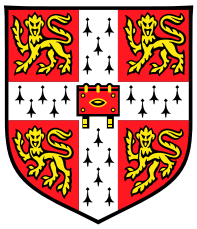
High performance rigid body tracking



Edward Rosten

Churchill College
University of Cambridge

Dissertation submitted for the degree of
Doctor of Philosophy

February 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 2002 and January 2006 under the supervision of Dr T.W. Drummond. The work described is original and a result of my own work, except where stated to the contrary. This dissertation contains 47 figures and is approximately 33,000 words in length. No part of it has already or is being currently submitted for any other qualification at any other university.

Edward Rosten

Acknowledgements

I would sincerely like to thank my supervisor, Tom Drummond and the members of the Fallside Lab: Georg Klein, Christopher Kemp, Timothy Gan, Ethan Eade and Gerhard Reitmayr for making it an amazing and fun place to work, as well as their help with computer vision in general, papers, hacking and bugs, not to mention many entertaining coffee breaks. A mention must also go to Georg Klein for writing "gvars" and saving many hours of compiling and Paul Smith in addition to those named above for their help in making libCVD what it is today.

Thanks must go to Susan for help, patience and taking the time to understand my work.

I finally wish to thank my family, Esther, Oliver and Judith for their support, and dedicate this thesis (and especially its contents page) to my father, Harvey.

Abstract

Many potential applications of computer vision require a 3D tracking system which can cope with very unpredictable, rapid motions of the camera while operating at frame rate. This cannot be achieved using previously available techniques. Robustness in tracking is achieved by combining different systems in a non trivial way. This allows trackers to be designed in such a way that they are not general purpose trackers, but instead excel at some other chosen property.

The problem of robustness is addressed by the development of a point based tracking system which uses high speed techniques for point detection and matching (allowing it to operate on full size frames) combined with a robust optimizer. This, coupled with a system which estimates the quality of a match, allows the system to track very rapid, unpredictable motions with considerable amounts of noise.

The full-frame extraction of feature points is vital to the robustness of the system. Machine learning is used to derive a feature detector which is significantly faster than previous methods. The repeatability of extracted features has been verified by comparison to other detectors. Despite being principally constructed for speed, the detector significantly outperforms existing feature detectors.

The problem of tracking curved surfaces is approached by developing a scheme for rapidly rendering the apparent contour from a predicted pose (a step required for tracking). This is done efficiently with a method for quickly tracing out the apparent contours, combined with family of high speed polygon intersection algorithms and a set of rules which determine visibility from the contour intersections.

Finally, a careful analysis of the properties of the trackers is performed. This is used to design a non-trivial filter for combining the measurements, and it shown that the design choices collectively lead to a very high performance tracker.

0. Contents

List of Figures

LIST OF FIGURES

List of Tables

1. Introduction

Tracking is the act of following an object or objects, so that the pose (and sometimes shape as well) relative to some reference frame is known. The focus of this thesis is tracking three dimensional rigid objects. The domain of this varies from tracking small, manufactured objects, to tracking the pose of the camera by tracking the inside of an immersive environment This is a 6-DOF (six degree of freedom) problem: the position and the orientation (for instance roll, pitch and yaw) both have three degrees of freedom.

Tracking is a useful technology and has many applications in robotics, augmented reality and beyond. These applications often have hard real-time requirements (i.e. a guaranteed upper bound on time taken by the process). Many applications, however, can be redesigned so that this hard requirement can be replaced by a soft real-time requirement in which typical processing speed is important and images are processed on average slightly above video rate so that the time requirement is met *almost* all the time. The techniques in this thesis are not real time but have been designed for video-rate processing and as a result they have been designed with computational efficiency in mind.

Many tracking technologies have been developed, such as as GPS (usually low data rate, works best in outdoor environments), 'Polhemous', 'Flock-of-Birds' and 'Hi-Ball' (provide high data rate and accuracy, but are expensive, and tracking can only work in instrumented environments). Visual tracking has several advantages as cameras are cheap, lightweight and ubiquitous. Techniques for visual tracking of fiducial markers have been developed, one of the most popular being ARToolKit [65]. However, this thesis is about markerless tracking. Markerless tracking does not require instrumentation of the environment, which can be a time consuming process, if it is possible at all: it may be impossible in outdoor environments, or where the environment might not be cooperative (such as in surveillance applications).

The focus of this thesis is high performance tracking, that is tracking which is both robust and computationally efficient. Achieving both robustness and efficiency is a difficult problem, and in fact, of the many trackers which have been presented, one common theme which can be extracted is the tradeoff between efficiency and robustness. This tradeoff will be described below, moving from brittle (but fast) to robust (but slow) tracking systems. Essentially, when tracking an object, the object's real position deviates from the predicted position, and that implies the following question:

How far am I going to look?

With increasing distance come a variety of problems. Larger search distances require that more of the image is being searched, requiring more computational time. When a larger proportion of the image is being searched, the system is more likely to get an incorrect measurement. Dealing with outlying measurements is difficult and furthermore, some techniques simply do not scale beyond certain distances: too many measurements will be incorrect for any system to cope with. In these cases a different (and usually much more complex) technique must be used instead (as described below and in Section 4.2).

There are several examples of the tradeoff in the 6-DOF tracking literature. The field can be split up into *edge-based* tracking (see Section 4.2 for a detailed overview), where tracking relies on strong gradients well localised in one dimension, and *point-based* tracking (see Section 2.2.1), where tracking relies on point-like features which are well localised in two dimensions.

Consider one of the early proposed frame-rate 6-DOF tracking systems, the RAPiD [51] tracking system. The tracking system uses a model to predict where edges should be in the image and a small area around each of the the predicted positions are searched. After searching, the model pose is altered to align predicted edges with the edges found. This tracking system requires very little computation, due to the tiny area of the image which is analysed. Since edges are only localisable in one dimension, only a one dimensional search is required to look for them.

The previous method is only applicable when the distance moved is small, but as the object moves further, and the search distance increases, the probability of finding the wrong edge (for instance due to clutter in the search region) increases greatly. In, [35] the computational resources are spent on longer search distances, and then a robust optimiser which greatly reduces the effect of mis-measured

edges. Extra computation is spent in [95] where the robust optimiser is preceded by a robust optimise on a reduced dimension model, in a coarse-to-fine strategy.

The system presented in [3] instead opts to spend the available computational time on getting better measurements of edges by extracting the edges using a robust optimiser and then fitting the model to the robust edges. [67] improves the search distance further by extracting all nearby edges and assigning each one a probability, before fitting the model to these edges with a robust optimiser. The multi-modal representation of edges makes the system more robust because it allows the correct edge to be present even if there is a dominant incorrect edge, but requires more computational resources.

Even with these techniques, once the distance exceeds a certain size, the assumption that edges in the image will appear near model edges fails. Attempting to use this assumption results in a very large number of bad measurements. One solution is to switch to a whole image approach, in which the entire image (as opposed to small areas near to predicted edges are searched). In [86], edge segments are extracted for the whole image (a process which at the time required specialised hardware for reasonable performance) and the model is fitted to these using a guided search in which model edges are aligned with extracted edge segments. Whereas [68] extracts edgels (as opposed to segments) and finds parts of the model which can be aligned more or less independently, a process requiring considerable computation. This improves robustness a great deal since the alignments operate in less than 6-DOF, so much more thorough searches can be performed.

One primary problem with edge based systems is that under common assumptions about what constitutes an edge, all edges look alike. Others have instead opted for point based tracking, since these features tend to be much easier to correspond (see Section 2.2.2 and 4.2). A good example of this is given in [145]. However, point-like features tend to require considerable computation: they require a two dimensional search to localise in an image, and a further two dimensional search to correspond features between frames. Further robustness can be achieved by expending enough computational resources to examine the entire image for feature points and allowing matching across arbitrary distances (see Section 2). This still requires that at least some part of the object is visible in consecutive frames.

Therefore, an alternative approach is to allow the object to move even further, and this is done by extracting feature points and matching them in to a database, without a strong prior on position. Once the prior contains no information, this technique becomes initialisation every frame. This comes at significant compu-

tational cost. When the database becomes large, matching is not only slow but effort must be expended in extracting features which are distinctive enough the be reasonably unique. The other computational cost arises from the need to match points across large changes in camera position. This requires considerable computational resources to extract features which are not only distinctive but also reasonable invariant to affine changes. Aside from the computational cost, large databases of points are not without their problems as will be discussed in Section 2.2.1. A good example of this type of system is presented in [44], though it still requires more computational resources to run at video-rate than is available in a commodity processor.

The final example to be given for trading computational cost for robustness is that of combining multiple tracking systems, for example [146], and [137]. These systems expend the computational effort in running multiple, more or less independent tracking systems. These then combine the results of the tracking systems, so if one tracker is under constrained (for instance there are too few features of the correct type), the other part will hopefully provide enough information to constrain the whole system properly. In the case of [46], a robust but approximate tracker is used to initialise a brittle but accurate tracker.

In this thesis, to achieve both robustness and performance, multiple tracking systems are used to combine the strengths and relieve most of the weaknesses. Because the tracking systems are to be combined, they no longer need to be built as general purpose system that have to work in as wide a variety of situations as possible. Instead, the tracking systems can each be designed to perform one task as well as possible without regard to some of the failure modes, relying instead on other trackers to make corrections if necessary.

The tracking systems to be combined are model-based in that they require a model of the object being tracked. In particular, a 3D surface model is required. These models are built by physically measuring the object to be tracked and building the model from these measurements 'by hand'.

1.1 Mathematical Tools

The linear algebra operations, including the ones described in this chapter, are provided by the C++ TooN software library [33]. Various matrix decompositions are provided by the BLAS [11] and LAPACK [2] libraries.

The camera modelling is provided as part of the CVD [120] computer vision library. Camera calibration was performed using the method described in [35], using a model with depths ranging from about 10cm to 5m, with the closest part of the model occupying the entire field of view. This is required for both the focal length and radial distortion parameters to be set accurately. A quintic camera model has been used throughout and is described in Section 1.3.

More information on 3D and projective geometry can be found in [53].

1.2 3D geometry

This chapter presents the framework and notation used throughout the rest of this thesis. 3D coordinates are written as homogeneous vectors, such that a point $\mathbf{p}$ at (x, y, z) is written as:

$$\mathbf{p} = \begin{bmatrix} \gamma x \\ \gamma y \\ \gamma z \\ \gamma \end{bmatrix}, \tag{1.1}$$

where γ is a degree of freedom which does not affect position. If $\mathbf{p}$ is in frame $\mathcal{A}$, it is written as (where γ is arbitrarily set to 1):

$$\mathbf{p}_{\mathcal{A}} = \begin{bmatrix} x_{\mathcal{A}} \\ y_{\mathcal{A}} \\ z_{\mathcal{A}} \\ 1 \end{bmatrix}. \tag{1.2}$$

Transforming $\mathbf{p}$ from $\mathcal{A}$ to $\mathcal{B}$ is achieved by multiplication with a 4×4 transformation matrix:

$$\mathbf{p}_{\mathcal{B}} = \mathbf{E}_{\mathcal{B}\mathcal{A}}\mathbf{p}_{\mathcal{A}}, \tag{1.3}$$

where $\mathbf{E}$ represents a Euclidean transformation and has the general form:

$$\mathbf{E} = \begin{bmatrix} & \mathbf{R} & & \mathbf{t} \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{1.4}$$

where $\mathbf{R}$ is a 3×3 rotation matrix (a special orthogonal matrix) and $\mathbf{t}$ is a translation vector in $\mathbb{R}^3$. $\mathbf{E}$, therefore has six degrees of freedom. There are various

ways of parameterising $\mathbf{E}$ (such as using Euler angles or unit quaternions [40] for parameterising $\mathbf{R}$), but in this case, $\mathbf{E}$ is parametrised using the exponential map:

$$\mathbf{E}(\boldsymbol{\mu}) = e^{\sum_{i=1}^{6} \mathbf{G}_i \mu_i}, \tag{1.5}$$

where $\mathbf{G}_1 \dots \mathbf{G}_6$, the generator matrices, are given by:

$$\begin{aligned}
\mathbf{G}_1 &= \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad \mathbf{G}_2 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad \mathbf{G}_3 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \\
\mathbf{G}_4 &= \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad \mathbf{G}_5 = \begin{bmatrix} 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad \mathbf{G}_6 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.
\end{aligned} \tag{1.6}$$

This is the SE(3) Lie group, further information about which can be found in [147]. Although this formulation has singularities, they occur far away from $\boldsymbol{\mu} = \mathbf{0}$. In this thesis, this framework is used to parameterise a small motion, $\mathbf{M}$, which occurs between a pair of frames, so the singularity can be easily avoided. If there is a point $\mathbf{p}_{\mathcal{O}}$ in the frame of the object, computing the motion of the point in the camera frame, $\mathcal{C}$, as the transformation varies by a small motion can be achieved in a straightforward manner by linearising:

$$\left.\frac{\partial \mathbf{p}_{\mathcal{C}}}{\partial \boldsymbol{\mu}}\right|_{\boldsymbol{\mu}=\mathbf{0}} = \left.\frac{\partial}{\partial \boldsymbol{\mu}} \mathbf{M}(\boldsymbol{\mu}) \mathbf{E}_{\mathcal{C}\mathcal{O}} \mathbf{p}_{\mathcal{O}}\right|_{\boldsymbol{\mu}=\mathbf{0}} = \left.\frac{\partial \mathbf{M}}{\partial \boldsymbol{\mu}}\right|_{\boldsymbol{\mu}=\mathbf{0}} \mathbf{E}_{\mathcal{C}\mathcal{O}} \mathbf{p}_{\mathcal{O}}, \tag{1.7}$$

and

$$\left.\frac{\partial \mathbf{M}}{\partial \mu_i}\right|_{\boldsymbol{\mu}=\mathbf{0}} = \mathbf{G}_i. \tag{1.8}$$

This formulation, (premultiplying the translation by the small motion $\mathbf{M}$ and linearising) makes the assumption that the camera is making small motions. Postmultiplying $\mathbf{E}$ by $\mathbf{M}$ would instead make the assumption that the object is making small motions. Making the correct assumption is important for numerical stability: consider the case where a camera makes a small rotation. This is equivalent to the model making a small rotation but a very large translation (if the camera and model centres are far apart). If the motion is computed by an optimizer then the optimizer will have to deal with a large spread in the computed parameters. In optimization schemes such as Gauss-Newton (the scheme predominantly used in this thesis), this results in the system inverting a matrix with a large spread in eigenvalues, which is ill-conditioned.

1.3 Projection and cameras

A 3D point is projected by an idealised camera—a pinhole camera centred at the origin and with unity focal length—to a point (u, v):

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}. \tag{1.9}$$

The derivatives of the projected point positions u and v with respect to $\boldsymbol{\mu}$ can be computed via the quotient rule:

$$\mathbf{J}_p = \frac{\partial \begin{bmatrix} u \\ v \end{bmatrix}}{\partial \boldsymbol{\mu}}, \quad \text{where} \quad \frac{\partial u}{\partial \mu_i} = \frac{z\frac{\partial x}{\partial \mu_i} - x\frac{\partial z}{\partial \mu_i}}{z^2}, \text{ etc.} \ldots \tag{1.10}$$

This computes the derivatives of a point in an ideal camera with respect to a motion. However, real cameras are not ideal. Often, they are modelled as linear where the pixel coordinates $[u', v']$ are given by:

$$\begin{bmatrix} u' \\ v' \end{bmatrix} = \begin{bmatrix} f_u & 0 & u_0 \\ 0 & f_v & v_0 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}. \tag{1.11}$$

However, this model is unsuitable for cameras with large amounts of radial distortion, such cameras with wide angle lenses. These are more suitably modelled by:

$$\begin{bmatrix} u' \\ v' \end{bmatrix} = \begin{bmatrix} f_u & 0 & u_0 \\ 0 & f_v & v_0 \end{bmatrix} \begin{bmatrix} \gamma u \\ \gamma v \\ 1 \end{bmatrix}, \tag{1.12}$$

where

$$\gamma = 1 + \alpha r^2 + \beta r^4, \tag{1.13}$$

and

$$r = \sqrt{u^2 + v^2}. \tag{1.14}$$

The Jacobian, $\mathbf{J}$, of the pixel coordinates, the derivatives of the pixel coordinates with respect to the motion of the 3D point, are computed by application of the chain rule:

$$\mathbf{J} = \mathbf{J}_c|_{[u,v]} \mathbf{J}_p, \tag{1.15}$$

where $\mathbf{J}_c$ is the camera Jacobian:

$$\mathbf{J}_c = \frac{\partial \begin{bmatrix} u' \\ v' \end{bmatrix}}{\partial [u, v]} = \begin{bmatrix} \dfrac{\partial u'}{\partial u} & \dfrac{\partial u'}{\partial v} \\ \dfrac{\partial v'}{\partial u} & \dfrac{\partial v'}{\partial v} \end{bmatrix}. \tag{1.16}$$

The full nonlinear projection of a point (x, y, z) in $\mathcal{C}$ is denoted by the function P:

$$\begin{bmatrix} u' \\ v' \end{bmatrix} = P([x, y, z, 1]^{\mathsf{T}}). \tag{1.17}$$

1.4 Layout of thesis

Chapter 2 presents a new feature based tracker which aims to be as robust and scalable as possible while keeping reasonable computational requirements. To achieve this, the system sacrifices the ability to make drift free measurements. To fully realise this system, a new feature detector (designed to be as computationally efficient as possible) is required and this is developed in Chapter 3. Dealing with arbitrary smooth curved surfaces is not trivial and is even harder to do at frame rate. In Chapter 4 a new edge based tracking system is described which is able to track at frame-rate, but it is developed with regard to computational efficiency as opposed to robustness. Chapter 5 presents a statistical analysis of the different tracking systems and develops a method whereby they can be combined resulting in an extremely robust overall tracking system which is still able to operate at video rate on standard hardware. Each section has a more detailed literature survey relevant to the material being presented.

2. Feature based tracking

2.1 Introduction

Despite advances that have been made in the domain of frame-rate visual tracking, there is still a need for systems which can tolerate the very rapid translations, rotations and accelerations which can occur in certain situations, such as unconstrained hand held or head mounted cameras. Even with a good motion model, these kinds of motions can lead to large prediction errors, so a system is needed which can deal with this. This kind of problem suggests the use of a point feature based tracking system: point features are relatively easy to localise and correspond between frames.

Typically, point feature based tracking works as follows. A 3D model is present in the form of a 3D point cloud with each of the 3D points annotated with appearance information. Correspondences between the 3D points and the 2D image are found. A position prior may be used to reduce the amount of searching that this requires. The model position is then adjusted to minimise the reprojection error between the 3D points and their 2D correspondences.

There are several different kinds of point based tracking system and these different systems have different properties in terms of robustness, drift, scalability, CPU usage and longevity of the model. It is the opinion of the author that a detection and matching based tracking system which does not maintain a static 3D point cloud is the best compromise (as discussed in Section 2.2.1). Because the point cloud is not static, drift in this kind of system is high and must be corrected by other means (see Section 5.4). The purpose of the tracker presented in this chapter is to provide the differential (i.e. subject to drift) measurements as robustly as possible.

2.2 Previous work

2.2.1 Tracking

Point feature based tracking systems broadly fall into one of two categories: *set-to-region*, where each member of a set of features known in frame n is matched against a region in frame $n+1$, e.g. by using NCC (Normalised Cross Correlation), and *set-to-set* matching, where a set of features detected in frame n is matched against a set of features in a database (which may be the features detected in frame $n+1$).

The Lucas-Kanade[6, 91] template tracking algorithm works by having a template (a patch of an image) and a model for distorting the patch. The parameters of the distortion (often translation and rotation but can also include complex warps such as projective warps and even appearance models) are adjusted to minimise the sum squared error between the distorted image patch and the image. Since this warp is arbitrary, it can be used to track patches and 3D pose [10, 15, 48, 62, 63, 97]. These tracking algorithms can be used in a set-to-region system by first extracting points and then tracking them in 2D across the image in subsequent frames.

A recent example (using NCC) of set-to-region matching is presented in [27], for a point feature based SLAM (Simultaneous Localisation and Mapping) system. A feature detector is infrequently run on the image to acquire new features and features are extracted as a small image patch. The system maintains full covariance for the feature position in 3D, and projects this into the image as a covariance ellipse. The position of the feature in this image is found by NCC between the extracted patch and the image over the area of the ellipse.

Set-to-region matching has both advantages and disadvantages compared to set-to-set matching. One of the advantages of set-to-region matching is that tracking of features can continue after the feature detector ceases to detect the feature in the correct place. Further, the localisation can be very accurate. However, there are downsides. The changing appearance of features (for instance due to aspect changes) can cause tracking failure—though the methods proposed in [97, 106] alleviate this to a large extent. The main problem is that the area of the search region, and hence the computational cost, increase very rapidly with the prediction error. This places limits on robustness which are too severe, and consequently, the choice has been to use a set-to-set tracking system.

As stated previously, the feature based tracking system requires a 3D point cloud of the object being tracked. Acquiring the point cloud, however is non trivial. There are several techniques. In [44], multiple views of the object are required *a priori* and an off-line structure-from-motion algorithm (see [53] for more details on this process) is used to create a fairly dense point cloud. Detected points are matched into the point cloud using the SIFT [88] algorithm, and the position is computed from the correspondences.

Currently, for modest size models the system requires several times the current available CPU speed to operate at frame rate. It is also not clear how well the system would scale to very large models, without using a tracking assumption to reduce the size of the database being matched into. Finally, structure from motion is also not without its problems: the further the camera moves from the starting point, the further the estimated pose will drift from the correct value. However, since the image is matched into the database in every frame, tracking can be regained after it is lost.

A prebuilt 3D surface model of the object being tracked is used by [145]. Several views of the model are taken from a known position, and these are denoted key-frames. The model starts from a known position and points are projected onto the model. Correspondences between frames are used to find the 2D motion of the points. The change in position and the original pose are jointly optimised, which helps to reduce the effect of alignment errors. The new pose is used to predict the position and warping of points from the closest key-frame. The pose is then updated using the key-frame. This allows the system to avoid problems associated with either a large database (since only one key-frame is used at a time) or wide baseline matching. The 3D point cloud represents the appearance of the model. Using key-frames keeps a static record of appearance. Without the key-frames, the system will drift. This is equivalent to appearance drift in the template tracking literature (see [97]).

When entered, the key-frames must be manually aligned. This is a tedious process and a very large number of key-frames are required to get coverage of a large model. A solution to this problem is proposed in [144] whereby key-frames are placed automatically. If too few key-frame features are visible, then a new key-frame is added at the position of the previous frame. Because there is no manual alignment, the position will drift as errors accumulate in the positions of successive key-frames. Systems which build models without absolute positioning (in this case, the 3D point cloud is being built) will always suffer from drift. The automatic placing of key-frames reduces the drift greatly compared to the equivalent system without key-frames, but cannot remove it in the general case. In

some cases, however, such as cyclic motion, the drift may be bounded.

There is a further problem with static point clouds: often (for instance in large, indoor environments such as offices), there are large number of useful features present due to clutter, such at items placed on a desk. The features are good for short term tracking, but they may not be persistent over long timescales as the clutter will change. Consequently, the point cloud will become out of date and tracking performance will suffer.

2.2.2 Feature extraction and matching

This section is about feature extraction and matching; feature detection is a large topic and is dealt with in Section 3.

One of the earliest developed techniques was to extract patches of the image around the detected feature, and to use these patches as the *descriptor* or *feature vector*. For matching purposes, the L_1-norm, L_2-norm or NCC (Normalised Cross Correlation) can be used to measure the difference between patches [91]. Using NCC gives affine invariance to lighting, and is equivalent to using the L_2 norm after removing the mean and setting the standard deviation of the feature vector to unity. To get further invariance, a detector can be used to determine the canonical rotation. The patch can be extracted at this rotation, and then the descriptor becomes automatically rotation invariant. As well as rotation, canonical scale [88] can be found to give scale invariance, and even the canonical pose in affine image space [102] can be found. Once the canonical pose is known, the local patch can be rectified and the descriptor can be extracted as an image patch.

Other proposed descriptors give different amounts of invariance to translation, rotation and other forms of distortion. These descriptors can be used with simple extraction to give some invariance to these distortions, or they can be used to reduce errors in the localisation of the features.

One approach is to process image patches with a filter bank. In this case, the feature vector is the response of each filter in the bank at the feature position. Kœnderink proposed the 'local jet' [77] of order N which consists of all derivatives of a Gaussian kernel up to order N. In [125] rotation invariance is achieved by multiplying together elements of the local jet (up to order 3 in this case) in various ways to create a feature vector, each element of which is rotation-

ally invariant. Examples are the local average intensity, gradient magnitude and Laplacian. Scale invariance is achieved by computing the feature at scales set apart by a factor of 1.2.

Steerable filters [37] allow non-isotropic filters at arbitrary orientations to be synthesised from a small number of such filters at fixed orientations. Complex filters [124] are defined over the unit disc by the equation

$$C_{mn}(x,y) = (x+iy)^m(x-iy)^n G(x,y)\,, \tag{2.1}$$

where G is a Gaussian. The descriptor proposed in [43] uses invariants computed from moments of various orders and degree:

$$M_{pqa} = \sum_x \sum_y x^p y^q I(x,y)^a\,, \tag{2.2}$$

(where $p+q$ is the order and a is the degree), of the image patch. These invariants were originally used on colour images.

Recently, there has been a considerable amount of interest in descriptors which build histograms of some property of an image patch. In the SIFT (Scale Invariant Feature Transform) algorithm, image patches are extracted at a canonical scale and gradients of the patch are computed. A gradient histogram is used to determine the overall orientation of the feature and the patch is rotated. Features with poor orientation localisation can be rejected, or if there are several strong peaks in the orientation histogram, then multiple descriptors can be made. The image patch is then split up into a 4×4 grid, and orientation histograms containing 8 bins are made in each of the grid cells, resulting in a 128 element feature descriptor. It is noted in [66] that a 128 element feature vector is probably larger than necessary. So, to reduce the size, a simplified SIFT like descriptor (the raw gradients in a 41×41 patch after the orientation has been determined) is used, features are extracted from a very large number of images of very different scenes and then PCA (Principal Component Analysis) [112] is performed on all the extracted vectors. The top 20 basis vectors are taken, and the components in these are used as the descriptor. A newer extension is GLOH [103] (Gradient Location Orientation Histogram), which is a 272 element SIFT like descriptor computed on a log-polar grid, with the dimensionality reduced to 128 using PCA.

The shape context [8, 9] algorithm performs edge detection on the area around the point, and computes vectors from the point to all edgels. The vectors are binned in a log polar histogram.

An intensity spin image [59, 60] is a two dimensional histogram, one dimension being distance from the centre, and the other being pixel intensity. This is rotationally invariant and is extended to give affine invariance in [80].

A comparison of many of these histogram based descriptors discussed in the previous paragraphs has been undertaken in [103]. The SIFT[88] based algorithms performed best.

Various techniques have been proposed to improve the quality of matching. In [152] features are matched from frame n to frame $n+1$. Correspondences are only kept if they are the same when matching from frame $n+1$ back to frame n. An alternative method proposed in [88] operates on a database of features. If a pair of features in the database is too close, then they could easily be ambiguous, so they are removed.

The difference (or distance) between descriptors are often measured using metric norms (typically L_2), and the best match is the one with the smallest distance. The Malahanobis distance is often used, but the vectors can be transformed such that the distance is the L_2-norm.

Finding the best match can be time consuming; matching features between frames is in the naïve case $\mathrm{O}(N^2)$ in the number of features per frame. A common method for speeding up the matching problem is to use a $k-$D tree [38]. A $k-$D tree in $\mathbb{R}^k$ recursively splits up the space with $k-1$ dimensional axis-aligned hyperplanes. All points end up in bins at the leaves of the tree. A depth first search with branch and bound is performed to find the closest point. If the data is quite sparse, i.e. $N \ll 2^k$, or there is not a good match in the database, then the $k-$D tree can end up searching a large amount of the database.

To further improve efficiency, the nearest neighbour search can be approximated by limiting the number of bins searched. The BBF [7] (Best Bin First) and priority $k-$D tree [4] improve matters by searching the closest bin first. See [5] and references therein for more information on approximate nearest-neighbour searching.

By making use of a 3D model used in a tracking system, [83] turns the problem of extraction and matching into a classification problem. Features are detected and image patches are extracted and projected back onto the 3D model. A large number of views are generated for each feature (called a *view-set*). The dimensionality of the image patches is reduced via PCA, and the view sets for each feature are clustered using k-means. Matching is then performed with a

nearest neighbour classifier. The system is efficient, but still too slow to operate at frame rate. The speed of the system is improved upon in [82] in which the feature vector consists of differences between all pairs of pixels in the image patch, quantised down to one ternary digit (0, −1 or 1). Random trees [1] asking ternary questions which examine only a small subset of this vector for all the features are built. At run time, the trees are used as weak classifiers of the features, the end result being a combination of all of the outputs. The resulting system is both accurate and very computationally efficient. Because only a small portion of the feature vector is examined, it does not need to be computed in its entirety.

As stated in [46], there is a tradeoff between invariance and discrimination of features. The more invariant features are to various transforms, the less ability they have to discriminate. As a result, the invariance of the feature descriptors should be matched to the transformations that the feature is expected to undergo. If there is too little invariance then the system will be unable to generate correct matches, since none of the features will 'look' the same. If there is too much invariance, then the matching quality will suffer, since different features may look like one another if extra parameters are allowed to vary.

2.3 Operation of the tracker

The basic operation of the point based tracker is shown in Figure 2.1. The tracker tracks a 3D point cloud, which is obtained by projecting detected features onto a 3D surface model. The 3D point cloud is very short term, it is only used between pairs of frames. Consider steps (D) and (E); features detected in (D) will be in slightly different places on the object than in (A) due to quantisation and image noise. Therefore, in (E), there will be residual errors and a corresponding error in the final pose. As a result, going round the loop again, in (B), the model will be in a slightly incorrect position, so the geometry of the point cloud will be slightly incorrect. This will lead to a larger pose error in step (E). As the tracker continues round the loop, these errors accumulate and the tracker diverges from the correct pose. In Section 2.2.1, various ways of reducing the drift of point based trackers are discussed, but none of these will be used due to the problems discussed in Section 2.2.1. Instead, independent measurements will be removed to eliminate the drift.

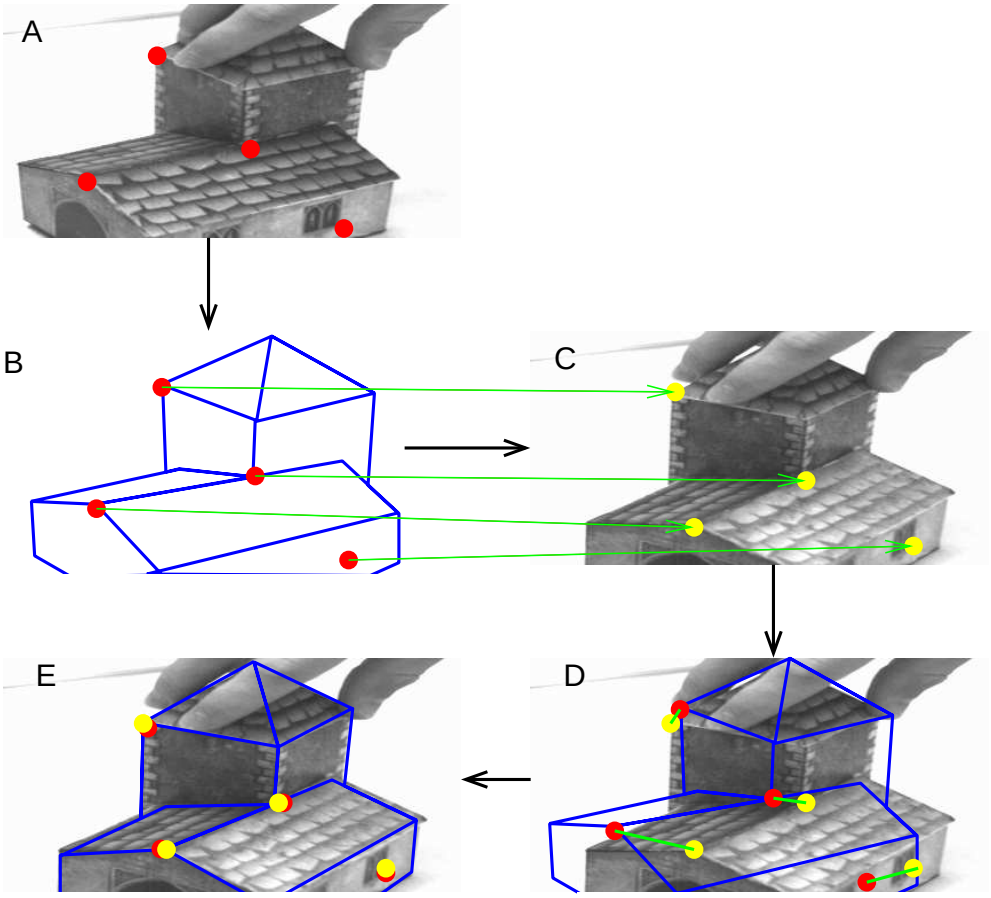


Figure 2.1: The point based tracking system. (A) Features are detected in frame n (red), and (B) are projected onto a 3D model to create a 3D point cloud. (C) Features are detected in frame $n+1$ (yellow) and feature correspondences are established between detected features and the point cloud. (D) The point cloud from frame n is shown superimposed on frame $n+1$. (E) The position is optimised to minimise the 2D distance between points in the 3D point cloud and the corresponding points in frame $n+1$.

2.4 Efficient feature matching

Features are detected in frames n and $n+1$, and feature vectors are extracted for each of the detected feature points. Detection is performed using the FAST feature detector, which is described in Section 3. The feature vector consists of the intensities of the 16 pixels in a Bresenham circle of radius 3 around the feature location, which are the pixels examined by the feature detector. The pixel intensities are not normalised.

For every feature in frame n, the feature in frame $n+1$ is chosen which minimises the SSD (sum squared difference) between the feature vectors. This allows multiple features in frame n to match to a single feature in frame $n+1$. This is in essence the simplest feature matching scheme. Although there are more advanced schemes, they all come at the cost of increased CPU usage. Furthermore the more advanced techniques increase the invariance of the descriptors, which is not necessarily advantageous. A common technique is to use NCC (Normalised Cross Correlation) for matching if image patches are used as the descriptor. This is equivalent to using the SSD if the mean is removed and the standard deviation is set to unity—which results in a descriptor which is invariant to affine changes in lighting. However, this is for a tracking system in which the features have no persistence in time—they are only ever used for inter-frame matching. Consequently, most of the features will experience small lighting changes. Adding full affine invariance to lighting makes feature matching worse since it reduces the ability of the descriptor to discriminate between features.

During the computation of the SSD between two features, the current value of the SSD is compared to the lowest SSD computed so far. If the current SSD exceeds the lowest SSD, then the features do not match and the computation can be terminated early In order to make more use of this, a rotation can be applied to the feature vector which compacts the energy into the first few terms. This is a technique common in image compression, especially JPEG [57] compression. For each frame, a rotation could be found which performs an optimal compaction, but this is computationally expensive. A rotation which is optimal for an entire dataset could be found, but it can still only be performed using an $O(N^2)$ matrix multiplication, whereas specific rotations can be performed faster. The DCT (Discrete Cosine Transform) can be performed using an $O(N \log N)$transform, and the HWT (Haar Wavelet Transform) can be performed using a particularly efficient $O(N)$ transform. Furthermore, the simplicity of the fast Haar transform means that very few multiplications (16) are required. Table 2.1 gives a comparison of the different methods.

Compaction type	None	Optimal	DCT	Haar
Number of MACs per test	4.61	3.24	3.52	3.69

Table 2.1: Average number of MACs (multiply-accumulates) required for each test during feature matching.

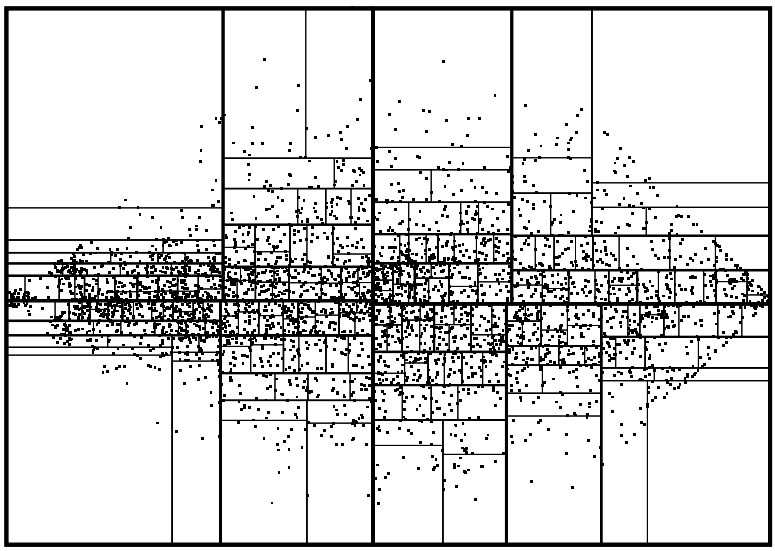


Figure 2.2: An example of a 2D $k-$D tree, shown for the first two components of HWT transformed image patches. Thicker lines are closer to the root of the tree.

If the matching method described is implemented in the simplest way possible, then the computational cost is $\mathrm{O}(N^2)$ in the number of features per frame. This cost is avoided by using a method analogous to a 1D k-D tree, which is given in Algorithm 1. In essence, the features in frame $n+1$ are sorted by the mean of the feature vectors. The feature with the closest mean to the feature being matched is found by binary search. The search for the best SSD starts from this point (a proof of this is given in Appendix A). This algorithm works because the SSD between the mean values of a pair of vectors provides a lower bound on the SSD between the vectors. Since the features are sorted, the search can be terminated on this bound. One of the primary benefits of this algorithm over the k-D tree is that very efficient, highly quality sorting (such as introsort [108]) and binary search algorithms exist in the C++ STL (Standard Template Library). Due to their widespread use, a large amount of time has gone into optimising these algorithms Furthermore, the inner loop of the algorithm is small and simple to implement.

Algorithm 1 Mean-bounded search for feature matching

BoundedSearch(Feature vector f, Database of feature vectors, D, **MaxSSD**)

find $\underset{i}{\operatorname{argmin}} \|f - d_i\|_2^2$

Sort D by the mean of the feature descriptors

$\mathbf{i} = \underset{i}{\operatorname{argmin}} |\overline{f} - \overline{d_i}|$ *Use binary search, to make this* $\mathrm{O}(\log N)$

Search outwards from d_i *until the difference in means bounds the search*
BestSSD $\leftarrow$MaxSSD
BestFeature $\leftarrow -1$
k $\leftarrow 0$
M *is the number of elements in the feature vector.*
while($M(\overline{f} - \overline{d_{i+k}})^2 <$ **BestSSD OR** $M(\overline{f} - \overline{d_{i-k-1}})^2 <$ **BestSSD**)
The entire SSD need not be computed if the SSD exceeds **BestSSD** *before the computation is complete*
if($\|f - d_{i+k}\|_2^2 <$ **BestSSD**)
BestSSD $\leftarrow \|f - d_{i+k}\|_2^2$
BestFeature $\leftarrow$**i+k**

if($\|f - d_{i-k-1}\|_2^2 <$ **BestSSD**)
BestSSD $\leftarrow \|f - d_{i-k-1}\|_2^2$
BestFeature $\leftarrow$**i** $-$ **k** $- 1$

k $\leftarrow$**k+1**
end
return BestFeature

To test the algorithm, the mean-bounded search needs to be compared to the naïve algorithm and a $k-$D tree. The $k-$D tree operates by splitting the dimension with the largest standard deviation with a hyper-plane positioned at the median of the points in that dimension. An example of a $k-$D tree is shown in Figure 2.2. The $k-$D tree is built such that there are M points in the leaf nodes, and due to the its small constant, the naïve comparison method is used within the leaf nodes. As M gets large, the poor algorithmic performance of the naïve method dominates, and as M gets small, the large constant factors of the $k-$D tree dominate. The tradeoff is shown in Figure 2.3. However, the time required to build the tree decreases with M. Table 2.2 shows the timing results. The

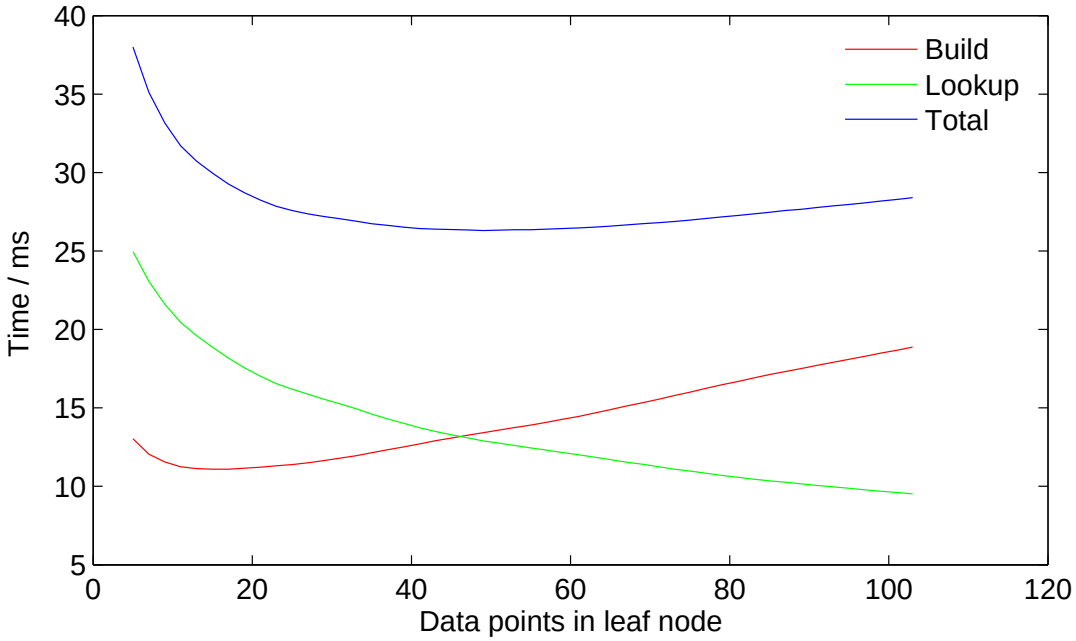


Figure 2.3: Timing results for building and indexing a $k-$D tree, against the number of data points in the leaf node. See Table 2.2 for details of the data used.

optimum for matching and overall timing has been shown.

The results in Table 2.2 show that the building time of the tree dominates the overall value of the time, and although the $k-$D tree is superior to other methods when matching is considered, the mean-bounded search performs best overall. As can be seen, the naïve algorithm is unsuitable for the task.

There is an obvious 'optimisation' for the mean-bounded search. Since both lists of features are sorted, the index in one list can be used as an approximation of the position in the second list, thereby eliminating the need for the binary search. It turns out, though that this 'scanning mean-bounded search' is not an optimisation.

The algorithm as given gives a significant speed increase relative to the $k-$D tree, but there are a few details which increase the speed considerably. The algorithm allows for a maximum permitted SSD, above which no match is considered to have been made. This is in essence an arbitrary 'magic number' (especially considering Section 2.5.1), however its presence speeds up the algorithm. If a feature disappears between frames n and $n+1$, then it will not in general look like any feature present in frame $n+1$, so the best SSD will be large. As a result, the

Algorithm	Naïve		$k-$D tree (min matching time)		$k-$D tree (min time)		**Mean-bounded search**		Scanning mean-bounded search	
comparisons	222700		17210		27570		**42980**		44170	
Setup time (ms)	0	(0)	2.64	(18.17)	1.78	(13.41)	**0.16**	**(1.24)**	0.16	(1.24)
Matching (ms)	20.04	(130.46)	1.69	(11.11)	2.05	(12.89)	**3.47**	**(22.00)**	3.52	(22.30)
Overall (ms)	20.04	(130.46)	4.33	(29.28)	3.83	(26.30)	**3.63**	**(23.24)**	3.68	(23.54)

Table 2.2: Comparison of k-D tree and mean bounded search. The results are taken from 1546 fields from a video of a laboratory, with on average 479.1 features per field. Timings were performed on an Opteron at 2.6GHz and (bracketed) a Pentium III at 850MHz. Maximum allowed SSD = ∞.

Algorithm	k-D tree: min time		Mean-bounded search		Maximum SSD
Matching time (ms)	2.05	(13.41)	3.47	(22.00)	∞
Overall time (ms)	3.83	(26.30)	3.63	(23.24)	
Matching time (ms)	2.05	(13.36)	3.01	(17.93)	5000
Overall time (ms)	3.83	(26.15)	3.17	(19.17)	

Table 2.3: Results of limiting the maximum allowed SSD. See Table 2.2 for details of the data used.

mean-bounding will not be able to terminate the search rapidly, and the feature will have to be tested against all (or nearly all) others. In practice, this has a significant effect on the speed of the algorithm, as is shown in Table 2.3.

The matching time dominates the mean-bounded search. In future, this could be improved (at no additional computational cost) by using the SSE3 vectorising instruction set, which provides a four way multiply and a horizontal add and would require no additional logic. The (on average) 3.69 multiply-accumulates could be replaced with a single one, leading to a large speed increase.

2.5 Position optimisation

Under large frame-to-frame motions, such as the one shown in Figure 2.4, feature points can change appearance significantly and this typically leads to a large number of mismatched features. In some sequences, consecutive frames have as few as 10% of points matched correctly. Even using SIFT [88] and a similar matching percentage was obtained.

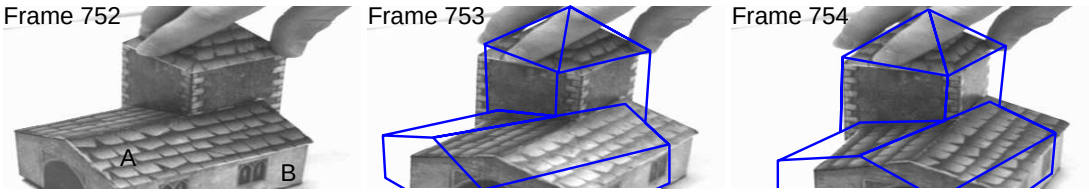


Figure 2.4: Three consecutive video frames are shown while the model is rotated rapidly (about 720 °/s) around its centre. The outline indicates the position of the model in the previous frame. Features on face A change their appearance and features on face B change shape significantly.

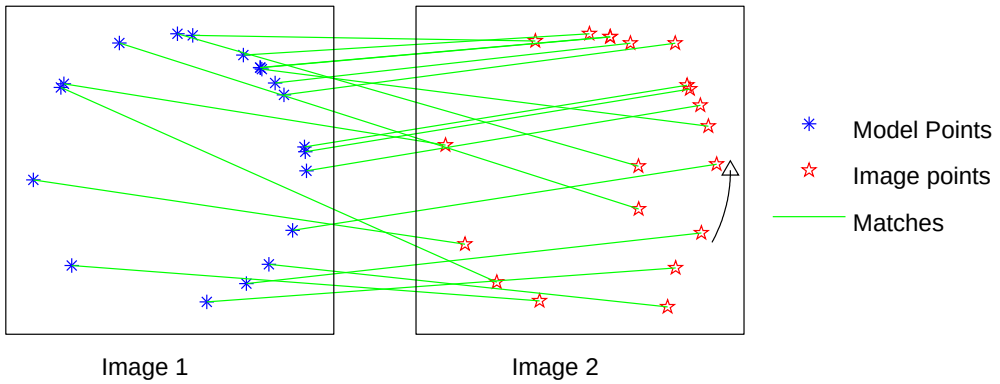


Figure 2.5: A synthetic example of feature matching between frames, where the model has a single parameter. Only a small fraction of the data is shown.

In classical Bayesian fashion, the most likely model parameters are computed by maximising the probability of the observed data given the parameters (with a weak prior). In frame n, we have two sets of features: one set of features on the model, $F_{O,n}$ (extracted in the previous frame, and reprojected under a motion $\boldsymbol{\mu}$), and another set, $F_{I,n}$, which have been extracted from the image. Between these features, we have a set of matches $M = \{m_1, ..., m_N\}$, where a match is given by $m_i = \{f_{O,i}, f_{I,i}\}$, where $f_{O,i} \in F_{O,n}$ and $f_{I,i} \in F_{I,n}$.

The set of matches can be regarded as being made up of correct (good) matches, $M_{\mathcal{G}}$, and incorrect (bad) matches, $M_{\mathcal{B}}$.

If $m_i \in M_{\mathcal{G}}$, then $f_{I,i}$ is in the same place that $f_{O,i}$ projects to under the motion to be recovered, with some added measurement noise. If $m_i \in M_{\mathcal{B}}$ then $f_{I,i}$ can appear anywhere in the image, with approximately uniform probability. If $\mathbf{e}_i$ is the Euclidean distance between $f_{O,i}$ and $f_{I,i}$ in the image, then the PDF

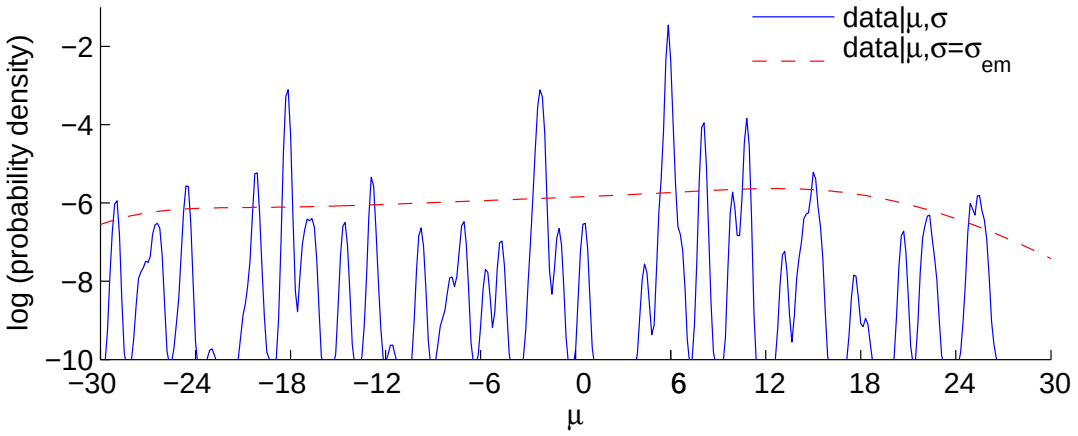


Figure 2.6: Probability density of observing M for the one dimensional example. PDFs are shown for the known value of σ, σ_g and where EM converges, σ_{em}, at the known value of α. The correct value of $\boldsymbol{\mu}$ is 6.

(Probability Density Function) of observing the matches is given by:

$$p(M|\boldsymbol{\mu}) = \prod_{i=1}^{N} \frac{(1-\alpha)}{A} + \alpha \frac{e^{(-\frac{\mathbf{e}_i^{\mathsf{T}}\mathbf{e}_i}{2\sigma^2})}}{2\pi\sigma^2}, \tag{2.3}$$

where A is the image area, α is the expected proportion of good matches and σ is the measurement noise.

In theory, one method for finding the most likely $\boldsymbol{\mu}$ is to use iterative reweighted least squares (IRWLS) where the reweighting function is the posterior probability that a match is an inlier. In practice, with a large number of mismatches, IRWLS will often not succeed because of local maxima in the likelihood. This will be demonstrated with an example with a one-dimensional model. Points are placed randomly on the unit circle, giving F_O. The model is then rotated by $\boldsymbol{\mu}$ radians, which is the parameter to be determined. To simulate matching, the new positions of the points are either corrupted by Gaussian noise with variance σ^2, or scattered about the unit square (Figure 2.5), giving F_I.

The likelihood of the data given $\boldsymbol{\mu}$ and the correct value of σ_g is shown in Figure 2.6 and visibly contains *many* local maxima with the absolute maximum being a narrow peak. Hence, in order to find the maximum, it is necessary to use a technique which is robust to local maxima. Generalised Monte-Carlo based techniques such as simulated annealing [70] can escape local optima by randomly perturbing the system. Perturbations are accepted with a probability based on

how energetically favourable it is to accept the perturbation. At higher temperatures, the cost function is raised to a small power, making perturbations more likely to be accepted. As the system is annealed, the power is increased, making it harder to escape from the current optimum. However, in this case, the global optimum is very narrow, so there is only a very small probability that a perturbation, or indeed any particle based method, will land in the optimum. Instead, it would be preferable to make the correct peak broader by convolving the likelihood with a Gaussian to blur it. This gives the following PDF:

$$p(m_i|\boldsymbol{\mu}) = \frac{(1-\alpha)}{A} + \alpha \frac{e^{(-\frac{\mathbf{e}_i^\mathsf{T}\mathbf{e}_i}{2(\sigma^2+\sigma_b^2)})}}{2\pi(\sigma^2+\sigma_b^2)}, \tag{2.4}$$

where σ_b is the size of the blur. This is equivalent to using a value of σ larger than the true value. Because for large σ, the peak will not be in the correct place, a schedule is needed to reduce it to its correct value and the most effective way to do this is by using the EM (Expectation Maximisation) algorithm [29] to find a PDF which fits the data. This is convenient because it gives us a framework for calculating unknown variables such as the true value of α and σ. The algorithm works as follows:

1. Estimate $p(m_i \in M_\mathcal{G})$ and $p(m_i \in M_\mathcal{B})$ given $\boldsymbol{\mu}$, σ_b and α. The mixture model is give in Equation 2.4.

2. Recompute the most likely $\boldsymbol{\mu}$ using the Gauss-Newton method. Let

$$\mathbf{K} = \begin{bmatrix} p(m_1 \in M_\mathcal{G})\mathbf{J}_1 \\ p(m_2 \in M_\mathcal{G})\mathbf{J}_2 \\ \vdots \end{bmatrix} \tag{2.5}$$

   and

$$\mathbf{f} = \begin{bmatrix} p(m_1 \in M_\mathcal{G})\mathbf{e}_1 \\ p(m_2 \in M_\mathcal{G})\mathbf{e}_2 \\ \vdots \end{bmatrix}, \tag{2.6}$$

   where $\mathbf{J}$ is defined in Equation 1.15. The motion is then recomputed as:

$$\boldsymbol{\mu} \leftarrow \boldsymbol{\mu} + (\mathbf{K}^\mathsf{T}\mathbf{K} + \lambda\mathbf{I})^{-1}\mathbf{f}, \tag{2.7}$$

   where $\mathbf{I}$ is the identity matrix, and λ introduces numerical stability.

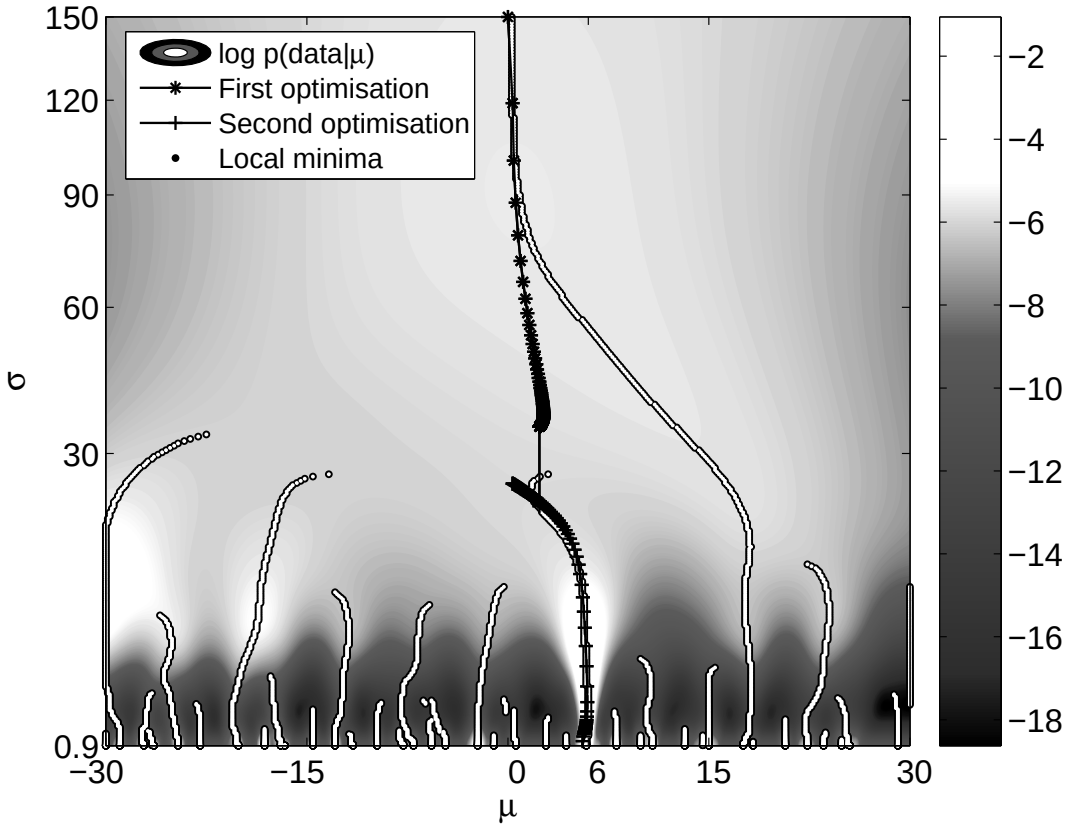


Figure 2.7: The greylevel plot shows the likelihood of the data given the model parameter, $\boldsymbol{\mu}$, for different levels of blur with the known value of α. The bottom row in this plot is the same as the graph in Figure 2.6. The graph also shows the path that EM takes through this space. The calculated values for α are not shown.

3. Recompute the new values of α and σ_b from the new value of $\boldsymbol{\mu}$.

This is shown operating on the 1D example in Figure 2.7.

In the presence of large amounts of noise, the EM algorithm can converge on a solution where σ is too large, as shown in Figure 2.6. Although the precise value of σ is unknown, we know its approximate value; a large proportion of the measurement error comes from pixel quantisation, hence $\sigma \sim 1$ pixel. If EM converges to $\sigma^2 \gg 1$, then the optimisation is given a 'kick' downwards by forcibly lowering σ, and recomputing α. The effect of this is shown in Figure 2.7.

It should be noted that Guided [139] MLESAC [140] could be used here instead of EM, and would take in the same prior information and produce an equivalent posterior, except it would not estimate α. It is unclear which would be computationally more efficient, but the current system is sufficient for frame-rate operation. However, the estimation of α is useful, as will be shown in Section 2.5.1.

EM jointly optimises the posterior probability $P(m_i \in M_\mathcal{G})$ for each point. If there is a prior estimate of this (see Section 2.5.1) the posterior becomes:

$$P(m_i \in M_\mathcal{G}) = \frac{P_p \alpha p_\mathcal{G}}{\frac{(1-P_p)(1-\alpha)}{A} + P_p \alpha} \tag{2.8}$$

where

$$P_p = P_{\text{prior}}(m_i \in M_\mathcal{G})$$

and

$$p_\mathcal{G} = \frac{e^{(-\frac{\mathbf{e}_i^\mathsf{T}\mathbf{e}_i}{2(\sigma^2+\sigma_b^2)})}}{2\pi(\sigma^2+\sigma_b^2)}.$$

The covariance of the posterior pose from the feature point tracker is:

$$\mathbf{C}_f = \left(\sigma^2 \sum_i P_{\text{post}}(m_i \in M_\mathcal{G}) \mathbf{J}_i^\mathsf{T} \mathbf{J}_i\right)^{-1}, \tag{2.9}$$

where $\mathbf{J}$ is the Jacobian relating the image motion (in pixels) of a point to the motion parameters (see Equation 1.15).

2.5.1 Calculation of the match prior from SSD

For a given feature point in F_O, f_n, its correspondence in F_I is calculated by finding the feature point in F_I which minimises the SSD of the feature vector between the feature points. The SSD is a measure of the difference in appearance of the two feature points, and we make the intuitive assumption that of all the features in frame $n+1$, those that look most like f_n (have the smallest SSD) are the most likely to be correctly matched. In other words, there is likely to be a relationship between the SSD and the probability that the feature match represents an inlier. We make explicit use of this relationship by on-line learning

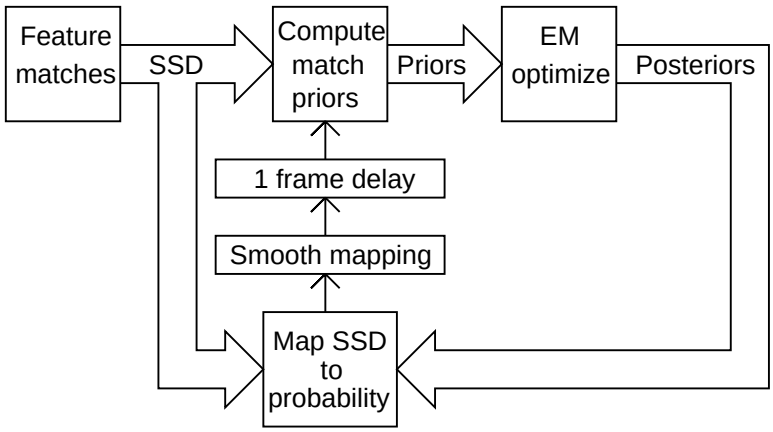


Figure 2.8: The feedback loop used to compute the matching prior based on the SSD, and the matching posteriors from the EM optimiser.

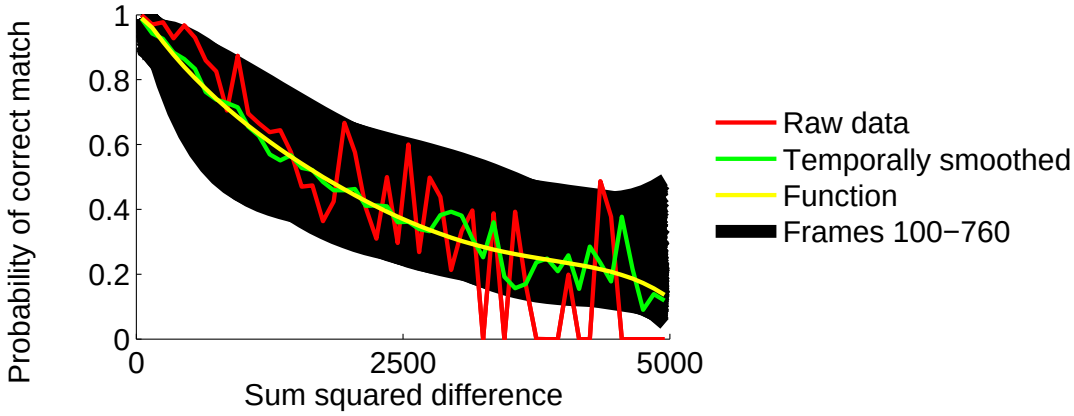


Figure 2.9: Graphs showing the function which maps SSD to match probability. The temporally smoothed function and raw data are given for frame 200 in a sequence.

of a function which maps SSD to inlier probability. This resembles the work of Tordoff and Murray [139], where the distributions of matching score (in this case, NCC) for matches and mismatches are modelled with a truncated Rayleigh distribution and a truncated quadratic respectively. These models are used to compute the probability of a given score being a correct match. The parameter of the model was then found from a large quantity of data in an off-line process, and the estimated probabilities are used to guide a RANSAC[36] based optimizer. This is extended in [19] where probabilities are not explicitly computed—only a monotonic relationship between matching score and probability is assumed.

The EM algorithm used in Section 2.5 calculates the posterior probability that each match is correct. This information, along with the SSD values for each match, can be used to provide an estimate of the relationship between SSD and inlier probability for the next frame. This is achieved by binning the SSD scores and computing the average inlier probability for each bin. To compensate for the limited amount of data in each frame (especially for large SSD values), the binned data is temporally smoothed by an IIR (infinite impulse response) filter with a time constant of 10 frames. A cubic polynomial is then fitted to the resulting values using least squares, and this is used as the SSD-to-inlier-probability mapping function for the next frame. The polynomial is used as a convenient way of creating a mapping; since it is not a representation of probabilities, it can in theory give invalid probabilities. As a result, the polynomial is fed through the following limiting function:

$$l(x) : x \rightarrow \begin{cases} 1-\epsilon, & & x & > & 1-\epsilon \\ x, & \epsilon & \geq & x & \geq & 1-\epsilon \\ \epsilon, & \epsilon & > & x & & \end{cases} . \tag{2.10}$$

Epsilon is set to a small value, (typically 0.01), which prevents the mapping completely saturating the probability of any of the points.

This feedback loop is illustrated in Figure 2.8. This technique provides a dramatic improvement in the performance of the point tracker, and the resulting mapping function can be seen in Figure 2.9. This figure also illustrates the substantial range of possible functions that can be generated over time, hence the necessity to use a dynamic as opposed to a statically learned model.

2.6 Results

2.6.1 Synthetic test of point tracking

The robust optimiser was tested using synthetically generated data using the following algorithm:

1. Place the virtual model in view of the virtual camera.
2. Synthesise corner detection by scattering points about the virtual camera frame, keeping the ones which land on the model.

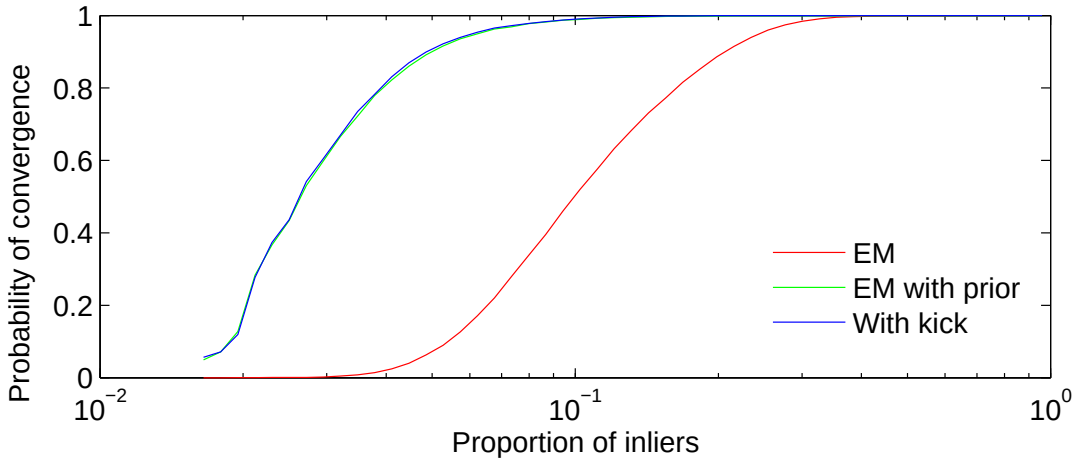


Figure 2.10: The synthetic results demonstrate that having a good estimate of $P(m_i \in M_\mathcal{G})$ greatly improves the probability of a successful convergence. The extra 'kick' downwards yields an improvement of about 1%.

3. Generate a random motion.
4. Reproject the points (with quantisation error) after the motion to simulate point matching.
5. Generate a probability that the match is correct. This corresponds to a probability obtained from the SSD.
6. Mismatching is simulated by making some of the points match to a random location in the image. The probability of this happening is based on the generated prior.
7. Optimise the position to find the motion.
8. Test the result against the known motion.

The random motion corresponds to a rotation of up to 15° and approximately 200 pixels of image motion due to translation. A total of 1000 virtual matches are generated in each frame, of which about 400 lie up on the model. Figure 2.10 shows the results of this and also clearly illustrates the improvement in performance that a good estimate of the matching prior can give. Without a good prior of $P(m_i \in M_\mathcal{G})$, 10% inliers yields a 50% probability of convergence. With a good estimate, only 3% inliers are needed to yield a 50% probability of convergence. This is important because frames with only 10% inliers are frequently experienced, and if the probability of convergence was only 50%, the system would fail frequently. Instead, it succeeds 99% of the time. The downwards kick (see Section 2.5) yields a small improvement in performance

Figure 2.11: This shows the tracking system coping with large translations. These consecutive frames are taken 1m apart, corresponding to a speed of 50ms^{-1} (112mph). In this scene, only the vertical partitions are modelled. As a result, a large number of the visible features (such as the contents of the desks) are structural clutter. The outline of the modelled geometry is shown.

2.6.2 Tests on images

The ability of the tracking system to deal with large inter-frame translations is shown in Figure 2.11. The two frames illustrated were tracked in isolation. The average image motion of point features between the two frames is 89 pixels, and the image motion of the edges closest to the camera is over 400 pixels.

This illustrates not only that the point based tracking system is capable of dealing with large inter-frame motions, but also that it is capable of dealing with large amounts of structural clutter without failing.

2.7 Conclusions

This chapter has presented a new and robust point based tracking system. It is able to converge on the correct pose even with large motions and a significant portion of outliers in the data. This has resulted from a combination of several items:

- An efficient matching scheme. This, along with the FAST feature detector allows the system to perform full frame matching: the distance that the object can be moved by is not limited by the matching scheme.
- A robust optimiser based on the EM algorithm which allows:
- A method for estimating the quality of a match based on the matching score from run time information.

The system is not designed to reduce drift, and as a result, long term tracking results do not apply to this system. Results of the complete system are given in Section 5.

3. Feature Detection

Corner detection is used as the first step of many vision tasks such as tracking, SLAM (simultaneous localisation and mapping), localisation, image matching and recognition. Hence, a large number of corner detectors exist in the literature. It is still true that when processing live video streams at full frame rate, existing feature detectors leave little if any time for further processing, even despite the massive increase in computing power since the inception of the detectors. However, this thesis is about video rate tracking, where computational resources are at a premium.

In the applications described above, corners are typically detected and matched into a database, thus it is important that the same real-world points are detected repeatably from multiple views [127]. The amount of variation in viewpoint under which this condition should hold depends on the application. It also follows that corner features must also be localisable, otherwise they would not reliably correspond to the same thing in the image.

This section will describe the development of a corner detector which is computationally efficient enough to be used as part of the frame-rate tracking system. In Section 3.2, the principle behind the feature detector is described. Section 3.3 describes how the feature detector can be implemented in a computationally efficient manner, and Section 3.4 illustrates how this can be improved generalised by using machine learning. The performance of the detector is then analysed in Section 3.5, both in terms of computational efficiency and the reliability of the features detected.

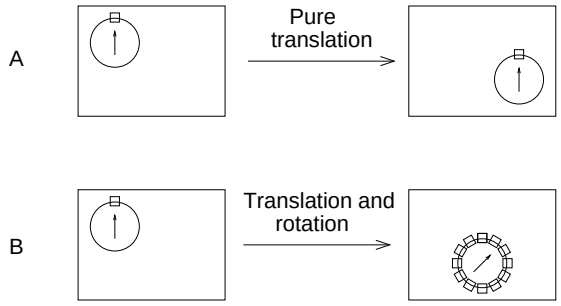


Figure 3.1: This illustrates a hypothetical feature detector which detects features which contain the top portion of a circle. In the first case (A), only pure image translation is allowed, so this feature (shown by a small box) can be reliably detected after the transformation. In the second (B) case rotation is allowed, so a detector capable of detecting the feature under the allowed transformations will detect all points on the circle. Allowing a wider range of transformations requires that stronger restrictions must be placed on what should constitutes a feature.

3.1 Previous work

3.1.1 Corner detectors

In the literature, the words "feature" and "corner" are used somewhat interchangeably to refer to small, two dimensional points of interest. These often arise as the result of geometric discontinuities, such as the corners of real world objects, but they may also arise from small patches of texture. Most algorithms are capable of detecting both kinds of points of interest, though the intuition behind the algorithms often comes explicitly from one or the other.

Corners have the following properties:

1. They should have variation in two directions. If there is no variation, then the feature cannot be localised beyond all uniform patches of the image of the same colour. If there is only one dimensional variation (an edge), then the the feature can be localised normal to the edge, but not tangent to it, since there is no variation tangent to the edge. More formally, the variation of the detected features must be matched to the transformation which will be present in the problem. This is illustrated in Figure 3.1.

2. Features need to be unique so that they are not mismatched. None of the detectors attempt to solve this problem. It is typically solved at a later stage.

Corner detectors can be categorised according to the kind of algorithm used. These are broadly:

1. Edge detection and chaining, followed by analysis of chained edges. [79, 84, 98, 105, 122]

2. Edge detection followed by local line fitting to find rapid changes in direction or appearance. [25, 50]

3. Edge detection followed by local analysis of first and/or second derivatives of the image. [71, 150]

4. Analysis of first and/or second derivatives of the image. [52, 69, 88, 90, 92, 101, 102, 109, 110, 130, 153]

5. Analysis of the SSD between an image patch and a shifted version of itself. [52, 69, 107, 110, 130, 153]

6. Analysis of a patch to see if it "looks" like a corner. [24, 32, 47, 88, 90, 107, 133, 142]

It should be noted that this is not absolute classification as many corner detectors belong to multiple classes.

The implementation of these feature detection algorithms is frequently done in the following manner:

1. A function, C, of an image, I, is applied to the image which computes corner strength at each pixel location in the image based on local information, resulting in a corner strength image.

2. The corner strength image is then thresholded, leaving a list of candidate pixels whose corner strength exceeds the threshold. This step discards most of the information in the image.

3. Some form of suppression is then performed on the list of candidate pixels to discard yet more information. This step typically (but not exclusively) used non maximal suppression so that only local maxima in the corner strength function remain.

The various corner detectors vary widely in how they perform these three steps. The most popular class of corner detectors operate by computing approximate second derivatives, either of the image or of some response of the image. Examples of this include computing image curvature or the Laplacian of the image. If a function only has first derivatives then it is a ramp in one direction, so it is completely uniform, and therefore self-similar in the direction orthogonal to the ramp. Despite the differences in approach, many of the detectors can be shown to be extracting the same information, up to some approximation.

An edge (usually a step change in intensity) in an image corresponds to the boundary between two regions. An edge exists where the gradient is both large and locally maximal. At corners of regions, this boundary changes direction rapidly. Several techniques were developed which involved detecting (for example by segmentation) and chaining edges with a view to analysing the properties of the edge. A comparison of some of these methods which use chain coded curves is undertaken in [84, 122]. Several techniques involve parameterising edges with cubic splines. Langridge [79] and Medioni [98] look for fast changes in the first derivative at points where the spline deviates a long way from the control point. The Curvature Scale Space [105] detector computes the radius of curvature of the contour and detects maxima of curvature where the maxima are significantly larger than the closest minima.

One of the problems with these techniques is the reliance on the method used to perform segmentation and chaining of the contours. As a result, many other techniques look for rapid changes in an edge by examining the local image properties instead.

Haralick and Shapiro [50] first detect edgels and use these as candidate points for corners. At each candidate point a line is fitted to the nearby edgels, and the image is examined where this line intersects a small circle around the candidate point. If the image gradient directions differ by more than a certain threshold, then the point is considered to be a corner since the edge direction is changing rapidly. They suggest using either a straight line or a cubic polynomial for the line fitting.

Cooper [25] proposes a detector which first finds edges and their directions. They then take a patch on an edge and compare it to the patches on either side in the direction of the local contour to detect self similarity.

Kitchen and Rosenfeld [71] look for rapid changes in the edge direction by measuring the derivative of the gradient direction along an edge, multiplied by the

magnitude of the gradient. The resulting corner response is

$$C = \frac{g_{xx}g_y^2 + g_{yy}g_x^2 - 2g_{xy}g_xg_y}{g_x^2g_y^2} \tag{3.1}$$

where, in general,

$$g_x = \frac{\partial g}{\partial x}, \quad g_{xx} = \frac{\partial^2 g}{\partial x^2}, \quad \text{etc.} \ldots,$$

and g is either the image or a bivariate polynomial fitted locally to the image. The best results which were reported used a quadratic polynomial.

Wang and Brady [150] propose a detector which searches for large total surface curvature on an image edge. The gradient of the image is ∇I, the normal is $\hat{\mathbf{n}} = \frac{\nabla I}{|\nabla I|}$ and $\hat{\mathbf{t}}$ is the tangent. Reformulating $I(x, y)$ as $I(t, n)$ where t and n parametrise a basis set aligned with $\hat{\mathbf{t}}$ and $\hat{\mathbf{n}}$, the curvature κ is then approximated as:

$$\kappa \approx \frac{\frac{\partial^2 I}{\partial t^2}}{|\nabla I|}, \text{where } |\nabla I|^2 \gg 1. \tag{3.2}$$

The algorithm is searching for high curvature, i.e. $\kappa^2 > S$, so the response function becomes

$$C = \frac{\partial^2 I}{\partial t^2} - S|\nabla I|^2. \tag{3.3}$$

Since the gradient has to be large, the points are also thresholded on gradient magnitude. This ensures that points only lie on an edge. The points are further restricted to lie on the steepest part of the edge, where $\frac{\partial^2 I}{\partial n^2} = 0$. This simplifies Equation 3.3 to:

$$C = \nabla^2 I - S|\nabla I|^2. \tag{3.4}$$

The authors note that the computation of first and second derivatives may require smoothing (especially if the source image is noisy), and that smoothing causes displacement of the corner. To alleviate this, they derive an expression which relates corner displacement to the smoothing factor under the assumption that the corner type is a 90° step corner.

The assumption that corners exist along edges is an inadequate model for patches of texture and point like features, and is difficult to use at 'T'-junctions. Moravec [107] proposed a feature detector which measures self similarity of an image by taking the SSD between an image patch and a shifted version of itself. The image patch is shifted horizontally, vertically and along the two diagonals and C is defined

to be the smallest SSD. Harris [52] built on this by computing an approximation to the second derivative of the SSD with respect to the shift. This is both computationally more efficient and can be made isotropic. The result is:

$$\mathbf{H} = \begin{bmatrix} \widehat{I_x^2} & \widehat{I_x I_y} \\ \widehat{I_x I_y} & \widehat{I_y^2} \end{bmatrix}, \tag{3.5}$$

where $\widehat{\ }$ denotes averaging performed over the area of the image patch. Further, Harris uses a smooth circular window over which to perform the averaging which results in an isotropic and less noisy response.

It is often claimed that $\mathbf{H}$ is equal to the negative second derivative of the autocorrelation. However, they are not the same (see Appendix B). Harris defines the corner response to be

$$C_H = |\mathbf{H}| - k(\text{trace}\,\mathbf{H})^2. \tag{3.6}$$

This is large if both eigenvalues are large, and it avoids explicit computation of the eigenvalues. It has been shown [109] that this is an approximate measure of the image curvature, though the approximation is different to the one used in Equation 3.1 and 3.4. Thresholding and non-maximal suppression is then used on the corner strength image.

Shi and Tomasi [130] conclude that under affine motion, it is better to use the smallest eigenvalue of $\mathbf{H}$ as the corner strength function:

$$C = \min\ \lambda_1, \lambda_2. \tag{3.7}$$

A number of other suggestions [52, 69, 110, 130] have been made for how to compute the corner strength from $\mathbf{H}$, and these have been shown to all be equivalent to various matrix norms of $\mathbf{H}$ [154].

Zheng *et al.* [153] perform an analysis of the computation of $\mathbf{H}$, and found some suitable approximations which allowed them to compute only two smoothed images, instead of the three previously required. They also derive a function $k(x, y)$ to replace k in Equation 3.6 in order to improve detection and stability. The resulting response function is

$$C = I_x^2 I_{yy}^2 + I_y^2 I_{xx}^2 - k(x, y)\,(I_x^2 + I_y^2). \tag{3.8}$$

They then perform an analysis of this function and show that it is approximated by:

$$C \approx \|\nabla\theta(x, y)\,\|^2 \tag{3.9}$$

where $\theta(x, y)$ is the direction of the image gradient. In other words, computation of the local SSD roughly measures the rate of change of edge direction.

In [109], Noble explains the Harris operator in terms of the first fundamental form of the image surface. From analysis of the second fundamental form, a new detector is proposed which detects points where the local surface is hyperbolic. The corner strength is defined as the probability that the classification of the surface being hyperbolic is correct.

A topographic approach is taken in [92]. Gradients are computed, and the scale of the magnetic vector potential (based on the gradients being elementary currents) is computed for every point. A topographic argument is used to derive a corner strength function based on the gradient of the potential.

An alternative approach to the problem of finding a scalar value which measures the amount of second derivative is to take the Laplacian of the image. To reduce the amount of noise (second derivatives greatly amplify noise), the smoothed Laplacian is computed by convolving the image with the LoG (Laplacian of a Gaussian). Since the LoG kernel is symmetric, an alternative interpretation is that this is performing matched filtering for features which are the same shape as a LoG. As a result, the variance of the Gaussian determines the size (or scale) of features of interest. It has been noted [102] that the locations of maxima of the LoG over different scales are particularly stable.

Lowe [88] obtains scale invariance by convolving the image with a DoG (Difference of Gaussians) kernel at multiple scales (3 per octave), retaining locations which are a maxima in both space and scale. DoG is used because it is a good approximation for LoG and much faster to compute, especially as blurred images at a large range of scales are required for other parts of the SIFT algorithm. The DoG (and therefore LoG) kernel responds quite strongly to edges. To reject edge like features, the eigenvalues of the Hessian of the image are computed at the correct scale for each detected feature. If the ratio of the eigenvalues is greater than 10, then the point is rejected, since the gradient is approximately constant in one direction. This method can be contrasted with Equation 3.4, where the Laplacian is compared to the magnitude of the edge response. In [58], several methods of computing DoG kernels are compared. Experimental evaluation show that the binomial kernel produces satisfactory results while giving a significant saving of computational resources. For this to work effectively at multiple scales, two scales per octave must be used, which also gives a further speed improvement.

Harris-Laplace [101] features are detected using a similar approach. An image pyramid is built (with a scale of 1.2 between successive layers of the pyramid), and features are detected by computing C_H (Equation 3.6) at each layer of the pyramid. Features are selected if they are a local maximum of C_H in the image plane and a local maxima of the LoG across scales.

Recently, scale invariance has been extended to consider features which are invariant to affine transformations [13, 102, 123, 124]. However, unlike the 3D scale space, the 6D affine space is too large to search, so all of these detectors start from corners detected in scale space. These in turn rely on 2D features selected in the layers of an image pyramid.

Another major class of corner detectors work by examining a small patch of an image to see if it "looks" like a corner. Since second derivatives are not computed, a noise reduction step, such as Gaussian smoothing, is not required. The result of this is that these corner detectors are computationally efficient since they only examine a small number of pixels for each corner detected. A corollary of this is that they tend to perform poorly on images with only large scale features such as blurred images. The corner detector presented in this section falls into this category.

The method presented in [47] assumes that a corner resembles a blurred wedge, and finds the characteristics of the wedge (the amplitude, angle and blur) by fitting it to the local image. The idea of the wedge is generalised in [133], where a method for calculating the corner strength is proposed which computes self similarity by looking at the proportion of pixels near to a centre, or *nucleus*, which are significantly different from the nucleus. The detector operates by computing a weighted sum of the number of pixels inside a disc whose intensity is within some threshold of the centre value. Pixels closer in intensity to the nucleus receive a higher weighting. This measure is known as the USAN (the univalue segment assimilating nucleus). A low value for the USAN indicates a two dimensional feature, since the centre pixel is very different from most of its surroundings. A set of rules is used to suppress qualitatively bad features, and then local minima of the USAN (SUSAN—Smallest USAN) are selected from the remaining candidates.

Trajkovic and Hedley [142] use a similar idea: that a patch is not self similar if pixels generally look different from the centre of the patch. This is measured by considering a circle. f_C is the pixel value at the centre of the circle, and f_P and $f_{P'}$ are the pixel values at either end of a diameter line across the circle. The response function is defined as

$$C = \min\ (f_P - f_C)^2 + (f_{P'} - f_C)^2. \tag{3.10}$$

This can only be large in the case where there is a 2 dimensional feature. The test is performed on a Bresenham circle. Since the circle is discretised, linear or circular interpolation is used in between discrete orientations in order to give the detector a more isotropic response. To this end, the authors present a method whereby the minimum response function at all interpolated positions between two pixels can be efficiently computed. Computing the response function requires performing a search over all orientations, but any single measurement provides an upper bound on the response. To speed up matching, the response in the horizontal and vertical directions only is checked. If the upper bound on the response is too low, then the potential corner is rejected. To speed up the method further, this fast check is first applied at a coarse scale.

A fast radial symmetry transform is developed in [90] to detect points. Points have a high score when the gradient is both radially symmetric, strong, and of a uniform sign along the radius. The scale can be varied by changing the size of the area which is examined for radial symmetry. The detected points have some resemblance DoG features.

An alternative method of examining a small patch of an image to see if it looks like a corner is to use machine learning to classify patches of the image as corners or non-corners. The examples used in the training set determine the type of features detected. In [32], a three layer neural network is trained to recognise corners where edges meet at a multiple of 45°, near to the centre of an 8×8 window. This is applied to images after edge detection and thinning. It is shown how the neural net learned a more general representation and was able to detect corners at a variety of angles. It is also noted that parallel nature of neural nets makes it particularly suitable for computer vision applications. In [24], features are chosen to be black discs on a white background (for part of a fiducial marker system). For computational efficiency, a cascade classifier is used. The first stage is a Bayes decision rule with a very low false negative rate and operating on only a pair of pixels. The second stage uses a condensed nearest-neighbour classifier. As the type of features becomes more complex, this (using machine learning to recognise corners) turns into the object recognition task.

3.1.2 Comparison of feature detectors

Considerably less work has been done on comparison and evaluation of feature detectors than on inventing new detectors (this is frequently noted in papers on this topic). Mohannah and Mokhtarian [104] evaluate performance by warping

test images in an affine manner by a known amount. They define the 'consistency of corner numbers' as

$$CCN = 100 \times 1.1^{-|n_w - n_o|}, \tag{3.11}$$

where n_w is the number of features in the warped image and n_o is the number of features in the original image. They also define accuracy as

$$ACU = 100 \times \frac{\frac{n_a}{n_o} + \frac{n_a}{n_g}}{2}, \tag{3.12}$$

where n_g are the number of 'ground truth' corners (marked by humans familiar with corner detection in general, but not the specific algorithm being tested) and n_a is the number of matched corners. This method unfortunately relies on subjective decisions.

Trajkovic and Hedley [142] define stability to be the number of 'strong' matches (matches detected over three frames in their tracking algorithm) divided by the total number of corners. In [138], a similar method is used: a corner in frame n is stable if it has been successfully tracked from frame 1 to frame n. Again, these measurements are clearly dependent on both the tracking and matching methods used. However, they have the advantage that they can be tested on the data used by the system, and therefore they evaluate the suitability of the corners for the specific system of interest.

When measuring reliability, the important factor is whether the same real-world features are detected from multiple views [127]. This is the definition which will be used here. For an image pair, a feature is 'detected' if it is extracted in one image and appears in the second. It is 'repeated' if it is also detected nearby in the second. The repeatability is the ratio of repeated features to detected features. In [127], the test is performed on images of planar scenes so that the relationship between point positions is a homography. Fiducial markers are projected onto the planar scene using an overhead projector to allow accurate computation of the homography.

3.2 The segment-test algorithm

The algorithm presented here belongs to the class of algorithms which examine a small patch around a candidate point to see if it "looks" like a corner. It uses the intuitive definition of what may make up a corner, that is an edge is a boundary between two regions and a corner occurs where the edge changes

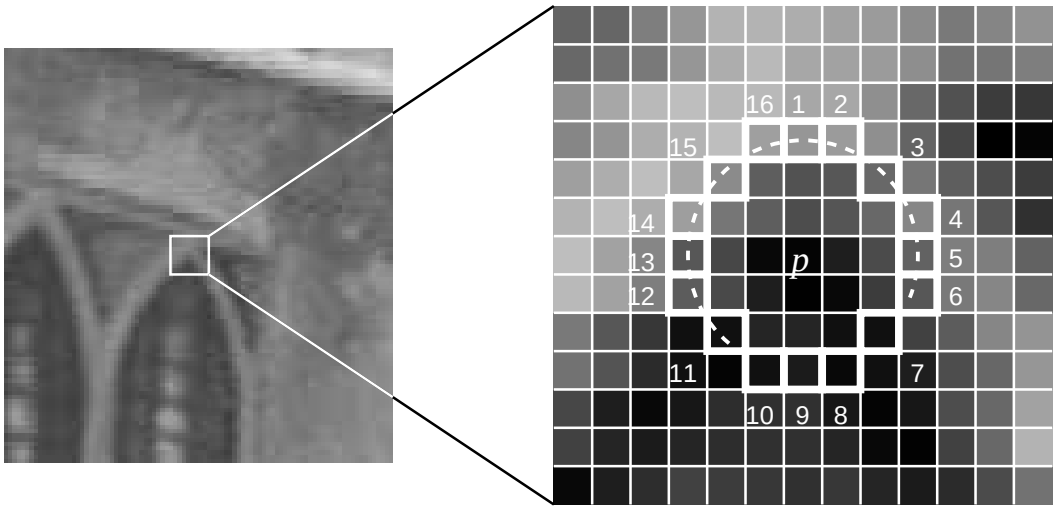


Figure 3.2: 12 point segment test feature detection in an image patch. The highlighted squares are the pixels used in the feature detection. The pixel at p is the centre of a candidate corner. The arc is indicated by the dashed line which passes through 12 contiguous pixels which are brighter than p by more than the threshold. In this case, $\theta = 3\pi/2$ and $\theta_t = 3\pi/2$ so the point is a corner.

directly suddenly. A point is on a corner if enough of the pixels around the point are in a different region from the point. Another interpretation is that a point is a corner if it is surrounded by different pixels (like USAN).

The segment-test algorithm implements this by considering a circle around the candidate point, p. It looks for the largest arc where the intensities of all the points on the arc are above the intensity of p (I_p) by some threshold, t, or the intensity of all points on the arc are below I_p by t. The point is a corner if

$$\theta \geq \theta_t, \tag{3.13}$$

where θ is the angle of the arc and θ_t is some threshold.

In practice, the image is discretised rather than continuous, so the measurements are made in a Bresenham circle of radius r around the candidate point. An approximation to θ_t is made by testing only at pixel locations on the Bresenham circle. This is used to define the *segment test criterion*:

Segment test criterion: There is a feature at p if, in a Bresenham circle of radius r around p, there are at least n contiguous pixels which are either all brighter than I_p by t or all darker than I_p by t.

A B C

Figure 3.3: Left: a test pattern containing polygons with lines joining at a variety of angles, and a selection of junctions. Centre: the first image from the Oxford basement sequence (512x512). Right: a photograph of King's College, Cambridge (768x576).

This is illustrated in Figure 3.2. The criterion given above describes an entire family of corner detectors since the radius of the circle over which the segment test is performed, and the number of contiguous pixels required, can both be varied. The remainder of this section will only consider detectors with $r = 3$ pixels. The detector will be demonstrated on a small set of test pictures shown in Figure 3.3.

The following notation will be used for the rest of this section. I is the image, p is a pixel, and I_p is the intensity of pixel p. For each location on the circle $x \in \{1 \dots 16\}$ (as shown in Figure 3.2), the pixel at position x relative to p is denoted $p \rightarrow x$, and its intensity is denoted $I_{p \rightarrow x}$.

An intuitive understanding about the kind of features detected can be found by comparing the Segment-Test detector to the LoG detector. Consider the Segment-Test detector with $\theta_t = 2\pi$. Features would be detected if they appear as a bright dot surrounded by dark pixels, or a dark dot surrounded by bright pixels. The result is a detector which is somewhat similar to a heavily quantised approximation to the LoG/DoG detector. Relaxing the requirements by allowing $\theta_t < 2\pi$ results in a detector which still responds to the same kind of features, except that they only need to look like LoG features within a segment of size θ_t, as opposed to the entire circle. The detector is rotationally invariant, so the segment of size θ_t can appear at any angle. The results of this algorithm applied to the test patterns (Figure 3.3) are shown in Figure 3.4.

Figure 3.4: The segment-test algorithm with $r = 3$ and $n = 12$. Left: (A) 318 features. Centre: (B) 1482 features ($t = 17$). Right: (C) 2594 features ($t = 50$). Features are indicated with a red dot showing the centre of the feature and a circle concentric with the dot showing the pixels that the segment test operates on.

3.3 FAST: accelerating the segment test

One of the key benefits of the algorithm is that for the case of $r = 3$ and $n = 12$, the detector can be written to be very computationally efficient. This is achieved using the FAST [118, 121] (Features from Accelerated Segment Test) algorithm. If n contiguous pixels are required, then a minimum of n tests are needed to determine if p is a feature. However, if p is not a feature, then this can often be determined with far fewer tests. Consider examining pixels at opposite sides of the circle, such as pixels 1 and 9 in Figure 3.2. If both of these pixels are close in intensity to p, then the largest contiguous ring of bright (or dark) pixels can be no more than seven pixels long. If this is the case, then p is not a feature, and this has been determined with only two tests. For the case of $n = 12$, it is reasonably easy to deduce a good ordering of tests. The full algorithm for $n = 12$ is given in Algorithm 2.

This detector exhibits high performance, but it has several weaknesses:

1. Multiple features are detected adjacent to one another.
2. The high-speed test does not generalise well for $n < 12$.

3. The choice and ordering of the fast test pixels contains implicit assumptions about the distribution of feature appearance.

4. Knowledge from the first 4 tests is discarded when the full segment test criterion is applied.

The first item is dealt with in Section 3.3.1, and the others are dealt with in Section 3.4.

Algorithm 2 The FAST algorithm for $n = 12$ and $r = 3$. Since examining pixels is expensive, care has been taken to order the tests so that the number of pixels examined is low.

Since $n = 12$, at least 3 out of four of the pixels 1, 5, 9 and 13 must be brighter than I_p by t or at least 3 out of four must be darker than I_p.

Examine pixels 1 and 9
if($|I_{p\to 1} - I_p| \leq t$ **AND** $|I_{p\to 9} - I_p| \leq t$)
    p cannot be a feature.
    The algorithm normally terminates here.

Examine pixel 13
if(2 or more examined pixels are brighter than I_p by t)
    if(Only 2 pixels are brighter than I_p by t)
        Examine pixel 5

    if(3 examined pixels are brighter than I_p by t)
        Test p using the full segment test criterion.
        This is slow, but rarely required.
    else
        p is not a feature.

else if(2 or more examined pixels are darker than I_p by t)
    if(Only 2 pixels are darker than I_p by t)
        Examine pixel 5

    if(3 examined pixels are darker than I_p by t)
        Test p using the full segment test criterion.
    else
        p is not a feature.

else
    p is not a feature.

3.3.1 Scoring and Filtering

The segment test algorithm tends to detect multiple adjacent features. To remove these, they must be scored, and features with a score which is not locally maximal are removed. Most algorithms operate by computing a C for each pixel in the image. If this is done, non-maximal suppression can be performed easily by testing the strength of a candidate corner against all nearby candidates to see if it is a local maximum. Typically, this will be done in a 3×3 square.

Since the segment test does not compute a corner response function, non maximal suppression cannot be applied directly to the resulting features. Consequently, a score function V must be computed for each detected corner, and non-maximal suppression must be applied to V to remove corners which have an adjacent (within a 3×3 square centred on p) corner with higher V.

There are several intuitive definitions for V:

1. The maximum value of n for which p would still be detected as a corner.
2. The maximum value of t for which p would still be detected as a corner.
3. The sum of the absolute differences in intensity between the pixels in the contiguous arc and the centre pixel.

Definitions 1 and 2 are very highly quantised measures: many pixels will score the same value if either definition 1 or 2 is used. For speed of computation, a slightly modified version of 3 is used. V is given by:

$$V = \max\left(\sum_{x\in B} |I_{p\to x} - I_p| - t \ , \ \sum_{x\in D} |I_p - I_{p\to x}| - t\right) \tag{3.14}$$

where

$$\begin{aligned} B &= \{x | I_{p\to x} \geq I_p + t\} \qquad \text{(Brighter pixels)} \\ D &= \{x | I_{p\to x} \leq I_p - t\} \qquad \text{(Darker pixels)} \end{aligned}$$

Subtracting t in Equation 3.14 biases the score towards favouring strong features (ones with large brightness differences) over weak ones with more contiguous pixels.

3.4 Even FASTer: a machine learning approach

In this section, the other shortcomings of the FAST detector are addressed by using machine learning to work out an efficient order for questions to be asked in. The process operates in two stages. In order to build a corner detector for a given n, first, corners are detected from a set of images (preferably from the target application domain) using the segment test criterion for n and a convenient threshold. This uses a slow algorithm which for each pixel simply tests all 16 locations on the circle around it.

Each of the 16 pixels surrounding p can have one of three different states: They can be much brighter than p, much darker than p, or roughly the same intensity. More formally, the state of $p \to x$ ($S_{p\to x}$) can have one of three states:

$$S_{p\to x} = \begin{cases} d, & & I_{p\to x} & \leq I_p - t & \text{(darker)} \\ s, & I_p - t < & I_{p\to x} & < I_p + t & \text{(similar)} \\ b, & I_p + t \leq & I_{p\to x} & & \text{(brighter)} \end{cases} \tag{3.15}$$

Choosing an x (choosing a pixel in the ring to be examined) and computing $S_{p\to x}$ for all $p \in P$ (the set of all pixels in all training images) partitions P into three subsets, P_d, P_s, P_b, where each p is assigned to $P_{S_{p\to x}}$, i.e. $P_\alpha = \{p | S_{p\to x} = \alpha\}$.

Let K_p be a Boolean variable which is true if p is a corner and false otherwise. Stage 2 employs the algorithm used in ID3 [116] and begins by selecting the x which yields the most information about whether the candidate pixel is a corner, measured by the entropy of K_p.

The entropy of K for the set P is:

$$H(P) = (c + \bar{c}) \log_2(c + \bar{c}) - c \log_2 c - \bar{c} \log_2 \bar{c} \tag{3.16}$$

$$\text{where} \quad c = \left|\{p | K_p \text{ is true}\}\right| \quad \text{(number of corners)}$$

$$\text{and} \quad \bar{c} = \left|\{p | K_p \text{ is false}\}\right| \quad \text{(number of non-corners)}$$

The choice of x then yields the information gain:

$$H(P) - H(P_d) - H(P_s) - H(P_b). \tag{3.17}$$

Having selected the x which yields the most information, the process is applied recursively to all three subsets i.e. x_b is selected to partition P_b into $P_{b,d}$, $P_{b,s}$, $P_{b,b}$, x_s is selected to partition P_s into $P_{s,d}$, $P_{s,s}$, $P_{s,b}$ and so on, where each x is chosen to yield maximum information about the set it is applied to. The process

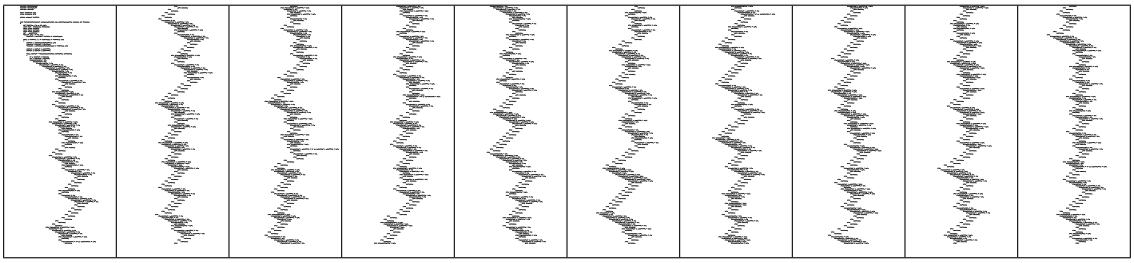


Figure 3.5: Approximately one third of the C-code generated for the 9 point FAST detector, shown in 0.3 point text. A total of 4235 lines of code are generated.

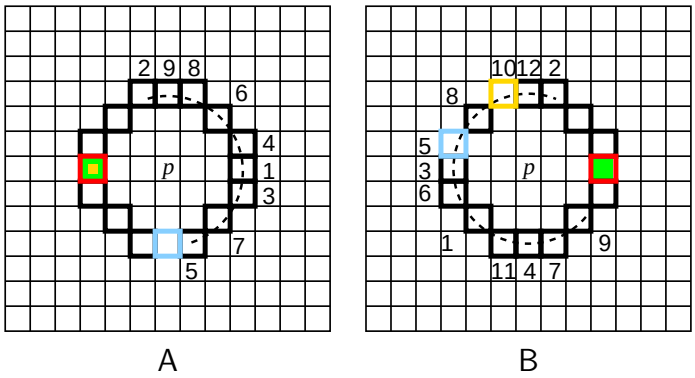
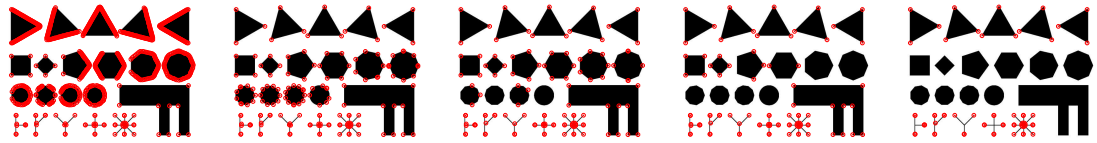


Figure 3.6: Ordering of tests for (A) the learned 9 point detector and (B) the learned 12 point detector, when all tests pass with a positive answer. If the first test fails, the red square is tested, if the second test fails, the blue square is tested, then green, then yellow.

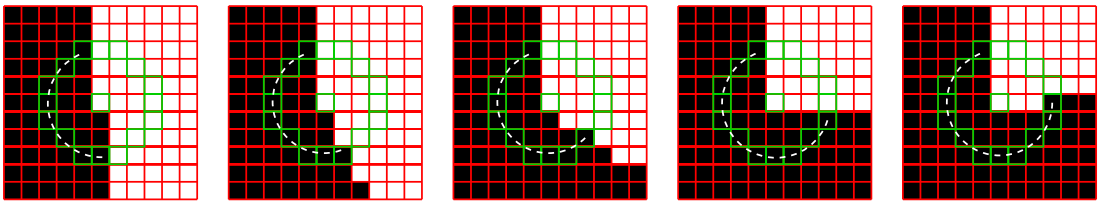
terminates when the entropy of a subset is zero. This means that all p in this subset have the same value of K_p, i.e. they are either all corners or all non-corners. This is guaranteed to occur since K is an exact function of the learning data.

This creates a ternary decision tree which can correctly classify all corners seen in the training set and therefore (to a close approximation) correctly embodies the rules of the chosen FAST corner detector. This decision tree is then converted into C-code, creating a long string of nested if-then-else statements which is compiled and used as a corner detector. This is illustrated in Figure 3.5. For full optimisation, the code is compiled twice, once to obtain profiling data on the test images and a second time with arc-profiling enabled in order to allow reordering optimisations. In some cases, two of the three subtrees may be the same. In this case, the Boolean test which separates them is removed.



A ($n = 8$) B ($n = 9$) C ($n = 10$) D ($n = 11$) E ($n = 12$)

Figure 3.7: Segment test corner detection on a test pattern with non-maximal suppression and $r = 3$.



A ($n = 8$) B ($n = 9$) C ($n = 10$) D ($n = 11$) E ($n = 12$)

Figure 3.8: Examples of detected corners where the angle is the maximum detectable angle.

3.4.1 Example detectors and features

An understanding about the decision tree can be gained by looking at a small part of the tree, which is shown in Figure 3.6. The figure shows how the detector orders tests such that two consecutive failures will result in a point being rejected.

The results of this corner detector with $r = 3$ and $n = 8 \ldots 12$ on the test pattern are shown in Figure 3.7. In Figure 3.7 A–D especially, a lack of rotational invariance in detection can be observed when the angle between line segments is close to the maximum detectable angle. An illustration of the type of points detected is given in Figure 3.8. This is especially clear for A. Although this is not a general edge detector, it does respond strongly to edges at some orientations. The reason for this is shown in Figure 3.8 A: the example corner is an edge slightly off the vertical.

When different corner detectors are run on the test pattern, the differences are very clear. However, when the detector is applied to real scenes, as shown in Figures 3.9 and 3.10, the difference between the detectors appears considerably less marked. As n decreases, the segment test detector becomes less susceptible to detecting patches of noise as features. This can be seen in Figure 3.10: in C, features are detected in the noisy patch on the ceiling, whereas in B, these are not

present. The reason for this is that t has been set such that the same number of features has been detected, regardless of n. As n decreases, more points pass the segment test, so in order to keep the number of features approximately constant, t has to increase. At lower values of n, the detector therefore becomes less prone to detecting noise.

One shortcoming of the segment test algorithm is shown in detail in Figure 3.10 E. The segment can respond to sloped delta edges because the circle of tested pixels misses the edge due to quantisation.

A ($n = 8$)

B ($n = 9$)

C ($n = 10$)

D ($n = 11$)

E ($n = 12$)

Figure 3.9: Segment test corner detection on a picture of King's College, Cambridge with non-maximal suppression and $r = 3$.

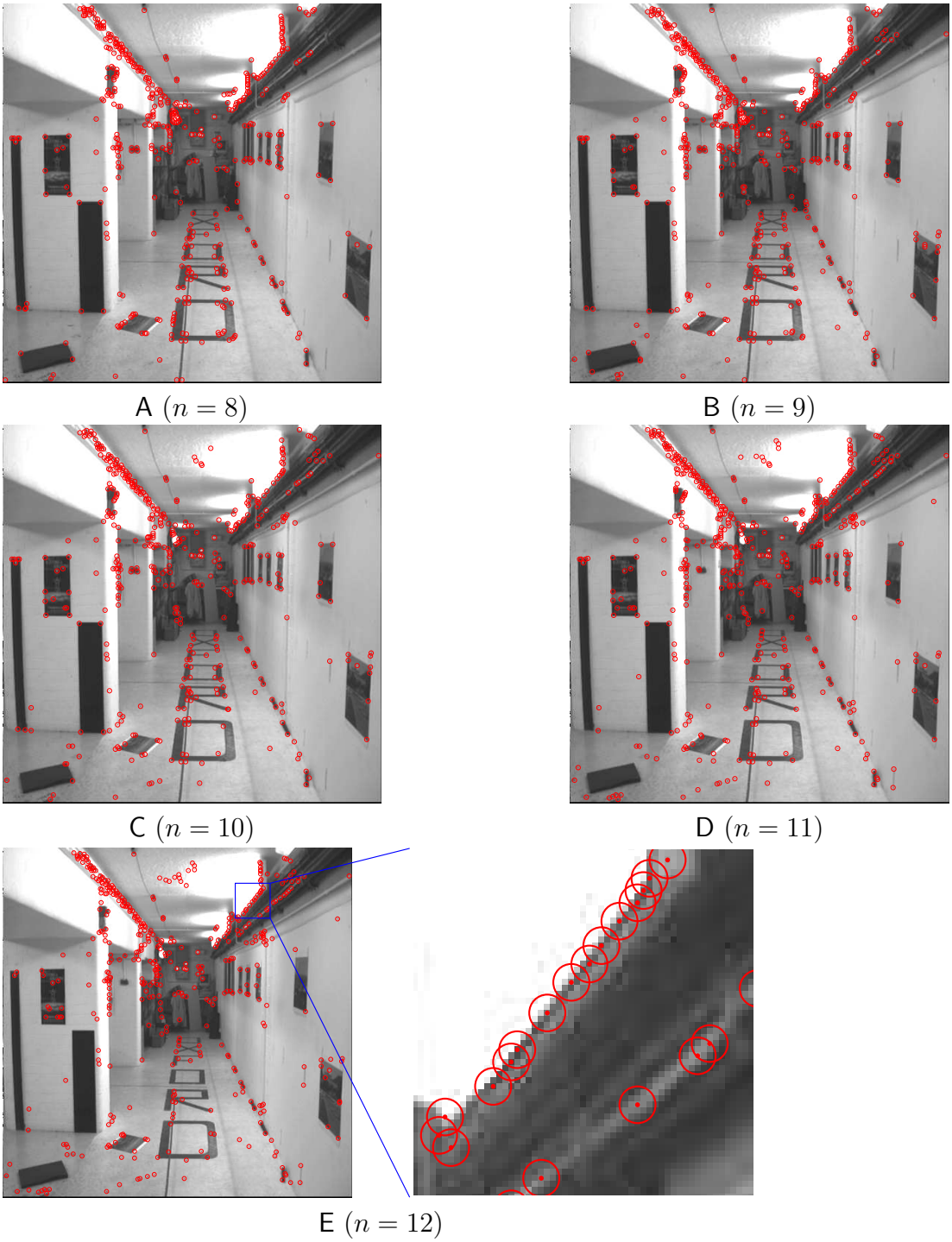


Figure 3.10: Segment test corner detection on the first picture of the Oxford corridor sequence, with non-maximal suppression and $r = 3$. The magnified cutout in E shows that the segment test algorithm responds quite strongly to sloped delta edges.

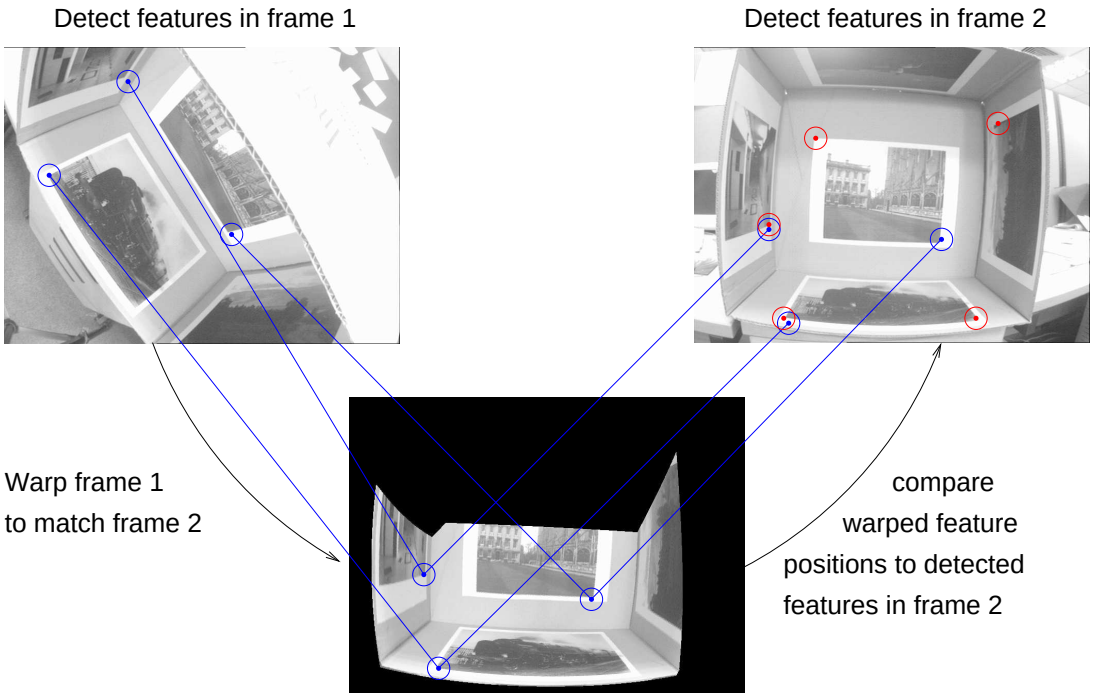


Figure 3.11: Illustration of the system used to test repeatability of a feature detector. A 3D CAD model of the scene is used to warp frame 1 correctly to match frame 2. In this example, one feature detected in frame 1 is missing in frame 2.

3.5 Evaluation

3.5.1 Repeatability

Repeatability is measured using the method of Schmid and *et al.* [127]: a feature detector has good repeatability if two different images of the same object have the same real-world features detected in both images. To measure this, two pictures of a scene are taken from different viewpoints, and features are detected in both images. If a feature in the first image appears near to a feature in the second image, then it is assumed that the same 'real-world' feature has been detected in both viewpoints. This measure provides an upper bound on the performance of a feature detection and matching scheme, since features have to be detected from multiple views before they can be matched across multiple views.

In [127], images of a planar scene were used to test the repeatability, making it

relatively straightforward to compute where points detected in one image should appear in another. Using planar scenes tests the ability of the detectors under mostly affine warps (since image features are small) under realistic conditions. This test is not all that well matched to the intended application domain of this feature detector, so instead a 3D surface model was used to compute where detected features should appear in other views (illustrated in Figure 3.11). This allows the repeatability of the detectors to be analysed on features caused by geometry such as corners of polyhedra, occlusions and 'T'-junctions. Bas-relief textures are also modelled as planar so that repeatability can be tested under non-affine warping.

A margin of error in position must be allowed when classifying points as the same 'real-world' feature because:

1. The alignment may not be perfect.
2. The model may not be perfect.
3. The camera model (especially regarding radial distortion) is not perfect.
4. The detector may find a maximum on a slightly different part of the feature in the two frames. This becomes more likely as the change in viewpoint and hence change in shape of the feature becomes large.

Instead of using fiducial markers, the 3D model is aligned to the scene by hand and is then optimised using a blend of simulated annealing and gradient descent to minimise the SSD between all pairs of frames and reprojections.

To compute the SSD between frame i and the reprojected frame j, the position of all points in frame j are found in frame i. The images are then bandpass filtered. High frequencies are removed to reduce noise, by using a Gaussian blur with $\sigma = 1$ pixel. Low frequencies are removed to reduce the impact of lighting changes by subtracting image blurred by a Gaussian with $\sigma = 20$ pixels. To improve the speed of the system, the SSD is only computed using 1000 random points (as opposed to every point). In a typical frame the maximum alignment error was reduced to under 5 pixels, so this was used as the margin of error in position.

In order to judge the quality of the FAST detector, the tests in this section compare it to several feature detectors, both well known and state of the art:

Figure 3.12: Box dataset: photographs taken of a test rig (consisting of photographs pasted to the inside of a cuboid) with strong changes of perspective, changes in scale and large amounts of radial distortion. This tests the corner detectors on planar textures.

1. SUSAN reference implementation [132]
2. SIFT (the multi-scale DoG detector) [88]
3. Harris [52]
4. Shi and Tomasi [130]
5. Harris-Laplace [101]

These detectors are tested against 224 image pairs from 3 different scenes. These scenes are shown in Figures 3.12, 3.13 and 3.14. The repeatability is computed as the number of corners per frame is varied. For comparison a scattering of random points is also included as a baseline measure, since in the limit that every pixel is detected as a corner, the repeatability is 100%. The FAST detectors are compared in Figure 3.15, which shows that the 9 point FAST detector is the best. Comparisons to other detectors are shown in Figure 3.16.

Figure 3.13: Maze dataset: photographs taken of a prop used in an augmented reality application. This set consists of textural features undergoing projective warps as well as geometric features. There are also significant changes of scale.

To test robustness to image noise, increasing amounts of Gaussian noise were added to the bas-relief dataset. It should be noted that the noise added is in addition to the significant amounts of camera noise already present (from thermal noise, electrical interference, and etc.).

Shi and Tomasi [130] derive their result for better feature detection on the assumption that the deformation of the features is affine. In the box and maze datasets, this assumption holds and it can be seen in Figure 3.16 A and B that the detector outperforms the Harris detector. In the bas-relief dataset, this assumption does not hold, and interestingly, the Harris detector outperforms the Shi and Tomasi detector in this case.

Mikolajczyk and Schmid [101] evaluate the repeatability of the Harris-Laplace detector by examining planar scenes as described in [126]. Their results show that Harris-Laplace points outperform both DoG points and Harris points in repeatability. For the box dataset, the results presented here verify that this is correct for up to about 1000 points per frame (a typical number, commonly used); the results are somewhat less convincing in the other datasets, where points undergo non-projective changes.

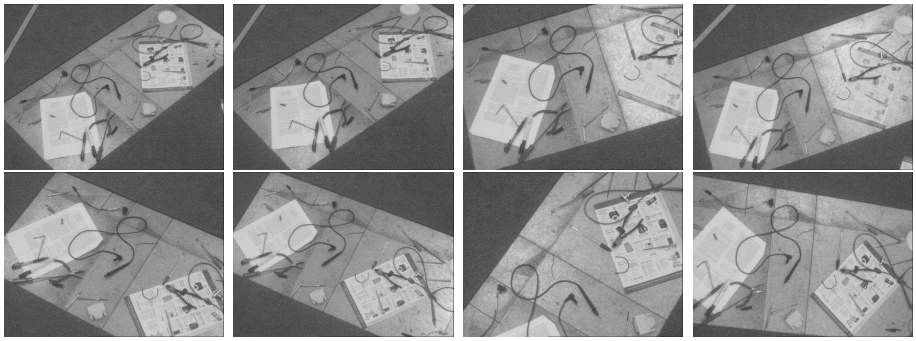


Figure 3.14: Bas-relief dataset: the model is a flat plane, but there are many objects with significant relief. This causes the appearance of features to change in a non affine way from different viewpoints.

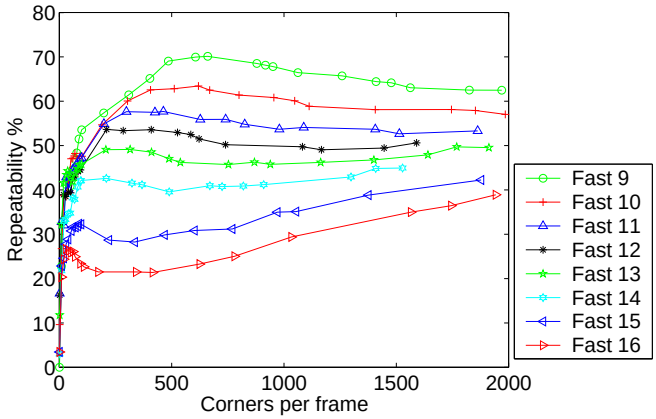


Figure 3.15: A comparison of the FAST detectors showing that $n = 9$ is the most repeatable. For $n \leq 8$, the detector starts to respond strongly to edges. The test is performed on the bas-relief dataset.

In the sample implementation of SIFT [89], approximately 1000 points are generated on the images from the test sets. The results confirm that this a good choice for the number of features since this appears to be roughly where the repeatability curve for DoG features starts to flatten off.

Smith and Brady [133] claim that the SUSAN corner detector performs well in the presence of noise since it does not compute image derivatives, and hence does not amplify noise. This claim is supported: although the noise results show that the performance drops quite rapidly with increasing noise to start with, it soon levels off and outperforms all but the DoG detector.

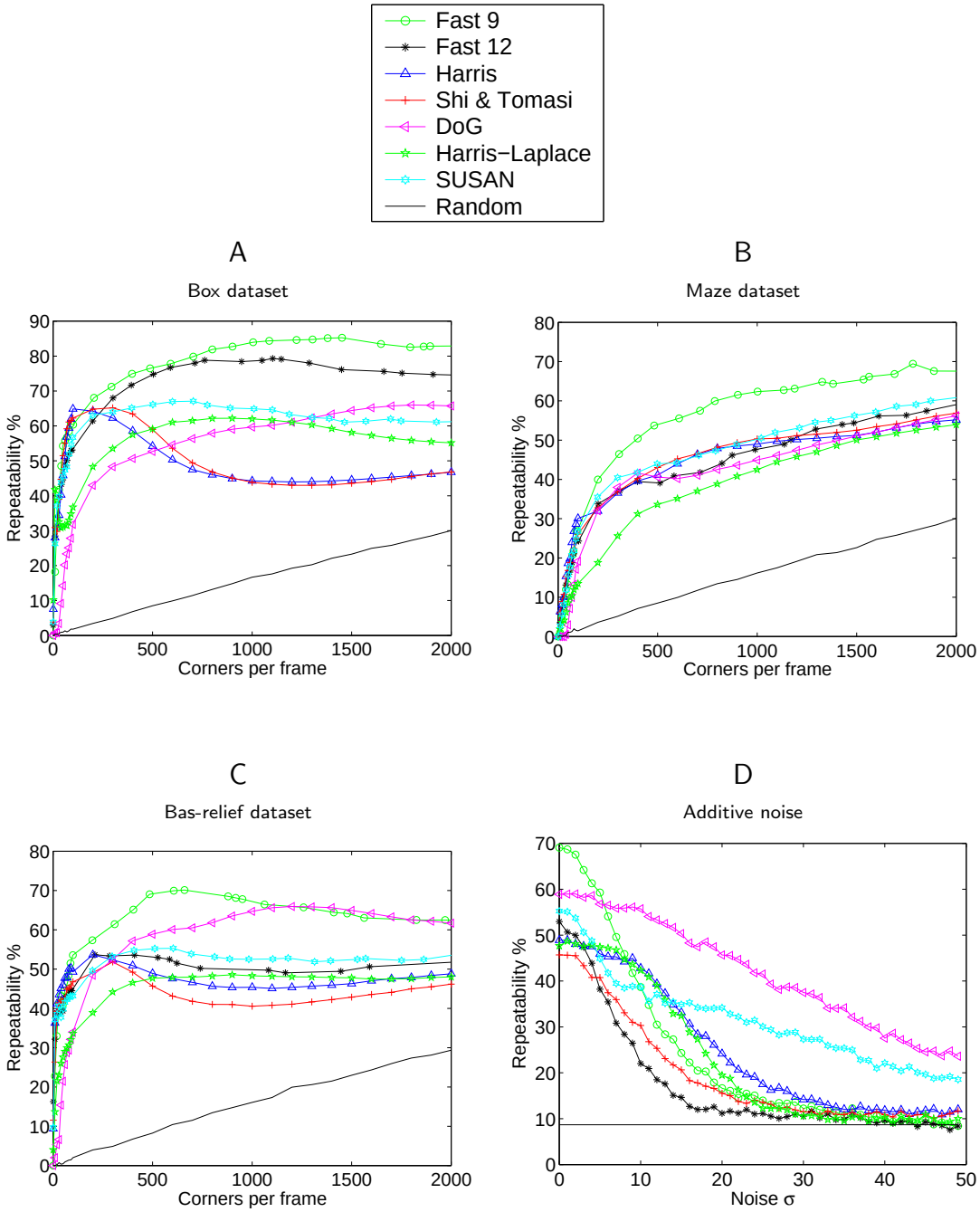


Figure 3.16: (A)–(C): Repeatability results for the three datasets as the number of features per frame is varied. (D): repeatability results for the bas-relief data set (500 features per frame) as the amount of Gaussian noise added to the images is varied. For FAST and SUSAN, the number of features cannot be chosen arbitrarily; the closest approximation to 500 features per frame achievable is used.

The big surprise of this experiment is that the FAST feature detectors, despite being designed only for speed, outperform the other feature detectors on these images (provided that more than about 200 corners are needed per frame). It can be seen in Figure 3.15, that the 9 point detector provides optimal performance, hence only this and the original 12 point detector are considered in the remaining graphs.

The DoG detector is remarkably robust to the presence of noise. Since convolution is linear, the computation of DoG is equivalent to convolution with a DoG kernel. Since this kernel is symmetric, this is equivalent to matched filtering for objects with that shape. The robustness is achieved because matched filtering is optimal in the presence of additive Gaussian noise [131].

FAST, however, is not very robust to the presence of noise. This is to be expected: since high speed is achieved by analysing the fewest pixels possible, the detector's ability to average out noise is reduced.

3.5.2 Performance

Timing tests were performed on a 2.6GHz Opteron and an 850MHz Pentium III processor. The timing data was taken over 1500 monochrome fields from a PAL video source (with a resolution of 768×288 pixels). The tree-based FAST detectors for $n = 9$ and 12 have been compared to the original FAST detector, to an implementation of the Harris and DoG (difference of Gaussians—the detector used by SIFT) and to the reference implementation of SUSAN [132].

It should be noted that the implementation of the Harris corner detector uses Gaussian smoothing for the averaging step in the computation of $\mathbf{H}$ (Equation 3.5). Instead, a box filter of an arbitrary size could be applied efficiently using an integral image [149]. Regardless of the filtering used, computation of I_x^2, I_y^2, $I_x I_y$ and C_H must still be performed for every pixel. Box convolution using an integral image is asymptotically faster than Gaussian convolution with the linear dimension of the kernel, though for very small kernels, it is likely that Gaussian convolution is more efficient due to its simplicity. Therefore, for the size of kernel used, box filtering is unlikely to lead to a dramatic increase in speed. Furthermore, using a box filter results in a response which is less rotationally invariant and it is also is noted by Harris[52] that it results in a noisier response. This would adversely affect the repeatability of the detector.

Detector	Opteron 2.6GHz		Pentium III 850MHz	
	ms	%	ms	%
Fast $n = 9$ (non-max suppression)	1.33	6.65	5.29	26.5
Fast $n = 9$ (raw)	1.08	5.40	4.34	21.7
Fast $n = 12$ (non-max suppression)	1.34	6.70	4.60	23.0
Fast $n = 12$ (raw)	1.17	5.85	4.31	21.5
Original FAST $n = 12$ (non-max)	1.59	7.95	9.60	48.0
Original FAST $n = 12$ (raw)	1.49	7.45	9.25	48.5
Harris	24.0	120	166	830
DoG	60.1	301	345	1280
SUSAN	7.58	37.9	27.5	137.5

Table 3.1: Timing results for a selection of feature detectors run on fields (768×288) of a PAL video sequence in milliseconds, and as a percentage of the processing budget per frame. Note that since PAL and NTSC, DV and 30Hz VGA (common for web-cams) have approximately the same pixel rate, the percentages are widely applicable. Approximately 500 features per field are detected.

As can be seen in Table 3.1, FAST in general offers considerably higher performance than the other tested feature detectors, and the tree-based FAST performs up to twice as fast as the handwritten version. Importantly, this process is able to generate an efficient detector for $n = 9$, which (as was shown in Section 3.5.1) is the most reliable of the FAST detectors. On modern hardware, FAST consumes only a fraction of the time available during frame-rate video processing, and on low power hardware, it is the only one of the detectors tested which is capable of video rate processing.

Examining the decision tree shows that on average 2.26 (for $n = 9$) and 2.39 (for $n = 12$) questions are asked per pixel to determine whether or not it is a feature. By contrast, the handwritten detector asks 2.8 questions on average.

Interestingly, the difference in speed between the learned detector and the original FAST are considerably less marked on the Opteron processor compared to the Pentium III. This is probably due to the Opteron having a less uniform cost per pixel than the Pentium III (the system assumes equal cost for all pixel accesses). To account for this, the entropy gain of Equation 3.17 could be multiplied by a weight, where computationally cheaper questions have lower weights.

In Table 3.1, the results are given for about 500 features per field which is a reasonable number given the repeatability results. Since speed is achieved by leaving the decision tree as early as possible, the speed of corner detection clearly

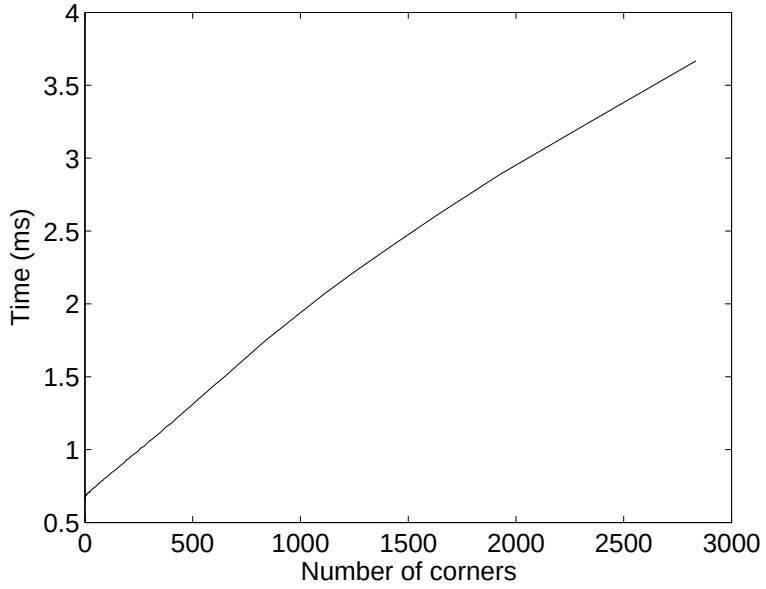


Figure 3.17: This graph shows the relationship between the number of corners detected and the speed of detection on fields of a PAL video sequence, for the 9 point detector. The test was performed on an Opteron 2.6 GHz.

depends on the number of corners detected. This is not the case for either DoG or Harris. To complete the picture, a graph of detection speed against average number of corners per frame is shown in Figure 3.17.

3.6 Conclusions

The FAST feature detector presented in this section is capable of detecting features at frame rate while leaving 93% of the CPU time available for further processing, which is over 5 times faster than the next fastest detector tested. The high speed of the detector has been achieved by careful design coupled with machine learning. Furthermore, the same features are detected reliably from a variety of viewing angles and scales. The tests of real data suggest that the new FAST detector produces more stable features than any of the commonly used detection algorithms.

4. Edge Based Tracking

4.1 Introduction

An image edge—so called because these features often correspond the the edges of the objects being viewed—is defined to be points in the image where the gradient is large, and often also defined to be locally maximal [14, 96]. Typically, edges are found on geometric edges, such as where one plane meets another at a different angle, occlusions, where one part of the object occludes another (or the background) and on surface texture, such as at strong, rapid changes in albedo. Edges are attractive features to track because the descriptor (a strong gradient) is remarkably stable under a very wide variety of transformations. A strong gradient will stay strong even under large changes in lighting strength and direction. Furthermore, edge features continue to appear on objects under strong changes of zoom, distance and perspective transformations, and radial distortion.

4.2 Previous work

One approach to edge based tracking is to use a model and the last known position of the model to predict approximately where in the image the edges should occur. Once that is done, the system only looks for edges near to the predicted edge positions. When the edges have been found, the object pose is then altered to make the rendered object line up with the image. Since the image is examined only where edges are expected to be found, this technique is computationally efficient. This system was first proposed as the RAPiD (Real-time Attitude and Position Determination) tracking system [51] and has served as the basis of many

edge based tracking systems, from which many modifications have been made.

The RAPiD system was designed to determine the six parameters of position and orientation of a 3D object in 3D space. The 3D position of the object in the previous frame and a Kalman filter [64] are used to predict the new position of the object in the current frame. The 3D model consists of a number of edges, along which control points are placed in 3D. Searches are performed normal to the edge at each control point to find a nearby image edge. RAPiD takes advantage of the aperture problem: searches are only preformed at angles of 0°, 45°, 90° and 135° for efficiency. The differential of the edge normal offset of the control points from the edge is computed with respect to the six motion parameters. The edge normal motion is determined from the searches, and this is used to update the pose to minimise the sum squared edge normal distances. This is illustrated in Figure 4.1. For efficiency (due to the limited computational resources available at the time), the control points are precomputed, and the visibility of these points (due to self-occlusion) is computed for a number of viewing positions.

More recently, several approaches to improving the robustness of RAPiD like systems have been proposed. The definition of an edge used by RAPiD , while useful, is somewhat vague. Merely detecting 'strong' edges can have variable results. Edges caused by a change in image intensity between two planes at different angles will disappear if the lighting on the two planes becomes too similar. Also, clutter may appear closer than the desired edge to the control point, causing clutter to be detected as the edge. The least-squares estimator used by the original system maximises the probability of observing the data given that errors are caused by random Gaussian noise. Since some edges are misdetected, the noise is not Gaussian. The use of an M-estimator is proposed in [35]. Based on the work of Tukey [143] and Huber [54, 55], these maximise the probability of observing the data given a non-Gaussian distribution, in this case, one in which large errors are more likely. In the original formulation, the precomputation of the visibility of the control points reduces the robustness for tracking complex objects. The object is also rendered in every frame using a BSP tree [111], and control points are placed dynamically along the visible lines.

An alternative method proposed in [95], which uses a a RAPiD like system with a robust estimator to compute affine motion of the model between two frames. The affine motion is then used to compute an approximation to the full 3D motion. The full pose is then refined by taking into account the direction of the image gradient. Specifically, $\frac{|\nabla I \cdot \hat{\mathbf{n}}|}{\|\nabla I\|}$ is computed for every control point, and the pose is adjusted to minimise the sum of these. Since $\hat{\mathbf{n}}$ is the predicted edge orientation,

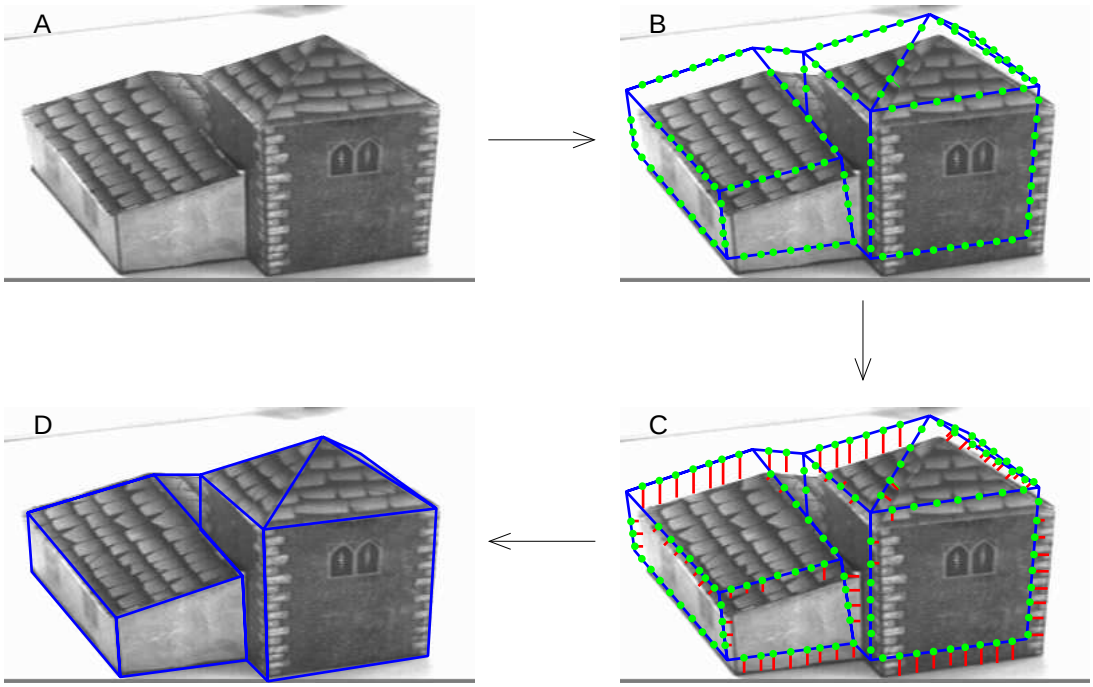


Figure 4.1: A RAPiD like edge tracker in operation. (A) A new frame arrives. (B) The previous pose is used to predict where the model and control points (green) should appear in the image. (C) Edge normal searches are performed from the control points to find the closest strong edge. (D) The pose of the model is updated to minimise the length of the edge normal searches.

ideally ∇I should be orthogonal to this. This method therefore favours positions where edges are both strong and in the correct direction.

However, there are problems with the RAPiD approach. As the amount of interframe motion allowed increases, the edge-normal search distances must also increase. After a point, the edge searches become increasingly likely to detect an incorrect edge before the correct one, and this puts limits on the robustness. A solution to this is presented in [72], where gyroscopes are used to measure angular velocity directly. This is then used to give a good prediction of the pose to make tracking more robust. To improve matters further, the gyroscopes are used to predict the amount of motion blur in the image, and this is used to create a matched filter for blurred edges, which substantially improves the robustness of the edge detection and hence tracking. A system [73] has more recently been proposed which replaces the physical gyroscopes with a purely visual system.

The RAPiD style approaches described so far make the assumption that the mea-

surements are independent. This is often not the case; a common case of bad measurements occurs when the contrast of an edge is too low. As a result, all measurements on that edge will be incorrect. Furthermore, if there is a nearby edge present, then the incorrect edge measurements will all be strongly correlated as well. This problem is tackled in [3]. The model is first split up into primitives with a known mathematical model (for instance line segments and conics). A sufficient number of control points are placed on each primitive to over-determine its parameters, and the parameters of each primitive are computed using RANSAC [36]. If a primitive has too little support, it is rejected. The pose of the object is then computed from all the primitives. Each primitive deleted in turn and the pose is recomputed. If the removal of a primitive results in a large drop in residual error, then the primitive is identified as mismeasured. The quality of primitives varies over time, so a decaying average of the frequency of correctness is kept for each primitive as a measure of confidence. The pose is then computed from all the good primitives, weighted by the confidence.

In [67], the primitives are fitted using MLESAC [140], but instead of keeping the best fit, all fitted edges (whether 'good' or 'bad') are kept, along with their scores. This enables the system to represent a multimodal probability distribution for the parameters of the edges. Edges are then selected in the next stage of RANSAC, with the probability of selection being guided [139] by the score. Groups of three of edges are chosen and the pose is computed from these. As an extra refinement, a texture based edge detector [128] is used which generalises the definition of an edge from a change in intensity to a change in pixel distributions from one side of the edge to another.

The texture edge detector is further extended in [129], by taking into account the correlated nature of edges. The edge detector produces a probability of the edge transition occurring for every pixel tested in the scanline. These are then linked across scanlines using a Hidden Markov Model, and a Viterbi algorithm is used to compute the most probable location of the edge.

As stated previously, the RAPiD style approaches need a good prior, otherwise correspondence is difficult. Therefore, another approach to edge tracking detects all edges first and attempts to fit the model to them afterwards. This is the approach adopted in [87]. First, straight line segments are extracted from the image using Marr-Hildreth [96] edge detection (hardware accelerated for speed) followed by chaining and straight segment extraction. The probability of each extracted segment belonging to each model segment is computed using the distance, orientation and model parameter covariance. The probabilities are then used to guide a search to attempt to match extracted segments to the line segments in

the model. The model pose is computed from the chosen line segments and if the residual error is small enough, then the search is terminated.

In [78], straight edge segments are extracted. The Mahalanobis distance between the model edge segment properties (length, position and orientation) and image edge segment properties are computed using the model uncertainty and image edge uncertainty for the weighting. A model edge is matched to the closest image edge measured by Mahalanobis distance. The pose is optimised and correspondences are recomputed. The whole process is iterated.

The system presented in [68] combines partitioning with the whole image approach. First, edgels are extracted in the whole image, using a modified version of the texture based edge detector which allows for multiple edge transitions. Measurements (model edges) are clustered into approximately rank-deficient clusters. These are ones where certain camera motions do not affect the appearance. For example, distant horizontal lines are affected by only by roll and pitch, but not by yaw and translation. Consequently, this cluster has only two (instead of six) degrees of freedom. As in RANSAC, a small number of edgels can be used to estimate possible values for the degrees of freedom, but since the dimensionality is greatly reduced, all combinations (as opposed to a random sampling) can be used. This gives a set of hypotheses for each cluster. Although one edge looks much like another, this does not apply to groups of edges, so correspondence is much easier to determine: typically clusters tend to contain only one or two plausible hypotheses. This approach allows the system to deal with large image motion, significant occlusion and very poor quality videos.

A thorough survey of 3D tracking (including edge based tracking) is given in [81].

4.2.1 Modelling and tracking of curved surfaces

The techniques described in the previous section deal largely with polyhedral objects. While tracking curved objects may be similar in principle, it poses some additional difficulties. Modelling arbitrary curved surfaces is more difficult than modelling arbitrary collections of planes. Furthermore, there may be fewer features on curved surfaces. Consider the case of a polyhedral surface where two planes meet at different angles. As long as the lighting is not uniform, there will be an image edge created, even if the surfaces have the same albedo. Curved surfaces do not necessarily have these features. In fact, the only feature that is always present on a curved surface is the *apparent contour*.

Figure 4.2: Dove of Peace by Pablo Picasso.

Apparent contours are therefore very important image features for curved surfaces. The apparent contour is defined as the points in the image where the viewing ray is tangent to the surface being viewed. This is broadly speaking the outline of the object. The apparent contour contains a large amount of information about the object as illustrated by the picture shown in Figure 4.2. This drawing shows only (with the exception of the eye) the apparent contour of the dove, but the object being depicted and its pose are clear. As well as having a high information content, there is another reason that makes the apparent contour the most important image feature of curved surfaces. There is in fact sufficient information present in the apparent contour of a curved surface to determine the surface shape and camera motion purely by looking at the deformations of the contour.

In [21] the deformation of apparent contours is used to determine the structure of curved objects under known camera motion. The method introduces and uses a formalism, namely the *epipolar parametrisation*. This defines corresponding points of two apparent contours in nearby views as the point where the contours intersect the epipolar plane defined by the two views. This can be used to parametrise a point on the contour with time as the camera position changes. A single view of the apparent contours give the direction of the surface normal. Two close views from known positions can be used to calculate the image speed of a point on the apparent contour, which gives the depth of the surface. This information can then be used to deduce the curvature of the surface. This epipolar parametrisation can fail at two points, cusps and frontier points (see Section 4.4.2.2 and [23]). The problem with cusps is solved in [22], by forming a

different parametrisation of cusps, the cusp locus. The motion of the cusps with time gives a different set of formulae to the epipolar parametrisation for the depth and curvature of the surface. The frontier points, on the other hand, have been put to other use in solving the structure and motion problem. A method for finding the motion of a camera by looking at frontier points is described in [20]. Given two views, a frontier point is a point where the epipolar plane is at a tangent to the surface. In both views, the apparent contour at this point is stationary with respect to to the normal of the epipolar line. The projection of a frontier point therefore completely defines an epipolar line since it gives a point on the line and the angle of the line. Two frontier points give the epipoles (intersection of the epipolar lines) in both views, and therefore the motion between the two views. The situation is not quite as simple as it appears, since the epipolar geometry is needed to define the frontier points. A guided search, followed by optimisation, is then used to find the camera motion. Once the camera motion is known, the surface shape can be found. However, [23] notes that in practice, more features than just the apparent contour would be used for determining camera motion.

Although this technique is appealing because it does not require a model, much greater robustness can be achieved if a model is used [67]. One way to model curved surfaces is to approximate the curved surface as a polygonal mesh. The surface can be represented directly as a polygonal mesh [86], or a technique such as marching cubes [85] can be used to turn an arbitrary surface into a polyhedral mesh. Once the surface has been approximated in this way, it can be tracked as if it were a polyhedral object. Other techniques centre around using segments of curved primitives for which the apparent contour can be easily computed. For example ellipsoids [18], truncated quadrics [34, 135], tapered superquadrics [39], spheres and truncated cones [28] have been used.

4.2.2 Implicit surfaces

A different approach is to use simple primitives that can be easily combined to make arbitrary shapes. Implicit surfaces are defined as the isosurface of a scalar function in $\mathbb{R}^3$. Primitive implicit shapes can be smoothly combined by summing their functions. There are several common kinds of primitives. Radial basis functions (RBFs) are widely used and are attractive from a modelling point of view since they can model smooth surfaces of arbitrary complexity and techniques exist [42] for fitting them to known 3-D data. The basis for these techniques involves fixing the centres of the RBFs and then formulating a set of linear equations in the RBF weightings which can be solved using numerical linear algebra methods.

Metaballs[1] (Gaussians in $\mathbb{R}^3$) are another commonly used primitive. Radially symmetric Metaballs are discussed extensively in [12]. This work includes a rapid surface rendering algorithm and notes a useful property of Gaussians, namely that they decay very quickly to an insignificant value which allows many calculations to be avoided, increasing efficiency. Exponents other than two are used to make hyper-ellipsoids in order to more easily model a wider variety of shapes. These other shapes require modifications to the rapid rendering techniques, since those techniques rely in part on spherical Gaussians being used. These are extended in a non-isotropic manner [151] to a class of shape known as 'Soft Objects', by using field sources as the centres of the RBF, as opposed to point sources. This allows much more accurate modelling of objects such as flat surfaces and cylinders than Metaballs. Basis functions other than exponentials have also been proposed. These mostly involve piecewise polynomial approximations of an exponential. The approximations are designed to reach zero with a zero gradient, at which point they are truncated. These are often more computationally efficient since the polynomial is faster to evaluate, and the range of influence is clearly defined. A summary of implicit surfaces for modelling is given in [99]. Since the surfaces are not defined explicitly, it can be computationally slow to locate them.

The problem of apparent contour calculation being inefficient has been tackled by [134]. The technique first preprocesses the model, restricting the state space of the tracking system to reduce the number of vastly different apparent contours allowed. Contours are considered different if the structure of the contours with intersections (which appear as T-junctions) is different. The tracker is then used to predict the pose and hence contour type. For small viewpoint changes, the shape of the contour is approximated by combining the most recent contour and the future contour. Certain applications are particularly unsuitable for preprocessing of the shape, especially high dimensional systems (such as jointed models) where the number of states that require preprocessing becomes very large. Implicit surfaces are used for off-line tracking in [113, 114], where points on the silhouette (part of the apparent contour) are found by casting a ray out from the camera.

Once the surface can be rendered, then the object can be tracked. For instance in [34], the surface consists of truncated quadrics and is therefore easy to render since quadrics project to conic curves. The object is tracked using a relatively standard technique.

A thorough mathematical analysis of the properties of curved surfaces is given

[1] Not to be confused with *meat*balls.

in [23], [74] and [41]. An analysis of apparent contours of implicit shapes under orthographic projection is given in [148].

4.3 The edge based tracking system in detail

The edge based tracking system is the same as the system presented in [35], with the exception of the technique used to measure edge strength (described below). Control points are placed every n pixels along the apparent contour. The high speed rendering techniques required to do this are the subject of the rest of this chapter. The camera frame is denoted $\mathcal{C}$.

A 3D control point in the coordinate frame of the model $\mathbf{X} = [x_{\mathcal{M}}, y_{\mathcal{M}}, z_{\mathcal{M}}, 1]^{\mathsf{T}}$ projects to $\mathbf{x} = \mathrm{P}(\mathbf{X})$. The edge direction at $\mathbf{X}$ is $\hat{\mathbf{T}}$, which projects (into the idealised camera) to direction $\hat{\mathbf{t}}$. The normal to the edge in the image is therefore:

$$\hat{\mathbf{n}}' = \mathbf{J}_c|_{P(\mathbf{X})} \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \hat{\mathbf{t}}. \tag{4.1}$$

The differential of the edge normal motion with respect to the motion parameters is:

$$\mathbf{J}_n = \hat{\mathbf{n}}' \, \mathbf{J}_c|_{P(\mathbf{X})} \, \mathbf{J}_p, \tag{4.2}$$

which is a 6×1 Jacobian.

Edge strength is given by the intensity normalised gradient: $\frac{|i_n - i_{n+1}|}{1 + i_n + i_{n+1}}$, where i_n and i_{n+1} are adjacent pixels. Inspiration for this is taken from the Self Quotient Image [49] which increases the contrast of edges in dark regions. Edge normal searches are performed at an angle quantised to the nearest 45°, and the closest edge where the strength is above a threshold is considered to be the correspondence. The edge normal distance is, of course $\hat{\mathbf{n}}' \cdot \mathbf{d}$, where $\mathbf{d}$ is the vector between the control point and its correspondence. The update to the pose of the edge tracker is computed from the Jacobian and the edge normal distance, by reweighted least squares (using regularised Gauss-Newton).

The entire edge tracking system is repeated several times for each frame. On each repetition, correspondence is recomputed and the model is relinearised. Also, on each iteration, the search length is reduced. A large search distance is required to deal with large prediction errors, but means that the system is more likely to detect erroneous edges. Reducing the search length reduces the effect of outliers

more as the position is refined more. It also leads to a large computational saving over just altering the reweighting function, since a large proportion of the computational effort goes into random access of pixels in the image.

4.4 Rapid rendering of implicit surfaces

Rendering is the process of finding which features of interest are visible and which are not. This is used to place control points on the visible part of the apparent contour. This section presents a rapid system for finding the apparent contour of the object and a system for rapid full surface rendering. For a large class of implicit functions, such as quadrics, the surface and its apparent contours can be computed analytically. However, in the general case, this cannot be done, and this section presents numerical techniques for rendering the surfaces. Section 4.4.1 presents the necessary theory of implicit surfaces. Section 4.4.2 presents the theory behind determining the visibility of the apparent contour. The mathematics described is then applied in Section 4.5, which details and compares techniques for calculating the apparent contour and determining its visibility. Section 4.4.3 then extends this theory from the apparent contours to the surface as a whole.

4.4.1 Calculating the apparent contour

A 3D shape S can be defined by a scalar function f of $\mathbb{R}^3$. A point $\mathbf{x}$ lies inside S if $f(\mathbf{x}) > 0$ and on the surface U of S if

$$f(\mathbf{x}) = 0. \tag{4.3}$$

If S is viewed from a camera centred at $\mathbf{c}$, one of the most notable features is the outline, or *visible apparent contour* of the shape. For a point $\mathbf{p}$ on this contour, there is a corresponding point $\mathbf{x}$ on U which projects to $\mathbf{p}$. A ray back projected from $\mathbf{p}$ will be at a tangent to U at $\mathbf{x}$, so

$$(\mathbf{x} - \mathbf{c}) \cdot \nabla f(\mathbf{x}) = 0 \tag{4.4}$$

(since $\nabla f(\mathbf{x})$ is normal to the surface). The points on U satisfying this equation define the *contour generators*, which are curves in $\mathbb{R}^3$. The projection of this curve by the camera at $\mathbf{c}$ is the *apparent contour*. Some parts of it are occluded by S and it is the visible parts which make up the *visible apparent contour*.

Equations 4.3 and 4.4 each provide one constraint. Combining these gives a one dimensional set of solutions which describe the contour generators. To solve the equation for $\mathbf{x}$, the the contour generator is first parameterised with the variable t, such that $\mathbf{x} = \mathbf{x}(t)$. Differentiating Equation 4.3 with respect to t gives

$$\mathbf{x}' \cdot \nabla f(\mathbf{x}) = 0. \tag{4.5}$$

Differentiating Equation 4.4 with respect to t gives:

$$\mathbf{x}' \cdot \nabla f(\mathbf{x}) + (\mathbf{x} - \mathbf{c}) \cdot \left(\mathcal{H}[f(\mathbf{x})]\, \mathbf{x}'\right) = 0 \tag{4.6}$$

where $\mathcal{H}$ is the Hessian operator. Substituting in Equation 4.5 and writing $\mathbf{H}(\mathbf{x}) = \mathcal{H}[f(\mathbf{x})]$ gives

$$(\mathbf{x} - \mathbf{c}) \cdot (\mathbf{H}(\mathbf{x})\, \mathbf{x}') = 0, \tag{4.7}$$

Using the notation from Cipolla and Giblin [23], in general:

$$\mathrm{II}(\mathbf{a}, \mathbf{r}') = -\frac{\mathbf{a} \cdot \tilde{\mathbf{n}}'}{\|\tilde{\mathbf{n}}\|}, \tag{4.8}$$

where II is the *second fundamental form*, $\mathbf{a}$ is a vector tangent to the surface, $\mathbf{r}(t)$ is a curve over the surface and $\tilde{\mathbf{n}}$ is the surface normal. Substituting $(\mathbf{x} - \mathbf{c})$ for $\mathbf{a}$, $\mathbf{x}(t)$ for $\mathbf{r}(t)$ and $\nabla f(\mathbf{x})$ for $\tilde{\mathbf{n}}$ and using the result from Equation 4.7 gives:

$$\mathrm{II}(\mathbf{x} - \mathbf{c}, \mathbf{x}') = 0. \tag{4.9}$$

In other words, the apparent contour is *conjugate* to the viewing ray. Taking Equation 4.7 and transposing gives:

$$\mathbf{x}' \cdot (\mathbf{H}(\mathbf{x})\, (\mathbf{x} - \mathbf{c})) = 0 \tag{4.10}$$

since the Hessian matrix is symmetric. Equations 4.5 and 4.10 give orthogonality constraints on $\mathbf{x}'$, so $\mathbf{x}'$ can be defined as

$$\mathbf{x}' = \alpha \left(\mathbf{H}(\mathbf{x})\, (\mathbf{x} - \mathbf{c})\right) \times \nabla f(\mathbf{x}), \tag{4.11}$$

with α giving an arbitrary scale.

The contour generator is now described by a starting point, $\mathbf{x}(0)$ and a first order differential equation. This can be solved using standard ODE integration techniques and an example is given in Figure 4.3. The visible apparent contour in Figure 4.3 B looks quite simple, but the contour generator corresponding to this is quite elaborate (C, D).

Equation 4.11 defines $\mathbf{x}'$ to be a vector field over all $\mathbf{x}$, not just on the apparent contours. Providing it does not become singular [148], this field has no divergence (see Appendix C) so it therefore defines closed contours. All the contour generators defined by Equation 4.11 and a suitable starting point are therefore closed.

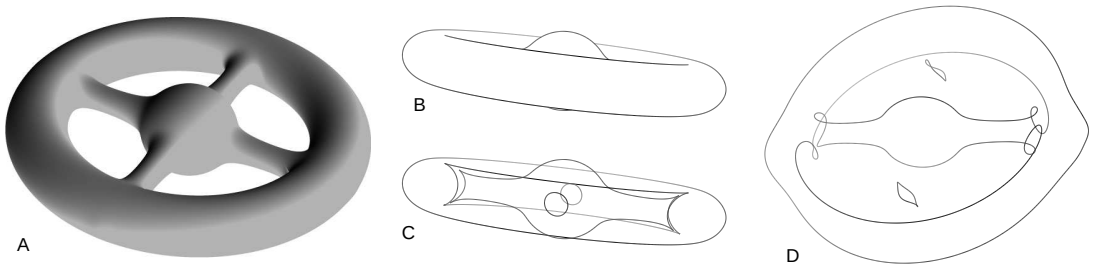


Figure 4.3: (A) A spoked wheel. (B) The visible apparent contour viewed from in front. (C) The apparent contour. (D) The contour generator from C, viewed from a different angle.

4.4.2 Determining the visibility of the apparent contour

Equation 4.11 yields a set of closed contours. Not all parts of these contours will necessarily be visible to the camera due to occlusion by S. A naïve solution to this would be to search along the ray from the camera to each point on each contour generator, testing for $f(\mathbf{x}) > 0$, but this is very inefficient.

In this section a technique is presented which allows the visibility to be determined more rapidly; an implementation of these techniques resulted in a thousandfold speed increase. This technique is based on the *occlusion depth* of points on the contour generator.

Occlusion depth:The occlusion depth of a point is the number of times a ray from the camera to the point intersects an opaque surface.

In other words, this is the number of times the ray intersects U. This is a powerful approach because the occlusion depth can only change at easily identified points as each contour is traversed. The occlusion depth and hence the visibility of the contour between these points is constant. As will be shown, the visibility of the least occluded part of each contour can then be determined by propagating information between the contours.

Figure 4.4 A shows a sphere occluding part of a torus and the image of the contours generated by Equation 4.11 is shown in Figure 4.4 B. The occlusion depth can only change in two ways. The most obvious occurs when the contour becomes occluded by another part of S, for example the point labelled 1 in Figure 4.4 B. This can easily be detected since it implies that the contours intersect. A less obvious change in occlusion depth occurs due to a cusp in the apparent contour,

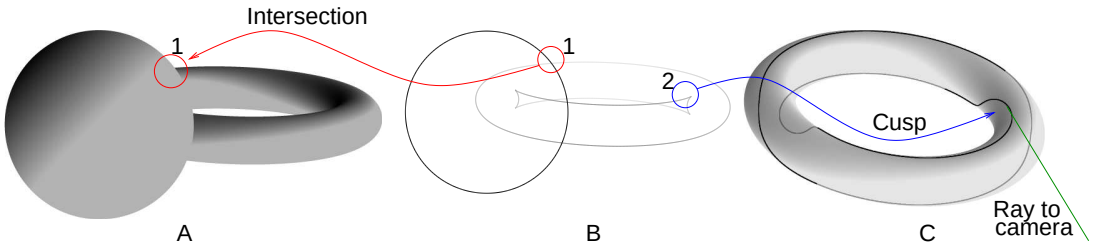


Figure 4.4: A torus partially occluded by a sphere. (A) Rendered image. (B) Apparent contour. (C) Contour generator of the torus rendered from a different angle.

at which the visible contour ends [76], as shown in Figure 4.4 B, labelled 2. This happens when the viewing ray is coincident with the contour generator [75]. This is shown in Figure 4.4 C. Note that at a cusp, the contour generator is smooth (it has continuous derivatives), even though the apparent contour is not.

4.4.2.1 Intersections

When two contours intersect in the image, the one with the generator furthest from the camera becomes occluded by part of S. When one part of the contour generator becomes occluded by another part of the shape, a ray from the camera to the occluded part must pass in and out of the shape again, i.e. it intersects U twice. At an intersection, therefore, the depth of the contour either increases by two or decreases by two, depending on the shape of the surface at the foremost contour generator. In order to determine this, the number of intersections between the ray and the foremost surface is calculated as the ray is moved along the rearmost contour. To do this, the ray through the point $\mathbf{x}_1 + \lambda \mathbf{x'}_2$ can be considered (where $\mathbf{x}_1$ is the point on the foremost surface and $\mathbf{x'}_2$ is the tangent of the rearmost contour).

Intersections between this ray and the foremost surface can then be represented as:

$$f((1+\alpha)(\mathbf{x}_1 + \lambda \mathbf{x'}_2)) = 0, \tag{4.12}$$

where α parametrises the position along the ray. Equation 4.12 can be solved locally at $\mathbf{x}_1$ by performing a Taylor expansion up to second order terms. Since the solution is local, α and λ are small, so terms in $\alpha\lambda$ can be ignored, leaving a quadratic equation in α:

$$\alpha^2 \mathbf{x}_1^\mathsf{T} \mathbf{H} \mathbf{x}_1 + 2\alpha\lambda \mathbf{x'}_2^\mathsf{T} \mathbf{H} \mathbf{x}_1 + \lambda^2 \mathbf{x'}_2^\mathsf{T} \mathbf{H} \mathbf{x'}_2 + 2\lambda \mathbf{x'}_2 \cdot \nabla f(\mathbf{x}) = 0, \tag{4.13}$$

where $\mathbf{H} = \mathbf{H}(\mathbf{x}_1)$. The number of solutions for α is given by the sign of the discriminant D of this equation. At $\lambda = 0$, $D = 0$, confirming that there is a repeated solution, i.e. the ray just brushes U at $\mathbf{x}_1$. Differentiating D with respect to λ gives:

$$d = \left.\frac{\partial D}{\partial \lambda}\right|_{\lambda=0} = -\left(\mathbf{x}_1^{\mathsf{T}}\mathbf{H}\mathbf{x}_1\right)\mathbf{x'}_2 \cdot \nabla f(\mathbf{x})\,. \tag{4.14}$$

If this derivative is positive, then going in the direction of $\mathbf{x'}_2$, D becomes positive, so the ray intersects U twice near $\mathbf{x}_1$. That is if the derivative is positive, then the depth of the rearmost contour increases by two, otherwise the depth decreases by two.

4.4.2.2 Cusps

A point $\mathbf{x}$ on the contour generator can be classified as belonging to *inner surface* or *outer surface*. If at $\mathbf{x}$ the ray to the camera dives into S then $\mathbf{x}$ is on an inner surface. If the ray moves away from S then $\mathbf{x}$ is on an outer surface. A ray from a point on an inner surface to the camera has an odd number of intersections with U, so the point has an odd numbered occlusion depth. Since its occlusion depth is nonzero, the point is not visible.

When a contour generator goes from being on an outer surface to being on an inner surface, its depth changes by one. At the point where this happens, the contour generator tangent is parallel to the ray to $\mathbf{x}$ as shown in Figure 4.4 C. Motion along the generator at this point results in no motion along the apparent contour in the image, so this appears as a cusp of the apparent contour. At a cusp, if the tangent of the contour generator is pointing away from the camera, the occlusion depth increases by one, otherwise it decreases by one. Essentially, moving away from the camera, the occlusion depth can only increase, and moving toward the camera can only cause a decrease.

4.4.2.3 Propagating depth information between contours

Tests for intersections give the relative change in depth of the contour, but cannot determine the minimum overall depth (i.e. whether the least occluded part of the contour can be seen). This can be resolved by propagating information between contours. Rule 1 is an invariant property of the occlusion depths at intersections.

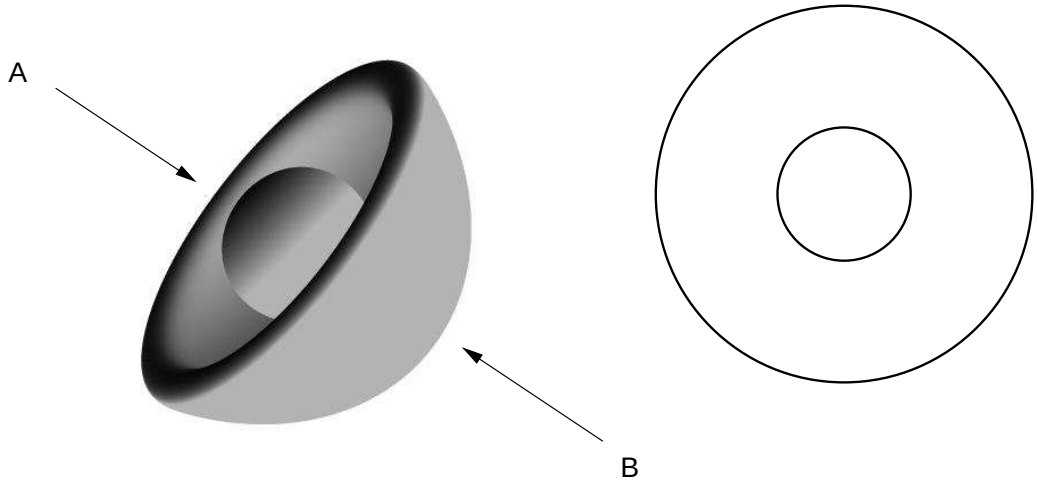


Figure 4.5: A cup and ball viewed from two different sides (left) and the apparent contours (right). In both cases, the apparent contours are identical. When viewed from A, the ball (and hence the inner apparent contour) is visible, but when viewed from B, the ball (and hence the inner apparent) contour is occluded. Note that even though in B the inner contour is occluded, every point on the inner contour is closer to the camera than every point on the outer contour.

Rule 1 *If contour A intersects contour B such that B becomes occluded, the depth of B just before the occlusion must be greater than or equal to the depth of A at the occlusion.*

Application of Rule 1 does not necessarily completely determine the visibility of all contours. If after application of this rule a contour contains a segment at depth 0 (and hence is potentially visible) but is completely contained within another contour then it may still be occluded. In this case a search must be performed to find its depth. The camera can be searched to find *if* the ray intersects U, rather than to find the *number* of intersections since this is sufficient for determining visibility. The second of these provides more information, but is slower. Search techniques are discussed in Section 4.4.3. Rule 1 must be reapplied if the segment is found to be occluded, since it is an invariant property of the system. After all segments requiring a search have been tested, then the visibility of every part of the apparent contour is known. An example of two sets of identical contours having different visibility is shown in Figure 4.5.

4.4.3 Determining Surface Visibility

Consider a scene consisting of a shape, S and an arbitrary contour, C. As C snakes through the world, it can become occluded in one of two ways. Either the contour can pass through U, in which case it becomes hidden because it is inside S, or it passes behind S. The latter of these can only happen where the projection of this curve intersects the apparent contour. If, instead of an arbitrary contour, C is restricted so that is does not pass through S, then the depth of it can be determined purely by using the rules described in Section 4.4.2, and this information can to some extent be propagated. If C is further restricted so that it can only lie on U, then it is known that anything directly behind C must also be behind S, so it must be occluded. As a result, C behaves in exactly the same way as the apparent contours. Surface patches can be created by laying many contours across the surface next to each other. Once the contour generators have been calculated, the surface patch contours can be added into the list of contours and the occlusion test can proceed as described in Section 4.4.2.

4.5 Rapid rendering of the visible apparent contour

Equation 4.11 defines the contour generator for any implicit surface. In the work so far, a sum of Gaussians in $\mathbb{R}^3$ are used to define the shape. The implicit function used has the form:

$$g_i(\mathbf{x}) = \beta_i e^{(\mathbf{x}-\mu_i)^\mathsf{T}\mathbf{C}_i(\mathbf{x}-\mu_i)} \tag{4.15}$$

$$f(\mathbf{x}) = \sum_i g_i(\mathbf{x}) - \frac{1}{2} \tag{4.16}$$

where $\mathbf{C}_i$ is the inverse covariance matrix for Gaussian i and is a real symmetric matrix. The gradient and Hessian are given by:

$$\nabla f(\mathbf{x}) = -\sum_i \mathbf{C}_i \left(\mathbf{x} - \mu_i\right) g_i(\mathbf{x}) \tag{4.17}$$

$$\mathbf{H}(\mathbf{x}) = \sum_i \left(2\mathbf{C}_i \left(\mathbf{x} - \mu_i\right)^\mathsf{T} \left(\mathbf{x} - \mu_i\right) \mathbf{C}_i - \mathbf{C}_i\right) g_i(\mathbf{x}). \tag{4.18}$$

Since $\mathbf{x}'$ is only defined up to an arbitrary scale, it has been set to be of unit length. The curve can be integrated using ODE integration techniques. which are discussed in section 4.5.1.

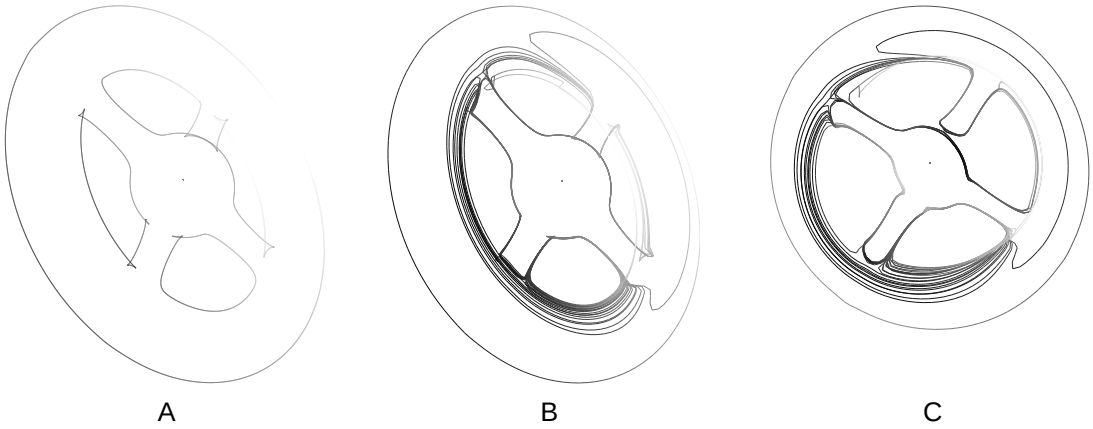


Figure 4.6: A set of contours from the spoked wheel in Figure 4.3. (A) The correct contour. (B) An incorrect solution of the contour caused by a large step size. (C) Contour B from a different viewpoint.

4.5.1 Solving the differential equation

4.5.1.1 Fixed step size solver

Equation 4.11 describes a differential equation, and this can be integrated using a standard 4^{th} order Runge-Kutta solver (finding $\mathbf{x}(0)$ is discussed in Section 4.5.2). An attempt was made to improve the solution by introducing a term that moves the solution back onto the apparent contour after each step. This technique was rejected since the choice of the step size is dominated by the need to avoid missing features in the contour, as opposed to getting a sufficiently accurate solution, so the extra accuracy added by the correction effectively adds nothing but computational effort.

The step size must be smaller than the feature size of the contour, otherwise the solver can skip over small, important features (such as points where the contour is nearly singular), and can jump onto another contour. An example of this is shown in Figure 4.6. The contour plotted in Figures 4.6 B and C has been calculated with a solver that has the step size set to be too large. In the solution of one of the contours, the solver jumps onto another contour, collecting a substantial error. It then loops around the other contours, jumping from one to another until eventually it happens to jump back onto the starting contour and end up where it started, terminating the solution.

The 4^{th} order Runge-Kutta method therefore suffers from two main problems. The first is that the step size has to be chosen at some point. This cannot be deduced easily from the model, so a conservative approach needs to be taken; the chosen step size must be smaller than necessary to make sure that the solution is unlikely to fail (the solution will always fail if the contour becomes singular). The other problem is that the smallest step size is only really needed at the small features and it is inefficient to take many very small steps where a few large ones would suffice.

The possible solutions [16, 30, 136] are either to make the steps more intelligent (like the Bulirsch-Stoer solvers), so that they do not miss the small features, or to choose the step size based on the local properties of the shape (like Runge-Kutta-Fehlberg solvers). The latter strategy has been investigated and is discussed in Section 4.5.1.2.

4.5.1.2 Variable step size solver

The idea behind a variable step size solver is to choose the step size in order to keep the error within certain bounds. Therefore, the error needs to be estimated, and a strategy for varying the step size is required. The algorithm used involves taking a 4^{th} and 5^{th} order Runge-Kutta step from the same point. The 5^{th} order step is considered to be closer to the ground truth than the 4^{th} order step, and is therefore used to estimate the error of the 4^{th} order step. By carefully choosing the way in which one takes the steps, the same function evaluations can be used for both of the steps. This is Fehlberg's method [115]. Given some function where:

$$\frac{dy}{dx} = f(x, y)\,, \tag{4.19}$$

the formula for the 5^{th} order step of length h is:

$$k_i = hf\left(x_n + a_i, y_n + \sum_{j=1}^{i-1} b_{ij} k_j\right) \tag{4.20}$$

$$y_{n+1} = y_n + \sum_{i=1}^{6} k_i c_i + \mathrm{O}\big(h^6\big)\,, \tag{4.21}$$

and the formula for the 4^{th} order step is:

$$\tilde{y}_{n+1} = y_n + \sum_{i=1}^{6} k_i \tilde{c}_i + \mathrm{O}\big(h^5\big)\,. \tag{4.22}$$

i	a_i	b_{ij} $_{j=}1$	2	3	4	5	c_i	$\tilde{c}_i$
1							$\frac{37}{378}$	$\frac{2825}{27648}$
2	$\frac{1}{5}$	$\frac{1}{5}$					0	0
3	$\frac{3}{10}$	$\frac{3}{40}$	$\frac{9}{40}$				$\frac{250}{621}$	$\frac{18575}{48384}$
4	$\frac{3}{5}$	$\frac{3}{10}$	$-\frac{9}{10}$	$\frac{6}{5}$			$\frac{125}{594}$	$\frac{13525}{55296}$
5	1	$-\frac{11}{64}$	$\frac{5}{2}$	$-\frac{70}{27}$	$\frac{35}{27}$		0	$\frac{277}{14336}$
6	$\frac{7}{8}$	$\frac{1631}{55296}$	$\frac{175}{512}$	$\frac{575}{13824}$	$\frac{44275}{110592}$	$\frac{253}{4096}$	$\frac{512}{1771}$	$\frac{1}{4}$

Table 4.1: The Cash-Karp [16] parameters for the Runge-Kutta-Fehlberg algorithm.

Algorithm 3 Estimating the optimum step size

given h_n, y_n, $\overline{\varepsilon}$
do
    calculate y_{n+1}, ε, using h_n as the step size
    estimate a better h_n based on ε and $\overline{\varepsilon}$
until $\varepsilon < \overline{\varepsilon}$
$h_{n+1} = h_n$

Therefore, the estimated error for the 4^{th} order step is:

$$\varepsilon = y_{n+1} - \tilde{y}_{n+1} = \sum_{i=1}^{6} (c_i - \tilde{c}_i) k_i. \tag{4.23}$$

The parameters used for the solver are the Cash-Karp parameters [16] (see Table 4.1) which have superseded Fehlberg's parameters in general usage.

If the maximum allowed error per step is $\overline{\varepsilon}$, then Algorithm 3 can be used for taking a step of a suitable size. The step is only recalculated if ε is larger than $\overline{\varepsilon}$, since the main problem with ε being too small is a lack of efficiency and this is not improved by recalculating y_{n+1} with a larger step. Once y_{n+1} is calculated, the optimum step size for that step can be estimated. The assumption is that the optimum step size for step n is approximately the same as the optimum step size for step $n+1$, so h_n is used as the first guess of h_{n+1}.

The error ε is approximately of $\mathrm{O}(h^5)$ with the step size, so the optimum step size, $\overline{h}$ could be calculated as:

$$\overline{h} = h\sqrt[5]{\left(\frac{\overline{\varepsilon}}{\varepsilon}\right)}. \tag{4.24}$$

However, if the error per step is $\mathrm{O}(h^n)$ then the *global* error is $\mathrm{O}(h^{n-1})$, since the *number* of errors introduced is $\mathrm{O}\left(\frac{1}{h}\right)$. The global error can appear to be better if the errors tend to cancel out as opposed to add up during the whole solution of the ODE. Since it is not clear exactly how the error varies with h, several strategies were implemented and tested. ε is taken to be the Euclidean distance between the two solutions. The different strategies are:

1. $$\overline{h} = \begin{cases} sh\sqrt[5]{\overline{\varepsilon}/\varepsilon} & \overline{\varepsilon} > \varepsilon \\ sh\sqrt[4]{\overline{\varepsilon}/\varepsilon} & \text{otherwise} \end{cases}. \tag{4.25}$$

   This is the conservative approach suggested by [115]; if the error is small, then increase the step size by the smaller amount, otherwise decrease it by the larger amount. A scaling factor s of slightly less than unity has been introduced because ε is usually an overestimate.

2. $$\overline{h} = sh\sqrt[5]{\overline{\varepsilon}/\varepsilon}. \tag{4.26}$$

3. $$\overline{h} = sh\sqrt[4]{\overline{\varepsilon}/\varepsilon}. \tag{4.27}$$

4. $$\overline{h} = \begin{cases} h/s & \overline{\varepsilon} > \varepsilon \\ hs & \text{otherwise} \end{cases}. \tag{4.28}$$

   This approach is based loosely on the earlier step-doubling techniques.

By finding the average optimum s over a number of contours and finding the total number of steps (including rejected ones) required, the relative merit of the four techniques can be tested. Methods 1–3 vary by less than 2% in the total number of steps required. Method 1 comes out slightly better (though this result could well be within measurement error), with an optimum of $s = 0.853$. Method 4 is atrocious by comparison, requiring about 1.6 times as many steps.

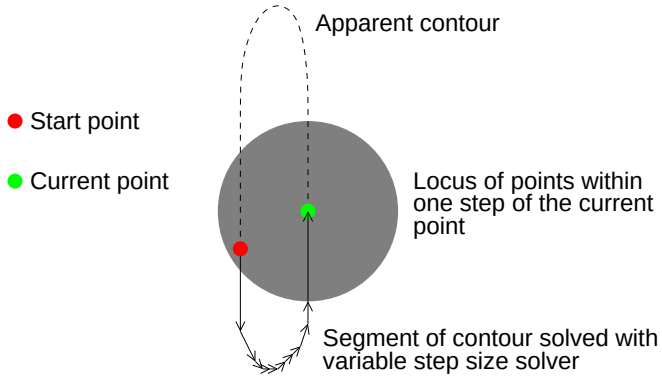


Figure 4.7: With a variable step sized solution, a non terminal point can pass within one step of the initial point.

4.5.1.3 Termination strategies

The discussion about solving the ODE to find the contour generators has skipped over a point, namely how the solution of the ODE terminates. A common termination strategy for the solution of ODEs is to run the solver for a fixed distance, i.e. until $\mathbf{x}$ reaches a certain value in the system described in Equation 4.19. Another common strategy is to run the solver until the solution converges (if indeed the ODE does converge). The ODE describing the contour generator is not amenable to either strategy. Firstly, the solver cannot be run for a fixed distance because the length of the generator is not known. Second, the contour generator is a closed loop, so the solution does not converge. Instead, it oscillates where the period of oscillation is the length of the generator.

The strategy employed for the fixed-step solver is to terminate the solution when the solution comes within a small multiple of the step size of the starting point. This is guaranteed to work because the step size must be smaller than the feature size, so if the solution is within approximately one step of the start, then it must be back at the start as opposed to near it but on a different feature. This does not work for a variable step sized solver. Figure 4.7 shows the solution of the variable step sized solver around a long, thin feature. After getting approximately one third of the way around, the current point is within one step of the end. Clearly the solution should not terminate here. There are several criteria which can be enumerated that a point, $\mathbf{x}$, must obey before it can be considered the end of the solution, namely:

1. $\mathbf{x}$ must be close to $\mathbf{x}(0)$

2. The contour at $\mathbf{x}$ must be aligned with the contour at $\mathbf{x}(0)$. That is:
   (a) The contours must be pointing in the same direction, i.e. $\mathbf{x}' \approx \mathbf{x}'(0)$
   (b) A step from the current point of the right length must land very close to the starting point.

It is possible to meet all but one of the conditions at a point which is not the end of the curve. These tests have been implemented and have led to significant improvements in efficiency. However, there are now four parameters which need tuning, instead of one for the fixed step sized solver.

4.5.2 Finding contours

In order to calculate contour generators, suitable starting points need to be found. Exactly one starting point on each contour needs to be found. The method for finding points is split into two stages, the first of which is performed offline. The offline stage scatters a number of points (typically 500–2000) over the surface of the object. Since the position of the surface is not known, this is done by scattering points randomly over the surface of the individual Gaussians and then moving the points to the surface using Newton's method. Each Gaussian has approximately the same number of points scattered over it. Broadly speaking, a high density of Gaussians corresponds to a high density of surface features, so this method (as opposed to scattering points evenly over the surface) ensures that there are more points where there is more detail.

At run time, candidate points on the surface close to the generators are chosen by selecting points where the surface is nearly parallel to the viewing ray, typically within 5°. An iteration scheme involving a Newton-Raphson step along the viewing ray, followed by a step toward the surface is used to move the points onto a contour generator. Points close to existing contour generators are then rejected. The proximity test is performed using the rapid search techniques described in Section 4.5.3.

4.5.3 Fast contour search techniques

In the process of rendering the visible contour, there are three operations which need to be performed rapidly on the contours and contour generators. These are

Algorithm 4 Fast proximity detection.

```
bool Proximity(Tree node T, Point p, Distance d)
    if(p is within d if the bounding box of T)
        if(T is a leaf node AND p is within d of the line segment in T)
            return true
        else
            return Proximity(T.left, p, d) OR Proximity(T.right, p, d)
    return false
end
```

Algorithm 5 Fast piecewise linear contour intersection.

```
form an empty list of pairs of line segments, L

Intersect(Tree Node m, Tree Node n)
    if(bounding boxes of m and n intersect)
        if(m and n are leaf nodes)
            if(the line segments in m and n intersect)
                add (m, n) to L
        else
            call Intersect on all pairs of children
end
```

testing the proximity of a point to a contour generator, determining if a point is inside a contour and finding all intersections between contours. The two and three dimensional contours consist of a loop of connected line segments. The ends of the line segments are at the points generated by integrating Equation 4.11. If the step size needed for accurate integration is small, then the contours can consist of a large number of line segments. As a result, a simplistic implementation of the tests runs very slowly.

A fast method of performing these tests has been developed which makes use of a balanced binary tree. The leaves of this tree contain a line segment and a bounding box. For each node, a bounding box is computed that exactly contains the bounding boxes of the child nodes. To determine if a point is near a contour, Algorithm 4 is used. The worst case execution time is $O(N)$, but the typical running time is $O(\log N)$. Algorithm 5 is used to calculate intersections between contours rapidly. The worst case execution time is $O(N^2)$, since there can be at most $O(N^2)$ intersections, but the typical running time is $O(\log N)$. If a point is

inside a polygon, then an infinite ray from that point will cross an odd number of edge segments of that polygon. A similar algorithm is used to perform this test in approximately logarithmic time.

Intersection of polygons is often performed using line-sweep based algorithms. In the system described here, the algorithms described perform algorithmically better. The slowest operation in the line-sweep algorithms is setting up the data structure. This requires turning the polygons into a sorted list of line segments, and as a result takes time of $\mathrm{O}(MN \log N)$, where M is the number of contours and N is the number of segments per contour. Calculating intersections between all contours is then linear in the number of line segments, taking $\mathrm{O}(MN)$ time. The algorithms presented here perform differently. The setup time is linear and the time taken to compare a pair of contours is $\mathrm{O}(\log N)$. However, all possible pairs of contours have to be tested, so the time taken is $\mathrm{O}(M^2 \log N)$. In all the cases tested so far, $N \gg M$, so the tree based algorithms perform better. It should be noted that the worst case performance for both algorithms is $\mathrm{O}((MN)^2)$, since there can be at most that many intersections.

4.6 Tracking

In order to track the object from an image, the derivatives of visible boundary points (in the image) with respect to the motion parameters must be calculated. As the camera moves, the boundary moves in the image. One component of this motion is due to the motion of the contour generator relative to the camera. The shape of the contour depends on the position of the camera, i.e. as the camera moves, the contour slips across U. This is the second component of the image motion.

The contour generator is by definition at a point where the surface is at a tangent to the viewing ray. Any small motion across the surface will therefore be along the viewing ray and will cause no image motion. More formally, $\mathbf{x}_0$ is a point on the contour generator and $\mathbf{x}_1$ is the point on U which corresponds to $\mathbf{x}_0$ after a small camera motion. There is a one parameter ambiguity in this correspondence. There are a family of points on the apparent contour visible after a small motion that could reasonable correspond to $\mathbf{x}_0$. The point $\mathbf{x}_1$ has to be defined. It is this point correspondence that is used for tracking and it can be shown that with this correspondence there is no image motion due to the contour slipping over the surface. Correspondence is found by defining a *correspondence plane*. The

plane is defined by the point ($\mathbf{x}_0$) in the first view. The corresponding point to $\mathbf{x}_0$, that is $\mathbf{x}_1$ is defined to be where the contour from the second view intersects this plane. The correspondence plane is described by the camera centre, $\mathbf{c}$, the initial point, $\mathbf{x}_0$ and the surface normal, $\nabla f(\mathbf{x}_0)$. The interpretation of this in the image is that a point on the second contour lies normal to the edge at the point on the first contour. The position of $\mathbf{x}_1$ can then be written as

$$\mathbf{x}_1 = \mathbf{x}_0 + a\left(\mathbf{x}_0 - \mathbf{c}\right) + b\left(\nabla f(\mathbf{x}_0)\right) \tag{4.29}$$

where a and b parametrise the position on this plane along the viewing ray and normal to the surface. We also know that $\mathbf{x}_1$ must lie on U, i.e. $f(\mathbf{x}_1) = 0$, so

$$f(\mathbf{x}_0 + a\left(\mathbf{x}_0 - \mathbf{c}\right) + b\left(\nabla f(\mathbf{x}_0)\right)) = 0. \tag{4.30}$$

Performing a Taylor series expansion of this up to first order terms gives

$$f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)\left(a\left(\mathbf{x}_0 - \mathbf{c}\right) + b\left(\nabla f(\mathbf{x}_0)\right)\right) = 0. \tag{4.31}$$

Substituting Equation 4.3 and Equation 4.4 gives

$$b\left\|\nabla f(\mathbf{x}_0)\right\|^2 = 0 \tag{4.32}$$

and therefore $b = 0$. Using this correspondence $\mathbf{x}_0$ corresponds to $\mathbf{x}_1$ where:

$$\mathbf{x}_1 = \mathbf{x}_0 + a(\mathbf{x}_0 - \mathbf{c}). \tag{4.33}$$

This shows that motion of $\mathbf{x}_0$ is only along a ray to a camera, and therefore produces no image motion of $\mathbf{p}$. For small motions, the visual motion of the contour generator is equivalent to the visual motion of a rigid wire frame. The shapes are tracked using a standard technique [35]. This tracker works by linearising the edge-normal image motion with respect to the six pose parameters, and therefore uses the correspondence described above.

When there is a cusp in the image, it is possible for the motion of the contour generator to lie outside the correspondence plane. When this happens, the approximation of the motion is inaccurate, which would make tracking of cusps inaccurate. This point has not been addressed because cusps are very weak image features, as can be seen in Figure 4.8. Since they are so weak, they are not particularly useful features to track. Further, the robust optimiser removes problems associated with any mismatching of edges which may occur at the cusps.

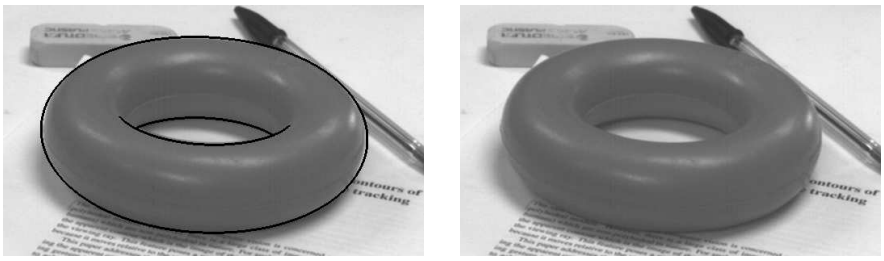


Figure 4.8: Cusps are very clear in the rendered outline of the torus (left), but in practice they are very hard to localise in the image (right).

4.6.1 Results

The system was tested by tracking a simple object (a torus) and a more complex object (a desk lamp). The torus can be seen in Figure 4.8. Figure 4.9 A is of a rendered image of the lamp model. Figure 4.9 B–D show the lamp being tracked in various orientations. The system was fast enough to track the incoming video stream at frame rate (25 frames per second). The model of the lamp contains 13 primitives (143 parameters), and approximately 0.025s (on a Pentium processor at 850 MHz) is required to calculate the visible apparent contour. The parameters along with annotations are given in Appendix D. The model involves collections of positive Gaussians with a scale of order unity for defining the geometry, and Gaussians with very large negative scalings to cut out flat sides.

4.7 Conclusions

This section has presented a method for efficiently rendering smooth surfaces described by an implicit function. The efficient rendering has arisen from a combination of several items:

- A differential equation for tracing out the contour generators, which can be integrated using standard, efficient techniques.
- A method of determining the the visibility of the apparent contours, based on their intersections.
- A new family of algorithms able to rapidly compute intersections of 2D contours and proximity to 3D contours. These are essential for efficiently determining starting points of the differential equation integrator and intersections for the visibility computation.

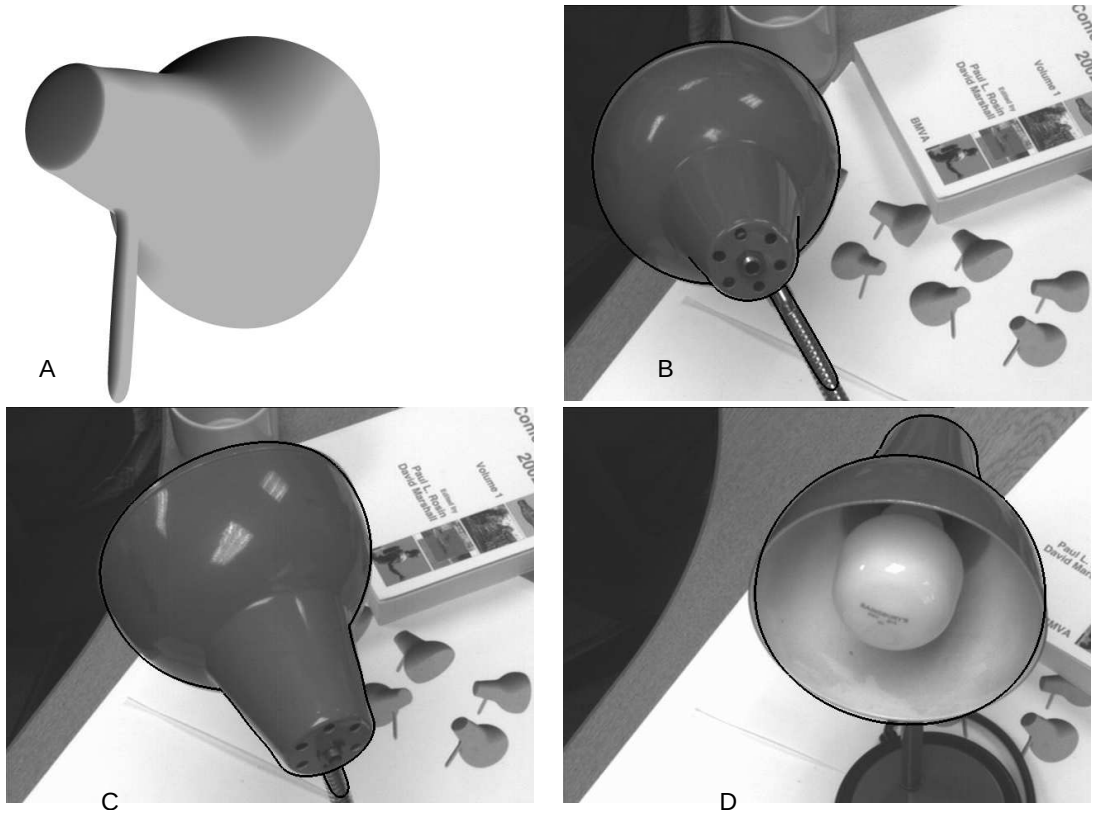


Figure 4.9: (A) the model of the lamp. (B)–(D) the lamp being tracked in various poses.

5. Sensor fusion

5.1 Introduction

The subject of this thesis is high performance (robust and efficient) tracking. This section is about how the systems presented so far can be combined in to a complete system which exhibits high performance. So far, two tracking systems have been presented, a point based tracking system and an edge based tracking system. These two tracking systems are complementary: the point based tracking system is very robust to large, unpredictable camera motions but suffers from drift and the point based tracking system is drift free but brittle (this will be discussed in more detail in Section 5.3). Because of this, it is natural construct a system to combine these in such a way that the point based tracking system makes the system robust, and the edge based system bakes it drift free.

5.2 Previous work

In multi-cue tracking, several cues in the image are used (for instance edges and points), by combining the measurements from multiple tracking systems. The idea behind this is that the system can then cope with failures in the different trackers.

Consider the case of combining two tracking systems. If the statistics of the system are Gaussian, and the dynamics are linear, then the Kalman filter is optimal [64]. This can be extended to nonlinear systems, with the two most popular techniques being the EKF (Extended Kalman Filter)—which linearises around the current point—and the UKF (Unscented Kalman Filter) [61], which

passes enough points through the nonlinearities to measure the new variance afterwards. This system is capable of dealing with certain kinds of failures. If one of the trackers is weakly constrained along some eigenvector, the measurement can still be accurate in the other dimensions (as long as regularisation is used in the optimisation step), so it still provides useful information when added into the Kalman filter. As long as the other tracking system constrains that dimension (though other dimensions can be unconstrained). When the measurements are added, the final pose will end up being well localised. This extends all the way to one tracker providing no information at all. This is the approach used in [146] in which point features and edges are combined. In [137], this is also combined with colour blob based tracking, and this takes advantage of the filter being able to use under constrained measurements, since the colour blob tracker only measures the 2D image motion of the object centroid.

This does not take into account some of the different properties of the tracking systems. In [46] multi-cue tracking is applied to the snake tracking problem. Snakes suffer from the edge correspondence problem, so point based tracking is performed before snake tracking to initialise the new position of the snake. This helps correct one failure mode due to the lack of robustness of one tracker, but no longer allows either of the trackers to fail. A system based on a large number of trackers is presented in [141]. These range from coarse and inaccurate but robust to highly accurate but brittle. Properties of the trackers and a finite state machine are used to select which tracker is to be used.

Colour, pattern detection and stereo are combined in [26] to track people. The trackers are combined robustly. If a high precision tracker is available, it is used, otherwise lower precision trackers are combined. The high precision trackers are tuned such that false negatives are greatly preferred to false positives. In other words, the system is designed to cope with missing rather than false information.

If the measurements do not have Gaussian noise, then the Kalman filter is not suitable, nor is any unimodal filter: multiplying the prior and measurement PDFs (probability density functions) will inevitably lead to multiple modes in the posterior distribution. This happens when a tracker can fail completely—it will provide accurate data most of the time, but once in a while, the answer will be wildly incorrect, even when there is plenty of data. An example of this is given in Figure 5.1. In this case, the prior is Gaussian, but the measurement is not (the measurement is represented as a mixture model, one component representing good measurements, the other representing bad ones), and the resulting posterior is multimodal. In other words, if the measurement was good, then the ground truth is probably near one mode, otherwise it is probably near the other.

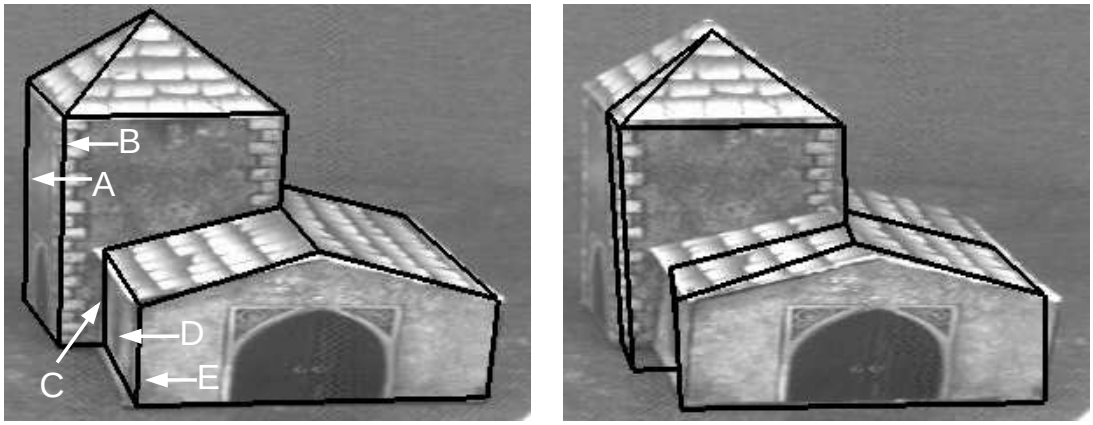


Figure 5.1: Left: model in the correct position. Right: edge tracking fails because model edges (A) and (B) lock onto image edge (B) and model edges (C) and (E) lock onto image edge (D).

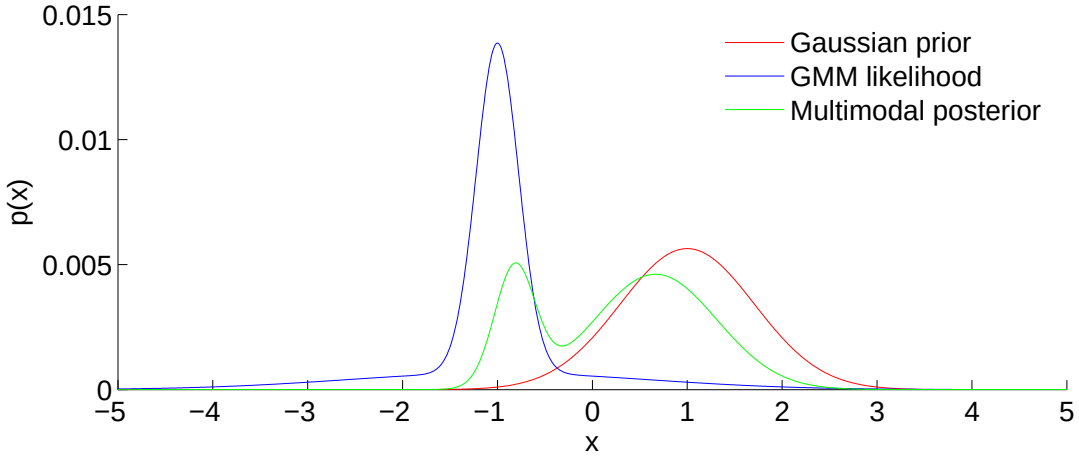


Figure 5.2: A 1D example of a Gaussian prior combining with a mixture model likelihood to produce a multimodal posterior.

An example of this is given in Figure 5.2.

CONDENSATION (CONditional DENSity propagATION) [56] is a tracking system which uses a multimodal representation. In each frame, the posterior is represented by N points, each weighted with a probability (particles). In a new frame, N new points are drawn randomly (taking the weighting into account) from the old ones, and drift (to take into account dynamics) and random noise are added to the particle positions. A measurement is taken at each particle to determine the probability of correctness (the measurement does not include an

optimisation step). In this case, the model is a spline curve and is used to track nonrigid objects in a video at frame rate. The particle propagation technique is also known as particle filtering [45].

This technique does not scale well with the number of dimensions; the number of particles required to adequately represent the distribution increases rapidly with the dimensionality. This is demonstrated in [31], where particle filtering is applied to human body tracking (29 DOF). Even with 40,000 particles (four seconds of video taking 30 hours to process), tracking is not successful. They propose *annealed particle filtering* to solve this problem. In this system, a broad measurement function is used to compute the particle weightings, effectively smoothing out the space, so far fewer particles are needed. Then the measurement function is narrowed, the system is resampled, and weightings are recomputed. This is repeated several times, allowing the system to converge on the global maximum with far fewer particles.

In some cases, *partitioned sampling* [93] can be used. In this case, two objects are being tracked simultaneously, and they can be tracked nearly independently. So, instead of tracking the pair together in a high dimensional space, they are tracked independently in the two low dimensional spaces, after which, the results can be combined. This is suitable for articulated objects and is applied to hand tracking in [94].

In particle filtering, measurements are used to compute the probability of the particles, and in Kalman filtering, the measurements are combined with the prior. These are combined in the *unscented particle filter* [100], where particle positions are updated by observations using a UKF. This would be useful in tracking situations where a tracker can be used which has a large radius of convergence: only one particle has to exist in the well for tracking to be successful. A parametric system which is evolved over time by tracking the modes of the distribution is used in [17]. This applies to high dimensional tracking (human body) where particle filtering is unsuitable.

5.3 Sensor analysis

Before attempting to combine these two approaches for pose estimation, it is first necessary to look in more detail at the *preconditions* and *postconditions* for each in order to understand the differences in statistical behaviour. A summary of these is first presented in order to provide a basis for the discussion that follows.

Point based tracking	
Preconditions	
3D point cloud/model.	$\mathcal{P}_1^-$
Postconditions	
Produces robust differential measurements...	$\mathcal{P}_1^+$
...with approximately Gaussian posterior.	$\mathcal{P}_2^+$
Posterior measurement covariance is inaccurate.	$\mathcal{P}_3^+$
Edge based tracking	
Preconditions	
Geometric 3D edge model.	$\mathcal{E}_1^-$
Accurate pose prior.	$\mathcal{E}_2^-$
Postconditions	
Non-Gaussian posterior.	$\mathcal{E}_1^+$
Drift-free measurement.	$\mathcal{E}_2^+$

5.3.1 Point features

Condition $\mathcal{P}_1^-$: In Section 2 it was argued that obtaining a static point cloud for large scenes is infeasible, and not necessarily useful. As a result, it is necessary to dynamically learn this model and this is achieved by back-projection onto a geometric 3D surface model. Because there are no static features in this point cloud, the tracker can only produce differential pose measurements (**Condition** $\mathcal{P}_1^+$).

Condition $\mathcal{P}_2^+$ arises because we can determine which point matches are correct and which are not with high probability (see Section 2.5 and Section 2.5.1). The measurement errors of the inliers are mostly due to pixel quantisation (we do not use a sub-pixel feature detector—see Section 3.3) and so are independent. The likelihood therefore approaches a Gaussian distribution by the central limit theorem.

Condition $\mathcal{P}_3^+$: Although the measurement errors are independent, the errors in the 3D point cloud are not. Points detected on unmodeled structural clutter are back-projected onto the closest modelled plane, which is almost always further from the camera than the clutter. This is simply because there are fewer unmodelled windows than unmodelled solid objects. The result is that any errors in the 3D point cloud may well be strongly correlated, and therefore it is found that the covariance obtained from the point matches is inaccurate. As a result

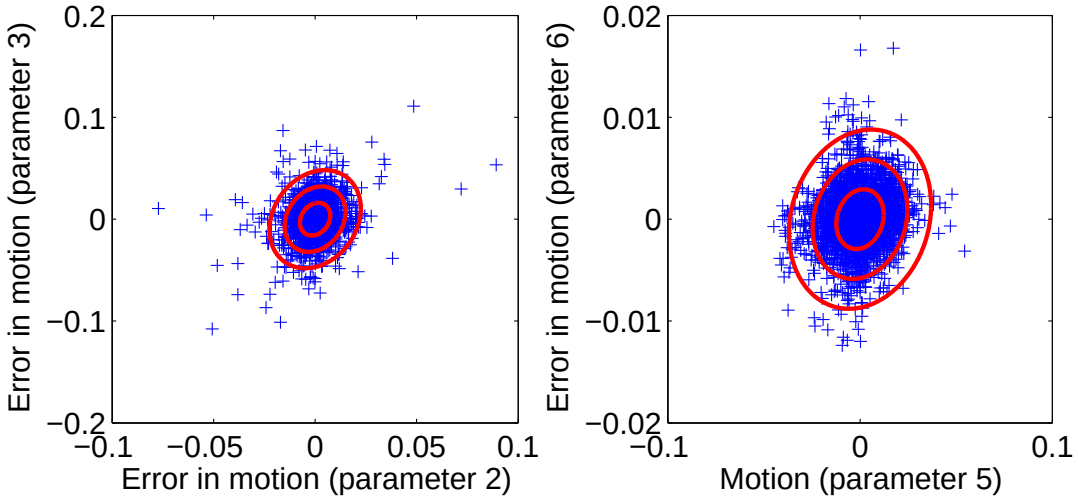


Figure 5.3: The errors between the point tracking posterior and the ground truth are well modelled by uncorrelated statistics. To demonstrate this, the two strongest correlations have been shown and even these are only weakly correlated.

the covariance must be modelled. Two models for the covariance are considered. The first model is that the covariance, $\mathbf{C}$, can be modelled as a function of the motion, $\boldsymbol{\mu}$:

$$\mathbf{C} = \mathbf{A} + \mathbf{B}\boldsymbol{\mu}\boldsymbol{\mu}^{\mathsf{T}}\mathbf{B}^{\mathsf{T}} \tag{5.1}$$

This is tested by using data which is obtained from a sequence where the pose is found by manual alignment in each frame. It is found that $\mathbf{A}$ is largely diagonal, and $\mathbf{B}$ consists of only very small values. The data corresponding to the largest off-diagonal element of $\mathbf{A}$ and the largest element of $\mathbf{B}$ is shown in Figure 5.3. The second model considered assumes that the shape of the covariance obtained from the data (see Section 2.5) is correct, but that it is over-saturated by a constant, k. The most likely value of k maximises the log-likelihood of the data and is given by:

$$k = \underset{k}{\operatorname{argmin}} \left(-\sum_i \mathbf{e}_i^{\mathsf{T}} (k\mathbf{C}_i)^{-1} \mathbf{e}_i - \ln \sqrt{(2\pi)^6 |k\mathbf{C}_i|}, \right) \tag{5.2}$$

where $\mathbf{C}_i$ is the computed covariance for frame i and $\mathbf{e}_i$ is the 6 DOF pose error for frame i, obtained from the ground truth data. It is found that $k \approx 7200$.

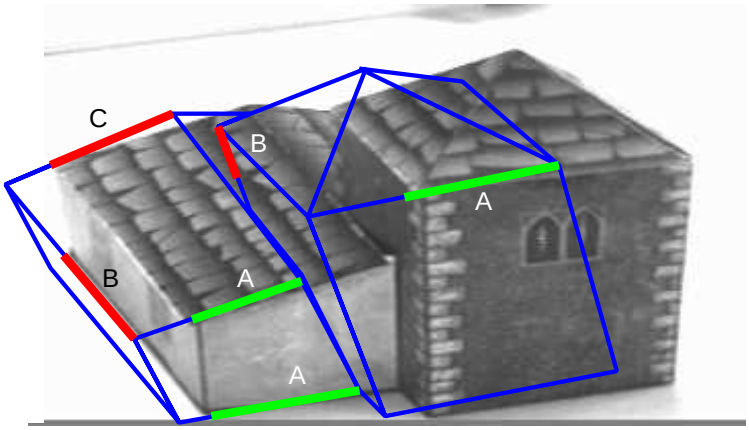


Figure 5.4: An example of edge based tracking failing when the initial position is not sufficiently close to the correct position. Some model edges (A) correspond to the correct image edges. Other model edges (B) have an incorrect image edge closer than the correct one. The worst model edges (C) do not have the correct image edge anywhere in the direction of the normal. This particular configuration of incorrect correspondences causes the tracker to converge on the position shown rather than the correct position.

5.3.2 Edge tracking

Condition $\mathcal{E}_1^-$: In order to perform edge tracking a 3D edge model of the object to be tracked. This is the model onto which features are projected for **Condition** $\mathcal{P}_1^-$. The model is created by hand. Edge features are invariant to lighting and perspective changes, hence the model can remain static. Because this model is static, the measurements obtained will be drift-free (**Condition** $\mathcal{E}_2^+$). Highly invariant features are not discriminative, so in order to avoid incorrect edge correspondences, a strong prior image edge position and hence model pose is required (**Condition** $\mathcal{E}_2^-$). Figure 5.4 illustrates the kind of failure mode that happens when the prior is too poor. Even with a good prior, edges can still be detected incorrectly, as illustrated in Figure 5.1. These incorrect measurements will often be strongly correlated and hence the pose estimate will contain a large error. The correlation in the error means that the posterior pose does *not* approach a Gaussian by the central limit theorem (**Condition** $\mathcal{E}_1^+$). We therefore model this distribution as a two component GMM (Gaussian Mixture Model), consisting of a narrow Gaussian (the distribution of poses where the correspondences are correct) and a very broad Gaussian (the distribution when edge correspondences are incorrect).

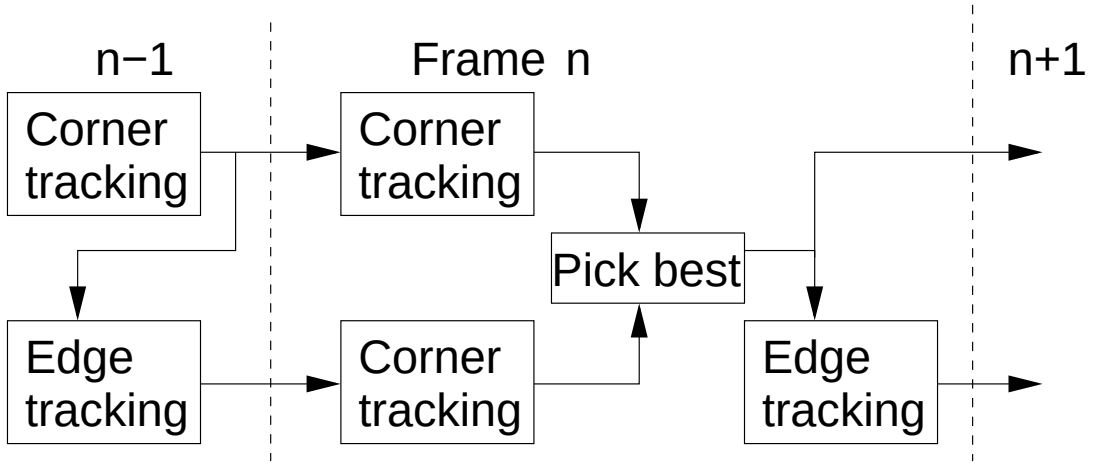


Figure 5.5: Block diagram showing data flow for the sensor fusion algorithm over three frames.

5.4 Sensor fusion

Both feature based and edge based tracking have failure modes, but these are complementary and so combining them leads to a more robust system. Because of the non-Gaussian statistics of the system, measurements cannot be trivially fused by using linear techniques such as Kalman filtering, so several strategies are needed to robustly combine the measurements. The above analysis leads to the following conclusions:

1. Points are robust to large motions (**Condition** $\mathcal{P}_1^+$), and lines need reasonably accurate initialisation (**Condition** $\mathcal{E}_2^-$), hence points should be treated first and lines second.

2. The statistics of line measurement are non-Gaussian (**Condition** $\mathcal{E}_1^+$) so a non-linear filter is needed.

The pose estimate covariance $\mathbf{C}_a$ and point cloud (**Condition** $\mathcal{P}_1^-$) from the previous frame are used. The point tracker adds a differential measurement (**Condition** $\mathcal{P}_1^+$) resulting in a posterior covariance $\mathbf{C}_b = \mathbf{C}_a + k\mathbf{C}_f$, where $\mathbf{C}_f$ is the covariance measured by the feature point tracker (see Section 2.5).

The brittle, but precise edge tracker is initialised (**Condition** $\mathcal{E}_2^-$) using robust differential measurements from **Condition** $\mathcal{P}_1^+$, as in [72]. Since the likelihood

given by the edge based tracker is a two-component GMM (**Condition** $\mathcal{E}_1^+$) and the prior is a Gaussian, the posterior is also a GMM which may have two modes (see Figure 5.2). This posterior for pose is then used to obtain the 3D point cloud needed for the next frame. Since the posterior can be bimodal, a separate point cloud is generated for each mode. Note that if the edge tracker is correct, this estimate of posterior pose is drift-free (**Condition** $\mathcal{E}_2^+$).

If both modes were to be propagated all the way through the next iteration, exponential mixture component explosion would follow since edge tracking doubles the number of mixture components at each iteration. This is avoided by comparing the performance of each point cloud on the subsequent point tracking stage. The GMM that gave rise to the point clouds gives their relative probability, while the difference in residual error in point tracking provides an estimate of their likelihoods. These are then combined and only the Gaussian component (and associated point cloud) with the highest posterior probability is retained for the edge tracking stage. The algorithm described above is summarised below:

1. A new frame arrives and point features are detected.
2. Correspondences are found between the new features and existing features on the model.
3. The probability that a match is correct is computed from the correspondence score.
4. The pose is robustly computed for both modes, and the most probable mode is kept.
5. The new pose is used to initialise and run an edge tracker.
6. The features are back-projected onto the model.
7. The learned relationship between matching score and matching probability is updated based on the posterior match probability.

This process is illustrated in Figure 5.5.

Results have been taken on 768×288 pixel fields from a 50Hz PAL source: a Pulnix TM-6EX video camera with a 4.2mm lens and the shutter speed set to the highest value. The system operated at frame rate on a Pentium 4 Xeon at 2.4GHz. The use of fields as opposed to frames removes problems associated with interlacing.

5.5 Results

The robustness of the overall algorithm, has been demonstrated on three different scenarios. The first is rapid rotational camera shaking. Shaking in this mode causes very large inter-frame image motion. However, the test scene is not particularly challenging for the edge based tracker, so this mainly tests the robustness of the feature based tracking system. However, the edge based tracker must still be present, otherwise the sequence would be untrackable due to drift. A graph of the angular position of the camera tracked over time in shown in Figure 5.6. Some example frames are shown from this sequence in Figure 5.7 and these are representative of the entire sequence. The average image motion of features throughout the scene is 79 pixels and the largest average image motion of a single frame is just over 204 pixels. In this sequence, a zero order motion model is used (no velocity information is tracked) and indeed, such a velocity model is unlikely to be of much use since the camera experiences angular accelerations of up to 88,500° s^{-2}.

The second test shows an explicit example of the advantage of the multimodal posterior propagation. A scene was constructed that contained a large number of strong unmodelled edges, both on the model itself, and also in the surrounding environment. The unmodelled edges frequently cause failure of the edge tracking system. An example is shown in Figure 5.8. As can be seen, the motion is not particularly challenging (especially compared to the camera shaking), but the edge tracker fails during rotation where a previously occluded edge becomes visible and locks onto the incorrect image edge. Since the tracking system can cope with failures of the edge based tracker, tracking is not lost.

The third test was performed on an extended sequence taken from a hand-held camera inside a laboratory and is 1300 fields in length. The maximum tracked camera velocity is 12ms^{-1}. Excerpts from this are shown in Figure 5.9. This figure also shows one of the limitations of the system: if the geometry of the world is too simple (for instance planar), then the point based tracker cannot make a useful guess as to the quality of the edge based tracking result. Another degeneracy (not shown) is pure camera rotation, since the point motion can be modelled as a homography.

This chapter has presented a high performance tracking system based on the combination of two different tracking systems with complementary behaviour and very different statistics. By employing a careful analysis of the requirements and behaviour of these systems, their synthesis into a single system has been

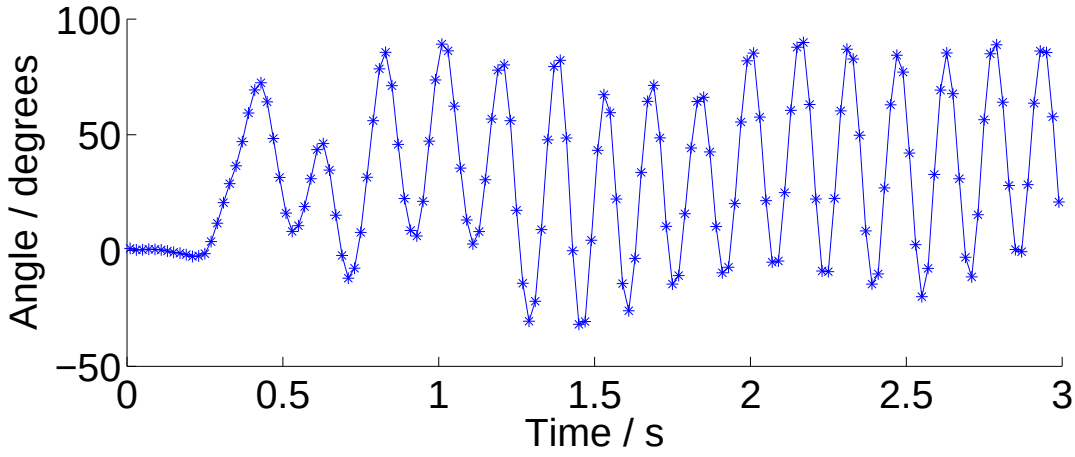


Figure 5.6: Graph showing the angle of a hand-held camera undergoing vigorous rotational shaking at approximately 6Hz. The limiting factor is the rate and range over which the author could shake the camera.

enhanced. This includes the use of one system to initialise the other, and a non-linear method for combining the two posteriors.

5.6 Conclusions

This chapter has presented a method for combining the two different tracking systems in such a way that the overall tracking system is both robust and drift-free. The nonlinear filter presented came from a careful analysis of the properties of both the tracking systems. The properties led to the conclusion that:

1. A nonlinear filter must be used.
2. A unimodal filtering system is not sufficient.

The is a tracking system which is considerable more robust than either of the two systems, and the results reflect this by showing cases where the edge tracker can make large corrections (required because of drift) and cases where the edge tracker is ignored (when it produces bad measurements).

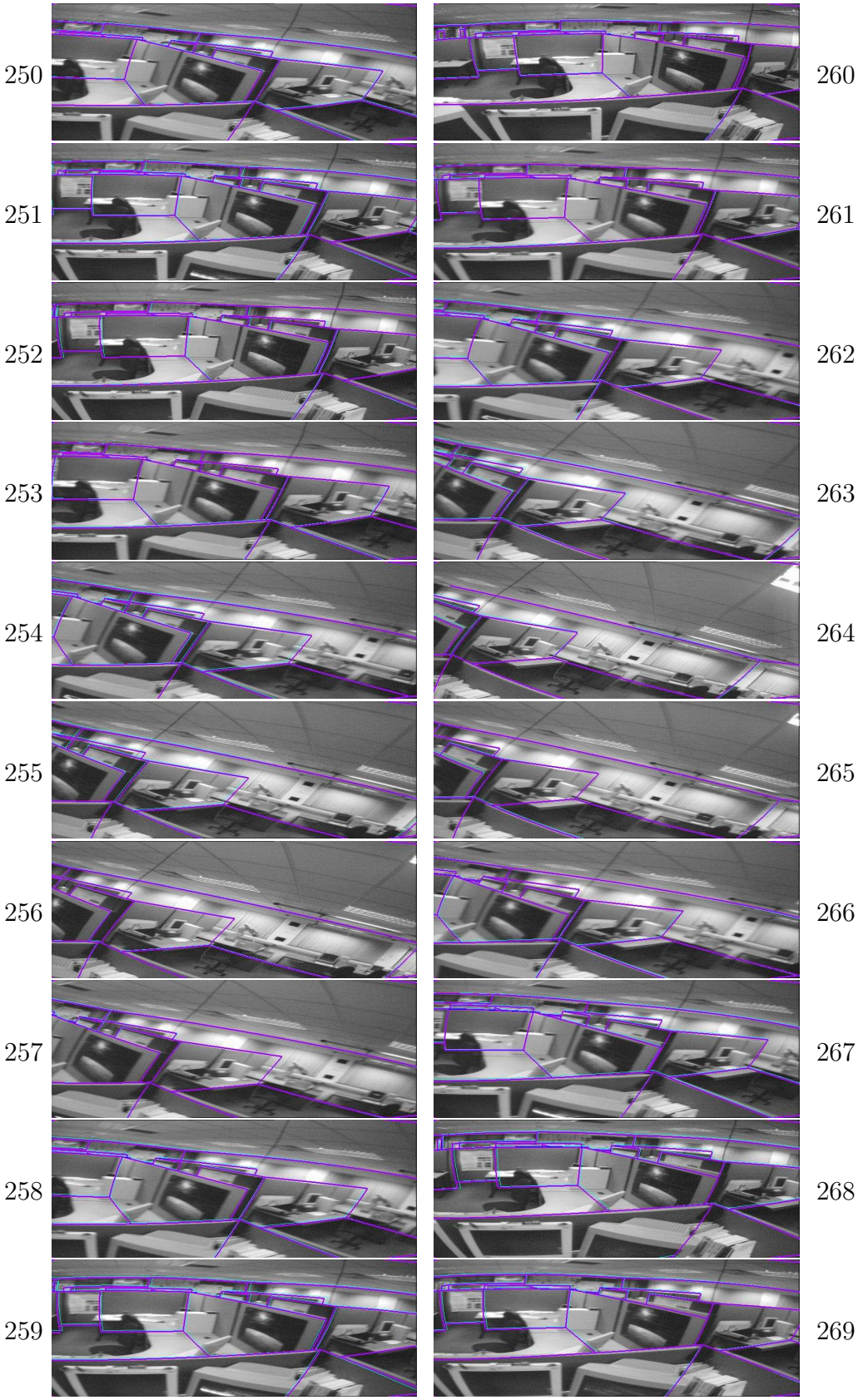


Figure 5.7: Frames 250–269 from the vigorous shaking sequence. The frames shown cover 0.4 seconds of the video.

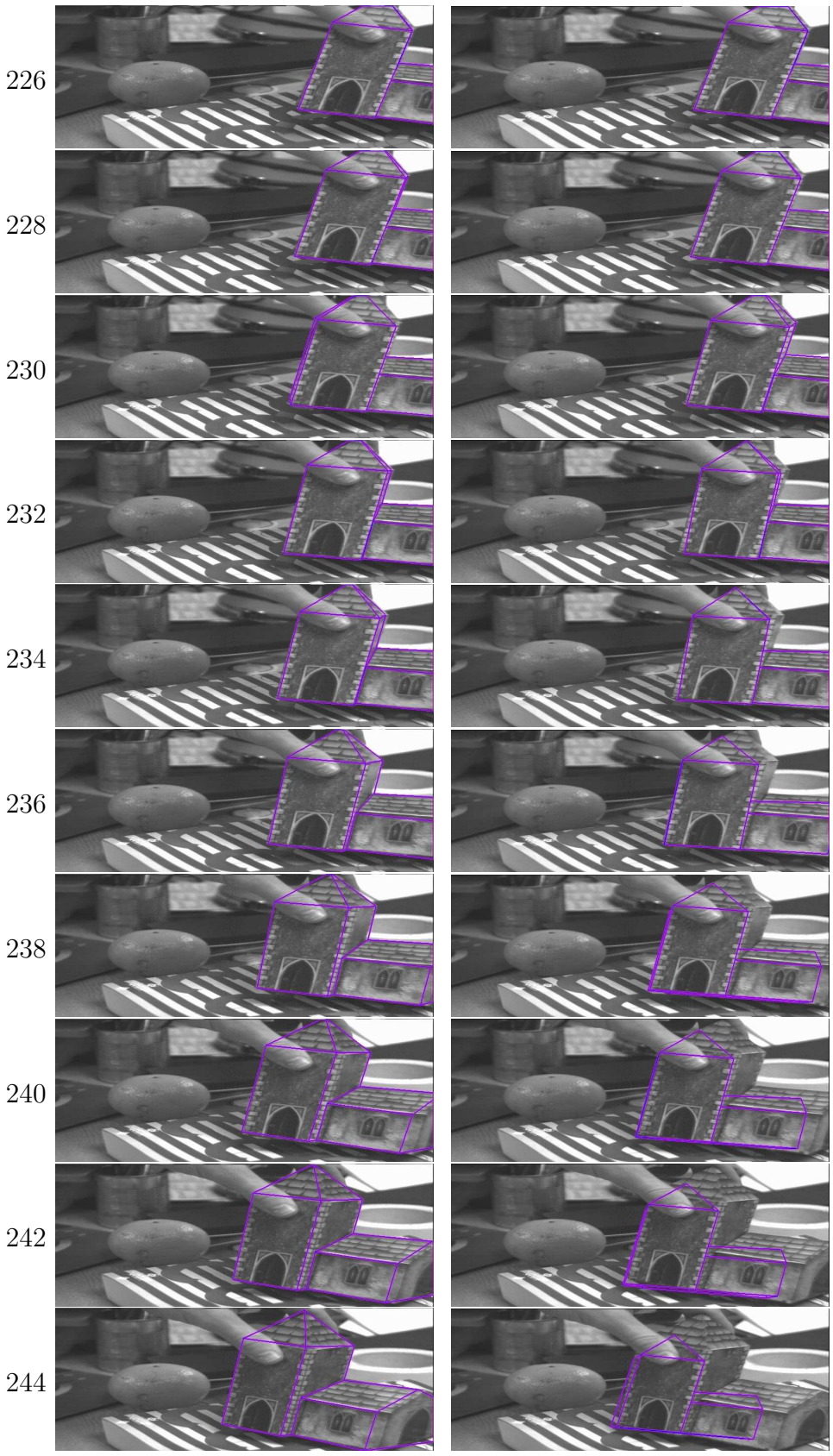


Figure 5.8: Frames from the video sequence designed to test the sensor fusion. Note the large number of strong unmodelled edges both on the model and in the environment. Tracking is shown for two filters, (left) multimodal posterior propagation and (right) point tracking followed by line tracking. Using the simple filter, tracking has been lost by the last frame.

Figure 5.9: Three excerpts from an extended sequence. Left (frames 556–565): rapid camera rotation about the optic axis. Centre (frames 769–768) some ill conditioning has caused the pose to drift slightly (as can be seen in the lower left corner). When it is judged to be good, the edge tracker can make large changes to the pose, and thus the drift is corrected. Right (frames 1286–1295): with very few edge features, the point tracker drifts. At the end, when edge features become visible, the edge tracker makes a large, incorrect update. Since the world is planar, the point based tracker is unable to judge which is better, and in this case it makes the wrong choice.

6. Conclusions

This thesis has made a number of significant contributions to the field of tracking and feature detection. These contributions were required to build a high performance tracking system and Figure 6.1 illustrates how all these components fit together. Essentially, FAST feature detection, efficient feature matching, a robust optimiser and on-line estimation of match quality are used to build an efficient and robust point tracking system. The point based tracking system suffers from drift, so an edge based tracking system has been used to tackle this. The two tracking systems have very different properties and robustness has been achieved by combining the tracking results in a nonlinear filter. The contributions of the components are detailed here:

An efficient and robust point based tracking system. This relies on efficient feature detection and matching, for which a new algorithm is described. Furthermore, by using EM as the optimisation algorithm, a feedback loop was created which allowed the system to estimate the quality of the match from the matching score. This is published in [118]. The resulting tracker is easily able to cope with prediction errors in translation causing 200 pixels image motion, and rotations of 15°, even when there are considerable quantities of outliers in the data.

FAST feature detector. The FAST detector was developed to allow full-frame feature detection to be performed sufficiently quickly that it could be used as a component in a tracking system. The original version is published in [118] and [121]. The feature detector was then extended to make it both faster and more reliable. This along with experimental validation is published in [119]. The resulting detector is both exceptionally fast (requiring under 7% of the available time to process a video stream on a modern workstation—5 times faster than then next best detector tested) and very repeatable compared to the other tested detectors.

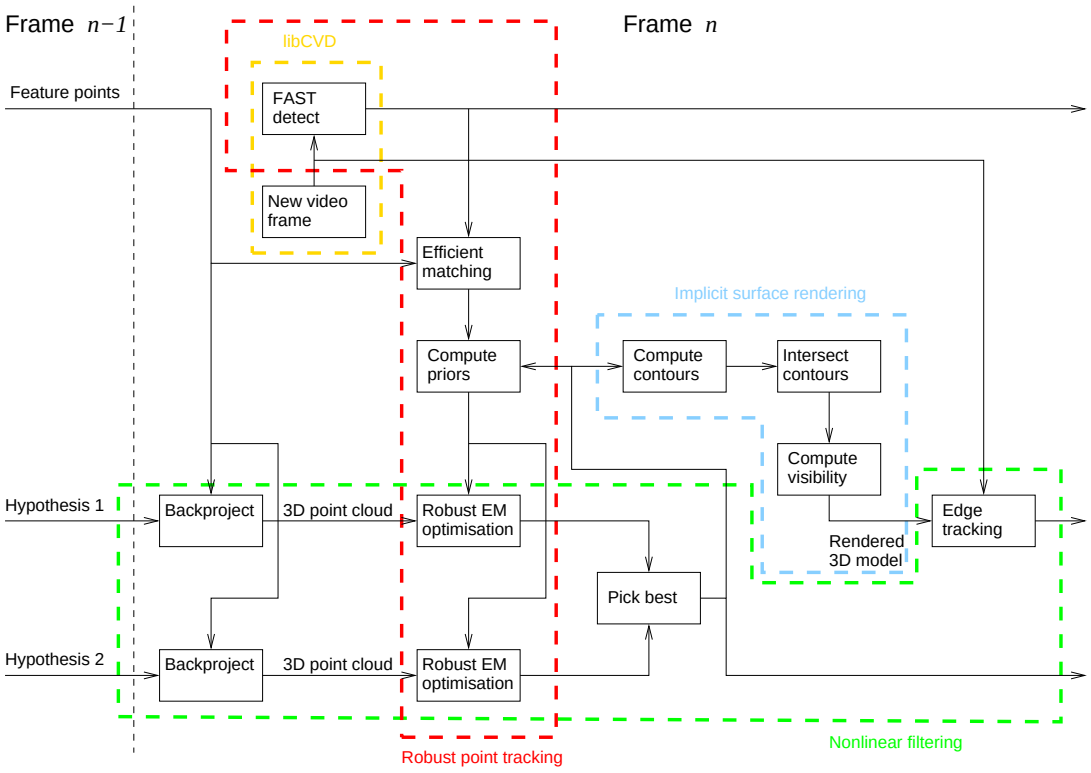


Figure 6.1: Block diagram of the complete tracking system. Subsystems containing multiple blocks are indicated inside the coloured dashed regions.

Implicit surface renderer. The edge based tracker for implicit surfaces was built using a new rendering system for arbitrary implicit surfaces. The renderer used a differential equation to trace out the apparent contour and determined occlusions by reasoning about contour intersections. These contributions are published in [117].

Fast polygon algorithms. A suite of fast polygon intersection algorithms was developed in order to improve efficiency of the implicit surface renderer. These algorithms perform polygon-polygon intersections, polygon-line intersections and polygon-point proximity detection in average $O(\log N)$ time. These are published in [117].

Nonlinear filter for combining trackers. Careful analysis of the tracking systems was used to justify a new nonlinear filter for combining the tracking results. Since either tracker can give an incorrect pose, the filter deferred evaluating the validity of the poses until more data arrived in the next frame. This resulted in

a tracking system with the robustness of the new point based tracking system, but with out drift. This is published in [118].

Software The FAST feature detector, along with various support facilities, such as video and image handling, is made available in the libCVD (Cambridge Vision Dynamics) library, of which the author is the maintainer. The library is available from [120].

6.1 Future work and open problems

Despite the robustness of the tracking system, there are still a considerable number of circumstances under which it can fail, such as when there are too few features in the view to constrain tracking adequately. While it is likely that further research can improve the robustness of the system (such as providing robustness to motion blur), one can always construct situations where tracking will be lost. Therefore, one area of future work would be to consider systems which could localise the object being tracked to allow tracking to continue.

Currently many localisation systems work by matching detected point features in to a static database. An interesting avenue for research would be to create a continually updated database of points for localisation from the 3D point clouds acquired each frame. For rapid localisation this would require that the database of point features can be indexed both spatially and by appearance. Furthermore, since the change in viewpoints between features in the database and features in the image could be quite large, it wold be useful to add computation of canonical scale to the FAST feature detector. This would allow it to be used for extraction of feature descriptors such as SIFT.

Another open question is that of the scalability of the tracking system. Both the point based tracker and the line based tracking system require rendering of the model (finding the visible edges, or the depth of arbitrary points in the image) for tracking, but are otherwise independent of the size of the model. Consequently, the scalability of the system depends on the scalability of the rendering engine, and modern rendering engines are capable of rapid rendering in vast environments. However, this makes the system scalable by putting the responsibility of making it scalable elsewhere. In practise, although the system could work with large models, acquiring large models is non trivial.

As stated in Section 1, models are built by measuring the object and then constructing a model. This process is both time consuming and error prone. In some cases where the object has been manufactured from a CAD (Computer Aided Design) model, this model could be used directly for tracking with little or no additional effort. However, few objects have these models readily available. Alternatively, surveying equipment (such as a total station) can be used to rapidly acquire accurate 3D coordinates of surveyed points. Capturing vertices during the surveying would allow the required 3D surface models to be constructed. With the aid of GPS equipped total stations, very large models can be surveyed.

Even with the addition of sufficiently advanced tools, considerable user input is required to create a model for a working tracking system. An appealing alternative is to use computer vision to generate a model. This could be done either with an off-line SFM (structure from motion) system or an on line SLAM (simultaneous localisation and mapping) system (sometimes known as 'causal SFM'). It should be noted that a purely visual system will suffer from drift. However, if absolute position is not required, then some systems can produce *consistent* models when loop closing occurs. If large scale models with absolute position are required, then an external measurement system such as GPS could be employed to remove drift.

A system such as this would allow new objects to be tracked easily, and in the case of a SLAM system, it would allow tracking to continue once the original model had ceased to be in the view. However, the system presented in this thesis requires a 3D surface model. The nature of the model allows the system to compute which parts of the object are occluded, as well as computing the depths of given points (which is essential to the robustness of the system). Computing this information is still very much an open problem and considerable further research is required.

A. Mean bounds SSD

Consider a column vector $\mathbf{x}$ in $\mathbb{R}^M$. The Euclidean length, l can be computed as:

$$l^2 = \|\mathbf{x}\|_2^2 = \mathbf{x}^\mathsf{T}\mathbf{x} = \sum_{i=1}^{M} x_i^2. \tag{A.1}$$

Since the elements of $\mathbf{x}$ are real, a single element puts a lower bound on l^2:

$$x_k^2 \leq \sum_{i=1}^{M} x_i^2. \tag{A.2}$$

If $\mathbf{x}$ is multiplied by an orthogonal matrix, $\mathbf{R}$, the length is unchanged:

$$(\mathbf{R}\mathbf{x})^\mathsf{T}\mathbf{R}\mathbf{x} = \mathbf{x}^\mathsf{T}\mathbf{R}^\mathsf{T}\mathbf{R}\mathbf{x} = \mathbf{x}^\mathsf{T}\mathbf{x} = l^2. \tag{A.3}$$

$\mathbf{R}$ can be constructed so that row k is $\left[\frac{1}{\sqrt{M}},\ldots,\frac{1}{\sqrt{M}}\right]$. Using this, Equation A.2 and Equation A.2 gives:

$$\mathbf{x}^\mathsf{T}\left[\tfrac{1}{\sqrt{M}},\ldots,\tfrac{1}{\sqrt{M}}\right]^\mathsf{T}\left[\tfrac{1}{\sqrt{M}},\ldots,\tfrac{1}{\sqrt{M}}\right]\mathbf{x} \leq l^2 \tag{A.4}$$

$$\left(\sum_{i=1}^{M} \frac{x_i}{\sqrt{M}}\right)^2 \leq l^2 \tag{A.5}$$

$$M\overline{x} \leq l^2, \tag{A.6}$$

where $\overline{x}$ is the mean of the elements of $\mathbf{x}$. Now consider $\mathbf{x} = \mathbf{v}_1 - \mathbf{v}_2$, i.e. l^2 is the SSD between $\mathbf{v}_1$ and $\mathbf{v}_2$. From Equation A.6 the length times SSD between the means is a lower bound on the SSD:

$$M(\overline{v_1} - \overline{v_2})^2 \leq \|\mathbf{v}_1 - \mathbf{v}_2\|_2^2. \tag{A.7}$$

B. Harris matrix and Cross Correlation

The Harris matrix, **H**, (see Equation 3.6) is derived by taking an image patch over the area (u, v), shifting it by (x, y) and computing the SSD. The local score, S is:

$$\begin{aligned} S &= \sum_u \sum_v \left(f(u,v) - f(u-x, v-y)\right)^2 \\ &= \sum_u \sum_v f(u,v)^2 - 2\underbrace{f(u,v)\, f(u-x, v-y)}_{X} + f(u-x, v-y)^2 \,, \end{aligned} \tag{B.1}$$

where $f(u, v)$ is the intensity of the image at pixel (u, v). The summation over the term labelled X is cross correlation. The Harris matrix is derived by taking the Hessian of S with respect to x and y around $x = y = 0$. It is often claimed that Hessian of S is equivalent to the Hessian of X, up to scale. This is not the case. Taking $S = \sum_u \sum_v 2s$, so that (from Equation B.1):

$$s = \frac{1}{2} f(u,v)^2 - f(u,v)\, f(u-x, v-y) + \frac{1}{2} f(u-x, v-y)^2 \,, \tag{B.2}$$

and differentiating this with respect to x and y gives:

$$\nabla s = -f(u,v)\, \nabla f(u-x, v-y) + f(u-x, v-y)\, \nabla f(u-x, v-y) \,. \tag{B.3}$$

Therefore, differentiating again to get the Hessian, $\mathcal{H}[s]$, gives:

$$\begin{aligned} \mathcal{H}[s] = \; & -f(u,v)\, \mathcal{H}[f(u-x, v-y)] + \nabla f(u-x, v-y)^\mathsf{T} \nabla f(u-x, v-y) \\ & + f(u-x, v-y)\, \mathcal{H}[f(u-x, v-y)] \,. \end{aligned} \tag{B.4}$$

The Harris matrix is given by $\mathcal{H}[S]\,|_{x=y=0}$, which is the well known result[52]:

$$\mathbf{H} = \sum_u \sum_v \nabla f(u,v)^\mathsf{T} \nabla f(u,v) = \begin{bmatrix} \widehat{\left(\frac{\partial f}{\partial x}\right)^2} & \widehat{\left(\frac{\partial f}{\partial x}\frac{\partial f}{\partial y}\right)} \\ \widehat{\left(\frac{\partial f}{\partial x}\frac{\partial f}{\partial y}\right)} & \widehat{\left(\frac{\partial f}{\partial y}\right)^2} \end{bmatrix} . \tag{B.5}$$

Instead, $\mathcal{H}[X]\,|_{x=y=0}$ is given by:

$$\mathbf{H}_X = \sum_u \sum_v f(u,v)\,\mathcal{H}[f(u,v)]\,. \tag{B.6}$$

Clearly, $\mathbf{H}$ and $\mathbf{H}_X$ are different. Consequently, the same feature detection algorithms can not be used. Algorithms based on $\mathbf{H}$ typically find features where the two eigenvalues, λ_1 and λ_2 are large. In particular, the Shi and Tomasi[130] detector defines the score to be:

$$\min \lambda_1, \lambda_2,$$

and the Harris[52] detector defines the score to be an approximation of this. Applying the same algorithms to $\mathbf{H}_X$ will not work, since unlike $\mathbf{H}$, it is not positive semi-definite. For instance, good corner points may occur where the eigenvalues are both large in magnitude and negative.

C. Proof that $\nabla \cdot \left(\mathbf{H}(\mathbf{x})\left(\mathbf{x}-\mathbf{c}\right) \times \nabla f(\mathbf{x})\right) = 0$

The definition for $\mathbf{x}'$ is derived by looking at constraints on the apparent contour. Nevertheless, $\mathbf{x}'$ is defined for any $\mathbf{x}$, so it is a vector field in $\mathbb{R}^3$. The definition for $\mathbf{x}'$ (up to scale) is:

$$\mathbf{x}' = \left(\mathcal{H}[f(\mathbf{x})]\,\mathbf{x}\right) \times g_i(\mathbf{x}) \tag{C.1}$$

Taking the divergence of $\mathbf{x}'$ and using the vector identity:

$$\nabla \cdot (\mathbf{a} \times \mathbf{b}) = \mathbf{b} \cdot \nabla \times \mathbf{a} - \mathbf{a} \cdot \nabla \times \mathbf{b}, \tag{C.2}$$

gives:

$$\nabla \cdot (\mathbf{H}\mathbf{x}) \times \nabla f = \nabla f \cdot \nabla \times \mathbf{H}\mathbf{x} - \mathbf{H}\mathbf{x} \cdot \nabla \times \nabla f, \tag{C.3}$$

where $\mathbf{H} = \mathcal{H}[f(\mathbf{x})]$ and $\nabla f = g_i(\mathbf{x})$. In general, $\nabla \times \nabla \mathbf{a} = 0$, so Equation C.3 becomes

$$\nabla \cdot \mathbf{x}' = \nabla f \cdot \nabla \times \mathbf{H}\mathbf{x}. \tag{C.4}$$

Expanding the second term of the dot product and writing $\mathbf{x} = \begin{bmatrix} x & y & z \end{bmatrix}^\mathsf{T}$ gives:

$$\begin{aligned}
\nabla \times \mathbf{Hx} &= \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \\ \frac{\partial}{\partial z} \end{bmatrix} \times \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial xy} & \frac{\partial^2 f}{\partial xz} \\ \frac{\partial^2 f}{\partial xy} & \frac{\partial^2 f}{\partial y^2} & \frac{\partial^2 f}{\partial yz} \\ \frac{\partial^2 f}{\partial xz} & \frac{\partial^2 f}{\partial yz} & \frac{\partial^2 f}{\partial z^2} \end{bmatrix} \begin{bmatrix} x \\ y \\ x \end{bmatrix} \\
&= \begin{bmatrix} \frac{\partial}{\partial y}\left(x\frac{\partial^2 f}{\partial xz} + y\frac{\partial^2 f}{\partial yz} + z\frac{\partial^2 f}{\partial z^2}\right) - \frac{\partial}{\partial z}\left(x\frac{\partial^2 f}{\partial xy} + y\frac{\partial^2 f}{\partial y^2} + z\frac{\partial^2 f}{\partial yz}\right) \\ \frac{\partial}{\partial z}\left(x\frac{\partial^2 f}{\partial x^2} + y\frac{\partial^2 f}{\partial xy} + z\frac{\partial^2 f}{\partial xz}\right) - \frac{\partial}{\partial x}\left(x\frac{\partial^2 f}{\partial xz} + y\frac{\partial^2 f}{\partial yz} + z\frac{\partial^2 f}{\partial z^2}\right) \\ \frac{\partial}{\partial x}\left(x\frac{\partial^2 f}{\partial xy} + y\frac{\partial^2 f}{\partial y^2} + z\frac{\partial^2 f}{\partial yz}\right) - \frac{\partial}{\partial y}\left(x\frac{\partial^2 f}{\partial x^2} + y\frac{\partial^2 f}{\partial xy} + z\frac{\partial^2 f}{\partial xz}\right) \end{bmatrix} \\
&= \begin{bmatrix} x\frac{\partial^3 f}{\partial xyz} + \frac{\partial^2 f}{\partial yz} + y\frac{\partial^3 f}{\partial y^2 z} + z\frac{\partial^3 f}{\partial yz^2} - x\frac{\partial^3 f}{\partial xyz} - y\frac{\partial^3 f}{\partial y^2 z} - \frac{\partial^2 f}{\partial yz} - z\frac{\partial^3 f}{\partial yz^2} \\ x\frac{\partial^3 f}{\partial x^2 z} + y\frac{\partial^3 f}{\partial xyz} + \frac{\partial^2 f}{\partial xz} + z\frac{\partial^3 f}{\partial xz^2} - \frac{\partial^2 f}{\partial xz} - x\frac{\partial^3 f}{\partial x^2 z} - y\frac{\partial^3 f}{\partial xyz} - z\frac{\partial^3 f}{\partial xz^2} \\ \frac{\partial^2 f}{\partial xy} + x\frac{\partial^3 f}{\partial x^2 y} + y\frac{\partial^3 f}{\partial xy^2} + z\frac{\partial^3 f}{\partial xyz} - x\frac{\partial^3 f}{\partial x^2 y} - \frac{\partial^2 f}{\partial xy} - y\frac{\partial^3 f}{\partial xy^2} - z\frac{\partial^3 f}{\partial xyz} \end{bmatrix} \\
&= \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}.
\end{aligned} \tag{C.5}$$

Substituting this result into Equation C.4 produces the result:

$$\nabla \cdot \mathbf{x}' = 0 \tag{C.6}$$

D. Lamp parameters

Using Equation 4.16, the parameters for the lamp (with annotations) are given below.

i	β	μ	$\mathbf{C}$
	Three long thin blobs make a roughly cylindrical body for the stem.		
1	1	$\begin{bmatrix} 0.25 \\ 0 \\ 2 \end{bmatrix}$	$\begin{bmatrix} 190 & 0 & 0 \\ 0 & 190 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
2	1	$\begin{bmatrix} 0.25 \\ 0 \\ 2.5 \end{bmatrix}$	$\begin{bmatrix} 190 & 0 & 0 \\ 0 & 190 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
3	1	$\begin{bmatrix} 0.25 \\ 0 \\ 3 \end{bmatrix}$	$\begin{bmatrix} 190 & 0 & 0 \\ 0 & 190 & 0 \\ 0 & 0 & 2 \end{bmatrix}$
	This cuts off the stem at the bottom to stop it being very long and pointed.		
4	-10000	$\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 10 \end{bmatrix}$

i	β	μ	$\mathbf{C}$
	Main 'hemisphere' at the front of the lamp.		
5	300	$\begin{bmatrix}1.6\\0\\4\end{bmatrix}$	$\begin{bmatrix}6 & 0 & 0\\0 & 4.5 & 0\\0 & 0 & 4.5\end{bmatrix}$
	Elongate the hemisphere slightly.		
6	300	$\begin{bmatrix}1.8\\0\\4\end{bmatrix}$	$\begin{bmatrix}30 & 0 & 0\\0 & 6 & 0\\0 & 0 & 6\end{bmatrix}$
	Sharply cut off the previous blobs to make a hemisphere. This misses the end.		
7	-800	$\begin{bmatrix}2.2\\0\\4\end{bmatrix}$	$\begin{bmatrix}100 & 0 & 0\\0 & .01 & 0\\0 & 0 & .01\end{bmatrix}$
	Remove the end of the blob.		
8	-300	$\begin{bmatrix}2.8\\0\\4\end{bmatrix}$	$\begin{bmatrix}30 & 0 & 0\\0 & .1 & 0\\0 & 0 & .1\end{bmatrix}$
	Create a cone at the rear of the lamp by stacking blobs.		
9	3	$\begin{bmatrix}1\\0\\4\end{bmatrix}$	$\begin{bmatrix}05 & 0 & 0\\0 & 05 & 0\\0 & 0 & 05\end{bmatrix}$
10	3	$\begin{bmatrix}0.5\\0\\4\end{bmatrix}$	$\begin{bmatrix}06 & 0 & 0\\0 & 06 & 0\\0 & 0 & 06\end{bmatrix}$
11	3	$\begin{bmatrix}0\\0\\4\end{bmatrix}$	$\begin{bmatrix}08 & 0 & 0\\0 & 08 & 0\\0 & 0 & 08\end{bmatrix}$
12	3	$\begin{bmatrix}-.5\\0\\4\end{bmatrix}$	$\begin{bmatrix}11 & 0 & 0\\0 & 11 & 0\\0 & 0 & 11\end{bmatrix}$
	Sharply cut off the last blob.		
13	-3000	$\begin{bmatrix}-0.75\\0\\4\end{bmatrix}$	$\begin{bmatrix}200 & 0 & 0\\0 & .01 & 0\\0 & 0 & .01\end{bmatrix}$

Bibliography

[1] Y. Amit and D. Geman. Shape quantization and recognition with randomized trees. *Neural Computation*, **9**(7):1545–1588, 1997. Cited on page 26

[2] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 3rd edition, 1999. Cited on page 15

[3] M. Armstrong and A. Zisserman. Robust object tracking. *Asian Conference on Computer Vision*, volume 1, 58–61. Singapore, 1995. Cited on pages 14 and 79

[4] S. Arya. *Nearest neighbor searching and applications*. Ph.D. thesis, College Park, MD, USA, 1995. Cited on page 25

[5] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching. *Journal of the ACM*, **45**:891–923, 1998. Cited on page 25

[6] S. Baker and I. Matthews. Lucas-kanade 20 years on: A unifying framework. *International Journal of Computer Vision*, 221–255, 2004. Cited on page 21

[7] J. S. Beis and D. G. Lowe. Shape indexing using approximate nearest-neighbour search in high-dimensional spaces. *11^{th} IEEE Conference on Computer Vision and Pattern Recognition*. Springer, 1997. Cited on page 25

[8] S. Belongie, J. Malik and J. Puzicha. Shape context: A new descriptor for shape matching and object recognition. *14^{th} Neural and Information Processing Systems Conference*, 831–837. Denver, Colorado, USA, 2000. Cited on page 24

[9] S. Belongie, J. Malik and J. Puzicha. Shape matching and object recognition using shape contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **24**(4):509–522, 2002. Cited on page 24

[10] M. Black and A. Jepson. Eigen-tracking: Robust matching and tracking of articulated objects using a view-based representation. *International Journal of Computer Vision*, **36**(2):63–84, 1998. Cited on page 21

[11] L. S. Blackford, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, M. Heroux, L. Kaufman, A. Lumsdaine, A. Petitet, R. Pozo, K. Remington and R. C. Whaley. An updated set of basic linear algebra subprograms (BLAS). *ACM Transactions Mathematical Software*, **28**(2):135–151, 2002. Cited on page 15

[12] J. F. Blinn. A generalization of algebraic surface drawing. *ACM Transcations on Graphics*, **1**(3):235–256, 1982. Cited on page 83

[13] M. Brown and D. G. Lowe. Invariant features from interest point groups. *13th British Machine Vision Conference*, 656–665. British Machine Vision Assosciation, Cardiff, 2002. Cited on page 51

[14] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **8**(6):679–698, 1986. Cited on page 76

[15] M. L. Cascia and S. Sclaroff. Fast, reliable head tracking under varying illumination: An approach based on registration of texture-mapped 3d models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **22**(4):322–336, 2002. Cited on page 21

[16] J. R. Cash and A. H. Karp. A variable order runge-kutta method for initial value problems with rapidly varying right-hand sides. *ACM Transactions on Mathematical Software*, **16**(3):201–222, 1990. Cited on pages 10, 93, and 94

[17] T. Cham and J. Rehg. A multiple hypothesis approach to figure tracking. *13th IEEE Conference on Computer Vision and Pattern Recognition*, 239–245. Springer, 1999. Cited on page 107

[18] G. K. M. Cheung, T. Kanade, J.-Y. Bouguet and M. Holler. A real time system for robust 3d voxel reconstruction of human motions. *14th IEEE Conference on Computer Vision and Pattern Recognition*, 2714–2720. Springer, 2000. Cited on page 82

[19] O. Chum and J. Matas. Matching with PROSAC - progressive sample consensus. *18th IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, 220–226. Springer, 2005. Cited on page 38

[20] R. Cipolla, K. Åström and P. Giblin. Motion form the frontier of curves surfaces. *5th IEEE International Conference on Computer Vision*, 269–275. Springer, Boston MA, USA, 1995. Cited on page 82

[21] R. Cipolla and A. Blake. Surface shape from the deformations of apparent contours. *International Journal of Computer Vision*, **9**(2):83–112, 1992. Cited on page 81

[22] R. Cipolla, G. J. Fletcher and P. J. Giblin. Surface geometry from cusps of apparent contours. *5th IEEE International Conference on Computer Vision*, 858–863. Springer, Boston MA, USA, 1995. Cited on page 81

[23] R. Cipolla and P. Giblin. *Visual Motion of Curves and Surfaces*. Cambridge University Press, 2000. Cited on pages 81, 82, 84, and 86

[24] D. Claus and A. Fitzgibbon. Reliable fiducial detection in natural scenes. *Proceedings of the 8th European Conference on Computer Vision, Prague, Czech Republic*, volume 3024, 469–480. Springer-Verlag, 2004. Cited on pages 46 and 52

[25] J. Cooper, S. Venkatesh and L. Kitchen. Early jump-out corner detectors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **15**(8):823–828, 1993. Cited on pages 46 and 47

[26] T. Darrell, G. Gordon, M. Harville and J. Woodfill. Integrated person tracking using stereo, color, and pattern detection. *International Journal of Computer Vision*, **37**(2):175–185, 2000. Cited on page 105

[27] A. J. Davison. Real-time simultaneous localisation and mapping with a single camera. *9th IEEE International Conference on Computer Vision*. Springer, Nice, France, 2003. Cited on page 21

[28] Q. Delamarre and O. D. Faugeras. 3d articulated models and multi-view tracking with silhouettes. *7th IEEE International Conference on Computer Vision*, volume 2, 716–721. Springer, Kerkyra, Corfu, Greece, 1999. Cited on page 82

[29] A. Dempster, N. Laird and R. D.B. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, **B 39**:1–38, 1977. Cited on page 35

[30] P. Deuflhard. Recent progress in extrapolation methods for ordinary differential equations. *SIAM Review*, **27**(4):505–535, 1985. Cited on page 93

[31] J. Deutscher, A. Blake and I. Reid. Articulated body motion capture by annealed particle filtering. *14th IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, 126–133. Springer, 2000. Cited on page 107

[32] P. Dias, A. Kassim and V. Srinivasan. A neural network based corner detection method. *IEEE International Conference on Neural Networks*, volume 4, 2116–2120. Perth, WA, Australia, 1995. Cited on pages 46 and 52

[33] T. Drummond. TooN: Tom's object-oriented numerics library, Accessed 2005. `http://savannah.nongnu.org/projects/toon`. Cited on page 15

[34] T. Drummond and R. Cipolla. Real-time tracking of highly articulated structures in the presence of noisy measurements. *8th IEEE International Conference on Computer Vision*, 315–320. Springer, Vancouver, Canada, 2001. Cited on pages 82 and 83

[35] T. Drummond and R. Cipolla. Real-time visual tracking of complex structures. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **24**(7):932–946, 2002. Cited on pages 13, 16, 77, 84, and 100

[36] M. A. Fischler and R. C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communcations of the ACM*, **24**(6):381–395, 1981. Cited on pages 38 and 79

[37] W. T. Freeman and E. H. Adelson. The design and use of steerable filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **13**(9):891–906, 1991. Cited on page 24

[38] V. Gaede and O. Günther. Multidimensional access methods. *ACM Computing Surveys*, **30**(2):170–231, 1998. Cited on page 25

[39] D. Gavrila and L. Davis. 3d model-based tracking of humans in action: A multi-view approach. *10th IEEE Conference on Computer Vision and Pattern Recognition*. Springer, 1996. Cited on page 82

[40] D. Gennery. Visual tracking of known three-dimensional objects. *International Journal of Computer Vision*, **7**(1):243–270, 1992. Cited on page 17

[41] P. J. Giblin, J. E. Rycrofy and F. E. Pollick. Moving surfaces. R. B. Fisher, ed., *Design and Application of Curves and Surfaces*, number 5 in Mathematics of Surfaces. Clarendon Press, 1994. Cited on page 84

[42] F. Girosi, M. Jones and T. Poggio. Priors stabilizers and basis functions: From regularization to radial, tensor and additive splines. Technical Report AIM-1430, MIT Computer Science and Artificial Intelligence Laboratory, 1993. Cited on page 82

[43] L. van Gool, T. Moons and D. Ungureanu. Affine/photometric invariants for planar intensity patterns. *4^{th} Euproean Conference on Computer Vision*, volume 1, 642–651. Springer, 1996. Cited on page 24

[44] I. Gordon and D. G. Lowe. Scene modelling, recognition and tracking with invariant image features. *3^{rd} IEEE and ACM International Symposium on Mixed and Augmented Reality*, 110–119. IEEE Computer Society, Jacksonville, USA, 2004. Cited on pages 15 and 22

[45] N. Gordon, D. Salmond and A. Smith. Novel approach to nonlinear/non-gaussian bayesian state estimation. *IEE-Proceedings-F*, 140, 107–113. 1993. Cited on page 107

[46] V. Gouet and B. Lameyre. Sap: A robust approach to track objects in video streams with snakes and points. *15^{th} British Machine Vision Conference*, volume 2, 737–746. British Machine Vision Assosciation, Kingston Upon Thames, 2004. Cited on pages 15, 26, and 105

[47] A. Guiducci. Corner characterization by differential geometry techniques. *Pattern Recognition Letters*, **8**(5):311–318, 1988. Cited on pages 46 and 51

[48] G. D. Hager and P. N. Belhumeur. Efficient region tracking with parametric models of geometry and illumination. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **20**(10):1025–1039, 1998. Cited on page 21

[49] Y. W. Haitao Wang, Stan Z Li. Face recognition under varying lighting conditions using self quotient image. *6^{th} IEEE International Conference on Automatic Face and Gesture Recognition*, 819. 2004. Cited on page 84

[50] R. M. Haralick and L. G. Shapiro. *Computer and robot vision*, volume 1. Adison-Wesley, 1993. Cited on pages 46 and 47

[51] C. Harris and C. Stennett. RAPID, a video rate object tracker. *1^{st} British Machine Vision Conference*, 73–77. British Machine Vision Association, Oxford, 1990. Cited on pages 13 and 76

[52] C. Harris and M. Stephens. A combined corner and edge detector. *Alvey Vision Conference*, 147–151. 1988. Cited on pages 46, 49, 68, 72, 124, and 125

[53] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2nd edition, 2004. Cited on pages 16 and 22

[54] P. J. Huber. Robust estimation of a location parameter. *Annals of Mathematical Statistics*, 73–101, 1964. Cited on page 77

[55] P. J. Huber. *Robust Statistics*. Wiley, 1981. Cited on page 77

[56] M. Isard and A. Blake. Condensation–conditional density propagation for visual tracking. *International Journal of Computer Vision*, **29**(8):5–28, 1998. Cited on page 106

[57] ISO/IEC 10918-1:1994 Digital compression and coding of continuous-tone still images: Requirements and guidelines. Technical comittee: JTC 1/SC 29, 1994. Cited on page 28

[58] O. R. James L. Crowley. Fast computation of characteristic scale using a half octave pyramid. *Scale Space 03: 4th International Conference on Scale-Space theories in Computer Vision*. Isle of Skye, Scotland, UK, 2003. Cited on page 50

[59] A. Johnson and M. Hebert. Using spin images for efficient object recognition in cluttered 3d scenes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **21**(5):433–449, 1999. Cited on page 25

[60] A. E. Johnson and M. Hebert. Surface matching for object recognition in complex 3-d scenes. *Image and Vision Computing*, **16**:635–651, 1998. Cited on page 25

[61] S. J. Julier and J. K. Uhlmann. A new extension of the kalman filter to nonlinear systems. *The Proceedings of AeroSense: The 11th International Symposium on Aerospace/Defense Sensing, Simulation and Controls*, 1628–1632. SPIE, Orlando, Florida, USA, 1997. Cited on page 104

[62] F. Jurie and M. Dhome. A simple and efficient template matching algorithm. *8th IEEE International Conference on Computer Vision*, 544–549. Springer, Vancouver, Canada, 2001. Cited on page 21

[63] F. Jurie and M. Dhome. Hyperplane approximation for template matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **24**(7), 2002. Cited on page 21

[64] R. E. Kalman. A new approach to linear filtering and prediction problems. *ASME Journal of Basic Engineering*, **82**:35–45, 1960. Cited on pages 77 and 104

[65] H. Kato and M. Billinghurst. Marker tracking and HMD calibration for a video-based augmented reality conferenencing system. *2^{nd} IEEE International Workshop on Augmented Reality*, 85–94. IEEE CS, San Francisco, CA, USA, 1999. Cited on page 12

[66] Y. Ke and R. Sukthankar. Pca-sift: A more distinctive representation for local image descriptors. *17^{th} IEEE Conference on Computer Vision and Pattern Recognition*, 506–513. Springer, 2004. Cited on page 24

[67] C. Kemp and T. Drummond. Multimodal tracking using texture changes. *15^{th} British Machine Vision Conference*. British Machine Vision Assosciation, Kingston Upon Thames, 2004. Cited on pages 14, 79, and 82

[68] C. Kemp and T. Drummond. Dynamic measurement clustering to aid real time tracking. *10^{th} IEEE International Conference on Computer Vision*, volume 2, 1500–1507. Springer, Beijing, China, 2005. Cited on pages 14 and 80

[69] C. S. Kenney, B. S. Manjunath, M. Zuliani, M. G. A. Hewer and A. V. Nevel. A condition number for point matching with application to registration and postregistration error estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **25**(11):1437–1454, 2003. Cited on pages 46 and 49

[70] S. Kirkpatrick, C. Gelatt and M. Vecchi. Optimization by simulated annealing. *Science*, **220, 4598**(4598):671–680, 1983. Cited on page 34

[71] L. Kitchen and A. Rosenfeld. Gray-level corner detection. *Pattern Recognition Letters*, **1**(2):95–102, 1982. Cited on pages 46 and 47

[72] G. Klein and T. Drummond. Tightly integrated sensor fusion for robust visual tracking. *Image and Vision Computing*, **22**(10):769–776, 2004. Cited on pages 78 and 111

[73] G. Klein and T. Drummond. A single frame visual gyroscope. *16th British Machine Vision Conference*. British Machine Vision Assosciation, Oxford, 2005. Cited on page 78

[74] J. J. Kœnderink. *Solid Shape*. MIT Press, Cambridge, Massachusetts, USA, 1990. Cited on page 84

[75] J. J. Kœnderink and A. J. van Doorn. The singularities of the visual mapping. *Biological Cybernetics*, **24**:51–59, 1976. Cited on page 88

[76] J. J. Kœnderink and A. J. van Doorn. The shape of smooth objects and the way contours end. *Perception*, **11**:129–137, 1982. Cited on page 88

[77] J. J. Kœnderink and A. J. van Doorn. Representation of local geometry in the visual system. *Biological Cybernetics*, **55**(6):367–375, 1987. Cited on page 23

[78] D. Koller, K. Daniilidisy and H.-H. Nagelyz. Model-based object tracking in monocular image sequences of road traffic scenes. *International Journal of Computer Vision*, **10**(3):257–281, 1993. Cited on page 80

[79] D. J. Langridge. Curve encoding and detection of discontinuities. *Computer Vision, Graphics and Image Processing*, **20**(1):58–71, 1987. Cited on pages 46 and 47

[80] S. Lazebnik, C. Schmid and J. Ponce. Sparse texture representation using affine-invariant neighborhoods. *16th IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, 319–324. Springer, 2003. Cited on page 25

[81] V. Lepetit and P. Fua. Monocular model-based 3d tracking of rigid objects: A survey. *Foundations and Trends in Computer Graphics and Vision*, **1**(1):1–89, 2005. Cited on page 80

[82] V. Lepetit, P. Lagger and P. Fua. Randomized trees for real-time keypoint recognition. *18th IEEE Conference on Computer Vision and Pattern Recognition*. Springer, 2005. Cited on page 26

[83] V. Lepetit, J. Pilet and P. Fua. Point matching as a classification problem for fast and robust object pose estimation. *17th IEEE Conference on Computer Vision and Pattern Recognition*. Springer, 2004. Cited on page 25

[84] H.-C. Liu and M. D. Srinath. Corner detection from chain code. *Pattern Recognition Letters*, **23**(1):51–68, 1990. Cited on pages 46 and 47

[85] W. E. Lorensen and H. E. Cline. Marching cubes: a high resolution 3d surface reconstruction algorithm. *Proceedings of SIGGRAPH*, volume 21, 163–169. 1987. Cited on page 82

[86] D. G. Lowe. Fitting parameterized 3-d models to images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **13**:441–450, 1991. Cited on pages 14 and 82

[87] D. G. Lowe. Robust model-based motion tracking through the integration of search and estimation. *International Journal of Computer Vision*, **8**(2):113–122, 1992. Cited on page 79

[88] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, **60**(2):91–110, 2004. Cited on pages 22, 23, 25, 32, 46, 50, and 68

[89] D. G. Lowe. Demo software: Sift keypoint detector. `http://www.cs.ubc.ca/~lowe/keypoints/`, Accessed 2005. Cited on page 70

[90] G. Loy and A. Zelinsky. A fast radial symmetry transform for detecting points of interest. *7th Euproean Conference on Computer Vision*, 358–368. Springer, 2002. Cited on pages 46 and 52

[91] B. D. Lucas and T. Kanade. An iterative image registration technique with application to stereo vision. *7th International Joint Conference on Artificial Intelligence*, 674–679. 1981. Cited on pages 21 and 23

[92] B. Luo, A. D. J. Cross and E. R. Hancock. Corner detection via topographic analysis of vector potential. *9th British Machine Vision Conference*. British Machine Vision Assosciation, Southampton, 1998. Cited on pages 46 and 50

[93] J. MacCormick and A. Blake. A probabilistic exclusion principle for tracking multiple objects. *7th IEEE International Conference on Computer Vision*, 572–578. Springer, Kerkyra, Corfu, Greece, 1999. Cited on page 107

[94] J. MacCormick and M. Isard. Partitioned sampling, articulated objects, and interface-quality hand tracking. *6th Euproean Conference on Computer Vision*, volume 2, 3–19. Springer, 2000. Cited on page 107

[95] E. Marchand, P. Bouthemy, F. Chaumette and V. Moreau. Robust real-time visual tracking using a 2D-3D model-based approach. *7th IEEE International Conference on Computer Vision*, volume 1, 262–268. Springer, Kerkyra, Corfu, Greece, 1999. Cited on pages 14 and 77

[96] D. Marr and E. Hildreth. Theory of edge detection. *Proceedings of the Royal Society of London B*, **207**:187–217, 1980. Cited on pages 76 and 79

[97] I. Matthews, T. Ishikawa and S. Baker. The template update problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **26**(6):810–815, 2004. Cited on pages 21 and 22

[98] G. Medioni and Y. Yasumoto. Corner detection and curve representation using cubic b-splines. *Computer Vision, Graphics and Image Processing*, **39**(3):279–290, 1987. Cited on pages 46 and 47

[99] J. Menon, B. Wyvill, C. Bajaj, J. Bloomenthal, B. Guo, J. Hart and G. Wyvill. An introduction to implicit techniques, siggraph course notes on implicit surfaces for geometric modelling and computer graphics, 1996. Cited on page 83

[100] R. van der Merwe, A. Doucet, N. de Freitas and E. Wan. The unscented particle filter. Technical report, Cambridge University Engineering Department, 2000. Cited on page 107

[101] K. Mikolajczyk and C. Schmid. Indexing based on scale invariant interest points. *8th IEEE International Conference on Computer Vision*, 525–531. Springer, Vancouver, Canada, 2001. Cited on pages 46, 51, 68, and 69

[102] K. Mikolajczyk and C. Schmid. An affine invariant interest point detector. *European Conference on Computer Vision*, 128–142. Springer, 2002. Copenhagen. Cited on pages 23, 46, 50, and 51

[103] K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **27**(10):1615–1630, 2005. Cited on pages 24 and 25

[104] F. Mohannah and F. Mokhtarian. Performenace evaluation of corner detection algorithms under affine and similarity transforms. T. F. Cootes and C. Taylor, eds., *12th British Machine Vision Conference*. British Machine Vision Assosciation, Manchester, 2001. Cited on page 52

[105] F. Mokhtarian and R. Suomela. Robust image corner detection through curvature scale space. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **20**(12):1376–1381, 1998. Cited on pages 46 and 47

[106] N. D. Molton, A. J. Davison and I. D. Reid. Locally planar patch features for real-time structure from motion. *15th British Machine Vision*

Conference. British Machine Vision Assosciation, Kingston Upon Thames, 2004. Cited on page 21

[107] H. Moravec. Obstacle avoidance and navigation in the real world by a seeing robot rover. *tech. report CMU-RI-TR-80-03, Robotics Institute, Carnegie Mellon University & doctoral dissertation, Stanford University.* Carnegie Mellon University, 1980. Available as Stanford AIM-340, CS-80-813 and republished as a Carnegie Mellon University Robotics Institue Technical Report to increase availability. Cited on pages 46 and 48

[108] D. R. Musser. Introspective sorting and selection algorithms. *Software Practice and Experience*, **27**(8):983, 1997. Cited on page 29

[109] A. Noble. Finding corners. *Image and Vision Computing*, **6**(2):121–128, 1988. Cited on pages 46, 49, and 50

[110] A. Noble. *Descriptions of image surfaces.* Ph.D. thesis, Department of Engineering Science, University of Oxford., 1989. Cited on pages 46 and 49

[111] M. S. Paterson and F. F. Yao. Efficient binary space partitions for hidden surface removal and solid modeling. *Discrete and Computational Geometry*, **5**(5):485–503, 1990. Cited on page 77

[112] K. Pearson. Principal components analysis. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, **6**(2):559, 1901. Cited on page 24

[113] R. Plänkers and P. Fua. Articulated soft objects for video-based body modeling. *8^{th} IEEE International Conference on Computer Vision.* Springer, Vancouver, Canada, 2001. Cited on page 83

[114] R. Plänkers and P. Fua. Model-based silhouette extraction for accurate people tracking. *7^{th} Euproean Conference on Computer Vision*, 325–339. Springer, 2002. Cited on page 83

[115] W. H. Press, S. A. Teukolsky, W. H. Vetterling and B. P. Flannery. *Numerical Recipes in C.* Cambridge University Press, 1999. Cited on pages 93 and 95

[116] J. R. Quinlan. Induction of decision trees. *Machine Learning*, **1**:81–106, 1986. Cited on page 60

[117] E. Rosten and T. Drummond. Rapid rendering of apparent contours of implicit surfaces for real-time tracking. *14^{th} British Machine Vision Conference*, volume 2, 719–728. British Machine Vision Assosciation, Norwhich, 2003. Cited on page 119

[118] E. Rosten and T. Drummond. Fusing points and lines for high performance tracking. *10^{th} IEEE International Conference on Computer Vision*, volume 2, 1508–1515. Springer, Beijing, China, 2005. Cited on pages 56, 118, and 120

[119] E. Rosten and T. Drummond. Machine learning for high speed corner detection. *9^{th} Euproean Conference on Computer Vision*, To appear. Springer, 2006. Cited on page 118

[120] E. Rosten, T. Drummond, E. Eade, G. Reitmayr, P. Smith, C. Kemp, G. Klein and T. Gan. libCVD: Cambridge vision dynamics library, Accessed 2005. `http://savannah.nongnu.org/projects/libcvd`. Cited on pages 16 and 120

[121] E. Rosten, G. Reitmayr and T. Drummond. Real-time video annotations for augmented reality. *International Symposium on Visual Computing*. 2005. Cited on pages 56 and 118

[122] W. S. Rutkowski and A. Rosenfeld. A comparison of corner detection techniques for chain coded curves. Technical Report 623, Maryland University, 1978. Cited on pages 46 and 47

[123] F. Schaffalitzky and A. Zisserman. Viewpoint invariant texture matching and wide baseline stereo. *8^{th} IEEE International Conference on Computer Vision*, 636–643. Springer, Vancouver, Canada, 2001. Cited on page 51

[124] F. Schaffalitzky and A. Zisserman. Multi-view matching for unordered image sets, or How do I organise my holiday snaps? *7^{th} Euproean Conference on Computer Vision*, 414–431. Springer, 2002. Cited on pages 24 and 51

[125] C. Schmid and R. Mohr. Local greyvalue invariants for image retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **19**(5):530–534, 1997. Cited on page 23

[126] C. Schmid, R. Mohr and C. Bauckhage. Comparing and evaluating interest points. *6^{th} IEEE International Conference on Computer Vision*, 230–235. Springer, Bombay, India, 1998. Cited on page 69

[127] C. Schmid, R. Mohr and C. Bauckhage. Evaluation of interest point detectors. *International Journal of Computer Vision*, **37**(2):151–172, 2000. Cited on pages 44, 53, and 66

[128] A. Shahrokni, T. Drummond and P. Fua. Texture boundary detection for real-time tracking. *8th Euproean Conference on Computer Vision*, volume 2, 566–577. Springer, 2004. Cited on page 79

[129] A. Shahrokni, T. Drummond and P. Fua. Fast texture-based tracking and delineation using texture entropy. *10th IEEE International Conference on Computer Vision*. Springer, Beijing, China, 2005. Cited on page 79

[130] J. Shi and C. Tomasi. Good features to track. *9th IEEE Conference on Computer Vision and Pattern Recognition*. Springer, 1994. Cited on pages 46, 49, 68, 69, and 125

[131] B. Sklar. *Digital Communications*. Prentice Hall, 1988. Cited on page 72

[132] S. M. Smith. `http://www.fmrib.ox.ac.uk/~steve/susan/susan2l.c`, Accessed 2005. Cited on pages 68 and 72

[133] S. M. Smith and J. M. Brady. SUSAN - a new approach to low level image processing. *International Journal of Computer Vision*, **23**(1):45–78, 1997. Cited on pages 46, 51, and 70

[134] K. Stark and T. Ihle. Visual tracking of solid objects based on an active contour model. *8th British Machine Vision Conference*. British Machine Vision Assosciation, Essex, 1997. Cited on page 83

[135] B. Stenger, P. R. S. Mendonça and R. Cipolla. Model-based 3d tracking of an articulated hand. *15th IEEE Conference on Computer Vision and Pattern Recognition*, 310–315. Springer, 2001. Cited on page 82

[136] J. Stoer and R. Bulirsch. *Introduction to Numerical Analysis (English Translation)*. Springer-Verlag, 1976. Cited on page 93

[137] G. Taylor and L. Kleeman. Fusion of multimodal visual cues for model-based object tracking. *Australian Conference on Robotics and Automation*. Brisbane, Austraila, 2003. Cited on pages 15 and 105

[138] P. Tissainayagam and D. Suter. Assessing the performance of corner detectors for point feature tracking applications. *Image and Vision Computing*, **22**(8):663–679, 2004. Cited on page 53

[139] B. Tordoff and D. W. Murray. Guided sampling and consensus for motion estimation. *7th Euproean Conference on Computer Vision*. Springer, 2002. Cited on pages 37, 38, and 79

[140] P. H. Torr and A. Zisserman. MLESAC: A new robust estimator with application to estimating image geometry. *Computer Vision and Image Understanding*, **78**:138–156, 2000. Cited on pages 37 and 79

[141] K. Toyama and G. D. Hager. Incremental focus of attention for robust vision-based tracking. *International Journal of Computer Vision*, **35**(1):45–63, 1999. Cited on page 105

[142] M. Trajkovic and M. Hedley. Fast corner detection. *Image and Vision Computing*, **16**(2):75–87, 1998. Cited on pages 46, 51, and 53

[143] J. Tukey. A survey of sampling from contaminated distributions. I. Olkin, ed., *Contributions to Probability and Statistics*, 448–485. Stanford University Press, 1960. Cited on page 77

[144] L. Vacchetti, V. Lepetit and P. Fua. Stable 3–d tracking in real-time using integrated context information. *16th IEEE Conference on Computer Vision and Pattern Recognition*. Springer, 2003. Cited on page 22

[145] L. Vacchetti, V. Lepetit and P. Fua. Stable real-time 3d tracking using online and offline information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **26**(10):1385–1391, 2004. Cited on pages 14 and 22

[146] L. Vacchetti and V. L. amd Pascal Fua. Combining edge and texture information for real-time accurate 3d camera tracking. *3rd IEEE and ACM International Symposium on Mixed and Augmented Reality*. IEEE Computer Society, Jacksonville, USA, 2004. Cited on pages 15 and 105

[147] V. Varadarajan. *Lie Groups, Lie Algebras and Their Representations*. Number 102 in Graduate Texts in Mathematics. Springer-Verlag, 1974. Cited on page 17

[148] G. Vegter and M. Szafraniec. Apparent contours of implicit surfaces. Technical Report ECG-TR-124102-03, ECG, 2002. Cited on pages 84 and 86

[149] P. A. Viola and M. J. Jones. Rapid object detection using a boosted cascade of simple features. *15th IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, 511–518. Springer, 2001. Cited on page 72

[150] H. Wang and M. Brady. Real-time corner detection algorithm for motion estimation. *Image and Vision Computing*, **13**(9):695–703, 1995. Cited on pages 46 and 48

[151] G. Wyvill, C. McPheeters and B. Wyvill. Data structure for soft objects. *The Visual Computer*, **2**(4):277–234, 1986. Cited on page 83

[152] Z. Zhang, R. Deriche, O. Faugeras and Q. Luong. A robust technique for matching two uncalibrated images through the recovery of the unknown epipolar geometry. *Artificial Intelligence*, **78**:87–119, 1995. Cited on page 25

[153] Z. Zheng, H. Wang and E. K. Teoh. Analysis of gray level corner detection. *Pattern Recognition Letters*, **20**(2):149–162, 1999. Cited on pages 46 and 49

[154] M. Zuliani, C. Kenney and B. Manjunath. A mathematical comparison of point detectors. *Second IEEE Image and Video Registration Workshop (IVR)*. Washington DC, USA, 2004. Cited on page 49