



UNIVERSITY OF
CAMBRIDGE

Department of Engineering

Visual Tracking for Augmented Reality

Georg Klein
King's College

A thesis submitted for the degree of
Doctor of Philosophy
January 2006

Declaration

This dissertation is submitted to the University of Cambridge in partial fulfilment for the degree of Doctor of Philosophy. It is an account of work undertaken at the Department of Engineering between October 2001 and January 2006 under the supervision of Dr T.W. Drummond. It is the result of my own work and includes nothing which is the outcome of work done in collaboration except where specifically indicated in the text. This dissertation is approximately 50,000 words in length and contains 39 figures.

Georg Klein

Acknowledgements

I thank my supervisor, Dr. Tom Drummond, and my colleagues in the Fall-side lab for their help and friendship. I have learned a great deal from them and hope they have enjoyed the last four years as much as I have.

I am grateful to the Gates Cambridge Trust for funding my research, and to Joe Newman for donating the Sony Glasstron display which made my work on Augmented Reality possible.

Finally I would like to thank my parents for their patience and continual support.

Abstract

In Augmented Reality applications, the real environment is annotated or enhanced with computer-generated graphics. These graphics must be exactly registered to real objects in the scene and this requires AR systems to track a user's viewpoint. This thesis shows that visual tracking with inexpensive cameras (such as those now often built into mobile computing devices) can be sufficiently robust and accurate for AR applications. Visual tracking has previously been applied to AR, however this has used artificial markers placed in the scene; this is undesirable and this thesis shows that it is no longer necessary.

To address the demanding tracking needs of AR, two specific AR formats are considered. Firstly, for a head-mounted display, a markerless tracker which is robust to rapid head motions is presented. This robustness is achieved by combining visual measurements with those of head-worn inertial sensors. A novel sensor fusion approach allows not only pose prediction, but also enables the tracking of video with unprecedented levels of motion blur.

Secondly, the tablet PC is proposed as a user-friendly AR medium. For this device, tracking combines inside-out edge tracking with outside-in tracking of tablet-mounted LEDs. Through the external fusion of these complementary sensors, accurate and robust tracking is achieved within a modest computing budget. This allows further visual analysis of the occlusion boundaries between real and virtual objects and a marked improvement in the quality of augmentations.

Finally, this thesis shows that not only can tracking be made resilient to motion blur, it can benefit from it. By exploiting the directional nature of motion blur, camera rotations can be extracted from individual blurred frames. The extreme efficiency of the proposed method makes it a viable drop-in replacement for inertial sensors.

Contents

1	Introduction	1
1.1	An Introduction to Augmented Reality	1
1.2	The Registration Challenge	2
1.3	Visual Tracking for Augmented Reality	3
1.4	AR with a Head-Mounted Display	5
1.5	AR with a Tablet PC	5
1.6	Exploiting Motion Blur	6
1.7	Layout of this Thesis	7
1.8	Publications	8
2	Background	9
2.1	Markerless Visual Tracking	10
2.1.1	Early Real-time Systems	10
2.1.2	Visual Servoing	14
2.1.3	Recent Advances in Visual Tracking	16
2.2	Tracking for Augmented Reality Applications	20
2.2.1	Passive Fiducial Tracking	21
2.2.2	Active Fiducial Tracking	25
2.2.3	Extendible Tracking	27
2.2.4	Inertial Sensors for Robustness	28

2.2.5	Combinations with Other Trackers	29
2.3	Augmented Reality Displays	31
2.3.1	Optical and Video See-through Displays	31
2.3.2	Advances in HMDs	33
2.3.3	HMD Calibration	34
2.3.4	Hand-held AR	37
2.3.5	Other AR Displays	40
2.4	Occlusions in AR	41
2.5	Motion Blur	46
3	Mathematical Framework	49
3.1	Coordinate frames	49
3.2	Motions	50
3.3	Uncertainty in Transformations	52
3.4	Software	53
4	Markerless Visual Tracking	54
4.1	Introduction	54
4.2	Tracking System Operation	57
4.2.1	Image Acquisition	58
4.2.2	Model Rendering and Camera Model	60
4.2.3	Image Measurement	62
4.2.4	Pose Update	63
4.2.5	Motion Model	64
4.3	Inertial Sensors	65
4.4	Sensor Fusion	66
4.4.1	Tracking System Initialisation	67
4.4.2	Parametric Edge Detector	68

4.4.3 Gyroscope Re-calibration	69
4.5 Results	71
5 HMD-Based Augmented Reality	74
5.1 Introduction	74
5.2 Head-Mounted Display	76
5.3 A Prototype Maintenance Application	78
5.4 Projection Model and Rendering	79
5.5 Registration	81
5.5.1 Registration for Optical See-through Displays	81
5.5.2 User Calibration Procedure	82
5.5.3 Nonlinear Optimisation	86
5.5.4 Dynamic Registration	87
5.6 Results	89
5.6.1 Maintenance Application	89
5.6.2 Calibration Performance	91
5.6.3 Dynamic Registration Error	92
5.6.4 Ergonomic Issues	94
6 Tablet-Based Augmented Reality	96
6.1 Introduction	96
6.2 A Tablet-based Entertainment Application	98
6.3 Tracking Strategy	100
6.4 Outside-in LED Tracking	103
6.5 Inside-Out Edge Tracking	109
6.6 Extended Kalman Filter	110
6.6.1 An Introduction to Kalman Filtering	110

6.6.2	Filter State	111
6.6.3	Prediction Step	113
6.6.4	Correction Step	114
6.6.5	Sensor Offset Calibration	115
6.7	Application Implementation	116
6.7.1	Kalman Filtering over the Network	116
6.7.2	Token detection	117
6.8	Rendering	118
6.9	Occlusion Refinement	123
6.10	Results	127
6.10.1	Real-time Performance	127
6.10.2	Errors	127
6.10.3	Dynamic Performance	128
6.10.4	Occlusion Refinement	129
6.10.5	The Tablet PC as an AR Format	130
7	A Visual Rate Gyroscope	132
7.1	Introduction	132
7.2	Method	135
7.2.1	Overview	135
7.2.2	Axis of Rotation	136
7.2.3	Blur magnitude	140
7.3	Results	141
7.4	Limitations	144
7.5	Combination with Edge Tracking	146
7.6	Conclusions	148

8 Conclusion	150
8.1 Summary	150
8.2 Contributions	151
8.3 Future Work	152
A Results Videos	154
B Projection Derivatives	157
B.1 Tracking Jacobian	157
B.2 HMD Calibration Jacobian	158
C M-Estimation	161
D Homographies	165
D.1 Estimating a Homography	165
D.2 Estimating Pose from a Homography	166
Bibliography	182

List of Figures

2.1	ARToolkit markers	24
2.2	An occlusion-capable optical see-through HMD	34
2.3	Occlusion handling in AR	44
4.1	Substantial motion blur due to 2.6 rad/s camera rotation	56
4.2	Tracking system loop	59
4.3	Lens comparison	60
4.4	Video edge search at a sample point	62
4.5	Rate gyroscopes affixed to camera	65
4.6	Long-term bias drift from three rate gyroscopes	66
4.7	Predicted motion blur vectors	69
4.8	Pixel and edge intensities from a tracked frame	70
4.9	Motion blur enlargements	72
5.1	Head-Mounted Display	75
5.2	Optical layout of the HMD	76
5.3	Image composition in the HMD	77
5.4	User's view vs. computer's view	81
5.5	Projection coordinate frames and parameters	82
5.6	Calibration seen through the display	85

5.7	Results captured from video	90
6.1	Maintenance application on the tablet PC	97
6.2	Tabletop game environment	99
6.3	Back of tablet PC	101
6.4	The game's sensors and coordinate frames	102
6.5	Tablet-mounted LEDs	104
6.6	Training procedure for LED matching	105
6.7	Run-time LED matching procedure	106
6.8	Predictor-Corrector cycle of the Kalman Filter	112
6.9	Token detection	117
6.10	A scene from Darth Vader vs. the Space Ghosts	120
6.11	Rendering Loop using z-Buffering	121
6.12	Occlusion error when using z-buffering	124
6.13	Occlusion refinement procedure	125
7.1	Canny edge-extraction of an un-blurred and a blurred scene	134
7.2	Operation of the Algorithm	137
7.3	Consensus evaluation	139
7.4	Rotation center placement results	142
7.5	Blur magnitude results	143
7.6	Failure modes	145
7.7	Pan across the AR game world	147

1

Introduction

1.1 An Introduction to Augmented Reality

Augmented Reality (AR) is the synthesis of real and virtual imagery. In contrast to *Virtual Reality* (VR) in which the user is immersed in an entirely artificial world, augmented reality overlays extra information on real scenes: Typically computer-generated graphics are overlaid into the user's field-of-view to provide extra information about their surroundings, or to provide visual guidance for the completion of a task. In its simplest form, augmented reality could overlay simple highlights, arrows or text labels into the user's view - for example, arrows might guide the user around a foreign city. More complex applications might display intricate 3D models, rendered in such a way that they appear indistinguishable from the surrounding natural scene.

A number of potential applications for AR exist. Perhaps the most demanding is AR-assisted surgery: In this scenario, the surgeon (using a suitable display device) can

view information from x-rays or scans super-imposed on the patient, e.g. to see important blood vessels under the skin before any incision is made. This is an example of *x-ray vision* which is made possible by AR: Information which is normally hidden from view can be revealed to the user in an understandable way. This information can be acquired from prior models (e.g. blueprints of a building, to show hidden plumbing) or acquired live (e.g. ultrasound in medicine.)

Augmented Reality can be used to give the user senses not ordinarily available. Data from arbitrary sensors can be presented visually in a meaningful way: For example, in an industrial plant, the sensed temperature or flow rate in coolant pipes could be visually represented by colour or motion, directly superimposed on a user's view of the plant. Besides visualising real data which is otherwise invisible, AR can be used to preview things which do not exist, for example in architecture or design: Virtual furniture or fittings could be re-arranged in a walk-through of a real building. Special effects of a movie scene could be previewed live in the movie set. Numerous entertainment applications are possible by inserting virtual objects or opponents into the real environment.

1.2 The Registration Challenge

The primary technical hurdle AR must overcome is the need for robust and accurate *registration*. Registration is the accurate alignment of virtual and real images without which convincing AR is impossible: A real chessboard with virtual pieces a few centimeters out of alignment is useless. Similarly, if a surgeon cannot be entirely certain that the virtual tumor he or she sees inside the patient's body is exactly in the right place, the AR system will remain unused - **Holloway** (1995) cites a surgeon specifying an offset of 1mm at a viewing distance of 1m when asked what the maximum acceptable error could be. The level of rendering precision required to operate a believable AR system far exceeds the requirements of VR systems, where the absence of the real world means small alignment errors are not easily perceived by the user.

When a user remains motionless and receives pixel-perfect alignment of real and virtual images, a system can be said to offer good *static* registration. However AR systems should further exhibit good *dynamic* registration: When moving, the user should notice no lag or jitter between real and virtual objects, even when undergoing rapid and erratic motion. Registration errors of either kind produce the effect of virtual objects ‘floating’ in space and destroy the illusion that they are of the real world.

The full requirements for achieving registration differ according to the display format employed, but a crucial component is almost invariably the continual knowledge of the user’s viewpoint, so that virtual graphics can be rendered from this. For the traditional case in which the user wears a head-mounted display (HMD), this amounts to tracking the 6-DOF pose of the user’s head. A variety of sensors have previously been used to track a user’s head, from accurate mechanical encoders (which restrict users to a small working volume) to magnetic or ultrasound sensors (which rely on appropriate emitters placed in the environment). This thesis shows that accurate and robust registration is possible without expensive proprietary sensors, using cheap off-the-shelf video cameras and *visual tracking*.

1.3 Visual Tracking for Augmented Reality

Visual Tracking attempts to track head pose by analysing features detected in a video stream. Typically for AR a camera is mounted to the head-mounted display, and a computer calculates this camera’s pose in relation to known features seen in the world. As the cost of computing power decreases and video input for PCs becomes ubiquitous, visual tracking is becoming increasingly attractive as a low-cost sensor for AR registration; further, in an increasingly large number of *video see-through* AR systems in which augmentations are rendered onto a video stream (cf. Section 2.3) a video camera is already present in the system.

Unfortunately, real-time visual tracking is not a solved problem. Extracting pose from a video frame requires software to make correspondences between elements in the im-

age and known 3D locations in the world, and establishing these correspondences in live video streams is challenging. So far, AR applications have solved this problem by employing *fiducials*, or artificial markers which are placed in the scene. These markers have geometric or color properties which make them easy to extract and identify in a video frame, and their positions in the world are known. Approaches to fiducial tracking are listed in Section 2.2.

Placing fiducials in the scene works very well for prototype applications in prepared environments, but is ultimately undesirable. This is not only because of aesthetic considerations: The placement and maintenance of fiducials may just not be practical when dealing with large environments or even multiple instances of the same environment. This thesis therefore focuses on *markerless* (also called *feature-based*) tracking which uses only features already available in the scene. Since matching features in real-time is difficult, markerless tracking systems typically operate under a number of simplifying assumptions (such as the assumption of smooth camera motion and frame-to-frame continuity) which result in a lack of *robustness*: Existing systems cannot track the range and rapidity of motion which a head-mounted camera in an AR application can undergo.

In Chapter 4 of this thesis I show how the required robustness can be achieved. A real-time tracking system which uses a CAD model of real-world edges is partnered with three *rate gyroscopes*: low-cost, low-power solid-state devices which directly measure the camera's rotational velocity. By equipping the visual tracking system with a very wide-angle lens and initialising pose estimates with measurements from these inertial sensors, gains in robustness can be achieved, but only up to a point. Soon, *motion blur* caused by rapid camera rotation causes sufficient image corruption to make traditional feature extraction fail. Chapter 4 shows how this motion blur can be predicted using the rate gyroscopes, and how the edge detection process used by the tracking system can accordingly be modified to detect edges even in the presence of large amounts of blur. This tightly-coupled integration of the two sensors provides the extra robustness needed for head-mounted operation.

1.4 AR with a Head-Mounted Display

Head pose tracking is a primary requirement for workable head-mounted AR, but it is not the only requirement. Chapter 5 of this thesis describes the development of an AR application based on an optically see-through HMD and shows what further steps are required to obtain good registration. A significant challenge for AR registration in this case is display *calibration*: This effectively determines the positions of the user's eyes relative to their head, and the projection parameters of the display hardware. Due to the curved mirrors commonly used in head-mounted displays, registration is often impaired by distortions in the projection of virtual graphics; I show that such distortions can be estimated as part of the user calibration procedure, and modern graphics hardware allows the correction of this distortion with very low cost. To provide acceptable dynamic registration, the delays inherent in visual tracking and graphical rendering need to be addressed; this is done by exploiting low-latency inertial measurements and motion predictions from a velocity model.

The completed HMD-based system (which re-implements a mock-up of **Feiner et al** (1993)'s printer-maintenance application) works, but reveals some shortcomings both of the tracking strategy used and of the HMD as a display device. This raises the question of whether a HMD should still be the general-purpose AR medium of choice, or whether better alternatives exist.

1.5 AR with a Tablet PC

Chapter 6 demonstrates that the *tablet PC* - a new class of device which combines a hand-held form-factor with a pen input and substantial processing power - can be used as a very intuitive medium for AR. Among a number of ergonomic advantages over the HMD previously used, the tablet PC offers brighter, higher-resolution, full-colour graphical overlays; however these increased graphical capabilities also create a new set of registration challenges. Small latencies are no longer an issue, but very high-accuracy overlays are required; graphics need no longer be brightly-coloured

wire-frame overlays, but the occlusion of real and virtual objects must now be correctly handled. Chapter 6 shows how local visual tracking on occluding edges of the video feed can be used to improve rendering accuracy, while blending techniques can improve the seamless integration of virtual graphics into the real world.

The potential of tablet PCs as an AR medium is demonstrated by a fast-paced entertainment application. The demands this application makes on a visual tracking system are different from those of the head-mounted system: Rapid head rotations are no longer the dominant cause of on-screen motion; instead, rapid close-range translations and repeated occlusions by the user's interaction with the application must be tackled by the tracking system. Chapter 6 shows that the required performance is achievable through a fusion of markerless inside-out tracking and fiducial-based outside-in tracking, which uses LEDs placed on the back of the tablet PC. This approach combines the high accuracy of the previously used markerless system with the robustness of fiducials, all without requiring the environment to be marked up with intrusive markers.

1.6 Exploiting Motion Blur

The majority of this thesis shows how visual tracking can be used to deal with specific requirements posed by AR applications; in Chapter 7, I instead present a technique which is not applied to any specific application, but which may be a useful addition to any visual tracking system used for AR.

So far, motion blur in video images has been considered an impediment to visual tracking, a degradation which must be dealt with by special means - for example by the integration of rate gyroscopes. Motion blur is however not completely undesirable, as it makes video sequences appear more natural to the human observer. Indeed, motion blur can give humans valuable visual cues to the motions happening in individual images; this suggests that computer tracking systems, too, should be able to gain information from blur in images.

If the number of sensors required for an AR system can be decreased, this reduces the system's complexity, cost, and bulk - hence, being able to replace physical rate gyroscopes with a visual algorithm would be advantageous. Chapter 7 shows how motion blur can be analysed to extract a camera's rotational velocity from individual video frames. To be a generally applicable replacement for rate gyroscopes, this algorithm should be extremely light-weight (so that no additional computing hardware is required) and should not rely on any features of individual tracking systems, such as edge models or fiducials. Chapter 7 shows that by imposing some sensible simplifying assumptions, rotation can be extracted from video frames in just over 2 milliseconds on a modern computer - less than a tenth of the computing budget available per frame.

1.7 Layout of this Thesis

The above introduction has outlined the main contributions described in the body of this thesis: Markerless tracking is described in Chapter 4, its applications to HMD- and tablet-based AR in Chapters 5 and 6 respectively, and the visual gyroscope algorithm is presented in Chapter 7.

A review of previous work in the fields of visual tracking and augmented reality is given in Chapter 2: The origins of the visual tracking system used, alternative tracking strategies, different categories of AR displays, strategies previously used to deal with occlusions, and existing approaches to tackle or exploit motion blur of real and virtual objects are described.

Chapter 3 then briefly introduces the mathematical framework used throughout the remainder of the thesis. This is based on the Euclidean group $SE(3)$ and its Lie algebra. The notation used for coordinate frames, their transformations, and motions are explained.

Chapter 8 concludes the body of the thesis with a summary of the contributions made and discusses issues meriting further investigation. Several appendices subsequently

describe some of the mathematical tools used in the thesis.

Most of the work presented in this thesis involves the analysis of live video feeds; consequently it is difficult to convey aspects of the methods used and results achieved using only still images. Several illustratory video files have been included on a CD-ROM accompanying this thesis, and they are listed in Appendix A.

1.8 Publications

The majority of the work described in this thesis has been peer-reviewed and presented at conferences. This is a list of the publications derived from this work:

Klein & Drummond (2002, 2004b): Tightly Integrated Sensor Fusion for Robust Visual Tracking. In *the proceedings of the British Machine Vision Conference (BMVC)*, Cardiff, 2002. Also appears in *Image and Vision Computing (IVC)*, Volume 22, Issue 10, 2004. Winner of the BMVA best industrial paper prize.

Klein & Drummond (2003): Robust Visual Tracking for Non-Instrumented Augmented Reality. In *the proceedings of the 2nd IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR)*, Tokyo, 2003.

Klein & Drummond (2004a): Sensor Fusion and Occlusion Refinement for Tablet-based AR. In *the proceedings of the 3rd IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR)*, Arlington, 2004. The system described in this paper was also presented live to conference delegates in a demo session.

Klein & Drummond (2005): A Single-frame Visual Gyroscope. In *the proceedings of the British Machine Vision Conference (BMVC)*, Oxford, 2005. Winner of the BMVA best industrial paper prize.

2

Background

This chapter describes past and present work in the fields of visual tracking and augmented reality. Initially, early real-time markerless tracking systems described by the computer vision community are reviewed, and then recent advances in markerless tracking are presented. Few of these systems have been applied to AR, so a review of the tracking strategies typically employed in AR systems follows. Further, this chapter compares and contrasts different display technologies used for AR, and the calibration procedures which have been applied to head-mounted displays. Approaches to occlusion handling in AR applications are presented. Finally, existing computer vision approaches which exploit (or are robust to) motion blur in images are examined.

2.1 Markerless Visual Tracking

2.1.1 Early Real-time Systems

The RAPiD (Real-time Attitude and Position Determination) system described by **Harris** (1992) was one of the earlier markerless model-based real-time 3D visual tracking systems. Such early real-time systems had access to far less computing power per frame than is available today; while one solution to this problem was the use of dedicated video processing hardware, Harris developed a technique which minimised the amount of data which needed to be extracted from the video feed. Many subsequent visual tracking systems (including the system used in this thesis) share principles of operation with Harris' work; therefore, a description of the system serves as a good introduction to marker-less visual tracking techniques.

In RAPiD, the system state contains a description of camera pose relative to a known 3D model. Pose is represented by six parameters, three representing translation and three representing rotation. Each video frame, the pose estimate is updated: first by a prediction according to a dynamic motion model, and then by measurements in the video input. Measurements in the video input are made by rendering model edges according to the latest predicted pose, and measuring the image-space difference between predicted and actual edge locations. Edge searches are local, one-dimensional, and confined to a small region near the rendered edges: This vastly reduces the computational burden of video edge-detection and enables real-time operation. The edge searches originate from a number of control points located along rendered edges and are in a direction perpendicular to the rendered edges. Typically 20-30 control points are used per frame.

Having measured the distances (errors) between rendered and actual image edges, pose is updated to minimise these errors. This is done by linearising about the current pose estimate and differentiating each edge distance with respect to the six pose parameters. Generally the system is over-determined, and errors cannot all be made zero: Instead, the least-squares solution is found.

RAPiD demonstrated real-time tracking of models such as fighter aircraft and the terrain near a landing strip. It required pre-processing of the models to determine visibility of control-points from various camera positions (this was achieved by partitioning a view-sphere around the model.) Increases in processing power and advances in technique have since given rise to many systems which take the basic ideas of RAPiD to the next level.

Alternative approaches to Harris' at the time included work by **Gennery** (1992) and **Lowe** (1992). Where RAPiD performs one-dimensional edge detection along a few short lines in the image, Lowe uses Marr-Hildreth edge detection on substantially larger portions of the image using hardware acceleration for image convolution.¹ The probabilities of a fit of individual model edge segments to extracted lines are calculated according perpendicular edge distance, relative orientation and model parameter covariance, and these likelihoods are used to guide a search to best match model edges to detected lines. This match is then used to update model parameters by least squares; if the least-squares residual error is found to be large, the match is rejected and the search repeated until a low-residual solution is found. The computational expense of this method limited the system to operation at 3-5 Hz compared to RAPiD which ran at 50Hz. However, the system already allowed the use of models with internal degrees of freedom and was robust in the face of background clutter.

Gennery (1992) also uses specialised edge-detection hardware, but searches the edge-detected image in a fashion similar to RAPiD's: For each predicted model edge, corresponding image edges are found by one-dimensional perpendicular² searches. However, where RAPiD uses few control points per edge, Gennery places a control point every three pixels. Furthermore, measurements are weighted according to the quality of an edge. Background boundary edges are rated as being of high quality; internal edges are weighted according to contrast and the angle difference between adjacent faces.

¹The algorithm and convolution hardware could handle full-frame edge detection, but the computational expense of edge-linking after convolution limits the implementation to perform edge-detection in a limited region around the predicted edge locations.

²Gennery only searches in the vertical or horizontal direction, whichever is closer to the perpendicular; RAPiD and others also search diagonally.

Gennery further proposes to penalise edges which deviate from the expected image orientation, and considers finding all edges near control points instead of the nearest edge only. Finally, the system can operate on point features instead of line features, and has provisions for the use of multiple cameras to remove position uncertainty along the direction of the optical axis. This is made possible by the propagation of uncertainty estimates using a Kalman filter (**Kalman**, 1960) (Kalman filters and Extended Kalman Filters (EKFs) which operate under non-linear dynamics are commonly used in tracking, and are described in Section 6.6.)

Both Harris and Gennery make use of a Kalman filter to predict motion. In both cases, the filter contains object pose and the first derivative (velocity.) Plant noise is modeled in both cases by assuming that acceleration is a zero-mean white random variable, and therefore uncertainty is added to the velocity component in the filter. Harris allows this uncertainty to propagate to position through the filter, whereas Gennery adds it explicitly. Lowe argues that a Kalman filter is inappropriate for dynamics as they occur in robotics, and replaces the filtering of the computed least-squares estimate with the inclusion of a prior stabilising weight matrix to the least-squares process (this can give results similar to the use of a Kalman filter which does not track velocity.)

Approaches to improving RAPiD's robustness are presented by **Armstrong & Zisserman** (1995). The authors identify a number of performance-degrading conditions often occurring in real-life tracking situations and propose steps to reduce their impact. Primarily, this is done by the identification and rejection of outliers, which is done at two levels.

First, the model of the object to be tracked is split into a number of primitives, which can be straight lines or conics. Control points are placed on these primitives as for RAPiD, but their number is such that redundant measurements are taken for each primitive. RANSAC (**Fischler & Bolles**, 1981) is then used to cull outlying control point measurements. For the case of a straight line primitive, for example, a number of hypothetical lines constructed from the measurements of two randomly chosen control points are evaluated according to how many of the other control point mea-

surements fall near the proposed lines. The highest-scoring line is used to eliminate control points which do not fall near it.

After outlying control points have been rejected, outlying primitives are identified and deleted. For each detected primitive in turn, a pose update is computed using only the control points of all the other primitives. The primitive is compared to its projection in this computed pose. If there is a significant difference, the primitive is classed an outlier and rejected. After all primitives have been tested and rejected if necessary, a pose update is computed from the remaining primitives only. The weight of each primitive in the pose update is determined by a confidence metric, which is calculated from the number of times a primitive has been previously deleted.

The removal of outliers is important for algorithms using least-squares optimisation. Standard least squares attempts to minimise the sum of the *squares* of errors: therefore, the influence of each error on the solution is directly proportional to its magnitude. Since outliers frequently produce errors of large magnitudes and hence have a large effect on a least-squares solution, their removal is desirable. An alternative to explicit outlier removal by case deletion or RANSAC is the use of a robust *M-estimator*: the influence of large errors is reduced by replacing the least-squares norm with an alternative function whose differential (and hence influence function) is bounded. M-estimators are described further in Appendix C.

Drummond & Cipolla (1999) employ an M-estimator to improve the robustness of a RAPiD-style edge tracking system. Further, RAPiD's view-sphere approach to determining control point visibility is replaced by a real-time hidden edge removal based on graphics acceleration hardware and a BSP-tree¹ representation of the model to be tracked; This allows the tracking of more complex structures than possible with RAPiD. This system forms the basis for the visual tracking systems described in this thesis and further details of its operation are given in Chapter 4.

Marchand et al (1999) use an M-estimator as part of a 2D-3D tracking system. A veloc-

¹A *Binary Space Partition* is a common graphics technique for sorting elements of 3D scenes in depth order according to the current viewpoint.

ity model such as a Kalman filter is not used; instead, motion in the image is computed using a 2D affine motion model. Perpendicular edge searches are performed around the object's outline and a robust M-estimator calculates the 2D affine transformation which best matches the newly detected edges. Subsequently, the 3D pose is refined by searching in the pose parameter space in an attempt to place model edges over image areas with a high intensity gradient. While the entire algorithm is computationally reasonably expensive, the affine tracking runs at frame rate and - due to the use of the M-estimator - is capable of handling partial occlusion and background clutter.

Robust estimators *and* outlier rejection are used by **Simon & Berger** (1998) to track a known model consisting of three-dimensional curves. Using a motion field estimate, snakes are fitted to image curves near the previous pose. Each visible model curve is projected and sampled, with distances to the snake taken each sample. The samples are combined using a robust estimator to produce an error value for the entire curve, the differentials of which with respect to pose are known. All of the curves' error values are then minimised with a global robust estimator. After this step, curves with a high residual error after the global optimisation are rejected, and the pose is refined, this time minimising the curve errors with standard least squares. The algorithm is not implemented in real-time, but deals well with snakes which curve around local minima. Besides curves, line and point features are also supported.

2.1.2 Visual Servoing

An alternative formulation to the visual tracking problem has emerged from the robotics community under the guise of *visual servoing*. Here a camera is typically attached to a robot and vision is used in closed-loop control system to position the robot relative to a visual target. The work of **Espiau & Chaumette** (1992) has served as the foundation for subsequent applications to the task of visual tracking.

Espiau & Chaumette (1992) use visual servoing to guide a six-jointed robot to a target position. At each time step, a Jacobian matrix relating the robot's degrees of freedom to the image error between current and target position is calculated, and the robot joint

velocities set so as to minimise the image error. Closed-form image derivatives under camera motion are derived for many different geometric primitives, including points, lines and spheres, and the work places emphasis on a stable control law; in terms of vision, experimental results are limited to servoing relative to four bright white disks, which is done at frame-rate.

This work in robotics has led to the concept of *virtual visual servoing* - this is essentially visual tracking from a different perspective: A camera model is considered to be attached to a virtual robot; each frame, this virtual robot is servoed in such a way as to match the image visible in the video camera, and so camera pose is determined. This formulation has been applied to augmented reality systems by **Sundareswaran & Behringer** (1998), **Behringer et al** (2002), and **Marchand & Chaumette** (2002); the approaches differ in type of image feature employed for measurement. **Sundareswaran & Behringer** (1998) use circular concentric ring fiducial markers which are placed at known locations on a computer case to overlay the hidden innards of the case on the camera's video feed. These markers are unique and can be found by searching the image; this allows the system to initialise itself from an unknown viewpoint and prevents tracking failure due to correspondence errors, but incurs a computational performance penalty. **Behringer et al** (2002) extend this approach to include edges and corners already present in the scene - these features are described in a CAD model.

Marchand & Chaumette (2002) do not employ markers but demonstrate visual servoing-based AR using a number of preexisting model features including points, lines and (the edges of) cylinders, all of which have known 3D locations. Interestingly, **Comport et al** (2005) compare the virtual visual servoing formulation with the system of **Drummond & Cipolla** (1999) and conclude that the fundamentals of both systems are very similar - although the system by Drummond & Cipolla (which forms the basis for the tracking used in this thesis) does not fully exploit the benefits of M-Estimators since it only performs one iteration per frame.

2.1.3 Recent Advances in Visual Tracking

A number of interesting developments in model-based tracking have been presented in recent years. One such advance is the integration of point features into edge-based trackers, which had been demonstrated by **Vacchetti et al** (2004) and **Rosten & Drummond** (2005).

The tracking of feature points on objects has the advantage that such feature points often provide a descriptor, allowing the correspondence of features from one frame to the next to be determined. This is in contrast to edge-based tracking, where correspondence is generally based on proximity to a prior estimate rather than the appearance of an edge; this can lead to edge-based trackers ‘locking on’ to incorrect edges, resulting in tracking failure. The disadvantage of using descriptors for interest points (which can be as simple as small image patches) is that they are often of limited invariance to aspect and lighting changes. To operate across these changes, many approaches update descriptors with time; this updating can however lead to feature *drift* where, after many frames, the 3D position of the feature no longer corresponds to its original location. Edges on the other hand are invariant to pose and illumination changes (even if their appearance may not be) and so tracking based on CAD edge models is drift-free.

Vacchetti et al (2004) combine edge-based tracking with earlier work on tracking Harris feature points. For every new frame, interest points are first found in the image and then matched with interest points in the closest of a number of offline reference frames; the 3D model positions of interest points in the reference frames are known, so this gives 2D-3D correspondences for the current frame, from which a pose estimate may be obtained. To avoid the jitter incurred by such potentially wide-baseline matching, and to handle objects which are not richly textured but are bounded by strong edges (e.g. white walls) this initial pose estimate is combined with both edge measurements and feature point correspondences with the previous frame; by optimising over the current frame’s pose, the previous frame’s pose and the 3D positions of the matched feature points, smooth camera trajectories are obtained. (The authors further

describe a multi-modal estimator allowing the use of multiple hypotheses from individual edgels, however this just corresponds to multiple iterations of standard RAPiD with the addition of an M-estimator, using the nearest hypothesis at each iteration.)

Rosten & Drummond (2005) apply the combination of edge and point tracking to an indoor environment. The authors argue that in such a scenario, the maintenance of up-to-date key-frames is an unrealistic proposition which further does not account for feature-rich environmental clutter. The offline model is restricted to edges, and corner points extracted using the novel FAST algorithm are matched from previous to current frame only, providing a motion estimate between the two frames which can exploit clutter rather than being corrupted by it. Very large inter-frame motions are tolerated by estimating the probability of each feature match being an inlier before using Expectation Maximisation to model the data likelihood distribution; inlier probabilities are estimated by learning (over time) a feature-vector-SSD to inlier probability mapping function. The Expectation Maximisation avoids the local minima associated with direct unimodal optimisations and allows for impressive inter-frame camera motions even in the presence of a large fraction of outliers. Following the point-based motion prediction, edge based tracking proceeds in the standard fashion.

Besides the additional use of point features, a number of other improvements to RAPiD-style edge tracking have been presented. Notable are the texture-based edge detector of **Shahrokni et al** (2004), which improves the quality of edge detection beyond the simple search for an intensity disparity, and the multi-modal extensions of **Kemp & Drummond** (2004, 2005), which tackle the tendency of a unimodal estimation system to ‘get stuck’ in local minima.

Shahrokni et al (2004) propose a replacement for the intensity-based edge detection which forms the basis for most RAPiD-style trackers. The 1D texture change-point detector models pixels along a line as belonging to one of two texture classes, where the point along the line at which the class changes being the detected edgel position. Textures are modeled as generated by either a zeroth- or a first-order Markov process and prior knowledge of the texture models can be used beneficially, but is not required.

The resulting algorithm is not substantially slower than a simple intensity-based detection.

Kemp & Drummond (2004) modify this texture-based edge detector so that it can yield multiple edgel locations rather than a single maximum. Instead of then using a unimodal estimator, edgel measurements are then combined using two stages of RANSAC: First, RANSAC is performed on each individual model edge to find hypotheses for the 2-DOF motion of each edge; then, a second stage of RANSAC finds 6-DOF camera pose from three edge hypotheses. In the second stage, the sampling of edge hypotheses is guided by a probabilistic extension to the RANSAC algorithm which approximates the posterior distribution of edge parameters for any given model edge. The use of multiple hypotheses makes this system markedly more robust to the local minima which unimodal RAPiD implementations (including the ones described in this thesis) are subject to.

The second stage of this approach samples only some of the possible combinations of three edge hypotheses, since the total number of such combinations may be vast and each hypothesis requires a 6×6 matrix inversion for evaluation. **Kemp & Drummond** (2005) introduce a dynamic clustering algorithm which allows edges to be grouped into a small number of clusters, where the edges in each cluster constrain fewer than the full six pose degrees of freedom - that is, lines of the measurement Jacobian are grouped to produce (approximately) rank-deficient groups. Each cluster might then only require a 4-DOF or 2-DOF parameter estimation. For example, vertical lines far away from the camera might be grouped into a cluster which is sensitive only to camera yaw and roll, but not to pitch or translation. The reduced cost of searching within each lower-dimensional cluster allows the authors to exhaustively search all the combinations of individual edge hypotheses rather than sampling a smaller number using RANSAC.

Besides improvements to model-based tracking, recent years have seen the emergence of several real-time systems capable of tracking a camera moving in a previously unknown scene. This class of system originates from robotics, where a robot (such as a

Mars lander) may be expected to explore an unknown environment. In AR, the usefulness of a system which can map completely unknown surroundings is less clear (if nothing is known about the environment, what useful annotations can one expect?) but such systems could well be adapted for *extendible tracking*, where the gaps in knowledge of sparse pre-existing models are filled in on-line.

Davison (2003) presents a real-time monocular vision-based SLAM (Simultaneous Localisation and Mapping) implementation. This system tracks feature points using small 2D image patches and cross-correlation; apart from four known features initialised at start-up, the 3D positions and appearances of all other features tracked are learned on-line. The coupled uncertainties of camera pose, velocity and feature point 3D locations are stored in a large Kalman filter. Real-time performance is achieved by restricting the expensive cross-correlation matching to the 2D ellipsoidal projections of likely feature positions given the current filter state.

Care is taken to avoid inserting non-Gaussian uncertainties into the filter state: when new features are detected using a roving saliency detector, they are not immediately inserted into the scene map (where they would have an infinite covariance in the viewing direction) but are first tracked in 2D for a number of frames while a particle filter is applied to the feature's depth estimate. Only when this particle filter converges on an approximately Gaussian depth PDF is the feature inserted into the Kalman filter.

While Davison's system is capable of handling respectably fast camera motion, it is based on a uni-modal filter and thus prone to unrecoverable tracking failures due to local minima or excessive motions. **Pupilli & Calway** (2005) present a particle-filter based tracking system which attempts to address these failure modes. Particle filters, which (given a sufficient number of samples) can be used to represent a distribution containing numerous modes (which need not be Gaussian) have been known to provide extraordinary tracking robustness in low-DOF real-time systems such as CONDENSATION (**Isard & Blake**, 1998). However the number of particles required rises exponentially with the number of degrees of freedom tracked; Pupilli and Calway demonstrate that by using a fast inlier/outlier count of tracked image features to evaluate individual particles, 6-DOF camera tracking using a particle filter is now

achievable in real-time. The estimated environment map however is not included in the particle filter and individual features positions are modeled as statistically independent (no full covariance) - thus while the system is more robust than Davison's to rapid camera motions and occlusions, the quality of map estimation is lower.

Several of the systems described here offer tracking performance superior to the system used for AR tracking in this thesis. However it is worth noting that this performance typically requires large amounts of processing power - few of systems just described would run at video rates on the tablet PC used in Chapter 6, let alone leave any spare processor time for AR rendering. Further, systems which use points and texture patches are often not operable in the presence of motion blur.

2.2 Tracking for Augmented Reality Applications

This section outlines the tracking strategies used in recent AR applications. A wide range of non-visual tracking technologies, such as magnetic and ultrasound, have been applied to AR as described in recent surveys by **Azuma et al** (2001) and **Rolland et al** (2000); however, the low cost of video cameras and the increasing availability of video capture capabilities in off-the-shelf PCs has inspired substantial research into the use of video cameras as sensors for tracking.

Despite this, *markerless* visual tracking is relatively rare in augmented reality applications. Recent years have seen the emergence of a few marker-less systems proposed for AR but most of these do not go beyond the “textured cube” or “teapot” stage: trivial insertions of simple graphics into the video feed of vision researchers’ experimental tracking systems. By contrast, it is still common for more demanding augmented reality applications to make use of *fiducials*: easily recognisable landmarks such as concentric circles placed in known positions around the environment. Such fiducials may be *passive* (e.g. a printed marker) or *active* (e.g. a light-emitting diode); both types of fiducial have been used in AR applications.

2.2.1 Passive Fiducial Tracking

One of the earlier passive-fiducial systems is described by **Mellor** (1995). Small (1cm) printed black rings are attached to a plastic skull to investigate registration accuracy for medical x-ray vision. The 3D positions of the fiducials are determined off-line using a laser range scanner. During operation, fiducial positions are extracted from the video image; the fiducials are identical in appearance, so correspondence between the 3D marker information and the detected 2D positions is resolved by an exhaustive search of the combinations to find the best match. Five or more correspondences are used to solve for the 12-DOF projection matrix, which is used to render a scanned model onto the captured video image.

Mellor's fiducial detection operates by searching all vertical and horizontal scan-lines for light-dark-light-dark-light patterns corresponding to the intersection of the scan-line with the ring fiducials. Matches from adjacent scan-lines are grouped to detect possible fiducials, which are then located using image moments. Both the fiducial detection and correspondence searches are sped up by using information from previous frames; fiducial detection is initially performed in windows around the previous frame's detected positions, and if this succeeds, correspondence is propagated. Only if enough fiducials are not found in this way is a full-frame extraction and correspondence search performed.

The system achieves a 2Hz frame-rate, but this is mostly due to the slow video-capture capability of the hardware used; the feature-extraction, correspondence and projection matrix calculations themselves are capable of interactive frame-rates (20 Hz.) The composited image is displayed on a CRT monitor.

Hoff et al (1996) use similar concentric circle markers for their head-mounted maintenance application. The user wears a head-mounted see-through display, and head pose is determined by a head-mounted camera. The fiducials are placed on a desktop computer case. The user is shown maintenance instructions such as arrows indicating parts or motions overlaid on the scene.

Fiducials are detected by first segmenting the image into black and white regions. Next, small regions of black and white are eliminated. Finally, the centroids of remaining connected single-colour regions are calculated. Where the centroid of a black region coincides with the centroid of a white region, a fiducial is found. Matching of fiducials is simplified by grouping five of the markers in a unique pattern in which three are co-linear: thus once three co-linear fiducials have been found in the image, the detection of the pattern and subsequent matching of the remaining features is straightforward. Four fiducials are used to calculate camera pose using a pre-calibrated camera model. The registration of multiple target objects is supported by using more than one unique cluster of five fiducials.

No attempt is made to re-use information from previous frames: rather, the system performs a localisation step every video frame. While this continual re-localisation overcomes the limitations of local search used by other systems, it requires a video capture board with built-in video processing functionality to process the entire image at every time step. Given that the system contains no notion of velocity (this was planned as future work) it cannot make any predictions of future camera motion; since an optical see-through display is used (see Section 2.3) substantial dynamic registration errors could be expected. On the other hand, the state estimate cannot be corrupted, so tracking cannot catastrophically fail.

Neumann & Cho (1996) simplify the correspondence problem by utilising full-colour frames. Stick-on paper circles and triangles in six different colours are used to annotate aircraft parts for assembly. Fiducials are detected by searching a heavily sub-sampled frame for the correct colour; this is initially done around the previous frame's position, or across the full frame if the local search fails. RANSAC is employed to find pose from any three correspondences. Due to sub-sampling and local search, the algorithm is rapid, requiring 0.03 seconds to calculate pose from a 320×240 pixel frame, however a slow digitizer slows the actual application down to 6Hz. The 3D positions of fiducials are acquired using a digitizer probe.

Cho et al (1997) extend this work by eliminating the need for a precise prior knowledge of the 3D position of all placed fiducials. A core set of initial fiducials has known

positions, but after this, new fiducials in the scene are detected and their pose automatically determined. New fiducials are initialised along an image ray with a covariance matrix describing the 3D uncertainty of the point's position in space; this uncertainty is initially set to be a sphere. With subsequent frames, the point's position is adjusted, and uncertainty shrunk, by slightly ad-hoc adjustments depending on the point's current uncertainty and new detected image positions. In contrast to later systems (in particular, in contrast to SLAM systems) the covariance of each point is independent; camera position uncertainty is not considered. **Cho & Neumann** (1998) also refine the concept of colour-coded fiducials by developing multi-ring coloured fiducials; instead of coloured circles, fiducials consisting of concentric coloured rings of different thicknesses are used, which both increases the number of unique features which can be generated and aids feature detection.

Koller et al (1997) also use fiducials, but propagate a state estimate with a 15-DOF Kalman filter, which contains pose, first derivatives, and second derivatives for translation: a constant angular velocity and constant linear acceleration motion model is used. Large squares are used as fiducials. On one colour channel, these contain a binary identification encoding which is used to solve the correspondence problem for tracking initialisation.

Tracking is performed locally around predicted fiducial positions, and the corners of each marker are detected by searching for edges, and then calculating the position where the edges intersect. The search range is determined by state uncertainty. The image locations are used directly as measurements for a state EKF and include measurement noise information. Should measurement noise for a landmark be particularly high, that landmark is flagged as unreliable; the system automatically re-initialises itself when too many landmarks become unreliable.

Stricker et al (1998) take a slightly different approach to square fiducial detection: fiducials are tracked by performing RAPiD-style edge searches about predicted positions. These measurements are combined directly using an M-estimator rather than first constructing edges and intersecting these to find corners of squares: it is claimed



Figure 2.1: ARToolkit markers are favoured for rapid AR application development. Samples shown here are distributed with the library.

that this extra step only reduces robustness in the face of outliers and partial occlusions. The system also rejects a full 3D motion model in favour of a simple 2D model which is faster to compute. As this system is used for AR applications, emphasis is placed on robustness and ease-of-use, and so the system can automatically be re-initialised when tracking fails. This is done by a blob-growing algorithm which locates all black squares in the image. Identification is again by colour bars along one side of the square.

While many passive fiducial-based tracking implementations for AR exist, none can match the ubiquity of *ARToolkit*. This popular library offers a reliable and easy-to-use fiducial tracking system geared towards AR applications. It is based on the work of **Kato & Billinghurst** (1999) and uses a heavy black square outline into which a unique pattern is printed; sample ARToolkit markers distributed with the library are illustrated in Figure 2.1. Fiducials are localised by image thresholding, line finding and extracting regions bounded by four straight lines; template matching yields the identity of such regions. Each marker, having four corners, yields a full 3D pose. Its popularity is in no small part due to its ease-of-use: Markers can quickly be produced on any printer and scanned in to the computer using a video camera and comprehensive software supplied with the toolkit. ARToolkit enabled the development of dozens of AR applications and is seen as the tool of choice when tracking is not the primary consideration in an AR prototype. The library is available for download as a fully-featured library¹ with an active development community. Work on improving the system is ongoing; for example, **Owen et al** (2002) replace the binary image inside the square rectangle with DCT basis functions, improving the system's resilience to noise and occlusion.

¹<http://artoolkit.sourceforge.net/>

Naimark & Foxlin (2002) present a circular fiducial which is not based on concentric ring colors but rather on fifteen binary cells arranged inside the circle. Algorithms for finding and identifying these fiducials are implemented in hardware to produce a self-tracking camera-sensor unit; since the fiducial extraction and identification cannot operate at full frame-rate on the hardware used, inertial sensors and prior information from previous frames are combined in a sensor fusion filter to reduce tracking workload. The resulting system has been developed into the Intersense VisTracker product¹.

2.2.2 Active Fiducial Tracking

Infra-red LEDs can output light across a very narrowly tuned wave-band, and if this band is matched at the sensor with a suitable filter, ambient lighting can be virtually eliminated. This means the only thing visible to the imaging sensor is the fiducials, and this vastly reduces the difficulty and computational requirements for tracking. For this reason, LEDs have long been used in commercially available tracking systems and real tracking applications; for example, LEDs mounted to a pilot's helmet can be used to track head pose in aircraft and simulators. Such applications are described in a survey by Ferrin (1991).

Tracking head-mounted LEDs with an external camera is an example of *outside-in tracking*, where the imaging sensor is mounted outside the space tracked. Outside-in tracking can be used to produce very accurate position results - especially when multiple cameras observe the tracked object - but they cannot generally provide measurements of orientation as accurately as *inside-out* systems, where the imaging sensor is itself head-mounted and any rotation of the user's head causes substantial changes in the observed image. Further, the range of an inside-out system is limited by the number of fiducials placed in the world rather than by the range of the world-mounted sensors; even for active fiducials such as LEDs, it is generally cheaper to install more fiducials than it is to install more cameras.

¹<http://www.intersense.com/products/>

Perhaps the best-known implementation of LED-based inside-out tracking is UNC's HiBall tracker which has its origins in the work of **Wang et al** (1990) and **Ward et al** (1992). Lateral-effect photodiode detectors are worn on the user's head: these detectors yield the 2D location of a single LED illuminating the imaging surface. Compared to standard video cameras, these detectors have the advantage of a higher resolution (1 part in 1000 angular accuracy) and much faster operating speeds ($660\ \mu\text{sec}$ to locate a single LED.) Large arrays of precisely-positioned ceiling-mounted LEDs can be controlled in software so that each sensor sees only one illuminated LED at any one time, solving the correspondence problem faced by all LED-based systems. **Welch & Bishop** (1997) replace the mathematical underpinnings of this system with the introduction of the SCAAT (single constraint at-a-time) framework, which is essentially a 12-DOF Extended Kalman Filter using under-constrained measurements: 2D measurements from each detected LED are inserted directly into the filter as they arrive, without first gathering enough measurements to produce a 6-DOF pose estimate. The system is repackaged and optimised as the commercialised HiBall tracker as described by **Welch et al** (1999); the 5-lens device weighs 300g and is capable of 2000Hz operation with sub-millimeter tracking accuracy.

A disadvantage of the HiBall system is that it does not operate as a camera; that is, it does not provide images which could be used for a video see-through AR system, and this is also true of any system which uses IR transmissive notch filters to isolate LEDs. An alternative is presented by **Matsushita et al** (2003) who combine a high-speed CMOS imaging chip with an FPGA to produce the ID CAM. The device is programmed to transmit alternating "scene" and "id" frames: each scene frame is a standard image of the scene as would be recorded by a monochrome 192×124 -pixel camera, whereas each id frame contains the image locations and identities of active LED markers. Markers transmit a strobing 8-bit id signal which is detected by the ID CAM as it samples the sensor at 2kHz to generate an id frame. The system is remarkably robust to any interfering ambient light and is intended to be embedded into hand-held devices such as mobile telephones.

2.2.3 Extendible Tracking

Purely fiducial-based visual tracking systems (particularly those using a single camera) are vulnerable to occlusions of fiducials, either because they may be covered by foreground objects e.g. the user's hand or because they may disappear from view when the user's head is turned. The use of natural features to extend the range and robustness of fiducial-based augmented reality systems has been proposed by **Park et al** (1998) and **Kanbara et al** (2001). In contrast to model-based tracking (and similarly to SLAM systems), these systems do not have prior information about the position of these natural features in space, but attempt to estimate them during operation.

Park et al (1998) use a single camera and recover feature positions with a structure-from-motion algorithm. Early during the system's operation, a sufficient number of fiducials is available to estimate camera pose: the fiducials are unique so that matching is trivial. The 2D positions of potentially trackable natural features is tracked, and used with a 3D filter to estimate the features' 3D positions (a "Recursive Average of Covariances" filter converges within circa 90 frames.) Individual features are tracked by an iterative optical flow approach which either converges on the feature or rejects it as un-trackable. Natural features may be either points or image regions. In the absence of fiducials, the positions of four natural features close to the four image corners are used to determine pose. The system presented was not capable of real-time operation.

Kanbara et al (2001) employ stereo vision instead of structure-from-motion. In the first obtained stereo frame pair, camera pose is reconstructed from the visible fiducials, which are found by colour matching. The frames are then split into a grid of small search windows, and each window is searched for a suitable natural feature with an 'interest operator.' Stereo information is used to recover the depth of these features.

During normal tracking, fiducials are detected by colour matching and natural features are detected by cross-correlation in the image with a template from the previous frame. In both cases, the search is limited to a small area about the predicted image position. A velocity model is not used; instead, three rate gyroscopes measure rotation

between frames. Once rotation has been estimated, the fiducial furthest away from the camera is found in the image. Its displacement from its rotation-predicted image position is then assumed to correspond to camera translation between frames. Hence camera rotation and translation are both predicted, allowing the use of a very local search for the remaining features. Tracked features are subjected to a confidence evaluation before their image positions are used to update camera pose by a least-squares method (weighted by feature confidence.)

2.2.4 Inertial Sensors for Robustness

The use of inertial sensors such as rate gyroscopes and accelerometers is wide-spread in virtual and augmented reality applications. Visual tracking systems perform best with low frequency motion and are prone to failure given rapid camera movements such as may occur with a head-mounted camera. Inertial sensors measure pose derivatives and are better suited for measuring high-frequency, rapid motion than slow movement where noise and bias drift outweigh. The complementary nature of visual and inertial sensors has led to the development of a number of hybrid tracking systems.

You et al (1999) combine rate gyroscopes with natural feature tracking. The system measures rotation only. Each frame, an initial rotation estimate is obtained from rate gyroscopes. Natural feature tracking as in **Park et al** (1998) is then used to correct this initial estimate. Natural feature tracking is replaced by fiducial tracking in **You & Neumann** (2001), which also adds linear accelerometers and introduces a state estimate in the form of an extended Kalman filter. Square fiducials with unique internal patterning are used: this patterning allows the tracking software to tell individual markers apart using principal component analysis. Marker positions are predicted for each frame and accurately determined using a coarse-to-fine detection procedure. The detected image positions are directly used as measurements for the EKF.

The same combination of inputs (6-DOF inertial data and fiducial-based vision) is also used by **Yokokohji et al** (2000): inertial sensors and an extended Kalman filter are used

not only to improve the robustness of a hardware-assisted landmark tracking system, but also to compensate for the end-to-end system delay by predicting future user head motion for the HMD display section. The authors go to great engineering lengths to minimise the system's delay so as to reduce dynamic registration error, specifying the use of a real-time operating system capable of running in time with camera refresh signals and treating video fields individually for both input and output video streams.

2.2.5 Combinations with Other Trackers

An alternative to visual and inertial tracking is commercially available in the form of *magnetic tracking*. Trackers such as the Ascension¹ Flock of BirdsTM can provide tracking accurate to few centimeters, which makes them suitable and popular for virtual reality applications. Augmented reality requires higher positioning accuracy for good registration. Thus, hybrid systems combining magnetic tracking with computer vision techniques have emerged. The major advantage of magnetic tracking when compared to inertial systems is that magnetic trackers give a direct measurement of pose: thus, even if a visual system should fail to track anything, the magnetic system can be used without error-prone integration of noisy measurements of derivatives and hence the systems do not fail catastrophically.

Bajura & Neumann (1995) present a video see-through system which relies mostly on magnetic tracking for its pose estimate. A single camera and LED fiducials are used to reduce registration errors. The LEDs are sufficiently bright that they can be detected by merely segmenting the image according to pixel intensity. Matching is done to the nearest predicted feature location. However, visual 3D pose reconstruction from detected fiducials is rejected on the grounds that it is too sensitive to noise and occlusions. Instead, virtual objects are temporarily displaced to line them up with the detected fiducials, with the assumption of correct camera pose and camera parameters. This displacement keeps the distance between virtual objects and camera constant to avoid apparent size changes. The system uses a radial lens distortion model to minimise static registration errors - this is done by un-distorting the video

¹<http://www.ascension-tech.com>

feed rather than by distorting virtual features. Further, the video see-through feed can be delayed until the augmented features are rendered, which achieves good dynamic registration. However, this causes the perceived view to lag behind the user's head motion.

State et al (1996) describe a fiducial/magnetic hybrid video see-through AR system. The visual system makes use of up to twelve unique two-colour concentric circles as fiducials, tracked with stereo cameras. Fiducials are detected by an exhaustive pixel search, performed on a small search window to speed up computation: firstly, the image is segmented to detect the colours present in the fiducial to be found. Next the image is searched for the outer ring of the fiducial and once this is found, detection of the inner circle is attempted. Once both these colour regions have been found, their center of mass is calculated.

Search windows are kept small by two methods: firstly, magnetic tracking is used to predict the new camera pose. Next, a single fiducial (which has previously been selected for ease of tracking) is found in the image, and camera pose refined by rotation about the camera center according to this fiducial's image position. The method for processing the remaining fiducials then depends on the number of them found: Should fewer than three be detected, the system is deemed under-determined and a rule-based system adjusts pose by one of six heuristic methods depending on which camera detects which fiducial. Should four or more be found, a least-squares solution is calculated. The case of three detected fiducials yields two possible solution which are compared to the magnetic tracker's prediction to choose between them. Once a pose update has been found, an internal estimate of the magnetic tracker's error is updated. This error estimate aids pose prediction in the next frame.

The resulting system is capable of sub-pixel static registration errors. However, dynamic errors are large due to the lack of synchronisation between the visual and magnetic sensors. The lack of a motion model means that state cannot be predicted. Further, the system is sensitive to lighting conditions due to the dependence on colour information in the fiducial detection.

Auer et al (1999) also combine magnetic and stereo visual tracking. Magnetic information is used to predict pose for a fiducial tracker based on black wall-mounted rectangles. Fiducial tracking is done in two stages: in the first, corners are roughly located as the intersection of image intensity edges. Subsequently, the corner positions are determined to sub-pixel accuracy by matching with a 5×5 pixel corner template. Many features (10-16) are tracked in the scene, and a form of RANSAC is used to eliminate outliers, after which a least-squares method refines pose. The inclusion of the visual system was found to improve the static positional accuracy over a purely magnetic alternative by a factor of three for translation and by a factor of ten for rotation.

Not all hybrid systems use magnetic information; the combination of two visual tracking systems, one inside-out and the other outside-in, has recently been studied by **Satoh et al** (2003). The inside-out tracker employs fiducials placed in the scene and is combined with outside-in tracking of one or more point-like markers attached to the user's head. For the case of only one head marker, measurements are combined by constraining the pose output from the inside-out tracking to lie on the line from the outside camera through the head marker. If more than one head marker is used the sum squared re-projection error of all markers in all cameras can be minimised. In either case, registration accuracy is improved beyond that provided by inside-out tracking by itself.

2.3 Augmented Reality Displays

2.3.1 Optical and Video See-through Displays

Augmented reality displays may be fundamentally split into two categories: *optical see-through* displays with which the user views the real world directly, and *video see-through* displays with which the user observes the real world in a video image of some sort. Each category of device has both advantages and disadvantages; these are most easily illustrated in the context of *Head-mounted Displays (HMDs)*.

An optical see-through HMD typically places a semi-silvered mirror before the user's eyes. The user can see the real world through the mirror, but can also see computer graphics drawn on miniature screens visible in the mirror's reflection: graphics are superimposed on the real world by *additive mixing*. This has the effect that graphical regions drawn as black appear transparent to the user, offering unmodified view of real objects in the same place.

A video see-through HMD does not let the user see the environment directly - the display mirror is fully reflective, and the user sees only what is drawn on the miniature screens. A view of the real world is typically provided by a video stream from display- or head-mounted cameras. Augmented graphics can be rendered directly into this video feed by any mixing required, so that e.g. black virtual objects can appear in the scene.

The primary advantage optical see-through HMDs have over video-based models is that they offer (in theory) a superior view of the real world: apart from an attenuation (and possible loss in contrast) the user's view of the real world is unmodified so objects can be observed in full resolution and without any time delays. By contrast, the view through a video-based display is of lower resolution and dynamic range, may have regions out of focus, and necessarily lags behind the real world somewhat.

The primary *disadvantage* of the optical displays is the inferior integration of virtual graphics into the real world. Since light from the real world strikes the user's eye directly, the computer cannot see what the user sees, but must instead guess where to draw graphics based on other position sensors and some display *calibration*: it is up to the user to tell the computer how to line up real and virtual images. By contrast, a computer has full access to the user's view in the case of video see-through displays. This means the computer can potentially draw annotations and graphics in *exactly* the right place, without user intervention. Further, video see-through displays are capable of drawing virtual graphics at the right time, since the real world and corresponding graphics appear to the user at exactly the same time; on optical see-through displays, the view of the real world is instantaneous, while the view of virtual graphics is neces-

sarily delayed. This can produce the unpleasant effect of graphics ‘swimming around’ the real world when the user’s head moves.

Apart from registration accuracy, the ability to insert objects by arbitrary mixing allows video see-through displays to produce richer composited scenes and allows for *diminished reality*: the removal of real components from the scene. By contrast, the only way to remove a real object in an additive-mixing display is to cover it with colors much brighter than the object to be obscured - this is obviously only possible up to a certain brightness of object to be masked.

2.3.2 Advances in HMDs

From the first twin-CRT-based display of **Sutherland** (1968), head-mounted displays have made great advances in resolution and compactness; however many issues remain outstanding, particularly in the field of AR. Some recent attempts have been made to address some of the shortcomings of the display format.

Kano et al (2004) present the reflex HMD, an optically see-through device which attempts to compensate for the time delay between sensing, rendering and display. Instead of rendering just the user’s expected field-of-view, a larger image is rendered which includes elements outside of the field-of-view. When the image is transmitted to the display, the LCD panel sync signals are modified so that only a portion of this large image is displayed: the signals are modified according to the integrated signal from rate gyroscopes so as to compensate for any head rotation which occurred between the original pose estimation and completion of associated rendering. This technique does incur distortions when a pinhole projection model is used, but dynamic registration errors near the center of the display are measurably reduced (**Kijima & Ojika**, 2002) and users are able to perform a “lock on” task faster than without this compensation.

Kiyokawa et al (2003) present the fourth-generation ELMO (Enhanced see-through display using an LCD panel for Mutual Occlusion). This optically see-through device

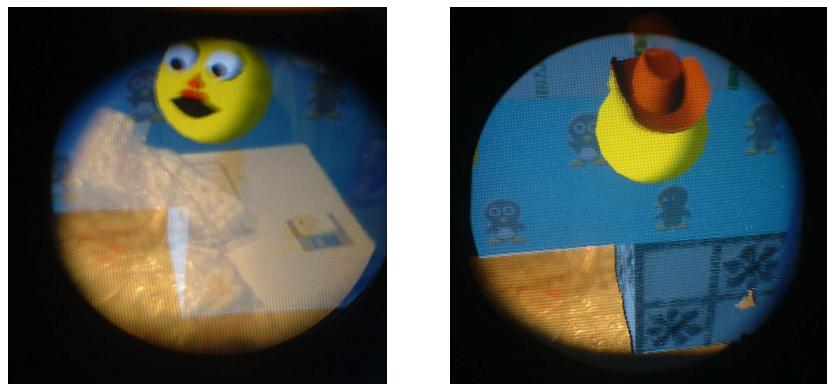


Figure 2.2: An occlusion-capable optical see-through HMD. An LCD panel in the focal plane of the ELMO display allows real objects to be hidden from view. Traditional optical see-through displays (left) produce transparent and insubstantial-appearing overlays while ELMO (right) gives objects a solid appearance. *Images courtesy of Kiyoshi Kiyokawa of Osaka University.*

contains an emitting micro-display like most other HMDs, but also inserts an LCD panel into the path of incoming light, allowing portions of the incoming light to be attenuated. This allows for much more flexible mixing of incoming and virtual light, allowing real-world objects to be properly occluded by virtual graphics and dark objects to be drawn. The view through the prototype device (with and without the occluding panel enabled) is shown in Figure 2.2. The disadvantage of this approach is the extra optics required to generate a new focal plane for the LCD; the twin ring-shaped optical paths increase the mass of the device (which includes back-of-head counterweights) to 2 kg. An effort to produce a more compact device achieving the same effect is being pursued by Cakmakci et al (2004), however this system is still in early stages of development.

2.3.3 HMD Calibration

The calibration of head-mounted displays for AR has also been the focus of substantial research. When video see-through displays are used, this calibration is not too difficult, since the computer can make a large number of measurements directly from the incoming video stream, and standard computer vision camera calibration techniques

can be employed - such as the method of **Tsai** (1987), who introduces a rich camera model including radial distortion and proposes a two-stage calibration technique consisting of an initial linear solution followed by a nonlinear optimisation.

When optical see-through displays are used, measurements can no longer be made automatically but must be supplied by the user. Further, calibration is no longer limited to camera-intrinsic parameters, but extrinsic parameters (the sensor-eye offset) must also be estimated. Ergonomics limit the number and accuracy of measurements a user can be expected to make, and hence the camera models employed are often simplified to reduce the number of degrees-of-freedom requiring calibration.

Approaches have differed in the method by which users are to make measurements: **Janin et al** (1993) compare the direct measurement of users' eye positions to an optimisation approach in which crosshairs are aligned with a real-world calibration target. Alignment can be performed both by aligning the crosshairs on-screen and by user head-motion. The optimisation minimises the re-projection error of all such measurements to calibrate a simple pin-hole projection model. The authors have found the optimisation method to produce results far more accurate than direct measurement.

Azuma & Bishop (1994) use a multi-step calibration procedure to determine intrinsic and extrinsic parameters. Some steps involve the positioning of software cursors in the display to determine intrinsic parameters, while other involve aligning the user's eye along a bore-sight to recover extrinsic parameters. An obvious disadvantage of this method is that the bore-sight-equipped calibration target must be present in the scene. The intrinsic parameter calibration used is relatively simple; no distortion is considered, and the aspect ratio of the display is considered known.

Tuceryan & Navab (2000) do not separate the calibration into separate intrinsic and extrinsic parameters and instead calibrate a 3×4 , 11-DOF projection matrix in a single linear step. Measurements are gathered using a single 3D point with known coordinates. The user sees a cross-hair in the display and aligns this with the 3D point by moving his or her head. Twelve measurements are made with the cross-hair in vari-

ous positions. The authors call this method SPAAM, for single point active alignment method.

Genc et al (2000) extend this system from the monocular to the stereo case; instead of having the user perform one individual calibration for each eye, both eyes are calibrated simultaneously. This is done by replacing the monocular cross-hair with a solid disk simultaneously visible in both eyes, with a horizontal offset between the disk positions for each eye; this gives the user the perception that the disk is located at a certain depth in the image. Again, a single 3D point is used as a target for alignment, but this is now performed in 3D: the user's head must be positioned at the correct depth so that the disks in both eyes line up with the target object. While this method decreases calibration time for the user, there is the danger that it will produce erroneous results if displays on left and right eye are vertically misaligned or slightly rotated.

The approaches described so far operate by aligning the user's head relative to fixed objects in the environment. **Fuhrmann et al** (1999b) take a different approach, and alignment of on-screen cross-hairs is against a hand-held tracked target (this hand-held device forms part of the AR application's user interface.) Eight points are sampled per eye: these are essentially the four corners of the viewing frustum, sampled at two different depths. This special geometric arrangement is used to produce simple equations for individual calibration parameters. Per-user calibration overhead can be reduced to four sampled points per eye once some parameters of the display are known. This work is followed up (**Fuhrmann et al**, 2000) with a mention of the potential for the calibration of a display's radial distortion; the authors proposes either dense sampling of a grid or the capture of an image through the display with a camera, but no results for optical see-through displays are presented. **Tang et al** (2003) have performed an evaluation of the accuracy of SPAAM versus Fuhrmann et al's stylus-based approach and find that the hand-held stylus method produces superior results.

Owen et al (2004) carry on to produce perhaps the most ambitious HMD calibration to date. The display is mounted on a mobile rig in front of calibration pattern, and

a 5-megapixel digital camera, also mounted on a mobile rig, is used to measure the shape of the display's virtual imaging plane - which the authors determine is not a flat plane as implied by a pinhole projection model, but rather a highly curved surface. Once the rich intrinsic description of the display has thus been acquired, projection parameters for any user can be calculated from knowledge of the user's eye offset from the calibrated display center; this offset is measurable using few user inputs, or indeed by physically measuring the user's inter-pupillary distance. Work on this method is ongoing, as is the incorporation of knowledge of the display's curved projection surface into a compensating rendering technique.

2.3.4 Hand-held AR

Head-mounted displays offer the enticing prospect of fully immersive AR experiences, but can in practice suffer from registration and usability difficulties. An alternative to HMDs is the use of hand-held display devices for AR. A device with a screen and on-board camera operates in video see-through mode and acts much like the viewfinder of a video camera, offering the user an unintrusive interface to augmented information without the bulk and cables typically involved without a head-mounted system.

Perhaps the first example of hand-held AR is presented by **Rekimoto** (1995). The author lists problems with previous HMD based systems and proposes the use of a hand-held palmtop display device, calling this the *magnifying glass approach*: "While a real magnifying glass optically enlarges the real world, a system based on this approach enlarges it with information". The NaviCam consists of a hand-held LCD-TV screen to which a small camera has been mounted, tethered to a computer and a video compositing engine. Bar-codes mounted on real objects in the world (such as a calendar) are recognised and pertinent information (such as the day's schedule) is overlaid when such bar-codes come into view.

The author conducts a user study in which users are required to "visit" three bar-codes using various AR display devices. The NaviCam allows the users to complete

the task in less than half the time required using a traditional HMD; further users find the hand-held device superior in almost all ergonomic aspects tested. Only the lack of hands-free operation and the weight of the device (430g) are considered negative aspects.

Since the work of Rekimoto, the computing power of hand-held devices has increased to the point that a tether to an external computer is no longer required for some AR applications. Three categories of device appear particularly promising as hand-held AR interfaces: Tablet PCs, PDAs and mobile telephones.

Of these classes, tablet PCs are the most powerful, but also the heaviest; furthermore the tablet PC is a relatively new class of computer and is not yet well-established in the market. Consequently, few tablet-based AR systems have been presented in the literature.

Vlahakis et al (2002) use a tablet PC as part of the larger ARCHEOGUIDE project. ARCHEOGUIDE aims to enhance the experience of guests visiting archaeological sites. For example, reconstructions of historic buildings are rendered in their original (now ruined or empty) locations. The project investigates multiple types of AR apparatus: HMDs are used for video see-through augmentations, while tablet and pocket PCs are used as replacements for paper guides. The tablets are equipped with differential GPS and a compass, and can replay multimedia streams chosen according to the user's location.

Zhu et al (2004) present the PromoPad, an augmented reality shopping assistant using video see-through augmentations on a tablet PC. Registration is marker-based and based on the Owen's earlier extension of the ARToolkit system (**Owen et al**, 2002). Emphasis is placed on the PromoPad's ability to detect context from the user's location and update the presented information accordingly: For example, products known to be of interest to a customer can be highlighted in the display, while less desirable products can be hidden (diminished reality.)

Personal Digital Assistants (PDAs) do not yet have the processing power of tablet PCs

- in particular, floating-point performance, 3D graphics and image-capture bandwidth are lacking - however they offer the attraction of a much smaller form-factor than tablet PCs. The devices now offer respectable integer processing power and built-in wireless connectivity, and both of these aspects have been exploited for AR.

Wireless networking can be used to work around limited processing power by transmitting images to a nearby workstation and treating the PDA as a "thin client". This approach is widely taken, for example by both **Geiger et al** (2000) and **Gausemeier et al** (2003) in the context of the AR-PDA project. In one prototype application, a user is instructed to insert a memory card into a digital camera by a virtual miniature man; in another, extra components for an electric oven are superimposed on the view visible on the PDA. In both of these approaches tracking and rendering is performed fully on the PC; images captured on the hand-held device are transmitted to a remote host, tracked and augmented, and the finished images are broadcast back to the PDA. **Pasman & Woodward** (2003) let the PDA perform some of the work: the camera image is thresholded before transmission to the server, the server sends back only the augmented graphics, and compositing is performed on the PDA. This approach reduces bandwidth requirements between the client and server. Operation over GSM is attempted but is very slow (5 seconds per frame.) **Newman et al** (2001) also operate the PDA as a thin client but it is tracked externally using the ultrasonic Bat tracker. Graphics are not overlaid onto a camera image, rather the user is presented with a VR-style view from the device's viewpoint.

Other approaches have attempted to run applications directly on the PDA. This has involved the development of a number of enabling technologies, notably the creation of fast integer-based OpenGL-like graphics libraries (e.g. KLIMT¹), and a port of the AR-Toolkit library to the pocket PC platform (**Wagner & Schmalstieg**, 2003a). Although pure pose estimation performance of up to 15 frames/sec is possible on an XScale CPU, current cameras cannot supply more than 8 frames/sec to the CPU; nevertheless, fully self-tracking PDA applications using this framework have been demonstrated by **Wagner & Schmalstieg** (2003b).

¹<http://studierstube.org/klimt/>

A further use of the PDA's wireless connectivity is the creation of multi-user AR applications; **MacWilliams et al** (2003) demonstrate that PDAs can usefully coexist with many other display modalities in a distributed AR framework but have encountered some rendering performance issues using a cross-compiled VRML renderer. **Wagner et al** (2005) present the widely-demonstrated "Invisible Train" multi-user application; the authors claim the ease-of-use and public acceptance of the form-factor exceed that previously encountered with HMDs.

Finally, the emergence of user-programmable mobile phones featuring built-in cameras has seen first attempts at harnessing this soon-to-be ubiquitous platform for AR applications: **Möhring et al** (2004) present a simple technology demonstration of a self-tracking camera phone, and more phone-based applications can be expected to follow.

2.3.5 Other AR Displays

Beyond head-mounted and hand-held AR a few other display devices have been employed for AR. These include video projectors, which combine some of the advantages of optical and video see-through systems: the computer can observe the registration of real and virtual objects, but only after a small delay. For example, **Karitsuka & Sato** (2003) use a backpack-mounted projector as a replacement for an optically see-through HMD.

A more specialist display is the *virtual showcase* (**Bimber et al**, 2001), a device based on a large semi-silvered mirror; this class of display mixes the user's view of a real object in a showcase with images generated on a computer monitor or projection surface. Magnetic head-tracking and shutter glasses worn by the user allow the graphics to be accurately positioned in relation to the real objects. The difficulties encountered by optical see-through mixing - i.e. virtual objects appearing transparent - can be addressed by illuminating real objects with structured illumination, darkening those regions which will appear behind virtual objects (**Bimber & Fröhlich**, 2002).

2.4 Occlusions in AR

When real and virtual objects coexist in a scene, virtual objects will sometimes be hidden behind real objects, and real objects may be obscured by virtual ones. In the context of video see-through AR, anything rendered by the computer will by default appear in front of the video image; occlusions of real objects by virtual ones are therefore not a problem. However, for real objects to hide virtual ones, the virtual ones must not be drawn. This can be achieved in a variety of ways. Early approaches to handling the occlusion of real and virtual objects include those of **Wloka & Anderson** (1995) and **Breen et al** (1996). Both investigate the use of stereo cameras to estimate a depth map of the real objects in the user's view, and use this depth map to handle the occlusion of augmented visuals.

Wloka & Anderson (1995) motivate this approach by pointing out that in most AR applications, the assumption of an unchanging scene is unrealistic because of (for example) the user's interaction. A speed-optimised depth-from-stereo algorithm is developed; this requires two cameras aligned such that epipolar lines are parallel to scan-lines. Sub-sampled images are vertically raster-scanned to detect intensity discontinuities: this is done to group pixels into vertical blocks of three or more pixel's length, where the pixels any block are of a similar intensity. Blocks in the left and right image are then matched according to length, intensity, and vertical position. The horizontal offset between the matched blocks then yields the z-position of this block, which is entered into the z-buffer¹ in both frames. Some pixels may be matched more than once and some not at all; such pixels use weighted contributions of the multiple measurements or their pixel neighbours. The authors go to great lengths to implement this algorithm (already designed for speed rather than accuracy) efficiently by imposing coherency constraints and parallelizing the algorithm for the dual-processor hardware available, and the algorithm executes in 370ms per quarter-size frame pair; strong artifacts are however visible in the composited images.

¹A *z-buffer* is a depth buffer in OpenGL. When a 3D primitive is rendered, each pixel's depth is first compared to the values in the z-buffer; if the pixel to be rendered is further away from the camera than the value in the z-buffer, the pixel is discarded; otherwise it is rendered and the z-buffer updated.

By contrast, **Breen et al** (1996) study the case of a static scene further. A polygonal depth map is not generated once-per-frame but only once for a given camera arrangement; the map is invalid as soon as the camera moves. The authors argue that for simple scenes for which 3D models can be generated, better results can be achieved by registering these in a calibrated 3D environment, and then rendering these 3D models as black into the scene (black corresponds to transparent using on the authors' video compositing system, and so the video is seen.) Using registered models, respectable interactive frame-rates are achieved.

More recent work on the application of depth-from-stereo to AR occlusion has been presented by **Kanbara et al** (2000). This approach differs from previous depth-from-stereo methods by calculating stereo depth only in the bounding boxes of the virtual objects to be rendered: as long as these occupy only a small portion of the screen, this leads to a significant increase in performance, and so the depth-from-stereo algorithm requires only 40ms per typical frame pair to execute. Kanbara uses blue markers in the image for tracking, and Sobel edges are extracted from the bounding box regions of both images in the stereo pair. Pixels on edges are matched between pairs using the sum-of-absolute-distances in 5×5 -pixel image patches, yielding edge depths; pixels between edges are assigned depths by interpolating between edge depths, and some consistency metrics are applied to reject outliers. Depth values are then written into the z-buffer and so subsequently rendered virtual graphics are clipped.

Kiyokawa et al (2003) employ a commercial depth-from-five-cameras system to estimate an occlusion map in a game application used to demonstrate their occlusion-capable ELMO HMD (cf. Section 2.3.2). This system yields a 280×240 pixel depth map at 30Hz. It requires five synchronised cameras, but the very compact cameras used on their display do not offer such a facility; this results in occasional errors in the recovered depth map, as illustrated in Figure 2.3.

Monocular approaches to occlusion handling also exist. **Berger** (1997) does not explicitly estimate depth and does not use 3D models of real occluding geometry. Rather, an estimate of occlusion regions is built by tracking 2D contours found in the image. By observing the motion of these contours over time, they can be labeled as being "in

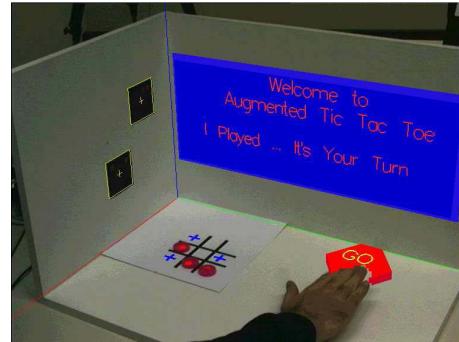
front of" or "behind" individual elements of virtual imagery. To generate clipping masks from groups of occluding contours (which may not form a closed shape), active snakes are employed. The set of edges is rendered and then blurred to produce an intensity field; a snake is initialised at the bounding box of the field, and allowed to shrink to fit the maxima (edges) of the intensity field. The resulting shape is used to clip the virtual graphics, and impressive results are presented for scenes without complex textures, as demonstrated in Figure 2.3. The computational performance of the algorithm is not discussed. **Lepetit & Berger** (2000) later extend this idea to a high-accuracy off-line scenario; by tracking user-seeded occluding curves through a video sequence, a 3D reconstruction of the occluding object is computed. The resulting accurate segmentation of the sequence into foreground and background allows virtual objects to be inserted into the video with high precision.

Recent real-time work on occlusion has focused on real objects (in particular, the user's hands) dynamically occluding virtual objects. **Stricker et al** (1998) present an interactive tic-tac-toe application. The player places a real playing piece on a real tic-tac-toe board and then pushes a virtual button, upon which the computer places its virtual piece onto the board. The computer's playing pieces (and the button) should not be drawn when the user's hand covers them. Occlusion is handled by detecting the user's hand using background subtraction; the area detected as not being part of the background is used to mask the z-buffer prior to rendering. This approach is made possible by a homogeneous background (white paper plus playing grid) and the assumption of a static camera. A resulting image is shown in Figure 2.3.

Fuhrmann et al (1999a) concentrate on the occlusion of virtual graphics by humans, use marker-based human motion capture to register occluding users in a collaborative scene. A 3D humanoid model aligned to the motion-capture pose is rendered into the z-buffer to occlude scientific visualisations in a collaborative application. For regions such as hands which can change shape in ways not tracked by the system, the "phantom" is blurred by rendering concentric shells of the object at various transparencies which represent the probability density of that portion of hand-space being occupied. Results of this procedure are illustrated in Figure 2.3.



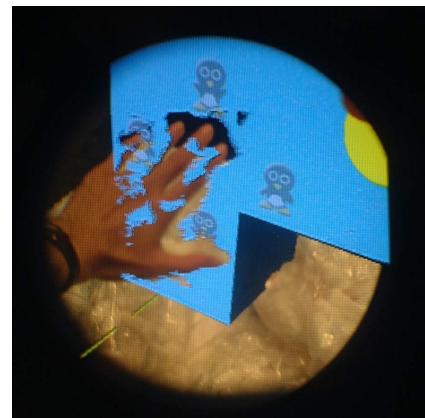
Berger (1997)
Courtesy of Marie-Odile Berger



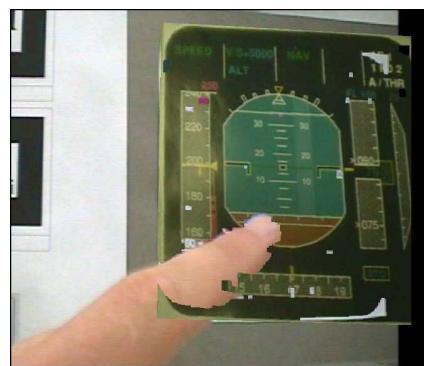
Stricker et al (1998)
Courtesy of Didier Stricker /
Fraunhofer IGD



Fuhrmann et al (1999a)
Courtesy of Anton Fuhrmann



Kiyokawa et al (2003)
Courtesy of Kiyoshi Kiyokawa of
Osaka University



Fischer et al (2003)
Courtesy of Jan Fischer

Figure 2.3: Occlusion handling in AR. This figure illustrates results from previous attempts to handle AR occlusions.

Fischer et al (2003) remove the limitations of Stricker's background-subtraction approach: instead of a simple unmoving background image, a 3D textured model of the scene is used. At each frame this is re-rendered using a camera pose obtained from ARToolkit tracking. This rendered version of the expected scene is then compared with the video feed to determine occluder positions. Since this comparison is done on a per-pixel basis, the alignment between rendered scene and viewed scene must be very accurate: for this purpose the scene is split into individual textured surfaces, and the position of each surface is refined by aligning feature points prior to rendering. Pixel comparison operates by a criterion in HSV-space designed to minimise the influence of lighting changes. The algorithm's complexity limits its run-time speed to circa 1 frame/second, but the authors suggest potential speed improvements; further, the color-based background subtraction produces occasional errors, as illustrated in Figure 2.3.

Mulder (2005) also employs background subtraction to determine occluder positions, but this is done in layered 3D, in the context of an enclosed manipulation space akin to a virtual showcase. Multiple static cameras observe a user's hands in front of a blue cardboard background. The resulting images are binarised into foreground or background. When rendering, these binary masks are then used to determine occlusion: the scene is first rendered into graphics hardware; subsequently, occlusion is determined at discrete depth intervals by considering several planes perpendicular to the user's viewing direction. For each plane, all cameras' binary masks are projected into the plane; where the occluding mask portions overlap, a real occluding object at that depth is assumed to exist, and virtual pixels behind this layer are overwritten. The algorithm uses a combination of stencil- and z-buffering to accomplish the procedure in graphics hardware and to further determine the position of real objects which should be occluded by virtual objects (since the display used is optically see-through, such occlusions of real objects by virtual ones must also be handled specially.)

2.5 Motion Blur

Motion blur occurs due to the non-instantaneous exposure time of video cameras. When an object is moving or a camera rotating, projections move across the imaging sensor during the time taken for exposure, blurring images in the direction of motion. Large motion blur which spans 10 pixels or more in the image can severely compromise the performance of many tracking systems: sharp points (e.g. LEDs) are turned into lines; sharp edges are turned into ramps; characteristic texture patches are turned into a blurred homogeneous regions. This makes the extraction and matching of features from blurred video frames difficult.

Motion blur is generally seen as a nuisance by the tracking community. A common strategy is to avoid blur altogether, by moving cameras slowly or by changing camera settings: for example, the popular unibrain Fire-i camera used in many vision systems allows the electronic selection of a short exposure time. In the context of augmented reality, neither approach is universally applicable: cameras are often head- or hand-mounted, making it hard to constrain the speed of motion, and cameras are often selected for their physical compactness or transmitting features rather than a rich electronic control set, and so motion blur remains in the image to be dealt with. In recent years, some real-time tracking systems which attempt to handle motion blur in software have emerged.

Claus & Fitzgibbon (2004) describe a machine-learning approach to creating a fiducial detector capable of coping with real-world tracking situations, which provide difficulties such as motion blur, clutter, and large variations in lighting and scale. Fiducials consist of four black dots arranged in a rectangle; machine learning is used to train a classifier which can label each pixel in the image as “black dot” or “not black dot”, and further processing rejects those matches which are not part of a rectangular fiducial. The circle detector is trained from a set of 12×12 -pixel input images which include dots under a range of lighting conditions and blur. This makes the detector somewhat robust to moderate amounts of motion blur encountered.

Gordon & Lowe (2004) obtain some resilience to motion blur by tracking features obtained by the scale-invariant feature transform. The SIFT transform (**Lowe**, 2004) extracts features of many scales from the image, and hence makes use of large-scale features which are less affected by moderate amounts of blur. Further, the system which can perform a localisation step at each frame and may so be considered robust to motion blur in that even if tracking fails during transient motions, tracking can be resumed once the camera rotation slows down. Many fiducial-based systems also exhibit this quality and so these systems are wide-spread in AR. The absence of pose estimates during rapid rotations may be acceptable for many applications.

The work described above, and indeed the edge-based tracking system described in Chapter 4, are examples of techniques used to overcome the *problems* presented by motion blur. A few attempts have also been made to exploit the *opportunities* presented by motion blur:

Lin (2005) analyses images of moving vehicles taken with a roadside camera. Motion blur is used to estimate the speed of vehicle motion; this estimate is then used to de-blur the image, allowing blurred registration plates to be read. The method uses a Fourier transform to detect the image orientation of blur, and blur magnitude is estimated by analysing intensity ramps in scan-lines where the vehicle borders a uniform background (a blue sky).

Rekleitis (1996) uses motion blur to estimate optical flow in an image. Steerable filters applied to the Fourier transform of image patches are used to determine the orientation of local blur. Once orientation has been determined, the 2D spectrum is collapsed to obtain the patch's spectrum along the direction of blur; blur length is extracted using cepstral analysis. Run-time performance is limited by the cost of 128×128 -pixel FFTs.

Favaro et al (2004) exploit both motion blur and distance-varying defocus present in images to reconstruct a scene's depth map, radiance and motion. The use of motion blur is an extension to the authors' previous work in the domain of shape from defocus. Blur is modeled as a diffusion process whose parameters are estimated by

minimising the discrepancy between input images to the output of the hypothesised diffusion process. This approach attempts to determine the maximum of information from every pixel of two or more input images; as such it differs greatly from the approach taken in Chapter 7, where a small set of parameters (camera rotation) is estimated from a single image in the shortest possible amount of time.

3

Mathematical Framework

This chapter introduces the mathematical framework employed throughout the remainder of this thesis.

3.1 Coordinate frames

Points in 3D space are represented as homogeneous coordinates of the form $(x \ y \ z \ 1)^T$. These coordinates are measured in a particular coordinate frame which may be denoted by a subscript. For example the coordinates of a point in the “World” coordinate frame may be denoted $(x_W \ y_W \ z_W \ 1)^T$.

The same point usually has different coordinates in different coordinate frames. Coordinates are transformed from frame \mathcal{A} to frame \mathcal{B} by left-multiplication with a 4×4 Euclidean transformation matrix denoted $E_{\mathcal{BA}}$, where the subscript \mathcal{BA} may be read

as “ \mathcal{B} from \mathcal{A} ”:

$$\begin{pmatrix} x_{\mathcal{B}} \\ y_{\mathcal{B}} \\ z_{\mathcal{B}} \\ 1 \end{pmatrix} = E_{\mathcal{B}\mathcal{A}} \begin{pmatrix} x_{\mathcal{A}} \\ y_{\mathcal{A}} \\ z_{\mathcal{A}} \\ 1 \end{pmatrix}. \quad (3.1)$$

Transformation matrices may be chained together by multiplication: The product $E_{\mathcal{C}\mathcal{A}} = E_{\mathcal{C}\mathcal{B}}E_{\mathcal{B}\mathcal{A}}$ transforms points from coordinate frame \mathcal{A} to coordinate frame \mathcal{C} . The transformations further have an inverse, e.g. $E_{\mathcal{A}\mathcal{B}}^{-1} = E_{\mathcal{B}\mathcal{A}}$.

Transformation matrices take the form

$$E = \begin{bmatrix} R & \mathbf{t} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.2)$$

where R is a 3D rotation matrix ($|R| = 1, R^T R = I$) and \mathbf{t} is a translation vector. The set of all possible E forms a representation of the 6-dimensional Lie Group SE(3), the group of rigid body transformations in \mathbb{R}^3 .

3.2 Motions

With time, the transformations between coordinate frames may change. Such a change is represented with a 4×4 motion matrix denoted M :

$$E_{\mathcal{B}\mathcal{A}|t+1} = M_{\mathcal{B}} E_{\mathcal{B}\mathcal{A}|t} \quad (3.3)$$

where $M_{\mathcal{B}}$ represents motion in coordinate frame \mathcal{B} . Motion matrices, also being rigid body transformations in 3D space, take the same form as E in Eq. (3.2).

Motion matrices M have six degrees of freedom and may be minimally parametrised as six-dimensional motion vectors μ , where μ_1, μ_2 and μ_3 represent translation along the x, y and z axes and μ_4, μ_5 and μ_6 describe rotation around these axes. Motion vectors, like motion matrices, are usually written with a subscript denoting their reference

frame (however this is sometimes dropped to aid readability.) For a given motion vector μ_B in frame B the corresponding motion matrix is given by the exponential map:

$$M_B = \exp(\mu_B) \equiv e^{\sum_{j=1}^6 \mu_{Bj} G_j} \quad (3.4)$$

where G_j are the group generator matrices. The generator matrices take the values:

$$G_1 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad G_2 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad G_3 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad (3.5)$$

$$G_4 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad G_5 = \begin{bmatrix} 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad G_6 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Differentiating 3.4 about the origin ($\mu = 0$), the partial derivatives of motion matrices with respect to the motion parameters are simply the corresponding generators:

$$\frac{\partial M}{\partial \mu_j} = G_j. \quad (3.6)$$

Further, complex coordinate frame transformations are easily differentiable by application of the chain rule:

$$\frac{\partial}{\partial \mu_{Bj}} (E_{CB} M_B E_{BA}) = E_{CB} G_j E_{BA}. \quad (3.7)$$

Closed forms of both the exponential and the logarithm ($\mu = \ln M$) exist. Further information on the Lie Group SE(3) and its properties may be found in **Varadarajan** (1974) and **Tomlin & Sastry** (1995).

It is sometimes useful to express motions in one coordinate frame in a different coordinate frame. For example, a certain motion M_B in coordinate frame B can be transformed to frame A in matrix form

$$M_A = E_{AB} M_B E_{BA}. \quad (3.8)$$

Alternatively motions can be transformed as vectors using the *adjoint operator*. The adjoint of a transformation matrix yields a 6×6 matrix such that

$$\mu_A = \text{Adj}(E_{AB}) \mu_B \quad (3.9)$$

and takes the value (writing the cross operator \wedge)

$$\text{Adj}(E) = \begin{bmatrix} R & t \wedge R \\ 0 & R \end{bmatrix}. \quad (3.10)$$

3.3 Uncertainty in Transformations

Often the true value of a transformation matrix is unknown and only a noisy estimate can be obtained. In the same way as motions are defined in a specific coordinate frame, errors are also relative to a frame. Considering the noisy estimate of the transformation from frame \mathcal{A} to frame \mathcal{B} and choosing (without loss of generality) to represent errors in frame \mathcal{B} , the relationship between estimate and true state (denoted by a hat $\hat{\cdot}$) is written

$$E_{\mathcal{B}\mathcal{A}} = \exp(\epsilon_{\mathcal{B}}) \hat{E}_{\mathcal{B}\mathcal{A}}. \quad (3.11)$$

The error 6-vector ϵ is modeled as normally distributed:

$$\epsilon_{\mathcal{B}} \sim N(\mathbf{0}, \Sigma_{\mathcal{B}}). \quad (3.12)$$

Here, $\Sigma_{\mathcal{B}}$ is the estimate's 6×6 covariance matrix in coordinate frame \mathcal{B} , and can be written in terms of expectation:

$$\Sigma_{\mathcal{B}} = E(\epsilon_{\mathcal{B}} \epsilon_{\mathcal{B}}^T). \quad (3.13)$$

If the errors in 3.11 were represented in frame \mathcal{A} instead of frame \mathcal{B} ,

$$E_{\mathcal{B}\mathcal{A}} = \hat{E}_{\mathcal{B}\mathcal{A}} \exp(\epsilon_{\mathcal{A}}), \quad (3.14)$$

then the distribution of $\epsilon_{\mathcal{A}}$ is different from that of $\epsilon_{\mathcal{B}}$. For example, an ambiguity in rotation in coordinate frame \mathcal{B} corresponds to a coupled translation and rotation ambiguity in coordinate frame \mathcal{A} . For this reason it is necessary to know how to transform covariance matrices from one coordinate frame to another.

Considering a single sample from the error distribution, this can be transformed from one coordinate frame to another just as motions in Eq. (3.9):

$$\epsilon_{\mathcal{A}} = \text{Adj}(E_{\mathcal{AB}}) \epsilon_{\mathcal{B}}. \quad (3.15)$$

Then by expectation,

$$\Sigma_{\mathcal{A}} = E(\epsilon_{\mathcal{A}} \epsilon_{\mathcal{A}}^T) \quad (3.16)$$

$$= E(\text{Adj}(E_{\mathcal{A}\mathcal{B}}) \epsilon_{\mathcal{B}} \epsilon_{\mathcal{B}}^T \text{Adj}(E_{\mathcal{A}\mathcal{B}})^T) \quad (3.17)$$

$$= \text{Adj}(E_{\mathcal{A}\mathcal{B}}) E(\epsilon_{\mathcal{B}} \epsilon_{\mathcal{B}}^T) \text{Adj}(E_{\mathcal{A}\mathcal{B}})^T \quad (3.18)$$

$$= \text{Adj}(E_{\mathcal{A}\mathcal{B}}) \Sigma_{\mathcal{B}} \text{Adj}(E_{\mathcal{A}\mathcal{B}})^T \quad (3.19)$$

and so error covariances also transform from one coordinate frame to the another by the adjoint.

3.4 Software

Software libraries which enable the rapid implementation of the above framework in the C++ programming language have been made available under a public license. These libraries have been written by members of the CUED Machine Intelligence Lab.

TooN - (Tom's Object Oriented Numerics) - is a templated vector/matrix library which provides an interface to BLAS/LAPACK linear algebra methods and contains classes encapsulating the functionality described in this chapter. It is available at
<http://savannah.nongnu.org/projects/toon>

libCVD - (Cambridge Vision Dynamics) - is a further library providing methods for video input/output, image manipulation and more. It is designed to facilitate the development of software such as that used in this thesis. The library may be obtained at <http://savannah.nongnu.org/projects/libcvd>

4

Markerless Visual Tracking

4.1 Introduction

This chapter describes a markerless visual tracking system which will be used as the primary registration sensor for the AR applications in this thesis. Such a system should deliver accurate pose estimates in real-time from live video feeds at up 50Hz and still leave computing resources for AR rendering; further, tracking needs to be robust to the rapid motions common in AR, for example the rotations of a user's head.

A markerless tracking system must track natural features which are already available in the scene, and so the system must find these features in the video feed. The choice of features to be used is a function of the amount of processing time available per frame: in some scenarios it is feasible to perform complex full-frame feature extraction to obtain richly described point features which can be matched from frame to frame

(using, for example, **Lowe** (2004)'s scale-invariant feature transform (SIFT)). However, such operations are mostly still too computationally expensive for real-time AR use.¹

Since fast operation is a primary requirement, the tracking system used in this thesis is based on the work of **Drummond & Cipolla** (1999). This system tracks the image edges of known three-dimensional objects for which a CAD model is available. Objects are tracked by comparing projected model edges to edges detected in a monochrome video feed. The advantage of tracking edges is that these can be detected very quickly by the use of a *prior*: an advance guess of where each edge will be in the image. As described in Section 4.2, this allows the system to perform very rapid edge searches using a 1D search along strips of pixels perpendicular to the predicted edge locations. The number of pixels searched in the image is thus vastly lower than for any full-frame feature extraction, and this results in fast operation. This system has previously been applied to the real-time control of articulated and mobile robots (**Drummond & Cipolla**, 1999; **Klein**, 2001; **Molana**, 2000).

The local, prior-based edge search is rapid, but is also essential to solve the correspondence problem. Fiducials can be designed with unique appearances so telling one from the next is easy; point features are often characterised by image patches or gradient measurements taken in their surroundings. Image edges, on the other hand, are merely modeled as one-dimensional intensity discontinuities and are therefore difficult to distinguish from one another. Hence, the tracking system employs the prior to resolve any ambiguities: if multiple image edges are found close to a model edge projected into the image, the tracking system assumes that the nearest image edge corresponds to the model edge.

For this reason (and also because the range of local search is limited) it is essential that the prior pose estimate for each frame is quite accurate. In robotics, good priors can be obtained by the use of motion models; since the robot inputs are known, the position of the camera for a new frame can be predicted using a model of the robot's dynamics. For a head-mounted camera, this is not possible, and a user's motions can

¹The recently developed FAST corner detector by **Rosten & Drummond** (2005) may be a notable exception.



Figure 4.1: Substantial motion blur due to 2.6 rad/s camera rotation, with 100x100 pixel enlargement

be quite unpredictable. Even if a first-order constant-velocity motion model is used to predict the user's motions, rapid head accelerations can result in priors which are far enough from the actual pose that tracking fails, either due to correspondence errors or due to features being outside the search range.

This chapter shows how low-cost inertial sensors (rate gyroscopes) can be added to the visual tracking system to provide the accurate priors necessary for robust operation. Due to drift, rate gyroscopes cannot reliably track head orientation by themselves: these devices measure angular velocity, and thus pose must be obtained by integration of noisy measurements, resulting in unbounded errors over time. Thus rate gyroscopes produce good high-frequency measurements but poor low-frequency ones; this is in contrast to the visual system, which fails under rapid motion but produces very accurate results when moving slowly. It is the fusion of these complementary sensors that yields good results in both circumstances - up to the point when images are sufficiently degraded by motion blur.

In anything but sunny outdoor lighting conditions, a camera's shutter must remain open for a considerable amount of time to capture enough light for a noise-free image; in fact many cameras' exposure times are almost the full amount of time available for a frame (in the case of a PAL camera, almost 20ms per field.) When a camera is moving, scene elements move across the image during exposure, resulting in motion blur: An example of a motion-blurred field captured from a camera rotating with 2.6 radians/s is shown in Figure 4.1. Motion blur is a problem because it degrades trackable features:

it smears sharp image edges into gradated blurs, point features into lines, and textured areas into homogeneous grays. In such conditions, most visual tracking systems fail.

This chapter shows that edge-based tracking can be made to operate even in the presence of very substantial blur. This is possible because the low-latency inertial sensors can be used to predict the direction and magnitude of blur at any point in the image. Using these predictions, the behaviour of edge detection can be adapted to suit the circumstances: instead of searching for an intensity step in the image, the tracking system can search for blurred edges instead. By fully exploiting the information which the rate gyroscopes make available, sequences with rotational velocities of up to 4.7 radians per second - in which features are blurred over distances as great as 50 pixels - have been tracked.

Section 4.2 describes the standalone operation of the edge-based tracking system, as it might be used for a robotics task. Section 4.3 describes inertial sensors which can be used to directly measure rotational velocity, and Section 4.4 shows how measurements from the visual and inertial sensors can be combined. The resulting tracking performance is described in Section 4.5.

4.2 Tracking System Operation

The tracking system continually updates an estimate of the position of a video camera relative to known objects in the world. This pose estimate is stored as a 4×4 coordinate frame transformation matrix $E_{\mathcal{C}\mathcal{W}}$ which transforms points $x_{\mathcal{W}}$ from the world coordinate frame \mathcal{W} , in which the model is described, to the camera-centered coordinate frame \mathcal{C} :

$$\mathbf{x}_{\mathcal{C}} = E_{\mathcal{C}\mathcal{W}} \mathbf{x}_{\mathcal{W}}. \quad (4.1)$$

This matrix, which completely describes the pose of the camera relative to the world, is updated by continually comparing the video image of the known features with their expected image positions. This process is illustrated in Figure 4.2, where a cardboard

model of a ship-part as might be encountered by a welding robot is placed at a known position. This tracking loop is performed once per video frame and consists of four steps:

1. **Image acquisition** Images are captured from a monochrome PAL video camera connected to the workstation via frame-grabber card.
2. **Model rendering** A CAD model of the known object or environment is rendered using an estimate of the camera's position. This determines which model edges are expected to be visible in the video image.
3. **Image measurement** Edges of the CAD model are compared to edges found in the video image by performing local searches for the nearest video edges.
4. **Pose update** Based on the measurements made, a new pose which bests aligns model to image edges is computed.

These individual stages are now described in more detail.

4.2.1 Image Acquisition

For tracking experiments, images are captured with a Pulnix TM-500 monochrome video camera. This camera delivers full-frame interlaced images with 576 lines at 25Hz or can deliver individual fields of 288 lines at 50Hz. The camera accepts C-mount lenses: typically an 8.5mm lens with a horizontal field-of-view of 36° was used for the robotics applications of **Drummond & Cipolla** (1999) and **Molana** (2000). To improve the tracking system's robustness, in **Klein** (2001) this lens was replaced with a 4.2mm wide-angle lens. With a horizontal field-of-view of 80°, this lens increases the number of trackable features visible in any frame and reduces apparent image motion due to camera rotation. This effect is illustrated in Figure 4.3.

Images are captured on a PC equipped with a video capture card based on the BT878 chip-set. Captured images are 8 bits per pixel greyscale and have a resolution of

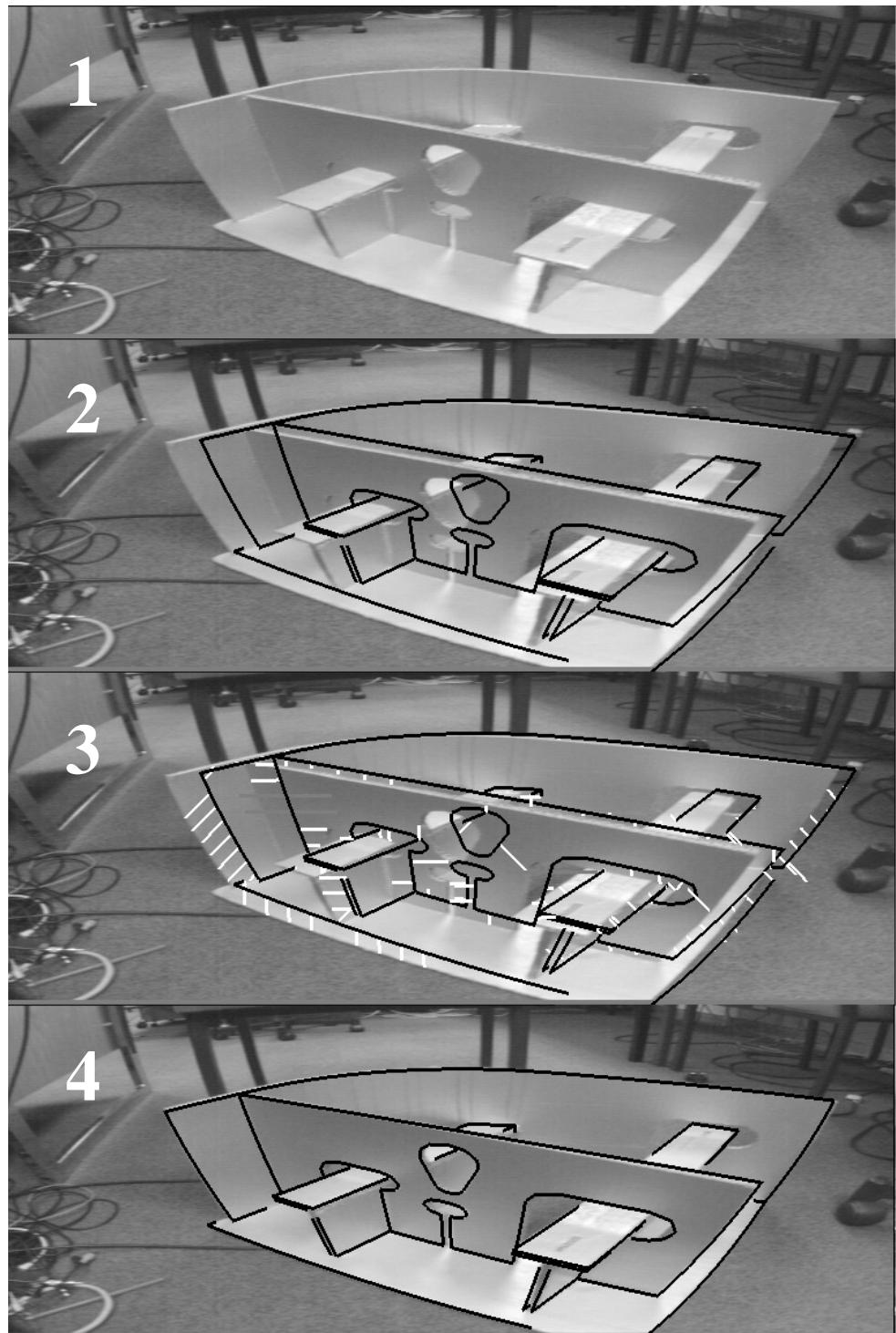


Figure 4.2: Tracking system loop. 1: Image acquisition, 2: Model rendering 3: Image measurement 4: Pose update

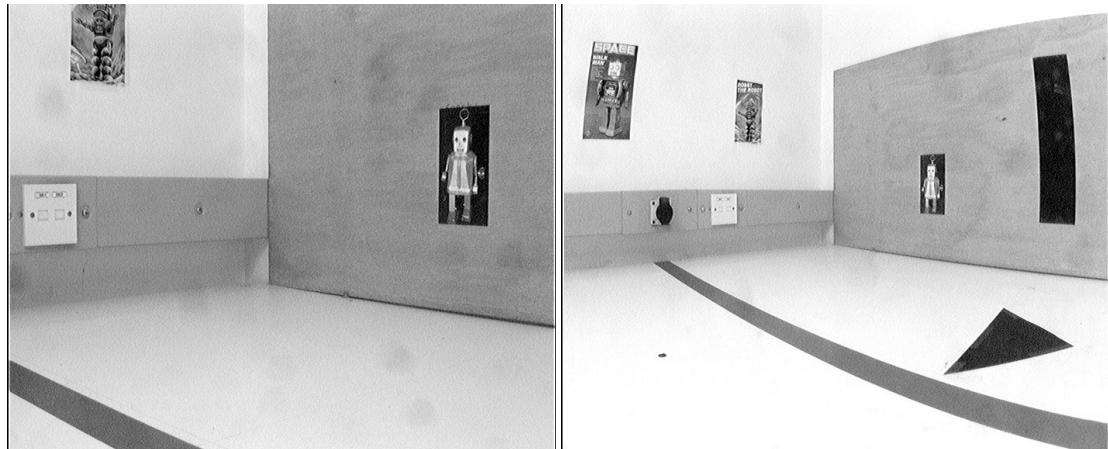


Figure 4.3: Comparison of 8.5mm (left) and 4.2mm (right) lenses from the same position. Using the 4.2mm lens, the system can ‘see’ more features for tracking, but straight lines are noticeably bent by lens distortion.

768×576 for full frames or 768×288 for individual fields. Images are displayed on the workstation using hardware-accelerated `glDrawPixels`.

4.2.2 Model Rendering and Camera Model

A CAD model describing the geometry of the object to be tracked is available. This model describes occluding faces and salient edges of the object and is measured in the world (\mathcal{W}) coordinate frame. After a frame is captured, the tracking system renders the edges of the model to the screen using the *prior pose estimate*, which may be just the pose estimate derived from the previous frame, or an estimate obtained from a motion model. Using the superscript “-” to denote the prior, a point in the world reference frame $x_{\mathcal{W}} = (x_{\mathcal{W}} \ y_{\mathcal{W}} \ z_{\mathcal{W}} \ 1)^T$ projects into the image as

$$\begin{pmatrix} u \\ v \end{pmatrix} = \text{CamProj} (\tilde{E}_{\mathcal{C}\mathcal{W}} x_{\mathcal{W}}) \quad (4.2)$$

where the CamProj() function represents the mathematical model of the camera and lens used. A standard pin-hole model would use a projection of the form

$$\text{LinearCamProj} \begin{pmatrix} x_c \\ y_c \\ z_c \\ 1 \end{pmatrix} = \begin{bmatrix} f_u & 0 & u_0 \\ 0 & f_v & v_0 \end{bmatrix} \begin{pmatrix} \frac{x_c}{z_c} \\ \frac{y_c}{z_c} \\ 1 \end{pmatrix} \quad (4.3)$$

where f_u and f_v describe the lens' focal length and u_0 and v_0 describe the optical center (all in pixel units). This projection model assumes that lenses exhibit no *radial distortion*; while this is often the case for telephoto lenses, wide-angle lenses often exhibit substantial amounts of *barrel distortion*. This effect can be observed in Figure 4.3.

The system uses a polynomial model of barrel distortion as found in **Ghosh** (1988). The relationship between a radius r in an undistorted, pin-hole projected image and its mapping \tilde{r} in a distorted image can be approximated by a polynomial in odd powers of \tilde{r} , where r , the undistorted radius, is calculated by

$$r = \sqrt{\left(\frac{x_c}{z_c}\right)^2 + \left(\frac{y_c}{z_c}\right)^2}. \quad (4.4)$$

The polynomial approximation takes the form

$$r = \tilde{r} + \alpha_1 \tilde{r}^3 + \alpha_2 \tilde{r}^5 + \dots \quad (4.5)$$

It is computationally more convenient to re-arrange this by reversion of series to the form

$$\tilde{r} = r - \beta_1 r^3 - \beta_2 r^5 + \dots \quad (4.6)$$

Only the first three terms of this expression are used, yielding

$$\tilde{r} = r - \beta_1 r^3 - \beta_2 r^5. \quad (4.7)$$

The full camera projection including radial distortion is given by

$$\text{CamProj} \begin{pmatrix} x_c \\ y_c \\ z_c \\ 1 \end{pmatrix} = \begin{bmatrix} f_u & 0 & u_0 \\ 0 & f_v & v_0 \end{bmatrix} \begin{pmatrix} \frac{\tilde{r} x_c}{r z_c} \\ \frac{\tilde{r} y_c}{r z_c} \\ 1 \end{pmatrix}. \quad (4.8)$$

Hidden edge removal is performed using hardware-accelerated OpenGL. The CAD model's faces can be rendered into the z-buffer using an un-distorted projection model. Points along the model's edges are then tested for visibility by drawing individual pixels at these points and using the GL_ARB_occlusion_query extension, which allows the success of the pixel's z-buffer test (and thus the visibility of the point) to be read back by the application.

4.2.3 Image Measurement

On every model edge, sample points are generated at 20-pixel intervals and tested for visibility. At each visible sample point, a local search for the nearest image edge is performed, and the distance to this detected image edge recorded. This process is illustrated in Figure 4.4.

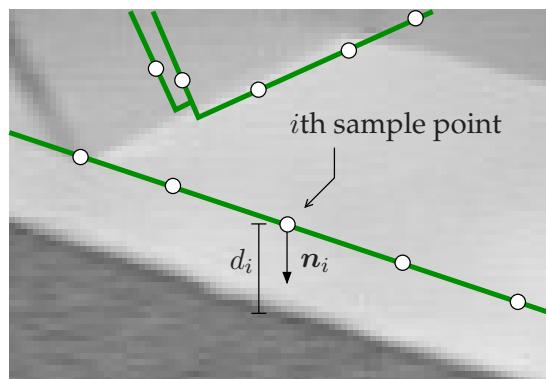


Figure 4.4: Video edge search at a sample point

At the i th sample point, 1-D edge detection is performed in the direction of the edge normal \mathbf{n}_i (the direction of this normal is adjusted to the nearest 45 degrees.) Edges are found by convolution with the kernel $(1 \ -1)^T$. The nearest local maximum which exceeds a threshold is accepted as the nearest edge, and the distance d_i to this edge is recorded. If no nearby edge is found, the sample point is skipped. All the edge distances measured in the image are stacked as an error vector \mathbf{d} .

4.2.4 Pose Update

Using these image measurements, the posterior pose which best aligns the rendered model to the edges in the video image is calculated. The posterior pose E_{CW} is expressed as a small motion from the prior pose:

$$E_{CW} = M_C \tilde{E}_{CW} \quad (4.9)$$

where M_C , the small motion in the camera frame, is parametrised by the motion 6-vector μ . To calculate μ , a Jacobian matrix J which describes the effect of each element of μ on each element of d is calculated by taking partial derivatives at the current camera pose estimate:

$$J_{ij} = \frac{\partial d_i}{\partial \mu_j} \quad (4.10)$$

The partial derivatives are found by differentiating the projection equations for each sample point w.r.t. the six motion parameters $\mu_1 \dots \mu_6$:

$$\frac{\partial d_i}{\partial \mu_j} = -\mathbf{n}_i \cdot \frac{\partial}{\partial \mu_j} \begin{pmatrix} u_i \\ v_i \end{pmatrix} \quad (4.11)$$

$$= -\mathbf{n}_i \cdot \frac{\partial}{\partial \mu_j} (\text{CamProj} (M_C \tilde{E}_{CW} \mathbf{x}_i)) \quad (4.12)$$

These differentials may be found by the chain rule as shown in Appendix B.1. For N sample points this yields an $N \times 6$ Jacobian matrix. The motion vector may then be found as the solution of the equation

$$J\mu = d. \quad (4.13)$$

This can be found by least-squares, using the pseudo-inverse of J

$$\mu = J^\dagger d \equiv (J^T J)^{-1} J^T d \quad (4.14)$$

however this solution is not very robust. Due to noise, mis-detection of features and occlusion, the statistics of d are significantly non-Gaussian. In particular, the error distribution typically encountered has higher-density tails than a Gaussian, and thus a direct least-squares solution is inappropriate. Instead, a robust estimation for μ is performed by finding the least-squares solution of

$$\begin{bmatrix} W^{\frac{1}{2}} J \\ P \end{bmatrix} \mu = \begin{pmatrix} W^{\frac{1}{2}} d \\ \mathbf{0} \end{pmatrix} \quad (4.15)$$

where P and W are diagonal matrices of size 6×6 and $N \times N$ respectively. The regularisation matrix P corresponds to a zero-motion prior and serves to stabilise the solution. W is a weighting matrix which scales down the influence of outliers on the estimation: the effect of each sample point is re-weighted by a decaying function of d to obtain a robust M-estimator. An introduction to M-estimators is given in Appendix C.

4.2.5 Motion Model

Once the posterior pose estimate has been found, the next frame is acquired from the camera and the tracking loop recommences. In the simplest implementation of the tracking system, the calculated posterior becomes the prior pose estimate for the next frame. This greatly restricts the range of velocities over which the tracking system can be used over, since even quite gentle camera pans can move features beyond the range of local edge detection if the detection is started at the previous frame's position. Instead, a decaying velocity model is used to predict the camera's pose at the next frame. For the current frame at time t , the prior for the next frame at time $t + \Delta t$ is

$$\tilde{E}_{\mathcal{CW}|t+\Delta t} = \tilde{M}_{\mathcal{C}} E_{\mathcal{CW}|t} \quad (4.16)$$

Where the predicted inter-frame motion $\tilde{M}_{\mathcal{C}}$ depends on the 6-DOF velocity estimate \mathbf{v} :

$$\tilde{M}_{\mathcal{C}} = \exp(\mathbf{v}\Delta t) \quad (4.17)$$

This velocity estimate is updated with the tracking system's motion vector $\boldsymbol{\mu}$ and a decay term which damps the system and aids stability:

$$\mathbf{v}_{|t+\Delta t} = 0.9(\mathbf{v}_{|t} + \boldsymbol{\mu}) \quad (4.18)$$

The resulting motion model helps the camera track large velocities, as long as these are relatively constant (limited acceleration) and as long as motion blur does not significantly corrupt the image. However, in some situations the motion model can have a negative effect; for example, a camera moving and then striking an object undergoes very rapid deceleration. A tracking system without motion model would have



Figure 4.5: Rate gyroscopes affixed to camera

no trouble tracking the now stationary camera, whereas the constant velocity motion model may not cope with the sharp change in speed and cause tracking to fail.

4.3 Inertial Sensors

To increase the tracker's robustness towards rapid rotations, three Futaba G301 piezo-electric gyroscopes were affixed to the camera as shown in Figure 4.5. These gyroscopes produce an output voltage which varies linearly with rotational velocity. The voltage is sampled using a 10-bit ADC and transmitted to the workstation via a serial link at 9600 bps, resulting in a sampling frequency used of 171 Hz. The circuitry used was developed by **Kumar (2001)**.

Each gyroscope produces an output voltage V linearly related to angular velocity:

$$V = b + \alpha\omega \quad (4.19)$$

where ω is rotational velocity about the gyroscope's axis, b a bias voltage and α the gyroscope sensitivity. At rest when $\omega = 0$, the bias b can be measured directly. For a

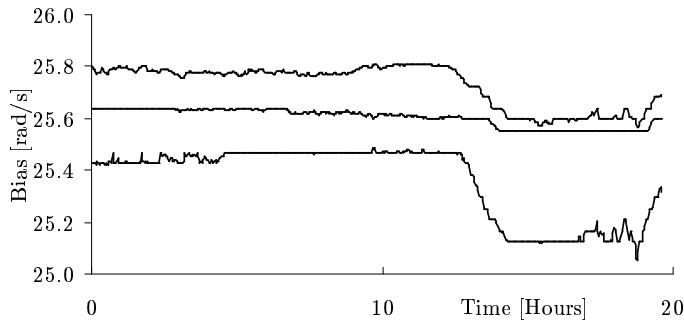


Figure 4.6: Long-term bias drift from three rate gyroscopes

single axis, angular displacement θ is found by integration over time:

$$\theta = \int \frac{V - b}{\alpha} dt \quad (4.20)$$

The parameter α can be determined by performing the above integration while rotating the gyroscope about a known angle.

The form of equation 4.20 means that estimates of angular position are very sensitive to errors in bias: small steady-state errors in b produce unbounded errors in θ . The rate gyroscopes used are sensitive to operating temperature and their bias can change markedly over time as shown in Figure 4.6. Hence, the value of b must be continually updated to ensure long-term robustness. The mechanism used for this is described in Section 4.4.3.

4.4 Sensor Fusion

The visual tracking system by itself offers highly accurate, drift-free measurements as long as motion is not too rapid; the rate gyroscopes provide reliable measurements of rapid motion but bias errors and noise result in drift when stationary. The complementary strengths and weaknesses of the two sensors make them ideal candidates for sensor fusion. This fusion is described here, and has three key components: an initialisation of the visual tracking system's pose estimate before video frame processing, a

modification of the edge detection process to support motion blur, and an update of the gyroscope's bias estimate.

4.4.1 Tracking System Initialisation

The visual tracker described in Section 4.2 uses local edge searches about a predicted prior position in the video feed. Furthermore, it linearises pose changes about the prior pose. As a result, the visual tracking system is best suited for correcting small pose errors and the prior needs to be accurate. If image motion beyond the range of the local edge search occurs (for example, due to sudden rapid camera rotation), the visual tracking system fails entirely.

The time needed to capture a video frame from the camera and transfer this frame from video hardware to the tracking system is large compared to the time needed to sample information from the gyroscopes. Hence, a record of gyroscope information corresponding to camera motion between the previous and the current video frames is always available to the tracking system before image processing commences. This information can be used to predict camera orientation for the new video frame, and replaces the rotation component of the motion model previously described. Linear accelerometers are not currently used, so the estimate of linear velocity $v_1 \dots v_3$ from Section 4.2.5 remains unchanged. The vector $\tilde{\mu}_c$ representing predicted change in camera pose is formed by combining the tracker's velocity estimates with gyroscope measurements:

$$\tilde{\mu}_c = (v_1\Delta t \quad v_2\Delta t \quad v_3\Delta t \quad \theta_1 \quad \theta_2 \quad \theta_3)^T \quad (4.21)$$

with θ_n evaluated as in Equation 4.20 and v_n being the predicted linear displacement along the n th axis. The prior prediction $\tilde{E}_{C\mathcal{W}|t+\Delta t}$ for the camera pose is computed using this estimated motion and the previous frame's posterior, $E_{C\mathcal{W}|t}$ as before:

$$\tilde{E}_{C\mathcal{W}|t+\Delta t} = \exp(\tilde{\mu}_c) E_{C\mathcal{W}|t}. \quad (4.22)$$

This operation provides the prior estimate needed for the model rendering step of the tracking system's operation.

4.4.2 Parametric Edge Detector

In Section 4.2.3 (which describes Step 3 of the tracking system loop) the camera was assumed to capture images using an ideal sampling function $f(t) = \delta(t)$. Under this assumption edge detection could be performed by constructing a vector \mathbf{i} of pixel intensities around each sampling point in the direction of the edge normal and convolving with the differential kernel $\mathbf{k}_{\text{diff}} = \begin{pmatrix} -1 & 1 \end{pmatrix}$ to give a vector of edge intensities, of which the nearest large local maximum is the detected edge position.

The assumption of a very short exposure time is not however valid for the camera used. Although cameras with very rapid exposure times exist, these usually require a high light intensity for operation and may not be suitable for operation in a standard environment, e.g. inside a normal building. Under these conditions, cameras often exhibit substantial motion blur, as illustrated in Figure 4.1. A better model of the sampling function of these cameras is a rectangular pulse:

$$f(t) = \frac{1}{t_e} \begin{cases} 1 & -\frac{t_e}{2} \leq t \leq \frac{t_e}{2} \\ 0 & \text{otherwise} \end{cases} \quad (4.23)$$

where t_e is the camera's exposure time. An image edge (step function) moving across the image at a rate of v pixels/second will thus appear as an intensity ramp of length vt_e pixels in the sampled field. The edge detection in Step 3 of the tracking system loop can be modified to detect blurred edges by using inertial sensor information to produce an estimate of camera motion during the sampling period:

$$\boldsymbol{\mu}_C = t_e \begin{pmatrix} 0 & 0 & 0 & \omega_1 & \omega_2 & \omega_3 \end{pmatrix}^T. \quad (4.24)$$

For each sample point, an estimate of the length b of an edge's motion blur in the direction of the edge normal can simply be found by multiplying with the corresponding row of Jacobian matrix J from Equation (4.10); i.e. for the i th sample point

$$l_i = J_i \boldsymbol{\mu}_C. \quad (4.25)$$

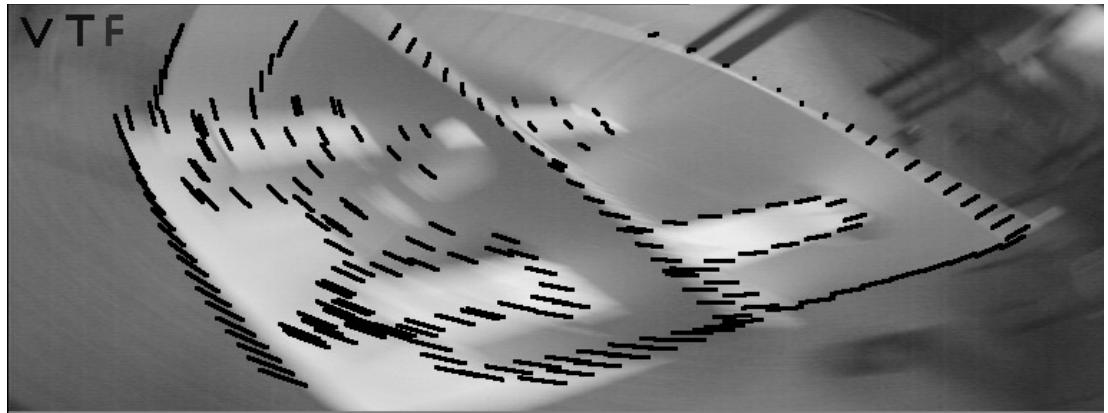


Figure 4.7: Image motion vectors of motion blur predicted from inertial sensors

Figure 4.7 shows the system's estimate of motion blur superimposed over a blurred video frame.¹ Edge detection is performed by convolution with a matched filter. The ramp kernel \mathbf{k}_{ramp} is used:

$$\mathbf{k}_{\text{ramp}} = \frac{1}{2l^2} \begin{pmatrix} -l & -l+2 & \dots & l-2 & l \end{pmatrix}^T. \quad (4.26)$$

When this kernel is convolved with the pixel intensities, the maxima indicate the sensed locations of blurred edges. The edge detection process is illustrated in Figure 4.8. The first plot shows image pixel intensity measured along the horizontal black line in the enlargement. These pixel intensities are convolved both with the differential kernel \mathbf{k}_{diff} (second plot) and a ramp kernel \mathbf{k}_{ramp} of length 36 (third plot.)

4.4.3 Gyroscope Re-calibration

As shown in Figure 4.6, the gyro bias parameters b are not constant. For long-term robustness, it is therefore necessary to update the system's bias estimate. This is done by comparing inertial predictions of rotational motion with measurements made by the visual system. If the rotational displacement around an axis between two visual measurements time Δt apart is Θ_n and the bias value is assumed to take the form

¹Figure 4.7 shows predicted blur vectors. For edge detection, only the edge normal component as calculated in Equation 4.25 is used.

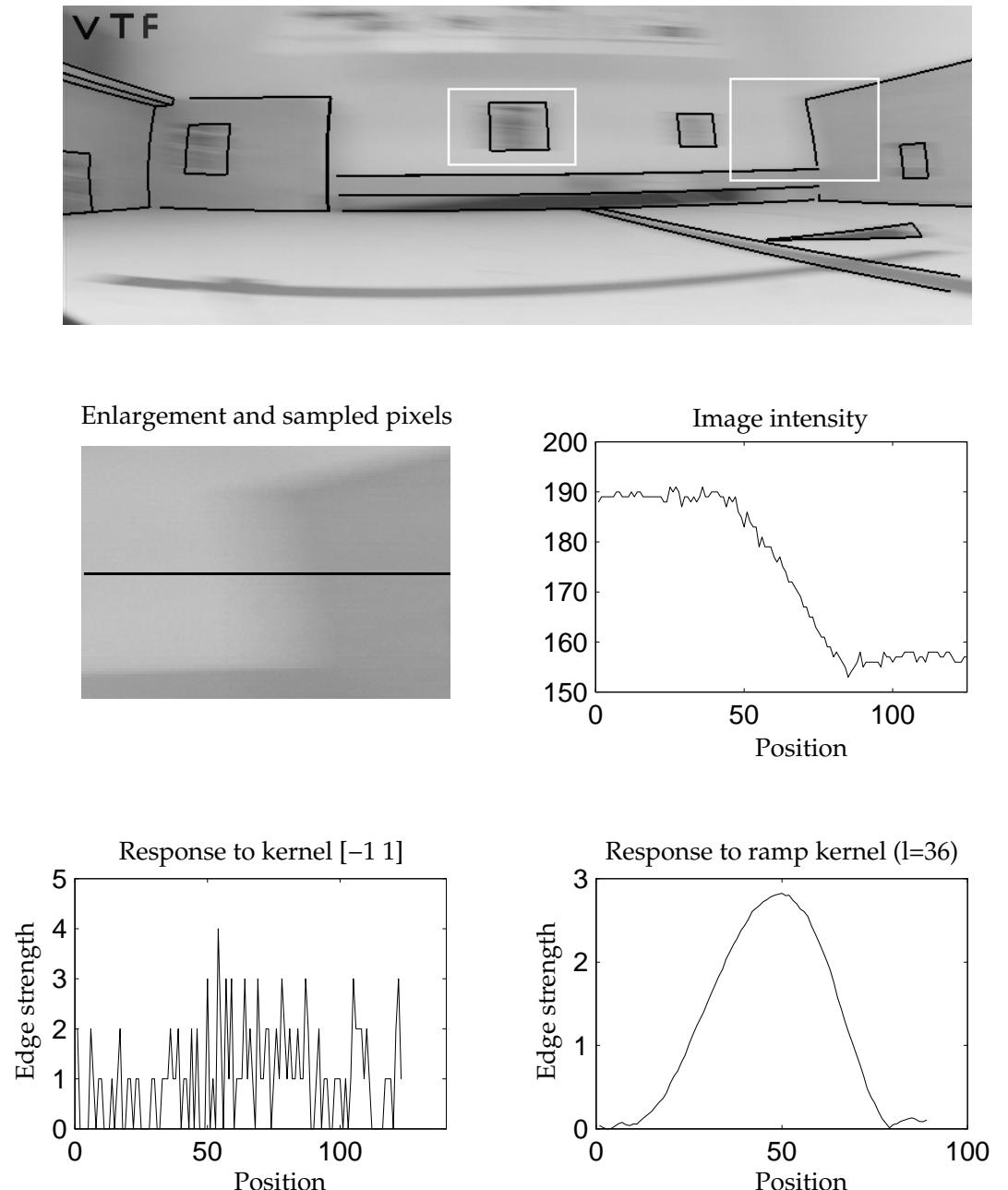


Figure 4.8: Still frame from a tracked sequence with enlargement and plots of pixel intensity and detected edge strengths taken along the black line in the enlargement

$b = b^{true} + e$ where e is bias error, it follows from equation 4.20 that (assuming perfect measurements)

$$e = \frac{\alpha(\Theta - \theta)}{\Delta t}. \quad (4.27)$$

In practice, differential measurements are noisy, and bias values are corrected by a small (typically $\frac{1}{100}$ th) fraction of the calculated error. This avoids corruption by spurious measurements and does not impact performance since bias drift is a gradual process.

4.5 Results

The tracking system presented was tested on three different test scenes with available edge models. The ‘tabletop’ scene (shown in Figure 4.8) places the camera in a simple immersive table-top environment as used for the visual guidance of a mobile robot in Klein (2001). The ‘ship’ scene points the camera at a model of a ship part (Figure 4.2) such as could be found in a visual servoing application (Drummond & Cipolla, 1999). The ‘cubicle’ scene contains a portion of a computer lab (Figure 4.1).

In each scene, the camera undergoes increasingly rapid motion relative to its target while the tracking system was run in three modes: without any inertial information, using inertial information to predict camera pose, and using inertial information both for pose and blur prediction. Table 4.1 shows the maximum rotational velocities at which tracking was sustainable. The tracking system’s performance differs greatly from scene to scene: While the ‘ship’ and ‘tabletop’ scenes contain many edges of modest contrast, the ‘cubicle’ scene contains high-contrast feature such as windows and light fittings and is trackable even at high rotational velocities. A video (`edge_tracking.avi`) demonstrating tracking performance is enclosed on the accompanying CD-ROM.

Figure 4.8 shows the tracking system correctly tracking a sequence in the ‘tabletop’ scene while the camera is rotating about its vertical axis with 3.1rad/s. This was the highest rotational velocity at which correct tracking was maintained for this scene.

Sequence:	Tabletop	Ship	Cubicle
Visual sensor only [rad/s] (pixels)	0.3 (3)	0.3 (3)	1.0 (11)
With pose initialisation	0.8 (8)	1.2 (13)	3.6 (38)
With blur prediction	3.1 (33)	2.0 (21)	4.7 (50)

Table 4.1: Tracking system performance for three scenes: Maximum trackable rotational velocities in rad/s (and corresponding motion blur in pixels)



Figure 4.9: Motion blur enlargements at various rotational velocities [rad/s]

Video image quality at different rotational speeds is compared in Figure 4.9, which shows enlargements corresponding to the central white outline in Figure 4.8. The first four enlargements show rotational velocities trackable with the combined system, the last was untrackable.

Fitting error was measured for the ‘tabletop’ test scene by fitting straight lines (in the undistorted space) to each edge’s sample points’ measurements and calculating the residual error after the line was fit. The mean error for sample points with no motion blur was found to be 1.1 pixels. The error increased to 4.2 pixels for sample points with a motion blur of 7 pixels and reached a maximum of approximately 5.5 pixels for motion blurs of 20-33 pixels.

The results demonstrate that the addition of an inertial pose prediction to the tracking system greatly increases the system’s robustness. Pose prediction by itself is however not sufficient when camera motion is such that motion blur corrupts image measurements. In this case, the estimation of motion blur and use of a parametric edge detection algorithm further increase the robustness of the system.

While the inertial sensors used can measure rotational velocity, linear velocity is still estimated from visual measurements. This shortcoming could be addressed by the

addition of linear accelerometers; however, since these devices measure the second differential of position and require the subtraction of a gravity vector, results are likely to be inferior to those for the rotational sensors.

It should be pointed out that neither the rate gyroscopes nor the linear accelerometers provide any information about possible motion of the objects tracked, and so the tracking of rapidly moving objects is not supported. Hand-held manipulation of objects being tracked, e.g. the ship part, generally produces motions much slower than those produced by camera shake, and is well-trackable.

The motion blur correction used is not suitable for parallel edges whose separation is comparable to the size of local motion blur. The use of more advanced rendering techniques (such as the use of multiple levels of detail) may help address this issue. However this would require suitably marked-up models and further increase the system's already considerable dependency on data prepared off-line.

5

HMD-Based Augmented Reality

5.1 Introduction

This chapter shows how a HMD-based AR application can be built around the markerless tracking system described in Chapter 4. It describes the development of an AR application based on a stereoscopic, semi-transparent Head-Mounted Display (HMD) which can overlay computer-generated graphics onto the user's view of the real world. By mounting a camera at a fixed offset to this display, the tracking system can be used as the position sensor necessary for accurate registration of the computer-generated visuals with objects in the real world. Figure 5.1 shows a picture of the helmet-mounted display and sensors.

While the tracking system of Chapter 4 can be used un-modified, tracking head-pose alone is not sufficient to render graphics in the right place: a calibration of the user's eye positions and of the display's projection parameters is also required. This chapter

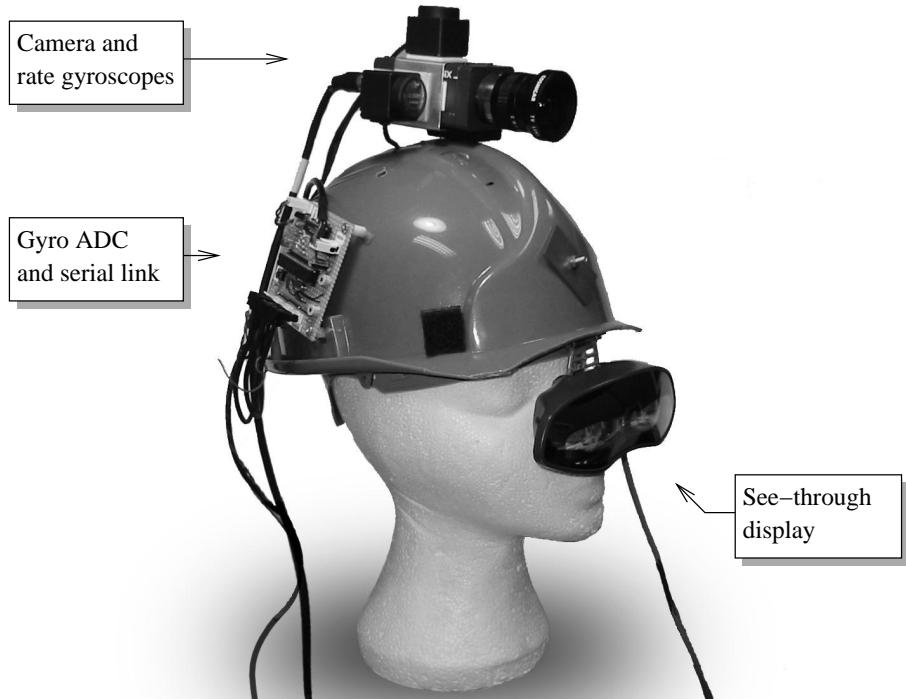
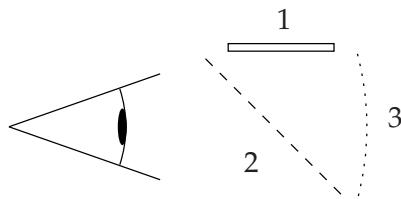


Figure 5.1: Head-Mounted Display. The display is mounted to a hard-hat to which camera and rate gyroscopes have also been attached.

shows how these parameters can be estimated using a simple calibration procedure and nonlinear optimisation. This optimisation allows the use of a nonlinear projection model which can compensate for the radial distortion produced by the display. Finally, to avoid virtual graphics lagging behind the real scene, velocity predictions from inertial sensors are used to improve the system's dynamic registration.

Section 5.2 describes the features of the display used, while the application it is used for is described in Section 5.3. Section 5.4 describes the rendering method used to draw augmented visuals, while Section 5.5 describes how the rendered visuals can be accurately registered with the view of the real world using calibration and prediction. Finally, Section 5.6 describes the performance of the resulting head-mounted AR system.



- 1: SVGA display
- 2: Semi-silvered mirror
- 3: Concave semi-silvered mirror

Figure 5.2: Optical layout of the HMD

5.2 Head-Mounted Display

The head-mounted display used is a Sony Glasstron LDI-100B. This display features one 800×600 pixel, 24 bpp display per eye. In its standard configuration, an SVGA signal is displayed identically on both displays; here, the display has been modified to produce frame-sequential stereo. Even-numbered frames from the computer are sent to one eye's display, odd-numbered frames to the other. The SVGA output is set to operate at 85Hz, resulting in a 42.5Hz update-rate for each eye.

The optical layout of the Glasstron HMD is illustrated in Figure 5.2. The display operates in optical see-through mode: light from the outside world passes through the semi-silvered mirrors to reach the user's eyes, giving the user a slightly darkened but otherwise undistorted view of the external environment. Simultaneously the user can see the images produced by the SVGA displays. This additive mixing of real and virtual images limits the scope of augmented visuals possible: As discussed in Section 2.3, it is not possible for real objects to be removed from the user's field of view. Virtual objects have a transparent appearance and cannot fully occlude real objects in the view (unless these are perfectly black.)

Figure 5.3 demonstrates how a user's view can be augmented with extra information using the HMD. Panel 1 shows the user's view of a scene without wearing the

HMD. Panel 2 shows the view when the HMD is worn, without any graphics being displayed: the view through the optics introduce a slight darkening and loss in clarity, and the user's field-of-view is reduced. To overlay annotations into the user's field of view, SVGA images are sent to the display: such an image is shown in Panel 3. Due to the additive mixing operation of the HMD, black portions of this image will not affect the user's view, and light portions of the image will appear as transparent overlays. The composited view is shown in Panel 4.

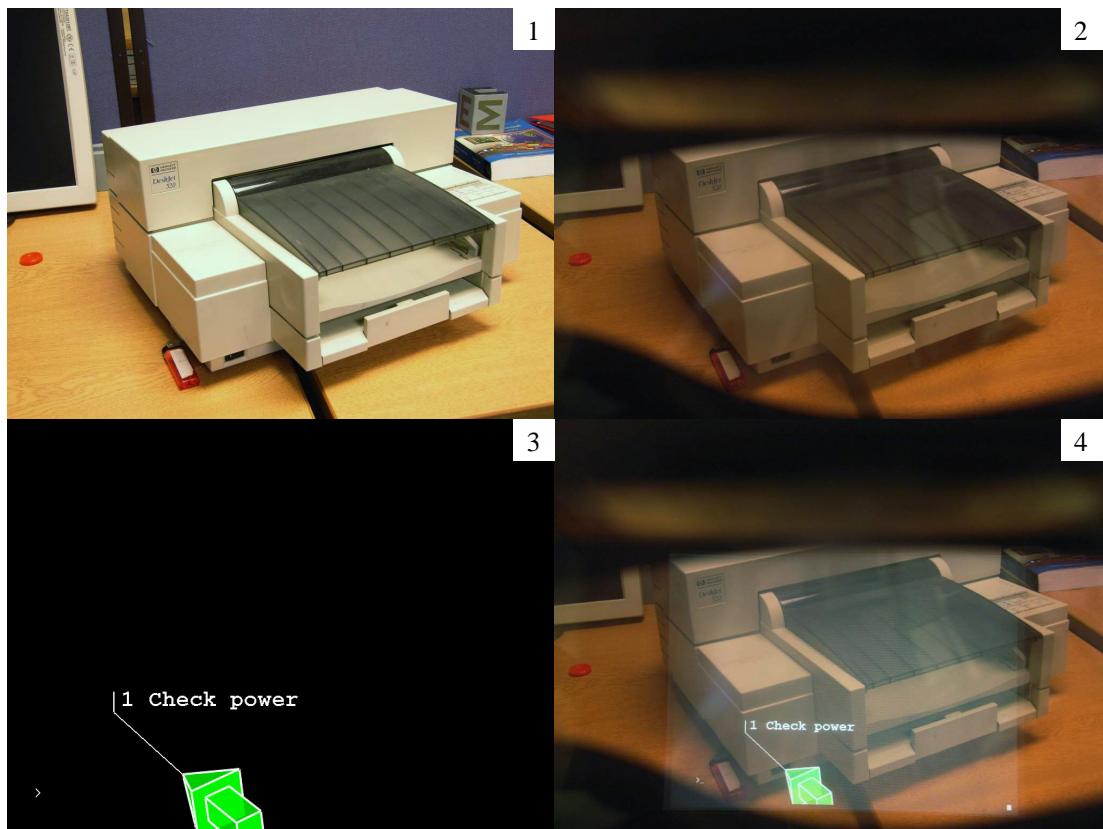


Figure 5.3: Image composition in the HMD. 1: User's view without an HMD. 2: The same view wearing the HMD. 3: Computer graphics sent to the HMD 4: Resulting composited view formed by additive mixing. Also faintly visible in this panel is the size of the rectangular area in which augmented visuals can be drawn.

5.3 A Prototype Maintenance Application

To evaluate the proposed AR system, a maintenance application mock-up was programmed. This application envisages an AR replacement for a paper-format maintenance checklist. For example, a mechanic performing a 100-step procedure on a gas turbine may currently refer to a paper checklist which describes and illustrates the steps to be taken. An AR replacement for this checklist would project instructions directly into the mechanic's field-of-view and directly highlight the relevant part in a step, eliminating the need to go back and forth between workpiece and checklist. For maintenance applications involving electronic devices, this application could further be sped up by automatically detecting when certain steps have been taken by the operator and advancing the checklist appropriately.

The application developed here demonstrates this principle on the basis of an ink-jet printer. A CAD model of this printer was constructed, allowing it to be tracked by the edge-based tracking system described in Chapter 4. A two-step checklist was programmed, by which the user is instructed to check the position of the power switch, and press one of the printer's buttons. The current activity is displayed as a text call-out which points at the location of the switch or button; further, an animated 3D arrow also shows the user where the next control is. Finally, the button to be pressed is highlighted with a ring which appears flush with the body of the printer. It is this printer maintenance application which appears on the display in Figure 5.3.

An AR printer maintenance application has previously been presented by **Feiner et al** (1993), who demonstrate the KARMA (Knowledge-based Augmented Reality for Maintenance Assistance) system: This system is designed to facilitate the difficult task of authoring AR applications by automatically deciding which things must be shown to the user in which order based on a maintenance instruction list. In this chapter the emphasis is on accurate registration rather than the function of the application; as a result, the application presented here does not approach the functionality of the KARMA system, but the quality of registration is improved.

Beyond the printer application, a CAD model of a server room was made and a second instance of the maintenance application generated: this one to guide a network technician to move a patch cable from one port in a network switch (located in a rack of network equipment) to another. This application is interesting because it is immersive - the tracked and augmented visuals surround the user (in contrast to the printer application in which only a single isolated object is viewed.) Again, text call-outs and arrows were used to illustrate the task. Further, in this application, individual equipment racks are given labels.

Illustrations of both of these applications and an evaluation of their performance is given in Section 5.6.

5.4 Projection Model and Rendering

3D graphics are rendered to the display by treating each eye's LCD screen as the image plane of a virtual camera located at that eye. This allows normal computer graphics projection models to be used. This section describes the projection model and rendering procedure used to render the view of each eye. Since each eye is treated individually with no shared parameters, the remainder of this section will describe the monocular case.

Given a calibrated display (the calibration procedure is described in Section 5.5) and a functioning tracking system, the position of the user's eye at any point in time is known and described by the matrix $E_{\mathcal{E}W}$ which describes the transformation from the world (\mathcal{W}) to the eye (\mathcal{E}) coordinate frame. The projection of a point $x_{\mathcal{W}}$ in the world to a pixel position on one eye's screen is given (similarly to Section 4.2) as

$$\begin{pmatrix} u \\ v \end{pmatrix} = \text{EyeProj} (E_{\mathcal{E}W} x_{\mathcal{W}}) \quad (5.1)$$

where $\text{EyeProj}()$ the eye-screen camera model. The display exhibits a small amount of radial distortion (in this case *pincushion* distortion); while the magnitude of this distortion is much smaller than the distortion exhibited by the 4.2mm tracking lens,

it is nonetheless plainly visible in the display, and a polynomial radial distortion approximation is included in the projection model. This differs from the model used for tracking in that only a single distortion parameter β is sufficient to model the weaker distortion:

$$\text{EyeProj} \begin{pmatrix} x_{\mathcal{E}} \\ y_{\mathcal{E}} \\ z_{\mathcal{E}} \\ 1 \end{pmatrix} = \begin{bmatrix} f_u & 0 & u_0 \\ 0 & f_v & v_0 \end{bmatrix} \begin{pmatrix} \frac{\tilde{r}}{r} x_{\mathcal{E}} \\ \frac{\tilde{r}}{r} z_{\mathcal{E}} \\ \frac{\tilde{r}}{r} y_{\mathcal{E}} \\ 1 \end{pmatrix} \quad (5.2)$$

with

$$r = \sqrt{\left(\frac{x_{\mathcal{E}}}{z_{\mathcal{E}}}\right)^2 + \left(\frac{y_{\mathcal{E}}}{z_{\mathcal{E}}}\right)^2} \quad (5.3)$$

$$\tilde{r} = r - \beta r^3. \quad (5.4)$$

Rendering is performed using accelerated z-buffered OpenGL. The display of lines and textured triangles is supported. For the evaluation of the AR display, the same CAD model as used for tracking is drawn, using black triangles (which appear transparent in the display) for hidden line removal. Rendering the edges of the tracking CAD model thus allows immediate visual inspection of the accuracy of tracking and calibration.

Radial distortion is not directly supported in the OpenGL rendering transformation, and therefore requires a separate step. In Section 4.2 radial distortion was performed on a point-by-point basis: only a few hundred points needed to be distorted very accurately, with no need for smoothly distorted lines or triangles. The requirements for AR rendering are different; the rendering of arbitrarily complex meshes may be required. Further, long line segments and large textured triangles should also be displayed with correct distortion, so a by-vertex distortion scheme is inappropriate.

Instead, the texturing approach presented in **Watson & Hodges** (1995) is used. An undistorted projection of the scene is first rendered into an off-screen buffer using full hardware acceleration. Radial distortion can then be applied by rendering a distorted grid to the screen using this buffer as a texture map. The vertices of the grid are pre-distorted with the inverse of the HMD's optical distortion to produce an undistorted

view for the user. A 20×20 grid has proven sufficiently accurate to avoid discontinuities at the grid cell boundaries.

5.5 Registration

5.5.1 Registration for Optical See-through Displays

For an AR application to function, the virtual graphics must be accurately registered, that is, correctly aligned with the relevant objects in the real world. For video see-through systems, in which the computer has access to the image the user sees, accurate registration can be achieved by simply employing visual tracking such as described in Chapter 4; the tracking system's projection model (E_{CW} and `CamProj()`) can then be used to render augmented visuals in the right place into the video feed.

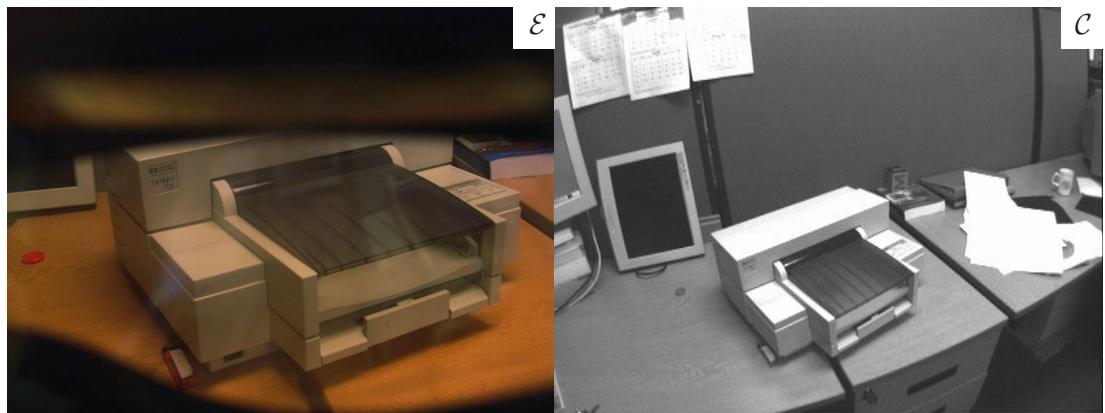


Figure 5.4: In an optical see-through system, the user and computer see different images. User's view (left) and computer's view (right).

Registration becomes more difficult for optical see-through systems such as used here. In these cases, the computer does not have access to the image of the real world that reaches the user's eyes. Instead, the computer uses video from a head-mounted camera which is mounted at some fixed offset to the display and most likely has very different projection parameters. This is illustrated in Figure 5.4, which shows the differ-

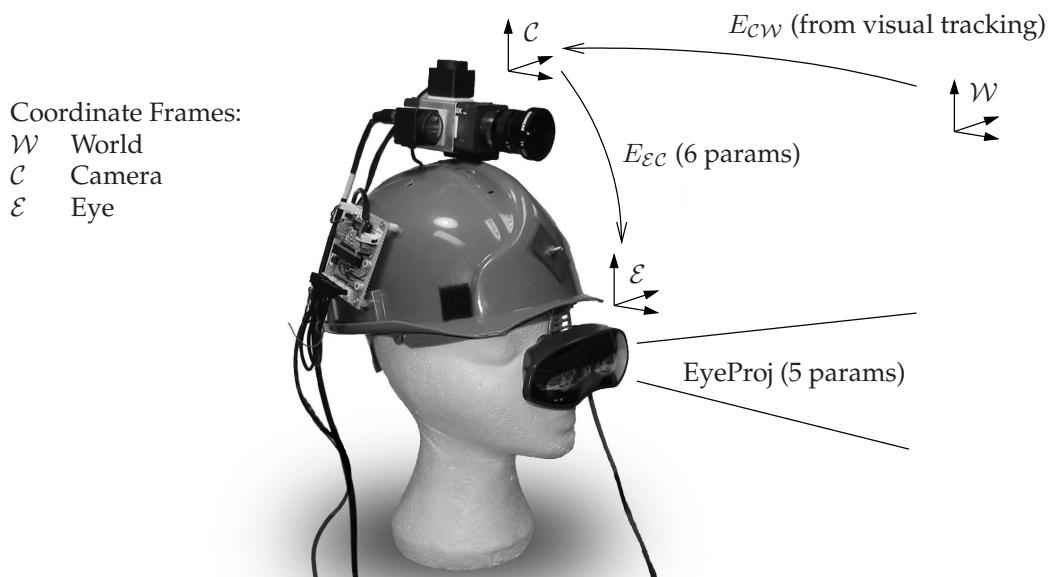


Figure 5.5: Coordinate frames and calibration parameters needed for projection

ence in the view the user and computer have of a scene. This difference in viewpoints must be accurately determined by a calibration procedure.

The tracking system measures E_{CW} , the transformation from world coordinate frame \mathcal{W} to the head-mounted camera's coordinate frame \mathcal{C} . To obtain the position of a user's eye $E_{\mathcal{E}W}$ required for rendering, an estimate of the transformation from camera to the user's eye E_{EC} must be obtained (from which $E_{\mathcal{E}W} = E_{EC}E_{CW}$). Further, the projection parameters parameters f_u, f_v, u_0, v_0 and β used for rendering of the augmented visual must also be calibrated. This gives a total of 11 parameters to be calibrated for each eye, as illustrated in Figure 5.5. Calibration is performed by gathering measurements from the user and then optimising projection parameters based on these measurements.

5.5.2 User Calibration Procedure

Due to differences in users' head geometries, the display parameters must be calibrated separately for each user, otherwise augmented features rendered are likely to

be significantly misaligned with their intended real-world locations. Indeed, even for a single user, a system perfectly aligned during one session can be very much misaligned when starting the next, as the AR display will not be in exactly the same place relative to the user's eyes. Very slight movements of the display relative to the eyes can have a large impact on the display's optical center and indeed on the magnitude of radial distortion encountered; focal length and modelled eye position are also affected.

Therefore, it is currently necessary to calibrate the display for every single use. Since the computer cannot see the view the user receives, calibration must rely on user input. Here, a calibration procedure which extends the approach of **Janin et al** (1993) is employed.

The user is provided with a mouse which controls cross-hairs visible in the AR display. To calibrate the projection parameters, the user is asked to align projected model vertices with their real-world location. To do this, the user first selects a projected vertex with the left mouse button, and then clicks on its real-world position with the right mouse button. A number of software measures are taken to make this task as accurate as possible:

- To reduce the negative impact of tracking jitter and user head motion on measurement accuracy, information from the tracking system is used to motion-stabilise the cross-hairs with respect to the projected geometry. If the mouse is not moved, the cross-hairs are not kept in a fixed screen position; instead, they are kept at a fixed screen offset from the current projection of the selected vertex. This appears intuitive in the display and means the user must not keep their head absolutely rigid during calibration. Further, the effect of tracking jitter on measurements is almost completely eliminated, since cross-hairs and projected vertex jitter by the same amount.
- the area immediately around the cross-hairs is cleared of any augmented visuals to provide the user with a clear view of the target geometry;¹

¹This effect is not shown in Figure 5.6 but can be observed in the video `hmd_calibration.mpg`.

- the eye not being calibrated is masked with a fully white screen, which obscures that eye's view of the real world sufficiently to ensure that the user calibrates the monocular view of the correct eye rather than a stereo view.

A high-contrast, accurately modelled calibration object is used to simplify the task. To ensure a good calibration, the user is encouraged to make measurements at many different places in the display, including the center and the extremities. Also, the user is encouraged to make measurements using vertices at many different distances - this is crucial for proper stereo alignment. If the tracking environment contains features at many depths, this is simply done by selecting some vertices which are near and some which are far away; if the environment is shallow, the user is encouraged to move nearer and further from the object during the course of the calibration.

Generally, the user will make five measurements at one range, then five at a different depth, and then make further measurements to clean up any remaining misaligned vertices as necessary. Figure 5.6 illustrates this procedure in six panels. The view in (a) is completely uncalibrated. The user has selected a vertex to adjust (indicated by a diamond) and moves the cross-hairs to the real-world position. The user clicks the right mouse button and the calibration is updated accordingly (b). The user then selects the next vertex (c) and clicks on the corresponding position (d). After three more measurements (for a total of five) the view is well aligned from this viewpoint (e); however, moving the closer to the target reveals errors (f) and more measurements will have to be made at this range. It should be pointed out that the very limited range of movement shown is due to constraints imposed by photography through the AR display, and a real user's head would move about more. The video file `hmd_calibration.mpg` which demonstrates the calibration procedure is enclosed.

The method presented here differs from the previous calibration methods discussed in Section 2.3.3. It is most similar to the method of **Janin et al** (1993), but differs in a few aspects: Perhaps the most significant is the use of a motion-stabilised cursor which greatly reduces the negative impact of tracking jitter on calibration and no longer requires the user to hold their head very still during calibration. Further, any vertex

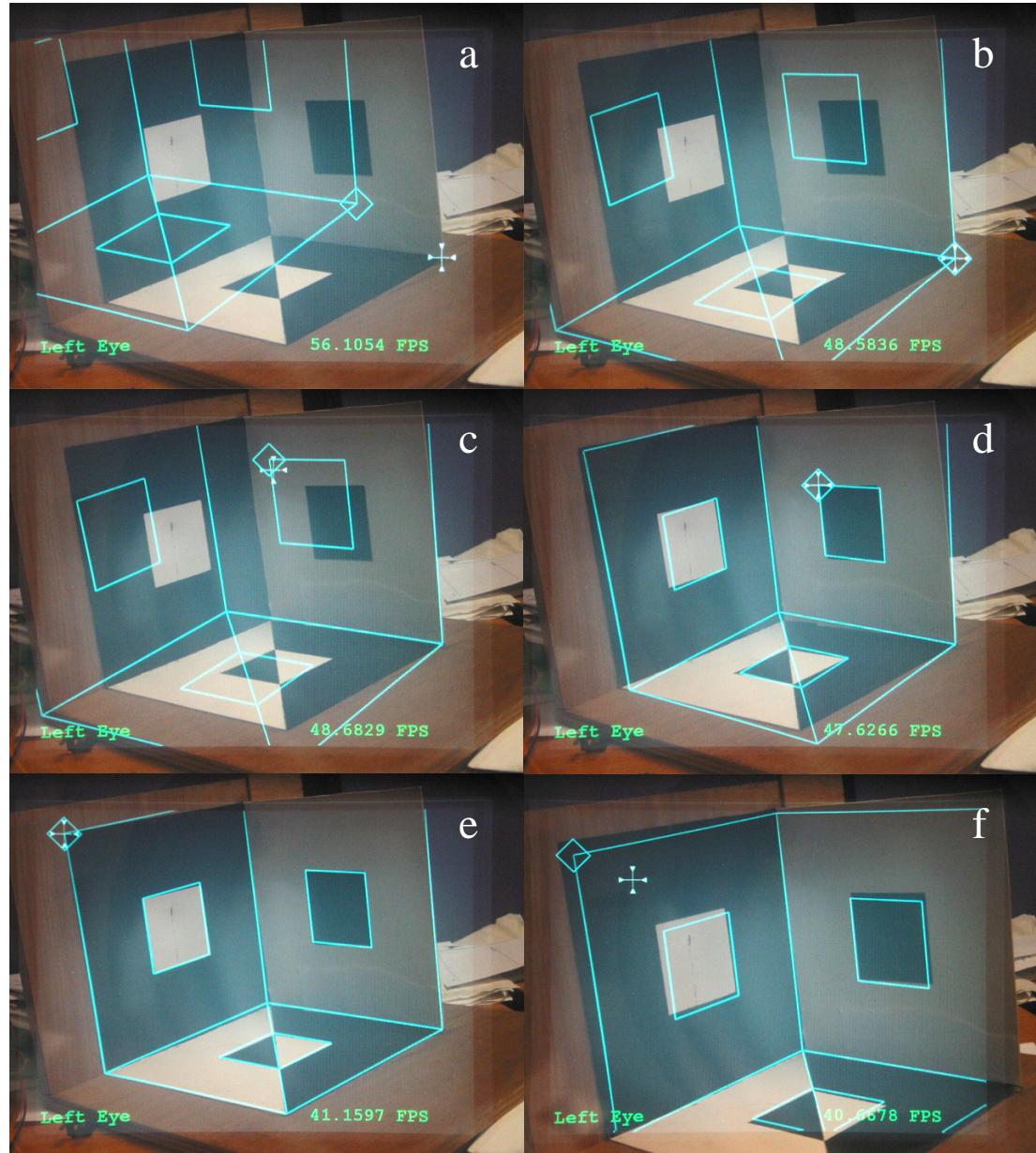


Figure 5.6: Calibration seen through the display

included in the tracked CAD model (rather than a single calibration point) can be used for calibration; this has the advantage that the user can always ‘fix’ the currently worst-projected part of the image and this leads to a more rapidly constrained calibration. Finally, the projection model used is richer (as described in Section 5.4).

5.5.3 Nonlinear Optimisation

Each user measurement provides the system with two pieces of information; a 2D display coordinate $(\tilde{u} \ \tilde{v})^T$ specified by the user’s cross-hairs, and a 3D sensor-camera coordinate $(x_c \ y_c \ z_c \ 1)^T$ which was the selected vertex’s camera-frame position at the time of the measurement. The ideal HMD calibration would project all the camera-frame coordinates to their measured display positions. Due to measurement noise and model inaccuracies, this is not achievable, and projection parameters are chosen to minimise the sum-squared re-projection error

$$\epsilon = \sum_{n=1}^N (\tilde{u}_n - u_n)^2 + (\tilde{v}_n - v_n)^2 \quad (5.5)$$

where N is the number of measurements made, and $(u_n \ v_n)^T$ is the projection of the n th sensor-camera coordinate. Re-writing the above in terms of an error vector e ,

$$\epsilon = |e|^2, \quad e = \begin{pmatrix} \tilde{u}_1 - u_1 \\ \tilde{v}_1 - v_1 \\ \vdots \\ \tilde{u}_N - u_N \\ \tilde{v}_N - v_N \end{pmatrix} \quad (5.6)$$

The minimisation of this error is equivalent to the tracking problem described in Section 4.2.4, with the addition of the minimisation of the camera parameters. Writing the display’s intrinsic projection parameters as a vector p ,

$$p = (f_u \ f_v \ u_0 \ v_0 \ \beta)^T, \quad (5.7)$$

and using a vector μ to update the camera-to-eye transformation with the equation

$$E_{EC|t+1} = \exp \left(\sum_{j=1}^6 \mu_j G_j \right) E_{EC|t}, \quad (5.8)$$

the re-projection error is minimised by solving

$$\begin{bmatrix} J^\mu & J^p \end{bmatrix} \begin{pmatrix} \boldsymbol{\mu} \\ \Delta \mathbf{p} \end{pmatrix} = \mathbf{e} \quad (5.9)$$

where

$$J_{ij}^\mu = \frac{\partial e_i}{\partial \mu_j} \quad J_{ij}^p = \frac{\partial e_i}{\partial p_j} . \quad (5.10)$$

The differentials can be calculated using the chain rule as presented in Appendix B.2.

In contrast to Section 4.2.4 where measurements were significantly non-Gaussian, the user's measurements can be considered less prone to clutter and feature mis-detection and so the error terms here are considered to follow a Gaussian distribution. Hence the pseudo-inverse solution to the above equation is used:

$$\begin{pmatrix} \boldsymbol{\mu} \\ \Delta \mathbf{p} \end{pmatrix} = \lambda \begin{bmatrix} J^\mu & J^p \end{bmatrix}^\dagger \mathbf{e} \quad (5.11)$$

One iteration of the above equation is performed per frame. The scale factor $\lambda \approx 0.1$ slows down convergence, aiding numerical stability and allowing the user to observe convergence in the display. Typically convergence takes less than a second after each new measurement.

5.5.4 Dynamic Registration

Once the calibration parameters have been accurately determined, the display should exhibit correct static registration: virtual graphics should be correctly aligned to the real world when the user does not move. *Dynamic* registration errors can however still occur when the user moves - this is because the user perceives the real world with no delay, whereas the virtual graphics are rendered with some latency. This time difference becomes most apparent when panning, and virtual annotations appear to 'lag' behind the real world.

Many factors contribute to the latency in the augmented visuals, such as the time required to capture a video field and transfer this into computer memory, the time to track this video image, the time required to render an augmented view, and the

time required for this rendered view to appear on the HMD. Most of these delays can be determined experimentally; knowledge of delay times can then be used to lower the apparent latency of the system. Information from the inertial sensor has a lower latency than information from the visual tracking system, and so extra information from the gyroscopes can be used to update the pose which is used for rendering the augmented visuals. Beyond this, a constant velocity model can be used to predict what a user's head pose will be at the time the graphics appear on the screen.

Given a time t_c which corresponds to the time at which the tracking system captured a frame, a time t_g which corresponds to the last available gyroscope measurement, and a time t_d at which the next frame is expected to appear in the display, then $t_c < t_g < t_d$. The matrix $E_{\mathcal{CW}}$ obtained from the tracking system corresponds to time t_c and will be written $E_{\mathcal{CW}|t_c}$. For rendering, this is replaced with a prediction of the pose at display time, $E_{\mathcal{CW}|t_d}$:

$$E_{\mathcal{CW}|t_d} = M_C E_{\mathcal{CW}|t_c} \quad (5.12)$$

with $M_C = \exp(\boldsymbol{\mu})$, and

$$\boldsymbol{\mu} = \begin{pmatrix} (t_d - t_c) \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} \\ \begin{pmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \end{pmatrix} + (t_d - t_g) \begin{pmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{pmatrix} \end{pmatrix}. \quad (5.13)$$

The linear motion components $\mu_1..μ_3$ are hence formed from the tracking system's velocity model (described in Section 4.2.5), whereas the rotational components $\mu_4..μ_6$ are formed in two parts: firstly an integral θ_i of the gyroscope measurements available between times t_c and t_g and secondly a prediction based on $\omega_i|_{t_g}$, the last measurement of rotational velocity made (c.f. Section 4.3). The integrals are as in Eq. (4.20):

$$\theta_i = \int_{t_c}^{t_g} \frac{V_i - b_i}{\alpha_i} dt . \quad (5.14)$$

A trade-off between good registration while panning and increased jitter (cf. Section 5.6.3) can be achieved by artificially moving t_d closer to t_g .

5.6 Results

5.6.1 Maintenance Application

Figure 5.7 shows typical views the user might see when running the maintenance application. These images were recorded by mounting a video camera behind the HMD and then calibrating the set-up through the video camera's viewfinder.

The left column shows the printer application, the right column shows scenes from the computer room. In both cases, CAD models of the scene are drawn in addition to the user instructions: these serve to evaluate registration. When the CAD model is well aligned with the real world, the user can be convinced that the instructional augmentations are drawn in the correct place. In the case of the printer, tracking is robust thanks to adequate lighting and the printer's prominent edges. This sequence also demonstrates the flexibility of visual tracking, in that when the user picks up the printer to examine the switch (third image), the tracking system is capable of tracking this motion. This would e.g. not be possible with magnetic tracking, unless the printer were also magnetically tracked.

The printer is tracked well as long as the user keeps it in the tracking camera's field-of-view: as soon as the user turns away from the printer far enough, tracking fails and must be re-initialised. However, manual re-initialisation when the printer is a working distance away is rapid.

The computer room sequence is more problematic. Subdued lighting and a large amount of clutter relative to the CAD model make tracking less stable. Any tracking failure requires the user to reset the tracking system which involves a trip back to the starting position, since a graceful failure recovery mechanism is not available.

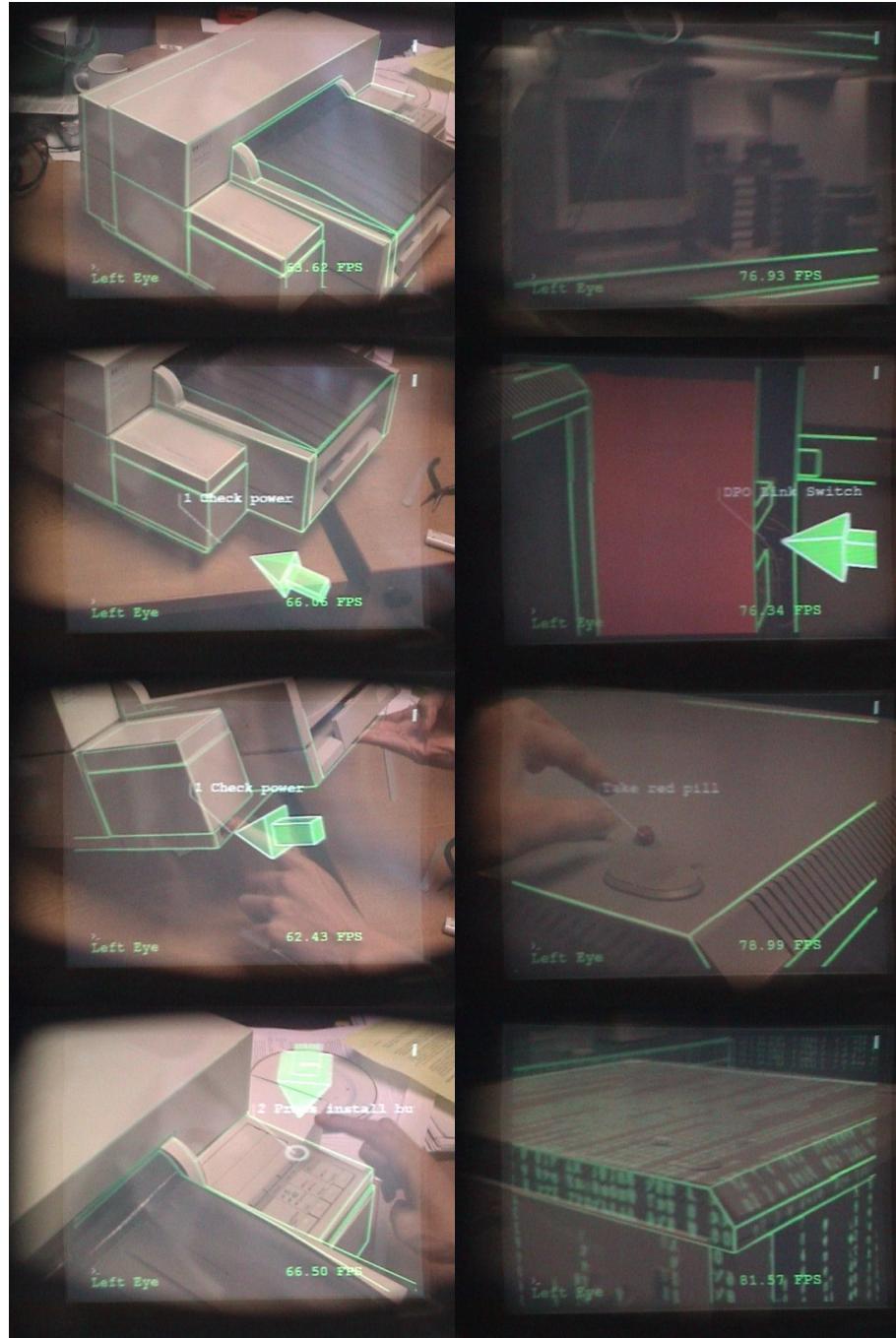


Figure 5.7: Captures from a video recorded through the HMD running the two example applications. Left column shows the printer maintenance application, right column the machine room.

Besides suffering from robustness issues, registration is affected at close range: the networking hardware to be manipulated is not modelled and tracked well enough to reliably highlight the correct network port; only the rectangular outline of the switch is used for tracking, and the ports are spaced at 1cm intervals.

The moderate lighting conditions in the machine room also very negatively impacts the usability of the display - the darkening effect of the see-through optics makes it difficult to clearly see the real world. Further, the constrained augmented field-of-view becomes very limiting in this rather cramped space: the network switch to be modified can only be seen in its entirety when standing at the other end of the room.

The network maintenance application was therefore abandoned and replaced with a simple entertainment demo modelled after the film ‘The Matrix’: the user is instructed to pick up a red pill, after which a matrix-like effect is projected on the walls and surfaces of the scene (bottom two pictures in Figure 5.7.)

5.6.2 Calibration Performance

A systematic evaluation of calibration performance is difficult to achieve since only the user can evaluate the final results; however a numerical measure of the quality of the static calibration can be obtained from the RMS re-projection error of all calibration points: ideally this error would be zero. This error was evaluated by performing eight stereo calibrations (sixteen monocular calibrations) using the test model shown in Figure 5.6. On average, each monocular calibration involved twelve measurements. The RMS residual error for all measurements is shown in Table 5.1.

To evaluate the effect of including the radial distortion term in the projection model, each calibration was also re-calculated without radial distortion. This resulted in a residual error which was on average greater by 0.7 pixels.

Subjectively, registration errors with distortion enabled appeared smaller than 1cm when viewing the object from a distance of 50-150 cm. With distortion disabled,

monocular registration errors were similar, however the effect on stereo perception was significant; graphics often appeared to float a few centimeters in front of the real object. This is supported by the numerical calibration results. On average, removing radial distortion from the projection model caused calibrated left-right eye separation to increase by 3mm, and also caused calibrated eye centers to move 1.8cm further forward.

Distortion Enabled	Yes	No
Mean Reprojection Error(pixels):	3.4	4.1
Rendering Speed (Frames/sec/eye)		
GF2MX Simple scene:	40	83
GF2MX Textured scene:	27	49
GF5900 Simple scene:	149	342
GF5900 Textured scene:	131	263

Table 5.1: Effect of radial distortion on calibration and rendering performance

5.6.3 Dynamic Registration Error

The full AR system, implemented on a 2.4GHz dual-Xeon machine with an nVidia Geforce 5900 video card, has a latency of 60-70ms, i.e. the virtual graphics lag 60-70ms behind the user's head motion. The rate gyroscopes can be used to reduce this latency by 20ms. The remaining 40ms of latency can be masked by prediction with a velocity model - this accurately aligns real and virtual imagery during panning, but increases jitter when the user's head is stationary.

The increased jitter is tabulated in Table 5.2, which measures RMS jitter of a single

Prediction used	Display Jitter (pixels)	
Lens used:	4.2mm	8.5mm
Tracking only:	1.5	0.9
Up to last gyro measurement (20ms:)	2.0	1.3
Full prediction (20+40ms:)	7.0	6.8

Table 5.2: Rendering jitter in the HMD while stationary

printer vertex rendered to the HMD over the course of 100 frames with the tracking camera held stationary 70cm from the printer. Jitter is measured for two tracking lenses and for three prediction strategies: One renders directly according to the latest tracking system pose; the next renders up the last available gyroscope measurements (20ms); the third predicts another 40ms beyond this. The results show a marked increase in jitter for the latter case: at these levels, jitter becomes very apparent to the user, however lag when panning disappears. A compromise could be to modify the prediction behaviour depending on the user's movements: when the user's head is not moving rapidly, prediction could be turned down.

Table 5.2 further compares jitter for the 4.2mm wide-angle lens typically used for tracking and an 8.5mm zoom lens used in previous systems which do not use radial distortion models e.g. **Drummond & Cipolla** (1999). The 4.2mm lens is used here to increase tracking robustness as detailed in Section 4.2. The disadvantage of using the wide-angle lens is a reduction in the tracking system's angular resolution. Table 5.2 shows that this increases jitter in the HMD. However, jitter from the tracking system is small compared to jitter introduced by prediction, and the advantages of using the wide-angle lens for tracking outweigh the slight increase in jitter.

Holloway (1995) has argued that correcting for radial distortion found in AR displays can have a net negative effect on registration, in that the static registration improvements gained by modelling radial distortion can be outweighed by the extra latency incurred. Table 5.1 shows how rendering performance changes when distortion is enabled, using two different graphics accelerators: an nVidia Geforce 2MX card and an nVidia Geforce 5900 card. With the older 2MX card, enabling distortion correction drops the frame-rate to under the display's maximum rate of 42.5 frames/second/eye; The distortion step contributes up to 8ms of latency to the system. The more modern card requires under 2ms of extra time per frame to perform the distortion correction. This cost is already small compared to the total system latency, and produces no measurable increase in jitter. Further, the cost will only decrease in future as graphics accelerators improve fill-rates; one may conclude that compensating for radial distortion no longer significantly impacts dynamic registration.

5.6.4 Ergonomic Issues

The HMD AR system has a number of ergonomic weaknesses and limitations which become apparent as the display is worn.

Bulk: The hard-hat with HMD, gyros and camera attached is uncomfortably bulky and heavy. The hard-hat must be done up tightly to prevent motion of the display during a session - this can cause discomfort.

Restricted view: The HMD offers a very limited field-of-view (30°) for the virtual graphics, which limits the system's usefulness when the user is close to the object being manipulated, or in immersive environments such as the machine room.¹ Further, the user's view of the real world is restricted by the HMD's frame and perceived contrast is reduced (c.f. Figure 5.3.)

Alignment sensitivity: Very small motions of the HMD relative to the user's eyes can cause large shifts in the virtual imagery. This is a problem for longer-term use of the display, as the hard-hat will shift on the user's head. The display may then need to be re-calibrated.

Eye Strain: The reduced contrast of the user's view of the real world, combined with imperfect stereo calibration and the fact that the real and virtual graphics are not always in focus together, can put strain on the user's eyes.

Tether: A large number of cables run from helmet to PC and this restricts the user's movements. The current implementation tethers the user to a desktop workstation, thus limiting the working space. It is possible to replace the workstation with a laptop;

¹Watson & Hodges (1995) point out that early binocular displays had horizontal fields-of-view of over 100 degrees; however, these displays had low resolution and also produced large distortions, which were not corrected at the time. As a result, manufacturers decreased field-of-view to avoid the resolution and distortion problems. The authors hypothesised that since distortion could now be corrected with their approach, manufacturers could revisit wide-FOV displays; unfortunately this does not appear to have happened.

however the need for stereo graphics mandates a workstation-class laptop, and these are typically heavy.

These HMD-related factors all contribute to the HMD being uncomfortable for long-term use. Further, successful use of the system is complicated by the inability of the tracking system used to automatically re-initialise after failure. The need to manually re-align the system after every tracking failure makes the system impractical for use outside laboratory conditions.

6

Tablet-Based Augmented Reality

6.1 Introduction

While the traditional delivery medium for AR has been the head-mounted display (as used in the previous Chapter) tethered to a workstation or laptop, the emergence of powerful PDAs and tablet PCs potentially provides an alternative medium particularly well suited for interactive AR applications. In these devices, the display, user-interface and processor are united in one compact device which can operate without trailing wires or a bulky backpack. Rather than augmenting the user's view of the world directly, they act as the viewfinder for a video camera and operate by augmenting the video feed as it is displayed (in the same manner as video feed-through HMD systems.) The great advantage is that here, small latencies do not matter. Further, no user calibration is necessary; any user can pick up the system and instantly make use of the application. Finally, a pen offers a highly intuitive user interface for interacting with AR applications.



Figure 6.1: Maintenance application on the tablet PC (montage with screenshot.)

While PDAs offer a very compact and unobtrusive form-factor, the video bandwidth and processing power they offer is unfortunately still rather limited. Tablet PCs on the other hand now offer the performance required for AR systems such as presented here; for example, the maintenance application which was used with the HMD in Chapter 5 can operate at full frame-rate completely on the HP Compaq TC1100 tablet PC illustrated in Figure 6.1. This device uses a 1GHz ULV Pentium-M processor. A 10" screen with 1024x768 pixels is driven by an NVidia Geforce4 420 Go graphics accelerator. Video input is provided by a unibrain Fire-i fire-wire camera which is attached to the back of the tablet and fitted with a wide-angle lens. This provides colour 640×480 pixel video at 30 frames per second.¹

Tablet-based AR is an emerging field with its own unique set of challenges which

¹Unfortunately most PCMCIA firewire cards provide no power to the camera, necessitating the use of power from the USB port and a replacement of the camera's power regulator.

must be addressed to produce truly usable systems. This chapter identifies these challenges and attempts to address them with suitable combinations of existing and novel technologies. An AR entertainment application, described in Section 6.2, has been developed for evaluation purposes: this application sets demanding standards for both tracking and rendering. A robust combination of multiple tracking systems including a novel outside-in LED tracker is described in Sections 6.3-6.6. Rendering techniques required to convincingly insert virtual objects into real-world footage are described in Sections 6.8-6.9; in particular, a novel method to refine the occlusion of virtual graphics by real objects is presented.

Details of the application's implementation are presented in Section 6.7, while results and an evaluation of the format's capabilities are presented in Section 6.10.

6.2 A Tablet-based Entertainment Application

This section describes *Darth Vader vs. the Space Ghosts*, an AR entertainment application written to showcase tablet-PC based AR. The application uses a real-world tabletop game environment as seen in Figure 6.2. This environment is a circa 90×90cm model which resembles a Cluedo board and models a single floor of a house. Thin 7cm-high wall-papered walls separate individual rooms and corridors.

The environment is physically empty, but contains virtual characters in the game: these can be seen by observing the environment with the tablet PC, which has a video camera attached to it: in the augmented view seen on the tablet's screen, the virtual characters appear to run around the tabletop environment. The player has control over one such character, Darth Vader: by touching the screen with the tablet's pen, the player can instruct Darth Vader to move to a specific location in the house. Darth Vader can collect virtual bonuses and power-up items which are scattered around the environment.

The world is further populated by Space Ghosts, which emerge from a cellar and at-



Figure 6.2: Tabletop game environment used for “Darth Vader vs. Space Ghosts”

tempt to catch Darth Vader. The player must make Darth Vader avoid contact with these ghosts or virtual lives are lost. To safely clear a level, all the space ghosts must be destroyed: this is accomplished by picking up the *Death Star* bonus and calling in laser strikes from the Death Star above. Once all ghosts have been destroyed, the next level commences with more ghosts.

Calling in a laser strike requires the user's physical interaction with the game world: the user holds a number of tokens (red playing pieces from the game Connect Four) which must be thrown into the game world. A virtual laser strike is then fired at the location the token comes to rest, and any space ghosts in a radius of it are destroyed.

Care is required when picking up power-ups, for these sometimes contain handicaps: For example, more ghosts may appear, or Darth Vader may become a "ghost magnet". Power-ups can be identified only by close examination, which requires the user to physically move the tablet close to the location of the power-up.

6.3 Tracking Strategy

The primary requirement for the correct operation of the application is an accurate and continuous knowledge of the pose of the tablet-mounted camera, so that visuals on the tablet can be drawn correctly.

The TC1100 tablet PC used is sufficiently powerful to run the edge-based tracking system described in Chapter 4 at 30 frames per second (A CAD model of the tabletop environment is used for tracking.) This system in itself provides all the pose and camera model information necessary to directly render virtual graphics onto the live video stream which is displayed on the tablet's screen. However, this system relies on a reasonably accurate prior pose estimate for every frame in order to operate: if such an estimate is not available, tracking fails and cannot be resumed without re-initialisation. Thus tracking can fail after any difficulty encountered, such as when the camera is pointed away from the tracked environment, moved very rapidly, or when



Figure 6.3: Back of the tablet PC showing fire-wire camera and infra-red LEDs

the environment is substantially occluded by the user; in these cases or at application start-up, a manual rough alignment is necessary to recommence tracking.

To increase tracking robustness to the point at which it is usable by un-trained users, a second tracking system operates concurrently with the edge-based tracker. This system uses an external camera which is mounted at the periphery of the playing environment and observes the tablet PC. The camera is attached to a workstation which computes the tablet's pose and sends this information to the tablet over a wireless network. To complement the edge-based tracking running on the tablet, this system should be capable of localising the tablet independently at each frame, i.e. not require a prior pose estimate: for this reason, fiducials are attached to the back of the tablet PC in the form of six small infra-red LEDs, as shown in Figure 6.3. Attaching fiducials to the tablet PC and using outside-in tracking means that it is not necessary to include any fiducials in the game world. The LED tracking system is described in Section 6.4.

Figure 6.4 illustrates the sensors and targets used to determine the tablet's pose. Four coordinate frames are defined in the application: the world coordinate frame of the table-top game environment \mathcal{W} , the tablet-mounted camera's coordinate frame \mathcal{C} , the tablet-mounted LEDs' coordinate frame \mathcal{T} , and the coordinate frame \mathcal{S} of the external camera that observes the LEDs.

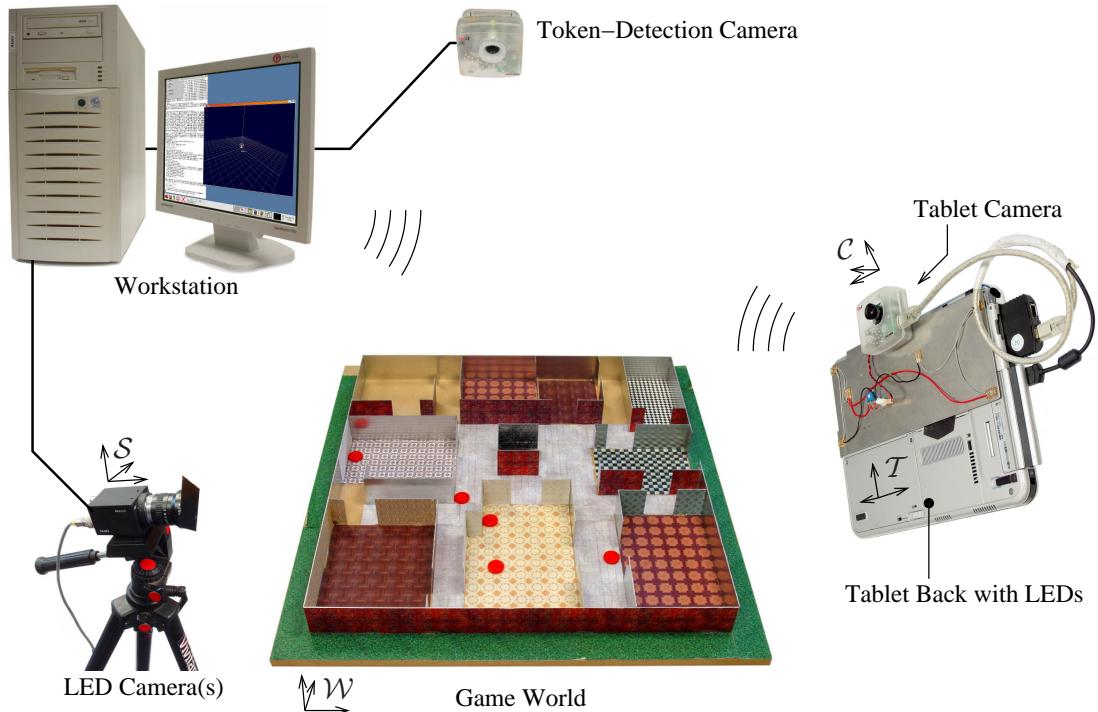


Figure 6.4: Overview of the game application’s sensors and coordinate frames

To combine measurements from inside-out and outside-in tracking in a meaningful way, the workstation runs a statistical filter (an *Extended Kalman Filter* or EKF) which combines measurements from both sensors. For this to be possible, the knowledge of the relative positions of the LED camera to the playing field, and of the tablet-mounted camera relative to the LEDs is required. The EKF and the calibration method employed to determine these coordinate frame transformations are described in Section 6.6. Changes made to the operation of the edge-based tracking system to combine it with the statistical filter are described in Section 6.5.

Besides tracking the pose of the tablet PC, the application also requires detection of tokens which the user throws into the game. This detection is performed by the LED-tracking workstation, which is connected to a ceiling-mounted camera that views the entire playing field. This camera is used solely to detect tokens which land in the playing area. Positions of detected tokens are sent to the tablet via wireless network. The token-detection procedure is described in Section 6.7.2.

6.4 Outside-in LED Tracking

This section describes the outside-in tracking system used to estimate the pose of the tablet PC from an external camera. For robustness and ease of tracking, a fiducial-based approach is chosen; however since the fiducials are in this case mounted to the display device rather than the environment, the aesthetic disadvantages of fiducial-based tracking are avoided. A further advantage of attaching the fiducials to the display device rather than the environment is that the display is powered, and so active fiducials can be used. Six infra-red emitting LEDs are attached to the back of the tablet PC and these are observed by one or more external cameras.

Infra-red LEDs are chosen as fiducials as they have a number of advantages over larger printed markers: their small size makes occlusion by users less likely¹; their size in the image changes insubstantially with distance; finally, they are easily detected in a video image. The monochrome Pulnix TM-500 external cameras used are sensitive to infra-red light, and an infra-red transmissive notch filter can be used to block all visible light, leaving only the LEDs visible - this is illustrated in Figure 6.5. Using this filter, the LEDs can be easily detected and false positive rates (during indoor operation) are negligible.

Localising the tablet from a calibrated external camera requires the system to be able to find four non-collinear points for which the positions on the tablet are known: the correspondence between detected features and fiducials on the tablet must be solvable. The disadvantage of using LEDs is that in contrast to paper markers which can have unique patterns printed in them, LEDs do not easily lend themselves to being individually distinguished by appearance. While it is possible to strobe LEDs to determine their identity (e.g. Welch et al, 1999), this requires information to be merged over many frames, whereas a full pose estimate each frame is desired here. Instead, LEDs are identified based on their relative positions in the image by exploiting the fact that the LEDs are mounted co-planarly to the back of the tablet in known positions.

¹While not attempted here, the small size of the LEDs also makes their inclusion directly into the device's casing a realistic possibility.

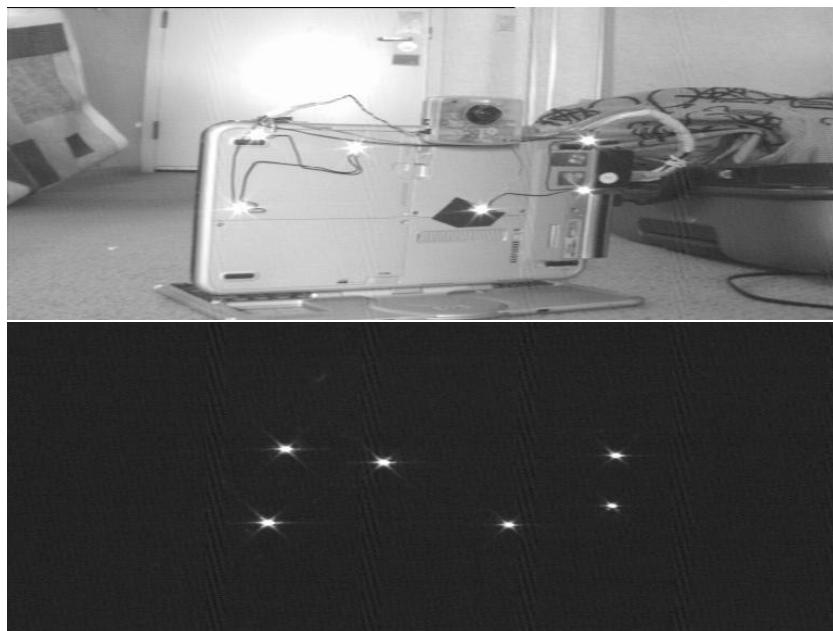


Figure 6.5: Tablet-mounted LEDs viewed through the external camera. Top, view with the LEDs on and no filter; Bottom, the same view when an IR-transmissive filter has been placed before the lens.

Each LED's physical position on the back of the tablet is known in the back-of-tablet coordinate frame \mathcal{T} . The position of the i th LED is $\mathbf{x}_{\mathcal{T}i} = (x_i \ y_i \ 0 \ 1)^T$ (all z -coordinates are zero since the LEDs are mounted in the plane of the tablet.) Figure 6.6(a) shows the six LEDs in frame \mathcal{T} represented as black dots.

The matching procedure used requires an offline training stage which allows fast correspondence at run-time. Permutations of four LEDs are repeatedly selected from the six, and for each permutation a numerical descriptor of this permutation is calculated: At run-time, these descriptors can be used to rapidly determine the identity of the detected LEDs. A single training permutation is shown in Figure 6.6(b). A 3×3 plane-to-plane homography H_T is generated which warps these four LEDs to the homogeneous unit square, as shown in Figure 6.6(c); H_T therefore satisfies

$$\begin{bmatrix} c_1a_1 & c_2a_2 & c_3a_3 & c_4a_4 \\ c_1b_1 & c_2b_2 & c_3b_3 & c_4b_4 \\ c_1 & c_2 & c_3 & c_4 \end{bmatrix} = H_T \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ 1 & 1 & 1 & 1 \end{bmatrix} \quad (6.1)$$

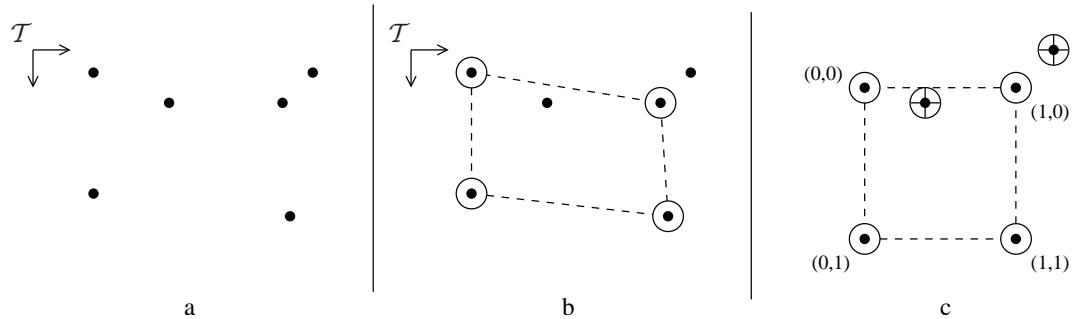


Figure 6.6: Training procedure for LED matching: Of the six tablet-mounted LEDs (a), each permutation of four (b) is warped to the unit square (c) and the warped positions of the remaining LEDs recorded.

with

$$\begin{bmatrix} a_1 & a_2 & a_3 & a_4 \\ b_1 & b_2 & b_3 & b_4 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \quad (6.2)$$

The homography H_T can be found using standard techniques (as described in Appendix D.1.) The remaining two LEDs are also transformed by this homography, as shown in Figure 6.6(c); their transformed positions $[a_5 \ b_5]^T$ and $[a_6 \ b_6]^T$ are stored as keys into a table of all the permutations considered.

For the 360 possible permutation of four from six LEDs, only 60 are considered; there are 15 possible combinations of four LEDs, and for each combination four permutations which form clockwise, planar quadrilaterals are considered.

At run-time, monochrome video images of 768×288 pixels are received at 50Hz. In each image, LEDs are detected using thresholding and flood-fill. The center of mass of each cluster of pixels which pass the threshold forms the detected LED center; the i th center is denoted $(u_i \ v_i)^T$. To establish correspondence, the LED center pixel coordinates are first un-projected into the image plane: this removes the effect of radial distortion caused by the camera lens. Using the same camera model as used for edge tracking in Section 4.2, the i th LED un-projects to the image plane as

$$\begin{pmatrix} \tilde{u}_i \\ \tilde{v}_i \end{pmatrix} = \text{CamUnproject} \begin{pmatrix} u_i \\ v_i \end{pmatrix} \quad (6.3)$$

$$= \begin{pmatrix} \frac{u_i - u_0}{f_u} \frac{r}{r'} \\ \frac{v_i - v_0}{f_v} \frac{r}{r'} \end{pmatrix} \quad (6.4)$$

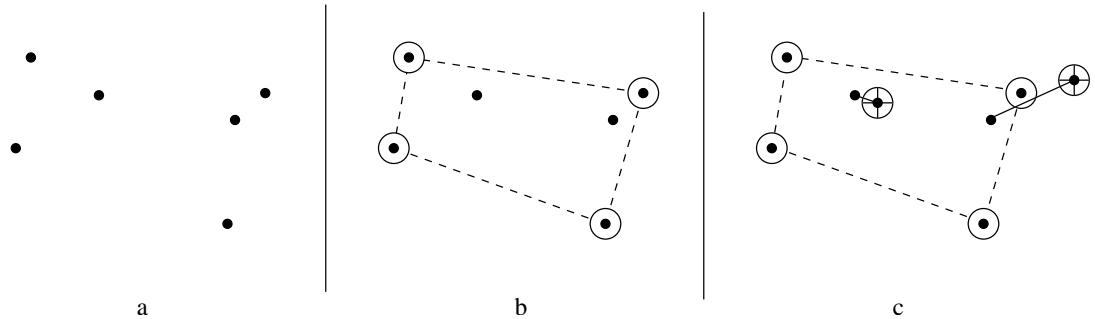


Figure 6.7: Run-time LED matching procedure: Of the six detected LEDs (a), four are selected (b) and used to compare the remaining two to trained permutations (c) to determine their identity.

where u_0 and v_0 are the camera's principal point and f_u and f_v the x- and y-focal lengths. The distorted radius r' is found directly from u_i and v_i as $r' = \sqrt{((u_i - u_0)/f_u)^2 + ((v_i - v_0)/f_v)^2}$; the corresponding un-distorted radius r is found by inverting Equation (4.7) using four Newton-Raphson iterations.

The six LEDs un-projected into the image plane are shown in Figure 6.7(a). Four LEDs which form a clockwise planar quadrilateral are chosen at random: these are highlighted in Figure 6.7(b). A homography H_R which projects the corners of the unit square to the image-plane positions of the four LEDs is calculated, i.e.

$$\begin{bmatrix} w_1\tilde{u}_1 & w_2\tilde{u}_2 & w_3\tilde{u}_3 & w_4\tilde{u}_4 \\ w_1\tilde{v}_1 & w_2\tilde{v}_2 & w_3\tilde{v}_3 & w_4\tilde{v}_4 \\ w_1 & w_2 & w_3 & w_4 \end{bmatrix} = H_R \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}. \quad (6.5)$$

This homography is then applied to the two key LED positions of each permutation considered in the training stage. The two key LEDs are projected into the image plane as illustrated in Figure 6.7(c). If the four LEDs used to create this permutation are the same four LEDs as those selected at run-time, the projections of the two key LEDs are expected to fall close to the image-plane positions of the remaining two LEDs in the image; hence the permutation with the smallest sum-squared error between projected key positions and image-plane detected LED positions yields the identity of the LEDs detected in the image.

Once the LEDs have been identified, the coordinate frame transformation E_{ST} which describes the pose of the tablet back relative to the observing camera can be found. This is done by first calculating a Homography H_P which maps the tablet-back LED positions into the camera's image plane:

$$\begin{bmatrix} w_1\tilde{u}_1 & w_5\tilde{u}_6 \\ w_1\tilde{v}_1 & \dots & w_5\tilde{v}_6 \\ w_1 & w_6 \end{bmatrix} = H_P \begin{bmatrix} x_1 & x_6 \\ y_1 & \dots & y_6 \\ 1 & 1 \end{bmatrix}. \quad (6.6)$$

This homography is used to calculate an initial estimate of the transformation E_{ST} ; this procedure is described in Appendix D.2. This estimate is then refined by optimising the pose estimate to minimise the sum-squared re-projection error $|e|^2$, where

$$e = \begin{pmatrix} u_1 - u'_1 \\ v_1 - v'_1 \\ \vdots \\ u_6 - u'_6 \\ v_6 - v'_6 \end{pmatrix} \quad (6.7)$$

and the reprojected pixel coordinates u'_i and v'_i are found as

$$\begin{pmatrix} u'_i \\ v'_i \end{pmatrix} = \text{CamProj} \left(E_{ST} \begin{pmatrix} x_i \\ y_i \\ 0 \\ 1 \end{pmatrix} \right). \quad (6.8)$$

and E_{ST} is updated by iterating

$$E_{ST|n+1} = \exp(\boldsymbol{\mu}_{\mathcal{S}}) E_{ST|n} \quad (6.9)$$

with

$$\boldsymbol{\mu}_{\mathcal{S}} = J^\dagger e \quad (6.10)$$

$$J_{ij} = \frac{\partial e_i}{\partial \mu_{\mathcal{S}j}} \quad (6.11)$$

where the derivatives J_{ij} are found as in Appendix B. Ten iterations are used for each frame.

To use the pose estimates in the Kalman Filter of Section 6.6, an estimate of the uncertainty of this pose estimate is required. A 6×6 pose covariance matrix Σ_S in the LED camera frame can be found by considering the pose estimate E_{ST} to be the true pose \hat{E}_{ST} corrupted by normally distributed errors in the LED camera's coordinate frame S :

$$E_{ST} = \exp(\epsilon_S) \hat{E}_{ST} \quad (6.12)$$

$$\epsilon_S \sim N(\mathbf{0}, \Sigma_S). \quad (6.13)$$

These errors originate from the estimation of the motion vector μ_S in Equation 6.10; this can be re-written $\mu_S = \hat{\mu}_S + \epsilon_S$. The error originates from the LEDs' detected image positions u_i and v_i , which are assumed to be corrupted by uncorrelated Gaussian noise of one pixel variance:

$$\begin{aligned} u_i &= \hat{u}_i + \epsilon_{ui} \\ v_i &= \hat{v}_i + \epsilon_{vi} \\ \epsilon_{ui}, \epsilon_{vi} &\sim N(0, 1). \end{aligned} \quad (6.14)$$

The error vector e of Equation (6.7) is hence also corrupted by noise of variance I_{12} (the 12×12 identity):

$$e = \hat{e} + \epsilon \quad (6.15)$$

$$\epsilon \sim N(\mathbf{0}, I_{12}) \quad (6.16)$$

The covariance Σ_S can then be found by expectation:

$$\begin{aligned} \Sigma_S &= E[\epsilon_S \epsilon_S^T] \\ &= E[(\mu_S - \hat{\mu}_S)(\mu_S - \hat{\mu}_S)^T] \\ &= E[(J^\dagger e - J^\dagger \hat{e})(J^\dagger e - J^\dagger \hat{e})^T] \\ &= E[(J^\dagger \epsilon)(J^\dagger \epsilon)^T] \\ &= J^\dagger I_{12} J^{\dagger T} \\ &= (J^T J)^{-1} \end{aligned} \quad (6.17)$$

The estimated pose, covariance, and frame time-stamp are transmitted to the Extended Kalman Filter every frame.

6.5 Inside-Out Edge Tracking

Since the LED tracking can continually provide the tablet with reasonable prior pose estimates (even if the edge tracker should fail), the edge tracker on the tablet PC need not be as robust as the tracker running the HMD-based application in Chapter 5: occasional edge-tracking failures will merely produce a brief glitch on the screen rather than require a re-initialisation. Further, the tablet PC, being a piece of computing equipment with moving parts, is unlikely to undergo the same accelerations as a head-mounted camera may encounter. For these reasons, the rate gyroscopes used in Chapters 4 and 5 to tolerate fast rotations are not needed here, and their removal from the system reduces bulk and weight attached to the tablet PC.

The operation of the tracking system is further modified by the use of an EKF to track the tablet's pose. The tracking system's motion model described in Section 4.2.5 is not used; instead, the EKF running on the workstation maintains a motion model and provides the tracking system with a pose prior for every frame. Further, the tablet's posterior from every frame is transmitted to the filter. Besides the current pose estimate, the filter also requires a covariance matrix, that is, an estimate of the amount of uncertainty present in each measurement.

To calculate this, the tracking system's image measurements (i.e. the edge normal distances d) are assumed to be corrupted by noise. Denoting the true normal distances by \hat{d} and assuming independent Gaussian noise of 1 pixel variance, the measurements relate to the true distances as

$$d = \hat{d} + \delta, \quad \delta \sim N(\mathbf{0}, I_N). \quad (6.18)$$

The errors propagate through the Equations (4.9) - (4.14) to produce a noisy motion estimate μ_c , which is related to the true motion $\hat{\mu}_c$ as

$$\mu_c = \hat{\mu}_c + \epsilon_c \quad (6.19)$$

$$\epsilon_c \sim N(\mathbf{0}, \Sigma_c). \quad (6.20)$$

The covariance $\Sigma_{\mathcal{C}}$ which describes the variance of camera pose in the camera reference frame \mathcal{C} can be found using expectation. Treating the estimation as using least-squares,

$$\Sigma_{\mathcal{C}} = \mathbb{E} [\boldsymbol{\epsilon}_{\mathcal{C}} \boldsymbol{\epsilon}_{\mathcal{C}}^T] \quad (6.21)$$

$$\begin{aligned} &= \mathbb{E} [(\boldsymbol{\mu}_{\mathcal{C}} - \hat{\boldsymbol{\mu}}_{\mathcal{C}})(\boldsymbol{\mu}_{\mathcal{C}} - \hat{\boldsymbol{\mu}}_{\mathcal{C}})^T] \\ &= \mathbb{E} [(J^\dagger \mathbf{d} - J^\dagger \hat{\mathbf{d}})(J^\dagger \mathbf{d} - J^\dagger \hat{\mathbf{d}})^T] \\ &= \mathbb{E} [(J^\dagger \boldsymbol{\delta})(J^\dagger \boldsymbol{\delta})^T] \\ &= J^\dagger \mathbb{E} [\boldsymbol{\delta} \boldsymbol{\delta}^T] J^{\dagger T} \\ &= (J^T J)^{-1} \end{aligned} \quad (6.22)$$

Here, the weight matrix W and stabilising prior P used in Eq. (4.15) have been omitted. The prior matrix P is not required since this role is now assumed by the Kalman filter. However, the weight matrix W used to obtain an M-Estimator is used for calculating pose but not covariance. This corresponds to the assumption that all image measurements are inliers which contribute information to the system, and will result in slight under-estimates of covariance.

6.6 Extended Kalman Filter

6.6.1 An Introduction to Kalman Filtering

To combine the measurements from the tablet-mounted camera and any fixed cameras observing the LEDs mounted on the back of the tablet, a Discrete Extended Kalman filter is employed. This section describes the filter and the steps required to combine the two sources of pose information used. The operational principles of the filter are described only as an introduction, and the description draws on more detailed works on the subject (Kalman, 1960; Maybeck, 1979; Welch & Bishop, 1995); in particular, the notation used here is loosely based on Welch and Bishop's tutorial.

An Extended Kalman Filter estimates the state of a system as time passes. The state estimate is based on noisy measurements which can be made from the actual system, and on a model of the system's dynamics which is used to predict the behaviour of the system with passing time. Central to a Kalman filter is it's model of uncertainty: Errors in measurements and in the current estimate of system state are all modeled as multivariate Gaussian distributions.

The propagation of a Kalman Filter's state can be described as a *Predictor-Corrector* cycle (Welch & Bishop, 1995). This is illustrated by a simplified 1D example in Figure 6.8, which shows a Kalman filter tracking the position of an object with time. An estimate of the object's position at time t is represented as a Gaussian PDF shown in Panel A. To estimate the position of the object at time $t+1$, the mean of this PDF is transformed by a motion model, which in this case predicts motion to the right; at the same time, since system inputs are not known, they are modeled as Gaussian *process noise* and hence the uncertainty in the object's position increases with time. This wider a-priori PDF for the object's position at time $t+1$ is shown in Panel B. The application of the motion model and increase in state uncertainty is the predictor stage of the Kalman filter.

At time $t+1$, a measurement of the system state is made; this measurement yields a Gaussian likelihood density function for the object's position as illustrated in Panel C. By combining the measurement with the prior position estimate, an a-posteriori estimate of the object's position can be formed (Panel D); the uncertainty of this estimate is now lower than that of either the prior or the measurement individually. The acquisition of measurements and the associated reduction of state uncertainty is the corrector stage of the Kalman filter. The filter continually loops through prediction and correction as time advances.

6.6.2 Filter State

The filter used for tracking tablet pose tracks a 12-DOF system state x which contains tablet camera pose (6-DOF) and velocity (6-DOF). The classical EKF formula-

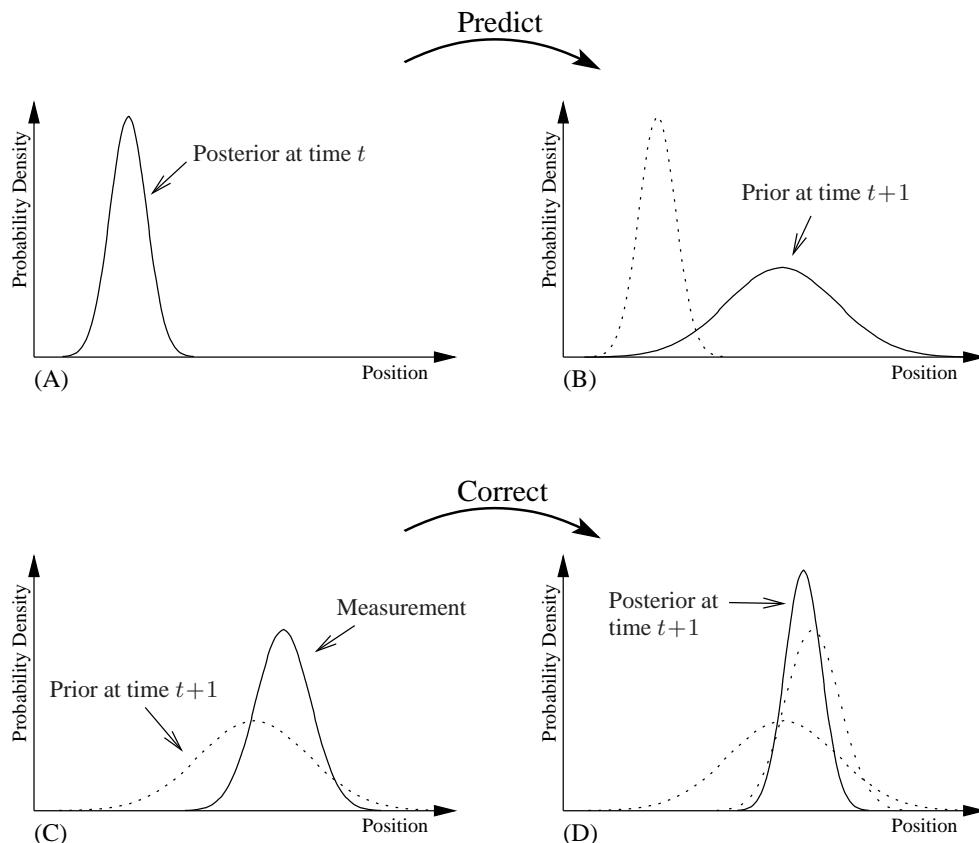


Figure 6.8: Predictor-Corrector cycle of the Kalman Filter

tion would use a 12-dimensional *state vector* to encode filter state in a fixed reference frame; in the application presented here, this could be done by storing pose and velocity relative to the world coordinate frame \mathcal{W} as two 6-vectors, with pose stored in its logarithm form. Covariance could similarly be expressed in the world coordinate frame.

However, this representation would introduce needless complications to virtually all calculations performed by the filter. Instead, the implementation here forgoes a fixed reference coordinate frame and performs all calculations relative to the tablet-camera frame \mathcal{C} . Filter state x is not stored as a 12-vector, but is instead represented as a transformation matrix $E_{\mathcal{C}\mathcal{W}}$ and a velocity 6-vector $v_{\mathcal{C}}$ in coordinate frame \mathcal{C} . Prior and posterior covariances are also expressed in tablet-camera coordinate frame \mathcal{C} .

While this is not the most compact representation of the state possible, it effectively linearises the filter about the current pose estimate and simplifies further calculations. (The representation used here is equivalent to operating a classical EKF with state stored as a 12-vector, where the first six elements of this vector represent the logarithm of camera pose in frame \mathcal{C} , i.e. with the first six elements of the state vector set to zero at all times.)

Thus, for time t , the filter's estimate x_t of the true (and hidden) system state \hat{x}_t is

$$x_t = \{E_{\mathcal{CW}|t}, \mathbf{v}_{\mathcal{C}|t}\}. \quad (6.23)$$

The state estimate relates to the true state as

$$\begin{aligned} E_{\mathcal{CW}|t} &= \exp(\boldsymbol{\epsilon}_{\text{pose}}) \hat{E}_{\mathcal{CW}|t} \\ \mathbf{v}_{\mathcal{C}|t} &= \boldsymbol{\epsilon}_{\text{vel}} + \hat{\mathbf{v}}_{\mathcal{C}|t} \end{aligned} \quad (6.24)$$

The state error 12-vector $\boldsymbol{\epsilon}_{\mathcal{C}|t}$ at time t is thus

$$\boldsymbol{\epsilon}_{\mathcal{C}|t} = \begin{pmatrix} \boldsymbol{\epsilon}_{\text{pose}} \\ \boldsymbol{\epsilon}_{\text{vel}} \end{pmatrix} \quad (6.25)$$

and is modeled as normally distributed

$$\boldsymbol{\epsilon}_{\mathcal{C}|t} \sim N(\mathbf{0}, P_t). \quad (6.26)$$

where P_t is the filter's state error covariance at time t .

6.6.3 Prediction Step

Pose predictions across a time interval δt from a previous state are made using a constant-velocity dynamic model, with acceleration modeled as zero-mean independent normally distributed process noise. Denoting prior estimates by the minus superscript ($-$), prior estimates of future state are given by the the time update function $f()$. This is given here sans unknown noise and driving function but parametrised by elapsed time δt :

$$\tilde{x}_{t+\delta t} = f(x_t, \delta t) = \{\exp(\mathbf{v}_{\mathcal{C}|t} \delta t) E_{\mathcal{CW}|t}, \mathbf{v}_{\mathcal{C}|t}\}. \quad (6.27)$$

The corresponding prior state covariance $\tilde{P}_{t+\delta t}$ is found by transforming the previous covariance by the time-update Jacobian A , and by adding process noise:

$$\tilde{P}_{t+\delta t} = AP_tA^T + \sigma_p^2 \begin{bmatrix} 0 & 0 \\ 0 & I_6 \end{bmatrix} \quad (6.28)$$

where σ_p is the system's process noise (acceleration) parameter and A takes the form

$$A = \begin{bmatrix} I_6 & \delta_t I_6 \\ 0 & I_6 \end{bmatrix} \quad (6.29)$$

in accordance with the system dynamics.

6.6.4 Correction Step

Measurements (denoted by the superscript m) from the tracking systems are in the form of pose estimates of the transformation $E_{\mathcal{CW}}^m$ with measurement covariances $\Sigma_{\mathcal{C}}^m$. To integrate information from pose measurements into the filter, measurements $E_{\mathcal{CW}}^m$ are converted to an *innovation motion* which describes the motion from the filter's corresponding prior state to the pose described by the measurement:

$$M_{\mathcal{C}} = E_{\mathcal{CW}|t+\delta t}^m \tilde{E}_{\mathcal{CW}|t+\delta t}^{-1} \quad (6.30)$$

Further, the measurement's covariance $\Sigma_{\mathcal{C}}^m$ is transformed into the filter's reference frame and used to compute the Kalman gain K . Dropping the subscript $t + \delta t$,

$$K = P^T H^T \left(H P^T H^T + \Sigma_{\mathcal{C}}^m \right)^{-1} \quad (6.31)$$

where the measurement interaction matrix takes the value $H = [I_6 \ 0]$ indicating that only pose and not velocity are measurable. The a posteriori pose estimate is found by weighting the innovation motion by the Kalman gain and applying the result to the prior pose:

$$E_{\mathcal{CW}} = \exp(K \log(M_{\mathcal{C}})) \tilde{E}_{\mathcal{CW}} \quad (6.32)$$

and the posterior covariance is appropriately reduced:

$$P = (I_6 - KH)\tilde{P}. \quad (6.33)$$

6.6.5 Sensor Offset Calibration

Since the inside-out tracking of Section 6.6 observes the table-top maze (\mathcal{W}) through the tablet-mounted camera (\mathcal{C}), it produces pose measurements of the form $\{E_{\mathcal{CW}}, \Sigma_c\}$, which can be directly filtered as described above. On the other hand, the LED tracking system described in Section 6.4 observes the tablet-mounted LEDs (\mathcal{T}) with an external camera (\mathcal{S}) and hence produces pose estimates of the form $\{E_{\mathcal{TS}}, \Sigma_T\}$; for these measurements to be filtered they must first be transformed into the filter's coordinate frame \mathcal{C} .

For this purpose, knowledge of the transformations E_{CT} (Tablet camera from tablet back) and E_{SW} (LED tracking camera from world) is required. Providing the tablet camera is rigidly attached to the tablet and the sensor camera rigidly mounted in the world, these transformations may be considered fixed and need be calibrated only once. Although the transformations can not be directly measured, they can be calculated by observing the changes in the two sensor measurements. Chaining together transformations,

$$E_{SW} = E_{ST} E_{TC} E_{CW}. \quad (6.34)$$

The transformation E_{SW} is unchanged by inserting an observed motion in the tablet camera frame M_C and a simultaneously observed motion in the tablet back frame M_T .

$$E_{SW} = E_{ST} M_T E_{TC} M_C E_{CW} = E_{ST} E_{TC} E_{CW} \quad (6.35)$$

Canceling and re-arranging,

$$\begin{aligned} M_T E_{TC} M_C &= E_{TC} \\ M_T E_{TC} &= E_{TC} M_C^{-1} \end{aligned} \quad (6.36)$$

Baillot et al (2003) have recently identified this problem as one studied in robotics as $AX = XB$. **Park & Martin** (1994) present a closed form solution which estimates E_{TC} from two or more sets of motion measurements, and this method is also used here. Once E_{TC} (and thus, simultaneously, E_{SW}) has been obtained, measurements from

LED tracking can be transformed into the tablet camera frame

$$\begin{aligned} E_{\mathcal{CW}}^m &= E_{\mathcal{TC}}^{-1} E_{\mathcal{TS}} E_{\mathcal{SW}} \\ \Sigma_{\mathcal{C}}^m &= \text{Adj}(E_{\mathcal{TC}}^{-1}) \Sigma_{\mathcal{T}} \text{Adj}(E_{\mathcal{TC}}^{-1})^T \end{aligned} \quad (6.37)$$

and so measurements from both inside-out and outside-in tracking are accommodated. This completes the equations required for implementation of the filter.

6.7 Application Implementation

This section describes some of the engineering aspects of the AR application implementation.

6.7.1 Kalman Filtering over the Network

Filtered estimates of the tablet's pose are primarily required by the inside-out edge-based tracker running on the tablet PC. However, the Kalman filter is implemented on the LED-tracking workstation. The two machines are connected by an 802.11b wireless network so that the communication of measurements and pose updates is possible. Aggressive NTP (Network Time Protocol) polling is used to ensure the clocks on the two machines are synchronised to sub-millisecond accuracy.

The tablet PC timestamps and transmits every measurement it makes to the Kalman filter, which also receives local measurements from the LED tracking system. Due to transmission delays, tablet measurements often arrive out-of-sequence in relation to LED measurements; a roll-back history of the filter is kept so that measurements arriving late can be inserted into a previous filter state and subsequent measurements re-applied.

To avoid a filter-query round-trip delay when the tablet needs a position prior for a new frame, the time-stamped 12-DOF state of every posterior calculated by the filter



Figure 6.9: Token detection. Tokens are recognisable by their colour and shape in the feed (shown here) from an overhead camera. Colour templates, shown in the top left, can be selected in the image to calibrate the system.

is broadcast to the tablet. From the latest posterior received, the tablet can trivially compute a prior appropriate to an incoming frame by applying a constant-velocity motion model.

6.7.2 Token detection

To detect tokens thrown into the game, the workstation employs a unibrain Fire-i firewire camera which is mounted on the ceiling above the maze. This camera delivers colour images of the playing field to the workstation at 30Hz. The tokens used in the game are red “connect-four” game-pieces; these can be identified in the video feed simply by their colour and approximate shape.

Detection of a new token is transmitted over the wireless link to the game running on the tablet PC: this indicates that a token has come to rest in a certain spot on the playing field, and a laser beam attack is then activated at that position. Therefore, tokens which are in motion (in mid-air, bouncing or rolling) should not be transmitted

to the application. The application ignores moving tokens by recording the positions of all tokens found in a video frame and by then checking that these are still present in the next video frame; if a token has moved less than a small threshold number of pixels, its position in the playing field is calculated using a plane-to-plane homography and transmitted to the tablet. This strategy can support very large numbers of tokens present on the board at once, and large numbers of tokens thrown into the game simultaneously (in practice the number is limited to ten by the rendering speed of the tablet PC rather than the detection process.)

Further, the positions of tokens which have already been transmitted to the application are stored so that if a user should occlude tokens during the course of the game, these are not registered as new tokens as soon as the occlusion passes. Token locations are cleared after completion of a game.

6.8 Rendering

The raison d'être of the tracking systems described in the previous sections is to provide a pose estimate with which virtual graphics can convincingly be rendered into the video feed provided by the tablet-mounted camera. The graphical compositing capabilities of the tablet PC are vastly superior to that of the HMD-based system of Chapter 5; while stereo display is not possible, graphics can be rendered at higher resolution and colour fidelity, and (most importantly) the system is not limited to additive mixing. Virtual objects can occlude real ones and can be drawn in many nuanced colours and in high detail (as opposed to high-visibility bright green outlines as used for the HMD.)

The disadvantage of this level of detail is that small rendering mistakes which would not be noticed in the HMD become apparent and even jarring on the tablet's screen. This section presents the standard graphical rendering algorithms used for the tablet's display. In Section 6.9 these will be shown to be lacking in areas where real objects

occlude virtual characters, and so an algorithm to refine the occlusions of virtual characters by real objects in the scene is described there.

The game running on the tablet PC operates a track-render loop; each frame arriving from the tablet-mounted camera is processed as follows:

1. **Colour-space conversion:** incoming video data is in YUV411 format. This is converted to RGB and greyscale prior to further processing.
2. **Prior calculation:** a pose prior for the current frame is calculated from the last received filter state.
3. **Edge-based tracking:** the tracking system of Section 6.5 is run on the greyscale image data.
4. **Rendering:** The current game-state and current video frame are rendered to the tablet's screen according to the calculated pose.
5. **Game mechanics:** User input is processed and the game state updated.

A typical resulting frame is shown in Figure 6.10, which is a screen-shot from the game application running: the virtual characters of Darth Vader and the two space ghosts as well as various other items appear correctly placed in the scene.

Rendering is performed using OpenGL, which is hardware-accelerated by the nVidia GeForce4 420 Go processor. The full screen with a resolution of 1024×768 pixels is used. The rendering loop used for each frame is illustrated in Figure 6.11.

As with the HMD application of Section 5.4, visuals are not directly rendered with radial distortion. Instead, rendering is initially performed using an undistorted pinhole-camera model; the rendered image is subsequently warped to match the camera lens' distortion. This approach requires the use of a visual buffer with destination-alpha support (that is, the alpha values of rendered OpenGL fragments are recorded). At the beginning of each frame, the screen is cleared to a transparent background with a



Figure 6.10: A scene from Darth Vader vs. the Space Ghosts

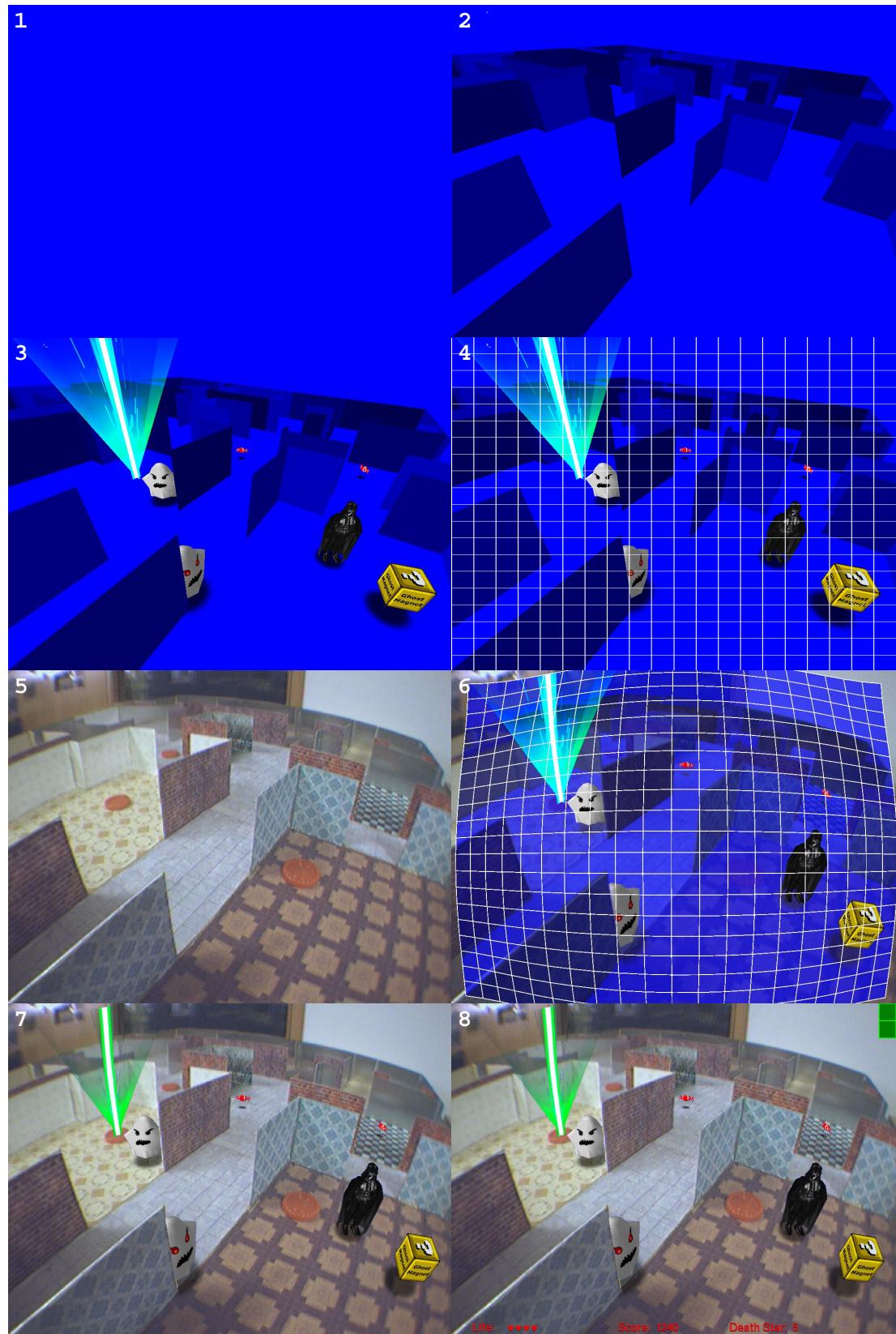


Figure 6.11: Rendering Loop using z-Buffering

distant z-value. This is illustrated in Panel 1 of Figure 6.11; in the illustration, a blue-screen analogy is used to indicate transparency (all blue objects are actually transparent in the application.)

Next, the real-world walls of the game world are rendered into the scene, again using a transparent colour (Panel 2) and the camera pose estimated by the tracking system. This has the effect of filling the graphic card's z-Buffer with the positions of these walls, which should hide any virtual objects located behind them.

When the z-buffer has been populated, game characters and items are rendered. Game characters are stored in id Software's .md2 file-format, which is widely documented and for which rendering code is available¹; other game items such as laser strikes are simply described by their OpenGL primitives. All game characters² and items are rendered using z-buffering so that portions hidden behind walls are not drawn. A semi-transparent circular shadow is rendered under each character or object to increase its apparent integration in the real world. The rendered characters appear in Panel 3 of figure 6.11.

The resulting image, which consists of clipped game objects against a transparent background, is now uploaded to a texture map (this process is represented by the grid in Panel 4). The current video frame is now rendered to fill the screen as shown in Panel 5.

To insert the game objects into this video image, the stored texture map is rendered using a distorted 20×20 grid of quadrilaterals, as illustrated in Panel 6. The grid is distorted by applying the effects of the camera's radial distortion to the un-distorted coordinates of the sampling grid (Panel 4) according to the procedure described by **Watson & Hodges** (1995). The blue portions in Panel 6 illustrate the transparent sections of the texture, so only the game objects appear in the composited image (Panel

¹A tutorial software implementation by DigiBen at <http://www.gametutorials.com> was until recently freely available but now requires purchase. This implementation is used here.

²The Darth Vader model used for the tablet AR application was created by Gilbert "Arcor" Arcand and is available at <http://www.planetquake.com/polycount/cottages/alcor/>. All other graphics are bespoke.

7), to which some user interface elements (such as status text and controls to interact with the tracking system) are added (Panel 8) to complete the rendered image.

6.9 Occlusion Refinement

In augmented reality applications such as the game presented here, virtual objects and the real world co-exist in close proximity. To appear believable, these objects should not only be well registered with the real world, they should also occlude real objects behind them, and be occluded by real objects in front. The accuracy of this occlusion greatly affects the user's perception that the virtual object belongs in the scene in the sense that occlusion errors are instantly recognisable and destroy the scene's believability.

The standard approach for handling occlusion is to use z-buffering. By populating the z-buffer with an estimate of the real world's depth, occlusion of the subsequently rendered virtual objects is automatically handled by the rendering hardware. The z-buffer can be filled with information generated from a stored 3D model and a pose estimate as done in Section 6.8; alternatively, data generated on-line can be used (e.g. **Wloka & Anderson** (1995) use depth from stereo.) Whichever method is used, the values in the z-buffer will occasionally not correspond to the real depth in the scene, and occlusion errors will occur.

Considering only those systems which assume knowledge of the occluding real geometry, the most obvious source of error is an inaccurate model of this geometry. However, even if the model is accurate, tracking errors (or jitter) or incorrect projection parameters can produce noticeable occlusion errors in the image. This is particularly true of systems in which the tracked feature and the occluding geometry are some image distance apart: in this case, any small rotational tracking error produces an amplified occlusion error.

By tracking the visible edges in the scene to obtain pose, the application presented here

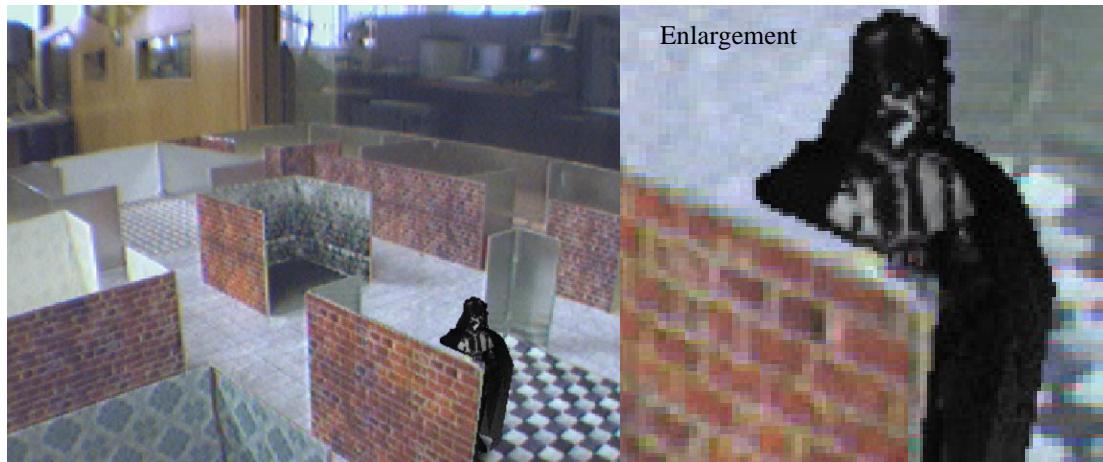


Figure 6.12: Occlusion error when using z-buffering to clip virtual characters

is also optimising for those features which cause occlusion and this goes a long way to providing a good level of realism. However, the system still produces substantial occlusion errors on occasion.

Figure 6.12 shows an example in which the occluding wall has been rendered into the z-buffer too high, so that too much of the virtual character is occluded. Even though the position of the occluding wall was measured during tracking, it is drawn in the position which best fits *all* measured edges. To solve this problem, an approach has been developed which optimises the location of an occluding edge using measurements from that edge only.

To achieve this, the rendering process described in the previous section must be modified. The new procedure is illustrated in Figure 6.13. Z-buffering is no longer used to determine object-world occlusions; instead, each character is initially rendered unoccluded (Figure 6.13, Panel 1) onto the transparent background.

Using software edge-based rendering techniques, the viewing frustum which contains the rendered character is intersected with the game-world geometry to produce a clipping polygon (Panel 2) which corresponds to the portions of the game world which should occlude the virtual character. This polygon can be used to remove hidden por-

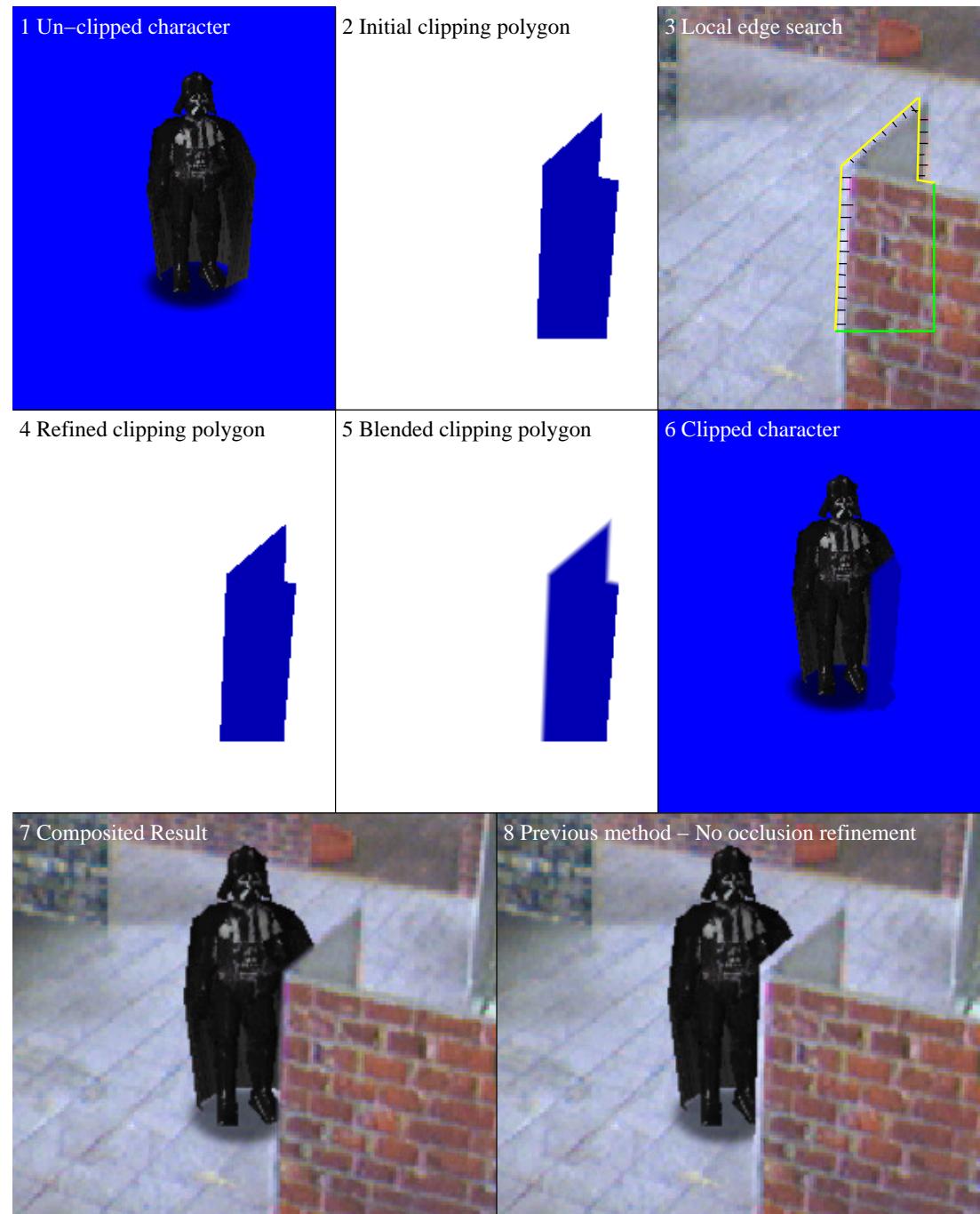


Figure 6.13: Occlusion refinement procedure

tions of the character by drawing it over the rendered character using a transparent colour¹ (illustrated as blue.) The result of clipping with this polygon would be identical to the results of the z-buffering approach described in the previous section.

Each individual occluding edge of this clipping polygon is now refined (Panel 3) by analysing the video input: Closely spaced sample points (akin to those used for tracking in Chapter 4) are initialised along the edges, and a perpendicular search for the nearest image edge is performed in the video image. A two-DOF optimisation adjusts the endpoints of each edge to best match the nearest detected image edge. The resulting refined clipping polygon is shown in Panel 4.

If the virtual character were clipped with this refined polygon, occlusion errors would be reduced; however, the resulting edge between virtual and real graphics can have an unnaturally sharp appearance. In the video image, occluding regions of geometry produce edges which are somewhat blurred due to the camera's optics and sampling. The pixel-precise clipping of virtual characters looks harsh by contrast (even despite the slight blur introduced by the radial distortion correction.) Furthermore, near-vertical or near-horizontal clipping lines are prone to producing "crawling jaggies" artifacts with slight camera motion or tracking jitter.

To make occluding boundaries appear more seamless, the clipping polygon's occluding edges are transformed into thin (the equivalent width of two video image pixels) alpha-gradated quadrilaterals as illustrated in Panel 5. The alpha-values of this blended, refined polygon are now used to over-write² alpha values of the rendered character. The resulting clipped character is visible in Panel 6.

The occlusion refinement procedure is repeated for each virtual object inserted into the scene. Z-buffering can still be used so that virtual objects properly self-occlude. Once all characters and objects have been clipped, the frame-buffer is copied to a texture and the AR rendering cycle continues as in the previous section. Panels 7 and 8 of

¹OpenGL channel masks are used so that only the alpha channel is modified.

²Suitable blending is used to preserve transparencies already present, as e.g. in the shadows or the transparent background.

Figure 6.13 compare the resulting composited images: Panel 8 shows the character incorrectly clipped using z-buffering, while Panel 7 shows the character clipped using the occlusion refinement procedure.

6.10 Results

6.10.1 Real-time Performance

The full AR occlusion-refining track-render system described in this chapter operates at the full video frame-rate of 30 frames per second on the tablet PC. CPU usage on the tablet registers at 70%. On the workstation, LED tracking, filtering and coin detection run at full frame-rate and occupy a total of 15% of processor time.

The occlusion refinement code requires approximately 1ms of execution time per object to be refined, however this can vary greatly with viewpoint (and hence the complexity and size of clipping polygons). This limits the number of objects which can simultaneously be drawn on-screen while maintaining full frame-rate to approximately 10.

6.10.2 Errors

Tracking jitter was evaluated by keeping the tablet static and observing the noise on incoming measurements. Typical RMS jitter values for the edge-based and LED tracking are tabulated in Table 6.1. Tracking jitter reduces the apparent registration of real and virtual objects, with virtual objects appearing to wobble on a static background.

	Edges Trans/	Rot	LEDs Trans/	Rot
Jitter (mm/deg)	1.1	0.17	5	0.5
σ (mm/deg)	1.0	0.15	8	2.85

Table 6.1: Tracking jitter compared to estimated standard deviation

The observed jitter of the edge-based tracking agrees with the expected error. The LED measurements yield lower observed jitter than the expected error. This is likely because the LED centroids in the image are extracted to sub-pixel accuracy and the $\sigma=1$ pixel assumption in Section 6.4 is overly pessimistic.

A systematic error between LED measurements and tablet camera measurements was observed in some situations. Depending on the position of the tablet in the playing volume, this error was as large as 2cm. It is likely that this errors is caused by inaccuracies in the calibration of E_{SW} and E_{TC} and errors in camera parameter values.

6.10.3 Dynamic Performance

A video file (`tablet_tracking.avi`) demonstrating tracking performance is enclosed. In this video, the tracked edges are rendered into the scene to allow visual inspection of tracking performance. During normal operation, these edges are not rendered.

In standalone operation, the edge-based tracking of the tablet camera is prone to failure on rapid motions. Further, there is a possibility of the edge-based tracking falling into local minima. These failure mechanisms are illustrated in the enclosed video file.

The LED tracking does not suffer any adverse effects from rapid motion. The LEDs are bright enough that the LED camera's exposure time can be set to a sufficiently small value to eliminate motion blur. However, the LED tracking by itself is not accurate enough to properly register the augmented visuals. This is due to both the systematic pose errors described above and the relatively large tracking jitter.

When edge-based tracking and LED tracking are combined, the LED tracking's initialisation is for the most part sufficiently accurate to allow the edge-based tracking to converge correctly. Recovery from total edge-tracking failure is possible as long as the LEDs are in view of the observing camera. It should be noted this volume is larger than appears in the result video, in which the tablet is tethered for filming.

The systematic error described above can produce oscillatory behaviour in the system state. However, since the augmented visuals are rendered using the edge tracking posterior, this oscillatory behaviour of the state is not observable in the AR display - there is however a small probability that at any given frame, edge-tracking will not converge correctly, and this causes occasional one-frame glitches in the display.

The use of ad-hoc re-weighting of innovations of the different sensors in the EKF to account for their differing error behaviours has shown great potential in reducing this oscillation. Simultaneously, re-convergence after edge-tracking failure can be sped up. Robust implementations of the EKF (e.g. **Cipra & Romera**, 1991) may offer performance increases but have not been implemented.

6.10.4 Occlusion Refinement

An accompanying video file (`tablet_refinement.avi`) demonstrates the effect of occlusion refinement. In most scenes, the use of occlusion refinement is beneficial to the composited appearance. However for some configurations the refinement introduces new errors. In particular this is the case if an occluding edge in the image is very low-contrast and in close proximity to other texture edges or shadows. In this case the edge position refinement converges on an incorrect edge, producing large, often dynamic, and mostly very noticeable occlusion errors. There remains scope for improvement of the occlusion refinement procedure, for example by using alternative line-fitting techniques, information from previous frames, or adjoining edges.

In the absence of correspondence failures, the occlusion refinement system enhances the appearance of the composited scene. In particular, the visible effect of tracking jitter is reduced. Further, the “crawling jaggies” effect when occluding edges are near-horizontal or near-vertical is mostly eliminated by the alpha-blended clipping procedure.

The occlusion refinement system presented here only works for edges which are known to be present in the scene; the occlusion of virtual graphics by other objects

is not attempted. Thus, a user's hand, if placed in front of the game world, will be ignored by the system and characters which should be covered will be rendered over the hand. Approaches to handle such occlusions by dynamic scene elements have been proposed and are discussed in Section 2.4, but are not implemented here. On the other hand, the application presented here usually requires both the user's hands to hold the tablet PC and pen (and/or tokens); they are therefore rarely seen in the video stream.

6.10.5 The Tablet PC as an AR Format

This chapter has described the implementation of an AR game application on a tablet PC. This application was demonstrated at the International Symposium on Mixed and Augmented Reality (ISMAR'04) in Arlington where an un-trained (albeit technical) audience was able to operate the game application. In contrast to the HMD-based application described in Chapter 5, which both trained and un-trained users find cumbersome and difficult to operate, users had no trouble picking up the tablet PC and immediately interacting with the AR application. In particular, the lack of per-user calibration and the pen interface allow intuitive use of the device. The fusion of two tracking strategies ensured that for the most part,¹ tracking was transparent; that is, users did not need to worry about the tracking system but could focus on the application.

In direct comparison to PDAs, the tablet PC at 1.4kg is uncomfortably heavy to hold one-handed for extended periods of time; two-handed use is not possible because one hand is required to hold the pen, and the user's choice of supporting hand positions are limited because the tablet's LEDs should not be obscured. Further, the device becomes rather hot, particularly around the area which the user holds. These two factors limited the amount of time users were comfortable holding the tablet to a few minutes. However, it is expected that as tablet PCs become lighter, or PDAs more powerful, full

¹Systematic failures occurred when the tablet-mounted camera or the external camera received a knock, changing the calibrated E_{SW} or E_{TC} matrices.

frame-rate hand-held AR will become a very practical and ergonomically acceptable possibility.

7

A Visual Rate Gyroscope

7.1 Introduction

Many visual tracking algorithms have difficulty tracking scenes in which the camera is undergoing rapid rotation. As discussed in Chapter 4, even moderate camera rotation can cause image features to move very rapidly across the image, and motion blur caused by non-zero exposure time can corrupt the visual features which tracking systems require for operation.

For this reason, practical AR systems using head-mounted vision either use fiducials (which can be localised every frame, ensuring tracking can be resumed even if it fails during rapid motion) or combine head-mounted tracking with alternative forms of input which are robust to rapid rotation. For example, the previous chapter employed outside-in LED tracking for robust tablet tracking, and rate gyroscopes were used to assist HMD tracking in Chapter 5.

Rate gyroscopes are relatively cheap, offer high-bandwidth output from an enclosed unit, and require virtually zero computational cost. Even so, they increase the volume, mass and power requirements of any AR system, and for this reason they are (for example) not used with the tablet PC in Chapter 6. This chapter describes a computer vision-based replacement for rate gyroscopes: an algorithm which estimates camera rotation from video images.

To be a viable replacement for rate gyroscopes this algorithm is designed to have minimal computational requirements. As demonstrated in Section 7.3, the algorithm can compute rotation estimates in under 3 milliseconds per frame on a 2.4 GHz computer. This is sufficiently rapid for it to be used as an initialiser for subsequent tracking algorithms while maintaining real-time performance. Further, as with rate gyroscopes, no prior information is required and so the performance of the algorithm is not affected by earlier tracking failures.

The algorithm operates by analysing the structure of the motion blur present in the image. The basis for its operation is the insight that in the presence of sufficient motion blur, the only sharp edges present in the image will be those parallel to the direction of blur; this allows the center of camera rotation to be computed rapidly, without the use of large-scale 2D image computations. Once the center of rotation is found, the magnitude of rotation can be quickly computed under some simplifying assumptions.

The algorithm's operation is described in Section 7.2 and results are presented in Section 7.3. The algorithm is subject to a number of limitations, some of which (notably the ambiguity of sign) are unavoidable and others which are the result of speed vs. accuracy trade-offs; these are described in Section 7.4.

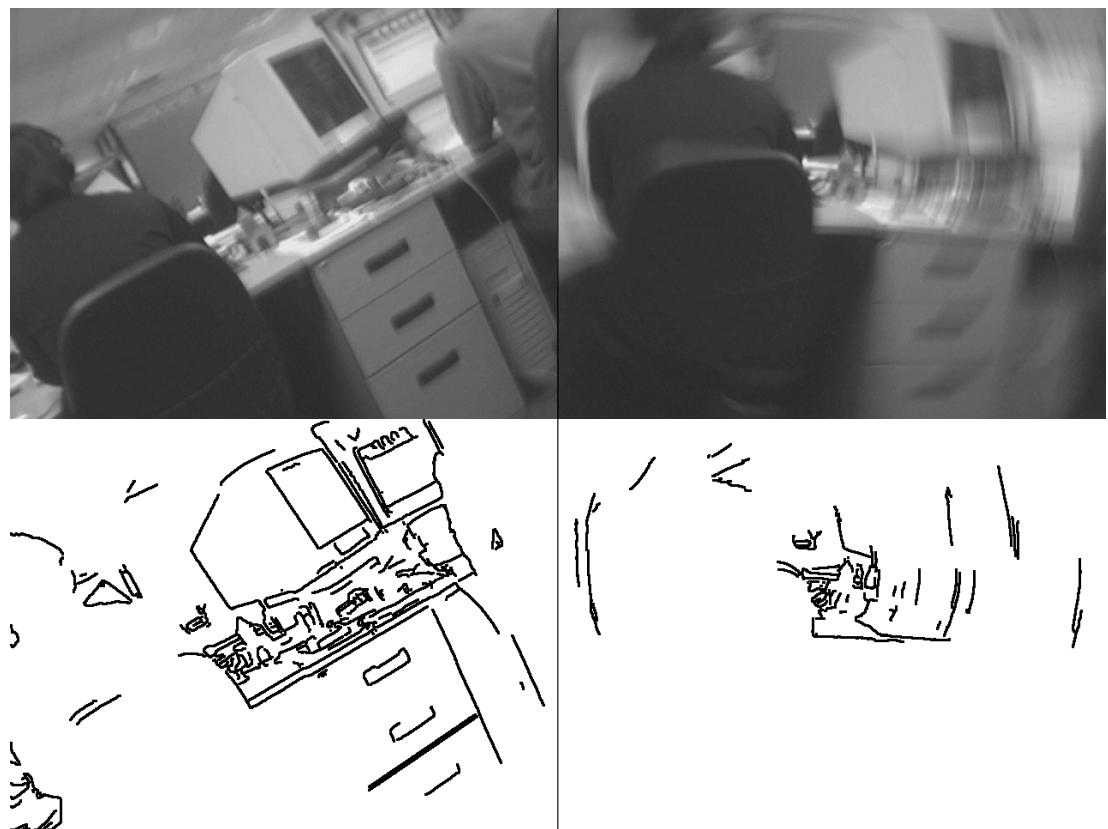


Figure 7.1: Canny edge-extraction of an un-blurred (left) and blurred scene (right). The un-blurred scene contains edges in all directions while in the blurred scene, edges in the direction of blur dominate.

7.2 Method

7.2.1 Overview

This section describes the algorithm to estimate camera rotation from the motion blur present in a single video frame. Since the algorithm is designed for speed rather than accuracy, a simple model of motion blur is used. Only rotation is considered: it is assumed that the camera is either not translating, or the translation is not contributing significantly to motion blur; further, the scene is assumed to be static.

During the frame's exposure, the camera is assumed to rotate with constant angular velocity about a center of rotation $(x_c \ y_c)^T$ in the image plane. Points d pixels away from this center will therefore be blurred across an arc of length θd , where θ is the angle the image rotates during exposure of the frame.

Considering projection by a standard pin-hole model with the camera center at the origin, the optical axis aligned with the z -axis and the image plane at $z = F$, the point about which the image rotates has coordinates

$$\mathbf{c} = \begin{pmatrix} x_c \\ y_c \\ F \end{pmatrix} \quad (7.1)$$

with all units in pixels. In 3D space, the camera rotates around an axis of rotation which passes through the origin and is described by the unit vector $\hat{\mathbf{a}}$. It follows that \mathbf{c} is the projection of $\hat{\mathbf{a}}$ into the image plane. In the case that $\hat{\mathbf{a}}$ is parallel to the image plane (i.e. when the camera is purely panning), \mathbf{c} is at infinity in the image plane, and the model turns arcs of blur in the image into straight lines.

Strictly speaking, the image locus swept by a point under camera rotation could be any conic section; the circular assumption corresponds to a very large focal length or a spherical imaging surface. However, the circular assumption yields faster computation and introduces only small errors, particularly when using lenses which exhibit barrel distortion (most wide-angle lenses are prone to exhibit this effect.)

The algorithm operates in two stages: Section 7.2.2 demonstrates how the axis of rotation \hat{a} can be found. Once this has been calculated, the blur length is estimated in Section 7.2.3. Apart from the pixel aspect ratio, the method requires no precise knowledge of camera parameters - the focal length F used in calculations can be very approximate, and here it is set to 640 pixels. Consequently, all quantities are calculated in pixel units. A conversion of these results to 3D coordinates is straightforward if camera focal length and optic center are known. Knowledge of frame exposure time can then be used to obtain rotational velocity from the calculated blur length.

7.2.2 Axis of Rotation

To determine the axis of rotation in a blurred image the directional nature of motion blur is exploited. Any point in the image is blurred tangentially to a circle centered on c , and not blurred in the perpendicular direction (radially towards c). It follows that image edges emanating radially from c are corrupted by blur, while intensity edges in tangential directions are preserved. This is illustrated in Figure 7.1: The top right image shows a frame affected by motion blur. Beneath it is the result of a Canny edge extraction (Canny, 1986) of this frame: edges parallel to the direction of blur dominate. Thus, the point c can be found as the point which is most perpendicular to all edges remaining in the blurred image.

Figure 7.2 illustrates the operation of the algorithm. To avoid the cost of full-frame edge extraction, the input image (Panel 1) is sparsely searched for edgels. This is done along a grid of vertical and horizontal lines spaced 10 pixels apart. The changes in intensity between adjacent pixels along these lines are computed: local maxima of instantaneous intensity change which exceed a threshold value are assumed to be edgels. Typically, between 100 and 600 edgels are found in this way, and the position $p = (x \ y \ F)^T$ of each edgel is recorded. Figure 7.2 shows edgels extracted by this grid search in Panel 2.

At each edgel site, the local image intensity gradient is found using the Sobel operator (Pingle, 1969). This yields the values G_x and G_y which describe local gradient in the

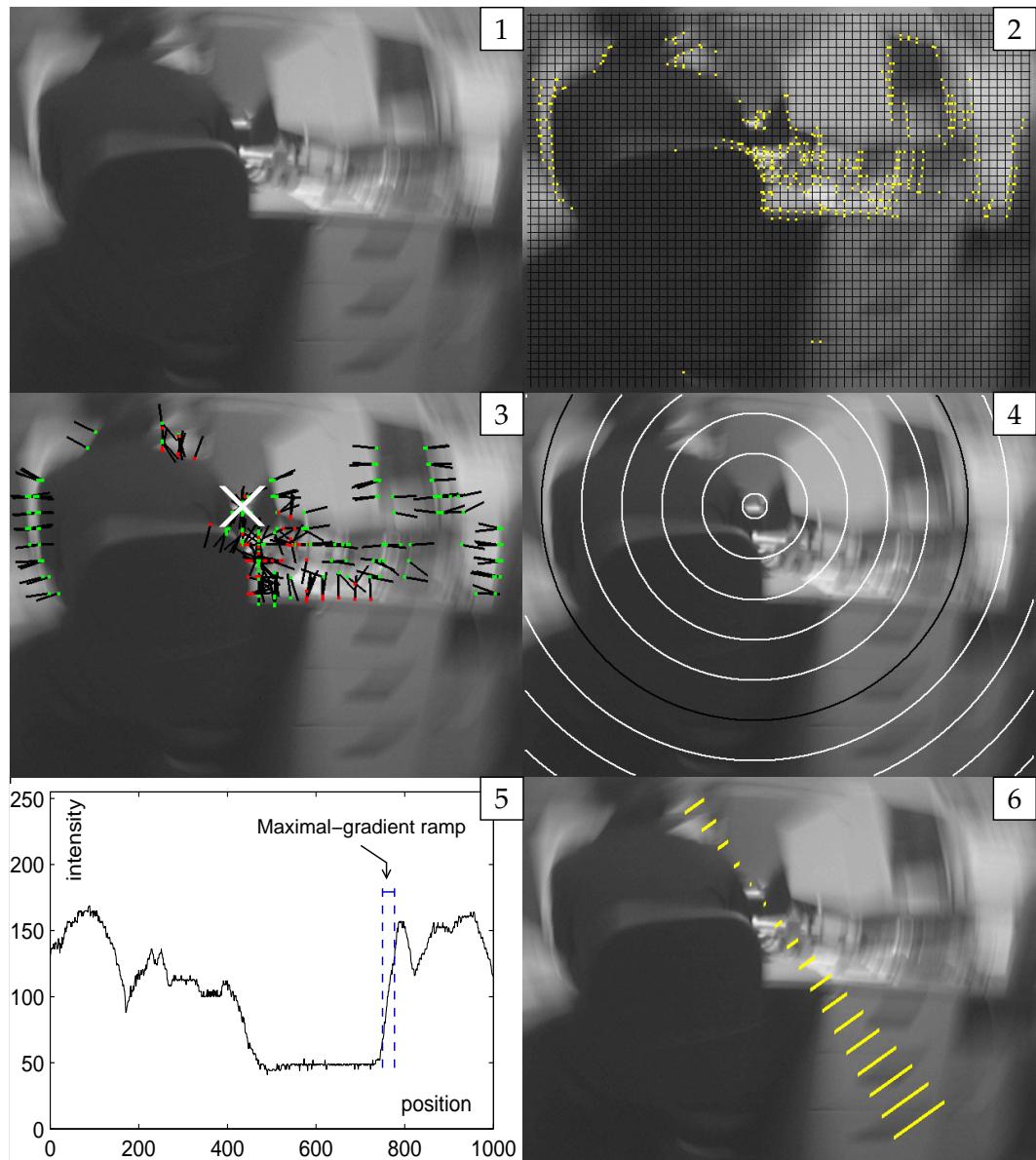


Figure 7.2: Operation of the Algorithm. (1) Blurred input picture. (2) The image is searched for local gradient maxima along a sparse grid. (3) Sobel extraction is performed along these 1D edgels and the orientation of maximal image gradient is indicated by black lines. The best intersection of the black lines (i.e., the center of rotation in the image; white \times) is found using RANSAC and optimisation. Green denotes the inlier set. (4) Pixels along concentric circles around the rotation center are sampled. (5) Plot of the pixel intensity values sampled clockwise along the black circle. The highest-gradient intensity ramp is indicated: this is interpreted as the least-blurred feature on the circle and used to estimate blur length. (6) By combining estimated blur lengths from all circles, the overall angular blur (and hence the estimated camera rotation) is found. The estimated blur length is drawn in the image.

x and y directions respectively. The vector $\mathbf{g} = (G_x \ G_y \ 0)^T$ then describes the direction of maximum gradient, which is normal to any edge in the image. Panel 3 of Figure 7.2 shows edgel normals extracted from the video frame.

Each edgel describes a line $\mathbf{l} = \mathbf{p} + \lambda\mathbf{g}$ in the image plane along which the rotation center \mathbf{c} is expected to lie. This line is more conveniently expressed as the intersection of the image plane with a plane \mathcal{N} passing through the origin; this plane is parametrised by its unit normal vector $\hat{\mathbf{n}}$, given by

$$\hat{\mathbf{n}} = \frac{\mathbf{p} \times \mathbf{g}}{|\mathbf{p} \times \mathbf{g}|}. \quad (7.2)$$

To find the image rotation center \mathbf{c} RANSAC (Random Sample Consensus, **Fischler & Bolles**, 1981) is employed, followed by an optimisation using only the consensus set. In the RANSAC stage, each hypothesis is formed by randomly selecting two edgels a and b . The image rotation center \mathbf{c} is the given by the intersection of lines \mathbf{l}_a and \mathbf{l}_b .

To support rotation centers at infinity, \mathbf{c} is not explicitly calculated. Instead, the algorithm calculates the axis of rotation \mathbf{a} . The image rotation center \mathbf{c} is thus implicitly described as the intersection of \mathbf{a} with the image plane.

The axis of rotation \mathbf{a} lies along the intersection of planes \mathcal{N}_a and \mathcal{N}_b :

$$\mathbf{a} = \hat{\mathbf{n}}_a \times \hat{\mathbf{n}}_b \quad (7.3)$$

To evaluate consensus for each hypothesis, the sum of angular error terms from all other edgels is found. For the i th edgel, θ_i is the angle at \mathbf{p}_i in the image between the line \mathbf{l}_i and the vector $\mathbf{c} - \mathbf{p}_i$; Figure 7.3 illustrates this quantity and the relevant vectors in the image plane.

Instead of calculating θ_i directly, it is more computationally efficient to approximate this angle with ϕ_i , the angle between the plane \mathcal{N}_i and the plane containing \mathbf{p}_i , \mathbf{c} and the optic center. The square of the sine of this angle is used for an error metric. In terms of the hypothesised axis of rotation \mathbf{a} , the error metric ϵ_i is

$$\epsilon_i = \frac{|(\mathbf{a} \times \mathbf{p}_i) \times \hat{\mathbf{n}}_i|^2}{|\mathbf{a} \times \mathbf{p}_i|^2} = \sin^2(\phi_i) \approx \sin^2(\theta_i). \quad (7.4)$$

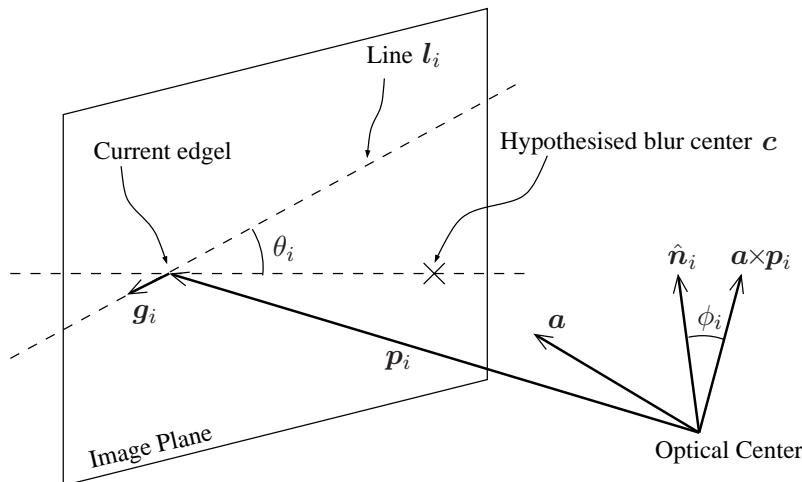


Figure 7.3: Consensus evaluation for the i th edgel and a center hypothesis

The error metric is capped at a threshold value ϵ_{max} and the hypothesis with the lowest summed error is selected. The consensus set for this hypothesis is the set of N edgels whose error metric is lower than ϵ_{max} .

The winning hypothesis a is normalised and then optimised to minimise the sum-squared error $|\epsilon|^2$, where ϵ is the vector of error metrics for the consensus set. This is done using four Gauss-Newton iterations. At each iteration,

$$\mathbf{a}' = \mathbf{a} + \Delta \mathbf{a} \quad (7.5)$$

$$\Delta \mathbf{a} = (J^T J)^{-1} J^T \epsilon \quad (7.6)$$

Where J is the $N \times 3$ Jacobian matrix describing partial derivatives of ϵ with $\Delta \mathbf{a}$

$$J_{ij} = \frac{\partial \epsilon_i}{\partial \Delta a_j} \quad (7.7)$$

and is found by differentiating Equation (7.4) w.r.t. $\Delta\mathbf{a}$:

$$J_{ij} = \frac{\partial}{\partial \Delta a_j} \left(\frac{|((\mathbf{a} + \Delta\mathbf{a}) \times \mathbf{p}_i) \times \hat{\mathbf{n}}_i|^2}{|(\mathbf{a} + \Delta\mathbf{a}) \times \mathbf{p}_i|^2} \right) \Big|_{\Delta\mathbf{a}=0} \quad (7.8)$$

$$= \frac{\partial}{\partial \Delta a_j} \left(\frac{u}{v} \right) = \frac{vu' - uv'}{v^2} \quad (7.9)$$

$$\text{with } u' = 2((\mathbf{a} \times \mathbf{p}_i) \times \mathbf{n}_i) \cdot ((I_j \times \mathbf{p}_i) \times \mathbf{n}_i) \quad (7.10)$$

$$v' = 2(\mathbf{a} \times \mathbf{p}_i) \cdot (I_j \times \mathbf{p}_i) \quad (7.11)$$

where I_j is the j th column of the 3×3 identity matrix. Once \mathbf{a} has been optimised, the image center of rotation c is found by extending \mathbf{a} to intersect the image plane. The extracted rotation center is indicated in Panel 3 of Figure 7.2.

7.2.3 Blur magnitude

Whereas the axis of rotation has been determined by analysing image structure in the direction perpendicular to local blur, the magnitude of blur is determined by looking at pixels in the blurred direction. Pixels are sampled from c -centered circles in the image using an incremental rasterisation algorithm. The circles are initialised at sparse radial intervals (typically 50 pixels apart) to maintain high run-time speed. This process is illustrated in Panel 4 of Figure 7.2. Each scanned circle produces a 1D signal of image intensity along the circle; this signal is assumed to have been convolved with a rectangular pulse of length d , which must be estimated. One such signal is shown in Panel 5 of Figure 7.2.

Rekleitis (1996) estimates this blur length using cepstral analysis: convolution of the signal with a rectangular pulse produces a sinc-pulse envelope in the frequency domain. The cepstrum (the inverse Fourier transform of the log power spectrum) is used to recover this envelope's fundamental frequency, which is proportional to the length of motion blur in the image. While this method is attractive in its conceptual elegance, it is unfortunately not appropriate here. To make the results of the Fourier- and cepstral analysis resilient to noise, a large number of image samples are needed. Further,

the minimum of the cepstrum becomes difficult to locate in images with large blur as camera noise starts to dominate.

Instead, an ad-hoc approach to blur length detection is adopted. Under the assumption that the axis of rotation has been correctly calculated and that the samples are therefore taken along the direction of blur, blur length cannot exceed the length of the shortest intensity ramp which was produced by an intensity step in the scene. However, merely measuring the minimum ramp length is unreliable, since two intensity steps of opposite sign in close proximity can produce arbitrarily short ramps in the image.

To avoid under-estimating blur length, only ramps which span a large (50 greyscale levels or more) intensity change are considered: These are assumed to have originated from large isolated intensity steps in the image. Under the further assumption that the largest intensity step in every scene spans approximately the same intensity increase, the gradient of the steepest ramp to span this change is then inversely proportional to the length of motion blur. This maximal-gradient ramp is found by a brute-force search in which the shortest distance to span the threshold intensity change is found. The maximal-gradient ramp thus found is illustrated in Panel 5 of Figure 7.2.

To combine the results of each circular scan, the maximal gradients of each scan are first scaled according to relative circle radius, and then combined by calculating their p -norm (with $p = 5$) to provide some resilience to outliers. The inverse of this result forms the estimate for the magnitude of camera rotation during the frame's exposure.

7.3 Results

This section describes the single-frame performance of the system for sequences where the system can operate correctly. There are many scenarios in which the system will always yield incorrect results. These cases are discussed in Section 7.4.



Figure 7.4: Rotation center placement results for four test scenes (un-blurred in top row.)

The visual gyroscope was implemented in C++ on a 2.4GHz Pentium-4 computer and tested on live video footage. Images are captured from a fire-wire camera at 30Hz, are greyscale and have a resolution of 640×480 pixels with a square pixel aspect ratio. The camera's exposure time was set to its maximum value of approximately 30ms.

Figure 7.4 shows the algorithm's choice of rotation center for a number of different motions in four test sequences. The center-extraction section operates fairly reliably for scenes with motion blur of 10 pixels length or greater, even when these scenes contain very bright elements which make blur length detection problematic.

Figure 7.5 compares the output of the algorithm to the output of a Futaba G301 piezo-electric rate gyroscope mounted to measure camera panning about the vertical axis. The left plot shows a series of camera shakes of increasing intensity recorded in an indoor setting (the same scene as used for Figure 7.2). Since the visual gyroscope produces measurements with a sign ambiguity, the results show the absolute value of horizontal image motion in pixel units. The plot on the right of Figure 7.5 compares

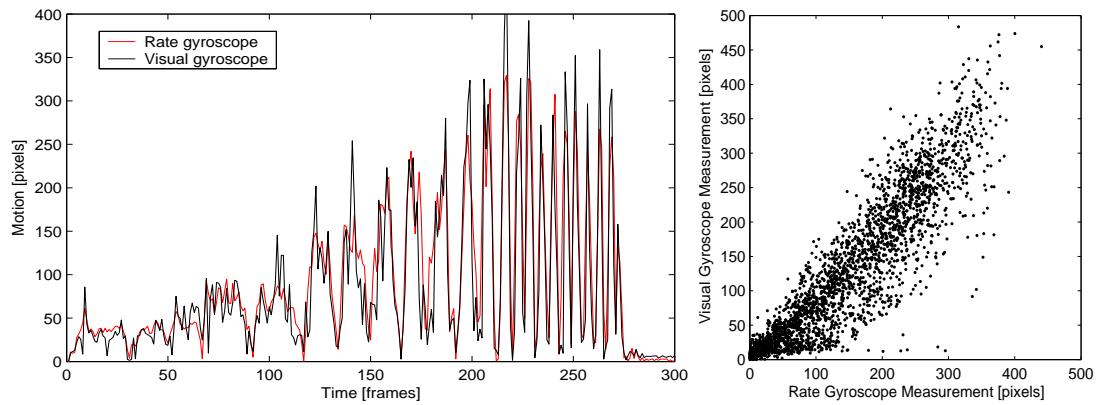


Figure 7.5: Estimated blur magnitude results compared to a rate gyroscope (taken as ground truth.)

2000 samples of rotational velocity taken from the visual and rate gyroscopes. Ideally, the left plots should be identical, and the plot on the right should show a $y = x$ line.

The execution speed of the algorithm is largely limited by the floating-point computations required to determine the center of rotation (RANSAC and optimisation). The cost of these computations scales linearly with the number of edgels used; it is therefore possible to tune execution speed by varying the maximum number of edgels processed per frame. To reliably obtain an execution time of under 3ms per frame, the algorithm processes only the 300 strongest edgels of the 300-2000 edgels typically found in an image. Average computation times of different stages of the algorithm are shown in Table 7.1.

Process	Time [msec]
Edgel extraction	0.50
Best edgel selection	0.05
RANSAC	0.65
Optimisation	0.35
Circular sampling	0.15
Blur length search	0.40
Total	2.10

Table 7.1: Timing results on a 2.4GHz Pentium 4

7.4 Limitations

This sections describes some known limitations of the proposed system.

1. **Sign ambiguity in blur magnitude:** Under the assumption of time-invariant illumination and an instantaneous shutter across the frame, there is no way of distinguishing the direction of time in a single frame. This information must be acquired elsewhere, e.g. by comparison with other frames or from a tracking system's motion model. A method for resolving this ambiguity using image patches from the previous frame is described in Section 7.5; it is conceivable that for multi-modal tracking systems (e.g. those employing a particle filter pose representation) the sign ambiguity is acceptable.
2. **Intolerance to strobing lights:** Illumination is assumed to be of constant intensity throughout a frame's exposure. Some objects, such as CRT screens, do not satisfy this assumption, and produce sharp edges in otherwise blurred images. If the rest of the image is strongly blurred, these sharp edges can cause both the axis of rotation and blur magnitude to be mis-detected.
3. **Requirement of edges in the scene:** Scenes completely devoid of sharp edges cannot be well distinguished from heavily blurred images. In such cases the system can produce erroneous results. Further, if a scene's edges are all oriented in similar directions the system will frequently mis-detect the axis of rotation. For example, a diagonal pan across a chess-board is poorly handled, since the rotation axis detection stage lacks sharp diagonal edges.
4. **Requirement of motion blur:** The system over-estimates blur magnitude for images with small amounts of motion blur, or no blur at all. For scenes with no blur, the center of location is effectively random. However, any sharp elements present in the scene will ensure that blur magnitude estimates are bounded. It would therefore be possible to simply discard estimates predicting small rotations.

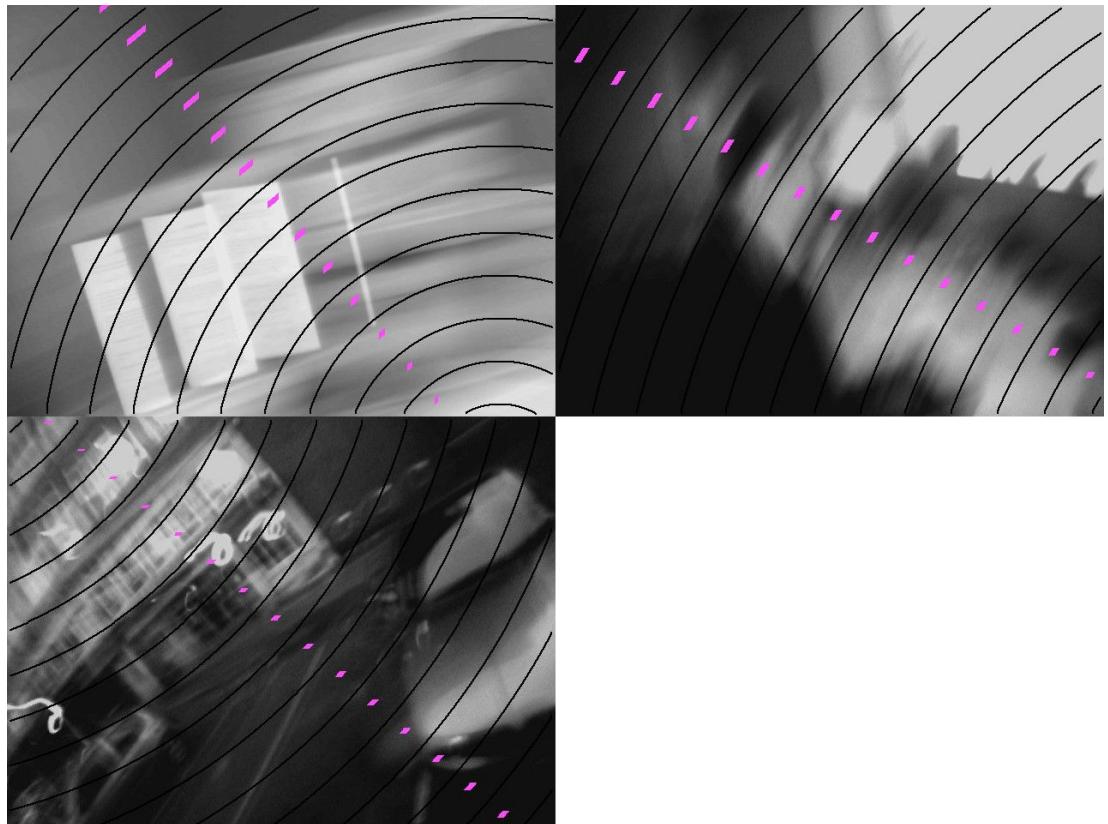


Figure 7.6: Some failure modes of the algorithm. Top left: a CRT computer monitor here refreshes three times in a single 30ms exposure, causing sharp edges in an otherwise blurred image. Top right: the window in the top right of image is bright enough to saturate the camera sensor, causing sharper-than-expected edges at its border. Bottom: The camera is undergoing sharp acceleration and not rotating with constant velocity (as can be observed by the locus of the bright light sources.) In all cases, blur length is under-estimated.

5. **Assumption of linear camera:** The intensity transfer function of the camera used is assumed to be linear ($\text{gamma}=1.0$). Scenes with bright light sources which saturate the camera sensor can be problematic, as their edges can produce high-gradient intensity ramps even under very large blur.
6. **Pure rotation assumption:** The algorithm assumes that motion blur is caused purely by camera-centered rotation and not by camera translation. In practice, when using a head-mounted camera, rotation does indeed contribute the largest component of motion blur, so this assumption is not unreasonable. There can however be cases in which translation is misinterpreted as rotation, particularly if the depth of the scene is very shallow.
7. **Fixed rotation center assumption:** The algorithm assumes that the axis of rotation is fixed during exposure. Very rapid camera acceleration can however cause the axis of rotation to change during a frame. In these cases the paths traced by points in the image no longer form concentric circles, and the detection of the rotation center can fail.

Some of these limitations are illustrated in Figure 7.6.

7.5 Combination with Edge Tracking

To test the algorithm's viability as an initialiser for prior-based tracking systems, the algorithm was combined with the edge-based tracking described in Chapter 4: instead of querying rate gyroscopes over the serial port, each video frame is passed to the visual gyroscope to obtain a rotation estimate. This rotation estimate is used to update camera pose and to modify the edge detection behaviour of the tracking system.

To resolve the directional ambiguity of the visual gyroscope's output, the tracking system's motion model (as described in Section 4.2.5) was employed. This motion model tracks the camera's velocity with time. Its linear component is used directly to



Figure 7.7: Eight consecutive frames of a pan across the AR game world. This sequence is trackable using the visual gyroscope combined with edge tracking.

predict the camera's pose, but its rotational component is used only to disambiguate forwards and backwards for the gyroscope's rotational estimate.

Using this motion model, the visual gyroscope is capable of assisting the edge-based tracking of camera pans. However, sudden pans initiated from rest can cause this strategy to fail: at rest, the motion contains zero velocity plus random noise, so if the first frame of motion contains substantial blur, the system effectively has a 50% chance of failure.

The attached video `visual_gyroscope.avi` illustrates this behaviour. Figure 7.7 shows some consecutive frames from this test sequence in which the camera pans rapidly starting from rest. The motion model has no chance to learn the direction of motion because tracking is immediately lost.

To reliably choose a forward direction, information from a previous frame is required. The system was modified to sample nine 33×33 -pixel image patches (arranged in a 3×3 grid) from each video frame. At the next frame, a camera rotation estimate is generated; to determine the direction of motion, the stored patches are compared to their projected positions in both the forward and backward directions by a simple sum-squared difference metric. Each patch votes for the direction yielding the smaller sum-squared difference, and the winning direction is chosen. The computational cost

of this procedure is under 0.3 milliseconds per frame. The attached video illustrates the performance of the resulting gyroscope replacement system running in standalone operation.

Even when the direction of motion can accurately be determined, some ad-hoc measures are required to track a sequence with rapid camera shake. Although the edge-based tracking system is designed to handle motion blur, it is unreliable when used with frames containing motion blur over 100 pixels, such as those found in the middle of the sequence in Figure 7.7. Edge-based tracking is hence disabled for frames with blur exceeding 70 pixels. Likewise, the visual gyroscope output is unreliable at low levels of blur, so this too is disabled at levels of blur under 5 pixels (at these moderate levels, the edge-based tracking system can often cope unaided.)

Using this combination of visual gyroscope and edge-based tracking, challenging sequences previously requiring the use of rate gyroscopes become (partially) trackable using only visual techniques, as demonstrated in the attached video sequence: tracking eventually fails due to a mis-estimation of translational error in the horizontal direction rather than orientation, an error that neither the algorithm presented here nor physical gyroscopes could correct.

7.6 Conclusions

This chapter has presented a simple method for calculating camera rotation up to a sign ambiguity from a single blurred image; further, the ambiguity can easily be resolved by the use of a neighbouring image. The use of sharp features remaining in the blurred image makes the method fast, permitting its use in combination with other real-time tracking systems.

While axis of rotation of the camera can be computed fairly reliably, measurements of blur magnitude are currently fairly noisy. Further, the system is limited to operation in favourable conditions (as described in Section 7.4). While these limitations mean

that the system can not yet fully replace the functionality of actual gyroscopes, it has nevertheless been demonstrated to be useful as an initialiser for the edge-based tracking used in previous parts of this thesis.

8

Conclusion

8.1 Summary

This thesis has investigated the application of visual tracking techniques to augmented reality. This has been done in the context of two AR systems: one based on a head-mounted display and one based on a tablet PC. Each system offers different challenges and opportunities and visual tracking techniques have been adapted to suit each in turn.

The HMD-based system exposes a head-mounted camera to the user's very rapid and unpredictable head rotations. Visual tracking must be robust to these rotations. To achieve this robustness, an existing edge-based tracker is equipped with a wide-angle lens and rate gyroscopes, and further modified so that tracking can proceed even when camera rotation substantially corrupts the image with motion blur. An existing nonlinear calibration procedure used to calibrate the tracking system's lens is adapted

and used to calibrate the head-mounted display; this allows nonlinear distortions in the display to be compensated for and results in improved registration for the user.

The tablet-based application presented is centered around a fixed playing area and can make use of external cameras. For this application the emphasis is less on robustness to rotations than on reliable tracking which can survive the intermittent failure of the tablet-mounted tracking system. The use of active fiducials attached to the tablet makes this possible without prominent markers in the game world. The higher-quality display capability of the tablet's screen sets higher demands on graphical rendering, and small rendering errors which are missed on the HMD become jarring on the video see-through display; one such class of errors (real objects not accurately occluding virtual ones) is addressed through further local tracking in the image being augmented.

Finally, this thesis has presented a software alternative to the use of rate gyroscopes. This algorithm was developed after the AR applications had been completed; due to time constraints it was therefore not directly tested in an AR context. Instead, it is presented as a tool which may be applicable to many tracking applications, and its potential is illustrated by integration with the edge-based tracker previously used.

8.2 Contributions

What follows is a list of the contributions made in this thesis.

- Edge tracking under very fast rotation. The use of a wide-angle lens and of rate gyroscopes to modify the tracking system's edge detection produces a tracker that can continue to operate using blurred images which few (if any) other tracking systems could support.
- An intuitive display calibration system which compensates for radial distortion. Instead of forcing users to move their head to satisfy arbitrary geometric constraints, the calibration system presented here merely requires the user to correct

any parts of the overlay which appear out-of-place using a simple point-and-click interface. The framework used permits the correction of radial distortion, improving registration accuracy.

- A robust method of tracking a tablet PC's pose using a novel outside-in fiducial tracker and inside-out edge tracking. This method combines the robustness of the outside-in tracker with the accuracy of the inside-out system, and does not require any markers to be placed in the tracked scene.
- A real-life demonstration of tablet PC-based AR. The tablet PC has a number of ergonomic advantages over head-mounted displays and the suitability of this device as an AR medium has been demonstrated by the development and public demonstration of a functional game application.
- An occlusion refinement procedure: a method of improving the integration of virtual graphics into real scenes by exploiting the availability of a scene CAD model to identify and then refine individual occluding edges.
- A fast algorithm to extract camera rotation from motion blur. The low computational cost of this algorithm makes it suitable as a drop-in replacement for rate gyroscopes.

8.3 Future Work

The work undertaken for the production of this thesis has revealed numerous areas in which more research and development are required. A number of these relate to the tracking techniques employed, but others to AR in general.

- A prior-based tracker in isolation will never be robust. The edge-based tracker can and will occasionally fail, rate gyroscopes or not. Only when combined with an alternative, absolute source of measurements (e.g. the outside-in tracker) can it be considered robust. When outside-in tracking is not available, a method of

detecting tracking failure, and a method to automatically re-initialise tracking are required.

- The calibration of optically see-through HMDs remains difficult. Methods to reduce the quantity of user input - for example, by sharing information between the eyes - are required. Further, the motion of a display on the user's head still destroys calibration; a method of correcting for such motions without resorting to a full calibration would be beneficial. Approaches along the lines of that taken by **Owen et al (2004)** are encouraging.
- The field-of-view (FOV) of consumer HMDs is too small for immersive applications. The field-of-view of optically see-through displays is typically small partially because wide-FOV displays incur large amounts of nonlinear distortion and this has historically made accurate registration difficult; however, with modern projection models and graphics hardware able to appropriately pre-distort images with little overhead, distortion should no longer stand in the way of wide-angle displays.

Appendix A

Results Videos

The majority of research described in this thesis involves the processing of video material. Results for video sequences are difficult to express in numbers or images, hence a CD-ROM with results videos is attached to this document. This appendix contains descriptions of the videos found on the CD-ROM.



`edge_tracking.mpg`

This video demonstrates the edge tracking presented in Chapter 4. The video compares tracking performance for the ‘cubicle’ environment, using no gyroscope information, gyro initialisation without blur prediction, and finally initialisation with full blur prediction.



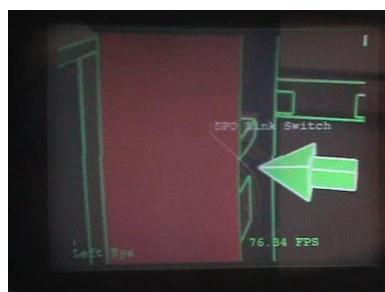
hmd_calibration.mpg

This video is shot live with a video camera looking through the Sony Glasstron HMD. It demonstrates the HMD parameter calibration described in Section 5.5.2. The user selects model vertices and clicks in the correct locations in the display, aligning the rendered model with the real world.



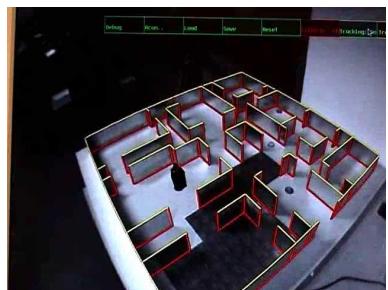
hmd_printer.avi

This video is shot live with a video camera looking through the Sony Glasstron HMD. It demonstrates the printer maintenance application described in Section 5.3. The user is visually guided through a checklist.



hmd_machineroom.avi

This video is shot live with a video camera looking through the Sony Glasstron HMD. It demonstrates the machine room application described in Section 5.3, and shows a 'Matrix'-style graphical overlay later on.



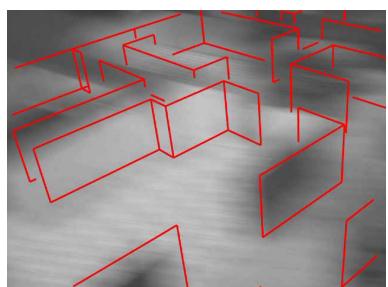
tablet_tracking.avi

This video is shot live with a video camera viewing the tablet PC's external monitor output. It shows that robust and accurate tablet tracking is achieved when inside-out and outside-in tracking are combined (as described in Section 6.3.)



tablet_refinement.avi

This video demonstrates the difference between rendering a character with standard z-buffer techniques, and rendering with the occlusion refinement code described in Section 6.9. The video alternates between the two rendering techniques as the character moves around the game world.



visual_gyroscope.avi

This video demonstrates the operation of the visual gyroscope described in Chapter 7. The standard edge-based tracker cannot track the rapid camera motions in the video. Tracking is improved by employing the single-frame gyroscope, but its directional ambiguity causes tracking to fail. The ambiguity can be resolved using information from the previous frame, resulting in increased (but not perfect) tracking robustness.

Appendix B

Projection Derivatives

B.1 Tracking Jacobian

This section describes the calculation of the tracking system Jacobian used in Chapter 4. The derivatives of the form $\frac{\partial d_i}{\partial \mu_j}$ as written in Equation (4.11) are best computed by splitting some of the projection Equations (4.2 ... 4.8) into smaller divisions:

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{bmatrix} f_u & 0 & u_0 \\ 0 & f_v & v_0 \end{bmatrix} \begin{pmatrix} A_1 \\ A_2 \\ 1 \end{pmatrix} \quad (\text{B.1})$$

$$\begin{pmatrix} A_1 \\ A_2 \end{pmatrix} = \frac{\tilde{r}}{r} \begin{pmatrix} C_1 \\ C_2 \end{pmatrix} \quad (\text{B.2})$$

$$\begin{pmatrix} C_1 \\ C_2 \end{pmatrix} = \begin{pmatrix} \frac{x_c}{z_c} \\ \frac{y_c}{z_c} \end{pmatrix}. \quad (\text{B.3})$$

where $r = \sqrt{C_1^2 + C_2^2}$ and $\tilde{r} = r - \beta_1 r^3 - \beta_2 r^5$. Defining further

$$B \equiv \frac{\tilde{r}}{r} = (1 - \beta_1 r^2 - \beta_2 r^4), \quad (\text{B.4})$$

Equation (B.2) becomes

$$\begin{pmatrix} A_1 \\ A_2 \end{pmatrix} = B \begin{pmatrix} C_1 \\ C_2 \end{pmatrix}. \quad (\text{B.5})$$

The partial derivatives of these individual equations are readily obtainable:

$$\text{Eq. B.1: } J_A = \begin{bmatrix} \frac{\partial u}{\partial A_1} & \frac{\partial u}{\partial A_2} \\ \frac{\partial v}{\partial A_1} & \frac{\partial v}{\partial A_2} \end{bmatrix} = \begin{bmatrix} f_u & 0 \\ 0 & f_v \end{bmatrix} \quad (\text{B.6})$$

$$\text{Eq. B.4: } J_B = \begin{bmatrix} \frac{\partial B}{\partial C_1} & \frac{\partial B}{\partial C_2} \end{bmatrix} = \begin{bmatrix} -2\beta_1 C_1 - \beta_2(4C_1^3 + 4C_1 C_2^2) \\ -2\beta_1 C_2 - \beta_2(4C_2^3 + 4C_2 C_1^2) \end{bmatrix}^T \quad (\text{B.7})$$

$$\text{Eq. B.5: } J_C = \begin{bmatrix} \frac{\partial A_1}{\partial C_1} & \frac{\partial A_1}{\partial C_2} \\ \frac{\partial A_2}{\partial C_1} & \frac{\partial A_2}{\partial C_2} \end{bmatrix} = \begin{bmatrix} B & 0 \\ 0 & B \end{bmatrix} + \begin{pmatrix} C_1 \\ C_2 \end{pmatrix} J_B \quad (\text{B.8})$$

$$\text{Eq. B.3: } J_D = \begin{bmatrix} \frac{\partial C_1}{\partial x_c} & \frac{\partial C_1}{\partial y_c} & \frac{\partial C_1}{\partial z_c} \\ \frac{\partial C_2}{\partial x_c} & \frac{\partial C_2}{\partial y_c} & \frac{\partial C_2}{\partial z_c} \end{bmatrix} = \begin{bmatrix} \frac{1}{z_c} & 0 & \frac{-x_c}{z_c^2} \\ 0 & \frac{1}{z_c} & \frac{-y_c}{z_c^2} \end{bmatrix}. \quad (\text{B.9})$$

Finally, the derivative of $\mathbf{x}_c = [x_c \ y_c \ z_c \ 1]^T$ w.r.t. μ_j follows from Equation (3.6):

$$\frac{\partial \mathbf{x}_c}{\partial \mu_j} = G_j E_{\mathcal{CW}} \mathbf{x}_{\mathcal{W}}. \quad (\text{B.10})$$

and combining these partial differentials with Equation (4.11), eliding the subscript i for the i th measurement, yields the derivatives

$$\frac{\partial d}{\partial \mu_j} = -\mathbf{n} \cdot J_A J_C \begin{bmatrix} J_D & 0 \\ 0 & 0 \end{bmatrix} G_j E_{\mathcal{CW}} \mathbf{x}_{\mathcal{W}}. \quad (\text{B.11})$$

B.2 HMD Calibration Jacobian

The partial differentials needed to calibrate the HMD in Section 5.5 are found analogously to the differentials in Section B.1. The HMD however uses a slightly simplified projection model (cubic as opposed to quintic distortion) and each feature point produces two lines of the Jacobian, since the aperture problem of edge tracking does not apply. Further, derivatives with respect to intrinsic projection parameters are required. Repeating the process of Section B.1, we split the motion-undergoing projection equation

$$\begin{pmatrix} u \\ v \end{pmatrix} = \text{EyeProj}(M_{\mathcal{E}} E_{\mathcal{EC}} \mathbf{x}_c) \quad (\text{B.12})$$

into smaller parts:

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{bmatrix} f_u & 0 & u_0 \\ 0 & f_v & v_0 \end{bmatrix} \begin{pmatrix} A_1 \\ A_2 \\ 1 \end{pmatrix} \quad (\text{B.13})$$

$$\begin{pmatrix} A_1 \\ A_2 \end{pmatrix} = B \begin{pmatrix} C_1 \\ C_2 \end{pmatrix} \quad (\text{B.14})$$

$$B = \frac{\tilde{r}}{r} = (1 - \beta r^2) \quad (\text{B.15})$$

$$\begin{pmatrix} C_1 \\ C_2 \end{pmatrix} = \begin{pmatrix} \frac{x_{\mathcal{E}}}{z_{\mathcal{E}}} \\ \frac{y_{\mathcal{E}}}{z_{\mathcal{E}}} \end{pmatrix} \quad (\text{B.16})$$

where $r = \sqrt{C_1^2 + C_2^2}$ and $\tilde{r} = r - \beta r^3$ in accordance with the simpler cubic projection model. Again, partial derivatives of these equations, and the of the point $x_{\mathcal{W}}$ are obtained:

$$\text{Eq. B.13: } J_A = \begin{bmatrix} \frac{\partial u}{\partial A_1} & \frac{\partial u}{\partial A_2} \\ \frac{\partial v}{\partial A_1} & \frac{\partial v}{\partial A_2} \end{bmatrix} = \begin{bmatrix} f_u & 0 \\ 0 & f_v \end{bmatrix} \quad (\text{B.17})$$

$$\text{Eq. B.15: } J_B = \begin{bmatrix} \frac{\partial B}{\partial C_1} & \frac{\partial B}{\partial C_2} \end{bmatrix} = \begin{bmatrix} -2\beta C_1 & -2\beta C_2 \end{bmatrix} \quad (\text{B.18})$$

$$\text{Eq. B.14: } J_C = \begin{bmatrix} \frac{\partial A_1}{\partial C_1} & \frac{\partial A_1}{\partial C_2} \\ \frac{\partial A_2}{\partial C_1} & \frac{\partial A_2}{\partial C_2} \end{bmatrix} = \begin{bmatrix} B & 0 \\ 0 & B \end{bmatrix} + \begin{pmatrix} C_1 \\ C_2 \end{pmatrix} J_B \quad (\text{B.19})$$

$$\text{Eq. B.16: } J_D = \begin{bmatrix} \frac{\partial C_1}{\partial x_{\mathcal{E}}} & \frac{\partial C_1}{\partial y_{\mathcal{E}}} & \frac{\partial C_1}{\partial z_{\mathcal{E}}} \\ \frac{\partial C_2}{\partial x_{\mathcal{E}}} & \frac{\partial C_2}{\partial y_{\mathcal{E}}} & \frac{\partial C_2}{\partial z_{\mathcal{E}}} \end{bmatrix} = \begin{bmatrix} \frac{1}{z_{\mathcal{E}}} & 0 & \frac{-x_{\mathcal{E}}}{z_{\mathcal{E}}^2} \\ 0 & \frac{1}{z_{\mathcal{E}}} & \frac{-y_{\mathcal{E}}}{z_{\mathcal{E}}^2} \end{bmatrix} \quad (\text{B.20})$$

$$\frac{\partial x_{\mathcal{E}}}{\partial \mu_j} = G_j E_{\mathcal{E}C} \mathbf{x}_C \quad (\text{B.21})$$

Combining these equations yields the motion of the projected geometry w.r.t. the motions μ_j :

$$\frac{\partial}{\partial \mu_j} \begin{pmatrix} u \\ v \end{pmatrix} = J_A J_C \begin{bmatrix} J_D & 0 \\ 0 & 0 \end{bmatrix} G_j E_{\mathcal{E}C} \mathbf{x}_C \quad (\text{B.22})$$

The negative of these derivatives form the Jacobian matrix J^μ of Equation (5.10). To find the matrix J^p , the projection equations are differentiated w.r.t. the intrinsic pro-

jection parameters. For one measurement,

$$J^p = - \begin{bmatrix} \frac{\partial u}{\partial f_u} & \frac{\partial u}{\partial f_v} & \frac{\partial u}{\partial u_0} & \frac{\partial u}{\partial v_0} & \frac{\partial u}{\partial \beta} \\ \frac{\partial v}{\partial f_u} & \frac{\partial v}{\partial f_v} & \frac{\partial v}{\partial u_0} & \frac{\partial v}{\partial v_0} & \frac{\partial v}{\partial \beta} \end{bmatrix} \quad (\text{B.23})$$

$$= - \begin{bmatrix} A_1 & 0 & 1 & 0 & -r^2 C_1 \\ 0 & A_2 & 0 & 1 & -r^2 C_2 \end{bmatrix} \quad (\text{B.24})$$

Appendix C

M-Estimation

This appendix introduces a robust estimation technique called *M*-Estimation. This technique is based largely on the insight of **Tukey** (1960) that least-squares solutions are not appropriate for outlier-contaminated data sets and subsequent work by **Huber** (1981, 1964) and others. M-Estimation is now commonly used in computer vision and excellent tutorials on the subject exist (**Zhang**, 1997).

Consider the tracking pose estimation of Section 4.2.4 as a least-squares estimation problem. Treating the problem as linear, given a vector of N observations \mathbf{d} and a Jacobian matrix J (of size N by M), the task is to find a vector of M parameters $\boldsymbol{\mu}$ which minimises the sum-squared residual error

$$|J\boldsymbol{\mu} - \mathbf{d}|^2. \quad (\text{C.1})$$

This can be formulated as in terms of an *objective function* $O(\boldsymbol{\mu})$. The task is then to find

$$\operatorname{argmin}_{\boldsymbol{\mu}} O(\boldsymbol{\mu}) \quad (\text{C.2})$$

where the objective function is

$$O(\boldsymbol{\mu}) = \sum_{i=1}^N \rho(J_i \boldsymbol{\mu} - d_i). \quad (\text{C.3})$$

J_i is the i th row of the measurement Jacobian, and each residual r contributes $\rho(r)$ towards the objective function, with $\rho(r) = r^2$ for the least-squares case. The objective function is minimised by setting its differentials w.r.t. μ to zero. Define (Hampel, 1974) the differential of $\rho()$ as the *influence function* :

$$\psi(r) = \frac{d}{dr}\rho(r) \quad (\text{C.4})$$

so for the sum-squared case $\psi(r) = 2r$. Differentiating w.r.t. the vector μ to find the minimum,

$$\frac{dO(\mu)}{\partial\mu} = \mathbf{0} \quad (\text{C.5})$$

$$\sum_{i=1}^N J_i^T \psi(J_i\mu - d_i) = \mathbf{0} \quad (\text{C.6})$$

$$\sum_{i=1}^N J_i^T J_i \mu = \sum_{i=1}^N J_i^T d_i \quad (\text{C.7})$$

$$J^T J \mu = J^T d \quad (\text{C.8})$$

$$\mu = (J^T J)^{-1} J^T d. \quad (\text{C.9})$$

This gives the standard pseudo-inverse solution for the least-squares problem.

Least-squares solutions are appropriate for cases in which measurements are corrupted by independent Gaussian noise. In the case of the visual tracking system, measurements are not only corrupted by small Gaussian noise but also contain many *outliers*, i.e. measurements which differ significantly from their true value due to feature mis-detection, occlusion or clutter. The error distribution of any individual measurement is therefore very *heavy-tailed* in comparison to a Gaussian distribution, since outside of a Gaussian region immediately surrounding the correct feature location, the probability of encountering outliers may be seen as uniform across the frame.

The influence function $\rho(r)$ for least-squares is proportional to the size of the residual. As can be deduced from Equation (C.6), outliers which produce large residuals thus have a scaled-up influence on the computed optimum. This means that least-squares estimation is not robust in the presence of any significant number of outliers.

A common strategy for robustly estimating parameters in the presence of outliers is to replace the objective function $\rho(x)$ with one of a set of robust functions designed to reduce the impact of outliers. Such functions are typically characterised by having a bounded influence function, and a whole range of potential functions is outlined in **Zhang** (1997).

To implement an M-estimator, the minimisation is re-formulated as recursive weighted least-squares problems. This is done by expressing $\psi(r)$ in terms of a *weight function* $\omega(r)$ where

$$\omega(r) = \frac{\psi(r)}{r} \quad (\text{C.10})$$

such that the influence function $\psi(r)$ may be replaced by a weighted residual. Weights for each residual are pre-computed at each iteration using the previous iteration's residual. For the n th iteration,

$$w_i = \omega(r_i|_{n-1}) = \omega(J_i\boldsymbol{\mu}|_{n-1} - d_i) \quad (\text{C.11})$$

so that the influence function $\psi(r_i)$ can be replaced by the weighted residual $r_i w_i$. Substituting Eqs. (C.10-C.11) into Eq. (C.6), the objective function is minimised:

$$\sum_{i=1}^N J_i^T (J_i\boldsymbol{\mu} - d_i) w_i = \mathbf{0} \quad (\text{C.12})$$

$$\sum_{i=1}^N J_i^T w_i J_i \boldsymbol{\mu} = \sum_{i=1}^N J_i^T w_i d_i \quad (\text{C.13})$$

$$J^T W J \boldsymbol{\mu} = J^T W d \quad (\text{C.14})$$

$$\boldsymbol{\mu} = (J^T W J)^{-1} J^T W d \quad (\text{C.15})$$

where W is a diagonal matrix containing the weights $w_{1\dots N}$. The solution corresponds to the least-squares solution of the problem $W^{\frac{1}{2}} J \boldsymbol{\mu} = W^{\frac{1}{2}} d$.

Smith (2001) employs local edge searches for the task of segmentation, and has experimentally observed that edge measurement statistics resemble a Laplacian (rather than Gaussian) distribution. The use of the “Fair” estimator which matches this distribution and contains few discontinuities is recommended, and this is the estimator used here. Table C.1 shows a comparison of this estimator to least-squares. The tuning constant c is set to 4 pixels for the tracking system.

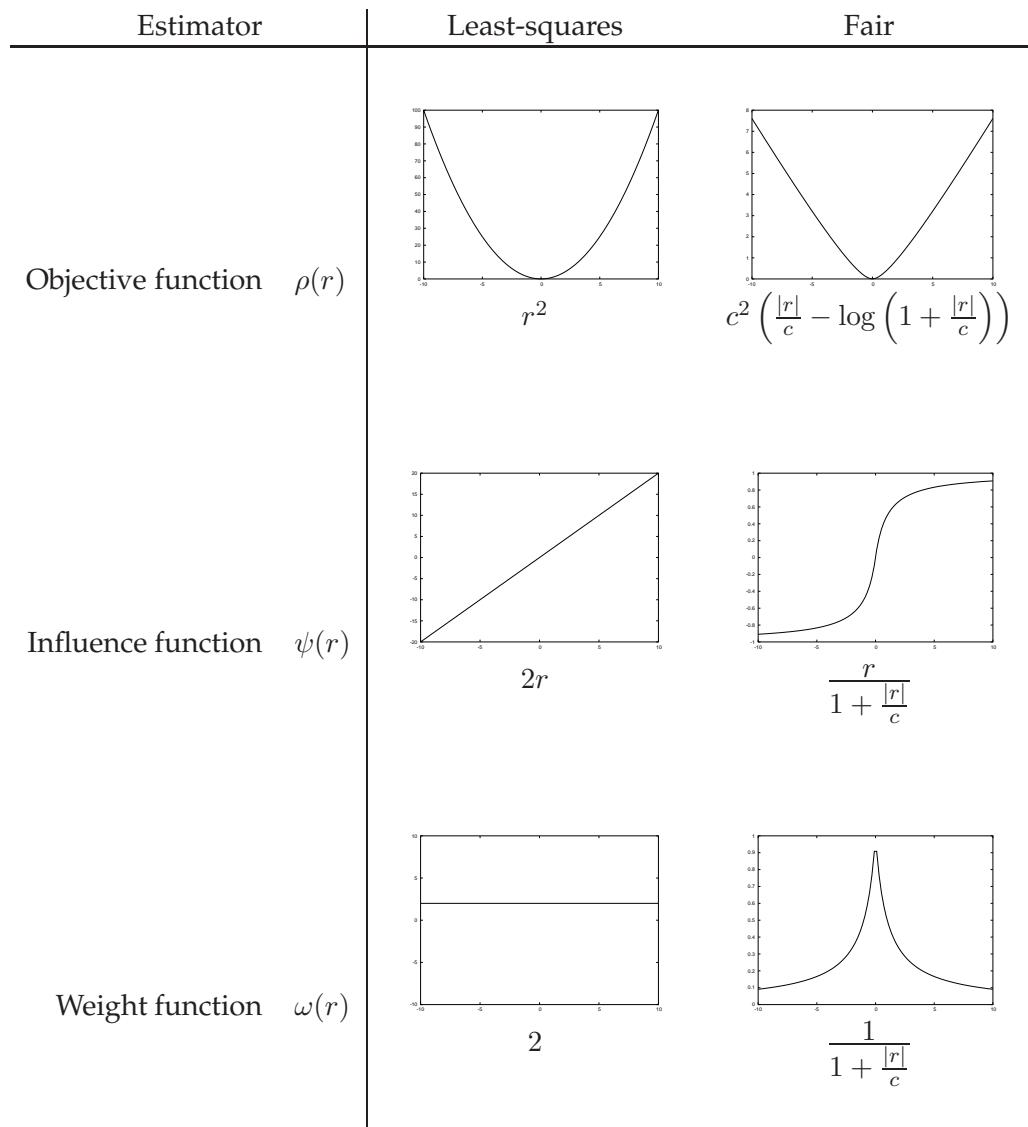


Table C.1: Comparison of least-squares and “Fair” estimators

Appendix D

Homographies

D.1 Estimating a Homography

This appendix describes an algorithm to estimate an 8-DOF plane-to-plane homography H of the form

$$\begin{bmatrix} w_1 u_1 & w_2 u_2 & w_3 u_3 & w_4 u_4 \\ w_1 v_1 & w_2 v_2 & w_3 v_3 & w_4 v_4 \\ w_1 & w_2 & w_3 & w_4 \end{bmatrix} = H \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ 1 & 1 & 1 & 1 \end{bmatrix} \quad (\text{D.1})$$

when the quantities u_n, v_n, x_n and y_n are known. Multiplying out to obtain u_1 ,

$$u_1 = \frac{w_1 u_1}{w_1} \quad (\text{D.2})$$

$$= \frac{H_{11}x_1 + H_{12}y_1 + H_{13}}{H_{31}x_1 + H_{32}y_1 + H_{33}}. \quad (\text{D.3})$$

Re-arranging this in terms of the elements of H

$$H_{11}x_1 + H_{12}y_1 + H_{13} - H_{31}x_1u_1 - H_{32}y_1u_1 - H_{33}u_1 = 0 \quad (\text{D.4})$$

and similarly for v_1

$$H_{21}x_1 + H_{22}y_1 + H_{23} - H_{31}x_1v_1 - H_{32}y_1v_1 - H_{33}v_1 = 0. \quad (\text{D.5})$$

The known quantities in Equation (D.1) thus yield eight equations, which can be written as $A\mathbf{h} = \mathbf{0}$:

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1u_1 & -y_1u_1 & -u_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1v_1 & -y_1v_1 & -v_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2u_2 & -y_2u_2 & -u_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2v_2 & -y_2v_2 & -v_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3u_3 & -y_3u_3 & -u_3 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3v_3 & -y_3v_3 & -v_3 \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4u_4 & -y_4u_4 & -u_4 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4v_4 & -y_4v_4 & -v_4 \end{bmatrix} \begin{pmatrix} H_{11} \\ H_{12} \\ H_{13} \\ H_{21} \\ H_{22} \\ H_{23} \\ H_{31} \\ H_{32} \\ H_{33} \end{pmatrix} = \mathbf{0}. \quad (\text{D.6})$$

These equations can be solved subject to $|\mathbf{h}| = 1$ by finding \mathbf{h} in the null-space of A using the Singular Value Decomposition (SVD) of A :

$$A = U\Lambda V^T \quad (\text{D.7})$$

\mathbf{h} is then the column of V corresponding to the smallest singular value. The homography could be further refined by nonlinear optimisation, however this is not required for the application presented here.

D.2 Estimating Pose from a Homography

This section describes a method of estimating an SE3 coordinate frame transformation from a homography. Given knowledge of the coordinates $\{x_{\mathcal{P}}, y_{\mathcal{P}}\}$ of coplanar points at $z_{\mathcal{P}} = 0$ in a frame P and knowledge of the normalised projections $\{u', v'\}$ in frame \mathcal{C} (this coordinate frame corresponds to a camera observing the coplanar points), a homography H can be computed such that

$$\begin{bmatrix} w_1u'_1 & \dots & w_nu'_n \\ w_1v'_1 & \dots & w_nv'_n \\ w_1 & & w_n \end{bmatrix} = H \begin{bmatrix} x_{\mathcal{P}1} & \dots & x_{\mathcal{P}n} \\ y_{\mathcal{P}1} & \dots & y_{\mathcal{P}n} \\ 1 & & 1 \end{bmatrix}. \quad (\text{D.8})$$

From this, an estimate of the transformation $E_{\mathcal{CP}}$ is desired. Applying $E_{\mathcal{CP}}$ to the points in frame P , we require

$$\begin{bmatrix} w_1u'_1 & \dots & w_nu'_n \\ w_1v'_1 & \dots & w_nv'_n \\ w_1 & & w_n \\ 1 & & 1 \end{bmatrix} = \begin{bmatrix} x_{\mathcal{C}1} & \dots & x_{\mathcal{C}n} \\ y_{\mathcal{C}1} & \dots & y_{\mathcal{C}n} \\ z_{\mathcal{C}1} & & z_{\mathcal{C}n} \\ 1 & & 1 \end{bmatrix} = E_{\mathcal{CP}} \begin{bmatrix} x_{\mathcal{P}1} & \dots & x_{\mathcal{P}n} \\ y_{\mathcal{P}1} & \dots & y_{\mathcal{P}n} \\ 0 & \dots & 0 \\ 1 & & 1 \end{bmatrix}. \quad (\text{D.9})$$

Comparing the structure of E_{CP} and H ,

$$E = \begin{bmatrix} | & | & | & | \\ r_1 & r_2 & r_3 & t \\ | & | & | & | \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (D.10)$$

$$H = \begin{bmatrix} | & | & | \\ h_1 & h_2 & h_3 \\ | & | & | \end{bmatrix} \quad (D.11)$$

reveals that h_1 and h_2 are the mapped directions of the plane's x - and y -axes in coordinate frame \mathcal{C} . This suggests that an approximation to E_{CP} can be formed by setting $t = h_3$ and using h_1 and h_2 as the basis for the rotation component of the SE3 matrix (having first scaled H so that the resulting rotation has determinant one.) In practice h_1 and h_2 are rarely perfectly orthogonal, and the rotation matrix may be found from h_1 and h_2 by Gram-Schmidt orthogonalisation, or more commonly by minimising the Frobenius-norm difference between $[h_1 h_2]$ and two computed orthonormal vectors $[r_1 r_2]$ (Sturm, 2000).

Here an alternative approach is used. Writing the Homography as

$$H = \begin{bmatrix} -a- & | \\ -b- & t \\ -c- & | \end{bmatrix} \quad (D.12)$$

the 2-vector c represents $\begin{bmatrix} \frac{\partial w}{\partial x} & \frac{\partial w}{\partial y} \end{bmatrix}$; for problems in which the plane in \mathcal{P} is close to parallel to the imaging plane, these parameters are small, and estimated poorly compared to the larger-valued a and b . Here the values of c are re-estimated, and the original values are only used to resolve a directional ambiguity.

Consider the SVD of $\begin{bmatrix} -a- \\ -b- \end{bmatrix}$:

$$\begin{bmatrix} -a- \\ -b- \end{bmatrix} = U \Lambda V^T \quad (D.13)$$

Where Λ is the diagonal matrix of singular values λ_1 and λ_2 with $\lambda_1 > \lambda_2$. The homography H has a scale freedom; before proceeding further, H is re-scaled by $1/\lambda_1$,

such that the SVD now becomes

$$\begin{bmatrix} -\mathbf{a}- \\ -\mathbf{b}- \end{bmatrix} = U \begin{bmatrix} 1 & 0 \\ 0 & \lambda \end{bmatrix} V^T \quad (\text{D.14})$$

$$= \begin{bmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & \lambda \end{bmatrix} \begin{bmatrix} \mathbf{v}_1 & \mathbf{v}_2 \end{bmatrix}^T \quad (\text{D.15})$$

The first column \mathbf{v}_1 of V , which corresponds to the unity singular value of the SVD, is the direction in the \mathcal{P}_{xy} -plane which produces the maximum change in x and y in frame \mathcal{C} . Geometrically, this maximum change in $x_{\mathcal{C}}$ and $y_{\mathcal{C}}$ occurs when there is no change in $z_{\mathcal{C}}$; the direction \mathbf{v}_1 in \mathcal{P} projects to a direction parallel to the image plane in \mathcal{C} .

The homography can be re-written as

$$H = H \begin{bmatrix} V & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} V^T & 0 \\ 0 & 1 \end{bmatrix} \quad (\text{D.16})$$

$$= \begin{bmatrix} -\mathbf{a}V- & | \\ -\mathbf{b}V- & \mathbf{t} \\ -\mathbf{c}V- & | \end{bmatrix} \begin{bmatrix} V^T & 0 \\ 0 & 1 \end{bmatrix} \quad (\text{D.17})$$

To obtain an orthonormal basis for the SE3's rotation matrix, c is now replaced with values which will be chosen to orthonormalise the first two columns of H . This results in a matrix H' from which an SE3 transformation is readily found.

$$H' = \begin{bmatrix} | & | & | \\ \mathbf{r}_1 & \mathbf{r}_2 & \mathbf{t} \\ | & | & | \end{bmatrix} \quad (\text{D.18})$$

$$= \begin{bmatrix} -\mathbf{a}V- & | \\ -\mathbf{b}V- & \mathbf{t} \\ \alpha & \beta & | \end{bmatrix} \begin{bmatrix} V^T & 0 \\ 0 & 1 \end{bmatrix} \quad (\text{D.19})$$

$$= \begin{bmatrix} u_{11} & \lambda u_{12} & | \\ u_{21} & \lambda u_{22} & \mathbf{t} \\ \alpha & \beta & | \end{bmatrix} \begin{bmatrix} V^T & 0 \\ 0 & 1 \end{bmatrix} \quad (\text{D.20})$$

The matrix V^T can be considered a rotation matrix which aligns plane coordinates in the frame \mathcal{P} to the axes \mathbf{v}_1 and \mathbf{v}_2 . The column $[u_{11} \ u_{21} \ \alpha]^T$ above then corresponds to the direction in \mathcal{C} of the direction \mathbf{v}_1 in the \mathcal{P}_{xy} plane. As noted earlier, this direction in \mathcal{C} is parallel to the image- (i.e. $z_{\mathcal{C}}$) plane; therefore $\alpha = 0$.

To calculate β , note that the second column $\begin{bmatrix} \lambda u_{12} & \lambda u_{22} & \beta \end{bmatrix}^T$ must be a unit vector, hence $\beta = \sqrt{1 - \lambda^2}$ since $(u_{12}^2 + u_{22}^2) = 1$. The sign of the root is chosen such that the dot-product of $\begin{bmatrix} 0 & \beta \end{bmatrix} V^T$ and c is positive.

Once H' has been determined, $r_3 = r_1 \times r_2$ completes the transformation matrix E_{CP} . The estimate may still be inaccurate and nonlinear optimisation of reprojection error is used to refine the transformation.

Bibliography

- Armstrong, M. & Zisserman, A.** (1995). Robust object tracking. In *Proc. Asian Conference on Computer Vision*, vol. I, pp 58–61. 2.1.1
- Auer, T., Brantner, S. & Pinz, A.** (1999). The integration of optical and magnetic tracking for multi-user augmented reality. In M. Gervaut, D. Schmalstieg & A. Hildebrand, eds., *Virtual Environments '99. Proc. of the Eurographics Workshop*, pp 43–52, Vienna. 2.2.5
- Azuma, R. & Bishop, G.** (1994). Improving static and dynamic registration in an optical see-through HMD. In *Proc. SIGGRAPH '94*, pp 197–204. 2.3.3
- Azuma, R., Baillot, Y., Behringer, R., Feiner, S., Julier, S. & Macintyre, B.** (2001). Recent advances in augmented reality. *IEEE Computer Graphics and Applications*, pp 34–47. 2.2
- Baillot, Y., Julier, S., Brown, D. & Livingston, M.** (2003). A tracker alignment framework for augmented reality. In *Proc. 2nd IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'03)*, pp 142–150, Tokyo. 6.6.5
- Bajura, M. & Neumann, U.** (1995). Dynamic registration correction in video-based augmented reality systems. *IEEE Computer Graphics and Applications*, **15**, pp 52–61. 2.2.5
- Behringer, R., Park, J. & Sundareswaran, V.** (2002). Model-based visual tracking for outdoor augmented reality. In *Proc. IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'02)*, Darmstadt, Germany. 2.1.2

- Berger, M.O.** (1997). Resolving occlusion in augmented reality: a contour based approach without 3D reconstruction. In *Proc. IEEE Intl. Conference on Computer Vision and Pattern Recognition (CVPR '97)*, 91, IEEE Computer Society. 2.4
- Bimber, O. & Fröhlich, B.** (2002). Occlusion shadows: Using projected light to generate realistic occlusion effects for view-dependent optical see-through displays. In *Proc. IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'02)*, Darmstadt, Germany. 2.3.5
- Bimber, O., Fröhlich, B., Schmalstieg, D. & Encarnaçao, L.M.** (2001). The virtual showcase. *IEEE Computer Graphics and Applications*, **21**, pp 48–55. 2.3.5
- Breen, D., Whitaker, R., Rose, E. & Tuceryan, M.** (1996). Interactive occlusion and automatic object placement for augmented reality. In *Proc. of Eurographics*, pp 11–22, Poitiers, France. 2.4
- Cakmakci, O., Ha, Y. & Rolland, J.P.** (2004). A compact optical see-through head-worn display with occlusion support. In *Proc. 3rd IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'04)*, pp 16–25, Arlington, VA. 2.3.2
- Canny, J.** (1986). A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, **8**, pp 679–698. 7.2.2
- Cho, Y. & Neumann, U.** (1998). Multi-ring color fiducial systems for scalable fiducial tracking augmented reality. In *Proc. Virtual Reality Annual International Symposium (VRAIS'98)*, 212, IEEE Computer Society, Washington, DC. 2.2.1
- Cho, Y., Park, J. & Neumann, U.** (1997). Fast color fiducial detection and dynamic workspace extension in video see-through self-tracking augmented reality. In *Proc. 5th Pacific Conference on Computer Graphics and Applications (PG'97)*, 168, IEEE Computer Society, Washington, DC, USA. 2.2.1
- Cipra, T. & Romera, R.** (1991). Robust kalman filter and its application in time series analysis. *Kybernetika*, **27**, pp 481–494. 6.10.3
- Claus, D. & Fitzgibbon, A.W.** (2004). Reliable fiducial detection in natural scenes. In *Proc. 8th European Conference on Computer Vision (ECCV'04)*, vol. 3022, pp 469–480, Prague. 2.5

- Comport, A., Kragic, D., Marchand, E. & Chaumette, F.** (2005). Robust real-time visual tracking: Comparison, theoretical analysis and performance evaluation. In *IEEE Int. Conf. on Robotics and Automation, ICRA'05*, pp 2852–2857, Barcelona, Spain. 2.1.2
- Davison, A.J.** (2003). Real-time simultaneous localisation and mapping with a single camera. In *Proc. 9th IEEE International Conference on Computer Vision (ICCV'03)*, pp 1403–1410, Nice. 2.1.3
- Drummond, T. & Cipolla, R.** (1999). Real-time tracking of complex structures with on-line camera calibration. In *Proc. British Machine Vision Conference (BMVC'99)*, vol. 2, pp 574–583, BMVA, Nottingham. 2.1.1, 2.1.2, 4.1, 4.2.1, 4.5, 5.6.3
- Espiau, B. & Chaumette, F.** (1992). A new approach to visual servoing. *IEEE Transactions on Robotics and Automation*, **8**, pp 313–326. 2.1.2
- Favaro, P., Burger, M. & Soatto, S.** (2004). Scene and motion reconstruction from defocused and motion-blurred images via anisotropic diffusion. In *Proc. 8th European Conference on Computer Vision (ECCV'04)*, vol. 3022, pp 257–269, Prague. 2.5
- Feiner, S., Macintyre, B. & Seligmann, D.** (1993). Knowledge-based augmented reality. *Communications of the ACM*, **36**, pp 53–62. 1.4, 5.3
- Ferrin, F.J.** (1991). Survey of helmet tracking technologies. In *Proc. SPIE Large Screen Projection, Avionic, and Helmet-Mounted Displays*, vol. 1456, pp 86–94. 2.2.2
- Fischer, J., Regenbrecht, H. & Baratoff, G.** (2003). Detecting dynamic occlusion in front of static backgrounds for AR scenes. In *EGVE '03: Proceedings of the workshop on Virtual environments 2003*, pp 153–161, ACM Press, New York. 2.4, 2.4
- Fischler, M. & Bolles, R.** (1981). Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, **24**, pp 381–395. 2.1.1, 7.2.2
- Fuhrmann, A., Hesina, G., Faure, F. & Gervautz, M.** (1999a). Occlusion in collaborative augmented environments. In *Proc. 5th EUROGRAPHICS Workshop on Virtual Environments*, Vienna. 2.4

- Fuhrmann, A., Schmalstieg, D. & Purgathofer, W.** (1999b). Fast calibration for augmented reality. In *VRST '99: Proceedings of the ACM symposium on Virtual reality software and technology*, pp 166–167, ACM Press, New York, NY, USA. 2.3.3
- Fuhrmann, A., Schmalstieg, D. & Purgathofer, W.** (2000). Practical calibration procedures for augmented reality. In *Proc. 6th Eurographics Workshop on Virtual Environments*. 2.3.3
- Gausemeier, J., Fründ, J., Matyszczok, C., Brüderlin, B. & Beier, D.** (2003). Development of a real time image based object recognition method for mobile AR-devices. In *Proc. 2nd ACM International Conference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa (SIGGRAPH AFRIGRAPH'03)*, Cape Town. 2.3.4
- Geiger, C., Kleinjohann, B., Reimann, C. & Stichling, D.** (2000). Mobile AR4ALL. In *Proc. IEEE and ACM International Symposium on Augmented Reality (ISAR'00)*, pp 181–182, Munich. 2.3.4
- Genc, Y., Sauer, F., Wenzel, F., Tuceryan, M. & Navab, N.** (2000). Optical see-through HMD calibration: A novel stereo method validated with a video see-through system. In *Proc. IEEE and ACM International Symposium on Augmented Reality (ISAR'00)*, pp 165–174, Munich. 2.3.3
- Gennery, D.** (1992). Visual tracking of known three-dimensional objects. *Int. Journal of Computer Vision*, **7**:3, pp 243–270. 2.1.1
- Ghosh, S.** (1988). *Analytical Photogrammetry, 2nd Edition*. Pergamon Press. 4.2.2
- Gordon, I. & Lowe, D.** (2004). Scene modelling, recognition and tracking with invariant image features. In *Proc. 3rd IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'04)*, pp 110–119, Arlington, VA. 2.5
- Hampel, F.** (1974). The influence curve and its role in robust estimation. *Journal of the American Statistical Association*, **69**, pp 383–393. C
- Harris, C.** (1992). Tracking with rigid models. In A. Blake, ed., *Active Vision*, chap. 4, pp 59–73, MIT Press. 2.1.1

- Hoff, W., Nguyen, K. & Lyon, T.** (1996). Computer vision-based registration techniques for augmented reality. *Intelligent Robots and Computer Vision XV*, **2904**, pp 538–548. 2.2.1
- Holloway, R.** (1995). *Registration Errors in Augmented Reality Systems*. Ph.D. thesis, University of North Carolina at Chapel Hill. 1.2, 5.6.3
- Huber, P.** (1981). *Robust Statistics*. Wiley. C
- Huber, P.J.** (1964). Robust estimation of a location parameter. *Annals of Mathematical Statistics*, pp 73–101. C
- Isard, M. & Blake, A.** (1998). CONDENSATION - conditional density propagation for visual tracking. *International Journal of Computer Vision*, **29**, pp 5–28. 2.1.3
- Janin, A.L., Mizell, D.W. & Caudell, T.P.** (1993). Calibration of head-mounted displays for augmented reality applications. In *Proc. IEEE Virtual Reality Annual International Symposium (VR'93)*, pp 246–255, Seattle. 2.3.3, 5.5.2, 5.5.2
- Kalman, R.** (1960). A new approach to linear filtering and prediction problems. *ASME Journal of Basic Engineering*, **82**, pp 35–45. 2.1.1, 6.6.1
- Kanbara, M., Okuma, T., Takemura, H. & Yokoya, N.** (2000). A stereoscopic video see-through augmented reality system based on real-time vision-based registration. In *Proc. IEEE Virtual Reality 2000 (VR2000)*, pp 255–262. 2.4
- Kanbara, M., Fujii, H., Takemura, H. & Yokoya, N.** (2001). A stereo vision-based mixed reality system with natural feature point tracking. In *The Second International Symposium on Mixed Reality (ISMР'01)*, pp 56–63. 2.2.3
- Kano, H., Kitabayashi, K. & Kijima, R.** (2004). Reflex head mounted display: Head mounted display for virtual reality with time lag compensation. In *Proc. Tenth International Conference on Virtual Systems and Multimedia (VSMM'04)*, pp 119–127. 2.3.2
- Karitsuka, T. & Sato, K.** (2003). A wearable mixed reality with an on-board projector. In *Proc. 2nd IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'03)*, pp 321–322, Tokyo. 2.3.5

- Kato, H. & Billinghurst, M.** (1999). Marker tracking and HMD calibration for a video-based augmented reality conferencing system. In *Proc. 2nd Int'l Workshop on Augmented Reality*, pp 85–94, San Francisco, CA. 2.2.1
- Kemp, C. & Drummond, T.** (2004). Multi-modal tracking using texture changes. In *Proc. British Machine Vision Conference (BMVC'04)*, BMVA, London. 2.1.3
- Kemp, C. & Drummond, T.** (2005). Dynamic measurement clustering to aid real time tracking. In *Proc. 10th IEEE International Conference on Computer Vision (ICCV'05)*, vol. 2, pp 1500–1507, Beijing. 2.1.3
- Kijima, R. & Ojika, T.** (2002). Reflex HMD to compensate lag and correction of derivative deformation. In *Proc. IEEE Virtual Reality Conference (VR'02)*, pp 172–179, IEEE Computer Society, Washington, DC, USA. 2.3.2
- Kiyokawa, K., Billinghurst, M., Campbell, B. & Woods, E.** (2003). An occlusion-capable optical see-through head mount display for supporting co-located collaboration. In *Proc. 2nd IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'03)*, pp 133–142, Tokyo. 2.3.2, 2.4
- Klein, G.** (2001). Visual guidance of a mobile robot. Fourth year project, Cambridge University Engineering Department. 4.1, 4.2.1, 4.5
- Klein, G. & Drummond, T.** (2002). Tightly integrated sensor fusion for robust visual tracking. In *Proc. British Machine Vision Conference (BMVC'02)*, vol. 2, pp 787 –796, BMVA, Cardiff. 1.8
- Klein, G. & Drummond, T.** (2003). Robust visual tracking for non-instrumented augmented reality. In *Proc. 2nd IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'03)*, pp 113–122, Tokyo. 1.8
- Klein, G. & Drummond, T.** (2004a). Sensor fusion and occlusion refinement for tablet-based AR. In *Proc. 3rd IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'04)*, pp 38–47, Arlington, VA. 1.8
- Klein, G. & Drummond, T.** (2004b). Tightly integrated sensor fusion for robust visual tracking. *Image and Vision Computing*, **22**, pp 769–776. 1.8

- Klein, G. & Drummond, T.** (2005). A single-frame visual gyroscope. In *Proc. British Machine Vision Conference (BMVC'05)*, vol. 2, pp 529–538, BMVA, Oxford. 1.8
- Koller, D., Klinker, G., Rose, E., Breen, D., Whitaker, R. & Tuceryan, M.** (1997). Real-time vision-based camera tracking for augmented reality applications. In D. Thalmann, ed., *ACM Symposium on Virtual Reality Software and Technology*, ACM Press, New York, NY. 2.2.1
- Kumar, P.** (2001). Visual tracking with inertial guidance. Fourth year project, Cambridge University Engineering Department. 4.3
- Lepetit, V. & Berger, M.O.** (2000). Handling occlusions in augmented reality systems: A semi-automatic method. In *Proc. IEEE and ACM International Symposium on Augmented Reality (ISAR'00)*, pp 197–146, Munich. 2.4
- Lin, H.** (2005). Vehicle speed detection and identification from a single motion blurred image. In *Proc. Seventh IEEE Workshop on Application of Computer Vision (WACV/MOTION'05)*, vol. 1, pp 461–467, Breckenridge, CO. 2.5
- Lowe, D.** (1992). Robust model-based motion tracking through the integration of search and estimation. *Intl. Journal of Computer Vision*, **8**, pp 113–122. 2.1.1
- Lowe, D.** (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, **60**, pp 91–100. 2.5, 4.1
- MacWilliams, A., Sandor, C., Wagner, M., Bauer, M., Klinker, G. & Brügge, B.** (2003). Herding sheep: Live system development for distributed augmented reality. In *Proc. 2nd IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'03)*, Tokyo. 2.3.4
- Marchand, E. & Chaumette, F.** (2002). Virtual visual servoing: a framework for real-time augmented reality. In G. Drettakis & H. Seidel, eds., *Proc. Eurographics 2002*, vol. 21,3, pp 289–298, Saarbrücken, Germany. 2.1.2
- Marchand, E., Bouthemy, P., Chaumette, F. & Moreau, V.** (1999). Robust real-time visual tracking using a 2D-3D model-based approach. In *Proc. 7th IEEE International Conference on Computer Vision (ICCV'99)*, vol. 1, pp 262–268, Kerkyra, Greece. 2.1.1

- Matsushita, N., Hihara, D., Ushiro, T., Yoshimura, S., Rekimoto, J. & Yamamoto, Y.** (2003). ID CAM: A smart camera for scene capturing and ID recognition. In *Proc. 2nd IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'03)*, pp 227–236, Tokyo. 2.2.2
- Maybeck, P.** (1979). *Stochastic models, estimation and control*, vol. 1, chap. 1. Academic Press. 6.6.1
- Mellor, J.P.** (1995). Enhanced reality visualization in a surgical environment. A.I. Technical Report 1544, Massachusetts Institute of Technology, Artificial Intelligence Laboratory. 2.2.1
- Möhring, M., Lessig, C. & Bimber, O.** (2004). Video see-through AR on consumer cell-phones. In *Proc. 3rd IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'04)*, pp 252–253, Arlington, VA. 2.3.4
- Molana, R.** (2000). Visual guidance of a mobile robot. Fourth year project, Cambridge University Engineering Department. 4.1, 4.2.1
- Mulder, J.D.** (2005). Realistic occlusion effects in mirror-based co-located augmented reality systems. In *Proc. IEEE Virtual Reality Conference (VR 2005)*, pp 203–208, Bonn. 2.4
- Naimark, L. & Foxlin, E.** (2002). Circular data matrix fiducial system and robust image processing for a wearable vision-inertial self-tracker. In *Proc. IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'02)*, pp 27–36, Darmstadt, Germany. 2.2.1
- Neumann, U. & Cho, Y.** (1996). A selftracking augmented reality system. In *Proc. ACM Symposium on Virtual Reality Software and Technology (VRST'96)*, pp 109–115. 2.2.1
- Newman, J., Ingram, D. & Hopper, A.** (2001). Augmented reality in a wide area sentient environment. In *Proc. IEEE and ACM International Symposium on Augmented Reality (ISAR'01)*, New York. 2.3.4
- Owen, C., Xiao, F. & Middlin, P.** (2002). What is the best fiducial? In *Proc. First IEEE International Augmented Reality Toolkit Workshop*, pp 98–105, Darmstadt. 2.2.1, 2.3.4

- Owen, C.B., Zhou, J., Tang, A. & Xiao, F.** (2004). Display-relative calibration for optical see-through head-mounted displays. In *Proc. 3rd IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'04)*, pp 70–78, Arlington, VA. 2.3.3, 8.3
- Park, F. & Martin, B.** (1994). Robot sensor calibration: Solving $AX=XB$ on the euclidean group. *IEEE Transactions on Robotics and Automation*, **10**, pp 717–721. 6.6.5
- Park, J., You, S. & Neumann, U.** (1998). Natural feature tracking for extendible robust augmented realities. In *Proc. Int. Workshop on Augmented Reality*. 2.2.3, 2.2.4
- Pasman, W. & Woodward, C.** (2003). Implementation of an augmented reality system on a PDA. In *Proc. 2nd IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'03)*, pp 276–277, Tokyo. 2.3.4
- Pingle, K.K.** (1969). Visual perception by a computer. In A. Grasselli, ed., *Automatic Interpretation and Classification of Images*, pp 277–284, Academic Press, New York. 7.2.2
- Pupilli, M. & Calway, A.** (2005). Real-time camera tracking using a particle filter. In *Proc. British Machine Vision Conference (BMVC'05)*, pp 519–528, BMVA, Oxford. 2.1.3
- Rekimoto, J.** (1995). The magnifying glass approach to augmented reality systems. In *Proc. International Conference on Artificial Reality and Tele-Existence Conference on Virtual Reality Software and Technology (ICAT/VRST'95)*, pp 123–132. 2.3.4
- Rekleitis, I.** (1996). Steerable filters and cepstral analysis for optical flow calculation from a single blurred image. In *Vision Interface*, pp 159–166, Toronto. 2.5, 7.2.3
- Rolland, J., Davis, L. & Baillot, Y.** (2000). A survey of tracking technologies for virtual environments. In W. Barfield & T. Caudell, eds., *Fundamentals of Wearable Computers and Augmented Reality*, chap. 3, Lawrence Erlbaum Assoc. 2.2
- Rosten, E. & Drummond, T.** (2005). Fusing points and lines for high performance tracking. In *Proc. 10th IEEE International Conference on Computer Vision (ICCV'05)*, vol. 2, pp 1508–1515, Beijing. 2.1.3, 1

- Satoh, K., Uchiyama, S., Yamamoto, H. & Tamura, H.** (2003). Robust vision-based registration utilizing bird's-eye view with user's view. In *Proc. 2nd IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'03)*, Tokyo. 2.2.5
- Shahrokni, A., Drummond, T. & Fua, P.** (2004). Texture boundary detection for real-time tracking. In *Proc. 8th European Conference on Computer Vision (ECCV'04)*, vol. 3022, pp 566–577, Prague. 2.1.3
- Simon, G. & Berger, M.O.** (1998). A two-stage robust statistical method for temporal registration from features of various type. In *Proc. 6th IEEE International Conference on Computer Vision (ICCV'98)*, pp 261–266, Bombay. 2.1.1
- Smith, P.A.** (2001). *Edge-based Motion Segmentation*. Ph.D. thesis, University of Cambridge, UK. C
- State, A., Hirota, G., Chen, D., Garrett, W. & Livingston, M.** (1996). Superior augmented reality tracking by integrating landmark tracking and magnetic tracking. In *Proc. SIGGRAPH'96*, pp 429–438. 2.2.5
- Stricker, D., Klinker, G. & Reiners, D.** (1998). A fast and robust line-based optical tracker for augmented reality applications. In *Proc. First International Workshop on Augmented Reality (IWAR'98)*, pp 129–145, AK Peters, San Francisco. 2.2.1, 2.4
- Sturm, P.** (2000). Algorithms for plane-based pose estimation. In *Proc. IEEE Intl. Conference on Computer Vision and Pattern Recognition (CVPR'00)*, pp 1010–1017, Hilton Head Island, South Carolina. D.2
- Sundareswaran, V. & Behringer, R.** (1998). Visual servoing-based augmented reality. In *Proc. First IEEE Workshop on Augmented Reality (IWAR'98)*, San Francisco. 2.1.2
- Sutherland, I.E.** (1968). A head-mounted three-dimensional display. In *Proc. Fall Joint Computer Conference*, pp 757–764, Washington, D.C. 2.3.2
- Tang, A., Zhou, J. & Owen, C.B.** (2003). Evaluation of calibration procedures for optical see-through head-mounted displays. In *Proc. 2nd IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'03)*, pp 161–168, Tokyo. 2.3.3

- Tomlin, C. & Sastry, S.** (1995). Control of systems on Lie groups. In S. Sastry, ed., *Memorandum No. UCB/ERL M95/8, Advanced Topics in Adaptive and Nonlinear Control*, University of California at Berkeley. 3.2
- Tsai, R.** (1987). 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*, RA-3, pp 323–344. 2.3.3
- Tuceryan, M. & Navab, N.** (2000). Single point active alignment method (spaam) for optical see-through HMD calibration for AR. In *Proc. IEEE and ACM International Symposium on Augmented Reality (ISAR'00)*, pp 149–158, Munich. 2.3.3
- Tukey, J.** (1960). A survey of sampling from contaminated distributions. In I. Olkin, ed., *Contributions to Probability and Statistics*, pp 448–485, Stanford University Press. C
- Vacchetti, L., Lepetit, V. & Fua, P.** (2004). Combining edge and texture information for real-time accurate 3D camera tracking. In *Proc. 3rd IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'04)*, pp 48–57, Arlington, VA. 2.1.3
- Varadarajan, V.** (1974). *Lie Groups, Lie Algebras and Their Representations*. No. 102 in Graduate Texts in Mathematics, Springer-Verlag. 3.2
- Vlahakis, V., Ioannidis, N., Karigiannis, J., Tsotros, M. & Gounaris, M.** (2002). Virtual reality and information technology for archaeological site promotion. In *Proc. 5th International Conference on Business Information Systems (BIS02)*, Poznan, Poland. 2.3.4
- Wagner, D. & Schmalstieg, D.** (2003a). ARToolKit on the PocketPC platform. Tech. Rep. TR-188-2-2003-23, Technical University of Vienna. 2.3.4
- Wagner, D. & Schmalstieg, D.** (2003b). First steps towards handheld augmented reality. In *7th Intl. Symposium on Wearable Computers (ISWC'03)*, pp 127–137, White Plains, NY. 2.3.4

- Wagner, D., Pintaric, T., Ledermann, F. & Schmalstieg, D.** (2005). Towards massively multi-user augmented reality on handheld devices. In *Proc. Third Intl. Conference on Pervasive Computing (Pervasive'05)*, pp 208–219, Munich. 2.3.4
- Wang, J., Azuma, R., Bishop, G., Chi, V., Eyles, J. & Fuchs, H.** (1990). Tracking a head-mounted display in a room-sized environment with head-mounted cameras. In *Proc. SPIE Helmet-Mounted Displays II*, vol. 1290, pp 47–57, Orlando, FL. 2.2.2
- Ward, M., Azuma, R., Bennett, R., Gottschalk, S. & Fuchs, H.** (1992). A demonstrated optical tracker with scalable work area for head-mounted display systems. In *Proc. 1992 Symposium on Interactive 3D graphics (SI3D'92)*, pp 43–52, ACM Press, New York, NY, USA. 2.2.2
- Watson, B. & Hedges, F.** (1995). Using texture maps to correct for optical distortion in head-mounted displays. In *Proc. IEEE Virtual Reality Annual Symposium (VRAIS'95)*, pp 172–178. 5.4, 1, 6.8
- Welch, G. & Bishop, G.** (1995). An introduction to the Kalman filter. Tech. Rep. TR 95-041, University of North Carolina at Chapel Hill, updated 2002. 6.6.1
- Welch, G. & Bishop, G.** (1997). SCAAT: incremental tracking with incomplete information. In *Proc. 24th annual conference on Computer graphics and interactive techniques (SIGGRAPH '97)*, pp 333–344, ACM Press, New York. 2.2.2
- Welch, G., Bishop, G., Vicci, L., Brumback, S., Keller, K. & Colucci, D.** (1999). The HiBall tracker: High-performance wide-area tracking for virtual and augmented environments. In *Proc. ACM Symposium on Virtual Reality Software and Technology*. 2.2.2, 6.4
- Wloka, M. & Anderson, B.** (1995). Resolving occlusion in augmented reality. In *Proc. Symposium on Interactive 3D Graphics*, pp 5–12, New York. 2.4, 6.9
- Yokokohji, Y., Sugawara, Y. & Yoshikawa, T.** (2000). Accurate image overlay on see-through head-mounted displays using vision and accelerometers. In *Proc. IEEE Conference on Virtual Reality*, pp 247–254. 2.2.4
- You, S. & Neumann, U.** (2001). Fusion of vision and gyro tracking for robust augmented reality registration. In *Proc. IEEE Conference on Virtual Reality*, pp 71–78. 2.2.4

- You, S., Neumann, U. & Azuma, R.** (1999). Hybrid inertial and vision tracking for augmented reality registration. In *Proc. IEEE Conference on Virtual Reality*, pp 260–267. 2.2.4
- Zhang, Z.** (1997). Parameter estimation techniques: a tutorial with application to conic fitting. *Image Vision Computing*, **15**, pp 59–76. C, C
- Zhu, W., Owen, C., Li, H. & Lee, J.H.** (2004). Personalized in-store e-commerce with the PromoPad: an augmented reality shopping assistant. *Electronic Journal for E-commerce Tools and Applications*, **1**. 2.3.4