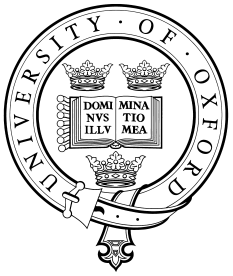
Tracking Non-Rigid Objects in Video

Aeron Buchanan
St Anne's College, 29th February, 2008



A thesis submitted for the degree of Doctor of Philosophy of the University of Oxford

Abstract

Video is a sequence of 2D images of the 3D world generated by a camera. As the camera moves relative to the real scene and elements of that scene themselves move, correlated frame-to-frame changes in the video images are induced. Humans easily identify such changes as scene motion and can readily assess attempts to quantify it. For a machine, the identification of the 2D frame-to-frame motion is difficult. This problem is addressed by the computer vision process of tracking.

Tracking underpins the solution to the problem of augmenting general video sequences with artificial imagery, a staple task in the visual effects industry. The problem is difficult because tracking in general video sequences is complicated by the presence of non-rigid motion, repeated texture and arbitrary occlusions. Existing methods provide solutions that rely on imposing limitations on the scenes that can be processed or that rely on human artistry and hard work. I introduce new paradigms, frameworks and algorithms for overcoming the challenges of processing general video and thus provide solutions that fill the gap between the 'automated' and 'manual' approaches. The work is easily sectioned into three parts, which can be considered separately or taken together for dealing with video without limitations.

The initial focus is on directly addressing practical issues of human interaction in the tracking process: a new solution is developed by explicitly incorporating the user into an interactive algorithm. It is a novel tracking system based on fast full-frame patch searching and high-speed optimal track determination. This approach makes only minimal assumptions about motion and appearance, making it suitable for the widest variety of input video. I detail an implementation of the new system using k-d trees and dynamic programming.

The second distinct contribution is an important extension to tracking algorithms in general. It can be noted that existing tracking algorithms occupy a spectrum in their use of global motion information. Local methods are easily confused by occlusions, repeated texture and image noise. Global motion models offer strong predictions to see through these difficulties and have been used in restricted circumstances, but are defeated by scenes containing independently moving objects or modest levels of non-rigid motion. I present a well principled way of combining local and global models to improve tracking, especially in these highly problematic cases. By viewing rank-constrained tracking as a probabilistic model of 2D tracks instead of 3D motion, I show how one can obtain a robust motion prior that can be easily incorporated in any existing tracking algorithm.

The development of the global motion prior is based on rank-constrained factorization of measurement matrices. A common difficulty comes from the frequent occurrence of occlusions in video, which means that the relevant matrices are often not complete due to missing data. This defeats standard factorization algorithms. To fully explain and understand the algorithmic complexities of factorization in this practical context, I present a common notation for the direct comparison of existing algorithms and propose a new family of hybrid approaches that combine the superb initial performance of alternation methods with the convergence power of the Newton algorithm.

Together, these investigations provide a wide-ranging, yet coherent exploration of tracking non-rigid objects in video.

Publications

A. M. Buchanan and A. W. Fitzgibbon, **"Damped Newton Algorithms for Matrix Factorization with Missing Data."** In Proceedings, IEEE International Conference on Computer Vision and Pattern Recognition (CVPR'05), Vol. 2, pages 316–322, San Diego, 2005.

A. M. Buchanan and A. W. Fitzgibbon, **"Interactive Feature Tracking using K-D Trees and Dynamic Programming."** In Proceedings, IEEE International Conference on Computer Vision and Pattern Recognition (CVPR'06), Vol. 1, pages 626–633, New York, 2006.

A. M. Buchanan and A. W. Fitzgibbon, **"Combining Local and Global Motion Models for Feature Point Tracking in General Scenes."** In Proceedings, IEEE International Conference on Computer Vision and Pattern Recognition (CVPR'07), Minneapolis, 2007.

Acknowledgements

I would very much like to thank my supervisor, Andrew Fitzgibbon, particularly for his excellent tuition and seemingly infinite patience. Thanks also go to all my family and friends, and especially the members of the Visual Geometry Group.

This work was funded by a three-year studentship from the Engineering and Physical Sciences Research Council (EPSRC).

Contents

Chapter 1

Introduction

As the ubiquity of video cameras grows, the study of the wealth of information embedded in image sequences becomes evermore pertinent. The cost of cameras is now low enough for a significant proportion of the population to have the capability to generate videos in their pocket. More overtly, the spread of CCTV cameras means that billions of hours of footage are generated every day. The growing conversion of film-makers from using celluloid film to employing digital cameras is a reflection of the fact that even professionally, the creation of moving pictures is becoming less expensive and more accessible. The proliferation of video generation is increasing the awareness of and demand for video analysis and manipulation. The reconstruction of the scene in front of the camera, the search for objects appearing in the video and the addition of computer generated imagery are, nowadays, common tasks performed on image sequences. Much of such video processing starts with obtaining knowledge of the observed movements of the objects and background in the video footage. The resulting motion information opens the door to a myriad of popular processing algorithms: it then becomes easy to augment the video with dinosaurs, floating candles and other computerized special effects. It becomes simple to stitch the individual frames together to create panoramic vistas. It even becomes possible to determine actions, interactions and sometimes the intents of the subjects of the video. Playing such a foundational role in computer vision,

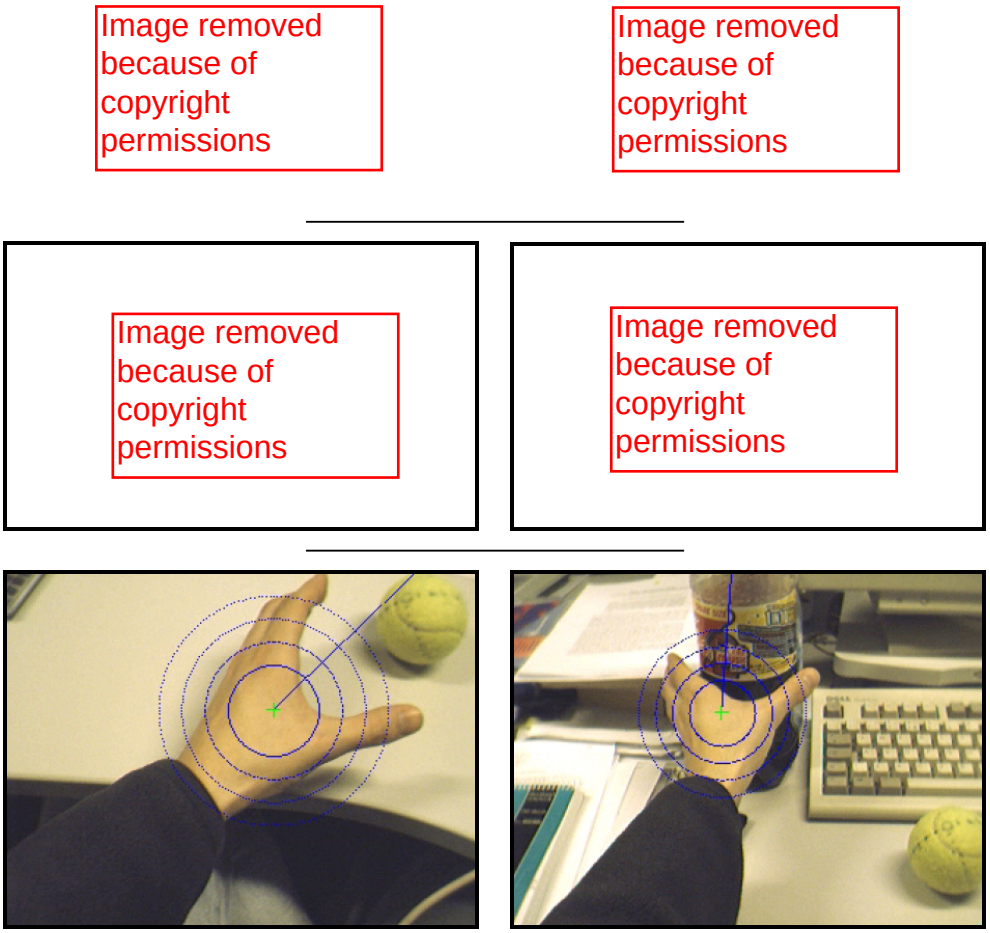


Figure 1.1: **Applications of motion tracking.** Top: panoramic stitching (images taken from the Panorama builder 7.0 software tutorial page, `www.panobuilder.com/learnmore_pb7.asp`). Middle: insertion of dinosaurs into live action footage (from the first and fourth 'Jurassic Park' movies). Bottom: recognition of grasping (Mayol et al. 2004).

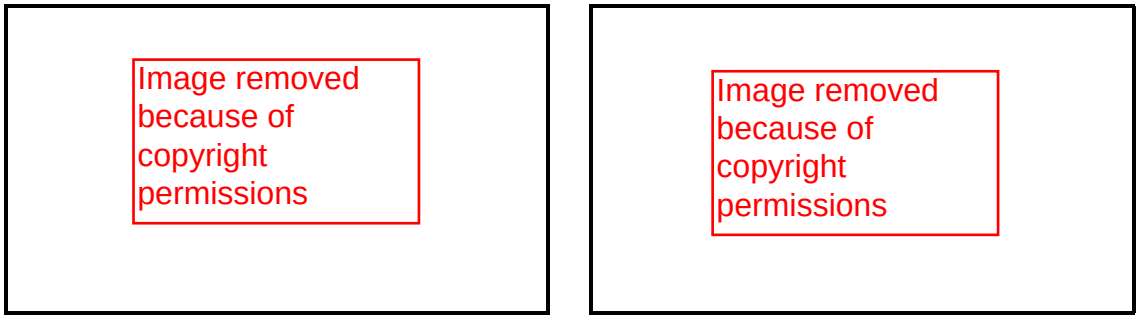
this *tracking* process is a extremely important and is the subject of this thesis.

Before explaining the thesis in any more detail, I would like to describe the context and motivation for the work a little further. It is based primarily on the feature film industry, which is close to my heart, formally professionally, and continuingly recreationally. To emphasize the importance and widespread appeal of film, let me present some statistics. For the decade ending 2006, total cinema admissions in the United Kingdom rose by 12.7% to 156.6 million tickets (UK Film Council 2007). In 2006 alone, 505 films were released in

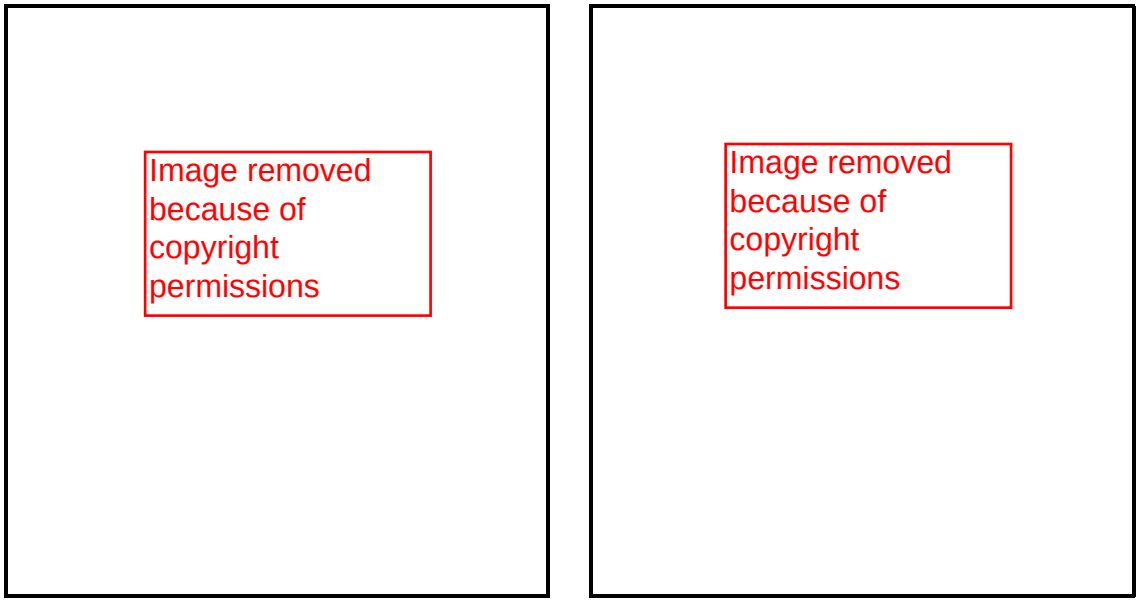
the UK and Ireland, the highest for at least the last five years. In total, box office takings in Britain have remained above £740 million since 2002, this year's takings making the increase for the last ten a healthy 55.8%. Within the film business, the visual effects industry[1] is now an integral part of the filmmaking process. Looking at the top twenty films in the UK, by box office takings for the last ten years, four fifths of them relied on digital effects for the final product, including 'Bridget Jones's Diary', which was worked on by no fewer than three visual effects companies (IMDB 2007). Of the four that did not employ effects artists in post-production, three are computer animated adventures. The only one for which computers were not used for the visuals at all was 'The Full Monty', which is one of those rare shoe-string budget success stories. It is now routine to digitize a film after it is shot (if it has not been captured digitally in the camera), even if it is only to perform colour grading or minor 'clean-up' artistry (where, for example, film crew who inadvertently appear in reflections and the like are removed). All this says nothing of the proliferation of visual effects on the small screen (television cameras have always been electronic), particularly conspicuous in many title sequences and music videos. The availability of digital manipulation for film and video, both professionally and in the amateur world, makes the research into tools for this purpose very relevant.

Because almost all video image sequences depict movement (indeed, even for a 'static' shot it is often hard to keep the camera still), the successful manipulation of those images almost always requires knowledge of the motion observed. The human visual system is sensitive to many temporal stimuli, and movement is one of these. As such, it is imperative that, for the integration of moving elements, the motion visible in the original footage is matched exactly. Any mismatch between the augmented visuals and the real life footage draws attention to the trickery and destroys the illusion the director is relying on. Figure 1.2 displays some examples from recent motion pictures of where digital augmentation of the

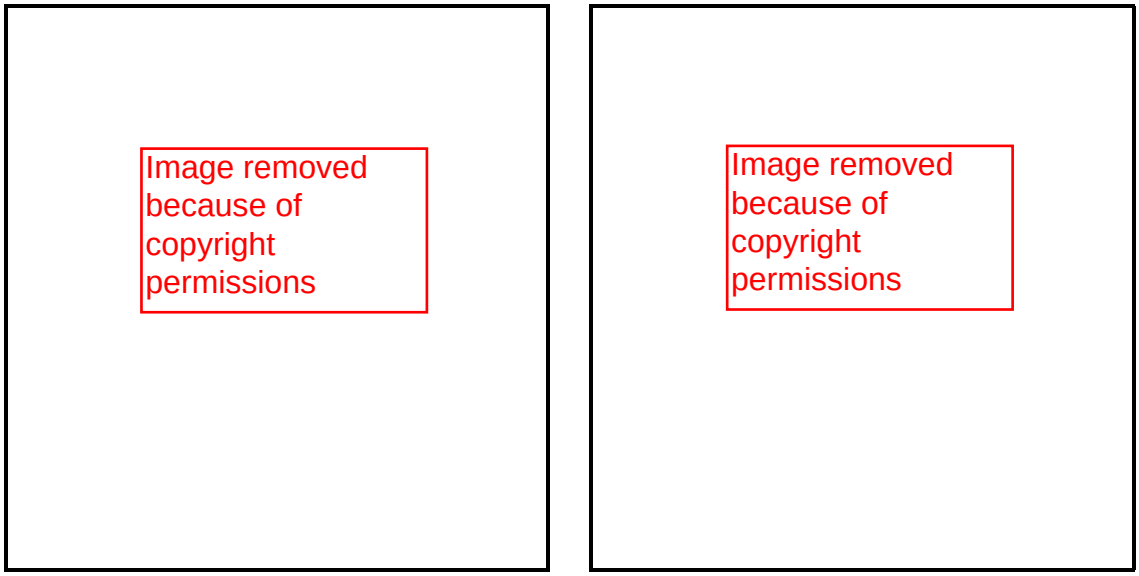
[1] Visual effects being the technically correct name for special effects that are created digitally in post-production; special effects are strictly only executed during filming



The near-future setting of 'Children of Men' required many scenes to have animated adverts on the sides of buses and buildings, sometimes on many independently moving objects in the same shot.



In order for the performance of the actor to show through in the computer-generated tentacles in the film 'Dead Man's Chest', the actor's very non-rigid face was carefully tracked.



A foul monster from 'Blade II' had an extending worm-like proboscis and a disturbing sideways opening jaw that had to blend perfectly with the real movement of the actor's jaw and cheeks.

Figure 1.2: **Video augmentation.** Examples of visual effects from some recent feature films.

original footage was used to create the desired effect. From the seemingly simple insertion of two dimensional posters on to buildings and buses to full three dimensional transitions of humans into horrific beasts, the initial acquisition of how the objects in the frame are moving is vital. It should be fairly understandable, therefore, that match-moving (or motion tracking as it is also known) is the basis of just about all visual effects.

In this thesis, the ways in which this underlying motion information is obtained is explored. Existing and commercially available methods fall short in two main areas: their inability to provide the right answer in many situations and their inadequate and cumbersome interface for handling the correction of their output when they do get it wrong. The answer for many in the visual effects industry is to simply complete the whole task by hand. This need not be the case. Therefore the aim of this thesis is to introduce new ways of obtaining motion information from video and thus be able to overcome the failings of existing methods by providing systems that work with any video footage at all.

1.1.1 Tracking

Tracking is the process of acquiring motion information from video. It can be said that there are three classes of tracking: *optic flow*, *feature point tracking* and *batch tracking*. The former is the process of determining the frame-to-frame motion at every single pixel in each image from the sequence (the local "visual slippage" across the whole image from one frame to the next). It takes a lot of computing power, but gives very dense motion information. Its shortcoming is that when these frame-to-frame motions are concatenated across the entire sequence, the inherent small errors and imprecisions build up unchecked, resulting in a drift away from the actual motion (constraints on optic flow have been realized, but only for extremely short video clips). I do not investigate optic flow techniques in this thesis, instead focusing on the other classes.

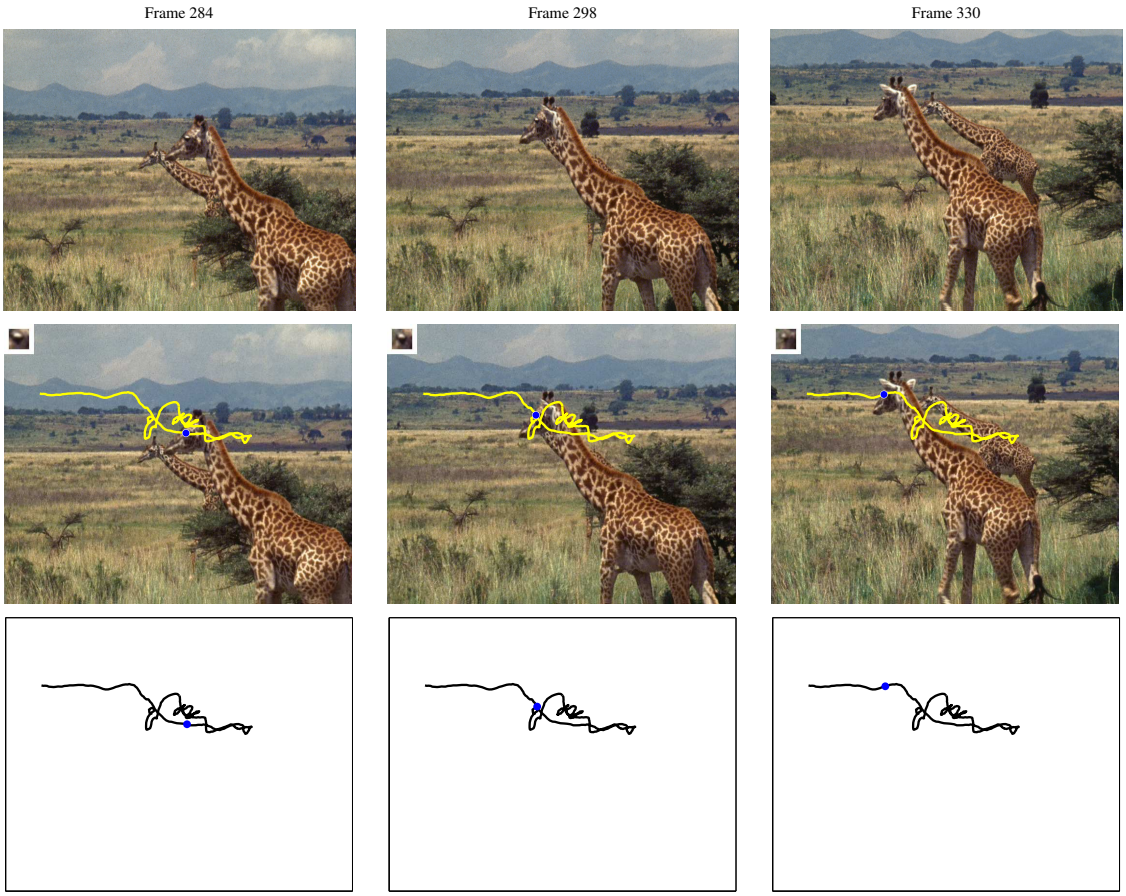


Figure 1.3: **Example of a typical track.** Top row: frames from a video sequence where a giraffe in the foreground walks in front of another giraffe further away. The camera pans to keep the foreground walking giraffe in frame. Middle row: the same frames with the track of a feature on the foreground giraffe (its eye in this case) overlaid. The position of the feature in each frame has been highlighted with a big dot. The image region around the feature point (the *feature patch*) has been included, double size, in the top left of each image. Bottom row: the shape of the track without the context of the images.

Feature Point Tracking

Feature point tracking is the determination of the motion of an individual *feature* as it appears in an image sequence. A feature is a single point on a real world object or surface that is in some way distinct. Distinctiveness often arises from variation in a surface's *texture* (which in this context is the pattern of colour and intensity across an object's surface, i.e. the reflectance function over real-world surface position, also known as albedo), but it can also be derived from the shape of an object, such as at corners. The key requirement of a feature is that its

differentiating characteristics should remain after they have been projected through a lens and appear in an image, making them identifiable for computer vision applications. For ease of expression, the term feature will also be sometimes used for their appearance in images as well as for the actual real-world entity. Taking a small window of pixels from around an image feature produces a *feature patch*. Feature patches are small sub-images (typically 3% of the full image's size) centred on a feature's location (see Figure 1.3). These are used in isolation of the image from which they are extracted. They are a special case of the more general *image patch*, is any sub-array of pixels from an image (i.e. not necessarily associated with a feature).

Having tracked a feature, we would know its location in every frame of the video. Collecting all these points together in temporal order provides a feature's *track*. Therefore, the high level description of the tracking process is simple: find the coordinates of the chosen feature in every frame and join them up. Of course, implementing this process for real world footage is difficult.

The difficulties arising when tracking a feature through an image sequence are slightly clearer with the actual tracking methodology in mind. Before moving on, therefore, let us consider what is involved in the process of obtaining a feature track. I will start by describing an example procedure:

Starting in the first frame

1. The user clicks on the feature; an image patch is extracted at that location.
2. In the next frame, the extracted patch is used to search for the feature within an area centred on the feature's position in the previous frame.
3. The position with the greatest similarity is chosen as the location of the feature in that frame.
4. If no satisfactory match is found, goto step 1, otherwise goto step 2.

This is but one of a great many possible algorithms. A more general framework for

tracking can be best described from a probabilistic viewpoint. Each frame is assigned a probability density function (PDF) over its pixels, $p(x, y)$, representing the probability that the projection of a real-world feature appears at any particular location. The feature track is inferred by considering all the PDFs together. Each PDF is derived from two types of information: feature appearance and feature position in all of the other frames[2].

Appearance. The feature's real 3D appearance is not known—only projections of it in the frames of the image sequence. Therefore, an idea of the feature's full appearance is only obtained from examination of feature patches extracted from pictures and frames of video. Furthermore, because a feature's appearance frequently changes over time (due to changes in lighting for example), very many patches may need to be collected. I will call such a collection of patches the *patch library* and it will be considered to contain (possibly correlated) draws from the distribution of all possible projected feature appearances.

Position. By definition, until the track is complete, the feature's location will only be known from its supposed locations in a sub-set of the video sequence's frames. The frame sub-set may be contiguous, but is not guaranteed to be so. Nevertheless, the set of known positions will be referred to as the feature's known *motion*.

Models of appearance change and projected movement are used to determine the influence of the evidence on the per-frame PDFs.

Initially, there is no information on the feature and so all PDFs are uniform. To start the process, a human operator, or *user*, must indicate the target feature. I am going to assume that no information on the features appearing in a particular video can be known from outside of the video itself. This is reasonable in the context of visual effects because of the vast

[2]Technically, there is, of course, only one source of information: the images of the sequence, but images, being intensity functions over $\mathbb{R}^2$, do provide two separate classes of evidence: intensity and position. Because the mapping is many-to-one and hence not reversible (colour does not imply position) there are two types of information in a real sense, e.g. consider the tracking problem in either one data type alone.

variability of video sequences that are worked on in post production. Therefore, the feature to be tracked is selected by the user clicking on it in a frame of the video. This click provides a definite coordinate and feature patch, so the PDF for that frame can be collapsed to a tight peak at that point and an appearance patch can be added to the patch library. These updates change the knowledge available, which results in a cascade of updates in all of the other PDFs. For example, if the motion model was Brownian motion, the influence of the knowledge of the first coordinate would be ever increasing Gaussians, expanding like space-time cones of influence. Once the updates are complete, it might be possible to determine the coordinate of the feature in another frame, creating a chain reaction of updates and determinations. If a location for the feature appears in every frame then the feature's track has been determined and the job is done. If not then the user may need to give another indication of the feature's trajectory to provide that extra bit of information (in appearance and position) to kick-start the process back into action.

The above description introduces all the aspects of the tracking process. To anchor it to reality, it also gave some implications regarding user interaction. Given the assumption of working solely with a single image sequence, these practical interjections do not affect the generality of the description. Indeed, it should be noticed that it can be trivially extended to cover the fully general case via the models. All existing trackers can be seen as approximating implementations of the above procedure. For example, the basic template tracker approximates the influence of motion by setting a rectangular 'top-hat' function centred on the feature's position in the previous frame and then determines the influence of appearance using a cross-correlation function on the image with the feature patch in the patch library.

This is the simple algorithm introduced above:

1. The user clicks on the feature; the extracted feature becomes the patch library.
2. In the next frame, the patch in the patch library is used to search for the feature within a rectangular region centred on the feature's position in the previous frame.
3. The position with the greatest similarity is chosen as the location of the feature.
4. If no satisfactory match is found, goto step 1, otherwise goto step 2.

This algorithm description omits some unimportant technical details, but otherwise represents the base position well. As presented, the patch library would hold only one patch defined by the location of the last user click. In this thesis, this approach is referred to as *track-to-first*, because it uses the appearance from the first frame of any given block to determine where the feature is in the current frame. A common alternative is to replace the patch in the patch library in every frame with the patch extracted from the previous frame. Here, this approach is called *track-to-previous*:

1. The user clicks on the feature; the extracted feature becomes the patch library.
2. In the next frame, the patch in the patch library is used to search for the feature within a rectangular region centred on the feature's position in the previous frame.
3. The position with the greatest similarity is chosen as the location of the feature; the extracted feature becomes the patch library.
4. If no satisfactory match is found, goto step 1, otherwise goto step 2.

One can imagine a range of variations, but specific implementations will not be described

until the next chapter.

With the underlying approach defined, it is easier to understand the complications that arise in real-world video and the ways in which existing tracking schemes fail to adequately account for them. I separate these difficulties into three types: *occlusions*, appearance changes, and *repeated texture*.

Occlusions. The first complication is the lack of guaranteed continuity of presence: the feature may not appear in some frames of the sequence. This may arise for two reasons: i) the feature moves out of frame, outside the visibility of the camera or ii) an object in the scene moves between the feature and the camera, obscuring it from view. In most sensible situations, the former only happens once the feature's position gets close to the edge of the picture and so is, in some sense, not unexpected. The latter instance, however, referred to as an occlusion, is much more unpredictable and makes tracks end in the middle of the frame and reappear in a location whose correlation to the last known position decreases with the length of time the feature is not visible. As a human it is often quite easy to spot occlusions due to our ability to analyse images semantically, but engineering a system that can account for such situations automatically is very difficult. Most existing trackers do not explicitly acknowledge occlusions at all (using the term for both types of disappearance) and instead roll together this problem with the next one ('appearance change' below) and simply give up when things get too tricky. This results in the forcing of the user to interact to get the process going again. Of course, sometimes trackers fail to notice anything is wrong (possibly due to the third problem: 'repeated texture' below) and continue to generate a false positive track that has to be reviewed and deleted by the user, making for great inefficiencies.

Appearance change. The second complication arises because of the fact that the appearance of the feature can, and invariably does, change as the image sequence progresses. There are many causes of appearance change, but they can be largely separated into two classes.

The first class is lighting effects, which can itself be attributed to several causes: i) changes in the scene's lights themselves or aperture adjustments in the camera, which results in alterations such as brightness and contrast changes; ii) relative movement between the light sources in the scene and the feature's object, which may also result in brightness or contrast changes, including lighting gradients across features; iii) the property that the appearance of a feature in a frame of video can be different if just the viewing direction changes (e.g. because of specular and anisotropic materials); iv) more complicated global lighting effects, such as shadows, which can create incoherent, sometimes drastic appearance changes. The basic, linear changes in this class can be accounted for relatively straightforwardly using normalization techniques to produce lighting-invariant feature patch description schemes, but the higher order effects can not be accounted for procedurally and so can pose a significant problem.

The other class of causes of appearance change is patch warping. Again, there are several ways the appearance of a patch can change in shape between two frames: one is from any relative movement between the camera and the feature's object. This causes the feature to be viewed from a different angle and therefore produces a different image. If the 3D structure of the feature's object is flat around the feature itself, then range of appearance changes is limited to distortions called *foreshortening* in most cases or general perspective warps in general. Otherwise, the distortions will be arbitrary from frame-to-frame, but coherent over the long term because of the constraint of the underlying object's shape being fixed. Of course, a fixed shape assumes that the object is *rigid*. If the feature lies on a surface that changes shape over time then it is said to belong to a *non-rigid* object. Non-rigid objects include almost all natural things (such as animals and most plants), any articulated structures (e.g. mechanical cranes and angle-poise lamps) and flexible structures (like clothes and flags). Non-rigid warping, therefore, comes from the possibility that an object's surface, around a feature, itself changes shape. In general, the distortions to the appearance of features on non-rigid objects are wide-ranging and can be incredibly complex and therefore are in general

difficult to deal with.

Changes in feature patch appearance creates problems for the image search phase of traditional trackers. Track-to-first algorithms are, in general, too strict and either lose the feature prematurely or succumb to the complication of repeated texture (below), because after a certain point they are effectively searching for the wrong appearance. The variety of appearance changes seen in real footage inspired the track-to-previous approaches, but they turn out to be too flexible. While in theory being able to cope with any changes in appearance, in practice the lack of constraint on what such trackers will accept mean that they quickly *drift* away from the target feature and end up tracking other parts of the scene (most often static background). The presence of occlusions exacerbates this problem as the patch search too readily accepts matches on the occluding object, thus losing the actual feature being tracked completely.

Repeated texture. The third and final complication is what I will refer to as repeated texture. This problem arises because sometimes a feature is not unique in appearance. Most simply, this situation can arise when there are multiple instances of an object in a scene, for example the wheels on a car in traffic or the stripes on a tiger. A naïve tracker might always pick the best match after doing a feature patch search, but it is easy to see that, given the propensity of appearance to vary, such an approach has a high likelihood of jumping to the wrong feature. Even when thought has been given to the motion model to help choose between competing matches, the problem is still significant and causes many solutions to fail.

Together, these complications constitute the hurdles to be overcome by any successful tracking algorithm. All existing methods fall at one or more of these hurdles for at least some sequences. However, for the visual effects industry it is crucial that the most general video sequence is accounted for because of the unconstrained variation of the footage that

they have to deal with. This thesis presents new approaches to the tracking problem that help to overcome these problems.

Batch Tracking

Batch tracking is very closely related to feature point tracking. It is the automated generation of a large number of tracks throughout a video sequence. Its output is used by systems that aim to recover a global aspect of the video, such as the three dimensional structure of the scene or the 3D motion of the camera itself. The key distinction in contrast to feature point tracking is the lack of specificity of the tracks it generates. While batch tracking strives to produce completely accurate tracking data, as with feature point tracking, the feature points chosen, their position in the frame and the duration of their tracks are not as important. The goal of batch tracking is to generate as many accurate tracks as possible, across as much of the frame as possible. The result is a set of typically short tracks, with every frame of the sequence having at least some starting there, so that the durations of all the tracks overlap one another. Each small track can be generated using an automated feature point tracker, so the issues described above are relevant to batch tracking as well. However, as there is little penalty in including short tracks, the troubles of tracking any one individual feature (as described above) can be dealt with by simply aborting that track. The more prominent concerns of a batch tracking system are the automatic detection of features and the qualitative criterion for individual track cessation. The relationship between batch and feature point trackers is one of the issues that will be discussed in this thesis, but only once a more detailed picture of the mechanisms involved has been presented.

1.1.2 Thesis Overview

Now that the basic concepts have been introduced, a more detailed picture of the work in this thesis can be given.

Chapter 2 goes straight to the issue of generating long accurate feature tracks, which are essential for the production of successful visual effects shots. The goal of the visual effects artist is the seamless integration of artificial imagery, so it is vital that the motion in the original video is matched perfectly, i.e. the user of the tracking software requires perfect tracking data. Existing software misses the mark by either trying to provide an automated solution, which frequently fails because of the complexity of the footage generally worked on, or by simply presenting a glossy interface for an otherwise fully manual system. The claim of this thesis is that a solution lies between these extremes, combining a user-oriented interface with computer vision. The proposed solution can be conceived as a power-assisted manual tracker, giving near instantaneous automated feedback, with the ability to cope fully with any type of video, including the all-too-common problem of occlusions. All interactive systems require at least one mouse click, to select what it is that is to be tracked, and with this system that is sometimes all that is required to obtain a full, perfect track, displayed on screen almost instantly. If the track presented is not perfect, the user simply scans through the video and selects another frame in which to make a correction, after which the full track is immediately updated. In this way the user, through intuitive actions, directs an iterative process and is constantly updated with the latest automatic result. Video sequences tens of thousand of frames long can be tracked to 100% satisfaction in minutes, orders of magnitude faster and more conveniently than existing solutions.

It is worth noting that the creation of an interactive feature tracking system has two further significant reasons to justify it: firstly, high precision accurate tracks of arbitrary features are required by many people beyond the visual effects industry. The tracking of motion in video has a high forensic value (e.g. in security, surveillance, police detective work and cricket); secondly, tracks are often required on fine detail that defeats optical flow algorithms (e.g. the plume of a feather, the tip of a spear) and on locations which have semantic importance (the corners of buildings, animals' eyes), but which cannot be expected to emerge from full-frame motion computation, possibly simply due to the signal to noise ratio of motion around such

features.

The tracking system described in Chapter 2 deals with the motion of a single feature point alone. The motion information in its calculations is local to the single track being generated. This makes it particularly hard to predict where the feature is going, particularly while it is occluded. However, image motion elsewhere in the frame often gives strong cues as to the trajectory of the target feature. Indeed, when the scene in front of the camera is completely static, knowing the motion of other points in that scene means that a target feature's position in the next frame can be calculated almost exactly. This is information that batch trackers routinely draw upon to greatly improve performance. I am interested in tracking features far beyond the restricted world of rigid scenes, but the point is that the global motion across the frame is a resource that should not be overlooked.

The question investigated in Chapter 3 then, is how global motion information can be distilled to a form suitable for incorporation into feature point trackers. This question is equivalent to setting up the outlier rejection scheme of batch trackers. As such, a good answer is known: rank-constrained matrix factorization. Indeed, this is also the basis for other important computer vision tasks, as well as batch tracking, such as surface reconstructions from changes in lighting. A significant problem here, though, is that the matrix concerned is often not complete. It is often the case that many of its entries are not known, because, for example, feature points were occluded in some frames making observation measurements impossible. Chapter 3 describes the mathematics of matrix factorization and describes the development of algorithms to overcome the important and common practical problem of missing data.

Having been thoroughly introduced, a role for matrix factorization in feature point tracking is easier to explain. Some form of motion model is always needed to guide the choice between candidate feature locations in any given frame. As mentioned above, batch trackers can obtain a consensus on the movement of features across the entire image. Embracing this global aspect has the distinct advantage that erroneous frame-to-frame candidate correspon-

dences (that go against the overall motion) can be isolated and rejected as outliers. Of course, the batch processing technique is not appropriate for user-initiated feature tracking: only a restricted subset of features in a given frame are incorporated and of those, any for which a corresponding point in the next frame cannot be found is dropped and ignored thereafter. A good feature point tracker must provide a track for any feature from any frame and should certainly be less blasé about false negatives. However, the global motion information provided by a batch tracking system can inform the motion discrimination used to reject false positives in the feature tracker. Chapter 4 describes the combining of global motion information into otherwise local feature tracking systems. To highlight the universal applicability of such incorporation, the tracking problem is recast in the general framework of probabilistic inference and the influence of the global information is presented as a motion prior. Naturally, the assumptions on the motion that is to be analyzed are kept to a minimum. This comes partly from the novelty of working in the two dimensions of 'screen space' rather than the more conventional idea of constraining the expected motion by modelling the camera and a three dimensional world in front of it. The result is a very flexible yet effective method of guiding feature point trajectory determination, even in video with considerable independent and non-rigid motion.

Chapter 2

Interactive Tracking

This chapter describes a new tracking system. It has been designed to generate long accurate tracks, despite common real-world problems, such as the need to track very small regions on sometimes highly non-rigid objects. In general, there can be no guarantees on minimum region size, object structure or surface motion. This is why a generalized interactive feature point tracker is the correct approach.

In the technology available to date, there is a large gulf between (semi-)automatic tracking algorithms (that sometimes give the user a degree of control to initiate or correct the tracks being generated) and the default technique of doing the whole process by hand (manual tracking). Existing software tends to be slow or cumbersome to use, sometimes to the degree that manual tracking is faster for the experienced visual effects artist. A new type of interactive tracker is required to fill the gap.

The aim of the work described in this chapter is a system that produces 100% perfect feature tracks with as little user interaction as possible. In contrast, the ethos behind existing software seems to be focussed on maximizing automation. While on the face of it, the goal of an automated tracker is to maximize output quality and minimize interventions, the imperative of perfect tracks is overlooked. So, in practice, when such systems fail, which they almost always do (often very quickly, being untuned for the vast majority of real-life

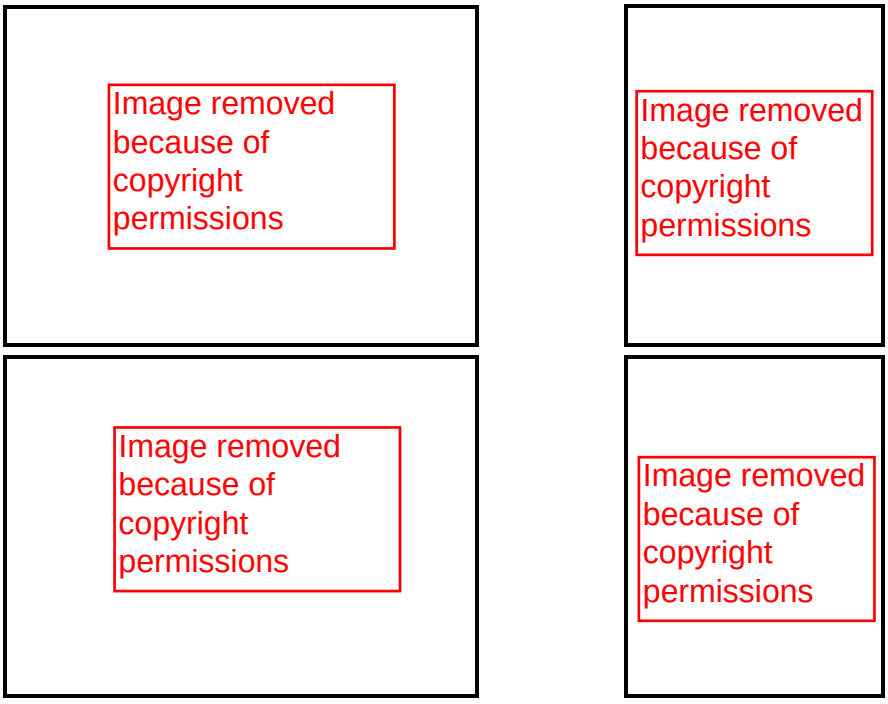


Figure 2.1: **Real world problems.** It is not uncommon to be required to track small flexible objects, such as for the removal of the backpack from this stunt parachutist playing a doomed character in 'The Good Shepherd' (left column), or highly non-rigid objects, like the back of a jacket from the film 'Driven' (right column).

sequences), the user is forced to intervene in what turns out to be a lengthly process of stopping and restarting.

Instead, I consider a system that explicitly models the user in the feedback loop. By building an algorithm around the user, the result is a hybrid manual/automatic tracking platform that slides gracefully from being completely automated to being completely manual, thus guaranteeing that the generated tracks completely fulfil the user's requirements, while minimizing user input.

2.1.3 Problem Overview

The aim is to design an "interactive feature tracker" which should be defined as the process of extracting long and accurate tracks of real world features observed in an image sequence,

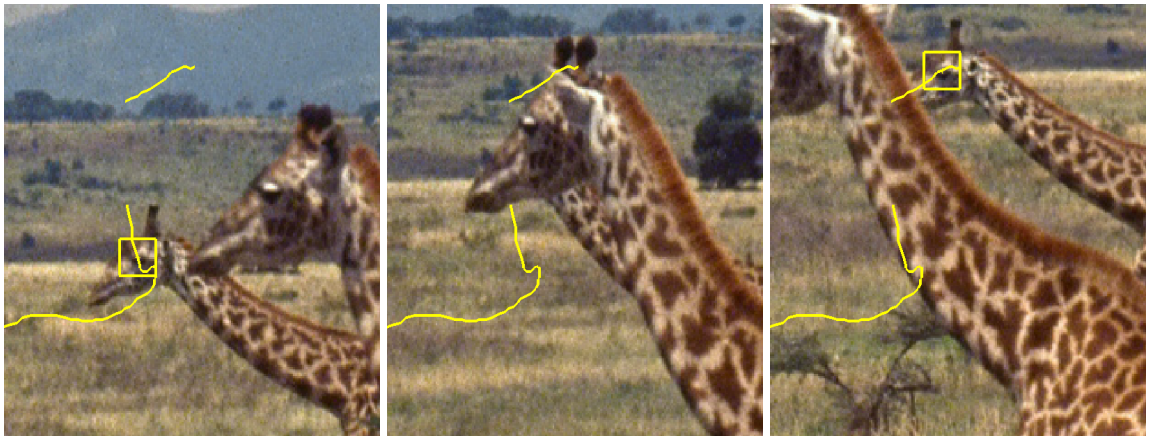


Figure 2.2: **An example track.** Generated using the system described on 140 frames of the walking giraffe sequence (720×576) from Figure 1.3. The eye of the background giraffe was selected in the first frame. Detection of 50 candidate matches from the entirety of every frame was conducted at over 50fps. Track optimization using dynamic programming, including the correct analysis of the occlusion in the middle of the sequence, ran at 175fps on a 3GHz PC in MATLAB.

in which manual initiation is expected and manual intervention is accepted, but where, crucially, automated responses are provided very rapidly. This definition distills the essence of what is being strived for and the limits thereof. Requiring accurate tracks is a trivial criterion of a successful system. Such a tracking system must also be able to deal with both short and long video sequences (i.e. more than 1000 frames) because applications demand it; some effects shots are edited to be less than a second long (generally as part of a film's action sequence) while it is quite possible to also have some lasting many minutes (in a dialogue scene, for example) and, of course, everything in between. It is also important that interactive means that corrective inputs by the operator are rapidly rewarded with an updated result. Quick responses allow for a workable feedback loop incorporating the user, efficiently harnessing all available resources (i.e. both the computer's processor and the user's brain). Lag is destabilizing in any feedback control system and in this case the effects are keenly felt by the operator of the software. However, although the engineering of a solution must, therefore, be built around the possibility of corrective intervention, it is still vital to minimize user effort, hence the use of the verb 'accepted' in the definition. On the other hand, it is important

that the process is initiated with a manual input because any object in a video sequence can be the target of post-production attention. As such, any feature point tracking operation will always necessarily require at least one user interaction. Any performance measure based on minimizing manual effort will be viewed in the context of this baseline. Real world features are mentioned to make reference to the general nature of the footage being dealt with.

Ideally, a tracking system would allow points of interest to be indicated with a single mouse click in one frame of the video and then give the desired output: the location of the point's 2D projection in every frame of the sequence. A perfect system would

- be robust to occlusions of the feature over arbitrary numbers of frames
- impose no inherent restriction on the way the underlying object moved (including the speed it moved at and the directions it moved in)
- cope with considerable appearance change due to any or all of the factors mentioned in Chapter 1
- avoid algorithmic artifacts such as drifting away from the point being tracked[1].

When a perfect result is not immediately presented, the system should allow interactive corrections.

2.1.4 Track Notation

Before going any further, I want to formalize the problem of track generation mathematically. This will help in the comparison of the new tracker presented here with existing routines, showing up the similarities and differences more clearly.

Let me start by defining precisely the term *feature track*. Consider an image sequence of length F frames, $\mathcal{I} = \{\mathtt{I}_t \,|\, t = 1 \ldots F\}$. At the core of a feature track is, of course, a set of F image locations $\mathcal{X} = \{\mathbf{x}_t \,|\, t = 1 \ldots F\}$. With each 2D location is associated an

[1] Often called the "template update problem" Matthews et al. (2003)

Symbol	*Meaning*
t	video frame number
F	total number of frames
$\mathtt{I}_t$	image for frame t
$\mathcal{I}$	the set of all the images in the sequence, $\{\mathtt{I}_t \,\vert\, t = 1 \dots F\}$
$\mathtt{I}_{\mathcal{W}_\mathbf{x}}$	a window of pixels from image $\mathtt{I}_t$, centred around $\mathbf{x}$, i.e. an image patch
$\mathtt{I}$	the identity matrix
$\mathbf{x}$	image coordinates
$\mathbf{x}_t$	feature location in frame t
$\mathcal{X}$	the set of all feature locations, $\{\mathbf{x}_t \,\vert\, t = 1 \dots F\}$ i.e. the track's trajectory
$\mathcal{W}_\mathbf{x}$	the set of all pixel coordinates within a fixed radius of $\mathbf{x}$
$\mathbf{p}_t$	the image patch representing the appearance of the feature in frame t
$\hat{\mathbf{p}}_t$	the vector descriptor for patch $\mathbf{p}_t$
$\mathcal{P}$	the set of all feature appearance descriptors, $\{\hat{\mathbf{p}}_t \,\vert\, t = 1 \dots F\}$
$\mathcal{T}$	all (location,appearance) pairs, $\{(\mathbf{x}_t, \hat{\mathbf{p}}_t) \,\vert\, t = 1 \dots F\}$, i.e. the track itself
$\mathbf{y}_i$	the i^{th} user-selected keyframe location
$\mathbf{q}_i$	the i^{th} user-selected keyframe appearance
$\mathcal{N}$	the set of frame numbers of user selected keyframes
$\mathcal{Q}$	all keyframe inputs, $\{(\mathbf{y}_t, \hat{\mathbf{q}}_t) \,\vert\, t = 1 \dots \vert\mathcal{N}\vert\}$, i.e. the patch library
l_t	candidate label selection variable for frame t
$\mathcal{L}$	the set of all labels, $\{l_t \,\vert\, t = 1 \dots F\}$
E	the track evaluation function
E_*	a track evaluation sub-function
$E_{*,t}$	a track evaluation sub-function integrand: $E_* = \lambda_* \sum_{t=1}^{F} E_{*,t}$

Figure 2.3: **Notation.** Summary of notation used in this chapter (fold-out copy at back of book).

appearance, $\mathbf{p}_t$, representing the projected image of the feature expected to be seen in the neighbourhood of $\mathbf{x}_t$ in frame t. The set of appearances is denoted $\mathcal{P}_\mathcal{X} = \{\hat{\mathbf{p}}_t \,\vert\, t = 1 \dots F\}$. To maximize the utility of the notation, each $\hat{\mathbf{p}}$ is taken as an appearance description vector. The description vector could be any way of representing an image patch. For example, it could be the actual pixel intensities within a radius (measured under the L_1 norm, L_2 norm or otherwise) of the feature location, in which case $\hat{\mathbf{p}} = \mathbf{p}$. It could also just be some hash function value of such a patch. It could even include some pixel weighting function of radius. My implementation is explained below, but is basically a transformation (or filter bank projection) that trades between discriminatory power and memory footprint. A track,

therefore, is the set of (location, descriptor) pairs, one for each frame of the sequence:

$$\mathcal{T} = \{(\mathbf{x}_t, \hat{\mathbf{p}}_t) \,|\, t = 1 \dots F\}\,. \tag{2.1}$$

The use of an intrinsic appearance variable has not seen much attention in the literature, but I think it is important to explicitly define and use these associated appearances because of the additional important information that they contain. Specifically, when quantitatively assessing the quality of a track, i.e. the possibility that it represents the 2D projection of the motion of the feature point of interest, it is clear that it is not just the 2D motion that determines how appropriate the candidate track is, but that it is also the 2D appearance of the neighbourhood about the chosen points that reflects the quality of the match. The use of the descriptor $\hat{\mathbf{p}}$ instead of $\mathbf{p}$ is deliberate for two reasons: the first is that I wish to emphasize the thought of an evolving feature patch being embedded on a manifold in description vector space (more on this below); the second is simply that it is the descriptor that is used in the resulting implementation.

Ideally, I would like to frame the determination of track parameters $\mathcal{T}$ as a probabilistic inference of positions $\mathbf{x}_t$ and appearances $\mathbf{p}_t$ from observations in the form of image pixel sets $\mathcal{I}$, i.e. treating $\mathcal{T}{=}(\mathbf{x}_t, \mathbf{p}_t)$ as latent variables. However, the need for performance efficiencies takes the actual implementation away from this ideal: firstly through the introduction and consideration of only a discrete subset of observations of position and appearance in each frame, $(\mathbf{z}_{i,t}, \mathbf{u}_{i,t})$; and secondly by using these observation variables directly for the resulting track during the interactive phase of the process (after which a final refinement phase produces a more precise track for the actual output). Nevertheless, the final implementation does retain references to this ideal and it is important to develop the explanations of the formation of the final algorithm by referring to $(\mathbf{x}_t, \mathbf{p}_t)$ as 'latent'. Also, for the end of chapter summary, it is possible to explain future extensions more concisely using this terminology.

It is also important to introduce an entity that can represent the user defined constraints

on the track. A track is initialized by the user indicating points in any number of *keyframes*: a subset $\mathcal{N}$ of the input frames. In every keyframe, an accurate observation of the location and appearance of the feature track is defined. For many tracks, the number of keyframes $|\mathcal{N}| = 1$, but if, for example, a feature changes appearance significantly while occluded, it will be necessary to insert an additional keyframe when the feature emerges from occlusion. The keyframes are also denoted by the set, or library of (location, descriptor) pairs $\mathcal{Q} = \{(\mathbf{y}_i, \hat{\mathbf{q}}_i) \,|\, i \in \mathcal{N}\}$, in the same way as tracks are defined.

Having these two entities defined makes it easy to generalize about off-line tracking algorithms in general; i.e. they can all be described as a system that takes $\mathcal{Q}$ as an input and returns $\mathcal{X}$ (or $\mathcal{T}$) as output, describing the position (and associated appearance) of the feature through the image sequence. Obviously the algorithms to get from input to output can, and do, vary widely, but having this concise view of the problem makes the situation clear and comparisons easier.

As well as enabling generalizations, this representation of the tracking problem crucially allows us to unambiguously define measures of the 'quality' of a track: given two tracks, $\mathcal{T}_1$ and $\mathcal{T}_2$, which best describes $\mathcal{Q}$ through the image sequence? I present a generic function that measures the quality of a track relative to an input keyframe set, $\mathcal{Q}$:

$$E(\mathcal{T}, \mathcal{I}, \mathcal{Q}) = \underbrace{E_{\mathcal{X}}(\mathcal{X})}_{\substack{\text{motion}\\ \text{smoothness}}} + \underbrace{E_{\mathcal{P}}(\mathcal{P})}_{\substack{\text{appearance}\\ \text{smoothness}}} + \underbrace{E_{\mathcal{Q}}(\mathcal{T}, \mathcal{Q})}_{\substack{\text{appearance}\\ \text{fidelity}}} + \underbrace{E_{\mathcal{I}}(\mathcal{T}, \mathcal{I})}_{\substack{\text{observation}\\ \text{coherence}}}. \tag{2.2}$$

This function is the sum, over all the frames, of four terms: two related to smoothness, one to keyframe relevance and one to the images themselves. There is a trajectory term, $E_{\mathcal{X}}$, with which the frame to frame position, assigned to the feature, can be assessed, or for penalizing patterns of longer-term motion if desired; an appearance update penalty, $E_{\mathcal{P}}$, being the equivalent to the position velocity term, but for the appearance vector; a measure of the deviation of appearance from the keyframe set $\mathcal{Q}$, $E_{\mathcal{Q}}$, enabling comparison with the

user's input to be made; and a comparison of the track with the actual images themselves, $E_{\mathcal{I}}$, used between keyframes, for example. In addition, when appropriate, I use the convention that

$$E_* = \lambda_* \sum_{t=1}^{F} E_{*,t}. \tag{2.3}$$

with the assumption that $\lambda_* = 1$ unless stated otherwise.

With the presentation of a quantitative quality function, the tracking problem may be restated as an optimization problem: choose $\mathcal{T}$ to minimize $E(\mathcal{T})$. Many trackers only approximate a solution, because they process the image sequence strictly sequentially. Actually performing this minimization leads to superior results, which, in the context of an interactive tracker, means fewer inputs from the user.

Example Realizations

To help ground the utility of the track evaluation function, E, I will now describe again the two basic tracking paradigms introduced in the Chapter 1: track-to-first and track-to-previous, but using the new standardized notation. Even though the notation is presented in terms of a global optimization, it is important to be mindful that all the existing algorithms described below are implemented so as to generate a track frame-by-frame. This is equivalent to the frame-wise expression inside the summation of E being minimized for frame t before the consideration of the next frame. Nevertheless, with this operational difference in mind, comparisons are still appropriate.

Track-to-first:

$$E_{\mathcal{X},t} \quad : \quad |\mathbf{x}_t - \mathbf{x}_{t-1}| \leq \mathbf{r} \quad (constraint) \tag{2.4}$$

$$E_{\mathcal{P}} \quad = \quad 0 \tag{2.5}$$

$$E_{\mathcal{Q},t} \quad = \quad \|\mathbf{p}_t - \mathbf{q}\|_2^2 \tag{2.6}$$

$$E_{\mathcal{I},t} \quad : \quad \mathbf{p}_t = \mathtt{I}_{\mathcal{W}_{\mathbf{x}_t}} \quad (constraint) \tag{2.7}$$

$$\mathcal{Q} \quad = \quad \{(\mathbf{x}_1, \mathbf{p}_1)\} \tag{2.8}$$

which, in a sequential framework, reduces to a 'sum of square differences' (SSD) search within a window, of radius $\mathbf{r}$, centered on the previous location:

1. $\mathbf{x}_1$ set by user click in first frame.
2. $\mathbf{p}_1 = \mathtt{I}_{\mathcal{W}_{\mathbf{x}_t}}$
3. move to next frame: $t \leftarrow t + 1$
4. $\mathbf{x}_t = \mathrm{argmin}_{|\mathbf{x}_t - \mathbf{x}_{t-1}| < \mathbf{r}} \|\mathtt{I}_{\mathcal{W}_{\mathbf{x}_t}} - \mathbf{p}_1\|$
5. if $t = F$ then STOP; otherwise goto step 3.

As with all the algorithms described, these steps do not (in general) result in the minimization of E because the consideration of later frames can not inform the calculations of any earlier frames. This is elaborated upon below.

Track-to-previous:

$$E_{\mathcal{X},t} \quad : \quad |\mathbf{x}_t - \mathbf{x}_{t-1}| \leq \mathbf{r} \quad (constraint) \tag{2.9}$$

$$E_{\mathcal{P},t} \quad = \quad \|\mathbf{p}_t - \mathbf{p}_{t-1}\| \quad (t > 1) \tag{2.10}$$

$$E_{\mathcal{Q},t} \quad = \quad \|\mathbf{p}_t - \mathbf{q}\|_2^2 \quad (t < 1) \tag{2.11}$$

$$E_{\mathcal{I},t} \quad : \quad \mathbf{p}_t = \mathtt{I}_{\mathcal{W}_{\mathbf{x}_t}} \quad (constraint) \tag{2.12}$$

$$\mathcal{Q} \quad = \quad \{(\mathbf{x}_1, \mathbf{p}_1)\} \tag{2.13}$$

i.e. when implemented as a sequential algorithm is an SSD search within a window centered on the previous location using the patch from the previous frame (which, again, will not find the global optimum of E).

Many implementations require embedded minimizations of auxiliary parameters. A classic example is the Kanade-Lucas-Tomasi, or KLT tracker. It was proposed by Tomasi and Kanade (1991), based on the image registration technique due to Lucas and Kanade (1981), and updated to account for affine warpings by Shi and Tomasi (1994) (adding rotations and scalings). It is an extension of track-to-first, but comparisons to the template patch are made through a warping function, W, that takes coordinates in frame t and warps them back to the original patch coordinates in frame 1. New warp parameters, θ_t are calculated each frame, giving:

$$E_{\mathcal{X},t} \quad : \quad |\mathbf{x}_t - \mathbf{x}_{t-1}| \leq \mathbf{r} \quad (constraint) \tag{2.14}$$

$$E_{\mathcal{P}} \quad = \quad 0 \tag{2.15}$$

$$E_{\mathcal{Q},t} \quad = \quad \min_{\theta_t} \| W(\mathbf{p}_t, \theta_t) - \mathbf{q} \|_2^2 \tag{2.16}$$

$$E_{\mathcal{I},t} \quad : \quad \mathbf{p}_t = \mathtt{I}_{\mathcal{W}_{\mathbf{x}_t}} \quad (constraint) \tag{2.17}$$

$$\mathcal{Q} \quad = \quad \{(\mathbf{x}_1, \mathbf{p}_1)\} \tag{2.18}$$

A paper by Hager and Belhumeur (1998) presents an advancement in the implementation of this formulation that gives significant computational efficiency improvements (through the factorization of the linearization into constant and time-varying components). They also demonstrate how the ability to track through illumination changes can be naturally integrated into their framework (using the photometric stereo equations discussed in Chapter 3). Their paper also provides a concise review of this approach.

Approximate Minimization

As noted above, despite the fact that developing a track sequentially, frame by frame, precludes the ability to globally optimize track quality, thus impeding the chance of being able to obtain the correct track, most existing tracking algorithms are based on this approach. Nevertheless, a sequential search can only be considered as an approximate minimizer of the global error. In contrast, the system introduced here performs the optimization on the entire track, allowing the global optimum to be found. Being able to perform global minimizations is a crucial part of overcoming the difficulties of the tracking problem, especially occlusions and repeated texture. If, for example, one is tracking a feature on a parked car, a sequential tracker may be distracted by a similar feature on an occluding car that drives past in front of it, taking the track out of the frame. A global optimization would be able to notice that once the passing car has gone, the target feature is still there. An example of this is given in Figure 2.4.

2.1.5 Tracking Literature

Published work has yet to explicitly address the issue of user interaction for feature tracking. In essence, two classes of feature tracking approaches exist. The first is mentioned for completeness, and is the trivial mode of the completely manual approach. This approach has the user indicating the location of the feature in every frame of the video. Slightly more advanced versions can be imagined that allow for just subsets of frames to have identifications (e.g. every fifth frame) and then interpolate the intermediate positions (using, for example, cubic splines). Such a system was employed to obtain the animated effect for the film 'A Scanner Darkly' (Shay 2006). When inter-frame motion is small, geometric interpolation is sometimes acceptable. However, with only the minimum of track processing (that takes no account of the video frames themselves), members of this class cannot be regarded as 'interactive feature trackers'.

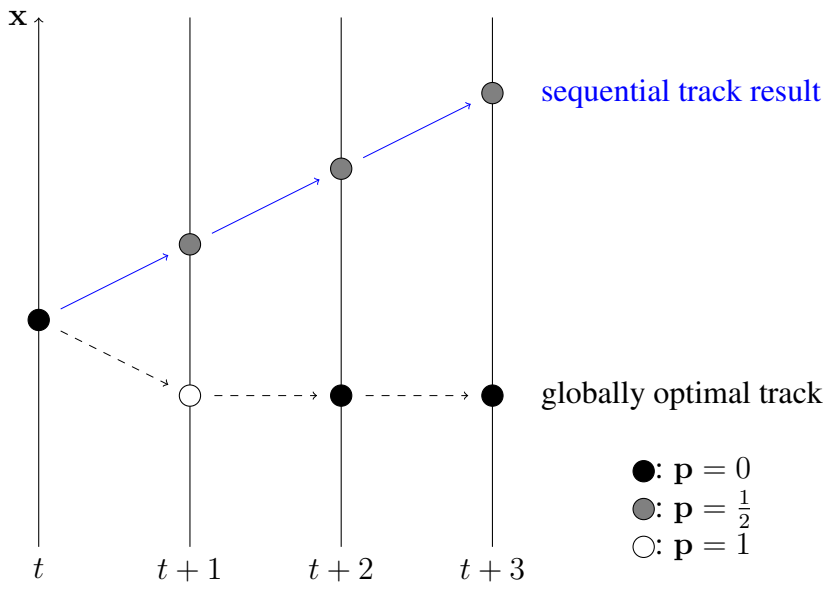


Figure 2.4: **Global vs sequential.** Approximating the tracking optimization using a sequential algorithm can miss the globally optimal solution. In this one dimensional example, a black dot in frame t is being tracked using track-to-first. In frame $t+1$ the choice is between a grey dot or a white dot both the same distance away. The sequential algorithm will pick the grey dot. This choice is fixed for the greedy algorithm, so the grey dot will continue to be tracked. The optimal track is via the white dot, which returns to its black appearance in frame $t+2$. A black to white appearance change is extreme, but equivalent situations arise in real tracking problems.

The second includes all (explicit or implicit) 'semi-automatic' tracking systems. The tracking algorithms for these cases have been designed to overcome some or all of the problems associated with feature patch tracking in an automated fashion. To date, almost all members of this class work through an image sequence sequentially, greedily determining the feature's trajectory up to the current frame before processing the next one. When in an 'interactive' setting, the mode of operation is simple: i) select the feature to be tracked in the first frame to initiate the tracking algorithm, ii) wait until tracking fails, iii) restart the algorithm by reselecting the feature in the failure frame. We have yet to see a solution that can automatically deal with any feature tracking task and so restarts are essentially inevitable, hence the term '*semi*-automatic'. However, there is a large variation in the degree of attention that the user must pay to this class of tracking process as it runs through a sequence. Some systems are able to stop before false positives start to be added to the end of a track,

allowing a more relaxed user to poll the tracking process and restart when necessary. Of course, to ensure that the user never has to correct false positives, the system must have a high restart rate, which goes against the aim of minimizing user interactions. At the other extreme are systems that do not address the possibility of failure and so require a constantly vigilant user to interrupt the tracking process when it goes wrong. This does theoretically mean that the minimum number of interactions are achieved for that tracking strategy, but in practice it results in the laborious process of reviewing the tracker's output to establish exactly when it started going wrong before it can be restarted. In general, those systems run not much faster than 'real-time' and so, in addition to the time-consuming restarts, the user must sit and watch the tracker's progress for at least the length of the video clip being worked on. A slight variation is the tracker built into the Boujou software package (2d3 2007), which tracks a feature between specified keyframes, but the user experience here is just that an addition input is required before tracking can commence. The second class, therefore, includes a range of approaches that lie on a scale that cannot be tuned to the point of being truly interactive.

It should now be more clear how the new system fills the gap in what is currently available, but it is worth having a more detailed look at the members of the second class ('semi-automatic'). An important mode of variation within this range of tracking algorithms is the way they choose the *template* (the image patch used to represent the feature being tracked) to find the location of the feature in the new frame. At one extreme is the track-to-first strategy, of those trackers that use a template defined by a single fixed frame (the first of the sequence) and ignore the fact that the feature's appearance will change over time. It is easy to see that these trackers will often quickly lose their target. A redeeming feature, beyond its simplicity, is that in situations where the appearance change of the feature being tracked results in the original appearance reappearing, possibly because of a cyclic motion or maybe due to temporary occlusion, it is possible that the target can be found again. However, even this would only be possible if the tracker is willing to search the entirety of every frame. Searching

across the full frame is a potentially time-consuming process. It is very common for trackers to depend on a motion model to limit computational cost in an attempt to counteract an inefficient or slow detection algorithm. However, using a motion model exactly means that assumptions are being made as to where the feature is expected to be in future frames. For general footage, the most common kind, such assumptions are unlikely to be valid, so, once a feature is lost, either due to occlusion or in the bad situation of a failing detector, it is highly unlikely that the program would be able to recover the feature track again because, having restricted the search area, it will be looking in the wrong place.

At the other extreme, the pixel window around the supposed position of the feature in the previous frame becomes the template for the current frame. This is the track-to-previous paradigm. At this end of the range, the appearance model is at its most flexible and can, in theory, cope with unlimited appearance variation, although only if it happens slowly. The problem is that there is too much flexibility in such an approach. The track will soon drift onto a stationary piece of background, because there is no mechanism to anchor the tracker on the desired feature. If there are occlusions then it can be impossible to tell if and when they have occurred, because the appearance will simply be adjusted to be that of the occluder.

Then, there is the intermediate scale of tactics for adapting the template appearance from frame to frame, which can together be referred to as *adaptive tracking*. These effectively try find a balance between track-to-first and track-to-previous. Simple versions, such as blending the template with the patch from the previous frame, tend to suffer a blend of the ills of both track-to-first and track-to-previous, without a satisfactory optimum.

This situation has been dubbed the ‘template update problem’ by Matthews et al. (2003), who propose the simultaneous use of both track-to-first and track-to-previous together, to try

and obtain the benefits of both while canceling out the disadvantages:

$$E_{\mathcal{X},t} \quad : \quad |\mathbf{x}_t - \mathbf{x}_{t-1}| \leq \mathbf{r} \quad (constraint) \tag{2.19}$$

$$E_{\mathcal{P},t} \quad = \quad \|\mathbf{p}_t - \mathbf{p}_{t-1}\| \quad (t > 1) \tag{2.20}$$

$$E_{\mathcal{Q},t} \quad = \quad \min_{\theta_t} \| W(\mathbf{p}_t, \theta_t) - \mathbf{q}\| \tag{2.21}$$

$$E_{\mathcal{I},t} \quad : \quad \mathbf{p}_t = \mathtt{I}_{\mathcal{W}_{\mathbf{x}_t}} \quad (constraint) \tag{2.22}$$

$$\mathcal{Q} \quad = \quad \{(\mathbf{x}_1, \mathbf{p}_1)\} \tag{2.23}$$

The sequential implementation is:

1. $\mathbf{x}_1$ set by user click in first frame.
2. $\mathbf{p}_1 = \mathtt{I}_{\mathcal{W}_{\mathbf{x}_1}}$
3. move to next frame: $t \leftarrow t + 1$
4. $\mathbf{x}_t = \mathrm{argmin}_{|\mathbf{x}_t - \mathbf{x}_{t-1}| < \mathbf{r}} \left\{ \|\mathtt{I}_{\mathcal{W}_{\mathbf{x}_t}} - \mathbf{p}_{t-1}\| + \min_{\theta_t} \| W(\mathbf{p}_t, \theta_t) - \mathbf{p}_1\| \right\}$
5. if $t = F$ then STOP; otherwise goto step 3.

The resulting tracker works well, but reasonable processing speeds are achieved only by restricting the search area, i.e. exactly the opposite of what is desired: minimal assumptions should be made about motion in order that any footage can be dealt with, particularly as after long occlusions, a feature may appear anywhere in the frame. As it stands, the issue of occlusions is not addressed in their work. Occlusion handling is part of a similar algorithm by Ishikawa et al. (2002), who robustify their tracker by ignoring sub-blocks of the region being tracked when calculating the match error. However, this does not go far enough, because in order to avoid problems of drift, they limit the maximum extent of occlusion to a small percentage of template area, and thus do not actually solve the real problem of occlusions.

Another approach is presented as the "wandering, stable, lost" algorithm, by Jepson et al. (2003), which defines tracking as an optimization over the eponymous and explicitly labelled detection states. The algorithm attempts to learn mixing parameters for each frame that

weight the relative contributions of recent frames as tracking progresses. An impression of their implementation can be summarized, with the introduction of a few new symbols, thus:

$$\textit{'lost'}\ E_{\mathcal{X},t} = \frac{1}{2}(\mathbf{x}_t - \boldsymbol{\xi})^\top \mathtt{V}_1^{-1}(\mathbf{x}_t - \boldsymbol{\xi}) + \frac{1}{2}(\mathbf{x}_t - \mathbf{x}_{t-1})^\top \mathtt{V}_2^{-1}(\mathbf{x}_t - \mathbf{x}_{t-1}) \quad (2.24)$$

$$\textit{'wandering'}\ E_{\mathcal{P},t} = \min_{\{o_{w,\mathbf{x}} | \mathbf{x} \in \mathcal{W}_{\mathbf{x}_t}\}} \frac{1}{2}\, \epsilon\, (\hat{\mathbf{p}}_t - \hat{\mathbf{p}}_{t-1})^\top \Sigma_{\mathcal{P}}^{-1} (\hat{\mathbf{p}}_t - \hat{\mathbf{p}}_{t-1}) \quad (2.25)$$

$$\textit{'stable'}\ E_{\mathcal{Q},t} = \min_{\{o_{s,\mathbf{x}} | \mathbf{x} \in \mathcal{W}_{\mathbf{x}_t}\}} \frac{1}{2} \left(\hat{\mathbf{p}}_t - \boldsymbol{\mu}_{t-1}\right)^\top \Sigma_{\mathcal{Q}}^{-1} \left(\hat{\mathbf{p}}_t - \boldsymbol{\mu}_{t-1}\right) \quad (2.26)$$

$$E_{\mathcal{I},t} : \mathbf{p}_t = \mathtt{I}_{\mathcal{W}_{\mathbf{x}_t}} \quad (constraint) \quad (2.27)$$

$$\mathcal{Q} = \left\{ \ldots \boldsymbol{\mu}_{t-1} \right\} \quad (2.28)$$

Their tracker works on an elliptical window of points, $\mathbf{x} \in \mathcal{W}_{\mathbf{x}_t}$, defined by the track locator $\mathbf{x}_t$, which is extended here to hold the rotation and scale of the elliptical region, as well as position. Their 'wandering' component becomes $E_{\mathcal{P}}$, the 'stable' component is $E_{\mathcal{Q}}$ and the 'lost' component $E_{\mathcal{X}}$. The vector $\boldsymbol{\xi}$ is the null transform (zero translation, zero rotation, unit scaling) and $\mathtt{V}_1$ and $\mathtt{V}_2$ are covariance matrices. The appearance descriptor used in this case is based on the phase responses of a steerable pyramid (Simoncelli and Freeman 1995). They aim for an online system and so the keyframe library is built frame-by-frame, with the appearance used for frame $t+1$ being that from frame $t-1$ approximately warped to match the change in rotation and scale from $\mathbf{x}_{t-1}$ to $\mathbf{x}_t$. Also incorporated are sets of ownership probabilities, $o_{*,\mathbf{x}}$, for $* = w, s, l$, over every location in the elliptical template region, which are chosen each frame to approximately maximize the likelihood of the appearance, with additional constraints that prevent them vanishing in order to minimize the energy. In the above formulation, these are incorporated into the covariance matrices for compactness:

$$\left[\Sigma_{\mathcal{P}}\right]_{\mathbf{x}} = \frac{1}{o_{w,\mathbf{x}}} \sigma_w^2 \quad (2.29)$$

$$\left[\Sigma_{\mathcal{Q}}\right]_{\mathbf{x}} = \frac{1}{o_{s,\mathbf{x}}} \sigma_{\mathbf{x},t}^2 \quad (2.30)$$

It should be noted that the above description has been presented for illustrative purposes and abuses of notation have hidden many details of their actual implementation.

The "wandering, stable, lost" algorithm shows impressive short-time occlusion resistance, coupled with fast adaptation to appearance change. However, it has two fundamental shortcomings: first, the flexibility in its template matching results in a location accuracy that is sufficient for object level tracking, but makes it unsuitable for precise feature point tracking; second, it has a relatively strong dependence on a motion model which means that if the object moves significantly while occluded it will be lost. These are two aspects that must be overcome by a successful feature point tracker.

A different approach is the 'mean-shift' tracker (Comaniciu et al. 2003), which has gained much attention in recent years (Peng et al. 2005, Yang et al. 2005). Mean shift trackers are based on the mean shift gradient descent optimization scheme, which is used to find local modes of some measure of target object presence. Implementations tend to represent the tracking target using histogram-based descriptors. This makes them essentially 'blob' trackers (Collins 2003) that do not have the accuracy demanded by the task at hand. In principle any appearance descriptor could be used, but the local nature of the optimization means that mean shift is inappropriate for any tracking application that strives to reject the assumptions of small inter-frame motion and no occlusions.

There is one final aspect of tracking that needs attension: tracking hypothesis management. This is a way of dealing with the shortcomings of the sequential approach to better approximate global solves and hence perform better in the face of temporary occlusions and repeated texture. It is also known as 'multiple hypothesis tracking'. The main concept is that a hierarchy of alternative track possibilities is stored and updated, rather than strictly maintaining a single possible track. It has been developed probabilistically by several authors, including Reid (1979) and Cox and Hingorani (1994), who have presented efficient algorithms for storing, updating and pruning the stored hypotheses. The context for these authors is, crutially, the real-time processing of streaming input and so they do not have the

off-line luxury of being able to globally consider all possible solutions in one go. My system can be considered as taking their underlying concept to the global extreme.

2.1.6 Detection Literature

[2]In the above literature, the methods used to find the position of the feature in the current frame are based on exhaustive searches, such as SSD. The overhead of performing an appearance evaluation at every pixel in the frame is significant. For example, Jepson et al. (2003) report that half of their per frame computation cost goes on calculating appearance descriptors for the new frame. Even accounting for the improvements in computer processor speeds since their publication, their system would still fall short of even real time tracking rates. The staple tactic is to reduce the search area using the assumption that inter-frame motion is limited. This is often acceptable in online or incremental systems that process frames one at a time. However, to maintain tracking through any significant occlusion, either a very strong motion model must be relied upon or the advantage of reduced search areas must be set aside. The motivation for the tracking system presented in this chapter is the problem of tracking almost unrestricted motion. The need to be able to track arbitrary trajectories means that anything beyond very short term motion models can not be used. This means that, given a fixed track location in a particular frame, the range of all possible motions quickly extends to the image's edges either forwards or backwards in time. My goal is a globally optimum system, where the track energy, E is optimized for the entire track simultaneously, so in addition to long range occlusion handling, full frame searches are also needed to ensure the full extents of all possible motions are available for evaluation. Furthermore, given that the user's first input is effectively random, both spatially and temporally, the system must be prepared for full frame searches in any frame. As such, having to restrict

[2]Here I consider the term 'detection' to be synonymous with what some call 'localization', i.e. the determination that the feature can be seen in a frame *and* the determination of its location. However, I prefer the term 'localization' to be used for the local optimization of a feature's position from an appropriate nearby initialization position.

the search area to achieve desired response speeds is an unacceptable drawback. Obviously, being able to search an entire image while providing rapid full track updates requires very efficient searching strategies. Fast detection is possible, but existing approaches are either not quite fast enough or have unacceptable disadvantages, mainly a need for learning specific target templates. This includes the use of fast classifier techniques, such as the combination of AdaBoost and 'integral images' due to Viola and Jones (2004). While very good detection speeds are achievable with their method, the dependance on a large number of training images and a significant time for the actual training means that they are unsuitable in the interactive tracking context.

The 'support vector machine' of Avidan (2003) also requires a significant object-specific training phase. Extensions by Williams et al. (2003) have created the "relevance vector machine" which can run slightly faster than video frame rate in 'tracking mode' and be trained from a single template. Nevertheless, the training stage still adds unacceptable delays to the interactivity and the actual full-frame detection time is reported to be of the order of one second per frame, well below an interactive target.

Another relevant technology is the group of techniques based on interest points and hashed descriptors such as SIFT (Lowe 2004). Comments at two levels can be made here. At the higher level, i.e. the full interest point detector, there is an incompatibility in that I wish to allow the user to explicitly specify the image location to be tracked. Even with generous interest-point detection thresholds, we can expect no more than of the order of 1000 detections per image. It is therefore unlikely in practice that an interest point is found near the features we wish to track. Systems which use interest points for tracking (e.g. Sivic and Zisserman 2003) depend on having relatively large objects which include several interest points in order to obtain a reliable track. This is even the case for attempts to combine SIFT feature detection with the "wandering, stable, lost" framework (Li 2004), which further falls short by taking several seconds to process each frame. At a lower level, i.e. the descriptors themselves, there are descriptor size issues relating specifically to the minimum practical

SIFT descriptor vector length and problems into this descriptor's applicability for use in the error function, E. This point is discussed in more detail in the system description section below.

2.1.7 Literature Summary

In summary, it is clear that an interactive system, one in which the principle of manual corrections is integral, is vital for practical tracking systems. In real-world applications, the system must be able to generate a track for every piece of footage that it is presented with, so if automation does not quite get the perfect track, there must be provision for the operator to intervene and correct it. No research papers on tracking have actually explicitly addressed the need for interactivity in tracking algorithms and most present algorithms involve processes that preclude the ability to be interactive, beyond the cumbersome 'stop-restart' default.

2.2 System Description

The proposed system fills the gap in the available technology by explicitly separating feature detection from trajectory determination. By divorcing image searching from tracking, arbitrary motion models and occlusions can be dealt with easily. The practical implication is that the options are opened regarding the choice of algorithm for each. The theoretical justification for the separation is two fold: firstly, as has been developed above, a full-frame search is effectively the only way to avoid the false negatives that restricted motion assumptions introduce. Full frame searching is by definition independent of trajectory determination. This naturally flows to the second point that the best way to determine a feature's motion is to consider it over the entire sequence, thereby side-stepping the problem of generating incorrect tracks by making wrong (but perhaps initially imperceptibly so) decisions early in the tracking process, as will always be an issue when restricted to frame-by-frame processing. Globally optimizing a feature track also means that the hitherto inadequately addressed

problem of occlusions can be integrated directly.

This separation makes explicit the practical distinction between the two aspects of feature tracking: detection and trajectory determination. In a perfect scenario, there would be only true positives and true negatives and the two aspects would be the same, but in practice the problems of appearance change, occlusions, image noise and repeated texture mean that false positives and false negatives must be dealt with, hence the practical need for a distinction. However, acknowledging the need for a two stage strategy makes more clear the opportunities for optimizations. Existing object detectors, such as the face detector of Viola and Jones, show that the process of detection can be speeded up greatly by pre-processing: in their case, the conversion of sequence frames to integral images and the training of classifiers tuned for a fast detection scheme. While object-specific learning is not appropriate for this application, context-free preprocessing is possible for high speed template matching. As will be seen, fast global trajectory determination algorithms are also possible.

We can now see that the most powerful approach to tracking is, in fact, the simplest description of the process: find the feature points in every frame of the video and then join them up. Beautifully, this scheme also lends itself perfectly to an interactive system. If the two-stage process can be completed rapidly, then it can be initiated each time a user selects a feature in any frame in an image sequence, returning a complete track ready for acceptance or further tweaking. The problem of appearance change is handled smoothly by collecting all the feature appearances indicated by the user's selections in a patch library. This has the dual advantage of being flexible enough to account for any type of appearance change that can be encountered, be it due to excessive foreshortening effects or extreme non-rigid motion, while not suffering from any of the problems associated with drift and localization imprecision plaguing existing schemes that attempt this.

The rest of this chapter describes an implementation that provides the rapid feedback I demand and all the associated benefits described. A graphical overview is presented in Figure 2.5. The process starts with the generation of fast look-up tree structures for full-

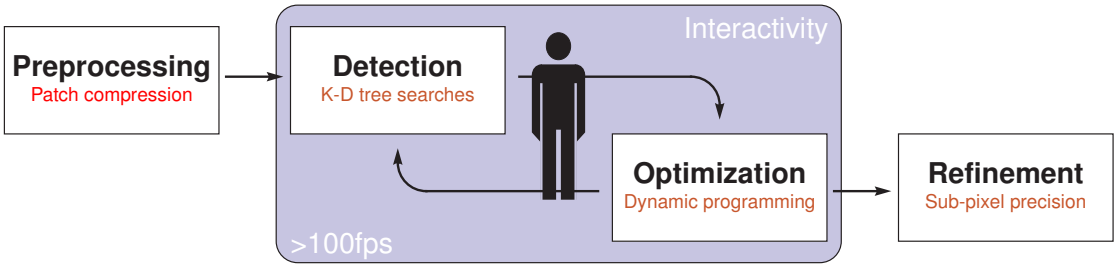


Figure 2.5: **System outline.** The proposed tracking system is split into an off-line preprocessing stage, which includes the generation of the k-d trees, an on-line interactive stage, directed by the user, and a single-pass postprocessing stage to refine the track to sub-pixel accuracy.

frame searching. This first stage is completely automatic and so does not require any user interaction (and as such its speed is not an issue). The next stage is the interactive generation of feature tracks. This is a user-controlled feedback loop: the user selects a point on the feature track and it is added to the keyframe library $\mathcal{Q}$. This initiates a search for the new appearance template across the whole image sequence, followed by a global optimization of the objective function E. If the resulting track is acceptable, the job is done, but if not the user can repeat the process by choosing another keyframe in order to further constrain the track. The final step is a sub-pixel refinement of the track's trajectory.

2.2.1 System Walk-through

Before delving into the details of the component parts of the final implementation, it may be helpful to have in mind a detailed descriptive overview of how the system is used and what processes are being run behind the graphical interface. As an example, imagine that the giraffe sequence from Figure 2.2 is being worked on and the track of the foreground giraffe's eye (as shown in the figure) is desired.

Firstly the frames of the sequence are loaded in turn and each is used to generate a k-d tree holding all image patches in that image. The stored image patches are not actually the full patches as extracted directly from the video frames. Rather, they are patch description vectors, generated by projecting the full image patch vector onto a smaller set of basis

vectors. As such they require less space to store them in memory. This is the sequence preprocessing stage.

Once all the k-d trees have been generated, the system is ready for the interactive tracking stage itself. The user is presented with a view showing the video and is allowed to select any frame of the sequence at any time. To generate the track of the giraffe's eye, the user may start with the first frame and click on the centre of the eye as it appears in that frame. This click initiates a four-step task: (i) an image patch centred on the user's click is extracted from the image and saved to the patch library, along with a note of the frame it came from and its coordinates in that frame; (ii) the extracted patch is projected onto the basis vectors to generate its description vector; (iii) the description vector is used as the search query to find the nearest neighbours in all the k-d trees generated during preprocessing—that is, to find the coordinates of the most similar patches in all the other frames of the sequence: there are now a number of (say ten) candidate coordinates for the location of the giraffe's eye in every frame of the whole sequence; (iv) the globally optimum track, i.e. the best combination of choosing one candidate coordinate from each frame under the objective function E, is determined using dynamic programming—all combinations are assessed efficiently and the global solution is chosen as being the output track. At this point, the user has two options: a) the track can be accepted or b) it can be found wanting. For the former, the system moves on to the postprocessing stage (below). Otherwise, the user needs to give further indications of the nature of the track: any one of the problems described in Section 1.1.1 could have arisen as the giraffe walks across the savanna. In this particular case there are dramatic appearance changes that complicate the tracking process: the giraffe turns its head away from the camera and then turns it back again (presenting significant variations in viewing angle) as well as performing a slow blink (radically altering the colours of the region being tracked). In cases of such major appearance change, it is usual for there to be no candidate coordinates covering the actual location of the target in the offending frames. The user corrects for this by fast forwarding to one of the frames in which these events occur (notable for not having

candidates in the correct place) and clicking again on the giraffe's eye. As above, the click will start the four-step cycle of (i) extract, (ii) transform, (iii) search, (iv) optimize. Note that the search will generate new candidate locations in each frame and these are added to those generated by earlier clicks, thus providing the optimization with more options. After each click, the new track can be reviewed as to whether it is satisfactory. If it is still not acceptable after candidate locations cover the true location in every frame, then performing further patch searches will not help and no more clicking needs to be done. Instead, the parameters of the objective function E can be varied to influence the global optimum, as might well be required in the case of occluded targets, where the occlusion penalty might have to be adjusted.

Finally, a postprecessing stage can refine the track to sub-pixel accuracy. This is required because the k-d tree nearest neighbour searching as implemented returns only patches at integer coordinate positions. Full details of this and the preceding stages are given in the next section.

2.3 Implementation

Resummarizing, in enumerated form, the key steps are:

1. Preprocessing
2. Detection (patch search in every frame)
3. Track optimization
4. Track refinement

Stages 2 and 3 are iterated under the control of the human operator, through feature selection in arbitrary frames, until a satisfactory track is presented. The instantiations of each of these processes will now be addressed in turn.

2.3.1 Preprocessing

Before an interactive session can begin, the sequence to be worked on must be processed to set-up the mechanism for fast full-frame template matching. An important criterion for the preprocessing is to not affect the potential for any feature in the sequence to be tracked. This is why approaches involving learning or training are not appropriate. Furthermore, because this is a template-based system, the result must be equivalent to the standard cross-correlation sweep over the whole image. The first fundamental drawback of the cross-correlation process is that extracting a sub-image from a pixel array is a costly memory read operation, especially for colour images. Therefore, a potentially useful preprocessor step would be to simply extract patches from every location in each frame. This would mean each patch would be contiguous in memory without altering the feature tracking possibilities as required. Obviously, this would drastically inflate the amount of disk space required to hold each image and would not address the second fundamental drawback of the correlation template matching: that comparing every corresponding pixel pair of the template and patch is also a costly calculation. Both problems are solved by converting each extracted patch into a compact appearance description vector. However, a linear search of all appearance vectors is inefficient. Holding them in a tree structure instead means that searching can be completed in logarithmic time. Therefore, the preprocessing stage for this implementation is undertaken with three steps for every frame in the sequence:

1. Extract a patch from every location in the frame.
2. Convert each patch into a compact appearance description vector.
3. Store the vectors in a tree structure for fast lookup in the detection stage.

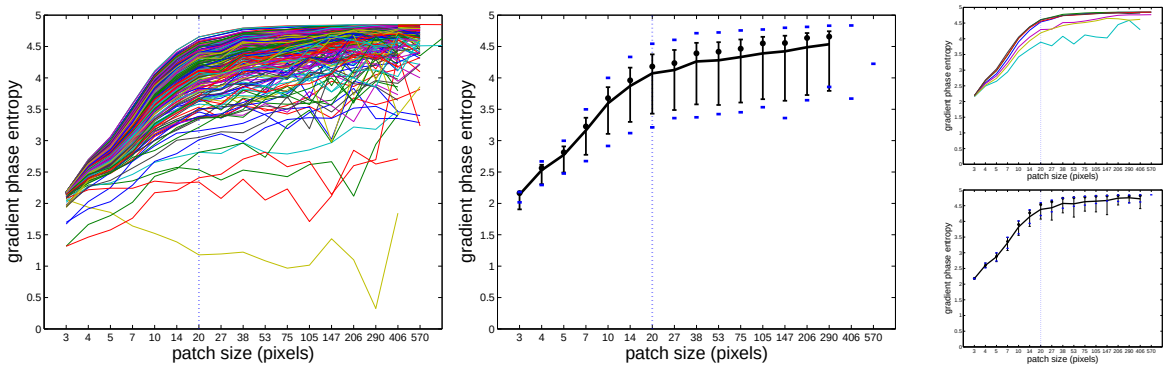


Figure 2.6: **Patch size choice.** Using 20×20 pixel patches is a good choice as it strikes a balance between maximizing information content and minimizing patch area. The data were generated by calculating the average 'gradient phase entropy' for 1000 patches of each patch size (fewer for very large sizes) randomly sampled from example images. The two large graphs show the same data, generated from a set of 209 images (taken from a professional photographer's daily "photo-blog", `www.topleftpixel.com`, during 2006), with the left plot showing the raw data (one curve per image) to give a sense of the correlation, and the central graph summarizing the data. The two small graphs on the right show the same information for an example frame from eight video sequences used in this report. In the summary plots, the graph is drawn through the mean value for each patch size, with an error bar extending to one standard deviation (calculated separately for above and below the mean), black dots are at the median value and blue dashes are at the 5% and 95% ranked points.

Patch Extraction

Moving to the next level of detail, it is clear that the size of the patches must be chosen and fixed at this initial stage. Having a fixed patch size is standard for feature point trackers and is adequate here because the focus is on the precise localization of small feature patches, rather than the less demanding task of region or object tracking (if the object of interest occupies a considerable portion of the frame, it may be tracked by down-sampling the image sequence, or by using a "bag of SIFT" representation (Sivic and Zisserman 2003)). The optimal patch size for feature tracking is a function of texture density and must balance the need for patches to be large enough to be distinctive (minimizing problems of repeated texture), yet small enough to attain high localization accuracy (minimizing the problems of structural appearance change). Instinctively, the right size for feature patches is about 3% of the frame size, which is roughly 20 pixels for PAL image frames. From a universal perspective, it is not clear how to quantitatively assess optimal patch size for feature tracking. In an attempt

to do so, I devised an indicator that is best described by the name 'gradient phase entropy'. I calculated this by working out the Shannon entropy of the angles, quantized in 128 bins, of the Sobel gradients whose magnitude is greater than 0.1% of the maximum intensity value. I looked at the change in 'gradient phase entropy' over a range of extracted patch sizes. The results are shown in Figure 2.6. A spectrum of image types (portraits, cityscapes, close-ups, etc) and sizes (dimensions from about 400 to 1600 pixels) were included, but a consistent pattern appears with a definite shoulder, at around 14–27 pixel square patches, before the curve starts to level off. The presence of a turning point suggests that above a certain patch size, increasing the dimensions does not significantly add to pattern information. This is probably related to texture density being roughly constant in real life footage because of the fractal nature of texture detail in the real world. In the work described in the rest of this chapter, constant 20×20 pixel patches were used. The remaining issue of scale change within a sequence can be accommodated in the system detailed here by absorbing the effects into the appearance change handling. In principle, the extension of the technique to multiple scales is well understood (Lowe 2004), by repeating all computations on down-sampled sequences, with an increase in computational cost of the order of $\sum_{i=1}^{\infty} \frac{1}{2}^{2i} \approx 33\%$.

Data Compression

For my choice of patch size, each patch is a 1200-element vector ($20\times20\times3$) in colour sequences. This is a big number, especially as one will have almost 400,000 patches for every frame of PAL video. A more compact representation is required. It is reasonable to expect that practical (i.e. representatively distinctive) lower dimensional representations exist because natural images, or at least coherent subsets of that class, are likely to approximately lie on low dimensional manifolds (e.g. Donoho and Grimes (2005), Hinton et al. (1997), Varma and Zisserman (2003)).

There has been much work on feature patch descriptors with a variety of motivations. For example, Meltzer et al. (2004) try and learn local manifolds from tracking data for lo-

cation recognition by mobile robots. For objects with discernable boundaries, the work of Belongie et al. (2002) is applicable. In attempts to construct robust tracking systems, colour histograms have also been used, but tend to be too invariant to translation to be accurate. Affine invariance (or partial invariance) for wide baseline matching has been developed by many authors, for example the SIFT features of Lowe (2004). While invariance is attractive, descriptor size is the most important factor and affine invariant descriptors are either too large for this application (see below) or the underlying assumptions are contradicted by the general context of the tracking system being developed here (e.g. planar textures). A common approach for obtaining low dimensional descriptors is employing a rank deficient matrix transform embodying a carefully chosen *filter bank*, such as steerable filters (Freeman and Adelson 1991), although any sensible orthonormal basis is sometimes satisfactory (Jones and Malik 1992). Koenderink and van Doom (1987) presented what they called a *local jet* which contains the responses of a number of gradient operators. The local gradient jets were further modified by Schmid and Mohr (1997) to create descriptors with intensity invariant properties. A review and the precision recall performance of many of the proposed descriptors (in the context of interest point matching) is given by Mikolajczyk and Schmid (2005).

The recommendation of Mikolajczyk and Schmid (2005) is to use SIFT descriptors. However, I decided not to employ SIFT to encode image patches in the per-frame patch look-up trees structures because of considerations regarding memory footprint and assumptions of the descriptor's behaviour relating to patch appearance evolution. This system requires the storage of a very large number of patches (e.g. one for every pixel of the image sequence[3]) and so I regarded descriptor size as the most important attribute for deciding upon which to use. The recommended size of a SIFT descriptor is a 128 element vector and as such is an order of magnitude larger than I require. I also suspect that, as well as the dramatic cut in dimension, the dramatic cut in precision that I eventually impose on the description vec-

[3] A 90-frame PAL sequence requires over 37 million patch descriptors

tors (to further reduce their size in bytes) would drastically reduce the effectiveness of SIFT. Furthermore, the approach developed here exploits the smooth evolution of appearance patch vectors through most sequences (i.e. most frame-to-frame appearance changes are incremental) and relies on this smoothness being transferred to the patch descriptor space. Hashing functions, such as SIFT, cannot guarantee that small appearance changes will result in small changes in the description vectors.

As a secondary comment, Mikolajczyk and Schmid (2005) note that affine invariant steerable filter bank approaches are a good choice among low-dimensionality options. I have employed a straightforward PCA projection filter bank descriptor. An explicitly 'steerable' basis set was not chosen, because an automatic degree of 'tuning' is possible given that in the system being designed, each sequence can use its own filter bank. More significantly, no affine invariance was built into the descriptors used, again to ensure minimal descriptor size. Instead, affine transformations (including scale changes, rotations and foreshortenings) are taken to be appearance changes and thus the system relies on the user to account for them. I fear that even affine invariant descriptors would generate too many false positives, at the very low dimensions used for the descriptors in the final implementation, to provide an adequate candidate list for track optimizations.

The filter bank is constructed via PCA of a large random selection of sample patches taken from the video to be worked on. In this way the most discriminating orthonormal basis set is used to define the low dimensional subspace for the description vectors. This assumes that all the patches from a given sequence lie in or very close to a low-dimensional hyperplane in the full patch vector space. PCA finds an ordered set of basis vectors that are aligned to the directions of greatest variability of a data set, i.e. projecting the data onto the first r vectors of this basis discards the minimal amount of variation in the spread of the data points (in a Euclidean distance sense), hence the term 'most discriminating'. The concepts involved are more rigourously explained in the next chapter. It was found that taking five hundred samples from each frame guaranteed a stable basis for the sequences investigated, although

fewer samples might well be adequate for particular sequences.

A PCA basis can be generated from a set of m dimensional data points, $\mathbf{x}_i$, in this case, by taking the singular value decomposition (SVD) of a matrix $\mathtt{X}$ which has $\mathbf{x}_i$ as its columns. The SVD of a matrix generates two square orthonormal matrices, $\mathtt{U}$ and $\mathtt{V}$, plus a rectangular diagonal matrix, Σ, of *singular values*, which are presented in decreasing order down the diagonal, such that

$$\mathtt{X} = \mathtt{U}\Sigma\mathtt{V}^\top. \tag{2.31}$$

Simply taking the first r columns of $\mathtt{U}$ provides the best linear basis for transforming image patches into patch descriptors in this context:

$$\mathtt{B} = \begin{bmatrix} & & & \\ \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_r \\ & & & \end{bmatrix} \tag{2.32}$$

by giving the best rank-r approximation to $\mathtt{X}$ (see Chapter 3)

$$\mathtt{X} \approx \mathtt{BC} \tag{2.33}$$

The matrix $\mathtt{C}$ holds sets of *filter coefficients*, one set per column, which describe the corresponding vectors in $\mathtt{X}$ (with $\mathtt{C}$ being calculated by multiplying the top left $r \times r$ sub-matrix of Σ with the first r columns of $\mathtt{V}$, transposed). These coefficients are the coordinates of the transformed vectors in the descriptor space defined by $\mathtt{B}$, and are taken as the patch descriptors. Now any m-vector, $\mathbf{x}$, from (or similar to) those in $\mathtt{X}$ can be approximated well by the r-vector, $\hat{\mathbf{x}}$, which satisfies

$$\mathbf{x} = \mathtt{B}\hat{\mathbf{x}} \quad \text{i.e.} \quad \hat{\mathbf{x}} = \mathtt{B}^+\mathbf{x} \tag{2.34}$$

where $\mathtt{B}^+$ is the pseudo-inverse of the basis, since $\mathtt{B}$ is rectangular, i.e.$\hat{\mathbf{x}}$ holds the coefficients of the linear combination of the basis vectors in $\mathtt{B}$ to approximately describe $\mathbf{x}$. It is arranged

here that $\mathtt{B}$ is orthonormal, so in fact $\mathtt{B}^{+} = \mathtt{B}^{\top}$.

An important point that has been omitted above is that the data, $\mathtt{X}$, should be *mean centred* before the SVD operation, otherwise a misleading bias would arise. In the context of choosing $\mathtt{B}$, the SVD can be taken as hyperplane fitting, but with no offset: the fitted hyperplane must go through the origin. Here, the SVD is being used to characterize image patch intensity information, which is always positive, so all data points (vectorized image patches) will be in the first (hyper-)quadrant. This means that, if the data are not mean-centred first, much of the basis would be 'wasted' accounting for this unimportant attribute. Mean-centred data simply have a mean of zero:

$$\bar{\mathtt{X}} = \mathtt{X} - \left[\frac{1}{n}\sum_{i=1}^{n}\mathbf{x}_i\right]\mathbf{1}^{\top}. \tag{2.35}$$

For this system, the data are the sample set of vectorized image patches, $\mathbf{p}_i$, packed horizontally into the data matrix, $\mathtt{P} = [\mathbf{p}_1\mathbf{p}_2\cdots\mathbf{p}_n]$, then mean-centred before the SVD is used to generate a basis. New patches, $\mathbf{p}$, from the same sequence can then be transformed into description vectors using

$$\hat{\mathbf{p}} = \mathtt{B}^{+}\left(\mathbf{p} - \bar{\mathbf{p}}\right) \tag{2.36}$$

where $\bar{\mathbf{p}}$ is specifically the mean of the columns of $\mathtt{P}$.

In practice, collecting patch vectors in a matrix and calculating its SVD directly would be somewhat memory intensive. Therefore, a slightly alternative route is taken. From the definition of the SVD:

$$\bar{\mathtt{P}}\bar{\mathtt{P}}^{\top} = \mathtt{U}\Sigma\mathtt{V}^{\top}\mathtt{V}\Sigma\mathtt{U}^{\top} \tag{2.37}$$

$$= \mathtt{U}\Sigma^{2}\mathtt{U}^{\top} \tag{2.38}$$

i.e.$\mathtt{U}$ can be obtained by the eigenvalue diagonalization of $\bar{\mathtt{P}}\bar{\mathtt{P}}^{\top}$, with the important additional step of reordering the eigenvectors such that the eigenvalues are presented in decreasing

order as required by Σ^2. Expanding out the mean-centred matrices

$$\bar{\mathtt{P}}\bar{\mathtt{P}}^\top = \left(\mathtt{P} - \bar{\mathbf{p}}\mathbf{1}^\top\right)\left(\mathtt{P} - \bar{\mathbf{p}}\mathbf{1}^\top\right)^\top \tag{2.39}$$

$$= \mathtt{P}\mathtt{P} - \left[\bar{\mathbf{p}}\mathbf{1}^\top\right]\mathtt{P}^\top - \mathtt{P}\left[\bar{\mathbf{p}}\mathbf{1}^\top\right]^\top + \left[\bar{\mathbf{p}}\mathbf{1}^\top\right]\left[\bar{\mathbf{p}}\mathbf{1}^\top\right]^\top \tag{2.40}$$

Therefore, considering the individual elements of $\bar{\mathtt{P}}\bar{\mathtt{P}}^\top$

$$\left[\bar{\mathtt{P}}\bar{\mathtt{P}}^\top\right]_{ij} = \left\{\sum_{k=1}^{n} p_k^i p_k^j\right\} - \bar{p}^i\sum_{k=1}^{n} p_k^j - \bar{p}^j\sum_{k=1}^{n} p_k^i + n\bar{p}^i\bar{p}^j \tag{2.41}$$

$$= \left\{\sum_{k=1}^{n} p_k^i p_k^j\right\} - \frac{1}{n}\sum_{k=1}^{n} p_k^i\sum_{k=1}^{n} p_k^j - \frac{1}{n}\sum_{k=1}^{n} p_k^j\sum_{k=1}^{n} p_k^i + \frac{n}{n^2}\sum_{k=1}^{n} p_k^i\sum_{k=1}^{n} p_k^j \tag{2.42}$$

$$= \left\{\sum_{k=1}^{n} p_k^i p_k^j\right\} - \frac{1}{n}\sum_{k=1}^{n} p_k^i\sum_{k=1}^{n} p_k^j \tag{2.43}$$

where x_k^i is the i^{th} element of vector $\mathbf{x}_k$. This formulation can conveniently be achieved incrementally

$$\bar{\mathtt{P}}\bar{\mathtt{P}}^\top = \left\{\sum_{k=1}^{n} \mathbf{p}_k\mathbf{p}_k^\top\right\} - \frac{1}{n}\bar{\mathbf{p}}\bar{\mathbf{p}}^\top \tag{2.44}$$

that is, while going through an image sequence collecting sample image patch vectors, instead of concatenating them for $\mathtt{P}$, one can equivalently keep two fixed-size entities: a matrix $\mathtt{S}$ and a vector $\mathbf{s}$, both keeping running summations of the sample patches. They must start full of zeros and then are updated by each vector sample thus

$$\mathtt{S} \leftarrow \mathtt{S} + \mathbf{p}_i\mathbf{p}_i^\top \tag{2.45}$$

$$\mathbf{s} \leftarrow \mathbf{s} + \mathbf{p}_i \tag{2.46}$$

which allows the following algorithm for the determination of $\mathtt{B}$:

$$\text{eigenvectors, eigenvalues} \leftarrow \text{eigenvalue decomposition of } \left(\mathtt{S} - \frac{1}{n}\mathbf{s}\right) \tag{2.47}$$

$$\mathtt{B} \leftarrow \text{first } r \text{ eigenvectors, ordered by decreasing eigenvalue} \tag{2.48}$$

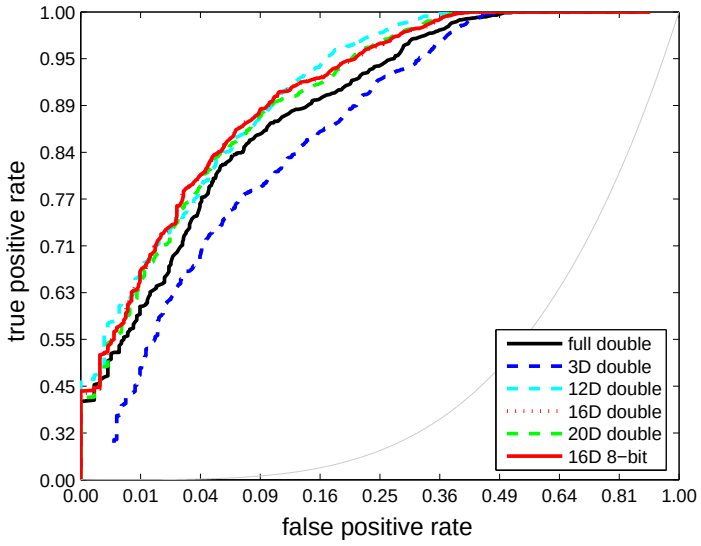


Figure 2.7: **Feature description vector dimension I.** ROC curves showing the performance of PCA-coefficient description vector representations of patches compared to using the norm of the full patch residual. A log scale has been used to exaggerate the differences between the plots; the grey line represents expected random performance. This representation is based on five ground-truth feature tracks in the walking giraffe sequence. Feature patches were matched against all the other patches taken from each feature's track and a set of random 'background' patches. Matching was performed using the full 20×20 patch and the feature vector descriptors over a range of dimensionalities, both with and without 8-bit quantization. Only a selection of the results are presented to increase clarity. The other results follow the trend seen here: very low dimensionalities perform the worst, increasing to an optimum around 16 dimensions and then slightly falling back with higher dimensions.

All that remains is the choice of r, the dimensionality of the descriptor. Figures 2.7 and 2.8 show the results of tests performed over a variety of dimensionalities, which suggest that beyond a minimum rank, all descriptors perform well. The low dimensional optimum suggests that having a descriptor of about 16 dimensions performs best. In fact, in receiver-operator performance, 16 dimensional descriptors perform better than the matching of the actual image patches themselves. It could that the smoothing applied by the transformation process aids in discrimination in some of the border-line cases, by overcoming the noise.

To further reduce the memory size of the descriptors, they are quantized from double precision to 8-bit integers. To maximize the use of the 8-bits for each frame, the upper and lower bounds of the points in the projection coefficient vector space are found for each k-d

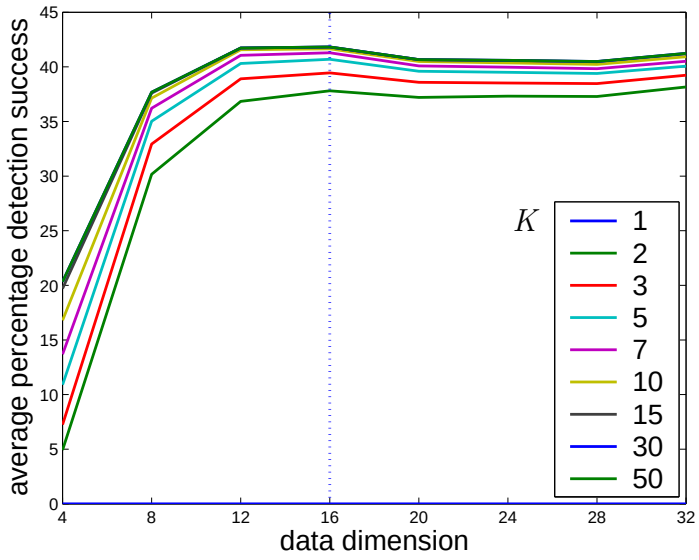


Figure 2.8: **Feature description vector dimension II.** The average detection success for five features tracked through the walking giraffe sequence as descriptor dimension is varied. For each feature, the detection system described below was used to search every frame of the sequence in which the feature was visible. Success was measured as the percentage of frames in which the localization was within one pixel of ground truth (thought to have a standard deviation from the real ground truth of about half a pixel). For each feature, the process was repeated using every feature appearance patch from the sequence, thus averaging over a simulation of a user selecting a feature in an arbitrary frame and demanding a track. Each curve represents a different number of nearest-neighbour matches, K, with success being counted if any matches localize to within one pixel. The low dimensional optimum is a 16 dimensional descriptor.

tree separately and used to normalize the data before quantization. However, this does mean that each dimension of the patch coefficient vectors stored in each k-d tree has been scaled separately. Theoretically, this is a problem because a descriptor generated for one frame might not be directly comparable to those of another, if the upper and lower bounds are different, thus affecting the k-d tree searches. However, Figure 2.7 shows that, surprisingly, in practice the quantization has a negligible effect, and that comparing 16 element vectors using either double precision or 8-bit integers results in almost identical performance. This was also seen in all other dimensionalities too.

The efficiency of this process is hinted at in the way that it captures many of the visual attributes of any particular sequence, including its predominant colour scheme and important

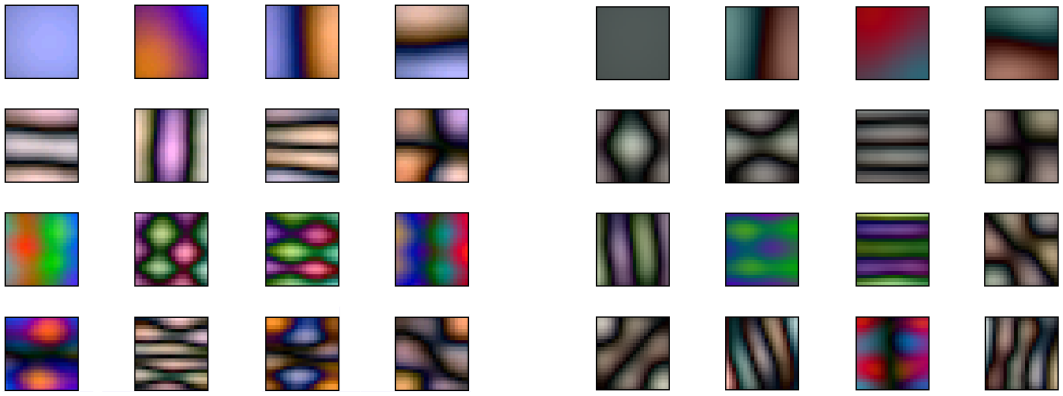


Figure 2.9: **PCA bases.** Two example basis sets, viewed as patches. For maximum impact the absolute value (magnitude) of the normalized pixel values is displayed. On the left are the 16 basis patches for the very blue ICCV'03 sequence (Figure 2.14), and on the right an indication of the many horizontal edges seen the 'reading' sequence (Figure 2.13). All patches from the respective sequences are treated by the descriptors as linear combinations of these basis filters.

geometric themes (Figure 2.9).

Geometric Indexing

The idea of the preprocessing stage is to create data structures with which it is quick to recall information. After projection onto the filter basis, we view each $W \times H$ input frame as a collection of 16-dimensional vectors $\{\hat{\mathbf{p}}_1, ..., \hat{\mathbf{p}}_{WH}\}$. The problem of searching the image for a query patch $\hat{\mathbf{q}}$ is reduced to a nearest-neighbour search in $\mathbb{R}^{16}$, and in particular it would be good to efficiently retrieve the locations of the M vectors most similar to $\hat{\mathbf{q}}$. When searching for a known vector, tree structures give an exponential time complexity speed up over exhaustive linear search and so provide an attractive solution. Here, k-d trees (Arya and Mount 1993) are employed as a general and flexible option for k dimensional vector lookups. K-d trees are but one of a variety of geometric data structures which accelerate closest-point computations. Other data structures (e.g. BKD-trees (Procopiuc et al. 2002), KDB-trees (Robinson 1981), Tree-Structured VQ (Chen et al. 1997), SS+-trees (Kurniawati et al. 1997), cell trees (Gunther 1986), hB-trees (Lomet and Salzberg 1990) and R-trees (Greene 1989, Guttman 1984, Jagadish 1990, Kamel and Faloutsos 1994, Sellis et al. 1987))

could have been used, but a detailed investigation was not undertaken. The basic k-d tree is simple, general and very adequately proves the concept of image preprocessing for fast system execution. Every video clip requires its own k-d tree set to be created, with one k-d tree per image. The preprocessing computations can be performed overnight or when the celluloid film is being digitally scanned. Each k-d tree holds at its leaves all possible patches extracted from its associated video frame. Template matching then becomes a fast k-d tree lookup.

The basic design of the k-d tree is relatively simple: it is a binary tree structure with every internal node, n_i having exactly two children, $\mathsf{n}_i^{\mathrm{L}}$ (left) and $\mathsf{n}_i^{\mathrm{R}}$ (right). Each node also has associated with it a threshold t_i and dimension number d_i. To discover in which leaf a vector, $\hat{\mathbf{q}}$, resides, the tree is traversed from the root node, n_0 and down[4] through the children with choices as to which child to pick determined by inspecting the value of the $\mathsf{d}_i^{\mathrm{th}}$ dimension of $\hat{\mathbf{q}}$ and comparing it to the threshold t_i. If $q_{\mathsf{d}_i} < \mathsf{t}_i$ then the next node to visit is $\mathsf{n}_i^{\mathrm{L}}$, whereas if $q_{\mathsf{d}_i} \geq \mathsf{t}_i$ then $\mathsf{n}_i^{\mathrm{R}}$ is chosen. This continues recursively until a leaf node is reached. In this way, the k-d tree effectively applies recursive hyperplane partitions to the data, one at each node of the tree, perpendicular to the chosen dimension, d_i, at that node. The leaf node that is reached is the hyperspace partition in which the query vector belongs. All the vectors stored in that leaf are then compared to find which is the nearest to the query. However, if $\hat{\mathbf{q}}$ is near to the partition boundary, the actual nearest neighbour in the tree might reside in a neighbouring partition, and would therefore be held in a different leaf node. To check for this possibility the descent route down the tree is *backtracked* and any partitions for which the boundary is within the distance to the current nearest neighbour are also investigated. Backtracking continues until the chosen neighbour is closer than any unsearched partitions. Appendix A provides more details.

A k-d tree is constructed for each frame, with the exact implementation designed very carefully for simultaneously maximizing lookup speeds and minimizing the size on disk.

[4]Despite the name, I will refer to the root node as being at the top of the tree, with the leaves below it.

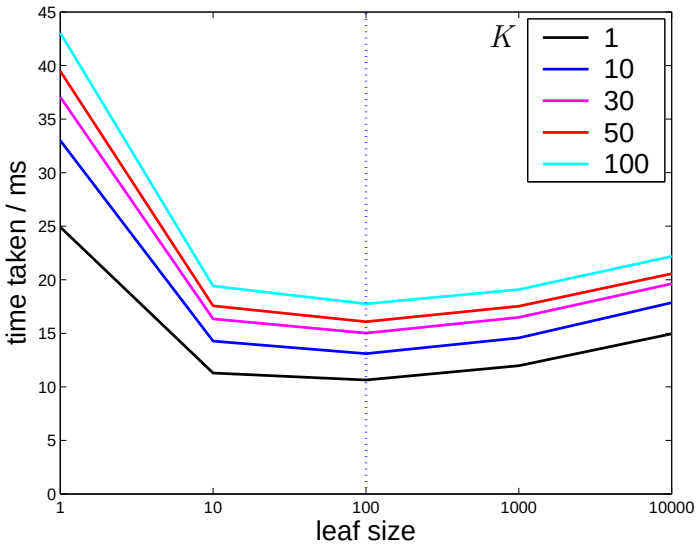


Figure 2.10: **Tree leaf density.** Average search times for K nearest neighbours in frames from the four video sequences used in this chapter. For each sequence, one hundred query points were taken randomly from across the entire sequence and searched for in every frame of that sequence. This was repeated for k-d trees built with the logarithmic range of leaf node data counts shown.

This is achieved, in part, by storing balanced k-d trees of constant depth h, chosen so that the average number of patches being stored at each leaf is about 100. Balanced binary trees are those for which at any internal node, half of all the vectors in leaves below that node are found under one child and the other half under the other child. A tree's depth is the number of internal nodes that are visited when traversing the tree from the root node to the leaves. The depth of fully balanced trees is constant across all leaves. Having an order of magnitude of leaf density around 100 was seen to give the best search times, as shown in Figure 2.10. This determines h for a frame resolution to be $h = \lceil \log_2(WH/100) \rceil$. When constructing the tree, the dimension indices and thresholds at each node are set to achieve an evenly balanced result (although it is not necessary to achieve perfect balance). The dimension index for a given node is chosen by looking at the variance of the data that will be below that node, separately for each of the 16 dimensions. The dimension with the highest variance is chosen for discrimination at that node (so this is like an axis re-ordering PCA with the first 'principal component' being chosen). The threshold for a node is taken as the median value of the data

in the chosen dimension. Using the median satisfies the requirement for leaf count evenness.

Another implementation feature that helps with the space/lookup efficiency is the layout of the tree in memory. A pointer-based data structure is not required, as the deterministic layout of a full (balanced) binary tree means that it is quick and easy to locate node or leaf attributes from within a linear array. By numbering the nodes sequentially, breadth first (starting at each level with the 'left' most node), with the root node having $i = 1$, the index of the left child of node i is simply $2i$ and the right node is $2i+1$. Therefore, the position of a node's attributes in linear arrays can be calculated directly, without having to use extra pointer attributes. Note that this indexing is equivalent to creating a binary vector, of length equal to the depth of a node, with a `1` as the first entry (representing the root), followed in order by a `0` for every left branch taken and a `1` for every right branch. Reading the binary vector as a number gives the index. For the leaves a separate array of information is held, because the information at leaves is different to the internal nodes. This means that the array index for the leaf nodes starts again at one for the left most leaf. However, the leaf indices can still be generated procedurally by using the same process as above, but labelling the root node zero instead of one, or equivalently, using the above system then halving and adding one.

Once a query has traversed the tree and arrived at a leaf, there are still one hundred possible matches. These are compared in a linear exhaustive mini-search. As such, all the data vectors held at a given leaf node are guaranteed to be accessed, so after building the tree, the patch descriptors are rearranged in `P` so that all descriptors at leaf number one come first, then those for leaf number two, etc. This makes them contiguous in memory and so maximizes memory access efficiency (cache coherency). This also means the data stored at each leaf node (in the leaf node array) need consist of only one number: the column index of the first description vector in the re-ordered `P`. However, it also means that one more extra array is needed. It is necessary to store the original column index of each descriptor in the un-ordered `P`, from which the coordinates in the associated video frame can be reconstructed.

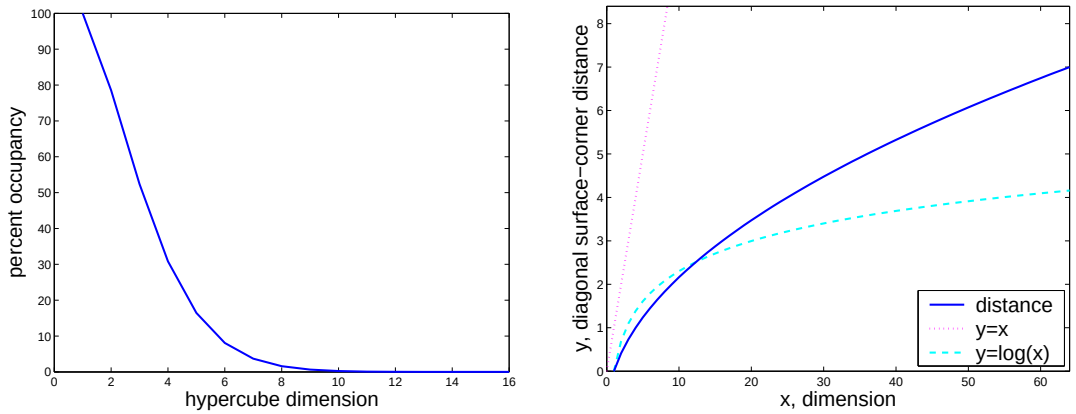


Figure 2.11: **Hypersphere inadequacies.** Left: the hypervolume occupied by unit radius hyperspheres in spaces within a range of dimensions as percentages of their bounding hypercubes (edge length 2). This becomes negligible at around dimension 10. Right: a plot of the distance between a corner of the hypercube and the nearest point on the surface of the bounded (unit radius) hypersphere. This simply reflects the fact that the corner of a hypercube gets further away from the center the larger the dimension. Linear and logarithmic plots are given for comparison. The rate of change of this measure becomes worse than logarithmic after about dimension 4 and twice as large after dimension 16. These two geometric features relate to the theoretical inefficiency of k-d trees as the data dimension increases.

Altogether, this creates a data structure for which the storage cost is no more than 24 bytes per pixel (it can be less for images whose pixel count is below the next power of two). Although this is eight times larger than a 'TrueColour' image, full-frame template matching using k-d trees can be over 200 times faster than standard cross-correlation across the same image for PAL resolutions.

A comparison of all the competing implementation details and other technical descriptions are given in Appendix A.

It is worth noting that k-d trees are sometimes considered inefficient in spaces of even moderately high dimension (Lowe 2004): this is probably due to the fact that for higher dimensions all points appear to roughly equally very far away from each other (Figure 2.11 tries to give a geometric idea of this). Although the remaining small differences in distances are enough for good discrimination, they are misleading when it comes to k-d tree partition occupancy. However, patch search in images is a special case. The combination of the

PCA projection of the data and natural appearance clustering places it in an ideal geometric configuration, meaning that, on average, lookups benefit from more than a threefold speedup over efficient exhaustive search in $\mathbb{R}^{16}$.

2.3.2 Image Searching

Searching the images for an input patch $\mathbf{q}$ is simply a matter of querying the k-d trees for each image in turn. The top M matches are retained from each search, i.e. the M nearest neighbours to the input patch. As shown by Figure 2.8, setting M to 30 achieves almost maximal performance and so that number was used throughout the experiments. Returning multiple nearest neighbour matches is easy to implement efficiently for k-d trees by maintaining an ordered list structure for the matches as they are compared (see appendix for details).

In practice small transformations (including translations) of a patch remain close to the original in patch space. This is suggested by the Taylor series expansion of patch transformations and will be elaborated on below. The real implication of this is that the output of the image searches present spatial clusters of matches appearing in the nearest neighbours. With a convolution-style strategy, a non-max suppression technique is often used to prune all adjacent matches above the threshold, leaving only the best match in each group. Non-max suppression is less straightforward for k-d tree searches because of the sparseness of the search output data.

I devised a clustering technique to realize the necessary non-max suppression for sparse coordinate data. Given a set of coordinates, each is taken in turn. The first creates its own cluster. Subsequent points are compared to every existing cluster. A point–cluster comparison involves calculating the distance of the point to each point in the cluster, stopping as soon as any point is seen to be less than the cluster threshold distance. If a new point is seen to be part of two or more existing clusters, they are merged. An optimal k-d tree nearest neighbour implementation returns the results in decreasing order of similarity, i.e. with the

best first. As such, the best match in each cluster is simply the first point in that cluster, thus giving an efficient sparse non-max suppression algorithm. I used an early bailout threshold of $\sqrt{2}$ pixels, and discarded the non-maximal matches. The final search result is generally about $\hat{M} \approx M/8$ candidate matches for each frame.

2.3.3 Track Path Optimization

At this stage there are, on average, not many more than $\hat{M}$ matches in each frame. Every possible candidate track, $\mathcal{T}_c$, takes one match from each image. The trajectory determination stage tries to find the best track (of the combinatorially explosive $\hat{M}^F$ candidate tracks), i.e. the one that minimizes the track error, $E(\mathcal{T}_c)$. Fortunately, this problem is solved by dynamic programming (DP), which finds the optimal choices of matches over the whole sequence, but searching over a greatly reduced number of candidates. DP finds the global optimum in $O(\hat{M}^2 F)$ time.

The use of dynamic programming has appeared many times in computer vision: it has been used to optimize trajectory segmentation (Mann et al. 2002), optical flow (Quenot 1992), radial contour tracking (Chen et al. 2001), template matching (Liu and Wang 2000) and full active contour tracking (Pardas and Sayrol 2000). It has also been used for tracking affine deformations (Yao and Li 2004), but only for templates containing many individual features. The only instance of its use for trajectory optimization of which I am aware, is in work by Dreuw et al. (2006) (undertaken in parallel with that described here) that aims to perform off-line analysis of sign-language footage for automated recognition. Although their cost function has similarities to what is presented here, they are not concerned about processing time and so the integration they employ is not entirely comparable.

Before discussing the actual track quality function and how it will be optimized by the dynamic programming routine, we need to look at the priors on inter-frame motion and appearance change that reflect beliefs as to what denotes a good track. Firstly, it is reasonable

to not expect a feature in frame t to appear far from its position in frame $t+1$. This is a common assumption, that, as explained at length in the previous section, hinders greatly a number of tracking approaches. However, that is because they are incorporated at the detection stage rather than the evaluation stage as proposed here. Other simple steps will also be taken to avoid the other pitfalls that have been identified in the criticisms raised above. Secondly, appearance changes tend to be gradual; at standard video frame rates, the image of a feature does not radically alter from one frame to the next. By the Taylor expansion of the linear transformation of image patches, it can be seen that all incremental patch warpings, such as those that occur from frame to frame due to rotation or small non-rigid deformations of the feature patch image, lead to smooth trajectories through 'patch space.' In turn, any conformal mapping[5], applied to the patch to transform it into a description vector, will maintain that smoothness and so the trajectory through 'description space' will be smooth as well. Therefore, we can say that $\mathbf{x}_t$ and $\hat{\mathbf{p}}_t$ move smoothly through their respective spaces. I implement these smoothness constraints using weak constant position models, as will now be seen.

Defining the Error Functions

The track evaluation function, E, will now be taken through a few small alterations as it is adjusted to fit into the environment of this interactive tracker. I begin with a reminder of the function:

$$E(\mathcal{T},\mathcal{I},\mathcal{Q}) = \underbrace{E_{\mathcal{X}}(\mathcal{X})}_{\substack{\text{motion}\\ \text{smoothness}}} + \underbrace{E_{\mathcal{P}}(\mathcal{P})}_{\substack{\text{appearance}\\ \text{smoothness}}} + \underbrace{E_{\mathcal{Q}}(\mathcal{T},\mathcal{Q})}_{\substack{\text{appearance}\\ \text{fidelity}}} + \underbrace{E_{\mathcal{I}}(\mathcal{T},\mathcal{I})}_{\substack{\text{observation}\\ \text{coherence}}}. \tag{2.49}$$

Discretization. The first alteration relates to the use of match candidates to define the domain of possible tracks. Each of the $\hat{M}$ candidates in each frame is assigned a label (for example, by enumeration) that is then referenced by the variable l_t. It is now possible to refer

[5]Obviously, discontinuous transforms for the creation of description vectors, e.g. hash functions, do not hold up this assumption.

to the candidate locations using $\mathbf{z}(l_t)$ and the associated appearances with $\mathbf{u}(l_t) = [\mathtt{I}_t]_{\mathcal{W}_{\mathbf{z}(l_t)}}$. All the label variables, one for each frame, are held in the label set, $\mathcal{L}$. The full optimization is then the determination of the latent track variables (trajectory and appearance) that maintain a smooth motion while keeping as close to the candidate observations as possible:

$$E_{\mathcal{X},t} = \|\mathbf{x}_t - \mathbf{x}_{t-1}\| \quad \textit{motion smoothness} \tag{2.50}$$

$$E_{\mathcal{P},t} = \|\hat{\mathbf{p}}_t - \hat{\mathbf{p}}_{t-1}\| \quad \textit{appearance smoothness} \tag{2.51}$$

$$E_{\mathcal{Q},t} = \min_i \|\hat{\mathbf{p}}_t - \hat{\mathbf{q}}_i\| \tag{2.52}$$

$$E_{\mathcal{I}} = \min_{\mathcal{L}} \sum_{t=1}^{F} \{\|\hat{\mathbf{p}}_t - \hat{\mathbf{u}}(l_t)\| + \|\mathbf{x}_t - \mathbf{z}(l_t)\|\} \tag{2.53}$$

In addition, the hard constraints that

$$\mathbf{x}_i = \mathbf{y}_i \quad \forall i \in \mathcal{N} \tag{2.54}$$

must be satisfied, although this has not been explicitly expressed in the quality functions because it is much easier to implement than understand when integrated into the formula.

An optimization over the continuous latent variable spaces is too costly to be performed in an interactive context and so it is approximated with a combination of an on-line optimization over the label space and the off-line postprocessing refinement stage. Switching the minimization to being over a discrete label set allows for the incorporation of DP and the associated efficiencies. The implication is that the latent variables are effectively discarded in favour of variables that are hard constrained to the candidate observations: $\mathbf{x}_t = \mathbf{z}(l_t)$ and $\hat{\mathbf{p}}_t = \hat{\mathbf{u}}(l_t)$. We now have

$$E(\mathcal{L}, \mathcal{I}, \mathcal{Q}) = E_{\mathcal{X}}(\mathcal{L}, \mathcal{I}) + E_{\mathcal{P}}(\mathcal{L}, \mathcal{I}) + E_{\mathcal{Q}}(\mathcal{L}, \mathcal{I}, \mathcal{Q}) \tag{2.55}$$

where $\mathcal{I}$ has been included in all functions to indicate the fact that only observed variables

are being used. The component terms become

$$E_{\mathcal{X},t} = \|\mathbf{z}(l_t) - \mathbf{z}(l_{t-1})\| \tag{2.56}$$

$$E_{\mathcal{P},t} = \|\mathbf{u}(l_t) - \mathbf{u}(l_{t-1})\| \tag{2.57}$$

$$E_{\mathcal{Q},t} = \min_i \|\mathbf{u}(l_t) - \hat{\mathbf{q}}_i\| . \tag{2.58}$$

Note that the hard constraint on keyframe matching becomes implicit by reducing the available labels to one for $t \in \mathcal{N}$, forcing the user selected point to be chosen in those frames.

It is also worth noting that higher order motion cost functions can be employed, but each extra position term in $E_{\mathcal{X},t}$ increases the time complexity of the DP by a factor of $\hat{M}$, e.g. an acceleration term, $\|\mathbf{x}_{t-1} - 2\mathbf{x}_t + \mathbf{x}_{t+1}\|$, would require an order $O(\hat{M}^3 F)$ computation.

Occlusion handling. No mention has yet been made as to how to deal with occlusions. Explicitly incorporating occlusion handling into this system requires another modification of the track evaluation function. For this, the possible labels for every non-keyframe t are extended to always include an 'occlusion appearance', which if chosen signifies that the feature is not visible in that frame. The terms of E can be updated as appropriate and an additional term is provided to penalize changes in occlusion state:

$$E(\mathcal{L}, \mathcal{I}, \mathcal{Q}) = E_{\mathcal{X}}(\mathcal{L}, \mathcal{I}) + E_{\mathcal{P}}(\mathcal{L}, \mathcal{I}) + E_{\mathcal{Q}}(\mathcal{L}, \mathcal{I}, \mathcal{Q}) + E_{\mathcal{L}}(\mathcal{L}). \tag{2.59}$$

with

$$E_{\mathcal{X},t} = \|\mathbf{z}(l_t) - \mathbf{z}(l_{t-1})\| \tag{2.60}$$

$$E_{\mathcal{P},t} = \begin{cases} 0 & l_t = \textit{occlusion} \\ \|\hat{\mathbf{u}}(l_t) - \hat{\mathbf{u}}(l_{t-1})\| & \textit{otherwise} \end{cases} \tag{2.61}$$

$$E_{\mathcal{Q},t} = \begin{cases} 0 & l_t = \textit{occlusion} \\ \min_i \|\hat{\mathbf{u}}(l_t) - \hat{\mathbf{q}}_i\| & \textit{otherwise} \end{cases} \tag{2.62}$$

$$E_{\mathcal{L},t} = \begin{cases} \kappa_r & l_{t-1} = l_t = \textit{occlusion} \\ \kappa_o & l_{t-1} \neq l_t = \textit{occlusion} \\ 0 & \textit{otherwise} \end{cases} \tag{2.63}$$

Appearance comparisons are suppressed during assigned occlusion periods, because the appearance $\hat{\mathbf{u}}$ is generated from the image frame itself and if the feature is occluded, it is not applicable to refer to the image to obtain a representation of the feature's appearance for that frame. Note that the evolution of the locations $\mathbf{x}_t$ will still be evaluated, even during occlusions. This means that there is an implicit cost associated with occlusions that penalizes the motion expected to have occurred while the feature was not visible. The new function is the explicit occlusion cost in the form of an occlusion state penalty function, $E_{\mathcal{L}}$, which has four modes: staying visible and becoming visible are unpenalized; staying occluded costs κ_r; and becoming occluded costs κ_o. In this way, any occlusion has a base cost and longer occlusions are penalized more heavily than shorter ones. This state transition cost is similar to that in Huang and Essa (2005). The contribution here is to show how this optimization can be computed efficiently, with a strong likelihood of finding the desired optimum, even in the presence of significant occlusion.

frame number:	1	2	3	4	5	6	7	8	...
input patches:	inf,nan	0.2, 4	inf,nan	inf,nan	inf,nan	inf,nan	1.5, 2	inf,nan	...
matches:	0.2,nan	inf,nan	0.4, 1	0.5, 2	1.0, 3	1.2, 2	inf,nan	1.8, 1	...
	0.2,nan	inf,nan	0.3, 1	0.6, 2	0.9, 2	1.1, 3	inf,nan	1.9, 1	...
	0.2,nan	inf,nan	0.4, 1	0.8, 5	inf,nan	1.2, 3	inf,nan	1.6, 1	...
	0.1,nan	inf,nan	inf,nan	0.7, 3	inf,nan	inf,nan	inf,nan	inf,nan	...
occlusion:	0.5,nan	inf,nan	0.7, 1	0.9, 5	1.1, 5	1.4, 2	inf,nan	2.0, 1	...

Figure 2.12: **Dynamic programming example.** The dynamic programming table is filled one column at a time from left to right. In this example, each element of the table holds a pair of numbers: the error of the optimal path up to that match in that frame and the number of the match in the previous frame on that path. Note that this approach naturally deals with multiple input patches, varying numbers of matches per frame and occlusions. Here, the best track is $\{4,1,2,3,2,2,1,4\}$.

Dynamic Programming.

Dynamic programming is a way of reducing the search space for problems where one of a number of states must be chosen for each of a series of successive stages. Here, the stages are the frames of the sequence and the states are the possible matches (detections) in each frame. It can be implemented by filling a table whose rows and columns represent the matches (states) and frames (stages) respectively, as shown in Figure 2.12. By the principle of optimality (Bellman 1952), the table's entries may be calculated one column at a time, forwards through the frames, so that each entry holds the error of the optimal track, ending on that match, through the whole sequence up to that frame. To calculate the entry of a particular match in a given frame (column), the following steps are taken:

1. cycle through each match of the previous frame,
2. add to the value of that previous match, the cost of extending the track from that match to the current one,
3. store the lowest error and the index of that chosen match in the current frame's table entry.

At the end of the sequence, the global optimum is the track whose error in the final column is lowest. To determine that optimal track, the indices stored in each table entry are used to retrace the best options back through the sequence.

The possibility for occlusion, $l_t = \textit{occlusion}$, is incorporated by adding an extra row to the table, thus making a 'match' to the state of occlusion possible in every frame. However, there can be no $\mathbf{x}_t$ explicitly associated with this occlusion state, as there is for all the candidate matches. This means that $E_{\mathcal{X},t}$ can not be evaluated during an occlusion. Therefore, to implement the minimization of $E_{\mathcal{X},t}$ through occlusions, there is actually a 'reappearance cost' calculated when a change from occlusion to visible arises. This cost is the minimum of $\sum E_{\mathcal{X},t}$ over the duration of the occlusion and is equivalent to making the trajectory a straight line from the last visible position to the point where the feature reappears. Note that for higher order (e.g. acceleration) motion cost functions, the minimum energy occlusion trajectory would be the appropriately ordered spline-like path.

I set κ_r to be $0.4\kappa_o$ and so there are only three coefficients for the user to choose values for: $\lambda_{\mathcal{X}}$, $\lambda_{\mathcal{P}}$ and $\lambda_{\mathcal{L}}$[6]. The calculation of the optimal track can be so fast that these can be adjusted by hand to get the best results; this furthers the interactive element of this system. In practice, playing with the system has found that good values can be found with little effort in very few iterations. It is mainly the tweaking of the occlusion cost, λ_o, that is most often needed.

2.3.4 Track Refinement

The k-d tree can only return matches from those locations for which it stores patches. Therefore, the results at this stage are pixel-aligned. To obtain sub-pixel accuracy a final correlation-based localization can be employed for track refinement. For this system, a KLT affine optimization from every point on the visible track is run, using the associated patch from the patch library: $\mathbf{q}_{l_t}$. The optimization of l_t in each frame becomes synonymous with the choice of keyframe appearance, because every match is already associated with a query patch in $\mathcal{Q}$, i.e. the one that was used to generate that match.

[6]The degree of freedom in κ_o is absorbed in that of $\lambda_{\mathcal{L}}$ in the full evaluation function.

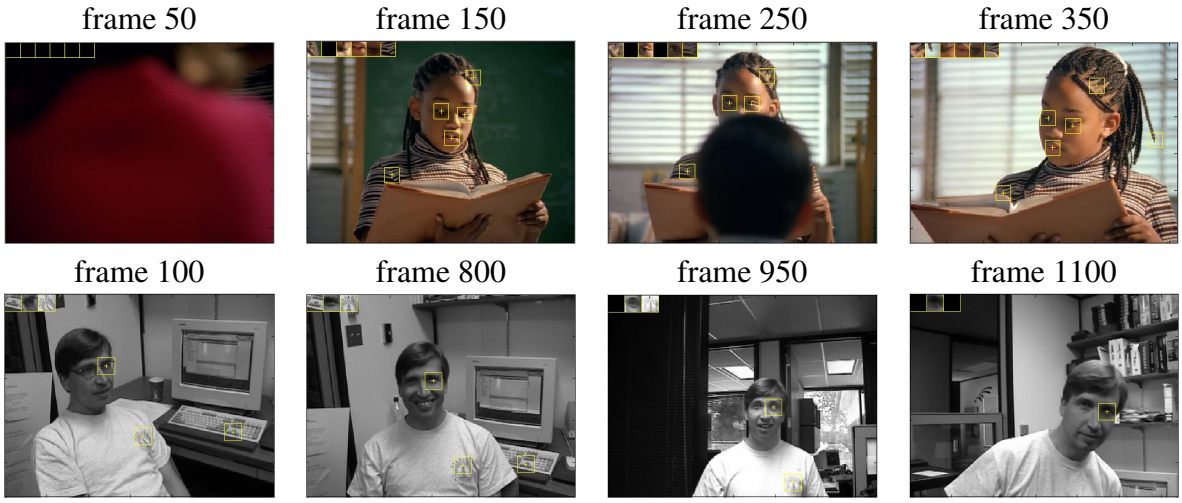


Figure 2.13: **Results I.** Top row: four frames from a 464 frame sequence (360×270) of a girl reading with six tracks. Detection rates averaged 135fps. Bottom row: three example tracks through the 1145 frame 'Dudek' sequence (320×240). Detection consistently ran at 200fps.

2.3.5 Adaptive Tracking

Finally, we return to the iterative nature of the user's feedback in the generation of a feature track. The above implementation description briefly overlooked the full interactive process, concentrating on how to deal with a single template patch. It is readily possible for it to be sufficient for the user to select a single frame in which to click on the target feature, i.e. the perfect track after the first selection. For example, other than the obvious easy feature tracks, in situations where the patch appearance varies cyclically (e.g. a face occasionally looking off to the side of the camera), a single selection can be very effective, because the frames where the match is good tend to "lock down" the track, and the patch smoothness term overrides the difference between even quite distant frames and the single template. However, many sequences have more abrupt or more severe appearance change, for example an eye which blinks. In these cases, the $\hat{M}$ candidate patches in a given frame may not include the correct match, in which case the track will not be complete and the user will obviously need to supply additional input. All additional user clicks are treated in the same way as the first: the new template is added to the patch library, $\mathcal{Q}$; the new template is used to search every

frame for candidate matches; and DP is used to update the optimal track. However, having $|\mathcal{N}|\hat{M}$ matches in each frame (i.e. a number of candidates linear in the number of user clicks) is not desirable, or indeed required. It is not desirable because the DP algorithm (for the evaluation function defined above) is sensitive to the number of candidate matches, having a time complexity quadratic in the number of possibilities per frame. Therefore, match pruning is performed as successive patches from $\mathcal{Q}$ are searched for. For the Matlab implementation used here, keeping the number of candidates per frame to about ten worked well. To decide which matches to keep and which to discard, both position and appearance can be compared. For example, it is not necessary to have similar looking patches very close to each other, so all but one of a group (most likely a pair) of neighbouring candidates should be removed, in the same way sparse non-max suppression is performed immediately after searching. Either the search distance threshold can be used or it could be expanded to be the convergence basin of the refinement process. Among neighbouring candidates, whichever has the best match to their most similar patch in $\mathcal{Q}$ is kept and the others are deleted. I also have a hard limit on the maximum number of candidates in a frame. If there remains too many, then they are all ranked by $\min_i \|\mathbf{p}_t^k - \mathbf{q}_i\|$ (where k is the candidate number) and those not in the top count are discarded.

2.4 Results

Direct comparisons to other systems are tricky because long term occlusions and interactivity are not addressed by existing methods. Therefore, this section is mostly a demonstration of what is achievable using the approach described above, making for a relatively short report.

The demonstration is made with four video sequences. The first is shown in Figure 2.2. A giraffe in the background is occluded by another giraffe walking in front of it. A single click on the rear giraffe's eye is enough to obtain a perfect track though all 140 PAL resolution colour frames. To give an idea of the scale of the parameters used, the coefficients were set

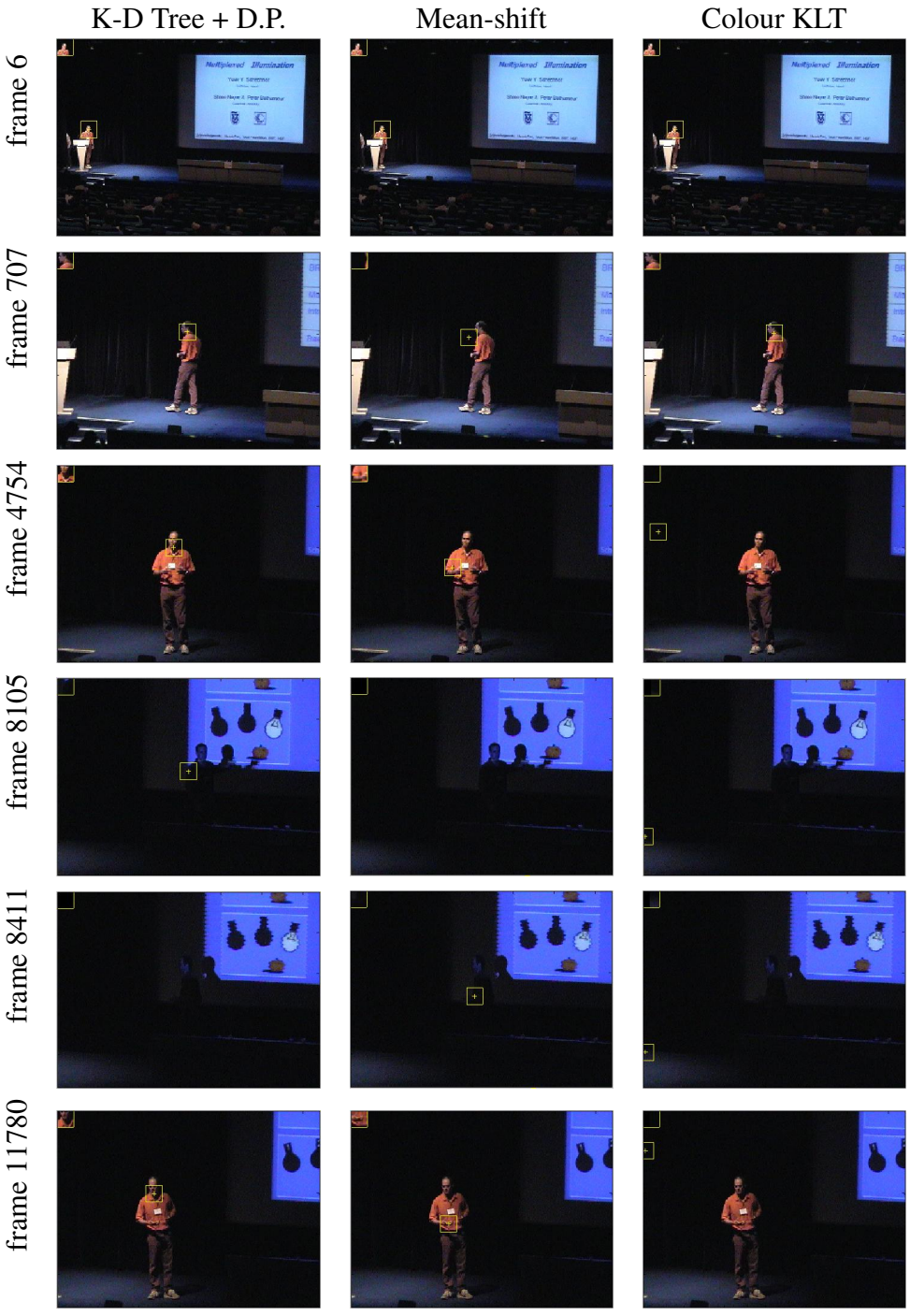


Figure 2.14: **Results II.** Six of the 12000 frames from the ICCV'03 sequence. A real-world problem is to track the speaker for aesthetic cropping. First column: with three clicks and seven parameter adjustments, an almost complete track was obtained. The head's center-line was successfully tracked for all 98.4% of the sequence, with full recoveries after the small number of wandering and lost tracks. Second column: the mean-shift tracker performed poorly. It failed during the zoom and was manually restarted in frame 830. It continually lost track (and was automatically restarted in the centre of the image). Third column: the KLT tracker was successful, even through the zoom, until about frame 3000 when it was distracted by the lectern.

to $\lambda_{\mathcal{P}} = 0.95$, $\lambda_{\mathcal{X}} = 0.67$ and $\lambda_{\mathcal{L}} = 80$.

The second sequence can be seen in Figure 2.14. It is a talk from ICCV'03. On the presentation DVD, a cropped version of this video has been used. A real-world task, therefore, is the tracking of the presenter for an aesthetic crop, keeping him central in the frame. Although technically, the speaker is never occluded, he walks to the projector screen several times, resulting in drastic lighting changes, in addition to continual pose variation. These attributes make this sequence surprisingly troublesome.

For the original DVD production, this task was attempted using a variety of modern tracking methods with little success: continual restarts were necessary throughout the sequence to overcome persistent tracking failures. The final track was created by interpolating between manually set keyframes (chosen to minimize the artifacts of the interpolation).

As a test, the production of an aesthetic-crop quality track was repeated using the interactive system and two tracking strategies from the literature: an advanced mean-shift tracker, implemented from Bibby and Reid (2005) and a colour patch KLT tracker. The two comparisons are shown in Figure 2.14.

Using the interactive system, I was able to track the presenter over the first 12000 frames with only three input clicks and seven parameter adjustments. Each patch search ran at almost 200fps and track updates took around 0.5 seconds. The total time spent was just over three minutes. To fit the preprocessed data into the limited memory of the test machine (a laptop PC with 1GB of RAM), the sequence was down-sampled to half resolution and loaded in frame steps of 15. Because of the flexibility of this system, neither of these processes had any detrimental effect on performance quality. Indeed, an excellent advantage of this system is that temporal sub-sampling is readily viable because there is no reliance on a motion model to restrict the motion between frames. Similar parameters were used as for the giraffe sequence, but with a much higher penalty for occlusions to minimize 'drop-outs'. Input patches were defined in frames 5, 725 and 4415. The resulting track is not as accurate as for the giraffe sequence, but is a more than satisfactory solution for the problem laid down

above. Throughout the video the track does wander slightly over the presenter's shoulders, occasionally significantly and at a couple of points loses him completely for a few frames. However, the system always recovers and the track is robustly restored each time.

Both the other two systems took considerably longer than three minutes to generate a complete track, even without intervention. Therefore, they can only be considered successful if they were able to produce an adequate track without supervision and so this is the comparison that is made here. The mean-shift tracker, which ran at real time, failed to give any satisfactory results. Firstly, it failed to track the speaker through the zoom at the start of the video. It was manually restarted after the zoom, but continued to rapidly lose the target. To obtain some long term tracking result, it was restarted automatically in the centre of the frame whenever it lost track. The KLT tracker performed much better, successfully tracking the speaker through the zoom, but by frame 3000, it had failed completely. There are obviously many parameters that could be adjusted to improve the performance of the KLT tracker for this particular sequence, but the time taken for each iteration is not interactive. Indeed, I spent several hours optimizing the algorithm to obtain the track described.

It is clear that whichever method is used, the entire track must be checked to ensure that there are no mistakes anywhere in the sequence. One may then ask whether watching an automated tracker is such a disadvantage. However, a human supervisor does not need to review a video at real-time frame rates to check the fidelity of a track. Scrolling through at twice, five times or even higher multiples of the final speed provides a fully adequate check on tracking success.

Two more example sequences, with six and three tracks respectively, are shown in Figure 2.13.

2.5 Closing Remarks

This chapter shows how the separation of the feature tracking problem into preprocessing and interactive stages allows for a significant improvement in interactive performance. Long sequences can be tracked with very little operator input, and tuning parameters can be varied in real time for maximal efficiency. It is also an interesting look into a redressing of the balance between calculation and storage, sacrificing fully automatic performance, but allowing the system to be applied to a wide variety of footage and use-cases.

This system was designed to overcome the current gap between the manual approach and automatic tracking algorithms which fail on oft-seen complex footage and those that give a user a degree of control to initiate or correct a semi-automated algorithm, but that are slow and/or cumbersome in their realization. By pre-processing the image sequence that is to be worked on, creating an 'image-patch database' in the most general sense, an incredibly fast feature tracker can work behind-the-scenes as a user scrolls through the video selecting points that lie on the desired track, at each point updating an automatically generated track that the user can continue to edit or accept as perfect. In the best case, a single click will return that complete track. At worse the system will become an improved version of existing trackers, where a sequence of intuitive selections and interactive parameter adjustments will be required to perfect the result.

The preprocessing stage is the system's major strength and weakness. It is a strength because it allows arbitrarily large sequences to be tracked, given arbitrarily large RAM in which to store the preprocessed data. Its weakness is the cost of RAM. The examples here were produced on a laptop with 1GB of RAM, allowing the storage of the order of 70 PAL frames worth of metadata. However, given the increasing popularity of compute clusters, it is easy to foresee "k-d tree servers", where each cluster node handles patch searching for a subset of the sequence (performed completely independently per subset), which comes from the advantages of the detection/optimization separation. The cluster nodes need large mem-

ories, but not fast processors. Then only the DP needs to be performed on the client machine, allowing interactive tracking of even long sequences. Note that the fact that the computation is orders of magnitude faster than raw correlation means that orders of magnitude fewer cluster nodes are required. Such a distributed setup has the further advantage that it need not be restricted to a single user: the nature of the user-driven process implies that k-d tree searches will be demanded only intermittently in relation to the time taken on any one search, meaning that many users can work on the same sequence simultaneously.

Several extensions to the described implementation are worthy of discussion: (i) the simultaneous tracking of multiple features on the same object, allowing some between-track coherence to strengthen the motion model, could be both a method to reduce the number of user interventions required and an extra tool in itself (i.e. accurate non-rigid region or object tracking). One can imagine having an iterative process of simultaneous DP optimizations, one for each feature, influencing each other between iterations, or incorporating all tracks into a single DP optimization. Which would make for a better balance between accuracy and speed does not appear to be a straightforward question; (ii) during the discussion of the development of the system implementation, much was made of patch descriptor size, this being a very important aspect affecting both the meta-data memory footprint and search times. The need to minimize descriptor size lead to the use of PCA bases for the patch descriptor generation, which demonstrated very good ROC performance. Nevertheless, alternative basis sets for the patch projection would be worth investigating. It might be that patch matching performance can be improved, e.g. through the incorporation of various invariances, without reducing search efficiencies (that is search speeds especially and memory usage preferably). Tests were performed to look into the use of a discrete scale space (incorporating into each k-d tree patches taken from a series of up- and down-scaled versions of the current frame, using an area ratio of 2, i.e. a linear scale of $\sqrt{2}$—it was decided that the increase in memory footprint and search time due to the increased number of patches stored per tree was acceptable compared to the extra search time that would be induced by searching with a series of

scale-adjusted target patches), but it was found that the PCA descriptor is too sensitive to scale and an overly fine discretization would be necessary. The use of completely different descriptors could help here. There could be an appropriate balance, in terms of search time performance, in the use of additional patches in the search trees (e.g. scale adjusted sets, as mentioned above, or rotation sequences, etc) and more complex descriptors with more elements (i.e. inherent invariance and larger description vectors), in order to obtain significant patch matching improvements; (iii) recall, however, that the described implementation does give rise to a small degree of translation invariance. This could mean that full-frame searching can be reduced to a search of spatially sub-sampled images, which would result in major improvements to patch search times; (iv) the patch search stage currently uses a Euclidean distance kernel to compare patches. Despite projection onto a small basis set and per-frame normalization, this basic norm performed very well. Nevertheless, this leaves much scope for investigation into alternative ways of measuring patch similarity. For example, what is the speed/accuracy trade-off in accounting for the projection? Furthermore, it would seem advantageous to use a robust kernel to cope with specularities, partial occlusions and object boundaries, but incorporating the effects of the descriptor transformation into the robust calculations could be interesting of itself; (v) the current DP implementation is much slower than need be. An optimized implementation would allow the use of second- or even higher-order motion models, or possibly make the proposal of feature-cluster (object) tracking more viable; (vi) the final implementation also saw a big departure away from the pure use of actual latent variables describing position and appearance, instead fixing these directly to observations made in the images. However, during occlusions, an implicit allocation of positions is made to the track, despite the necessarily absent observations. This echo of the latent variable formulation (which would be more convincing if higher order motion models were incorporated) could also be included for the appearance, thus providing full track outputs and possibly improving optimization results. There is also the option of converting the post-processing stage to a full latent variable optimization, rather than a simple refinement

step; (vii) extra features can be added to the on-line stage also. The possibility of an 'auto-track' was investigated to reduce the number of input patches that need to be selected by the operator. The present idea is a simple iteration comprising two steps. Firstly, the track is optimized, as described above, and the local minimum of the track error is found as normal. Secondly, for those frames which have not yet been used to define input patches, the feature appearance is copied from $\mathcal{P}$ and the one most unlike the patches in $\mathcal{Q}$ is added to $\mathcal{Q}$ as a new patch. The sequence is searched for the new patches and two steps are repeated. This is continued until no new patches are added or the error of the track does not decrease. This could be a useful tool for dealing with false negatives. However, it assumes that a current track contains only a small proportion of false positives, if any. This is a reasonable assumption given that the process is user-driven. In summary, there are many variations on the components of the basic system which need to be investigated, but I feel that the argument for this approach is strong.

Chapter 3

Factorization of Incomplete Matrices

3.1 Introduction

The previous chapter described the development of a framework explicitly incorporating user interaction. The need for speedy response times diverted the focus away from an exploration of ways to assess the plausibility of candidate frame-to-frame feature motions. This chapter can be seen as an extended introduction for the following chapter that addresses that very issue. However, it is more that just an introduction: firstly, it describes the concepts involved with rank truncation and matrix factorization (being the essential mathematical background to the next chapter), but secondly it describes the wide-ranging applications that are based on matrix factorization itself and therefore motivate a full investigation into this topic. In addition to the global prior tracking application that will be discussed in detail in the next chapter, matrix factorization is a powerful tool for finding solutions to other central computer vision problems. For example *structure from motion* (Tomasi and Kanade 1992b, Triggs 1996), non-rigid model tracking (Brand 2001, Bregler et al. 2000, Torresani and Hertzmann 2004), non-rigid object reconstruction (Llado et al. 2005, Torresani et al. 2004) and *shape from shading* (Belhumeur and Kriegman 1996), the first three of which are all typically based on tracking data, such as is generated by the systems described in this thesis. To cement the

motivation for investigating factorization, these applications are described in detail later in this chapter. It will become clear that a major hurdle for computer vision in all these applications is missing data. Standard factorization algorithms (e.g. singular value decomposition, SVD) cannot provide solutions when data are missing. If we want to overcome real-life challenges, the ability to cope with missing data must be part of the algorithms we intend to use. To date, there have been many algorithms introduced to provide a solution. However, none manage to consistently provide either good performance, satisfactory solutions or both. An actual example of three applications mentioned above is introduced in the next section to illustrate the problems in a more real sense. It is then important to review the mathematics that is common to all the applications that are covered. Section 3.1.2 sets the mathematical scene and comments on some of the finer points of the problems to ensure that the important aspects of the problem can be differentiated from those which are secondary and so are not considered in this thesis. Section 3.2 presents the many algorithms that have been proposed in the literature. Newton-based methods are presented next, being a class of algorithms that have not yet been explored for this problem. A comparison of all these schemes is given in Section 3.4 using synthetic tests and the real example problems introduced in this chapter. Finally, the conclusions summarize the chapter and describe the direction of further research.

3.1.1 Example Problems

The rest of this section will provide the mathematical details for these examples, so here is a more qualitative description of some of the wide range of applications for factorization algorithms.

Figure 3.1 shows selected frames from three real image sequences. The first is an example of a structure from motion (SFM) problem. It is a toy dinosaur rotating on a turntable. It is a rigid scene seen from many angles. The sequence is given to tracking software that finds visible features in the scene and tracks the image of each feature as it moves around

Figure 3.1: **Real problems.** Three sequences demonstrating actual problems requiring matrix factorization. Top row: a toy dinosaur rotates on a turntable (a sequence of 36 image frames). Middle row: the walking giraffe sequence introduced in Chapter 1—a background giraffe walks out from behind a bush and is momentarily occluded by a giraffe in the foreground walking in front of it (120 frames). Bottom row: a static scene is illuminated from many directions (20 frames).

in image-space. A *measurement matrix* is filled by the lists of the 2D image coordinates of each feature. Impressively, both the structure of the scene (the 3D coordinates of the tracked features on the dinosaur) and the position and orientation of all the cameras can be obtained from the measurement matrix alone. Holes in the measurement matrix arise because not every feature is physically visible in every frame and because the tracking software is not

completely reliable and fails to find a visible feature in some frames, so SFM is a missing data problem in practice.

The second sequence is an example of a non-rigid scene. It is non-rigid on two levels; at the scene level (where there are three independently moving objects, namely the effectively rigid scenery and two giraffes) and at the object level (the giraffes are very much deforming objects). The segmentation of the three objects (i.e. the scene level non-rigidity) is not considered in this report and is left for future work (see Section 3.5). Instead, the background giraffe is considered in isolation. Again, tracking software is used to generate the data that is used to create the measurement matrix. The reasons for missing data in the non-rigid reconstruction problem are the same as for the rigid situation. Here the occlusions are very clear in the first frame shown where the giraffe is behind a bush and in the second frame shown where the foreground giraffe walks in front.

The third sequence demonstrates the setup for illumination based reconstructions (shape from shading). A static scene is lit by a distant light source from different directions. During the filming, the movement of the light source was controlled so that the distance between the bulb and the face was very much larger than the size of the face, thus approximating the requirement of it being at an infinite distance. In this case the measurement matrix is generated directly from the intensity values of the images. Shadows and specular highlights have to be omitted from the measurement matrix because such visual events are not part of the diffuse model (as will be explained in Section 3.1.4 on page 88), resulting in the matrix being incomplete.

3.1.2 Matrix Factorization

Let us start with a review of the mathematical problem at hand and continue on to specific applications and their derivations.

The general problem being addressed in this chapter is matrix factorization. Given a

Symbol	*Meaning*
$\hat{\mathtt{M}}$	the acquired *measurement matrix*. This is the only variable (other than $\mathtt{W}$) whose elements are known at all. It is modelled as a noisy version of $\mathtt{M}$
$\mathtt{W}$	the *weight matrix*. A matrix whose elements describe the confidence in the knowledge of the corresponding elements of $\hat{\mathtt{M}}$
$\mathtt{M}$	the rank r matrix to be factorized
$\mathtt{A}$	a column space for $\mathtt{M}$
$\mathtt{B}$	a row space for $\mathtt{M}$
m	the number of rows in $\mathtt{M}$, $\hat{\mathtt{M}}$, $\mathtt{W}$ and $\mathtt{A}$
n	the number of columns in $\mathtt{M}$, $\hat{\mathtt{M}}$ & $\mathtt{W}$ and the number of rows in $\mathtt{B}$
r	the rank of $\mathtt{M}$ and the number of columns in $\mathtt{A}$ and $\mathtt{B}$
m_{ij}	the element of $\mathtt{M}$ on the i^{th} row and j^{th} column
$\mathbf{m}^i$	the i^{th} row of $\mathtt{M}$ as a column vector
$\mathbf{m}_j$	the j^{th} column of $\mathtt{M}$
$\mathtt{M}_{(m\times n)}$	dimensions of vectors and matrices are given in subscripted parentheses
δ_{ij}	a Kronecker delta function equalling one when $i=j$ and zero otherwise
$\odot$	the Hadamard *element-wise* matrix product: $\mathtt{R}=\mathtt{P}\odot\mathtt{Q} \implies r_{ij}=p_{ij}q_{ij}$
$\mathtt{M}^+$	the pseudo-inverse of $\mathtt{M}$ allowing $\mathtt{M}^+\mathtt{M}=\mathtt{I}$ for $r=n<m$
$\mathtt{M}\downarrow_r$	*rank truncation*: the closest rank r matrix to $\mathtt{M}$ by the Frobenius norm
$\mathtt{M}(:)$	column-wise vectorization of $\mathtt{M}$: $\mathtt{M}(:)=(\mathbf{m}_1^\top\ \mathbf{m}_2^\top\ \ldots\ \mathbf{m}_n^\top)^\top$
$\mathtt{M}(\cdot\cdot)$	row-wise vectorization of $\mathtt{M}$: $\mathtt{M}(\cdot\cdot)=(\mathbf{m}^{1\top}\ \mathbf{m}^{2\top}\ \ldots\ \mathbf{m}^{m\top})^\top$

Figure 3.2: **Notation.** Summary of notation used in this chapter (fold-out copy at back of book).

matrix, $\mathtt{M}$, of known rank, r, find two smaller matrices, $\mathtt{A}$ and $\mathtt{B}$, such that:

$$\mathtt{M}_{(m\times n)}=\mathtt{A}_{(m\times r)}\mathtt{B}^\top_{(n\times r)} \tag{3.1}$$

The previous section described examples of real problems. The process common to the applications is as follows; an imaging device records one or more images of the real world in a digital form. These images are processed and the entries of $\mathtt{M}$ are filled. The solution sought is an $\mathtt{A}$ and $\mathtt{B}$ that satisfy Equation 3.1. In practice there are four significant aspects that make a solution to Equation 3.1 difficult:

1. There is not a unique answer to any such problem, as any invertible $r \times r$ matrix, $\mathtt{G}$, may always be introduced as a transform on the factoring matrices:

$$\mathtt{M} = (\mathtt{AG})(\mathtt{G}^{-1}\mathtt{B}^{\top}) = \mathtt{A}'\mathtt{B}'^{\top} \tag{3.2}$$

Here, $\mathtt{G}$ represents the *gauge freedom* (Triggs et al. 2000). It turns out that this is not actually as inconvenient as it may seem. In most applications it can either be disregarded or easily circumnavigated, e.g. by registration or normalization.

2. $\mathtt{M}$ is not known exactly. Instead, a noisy version of $\mathtt{M}$ is acquired. For convenience, the differences between the known noisy elements and the supposed true values are assumed to be independently drawn from a particular distribution. Although no one distribution may be used in general, the algorithms reviewed in this report all assume that a zero-mean Gaussian distribution is valid. The algorithms I propose in Section 3.3 are not constrained to a particular distribution, but using the Normal distribution we have:

$$\hat{\mathtt{M}} = \mathtt{M} + \mathcal{N}(0, \sigma^2) \tag{3.3}$$

where $\hat{\mathtt{M}}$ is the measurement matrix. It is so likely that $\hat{\mathtt{M}}$ will have a rank greater than r that the alternatives are ignored. Because the expectation, $<\hat{\mathtt{M}}>$, *is* $\mathtt{M}$, the problem to be solved becomes that of finding the rank r matrix that is closest to $\hat{\mathtt{M}}$, i.e.

$$\min_{\mathtt{A},\mathtt{B}} \left\| \; \hat{\mathtt{M}} - \mathtt{A}\mathtt{B}^{\top} \; \right\| \tag{3.4}$$

When the Frobenius or 2-norm is taken as a distance measure, singular value decomposition gives the correct answer (see Golub and Van Loan (1996), page 72 and Reid and Murray (1996)).

3. For various application-specific reasons (see previous section), not all the elements of $\hat{\mathtt{M}}$ are known. Any minimization scheme employed must be adapted to ignore the elements of $\hat{\mathtt{M}}$ that do not have values. Here, we introduce a *weight matrix*, $\mathtt{W}$, whose elements reflect the knowledge of the corresponding elements of $\hat{\mathtt{M}}$. Zeros in $\mathtt{W}$ mean that those positions in $\hat{\mathtt{M}}$ do not have values and ones mean they do. Values between zero and one may also be used, denoting relative confidences in the values in $\hat{\mathtt{M}}$. The matrix $\mathtt{W}$ can be included in the minimization thus:

$$\min_{\mathtt{A},\mathtt{B}} \left\| \mathtt{W} \odot \left(\hat{\mathtt{M}} - \mathtt{A}\mathtt{B}^{\top} \right) \right\| \tag{3.5}$$

where the $\odot$ operator denotes the Hadamard matrix product: $\mathtt{R} = \mathtt{P} \odot \mathtt{Q} \implies r_{ij} = p_{ij}\, q_{ij}$. This problem cannot be solved using standard factorization techniques (such as singular value decomposition).

4. In real applications, it cannot be guaranteed that all the data entered into the measurement matrix $\hat{\mathtt{M}}$ are consistent with the model. Maybe because the model does not fully explain all observed situations (such as shadows and specularities in the illuminated face example) or because the process that generates the elements of $\hat{\mathtt{M}}$ from an image sequence is not completely reliable (e.g. feature tracking software for SFM problems sometimes present a track that is actually jumping between more than one similar looking feature as the path of a single feature point). Such data are called outliers and can significantly mislead algorithms, encouraging them to return erroneous results. Algorithms that are not distracted by outliers and in some way manage to give correct solutions despite their presence are called robust algorithms. One way to deal with outliers is through the use of appropriate distributions on the noise model, making this a special case of Point 2 above. However, outliers can be, and are, dealt with separately, hence the separate point made here. None of the missing data algorithms reviewed in this report are robust and outlier rejection is left for future investigation.

Summary

In Equation 3.5 we have the focus of this chapter. It is used by all the applications described and as such, all the algorithms included in this chapter, which have been introduced as ways to tackle this formula, may be used for finding solutions to any of the forms of the problem.

3.1.3 Notation

In the literature there are many labels used for the components of Equation 3.5. In this chapter the same variable names will be used throughout, aiding direct comparisons of the approaches that have been proposed over the years. Specifically, the measurement matrix, $\hat{\mathtt{M}}$, is the only matrix with known elements—unknown elements being signified by the corresponding zero elements of the weight matrix, $\mathtt{W}$—and is assumed to be a noisy version of $\mathtt{M}$. The noise on each element is taken as being independently drawn from a zero-mean Normal distribution, with variance σ^2. The desired factoring matrices are denoted by $\mathtt{A}$ and $\mathtt{B}$. $\mathtt{M}$ is known to be rank r and so, given $\mathtt{M} \in \mathbb{R}^{m\times n}$, we have $\mathtt{A} \in \mathbb{R}^{m\times r}$ and $\mathtt{B} \in \mathbb{R}^{n\times r}$ with $\mathtt{W}, \hat{\mathtt{M}} \in \mathbb{R}^{m\times n}$. The weight matrix will mostly be considered as a matrix of zeros and ones. Also note that $\mathtt{M}$, without loss of generality (see Section 3.2.1), is always considered to be portrait, with $m > n$.

More generally, and in line with the other chapters, scalars are italicized (s), vectors are bold ($\mathbf{v}$) and matrices are presented in the 'typewriter' typeface ($\mathtt{M}$). Different styles of the same letter will always refer to the same matrix, namely: m_{ij} is the element on the i^{th} row and in the j^{th} column, $\mathbf{m}_j$ the j^{th} column and $\mathbf{m}^i$ the transpose of the i^{th} row of matrix $\mathtt{M}$. Subscript and superscript indices will also be used to represent column and row numbers, respectively, for block segmentation of matrices. Vectors should always be taken as column vectors, with row vectors being represented using transpose notation.

The rank truncation of a matrix, $\mathtt{M}$, to a lower rank r (that is, the closest rank r matrix to $\mathtt{M}$), is denoted with $\mathtt{M}\downarrow_r$, i.e. $\tilde{\mathtt{M}} = \mathtt{M}\downarrow_r$ such that $\|\mathtt{M} - \tilde{\mathtt{M}}\|$ is minimized. This can be achieved by

taking the singular value decomposition (SVD) of $\mathtt{M}$ as $\mathtt{U\Sigma V}^\top$, then zeroing out all the diagonal entries of Σ beyond the r^{th} to give $\tilde{\Sigma}$ and finally rebuilding the matrix: $\mathtt{M}\downarrow_r = \mathtt{U}\tilde{\Sigma}\mathtt{V}^\top$.

For the row-wise vectorization of $\mathtt{M}$, $\mathtt{M}(\cdot)$ is used, i.e. $(\mathbf{m}^{1\top}\ \mathbf{m}^{2\top}\ \ldots\ \mathbf{m}^{m\top})^\top$.

Finally, the pseudo-inverse of a matrix $\mathtt{M}$ is represented by $\mathtt{M}^+$. It is used when an actual inverse of $\mathtt{M}$ does not exist (e.g. when $\mathtt{M}$ is rectangular). If $\mathtt{M}$ is square and full rank then $\mathtt{M}^+{=}\mathtt{M}^{-1}$, whereas if $r{=}n{<}m$ then $\mathtt{M}^+{=}(\mathtt{M}^\top\mathtt{M})^{-1}\mathtt{M}^\top$. In general, the pseudo-inverse of a rank r matrix $\mathtt{M}$ may be constructed using the SVD thus:

$$\mathtt{M}^+ = \mathtt{V}\Sigma^+\mathtt{U}^\top \quad \text{where} \quad \Sigma^+ = \mathrm{diag}(\frac{1}{\sigma_1}, \ldots, \frac{1}{\sigma_r}, 0, \ldots, 0) \tag{3.6}$$

with σ_i being the i^{th} singular value.

3.1.4 Applications

With the general formulation described above, it is now easy to explain how the computer vision applications mentioned at the start of the chapter can be cast as factorization problems. The rest of this section goes through the models used in each application.

Structure from Motion

The term 'structure from motion' (and more correctly, but less commonly 'structure recovery from motion') refers to the process of recovering scene structure (the world positions of 3D points and cameras) from the motion of the 2D projections of the points through an image sequence. It is this problem that is given the most attention in this report, mainly because it is the one which is treated most frequently in the literature. Tomasi and Kanade (1992b) presented factorization as a way of solving the SFM problem. The basis is the affine camera model, which describes cameras as orthographically projecting scene points onto an image

plane. For F views (or frames) of P points, the equation of projection is

$$\mathbf{m}_p^f = \mathtt{P}^f \mathbf{x}_p + \mathbf{t}^f \quad f = 1, \ldots, F; p = 1, \ldots, P \tag{3.7}$$

with $\mathbf{m}_p^f$ being the two-dimensional vector encoding the f^{th} image of the p^{th} point, $\mathtt{P}^f$ a 2×3 matrix describing the f^{th} camera's orientation and scaling effects, $\mathbf{t}^f$ the translation vector moving the image of the world coordinate origin to the f^{th} frame's origin and $\mathbf{x}_p$ being the three-vector for the global position of the p^{th} point. The equations for all the points in one view may be concatenated to give

$$\begin{bmatrix} \mathbf{m}_1^f & \mathbf{m}_2^f & \cdots & \mathbf{m}_P^f \end{bmatrix} = \mathtt{P}^f \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \ldots & \mathbf{x}_P \end{bmatrix} + \mathbf{t}^f \mathbf{1}^\top \tag{3.8}$$

$$or \quad \mathtt{M}^f = \mathtt{P}^f \mathtt{X} + \mathbf{t}^f \mathbf{1}^\top \tag{3.9}$$

where $\mathbf{1}$ is a P-vector of ones. All the views may then be concatenated vertically for a conveniently compact matrix form of the model:

$$\begin{bmatrix} \mathtt{M}^1 \\ \mathtt{M}^2 \\ \vdots \\ \mathtt{M}^F \end{bmatrix} = \begin{bmatrix} \mathtt{P}^1 \\ \mathtt{P}^2 \\ \vdots \\ \mathtt{P}^F \end{bmatrix} \mathtt{X} + \begin{bmatrix} \mathbf{t}^1 \\ \mathbf{t}^2 \\ \vdots \\ \mathbf{t}^F \end{bmatrix} \mathbf{1}^\top \tag{3.10}$$

$$or \quad \mathtt{M} = \mathtt{PX} + \mathbf{t}\mathbf{1}^\top \tag{3.11}$$

which, in turn, may be written as a single matrix multiplication, putting the problem into the form addressed in this report:

$$\mathtt{M} = \begin{bmatrix} \mathtt{P} & \mathbf{t} \end{bmatrix} \begin{bmatrix} \mathtt{X} \\ \mathbf{1}^\top \end{bmatrix} = \mathtt{AB}^\top \tag{3.12}$$

The measurement matrix, $\mathtt{M}$, collects all of the coordinates of all the images of all the points. It is the known component of Equation 3.12, obtained from the image sequence (Tomasi and Kanade 1992a). Under the affine camera model, it is a rank 4 matrix, giving $r = 4$, $m = 2F$ and $n = P$.

The inclusion of the vector $\mathbf{t}$ and hence the constraint that the last column of $\mathtt{B}$ should be all ones is inconvenient. Here, we can turn to the gauge freedom for help. Consider a matrix $\mathtt{G}$ of the form:

$$\mathtt{G} = \begin{bmatrix} \mathtt{I} & -\mathbf{v} \\ \mathbf{0}^\top & 1 \end{bmatrix} \tag{3.13}$$

and insert $\mathtt{G}\mathtt{G}^{-1}$ into Equation 3.12 to give the following transformation:

$$\mathtt{M} = \begin{bmatrix} \mathtt{P} & \mathbf{t} \end{bmatrix} \begin{bmatrix} \mathtt{I} & -\mathbf{v} \\ \mathbf{0}^\top & 1 \end{bmatrix} \begin{bmatrix} \mathtt{I} & \mathbf{v} \\ \mathbf{0}^\top & 1 \end{bmatrix} \begin{bmatrix} \mathtt{X} \\ \mathbf{1}^\top \end{bmatrix} \tag{3.14}$$

$$= \begin{bmatrix} \mathtt{P} & -\mathtt{P}\mathbf{v}+\mathbf{t} \end{bmatrix} \begin{bmatrix} \mathtt{X} + \mathbf{v}\mathbf{1}^\top \\ \mathbf{1}^\top \end{bmatrix} \tag{3.15}$$

$$= \begin{bmatrix} \mathtt{P} & \mathbf{t}' \end{bmatrix} \begin{bmatrix} \mathtt{X}' \\ \mathbf{1}^\top \end{bmatrix} \tag{3.16}$$

Using the new factorization, let us look at the row-wise summation of the elements of $\mathtt{M}$:

$$\sum_{j=1}^{n} \mathbf{m}_j = \sum_{j=1}^{n} \mathtt{P}\mathbf{x}'_j + \mathbf{t}'\mathbf{1}_j = n\mathbf{t}' + \mathtt{P}\sum_{j=1}^{n} \mathbf{x}'_j \tag{3.17}$$

We can eliminate the first term of the right-hand side of Equation 3.17 if we can make

$\sum_{j=1}^{n} \mathbf{x}'_j$ equal zero.

$$\sum_{j=1}^{n} \mathbf{x}'_j = 0 \quad \Longrightarrow \quad \sum_{j=1}^{n} \mathbf{x}_j + \mathbf{v} = n\mathbf{v} + \sum_{j=1}^{n} \mathbf{x}_j = 0 \tag{3.18}$$

$$or \quad \mathbf{v} = \frac{1}{n} \sum_{j=1}^{n} \mathbf{x}_j \tag{3.19}$$

So, the gauge freedom of the problem can be used to set the origin of the 3D world points to be at their centroid. Now, following on from Equation 3.17, we have

$$\sum_{j=1}^{n} \mathbf{m}_j = n\mathbf{t}' \quad or \quad \mathbf{t}' = \frac{1}{n} \sum_{j=1}^{n} \mathbf{m}_j \tag{3.20}$$

This says that the image of the centroid of a set of points is the centroid of the images of all the points, bringing us to the operation of mean-centring the measurement matrix:

$$\bar{\mathtt{M}} = \mathtt{M} - \bar{\mathbf{m}}_j \mathbf{1}^\top \quad \text{with} \quad \bar{\mathbf{m}}_j = \frac{1}{n} \sum_{j=1}^{n} \mathbf{m}_j \tag{3.21}$$

Equation 3.12 is affine projection in homogeneous coordinates and highlights the fact that $\mathtt{M}$ is actually a three-dimensional structure embedded in a four-dimensional space, i.e. a hyperplane that does not pass through the origin. By mean-centring the measurement matrix, the hyperplane is translated to the origin and $\bar{\mathtt{M}}$ becomes rank 3, with no constraints:

$$\bar{\mathtt{M}} = \mathtt{P}'\mathtt{X}' \tag{3.22}$$

Sadly, manipulating $\mathtt{M}$ to be mean-centred is an option that is not available when elements of $\mathtt{M}$ are unknown and so affine SFM with missing data remains the larger rank 4 problem in Equation 3.12. The way minimization is affected by this situation is discussed in the next section (Section 3.1.5).

As mentioned above, with all three of the reconstruction from motion applications men-

tioned here, the practicalities of missing data is all too apparent when it comes to their implementations. When filling $\mathtt{M}$ from the coordinates of tracked scene points, occlusions or failures in the tracking process leave elements for which values are not known, sometimes making the measurement matrix very sparse.

Non-Rigid Structure from Motion

Extending this model to account for non-rigid motion simply adds another layer of factorization. Assume that non-rigid objects can change shape by being a weighted sum of a set of B *basis shapes*, $\mathtt{S}^n$, which are $3 \times P$ matrices holding the positions all the 3D points (in order) for each shape. So, for frame f, we have:

$$\mathtt{X}^f = \lambda_1^f \mathtt{S}^1 + \lambda_2^f \mathtt{S}^2 + \ldots + \lambda_B^f \mathtt{S}^B = \sum_{i=1}^{B} \lambda_i^f \mathtt{S}^i \tag{3.23}$$

and so

$$\mathtt{M}^f = \mathtt{P}^f \mathtt{X}^f = \sum_{i=1}^{B} \lambda_i^f \mathtt{P}^f \mathtt{S}^i \tag{3.24}$$

leading to the points for all frames to be written in one expression:

$$\mathtt{M}_{(2F\times P)} = \begin{bmatrix} \lambda_1^1\mathtt{P}^1 & \lambda_2^1\mathtt{P}^1 & \cdots & \lambda_B^1\mathtt{P}^1 \\ \lambda_1^2\mathtt{P}^2 & \lambda_2^2\mathtt{P}^2 & \cdots & \lambda_B^2\mathtt{P}^2 \\ \vdots & \vdots & & \vdots \\ \lambda_1^F\mathtt{P}^F & \lambda_2^F\mathtt{P}^F & \cdots & \lambda_B^F\mathtt{P}^F \end{bmatrix}_{(2F\times 3B)} \begin{bmatrix} \mathtt{S}^1 \\ \mathtt{S}^2 \\ \vdots \\ \mathtt{S}^B \end{bmatrix}_{(3B\times P)} \tag{3.25}$$

$$= \mathtt{A}\mathtt{B}^\top \tag{3.26}$$

i.e. the measurement matrix is a rank $3B$ matrix, with the special feature that $\mathtt{A}$ has a very specific structure (two-row sub-blocks can be reordered into rank 1 matrices[1]). See Bregler

[1]Consider an odd-even row pair sub-block of $\mathtt{A}$: $\mathtt{A}' = [\lambda_1\mathtt{P}\ \lambda_2\mathtt{P}\ \cdots\ \lambda_B\mathtt{P}]$ and let $\mathbf{p} = \mathtt{P}(:)$. It is straightforward to rearrange $\mathtt{A}'$ to obtain the rank 1 matrix $[\lambda_1\mathbf{p}\ \lambda_2\mathbf{p}\ \cdots\ \lambda_B\mathbf{p}] \equiv \mathbf{p}\,[\lambda_1\ \lambda_2\ \cdots\ \lambda_B]$.

et al. (2000) for more details.

Structure from Perspective Motion

The perspective camera model is essentially the perspective version of Equation 3.7, i.e. using homogeneous coordinates in a projective space:

$$\lambda_p^f \mathbf{m}_{p(3\times1)}^f = \mathtt{P}_{(3\times4)}^f \mathbf{x}_{p(4\times1)} \quad f = 1,\ldots,F; p = 1,\ldots,P \tag{3.27}$$

where λ is an arbitrary scale factor introduced to deal with the lack of uniqueness inherent with representing coordinates using homogeneous vectors. As before, the equations for all the points and all the frames can be stacked into one matrix equation:

$$\begin{bmatrix} \lambda_1^1\mathbf{m}_1^1 & \lambda_2^1\mathbf{m}_2^1 & \cdots & \lambda_P^1\mathbf{m}_P^1 \\ \lambda_1^2\mathbf{m}_1^2 & \lambda_2^2\mathbf{m}_2^2 & \cdots & \lambda_P^2\mathbf{m}_P^2 \\ \vdots & \vdots & \ddots & \vdots \\ \lambda_1^F\mathbf{m}_1^F & \lambda_2^F\mathbf{m}_2^F & \cdots & \lambda_P^F\mathbf{m}_P^F \end{bmatrix} = \begin{bmatrix} \mathtt{P}^1 \\ \mathtt{P}^2 \\ \vdots \\ \mathtt{P}^F \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_P \end{bmatrix} \tag{3.28}$$

$$= \mathtt{A}\mathtt{B}^\top \tag{3.29}$$

The addition of equality up to scale is the crucial factor that sets this problem apart from the simple affine model. Only two-thirds of the measurement matrix is known here, as the image coordinates of the points only give the inhomogeneous part, and not the scale factor. Having said that, we can actually choose $F+P$ of them arbitrarily, but the rest must be calculated. The calculation of the *projective depths* proceeds by operating on the epipolar geometry of frame pairs (Triggs 1996). Once the measurement matrix is complete, it is known to be rank 4 and so can be factorized. This model is mentioned here for completeness and will not be discussed further in this report.

Surface from Illumination

Another problem that can be formulated as a single matrix product is that of recovering surface normals from a series of varied illumination images. Considering only Lambertian illumination, the light reflected from a flat surface with normal $\mathbf{n}$ from a light source at infinity in direction $\mathbf{d}$ is simply the dot-product of the two vectors. The intensity of the incoming light and the absorption (albedo) of the surface can be encoded as the magnitude of the appropriate vectors, i.e. $\mathbf{n} = c\hat{\mathbf{n}}$ and $\mathbf{d} = l\hat{\mathbf{d}}$. Therefore, given a set of M images of a Lambertian object under varied lighting conditions (with the camera's orientation relative to the object constant throughout) we can take the object's surface as being made up of planar segments—one under each pixel—to build up a measurement matrix by entering all the pixel intensity values from each image along each row. Each column is therefore the intensity of a single point on the surface over all illumination directions. Each element of the measurement matrix, thus each pixel (i.e. surface point), is calculated as

$$m_{ij} = \mathbf{d}^{i\top}\mathbf{n}_j \quad \forall ij \tag{3.30}$$

and concatenating these together in the usual way gives

$$\mathtt{M}_{(M\times N)} = \begin{bmatrix} \mathbf{d}^{1\top} \\ \mathbf{d}^{2\top} \\ \vdots \\ \mathbf{d}^{F\top} \end{bmatrix}_{(M\times 3)} \begin{bmatrix} \mathbf{n}_1 & \mathbf{n}_2 & \cdots & \mathbf{n}_P \end{bmatrix}_{(3\times N)} \tag{3.31}$$

$$= \mathtt{A}\mathtt{B}^{\top} \tag{3.32}$$

i.e. the factorization of the measurement matrix gives the illumination direction for every frame and surface normal for every image pixel.

The lack of perfectly diffuse surfaces in the real world means that implementations join

the ranks of missing data problems. It was introduced earlier that elements of the measurement matrix must be ignored when a pixel's colour value shows the surface to be exhibiting reflectance outside the diffuse model. This is covered by values below a low threshold representing a surface going into shadow and values above a high threshold representing the reflectance becoming specular. Helpfully, specularities are often small and a moving light source, which is inherent to this application, means that both specularities and shadows will move across the surface leaving a good proportion of the measurement matrix with data for processing.

3.1.5 The Error Function

The problem at the heart of this chapter is the minimization of

$$F(\mathtt{A},\mathtt{B}) = \left\| \mathtt{W} \odot \left(\hat{\mathtt{M}} - \mathtt{A}\mathtt{B}^{\top}\right)\right\|_F^2 . \tag{3.33}$$

When all of the elements in $\hat{\mathtt{M}}$ are known and $\mathtt{W}$ contains only ones, the error function has one minimum and that is found by the SVD. Nonlinear minimization algorithms also quickly find the global minimum. When data are missing, only iterative approaches can provide solutions. Their job is made difficult because the weighted error function, F, has many minima (see Figure 3.3). By introducing the weight matrix, the error surface is drastically changed. As well as multiple minima, the global minimum in the full data case is, in general, no longer a minimum in the missing data case. Not only must algorithms deal with local minima, they must also incorporate useful regularization to get closer to the answer that would be obtained if all data were available.

Looking further into the space in which F is to be minimized, we have the following

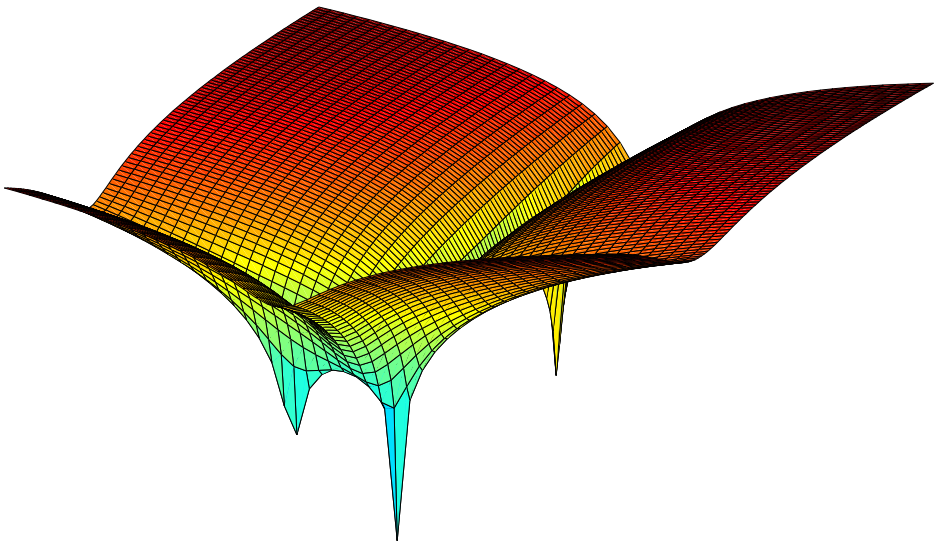


Figure 3.3: **Multiple minima I.** A cross-section through a real error surface showing three minima. It was generated by evaluating the error function using the data from the dinosaur sequence. The error is a function of many more than two variables so this cross-section was chosen by picking three minima found by the damped Newton algorithm (see Section 3.3). Evaluation of the all the derivatives confirms that these are minima in all dimensions.

insight. Let

$$\mathbf{x} = \begin{pmatrix} \mathtt{A}(\cdot\cdot) \\ \mathtt{B}(\cdot\cdot) \end{pmatrix} \tag{3.34}$$

be the row-wise unravelling (page 81) of both $\mathtt{A}$ and $\mathtt{B}$ into an $r(m+n)$-vector. The error can now be written in vector form relating directly to the $r(m+n)$-dimensional space of the error surface.

$$F(\mathbf{x}) = \sum_{i,j=1}^{m,n} w_{ij}^2 \left(\hat{m}_{ij} - \mathbf{x}^\top \mathtt{S}_{ij} \mathbf{x}\right)^2 \tag{3.35}$$

Here, $\mathtt{S}_{ij}$ is the ij^{th} $r(m+n)$-square *selection* matrix, forming the appropriate dot-product between the rows of $\mathtt{A}$ and $\mathtt{B}$. It has a very simple structure:

$$\mathtt{S}_{ij} = \left[\begin{array}{ccc|ccc} & & & \mathtt{0}_{r\times r} & \cdots & \mathtt{0}_{r\times r} \\ & & & & \ddots \quad \iddots & \\ & \mathtt{0}_{mr\times mr} & & \vdots & \mathtt{0}_{r\times r} \quad \frac{1}{2}\mathtt{I}_{r\times r} & \vdots \\ & & & & \iddots \quad \ddots & \\ & & & \mathtt{0}_{r\times r} & \cdots & \mathtt{0}_{r\times r} \\ \hline \mathtt{0}_{r\times r} & \cdots & \mathtt{0}_{r\times r} & & & \\ & \ddots \quad \iddots & & & & \\ \vdots & \mathtt{0}_{r\times r} & \vdots & & \mathtt{0}_{nr\times nr} & \\ & \iddots \quad \frac{1}{2}\mathtt{I}_{r\times r} \quad \ddots & & & & \\ \mathtt{0}_{r\times r} & \cdots & \mathtt{0}_{r\times r} & & & \end{array}\right] \tag{3.36}$$

that is, $\mathtt{S}_{ij}$ is a matrix of all zeros, except for a half identity matrix in the ij^{th} $r\times r$ block of the upper right segment and another in the ji^{th} block of the lower left segment. The halves have been introduced to make $\mathtt{S}_{ij}$ symmetric. Looking at a cross-section through this error surface, from a point $\mathbf{p}$ and in the direction $\mathbf{q}$, by substituting $\mathbf{x}$ with $\mathbf{p}+t\mathbf{q}$, we can see that

the error surface can be quartic:

$$F(t) = \sum_{i,j=1}^{m,n} w_{ij}^2 \left(\hat{m}_{ij} - (\mathbf{p} + t\mathbf{q})^\top \mathtt{S}_{ij} (\mathbf{p} + t\mathbf{q})\right)^2 \tag{3.37}$$

$$= \sum_{i,j=1}^{m,n} w_{ij}^2 \left(\kappa_{0ij} + \kappa_{1ij} t + \kappa_{2ij} t^2 + \kappa_{3ij} t^3 + \kappa_{4ij} t^4\right) \tag{3.38}$$

with

$$\kappa_{0ij} = \left(\hat{m}_{ij} - \mathbf{p}^\top \mathtt{S}_{ij} \mathbf{p}\right)^2 \tag{3.39}$$

$$\kappa_{1ij} = 4 \left(\mathbf{p}^\top \mathtt{S}_{ij} \mathbf{p} - \hat{m}_{ij}\right) \mathbf{p}^\top \mathtt{S}_{ij} \mathbf{q} \tag{3.40}$$

$$\kappa_{2ij} = 2 \left(\mathbf{p}^\top \mathtt{S}_{ij} \mathbf{p} - \hat{m}_{ij}\right) \mathbf{q}^\top \mathtt{S}_{ij} \mathbf{q} + 4 \left(\mathbf{p}^\top \mathtt{S}_{ij} \mathbf{q}\right)^2 \tag{3.41}$$

$$\kappa_{3ij} = 4 \left(\mathbf{p}^\top \mathtt{S}_{ij} \mathbf{q}\right) \left(\mathbf{q}^\top \mathtt{S}_{ij} \mathbf{q}\right) \tag{3.42}$$

$$\kappa_{4ij} = \left(\mathbf{q}^\top \mathtt{S}_{ij} \mathbf{q}\right)^2 \tag{3.43}$$

noting that $\mathbf{p}^\top \mathtt{S}_{ij} \mathbf{q} = \mathbf{q}^\top \mathtt{S}_{ij} \mathbf{p}$. If the direction $\mathbf{q}$ is chosen to be a coordinate direction (i.e. a vector of zeros except for one entry set to one), $\mathbf{q}^\top \mathtt{S}_{ij} \mathbf{q}$ is zero and the error function is a sum of quadratics. This is used by alternation to form closed-form solutions as will be seen in Section 3.2.2. However, directions can be chosen to reveal the quartic nature of the surface (this is similar to the fact that a ruled quadric can look linear in cross-section).

It may be helpful to consider the situation in an algebraic light. Taking the columns of $\mathtt{M}$ to be points in an m-dimensional space, factorization is the a process that fits an r-dimensional hyperplane though the origin to those points, minimizing the squared perpendicular distance from the plane to the points (i.e. minimizing Equation 3.4). If the points are thought to lie on an $(r-1)$-dimensional hyperplane, that does not include the origin, factorization can provide the answer in that situation as well. As a brief example, here is the least-squares

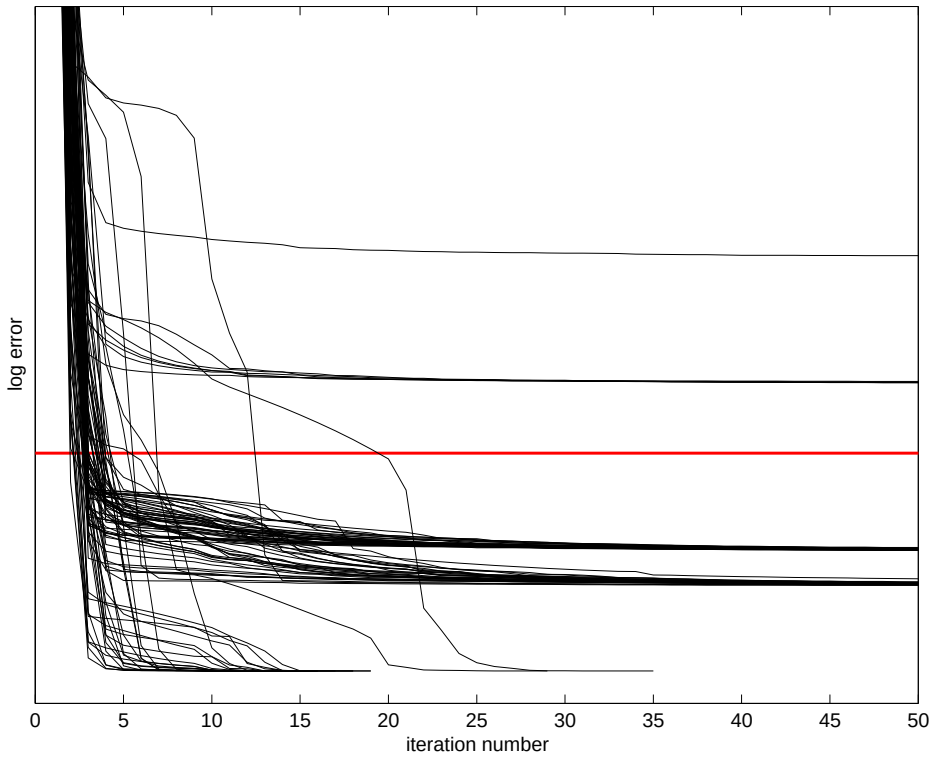


Figure 3.4: **Overfitting I.** A graph of the missing-data error as it is minimized by the algorithm described in Algorithm 3.16 from 100 random starting points. The red horizontal line is the error of the answer given by the SVD of the full measurement matrix, i.e. the answer we want. The iterative scheme has found five minima, none of which are a satisfactory answer. Although three of the minima appear to be better than the SVD solution, it must be emphasized that the error function is only looking at a subset of residuals and so can favour solutions that are poor when compared to the full input. This test was performed on random synthetic data.

error function for fitting a line to a set of 3D points:

$$F(\mathbf{d}, \boldsymbol{\alpha}, \mathbf{v}) = \left\| \mathtt{X} - \left(\mathbf{d}\boldsymbol{\alpha}^\top + \mathbf{v}\mathbf{1}^\top \right) \right\|_F^2 \tag{3.44}$$

where $\mathtt{X}$ holds all the points to be fitted as columns, $\mathbf{d}$ is the direction of the line, $\boldsymbol{\alpha}$ holds all the distances along the line of the nearest points on the line to each point in $\mathtt{X}$ and $\mathbf{v}$ is the offset of the line from the origin. The α_i in this case would be measured from $\mathbf{v}$ in 'lengths-of-$\mathbf{d}$'. As with the SFM formulation, we may stack $\mathbf{d}$ and $\mathbf{v}$ into one matrix and put $\boldsymbol{\alpha}$ and $\mathbf{1}$ into another to give the form of the function in Equation 3.4. Hence,

constraining the last column of B to be all ones, finds the least squares solution to fitting an offset $(r-1)$-dimensional hyperplane. Note that this is the same as performing principal component analysis on a set of points without the origin as their centroid (the mean point), i.e. the centroid must be found in the minimization along with the principal components.

When the weight matrix is included, it is like a window that the minimization looks through. Seeing only a proportion of the residual matrix means that it can reach a lower error than the theoretical minimum. Figure 3.4 shows an example of an iterative algorithm falling into local minima. It is an example of the situation where the solution that is sought (the solution given by the SVD of the full measurement matrix) has a higher error than solutions attained by iterative schemes. Because the error function being minimized by the iterative scheme only looks at a subset of elements from the *residual matrix* (the difference between the input, M, and a pair of proposed factors, A and B), it has a distorted view of how well it is doing, hence the seemingly superior results. In practical situations there is no way of knowing how far the solution given by the missing-data minimization is away from the 'true' answer.

3.2 Existing Solutions

Here is a review of the existing solutions to the problem of the decomposition of incomplete matrices.

3.2.1 Direct Methods

A space can be described by a span of vectors, $\{\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_r\}$, linear combinations of which form all points in that space. If all the vectors in the span are linearly independent then it is a minimal span, or basis, for the space. That space may exist, for convenience, in more dimensions than the number of vectors needed to describe it, i.e. as r m-vectors with $m>r$, and so can be thought of as a subspace. An r-dimensional subspace in an m-dimensional space

may be represented in matrix form by an $m \times r$ matrix with the basis vectors as columns, denoted here by $\mathtt{S}$. Now, a rank r matrix, $\mathtt{M}$, with $r < \min(m, n)$ is rank deficient and has linearly dependent rows and columns. Therefore, the columns of such a matrix can be taken as points in an r-dimensional subspace. The span of that subspace is called the *column space* of the matrix. Each matrix column can be described with a set of coefficients, or coordinates:

$$\mathbf{m}_j = \mathbf{s}_1\lambda_{1j} + \mathbf{s}_2\lambda_{2j} + \ldots + \mathbf{s}_r\lambda_{rj} = \mathtt{S}\boldsymbol{\lambda}_j \quad \forall j \tag{3.45}$$

Concatenating all these equations together horizontally gives the factorization equation with which we are now familiar:

$$\mathtt{M} = \begin{bmatrix} \mathbf{m}_1 & \mathbf{m}_2 & \cdots & \mathbf{m}_n \end{bmatrix} = \mathtt{S} \begin{bmatrix} \boldsymbol{\lambda}_1 & \boldsymbol{\lambda}_2 & \cdots & \boldsymbol{\lambda}_n \end{bmatrix} = \mathtt{S}\Lambda \equiv \mathtt{A}\mathtt{B}^\top \tag{3.46}$$

The order of the stacking does not affect the calculation of any one column's coefficients, drawing attention to invariance of factorization to column ordering. Indeed, any column-wise operations may be performed prior to factorization and the process is still valid. Undoing those operations on the matrix afterwards will always give a valid decomposition of the original matrix. Transposing $\mathtt{M}$ is also a valid pre-factorization event. That is, given $\mathtt{M} = \mathtt{A}\mathtt{B}^\top$, $\mathtt{M}' = \mathtt{A}'\mathtt{B}'^\top$ and $\mathtt{M}' = \mathtt{M}^\top$, then $\mathtt{A}' = \mathtt{B}$ and $\mathtt{B}' = \mathtt{A}$. As such, the terms 'row' and 'column' may be interchanged in these explanations (if done consistently) and we can always assume that $m > n$, i.e. that $\mathtt{M}$ is portrait.

A Special Case Solution

The idea of row and column spaces and them being coefficients for each other in the construction of their matrix, immediately suggests a way of filling a specific group of matrices with missing elements: those that have r complete (i.e. all elements are known), linearly independent columns. Let the matrix with missing elements be $\mathtt{M} \in \mathbb{R}^{m \times n}$ and be rank de-

ficient, i.e. $r < \min(m, n)$. Let the complete columns be $\{\mathbf{s}_{j'}\}$, for $j' = 1 \ldots r$. Let $\mathtt{S}$ be an $m \times r$ matrix with all these $\mathbf{s}_{j'}$ vectors as its columns. $\mathtt{S}$ is, of course, a valid column space of the matrix $\mathtt{M}$. All other columns are linear combinations of the columns of $\mathtt{S}$. Take $\mathbf{m}_j$ as one of the columns with missing elements. If at least r elements of $\mathbf{m}_j$ are known, then we have at least r equations of the form

$$s_{i'1}\lambda_1 + s_{i'2}\lambda_2 + \ldots + s_{i'r}\lambda_r = m_{i'j} \quad \forall i' \tag{3.47}$$

for i' the indices of the known elements of $\mathbf{m}_j$. By creating an r-vector from these known elements, $\hat{\mathbf{m}}_j$, the following equation can be formed to determine the coefficients for this column:

$$\hat{\mathtt{S}}_{(r\times r)}\boldsymbol{\lambda}_{(r\times 1)} = \hat{\mathbf{m}}_{j(r\times 1)} \tag{3.48}$$

with $\hat{\mathtt{S}}$ constructed by taking the rows of $\mathtt{S}$ corresponding to those elements taken from $\mathbf{m}$. As $\hat{\mathtt{S}}$ has full rank, this equation can be solved for the coefficients, $\boldsymbol{\lambda} = \hat{\mathtt{S}}^{-1}\hat{\mathbf{m}}_j$, and the missing elements can be found directly:

$$\mathbf{m}_j = \mathtt{S}\hat{\mathtt{S}}^{-1}\hat{\mathbf{m}}_j \tag{3.49}$$

Based on the above, an algorithm for rebuilding a rank-deficient matrix with missing elements, hence a factorization for it, can be formed. It is given in Algorithm 3.1. Of course, this algorithm will only work on matrices for which r full columns are known. Despite this being a rare occurrence in practice, Rother and Carlsson (2001) considered the SFM problem in which a 'ground-plane' was visible in all views and so satisfied the condition and motivated their algorithm based on this column space idea using the concept of infinite homographies.

It should be clear that the minimum number of elements that must be known for complete factorization is $r(m+n-r)$, which is illustrated by an example in Figure 3.5. We need an $m \times r$ element column space and n sets of r coefficients to describe all the columns. If

Algorithm 3.1 Factoring a matrix with missing elements in all but r columns.

inputs $\mathtt{M} \in \mathbb{R}^{m \times n}$, $\mathtt{W} \in \{0,1\}^{m \times n}$, $(m, n, r) \in \mathbb{Z}$

// $\mathtt{M}$ is rank r and has unknown values in all but r linearly independent columns

declare $\mathbf{v} \in \mathbb{N}^{n-r}$, $\mathtt{A} \in \mathbb{R}^{m \times r}$, $\mathtt{B} \in \mathbb{R}^{n \times r}$
$c \leftarrow 1$
$col \leftarrow 1$
while $(col - c < n - r)$ **do** *// fill $\mathtt{A}$ with r complete columns*
  if $\mathrm{sum}(\mathbf{w}_{col}) = m$ **then** *// column $\mathbf{m}_{col}$ is complete*
    $\mathbf{a}_c \leftarrow \mathbf{m}_{col}$
    $\mathbf{b}^{col} \leftarrow \mathbf{0}$
    $b_{col,c} \leftarrow 1$
    $c \leftarrow c + 1$
  else *// record column as having missing elements*
    $v_{col-c+1} \leftarrow col$
  end if
  $col \leftarrow col + 1$
end while
for $i = 1$ to $n - r$ **do** *// process the incomplete columns*
  $j \leftarrow v_i$
  declare $\hat{\mathbf{m}} \in \mathbb{R}^r$, $\hat{\mathtt{S}} \in \mathbb{R}^{r \times r}$
  $c \leftarrow 1$
  $row \leftarrow 1$
  while $(c < r)$ & $(row < m)$ **do** *// take r known elements of column*
    if $w_{col,j} = 1$ **then** *// element is known*
      $\hat{m}_c \leftarrow m_{row,j}$
      $\hat{\mathbf{s}}^c \leftarrow \mathbf{s}^{row}$
      $c \leftarrow c + 1$
    end if
    $row \leftarrow row + 1$
  end while
  if $c = r$ **then** *// r known elements found*
    $\mathbf{b}^j \leftarrow \hat{\mathtt{S}}^{-1}\hat{\mathbf{m}}$
  else
    cannot reconstruct this column
  end if
end for

outputs $\mathtt{A}$, $\mathtt{B}$

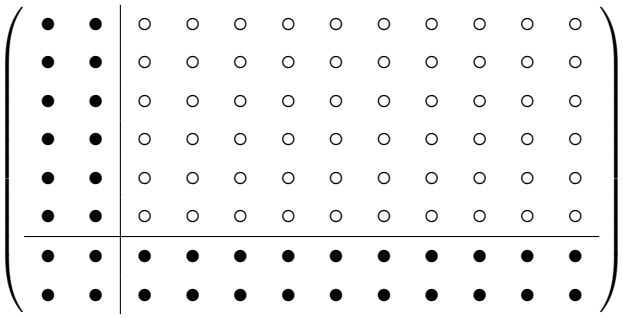


Figure 3.5: **Minimal configuration.** An example of a configuration of the smallest number of elements needed to reconstruct an 8×12 matrix if it has a rank of 2. Full circles represent known values and empty circles unknown values.

the column space is taken directly from the columns of the matrix itself, then r sets of r coefficients are already known to be the identity matrix and so can be subtracted from the count (this is the gauge freedom).

A General Solution

In general, to determine a specific unknown element of $\mathbf{m}_j$, $\hat{\mathtt{S}}$ need not be formed from complete columns; if the indices of r known elements in $\mathbf{m}_j$ are again i' and the index of an unknown element is i, then any columns of $\mathtt{M}$ with known values in the i^{th}, plus all the i'^{th} entries, can be used to determine m_{ij}. The algorithm is similar to Algorithm 3.1, but rather than starting with a search for an all encompassing column space, instead there are searches for partial column spaces in the inner loop. An outline is given in Algorithm 3.2.

It should now be easy to see that $\mathtt{M}$ must have at least r elements known on every column (otherwise deficient columns can not have their coefficients calculated) and at least r elements on every row (otherwise no other elements on that row can be determined). Obviously, these minimum conditions are necessary, but not sufficient. The configuration of missing elements is the crucial factor. Having exactly the minimum number of known values can be achieved with any matrix of the form shown in Figure 3.5, but it is easy to create matrices with these minimum properties that will fail (see Figure 3.6). Helpfully, the larger the ratio of matrix size to rank, the larger the number of elements that can be missing without

Algorithm 3.2 Rebuilding a matrix with missing elements.

inputs $\mathtt{M} \in \mathbb{R}^{m \times n}$, $\mathtt{W} \in \{0,1\}^{m \times n}$, $(m, n, r) \in \mathbb{Z}$

// $\mathtt{M}$ is rank r and has at least $r(m+n-r)$ known elements—see text.

repeat
  for $j = 1$ to n **do** *// each column*
    $l \leftarrow \mathrm{sum}(\mathbf{w}_j)$
    declare $\hat{\mathbf{m}}_j \in \mathbb{R}^l$, $\hat{\mathtt{S}} \in \mathbb{R}^{l \times r}$, $\hat{\mathbf{s}} \in \mathbb{R}^r$
    $\{i'\} \leftarrow$ indices of known elements in $\mathbf{m}_j$
    $\{i\} \leftarrow$ indices of unknown elements in $\mathbf{m}_j$
    $\hat{\mathbf{m}}_j \leftarrow m_{\{i'\}j}$
    for all $i \in \{i\}$ **do** *// each unknown element*
      $\{j'\} \leftarrow$ indices of all columns of $\mathtt{M}$ with known elements on rows $\{i'\} \cap i$
      if size($\{j'\}$) $\geq r$ **then**
        $\hat{\mathtt{S}} \leftarrow m_{\{i'\}\{j'\}}$
        $\hat{\mathbf{s}} \leftarrow m_{i\{j'\}}$
        $m_{ij} \leftarrow \left(\hat{\mathtt{S}}^{+}\hat{\mathbf{m}}_j\right)^{\top} \hat{\mathbf{s}}$
        $w_{ij} \leftarrow 1$
      end if
    end for
  end for
until matrix full

outputs $\mathtt{M}$

loss of information. This is particularly good in a SFM context because it tends to be easy to make the number of frames in a video sequence much larger than the rank constraint of the scene (unless one is after excessively non-rigid structure). The bad side of this view is that increasing the number of frames does not necessarily increase the number of tracked feature points and computers only have finite resources and are unable to deal with arbitrarily large computations.

Why is Algorithm 3.2 not the solution we are looking for? The reason is noise. Noisy measurements make it much harder to guess the original matrix, especially when only a proportion of the matrix entries are known. At this stage, there are two main strategies which help deal with noisy matrix entries. The first is to use as many elements as possible when calculating each column's coefficients, rather than only the $r(r+2)$ that are required when the data is noise-free. The pseudo-inverse is an invaluable tool in implementations of

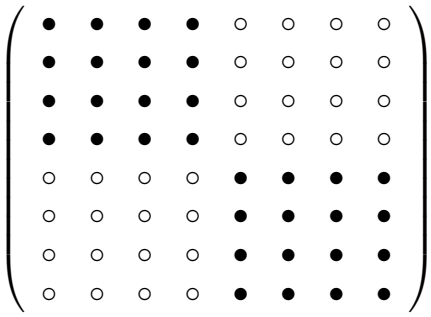


Figure 3.6: **Degenerate configuration.** An example of a configuration having the minimum number of elements for reconstruction, but which may not be factorized. The best that may be done is the factorization of the two blocks, but their connection can only be arbitrary.

the first strategy. The second strategy is to use newly calculated entries to calculate further coefficients. Note that Algorithm 3.2 has been presented in a way that incorporates both these strategies. Algorithm 3.2 is, in fact, a generalized version of an algorithm included in the original paper by Tomasi and Kanade (1992b). Their implementation uses only the second strategy, but it does explicitly deal with non-mean-centred measurement matrices. An initialization method in a paper by Guerreiro and Aguiar (2002b), which can be described as *column space stitching*, is an alternative, but still equivalent, implementation. Algorithm 3.2 has, very recently, been proposed again by Chen and Suter (2004). A perspective version has been suggested by Saito and Kamijima (2003), although it is more philosophically similar than algorithmically equivalent.

Error Magnification

One illustration of how having holes in the measurement matrix makes reconstruction, and hence factorization, hard was presented in Section 3.1.5. In the context of Algorithm 3.2, the difficulty can be seen as error propagation. Errors in the reconstruction of the first parts of the matrix are magnified when those parts are used in the calculations for the rest of the matrix. Successive mini-column spaces are adjusted to align with those previously extracted to build up the full column space. Each slightly noisy column space stitch adds in successively more error. In the full data case, the SVD uses all elements to get an idea of the noise and so

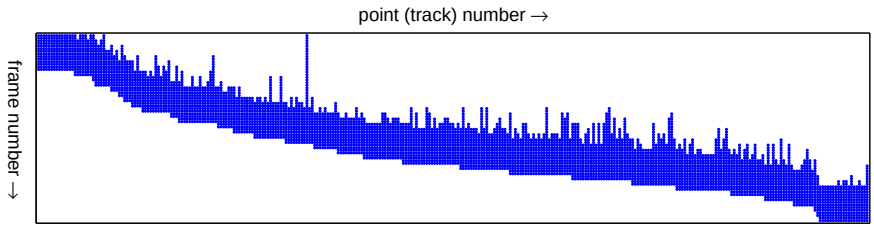


Figure 3.7: **Sparsity pattern.** The sparsity pattern of a real measurement matrix. This from the rotating dinosaur sequence (see Figure 3.1). The columns have been ordered by track end frame to emphasize the structure of the sparsity pattern. Note the similarity to Figure 3.8. As the toy dinosaur rotates, feature points appear and disappear from view depending on angle (i.e. frame number), giving the classic diagonal pattern.

deals with it in an optimal fashion. For the noisy missing data case, strategy one above is a step towards this. Strategy two tries to help, but ends up propagating errors. Unfortunately measurement matrices generated from real-life data (an example is shown in Figure 3.7) often have structures closer to that shown in Figure 3.8 than that in Figure 3.5 and the second strategy becomes a necessity.

Jacobs' Reconstruction Method

An alternative approach has been introduced by Jacobs (2001). Rather than effectively building up a column space, Jacobs suggests working down to one from the full m-dimensional space in which the column space exists. Jacobs shows that an orthogonal complement of $\mathtt{S}\equiv\mathtt{A}$ can be incrementally built up until it describes a subspace of the desired size, i.e. a span of $(m-r)$ m-vectors.

The orthogonal complement of a matrix $\mathtt{S}$ is defined as

$$\mathtt{N} = \mathrm{null}(\mathtt{S}^\top) \tag{3.50}$$

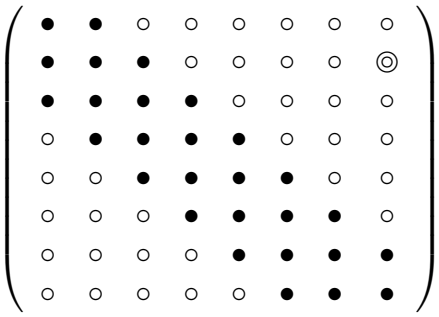


Figure 3.8: **Rank 2 reconstruction example.** Let filled be known and open unknown. The circled unknown entry can not be reconstructed directly as there are no other columns with known entries on the 2^{nd}, 7^{th} and 8^{th} rows. However, there are enough known elements to rebuild the matrix and hence factorize it. The 9^{th} element of every 3×3 block with 8 known elements can be calculated, creating more 3×3 blocks with 8 known entries. This recursive filling is implemented in Algorithm 3.2. It is fine for the noise-free situation, but errors propagate in the noisy matrix and become ever larger towards the two unknown corners. Note that this structure has the required $r(m+n-r)$ known elements.

It is the span of all the vectors that are orthogonal to the span of the vectors in $\mathtt{S}$, i.e. $\mathtt{S}^\top\mathtt{N} = \mathtt{0}$.

Jacobs' algorithm uses orthogonal complements to carve away at $\mathbb{R}^m$ to find the r-dimensional subspace from which the columns of $\mathtt{M}$ have been taken. As the algorithm progresses, $\mathtt{N}$ is built up. The final step is to set $\mathtt{S} = \mathrm{null}(\mathtt{N}^\top)$. An advantage here is that when dealing with a noisy measurement matrix, an excess of columns may be added to $\mathtt{N}$ by using as many elements from $\hat{\mathtt{M}}$ as possible, so that all those values can be used in the calculation of $\mathtt{S}$. The final step would then be a minimization step rather than an actual null space calculation, but it is easily implemented with the SVD.

See Algorithm 3.3 for an overview of the general process. For a noise-free $\mathtt{M}$, take r columns of $\mathtt{M}$ to form a matrix $\mathtt{S}_i$. If all the elements of $\mathtt{S}_i$ are known then it is a valid column space and the process can terminate. If they are not, then the algorithm proceeds as follows. Firstly, note and remove all the rows of $\mathtt{S}_i$ in which there is an unknown element to form $\hat{\mathtt{S}}_i$. Calculate $\hat{\mathtt{N}}_i$, the orthogonal complement of $\hat{\mathtt{S}}_i$. Form $\mathtt{N}_i$ by inserting into $\hat{\mathtt{N}}_i$ zero rows corresponding to those that were removed from $\mathtt{S}_i$. Concatenate $\mathtt{N}_i$ onto $\mathtt{N}$. If $\mathtt{N}$ is the required size, i.e. has at least $(m-r)$ columns, and is rank r, then set $\mathtt{S} = \mathrm{null}(\mathtt{N}^\top)$, otherwise continue. Every span, $\mathtt{N}_i$, that is added to $\mathtt{N}$ is orthogonal to $\mathtt{S}$ and so when $\mathtt{N}$ is complete it

Algorithm 3.3 Jacobs' matrix factorization method.

inputs $\mathtt{M} \in \mathbb{R}^{m \times n}$, $\mathtt{W} \in \{0,1\}^{m \times n}$, $(m, n, r) \in \mathbb{Z}$

declare $\mathtt{S} \in \mathbb{R}^{m \times r}$, $\mathtt{S}_i \in \mathbb{R}^{m \times r}$, $\mathtt{N} \in \mathbb{R}^{m \times 0}$
for i all n-choose-r column selections **do**
  $\mathtt{S}_i \leftarrow i^{th}$ r-tuple of columns of $\mathtt{M}$
  $\{i'\} \leftarrow$ indices of complete rows in $\mathtt{S}_i$
  $k \leftarrow$ size $\{i'\}$
  if $k > r$ **then**
    declare $\hat{\mathtt{S}}_i \in \mathbb{R}^{k \times r}$
    $\hat{\mathtt{S}}_i \leftarrow$ the $\{i'\}$ rows of $\mathtt{S}_i$
    $l \leftarrow k - \mathrm{rank}(\hat{\mathtt{S}}_i)$
    declare $\mathtt{N}_i \in \mathbb{R}^{m \times l}$, $\hat{\mathtt{N}}_i \in \mathbb{R}^{k \times l}$
    $\hat{\mathtt{N}}_i \leftarrow \mathrm{null}(\hat{\mathtt{S}}_i^\top)$
    $\mathtt{N}_i \leftarrow$ rows $\{i'\}$ from $\hat{\mathtt{N}}_i$, $\mathbf{0}^\top$ otherwise
    $\mathtt{N} \leftarrow \begin{bmatrix} \mathtt{N} & \mathtt{N}_i \end{bmatrix}$
  end if
end for

outputs $\mathtt{S} = \mathrm{null}(\mathtt{N}^\top)$

exactly defines $\mathtt{S}$.

The procedure of truncating $\mathtt{S}_i$ and then rebuilding $\mathtt{N}_i$ afterwards is an algorithmic way of dealing with the fact that unknown elements represent unknown freedom in the subspace $\mathtt{S}_i$. To understand this, consider an example $\mathtt{S}_i$—two columns taken from a rank 2 matrix (with asterisks representing unknown entries)—and the formulation of its orthogonal complement:

$$\begin{pmatrix} * & 3 & 2 & 7 & 5 \\ 2 & 4 & * & 1 & 3 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \\ e \end{pmatrix} = \mathbf{0} \tag{3.51}$$

Because there could be any value in the positions taken by the asterisks, a and c must be set to zero, leaving b, d and e to be calculated from the second, fourth and fifth columns of $\mathtt{S}_i^\top$. The vector $(0, b, 0, d, e)^\top$ is still perpendicular to $\mathtt{S}_i$ and so is a member of the subspace defined by $\mathtt{N}_i = \mathrm{null}(\mathtt{S}_i^\top)$.

When processing a noisy measurement matrix, the conditions for stopping are less well defined. In Algorithm 3.3, it is suggested that all column selections should be tested, but n-choose-r can easily be very large and so alternatives must be employed in practice. For example, simply randomize the selection of r-tuples of columns and set a hard limit for the number of iterations (Jacobs' own implementation uses this method). Alternatively, the singular values of $\mathtt{N}$ could be monitored as it grows beyond the minimum $(m-r)$ columns and termination could occur when they stop changing (potentially very costly). Taking more than r columns to form each $\mathtt{S}_i$ is another way to help overcome the problem of noise. Unfortunately, the more columns that are chosen, the smaller the number of complete rows. The minimum number of complete rows in $\mathtt{S}_i$ that is needed is $r+1$ because an $\hat{\mathtt{S}}_i$ from a noisy measurement matrix will invariably have full rank and it must have a nullity of at least one for $\hat{\mathtt{N}}_i$ to exist. A subtle alternative to Algorithm 3.3 takes selections of $r+1$ rows and collects all the columns with known elements on those rows. The full version would process m-choose-$(r+1)$ row combinations, but is otherwise the same. To minimize execution time, the transpose of a measurement matrix should always be considered as the input, though the correct orientation to use depends on the implementation. The time complexities of the two versions when all row or column combinations are worked through is dominated by the number of possibilities. A time complexity of $O(k\text{-choose-}l)$ can be as bad as $O(2^k)$, but here (with $k \ll l$) it is approximately $O(k^l)$. So, for the column-wise approach, an input matrix with fewer columns than rows is better, whereas for the row-wise approach the converse is preferable. If the number of iterations is fixed in some way, then the dominant operation within the loop is the important factor. In both versions, it is the null space calculation that will influence the time complexity. The SVD can be used for that operation and so the time complexities become $O(mr^2)$ for column-wise and $O(n(r+1)^2)$ for row-wise, meaning that portrait matrices should now be given to the row-wise algorithm and vice-versa.

As Jacobs points out in his paper, this algorithm is not optimal, but he suggests that it is a good initialization technique for iterative schemes. The most notable developments are

that of Martinec and Pajdla who firstly extended this approach (Martinec and Pajdla 2002; 2003) with the unknown scale factors in the projective SFM solution due to Triggs (1996) and then developed an almost equivalent method, but working in 'normal' space rather than 'null' space (Martinec and Pajdla 2005) which deals much better with noise.

3.2.2 Alternation Approaches

Firstly, consider the error function of simple factorization:

$$F(\mathtt{A},\mathtt{B}) = \left\| \mathtt{M} - \mathtt{A}\mathtt{B}^\top \right\|_F^2 \tag{3.52}$$

From a given point on the error surface, closed form solutions for new arguments that minimize the function in each variable can be derived in various ways (see Appendix B). Alternation is the strategy of solving for one and then the other variable, iteratively in turn. Algorithm 3.4 demonstrates the simplicity of the alternation scheme. In most cases it will converge on the answer quickly, although it is prone to crawling along shallow valleys when they are encountered.

Algorithm 3.4 Alternation for factorization: $\operatorname{argmin}\{F(\mathtt{A},\mathtt{B}) = \|\mathtt{M} - \mathtt{A}\mathtt{B}^\top\|_F^2\}$

inputs $\mathtt{M} \in \mathbb{R}^{m\times n}$, $\mathtt{A} \in \mathbb{R}^{m\times r}$, $\mathtt{B} \in \mathbb{R}^{n\times r}$, $(m,n,r) \in \mathbb{Z}$

1: **repeat**
2: $\quad \mathtt{A}^\top \leftarrow \left(\mathtt{B}^\top\mathtt{B}\right)^{-1}\mathtt{B}^\top\mathtt{M}^\top$
3: $\quad \mathtt{B}^\top \leftarrow \left(\mathtt{A}^\top\mathtt{A}\right)^{-1}\mathtt{A}^\top\mathtt{M}$
4: **until** no change

outputs A, B

This algorithm can be extended slightly by allowing entire rows and columns of A and B respectively, to be weighted (for example, in the context of SFM, if there is more confidence in the tracking of certain points, or in the detection of points in certain views). $\mathtt{W}_{rows}$ and

$\mathtt{W}_{cols}$ are diagonal matrices to allow

$$F(\mathtt{A},\mathtt{B}) = \left\|\mathtt{W}_{rows}(\mathtt{M} - \mathtt{A}\mathtt{B}^\top)\mathtt{W}_{cols}\right\|_F^2 \tag{3.53}$$

Again, a simple alternation strategy can be derived, as shown in Algorithm 3.5.

Algorithm 3.5 Row and column weighted alternation: $\operatorname{argmin}\{F(\mathtt{A},\mathtt{B})=\|\mathtt{W}_{rows}(\mathtt{M}-\mathtt{A}\mathtt{B}^\top)\mathtt{W}_{cols}\|_F^2\}$

inputs $\mathtt{M}\in\mathbb{R}^{m\times n}$, $\mathtt{A}\in\mathbb{R}^{m\times r}$, $\mathtt{B}\in\mathbb{R}^{n\times r}$, $\mathtt{W}_{rows}\in \operatorname{diag}\mathbb{R}^m$, $\mathtt{W}_{cols}\in \operatorname{diag}\mathbb{R}^n$, $(m,n,r)\in\mathbb{Z}$

1: **repeat**
2: $\quad \mathtt{A}^\top \leftarrow \left(\mathtt{B}^\top\mathtt{W}_{cols}^2\mathtt{B}\right)^{-1}\mathtt{B}^\top\mathtt{W}_{cols}^2\mathtt{M}^\top$
3: $\quad \mathtt{B}^\top \leftarrow \left(\mathtt{A}^\top\mathtt{W}_{rows}^2\mathtt{A}\right)^{-1}\mathtt{A}^\top\mathtt{W}_{rows}^2\mathtt{M}$
4: **until** no change

outputs A, B

Weighting on entire rows and/or columns is a bit too restrictive and, indeed, unhelpful when it comes to missing data problems. Fortunately, it is easy to extend alternation further, to include element-wise weighting. We now go to the error function

$$F(\mathtt{A},\mathtt{B}) = \left\|\ \mathtt{W}\odot(\mathtt{M} - \mathtt{A}\mathtt{B}^\top)\right\|_F^2 \tag{3.54}$$

and proceed as above. The difference, having introduced the Hadamard product, is that the equations may only be collected to the vector level and not to the matrix level as with the above two formulations. This gives Algorithm 3.6.

Algorithm 3.6 Weighted alternation: $\operatorname{argmin}\{F(\mathtt{A},\mathtt{B}) = \|\ \mathtt{W}\odot(\mathtt{M} - \mathtt{A}\mathtt{B}^\top)\|_F^2\}$

inputs $\mathtt{M}\in\mathbb{R}^{m\times n}$, $\mathtt{A}\in\mathbb{R}^{m\times r}$, $\mathtt{B}\in\mathbb{R}^{n\times r}$, $\mathtt{W}\in\mathbb{R}^{m\times n}$, $(m,n,r)\in\mathbb{Z}$

1: **repeat**
2: $\quad \forall i \quad \mathbf{a}^i \leftarrow \left(\mathtt{B}^\top\operatorname{diag}(\mathbf{w}^i)^2\mathtt{B}\right)^{-1}\mathtt{B}^\top\operatorname{diag}(\mathbf{w}^i)^2\mathbf{m}^i$
3: $\quad \forall j \quad \mathbf{b}^j \leftarrow \left(\mathtt{A}^\top\operatorname{diag}(\mathbf{w}_j)^2\mathtt{A}\right)^{-1}\mathtt{A}^\top\operatorname{diag}(\mathbf{w}_j)^2\mathbf{m}_j$
4: **until** no change

outputs A, B

In the specific application of affine SFM an additional constraint arises: the translation of

coordinate spaces. As seen in Equation 3.11 above, this adds an extra condition:

$$\mathtt{M} = \mathtt{P}\mathtt{X} + \mathbf{t}\mathbf{1}^\top \tag{3.55}$$

$$= \begin{bmatrix} \mathtt{P} & \mathbf{t} \end{bmatrix} \begin{bmatrix} \mathtt{X} \\ \mathbf{1}^\top \end{bmatrix} \tag{3.56}$$

namely, that the last column of $\mathtt{B}$ should be made up of ones (i.e. the three-dimensional scene points lie on an offset hyperplane in projective space). The alternation algorithm can be modified to incorporate this, giving Algorithm 3.7, overcoming the problem of mean-centring the measurement matrix when elements are unknown: the iteration scheme in Algorithm 3.7 estimates both the offset 3D hyperplane and the centroid of the data (as described in Section 3.1).

Algorithm 3.7 Alternation for SFM: $\operatorname{argmin}\{F(\mathtt{A},\mathtt{B}) = \|\,\mathtt{W} \odot (\mathtt{M} - \mathtt{A}\mathtt{B}^\top)\|_F^2\}$ with $\mathbf{b}_r = \mathbf{1}$

inputs $\mathtt{M} \in \mathbb{R}^{m\times n}$, $\mathtt{A} \in \mathbb{R}^{m\times r}$, $\mathtt{B} \in \mathbb{R}^{n\times r}$, $\mathtt{W} \in \mathbb{R}^{m\times n}$, $(m,n,r) \in \mathbb{Z}$

1: **declare** $\mathtt{X} \in \mathbb{R}^{n\times r-1}$, $\mathtt{P} \in \mathbb{R}^{m\times r-1}$, $\mathbf{t} \in \mathbb{R}^m$
2: **repeat**
3: $\quad \mathbf{a}^i \leftarrow \left(\mathtt{B}^\top \operatorname{diag}(\mathbf{w}^i)^2 \mathtt{B}\right)^{-1} \mathtt{B}^\top \operatorname{diag}(\mathbf{w}^i)^2 \mathbf{m}_j \quad \forall i$
4: $\quad \begin{bmatrix} \mathtt{P} & \mathbf{t} \end{bmatrix} \leftarrow \mathtt{A}$
5: $\quad \mathbf{x}_j \leftarrow \left(\mathtt{P}^\top \operatorname{diag}(\mathbf{w}_j)^2 \mathtt{P}\right)^{-1} \mathtt{P}^\top \operatorname{diag}(\mathbf{w}_j)^2 (\mathbf{m}_j - \mathbf{t}) \quad \forall j$
6: $\quad \mathtt{B} \leftarrow \begin{bmatrix} \mathtt{X}^\top & \mathbf{1} \end{bmatrix}$
7: **until** no change

outputs $\mathtt{A}$, $\mathtt{B}$

After each iteration, all these algorithms reduce the error. It is therefore guaranteed that they all reach a local minimum.

In 1976, Wiberg (1976) introduced Algorithm 3.7. Shum et al. (1995) came back to it in 1995 and, with minor algorithmic modifications, applied it to the measurement matrix of Tomasi and Kanade's factorization approach. More recently, Guerreiro and Aguiar (2002a;b; 2003) have presented this algorithm again. Aguiar and Moura (2000; 2003) have suggested using the slightly less useful Algorithm 3.5. It was shown by Roweis (1997) that alternation

can, in fact, be derived within an EM framework.

There have also been several propositions that are variations on Wiberg's approach. Hartley and Schaffalitzky (2003) suggest adding a normalization step ("PowerFactorization" as employed by Vidal and Hartley (2004)—Algorithm 3.8). Huynh et al. (2003) proposed performing alternation on a continually updated version of $\mathtt{M}$. Algorithm 3.9 outlines their algorithm.

Algorithm 3.8 "PowerFactorization": $\operatorname{argmin}\{F(\mathtt{A},\mathtt{B}) = \|\,\mathtt{W} \odot (\mathtt{M} - \mathtt{A}\mathtt{B}^\top)\|_F^2\}$

inputs $\mathtt{M} \in \mathbb{R}^{m\times n}$, $\mathtt{A} \in \mathbb{R}^{m\times r}$, $\mathtt{B} \in \mathbb{R}^{n\times r}$, $\mathtt{W} \in \mathbb{R}^{m\times n}$, $(m,n,r) \in \mathbb{Z}$

1: **repeat**
2: $\quad \mathbf{b}^j \leftarrow \left(\mathtt{A}^\top \operatorname{diag}(\mathbf{w}_j)^2 \mathtt{A}\right)^{-1} \mathtt{A}^\top \operatorname{diag}(\mathbf{w}_j)^2 \mathbf{m}_j \quad \forall j$
3: $\quad \mathtt{B} \leftarrow$ column-wise orthonormalization of $\mathtt{B}$
4: $\quad \mathbf{a}^i \leftarrow \left(\mathtt{B}^\top \operatorname{diag}(\mathbf{w}^i)^2 \mathtt{B}\right)^{-1} \mathtt{B}^\top \operatorname{diag}(\mathbf{w}^i)^2 \mathbf{m}_j \quad \forall i$
5: **until** no change

outputs $\mathtt{A}$, $\mathtt{B}$

Algorithm 3.9 Huynh et al. (2003): $\operatorname{argmin}\{F(\mathtt{A},\mathtt{B}) = \|\,\mathtt{W} \odot (\mathtt{M} - \mathtt{A}\mathtt{B}^\top)\|_F^2\}$

inputs $\mathtt{M} \in \mathbb{R}^{m\times n}$, $\mathtt{A} \in \mathbb{R}^{m\times r}$, $\mathtt{B} \in \mathbb{R}^{n\times r}$, $\mathtt{W} \in \mathbb{R}^{m\times n}$, $(m,n,r) \in \mathbb{Z}$

1: **declare** $\mathtt{X} \in \mathbb{R}^{n\times r-1}$, $\mathtt{P} \in \mathbb{R}^{m\times r-1}$, $\mathbf{t} \in \mathbb{R}^m$
2: **repeat**
3: $\quad \mathbf{a}^i \leftarrow \left(\mathtt{B}^\top \operatorname{diag}(\mathbf{w}^i)^2 \mathtt{B}\right)^{-1} \mathtt{B}^\top \operatorname{diag}(\mathbf{w}^i)^2 \mathbf{m}_j \quad \forall i$
4: $\quad \mathtt{M} \leftarrow$ update elements of $\mathtt{M}$ with large residuals
5: $\quad \begin{bmatrix} \mathtt{P} & \mathbf{t} \end{bmatrix} \leftarrow \mathtt{A}$
6: $\quad \mathbf{x}_j \leftarrow \left(\mathtt{P}^\top \operatorname{diag}(\mathbf{w}_j)^2 \mathtt{P}\right)^{-1} \mathtt{P}^\top \operatorname{diag}(\mathbf{w}_j)^2 (\mathbf{m}_j - \mathbf{t}) \quad \forall j$
7: $\quad \mathtt{B} \leftarrow \begin{bmatrix} \mathtt{X}^\top & \mathbf{1} \end{bmatrix}$
8: **until** no change

outputs $\mathtt{A}$, $\mathtt{B}$

Aanæs et al. (2002) have put forward another method that works on an updated version of the measurement matrix. They use one alternation step after a subspace projection to give Algorithm 3.10. Although not actually related, Guerreiro and Aguiar (2003; 2002a;b), along with basic alternation, also present a similar *project and merge* iteration scheme. See Algorithm 3.11.

Many of the algorithms here do not explicitly deal with the fact that a measurement

Algorithm 3.10 Aanæs et al. (2002): $\operatorname{argmin}\{F(\mathtt{A},\mathtt{B}) = \|\,\mathtt{W}\odot(\mathtt{M}-\mathtt{A}\mathtt{B}^\top)\|_F^2\}$

inputs $\mathtt{M}\in\mathbb{R}^{m\times n}$, $\mathtt{A}\in\mathbb{R}^{m\times r}$, $\mathtt{B}\in\mathbb{R}^{n\times r}$, $\mathtt{W}\in\mathbb{R}^{m\times n}$, $(m,n,r)\in\mathbb{Z}$

declare $\hat{\mathtt{M}}\in\mathbb{R}^{m\times n}$
$\hat{\mathtt{M}} = \mathtt{M}$
repeat
  $\mathtt{A}\leftarrow\hat{\mathtt{M}}\downarrow_3$
  $\mathbf{b}^j\leftarrow\left(\mathtt{A}^\top\operatorname{diag}(\mathbf{w}_j)^2\mathtt{A}\right)^{-1}\mathtt{A}^\top\operatorname{diag}(\mathbf{w}_j)^2\mathbf{m}_j\quad\forall j$
  $\hat{\mathtt{M}}\leftarrow\mathtt{W}\odot\mathtt{M}+(\mathtt{1}-\mathtt{W})\odot(\mathtt{A}\mathtt{B}^\top)$
until no change

outputs A, B

Algorithm 3.11 Guerreiro & Aguiar's two-step algorithm: $\operatorname{argmin}\{F(\mathtt{A},\mathtt{B})=\|\,\mathtt{W}\odot(\mathtt{M}-\mathtt{A}\mathtt{B}^\top)\|_F^2\}$

inputs $\mathtt{M}\in\mathbb{R}^{m\times n}$, $\mathtt{W}\in\mathbb{R}^{m\times n}$, $(m,n,r)\in\mathbb{Z}$

declare $\tilde{\mathtt{M}}\in\mathbb{R}^{m\times n}$, $\hat{\mathtt{M}}\in\mathbb{R}^{m\times n}$
$\tilde{\mathtt{M}}\leftarrow\mathtt{M}\downarrow_r$
repeat
  $\hat{\mathtt{M}}\leftarrow\mathtt{W}\odot\mathtt{M}+(\mathtt{1}-\mathtt{W})\odot\tilde{\mathtt{M}}$
  $\tilde{\mathtt{M}}\leftarrow\hat{\mathtt{M}}\downarrow_r$
until no change

outputs $\tilde{\mathtt{M}}$

matrix with missing entries cannot be mean-centred. Brandt (2002) has formulated a closed form solution for B directly addressing this point. He has built an algorithm very similar to Algorithm 3.10, but using his equations to update B (which are too involved to present here). A summary of how all the alternation algorithms presented above are related is given in Figure 3.9.

Finally, a point that must be raised is the initialization of these algorithms. All require an initial estimate of A and B. Although alternation is guaranteed to reach a minimum, it will only be a local minimum and so the starting guess is important. The answer returned by alternation schemes is very sensitive to initialization (see Figure 3.15). The algorithms of Section 3.2.1 can be used to fill the measurement matrix to obtain a place from which to start. The only other alternative is to set up A and B with random values and see where it goes. In Section 3.4 it is shown how effective these starting strategies are.

ALT	PF	SIR	HHH	BDT	AFAC	GA	
							inputs: A,B, M, W, r
1	1	1	1	1	1	1	1: $\hat{\mathtt{M}} \leftarrow \mathtt{M}$
2	2	2	2	2	2	2	2: **repeat**
						3	3: $\tilde{\mathtt{M}} \leftarrow$ truncate $\hat{\mathtt{M}}$ to rank r (via SVD)
4	4	4	4				4: $\mathbf{a}^i \leftarrow (\mathtt{B}^\top \mathrm{diag}(\mathbf{w}^i)^2 \mathtt{B} + \lambda_1 \mathtt{I})^{-1} \mathtt{B}^\top \mathrm{diag}(\mathbf{w}^i)^2 \hat{\mathbf{m}}_i \quad \forall i$
			5				5: $\hat{\mathtt{M}} \leftarrow$ update elements of $\hat{\mathtt{M}}$ with large residuals
				6	6		6: $\mathtt{A} \leftarrow$ first left singular vectors of $\hat{\mathtt{M}}$
		7	7				7: $\begin{bmatrix} \mathtt{A} & \mathbf{t} \end{bmatrix} \leftarrow \mathtt{A}$; $\tilde{\mathtt{M}} \leftarrow \hat{\mathtt{M}} - \mathbf{t}\mathbf{1}^\top$
8	8	8			8		8: $\mathbf{b}^j \leftarrow (\mathtt{A}^\top \mathrm{diag}(\mathbf{w}_j)^2 \mathtt{A} + \lambda_2 \mathtt{I})^{-1} \mathtt{A}^\top \mathrm{diag}(\mathbf{w}_j)^2 \hat{\mathbf{m}}_j \quad \forall j$
	9						9: $\mathtt{B} \leftarrow$ column-wise orthonormalization of B
				10			10: $\mathtt{B} \leftarrow$ Brandt closed-form update of B
		11	11	11			11: $\mathtt{B} \leftarrow \begin{bmatrix} \mathtt{B} & \mathbf{1} \end{bmatrix}$; $\mathtt{A} \leftarrow \begin{bmatrix} \mathtt{A} & \mathbf{t} \end{bmatrix}$
		12	12				12: $\hat{\mathtt{M}} \leftarrow \tilde{\mathtt{M}} + \mathbf{t}\mathbf{1}^\top$
					13		13: $\tilde{\mathtt{M}} \leftarrow (\mathtt{A}\mathtt{B}^\top)$
					14	14	14: $\hat{\mathtt{M}} \leftarrow \mathtt{W} \odot \mathtt{M} + (1 - \mathtt{W}) \odot \tilde{\mathtt{M}}$
15	15	15	15	15	15	15	15: **until** convergence
•	•	•	•	•	•		**outputs:** A, B
						•	**outputs:** $\tilde{\mathtt{M}}$

Figure 3.9: **Algorithm comparison.** A table to compare basic alternation with the variants for the minimization of equation 3.5. Columns denote which lines are used by which algorithms. ALT = alternation; PF = PowerFactorization; SIR = Shum, Ikeuchi, and Reddy; HHH = Huynh, Hartley, and Heyden; AFAC = Aanaes, Fisker, Åström, and Carstensen; GA = Guerreiro and Aguiar; BDT = Brandt.

3.2.3 Other Literature

Here is a very brief review of other literature that has not been included above.

Batch methods, such as those employed by Fitzgibbon and Zisserman (1998) and Guilbert and Bartoli (2003), have not been tested for comparison. These necessarily work on subsections of the measurement matrix and proceed in a hierarchical fashion to the full solution. Although it is desirable to use as much of the data in one go in order to see past the noise and avoid error magnification (as described in Section 3.2.1), the size of the problems can make an immediate solution intractable. Working on smaller sub-problems and combining the results is a way to cope with large problems. Furthermore, initially considering only short sub-sequences also has the advantage that much lower levels of missing data can be expected, which means that direct solutions, such as those used by the above authors, are applicable. Another example is the projective solution of Schaffalitzky et al. (2000), which requires six point tracks to be visible across at least four frames, with the advantage that if the

six tracks extend beyond four frames, the measurement data from the entire (sub-)sequence in which they exist can be included in the 'leaf-node' solution. Such batch methods would be an interesting avenue for future work.

Also, yet to be investigated are extensions to principal component analysis (PCA). A few authors (Brand 2002, Skocaj and Leonardis 2002) have looked at incremental PCA, which in itself is not directly helpful to solving the missing data problem. However, the schemes naturally include the ability to weight data as they are incorporated into the analysis.

Robustness is an attribute omitted by the algorithms in this report, despite its importance. On this topic, the work by Torre and Black (2001), who include an outlier term in the error function for their robust PCA and so outlier rejection becomes part of the minimization, is very interesting.

Another approach that has a high potential is expectation-maximization (EM). It has been employed by a couple of authors (Aguiar and Moura 1999, Gruber and Weiss 2004), but most notably by Torresani and Hertzmann (2004), Torresani et al. (2001; 2004), Bregler et al. (2000). There is also work outside the computer vision field on accelerating and adjusting EM algorithms (Jamshidian and Jennrich 1997), as well as on the equivalence between EM and gradient methods (Lange 1995a).

Interestingly, robust PCA and EM algorithms can be described as alternation schemes in that, for each iteration, they consider subsets of parameters in turn while assuming the others are fixed. The robust PCA of Torre and Black (2001) does not use closed form solutions, but EM algorithms do, hence a strong connection. Yuille et al. (1999) also propose an algorithm that falls into this category. They tackle the problem of illumination based reconstruction with a method that incorporates shadow detection (i.e. partial outlier rejection—specularities were not included). Their scheme updates one set of parameters using a closed form solution and another two sets of parameters using gradient descent within each iteration. At the end of each iteration, they update their weight matrix based on the updated parameter values.

The work of Irani and Anandan (2002) on factorization with uncertainty is also inter-

esting, although it does not address the missing data problem itself. They rearrange the measurement matrix from a $2F \times P$ matrix into a $F \times 2P$ form with x and y image coordinates in separate columns. The alternative matrix layout allows the Mahalanobis distance (between factorization and measurement) to be minimized directly.

Finally, it is worth mentioning that this chapter covers some of the same ground as the bundle adjustment review by Triggs et al. (2000), but here the general problem of factorization when data are missing is investigated, rather than looking generally at the specific problem of SFM.

3.3 Error Surface Exploration

Let us continue looking at the missing-data error function

$$F(\mathtt{A},\mathtt{B}) = \left\| \mathtt{W} \odot \left(\hat{\mathtt{M}} - \mathtt{A}\mathtt{B}^\top\right) \right\|_F^2 \tag{3.57}$$

Section 3.2.2 dealt with alternation, which can be viewed as a coordinate-descent scheme, where the error surface is descended in each dimension of the parameter space in turn until there is nowhere else to go. Of course, there are several more ways to skate down the error surface. Here, gradient descent, Newton and the damped-Newton method are considered. These have not been investigated as methods to minimize Equation 3.57 in the computer vision literature. A hybrid method is also proposed.

3.3.1 Gradient Descent

As in Section 3.1.5, let us vectorize the arguments to the error function, A and B, as $\mathbf{x}$, for example, row-wise vectorization:

$$\mathbf{x} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1r} & \cdots & a_{mr} & b_{11} & b_{12} & \cdots & b_{nr} \end{pmatrix}^\top \tag{3.58}$$

Algorithm 3.12 Gradient descent: $\operatorname{argmin}\{F(\mathtt{A},\mathtt{B}) = \|\,\mathtt{W} \odot (\mathtt{M} - \mathtt{A}\mathtt{B}^\top)\|_F^2\}$

inputs $\mathtt{M} \in \mathbb{R}^{m\times n}$, $\mathtt{A} \in \mathbb{R}^{m\times r}$, $\mathtt{B} \in \mathbb{R}^{n\times r}$, $\mathtt{W} \in \mathbb{R}^{m\times n}$, $(m,n,r) \in \mathbb{Z}$

1: **declare** function $F = \|\,\mathtt{W} \odot (\mathtt{M} - \mathtt{A}\mathtt{B}^\top)\|_F^2$
2: **declare** $\mathbf{x} \in \mathbb{R}^{r(m+n)}$, $\mathbf{d} \in \mathbb{R}^{r(m+n)}$
3: $\mathbf{x} \leftarrow$ vectorize(A,B)
4: **repeat**
5: $\quad \mathbf{d} \leftarrow \frac{\partial F}{\partial \mathbf{x}}$
6: $\quad \lambda \leftarrow$ minimize $F(\mathbf{x} - \lambda\mathbf{d})$
7: $\quad \mathbf{x} \leftarrow \mathbf{x} - \lambda\mathbf{d}$
8: **until** no change

outputs $\mathtt{A}, \mathtt{B} \leftarrow$ unvectorize($\mathbf{x}$)

Further, let us consider the error function in terms of individual matrix elements,

$$F(\mathbf{x}) = \sum_{i=1}^{m}\sum_{j=1}^{n}\left[w_{ij}\left(m_{ij} - \sum_{k=1}^{r} a_{ik}b_{jk}\right)\right]^2 \tag{3.59}$$

$$= \sum_{i=1}^{m}\sum_{j=1}^{n} w_{ij}^2\left(m_{ij}^2 - 2m_{ij}\sum_{k=1}^{r} a_{ik}b_{jk} + \left[\sum_{k=1}^{r} a_{ik}b_{jk}\right]^2\right) \tag{3.60}$$

from which the partial derivatives of F with respect to all the elements of A and B can be more easily seen (see Appendix B).

$$\frac{\partial F}{\partial a_{ab}} = -2\sum_{j=1}^{n} w_{aj}b_{jb}\left(m_{aj} - \sum_{k=1}^{r} a_{ak}b_{jk}\right) \tag{3.61}$$

$$\frac{\partial F}{\partial b_{cd}} = -2\sum_{i=1}^{n} w_{ic}a_{id}\left(m_{ic} - \sum_{k=1}^{r} a_{ik}b_{ck}\right) \tag{3.62}$$

With these partial derivatives, the complete vector derivative of F can be calculated.

The gradient descent algorithm performs a line search in the direction of the gradient at the current point, and moves to a point on the line that has lower error. Logarithmic or bisection-style searches can be employed, though once a few points on the surface are known then quadratic or cubic fits might provide a faster descent for problems in a small number of dimensions. See Algorithm 3.12.

Algorithm 3.13 Newton's method: $\operatorname{argmin}\{F(\mathtt{A},\mathtt{B}) = \|\,\mathtt{W} \odot (\mathtt{M} - \mathtt{A}\mathtt{B}^\top)\|_F^2\}$

inputs $\mathtt{M} \in \mathbb{R}^{m\times n}$, $\mathtt{A} \in \mathbb{R}^{m\times r}$, $\mathtt{B} \in \mathbb{R}^{n\times r}$, $\mathtt{W} \in \mathbb{R}^{m\times n}$, $(m,n,r) \in \mathbb{Z}$

1: **declare** function $F = \|\,\mathtt{W} \odot (\mathtt{M} - \mathtt{A}\mathtt{B}^\top)\|_F^2$
2: **declare** $\mathbf{x} \in \mathbb{R}^{r(m+n)}$, $\mathbf{d} \in \mathbb{R}^{r(m+n)}$, $\mathtt{H} \in \mathbb{R}^{r(m+n)\times r(m+n)}$
3: $\mathbf{x} \leftarrow$ vectorize(A,B)
4: **repeat**
5: $\quad \mathbf{d} \leftarrow \frac{\partial F}{\partial \mathbf{x}}$
6: $\quad \mathtt{H} \leftarrow \frac{\partial^2 F}{\partial \mathbf{x}^2}$
7: $\quad \lambda \leftarrow$ minimize $F(\mathbf{x} - \lambda \mathtt{H}^{-1}\mathbf{d})$
8: $\quad \mathbf{x} \leftarrow \mathbf{x} - \lambda \mathtt{H}^{-1}\mathbf{d}$
9: **until** no change

outputs $\mathtt{A}, \mathtt{B} \leftarrow$ unvectorize($\mathbf{x}$)

3.3.2 Newton

The second derivatives of the error function are also readily available and so Newton's method can be employed. The function is approximated as being quadratic, for which there is a closed form solution:

$$\underset{\mathbf{x}}{\operatorname{argmin}} \left(a + \mathbf{b}^\top \mathbf{x} + \frac{1}{2}\mathbf{x}^\top \mathtt{C}\mathbf{x} \right) = -\mathtt{C}^{-1}\mathbf{b} \tag{3.63}$$

From the Taylor series expansion, the local quadratic approximation of the function F about $\mathbf{x}$ is

$$F(\mathbf{x} + \boldsymbol{\delta}) \approx F(\mathbf{x}) + \nabla^\top F(\mathbf{x})\,\boldsymbol{\delta} + \frac{1}{2}\,\boldsymbol{\delta}^\top \mathtt{H}(\mathbf{x})\boldsymbol{\delta} \tag{3.64}$$

Algorithm 3.14 Damped Newton: $\operatorname{argmin}\{F(\mathtt{A},\mathtt{B}) = \|\,\mathtt{W} \odot (\mathtt{M} - \mathtt{A}\mathtt{B}^\top)\|_F^2\}$

inputs $\mathtt{M} \in \mathbb{R}^{m\times n}$, $\mathtt{A} \in \mathbb{R}^{m\times r}$, $\mathtt{B} \in \mathbb{R}^{n\times r}$, $\mathtt{W} \in \mathbb{R}^{m\times n}$, $(m,n,r) \in \mathbb{Z}$

declare function $F = \|\,\mathtt{W} \odot (\mathtt{M} - \mathtt{A}\mathtt{B}^\top)\|_F^2$
declare $\mathbf{x} \in \mathbb{R}^{r(m+n)}$, $\mathbf{y} \in \mathbb{R}^{r(m+n)}$, $\mathbf{d} \in \mathbb{R}^{r(m+n)}$, $\mathtt{H} \in \mathbb{R}^{r(m+n)\times r(m+n)}$
$\mathbf{x} \leftarrow$ vectorize(A,B)
$\lambda \leftarrow 0.01$
repeat
  $\mathbf{d} = \frac{\partial F}{\partial \mathbf{x}}$
  $\mathtt{H} = \frac{\partial^2 F}{\partial \mathbf{x}^2}$
  repeat
    $\lambda \leftarrow \lambda \times 10$
    $\mathbf{y} = \mathbf{x} - (\mathtt{H} + \lambda \mathtt{I})^{-1}\mathbf{d}$
  until $F(\mathbf{y}) < F(\mathbf{x})$
  $\mathbf{x} \leftarrow \mathbf{y}$
  $\lambda \leftarrow \lambda \div 10$
until no change

outputs $\mathtt{A}, \mathtt{B} \leftarrow$ unvectorize($\mathbf{x}$)

with the entries of the Hessian matrix, $\mathtt{H}$, for a point $\mathbf{x}$, provided by the second derivatives:

$$\frac{\partial^2 F}{\partial a_{ab}\partial a_{ef}} = 2\delta_{ae}\sum_{j=1}^{n} w_{aj}^2 b_{jb} b_{jf} \tag{3.65}$$

$$\frac{\partial^2 F}{\partial b_{cd}\partial a_{ef}} = 2w_{ec}^2\left(a_{ed}b_{cf} + \delta_{df}\left(-m_{ec} + \sum_{k=1}^{r} a_{ek}b_{ck}\right)\right) \tag{3.66}$$

$$\frac{\partial^2 F}{\partial a_{ab}\partial b_{gh}} = 2w_{ag}^2\left(a_{ah}b_{gb} + \delta_{bh}\left(-m_{ag} + \sum_{k=1}^{r} a_{ak}b_{gk}\right)\right) \tag{3.67}$$

$$\frac{\partial^2 F}{\partial b_{cd}\partial b_{gh}} = 2\delta_{cg}\sum_{i=1}^{m} w_{ic}^2 a_{id} a_{ih} \tag{3.68}$$

However, rather than jumping to the minimum of this approximation to the surface at $\mathbf{x}$, the gradient direction should be used to guide a line search. Newton's method is given in Algorithm 3.13.

3.3.3 Damped Newton

Newton's method assumes that a quadratic surface is always a helpful approximation to the actual error surface. Sometimes it may be better to simply perform a gradient descent step, for example when $\mathtt{H}$ is not positive-definite. Newton's method can be easily modified to act in the fashion of the Levenberg Marquardt algorithm. Also known as trust region minimization, one such procedure is presented in Algorithm 3.14.

3.3.4 Damped Newton with Line Search

Matrix inversion is computationally expensive. Every run through the inner loop of the damped Newton algorithm includes a potentially large matrix inversion. An extension that attempts to reduce the number of inversions combines the damped Newton iteration with the standard line search (Algorithm 3.15). Having calculated the descent vector as $(\mathtt{H} + \lambda\mathtt{I})^{-1}\mathbf{d}$ the next step is to perform a line search in this direction rather than a rejection-update inner loop. The parameter λ is then updated (step 10 of 3.15) based on where on the line a minimum was found. If the minimum was very close to the start point for that iteration, i.e. a small step, λ is increased proportionally to emulate that small step for the next iteration. Conversely, if a large step was performed, λ is reduced.

3.3.5 Hybrid System

Alternation is very fast initially, but often gets stuck in the many shallow valleys that appear when the number of unknown elements increases. Damped Newton is fast in valleys, but fairly ineffectual when far from minima where alternation is better. A graphical depiction of this is given in the results section (Figure 3.16). An interesting proposal, therefore, is to combine the two. The combination is suggested in passing by Little and Rubin (2002), but no references about its performance have been found, especially in the computer vision literature.

Algorithm 3.15 Damped Newton with line search: $\operatorname{argmin}\{F(\mathtt{A},\mathtt{B}) = \| \mathtt{W} \odot (\mathtt{M} - \mathtt{A}\mathtt{B}^\top)\|_F^2\}$

inputs $\mathtt{M} \in \mathbb{R}^{m\times n}$, $\mathtt{A} \in \mathbb{R}^{m\times r}$, $\mathtt{B} \in \mathbb{R}^{n\times r}$, $\mathtt{W} \in \mathbb{R}^{m\times n}$, $(m,n,r) \in \mathbb{Z}$

declare function $F = \| \mathtt{W} \odot (\mathtt{M} - \mathtt{A}\mathtt{B}^\top)\|_F^2$
declare $\mathbf{x} \in \mathbb{R}^{r(m+n)}$, $\mathbf{d} \in \mathbb{R}^{r(m+n)}$, $\mathtt{H} \in \mathbb{R}^{r(m+n)\times r(m+n)}$
$\mathbf{x} \leftarrow$ vectorize(A,B)
$\lambda \leftarrow 0.1$
repeat
  $\mathbf{d} = \frac{\partial F}{\partial \mathbf{x}}$
  $\mathtt{H} = \frac{\partial^2 F}{\partial \mathbf{x}^2}$
  $\alpha \leftarrow$ minimize $F(\mathbf{x} - \alpha(\mathtt{H} + \lambda\mathtt{I})^{-1}\mathbf{d})$
  $\mathbf{x} \leftarrow \mathbf{x} - \alpha(\mathtt{H} + \lambda\mathtt{I})^{-1}\mathbf{d}$
  $\lambda \leftarrow \lambda \div \alpha$
until no change

outputs $\mathtt{A}, \mathtt{B} \leftarrow$ unvectorize($\mathbf{x}$)

The main decision regarding the implementation of the hybrid scheme is how to determine when to switch from one method to the other. The λ parameter of damped Newton is useful in this respect. When the damped Newton method starts to resort to the gradient descent strategy, i.e. when the λ parameter is large, the step sizes are getting very small and progress will be very slow. Another interpretation of λ is as a measure of the fit of the quadratic approximation. The larger the value of λ the worse the fit. Therefore, switching to alternation when λ is large could be advantageous.

There are many ways in which switching between alternation and damped Newton based on the value of λ can be implemented. Algorithm 3.16 gives one example: λ reaching a threshold value is used to initiate the switch from damped Newton to alternation. The switch back is made after a set number of alternation steps, with the hope that a place more suited to the Newton method has been found.

Two other strategies were implemented: a very basic version which calculated the new state using both alternation and damped Newton and chose between them using the new error for each (the line search method was used instead of the full damped Newton scheme for speed); and a more complicated version that monitored the value of λ within the damped Newton optimization loop (the innermost loop) and bailed out when λ became too large. A

comparison of these implementations is given in the results.

Algorithm 3.16 The alternation/damped Newton hybrid: $\operatorname{argmin}\{F(\mathtt{A},\mathtt{B}) = \| \mathtt{W} \odot (\mathtt{M} - \mathtt{A}\mathtt{B}^\top) \|_F^2\}$

inputs $\mathtt{M} \in \mathbb{R}^{m \times n}$, $\mathtt{A} \in \mathbb{R}^{m \times r}$, $\mathtt{B} \in \mathbb{R}^{n \times r}$, $\mathtt{W} \in \mathbb{R}^{m \times n}$, $(m, n, r) \in \mathbb{Z}$

declare function $F = \| \mathtt{W} \odot (\mathtt{M} - \mathtt{A}\mathtt{B}^\top) \|_F^2$

declare $\mathbf{x} \in \mathbb{R}^{r(m+n)}$, $\mathbf{y} \in \mathbb{R}^{r(m+n)}$, $\mathbf{d} \in \mathbb{R}^{r(m+n)}$, $\mathtt{H} \in \mathbb{R}^{r(m+n) \times r(m+n)}$

$\lambda \leftarrow 0.01$

SWITCH $\leftarrow 100$ *// the value of λ for a change to alternation*

COUNT $\leftarrow 4$ *// the number of alternation steps before returning to Newton*

$c \leftarrow$ COUNT

repeat

  if $c > 0$ **then** *// alternation*

    $c \leftarrow c - 1$

    $\mathbf{a}^i \leftarrow \left(\mathtt{B}^\top \operatorname{diag}(\mathbf{w}^i)^2 \mathtt{B}\right)^{-1} \mathtt{B}^\top \operatorname{diag}(\mathbf{w}^i)^2 \mathbf{m}_j \quad \forall i$

    $\mathbf{b}^j \leftarrow \left(\mathtt{A}^\top \operatorname{diag}(\mathbf{w}_j)^2 \mathtt{A}\right)^{-1} \mathtt{A}^\top \operatorname{diag}(\mathbf{w}_j)^2 \mathbf{m}_j \quad \forall j$

  else *// damped Newton*

    $\mathbf{x} \leftarrow$ vectorize(A,B)

    $\mathbf{d} = \frac{\partial F}{\partial \mathbf{x}}$

    $\mathtt{H} = \frac{\partial^2 F}{\partial \mathbf{x}^2}$

    repeat

      $\lambda \leftarrow \lambda \times 10$

      $\mathbf{y} = \mathbf{x} - (\mathtt{H} + \lambda \mathtt{I})^{-1} \mathbf{d}$

    until $F(\mathbf{y}) < F(\mathbf{x})$

    $\mathbf{x} \leftarrow \mathbf{y}$

    $\lambda \leftarrow \lambda \div 10$

    $\mathtt{A}, \mathtt{B} \leftarrow$ unvectorize($\mathbf{x}$)

  end if

  if $\lambda \geq$ SWITCH **then**

    $\lambda \leftarrow \lambda \div 10$ *// start safely on return to Newton*

    $c \leftarrow$ COUNT

  end if

until no change

outputs A, B

3.4 Results

Experiments were carried out to discover the relative performance of the algorithms presented in Sections 3.2 and 3.3. Several separate tests explore different aspects of the problem. Firstly, the results from synthetic tests are presented to give an idea of the general relative performance of the algorithms. Not all algorithms were implemented at the time of these tests, but the comparisons that are drawn reflect well the overall picture. Then we return to the example problems introduced in Section 3.1.1. All the algorithms were run on these examples and the results are informative. Finally, the dinosaur sequence is used to demonstrate the suitability of the error function that all the algorithms described above have been designed to minimize.

3.4.1 Synthetic Performance Comparison

Synthetic tests were run to compare the performance of six algorithms: alternation, PowerFactorization, Aanaes *et al.*'s variation, damped Newton, damped Newton plus line search and the hybrid method. All were implemented to be as efficient as possible. Each test was simply a general factorization problem of a noisy, incomplete, rank deficient matrix. A summary of the results of 10000 random synthetic tests is shown in Figures 3.10 and 3.11. The graphs show that the relative success of the algorithms is varied. It is the three Newton-based algorithms that perform most successfully. Alternation is less good at minimizing the error function. The PowerFactorization and Aanaes *et al.*'s algorithms display the worst performance in these tests. Comparing their relative speeds also shows a range of attainments. Damped Newton and damped Newton with line search are the slowest algorithms (with the line search version being slightly quicker as expected). The Aanaes *et al.* algorithm is also slow, which can be attributed to its use of the SVD in each iteration. Alternation and PowerFactorization are the quickest, with the hybrid coming in about twice as slow. The number of iterations until convergence is also interesting because it is less implementation sensitive

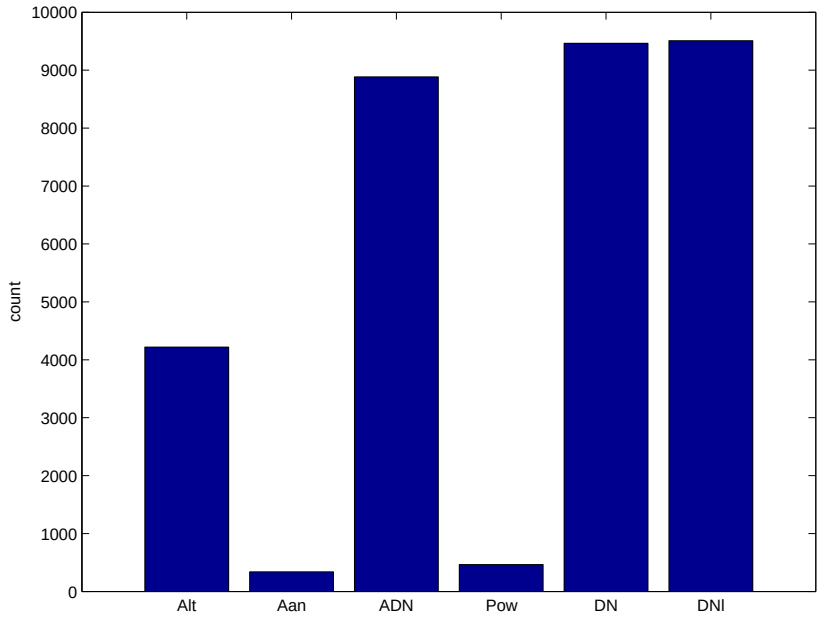


Figure 3.10: **Synthetic evaluation I.** Results from 10000 synthetic random general tests. The bar chart displays the number of tests in which each of the six algorithms attained the lowest error of all the methods for individual tests. The same initial guess was used by all algorithms in each test. Alt = alternation; Aan = Aanaes *et al.*; ADN = hybrid method; Pow = PowerFactorization; DN = damped Newton; DNl = damped Newton plus line search.

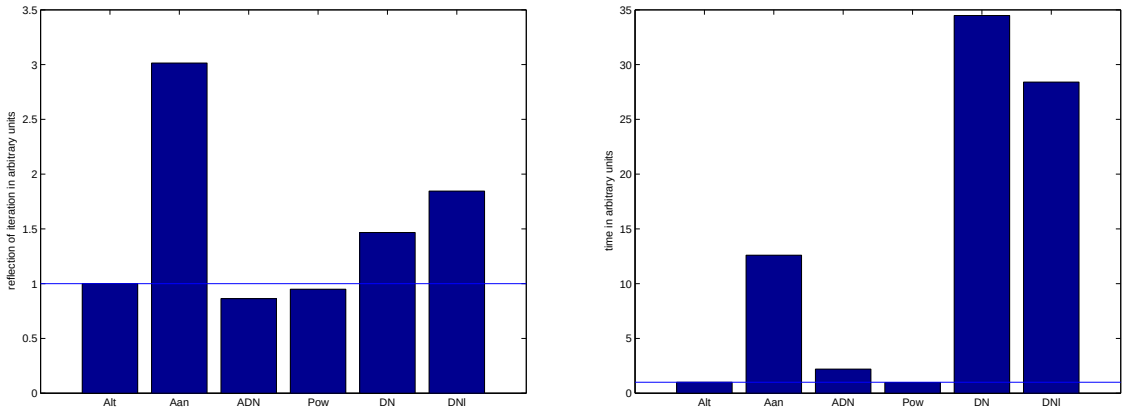


Figure 3.11: **Synthetic evaluation II.** Results from the same tests used for Figure 3.10. The left chart shows the average relative number of iterations and the right the average time for convergence for the six algorithms. As the tests took different parameters, the number of iterations and times taken are not comparable across all tests. For these graphs, the statistics were normalized to the alternation algorithm for each trail before being averaged. Alt = alternation; Aan = Aanaes *et al.*; ADN = hybrid method; Pow = PowerFactorization; DN = damped Newton; DNl = damped Newton plus line search.

(although it must be kept in mind that iteration costs vary substantially across the algorithms, e.g. alternation is $O\left(r(r^2 + \max(m,n)^2)\right)$ and Newton is $O\left(r^3(m+n)^3\right)$). The comparison here shows the hybrid method to be the best and the Aanaes *et al.* method the worst.

It should be noted that the above synthetic test results give a very strict view on the algorithms' performance in terms of minimization ability, looking only the 'race winners' in that category. One might prefer to use an algorithm that does not necessarily ever reach the lowest minima, but gets quite close most of the time (as a precursor to bundle adjustment for example). I am interested in each algorithm's ability to find the global minimum and feel that the above analysis reflects this better. Nevertheless, the relative performance of the algorithms at the higher level of detail is demonstrated in tests on real data below.

Coupling

The distribution of known elements in the measurement matrix has a large impact on the success of the factorization process. As discussed in Section 3.2.1, if the configuration of known elements is such that blocks of the `M` are effectively independent of each other (see Figure 3.6) then, in general, it is impossible to reconstruct the full matrix. Matrices with this structure are deemed to have poor *coupling*, the extent to which regions of `M` are linked to each other.

Figure 3.12 shows the results of three experiments in a set of many tests covering a large range of weight matrix configurations. The examples in the figure are representative of the trends in the full results. The experiment at the top of the figure used a dense weight matrix and so had no coupling issues. The second used a `W` that is almost decoupled, having very few elements that overlap with both the diagonal quadrants. In the last experiment shown, `W` is completely decoupled and the two diagonal quadrants are entirely independent of each other. The results displayed for each experiment comprise: the structure of the weight matrix on the left, with black representing 'known' data and white signifying 'missing' data; and a set of six residual matrices showing the success of the three algorithms used in the test. The

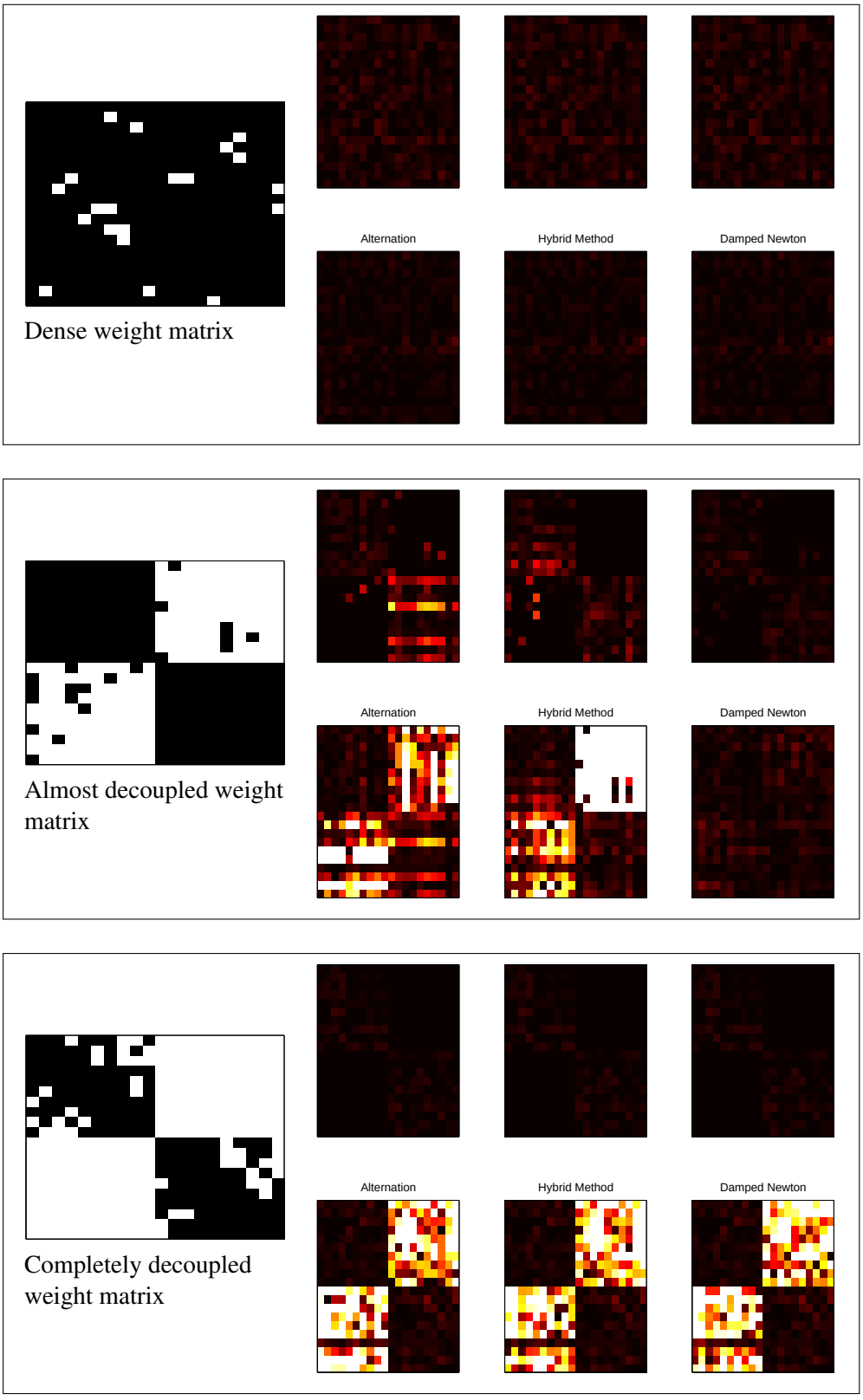


Figure 3.12: **Coupling.** A demonstration of the effect *coupling* has on the reprojection error (see Section 3.4.1). `W` on left, residuals on right. Error function residuals in top row, actual residuals on bottom row.

two leftmost matrices are alternation, a hybrid method is in the centre and the residuals from the damped Newton algorithm are on the right. The results from the three hybrid methods that were implemented are so similar that the results from only one can be shown to convey the trend clearly (see Section 3.4.2). The top row of residuals for each experiment show are the differences between the measurement matrix passed to the algorithms (a noisy version of the actual measurement matrix with the noise added having a magnitude 5% of the true matrix element values) and the results they returned, filtered using the weight matrix (i.e. the errors summed in the calculation of the error function). The bottom row are the residuals of the solutions given by the algorithms compared to the actual (noise free) measurement matrix. The residual matrices are coloured such that 'hotter' colours denote larger differences between the appropriate measurement matrices and the reprojections. Only the weight matrix changed between the tests; all other matrices were kept constant, including the initial guess.

When the known data is dense, none of the algorithms have any trouble finding a solution. The pattern of the residuals is identical for all three solutions, suggesting that global minimum has been found.

As the weight matrix approaches the decoupled state, alternation and the hybrid methods start to become unable to reconstruct all of the missing elements of the measurement matrix. Alternation's coordinate descent scheme, where `A` and `B` are optimized separately of each other, means that the few known elements that link the two large known blocks cannot be used in any significant way. Damped Newton, on the other hand, using a scheme that optimizes both `A` and `B` simultaneously, manages to use the few elements that are known in the off-diagonal quadrants to successfully recover those regions.

Once all elements in the off-diagonal quadrants are lost, there is nothing that can be done to reconstruct those portions of `M`. Note that all three algorithms are told only about the weighted error (generated by the upper residual matrices shown for each experiment in the figure), which shows them all to be doing roughly equally well: the weight matrix is hiding the poor fit of the two unknown blocks.

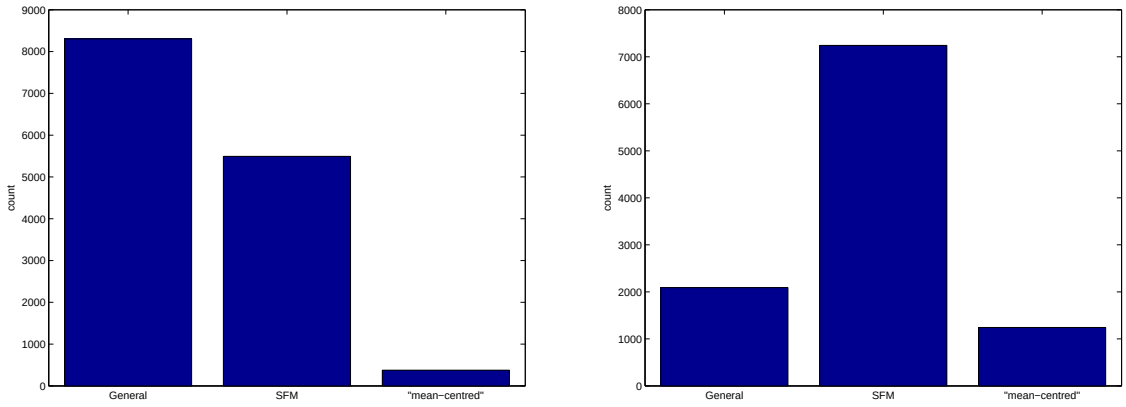


Figure 3.13: **SFM vs general.** The overall results of 10000 tests with independently varied matrix size, noise level and visibility (fraction of missing data). All tests were the factorization of a rank 4 synthetic measurement matrix whose elements actually exist in an offset 3D hyperplane, simulating the situation in SFM. Three algorithms were run in each test: general alternation, SFM alternation (Wiberg) and "partially mean-centred" alternation. The left chart shows counts of each of the algorithms obtaining a solution with the lowest error (using the weighted error function) of the three methods for each test. The right chart is the count of the solution being the closest to the actual answer.

SFM vs General Factorization

The structure from motion problem has the special constraint that the last column of B should be $\mathbf{1}$, i.e. the three-dimensional scene points lie in an offset hyperplane in a four-dimensional space. The general factorization algorithm does not account for this. Wiberg's modification to the alternation algorithm is the template for how to perform the correct minimization in this specific situation.

Synthetic measurement matrices were generated by multiplying together random four-column As and Bs (with the last column of B set to one). Noise was added and it was then passed, with a random weight matrix, to effectively three algorithms: (1) alternation looking for a general rank four solution, (2) Wiberg's alternation and (3) alternation looking for a general rank 3 factorization to a "partially mean-centred" measurement matrix. The last set-up took the input matrix and then subtracted a weighted column sum using the weight matrix. This is the missing data equivalent to mean-centring the measurement matrix in an attempt to convert the problem to a rank three general factorization (see Section 3.1.4).

10000 tests were executed over a variety of problems. The size, noise and visibility of the input measurement matrix were all independently randomized for each test.

The results are shown in Figure 3.13. They show that the general factorization is better at minimizing the missing-data error function, but that Wiberg's alternation for SFM gets answers that are closer to the actual solution. Of course, in actual problems, the real solution is not known and this comparison cannot be made. Attempting to mean-centre the measurement matrix to make it a rank 3 problem by finding the row-wise average of the visible elements is not effective. Optimizing for the 3D hyperplane and the offset using Wiberg's formulation is best option.

3.4.2 Real Problems

The example applications presented in Section 3.1.1 were used to compare all the algorithms described in this report. A summary of the problems with an overview of the results is given by Figure 3.14. For each problem, all the algorithms were run using two classes of initialization. The algorithms were firstly started from the answer given by Jacobs' reconstruction algorithm and secondly from 500 or 1000 random starting points. The large number of random initializations was undertaken in an effort to gain a picture of the converge basins of the local minima. The parameter space in which the error function is being minimized is very large and an exhaustive search is intractable. Fortunately it turns out that the convergence basins of the minima in these problems are large enough for many minima to be isolated using only hundreds of initial estimates. In all three problems one minimum was significantly more prominent than all other final error values. This error has been taken as the global minimum of each problem. Although this cannot be said for sure, the evidence strongly suggests that this is the case. The last row of Figure 3.14 is a basic summary of the three base algorithms' performance and reflects the size of the convergence basins for them. The full plots are given in the following subsections. In these cumulative frequency plots, the

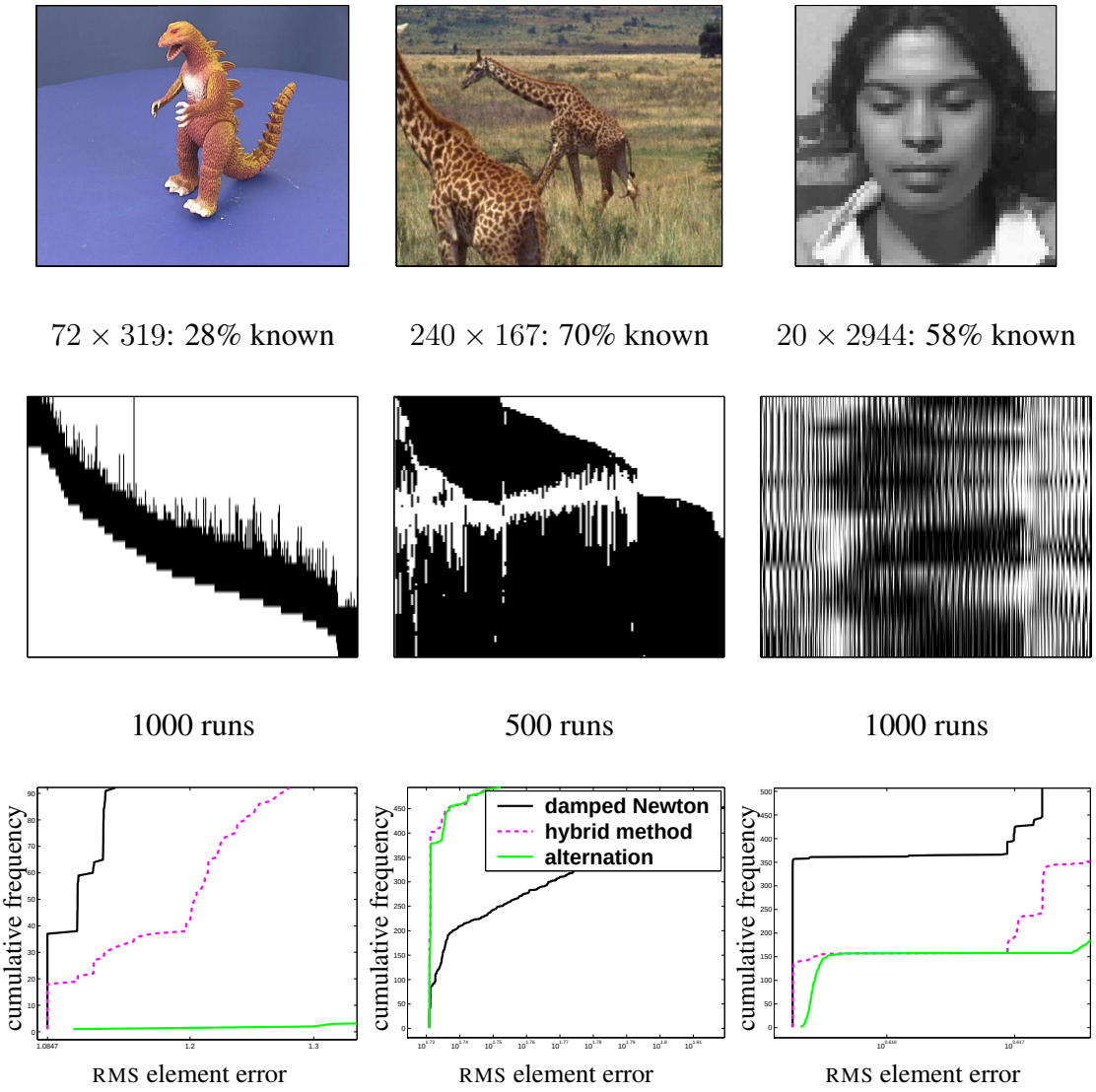


Figure 3.14: **Real test sequences.** Summary of the results from real tests. All algorithms presented in the text were used to solve the factorization problem associated with each of the three examples introduced in Section 3.1.1. The three columns correspond to the three problems: the recovery of 1) the rigid turntable motion of a toy dinosaur; 2) the non-rigid occluded motion of a giraffe in the background; and 3) the light directions and surface normals of a static face with a moving light source. The first row shows a frame from the sequence. The second represents the sparsity of the measurement matrix. The third row is a detail of the accumulation histogram based on the final error and counted over all runs.

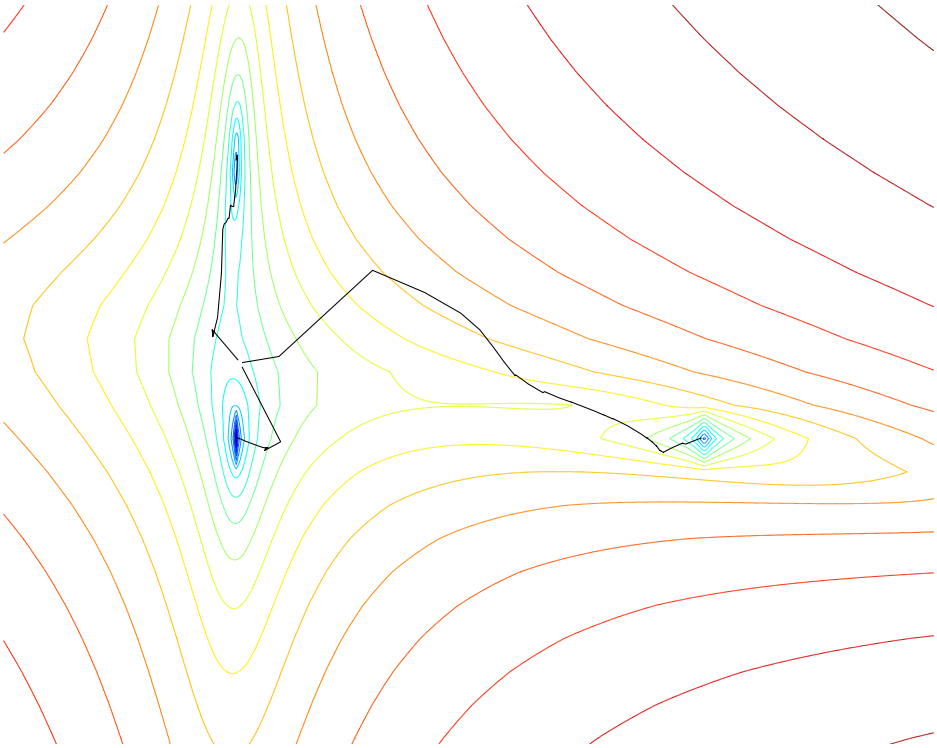


Figure 3.15: **Multiple minima II.** A contour plot of the same error cross-section shown in Figure 3.3. The descent paths from three starting points, projected onto this cross-section are overlaid. Although the starting points are very close, three separate widely spaced minima have been found. In another 997 runs from similarly close starting points, very few minima were visited more than once.

height of each step in the curve is representative of the size of the convergence basin for the minimum with that error. The first step corresponds to the global minimum and so the height of the first step can be taken as a measure of algorithm success.

Sensitivity to initialization is a large aspect of the practical situation. Figure 3.15 shows a contour plot of a slice through the error surface (seen in Figure 3.4) around three minima for the dinosaur problem. Superimposed onto the plot are three descent trajectories taken by the damped Newton method as it minimized the error function from three different starting points. Note that the relative spread of the minima in comparison to the spread of the initial states is large. Initial guesses must be close to the global minimum for it to be found. Direct methods (Section 3.2.1) are often suggested as ways of obtaining a good initial guess. This assertion was tested using Jacobs' method. The reconstruction given by Jacobs' algorithm

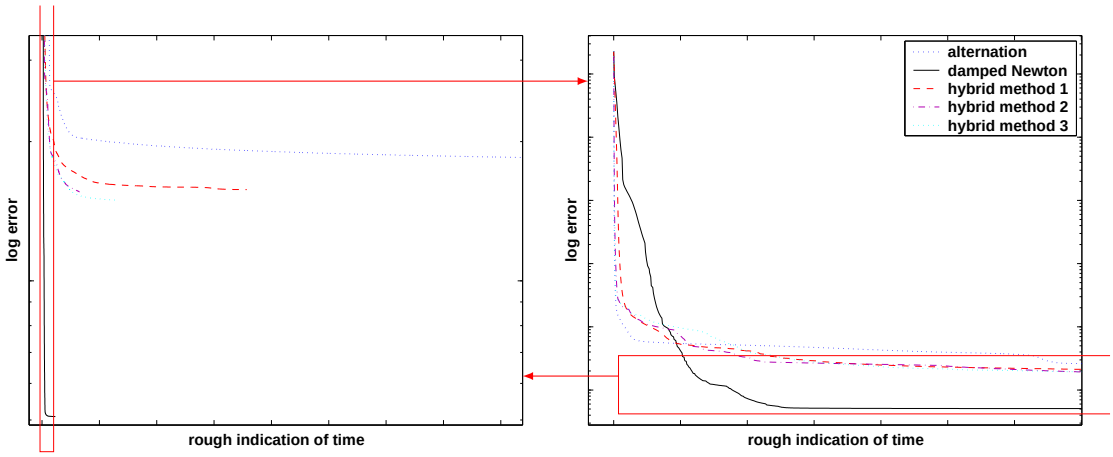


Figure 3.16: **Example minimization.** Typical error minimization graphs for alternation, three hybrid methods and damped Newton. The plot on the left shows a large range of iterations (missing the detail of the first few iterations). Note how alternation is still running, long after the other four algorithms have converged. The plot on the right is the detail of the first few iterations showing how fast alternation converges initially, before damped Newton catches up.

and the results from all algorithms initialized using it did not give the lowest errors seen for any of the problems, i.e. they were not near the global minimum of the error function. For missing data problems it is clear that Jacobs' method cannot be relied upon for a solution or as a good initialization for any of the iterative schemes that have been tested.

It is important to note that outliers were not included in any of the example problems. Although some algorithms (including the Newton based algorithms) can be extended to be robust to outlying data, none of those tested could cope with outliers. All three measurement matrices were 'clean' of erroneous data entries when passed to the algorithms being tested.

Hybrid Methods

Before looking at the results of all the algorithms' performance on the real problems, here is a quick summary of the performance of the hybrid methods. Looking at typical attempts to solve for the structure and motion of the rotating dinosaur sequence, it is clear that there is a big difference between damped Newton and alternation. Figure 3.16 shows the error being reduced by the two base algorithms and the three hybrid schemes described at the end of Section 3.3 for one such typical attempt. Across all attempts, examples can be found of each

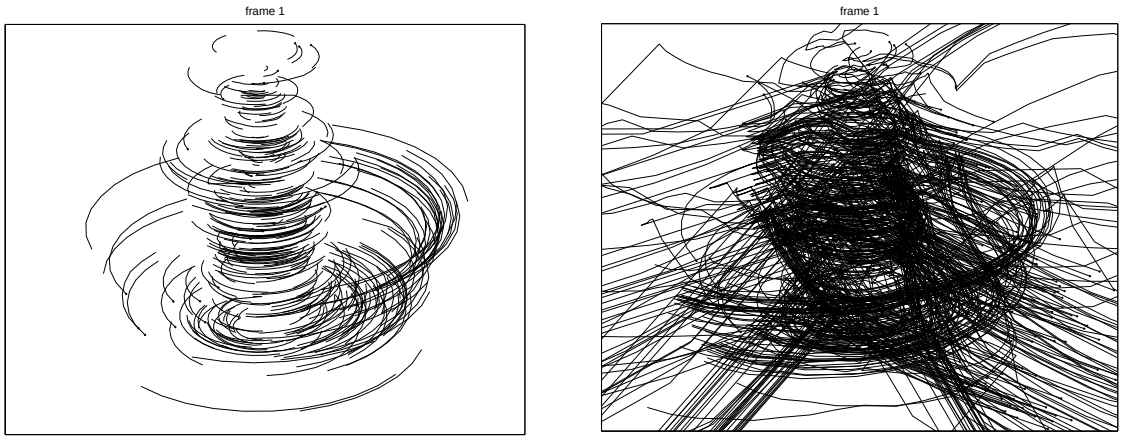


Figure 3.17: **Overfitting II.** On the left are the tracks of points detected in a video sequence of a rotating toy dinosaur. These tracks are enough to fill just over a quarter of the measurement matrix. Note the elliptical trajectories. On the right is a typical reconstruction from any one of the stitched results. The measurement matrix has been filled, but elliptical tracks have not been generated throughout the reconstruction. See Figure 3.18 for a more concise view of this measurement matrix.

outperforming the other, but the run shown is the mode behaviour.

The key points in Figure 3.16 are that a) alternation converges much quicker than damped Newton initially and b) damped Newton eventually catches up and overtakes alternation, leaving alternation to converge very slowly. The aim of introducing the hybrid schemes is to capture the positive attributes of both algorithms to get a method that can converge quickly both initially and ultimately. Figure 3.16 also shows the descent for the three hybrid schemes. They are very similar and lie between alternation and damped Newton. In almost all tests, the hybrid schemes performed almost identically. For the rest of this chapter they will be all referred to together and be represented in figures by just one line to increase clarity.

Rotating Dinosaur

Consider again the rotating dinosaur sequence. Automated tracking software generated feature tracks which were used to fill the measurement matrix. The tracks were then filtered for outliers using the knowledge that motion is rotational (i.e. elliptical tracks—see Section 3.4.3). The known values fill about 28% of the 72×319 element matrix. Because it is

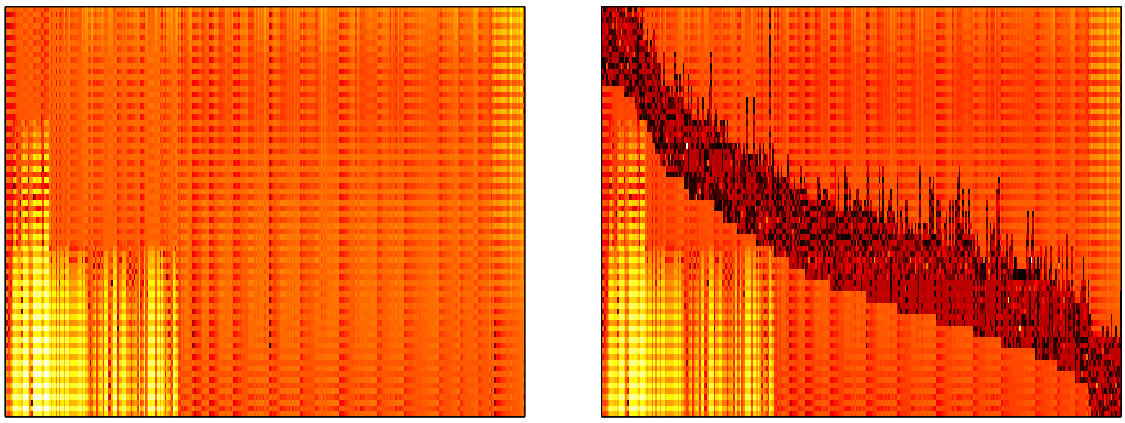


Figure 3.18: **Overfitting III.** Images of the measurement matrix from the rotating dinosaur sequence solution obtained by stitching (see Figure 3.17). The hotter the colour, the larger the value in the matrix. On the left is the reconstruction and on the right is the same matrix with the known elements darkened for comparison. Notice that, as the elements that have been filled by the process get further from the central diagonal band of known elements, the magnitude of the residual gets much larger. As all points should remain in view for the whole sequence this image is almost directly related to the error with respect to the unknown ground truth.

a video of rigid motion, the measurement matrix has rank 4, so theoretically only 5% of the matrix needs to be known. However, it is still a very sparse matrix.

Each algorithm was run 1000 times from random starting points. To test the algorithms further, they were all run on sub-blocks of the matrix as well. The sequence was split into two- and five-block subsequences, processed and then stitched back together to return to the full measurement matrix.

A representative reconstruction of the stitched solutions is shown in Figures 3.17 and 3.18. For this example, stitching together solutions to the smaller problems exaggerated the problems of error magnification (Section 3.2.1) leaving a very poor result. The runs on the measurement matrix taken as a whole proved to give better results. The left hand pane of Figure 3.23 shows the best reconstruction (at the assumed global minimum) of all the full factorization tests. Here we have a much more plausible answer, but still an unsatisfactory suggestion for turntable motion. This is discussed more at the end of this chapter.

The cumulative frequency plot in Figure 3.19 shows the relative successes of the algo-

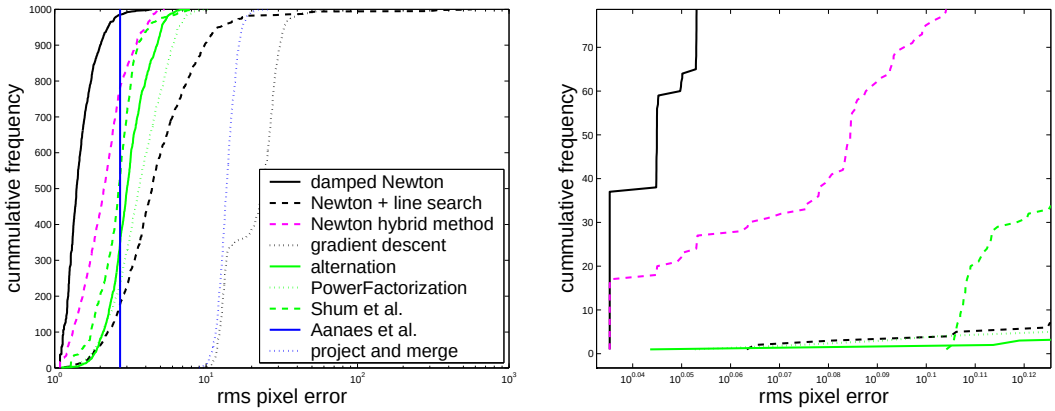


Figure 3.19: **Rotating dinosaur.** Cumulative frequency graphs for the results of 1000 randomly initialized runs on the dinosaur sequence. The larger the initial step, the better the algorithm can be considered to be. The left pane shows the full plot, the right shows a detail of the bottom left.

rithms in factorizing the full rotating dinosaur measurement matrix. As explained above, the higher the curve, the more successful the algorithm has been on this problem. The 'project and merge' algorithm, together with gradient descent, proved to be the worst schemes. All the alternation style methods performed with a similar level of success. Damped Newton has the best performance, with the hybrid methods appearing to be almost as good. None of the other algorithms, including the damped Newton with line search algorithm, found the global minimum from any of the 1000 random starting points of these runs.

Occluded Giraffe

The giraffe sequence provides a non-rigid motion example. Features on a giraffe walking behind another giraffe were tracked by hand to give a measurement matrix free of outliers. For this 240×167 element factorization (being larger than the dinosaur) each algorithm was run only 500 times. The damped Newton with line search algorithm was not run on the giraffe sequence and so can not be included in the comparison.

Refer to the cumulative frequency graph in Figure 3.20. Again, 'project and merge' and gradient descent proved to be the worst schemes. This time, two of the alternation type methods found the global minimum: alternation and PowerFactorization. They both did

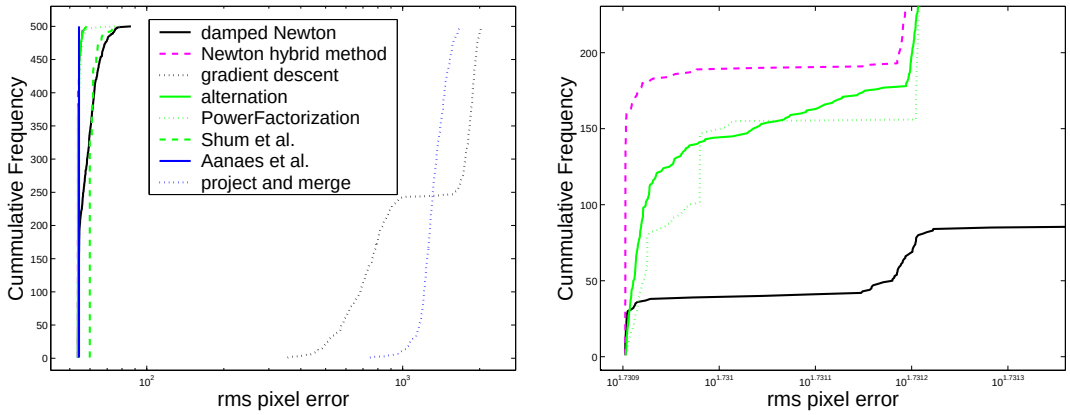


Figure 3.20: **Occluded giraffe.** Cumulative frequency graph for the results of 500 randomly initialized runs on the giraffe sequence. The full plot in on the left, with a detail from the bottom left of the plot on the right.

better than damped Newton. However, the hybrid methods showed themselves as the best algorithms for this problem. None of the other algorithms found the global minimum.

Figure 3.21 demonstrates the variation in success between one of the best performers (alternation) and the worst (gradient descent). For the frame shown, alternation (and the other successful algorithms) managed to propose very plausible positions for the features that could not be seen in that image. Gradient descent found a minimum where even the visible points reproject very poorly.

Illuminated Face

For a demonstration of illumination based reconstruction, an image sequence of a static face lit from twenty different directions has been used. As described in Section 3.1, the pixel intensity values are transferred directly into the measurement matrix (here of size 20×2944). Again, outliers must not be included and so all pixels for which the intensity value is not within diffuse illumination limits (pixels representing shadows and specularities) must be omitted. The algorithms were run 1000 times with random starting points for `A` and `B`. The results from the 'project and merge' algorithm have not been included because no runs using that method ever converged at all.

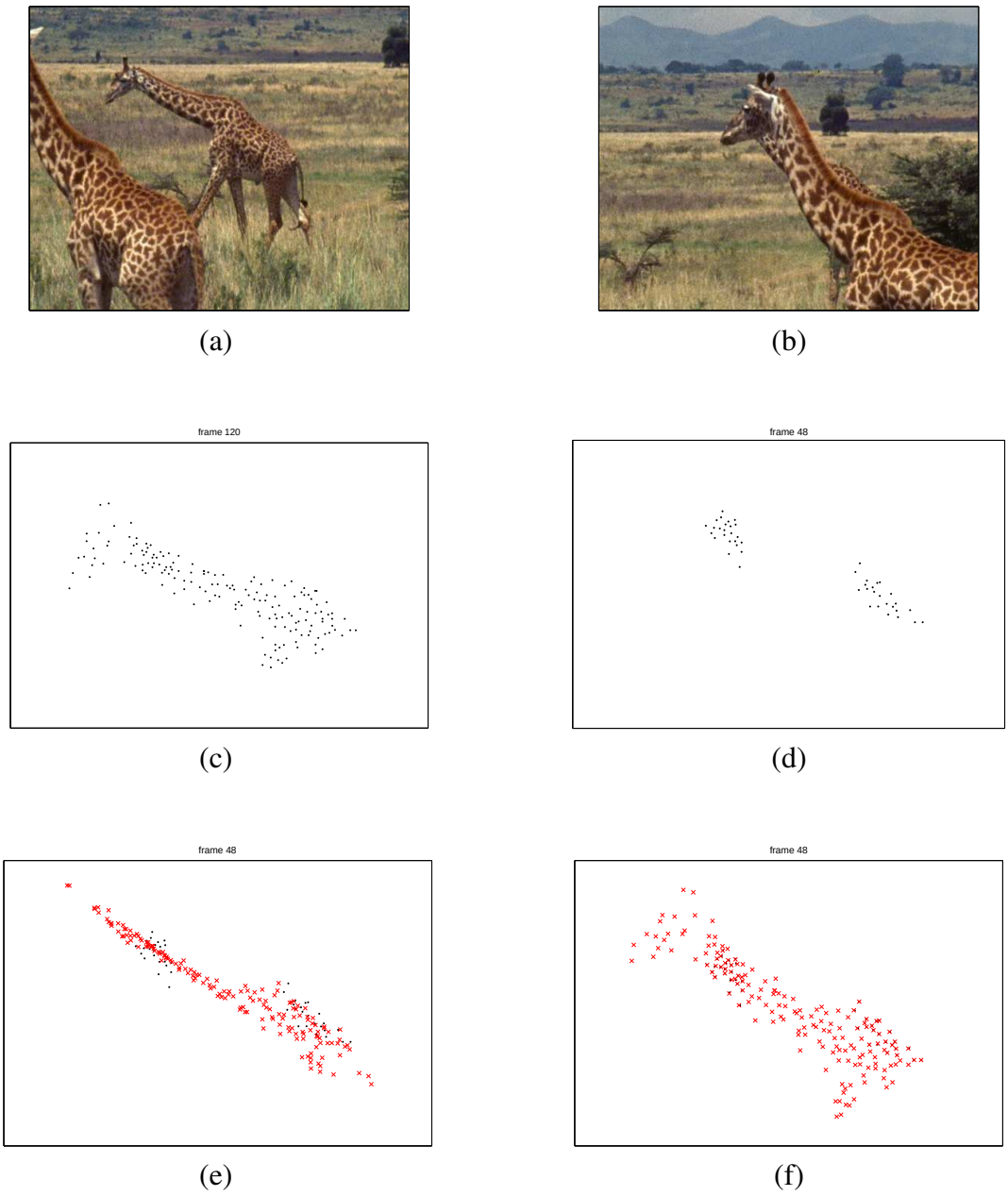


Figure 3.21: **Example reconstruction.** Example results for the giraffe sequence. (a) Frame 120 showing the whole background giraffe. (b) Frame 48 with the background giraffe occluded by the foreground giraffe. (c) The tracked points in Frame 120 (known coordinates as entered into the measurement matrix). (d) The tracked points in Frame 48. (e) A typical reconstruction of Frame 48 from the output of the gradient descent algorithm. (f) A typical result (Frame 48 again) of those algorithms that actually found the global minimum (e.g. alternation and the Newton based methods).

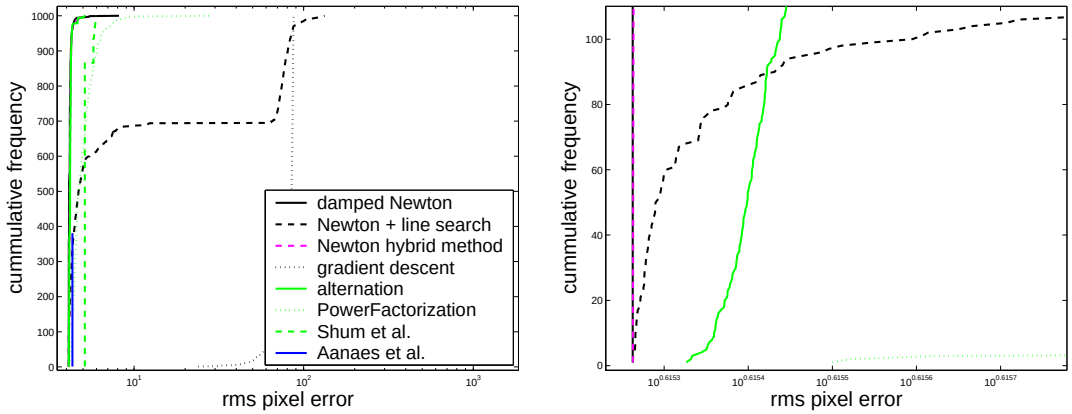


Figure 3.22: **Illuminated Face.** Cumulative frequency graph for the results of 1000 randomly initialized runs on the illuminated face sequence. On the left is the full plot and on the right is a detail of the plot.

Figure 3.22 displays the results of the runs on this problem. Here, we can see that all three Newton based methods (damped Newton, line search and hybrid) performed well. The line search method only just made it to the global minimum. Alternation was the only other notable algorithm, but it did not find the lowest minimum. All the other algorithms performed very badly.

3.4.3 Priors

An important question is whether the error function that all the algorithms tested are minimizing is one that can provide satisfactory solutions. As Figure 3.23 shows, minimizing the error function of Equation 3.5 (pure reprojection error) does not give the most satisfactory results. In the same figure is the global minimum of the same error function plus a regularizing term that penalized unorthonormal cameras:

$$F(\mathtt{A},\mathtt{B}) = \left\| \mathtt{W} \odot \left(\mathtt{M} - \mathtt{A}\mathtt{B}^{\top} \right) \right\| + \sum_{f=1}^{m/2} \left\{ \mathbf{p}_{2f-1}^{\top} \mathbf{p}_{2f} \right\} \tag{3.69}$$

Here, $\mathbf{p}_i$ has been used to denote the first three entries of the i^{th} row of the matrix $\mathtt{A}$. Recall that every pair of rows (more specifically, every odd plus even row pair) of $\mathtt{A}$ has the inter-

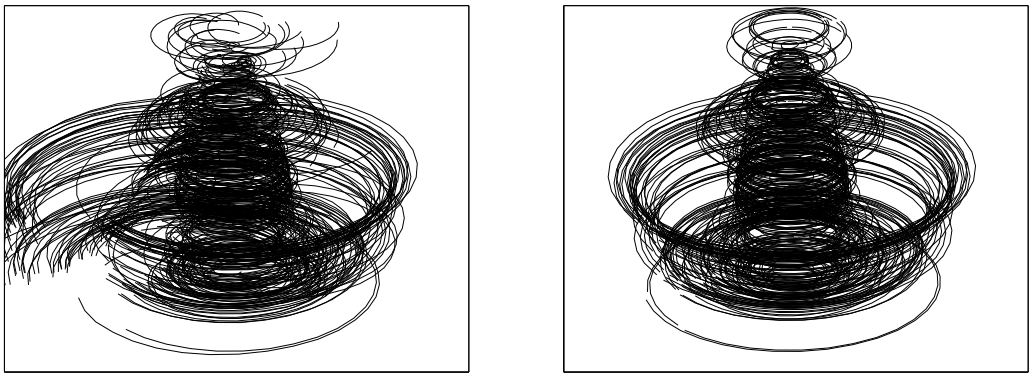


Figure 3.23: **Influence of constraints.** Point tracks for the dinosaur turntable sequence. See Figure 3.17 for the input tracks. On the left are the best tracks obtained using the algorithms discussed. On the right are tracks obtained using simple priors (orthonormality of the camera matrices in the column space).

pretation $[\,\mathtt{P}_f\ \mathbf{t}_f\,]$, hence the regularizer used. The improvement in the results through the use of a simple prior is pronounced. I think this clearly shows that the incorporation of prior knowledge of the problem into the minimization is very important. In one respect, an even simpler prior has already been introduced earlier in the report. The knowledge that for the SFM problem, the columns of the measurement matrix should lie on an offset hyperplane of dimension one less than the rank lead to the adaptation of the alternation algorithm used by Wiberg and Shum *et al*. In that case the inclusion of the prior allowed the retention of the closed form solutions required by alternation. However, in general, it is hard to develop priors that facilitate closed form solutions. The regularized error function above was minimized by damped Newton to give the result shown in Figure 3.23.

3.5 Final Remarks

Factorization is an important tool for computer vision. However, real applications must deal with missing data to be effective and standard factorization algorithms are unable to deal with missing data. As such, many iterative algorithms have been proposed in the computer vision literature to repair this deficiency. In addition to a review of these existing methods,

this report presents several routines based on the Newton method. Such methods have not been presented before in the computer vision literature for the problem of factorization with missing data. All existing methods published on this problem are in some way based on alternation. Hybrid methods, which have also not been covered in the literature, combine these two approaches and have also been described. A comparison of the three classes of algorithm showed that alternation is not the best choice, mainly due to the excessively slow convergence rates, even though such methods give good initial convergence. Newton optimization can also fail to provide good results on some sequences. Hybrid methods performed consistently and relatively well across the three real problems used for comparison.

The handling of outliers in the measurement matrix has not been addressed here, but is an important avenue for future research. As reported, 'robust' alternation has been attempted, but the resulting algorithm is ineffective. Newton strategies lend themselves to the incorporation of robust energy functions. In a brief investigation of robust least squares kernels (Hartley and Zisserman 2004), I found that their inclusion made the optimization unusably unstable. Further work is needed.

An important conclusion of this chapter regards the suitability of the basic error function. Unconstrained optimizations in the very high dimensional spaces typical of the problems described here will always be plagued by the fact that the error surface appears incredibly flat for much of the space. Furthermore, without the incorporation of regularizers to encapsulate prior knowledge of a problem, overfitting in the extra freedoms enabled by missing data invariably means that unsatisfactory results are obtained. Formulating priors such that alternation-style closed form solutions exist for the augmented error function is very difficult. On the other hand, adding priors that are twice differentiable is considerably easier and so Newton based methods provide a flexible framework for factorization and are my preferred strategy.

Chapter 4

Incorporating Global Motion

This chapter describes a motion prediction model which can be easily incorporated as a motion prior in order to improve the tracking performance of feature point tracking systems. It is designed to achieve two things: one, to bridge the gap between those techniques that impose a motion constraint on entire video sequences and those that do not employ any sort of global motion prior; and two, to enable the incorporation of global motion information into the determination of the motion tracks of arbitrary feature points.

The concepts of matrix factorization introduced in the last chapter form the base mechanism of the motion prior's realization. The broad aim is the ability to incorporate the motion seen globally across an entire frame of video over the recent frame history into any feature point tracking algorithm. More specifically, there is an extensive array of research exploring ways in which the motion seen across the entire frame of a video sequence is constrained by an underlying coherence, such as rigid structure, as covered in detail in the last chapter. It is now time to show how the global essence can be instilled within the otherwise local feature point tracking process in a general and flexible way.

4.1 Introduction

There are two shortfalls of existing algorithms that exploit a global motion constraint, in relation to feature point tracking in general scenes: one is that the use of interest points prevents direct generation of user-chosen feature point tracks; and the other is that indirect global constraints on frame-to-frame feature point matching do not constrain the motion appropriately.

The first issue comes from the use of interest point detectors in automated batch tracking systems, such as Fitzgibbon and Zisserman (1998). These include the likes of Harris corner detectors (Harris and Stephens 1988) or difference of Gaussian interest point detectors (e.g. Lowe 2004). The advantage of automatic feature point selection is that the overlying system can obtain high numbers of distinctive features quickly and without acquiring user approval. Each feature need only be tracked over short periods, because more features are selected in every successive frame, replacing any tracks that are cut short. This is sufficient for the determination of indirect entities, such as in camera localization or scene structure reconstruction, because the combined information from a number of short tracks of semantically arbitrary features is enough to constrain the solution. However, feature point tracking demands very long tracks on semantically significant features. It is unlikely that the point chosen by the user will be in the subset isolated by selecting features procedurally. Furthermore, it is far from guaranteed that a batch tracking process will have tracked any point for a significant percentage of the sequence length. Therefore the sparse information can not be used directly. It must be referenced indirectly: calculated in a first pass and used in a second if full flexibility is to be accomplished. This is the approach in this chapter. Irani (1999) showed how hard global motion constraints can be incorporated into direct optical flow methods for a range of rigid motion models, but even extending this to non-rigid motion constraints would not be enough, because of the second problem.

The second issue relates to the imposition of the constraining model. Straightforward,

completely rigid scenes fulfil the assumptions of rigid motion, so in those limited situations the process is successful. However, I am interested in general motion and footage that could contain considerable non-rigid motion or multiple bodies. Several authors have dealt with generalizations of the rigidity assumption: for example, Bregler et al. (2000) show how the rigidity assumption could be generalized to include non-rigid deformation, while Torresani and Hertzmann (2004) have extended such techniques and describe an automatic process for simultaneously tracking sets of feature points and fitting a non-rigid model to constrain the motion. Nevertheless, they all share the same drawback: their global motion model must apply to the whole sequence. As such, long sequences containing complicated non-rigid or multiple body motion must be described by a very general motion model in order to capture the complex movements seen across the whole video. As a result, the feature tracks for any particular subset of the sequence cannot be reliably constrained. Then, if a more specific motion model is used, the full range of motion becomes unattainable and tracks on small moving objects, for example, will be overlooked.

There is also a class of semi-local models. Sand and Teller (2006) simultaneously compute the trajectory of a large set of image features under a piecewise smoothness model implemented by first computing flow vectors under a gradient-weighted smoothness constraint, and then bilateral filtering of the flow. Similarly, Smith et al. (1998) refine matches using a median flow filter, again imposing a form of piecewise smoothness. These methods, although smooth globally, can exacerbate the feature drift problems of purely local methods. Furthermore, they lack the predictive power of the global methods, where trajectories in previous frames can be used to predict feature positions in future frames.

The work described in this chapter combines local and global models by continuously updating a non-rigid motion model over a sliding temporal window (typically of the order of 10 frames). The motion model itself is a rank-constrained model similar to that used by some of the above authors, but crucially imposed directly on the 2D motion of the tracks rather than with any reference to some underlying world model. Together, these two elements mean that

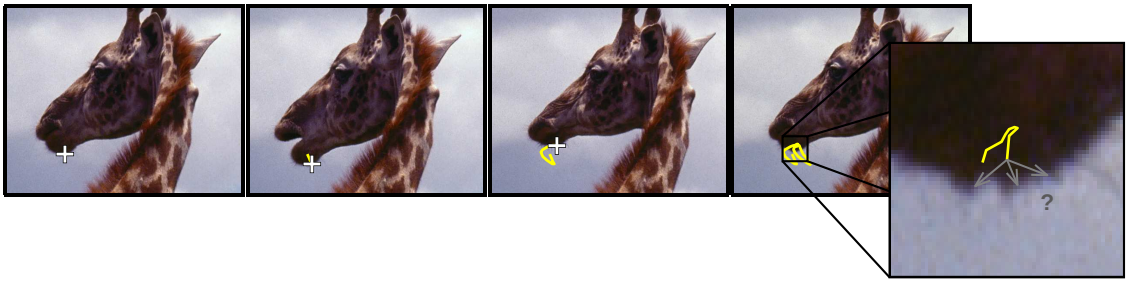


Figure 4.1: **Guided tracking I.** The task is to track the tip of the chin of a masticating giraffe. In this case, the hairs on the giraffe's chin are very similar and standard template matching gets confused as to which one is the target. Being able to predict the motion using motion models can help overcome this difficulty (see Figure 4.3).

the widest variety of motions can be captured by the model. When tracking an individual feature, this motion model can be used as a motion prior in any feature point tracking system. Using this prior, tracking systems can draw on the strong information held in the local optical flow information and the weaker global motion information used by the methods described above.

For illustrative purposes, the example feature point tracking system will be a relatively straightforward Bayesian template tracker based on the work presented by Arulampalam et al. (2002) and Matthews et al. (2003). I acknowledge that the system being described here encompasses many of the flaws that must be avoided in a good feature point tracker, mainly an inability to deal with occlusions. However, it is the development of the global motion prior and its ability to be integrated into a local algorithm that is the focus here, and so the engineering of the tracker itself is very much secondary. It will be explained in full, but it must be noted that it was chosen as a sophisticated baseline, with the hope that it would be intuitive enough to interested parties for them to make useful interpolations regarding performance and integration elsewhere.

Symbol	*Meaning*
t	sequence frame number
$\mathbf{x}_t$	hidden feature position in frame t
$\mathbf{z}_t$	observed feature position in frame t
M	size of temporal window for global motion model
$\mathbf{M}$	list of decreasing numbers from $M-1$ to 1
$\mathrm{p}(\mathbf{x}_t \mid \mathbf{z}_{1..t})$	the posterior probability of position given all observations
$\mathrm{p}(\mathbf{z}_t \mid \mathbf{x}_t)$	the likelihood of observed feature position over the image
$\mathrm{p}(\mathbf{x}_t \mid \mathbf{x}_{1-\mathbf{M}})$	the motion model
$\mathrm{p}(\mathbf{x}_{1-\mathbf{M}} \mid \mathbf{z}_{1..t})$	the prior
$q(\mathbf{x}_t)$	the *diffused prior* $\int \mathrm{p}(\mathbf{x}_t \mid \mathbf{x}_{t-\mathbf{M}})\mathrm{p}(\mathbf{x}_{t-\mathbf{M}} \mid \mathbf{z}_{1..t-1})$
R	rank of subspace approximating global motion
$\mathtt{M}, \mathtt{M}_t$	measurement matrix of precomputed tracks in frames $t-M+1$ to t
$\mathtt{B}, \mathtt{B}_t$	rank R motion basis for frames $t-M+1$ to t
$\mathtt{P}$	the motion predictor matrix: $\langle \mathbf{x}_t \rangle = \mathtt{P}\mathbf{x}_{t-\mathbf{M}}$
$\Sigma_{t-\mathbf{M}}$	the covariance matrix on likelihood
$\mathtt{A}_t, \mathtt{A}^*, \mathtt{A}'$	affine warp matrices
W	the affine warp function: $W(\mathbf{x}) = \mathtt{A}\mathbf{x} + \mathbf{t}$

Figure 4.2: **Notation.** Summary of notation used in this chapter (fold-out copy at back of book).

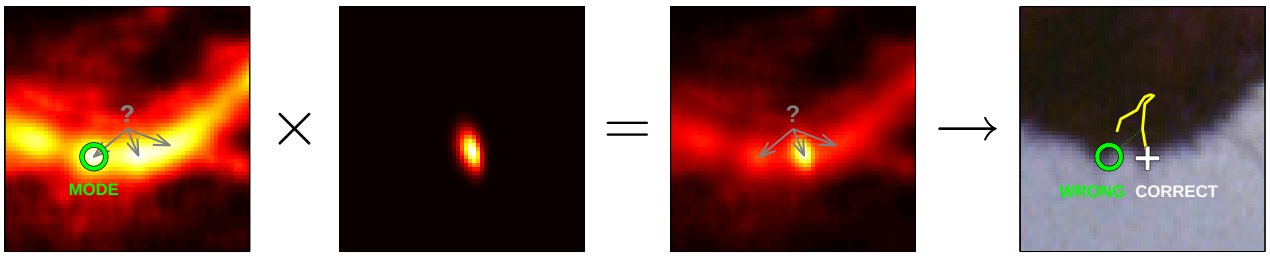
4.2 Notation

The notation used in this chapter follows on from the previous chapters. The little that is extra is primarily for making references to sets and indices more compact. Sets are often implicitly introduced using '..' in a subscript of the common name of the members, namely:

$$\mathbf{v}_{1..N} = \{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_N\}. \tag{4.1}$$

When the above term appears where a vector is expected (in a matrix multiplication for example), then the appropriate vector is constructed by stacking vertically the member of the implied set:

$$\vec{\mathbf{v}}_{1..N} = \begin{bmatrix} \mathbf{v}_1 & \mathbf{v}_2 & \cdots & \mathbf{v}_N \end{bmatrix}^\top \tag{4.2}$$



Likelihood *Diffused Prior* Posterior Surface Corrected Track

Figure 4.3: **Guided tracking II.** Frame 38 from the munching giraffe sequence (see Figure 4.1). From left to right: the likelihood function over position $\mathrm{p}(\mathbf{z}_t \,|\, \mathbf{x}_t)$. Note that the mode is far from the correct peak—the naïve maximum likelihood estimate is confused by the repeated texture; the diffused prior $q(\mathbf{x}_t) = \mathrm{p}(\mathbf{x}_t \,|\, \mathbf{z}_{t-\mathbf{M}})$; the posterior $\mathrm{p}(\mathbf{x}_t \,|\, \mathbf{z}_{t,t-\mathbf{M}})$ showing that the use of a motion model has resulted in the erroneous location being avoided and the correct point being chosen.

4.3 Example Tracking Framework

To help introduce the notion and the context for the motion prior, let me first introduce the example tracking framework into which it will be integrated, as the integration itself is an important part of the use of the prior.

The task of a feature point tracker is to return an accurate feature point trajectory through an image sequence of a scene undergoing general motion, including multiple bodies and non-rigidity, given the location of the feature in a single starting frame. For each frame, t, we wish to know the underlying position $\mathbf{x}_t$ of the feature, given the observations of position, $\mathbf{z}_{1..t}$, from the images in the current and all previous frames. The standard Bayesian tracking formulation (Arulampalam et al. 2002) poses the search for the probability density over position as

$$\underbrace{\mathrm{p}(\mathbf{x}_t \,|\, \mathbf{z}_{1..t})}_{\text{posterior}} \propto \underbrace{\mathrm{p}(\mathbf{z}_t \,|\, \mathbf{x}_t)}_{\text{likelihood}} \int \underbrace{\mathrm{p}(\mathbf{x}_t \,|\, \mathbf{x}_{t-1})}_{\text{motion model}} \underbrace{\mathrm{p}(\mathbf{x}_{t-1} \,|\, \mathbf{z}_{1..t-1})}_{\text{prior}} \mathrm{d}\mathbf{x}_{t-1} \tag{4.3}$$

with the posterior at time t becoming the prior at time $t+1$. There is much that this derivation overlooks, but it is still a popular starting point. A discussion of the assumptions and implications for the implementation described in this chapter is given in Appendix C. Nevertheless, it can be shown that the above equation is the probabilistic basis for the Kalman filter, in the case of a linear motion model, and particle filters for more complicated motion

predictors. One of the assumptions of the above formulation is that it is a first order Markov process on the feature's position, so any trajectory information beyond the previous frame is technically ignored[1]. I am proposing that the global motion from across the previous $M-1$ frames should be used to inform on the current frame's position, i.e. extending the scheme to an order $M-1$ Markov process. In this way, complex, non-rigid motion, consistent with that in the rest of the scene, can be captured and used to guide the tracking. The updated formulation is

$$\mathrm{p}(\mathbf{x}_t \,|\, \mathbf{z}_{1..t}) \propto \mathrm{p}(\mathbf{z}_t \,|\, \mathbf{x}_t) \int \mathrm{p}(\mathbf{x}_t \,|\, \mathbf{x}_{t-\mathbf{M}}) \mathrm{p}(\mathbf{x}_{t-\mathbf{M}} \,|\, \mathbf{z}_{1..t-1}) \mathrm{d}\mathbf{x}_{t-\mathbf{M}} \tag{4.4}$$

where $\mathbf{M}$ has been used for the list of decreasing numbers $M-1, \ldots, 1$, denoting the fact that the last $M-1$ frames are considered in the determination of the location of the feature in the current frame. A significant departure from the standard equation is that now the posterior can not be used directly as the prior in the calculations for frame $t+1$: the prior must be recalculated for every frame. Fortunately, updating the prior from frame-to-frame is a straightforward calculation, as will be shown below.

The next few sections will be used to describe the implementation of the example tracker, taking each of the three components: the motion model, the prior and the likelihood in turn.

4.4 The Global Motion Prior

The implementation of the motion model dominates the implementation of the tracker and so it is now time to give a slightly more detailed overview of the algorithm that will emerge. The algorithm follows the "guided matching" framework, where motion models are first computed for sub-sequences and then used to constrain tracking on a second pass. It is

[1]The incorporation of other state variables, such as velocity, effectively mean that a slightly longer term local motion model is employed, but this is not explicit in the derivation.

convenient to take the integral

$$q(\mathbf{x}_t) = \int \mathrm{p}(\mathbf{x}_t|\mathbf{x}_{t-\mathbf{M}})\mathrm{p}(\mathbf{x}_{t-\mathbf{M}}|\mathbf{z}_{1..t-1})\mathrm{d}\mathbf{x}_{t-\mathbf{M}} \tag{4.5}$$

as a single entity, which I will refer to as the *diffused prior distribution*, which incorporates the prior and the motion model. The main steps are as follows.

First pass: Fit motion models.

1. Detect interest points on each frame and match to interest points in successive frames to generate tracks through the sequence. This is a standard batch tracking procedure.

2. Robustly fit a rank R non-rigid motion model to the tracks in each (overlapping) M-frame subsequence. I have used $M = 10$ and $R = 6$.

Second pass: For a single track, initialize the track by standard tracking (i.e. without the motion model proposed here) for $M-1$ frames in order to form observations $\mathbf{z}_{1..M-1}$. Now for each subsequent frame t,

1. Predict the position in frame t as the diffused prior distribution.

2. Measure the likelihood $\mathrm{p}(\mathbf{z}_t|\mathbf{x}_t)$.

3. Choose $\mathbf{z}_t$ as the point which maximizes the product of likelihood and diffused prior.

4. Approximate the posterior with a Gaussian.

Thus the basic tracker is constrained by the global motion model $\mathrm{p}(\mathbf{x}_t|\mathbf{x}_{t-\mathbf{M}})$, reducing the likelihood of drift and of jumping to incorrect matches. Because the global model is based on a sliding temporal window it can model complex motions, while not being bogged down by distant trajectory history. I will now describe the key algorithm steps in more detail.

4.4.1 First Pass: Fitting the Motion Model

The motion model is similar to the non-rigid factorization model of Bregler et al. (2000). This is as described by the previous chapter, but a brief outline is now given to serve as a reminder and make clear the deviations. Given a sequence of M images, a tracked point is represented by a $2M$ long column vector $\mathbf{X} = [x_M, y_M, ..., x_1, y_1]^\top$. Given N tracked points in the sequence, the column vectors are concatenated horizontally into the $2M \times N$ measurement matrix, $\mathtt{M}$, holding each track, $\mathbf{X}_i$, in a separate column and the positions of all the features in a particular frame in consecutive pairs of rows.

The motion model takes the form of a basis that describes the motion seen in $\mathtt{M}$. This basis, $\mathtt{B}$, is a $2M \times R$ matrix. It assumes that all scene motion for the M-frame sequence exists in a rank R subspace, that is $\mathtt{M} = \mathtt{BC}$ for $\mathtt{C}$ an $R \times N$ matrix of coefficients. It is important to note that $\mathtt{B}$ is obtained from the direct factorization of $\mathtt{M}$, i.e. without mean-centering, thus providing a complex and desirable spatial consistency for predictions. The rank, R, is the main tuning parameter of the algorithm chosen to represent the complexity of motion in the scene, while avoiding over-fitting.

The prior used in the second pass is based on the robust generation of this global motion model using standard template tracking of interest points over short subsequences of the long input sequence. I build measurement matrices $\mathtt{M}$ for all M-frame subsequences of the sequence. It is acknowledged that $\mathtt{M}$ will contain incomplete and erroneous tracks. However, this will be dealt with robustly, as described next. The output of this stage is a $2M \times R$ basis matrix $\mathtt{B}_t$ for each frame of the sequence, with $\mathtt{B}_t$ modelling the motion over the previous $M-1$ frames, from t down to $t-M+1$.

4.4.2 RANSAC for Subsequence Motions

To summarize, the main task of the first pass is to compute the best rank-R basis for a measurement matrix $\mathtt{B}$, covering a subsequence of M frames. For an $\mathtt{M}$ containing no incorrect

tracks, the optimal rank R basis is obtained by (e.g. SVD) factorization and truncation to rank R. If occlusions and false positives have been recorded then a robust factorization algorithm should be used. I now describe a fast RANSAC-based algorithm that I have found to work well in practice.

Any robust batch tracking algorithm may be employed to generate the $\mathtt{M}$ for each subsequence, but faster and simpler approaches are more desirable for maximizing throughput and the ability to capture arbitrary motion respectively. I have chosen to use a straightforward Harris-based batch tracker to create the $\mathtt{M}$s, followed by this RANSAC scheme to obtain faithful bases. The steps of the RANSAC stage are:

1. pick R complete columns of the measurement matrix to form a candidate basis, $\mathtt{B}'$
2. calculate the reprojection error for all tracks in $\mathtt{M}$ using $e(\mathbf{x}) = \|(\mathtt{I} - \mathtt{B}'\mathtt{B}'^{+})\mathbf{x}\|$
3. count the support as the number of tracks (considering only complete columns at this stage) with a reprojection error less than a threshold;
4. choose the candidate basis with the largest support;
5. calculate the best basis, $\mathtt{B}$, from all the tracks in the support for the best candidate $\mathtt{B}'$ using SVD and rank truncation.

The above RANSAC process alone tends to be too aggressive and leaves out many correct tracks. To further improve the quality of the basis for each subsequence, I employ a "basis growing" procedure, expanding it to include as many of the inliers as possible in a simple, yet surprisingly effective way. The algorithm is as follows. Until the number of tracks in the support stops increasing, recalculate the reprojection error of all the tracks using the new basis and reclassify them all (as inliers or outliers), using a threshold, to generate a new support. Calculate the new basis by rank truncation as above and iterate. A variation that can be slightly more conservative is to reduce the threshold to that of the worst reprojection error of the support in any given iteration.

If the sequence needs to be re-tracked using a different rank constraint, the same measurement matrices are used to generate new bases. The time taken to robustly find and grow a basis for a subsequence takes about the same amount of time as loading an image from disk.

4.4.3 Second Pass: Computing the Motion Model

In order to compute the prediction of position for the current frame, we use the motion basis $\mathtt{B}_t$, which for the rest of this chapter will be denoted by $\mathtt{B}$. Concatenating the prediction $\mathbf{x}_t$ and history $\mathbf{x}_{t-\mathbf{M}}$ into a single $2M\times 1$ column vector, we obtain

$$\begin{bmatrix} \mathbf{x}_t \\ \mathbf{x}_{t-\mathbf{M}} \end{bmatrix} = \mathtt{B}\mathbf{c} = \begin{bmatrix} \mathtt{B}_0 \\ \mathtt{B}_\mathbf{M} \end{bmatrix} \mathbf{c} \tag{4.6}$$

where $\mathtt{B}_0$ is the first two rows of $\mathtt{B}$, and $\mathtt{B}_\mathbf{M}$ denotes the remaining rows. The coefficient vector $\mathbf{c}$ may be computed from $\mathbf{x}_{t-\mathbf{M}}$ as

$$\mathbf{c} = \mathtt{B}_\mathbf{M}^{+}\mathbf{x}_{t-\mathbf{M}} \tag{4.7}$$

where $\mathtt{B}_\mathbf{M}^{+}$ is the pseudoinverse of $\mathtt{B}_\mathbf{M}$, and thus the prediction of $\mathbf{x}_t$ is

$$\langle \mathbf{x}_t \rangle = \mathtt{B}_0\mathtt{B}_\mathbf{M}^{+}\mathbf{x}_{t-\mathbf{M}} = \mathtt{P}\mathbf{x}_{t-\mathbf{M}} \tag{4.8}$$

where the $2\times 2(M-1)$ matrix $\mathtt{P} = \mathtt{B}_0\mathtt{B}_\mathbf{M}^{+}$ projects the track $\mathbf{x}_{t-\mathbf{M}}$ into frame t. Even though this is a simple linear projection, it can model complex non-rigid motions (including sharp cusps) because of the use of the medium length time history and global support. We can then

define the distribution as

$$\mathrm{p}(\mathbf{x}_t|\mathbf{x}_{t-\mathbf{M}}) = \exp\left(-\gamma\|\mathbf{x}_t - \mathtt{P}\mathbf{x}_{t-\mathbf{M}}\|^2\right) \tag{4.9}$$

$$\equiv \mathcal{N}(\mathbf{x}_t \mid \mathtt{P}\mathbf{x}_{t-\mathbf{M}},\ \gamma^{-1}\mathtt{I}) \tag{4.10}$$

where γ is a tuning parameter called the *diffusion coefficient*. Large values of γ mean high confidence in the prediction, low values mean that tracking tends not to trust the motion model. I set $\gamma = 10$ for all tests.

4.4.4 Computing the Diffused Prior

Next is the computation of the diffused prior $q(\mathbf{x}_t)$ over position, which will give the search window in image t. Section 4.4.3 defines the predicted position of $\mathbf{x}_t$ given the previous positions $\mathbf{x}_{t-\mathbf{M}}$. Recall, however, that the $\mathbf{x}_{t-\mathbf{M}}$ are hidden variables, and that what was observed were the 2D positions $\mathbf{z}_{t-\mathbf{M}}$.

As will be discussed in Section 4.5, the prior will be expressed as a single Gaussian at the end of each tracking iteration. The mean of the Gaussian is $\mathbf{z}_{t-\mathbf{M}}$, and let $\Sigma_{t-\mathbf{M}}$ be the $2(M-1)\times 2(M-1)$ covariance matrix (discussed below). Thus

$$\mathrm{p}(\mathbf{x}_{t-\mathbf{M}}|\mathbf{z}_{t-\mathbf{M}}) = \mathcal{N}(\mathbf{x}_{t-\mathbf{M}} \mid \mathbf{z}_{t-\mathbf{M}},\ \Sigma_{t-\mathbf{M}}) \tag{4.11}$$

and the integral which gives the diffused prior over position in the new frame is

$$q(\mathbf{x}_t) = \int \mathrm{p}(\mathbf{x}_t|\mathbf{x}_{t-\mathbf{M}})\mathrm{p}(\mathbf{x}_{t-\mathbf{M}}|\mathbf{z}_{t-\mathbf{M}})\mathrm{d}\mathbf{x}_{t-\mathbf{M}}. \tag{4.12}$$

Substituting (4.11) and (4.10), we obtain

$$q(\mathbf{x}_t) \propto \int \exp\left(-\gamma\|\mathbf{x}_t - \mathtt{P}\mathbf{u}\|^2\right) \times \exp\left(-(\mathbf{u} - \mathbf{z}_{t-\mathbf{M}})^\top \Sigma_{t-\mathbf{M}}^{-1}(\mathbf{u} - \mathbf{z}_{t-\mathbf{M}})\right) \mathrm{d}\mathbf{u} \tag{4.13}$$

where the variable of integration has been written $\mathbf{u}$ to aid legibility. Evaluating the integral (see Appendix C) yields a normal distribution for $\mathbf{x}_t$ of the form

$$q(\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_t \mid \mathtt{P}\mathbf{z}_{t-\mathbf{M}},\ \gamma^{-1}\mathtt{I} + \mathtt{P}\Sigma_{t-\mathbf{M}}\mathtt{P}^{\top}). \tag{4.14}$$

4.5 Prior Update

Multiplying the likelihood by the diffused prior as above gives a posterior response surface $S(\mathbf{x}) = p(\mathtt{I}_t|\mathbf{x})q(\mathbf{x})$. The posterior distribution $p(\mathbf{x}_t|\mathbf{z}_t)$ is set as an approximation of S using a Gaussian with mean given by the mode of the posterior and a diagonal covariance matrix $\Sigma_t = d^2\mathtt{I}_{2\times 2}$ with d heuristically set to the L_1 distance between the mode of $q(\mathbf{x}_t)$ and $\mathbf{z}_t$.

The prior for time $t{+}1$ can then be updated directly with the posterior distribution. The mean of the full prior is simply the stacked observations from the last $M{-}1$ frames and the covariance is the top-left $2(M{-}1)$-square sub-matrix of the block diagonal matrix

$$\begin{bmatrix} \Sigma_t & \mathtt{0} \\ \mathtt{0} & \Sigma_{t-\mathbf{M}} \end{bmatrix}. \tag{4.15}$$

4.6 Computing the Likelihood

Up until this point, all references to observations have assumed that feature coordinates are directly available. In practice, of course, it is image intensities over position that are observed. The main implications of this are presented in Appendix C. The main point is that some way of handling appearance must be devised. Here, I have employed the work of Matthews et al. (2003). It tries to capture greater operational efficiencies and flexibility by combining the affine warping version of track-to-first (Tomasi and Kanade 1991), known as the Kanade-Lucas-Tomasi or KLT tracker, with track-to-previous. Using this technique, the

likelihood is calculated by cross-correlating the appearance patch extracted from the previous frame with the image of the current one (i.e. track-to-previous). The problems of drift are partly addressed by initiating a KLT optimization from the mode of the posterior and using the result as the observation $\mathbf{z}_t$ to be used in subsequent frames. I have found that the remaining problems of repeated texture and appearance change can be greatly mitigated by using the motion prior described above.

4.6.1 The KLT Update

The KLT update is an image registration technique due to Lucas and Kanade (1981) and its use in a tracking context was presented by Tomasi and Kanade (1991). It models the frame-to-frame appearance change of the template as a cumulative affine warp of the original feature patch:

$$W(\mathbf{x}) = \mathtt{A}\mathbf{x} + \mathbf{t} \tag{4.16}$$

so that

$$[\mathbf{p}_t]_{W_t(\mathbf{x})} = [\mathbf{p}_1]_{\mathbf{x}} \quad \forall \mathbf{x} \tag{4.17}$$

noting that the warp parameters are likely to be different after every frame transition. The warp parameters are determined by a Newton optimization minimizing the sum of squared differences between the warped patch and frame t:

$$\min_{W_t} \|\mathbf{p}_1 - \mathtt{I}_{W_t(\mathcal{X})}\|_2^2 \tag{4.18}$$

The entries in $\mathtt{A}$ are stored and used as the starting point for the next frame's optimization. The translation, $\mathbf{t}$, is just a local offset and so is initiated as the mode of the posterior. The observation, $\mathbf{z}_t$ is set as the value to which $\mathbf{t}$ optimizes.

4.7 Additional Implementation Details

The above steps describe the essential components of the tracking algorithm. It must be emphasized that main novelty of this chapter is in the use of rank constraints to support a medium-range (10-frame) motion model which allows accurate track predictions even with complex motion. In this section the remaining implementation strategies employed are covered.

4.7.1 Multiple Predictions

To get a more robust motion estimate, I actually employ a range of bases to get a series of predictions for each point in each frame, using the range $M = \{6, 7, 8, 9, 10\}$ for $R = 6$. Values of $2M$ too close to the rank generally give motion predictions that are too erratic; making over-constrained estimations tends to be more effective. However, considering previous motion over too many frames leads to the low rank motion approximation breaking down, hence the balance. The posterior is then generated from the mixture of the resulting Gaussians.

4.7.2 KLT Refinement

As the track proceeds, I maintain a set of warp parameters, $\mathtt{A}_t$, holding the affine transformation of the starting feature's appearance, matching it to the current frame. It is initially set to the identity matrix, i.e. no warp. Starting the optimization at the current estimate location (given by maximizing the posterior estimate) and the previous frame's warp, $\mathtt{A}_{t-1}$, the local minimum SSD fit of the original template to the current image is found (Section 4.6.1 above) to give the optimal warp for this frame: $\mathtt{A}^*$. Using the raw output of this KLT optimization is rather unstable in practice and so an ad-hoc scheme was created to soften the progress of the warp parameters. $\mathtt{A}_t$ is generated in two steps. The first blends the previous frame's matrix with the new warp according to the 'plausibility' of the new warp, $\mathtt{A}^*$. If the new warp parameters seem too drastic, they are ignored. Because we can expect little or no scaling

between consecutive frames, this first blend is based on the eigenvalues of new affine warp matrix, plus the difference between the new and old:

$$\mathtt{A}' = \alpha_1 \mathtt{A}^* + (1 - \alpha_1)\mathtt{A}_{t-1} \tag{4.19}$$

with

$$\alpha_1 = \frac{k_1 \exp\left(-k_2(\sigma_1^2 - \sigma_2^2)^2\right) + k_3 \exp\left(k_4 \|\mathtt{A}^* - \mathtt{A}_{t-1}\|^2\right)}{k_1 + k_3} \tag{4.20}$$

where σ_1 and σ_2 are the eigenvalues of $\mathtt{A}^*$.

Furthermore, as parameter drift is a real threat in the cumulative update paradigm, so an attempt is made to counteract the potential of the KLT fitting process to take the warp to extreme distortions. The is done by a) using the absolute difference in appearance between the original template and the warped match in the current frame, and b) adding a prior on the scaling effect of the warp. These two attributes modulate a second blend of the warp, $\mathtt{A}'$, with the identity matrix. This provides a helpful restraining influence on run-away optimizations of (4.18):

$$\mathtt{A}_t = \alpha_2 \mathtt{A}' + (1 - \alpha_2)\mathtt{I} \tag{4.21}$$

with

$$\alpha_2 = \frac{k_5 \exp\left(-k_6 \|\mathbf{p}_1 - \mathtt{I}_W\|^2\right) + k_7 \exp\left(-k_8(\sigma_3^2 + \sigma_4^2)\right) + k_9 \exp\left(-k_{10}(\sigma_3^2 - \sigma_4^2)^2\right)}{k_5 + k_7 + k_9} \tag{4.22}$$

where σ_3 and σ_4 are the eigenvalues of $\mathtt{A}'$.

The parameter constants $\mathbf{k}$ were set by hand to achieve good performance across all the test sequences: $\mathbf{k} = [1, 1, 0.8, 4, 1, k_6, 0.3, 0.635, 1, 5]$, with k_6 being set to 30% of the maximum pixel intensity level squared divided by the number of pixels in the image patches used.

4.7.3 Robustifying the Diffused Prior

In the form in (4.10), too much confidence is placed in the prediction of the motion model for practical use, so I implement a robust prior. I scale the prior to have maximum value one and make a mixture with a uniform distribution. The more robust prior, denoted q', is then defined by

$$q'(\mathbf{x}_t) \propto \alpha \frac{q(\mathbf{x}_t)}{\max_{\mathbf{x}} q(\mathbf{x})} + (1 - \alpha). \tag{4.23}$$

The blend coefficient, α, reflects the confidence we have in the model's predictive powers for the trajectory $\mathbf{z}_{t-\mathbf{M}}$. It can be quantified by comparing the coefficient vector $\mathbf{c}$ of the preceding trajectory $\mathbf{z}_{t-\mathbf{M}}$, given by $\mathbf{c} = \mathtt{B}^{+}\mathbf{z}_{t-\mathbf{M}}$, to the coefficients of all the inlying motion in the measurement matrix, $\mathtt{M}$. If the motion of the tracked feature currently matches that seen in the rest of the scene, then the prediction made by the basis can be taken as being good, hence

$$\alpha = e^{-\beta d_{min}} \tag{4.24}$$

for a scaling parameter β controlling the speed at which α decays with distance (all reported tests used $\beta = 0.0005$) and

$$d_{min} = \min_{i}(\|\mathbf{c} - \mathbf{c}_i\|_2) \tag{4.25}$$

where $\mathbf{c}_i$ is the i^{th} column of $\mathtt{B}^{+}\mathtt{X}$, for $\mathtt{X}$ the 'measurement matrix' of inliers used to determine $\mathtt{B}$. Effectively, this is a nearest neighbor estimate of the density of the basis coefficients.

When using multiple predictions, these α values are useful as weights for when the predictions are combined. I use them when averaging the modes of $q(\mathbf{x}_t^i)$ for the calculation of d in Equation (4.15).

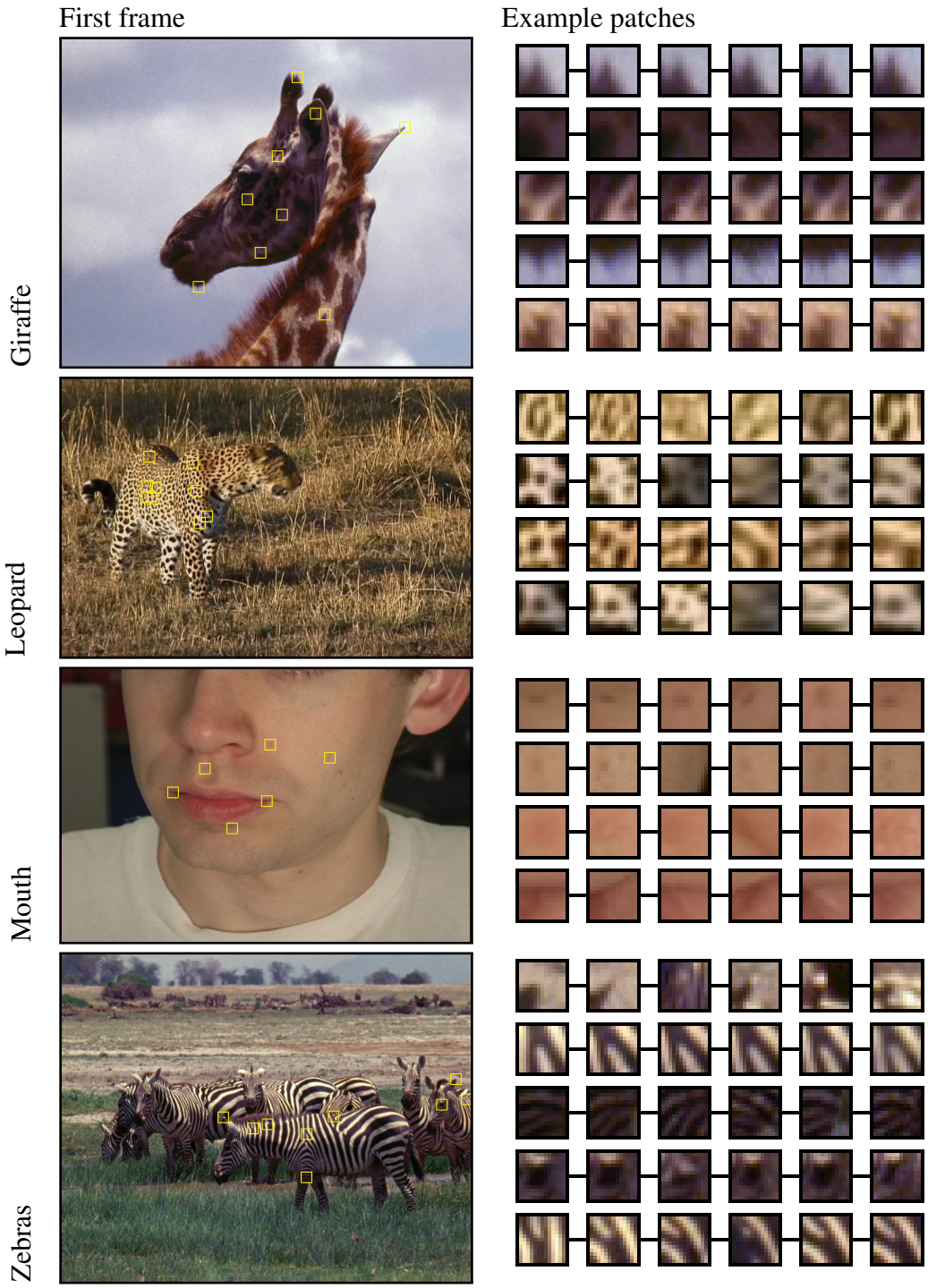


Figure 4.4: **Test sequences.** The first frame of each sequence used in the evaluation, with the start of the ground truth tracks shown. On the right is an indication of the range of appearance variation in the feature tracks used. Note that feature choice was limited to features that a) could be tracked consistently and accurately by a human and b) were continuously visible for the whole sequence. See text for more comments.

4.8 Experiments

Ground truth for tracking on four sample image sequences, summarized in Table 4.1, was obtained manually and for features as illustrated in Figure 4.4. Tracking challenges in the sequences include appearance variation, lighting changes and considerable motion blur.

As the main contribution is in the form of the prior, I ran, along with the tracker described, the same Bayesian KLT tracker employing three other functions for calculating the prior:

Uniform. A uniform prior over a constant-size search window (61×61 pixels).

Acceleration. A constant acceleration model (covariances as described in Section 4.5) whose motion parameters are re-estimated every frame using the last three observations.

Median. The median two-frame global motion within a 30 pixel radius (Smith et al. 1998) with covariances based on support.

The first experiment measures tracker reliability. Throughout the testing, 13×13 image patch templates were used and rank-6 motion bases were employed. The trackers were initialized on the ground truth track positions in image 1 of each sequence. When the track drifted off position by more than Δ pixels, a track failure was recorded. This was then repeated, starting the trackers in each of the first 100 frames of the sequences (first 50 for the shorter giraffe sequence) in order to average out any artifacts that may occur due to starting in any particular frame. This average track length is an important predictor of performance on many tracking tasks (e.g. structure and motion recovery (Hartley and Zisserman 2004)). Table 4.2 summarizes the average track length improvements of my proposed motion prior compared to the three models described above for $\Delta = 4$. Average track lengths increased in all cases, with improvements of up to 80%, 45% and 12% over the uniform, acceleration and median models respectively. Figure 4.5 shows how the mean track length improvement, averaged over the four sequences, increases roughly linearly with Δ.

Name	Resolution	Length	Texture	Objects	Tracks
Giraffe	720×576	100	med	1 (N)	9
Leopard	700×475	242	high	2 (N+N)	9
Mouth	720×480	346	low	2 (N+R)	6
Zebras	720×576	171	high	10 (9N+R)	9

Table 4.1: **Ground-truth test sequences.** "Resolution" is in pixels. "Length" is measured in frames. "Texture" indicates the density of texture in the scene. "Objects" is the number of independently moving objects in the sequence as a whole (N: non-rigid, R: rigid). "Tracks" is the number of ground-truth tracks evaluated on each.

Proposed vs.	Giraffe	Leopard	Mouth	Zebras
Uniform	**46.9%**	**80.9%**	**44.8%**	**16.0%**
Acceleration	**45.4%**	**14.1%**	**18.1%**	**6.0%**
Median	**12.6%**	**3.8%**	**3.0%**	**3.8%**

Table 4.2: **Results: track length I**. Average improvement of correct track length in example sequences of the proposed motion model over the three existing models.

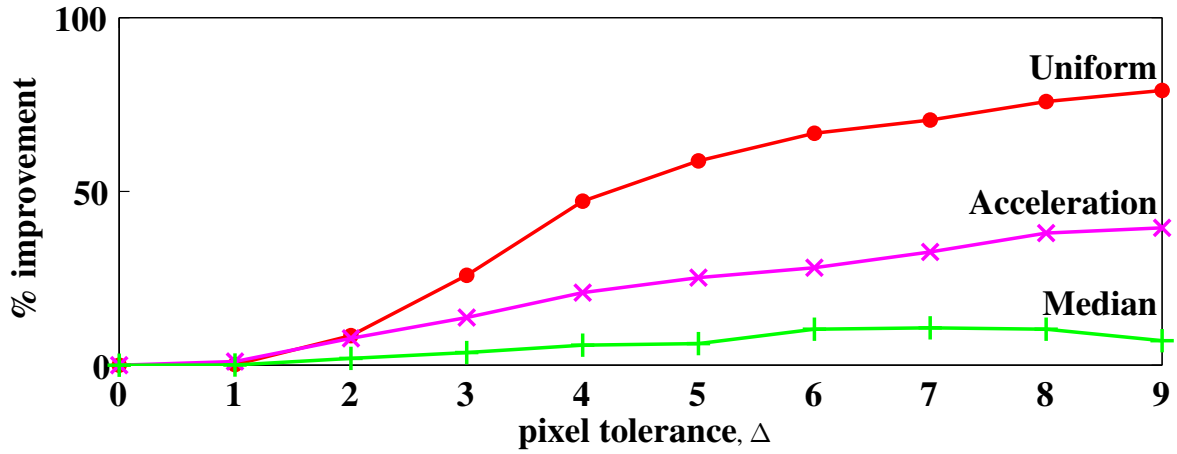


Figure 4.5: **Results: track length II**. Average improvement in track lengths over the existing motion models across all four sequences against the accuracy threshold Δ (deviation from ground-truth). The new proposal outperforms the alternatives at all levels.

A few track case-studies are worth comments. In the giraffe sequence, the motion on the ear is very different to that in the rest of the scene. Even though it is a relatively small object, its motion was successfully captured by the motion model. The mouth video is challenging mainly because it lacks texture. Despite this, the global motion model was able to successfully support motion in the problematic regions. The leopard sequence also provides a difficult set of features to track, having large non-rigid deformations, significant motion blur and very self-similar texture. However, the combination of local and global motion information is able to guide the tracker through these difficulties. The zebras are a good example of the power of the sliding temporal window: methods that try to find a global solution are likely to fail here because of the large number of independent non-rigid objects in the scene. By considering 10-frame sub-sequences, rank-6 motion models were adequate to cover the complex motions observed.

A second experiment investigates the *predictive power* of the proposed motion model, compared to the three alternatives. It was calculated as the RMS pixel error of the predictions made by each model using the ground truth data as the prior observations. The results, presented in Table 4.3, show that improvements of up to 80%, 50% and 20%, over each of the standard models respectively, are possible. On the giraffe sequence, the median filter predicted slightly better (despite performing less well when the whole system is considered), probably due to the high texture density in this scene.

Only a small number of tracks, those for which I have ground truth, have been discussed here, but the qualitative performance is typical. It is important to note that the quality of the tracks were improved in all cases through the use of the motion prior, that is, the use of the extra information very rarely degrades the results and often provides a substantial increase in performance.

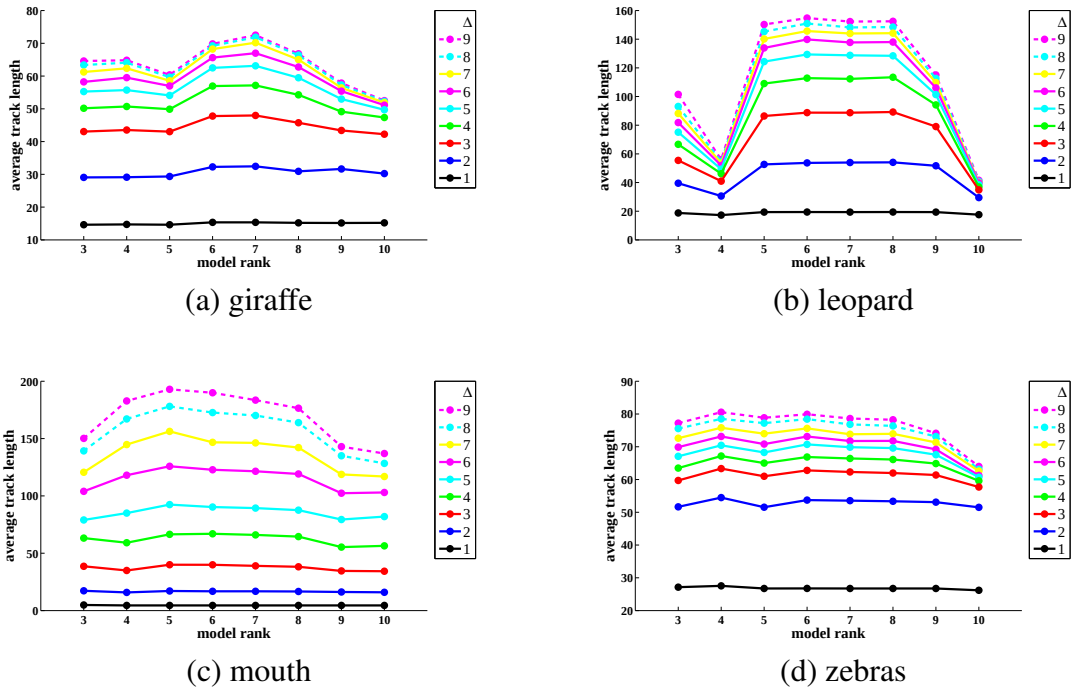
The impact on computational speed is primarily in the computation of the motion models, with complexity approximately equivalent to the second-pass stage, so that the addition of priors approximately doubles the computational complexity. For an interactive tracking

Sequence	**Proposed**	Uniform	Acceleration	Median
Giraffe	**1.85**	2.92 (**36.6%**)	2.62 (**29.4%**)	1.67 (**-10.5%**)
Leopard	**1.61**	3.79 (**57.4%**)	2.59 (**37.6%**)	1.83 (**11.9%**)
Mouth	**1.72**	7.89 (**78.2%**)	3.49 (**50.6%**)	1.75 (**1.4%**)
Zebras	**1.22**	2.68 (**54.6%**)	2.03 (**40.1%**)	1.52 (**20.0%**)

Table 4.3: **Results: predictive power**. The average RMS error, in pixels, of predictions using the ground truth data. Percentage improvement achieved by my proposal is given in brackets. Here, 'Uniform' is the constant position model and gives a scale reference.

scenario, the primary speed requirement is in pass two, as the motion models can be precomputed when the sequence is loaded from tape or scanned. In this case, a speed advantage is enjoyed when the prior is tight because the size of the search window is reduced.

As mentioned, all the above results are reported for tracking using the motion model with the basis rank set to 6. Six was chosen because it was seen to provide adequate motion models in the majority of cases. As such, it is the recommended default value and was used for these results for better direct comparisons. However, because ground truth tracks are available for these trials, it is possible to go further behind the scenes than would be the case in normal situations and observe the performance over a range of basis ranks. Figure 4.6 shows the variation of track length, averaged over start frames and feature tracks (as above), against rank for each of the four sequences. The variation in the performance profile varies between the sequences, but broadly speaking, for these sequences, the best performance was obtained with ranks around six, with performance degrading for higher and lower choices. The effect of varying the rank is exaggerated for larger tolerances. For $\Delta = 4$ (as for the results above) the best performing ranks were 7, 8, 6, 4 for the giraffe, leopard, mouth and zebras respectively. As it happens, for the three sequences whose top rank value was not six, six was still the second best choice. Comments could be made in attempts to explain physical interpretations of these numbers (e.g. planar plus two basis shapes), but I will leave this to motivated readers, as I would prefer to emphasize the lack of any reference to underlying physical models in the formulation by omitting any such interpretations.



(a) giraffe (b) leopard

(c) mouth (d) zebras

Figure 4.6: **Results: track length III.** Average track length (over start frames and feature tracks, as above) against basis rank for each of the four test sequences, plotted for a range of tolerances. Values below 5 and above 8 tend to generate worsening performance, with 6 being a good, 'catch all' choice.

4.9 Concluding Remarks

This chapter has demonstrated that local feature tracking can be guided using global scene motion information in an efficient algorithm. By making predictions of the new location of a feature point using a low rank approximation to the motion in the rest of the frame, tracking algorithms can be made to generate longer and more accurate tracks. The confusion caused to standard tracking algorithms by repeated texture and ill-defined templates can be mitigated with this prior.

Some characteristics of the algorithm are worthy of note. Even though the track prediction is a simple linear projection (Equation 4.8), it can model complex non-rigid motions and can predict from tracks with jumps and cusps, because of the use of the long time history and because of the fitting of the global model. In effect I have presented a high-order Kalman filter where the state transition matrix is specialized to every frame of the sequence.

Technically, this "prior" is not independent of the information used in the calculation of the likelihood, both being derived from the pixel intensity values of the image sequence. In practice, though, the distance between the feature being tracked and the points used for the motion model is large enough, on average, to allow us to treat the two as being independent.

I have used a relatively simple base tracker into which to impose the priors, ignoring much of the recent work on template updating, occlusion, and other non-motion priors (see Chapter 2). It would appear reasonable, however, to expect that the global motion model will improve any base tracker that uses a simpler (or no) motion model.

A salient issue not addressed here was that of initialization. For the first `M` frames of a sequence, when motion predictions can not be made, an alternative strategy must be used. For the examples given in this paper, the prior was set to be uniform for these initial frames, i.e. the trackers compared had identical behaviour for those first frames to aid the comparison. More sophisticated methods can easily be employed, such as multiple hypothesis techniques. Once enough frames have been tracked, selection between alternative hypotheses can be aided using the prior for the full tracks, i.e. using the motion basis for those frames to calculate reprojection errors and coefficient distances.

Chapter 5

Conclusions

Tracking is an important task in computer vision. In practice, many of the target image sequences possess attributes that pose severe difficulties for existing tracking algorithms: non-rigid motion, multiple independent objects, repeated texture, occlusions, small objects and the like. When tracking fails, human intervention is required to obtain the desired track. I have presented new methods to overcome the difficulties and hence reduce the level of intervention required.

Firstly, I present a novel tracking system. Frequently, people need specific tracking data from complex video sequences. The approach is one of an iterative process of automated attempts and manual corrections. Existing solutions use inadequate automation and tend to require excessive manual input. I addressed both these issues by considering a global view on automatic track generation and designing a framework to maximize the work flow involved in track generation. I conclude that, in the off-line context, there are two significant points that should attract more attention: i) that the image sequence can be pre-processed to dramatically decrease the time taken to construct a track, and ii) that the global optimization of the track's trajectory, taking into account the progression of appearance, produces far superior results.

Secondly, I introduce a new form of motion model. Existing feature point tracking al-

gorithms tend to rely only on local motion information, i.e. the trajectory of the track itself. A much better idea of feature motion can be gained by looking at the motion across the rest of the frame. The new motion model allows global frame motion to guide the feature point tracking process without the problems inherent in alternative approaches. This is achieved by i) considering the global motion over a medium term temporal window, and ii) using the 2D frame motion directly, without underlying assumptions about the image generation process. The result is a flexible and successful motion prior that can be incorporated into any tracking scheme.

Thirdly, I conducted a review of matrix factorization techniques for situations when not all the elements of the matrix are known. This is a significant topic because it plays a foundational role in many important computer vision tasks. It is also directly applicable to the generation of the new model motions described in this thesis. In addition, I propose a new class of hybrid algorithms which combine the two most prominent existing techniques for performance, which is, on average, superior to both. Furthermore, it is noted that the tailoring and optimization of the factorization process to a specific target task is important if the best results are to be achieved.

There is naturally considerable scope for extension of this work. For the interactive tracker, many optimizations are possible to improve on the time statistics reported. In addition, the presentation of trajectory optimization parameters for tweaking during use is not ideal if the system were to be presented to the general public. Ways to automatically set these parameters, on a track-by-track basis, should be developed, may be with more intuitive options if the user were to be involved (for example, asking whether the sequence contained 'no occlusions' or 'heavy occlusions'). For the factorization work, investigations into robustification and context-specific constraints, for example, would be pertinent. For the global motion model, I would like to develop the full use of the matrix factorization techniques discussed in this thesis to provide better motion estimates in more difficult video. Also, it would be beneficial to incorporate the prior into other tracking frameworks. Using it for the

interactive tracker would be particularly interesting because great care would be needed to ensure that high processing rates were maintained.

Appendix A

K-d Trees

K-d trees were introduced in Chapter 1, but only an overview of their implementation and efficiencies was given. I now give a description of some of the various ways in which k-d trees can be realized. Following on from the description of a k-d tree given in Chapter 1, the variables that need to be defined are:

D	dimension of the data
P	number of initial data points
$h \leq \lceil \log_2 P \rceil$	height of the tree (excluding leaf nodes)
$N = 2^h$	number of leaf nodes
$\mathtt{X} = [\mathbf{x}_1\ \mathbf{x}_2\ \cdots\ \mathbf{x}_P]$	the initial data set
$\mathbf{d}$	vector of per-node dimension attributes
$\mathbf{t}$	vector of per-node threshold attributes
$\mathbf{l}$	vector of per-leaf "data index list" pointers

There are two stages associated with k-d tree use: building and searching. The building phase is the construction of the tree, in which the entries in the vectors $\mathbf{d}$, $\mathbf{t}$ and $\mathbf{l}$ are determined. These three vectors, together with the original data, $\mathtt{X}$, constitute the tree. Algorithm A.1 describes the process. The fractal nature of binary trees makes it possible to use a recursive formulation. I have used recursion here to take advantage of the concise nature of the

resulting algorithms.

Algorithm A.1 Function 'build k-d tree'.

inputs $\mathtt{X} \in \mathbb{R}^{D \times P}$, $h \in \mathbb{N}$

declare $N = 2^h$
declare $\mathbf{d} \in \mathbb{N}^{N-1}$, $\mathbf{t} \in \mathbb{R}^{N-1}$, $\mathbf{l} \in \mathbb{N}^{N}$
recurse ($\mathtt{M}$, i, c)
  if <first call> **then** *// initialize*
    declare (local) $\mathtt{M} = \mathtt{X}$
    declare (local) $i = 1$
    declare (local) $c = 1$
  end if
  $d_i = f_{\text{dim}}(i, \mathtt{M})$ *// set i^{th} entry in* $\mathbf{d}$
  $t_i = \text{median}(\mathbf{m}^{d_i})$ *// median of d_i^{th} row of* $\mathtt{M}$
  // partition data in $\mathtt{M}$ *by value in d_i^{th} dimension*
  $\{j\} \leftarrow$ indices for which $m_j^{d_i} < t_i$
  $\{k\} \leftarrow$ indices for which $m_k^{d_i} \geq t_i$
  if $c \leq h$ **then**
    recurse ($\mathtt{M}_{\{j\}}$, $2i$, $c+1$) *// send columns left*
    recurse ($\mathtt{M}_{\{k\}}$, $2i+1$, $c+1$) *// send columns right*
  else
    l_{2i-N+1} = pointer to permanent storage of $\{j\}$
    l_{2i-N+2} = pointer to permanent storage of $\{k\}$
  end if

outputs $\mathbf{d}$, $\mathbf{t}$, $\mathbf{l}$

The 'build k-d tree' function also requires there to exist another function (denoted f_{dim} in the code listing): a function that chooses the data dimension in which partitions are made. In its simplest form, this function would return a constant, e.g. $f_{\text{dim}}(i, \mathtt{M}) = 1$. In my initial implementations, I set this function to cycle through dimensions, starting from 1:

$$f_{\text{dim}}(i, \mathtt{M}) = \mod (c-1, D) + 1 \tag{A.1}$$

noting that the current node depth and the dimensionality of the data can be calculated from i and $\mathtt{M}$ respectively. In my second generation of implementations, I set f_{dim} to return the

dimension which holds the largest standard deviation of the values in $\mathtt{M}$:

$$f_{\text{dim}}(i, \mathtt{M}) = \underset{d}{\text{argmin}}(\sigma_d) \quad \text{for} \quad \sigma_d = \text{std}(\mathbf{m}^d). \tag{A.2}$$

The other function needed for the use of k-d trees is the search function. There are several subtly different embodiments of the k-d tree search, but I will present two here: the 'partition search', which returns a list of the points in $\mathtt{X}$ that are in the same partition (k-d tree leaf) as the query point $\mathbf{q}$, and the 'k nearest neighbour search', which provides a list of the k points in $\mathtt{X}$ that are closest to $\mathbf{q}$. The other variations of search (for example finding all points in $\mathtt{X}$ which are within a given radius of the query point $\mathbf{q}$) and are mostly trivial alterations of the 'k nearest-neighbour search algorithm' (hereafter referred to as 'kNN' searching).

Algorithm A.2 Function 'partition search'.

inputs $\mathtt{X} \in \mathbb{R}^{D \times P}$, $\mathbf{d} \in \mathbb{N}^{N-1}$, $\mathbf{t} \in \mathbb{R}^{N-1}$, $\mathbf{l} \in \mathbb{N}^{N}$, $\mathbf{q} \in \mathbb{R}^{D}$

declare $\mathtt{m} \in \mathbb{R}^{D \times P}$
recurse (i, c)
  if <first call> **then** *// initialize*
    declare (local) $i = 1$
    declare (local) $c = 1$
  end if
  if $c \leq h$ **then**
    if $\mathbf{q}_{d_i} < t_i$ **then**
      recurse($2i, c + 1$) *// search left*
    else
      recurse($2i + 1, c + 1$) *// search right*
    end if
  else
    $\mathtt{M} \leftarrow \mathtt{X}_{\{l_{i-N+1}\}}$
  end if

outputs $\mathtt{M}$

The 'partition search' function is presented in Algorithm A.2. I have started with this as it is the base for all other k-d tree search functions. The number of points that exist at any given leaf node is a function of the number of data points, P, their spread and the number of

leaf nodes, N. There is the obvious relation that if $P \approx N$ then there will be roughly one data point per leaf, but if $P > N$ then we can expect P/N points per leaf. More interestingly, if the chosen partition dimensions ever happen to contain more than half the points with the same value in that dimension, then more points will be sent to the righthand child than to the left, making the tree unbalanced and giving some leaves more points than others (which is acceptable). However, knowing all the points that are in the search partition does not guarantee that the nearest point has been found. In an extremely unbalanced tree, that leaf could hold no points. More commonly, if $\mathbf{q}$ is close to the partition boundary then the nearest point could be on the other side of that boundary, in a neighbouring partition. For this reason, the process of *backtracking* must be introduced. Conceptually, this is a process of stepping back up the tree after the leaf nodes have been reached to explore other branches: the tree is traversed as in Algorithm A.2, and the distances to all the points at the resulting leaf node are calculated; then the visited internal nodes are revisited and their other child nodes are searched in the normal way if the distance to the threshold at that node is less than the distance to the current closest point; in this way other leaf nodes are visited and the closest point is updated as the distances to new points are calculated. This process is presented in Algorithm A.3. It has been extended to work for the 'kNN' search, which requires that a list structure be introduced to hold the top k neighbours as the search progresses. However, the exact implementation of this data structure is open for optimization. It must simply provide two functions: one is to return the distance to the furthest point from $\mathbf{q}$ it holds; and the other is to appropriately add new points to itself. By appropriate, I mean that if the point is further away from $\mathbf{q}$ than any of the points already there, it should be discarded (unless the list is not yet full), otherwise it should be added to the list and the worst existing point should be discarded.

The main aspects for variation and optimization of the 'kNN' algorithm are:

- The **kNN data structure** used to implement the list $\mathcal{Q}$

Algorithm A.3 Function 'k nearest neighbour search'.

inputs $\mathtt{X} \in \mathbb{R}^{D \times P}$, $\mathbf{d} \in \mathbb{N}^{N-1}$, $\mathbf{t} \in \mathbb{R}^{N-1}$, $\mathbf{l} \in \mathbb{N}^{N}$, $\mathbf{q} \in \mathbb{R}^{D}$, $k \in \mathbb{N}$

declare $k - \text{list}\mathcal{Q}$ *// holds k (index,distance) pairs*
recurse (i, c)
  if <first call> **then** *// initialize*
    declare (local) $i = 1$
    declare (local) $c = 1$
  end if
  if $c \leq h$ **then**
    if $q_{d_i} < t_i$ **then**
      recurse $(2i, c + 1)$ *// search left*
      if $\|q_{d_i} - t_i\| < \text{argmax}_{\text{distance}}\, \mathcal{Q}$ **then**
        recurse($2i + 1, c + 1$) *// search right*
      end if
    else
      recurse($2i + 1, c + 1$) *// search right*
      if $\|q_{d_i} - t_i\| < \text{argmax}_{\text{distance}}\, \mathcal{Q}$ **then**
        recurse($2i, c + 1$) *// search left*
      end if
    end if
  else
    for all $j \in \{l_{i-N+1}\}$
      add $(j, \|\mathbf{q} - \mathbf{x}_j\|)$ to $\mathcal{Q}$
    end for
  end if

outputs $\mathtt{X}_{\mathcal{Q}}$

- The **distance calculation** between $\mathbf{q}$ and the data points $\mathbf{x}_i$
- The **node visitation order** specifying the backtracking strategy
- The **partition distance function** of $\mathbf{q}$ and the splitting planes

I will now describe the options I considered for each aspect.

List data structure. As mentioned above, the implementation of $\mathcal{Q}$ must provide for the execution of two functions. The first is making available the longest distance (from $\mathbf{q}$ to the points held in it). If the points are kept in an ordered list, then this operation becomes an $O(1)$ look-up. Maintaining an ordered list also helps the efficiency of the add operation,

because the decision about whether a new point should be added and which should be replaced becomes a constant time operation too. The remaining question is how to maintain the ordering as new points are added. I considered three options: a) a linked-list with bubble sort, b) a linked-list with insert sort, and c) a priority heap.

Distance calculations. It is quite possible that a leaf node holds multiple points. These are searched exhaustively when a leaf node is visited. However, it is not always necessary to fully calculate the distance from $\mathbf{q}$ to every point in that leaf. During the calculation of the Euclidean distance, the running summation can be compared to the worst distance in $\mathcal{Q}$. If the summation exceeds that distance, then the rest of the distance calculation can be skipped. This is sometimes referred to as an 'early bailout' scheme.

Node visitation order. The order in which the internal nodes are visited can be adjusted in an attempt to minimize the amount of backtracking undertaken. A natural ordering is achieved implicitly using the recursive formulation above. Alternatively, a list structure, similar to $\mathcal{Q}$ can be used to decide which internal node to visit next. I implemented a LIFO (last in, first out) stack, where, upon visiting a node, its two children are added to the stack in reverse order of relevance (which is actually equivalent to the recursive approach, but was investigated in order to test function call efficiency) and also a priority queue, where a node's children are added in the same way as for a stack, but the most promising node is visited next (allowing the algorithm to jump across the tree when it is seemingly advantageous).

Partition distance function. It is possible to account for the fact that at each node, a splitting dimension can be different. The standard implementation sees the use of the perpendicular distance to the current splitting plane for every node traversal decision. Instead, one can use an estimate of the distance to the closest point in the neighbouring partition concerned. This is achieved by using the splitting dimensions seen at all the nodes from the root to the one at hand. This has been dubbed the 'incremental distance calculation' (Arya and Mount

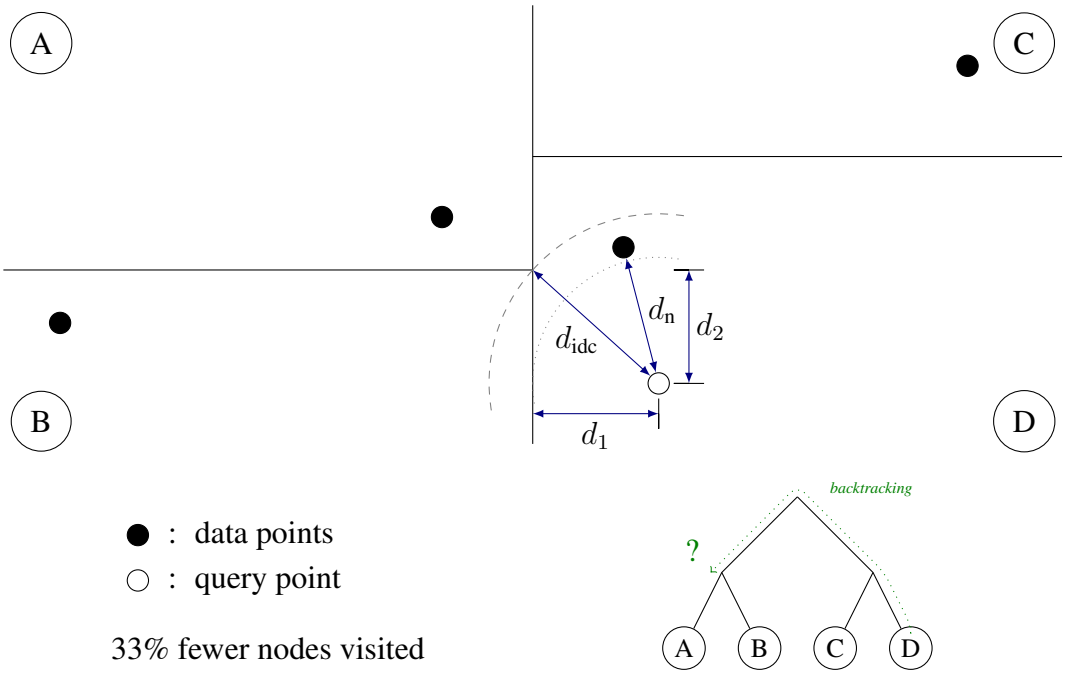


Figure A.1: **Incremental distance calculation.** A query point is in partition D of a four leaf k-d tree. The nearest point in this partition is at a distance of d_n. Backtracking to the C-D internal node reveals that no points in C can be closer than d_n, so the option of visiting C is skipped and backtracking continues to the root node. The distance to the root partition is d_1, which is less than d_n and so a closer point could exist in A or B. Backtracking therefore visits internal node A-B. The query point is on the lower side of the A-B boundary and so B is visited first. As it happens, the point in B is further away than d_n. Backtracking then considers visiting node A. In standard k-d tree implementations, distance d_2 would be used to decide whether to visit A. As d_2 is less than d_n, the answer would be yes. However, taking d_1 into account shows that the nearest point in A would be at a distance of d_{idc} and so it is not actually worth visiting A after all. The 'incremental distance calculation' scheme is a way of taking into account the distances to all the partitions as a k-d tree is descended from the root, making it possible for backtracking to visit fewer leaf nodes.

1993). This distance is then used, instead of the last threshold difference distance, to decide whether to backtrack into any particular subtree. See Figure A.1 for a demonstration.

Fourteen combinations of the above possibilities were tested and their performance over a range of data sets (both from image sequence data and synthetic data), dimensions, leaf counts and k (nearest neighbour) values was evaluated. The winning implementation consistently outperformed all other methods. It used early bailout exhaustive search, insert sort on $\mathcal{Q}$, recursive back-tracking and incremental partition distance calculations. The results

showed that, up to at least 36 dimensions, using this optimal implementation is always at least as good as optimized exhaustive search over the whole data set. For low dimensional data ($D<10$), k-d trees are the best choice by many orders of magnitude. For higher dimensions, performance on random data is effectively the same as for exhaustive search. However, when the data are clustered or structured (like a mixture of Gaussians, for example), a constant linear speed-up (a threefold time saving is not uncommon) is achieved by k-d trees. My recommendation, therefore, is that if exact nearest neighbour searches are to be performed, k-d trees should always be considered.

Appendix B

Alternation Derivatives

The error function, $F(\mathtt{A},\mathtt{B}) = \|\mathtt{W} \odot (\mathtt{M} - \mathtt{A}\mathtt{B}^\top)\|_F^2$, can be minimized, with respect to either of the two parameters (given a particular value for the other), using a closed form solution. Here, the derivation via zero-crossings of the gradient function is presented. The solution for $\mathtt{A}$, for a given $\mathtt{B}$, is laid out. The solution for $\mathtt{B}$ is very similar and so will not be expanded. Firstly, F is rewritten in terms of the rows of $\mathtt{A}$.

$$F(\mathtt{A},\mathtt{B}) = \left\| \mathtt{W} \odot \left(\mathtt{M} - \mathtt{A}\mathtt{B}^\top\right) \right\|_F^2 \tag{B.1}$$

$$= \sum_{i=1}^{m} \left\| \mathbf{w}^{i\top} \odot \left(\mathbf{m}^{i\top} - \mathbf{a}^{i\top}\mathtt{B}^\top\right) \right\|_F^2 \tag{B.2}$$

$$= \sum_{i=1}^{m} \left\| \operatorname{diag}(\mathbf{w}^i) \left(\mathbf{m}^i - \mathtt{B}\mathbf{a}^i\right) \right\|_F^2 \tag{B.3}$$

$$= \sum_{i=1}^{m} \left(\mathbf{m}^i - \mathtt{B}\mathbf{a}^i\right)^\top \operatorname{diag}(\mathbf{w}^i)^2 \left(\mathbf{m}^i - \mathtt{B}\mathbf{a}^i\right) \tag{B.4}$$

$$= \sum_{i=1}^{m} \mathbf{m}^{i\top}\operatorname{diag}(\mathbf{w}^i)^2\mathbf{m}^i - 2\mathbf{m}^{i\top}\operatorname{diag}(\mathbf{w}^i)^2\mathtt{B}\mathbf{a}^i + \mathbf{a}^{i\top}\mathtt{B}^\top\operatorname{diag}(\mathbf{w}^i)^2\mathtt{B}\mathbf{a}^i \tag{B.5}$$

Taking the first and second derivatives of this sum with respect to $\mathbf{a}^i$ gives

$$\frac{\partial F}{\partial \mathbf{a}^i} = \left(-2\mathbf{m}^{i\mathsf{T}}\mathrm{diag}(\mathbf{w}^i)^2\mathtt{B} + 2\mathbf{a}^{i\mathsf{T}}\mathtt{B}^\top\mathrm{diag}(\mathbf{w}^i)^2\mathtt{B}\right)^\top \tag{B.6}$$

$$= 2\left(\mathtt{B}^\top\mathrm{diag}(\mathbf{w}^i)^2\mathtt{B}\mathbf{a}^i - \mathtt{B}^\top\mathrm{diag}(\mathbf{w}^i)^2\mathbf{m}^i\right) \tag{B.7}$$

$$\frac{\partial^2 F}{\partial \mathbf{a}^{i2}} = 2\mathtt{B}^\top\mathrm{diag}(\mathbf{w}^i)^2\mathtt{B} \tag{B.8}$$

The second derivative is at least positive semi-definite and if there are more than r non-zero entries along the diagonal of $\mathtt{W}^i$ then it is positive-definite (remember that $\mathtt{B}$ is defined as a rank r, $n \times r$ matrix). In the latter case, there is only one zero-crossing of the first derivative and that is a minimum. Setting the gradient to zero gives the solution:

$$\frac{\partial F}{\partial \mathbf{a}^i} = 0 \tag{B.9}$$

$$\implies \quad \mathtt{B}^\top\mathrm{diag}(\mathbf{w}^i)^2\mathtt{B}\mathbf{a}^i = \mathtt{B}^\top\mathrm{diag}(\mathbf{w}^i)^2\mathbf{m}^i \tag{B.10}$$

$$\implies \quad \mathbf{a}^i = \left(\mathtt{B}^\top\mathrm{diag}(\mathbf{w}^i)^2\mathtt{B}\right)^{-1}\mathtt{B}^\top\mathrm{diag}(\mathbf{w}^i)^2\mathbf{m}^i = (\mathrm{diag}(\mathbf{w}^i)\mathtt{B})^+\mathrm{diag}(\mathbf{w}^i)\mathbf{m}^i \tag{B.11}$$

The solution for $\mathtt{B}$, given a matrix $\mathtt{A}$, is very similar. The result comes out to be

$$\mathbf{b}^i = \left(\mathtt{A}^\top\mathrm{diag}(\mathbf{w}_j)^2\mathtt{A}\right)^{-1}\mathtt{A}^\top\mathrm{diag}(\mathbf{w}_j)^2\mathbf{m}_j = (\mathrm{diag}(\mathbf{w}_j)\mathtt{A})^+\mathrm{diag}(\mathbf{w}_j)\mathbf{m}_j \tag{B.12}$$

Higher Order Terms

Differentiating the error function further reveals more of the quartic nature of the main error function. The following derivatives are made with respect to the individual elements of $\mathtt{A}$ and $\mathtt{B}$, continuing from the equations in Section 3.3.

Third derivatives:

$$\frac{\partial^3 F}{\partial a_{ab}\partial a_{ef}\partial a_{op}} = 0 \tag{B.13}$$

$$\frac{\partial^3 F}{\partial b_{cd}\partial a_{ef}\partial a_{op}} = 2w_{ec}^2\delta_{eo}\left(\delta_{df}b_{cp} + \delta_{dp}b_{cf}\right) \tag{B.14}$$

$$\frac{\partial^3 F}{\partial a_{ab}\partial b_{gh}\partial a_{op}} = 2w_{ag}^2\delta_{ao}\left(\delta_{bh}b_{gp} + \delta_{hp}b_{gb}\right) \tag{B.15}$$

$$\frac{\partial^3 F}{\partial b_{cd}\partial b_{gh}\partial a_{op}} = 2w_{oc}^2\delta_{eg}\left(\delta_{dp}a_{oh} + \delta_{hp}a_{od}\right) \tag{B.16}$$

$$\frac{\partial^3 F}{\partial a_{ab}\partial a_{ef}\partial b_{qr}} = 2w_{aq}^2\delta_{ae}\left(\delta_{br}b_{qf} + \delta_{fr}b_{qb}\right) \tag{B.17}$$

$$\frac{\partial^3 F}{\partial b_{cd}\partial a_{ef}\partial b_{qr}} = 2w_{ec}^2\delta_{cq}\left(\delta_{df}a_{er} + \delta_{fr}a_{ed}\right) \tag{B.18}$$

$$\frac{\partial^3 F}{\partial a_{ab}\partial b_{gh}\partial b_{qr}} = 2w_{ag}^2\delta_{gq}\left(\delta_{bh}a_{ar} + \delta_{br}a_{ah}\right) \tag{B.19}$$

$$\frac{\partial^3 F}{\partial b_{cd}\partial b_{gh}\partial b_{qr}} = 0 \tag{B.20}$$

Forth derivatives:

$$\frac{\partial^4 F}{\partial a_{ab}\partial a_{ef}\partial a_{op}\partial a_{st}} = 0 \tag{B.21}$$

$$\frac{\partial^4 F}{\partial b_{cd}\partial a_{ef}\partial a_{op}\partial a_{st}} = 0 \tag{B.22}$$

$$\frac{\partial^4 F}{\partial a_{ab}\partial b_{gh}\partial a_{op}\partial a_{st}} = 0 \tag{B.23}$$

$$\frac{\partial^4 F}{\partial b_{cd}\partial b_{gh}\partial a_{op}\partial a_{st}} = 2w_{oc}^2\delta_{eg}\delta_{os}\left(\delta_{dp}\delta_{ht} + \delta_{hp}\delta_{dt}\right) \tag{B.24}$$

$$\frac{\partial^4 F}{\partial a_{ab}\partial a_{ef}\partial b_{qr}\partial a_{st}} = 0 \tag{B.25}$$

$$\frac{\partial^4 F}{\partial b_{cd}\partial a_{ef}\partial b_{qr}\partial a_{st}} = 2w_{ec}^2\delta_{cq}\delta_{es}\left(\delta_{df}\delta_{rt} + \delta_{fr}\delta_{dt}\right) \tag{B.26}$$

$$\frac{\partial^4 F}{\partial a_{ab}\partial b_{gh}\partial b_{qr}\partial a_{st}} = 2w_{ag}^2\delta_{gq}\delta_{as}\left(\delta_{bh}\delta_{rt} + \delta_{br}\delta_{ht}\right) \tag{B.27}$$

$$\frac{\partial^4 F}{\partial b_{cd}\partial b_{gh}\partial b_{qr}\partial a_{st}} = 0 \tag{B.28}$$

$$\frac{\partial^4 F}{\partial a_{ab}\partial a_{ef}\partial a_{op}\partial b_{uv}} = 0 \tag{B.29}$$

$$\frac{\partial^4 F}{\partial b_{cd}\partial a_{ef}\partial a_{op}\partial b_{uv}} = 2w_{ec}^2\delta_{eo}\delta_{cu}\left(\delta_{df}\delta_{pv} + \delta_{dp}\delta_{fv}\right) \tag{B.30}$$

$$\frac{\partial^4 F}{\partial a_{ab}\partial b_{gh}\partial a_{op}\partial b_{uv}} = 2w_{ag}^2\delta_{ao}\delta_{gu}\left(\delta_{bh}\delta_{pv} + \delta_{hp}\delta_{bv}\right) \tag{B.31}$$

$$\frac{\partial^4 F}{\partial b_{cd}\partial b_{gh}\partial a_{op}\partial b_{uv}} = 0 \tag{B.32}$$

$$\frac{\partial^4 F}{\partial a_{ab}\partial a_{ef}\partial b_{qr}\partial b_{uv}} = 2w_{aq}^2\delta_{ae}\delta_{qu}\left(\delta_{br}\delta_{fv} + \delta_{fr}\delta_{bv}\right) \tag{B.33}$$

$$\frac{\partial^4 F}{\partial b_{cd}\partial a_{ef}\partial b_{qr}\partial b_{uv}} = 0 \tag{B.34}$$

$$\frac{\partial^4 F}{\partial a_{ab}\partial b_{gh}\partial b_{qr}\partial b_{uv}} = 0 \tag{B.35}$$

$$\frac{\partial^4 F}{\partial b_{cd}\partial b_{gh}\partial b_{qr}\partial b_{uv}} = 0 \tag{B.36}$$

Particularly note that the fourth derivatives are functions of the weights only.

Appendix C

Bayesian Tracking

In Chapter 4 a Bayesian formulation for feature tracking is given. There, the posterior is conditioned on the observations of the trajectory up to the current frame. In practice the observations are made from images. The terminology of Chapter 4 makes no commitment as to the way the observed images are used. Theoretically, the images themselves should be part of the probabilistic formulation so that the way in which position coordinate observations are made can be derived directly.

Before going any further, here are a couple of notes on the notation that will be used in this section. Images, while still being written in matrix notation, $\mathtt{I}$, are treated mathematically as vector entities, i.e.

$$\mathtt{I}_t \leftarrow \mathtt{I}_t(:) \quad \therefore \quad \mathtt{I}_t^\top \mathtt{I}_t = scalar. \tag{C.1}$$

Therefore, for completeness, there needs to be a function that converts coordinate pairs, $\mathbf{x}$, that reference a 2D location in the image, to the linear index of the same location in the vectorized form: $i(\mathbf{x})$, thus $\mathtt{I}_{i(\mathbf{x})}$. However, this is abbreviated to $\mathtt{I}_\mathbf{x}$ without interference with the rest of the notation. This will also sometimes be used for patches, with the extra assumption that the central pixel of a patch is referenced by the coordinate $(0, 0)$, making possible

$\mathbf{p}_{\mathbf{x}-\mathbf{x}_t}$ when $\mathbf{x}$ and $\mathbf{x}_t$ are in 'image coordinates'. Also note that, because $\mathbf{x}$ is so synonymous with coordinates, it will be used for two different coordinate entities in this chapter. Without a subscript, it will be a temporary generic coordinate pair, used, for example, in summations or local scope function definitions. With a subscript, i.e. $\mathbf{x}_t$, it is the important latent variable of the feature track as defined in Chapter 2.

In Chapter 2, the importance of the feature's appearance as part of the feature track's frame-to-frame state was made explicit in the definition of a track. Figure C.1 displays the graphical model that is implied. The Bayesian tracking formulation is then updated to:

$$\begin{aligned} \mathrm{p}(\mathbf{x}_t \,|\, \mathtt{I}_{1..t}, \mathbf{p}_{1..t-1}) &\propto \mathrm{p}(\mathtt{I}_t \,|\, \mathbf{x}_t, \mathbf{p}_{1..t-1}, \mathtt{I}_{1..t-1}) \mathrm{p}(\mathbf{x}_t \,|\, \mathbf{p}_{1..t-1}, \mathtt{I}_{1..t-1}) && \text{(C.2)} \\ &\propto \mathrm{p}(\mathtt{I}_t \,|\, \mathbf{x}_t, \mathbf{p}_{t-1}) \int \mathrm{p}(\mathbf{x}_t \,|\, \mathbf{x}_{t-1}) \mathrm{p}(\mathbf{x}_{t-1} \,|\, \mathtt{I}_{1..t-1}) d\mathbf{x}_{t-1}. && \text{(C.3)} \end{aligned}$$

As mentioned in Chapter 4, I am going to overlook the debate on modeling appearance and image generation and employ the simplest of generative models for the observed image[1]:

$$\begin{aligned} \mathrm{p}(\mathtt{I}_t \,|\, \mathbf{x}_t, \mathbf{p}_t) &= \mathcal{N}(\mathtt{I}_t | \mu_{\mathtt{I}_t}, \Sigma_{\mathtt{I}_t}) && \text{(C.4)} \\ \left[\mu_{\mathtt{I}_t}\right]_{\mathbf{x}} &= \begin{cases} 0 & \mathbf{x} \notin \mathcal{W}_{\mathbf{x}_t} \\ \left[\mathbf{p}_t\right]_{\mathbf{x}-\mathbf{x}_t} & \mathbf{x} \in \mathcal{W}_{\mathbf{x}_t} \end{cases} && \text{(C.5)} \\ \left[\mathrm{diag}(\Sigma_{\mathtt{I}_t})\right]_{\mathbf{x}} &= \begin{cases} \infty & \mathbf{x} \notin \mathcal{W}_{\mathbf{x}_t} \\ \sigma_{\mathtt{I}} & \mathbf{x} \in \mathcal{W}_{\mathbf{x}_t} \end{cases} && \text{(C.6)} \end{aligned}$$

and the evolution of the appearance:

$$\begin{aligned} \mathrm{p}(\mathbf{p}_t \,|\, \mathbf{p}_{t-1}) &= \mathcal{N}(\mathbf{p}_t | \mu_{\mathbf{p}_t}, \Sigma_{\mathbf{p}_t}) && \text{(C.7)} \\ \mu_{\mathbf{p}_t} &= \mathbf{p}_{t-1} && \text{(C.8)} \\ \Sigma_{\mathbf{p}_t} &= \sigma_{\mathbf{p}} \mathtt{I} && \text{(C.9)} \end{aligned}$$

[1]This simple generative model for the image observations justifies, to some extent, the assumption in the independence of the pre-computed motions.

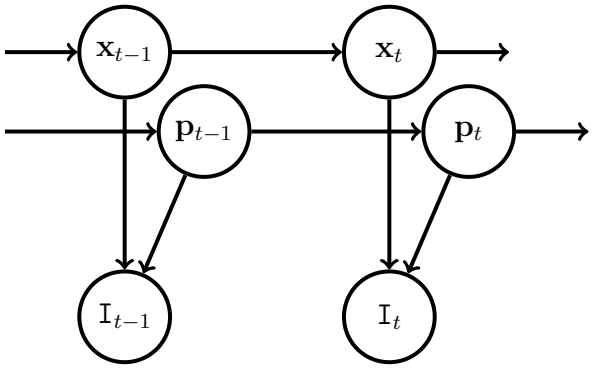


Figure C.1: **Bayesian tracking.** The graphical model for the tracking process. Observed image variables, $\mathtt{I}_t$, are generated by two hidden variables: the feature location, $\mathbf{x}_t$, and the feature appearance, $\mathbf{p}_t$.

There is considerable research on how to maintain and update the appearance density, from the weighted temporal sum of patches (Jepson et al. 2003) to Kalman filters on every pixel (Nguyen and Smeulders 2004). Going ahead with the simple appearance model above, the likelihood can be derived. From now on, $\mathbf{I} = [\mathtt{I}_t]_{\mathcal{W}_{\mathbf{x}_t}}$ is the window of pixels extract from $\mathtt{I}_t$ around $\mathbf{x}_t$.

$$p(\mathtt{I}_t|\mathbf{x}_t, \mathbf{p}_{t-1}) = \int p(\mathtt{I}_t|\mathbf{x}_t, \mathbf{p}_t) p(\mathbf{p}_t|\mathbf{p}_{t-1}) d\mathbf{p}_t \tag{C.10}$$

$$= k \int \exp\left(-P\right) d\mathbf{p}_t \tag{C.11}$$

Now

$$P = \frac{\| [\mathtt{I}_t]_{\mathcal{W}_{\mathbf{x}_t}} - \mathbf{p}_t \|_2^2}{2\sigma_{\mathtt{I}}^2} - \frac{\|\mathbf{p}_t - \mathbf{p}_{t-1}\|_2^2}{2\sigma_{\mathbf{p}}^2} \tag{C.12}$$

$$= \frac{\mathbf{I}^\top\mathbf{I} - 2\mathbf{I}^\top\mathbf{p}_t + \mathbf{p}_t^\top\mathbf{p}_t}{2\sigma_{\mathtt{I}}^2} + \frac{\mathbf{p}_t^\top\mathbf{p}_t - 2\mathbf{p}_{t-1}^\top\mathbf{p}_t + \mathbf{p}_{t-1}^\top\mathbf{p}_{t-1}}{2\sigma_{\mathbf{p}}^2} \tag{C.13}$$

$$= \frac{\mathbf{I}^\top\mathbf{I}\sigma_{\mathbf{p}}^2 + \mathbf{p}_{t-1}^\top\mathbf{p}_{t-1}\sigma_{\mathtt{I}}^2}{2\sigma_{\mathtt{I}}^2\sigma_{\mathbf{p}}^2} - \frac{\mathbf{I}^\top\sigma_{\mathbf{p}}^2 + \mathbf{p}_{t-1}^\top\sigma_{\mathtt{I}}^2}{2\sigma_{\mathbf{p}}^2\sigma_{\mathtt{I}}^2}\mathbf{p}_t + \frac{\sigma_{\mathbf{p}}^2 + \sigma_{\mathtt{I}}^2}{2\sigma_{\mathbf{p}}^2\sigma_{\mathtt{I}}^2}\mathbf{p}_t^\top\mathbf{p}_t \tag{C.14}$$

$$= \frac{K}{2\sigma'^2} - \frac{2L^\top}{2\sigma'^2}\mathbf{p}_t + \frac{1}{2\sigma'^2}\mathbf{p}_t^\top\mathbf{p}_t \tag{C.15}$$

with

$$K = \frac{\mathbf{I}^\top \mathbf{I}\sigma_\mathbf{p}^2 + \mathbf{p}_{t-1}^\top \mathbf{p}_{t-1}\sigma_\mathtt{I}^2}{\sigma_\mathtt{I}^2 + \sigma_\mathbf{p}^2} \tag{C.16}$$

$$L^\top = \frac{\mathbf{I}^\top \sigma_\mathbf{p}^2 + \mathbf{p}_{t-1}^\top \sigma_\mathtt{I}^2}{\sigma_\mathtt{I}^2 + \sigma_\mathbf{p}^2} \tag{C.17}$$

$$\sigma' = \sqrt{\frac{\sigma_\mathbf{p}^2 \sigma_\mathtt{I}^2}{\sigma_\mathbf{p}^2 + \sigma_\mathtt{I}^2}} \tag{C.18}$$

$$k = \frac{1}{\left(2\pi\sigma_\mathtt{I}^2\right)^{\frac{n}{2}}} \frac{1}{\left(2\pi\sigma_\mathbf{p}^2\right)^{\frac{n}{2}}} \tag{C.19}$$

therefore

$$L^\top L = \frac{\mathbf{I}^\top \mathbf{I}\sigma_\mathbf{p}^4 + 2\mathbf{p}_{t-1}^\top \mathbf{I}\sigma_\mathbf{p}^2\sigma_\mathtt{I}^2 + \mathbf{p}_{t-1}^\top \mathbf{p}_{t-1}\sigma_\mathtt{I}^4}{\left(\sigma_\mathbf{p}^2 + \sigma_\mathtt{I}^2\right)^2} \tag{C.20}$$

and so, for $M = L^\top L - K$, we have

$$M = \frac{\mathbf{I}^\top \mathbf{I}\sigma_\mathbf{p}^2}{\sigma_\mathbf{p}^2 + \sigma_\mathtt{I}^2}\left(\frac{\sigma_\mathbf{p}^2}{\sigma_\mathbf{p}^2 + \sigma_\mathtt{I}^2} - 1\right) + \frac{2\mathbf{p}_{t-1}^\top \mathbf{I}\sigma_\mathtt{I}^2\sigma_\mathbf{p}^2}{\left(\sigma_\mathbf{p}^2 + \sigma_\mathtt{I}^2\right)^2} + \frac{\mathbf{p}_{t-1}^\top \mathbf{p}_{t-1}\sigma_\mathtt{I}^2}{\sigma_\mathbf{p}^2 + \sigma_\mathtt{I}^2}\left(\frac{\sigma_\mathtt{I}^2}{\sigma_\mathbf{p}^2 + \sigma_\mathtt{I}^2} - 1\right) \tag{C.21}$$

which means

$$p(\mathtt{I}_t|x_t, \mathbf{p}_{t-1}) = k\int \exp\left(-\left[\frac{L^\top L - M}{2\sigma'^2} - \frac{2L^\top}{2\sigma'^2}\mathbf{p}_t + \frac{1}{2\sigma'^2}\mathbf{p}_t^\top \mathbf{p}_t\right]\right) d\mathbf{p}_t \tag{C.22}$$

$$= \exp\left(\frac{M}{2\sigma'^2}\right) k \int \exp\left(-\frac{\|L - \mathbf{p}_t\|}{2\sigma'^2}\right) d\mathbf{p}_t \tag{C.23}$$

$$= \exp\left(\frac{M}{2\sigma'^2}\right) k \left(2\pi\sigma'^2\right)^{\frac{n}{2}} \tag{C.24}$$

and finally, with $S = \left(\sigma_\mathbf{p}^2 + \sigma_\mathtt{I}^2\right)^2$,

$$M = -\left[\frac{\mathbf{I}^\top \mathbf{I}\sigma_\mathbf{p}^2}{S}\left(\left(\sigma_\mathbf{p}^2 + \sigma_\mathtt{I}^2\right) - \sigma_\mathbf{p}^2\right) - \frac{2\mathbf{p}_{t-1}^\top \mathbf{I}\sigma_\mathtt{I}^2\sigma_\mathbf{p}^2}{S} + \frac{\mathbf{p}_{t-1}^\top \mathbf{p}_{t-1}\sigma_\mathtt{I}^2}{S}\left(\left(\sigma_\mathbf{p}^2 + \sigma_\mathtt{I}^2\right) - \sigma_\mathtt{I}^2\right)\right] \tag{C.25}$$

which simplifies to

$$M = -\left[\frac{\mathbf{I}^\top \mathbf{I}\sigma_{\mathbf{p}}^2\sigma_{\mathtt{I}}^2}{S} - \frac{2\mathbf{p}_{t-1}^\top \mathbf{I}\sigma_{\mathtt{I}}^2\sigma_{\mathbf{p}}^2}{S} + \frac{\mathbf{p}_{t-1}^\top \mathbf{p}_{t-1}\sigma_{\mathtt{I}}^2\sigma_{\mathbf{p}}^2}{S}\right] \tag{C.26}$$

$$= -\frac{\sigma_{\mathbf{p}}^2\sigma_{\mathtt{I}}^2}{(\sigma_{\mathbf{p}}^2+\sigma_{\mathtt{I}}^2)^2}\,\|\mathbf{I}-\mathbf{p}_{t-1}\| \tag{C.27}$$

$$= -\frac{\sigma'^2}{\sigma_{\mathbf{p}}^2+\sigma_{\mathtt{I}}^2}\,\|\mathbf{I}-\mathbf{p}_{t-1}\| \tag{C.28}$$

revealing

$$p(\mathtt{I}_t|\mathbf{x}_t,\mathbf{p}_{t-1}) = k\left(2\pi\sigma'^2\right)^{\frac{n}{2}}\exp\left(\frac{M}{2\sigma'^2}\right) \tag{C.29}$$

$$= \frac{1}{\left(2\pi\left(\sigma_{\mathbf{p}}^2+\sigma_{\mathtt{I}}^2\right)\right)^{\frac{n}{2}}}\exp\left(-\frac{\|\mathbf{I}-\mathbf{p}_{t-1}\|}{2(\sigma_{\mathbf{p}}^2+\sigma_{\mathtt{I}}^2)}\right) \tag{C.30}$$

$$= \mathcal{N}\left([\mathtt{I}_t]_{\mathcal{W}_{\mathbf{x}_t}}\,|\,\mathbf{p}_{t-1},\mathtt{I}(\sigma_{\mathbf{p}}^2+\sigma_{\mathtt{I}}^2)\right) \tag{C.31}$$

i.e. an SSD of the previous patch with the current frame. By taking the maximum likelihood estimate of appearance, $\mathbf{p}_{t-1}\leftarrow\mathtt{I}_{\mathcal{W}_{\mathbf{x}_{t-1}}}$, we have the track-to-previous algorithm. It is now easy to see why track-to-previous is so susceptible to the problems of drift and repeated texture. This is why the tracker used in Chapter 4 included a version of the template update scheme of Matthews et al. (2003).

Evaluation of the Diffused Prior

Here, we can go through the derivation of Equation (4.14), i.e. the evaluation of the integral in (4.13) and the determination of the diffused prior, $q(\mathbf{x}_t)$. The first steps are manipulations of the integrand in (4.13), which takes the form $\exp(-\gamma E(\mathbf{x},\mathbf{u}))$. For brevity, $\mathbf{x}_t$ will be written simply as '$\underline{\mathbf{x}}$' and $\mathbf{z}_{t-\mathbf{M}}$ will be represented by '$\underline{\mathbf{z}}$'. We start by defining

$$A = \gamma^{-1}\Sigma_{t-\mathbf{M}}^{-1}, \tag{C.32}$$

so we can write

$$E = \|\underline{\mathbf{x}} - \mathtt{P}\mathbf{u}\|^2 + (\mathbf{u} - \underline{\mathbf{z}})^\top A(\mathbf{u} - \underline{\mathbf{z}}) \tag{C.33}$$

$$= \mathbf{u}^\top \mathtt{P}^\top \mathtt{P}\mathbf{u} - 2\underline{\mathbf{x}}^\top \mathtt{P}\mathbf{u} + \underline{\mathbf{x}}^\top \underline{\mathbf{x}} + \mathbf{u}^\top A\mathbf{u} - 2\underline{\mathbf{z}}^\top A\underline{\mathbf{z}} + \underline{\mathbf{z}}^\top \underline{\mathbf{z}} \tag{C.34}$$

$$= \mathbf{u}^\top (\mathtt{P}^\top \mathtt{P} + A)\mathbf{u} - 2\mathbf{u}^\top (\mathtt{P}^\top \underline{\mathbf{x}} + A^\top \underline{\mathbf{z}}) + \underline{\mathbf{x}}^\top \underline{\mathbf{x}} + \underline{\mathbf{z}}^\top \underline{\mathbf{z}}. \tag{C.35}$$

Now, with $C = (\mathtt{P}^\top \mathtt{P} + A)$ and $c = C^{-1}(\mathtt{P}^\top \underline{\mathbf{x}} + A^\top \underline{\mathbf{z}})$:

$$E = (\mathbf{u} - c)^\top C(\mathbf{u} - c) - c^\top C\, c + \underline{\mathbf{x}}^\top \underline{\mathbf{x}} + \underline{\mathbf{z}}^\top \underline{\mathbf{z}}, \tag{C.36}$$

allowing the diffused prior to be seen as

$$q(\underline{\mathbf{x}}) \propto \int e^{-\gamma g(\mathbf{u})} e^{-\gamma f(\underline{\mathbf{x}})} e^{-\gamma \underline{\mathbf{z}}^\top \underline{\mathbf{z}}} \mathrm{d}\mathbf{u} \tag{C.37}$$

i.e. $q(\underline{\mathbf{x}}) \propto e^{-\gamma f(\underline{\mathbf{x}})}$, since the $\underline{\mathbf{x}}$s in $g(\mathbf{u})$ are contained only in the 'mean' term (disappearing on integration). Examining $f(\underline{\mathbf{x}})$ (remembering that C is symmetric):

$$f(\underline{\mathbf{x}}) = \underline{\mathbf{x}}^\top \underline{\mathbf{x}} - c^\top C\, c \tag{C.38}$$

$$= \underline{\mathbf{x}}^\top \underline{\mathbf{x}} - (\mathtt{P}^\top \underline{\mathbf{x}} + A^\top \underline{\mathbf{z}})^\top C^{-1} C\, C^{-1} (\mathtt{P}^\top \underline{\mathbf{x}} + A^\top \underline{\mathbf{z}}) \tag{C.39}$$

$$= \underline{\mathbf{x}}^\top \underline{\mathbf{x}} - (\underline{\mathbf{x}}^\top \mathtt{P} + \underline{\mathbf{z}}^\top A)(\mathtt{P}^\top \mathtt{P} + A)^{-1} (\mathtt{P}^\top \underline{\mathbf{x}} + A^\top \underline{\mathbf{z}}) \tag{C.40}$$

$$= \underline{\mathbf{x}}^\top (\mathtt{I} - \mathtt{P}(\mathtt{P}^\top \mathtt{P} + A)^{-1} \mathtt{P}^\top)\underline{\mathbf{x}} + \\ -2\underline{\mathbf{x}}^\top \mathtt{P}(\mathtt{P}^\top \mathtt{P} + A)^{-1} A^\top \underline{\mathbf{z}} + \ldots \tag{C.41}$$

The next stage follows from equating $e^{-\gamma f(\underline{\mathbf{x}})}$, and hence $q(\underline{\mathbf{x}})$, with a Gaussian,

$$\exp(-(\underline{\mathbf{x}} - \mu)^\top \Sigma^{-1} (\underline{\mathbf{x}} - \mu)) \tag{C.42}$$

and from there determining the Gaussian's parameters:

$$\gamma f(\underline{\mathbf{x}}) \equiv \underline{\mathbf{x}}^\top \Sigma^{-1} \underline{\mathbf{x}} - 2\underline{\mathbf{x}}^\top \Sigma^{-1} \mu + \ldots \tag{C.43}$$

$$\therefore \quad \Sigma^{-1} = \gamma(\mathtt{I} - \mathtt{P}(\mathtt{P}^\top \mathtt{P} + A)^{-1}\mathtt{P}^\top) \tag{C.44}$$

$$= \gamma(\mathtt{I} + \mathtt{P}A^{-1}\mathtt{P}^\top)^{-1} \tag{C.45}$$

$$= (\gamma^{-1}\mathtt{I} + \mathtt{P}\Sigma_{t-\mathbf{M}}\mathtt{P}^\top)^{-1} \tag{C.46}$$

$$\text{and} \quad \Sigma^{-1}\mu = \gamma \mathtt{P}(\mathtt{P}^\top \mathtt{P} + A)^{-1} A^\top \underline{\mathbf{z}} \tag{C.47}$$

$$= \gamma(\mathtt{I} - \mathtt{P}(\mathtt{P}^\top \mathtt{P} + A)^{-1}\mathtt{P}^\top)\mathtt{P}\underline{\mathbf{z}} \tag{C.48}$$

$$= \Sigma^{-1}\mathtt{P}\underline{\mathbf{z}} \tag{C.49}$$

where (C.45) is an application of the Sherman–Morrison–Woodbury identity:

$$(\mathtt{K} + \mathtt{UAV})^{-1} = \mathtt{K}^{-1} - \mathtt{K}^{-1}\mathtt{U}(\mathtt{A}^{-1} + \mathtt{V}\mathtt{K}^{-1}\mathtt{U})^{-1}\mathtt{V}\mathtt{K}^{-1} \tag{C.50}$$

with $\mathtt{K} = \mathtt{I}$, $\mathtt{U} = \mathtt{P}$, $\mathtt{A} = A^{-1}$ and $\mathtt{V} = \mathtt{P}^\top$, plus (C.48) employs

$$A = A^\top \tag{C.51}$$

$$(B + A)^{-1} A = \mathtt{I} - (B + A)^{-1} B \tag{C.52}$$

and the Newton's Cradle identity:

$$\mathtt{V}(\mathtt{I} + \mathtt{UV}) = (\mathtt{I} + \mathtt{VU})\mathtt{V} \tag{C.53}$$

Equation (4.14) follows.

Bibliography

2d3. *Boujou*, 2007. URL `www.2d3.com`.

H. Aanæs, R. Fisker, K. Åström, and J. M. Carstensen. Robust factorization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(9):1215–1225, 2002.

P. M. Q. Aguiar and J. M. F. Moura. Weighted factorization. In *Proceedings, IEEE International Conference on Image Processing*, volume 1, pages 549–552, 2000.

P. M. Q. Aguiar and J. M. F. Moura. Rank 1 weighted factorization for 3D structure recovery: Algorithms and performance analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(9):1049–1134, 2003.

P. M. Q. Aguiar and J. M. F. Moura. Maximum likelihood inference of 3D structure from image sequences. In *Lecture Notes in Computer Science, International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition*, pages 285–300, 1999.

S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for on-line non-linear/non-Gaussian Bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2):174–188, 2002.

S. Arya and D. M. Mount. Algorithms for fast vector quantization. In *Proceedings, IEEE Data Compression Conference*, pages 381–390, 1993.

S. Avidan. Subset selection for efficient SVM tracking. In *Proceedings, IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages 85–94, 2003.

P. Belhumeur and D. Kriegman. What is the set of images of an object under all possible lighting conditions? In *Proceedings, IEEE Conference on Computer Vision and Pattern Recognition*, pages 270–277, 1996.

R. Bellman. On the theory of dynamic programming. In *Proceedings, National Academy of Sciences*, 1952.

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.

C. Bibby and I. Reid. Visual tracking at sea. In *Proceedings, International Conference on Robotics and Automation*, 2005.

M. Brand. Morphable 3d models from video. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 456–463, 2001.

M. Brand. Incremental singular value decomposition of uncertain data with missing values. In *Proceedings, European Conference on Computer Vision*, pages 707–720, 2002.

S. Brandt. Closed-form solutions for affine reconstruction under missing data. In *Proceedings, Statistical Methods for Video Processing Workshop, European Conference on Computer Vision*, pages 109–114, 2002.

C. Bregler, A. Hertzmann, and H. Biermann. Recovering non-rigid 3D shape from image streams. In *Proceedings, IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 690–696, 2000.

J.-Y. Chen, C. Bouman, and J. Allenbach. Fast image database search using Tree-Structure VQ. In *Proceedings, The IEEE International Conference on Image Processing*, pages 827–830, 1997.

P. Chen and D. Suter. Recovering the missing components in a large noisy low-rank matrix: Application to SFM. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(8):1051–1063, 2004.

Y. Chen, T. Huang, and Y. Rui. Optimal radial contour tracking by dynamic programming. In *Proceedings, IEEE International Conference on Image Processing*, 2001.

R. T. Collins. Mean-shift blob tracking through scale space. In *Proceedings, IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 234–240, 2003.

D. Comaniciu, V. Ramesh, and P. Meer. Kernel-based object tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(5):564–577, 2003.

I. J. Cox and S. L. Hingorani. An efficient implementation and evaluation of reid's multiple hypothesis tracking algorithm for visual tracking. In *Proceedings, International Conference on Pattern Recognition*, pages 437–442, 1994. URL `citeseer.ist.psu.edu/cox94efficient.html`.

D. L. Donoho and C. Grimes. Image manifolds which are isometric to euclidean space. *Journal of Mathematical Imaging and Vision*, 23(1):5–24, 2005. ISSN 0924-9907.

P. Dreuw, T. Deselaers, D. Rybach, D. Keysers, and H. Ney. Tracking using dynamic programming for appearance-based sign language recognition. In *Proceedings, IEEE International Conference on Automatic Face and Gesture Recognition*, pages 293–298, 2006.

UK Film Council. Statistical yearbook 2006/07, 2007. URL `www.ukfilmcouncil.org.uk/information/statistics/yearbook/?y=2006&c=1`.

A. W. Fitzgibbon and A. Zisserman. Automatic camera recovery for closed or open image sequences. In *Proceedings, European Conference on Computer Vision*, pages 311–326. Springer-Verlag, 1998.

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.

G. H. Golub and C. F. Van Loan. *Matrix Computations, Third Edition*. The Johns Hopkins University Press, 1996.

D. Greene. An implementation and performance analysis of spatial data access methods. In *Proceedings, International Conference on Data Engineering*, pages 606–615, 1989.

A. Gruber and Y. Weiss. Factorization with uncertainty and missing data: Exploiting temporal coherence. In S. Thrun, L. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems*, volume 16. MIT Press, Cambridge, MA, 2004.

R. F. C. Guerreiro and P. M. Q. Aguiar. Factorization with missing data for 3D structure recovery. In *Proceedings, IEEE International Workshop on Multimedia Signal Processing*, pages 105–108, 2002a.

R. F. C. Guerreiro and P. M. Q. Aguiar. 3D structure from video streams with partially overlapping images. In *Proceedings, IEEE International Conference on Image Processing*, volume 3, pages 897–900, 2002b.

R. F. C. Guerreiro and P. M. Q. Aguiar. Estimation of rank deficient matrices from partial observations: Two-step iterative algorithms. In *Lectures Notes in Computer Science: Energy Minimization Methods in Computer Vision and Pattern Recognition*, volume 2683. Springer-Verlag, 2003.

N. Guilbert and A. Bartoli. Batch recovery of multiple views with missing data using direct sparse solvers. In *Proceedings, British Machine Vision Conference*, 2003.

O. Gunther. The cell tree: An index for geometric data. Technical Report Memorandum No. UCB/ERL M86/89, Univ. of California, Berkeley, 1986.

A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings, Association for Computing Machinery Special Interest Group on Managment of Data*, pages 47–57, 1984.

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.

C. Harris and M. Stephens. A combined corner and edge detection. In *Proceedings, The Fourth Alvey Vision Conference*, pages 147–151, 1988.

R. Hartley and F. Schaffalitzky. Powerfactorization: an approach to affine reconstruction with missing and uncertain data. In *Australia-Japan Advanced Workshop on Computer Vision*, 2003.

R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, second edition, 2004. ISBN 0521540518.

G. E. Hinton, P. Dayan, and M. Revow. Modelling the manifolds of images of handwritten digits. *IEEE Transactions on Neural Networks*, 8(1):65–74, 1997.

Y. Huang and I. Essa. Tracking multiple objects through occlusions. In *Proceedings, IEEE Conference on Computer Vision and Pattern Recognition*, 2005.

D. Q. Huynh, R. Hartley, and A. Heyden. Outlier correction in image sequences for the affine camera. In *Proceedings, IEEE International Conference on Computer Vision*, volume 1, pages 585–590, 2003.

IMDB. Internet movie database, 2007. URL `www.imdb.com/title/tt0243155/fullcredits#cast`.

M. Irani. Multi-frame optical flow estimation using subspace constraints. In *Proceedings, IEEE International Conference on Computer Vision*, 1999.

M. Irani and P. Anandan. Factorization with uncertainty. *International Journal of Computer Vision*, 49:101–116, 2002.

T. Ishikawa, I. Matthews, and S. Baker. Efficient image alignment with outlier rejection. Technical Report CMU-RI-TR-02-27, Carnegie Mellon University, 2002.

D. W. Jacobs. Linear fitting with missing data for structure-from-motion. *Computer Vision and Image Understanding*, 82(1):57–81, 2001.

H. Jagadish. Spatial search with polyhedra. In *Proceedings, International Conference on Data Engineering*, 1990.

M. Jamshidian and R. Jennrich. Acceleration of the EM algorithm by using quasi-newton methods. *Journal of the Royal Statistical Society*, 59(3):569–587, 1997.

A. Jepson, D. Fleet, and T. El-Maraghi. Robust online appearance models for visual tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(10), 2003.

D. Jones and J. Malik. A computational framework for determining stereo correspondence from a set of linear spatial filters. *Image and Vision Computing*, 10(10):699–708, 1992.

I. Kamel and C. Faloutsos. Hilbert r-tree: An improved r-tree using fractals. In *Proceedings, International Conference on Very Large Data Bases*, pages 500–509, 1994.

J. J. Koenderink and A. J. van Doom. Representation of local geometry in the visual system. *Biological Cybernetics*, 55(6):367–375, 1987.

R. Kurniawati, J. Jin, and J. Shepherd. The SS+-tree: An improved index structure for similarity searches in a high-dimensional feature space. In *Proceedings, Conference on Storage and Retrieval for Image and Video Databases*, volume 3022, pages 110–120, 1997.

K. Lange. A gradient algorithm locally equivalent to the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 57(2):425–437, 1995a.

L. Li. On-line visual tracking with feature-based adaptive models. Master's thesis, Univerisity of British Columbia, Vancover, Canada, 2004.

R. J. A. Little and D. B. Rubin. *Statistical Analysis with Missing Data, Second Edition*. Wiley, Hoboken, New Jersey, 2002. ISBN 0-471-18386-5.

Z. Liu and Y. Wang. Face detection and tracking in video using dynamic programming. In *Proceedings, IEEE International Conference on Image Processing*, 2000.

X. Llado, A. Del Bue, and L. Agapito. Non-rigid 3D factorization for projective reconstruction. In *Proceedings, British Machine Vision Conference*, 2005.

D. Lomet and B. Salzberg. The hb-tree: A multiattribute indexing method with good guaranteed performance. *Association for Computing Machinery Transactions on Database Systems*, 15(4): 625–658, 1990.

D. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

B. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *International Joint Conferences on Artificial Intelligence*, pages 674–679, 1981.

R. Mann, A. Jepson, and T. El-Maraghi. Trajectory segmentation using dynamic programming. In *Proceedings, International Conference on Pattern Recognition*, 2002.

D. Martinec and T. Pajdla. Structure from many perspective images with occlusions. In *Proceedings, European Conference on Computer Vision*, pages 355–369, 2002.

D. Martinec and T. Pajdla. Automatic factorization-based reconstruction from perspective images with occlusions and outliers. In *Proceedings, Computer Vision Winter Workshop*, pages 147–152, 2003.

D. Martinec and T. Pajdla. 3D reconstruction by fitting low-rank matrices with missing data. In *Proceedings, IEEE Conference on Computer Vision and Pattern Recognition*, 2005.

I. Matthews, T. Ishikawa, and S. Baker. The template update problem. In *Proceedings, British Machine Vision Conference*, 2003.

W. W. Mayol, A. J. Davison, B. J. Tordoff, N. D. Molton, and D. W. Murray. Interaction between hand and wearable camera in 2D and 3D environments. In *Proceedings, British Machine Vision Conference*, 2004.

J. Meltzer, M.-H. Yang, R. Gupta, and S. Soatto. Multiple view feature descriptors from image sequences via kernel principal component analysis. In *Proceedings, European Conference on Computer Vision*, volume 1, pages 215–227, 2004.

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

H. T. Nguyen and A. W. M. Smeulders. Fast occluded object tracking by a robust appearance filter. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(8):1099–1104, 2004.

M. Pardas and E. Sayrol. A new approach to tracking with active contours. In *Proceedings, IEEE International Conference on Image Processing*, 2000.

N. S. Peng, J. Yang, and Z. Liu. Mean shift blob tracking with kernel histogram filtering and hypothesis testing. *Pattern Recognition Letters*, 26(5):605–614, 2005.

O. Procopiuc, P. Agarwal, L. Arge, and J. Vitter. Bkd-tree: A dynamic scalable kd-tree, 2002. URL `citeseer.ist.psu.edu/procopiuc02bkdtree.html`.

G. Quenot. The 'orthogonal algorithm' for optical flow detection using dynamic programming. In *Proceedings, International Conference on Accoustics, Speech and Signal Processing*, 1992.

D. B. Reid. An algorithm for tracking multiple targets. *IEEE Transactions on Automatic Control*, 24 (6):843–854, 1979.

I. D. Reid and D. W. Murray. Active tracking of foveated feature clusters using affine structure. *International Journal of Computer Vision*, 18(1):41–60, 1996.

J. Robinson. The k-d-b-tree: a search structure for large multidimensional dnamic indexes. In *Proceedings, Association for Computing Machinery Special Interest Group on Managment of Data*, pages 10–18, 1981.

C. Rother and S. Carlsson. Linear multi view reconstruction and camera recovery. In *Proceedings, IEEE International Conference on Computer Vision*, volume 1, pages 42–50, 2001.

S. Roweis. EM algorithms for PCA and SPCA. In *Proceedings, Neural Information Processing Systems*, volume 10, pages 626–632, 1997.

H. Saito and S. Kamijima. Factorization method using interpolated feature tracking via projective geometry. In *Proceedings, British Machine Vision Conference*, volume 2, pages 449–458, 2003.

P. Sand and S. Teller. Particle video: Long-range motion estimation using point trajectories. In *Proceedings, IEEE Conference on Computer Vision and Pattern Recognition*, 2006.

F. Schaffalitzky, A. Zisserman, R. I. Hartley, and P. H. S. Torr. A six point solution for structure and motion. In *Proceedings, European Conference on Computer Vision*, volume 1, pages 632–648, 2000. URL `citeseer.ist.psu.edu/article/schaffalitzky00six.html`.

C. Schmid and R. Mohr. Local grayvalue invariants for image retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(5):530–535, 1997.

T. Sellis, N. Roussopoulos, and C. Floutsos. The R+ tree: A dynamic index for multi-dimensional objects. In *Proceedings, International Conference on Very Large Data Bases*, pages 507–518, 1987.

E. Shay. Sterling allen on a scanner darkly. *Cinefex*, 107, 2006.

J. Shi and C. Tomasi. Good features to track. In *Proceedings, IEEE Conference on Computer Vision and Pattern Recognition*, pages 593–600, 1994.

H. Shum, K. Ikeuchi, and R. Reddy. Principal component analysis with missing data and its application to polyhedral object modeling. *Transactions on Pattern Analysis and Machine Intelligence*, 17(9):855–867, 1995.

Eero P. Simoncelli and William T. Freeman. The steerable pyramid: A flexible architecture for multi-scale derivative computation. In *International Conference on Image Processing*, volume 3, pages 444–447, 1995.

J. Sivic and A. Zisserman. Video Google: A text retrieval approach to object matching in videos. In *Proceedings, IEEE International Conference on Computer Vision*, 2003.

D. Skocaj and A. Leonardis. Weighted incremental subspace learning. In *Proceedings, Workshop on Cognitive Vision*, 2002.

P. Smith, D. Sinclair, R. Cipolla, and K. Wood. Effective corner matching. In *Proceedings, British Machine Vision Conference*, 1998.

C. Tomasi and T. Kanade. Detection and tracking of point features. Technical report, Carnegie Mellon University Technical Report CMU-CS-91-132, 1991.

C. Tomasi and T. Kanade. Selecting and tracking features for image sequence analysis. *Robotics and Automation*, 1992a.

C. Tomasi and T. Kanade. Shape and motion from image streams under orthography: a factorization method. *International Journal of Computer Vision*, 9(2):137–154, 1992b.

D. Torre and M. Black. Robust principal component analysis for computer vision. In *Proceedings, IEEE International Conference on Computer Vision*, volume 1, pages 362–369, 2001.

L. Torresani and A. Hertzmann. Automatic non-rigid 3D modeling from video. In *Proceedings, European Conference on Computer Vision*, pages 299–312, 2004.

L. Torresani, D. B. Yang, E. J. Alexander, and C. Bregler. Tracking and modeling non-rigid objects with rank constraints. In *Proceedings, IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages 493–500, 2001.

L. Torresani, A. Hertzmann, and C. Bregler. Learning non-rigid 3D shape from 2D motion. In S. Thrun, L. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems*, volume 16. MIT Press, Cambridge, MA, 2004.

B. Triggs. Factorization methods for projective structure and motion. In *Proceedings, IEEE Conference on Computer Vision and Pattern Recognition*, pages 845–851, 1996.

B. Triggs, P. McLauchlan, R. Hartley, and A. Fitzgibbon. Bundle adjustment – A modern synthesis. In W. Triggs, A. Zisserman, and R. Szeliski, editors, *Vision Algorithms: Theory and Practice*, Lecture Notes in Computer Science, pages 298–375. Springer Verlag, 2000.

M. Varma and A. Zisserman. Texture classification: Are filter banks necessary? In *Proceedings, IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 691–698, 2003.

R. Vidal and R. Hartley. Motion segmentation with missing data using powerfactorization and GPCA. In *Proceedings, IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 310–316, 2004.

P. Viola and M. Jones. Robust real-time face detection. *International Journal of Computer Vision*, 57 (2):137–154, 2004.

T. Wiberg. Computation of principal components when data are missing. *Proceedings, Second Symposium of Computational Statistics*, pages 229–326, 1976.

O. Williams, A. Blake, and R. Cipolla. A sparse probabilistic learning algorithm for real-time tracking. In *Proceedings, IEEE International Conference on Computer Vision*, 2003.

C. Yang, R. Duraiswami, and L. Davis. Efficient mean-shift tracking via a new similarity measure. In *Proceedings, IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages 176–183, 2005.

Z. Yao and H. Li. Tracking a detected face with dynamic programming. In *Proceedings, IEEE Conference on Computer Vision and Pattern Recognition Workshop*, volume 5, page 63, 2004.

A. Yuille, D. Snow, R. Epstein, and P. Belhumeur. Determining generative models of objects under varying illumination: Shape and albedo from multiple images using SVD and integrability. *International Journal of Computer Vision*, 35(3):203–222, 1999.

Symbol	*Meaning*
t	video frame number
F	total number of frames
$\mathtt{I}_t$	image for frame t
$\mathcal{I}$	the set of all the images in the sequence, $\{\mathtt{I}_t \,\vert\, t = 1 \dots F\}$
$\mathtt{I}_{\mathcal{W}_\mathbf{x}}$	a window of pixels from image $\mathtt{I}_t$, centred around $\mathbf{x}$, i.e. an image patch
$\mathtt{I}$	the identity matrix
$\mathbf{x}$	image coordinates
$\mathbf{x}_t$	feature location in frame t
$\mathcal{X}$	the set of all feature locations, $\{\mathbf{x}_t \,\vert\, t = 1 \dots F\}$ i.e. the track's trajectory
$\mathcal{W}_\mathbf{x}$	the set of all pixel coordinates within a fixed radius of $\mathbf{x}$
$\mathbf{p}_t$	the image patch representing the appearance of the feature in frame t
$\hat{\mathbf{p}}_t$	the vector descriptor for patch $\mathbf{p}_t$
$\mathcal{P}$	the set of all feature appearance descriptors, $\{\hat{\mathbf{p}}_t \,\vert\, t = 1 \dots F\}$
$\mathcal{T}$	all (location,appearance) pairs, $\{(\mathbf{x}_t, \hat{\mathbf{p}}_t) \,\vert\, t = 1 \dots F\}$, i.e. the track itself
$\mathbf{y}_i$	the i^{th} user-selected keyframe location
$\mathbf{q}_i$	the i^{th} user-selected keyframe appearance
$\mathcal{N}$	the set of frame numbers of user selected keyframes
$\mathcal{Q}$	all keyframe inputs, $\{(\mathbf{y}_t, \hat{\mathbf{q}}_t) \,\vert\, t = 1 \dots \vert\mathcal{N}\vert\}$, i.e. the patch library
l_t	candidate label selection variable for frame t
$\mathcal{L}$	the set of all labels, $\{l_t \,\vert\, t = 1 \dots F\}$
E	the track evaluation function
E_*	a track evaluation sub-function
$E_{*,t}$	a track evaluation sub-function integrand: $E_* = \lambda_* \sum_{t=1}^{F} E_{*,t}$

Figure 2: **Notation I.** Summary of notation used in Chapter 2.

Symbol	*Meaning*
$\hat{\mathtt{M}}$	the acquired *measurement matrix*. This is the only variable (other than $\mathtt{W}$) whose elements are known at all. It is modelled as a noisy version of $\mathtt{M}$
$\mathtt{W}$	the *weight matrix*. A matrix whose elements describe the confidence in the knowledge of the corresponding elements of $\hat{\mathtt{M}}$
$\mathtt{M}$	the rank r matrix to be factorized
$\mathtt{A}$	a column space for $\mathtt{M}$
$\mathtt{B}$	a row space for $\mathtt{M}$
m	the number of rows in $\mathtt{M}$, $\hat{\mathtt{M}}$, $\mathtt{W}$ and $\mathtt{A}$
n	the number of columns in $\mathtt{M}$, $\hat{\mathtt{M}}$ & $\mathtt{W}$ and the number of rows in $\mathtt{B}$
r	the rank of $\mathtt{M}$ and the number of columns in $\mathtt{A}$ and $\mathtt{B}$
m_{ij}	the element of $\mathtt{M}$ on the i^{th} row and j^{th} column
$\mathbf{m}^i$	the i^{th} row of $\mathtt{M}$ as a column vector
$\mathbf{m}_j$	the j^{th} column of $\mathtt{M}$
$\mathtt{M}_{(m\times n)}$	dimensions of vectors and matrices are given in subscripted parentheses
δ_{ij}	a Kronecker delta function equalling one when $i=j$ and zero otherwise
$\odot$	the Hadamard *element-wise* matrix product: $\mathtt{R}=\mathtt{P}\odot\mathtt{Q} \implies r_{ij}=p_{ij}q_{ij}$
$\mathtt{M}^+$	the pseudo-inverse of $\mathtt{M}$ allowing $\mathtt{M}^+\mathtt{M}=\mathtt{I}$ for $r=n<m$
$\mathtt{M}{\downarrow_r}$	*rank truncation*: the closest rank r matrix to $\mathtt{M}$ by the Frobenius norm
$\mathtt{M}(:)$	column-wise vectorization of $\mathtt{M}$: $\mathtt{M}(:)=(\mathbf{m}_1^\top\ \mathbf{m}_2^\top\ \dots\ \mathbf{m}_n^\top)^\top$
$\mathtt{M}(\cdot\cdot)$	row-wise vectorization of $\mathtt{M}$: $\mathtt{M}(\cdot\cdot)=(\mathbf{m}^{1\top}\ \mathbf{m}^{2\top}\ \dots\ \mathbf{m}^{m\top})^\top$

Figure 3: **Notation II.** Summary of notation used in Chapter 3.

Symbol	*Meaning*
t	sequence frame number
$\mathbf{x}_t$	hidden feature position in frame t
$\mathbf{z}_t$	observed feature position in frame t
M	size of temporal window for global motion model
$\mathbf{M}$	list of decreasing numbers from $M-1$ to 1
$\mathrm{p}(\mathbf{x}_t \mid \mathbf{z}_{1..t})$	the posterior probability of position given all observations
$\mathrm{p}(\mathbf{z}_t \mid \mathbf{x}_t)$	the likelihood of observed feature position over the image
$\mathrm{p}(\mathbf{x}_t \mid \mathbf{x}_{1-\mathbf{M}})$	the motion model
$\mathrm{p}(\mathbf{x}_{1-\mathbf{M}} \mid \mathbf{z}_{1..t})$	the prior
$q(\mathbf{x}_t)$	the *diffused prior* $\int \mathrm{p}(\mathbf{x}_t \mid \mathbf{x}_{t-\mathbf{M}})\mathrm{p}(\mathbf{x}_{t-\mathbf{M}} \mid \mathbf{z}_{1..t-1})$
R	rank of subspace approximating global motion
$\mathtt{M}, \mathtt{M}_t$	measurement matrix of precomputed tracks in frames $t-M+1$ to t
$\mathtt{B}, \mathtt{B}_t$	rank R motion basis for frames $t-M+1$ to t
$\mathtt{P}$	the motion predictor matrix: $\langle \mathbf{x}_t \rangle = \mathtt{P}\mathbf{x}_{t-\mathbf{M}}$
$\Sigma_{t-\mathbf{M}}$	the covariance matrix on likelihood
$\mathtt{A}_t, \mathtt{A}^*, \mathtt{A}'$	affine warp matrices
W	the affine warp function: $W(\mathbf{x}) = \mathtt{A}\mathbf{x} + \mathbf{t}$

Figure 4: **Notation III.** Summary of notation used in Chapter 4.