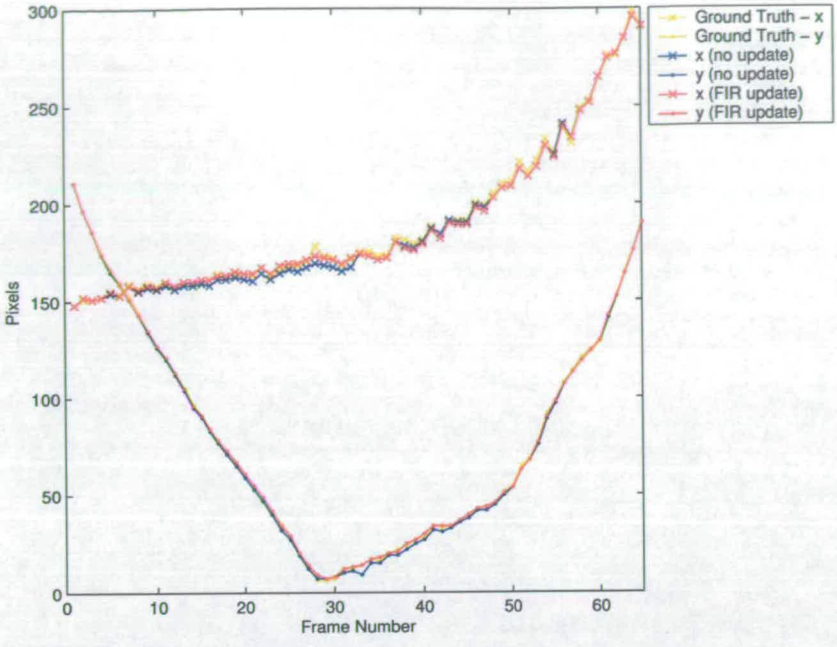


Figure A.59: *Contour Tracking – Track* `tom4`

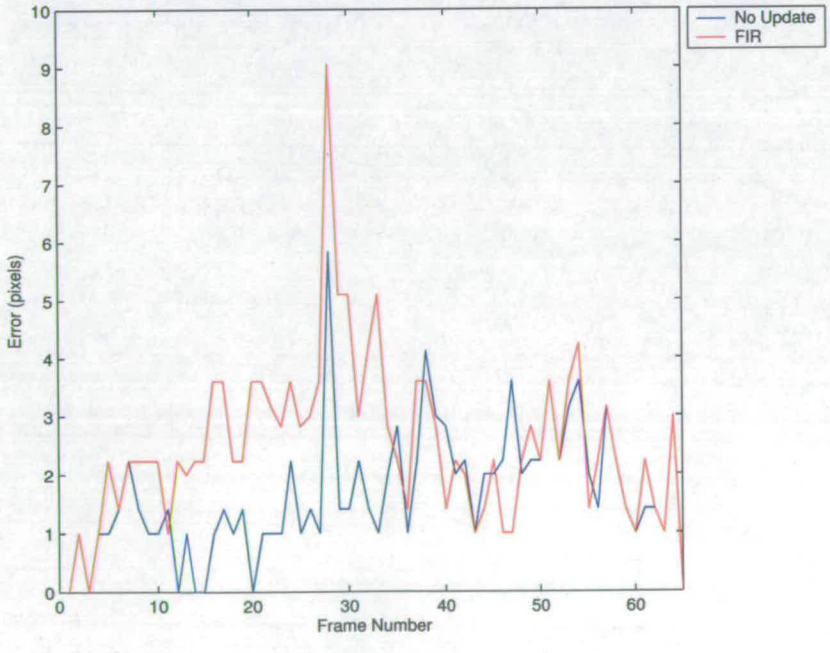


Figure A.60: *Contour Tracking – Errors* `tom4`

Appendix B
Numerical Optimisation

B.1 Introduction

It is often necessary to estimate the parameters of a non-linear function given a set of observations. This appendix deals with various non-linear methods of estimating parameters using methods such as Newton iteration and Levenberg-Marquardt iteration. A method to globally optimise multiple parameters using bundle adjustment is also introduced.

B.2 General optimisation problem

In most cases it is possible to estimate several parameters of some non-linear function by arranging them in a vector giving the vector relationship $\mathbf{X} = \mathbf{f}(\mathbf{P})$ where $\mathbf{X}$ is the measurement or observation vector and $\mathbf{P}$ is the parameter vector.

Normally $\mathbf{f}(\mathbf{P})$ takes the form of a cost function and the optimisation process attempts to find the estimated parameter vector $\hat{\mathbf{P}}$ such that $\hat{\mathbf{X}} = \mathbf{f}(\hat{\mathbf{P}}) - \epsilon$ where $\|\epsilon\|$ is minimised. An iterative process will attempt to minimise the error term ϵ by incrementally changing an initial parameter vector $\mathbf{P}_0$. The iterative process is stopped by a condition such as a measure of the change in the value of the cost function or that the maximum number of iterations has been reached.

B.3 Newton's Method

As described above there is a cost function $\mathbf{X} = \mathbf{f}(\mathbf{P})$ which will be used to estimate the parameter $\mathbf{P}$, starting with the initial parameter vector $\mathbf{P}_0$ by minimising the error. The dimensions of $\mathbf{P}$ and $\mathbf{X}$ can be different. Assuming the function $\mathbf{f}(\mathbf{P})$ is locally linear, the following second order Taylor series can approximate the estimate of the cost function with a small increment Δ in the parameter vector:

$$\mathbf{f}(\mathbf{P} + \Delta) \approx \mathbf{f}(\mathbf{P}) + \mathbf{g}\Delta + \tfrac{1}{2}\Delta^{\mathsf{T}}\mathbf{H}\Delta \tag{B.1}$$

$$\text{where} \quad \mathbf{g} \equiv \frac{d\mathbf{f}(\mathbf{P})}{d\mathbf{P}} \quad \text{gradient vector}$$
$$\mathbf{H} \equiv \frac{d^2\mathbf{f}(\mathbf{P})}{d\mathbf{P}^2} \quad \text{Hessian matrix}$$

The solution is located at the cost function's local minimum so finding the stationary point (where the derivatives are zero) results in the Newton step:

$$\mathbf{\Delta} = -\mathbf{H}^{-1}\mathbf{g} \tag{B.2}$$

Iterating the Newton step and updating the parameter vector until the stationary point is reached is Newton's Method. Once the cost function is stationary the current estimate of the parameter will be the solution to the optimisation. See Hartley and Zisserman[34] Appendix 6 for a full derivation.

B.3.1 Gauss-Newton Method

Given a cost function $\mathbf{X} = \mathbf{f}(\mathbf{P})$ where $\mathbf{X}$ is the measurement vector and $\mathbf{P}$ is the parameter vector. We wish to find the estimated parameter vector $\hat{\mathbf{P}}$ such that $\mathbf{X} = \mathbf{f}(\hat{\mathbf{P}}) - \epsilon$ for which $\|\epsilon\|$ is minimised. The residual $\epsilon = \overline{\mathbf{X}} - \hat{\mathbf{X}}$ where $\overline{\mathbf{X}}$ is the true value of estimated $\hat{\mathbf{X}} = \mathbf{f}(\hat{\mathbf{P}})$.

The approach to the problem is similar to solving a linear least squares method, however no closed form solution exists as the cost function is non-linear, so an iterative method must be used. Consider the following cost function with a weighting matrix $\mathbf{W}$ and prediction error ϵ as defined above:

$$\mathbf{f}(\mathbf{P}) = \tfrac{1}{2}\epsilon^{\mathsf{T}}\mathbf{W}\epsilon \tag{B.3}$$

The Jacobian matrix $\mathbf{J} \equiv \frac{\partial \mathbf{f}}{\partial \mathbf{P}}$ can be used to approximate the second order gradient vector $\mathbf{g}$ and Hessian matrix $\mathbf{H}$ as:

$$\mathbf{g} = \frac{d\mathbf{f}}{d\mathbf{P}} = \epsilon^{\mathsf{T}}\mathbf{W}\mathbf{J} \tag{B.4}$$

$$\mathbf{H} = \frac{d^2\mathbf{f}}{d\mathbf{P}^2} = \mathbf{J}^{\mathsf{T}}\mathbf{W}\mathbf{J} + \sum_i (\epsilon^{\mathsf{T}}\mathbf{W})_i \frac{\partial^2 \epsilon_i}{\partial \mathbf{P}^2} \tag{B.5}$$

The $\frac{\partial^2 \epsilon_i}{\partial \mathbf{P}^2}$ is normally small and dropped to give the first order Gauss-Newton approximation,

which are also known as the normal equations:

$$\mathbf{J}^{\mathsf{T}}\mathbf{W}\mathbf{J}\boldsymbol{\Delta} = -\mathbf{J}^{\mathsf{T}}\mathbf{W}\epsilon \tag{B.6}$$

The Hessian matrix $\mathbf{H}$ is now approximated by the Jacobian as $\mathbf{J}^{\mathsf{T}}\mathbf{W}\mathbf{J}$ which is advantageous as the second derivative of the cost function is sometimes hard to calculate or implement[43]. An initial value for the parameter vector $\mathbf{P_0}$ is chosen and the solution to the problem is found on successive iterations of $\mathbf{P}_{n+1} = \mathbf{P}_n + \Delta_n$. If the initial parameter vector places the function in a local minimum it will eventually converge to a solution.

B.3.1.1 Weighted iteration

If the measurement vector $\mathbf{X}$ is assumed to have a Gaussian distribution with a covariance matrix[1] $\Sigma_{\mathbf{X}}$ then it is possible to minimise the Mahalanobis distance $\|\mathbf{f}(\hat{\mathbf{P}}) - \mathbf{X}\|_{\Sigma}$. The weighting matrix $\mathbf{W} = \Sigma^{-1}$ which gives the normal equations as:

$$\mathbf{J}^{\mathsf{T}}\Sigma^{-1}\mathbf{J}\boldsymbol{\Delta} = -\mathbf{J}^{\mathsf{T}}\Sigma^{-1}\epsilon \tag{B.7}$$

B.3.1.2 Parameter Covariance

The inverse of the Hessian matrix $\mathbf{H}^{-1}$ gives an indication of the covariance of the parameters. As most optimisation problems are overdetermined the parameter covariance matrix is the Moore-Penrose pseudoinverse of the Hessian:

$$\boldsymbol{\Sigma}_{\mathbf{P}} = \mathbf{H}^{+} = (\mathbf{J}^{\mathsf{T}}\Sigma^{-1}{}_{\mathbf{X}}\mathbf{J})^{+} \tag{B.8}$$

With partitioned parameter vectors and sparse implementations performance improvements are possible as the entire Hessian need not be calculated. This is discussed for several types of parameter partition methods in Hartley and Zisserman[34] Appendix 6.3.

[1] For most image processing problems the measurement matrix $\mathbf{X}$ is normally distributed and observations are close to their true values, so the covariance matrix $\Sigma_{\mathbf{X}}$ can be approximated by the identity matrix when it is not possible to calculate it directly.

B.4 Levenberg-Marquardt minimisation

Newton's method has a fixed step size and often takes many iterations to approach the local minimum given less than optimal initial conditions. Levenberg-Marquardt minimisation uses a mixture of gradient descent and Gauss-Newton update equations to approach the local minimum quickly. A scalar parameter λ is introduced to increase or decrease the step size taken in each iteration. The normal equations are *augmented* to become:

$$(\mathbf{J}^\top \mathbf{W} \mathbf{J} + \lambda \mathbf{I})\boldsymbol{\Delta} = -\mathbf{J}^\top \mathbf{W} \epsilon \tag{B.9}$$

The value of λ is initially chosen to be small, say 10^{-3}. After an iteration step if the error has decreased then the step size is reduced by a factor of 10 and the normal equations approximate those of a Gauss-Newton method. If the error has increased then the step sized is increased by a factor of 10 and the iteration method now switches to gradient descent hopefully moving towards the solution quicker. Triggs[43] offers several implementations of Levenberg-Marquardt iteration with different convergence properties.

B.5 Block Matrices

Often matrices have a structure that allows them to be divided up into sub-matrices or blocks. When a block matrix is sparse, i.e. many of its elements are zero, matrix operations such as inversion can be implemented to execute very quickly.

When a block matrix only has non-zero entries on its diagonal the inverse is found by inverting each of the matrices on the diagonal. Computationally this is many times faster than inverting the entire matrix.

$$\mathbf{A}^{-1} = \begin{bmatrix} \mathbf{A_{11}} & & & \\ & \mathbf{A_{22}} & & \\ & & \ddots & \\ & & & \mathbf{A_{nn}} \end{bmatrix}^{-1} = \begin{bmatrix} \mathbf{A_{11}^{-1}} & & & \\ & \mathbf{A_{22}^{-1}} & & \\ & & \ddots & \\ & & & \mathbf{A_{nn}^{-1}} \end{bmatrix} \tag{B.10}$$

This property is useful in estimating parameters that depend only on one other parameter in an optimisation. This leads to a block matrix structure and the inverse can be found quickly by

evaluating only portions of the matrix as the other portions do not contribute to the inverse.

Methods such as QR decomposition, Gauss-Jordan elimination or Cholesky decomposition can be applied recursively to block matrices as if they were normal matrices. For example, to find the Cholesky decomposition of a block matrix, find the Cholesky decomposition of the block matrix as if it were a normal matrix and then find the Cholesky decomposition of each of the element matrices of the factorised block matrix. Triggs[43] suggests a method of recursive Cholesky decomposition for block matrices that recursively performs Cholesky decompositions on the elements of the matrix until the element is scalar. The Cholesky decomposition of a scalar is the square root of the value. This method only works with symmetrical positive definite matrices. It happens that these are common structures of the Hessian matrix in bundle adjustment problems. Often variables can be re-ordered to create a matrix which can be factored by Cholesky decomposition, resulting in speed improvements of several times.

B.6 Numerical Jacobian and Hessian Matrices

For many problems closed form solutions for the Jacobian and Hessian matrices can be implemented in simple cases. Most implementations (for example LINPACK[169] and Numerical Recipes in C[115]) use numerical approximations for the Jacobian, calculated by forward differences.

B.6.1 Jacobian Matrix

The Jacobian $\mathbf{J} = \frac{\partial \mathbf{f}(\mathbf{x})}{\partial \mathbf{x}}$ is the partial vector derivative of the vector function $\mathbf{y} = \mathbf{f}(\mathbf{x}) = (y_1, y_2, \ldots, y_n)^\mathsf{T}$ with respect to $\mathbf{x} = (x_1, x_2, \ldots, x_m)^\mathsf{T}$ with dimensions n and m respectively. This results in an $m \times n$ matrix where $\frac{\partial \mathbf{y}_i}{\partial \mathbf{x}_j}$ is the i-th dimension of the partial derivative of $\mathbf{f}$ with respect to the j-th dimension of x:

$$\mathbf{J} = \begin{bmatrix} \frac{\partial \mathbf{y}_1}{\partial \mathbf{x}_1} & \frac{\partial \mathbf{y}_2}{\partial \mathbf{x}_1} & \cdots & \frac{\partial \mathbf{y}_n}{\partial \mathbf{x}_1} \\ \frac{\partial \mathbf{y}_1}{\partial \mathbf{x}_2} & \frac{\partial \mathbf{y}_2}{\partial \mathbf{x}_2} & \cdots & \frac{\partial \mathbf{y}_n}{\partial \mathbf{x}_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \mathbf{y}_1}{\partial \mathbf{x}_m} & \frac{\partial \mathbf{y}_2}{\partial \mathbf{x}_m} & \cdots & \frac{\partial \mathbf{y}_n}{\partial \mathbf{x}_m} \end{bmatrix} \tag{B.11}$$

Calculating a numerical solution using a step size of δ gives the Jacobian as a matrix of row

vectors:[2]

$$\mathbf{J} = \begin{bmatrix} \left(\frac{\mathbf{f}\left((x_1+\delta,x_2,\ldots,x_m)^\mathsf{T}\right)-\mathbf{f}(\mathbf{x})}{\delta}\right)^\mathsf{T} \\ \left(\frac{\mathbf{f}\left((x_1,x_2+\delta,\ldots,x_m)^\mathsf{T}\right)-\mathbf{f}(\mathbf{x})}{\delta}\right)^\mathsf{T} \\ \vdots \\ \left(\frac{\mathbf{f}\left((x_1,x_2,\ldots,x_m+\delta)^\mathsf{T}\right)-\mathbf{f}(\mathbf{x})}{\delta}\right)^\mathsf{T} \end{bmatrix} \tag{B.12}$$

From this it can be seen that the Jacobian matrix for vector functions is the change in each dimension of the function value $\mathbf{y}$ with respect to the change in each dimension of the parameter vector $\mathbf{x}$. For example, in a structure from motion problem two Jacobians exist giving the change in a camera matrix with respect to a change in each dimension of the point and vice versa.

B.6.2 Hessian Matrix

The Hessian $\mathbf{H} = \frac{\partial^2 \mathbf{f}(\mathbf{x})}{\partial \mathbf{x}^2}$ is the second partial vector derivative of the vector function $\mathbf{y} = \mathbf{f}(\mathbf{x}) = (y_1, y_2, \ldots, y_n)^\mathsf{T}$ with respect to $\mathbf{x} = (x_1, x_2, \ldots, x_m)^\mathsf{T}$ with dimensions n and m respectively. This results in an $m \times n$ matrix where $\frac{\partial^2 \mathbf{y}_i}{\partial \mathbf{x}_j^2}$ is the i-th dimension of the partial derivative of $\mathbf{f}$ with respect to the j-th dimension of x:

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 \mathbf{y}_1}{\partial \mathbf{x}_1^2} & \frac{\partial^2 \mathbf{y}_2}{\partial \mathbf{x}_1^2} & \cdots & \frac{\partial^2 \mathbf{y}_n}{\partial \mathbf{x}_1^2} \\ \frac{\partial^2 \mathbf{y}_1}{\partial \mathbf{x}_2^2} & \frac{\partial^2 \mathbf{y}_2}{\partial \mathbf{x}_2^2} & \cdots & \frac{\partial^2 \mathbf{y}_n}{\partial \mathbf{x}_2^2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 \mathbf{y}_1}{\partial \mathbf{x}_m^2} & \frac{\partial^2 \mathbf{y}_2}{\partial \mathbf{x}_m^2} & \cdots & \frac{\partial^2 \mathbf{y}_n}{\partial \mathbf{x}_m^2} \end{bmatrix} \tag{B.13}$$

The finite difference approximation is given below with the step size δ is defined as in the Jacobian above:

[2] Hartley and Zisserman[34] suggest using values of 10^{-6} or $(10^{-4} \times x_i)$ for δ.

$$\mathbf{H} = \begin{bmatrix} \left(\left(\frac{\mathbf{f}\left((x_1-\delta,x_2,\ldots,x_m)^\mathsf{T}\right)-\mathbf{f}(\mathbf{x})}{\delta} - \frac{\mathbf{f}(\mathbf{x})-\mathbf{f}\left((x_1+\delta,x_2,\ldots,x_m)^\mathsf{T}\right)}{\delta} \right)/2\delta \right)^\mathsf{T} \\ \vdots \\ \left(\left(\frac{\mathbf{f}\left((x_1,x_2,\ldots,x_m-\delta)^\mathsf{T}\right)-\mathbf{f}(\mathbf{x})}{\delta} - \frac{\mathbf{f}(\mathbf{x})-\mathbf{f}\left((x_1,x_2,\ldots,x_m+\delta)^\mathsf{T}\right)}{\delta} \right)/2\delta \right)^\mathsf{T} \end{bmatrix} \tag{B.14}$$

B.7 Bundle adjustment

Bundle adjustment was developed by Brown[163] for the U.S. Air Force in the late 1950's to align images for aerial cartography. Triggs[43] provides a thorough review of bundle adjustment methods and the historical aspects of their use. Recently bundle adjustment has been largely used for structure from motion and self-calibration problems[27–29]. Most current methods use some form of non-linear least squares minimisation to do the bundle adjustment, these include methods such as Gauss-Seidel iteration or Levenberg-Marquardt minimisation. For many types of problems, profile Cholesky decomposition has been used to improve performance by a factor of two.

This section will discuss two implementations of bundle adjustment that can be used with 2D homographies or 3D structure from motion problems. The first type with two partitions is useful when optimising a single camera matrix or homography over several 2D or 3D points. The second type is a structure from motion type problem that globally estimates several camera parameters with image and global points.

For most structure from motion problems, points are not always visible in all camera views. This results in sparse matrix structures for optimisation, which provides a useful performance increase as not all elements of every matrix need to be calculated. Sparse techniques also allow recursive factorisation to used to be find matrix inverses.

The notation used is based on that used by Hartley and Zisserman[34] and a summary can be found at the front of this thesis.

Bundle adjustment problems can be solved by using Levenberg-Marquardt iteration. The normal equations of Equation B.6 are augmented with the weighting parameter $(1 + \lambda)$ to give Equation B.7. The weighting matrix is the covariance of the measurement parameter $\mathbf{X}$, which

for a structure from motion type problem is considered to be the identity matrix. The normal equations then simplify as follows:

$$\begin{aligned}(\mathbf{J}^\mathsf{T}\mathbf{W}\mathbf{J} + \lambda\mathbf{I})\boldsymbol{\Delta} &= -\mathbf{J}^\mathsf{T}\mathbf{W}\epsilon \\ (\mathbf{J}^\mathsf{T}\boldsymbol{\Sigma}_\mathbf{X}^{-1}\mathbf{J} + \lambda\mathbf{I})\boldsymbol{\Delta} &= -\mathbf{J}^\mathsf{T}\boldsymbol{\Sigma}_\mathbf{X}^{-1}\epsilon \\ (\mathbf{J}^\mathsf{T}\mathbf{J} + \lambda\mathbf{I})\boldsymbol{\Delta} &= -\mathbf{J}^\mathsf{T}\epsilon \end{aligned} \tag{B.15}$$

where	$\mathbf{J}^\mathsf{T}$	Jacobian of the parameter vector $\mathbf{P}$
	ϵ	Residual error vector, $\epsilon = \mathbf{X} - \hat{\mathbf{X}}$
	$\mathbf{X}$	True observed measurement vector
	$\hat{\mathbf{X}}$	Estimated measurement vector
	$\boldsymbol{\Sigma}_\mathbf{X}^{-1}$	Covariance of measurement vector $\mathbf{X}$, which is the identity matrix
	$\boldsymbol{\Delta}$	Increment vector to update parameter vector $\mathbf{P}$ after each iteration
	λ	Step size control

The parameter λ controls the step size when the Levenberg-Marquardt algorithm is in gradient descent mode. This is achieved by multiplying the diagonal entries of the Hessian matrix by $(1 + \lambda)$. Large values of λ help in ensuring that the Hessian is positive definite, resulting in a unique solution for $\boldsymbol{\Delta}$.

B.7.1 Partitioning parameters

The purpose of a bundle adjustment problem is to estimate the values of the parameter vector $\mathbf{P}$ given an observation vector $\mathbf{X}$. For example, if the parameters being optimised are camera parameters for two cameras and the observation vector is a list of points visible in each camera view, then it is logical to partition them as follows:

$$\begin{aligned}\mathbf{P} &= (\text{camera } 1, \text{camera } 2)^\mathsf{T} \\ \mathbf{X} &= (\text{point } 1, \text{point } 2, \ldots, \text{point } 9)^\mathsf{T}\end{aligned}$$

If each of the camera parameters is a 3-vector and the nine observations are 2-vectors, the above expressions would expand to:

$$\mathbf{P} = (c_{11}, c_{12}, c_{13}, c_{21}, c_{22}, c_{23})^{\mathsf{T}}$$
$$\mathbf{X} = (x_{11}, x_{12}, x_{21}, x_{22}, \ldots, x_{91}, x_{92})^{\mathsf{T}}$$

The camera parameters for each camera are independent (as are the observations) therefore the dependence of each parameter on every other parameter can be viewed as the covariance of $\mathbf{P}$, shown in Figure B.1.

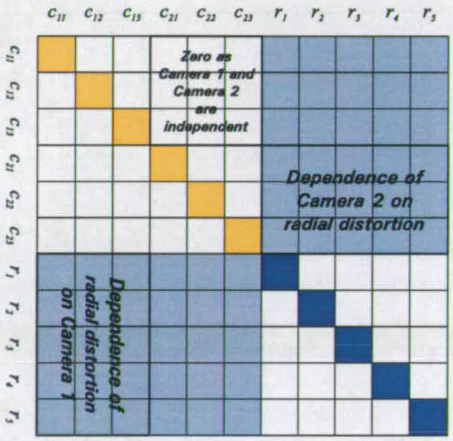


Figure B.1: *Dependence of each parameter on other parameters in the parameter vector*

It is also possible to include parameters which are common to both cameras, such as radial distortion. If the cameras are identical (or the same camera which has moved) and the radial distortion vector is defined as $\mathbf{R} = (r_1, r_2, r_3, r_4, r_5)^{\mathsf{T}}$, the parameter vector becomes:

$$\mathbf{P} = (c_{11}, c_{12}, c_{13}, c_{21}, c_{22}, c_{23}, r_1, r_2, r_3, r_4, r_5)^{\mathsf{T}}$$

If the observations are included as parameters in the optimisation, as well as the camera and radial distortion parameters, then the covariance matrix will have a structure as shown in Figure B.2.

The partitioning of parameters is important as the arrangement of parameters can affect the performance of the non-linear minimisation process. Triggs[43] covers methods of re-sectioning

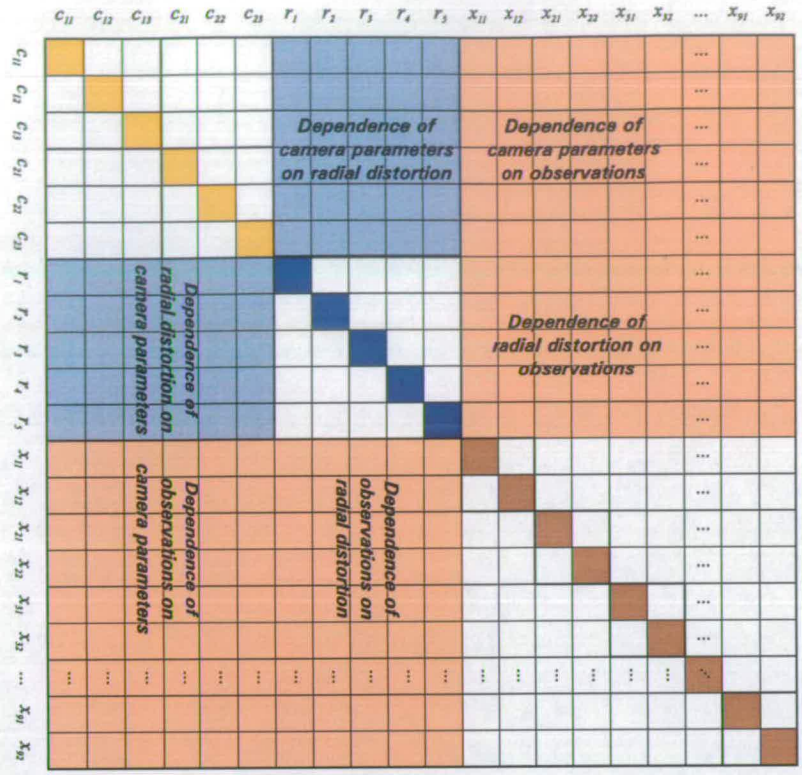


Figure B.2: *Dependence of each parameter on other parameters in the parameter vector*

of parameters and re-ordering parameters to allow methods such as profile Cholesky decomposition to be used for performance gains.

Most camera calibration and structure from motion problems can be solved by partitioning the parameter vector into either two or three partitions. Each of these schemes is discussed later in Section B.7.3 and Section B.7.4 respectively.

B.7.2 Implementation

An outline of an implementation of a bundle adjustment algorithm is shown in pseudocode in Figure B.3. All of the Jacobians and the Hessian matrix should be implemented as sparse matrices. This algorithm is redefined in Section B.7.3 and Section B.7.4 for two and three parameter vector partitions respectively. It is recommended that the sparse matrix classes from VXL[21] are used in a software implementation. In order to invert each matrix in a block matrix, QR-decomposition should be used – VXL provides a reliable implementation of this method. The Jacobians can be calculated by forward difference or the analytical solutions can be used. Sometimes, the analytical solutions are more stable, however, they require extra work.

1. Partition parameter vector $\mathbf{P}$ into either two or three partitions: $\mathbf{P} = (\mathbf{a}^\mathsf{T}, \mathbf{b}^\mathsf{T})^\mathsf{T}$ or $\mathbf{P} = (\mathbf{a}^\mathsf{T}, \mathbf{b}^\mathsf{T}, \mathbf{c}^\mathsf{T})^\mathsf{T}$
2. Set step control parameter $\lambda = 0.001$
3. Find the estimate of the observation vector $\hat{\mathbf{X}} = \mathbf{f}(\mathbf{P})$
4. Calculate the following:
   (a) Residual error between estimate of observations and the true observations $\epsilon = \mathbf{X} - \hat{\mathbf{X}}$
   (b) Jacobians $\mathbf{J}$ of each partition, i.e. $\frac{\partial \hat{\mathbf{X}}}{\partial \mathbf{a}}$, $\frac{\partial \hat{\mathbf{X}}}{\partial \mathbf{b}}$ (and $\frac{\partial \hat{\mathbf{X}}}{\partial \mathbf{c}}$)
   (c) Create Hessian matrix from the Jacobian block matrices $\mathbf{H} = \mathbf{J}^\mathsf{T}\mathbf{J}$
5. Augment the Hessian matrix by multiplying $\mathbf{H}$ by $(1 + \lambda)$
6. Find the inverse of the Hessian matrix and calculate the parameter increments δ for each partition by matrix factorisation
7. Update the parameter vector by adding the increments $\hat{\mathbf{P}}_{n+1} = \hat{\mathbf{P}}_n + \delta$
8. Find the new estimate of the observation vector $\hat{\mathbf{X}}_{\mathbf{n+1}} = \mathbf{f}(\mathbf{P}_{n+1})$
9. Calculate the new residual error
   (a) If the residual is smaller, then accept the new value of $\mathbf{P}$ and divide λ by 10
   (b) If the residual is larger, then revert to the old value of $\mathbf{P}$ and multiply λ by 10
10. Repeat from step 2 until the residual error is below is threshold or a maximum number of iterations is reached

Figure B.3: *Outline of a bundle adjustment algorithm*

Some examples of analytical solutions for the Jacobians for camera calibration are shown in Appendix C.

B.7.3 Bundle Adjustment with two partitions

The parameter vector in a two partition bundle adjustment should be divided into camera parameters and global feature points. The function $\mathbf{f}(\mathbf{P})$ transforms the global feature points into the points in each image $\mathbf{X}$. The parameter vector can be divided into two partitions for each part of the problem (camera matrices and global points), $\mathbf{P} = (\mathbf{a}^\mathsf{T}, \mathbf{b}^\mathsf{T})^\mathsf{T}$, where

$\mathbf{a}$ $1 \ldots m$ camera matrices (j-th image or view)

$\mathbf{b}$ $1 \ldots n$ optimised global points (i-th point)

$\mathbf{X}$ $1 \ldots p$ measurement points (k-th measurement)

A modification of the algorithm shown in Figure B.3 for two partitions is presented in Figure B.4.

The normal equations $(\mathbf{J}^\mathsf{T}\mathbf{J} + \lambda \mathtt{I})\boldsymbol{\Delta} = \mathbf{J}^\mathsf{T}\epsilon$ can be represented using block matrices as follows:

$$\begin{bmatrix} \mathtt{U}^* & \mathtt{W} \\ \mathtt{W}^\mathsf{T} & \mathtt{V}^* \end{bmatrix} \begin{pmatrix} \delta_\mathbf{a} \\ \delta_\mathbf{b} \end{pmatrix} = \begin{pmatrix} \epsilon_\mathtt{A} \\ \epsilon_\mathtt{B} \end{pmatrix} \tag{B.16}$$

where:

$$\mathtt{A} = \frac{\partial \hat{\mathbf{X}}}{\partial \mathbf{a}} = \Big[p \times m\Big] \text{block matrix}$$

$$\mathtt{B} = \frac{\partial \hat{\mathbf{X}}}{\partial \mathbf{b}} = \Big[p \times n\Big] \text{block matrix}$$

$$\mathtt{U} = \mathtt{A}^\mathsf{T}\mathtt{A} = \Big[m \times m\Big] \text{block matrix}$$

$$\mathtt{V} = \mathtt{B}^\mathsf{T}\mathtt{B} = \Big[n \times n\Big] \text{block matrix}$$

$$\mathtt{W} = \mathtt{A}^\mathsf{T}\mathtt{B} = \Big[m \times p\Big]\Big[p \times n\Big] = \Big[m \times n\Big] \text{block matrix}$$

$$\mathtt{U}^* = \mathtt{U}(1 + \lambda)$$

$$\mathtt{V}^* = \mathtt{V}(1 + \lambda)$$

1. Partition parameter vector $\mathbf{P}$ into two partitions: $\mathbf{P} = (\mathbf{a}^\mathsf{T}, \mathbf{b}^\mathsf{T})^\mathsf{T}$
2. Set step control parameter $\lambda = 0.001$
3. Find the estimate of the observation vector $\hat{\mathbf{X}} = \mathbf{f}(\mathbf{P})$
4. Calculate the following:
   (a) Residual error between estimate of observations and the true observations $\epsilon = \mathbf{X} - \hat{\mathbf{X}}$
   (b) Jacobians $\mathbf{J}$ of each partition: $\frac{\partial \hat{\mathbf{X}}}{\partial \mathbf{a}}$ and $\frac{\partial \hat{\mathbf{X}}}{\partial \mathbf{b}}$
   (c) $\mathtt{U} = \mathtt{A}^\mathsf{T}\mathtt{A} \qquad \mathtt{V} = \mathtt{B}^\mathsf{T}\mathtt{B}$
   (d) $\epsilon_\mathtt{A} = \mathtt{A}^\mathsf{T}\epsilon \qquad \epsilon_\mathtt{B} = \mathtt{B}^\mathsf{T}\epsilon$
5. Augment the blocks of the Hessian matrix by multiplying each by $(1+\lambda)$ to get $\mathtt{U}^*$ and $\mathtt{V}^*$
6. Find the parameter increments $\delta_\mathbf{a}$ and $\delta_\mathbf{b}$ by solving the following by QR-decomposition and back-substitution:
   (a) $(\mathtt{U}^* - \mathtt{W}\mathtt{V}^{*-1}\mathtt{W}^\mathsf{T})\delta_\mathbf{a} = \epsilon_\mathtt{A} - \mathtt{W}\mathtt{V}^{*-1}\epsilon_\mathtt{B}$
   (b) $\mathtt{V}^*\delta_\mathbf{b} = \epsilon_\mathtt{B} - \mathtt{W}^\mathsf{T}\delta_\mathbf{a}$
7. Update the parameter vector by adding the increments $\hat{\mathbf{P}}_{n+1} = ((\mathbf{a} + \delta_\mathbf{a})^\mathsf{T}, (\mathbf{b} + \delta_\mathbf{b})^\mathsf{T})^\mathsf{T}$
8. Find the new estimate of the observation vector $\hat{\mathbf{X}}_{\mathbf{n+1}} = \mathbf{f}(\mathbf{P}_{n+1})$
9. Calculate the new residual error
   (a) If the residual is smaller, then accept the new value of $\mathbf{P}$ and divide λ by 10 and go to step 3 or terminate if residual below a threshold
   (b) If the residual is larger, then revert to the old value of $\mathbf{P}$ and multiply λ by 10 and go to step 5 or terminate if residual below a threshold

Figure B.4: *Bundle adjustment algorithm for two partitions*

In order to solve for the parameter increments $\delta_{\mathbf{a}}$ and $\delta_{\mathbf{b}}$, the matrix on the left-hand side needs to be factored:

$$\begin{bmatrix} \mathtt{I} & -\mathtt{W}\mathtt{V}^{*-1} \\ \mathtt{0} & \mathtt{I} \end{bmatrix} \begin{bmatrix} \mathtt{U}^* & \mathtt{W} \\ \mathtt{W}^\mathsf{T} & \mathtt{V}^* \end{bmatrix} \begin{pmatrix} \delta_{\mathbf{a}} \\ \delta_{\mathbf{b}} \end{pmatrix} = \begin{bmatrix} \mathtt{I} & -\mathtt{W}\mathtt{V}^{*-1} \\ \mathtt{0} & \mathtt{I} \end{bmatrix} \begin{pmatrix} \epsilon_{\mathtt{A}} \\ \epsilon_{\mathtt{B}} \end{pmatrix}$$

$$\begin{bmatrix} \mathtt{U}^* - \mathtt{W}\mathtt{V}^{*-1}\mathtt{W}^\mathsf{T} & \mathtt{0} \\ \mathtt{W}^\mathsf{T} & \mathtt{V}^* \end{bmatrix} \begin{pmatrix} \delta_{\mathbf{a}} \\ \delta_{\mathbf{b}} \end{pmatrix} = \begin{pmatrix} \epsilon_{\mathtt{A}} - \mathtt{W}\mathtt{V}^{*-1}\epsilon_{\mathtt{B}} \\ \epsilon_{\mathtt{B}} \end{pmatrix} \tag{B.17}$$

This allows $\delta_{\mathbf{a}}$ to be found first by finding the QR-decomposition of the left-hand side of the expression below and back-substituting:

$$(\mathtt{U}^* - \mathtt{W}\mathtt{V}^{*-1}\mathtt{W}^\mathsf{T})\delta_{\mathbf{a}} = \epsilon_{\mathtt{A}} - \mathtt{W}\mathtt{V}^{*-1}\epsilon_{\mathtt{B}} \tag{B.18}$$

By substituting $\delta_{\mathbf{a}}$ into the following expression, it is possible to find $\delta_{\mathbf{b}}$ by the same QR-decomposition method as above:

$$\mathtt{V}^*\delta_{\mathbf{b}} = \epsilon_{\mathtt{B}} - \mathtt{W}^\mathsf{T}\delta_{\mathbf{a}} \tag{B.19}$$

The parameter vector $\mathbf{P}$ can then be updating as follows:

$$\mathbf{P} = ((\mathbf{a} + \delta_{\mathbf{a}})^\mathsf{T}, (\mathbf{b} + \delta_{\mathbf{b}})^\mathsf{T})^\mathsf{T} \tag{B.20}$$

It is then possible to iterate until the residual error drops below a threshold or a number of iterations have occurred. The parameter $\mathbf{P}$ will then represent the globally optimal solution which minimises the residual error produced by the function $\mathbf{f}(\mathbf{P}) - \mathbf{X}$. In the case of camera calibration, this will represent the minimum reprojection error between global features and the feature positions in each view.

B.7.4 Bundle Adjustment with three partitions

In many bundle adjustment problems there are parameters which are independent (such as the camera parameters for each camera view) and those which are common to all camera views (such as the radial distortion of the camera lens). For example, in a camera calibration, the rotation parameters for each camera view are not dependent on each other, although the rotation

does depend on the estimate of the radial distortion parameters. For this reason the parameter vector can be divided into three partitions for each part of the problem (camera matrices, points and fixed parameters such as radial distortion), $\mathbf{P} = (\mathbf{a}^\mathsf{T}, \mathbf{b}^\mathsf{T}, \mathbf{c}^\mathsf{T})^\mathsf{T}$, where

- **a** $1 \ldots m$ camera matrices (j-th image or view)
- **b** $1 \ldots q$ constant parameters over all images (h-th constant)
- **c** $1 \ldots n$ estimates of points in each image (i-th point)

A modification of the algorithm shown in Figure B.3 for two partitions is presented in Figure B.5.

The measurement vector $\mathbf{X}$ can be divided into $\mathbf{X_i} = (\mathbf{x}_{i1}^\mathsf{T}, \mathbf{x}_{i2}^\mathsf{T}, ..., \mathbf{x}_{im}^\mathsf{T})^\mathsf{T}$ where $\mathbf{x}_{ij}$ is the i-th point in the j-th image. However, not every point is visible in each image so the measurement vector is reduced to only points that exist. This results in the measurement vector of p points with $\mathbf{x}_k = \mathbf{x}_{ij}$ becoming the k-th point if the i-th point exists in the j-th view. If each measurement point is a 3-vector, $\mathbf{X}$ is vector of dimension $3p$.

The Jacobian matrices for each partition are given by:

$$\mathtt{A} = \frac{\partial \hat{\mathbf{X}}}{\partial \mathbf{a}} = \Big[p \times m\Big] \text{block matrix}$$
$$\mathtt{B} = \frac{\partial \hat{\mathbf{X}}}{\partial \mathbf{b}} = \Big[p \times q\Big] \text{block matrix}$$
$$\mathtt{C} = \frac{\partial \hat{\mathbf{X}}}{\partial \mathbf{c}} = \Big[p \times n\Big] \text{block matrix}$$

The Jacobian matrix is a block matrix, so a Jacobian for each partition can be calculated as follows for the k-th estimate of the measurement vector $\hat{\mathbf{x}}_k$:

$$\mathtt{A}_k = \frac{\partial \hat{\mathbf{x}}_k}{\partial \mathbf{a}_j} \quad \mathtt{B}_k = \frac{\partial \hat{\mathbf{x}}_k}{\partial \mathbf{b}_h} \quad \mathtt{C}_k = \frac{\partial \hat{\mathbf{x}}_k}{\partial \mathbf{c}_i} \tag{B.21}$$

To solve the normal equations $(\mathbf{J}^\mathsf{T}\mathbf{J} + \lambda \mathtt{I})\boldsymbol{\Delta} = \mathbf{J}^\mathsf{T}\epsilon$ for block matrices the Jacobian and Hessian matrices are found for each partition as follows. The structure of the Hessian matrix is shown in Figure B.6.

1. Partition parameter vector $\mathbf{P}$ into three partitions: $\mathbf{P} = (\mathbf{a}^\mathsf{T}, \mathbf{b}^\mathsf{T}, \mathbf{c}^\mathsf{T})^\mathsf{T}$
2. Set step control parameter $\lambda = 0.001$
3. Find the estimate of the observation vector $\hat{\mathbf{X}} = \mathbf{f}(\mathbf{P})$
4. Calculate the following:
   (a) Residual error between estimate of observations and the true observations $\epsilon = \mathbf{X} - \hat{\mathbf{X}}$
   (b) Jacobians $\mathbf{J}$ of each partition: $\mathtt{A} = \frac{\partial \hat{\mathbf{X}}}{\partial \mathbf{a}}$, $\mathtt{B} = \frac{\partial \hat{\mathbf{X}}}{\partial \mathbf{b}}$ and $\mathtt{C} = \frac{\partial \hat{\mathbf{X}}}{\partial \mathbf{c}}$
   (c) $\mathtt{U} = \mathtt{A}^\mathsf{T}\mathtt{A}$ $\quad \mathtt{V} = \mathtt{B}^\mathsf{T}\mathtt{B}$ $\quad \mathtt{R} = \mathtt{C}^\mathsf{T}\mathtt{C}$
   $\mathtt{W} = \mathtt{A}^\mathsf{T}\mathtt{B}$ $\quad \mathtt{P} = \mathtt{A}^\mathsf{T}\mathtt{C}$ $\quad \mathtt{Q} = \mathtt{B}^\mathsf{T}\mathtt{C}$
   $\epsilon_\mathtt{A} = \mathtt{A}^\mathsf{T}\epsilon$ $\quad \epsilon_\mathtt{B} = \mathtt{B}^\mathsf{T}\epsilon$ $\quad \epsilon_\mathtt{C} = \mathtt{C}^\mathsf{T}\epsilon$
5. Augment the blocks of the Hessian matrix by multiplying each by $(1 + \lambda)$ to get $\mathtt{U}^*$ and $\mathtt{V}^*$
6. Find the parameter increments $\delta_\mathbf{a}$ and $\delta_\mathbf{b}$ by solving the following by QR-decomposition and back-substitution:
   (a) $\left[\mathtt{U}^* - \mathtt{P}\mathtt{R}^{*-1}\mathtt{P}^\mathsf{T} - \mathtt{Y}(\mathtt{W}^\mathsf{T} - \mathtt{Q}\mathtt{R}^{*-1}\mathtt{P}^\mathsf{T})\right]\delta_\mathbf{a} = \epsilon_\mathtt{A} - \mathtt{Y}\epsilon_\mathtt{B} - \mathtt{P}\mathtt{R}^{*-1}\epsilon_\mathtt{C} + \mathtt{Y}\mathtt{Q}\mathtt{R}^{*-1}\epsilon_\mathtt{C}$
   (b) $\delta_\mathbf{b} = \left[\mathtt{V}^* - \mathtt{Q}\mathtt{R}^{*-1}\mathtt{Q}^\mathsf{T}\right]^{-1}\left[\epsilon_\mathtt{B} - \mathtt{Q}\mathtt{R}^{*-1}\epsilon_\mathtt{C} - (\mathtt{W}^\mathsf{T} - \mathtt{Q}\mathtt{R}^{*-1}\mathtt{P}^\mathsf{T})\delta_\mathbf{a}\right]$
   (c) $\delta_\mathbf{c} = \mathtt{R}^{*-1}(\epsilon_\mathtt{C} - \mathtt{P}^\mathsf{T}\delta_\mathbf{a} - \mathtt{Q}^\mathsf{T}\delta_\mathbf{b})$
7. Update the parameter vector by adding the increments: $\hat{\mathbf{P}}_{n+1} = ((\mathbf{a} + \delta_\mathbf{a})^\mathsf{T}, (\mathbf{b} + \delta_\mathbf{b})^\mathsf{T}, (\mathbf{c} + \delta_\mathbf{c})^\mathsf{T})^\mathsf{T}$
8. Find the new estimate of the observation vector $\hat{\mathbf{X}}_{\mathbf{n+1}} = \mathbf{f}(\mathbf{P}_{n+1})$
9. Calculate the new residual error
   (a) If the residual is smaller, then accept the new value of $\mathbf{P}$ and divide λ by 10 and go to step 3 or terminate if residual below a threshold
   (b) If the residual is larger, then revert to the old value of $\mathbf{P}$ and multiply λ by 10 and go to step 5 or terminate if residual below a threshold

Figure B.5: *Bundle adjustment algorithm for three partitions*

$$\begin{bmatrix} U^* & W & P \\ W^T & V^* & Q \\ P^T & Q^T & R^* \end{bmatrix} \begin{pmatrix} \delta_a \\ \delta_b \\ \delta_c \end{pmatrix} = \begin{pmatrix} \epsilon_A \\ \epsilon_B \\ \epsilon_C \end{pmatrix} \tag{B.22}$$

The Hessian matrix is also a block matrix and be approximated by the blocks of the Jacobian and the error $\epsilon_k = \mathbf{x_k} - \hat{\mathbf{x}}_\mathbf{k}$ for each point:

$$\begin{aligned} U = A^T A \quad V = B^T B \quad R = C^T C \\ W = A^T B \quad P = A^T C \quad Q = B^T C \\ \epsilon_A = A^T \epsilon \quad \epsilon_B = B^T \epsilon \quad \epsilon_C = C^T \epsilon \end{aligned}$$

$$Y = (W - PR^{*-1}Q^T)(V^* - QR^{*-1}Q^T)^{-1}$$

In order to find the parameter increment Δ it is necessary to invert the Hessian. By factoring the Hessian it is possible to isolate the first partition increment $\delta_\mathbf{a}$. After calculating $\delta_\mathbf{a}$ it is possible to back substitute for $\delta_\mathbf{b}$ and $\delta_\mathbf{c}$. Factoring the Hessian is done by performing block Jordan factoring in the same way used to solve linear equations by Gauss-Jordan elimination. By defining the matrix $Y = (W - PR^{*-1}Q^T)(V^* - QR^{*-1}Q^T)^{-1}$ and finding the upper matrix using Gauss-Jordan elimination and multiply the left hand side by it gives:

$$\begin{aligned} &\begin{bmatrix} I & -Y & -PR^{*-1} + YQR^{*-1} \\ 0 & I & -QR^{*-1} \\ 0 & 0 & I \end{bmatrix} \begin{bmatrix} U^* & W & P \\ W^T & V^* & Q \\ P^T & Q^T & R^* \end{bmatrix} \begin{pmatrix} \delta_a \\ \delta_b \\ \delta_c \end{pmatrix} \\ &= \begin{bmatrix} U^* - PR^{*-1}P^T - Y(W^T - QR^{*-1}P^T) & 0 & 0 \\ W^T - QR^{*-1}P^T & V^* - QR^{*-1}Q^T & 0 \\ P^T & Q^T & R^* \end{bmatrix} \begin{pmatrix} \delta_a \\ \delta_b \\ \delta_c \end{pmatrix} \end{aligned} \tag{B.23}$$

Multiplying the right hand side by the upper matrix results in:

$$\begin{bmatrix} I & -Y & -PR^{*-1} + YQR^{*-1} \\ 0 & I & -QR^{*-1} \\ 0 & 0 & I \end{bmatrix} \begin{pmatrix} \epsilon_A \\ \epsilon_B \\ \epsilon_C \end{pmatrix} = \begin{pmatrix} \epsilon_A - Y\epsilon_B - PR^{*-1}\epsilon_C + YQR^{*-1}\epsilon_C \\ \epsilon_B - QR^{*-1}\epsilon_C \\ \epsilon_C \end{pmatrix} \tag{B.24}$$

The problem has now been broken down in three independent equations that can be evaluated simultaneously:

$$\left[U^* - PR^{*-1}P^T - Y(W^T - QR^{*-1}P^T)\right]\delta_a = \epsilon_A - Y\epsilon_B - PR^{*-1}\epsilon_C + YQR^{*-1}\epsilon_C \tag{B.25}$$

$$\delta_b = \left[V^* - QR^{*-1}Q^T\right]^{-1}\left[\epsilon_B - QR^{*-1}\epsilon_C - (W^T - QR^{*-1}P^T)\delta_a\right] \tag{B.26}$$

$$\delta_c = R^{*-1}(\epsilon_C - P^T\delta_a - Q^T\delta_b) \tag{B.27}$$

It is possible to solve for δ_a from Equation B.25 by QR decomposition and back substitution. Equation B.26 and Equation B.27 can then be used to solve for δ_b and δ_c respectively. Each of the partitions of $\mathbf{P}$ are updated and the cost function is evaluated for $\mathbf{P} + \delta$. If the cost function has decreased then the step size λ is decrease by 10 and the incremented parameters are accepted. Otherwise it is increased by 10 and the incremented parameters are rejected. This process is repeated until a maximum number of iterations is performed or until the value of the cost function becomes stable.

The covariance of the parameter vector at each step is calculated by finding the inverse of the Hessian matrix as discussed in Appendix B.3.1.2. As the Hessian matrix has been factored into three independent equations (Equation B.25 – Equation B.27) it is possible to find the covariance of each partition of the parameter vector separately, without having to invert the entire Hessian matrix. The covariance of each partition of the parameter vector is found as follows:

$$\Sigma_a = \left[U - PR^{-1}P^T - Y(W^T - QR^{-1}P^T)\right]^+ \tag{B.28}$$

$$\Sigma_b = \left[V - QR^{-1}Q^T\right]^+ \tag{B.29}$$

$$\Sigma_c = R^+ \tag{B.30}$$

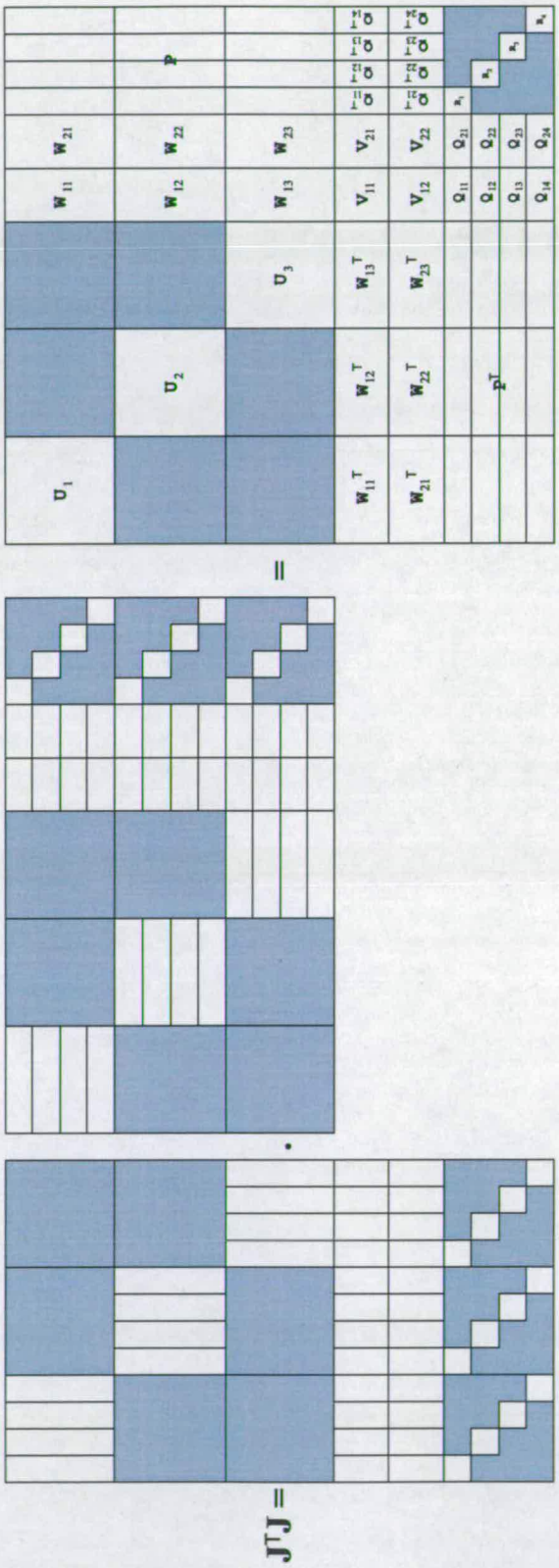


Figure B.6: *Structure of the Jacobians and Hessian matrix for three partitions*

Appendix C
Pan/Tilt/Zoom Camera Calibration

This appendix contains the details of several aspects of calibrating a PTZ camera using self-calibration methods and bundle adjustment. The following table summarises the analytical Jacobians which are used for each of the bundle adjustment steps during the calibration method described in Chapter 6:

Calibration Step	*Parameters optimised*	*Partitions*	
Pairwise homographies	Pairwise homographies	2	App C.1 pg 316
Bundle I	Homographies, Radial Distortion	3	App C.2 pg 318
Bundle II	Intrinsic Parameters, Radial Distortion, Principal Point	3	App C.3 pg 322
Zoom Final Bundle	All Intrinsic and Radial Distortion Parameters	2	App C.4 pg 326

A method for normalising a matrix so that its determinant is equal to one is given in Section C.5. This normalisation is necessary when using Hartley's linear method[30] for finding the rotation and intrinsic parameters from several pairwise homographies.

C.1 Jacobians – Pairwise homographies

The pairwise homography $\mathtt{H}$ projects a point $\mathbf{x}$ from image 1 to $\mathbf{x}'$ in image 2. As the observation of the true point in image 2 is an inhomogeneous point, $\mathbf{x}'$ is divided through by w' to give $\tilde{\mathbf{x}} = (\tilde{x}, \tilde{y})$. The following block diagram illustrates this process:

2D Point in Image 1	3-vector	$\mathbf{x} = (x, y, w)$
$\downarrow$		
$\boxed{\mathbf{x}' = \mathtt{H}\mathbf{x}}$		homography
$\downarrow$		
2D Point in Image 2	3-vector	$\mathbf{x}' = (x', y', w')$
$\downarrow$		
$\boxed{(x', y', w') \mapsto (\tilde{x}, \tilde{y})}$		projection
$\downarrow$		
Inhomogeneous 2D Image Point	2-vector	$\tilde{\mathbf{x}} = (\tilde{x}, \tilde{y})$

The Jacobians for each partition can be calculated easily by considering the structure of the function which maps the parameter vector to the estimate of the measurement vector:

$$\begin{aligned} \hat{\mathbf{X}} &= \mathbf{f}(\mathbf{P}) \\ &= \mathbf{f}(\mathbf{a}, \mathbf{b}) \\ &= \mathbf{f}(\mathbf{h}, \mathbf{X}, \mathbf{X}') \end{aligned} \tag{C.1}$$

The Jacobian matrix can be divided into a block matrix. The Jacobian of each sub-matrix is then calculated as follows for each partition:

$$\mathtt{A}_i = \frac{\partial \hat{\mathbf{X}}_i}{\partial \mathbf{h}} = \begin{bmatrix} \mathtt{0}_{2\times 9} \\ \frac{\partial \hat{\mathbf{x}}_i'}{\partial \mathbf{h}} \end{bmatrix} \tag{C.2}$$

$$\mathtt{B}_i = \frac{\partial \hat{\mathbf{X}}_i}{\partial \hat{\mathbf{x}}_i} = \begin{bmatrix} \mathtt{I}_{2\times 3} \\ \frac{\partial \hat{\mathbf{x}}_i'}{\partial \hat{\mathbf{x}}_i} \end{bmatrix} \tag{C.3}$$

C.1.1 Partition A

The derivation of the Jacobians for partition A are as follows:

$$\begin{aligned}\frac{\partial \hat{\mathbf{x}}}{\partial \mathbf{a}} &= \frac{\partial \tilde{\mathbf{x}}}{\partial \mathbf{a}} \\ &= \frac{\partial \tilde{\mathbf{x}}}{\partial \mathbf{x}'}\frac{\partial \mathbf{x}'}{\partial \mathbf{a}} \\ &= \frac{\partial \tilde{\mathbf{x}}}{\partial \mathbf{x}'}\left(\frac{\partial \mathbf{x}'}{\partial \mathbf{x}}\frac{\partial \mathbf{x}}{\partial \mathbf{a}} + \frac{\partial \mathbf{x}'}{\partial \mathbf{h}}\frac{\partial \mathbf{h}}{\partial \mathbf{a}}\right)\end{aligned}$$

But as $\mathbf{x}$ is independent of $\mathbf{a}$ this gives $\frac{\partial \mathbf{x}}{\partial \mathbf{a}} = \mathtt{0}$ and $\mathbf{a} = \mathbf{h}$ so $\frac{\partial \mathbf{h}}{\partial \mathbf{a}} = \mathtt{I}$. The expression above then simplifies to:

$$\frac{\partial \hat{\mathbf{x}}}{\partial \mathbf{a}} = \frac{\partial \tilde{\mathbf{x}}}{\partial \mathbf{x}'}\frac{\partial \mathbf{x}'}{\partial \mathbf{h}} \tag{C.4}$$

$$= \begin{bmatrix} \frac{1}{w'} & 0 & -\frac{x'}{w'^2} \\ 0 & \frac{1}{w'} & -\frac{y'}{w'^2} \end{bmatrix} \begin{bmatrix} x & y & w & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & x & y & w & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & x & y & w \end{bmatrix}$$

C.1.2 Partition B

The derivation of the Jacobians for partition B are as follows:

$$\begin{aligned}\frac{\partial \hat{\mathbf{x}}}{\partial \mathbf{b}} &= \frac{\partial \tilde{\mathbf{x}}}{\partial \mathbf{b}} \\ &= \frac{\partial \tilde{\mathbf{x}}}{\partial \mathbf{x}'}\frac{\partial \mathbf{x}'}{\partial \mathbf{b}} \\ &= \frac{\partial \tilde{\mathbf{x}}}{\partial \mathbf{x}'}\left(\frac{\partial \mathbf{x}'}{\partial \mathbf{x}}\frac{\partial \mathbf{x}}{\partial \mathbf{b}} + \frac{\partial \mathbf{x}'}{\partial \mathbf{h}}\frac{\partial \mathbf{h}}{\partial \mathbf{b}}\right)\end{aligned}$$

But $\mathbf{x} = \mathbf{b}$ so $\frac{\partial \mathbf{x}}{\partial \mathbf{b}} = \mathtt{I}$ and $\mathbf{b}$ is not dependent on $\mathbf{h}$ this gives $\frac{\partial \mathbf{h}}{\partial \mathbf{b}} = \mathtt{0}$. The expression above then simplifies to:

$$\frac{\partial \hat{\mathbf{x}}}{\partial \mathbf{b}} = \frac{\partial \tilde{\mathbf{x}}}{\partial \mathbf{x}'}\frac{\partial \mathbf{x}'}{\partial \mathbf{x}} \tag{C.5}$$

$$= \begin{bmatrix} \frac{1}{w'} & 0 & -\frac{x'}{w'^2} \\ 0 & \frac{1}{w'} & -\frac{y'}{w'^2} \end{bmatrix} \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix}$$

C.2 Jacobians – Bundle I

This bundle adjustment optimises the homographies between the reference view and each camera view as well as the radial distortion parameters (excluding the principal point) and each global feature position. The radial distortion parameters are common to each view and produce dense Jacobians. The block diagram below illustrates the transfer of a global feature in the reference to its position in a camera view:

2D Global Feature Point	3-vector	$\mathbf{x} = (x, y, w)$
$\downarrow$		
$\boxed{\mathbf{x}' = \mathtt{H}\mathbf{x}}$		homography
$\downarrow$		
2D Image Point	3-vector	$\mathbf{x}' = (x', y', w')$
$\downarrow$		
$\boxed{(x', y', w') \mapsto (\tilde{x}, \tilde{y})}$		projection
$\downarrow$		
Inhomogeneous 2D Image Point	2-vector	$\tilde{\mathbf{x}} = (\tilde{x}, \tilde{y})$
$\downarrow$		
$\boxed{\tilde{\mathbf{x}}_d = f(\tilde{\mathbf{x}}, \mathbf{c}) = f(\tilde{\mathbf{x}}, \kappa_1, \kappa_2, \kappa_3)}$		radial distortion function
$\downarrow$		
Distorted 2D Image Point	2-vector	$\tilde{\mathbf{x}}_d = (\tilde{x}_d, \tilde{y}_d)$

The following function maps the parameter vector to the estimate of the measurement vector. It also shows the partitioning of the parameter vector into three partitions:

$$\begin{aligned}\hat{\mathbf{X}} &= \mathbf{f}(\mathbf{P}) \\ &= \mathbf{f}(\mathbf{a}, \mathbf{b}, \mathbf{c}) \\ &= \mathbf{f}(\mathbf{h}, \mathbf{x}, \mathbf{x}', \mathbf{c})\end{aligned} \tag{C.6}$$

C.2.1 Partition A

The Jacobians in Partition A measure the change in the measurement vector with respect to the homographies for each camera view. Where a feature is not visible in the camera view, the

Jacobian is the zero matrix.

$$\begin{aligned}\frac{\partial \hat{\mathbf{x}}}{\partial \mathbf{a}} &= \frac{\partial \tilde{\mathbf{x}}_d}{\partial \mathbf{a}} \\ &= \frac{\partial \tilde{\mathbf{x}}_d}{\partial \tilde{\mathbf{x}}}\frac{\partial \tilde{\mathbf{x}}}{\partial \mathbf{a}} + \frac{\partial \tilde{\mathbf{x}}_d}{\partial \mathbf{c}}\frac{\partial \mathbf{c}}{\partial \mathbf{a}}\end{aligned}$$

But as $\mathbf{x}$ is independent of $\mathbf{a}$ this gives $\frac{\partial \mathbf{c}}{\partial \mathbf{a}} = 0$, resulting in the simplified expression:

$$\begin{aligned}&= \frac{\partial \tilde{\mathbf{x}}_d}{\partial \tilde{\mathbf{x}}}\frac{\partial \tilde{\mathbf{x}}}{\partial \mathbf{a}} \\ &= \frac{\partial \tilde{\mathbf{x}}_d}{\partial \tilde{\mathbf{x}}}\frac{\partial \tilde{\mathbf{x}}}{\partial \mathbf{x}'}\frac{\partial \mathbf{x}'}{\partial \mathbf{a}} \\ &= \frac{\partial \tilde{\mathbf{x}}_d}{\partial \tilde{\mathbf{x}}}\frac{\partial \tilde{\mathbf{x}}}{\partial \mathbf{x}'}\left(\frac{\partial \mathbf{x}'}{\partial \mathbf{x}}\frac{\partial \mathbf{x}}{\partial \mathbf{a}} + \frac{\partial \mathbf{x}'}{\partial \mathbf{h}}\frac{\partial \mathbf{h}}{\partial \mathbf{a}}\right)\end{aligned}$$

But as $\mathbf{x}$ is independent of $\mathbf{a}$ this gives $\frac{\partial \mathbf{x}}{\partial \mathbf{a}} = 0$ and $\mathbf{a} = \mathbf{h}$ so $\frac{\partial \mathbf{h}}{\partial \mathbf{a}} = \mathtt{I}$. The expression above then simplifies to:

$$\frac{\partial \hat{\mathbf{x}}}{\partial \mathbf{a}} = \frac{\partial \tilde{\mathbf{x}}_d}{\partial \tilde{\mathbf{x}}}\frac{\partial \tilde{\mathbf{x}}}{\partial \mathbf{x}'}\frac{\partial \mathbf{x}'}{\partial \mathbf{h}} \tag{C.7}$$

$$= \begin{bmatrix} 2(\tilde{x}-x_c)^2(\kappa_1+2\kappa_2 r^2+4\kappa_3 r^4)+L(r) & 2(\tilde{x}-x_c)(\tilde{y}-y_c)(\kappa_1+2\kappa_2 r^2+4\kappa_3 r^4) \\ 2(\tilde{x}-x_c)(\tilde{y}-y_c)(\kappa_1+2\kappa_2 r^2+4\kappa_3 r^4) & 2(\tilde{y}-y_c)^2(\kappa_1+2\kappa_2 r^2+4\kappa_3 r^4)+L(r) \end{bmatrix}$$
$$\begin{bmatrix} \frac{1}{w'} & 0 & -\frac{x'}{w'^2} \\ 0 & \frac{1}{w'} & -\frac{y'}{w'^2} \end{bmatrix} \begin{bmatrix} x & y & w & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & x & y & w & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & x & y & w \end{bmatrix}$$

$$\begin{aligned}\text{where} \quad & L(r) = (1+\kappa_1 r^2+\kappa_2 r^4+\kappa_3 r^6) \\ & r = \sqrt{(\tilde{x}-x_c)^2-(\tilde{y}-y_c)^2}\end{aligned}$$

C.2.2 Partition B

Partition B contains all of the global feature points, stored as 2D homogenous points. The positions of the global feature points will also be optimised, however, the optimised values are never used to update the global feature points as errors in the optimisation could be compounded in future steps.

$$\begin{aligned}\frac{\partial \hat{\mathbf{x}}}{\partial \mathbf{b}} &= \frac{\partial \tilde{\mathbf{x}}_d}{\partial \mathbf{b}} \\ &= \frac{\partial \tilde{\mathbf{x}}_d}{\partial \tilde{\mathbf{x}}}\frac{\partial \tilde{\mathbf{x}}}{\partial \mathbf{b}} + \frac{\partial \tilde{\mathbf{x}}_d}{\partial \mathbf{c}}\frac{\partial \mathbf{c}}{\partial \mathbf{b}}\end{aligned}$$

But as $\mathbf{c}$ is independent of $\mathbf{b}$ this gives $\frac{\partial \mathbf{c}}{\partial \mathbf{b}} = 0$, resulting in the simplified expression:

$$\begin{aligned}&= \frac{\partial \tilde{\mathbf{x}}_d}{\partial \tilde{\mathbf{x}}}\frac{\partial \tilde{\mathbf{x}}}{\partial \mathbf{b}} \\ &= \frac{\partial \tilde{\mathbf{x}}_d}{\partial \tilde{\mathbf{x}}}\frac{\partial \tilde{\mathbf{x}}}{\partial \mathbf{x}'}\frac{\partial \mathbf{x}'}{\partial \mathbf{b}} \\ &= \frac{\partial \tilde{\mathbf{x}}_d}{\partial \tilde{\mathbf{x}}}\frac{\partial \tilde{\mathbf{x}}}{\partial \mathbf{x}'}\left(\frac{\partial \mathbf{x}'}{\partial \mathbf{x}}\frac{\partial \mathbf{x}}{\partial \mathbf{b}} + \frac{\partial \mathbf{x}'}{\partial \mathbf{h}}\frac{\partial \mathbf{h}}{\partial \mathbf{b}}\right)\end{aligned}$$

But as $\mathbf{h}$ is independent of $\mathbf{b}$ this gives $\frac{\partial \mathbf{h}}{\partial \mathbf{b}} = 0$ and $\mathbf{x} = \mathbf{b}$ so $\frac{\partial \mathbf{x}}{\partial \mathbf{b}} = \mathtt{I}$. The expression above then simplifies to:

$$\frac{\partial \hat{\mathbf{x}}}{\partial \mathbf{b}} = \frac{\partial \tilde{\mathbf{x}}_d}{\partial \tilde{\mathbf{x}}}\frac{\partial \tilde{\mathbf{x}}}{\partial \mathbf{x}'}\frac{\partial \mathbf{x}'}{\partial \mathbf{x}} \tag{C.8}$$

$$= \begin{bmatrix} 2(\tilde{x}-x_c)^2(\kappa_1+2\kappa_2 r^2+4\kappa_3 r^4)+L(r) & 2(\tilde{x}-x_c)(\tilde{y}-y_c)(\kappa_1+2\kappa_2 r^2+4\kappa_3 r^4) \\ 2(\tilde{x}-x_c)(\tilde{y}-y_c)(\kappa_1+2\kappa_2 r^2+4\kappa_3 r^4) & 2(\tilde{y}-y_c)^2(\kappa_1+2\kappa_2 r^2+4\kappa_3 r^4)+L(r) \end{bmatrix}$$

$$\begin{bmatrix} \frac{1}{w'} & 0 & -\frac{x'}{w'^2} \\ 0 & \frac{1}{w'} & -\frac{y'}{w'^2} \end{bmatrix} \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix}$$

$$\begin{aligned}\text{where} \quad & L(r) = (1+\kappa_1 r^2+\kappa_2 r^4+\kappa_3 r^6) \\ & r = \sqrt{(\tilde{x}-x_c)^2-(\tilde{y}-y_c)^2}\end{aligned}$$

C.2.3 Partition C

The radial distortion parameters, with the principal point fixed at the centre of the image, are optimised in Partition C. As the radial distortion parameter effect the homographies and the global feature positions, the resulting Jacobians will be dense i.e. the Jacobian will be non-zero for all of the other parameters.

$$\frac{\partial \hat{\mathbf{x}}}{\partial \mathbf{c}} = \frac{\partial \tilde{\mathbf{x}}_d}{\partial \mathbf{c}} \tag{C.9}$$

$$= \begin{bmatrix} r^2(\tilde{x} - x_c) & r^4(\tilde{x} - x_c) & r^6(\tilde{x} - x_c) \\ r^2(\tilde{y} - y_c) & r^4(\tilde{y} - y_c) & r^6(\tilde{y} - y_c) \end{bmatrix}$$

where $r = \sqrt{(\tilde{x} - x_c)^2 - (\tilde{y} - y_c)^2}$

C.3 Jacobians – Bundle II

The transfer of a point $\mathbf{x}$ between two views where the camera has only rotated are given by $\mathbf{x}' = \mathbf{K}'\mathbf{R}'\mathbf{R}^{-1}\mathbf{K}^{-1}\mathbf{x}$. As the intrinsic parameters have remained constant and the reference view rotation matrix $\mathbf{R}^{-1}$ is the identity matrix, the above expression reduces to:

$$\mathbf{x}' = \mathbf{K}\mathbf{R}\mathbf{K}^{-1}\mathbf{x} \tag{C.10}$$

where $\mathbf{K}$ Intrinsic parameter matrix

$\mathbf{R}$ Relative rotation matrix between the view and the reference view

This is equivalent to the transfer of points between views in the previous section and $\mathbf{K}\mathbf{R}\mathbf{K}^{-1}$ is equivalent to a homography. The partitions are parameterised as follows:

$\mathbf{a} = (\theta_1, \phi_1, \theta_2, \phi_2, \ldots, \theta_m, \phi_m)^\mathsf{T}$	Rotations for each view
$\mathbf{b} = (\kappa_1, \kappa_2, \kappa_3, x_c, y_c, \alpha, f_y)^\mathsf{T}$	Five radial distortion and two intrinsic parameters
$\mathbf{c} = (x_1, y_1, w_1, x_2, y_2, w_2, \ldots, x_n, y_n, w_n)^\mathsf{T}$	Global feature points

The transfer of points from the reference view into each camera view is shown in the following block diagram:

2D Global Feature Point	3-vector	$\mathbf{x} = (x, y, w)$
↓		
$\boxed{\mathbf{x}' = \mathbf{K}\mathbf{R}\mathbf{K}^{-1}\mathbf{x}}$		homography
↓		
2D Image Point	3-vector	$\mathbf{x}' = (x', y', w')$
↓		
$\boxed{(x', y', w') \mapsto (\tilde{x}, \tilde{y})}$		projection
↓		
Inhomogeneous 2D Image Point	2-vector	$\tilde{\mathbf{x}} = (\tilde{x}, \tilde{y})$
↓		
$\boxed{\tilde{\mathbf{x}}_d = f(\tilde{\mathbf{x}}, \mathbf{c}) = f(\tilde{\mathbf{x}}, \kappa_1, \kappa_2, \kappa_3, x_c, y_c)}$		radial distortion function
↓		
Distorted 2D Image Point	2-vector	$\tilde{\mathbf{x}}_d = (\tilde{x}_d, \tilde{y}_d)$

C.3.1 Partition A

$$\frac{\partial \hat{\mathbf{x}}}{\partial \mathbf{a}} = \frac{\partial \tilde{\mathbf{x}}_d}{\partial \tilde{\mathbf{x}}} \frac{\partial \tilde{\mathbf{x}}}{\partial \mathbf{x}'} \frac{\partial \mathbf{x}'}{\partial \mathbf{a}} \tag{C.11}$$

$$= \begin{bmatrix} 2(\tilde{x}-x_c)^2(\kappa_1+2\kappa_2 r^2+4\kappa_3 r^4)+L(r) & 2(\tilde{x}-x_c)(\tilde{y}-y_c)(\kappa_1+2\kappa_2 r^2+4\kappa_3 r^4) \\ 2(\tilde{x}-x_c)(\tilde{y}-y_c)(\kappa_1+2\kappa_2 r^2+4\kappa_3 r^4) & 2(\tilde{y}-y_c)^2(\kappa_1+2\kappa_2 r^2+4\kappa_3 r^4)+L(r) \end{bmatrix}$$
$$\begin{bmatrix} \frac{1}{w'} & 0 & -\frac{x'}{w'^2} \\ 0 & \frac{1}{w'} & -\frac{y'}{w'^2} \end{bmatrix} \begin{bmatrix} d_{11} & d_{12} \\ d_{21} & d_{22} \\ d_{31} & d_{32} \end{bmatrix}$$

where $L(r) = (1+\kappa_1 r^2+\kappa_2 r^4+\kappa_3 r^6)$

$r = \sqrt{(\tilde{x}-x_c)^2-(\tilde{y}-y_c)^2}$

$\mathbf{a} = (\theta, \phi)^\mathsf{T}$

θ is the pan angle

ϕ is the tilt angle

$$d_{11} = x\frac{-\alpha f_y \sin\theta - x_c\cos\phi\cos\theta}{f_y\alpha} - w\Big[\frac{-\alpha f_y\sin\theta - x_c\cos\phi\cos\theta}{f_y\alpha}x_c - \alpha f_y\cos\theta + x_c\cos\phi\sin\theta\Big]$$
$$d_{12} = x\frac{x_c\sin\phi\sin\theta}{f_y\alpha} + y\frac{x_c\cos\phi}{f_y} - wx_c\Big[\frac{x_c\sin\phi\sin\theta}{f_y\alpha} + \frac{\cos\phi}{f_y y_c} + \sin\phi\cos\theta\Big]$$

$$d_{21} = x\frac{f_y\sin\phi\cos\theta - y_c\cos\phi\cos\theta}{f_y\alpha} + w\Big[-x_c\frac{f_y\sin\phi\cos\theta - y_c\cos\phi\cos\theta}{f_y\alpha} + f_y\sin\phi\sin\theta - y_c\cos\phi\sin\theta\Big]$$
$$d_{22} = x\frac{f_y\cos\phi\sin\theta + y_c\sin\phi\sin\theta}{f_y\alpha} + y\frac{-f_y\sin\phi + y_c\cos\phi}{f_y}$$
$$- w\Big[\frac{f_y\cos\phi\sin\theta + y_c\sin\phi\sin\theta}{f_y\alpha}x_c + \frac{(-f_y\sin\phi + y_c\cos\phi)}{f_y}y_c + f_y\cos\phi\cos\theta + y_c\sin\phi\cos\theta\Big]$$

$$d_{31} = x\frac{-\cos\phi\cos\theta}{f_y\alpha} + w\Big[\frac{\cos\phi\cos\theta x_c}{f_y\alpha} - \cos\phi\sin\theta\Big]$$
$$d_{32} = \frac{\sin\phi\sin\theta}{f_y\alpha} + y\frac{\cos\phi}{f_y} - w\Big[\frac{\sin\phi\sin\theta x_c}{f_y\alpha} + y_c\frac{\cos\phi}{f_y} + \sin\phi\cos\theta\Big]$$

C.3.2 Partition B

Partition B is create from the radial distortion parameters $(\kappa_1, \kappa_2, \kappa_3, x_c, y_c)^\mathsf{T}$ and the intrinsic parameters $(\alpha, f_y)^\mathsf{T}$. All of these parameters are constant over all camera views.

$$\frac{\partial \hat{\mathbf{x}}}{\partial \mathbf{b}} = \frac{\partial \tilde{\mathbf{x}}_d}{\partial \tilde{\mathbf{x}}} \frac{\partial \tilde{\mathbf{x}}}{\partial \mathbf{x}'} \frac{\partial \mathbf{x}'}{\partial \mathbf{x}} \tag{C.12}$$

$$= \begin{bmatrix} 2(\tilde{x}-x_c)^2(\kappa_1+2\kappa_2 r^2+4\kappa_3 r^4)+L(r) & 2(\tilde{x}-x_c)(\tilde{y}-y_c)(\kappa_1+2\kappa_2 r^2+4\kappa_3 r^4) \\ 2(\tilde{x}-x_c)(\tilde{y}-y_c)(\kappa_1+2\kappa_2 r^2+4\kappa_3 r^4) & 2(\tilde{y}-y_c)^2(\kappa_1+2\kappa_2 r^2+4\kappa_3 r^4)+L(r) \end{bmatrix}$$

$$\begin{bmatrix} \frac{1}{w'} & 0 & -\frac{x'}{w'^2} \\ 0 & \frac{1}{w'} & -\frac{y'}{w'^2} \end{bmatrix} \begin{bmatrix} d_{11} & d_{12} \\ d_{21} & d_{22} \\ d_{31} & d_{32} \end{bmatrix}$$

$$\text{where} \quad L(r) = (1+\kappa_1 r^2+\kappa_2 r^4+\kappa_3 r^6)$$
$$r = \sqrt{(\tilde{x}-x_c)^2-(\tilde{y}-y_c)^2}$$

$$\begin{aligned} d_{11} =& x\frac{\cos\theta}{\alpha} - x\frac{\alpha * f_y * \cos\theta - x_c * \cos\phi * \sin\theta}{f_y\alpha^2} \\ &+ w\Big[-\frac{\cos\theta * x_c}{\alpha} + x_c\frac{\alpha f_y\cos\theta - x_c\cos\phi\sin\theta}{f_y\alpha^2} + f_y\sin\theta\Big] \\ d_{12} =& x\frac{\cos\theta}{f_y} - x\frac{\alpha * f_y * \cos\theta - x_c * \cos\phi * \sin\theta}{f_y^2\alpha} - y\frac{x_c\sin\phi}{f_y^2} \\ &+ w\Big[-\frac{\cos\theta x_c}{f_y} + x_c\frac{\alpha f_y\cos\theta - x_c\cos\phi\sin\theta}{f_y^2\alpha} + x_c y_c\frac{\sin\phi}{f_y^2} + \alpha\sin\theta\Big] \end{aligned}$$

$$\begin{aligned} d_{21} =& -x\frac{f_y\sin\phi\sin\theta - y_c\cos\phi\sin\theta}{f_y\alpha^2} + wx_c\frac{f_y\sin\phi\sin\theta - y_c\cos\phi\sin\theta}{f_y\alpha^2} \\ d_{22} =& x\frac{\sin\phi\sin\theta}{f_y\alpha} - x\frac{f_y\sin\phi\sin\theta - y_c\cos\phi\sin\theta}{f_y^2\alpha} + y\frac{\cos\phi}{f_y} - y\frac{f_y\cos\phi + y_c\sin\phi}{f_y^2} \\ &+ w\Big[\frac{-\sin\phi\sin\theta x_c}{f_y\alpha} + x_c\frac{f_y\sin\phi\sin\theta - y_c\cos\phi\sin\theta}{f_y^2\alpha} - y_c\frac{\cos\phi}{f_y} + y_c\frac{f_y\cos\phi + y_c\sin\phi}{f_y^2} - \sin\phi\cos\theta\Big] \end{aligned}$$

$$\begin{aligned} d_{31} =& x\frac{\cos\phi\sin\theta}{f_y\alpha^2} - w\Big[\frac{\cos\phi\sin\theta x_c}{f_y\alpha^2}\Big] \\ d_{32} =& x\frac{\cos\phi\sin\theta}{f_y^2\alpha} - \frac{\sin\phi}{f_y^2} + w\Big[\frac{-\cos\phi\sin\theta x_c}{f_y^2\alpha} + y_c\frac{\sin\phi}{f_y^2}\Big] \end{aligned}$$

C.3.3 Partition C

Partition C contains all of the global feature points, stored as 2D homogenous points. As before, the global features can be related to each view by a homography. In this case the homography

can be found by evaluating $\mathtt{H} = \mathbf{KRK}^{-1}$.

$$\frac{\partial \hat{\mathbf{x}}}{\partial \mathbf{c}} = \frac{\partial \tilde{\mathbf{x}}_d}{\partial \tilde{\mathbf{x}}} \frac{\partial \tilde{\mathbf{x}}}{\partial \mathbf{x}'} \frac{\partial \mathbf{x}'}{\partial \mathbf{x}} \tag{C.13}$$

$$= \begin{bmatrix} 2(\tilde{x}-x_c)^2(\kappa_1+2\kappa_2 r^2+4\kappa_3 r^4)+L(r) & 2(\tilde{x}-x_c)(\tilde{y}-y_c)(\kappa_1+2\kappa_2 r^2+4\kappa_3 r^4) \\ 2(\tilde{x}-x_c)(\tilde{y}-y_c)(\kappa_1+2\kappa_2 r^2+4\kappa_3 r^4) & 2(\tilde{y}-y_c)^2(\kappa_1+2\kappa_2 r^2+4\kappa_3 r^4)+L(r) \end{bmatrix}$$

$$\begin{bmatrix} \frac{1}{w'} & 0 & -\frac{x'}{w'^2} \\ 0 & \frac{1}{w'} & -\frac{y'}{w'^2} \end{bmatrix} \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix}$$

$$\text{where} \quad L(r) = (1+\kappa_1 r^2+\kappa_2 r^4+\kappa_3 r^6)$$
$$r = \sqrt{(\tilde{x}-x_c)^2-(\tilde{y}-y_c)^2}$$

C.4 Jacobians – Zoom Final Bundle

A point is transferred from one image to another after zooming by: $\mathbf{x}' = \mathbf{K}'\mathbf{K}^{-1}\mathbf{x}$. However, if the global feature points are used then this expression becomes: $\mathbf{x}' = \mathbf{K}'\mathbf{x}_{\text{global}}$. The transformation of a global feature point into an image point in a zooming view is shown in the following block diagram:

2D Global Feature Point	3-vector	$\mathbf{x} = (x, y, w)$
$\downarrow$		
$\boxed{\mathbf{x}' = \mathbf{K}'\mathbf{x}}$		homography
$\downarrow$		
2D Image Point	3-vector	$\mathbf{x}' = (x', y', w')$
$\downarrow$		
$\boxed{(x', y', w') \mapsto (\tilde{x}, \tilde{y})}$		projection
$\downarrow$		
Inhomogeneous 2D Image Point	2-vector	$\tilde{\mathbf{x}} = (\tilde{x}, \tilde{y})$
$\downarrow$		
$\boxed{\tilde{\mathbf{x}}_d = f(\tilde{\mathbf{x}}, \mathbf{c}) = f(\tilde{\mathbf{x}}, \kappa_1, \kappa_2, \kappa_3, x_c, y_c)}$		radial distortion function
$\downarrow$		
Distorted 2D Image Point	2-vector	$\tilde{\mathbf{x}}_d = (\tilde{x}_d, \tilde{y}_d)$

The parameter vector is partitioned into two as follows:

$$\mathbf{a} = (f_{y1}, x_{c1}, y_{c1}, \kappa_{11}, \kappa_{21}, \ldots \quad f_{y2}, x_{c2}, y_{c2}, \kappa_{12}, \ldots, f_{ym}, x_{cm}, y_{cm})^{\mathsf{T}}$$ Intrinsic parameters for each view

$$\mathbf{b} = (x_1, y_1, w_1, x_2, y_2, w_2, \ldots, x_n, y_n, w_n)^{\mathsf{T}}$$ Global feature points

C.4.1 Partition A

The parameters in Partition A consist of the intrinsic parameters for the zoomed view concatenated with the radial distortion parameters. Although three radial distortion parameters are shown, it is possible to optimise anywhere between zero and three parameters by removing the associated columns of the matrices below and setting the appropriate value of κ_n to zero.

$$\begin{aligned}\frac{\partial \hat{\mathbf{x}}}{\partial \mathbf{a}} &= \frac{\partial \tilde{\mathbf{x}}_d}{\partial \tilde{\mathbf{x}}}\frac{\partial \tilde{\mathbf{x}}}{\partial \mathbf{k_1}} + \frac{\partial \tilde{\mathbf{x}}_d}{\partial \mathbf{k_2}}\frac{\partial \mathbf{k_2}}{\partial \mathbf{a}} \\ &= \frac{\partial \tilde{\mathbf{x}}_d}{\partial \tilde{\mathbf{x}}}\frac{\partial \tilde{\mathbf{x}}}{\partial \mathbf{x}'}\frac{\partial \mathbf{x}'}{\partial \mathbf{k_1}} + \frac{\partial \tilde{\mathbf{x}}_d}{\partial \mathbf{k_2}}\frac{\partial \mathbf{k_2}}{\partial \mathbf{a}}\end{aligned}$$

where $\mathbf{k_1} = (f_y, x_c, y_c, 0, 0)^\mathsf{T}$ Intrinsic parameters

$\mathbf{k_2} = (0, 0, 0, \kappa_1, \kappa_2, \kappa_3)^\mathsf{T}$ Radial distortion coefficients

$$= \begin{bmatrix} 2(\tilde{x}-x_c)^2(\kappa_1 + 2\kappa_2 r^2 + 4\kappa_3 r^4) + L(r) & 2(\tilde{x}-x_c)(\tilde{y}-y_c)(\kappa_1 + 2\kappa_2 r^2 + 4\kappa_3 r^4) \\ 2(\tilde{x}-x_c)(\tilde{y}-y_c)(\kappa_1 + 2\kappa_2 r^2 + 4\kappa_3 r^4) & 2(\tilde{y}-y_c)^2(\kappa_1 + 2\kappa_2 r^2 + 4\kappa_3 r^4) + L(r) \end{bmatrix}$$

$$\begin{bmatrix} \frac{1}{w'} & 0 & -\frac{x'}{w'^2} \\ 0 & \frac{1}{w'} & -\frac{y'}{w'^2} \end{bmatrix} \begin{bmatrix} \alpha x & w & 0 & 0 & 0 \\ y & 0 & w & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$+ \begin{bmatrix} 0 & -2(\tilde{x}-x_c)^2 L_2(r) - L(r) & -2(\tilde{x}-x_c)(\tilde{y}-y_c)L_2(r) - L(r) \\ 0 & -2(\tilde{x}-x_c)(\tilde{y}-y_c)L_2(r) - L(r) & -2(\tilde{y}-y_c)^2 L_2(r) - L(r) \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & \mathtt{I}_{5\times 5} \end{bmatrix}$$

where $L(r) = (1 + \kappa_1 r^2 + \kappa_2 r^4 + \kappa_3 r^6)$

$L_2(r) = (\kappa_1 + 2\kappa_2 r^2 + 4\kappa_3 r^4)$

$r = \sqrt{(\tilde{x}-x_c)^2 - (\tilde{y}-y_c)^2}$

C.4.2 Partition B

As the homography between the two views is essentially the intrinsic parameters of the zoomed view, the Jacobians for optimising the global feature points in Partition B are similar to previous sections:

$$\frac{\partial \hat{\mathbf{x}}}{\partial \mathbf{b}} = \frac{\partial \tilde{\mathbf{x}}_d}{\partial \tilde{\mathbf{x}}}\frac{\partial \tilde{\mathbf{x}}}{\partial \mathbf{x}'}\frac{\partial \mathbf{x}'}{\partial \mathbf{x}} \tag{C.14}$$

$$= \begin{bmatrix} 2(\tilde{x}-x_c)^2(\kappa_1+2\kappa_2 r^2+4\kappa_3 r^4)+L(r) & 2(\tilde{x}-x_c)(\tilde{y}-y_c)(\kappa_1+2\kappa_2 r^2+4\kappa_3 r^4) \\ 2(\tilde{x}-x_c)(\tilde{y}-y_c)(\kappa_1+2\kappa_2 r^2+4\kappa_3 r^4) & 2(\tilde{y}-y_c)^2(\kappa_1+2\kappa_2 r^2+4\kappa_3 r^4)+L(r) \end{bmatrix}$$

$$\begin{bmatrix} \frac{1}{w'} & 0 & -\frac{x'}{w'^2} \\ 0 & \frac{1}{w'} & -\frac{y'}{w'^2} \end{bmatrix} \begin{bmatrix} \alpha f_y & 0 & x_c \\ 0 & f_y & y_c \\ 0 & 0 & 1 \end{bmatrix}$$

$$\text{where} \quad L(r) = (1+\kappa_1 r^2+\kappa_2 r^4+\kappa_3 r^6)$$
$$r = \sqrt{(\tilde{x}-x_c)^2-(\tilde{y}-y_c)^2}$$

C.5 Matrix normalisation

Hartley's linear method[30] for finding the rotation and intrinsic parameters between views when the camera only rotates, requires that the homographies between pairs of views are 'normalised' such that their determinants are equal to one. To normalise an $N \times N$ matrix $\mathbf{A}$ such that $\det \mathbf{A}' = 1$, find the normalising factor α so that:

$$\mathbf{A}' = \alpha \mathbf{A} \tag{C.15}$$

If $\mathbf{A}$ is positive definite then:

$$\mathbf{B} = \alpha \mathbf{A}$$
$$\det \mathbf{B} = \det(\alpha \mathbf{A})$$
$$= \alpha^N \det \mathbf{A}$$
$$= 1$$

$$\alpha = \frac{1}{\sqrt[N]{\det \mathbf{A}}} \tag{C.16}$$

If $\mathbf{A}$ is not positive definite then α and $\mathbf{B}$ will be complex.

Appendix D
Software

This appendix briefly outlines some of the software packages used in this thesis as well as the software libraries and applications developed to produce the tracking methods discussed in the main body of this thesis. A brief discussion of the control of the Sony EVI-D31 PTZ camera via Sony's VISCA protocol is included in Section D.2.

D.1 Povray

Povray is a freeware ray-tracer that produces high quality ray-traced images from a scripting language. Image sequences and animations are also easily generated. Povray was chosen as the rendering engine for the synthetic sequences as it is freeware allowing it to be rendered on several machines simultaneously to speed up the rendering process. Povray can be downloaded from `http://www.povray.org` along with detailed documentation. The scripting langauge can do matrix multiplications and point transforms and output the results as text to produce ground truth and other information about camera positions and zooms. A large Povray community exists along with example scripts and 3D models of common objects. The Povray camera discussed in Section 3.6.4 can be setup to emulate the model of the Sony EVI-D31 PTZ camera. The resolution of the rendered image and the dynamics of the camera could be modelled by generating a rendering script from an application written in C/C++.

For all of the rendering tasks in this thesis, Povray was run simultaneously to render each frame of the sequence on several Linux PC's. A single master remotely executed Povray and collated the frames after creating the ground truth for all the frames. A single script file was used. This cut down rendering times from several hours to a few minutes allowing a rapid development turn-around. Other graphics systems such as OpenGL could have been used to generate the synthetic sequences, however, it would have been necessary to write a large number of data handling classes, such as a class to read in the vertices of a 3D model. Povray has a well defined scripting language which is far more flexible than a compiled program.

D.2 Sony EVI-D31 Serial Driver

One of the most annoying aspects of the software developed in this thesis was creating a driver for the Sony EVI-D31 PTZ camera. As the VISCA protocol is proprietary to Sony, there is very little documentation freely available. The user manual is extremely vague and brief. Although there are some drivers available on the Internet, they are mainly directed towards applications which use the Sony EVI-D31 as a webcam which can be controlled by remote users. As the response time in this type of application is not critical most of the available software does not try to extract the maximum data rate from the camera serial link.

The pan, tilt and zoom parameters need to be read from the camera as quickly as possible so that the parameters reported do not lag the frames captured. However, in practice it is impossible to read the camera parameters at speeds comparable to the frame rate. This is largely due to the transmission speeds of 9600bps and the protocol overhead.

D.2.1 VISCA Protocol

VISCA commands are divided into two categories: commands to adjust camera parameters (for example the pan/tilt position and the white balance) and information requests (for example a query to return the pan/tilt position). The various types of VISCA packets are summarised in Figure D.1.

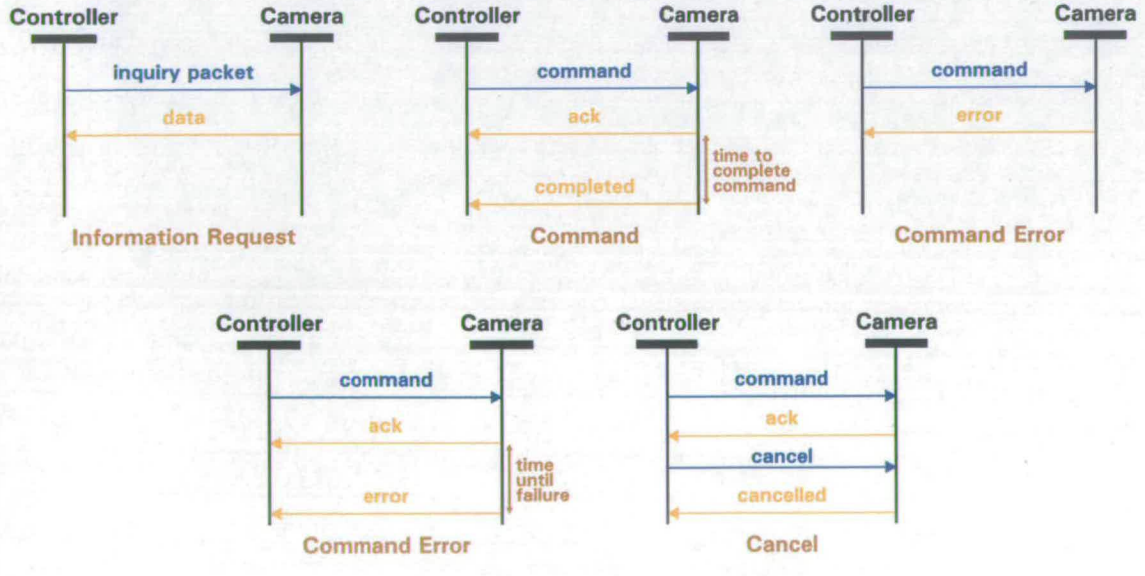


Figure D.1: *Types of VISCA packets and command sequences*

Figure D.2 shows an example timing diagram for specifying the pan, tilt and zoom position while reading the pan/tilt information back from the camera. Certain commands must not be interrupted and there can be no data transfer until either an ACK packet or an INFO packet have been returned.

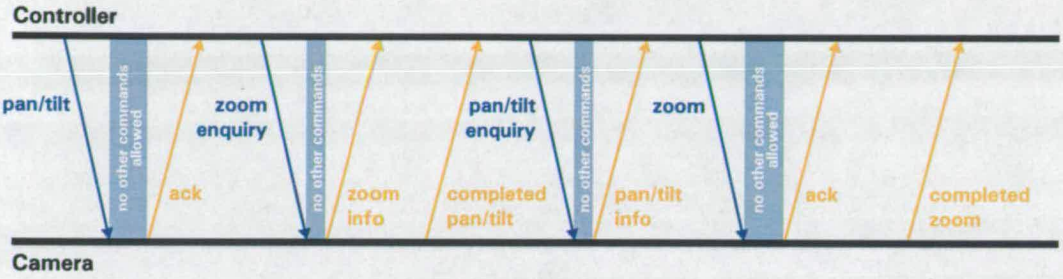


Figure D.2: *Example VISCA timing diagram*

D.2.2 Implementation

The order in which the packets can be returned is dependant on the time the command takes to complete. After issuing a command to pan and tilt the camera it is possible to issue other commands once the ACK packet has been received. In order to achieve the maximum performance it is necessary to have a threaded driver with blocking calls for operations such as pan/tilt commands as shown in Figure D.3.

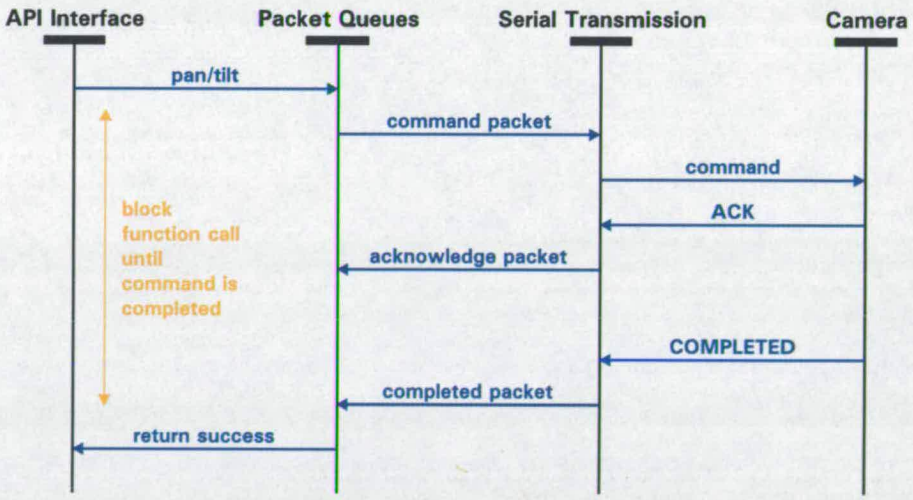


Figure D.3: *VISCA driver layer diagram*

Figure D.4 shows the necessity for implementing threaded receive and transmit queues. The arrival of packets is fairly asynchronous. A synchronous system could also be implemented, however nothing could be transmitted while waiting for the completed packet. Many of the ex-

isting VISCA drivers for the Sony EVI-D31 are implemented as synchronous functions which write data to the serial port and read packets back until the desired response is return. This strategy can be dangerous unless of the packet types are implemented and detected as packets can arrive in any order. In a queuing system, it is possible setup events which are triggered once a packet of a certain type of received and the waiting threads can then continue to execute. This method also frees up the processor to perform other tasks as it does not need to sit in a wait loop.

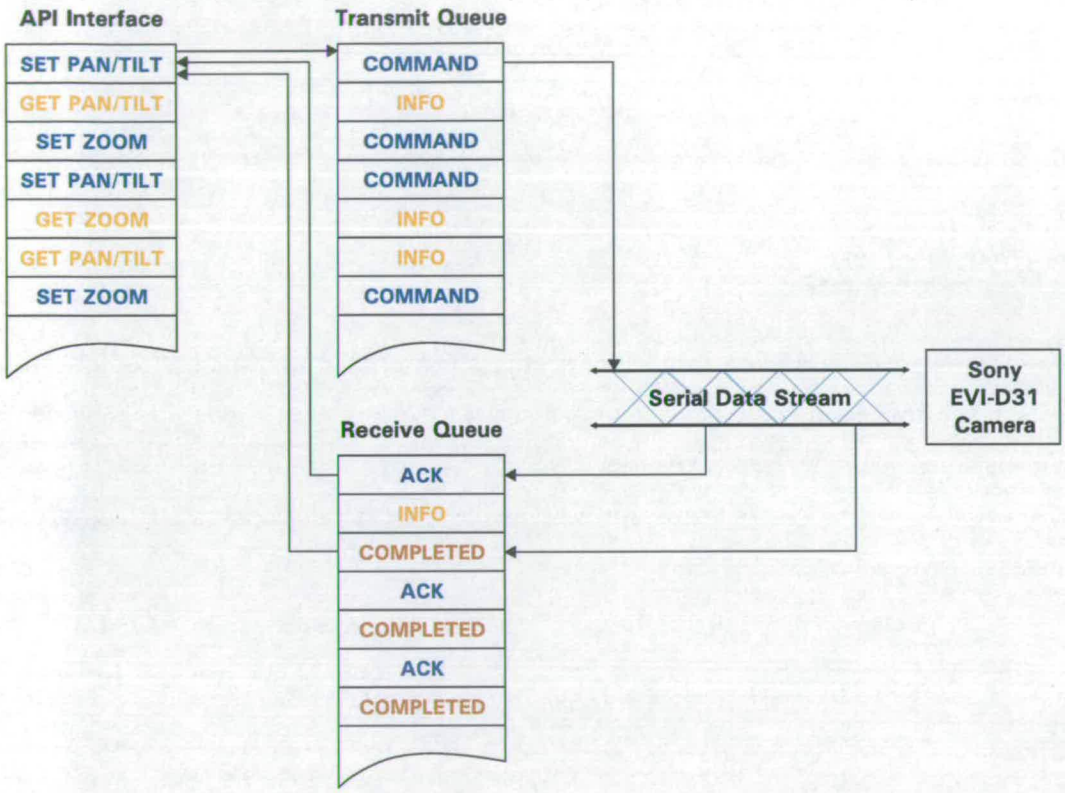


Figure D.4: *VISCA driver transmit and receive queues*

D.3 Vision C/C++ Library

A library of computer vision methods and mathematical operators was created such that algorithms could be written to run in real-time. An existing library at the University of Edinburgh developed by Andrew Peacock, Peter Hillman and Chris Haworth was used as the foundation for many of the classes, however they lacked the ability to run in real-time and needed to be optimised. As the PXC200A frame grabber had convenient Windows drivers, it was decided to use a Windows platform to develop all software on. The differences between Unix and

Windows are slight when user interface components are not used and it would be possible to port most of the library to Unix fairly easily. This section briefly describes all of the software developed providing example screenshots and simple class organisation diagrams.

D.3.1 Organisation

All applications were developed using Microsoft Visual C++ 7.0 on Win32 platform, specifically Windows XP. The standard Microsoft compiler was used. The following additional libraries were used: OpenCV, VTK and VXL. All other software was developed from the internal Vision Library developed originally by Andrew Peacock, Chris Haworth and Peter Hillman. The goal of the software produced was to allow the real-time capture and tracking of targets using the Sony EVI-D31 PTZ camera. All of the software developed could be easily ported to a Unix platform with the exception of the camera capture drivers and the serial communications classes of the VISCA protocol. Capture classes were also developed to allow the real-time capture of frames from the framegrabber to be used in real-time image processing methods.

Figure D.5 is an overview of the C++ classes developed. From the names and structure of the classes it is obvious what the functionality of each class is. Anyone wishing to use these classes can arrange to do so through the author or the School of Engineering and Electronics.

D.4 Applications

Several applications were developed in order to capture sequences and produce ground truth. Tracking applications and visualisation applications were also developed. The following sections show screenshots of two support applications referred to in the main text of the thesis.

D.4.1 Ground Truth

Figure D.6 shows the application developed to produce hand-tracked ground truth for all of the test sequences. An enlarged region of the image around the match position allows easy matching of the target in each frame. The ground truth is saved as a text file.

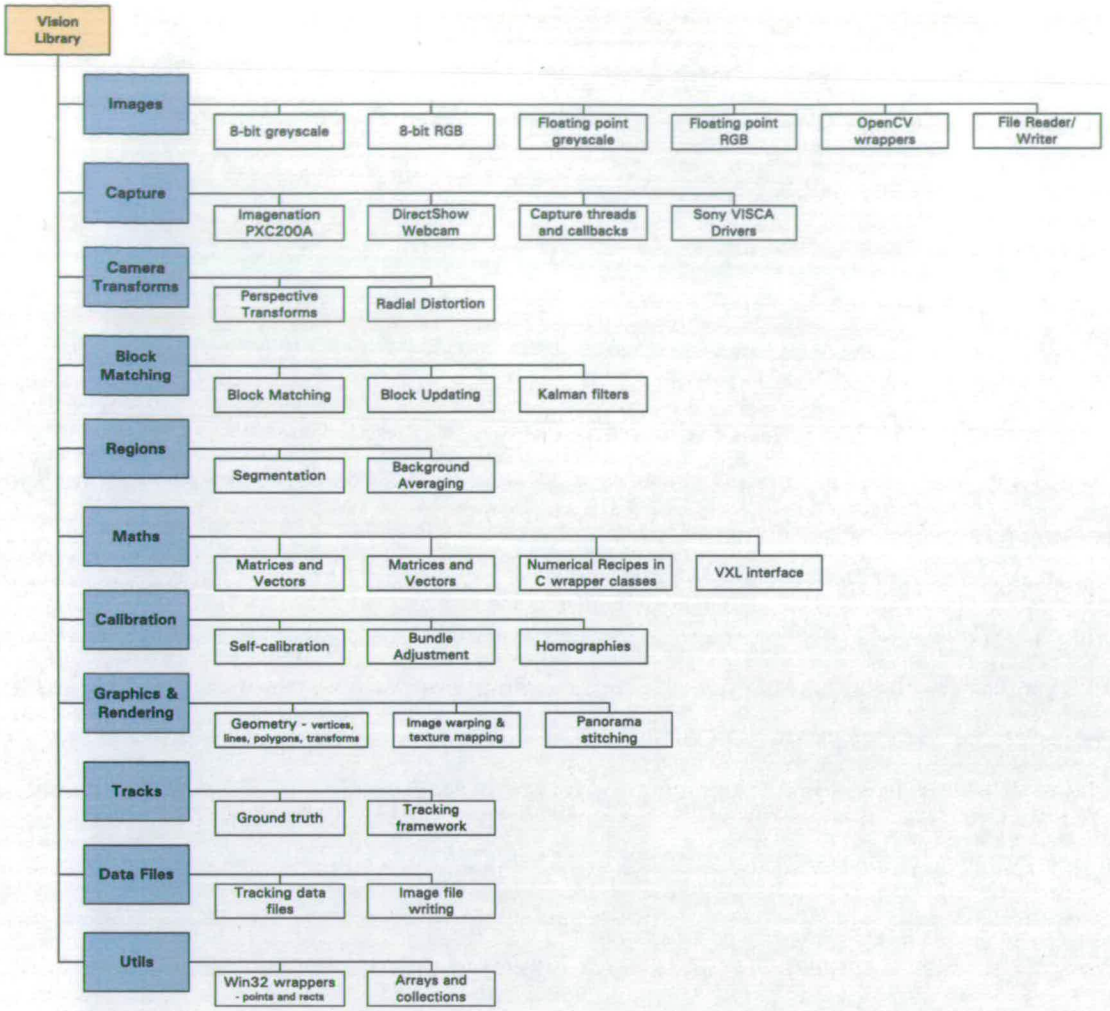


Figure D.5: *Vision library class diagram*

D.4.2 Grid Capture

A screenshot of the application used to capture pan, tilt and zoom sequences is shown in Figure D.7. The preview allows the user to select a path along which the camera will pan, tilt and zoom. The number of frames in each section of the trajectory can be adjusted. This allows easy timing of a sequence and good repeatability for re-shooting a sequence.

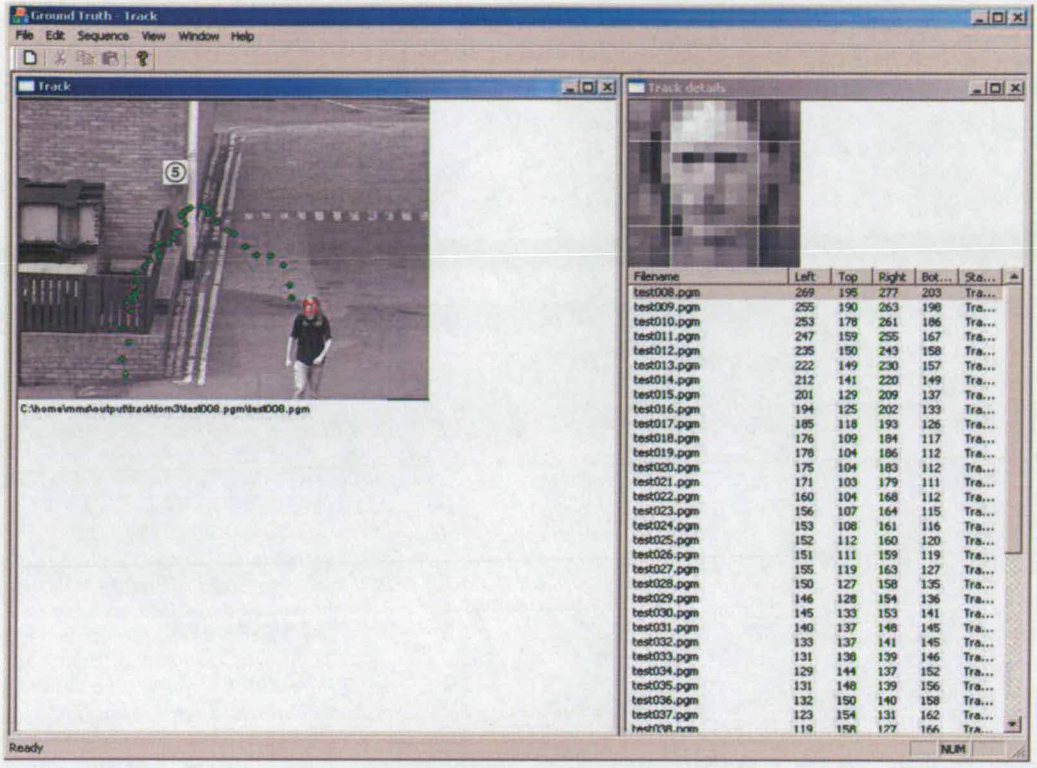


Figure D.6: *Ground Truth Application*

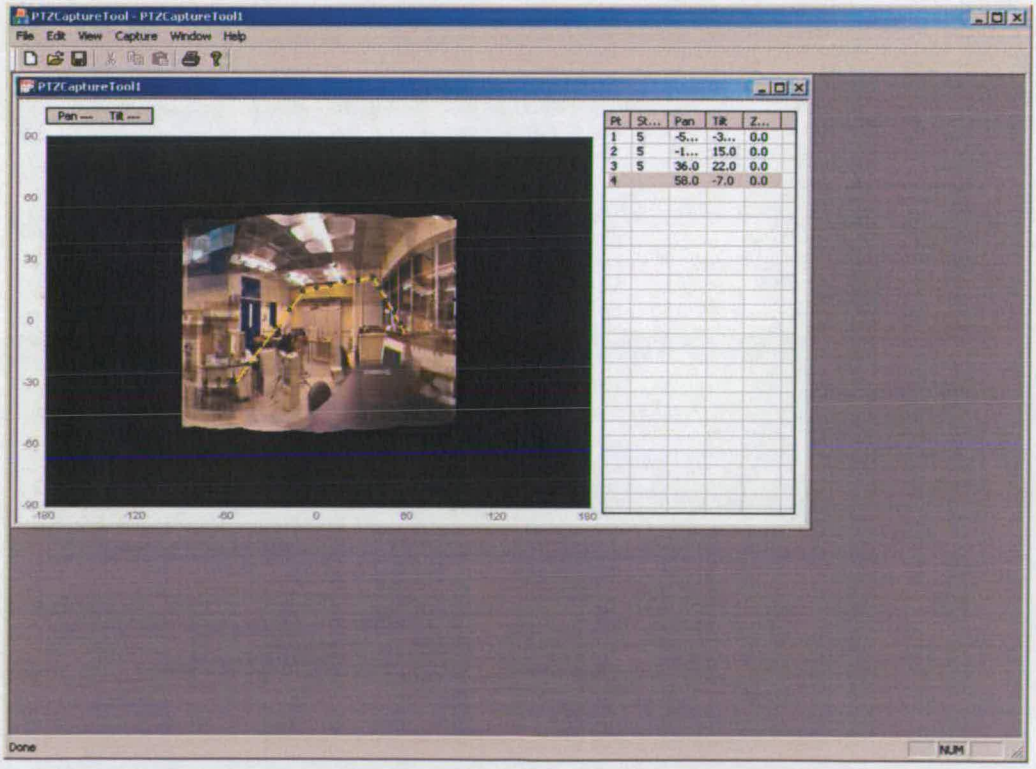


Figure D.7: *Grid Capture Application*