Video Object Segmentation From Long Term Trajectories

A dissertation submitted to the University of Dublin
for the degree of Doctor of Philosophy

Gary Baugh
Trinity College Dublin, February 2011

Signal Processing and Media Applications
Department of Electronic and Electrical Engineering
Trinity College Dublin

Declaration

I hereby declare that this thesis has not been submitted as an exercise for a degree at this or any other University and that it is entirely my own work.

I agree that the Library may lend or copy this thesis upon request.

Signed,

Gary Baugh

February 1, 2011.

Abstract

In this thesis we explore the problem of video object segmentation and also propose techniques that have better performances compared to other state-of-the -art techniques presented to date. We define video object (VO) segmentation as the task of labelling the various regions in an image sequence as belonging to a particular object. The main focus in this thesis is on two categories of VO segmentation techniques we define as *sparse trajectory segmentation techniques* and *dense pixel segmentation techniques*. A sparse trajectory segmentation approach labels a sparse set of pixel points tracked over a sequence. Here tracking of sparse pixel locations from frame to frame in a sequence generates *feature point trajectories*. Unlike a *sparse segmentation* where only trajectories are labelled, a dense pixel segmentation process labels all the pixels in a sequence as corresponding to the objects of interest in a sequence.

The major challenge in dense pixel segmentation is providing temporal and spatial consistency from frame to frame in the segmentation produced for a sequence. The majority of previous dense segmentation techniques fail to provide a robust solution to this problem. Hence we propose a dense segmentation Bayesian framework that utilizes long term trajectory (temporal) information in order to introduce spatiotemporal consistency.

Our technique first classifies the sparse trajectories into sparsely defined objects in the sparse trajectory segmentation step. Then the sparse object trajectories together with motion model side information are used to generate a dense segmentation of each video frame. Unlike previous work, we do not use the sparse trajectories only to propose motion models, but instead use their position and motion throughout the sequence as part of the classification of pixels in the dense segmentation. Furthermore, we introduce novel colour and motion priors that employ the sparse trajectories to make explicit the spatiotemporal smoothness constraints.

We are mainly motivated to produce techniques that can be used in cinema post-production applications. Semi-automated object segmentation is an important step in the cinema post-production workflow. Hence, we have two adaptations of our general segmentation framework that produce semi-automatic and automatic segmentations respectively. In the semi-automatic segmentation process a user guides the segmentation process, while in the automatic process no user information is required.

The long term tracking of sparse features in an image is important for many applications other that VO segmentation, such as motion estimation and people surveillance. The majority of existing tracking frameworks are based on some kind of prediction/correction idea. However, given a careful selection of interest points throughout the sequence, the problem of tracking can be solved with the Viterbi algorithm. We present in this thesis a novel approach to interest point selection for tracking using the Mean Shift algorithm over short time windows. The resulting points are then articulated within a Viterbi algorithm for creating very long term tracking data. The tracks are shown to be more accurate than traditional KLT implementations and also do not suffer from accumulation of error with time.

Acknowledgments

In this thesis we address the problem of video object segmentation. The ideas presented in this work have been influenced directly and/or indirectly by my supervisors (present and past), other researchers, my colleagues, family, friends, strangers and the various social cultures I have been exposed to over my life.

I would like to thank my undergraduate project supervisor at the University of the West Indies Dr. Cathy-Ann Radix for getting me started in the field of image/video processing. She also introduced me to my current supervisor at Trinity College Prof. Anil Kokaram. If I should write a book about my postgraduate studies, Dr. Radix and Prof. Kokaram would definitely be mentioned in the 'Genesis' (*In the beginning...*) chapter of this book.

Thanks to Anil and Akash ('Pon de Scene' Pooransingh) for arranging my first cricket game in Ireland, just a day after I arrived. They created a convincing caribbean atmosphere, which made me feel as if I never left my homeland of Jamaica, '*yeah man*'. Since we are all from the West Indies, the rest of the world must understand that "we don't like cricket, oh no, we love it!"[1] Special thanks to past and present members of the Dublin University Museum Players (DUMP) cricket team.

I would like to thank present and past members of the SIGMEDIA and other signal processing groups in the Electronic and Electrical Engineering (EEE) Department at Trinity College. Special mentions for Darren Kavanagh, Cedric Assambo, Franklyn Burke, Ranya Bechara, Claire Masterson, Craig Berry, Kangyu Pan, Mo*ham*ed Ahmed, Andrew Hines, Félix Raimbault, Finnian Kelly, Luca Cappelletta, Ken 'Cool Zeen' Sooknanan, Róisín Rowley-Brooke, Andrew Rankin, John Squires, Riccardo Veglianti, Daire Lennon, Valeria Rongione, Phil Parsonage, Dr. Gavin Kearney, Dr. Rozenn Dahyot, Dr. Hugh Denman, Dr. Damien Kelly, Dr. Dan Ring, Dr. Brian Foley, Prof. Frank Boland and Dr. Deirdre O'Regan. Thanks to the O'Regan family for taking care of me for a few Irish Christmases. Dr. François Pitié, Dr. David Corrigan and Dr. Noami Harte all provided me with useful advice which greatly influenced my work. Thanks to François and Postdoc Dave for the chess games and a roof over my head (with Skysports) respectively.

Special thanks to the support staff of the EEE Department; Teresa Lawlor, Robbie Dempsey, Bernadette Clerkin, Nora Moore and Conor Nolan. Conor provided the tool (Remote Desktop) that allowed me to check my simulations running on both Dave's (pc190) and Félix's (pc167) workstations from anywhere in the world. Thanks to both these guys for tolerating my CPU and memory hungry simulations.

Funding for my research has been provided by the Royal College of Surgeons in Ireland (RCSI), Science Foundation Ireland (SFI) eLearning project (SFI-RFP06- 06RFPENE004), Enterprise Ireland (EI) Dumping Detective project, the European Union (EU) FP7 research project

[1] From the song "*Dreadlock Holiday*" by 10CC.

i3DPost (Contract Number 211471), and Adobe Systems.

My family and friends have always been supportive of me pursuing my passion for Engineering. Thanks to my friends from Canada Hall (Hermits), Univerisity of the West Indies; Bread Board, Cabbage, Potter, Wall This, Rat Boy, Double Blank, Marks, Blue and Jezzy. Thanks to my family Mummy, Daddy, Stephanie, Demoy (Toto Strong Man), Kurt, Sheryl and Julie-Marie.

I must say a special thanks to Prof. Anil Kokaram who has not only been my supervisor, but also my team mate, captain and friend. I deeply appreciate everything you have done for me over the past four years.

Contents

List of Acronyms

DFD Displaced Frame Difference

EM Expectation Maximisation

GME Global Motion Estimation

HMM Hidden Markov Model

ICM Iterated Conditional Modes

MAP Maximum *a posteriori*

RANSAC Random Sample Consensus

SIFT Scale Invariant Feature Transform

MSER Maximally Stable Extremal Regions

KLT Kanade-Lucas-Tomasi

VO Video Object

Semi-Auto Semi-automatic

SVD Singular Value Decomposition

Contents

1 Introduction

Video cameras have become ubiquitous items in most households since the early 2000s. The majority of people living in developed countries own a mobile phone which usually has a video camera. The availability of cameras to the general population has facilitated the generation of an overwhelming amount of video content. People create videos for various requirements such as documenting their lives, entertainment and social interactions. The internet acts as a catalyst for video content production as video sharing sites such as *youtube.com* allow people to share their content with a global audience.

Prior to the invention of video and audio capturing devices, civilizations recorded their knowledge and history mostly through writing books, and indirectly by creating art and architecture. Books are written in a language for which the basis unit of information is usually a *word*. With the semantics of various languages understood, humans were then able to manipulate written content, for the purpose of generating new content or transmitting already generated content.

We can consider video to be playing a similar role for the current human civilization as books did for previous civilizations. Knowledge is increasingly being recorded in videos, and we must in some way be able to decompose a video into some basic atomic units of information. That is, find the equivalent for video content of what *words* are for written content. The popular postulation in the video research community is that *objects* are the '*words*' of video content. Note that we are considering a video to be only a sequence of images with no corresponding audio content.

A video is a sequence of 2D projections (images) of a 3D visual world. The concept of this 3D visual world being comprised of *objects* naturally applies to video. The task of identifying

the various objects in a video is defined as *video object segmentation*. Once we are able to identify the *objects* in some video content, we can then proceed to manipulating this content, for generating new content or transmitting already existing content in more efficient ways.

This thesis is concerned with the problem of *video object segmentation*. We propose techniques for extracting the *objects* in a video sequence. Some important modern application areas for video object (VO) segmentation are video compression, visual surveillance, video compositing, object detection, behaviour recognition. In the next section we will discuss how some of these modern applications can benefit from using *video object segmentation* techniques.

1.1 Application Relevance

Videos in general contain a significant amount of redundant information. The background scene may remain the same for several images in a sequence. Hence decomposing a video into a collection of *objects* allows this redundancy to be identified. This redundancy manifests itself as *objects* being repeated over several frames. The idea behind improving video compression techniques is basically encoding a single *object* representation for all the respective repeated *objects*. This saves vital video transmission bandwidth and disk storage space, which is important for sharing video content online.

Surveillance using video cameras is an obvious way of facilitating the increasing need for security in a ever growing population. Current visual surveillance systems usually have a human in the loop who monitors several simultaneous video feeds, and he is required to identify improper activities. This human is obviously going to make more errors as he becomes fatigued and also when the number of video feeds increases. Hence there is an insatiable need for automated visual surveillance systems. An essential requirement for these systems is being able to identify *objects*, which can be done by utilizing a video object segmentation technique. Once objects of interest are identified through *object detection*, we can proceed to analyzing the behaviour of these *objects*. *Behaviour recognition* in some way involves constructing 'sentences' that describe the activities in a scene in a 'language' in which the basis building blocks are '*objects*'.

Writing about something that did not occur in real life is called *fictional* writing, while the opposite scenario is called *non-fictional* writing. In a similar fashion we may consider a video of a scene that did occur in real life as being a *non-fictional* video. If the images in a video are manufactured/synthesized in some way this video may be considered as a *fictional* video. The art of creating *fictional* videos is defined as video compositing, which is used in the movie and television industries to tell convincing visual stories. The majority of the *objects* required to create *fictional* videos (some objects are computer generated) are extracted from *non-fictional* videos using a video object (VO) segmentation technique. Hence the quality of the video composites produced is directly related to the VO segmentation technique used.

Libraries for storing written content are possible because *words* can be indexed for searching through the gross content available. The content of interest can be found by specifying some

keywords. There is already a need for accessing video content in a similar way, by specifying '*keyobjects*'. This would be especially useful for allowing users to find content they are interested in with more discrimination power. The success of such a video content search engine will depend on a video object segmentation technique in order for *objects* to be identified and indexed.

The qualities of the video object segmentations required vary according to the application of interest. Post production for example demands high quality segmentations, where the objects of interest must be properly delineated. However, for surveillance a segmentation quality below post production quality is usually acceptable. In surveillance applications we are more interested in where the objects in a scene are generally located. Hence precise edge delineations are not required. We aim at producing post production quality segmentations with the video object segmentation techniques proposed in this thesis.

1.2 Thesis outline

The remainder of this thesis is organised as follows.

Chapter 2: Video Object Segmentation: A Review

A review of previous *video object segmentation* techniques is presented in this chapter. The main focus in this review is on techniques that can be utilized in video compositing (post production application). We discuss the major issues with producing high quality segmentations for video sequences, and how the various approaches address these issues. At the end of this chapter, we present a VO segmentation technique we proposed in 2008 [15] for a visual surveillance application. Some of the ideas in this technique are used in the design of a VO segmentation framework present later in this thesis.

Chapter 3: A Viterbi Tracker for Local Features

Previous VO segmentation techniques have reported improved segmentation performances by utilizing long term sparse feature point trajectories. These trajectories were used to estimate more accurate motion models with a lower computational cost compared to the conventional method of using optical flow [53]. The most popular trajectories used are Kanade-Lucas-Tomasi (KLT) tracks [55], which are trajectories of corner features tracked over several frames in a sequence.

In this chapter we present a new local feature tracking framework that utilizes a Viterbi [90] tracking strategy. We use this framework to track SIFT [73] features over an image sequence. These SIFT [73] feature trajectories allow much more accurate motion models to be estimated for a sequence compared to KLT trajectories. This conclusion is supported by the quantitative analysis done at end of this chapter.

Chapter 4: Sparse Trajectory Segmentation

In order to estimate motion models using feature point trajectories, these trajectories must be grouped according to their 2D image motion. A set of trajectories are grouped together if they have similar motion according to a define motion model. The task of identifying these groups is defined as *sparse trajectory segmentation*.

A *sparse trajectory segmentation* technique is proposed in this chapter. This technique has two main steps; a *initialization* and a *refinement* step. In the *initialization* step an rough initial grouping of the trajectories is estimated using ideas from previous techniques. The final *refinement* stage improves the initial grouping estimate by applying spatially and temporal smoothness strategies.

This chapter and the next give details of the *initialization* and *refinement* step respectively.

Chapter 5: Sparse Trajectory Segmentation: Model Refinement with Spatial Smoothness

This chapter is a continuation of the proposed *sparse trajectory segmentation* in the previous chapter. Here we give details of the *refinement* step, where a Bayesian framework is presented for labelling the trajectories as belonging to a particular motion group.

Chapter 6: Sparse Trajectory Segmentation Performance Evaluation

The performance of our *sparse trajectory segmentation* technique is assessed in this chapter. We compare our segmentation results with five state-of-the-art 3D segmentation technique, and two 2D sparse segmentation techniques. All these techniques are discussed in the review chapter (chapter 3). The two 2D segmentation technique are defined as *J-Linkage* and *Region Growing* respectively.

We use the Hopkins dataset [117] of 155 sequences supplied with ground truth trajectories segmentations to compare our segmentations with those of the 5 state-of-the-art 3D sparse segmentation techniques and the *J-Linkage* technique. For the *Region Growing* technique a quantitative analysis was not possible, so we conducted visual comparisons instead. We also did visual comparisons for the *J-Linkage* technique as well.

At the end of this chapter we demonstrate that unlike previous techniques our *sparse trajectory segmentation* technique produces satisfactory segmentations for real world sequences. These real world sequences contain a significant amount of non-rigid articulated motions.

Chapter 7: Dense Pixel Segmentation

Unlike our *sparse trajectory segmentation* technique which assigns object labels to a sparse set of pixel sites in a sequence, our *dense pixel segmentation* techniques label every pixel site in a sequence. We propose two *dense pixel segmentation* techniques in this chapter, one is

unsupervised (fully automatic) while the other is *supervised* (requires user assistance). Both *techniques* however use the same general Bayesian framework, and utilize the segmentations produced by *sparse trajectory segmentation* process.

The *sparse trajectory segmentation* step provides motion models (used in likelihood design) for the objects in a sequence. We use these models along with the spatial locations of the trajectories to introduce spatial and temporal smoothness strategies in our *dense pixel segmentation* process.

Chapter 8: Dense Segmentation Performance Evaluation

The performances of our *dense pixel segmentation* techniques are evaluated in this chapter. We compare our segmentations with those of three previous techniques that are discussed in the review chapter (chapter 3). One these techniques is based the work of Kokaram [62] where sequences are automatically segmented in foreground and background regions. The other two techniques are referred to as *Feature-Cut* and *Video SnapCut* respectively. Both these techniques presented in 2009 are state-of-the-art *supervised* techniques proposed by Ring [100] and Bai *et al.* [12] respectively.

We assess the performance of these techniques quantitatively using manually segmented ground truths for three sequence.

Chapter 9: Conclusions

The final chapter assesses the contributions of this thesis and outlines some directions for future work.

Demonstration DVD

Videos of the results presented in this thesis are included in an accompanying DVD. Appendix E gives details of how this DVD is organised.

1.3 Contributions of this thesis

The new work described in this thesis can be summarised by the following list:

- A Viterbi tracking framework for local features.
  - SIFT tracker
  - Outlier detection
- A sparse trajectory segmentation technique that enforces spatiotemporal smoothness.
  - Extensive comparisons with previous work

- A dense pixel segmentation framework for producing automatic and/or semi-automatic video object segmentations.
  - Occlusion detection
  - Automatic segmentation of a sequence using geodesic distances
  - Extensive comparisons with previous work

1.4 Publications

Portions of the work described in this thesis have appeared in the following publications:

- "Feature-based Object Modelling for Visual Surveillance" by Gary Baugh and Anil Kokaram, in *International Conference on Image Processing*, San Diego, CA, USA, October 2008.
- "A Viterbi tracker for local features" by Gary Baugh and Anil Kokaram, in *Proceedings of SPIE Visual Communications and Image Processing*, San Jose, CA, USA, January 2010.
- "Semi-Automatic Motion Based Segmentation using Long Term Motion Trajectories" by Gary Baugh and Anil Kokaram, in *International Conference on Image Processing*, Hong Kong, China, September 2010.

2

Video Object Segmentation: A Review

Video object (VO) segmentation is the task of selecting objects of interest from a sequence of images. We define these objects as constituting the *foreground* of the image sequence. The foreground is naturally composited into the sequence via the video/image capturing process. Hence, a VO segmentation process is required to extract this foreground in some way. Generally a segmentation process uses *mattes* (masks) to indicate which pixels in a sequence belong to the foreground. These *mattes* are commonly referred to as *alpha mattes* in the VO segmentation literature [27, 93, 123, 125].

The VO segmentation problem has received a great deal of attention in the literature as it is important for many video applications, such as surveillance [15], tracking [51], action recognition [8, 36], object detection [72, 96], scene reconstruction [14], and video matting [6, 10, 27, 30, 62, 74, 125]. A robust solution remains elusive for segmenting sequences with dynamic scenes containing both camera and multiple object motions, especially non-rigid motions.

In this chapter we will review previous video object segmentation techniques that are relevant to our work. We are mainly interested in techniques that can be applied to the video matting problem, which is important in the post production industry. We will also discuss some of the main challenges in producing quality segmentations.

First, we introduce in the next section how a VO segmentation technique generally specifies the set of foreground objects in an image sequence. This is important for future discussions.

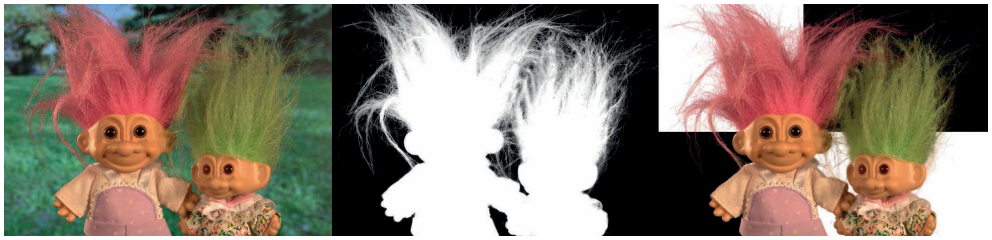

Figure 2.1: Example of alpha matte extraction and compositing of an image. **Left**: Image (800×563 pixels) from the training dataset provided by Rhemann [99]. **Center**: Alpha matte computed with technique of Gastal [47]. **Right**: Composite of the extracted foreground on a new background.

2.1 Mattes and Modelling

In a sense, all VO segmentation techniques consider that the observed image in each frame is created by mixing different image layers. For clarity assume that there is just one foreground image layer $I_f^{\mathcal{F}}(\mathbf{s})$ and a background image layer $I_f^{\mathcal{B}}(\mathbf{s})$ in frame f, where $\mathbf{s} = (x, y)$ is the spatial coordinates of a pixel in this frame. Then the observed image $I_f(\mathbf{s})$ is the result of the following mixing or modelling equation.

$$I_f(\mathbf{s}) = \alpha_f(\mathbf{s}) I_f^{\mathcal{F}}(\mathbf{s}) + (1 - \alpha_f(\mathbf{s}))\, I_f^{\mathcal{B}}(\mathbf{s}) \tag{2.1}$$

The idea was first made explicit by Pentland [35, 87, 88] and Wang [127]. $\alpha_f(\mathbf{s})$ is therefore the value of the *alpha matte* for pixel site $\mathbf{s}$ at frame f in the sequence. $\alpha_f(\mathbf{s})$ can be binary ($\alpha_f(\mathbf{s}) \in \{0, 1\}$) or a continuous value between 0 and 1 (real number in $[0, 1]$), depending on the granularity of the VO segmentation process. A continuous *alpha matte* is required to do a high quality composition of an extracted foreground onto a new background (application in post production), especially for foreground objects that are translucent in some way (e.g. hairs, thin fabrics). Most VO segmentation approaches first generate binary *alpha mattes* and then use these mattes to create more refined continuous *alpha mattes* [12].

Fig. 2.1 shows an an image (left) segmented using a technique by Gastal [47] to produce a continuous *alpha matte*. Here the *alpha matte* is shown in the center, where $\alpha_f(\mathbf{s}) = 1$ and $\alpha_f(\mathbf{s}) = 0$ are coloured white and black respectively. The various shades of gray indicate values of $\alpha_f(\mathbf{s})$ between 0 and 1. Note that the majority of the 'gray' values of $\alpha_f(\mathbf{s})$ correspond to the hairs of the dolls in fig. 2.1. The colours of the pixels corresponding to the ends of these hair strands are influenced by both the background and foreground colours. Hence a fractional (gray) $\alpha_f(\mathbf{s})$ here indicates how much the foreground colour influences site $\mathbf{s}$. With this continuous *alpha matte* (center of fig. 2.1), a high quality composition of the foreground on to a new background (right) can be created.

Foreground and background are indicated in a binary *alpha matte* by having $\alpha_f(\mathbf{s}) = 1$ and $\alpha_f(\mathbf{s}) = 0$ respectively. Fig. 2.2 shows examples of binary *alpha mattes* for frame 1 (left), 12

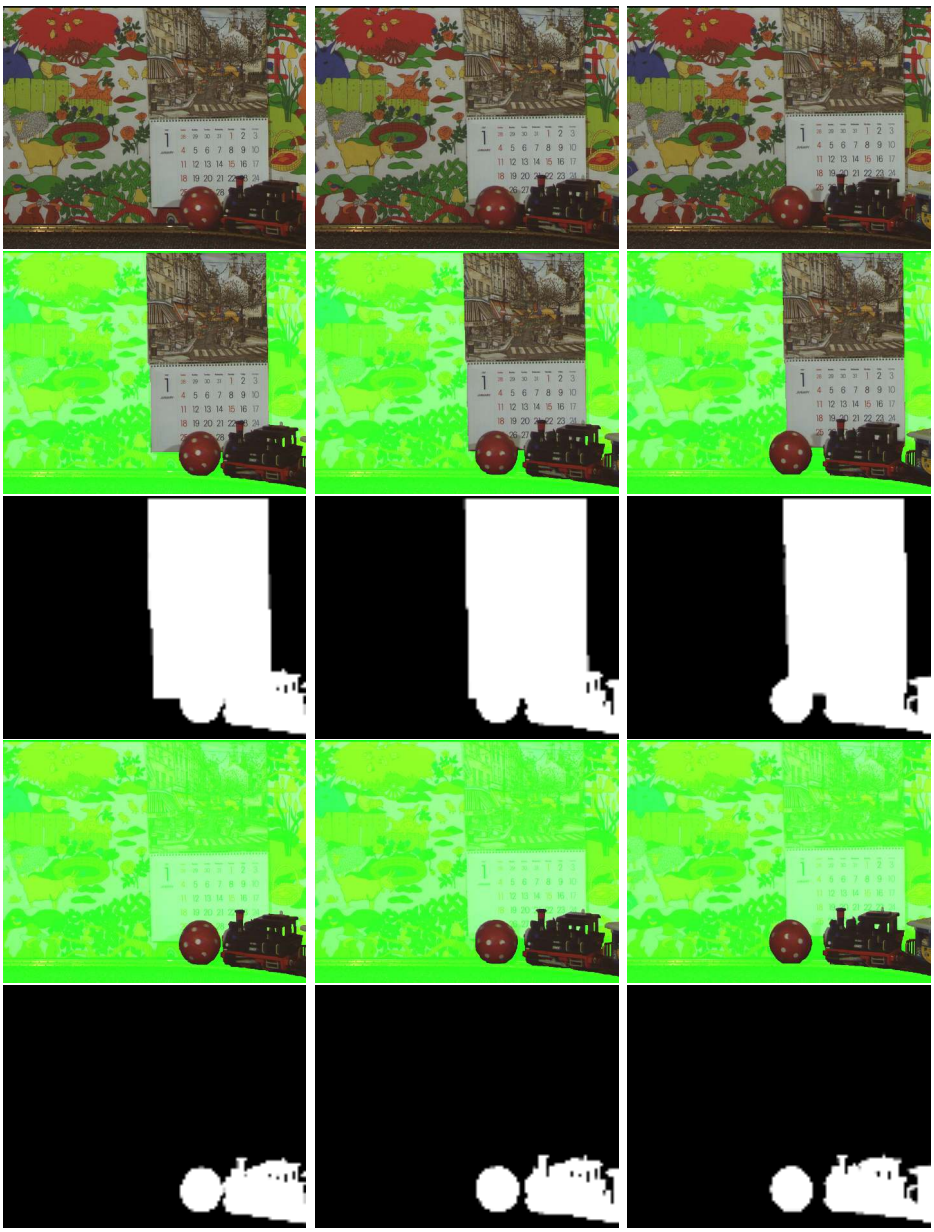

Figure 2.2: *Examples of the binary* ***alpha mattes*** *for frames 1 (left), 12 (center) and 25 (right) in the* ***Calendar and Mobile*** *sequence, where these frames are shown in the* ***top*** *row.* ***Third row****: The* ***alpha mattes*** *considering the calendar, train and ball all constitute the foreground. The foreground extracted with these mattes are shown in the* ***second*** *row. Here the background is shaded in green.* ***Bottom row****: The* ***alpha mattes*** *considering only the train and ball constitute the foreground. The foreground extracted with these mattes are shown in the* ***fourth*** *row. A pixel site* **s** *coloured* ***black*** *or* ***white*** *in an* ***alpha matte*** *correspond to* $\alpha_f(\mathbf{s}) = 0$ *and* $\alpha_f(\mathbf{s}) = 1$ *respectively.*

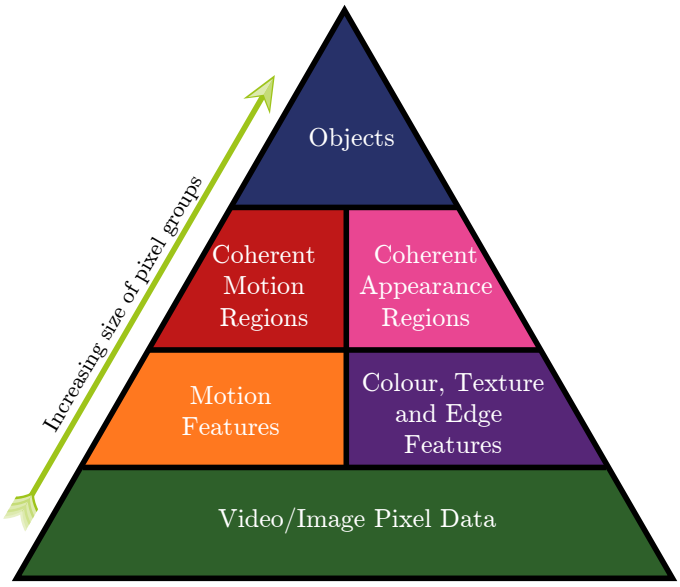

Figure 2.3: *The heuristics used in generating video object segmentations. Motion (**orange**), colour, texture and edge (**purple**) features are extracted from the pixel data (**green**). These features are then utilized in finding spatially and temporally coherent motion (**maroon red**) and appearance (**pink**) regions in the images of a sequence. These coherent regions are expected to exclusively represent a single object (**blue**) in scene. As we move up (**green arrow**) this heuristic pyramid we increase the number of image pixels that are grouped together.*

(center) and 25 (right) in the well known *Calendar and Mobile* sequence. The *third* row shows the mattes where the calendar, train and ball are all considered to constitute the foreground. Also shown in the *bottom* row are the mattes where only the ball and the train are foreground. The *second and third* rows (fig. 2.2) show the foregrounds extracted with the corresponding mattes in the *third and bottom* rows respectively. Here the background in each image is shaded green.

2.2 A Pyramid of Techniques

Previous VO segmentation techniques generally rely on motion and appearance cues in order to generate the required segmentations. However, there are some techniques where only motion or appearance information is used exclusively [23, 127]. All previous approaches in some way generate heuristics from motion and appearance cues to facilitate the segmentation of an image sequence.

Fig. 2.3 illustrates how different levels of heuristics are generated from motion and appearance cues. The pyramid structure demonstrates the hierarchy of these heuristics. Representing a sequence as a composition of objects (*blue*) is the highest level of semantic reasoning. To date

there are no computer vision algorithms that can reason about real scenes at this level in a useful way. Only human perception can be used to decompose a complex visual scene into objects.

The next level down from the objects in a scene (fig. 2.3) consists of spatially and temporally coherent motion (*maroon red*) and appearance (*pink*) image regions. VO segmentation techniques can be considered to operate at this level. These coherent regions can be identified in a sequence using a combination of motion (*orange*), colour, texture and edge (*purple*) features. These features are extracted from the raw video/image pixel data (*green*) at the lowest level.

All VO segmentation techniques can be considered to be estimating an *alpha matte* $\alpha_f(\mathbf{s})$ for image $I_f(\mathbf{s})$, given a set of model parameters $\Theta_f(\mathbf{s})$. This task can be expressed in the Bayesian framework below, where the solution to the segmentation problem is the Maximum a posteriori (MAP) estimate of $\alpha_f(\mathbf{s})$.

$$p(\alpha_f(\mathbf{s})|I_f(\mathbf{s}), \Theta_f(\mathbf{s}), \alpha_f(\sim \mathbf{s})) \propto p_l(I_f(\mathbf{s}), \Theta_f(\mathbf{s})|\alpha_f(\mathbf{s}))p_s(\alpha_f(\mathbf{s})|\alpha_f(\sim \mathbf{s}), \Theta_f(\mathbf{s})) \tag{2.2}$$

Where $p_l(I_f(\mathbf{s}), \Theta_f(\mathbf{s})|.)$ and $p_s(\alpha_f(\mathbf{s})|.)$ are the likelihood and prior distributions respectively. $\alpha_f(\sim \mathbf{s})$ are the labels in some defined neighbourhood of pixel site $\mathbf{s}$ at frame f. VO segmentation techniques generally differ in how they go about determining the various models and their corresponding parameters $\Theta_f(\mathbf{s})$. These models are used to describe the coherent motion and appearance regions in fig. 2.3. With the right parameters $\Theta_f(\mathbf{s})$, these models explain the observed motion, colour, texture and edge features at the lower heuristics levels, which usually are part of the likelihood design. The design of the prior $p_s(\alpha_f(\mathbf{s})|.)$ typically fulfills the requirement for the coherent regions to be spatially and temporally smooth over a sequence.

In the case where all the object in an image sequence are rigid and consistent in appearance and motion, there are VO segmentation techniques that will segment this sequence at the 'objects' level. That is, every object with unique motion and appearance can be completely extracted from these sequences. However, real world sequences of interest do not fall into this restricted category of sequences that comprise of only rigidly moving objects. Real world sequences usually contain articulated non-rigid objects that may have inconsistent motions and appearances. There might be several coherent motion and appearance regions (fig. 2.3) associated with each of these objects. A human would be required to associate these coherent regions with a particular object.

2.2.0.1 Technique Categories

The need for high quality segmentations of real world sequences has motivated a category of VO segmentation techniques that we defined as *supervised* techniques. These *supervised* techniques include a user in the segmentation loop, in order to handle difficult sequences. These techniques emerged in the late 90s, where the technique proposed by Mitsunaga [76] in 1995 is an early example of a *supervised* VO segmentation approach.

We define a VO segmentation technique where no user assistance is utilized as an *unsupervised* technique. Three of the earliest examples of these techniques were proposed by Potter [92] in

the in the late 70s, Adelson [3] and Thompson [110] in the early 80s. Since there is no human in the segmentation loop, these techniques produce much lower quality segmentations compared to *supervised* techniques.

In *supervised* VO segmentation techniques the user usually interacts with the system at the 'objects' level (fig. 2.3). Here the user may supply reference *alpha mattes* for some of the frames in a sequence and then allow the system to segment the remaining frames [11, 12, 100]. In these user supplied *alpha mattes* the objects of interest are specified to the system. These object specifications at the 'objects' level in the heuristics pyramid (fig. 2.3), constrain the lower level heuristics (coherent motion and appearance regions, etc). Hence the various heuristics are constrained implicitly from the top ('objects') of the pyramid to the bottom ('video/image pixel data'). These constraints allow *supervised* systems to discriminate better between the features (motion and appearance) of the various objects in a sequence.

Fig. 2.2 shows examples of user supplied *alpha mattes* for frame 1 (left), 12 (center) and 25 (right) for the *Calendar and Mobile* sequence. In the *third* row the user specifies that the foreground consists of the calendar, train and ball. However, in the *bottom* row the foreground consists of the ball and the train only. This user interactivity makes the *supervised* system more useful in modern applications such as post production.

Unlike a *supervised* technique, an *unsupervised* technique has to gather the various heuristics from the bottom ('video/image pixel data') of the pyramid (fig. 2.3) to the top ('objects'). Since these *unsupervised* systems have no notion of what constitutes an object, there are no guarantees that the segmentations produced would be in some way useful.

2.2.0.2 Challenges of VO Segmentation Techniques

There are some fundamental issues which affect all video object segmentation techniques, irrespective of whether they are *supervised* or *unsupervised*. Perhaps the most important is maintaining spatial and temporal consistency from frame to frame for the segmentation of an image sequence. The human visual system is very sensitive to temporal inconsistencies that may occur in an image sequence [122]. Segmenting the frames in a sequence independent of each other often results in temporal incoherencies. Hence learnt heuristics at each frame must be propagated in some way over a sequence. Typical video cameras only capture at 25 fps, thus the sampling rate is too low for dealing with the motion of fast objects and often introduces motion blurs. Hence building correspondence across frames and maintaining temporal coherence is very difficult in this case.

Another issue which affects only *supervised* techniques is how to effectively use the information supplied by the user in segmenting the sequence. Here the user may specify *alpha mattes* for certain frames, and the design of the segmentation technique will determine which of these mattes should influence the segmentation of a particular frame. Effective use of the user information is expected to reduce the amount of corrections that must be applied to the final

segmentations.

We will discuss later how the various VO segmentation techniques handle these issues.

2.3 Unsupervised Techniques

As previously mentioned, *unsupervised* techniques do not utilize any user supplied information. The majority of these techniques employ the idea of using 'layers' to represent the image regions in a sequence. Here a 'layer' simply refers to an image region where all the pixels have similar motion and/or appearance according to a specified model. For example, an Affine motion model may be used for specifying motion layers. 'Layers' correspond to the coherent motion and appearance regions in fig. 2.3 as discussed previously.

The idea of segmenting an image into layers was introduced by Wang [127] and Pentland [35, 87, 88] during the early 90s. This idea become very popular because it is an intuitive way of thinking about how images are structured.

Fig. 2.4 shows examples of typical layers for frames 1 (*first* row) and 25 (*third* row) in the *Calendar and Mobile* sequence. Here the pixels for the background (left), calendar (left center), train (right center) and ball (right) constitute the four layers shown respectively. The *alpha mattes* of the layers for frames 1 and 25 are shown in the *second* and *fourth* rows respectively.

An image can be composited from a set of layers (specifically the image data) and the corresponding *alpha mattes* for that image. Also the depth order of these layers must be specified. Fig. 2.5 illustrates the composition of frame 1 in the *Calendar and Mobile* sequence from the four layers for this frame (fig. 2.4).

2.3.1 Layer-based Techniques

With the introduction of the layer representation [35, 87, 88, 127], researchers were given a platform for integrating spatial and temporal smoothness strategies into the segmentation process. In the original work of Wang, optical flow [53] between adjacent frame pairs was used to extract motion layers. The use of only two frames at a time led to temporal inconsistency in the layers extracted. Also no spatial smoothness schemes were utilized in assigning pixels to the various layers. Hence there was room for improvement in this foundational work.

Some techniques were proposed for more robust extraction of motion layers, using sparse feature point correspondences across frames [129, 130], and deriving new motion features based on subspace decomposition [60]. Jojic [58] presented a technique where a set of layers are learned from a sequence using the Expectation Maximization (EM) algorithm [38], considering both appearance (pixel intensity) and motion features (optical flow). These layers are defined as 'flexible sprite' in [58]. Each sprite changes its shape from frame to frame according to estimated 2D image transformations. Each frame in a sequence can then be synthesized from compositing the various learnt sprites (layers). Temporal and spatial consistency is implicitly achieved in

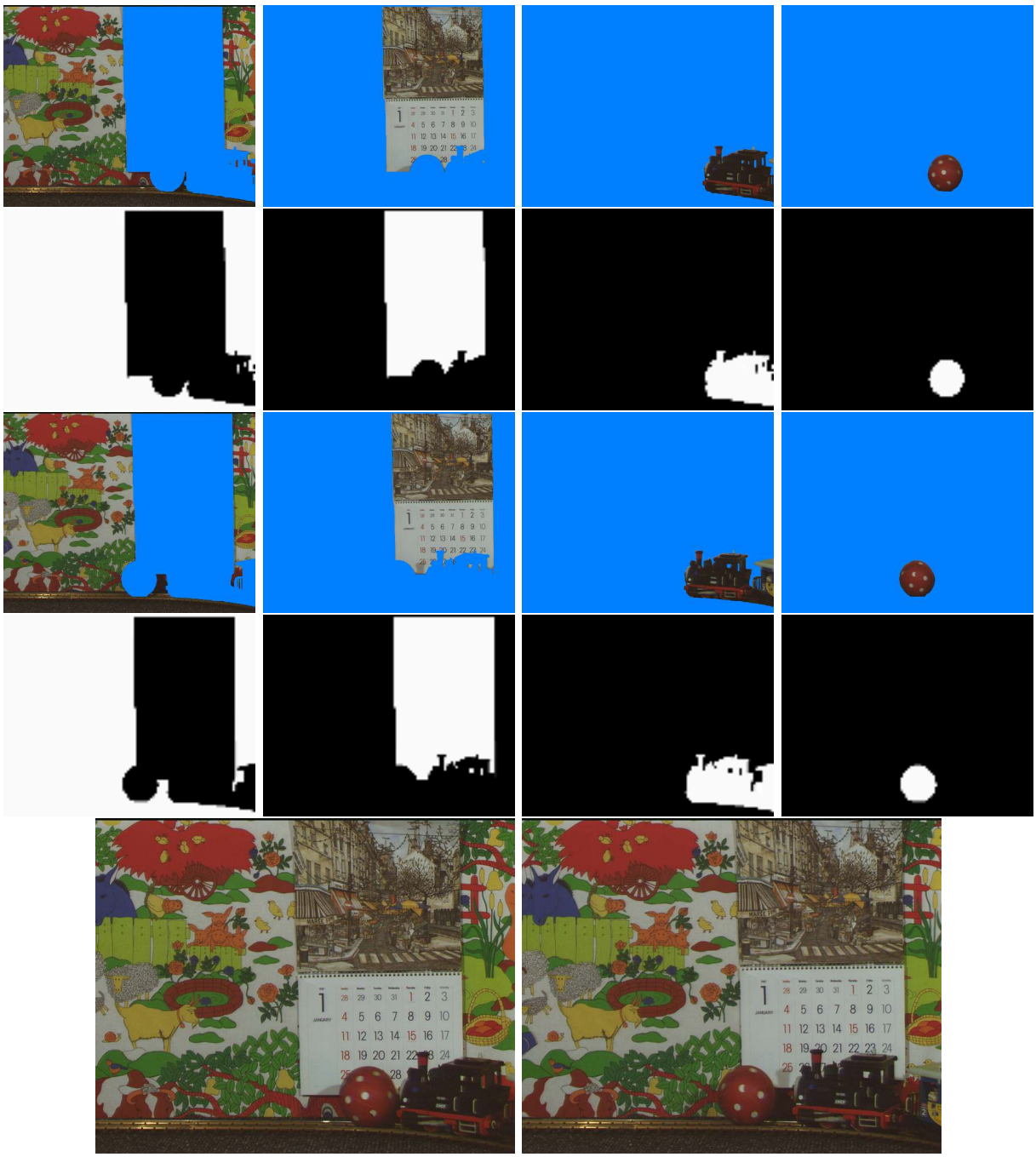

Figure 2.4: *The layer representation of frames 1 (bottom left) and 25 (bottom right) in the* ***Calendar and Mobile*** *sequence.* ***Top and third rows****: The layer images for the background, calendar, train and ball for frame 1 and 25 respectively.* ***Second and fourth rows****: The alpha mattes (α) for the corresponding images in the top and third rows. Here a pixel site* $\mathbf{s}$ *coloured* ***black*** *or* ***white*** *correspond to* $\alpha_f(\mathbf{s}) = 0$ *and* $\alpha_f(\mathbf{s}) = 1$ *respectively.*

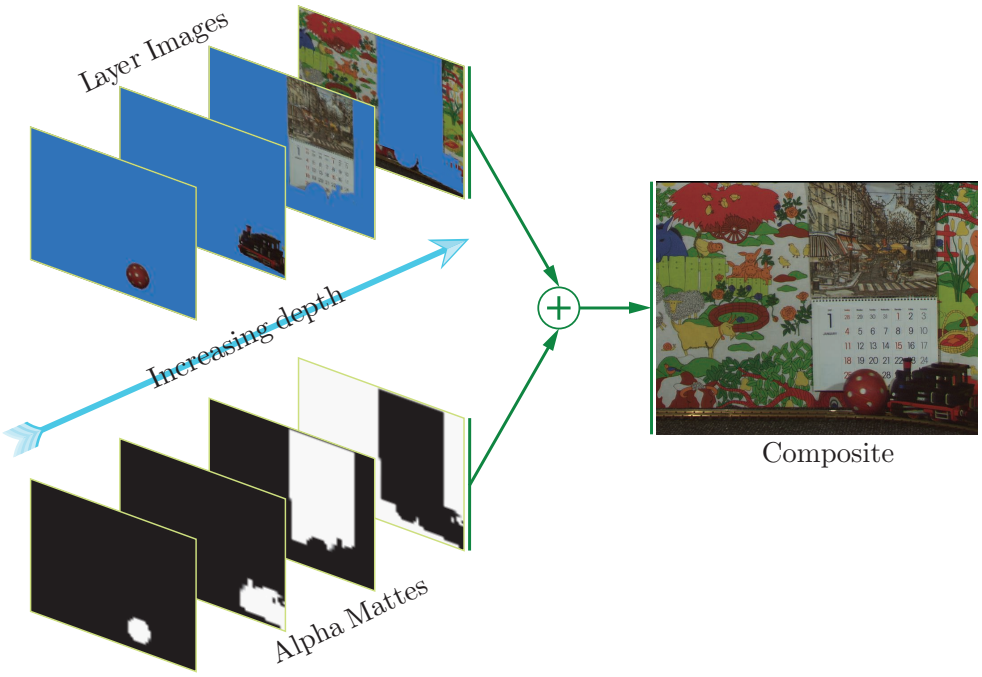

Figure 2.5: *The composition of frame 1 in the* ***Calendar and Mobile*** *from the layer images and the alpha mattes. The depth ordering determines how the pixels from the various layers occlude each other.*

this technique since the sprite maintains a consistent appearance throughout an image sequence. However, this technique is only effective on sequences where the motion of the objects can be approximated with parametric models.

Prior to Jojic [58], Ayer [9] and Sawhney [105] both presented techniques that also utilized EM for determining motion layers using optical flow. Smith [109] identified that these EM based approaches only work well when the motion of the objects in a sequence are very distinct. If the motions of the objects are quite close, the EM algorithm may fail to converge when estimating the layers. This non-convergence leads to the overall failure of the segmentation process. Also most of these approaches are prone to local minima, namely those described in [32, 58, 105, 116], that use EM or variational methods for learning the parameters of the layers.

Another EM based approach was proposed by Weiss [128], where a Markov Random Field [48] (MRF) model was used to encourage label smoothness in the assignment of pixel to the layers. Employing MRFs for enforcing spatial smoothness is common in a lot of approaches [34, 40, 64, 79–81]. Boykov [25, 135] provided a convenient way of finding the optimal solution to MRF problems using graphs (Graphcuts). Hence researchers [30, 56, 131] in the 2000s started formulating the layer segmentation problem in MAP-MRF frameworks which were then solved with Graphcuts.

The work of Kumar [64] is similar in some ways to the approach of Jojic [58]. However,

Kumar uses a MAP-MRF framework which is solved with loopy belief propagation. Also unlike Jojic, Kumar has more tolerance for the change in motion and appearance of the layers over the duration of a sequence (flexible sprites).

Criminisi [33] proposed a real-time layer-based technique that used a Conditional Random Field (CRF) [65] and a second order Hidden Markov Model prior to enforce spatial and temporal smoothness respectively. CRFs play a similar role to MRFs, except they are not prone to the label bias problem [65]. Hence the use of CRFs instead of MRFs should provide better segmentations. However the results of Criminisi [33] were not significantly better than previous layer-based techniques. This was probably because Criminisi's approach had a real-time constraint, where future frames are not available for the current segmentation.

2.3.1.1 Two Layers

Determining the optimal number of layers for a sequence is a challenging task. Hence some techniques simply assume a two layer representation [16,34,62,82]. Here one layer represent the background, and all other objects are assigned to the other layer (foreground). The motion of the background layer is assumed to be the estimated global motion. All pixels which do not follow the motion of the background, are assigned to the foreground layer.

The two layer technique by Kokaram [62] in 2005 is typical of these two layer approaches. It has been implemented in a software package called *NUKE* developed by The Foundry [2]. The algorithm is included in the MotionMatte plugin bundled into version 4.7 and all subsequent versions of *NUKE*. We will use this technique as a baseline to compare performance later.

Kokaram [62] presented a Bayesian framework where the likelihood design was based on motion compensated displaced frame differences (DFDs) for the current frame with respect to multiple past and future frames. The corresponding past and future frames were motion compensated with respect to the motion of the background layer. The motion of this layer was assumed to be the global motion, and this motion was estimated with a technique in [63] which disregards the motion of the foreground. The likelihood design here constrains the *alpha* value $\alpha_f(\mathbf{s})$ for pixel site $\mathbf{s}$ at frame f to be 1 (foreground) when the motion compensated DFDs for this site are large in both future and past temporal directions. Spatial smoothness was injected via an MRF prior. The MAP estimates for the binary *alpha mattes* were then generated using the Iterated Conditional Modes (ICM) algorithm [17].

2.3.2 Region Merging and Hierarchical Clustering Techniques

Motion layers extraction using optical flow [53] can be erroneous at the boundaries of objects. These undesired errors are inherited from use of optical flow. It is well known that incorrect optical flow fields usually correspond to image regions that have been revealed or occluded. These image regions naturally occur at the boundaries of objects, as these objects move at relative depths to each other in the image plane.

To overcome the errors at the boundaries of motion layers inherited from the use of optical flow, some researchers have proposed other *unsupervised* techniques. Some of these techniques can be placed into two categories, which we define as *region merging* and *hierarchical clustering* techniques.

Region merging approaches [4, 39, 78] generally perform two separate motion and colour segmentation [28] processes. It is assumed that the colour segments are almost always a subset of motion segments. That is, the majority of the pixels in a colour segment usually correspond to one motion segment. Unlike the boundaries between motion segments, the boundaries between the colour segments are assumed to be more closely related to the actual object boundaries in an image. Hence the colour segments are combined in a strategic way using the motion segments as guides. These combined colour segments represent the final segmentation, in which the boundaries of the objects are better defined compared to those in the motion segmentation result.

Obviously *region merging* techniques fail when the foreground and background colours are similar, since the colour segments would extend across both the foreground and background image regions. There is also a dependence in these techniques on the quality of the motion segmentation step. Here a poor motion segmentation will lead to a poor final segmentation.

2.3.2.1 Hierarchical Clustering

Hierarchical clustering techniques [50, 85, 86] treat an image sequence as a 3D spatiotemporal volume [61], and typically use a variant of the Mean shift algorithm [28] for segmentation [37, 126] based on colour only. These techniques started to emerged in the early 2000s, with the improvement in the amount of computational resources available in standard computers. These approaches require a significant amount of memory since all the video data usually need to be manipulated at once.

The approach presented by Grundmann [50] in 2010 is a typical example of a *hierarchical clustering* technique. Grundmann [50] extended an image segmentation technique by Felzenszwalb [41] to handle the segmentation of videos. In the work of Felzenszwalb [41] an image is segmented by using a graph-based clustering technique. The pixels in an image are the nodes in a derived graph structure. In this graph each pixel site is connected to its 8 immediate neighbours via edges. Edge weights are derived from per-pixel normalized colour differences. Subsequently, pixels are merged into image regions by traversing the edges and evaluating whether the edge weights are smaller than some estimated local thresholds. These thresholds are generated according to local image texture. The final set of image regions constitute the segmentation.

Grundmann [50] applied the technique of Felzenszwalb [41] to image sequences by defining a spatiotemporal neighbourhood for each pixel. The neighbourhood of each pixel is set to the 26 immediate neighbours in the spatial and temporal dimensions. In the temporal dimension, the neighbours are located along motion trajectories obtained from optical flow [53]. The local

Figure 2.6: *Segmentation results taken the paper by Grundmann [50]. Shown are two frames from four sequences with their corresponding segmentations. The coherent appearance regions are coloured in a similar ways for the segmentations of each sequence. For example, the face of the actor in the* ***bottom*** *row is coloured orange in both segmentations.* ***Top and second rows****: A sequence of a water-skier and an America football game respectively.* ***Third row****: A sequence from the movie* ***Public Enemies****, © 2009 Universal Pictures.* ***Bottom row****: A sequence from the movie* ***No Country for Old Men****, © 2007 Miramax Films.*

thresholds for merging are varied to produce a tree of various segmentation levels. Here small conservative thresholds over segment the sequence, while large thresholds have the opposite effect. Therefore as we move down this tree the sequence become more segmented (over segmented at the lowest level). The idea is that the user can then select the level of segmentation required from those in the generated segmentation tree.

The implementation of Grundmann [50] is quite sophisticated in order to work around computational memory issues. Fig. 2.6 shows the segmentations reported by Grundmann [50] on four image sequences. Note here that only the segmentations at a selected level in the segmentation tree are shown. From these segmentations it may be observed that the technique of Grundmann does not provide good segmentation for some of the boundaries between the objects in these four sequences. This technique is mainly dependent on appearance (colour) information, hence it suffers from not utilizing motion information in a useful way.

Of course these ideas can be extended to exploit motion itself as a feature but memory then becomes an even bigger issue.

2.4 Supervised Techniques

As previously defined a *supervised* VO segmentation technique utilizes user supplied information in order to generate high quality segmentations. The user may supply for some frames in a sequence partially labelled *mattes* called *trimaps*, or fully segmented binary *alpha mattes*, which we define as *key frames*. Also the user may just supply a few foreground and background paint strokes on selected frames.

In a *trimap*, image regions of definite foreground ($\alpha_f(\mathbf{s}) = 1$) and background ($\alpha_f(\mathbf{s}) = 0$) are labelled along with unknown regions ($\alpha_f(\mathbf{s}) = ?$). The segmentation system is subsequently required to generate *alpha* values for the pixels in these unknown region ($\alpha_f(\mathbf{s}) \in [0, 1]$), thus producing a continuous *alpha matte*. Note that binary and continuous *alpha mattes* are sometimes referred to as 'hard' and 'soft' *alpha mattes* respectively in the literature. Also VO segmentation techniques produce 'soft' *alpha mattes* for the frames in a sequence are called *video matting* techniques. For the rest of this discussion it should be assumed that all VO segmentation techniques mentioned produce binary ('hard') *alpha mattes* unless we specify otherwise.

2.4.1 Adaptation of Image Segmentation Techniques

Some *supervised* VO segmentation techniques [10, 27, 52] to date are simply adaptations of successful interactive image (*supervised*) segmentation techniques [24,66,71,102,103,134] to the problem of segmenting video sequences. Wang and Cohen [125] provide a comprehensive review of these interactive image segmentation approaches.

Operating on each individual frame of a video independently using only an interactive image segmentation (IIS) technique would create some undesired problems. These IIS techniques [21, 26, 67, 71, 102, 103, 124, 134] usually require that the user supplies for an image a *trimap* or paint foreground and background strokes. Doing this for every frame in a video would be a tedious task. An additional problem is that slight differences in the extraction of the foreground from frame to frame would lead to obvious temporal inconsistencies, especially along the edges of the foreground.

Unlike still image segmentation techniques, VO segmentation techniques can exploit motion information. Each video frame is temporally correlated with the nearby frames. Hence the challenge for *supervised* VO segmentation techniques is to accurately identify foreground regions with minimal user assistance by utilizing all available motion and appearance information.

Chuang *et al.* [27] in 2002 proposed a *video matting* technique based on a Bayesian image matting technique [134] they presented earlier. In this technique [27] optical flow was used to propagate a set of user drawn *trimap* to the unspecified frames in a sequence. A clean background plate was estimated as well to assist in the segmentation process. With a *trimap* propagated to every frame, the Bayesian image matting technique [134] which utilized this clean plate for estimating background colours was then used to generate *alpha mattes* for each frame in a sequence. The performance of this VO segmentation technique is limited by the accuracy of

the optical flow estimation, which is often quite erroneous. Furthermore, the background plate estimation assumes the background undergoes only planar-perspective transformation, which is not true of most real world sequences of interest. This technique would fail if the objects in the background are moving.

2.4.2 Propagation of Binary Mattes

A subcategory of *supervised* VO segmentation techniques [11, 12, 70, 100, 123] require the user to supply fully segmented binary *alpha mattes* (key frames) for selected frames in a sequence. The segmentation system then automatically produces binary *alpha mattes* for the remaining frames in the sequence by propagating the labels in these *key frame* in some way. The binary *alpha mattes* are then generally used to automatically generate *trimaps* for each frame in a sequence. Once these *trimaps* are estimated, an image matting technique can then be used to create continuous ('soft') *alpha mattes*. These VO segmentation techniques [11, 12, 70, 100, 123] reduces the work load of the user, as *trimaps* do not have to be specified manually for every frame in a sequence. Hence less user interactions are required compared to the previously discussed approaches that are adaptations of image segmentation techniques.

Li *et al.* [70] in 2005 extended the pixel-level 3D Graph cut technique proposed by Boykov [26] to better handle video sequences. They use three main steps in their technique, which are a 3D Graph cut binary segmentation, tracking user defined local image windows for correcting the binary segmentation, and extracting continuous *alpha mattes* with an image matting technique [107]. The first two steps are used to produce accurate binary *alpha mattes*, from which *trimaps* are generated for the final image matting step. The user initially supplies fully segmented binary *key frames* for some of the frames in a sequence. These *key frames* are articulated in the 3D Graph cut segmentation step, and the user then corrects any undesired segmentation errors.

In the next two sections we will discuss the *Video SnapCut* [12] and *Feature-Cut* [100] techniques, as we compare our work to both techniques later in this thesis. Both techniques proposed in 2009 are state of the art *unsupervised* VO segmentation techniques which propagate user defined binary *alpha mattes*.

2.4.2.1 Video SnapCut

The *Video SnapCut* segmentation algorithm was proposed by Bai *et al.* [12]. An implementation of this algorithm is bundled into Adobe **After Effects CS5** [1] as a matting tool called *Roto Brush*.

This algorithm uses local motion information to propagate appearance information from a segmented frame t to the current unsegmented frame $t + 1$. This appearance information is captured in overlapping rectangular windows of fixed sizes placed along the contour of the foreground. These local windows are defined as *local classifiers* in the paper [12]. The *top row* of fig. 2.7 shows examples of these *local classifiers* W_k^t as *yellow squares*. The *local classifiers*

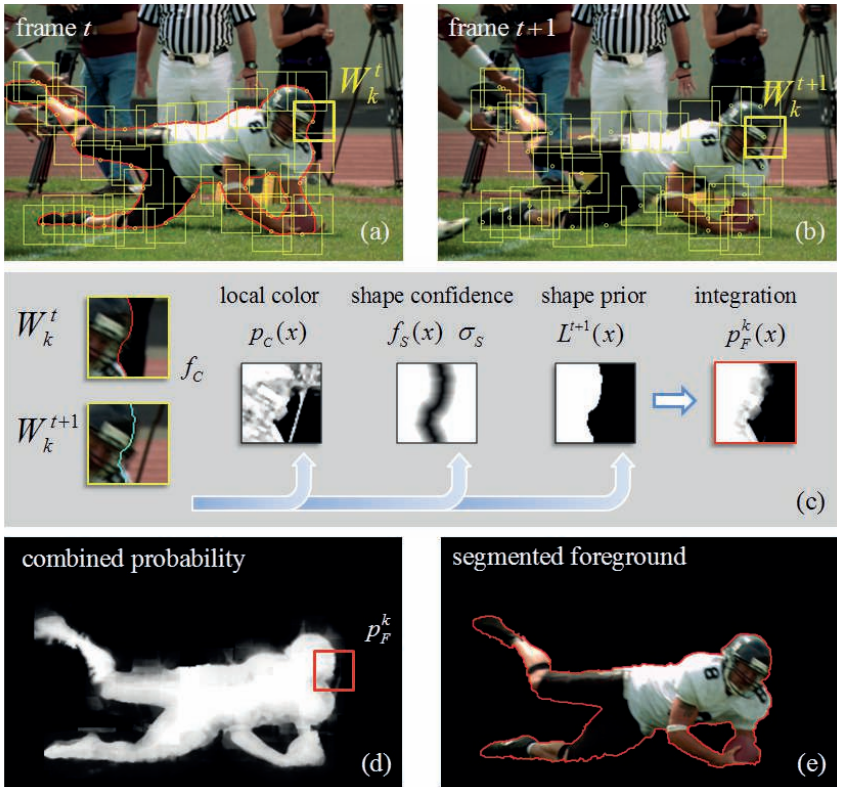

Figure 2.7: *This figure is taken from the paper by Bai et al. [12] which is an illustration of the local classifiers proposed in the paper.* ***(a)****: Overlapping classifiers (****yellow squares****) are initialized at frame t along the contour of the foreground (****red curve****).* ***(b)****: These classifiers are then propagated onto the next frame using local motion information.* ***(c)****: Each classifier contains a local colour and shape model, which are initialized at frame t and refined if necessary at frame* $t+1$. ***(d)****: Local foreground/background classification results are then combined to generate a global likelihood for all the pixels in frame* $t+1$. *(e): The final segmentation for frame* $t+1$. *Video courtesy of* ***Artbeats****.*

W_k^t in frame t (left) and propagated to the current frame $t+1$ (right) using motion information obtained from optical flow [53] and SIFT [73] feature matching. Here the *local classifiers* W_k^t (window) in frame t correspond to W_k^{t+1} in frame $t+1$. In each *local classifier* colour and shape models (*middle row* of fig. 2.7) are generated for the foreground/background image region inside the corresponding window. These models are then used in the design of a posterior distribution for the *alpha* labels for the current frame $t+1$. Finally, a MAP estimate for the binary *alpha matte* for this frame $t+1$ is generated using a Graph cut [135] solution. The *bottom row* of fig. 2.7 shows an example of the likelihood probability map on the left (d) for the frame segmented on the right (e).

In the *Video SnapCut* segmentation approach [12], the user fully segments the first frame (1^{st}

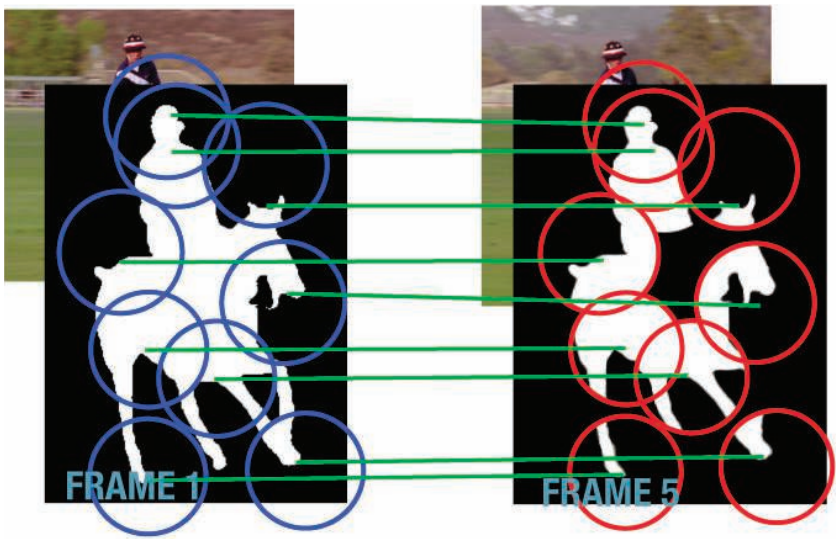

Figure 2.8: *This figure is taken from the paper [100] by Ring. This is an example of propagating a user define* ***alpha matte*** *for frame 1 to the an unsegmented* 5^{th} *frame in a sequence of a Polo player. Using feature correspondences (****green lines****) between these frames, regions of the known matte at frame 1 are 'pushed' into frame 5. Each of the* ***blue*** *and* ***red*** *circles represent the neighbourhoods of the respective SIFT features that have been matched in both frames. The partial matte in frame 5 is missing some foreground regions, but the subsequent optimization process (MAP estimation for labels) usually is able to fill in these missing region.*

key frame) in a sequence, and all subsequent frames are processed in the order 2 to T (where T is the number of frames in the sequence). The user is allowed to correct any segmentation errors generated for the current frame $t+1$, and by default these changes only affect the frames after $t+1$ (*forward* propagation), i.e. frames $t+2, \ldots, T$. However the user can make the current frame $t+1$ a *key frame*, and allow previous frames (2 *to* t) to be influenced by this *key frame* (*backward* propagation).

It will be shown later in this thesis that this technique generally performs well. However, it has some issues with distinguishing revealed background regions from legitimate foreground regions.

2.4.2.2 Feature-Cut

The *Feature-Cut* segmentation algorithm proposed by Ring [100], uses SIFT features matches between frames to propagate a collection of user *key frames* to the unsegmented frames in a sequence. Feature matches are used to identify corresponding image regions between a *key frame* and the current unsegmented frame of interest. These *key frames* contain foreground ($\alpha_f(\mathbf{s}) = 1$) and background labels ($\alpha_f(\mathbf{s}) = 0$), and the image region correspondences are used to propagate these labels to the unsegmented frames. Fig. 2.8 demonstrates this idea, using two

frames from a sequence of a Polo player. Here the user supplies a binary *alpha matte* for frame 1 (left), and this mattes is 'pushed' into the unsegmented 5^{th} frame (right). At each pixel site $\mathbf{s}$ in the unsegmented frame f, votes are accumulated whether site $\mathbf{s}$ is foreground or background. These votes are accumulated from all the user supplied *alpha mattes* within a temporal window of the the current unsegmented frame f. These foreground/background votes for each pixel site are utilized in the design of a likelihood distribution. Ring then proceeds to generate a MAP estimate of the binary *alpha matte* for frame f, using a Graph cut solution, and a MRF prior distribution.

This technique does not utilize appearance information in the traditional sense of colour and/or shape modelling. The author use the intensity of matched pixels in SIFT neighbourhoods to influence the foreground/background votes. Hence, this technique has some issues with the temporal consistency of the segmentations produced from frame to frame. Also for image sequences with low texture there are usually not enough SIFT feature matches between the frames for effective matte propagations. Therefore poor segmentations are usually produced for sequences with relatively low image texture.

2.5 Sparse Video Object Segmentation

The techniques discussed previously produce a label for every pixel in a sequence. We define these techniques as producing a *dense* segmentation. However, there is another category of video object segmentation techniques that only label a sparse set of pixel sites in a sequence. We define these techniques as producing a *sparse* segmentation. In general, these *sparse* segmentation techniques utilize feature point trajectories such as Kanade-Lucas-Tomasi (KLT) tracks [55] for estimating the various coherent regions (layers) in a sequence. Hence these techniques explicitly use only motion information.

Fig. 2.9 shows examples of *dense* (center) and *sparse* (right) segmentations for frame 5 in the *Calendar and Mobile* sequence. The spatial locations of the trajectories at frame 5 are indicated with coloured 'dots' in the *sparse* segmentation on the right. The motion history of the trajectories are indicated with lines that extend from the corresponding 'dots'. Both segmentations (dense and sparse) were produced with unsupervised techniques we will propose later in this thesis. The dense segmentation is estimated from the sparse segmentation by associating in some way the pixels in a sequence with the various trajectory groups (bundles). The corresponding sparse and dense image regions are coloured in a similar way in both segmentations. For example the pixels and trajectories for the calendar are coloured *yellow* in the *dense* and *sparse* segmentations respectively.

Sparse VO segmentation techniques are usually used in applications where the objects in a sequence do not have to be accurately delineated. Examples of these applications are visual surveillance [5, 15, 89], human behaviour analysis [77] and video summarization [49, 108]. However, Wills *et al.* [129,130] in 2003 proposed a *dense* segmentation technique that first generates

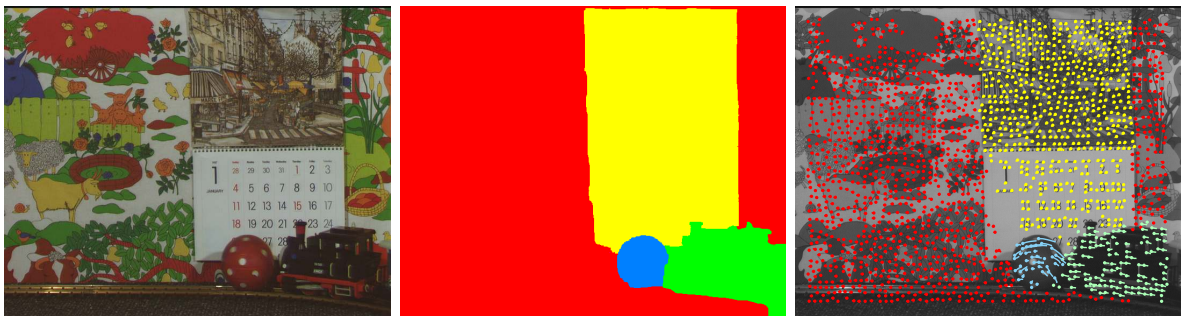

Figure 2.9: *Examples of the* ***dense*** *(center) and* ***sparse*** *(right) segmentations of frame 5 (left) in the* ***Calendar and Mobile*** *sequence. The* ***dense*** *segmentation is produced using the* ***sparse*** *segmentation in a technique we will propose later in this thesis. Corresponding sparse and dense segmentation regions are coloured in a similar way.* ***Right****: The* ***sparse*** *segmentation for frame 5, where the spatial location of the trajectories are indicated with coloured* ***dots****. The line that extends from each dots show the motion history of the corresponding trajectory.*

a *sparse* segmentation for 'guiding' this *dense* segmentation process. Here the objects in the final *dense* segmentation were of a reasonable quality compared to other equivalent unsupervised VO segmentations techniques. Wills *et al.* [129] were the first to propose this two step idea, where a *sparse* and then a *dense* segmentation is performed.

For the technique of Wills *et al.* [129], the motion layers were estimated in the *sparse* segmentation step, and subsequently the pixels in a sequence were assigned to these layers in the final *dense* segmentation step. Feature point trajectories are generated for the *sparse* segmentation step by matching Förstner interest points [45] across adjacent frames in a sequence. These trajectories were then clustered using a variant RANSAC algorithm [42], where each cluster represents a specific motion layer. The motion models for the layers were proven to be more reliable than those obtained with a technique that uses a dense optical flow field instead of sparse point trajectories. This idea was supported by Xiao [56] in 2005, who presented a similar two step (sparse then dense) approach.

2.5.1 Categories for Sparse Techniques

We define two categories for *sparse* VO segmentation techniques proposed to date, where these categories are 2D and 3D segmentation techniques. In a *sparse* 3D segmentation technique the feature point trajectories must be labelled according to the object that generated them. That is, if there are N objects in a sequence, each trajectory takes a label $n \in \{1 : N\}$ according to the object it corresponds to. Hence all the trajectories with label n correspond to the same object. Fig. 2.10 shows an ideal *sparse* 3D segmentation (center) for a sequence of a *truck* driving on a road. The truck and background are the two objects in this sequence, and the trajectories for both objects are labelled *yellow* and *red* respectively.

Figure 2.10: ***Left**: A truck shown at frames 1 in a motion sequence from the Hopkins dataset [117] available online. Some roughly planar surfaces on the truck are highlighted in red, green, yellow, purple, and cyan.* ***Center**: The desired labels for a 3D segmentation technique. The points along the trajectories for the background and truck are **red** and **yellow** dots respectively.* ***Right**: The trajectory segmentation result produced by our proposed **sparse** segmentation technique. Each trajectory bundle shown in a different colour roughly represents a single surface of the truck.*

A *sparse* 3D segmentation technique can be considered to group trajectories of similar 3D motion. However, a 2D *sparse* segmentation technique groups trajectories of coherent 2D motion in the image plane, according to a defined motion model (e.g. Affine). The number of 2D trajectory groups for a sequence may not correspond to the number of 3D objects. The illustration on the right of fig. 2.10 shows the 2D segmentation for the 'truck' sequence produced by a technique we will propose later in this thesis. Here each of the 10 trajectory group labels is indicated with a different colour. The perspective of the scene with respect to the camera causes different image regions to have varying 2D motion in the image plane. For example the front of the truck moves faster than the side in the image plane.

2.5.2 Sparse 2D Segmentation Techniques

As previously mentioned, *sparse* 2D segmentation techniques group feature point trajectories with coherent 2D motion in the image plane according to a specified motion model. The *sparse* segmentation performed by Wills [129, 130] discussed previously is the earliest example of these techniques. Even though there were earlier techniques that segment feature point trajectories in some way, we are only interested in techniques that are in some way applicable to the *dense* segmentation problem. Therefore we will only discuss techniques that fit this criterion.

Two of these techniques were proposed by Pundlik [94, 95] and Fradet [46] in 2007 and 2009 respectively. We will use these techniques as baselines for comparing the performance of our proposed *sparse* segmentation technique later.

2.5.2.1 Affine Region Growing

The approach proposed by Pundlik groups KLT [55] trajectories with coherent motion according to Affine motion models. We define each group of trajectories with coherent motion as a *trajectory bundle*. Each *trajectory bundle* represents a single motion layer. The segmentation system of Pundlik is causal, therefore in his design the segmentation for the current frame depends only on the trajectories that exist for the current frame f, and the previous frame $f-1$.

Let there be T trajectories $\mathcal{X}_t$, $t \in \{1:T\}$, where $\mathcal{X}_t$ is a trajectory that exist at both the current and previous frames at the minimum. Pundlik determines the neighbourhood structure for these spare trajectories $\mathcal{X}_t$ by performing Delaunay triangulations on the spatial locations of these trajectories at the previous frame $f-1$. A trajectory bundle (motion layer) is discovered by first selecting a trajectory at random and examining if its motion is similar to the motion of its neighbouring trajectories. This trajectory and its neighbours are grouped together if the defined error for each trajectory using the Affine motion model is below a threshold. The group is then allowed to expand until no more neighbouring trajectories can be placed in the group. When the first group/bundle is completely discovered, another ungrouped trajectory $\mathcal{X}_t$ is selected at random, and the process is repeated to discover the other bundles.

At every frame $f \in \{2:F\}$ trajectory bundles are discovered in the manner described above, where F is the number of frames in a sequence. Pundlik tries to assign any new trajectories at frame f to the bundles previously discovered in frame $f-1$ if this is possible, else new bundles are formed. This ad hoc method of identifying trajectory bundles leads to temporal inconsistencies in the motion layers discovered. Also just using motion information over two adjacent frame f and $f-1$ at a time does not allow this technique to discriminate well between the motions of the various trajectory bundles (motion layers).

2.5.2.2 J-Linkage

Fradet [46] proposed a technique where Affine motion models were estimated for a sequence using the J-Linkage algorithm [112]. The J-Linkage algorithm presented by Toldo [112] in 2008 is a model fitting algorithm similar to RANSAC [42], where the aim is to identify a set of models that describe some given noisy data that may contain outliers.

Using the J-Linkage algorithm a set of random Affine motion models are generated from the trajectories (KLT [55] trajectories are used for most sequences) with the longest duration in a sequence. The models that best explain the observed trajectory data are kept. Each model kept describes the motion of a particular trajectory bundle. Hence a trajectory $\mathcal{X}_t$ is assigned to bundle n if the corresponding motion for bundle n describes the motion of trajectory $\mathcal{X}_t$ the best out of all the bundles.

In this technique no spatial smoothness strategy is used in assigning the trajectories to the bundles, which results in spatial inconsistencies. Also, the motion models are estimated using the trajectories with the longest durations in a sequence, which means the motion of trajectories

with short durations are not described well. These short trajectories usually correspond to non-rigid objects. Hence, this technique may not provide motion layers that describe non-rigid motions in a useful way.

2.5.3 Sparse 3D Segmentation Techniques

Sparse 3D segmentation techniques [59,97,119,132] group trajectories with coherent 3D motion. In general these techniques derive a motion feature space by factorizing (usually with SVD) a matrix formed from the spatial locations of the trajectories in a sequence. The idea is that trajectories that have similar 3D motion reside in a low dimensional subspace in this derived feature space. Hence the solution to the segmentation problem is locating each subspace corresponding to an independently moving 3D object. To locate these subspaces, a clustering or polynomial fitting technique is usually applied to the trajectory points in the derived feature space. Tron [117] provides a comprehensive review of these techniques.

These 3D segmentation techniques have limited applications, since they work only for sequences with rigidly moving objects. Also the quality of the segmentations produced by these techniques deteriorate as the number of objects with different motions increases in a sequence. To date, the state-of-the-art 3D segmentation techniques can successfully segment sequences with a maximum of three independently moving objects. Another issue with these techniques is that the number of objects in the sequence most be specified by a user.

2.5.4 Spatial Inference from Sparse Segmentations

It seems sensible that doing a *sparse trajectory segmentation* should be a powerful step in a subsequent *dense pixel segmentation* of an image sequence. In 2008 we began to explore this idea for a visual surveillance application [15] where inaccurate foreground delineation was tolerable. We only wanted to detect if a new object had been introduced in an outdoor scene of interest. Fig. 2.11 shows two sequences; *Dumping Pedestrian* (687 frames) and *Dumping Car* (1099 frames) which are typical examples of the type of sequences we are interested in for this surveillance application. In both sequences black garbage bags are placed into the respective scenes. We are able to detect the presence of these new objects from the segmentations performed by our technique.

First, the background is modelled as a collection of sparse SIFT features $\mathcal{B}$ learnt from a sequence of training image, which is typically 100 frames of only the background scene. For a frame f that occurs when the surveillance system is online and monitoring a scene, a set of SIFT features $\mathcal{S}_f$ is extracted from this frame. The features for the foreground at frame f are assumed to be those in $\mathcal{S}_f$ that do not have a matching feature in the learnt background feature set $\mathcal{B}$. The *second* row of fig. 2.12 shows the foreground (red '$\times$') and background (yellow '.') features detected for frames 283 (left), 529 (center) and 278 (right) in the *Dumping Pedestrian* sequence. Here the background features are those in the extracted feature set $\mathcal{S}_f$ for frame f that have

Figure 2.11: *Example sequences our surveillance application is expected to segment.* ***Top row****: Frames 1, 321 and 613 in the* ***Dumping Pedestrian*** *sequence of 687 frames. In this sequence a pedestrian enters and exits the scene leaving a black garbage bag behind.* ***Bottom row****: Frames 1, 545 and 1097 in the* ***Dumping Car*** *sequence of 1099 frames. In this sequence a car enters and exits the scene leaving a small black garbage bag behind.*

matching features in the background model. The *top* row shows the corresponding frames for the feature maps in the *second* row. Note that there are features classified as foreground that actually correspond to background objects. These features are generated due to environmental changes subsequent to the background modelling step. Hence these features are not included in the background model. However, we design a foreground pseudo-likelihood map which helps to identify legitimate foreground features from these new background features generated due to environmental changes.

The pseudo-likelihood map generated for frame f considers the spatial locations of the foreground and background features. The idea is that an image region is more likely to be foreground if it has a high density of foreground features. Also this image region must have a low density of background features as well. The foreground and background features are superimposed on these foreground pseudo-likelihood maps in the *second* row of fig. 2.12. Here high and low foreground likelihood values are coloured in *bright orange* and *black* respectively.

2.5.4.1 Tracking Foreground Regions

We apply a threshold to the pseudo-likelihood map in order to identify blobs of foreground pixels. These foreground blobs are then tracked using a Viterbi tracking strategy proposed by Pitie

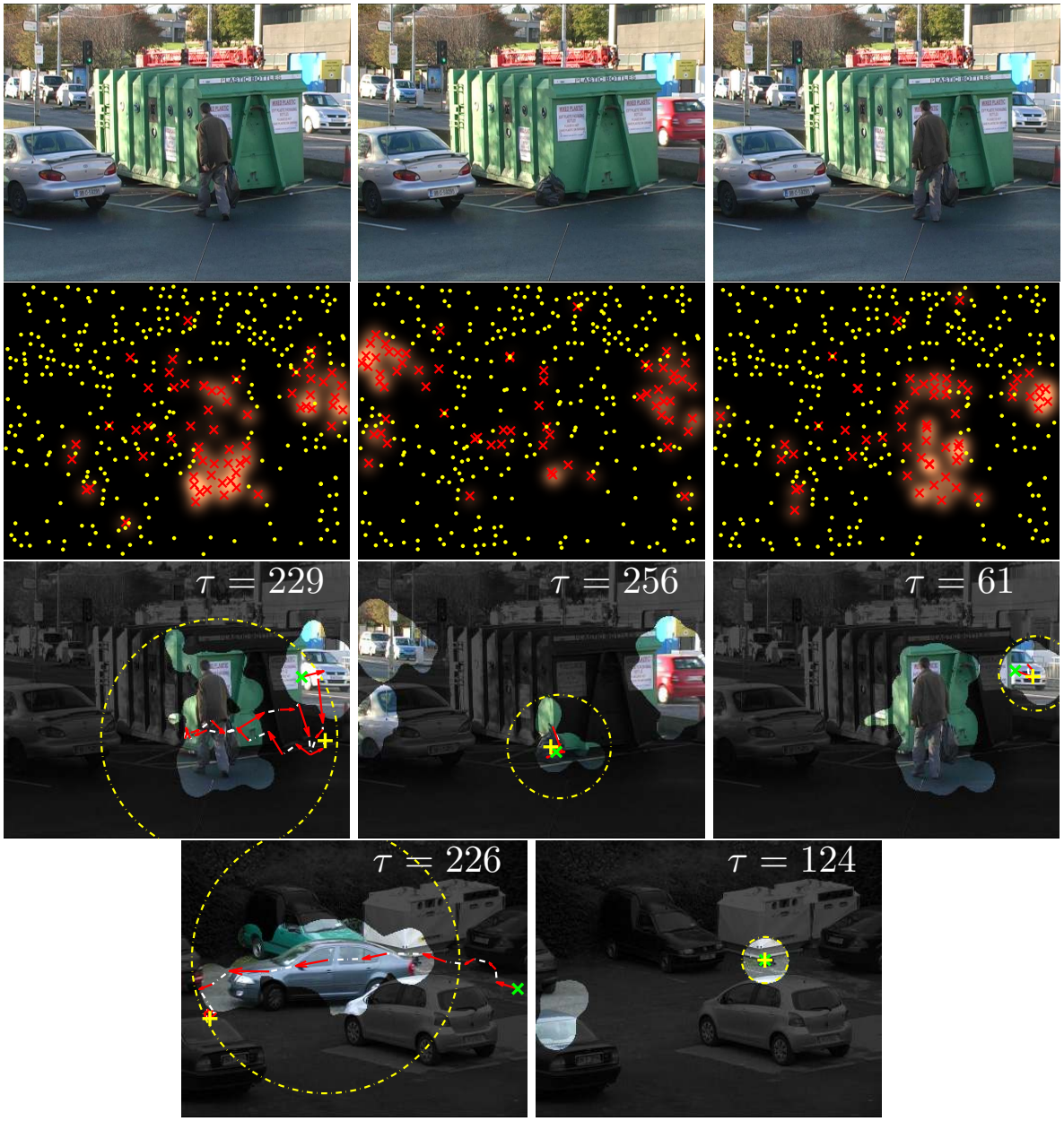

Figure 2.12: ***Top row***: *Frames 283 (left), 529 (center) and 278 (right) in the* ***Dumping Pedestrian*** *sequence.* ***Second row***: *Foreground (×) and background (.) feature points superimposed on a foreground pseudo-likelihood map, for the frames in the top row. In this pseudo-likelihood map high and low foreground likelihoods are coloured* ***bright orange*** *and* ***black*** *respectively.* ***Third row***: *Blob sequences for the* ***Dumping Pedestrian*** *sequence, superimposed on the frames in the* ***top*** *row. The background regions are shaded black, while the foreground is extracted using the foreground/background matte obtained from applying a threshold to the pseudo-likelihood map in the* ***second*** *row. The arrows indicate the transition of the centers of the blobs for a blob trajectory which is τ frames long. The blobs shown (circled) occur at the temporal middle frame in the respective blob trajectories, where the centers of the starting and ending blobs in each trajectory are indicated with $\times$s and $+$s respectively.* ***Bottom row***: *Blob trajectories for the* ***Dumping Car*** *sequence.*

et al. [90]. Hence we obtain trajectories of foreground and background blobs from tracking with the Viterbi algorithm. We are able to differentiate between foreground and background blob trajectories based on their temporal durations. In general, unlike foreground trajectories, background trajectories do not exist for more than 20 frames. Hence a blob trajectory is considered to correspond to a legitimate foreground object if it persists for more than 20 frames.

The *third* row of fig. 2.12 shows the generated foreground blob trajectories for the *Dumping Pedestrian* sequence. Here the length in frames of each blob trajectory is τ. For example $\tau = 229$ for the blob trajectory corresponding to the pedestrian (left). The centers of the starting and ending blobs in each trajectory are indicated with a *green* $\times$ and a *yellow* $+$ respectively. The *red* arrows indicate the displacement between the centers of the blobs in a sequence from frame to frame. We correctly estimate trajectories for the pedestrian and the garbage bag. However, a foreground blob trajectory was generated for a background image region, because this region corresponded to a background object that was stationary during background modelling, but subsequently started to move. This is an understandable result, and some cues other than temporal duration would have to be used to distinguish this trajectory from a legitimate foreground trajectory. For a sequence like the *Dumping Car* sequence (fig. 2.11) where the background objects remain roughly stationary during background modelling and online scene monitoring, we generate the correct number of foreground blob trajectories. The *bottom* row of fig. 2.12 shows the two correctly detected foreground blob trajectories for the *Dumping Car* sequence. These blob trajectories correspond to the car (left) and garbage bag (right) respectively.

2.6 Summary

VO segmentation techniques proposed to date generally struggle to impose temporal and spatial consistency in the segmentations produced. Previous work has shown that using long term feature point trajectories can help to introduce spatiotemporal consistency into the segmentation process. Hence our focus in this thesis is to design VO segmentation techniques with better spatiotemporal consistency. Our *sparse* to *dense* segmentation technique applied to the problem of visual surveillance, proved that the spatial locations of sparse features can be a reliable cue for discerning the general vicinity of the objects in a scene. We therefore propose a technique later that utilizes sparse point trajectories for estimating motion layers, and we design a new likelihood term that considers the spatial locations of the trajectories in each frame. The spatial locations of trajectories were never used in previous work to influence a *dense* segmentation process.

3

A Viterbi Tracker for Local Features

The long term tracking of sparse local features in an image is important for many applications including video object segmentation, camera calibration for stereo applications, camera or global motion estimation and people surveillance. The majority of existing tracking frameworks are based on some kind of prediction/correction idea e.g. KLT [13] and Particle Filters [7]. However, given a careful selection of interest points throughout the sequence, the problem of tracking can be solved with the Viterbi algorithm.

This chapter introduces a novel approach to interest point selection for tracking using the Mean Shift algorithm over short time windows. The resulting points are then articulated within a Viterbi algorithm for creating very long term tracking data. The tracks are shown to be more accurate than traditional KLT implementations and also do not suffer from accumulation of error with time.

The tracking of local regions involves matching them in adjacent images throughout a sequence. It is widely accepted that the ideal is to generate long contiguous tracks. Such tracking data can then be used for camera calibration, surveillance [15], and sparse to fine motion estimation or segmentation [95]. Most point tracking systems are two step: first selecting interest points and then tracking the image material around those points through the sequence. The popular KLT tracker [13], and various Particle trackers [7] follow this idea. However, given that it is possible to locate interest points in all the frames of the sequence, it is sensible to constrain a tracker to match only that image material around those detected interest points. Unfortunately, Mikolajczyk [75] reports that matching local descriptors is unreliable in terms of recall vs precision, since they are only invariant to a few classes of image transformations (affine

for example) encountered in practice. However that experiment was conducted using direct SIFT [73] descriptor matching without any constraints about motion or local image appearance information.

In this work we propose a strategy for selecting interest points more suitable for tracking, and we design a more appropriate feature vector using a colour quantisation step. The idea is first to process the existing interest points to generate a set of 3-frame long seed tracks of high confidence, using a low cost process. The tracking problem is then to extend these seeds using the available interest points. That is best solved with the Viterbi algorithm [43]. This idea enables long term tracking and also is less prone to accumulation of tracking error with time. We discuss track seed selection first, then present the feature vector design and finally construct the elements of the Viterbi tracker.

3.1 Selecting seed tracks

We adopt the use of the SIFT interest point detector [73] for identifying interest points in each frame of an image sequence. It is advantageous in any tracker to be able to select points which are known to be trackable, before attempting to generate long term tracking data. This is not attainable in practice, but what is achievable is to generate a set of short, high confidence tracks before proceeding to long term tracking. To do this we make the interesting observation that by tightly clustering SIFT descriptors (using a fast approximation to the Mean Shift (MS) algorithm [91]) associated with the SIFT interest points over 3 frames, each cluster automatically provides a seed path. The idea here is that by using a small kernel bandwidth (0.3), the SIFT descriptors that are clustered together are so close in the feature space that they can potentially arise from the same point in each of the three frames. In addition, points in the cluster should come from different frames, and be separated by less than D pels hence enforcing a maximum motion constraint. After MS clustering therefore, those clusters containing exactly 3 points and where the points in the cluster are separated by less than D=50 pels in the image plane, are selected as a seed track. In figure 3.1 a simplified view of the 128-dimensional SIFT descriptor space is shown to illustrate the idea.

Seed tracks are generated initially using frame windows of 3 frames with an overlap of 1 frame. Hence the first two frame windows have image indexes of $(1, 2, 3)$, and $(3, 4, 5)$. Where the end of one 3 frame path coincides with the start of the next 3 frame path, the paths are collated into a new longer seed path. This is performed over the entire sequence resulting in seed paths of arbitrary lengths. The next step is to extend the paths using more region information as well as motion constraints. The appearance information is presented below.

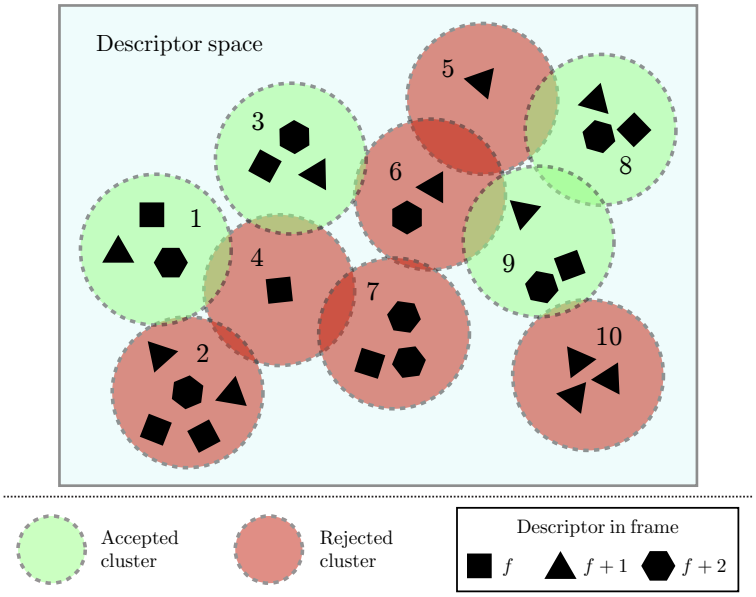

Figure 3.1: *Simplified 2-dimensional illustration of the clustering of the descriptor space for the image subsequence* $(f, f+1, f+2)$*. The size of the clusters are determined by the Mean Shift kernel bandwidth (radius of circle). Accepted clusters (1,3,8,9) contain only a single descriptor from all three images* $f, f+1$ *and* $f+2$*.*

3.2 The Feature Vector and Tracking Preliminaries

In order to describe the local region appearance, a feature vector $\mathbf{v} = (\mathbf{x},\ \mathbf{s},\ \mathbf{g},\ \mathbf{c})$ is associated with each interest point. The vector contains the position of the point $\mathbf{x} = (x, y)^T$, scale and dominant orientation from the SIFT descriptor, $\mathbf{s} = (\sigma\ o)^T$, the 128 point SIFT descriptor itself $\mathbf{g}$ and an adaptively quantised colour vector $\mathbf{c}$ (discussed later).

The ith track, $\mathcal{P}_f^i$, that starts in frame f is now considered to be a sequence of matched feature vectors $\mathbf{v}_f^i$. Figure 3.3 shows this situation for the seed tracks from the previous step. The track $\mathcal{P}_f^3$, for instance, consists of the feature vectors $\left(\mathbf{v}_f^3,\ \mathbf{v}_{n+1}^3,\ \mathbf{v}_{n+2}^3\right)$.

3.2.1 Colour Feature

Colour information should be collected from a region around the interest point. The size of this region should relate somehow to the influence that local image material has on the location of the interest point. The region should be circular (with radius r) since the detector is based on circularly symmetric Gaussians, but depending on the scale the size of the region of interest will change. This relationship is approximately $r \approx 11.4811\sigma + 2.5419$ (obtained from experimentation), where σ is the scale of the local region, obtained from the SIFT interest point detector. With this approximation of r, a pixel at site (u, v) is defined to be in the neighbourhood of an

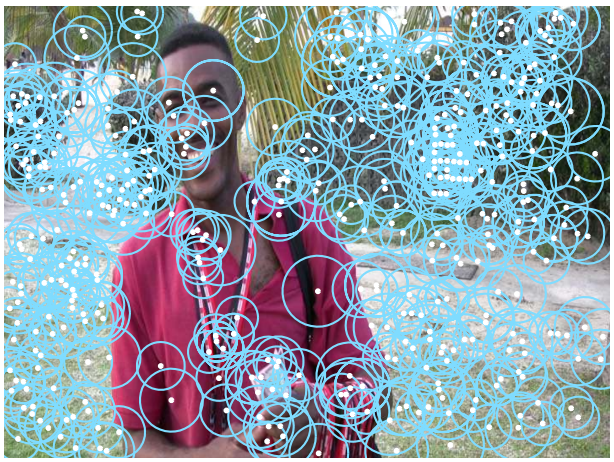

Figure 3.2: *Examples of local regions ('dot' indicate center of region* $(x\ y)$ *) and their corresponding circular neighbourhoods. Note that only a few regions are shown at selected scales for clarity of illustration.*

interest point at (x, y) if $(x-u)^2 + (y-v)^2 \leq r^2$.

Figure 3.2 shows example local regions and their corresponding neighbourhoods. The colour feature vector $\mathbf{c}$ is a normalized histogram of quantized colours in this neighbourhood, using the YUV colour space. It is important to note that the quantization scheme is non-linear and is discussed next. Note that spatiograms suggested by Birchfield [20] could be used as a simplified alternative for creating this vector.

3.2.1.1 Colour Space Quantization

Figure 3.2 shows that the neighbourhoods of the local regions overlap substantially and the union of these regions does not cover the whole image. Therefore some colours are more essential than others in representing these regions. By assuming that the colour profile of a local neighbourhood only changes slightly when tracked over a frame window, due to noise and photometric effects, it is sensible to express the region colours using only the colours in the union of all these local regions. Colours outside these regions are irrelevant. Furthermore, assuming that the colour content does not change wildly over a shot, a compact colour representation derived from the first few frames of the sequence would be sufficient to describe colour content over the subsequent frames. This compact representation is achieved by adaptive colour space quantisation.

Colour space quantization is a well researched topic [54,84,106]. There are two steps: palette design, in which a reduced number of palette colours is chosen, and pixel mapping in which each colour pixel is assigned to one of the colours in the palette. In this work we first create palettes for all local neighbourhoods, and then combine them to obtain a global palette, which is then used to generate the colour feature vectors $\mathbf{c}$. The first 3-frame window is used for quantisation and the resulting quantised colour space is used for the whole sequence.

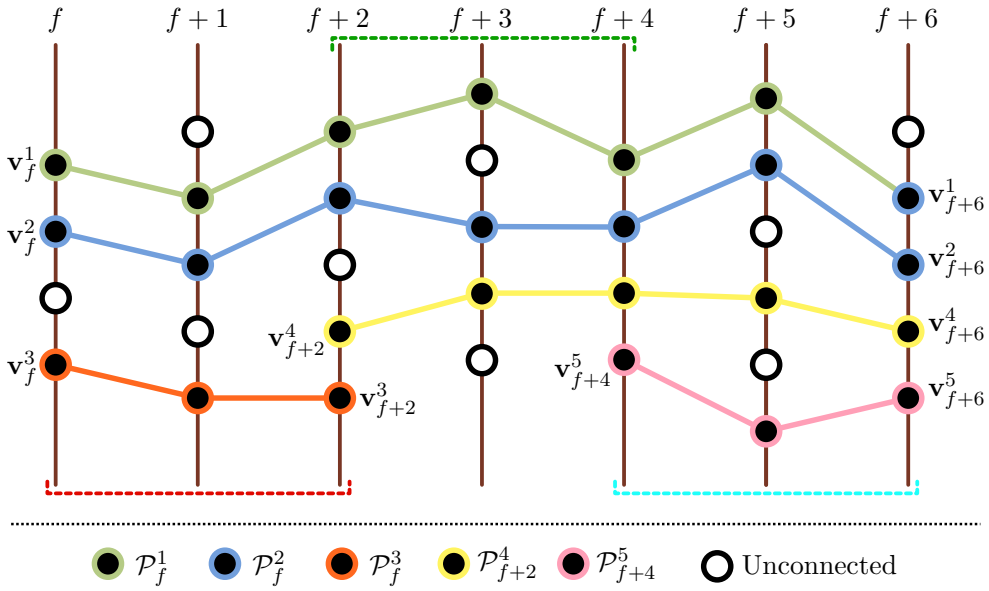

Figure 3.3: *Example of a trellis for local region tracking. Each node (circle) represents a feature vector* $\mathbf{v}_\beta^\alpha$ *for a local region, where* β *is the image at which the region was detected, and* α *is the track index. Three overlapping frame windows are shown* $(f, f+1, f+2)$,$(f+2, f+3, f+4)$, *and* $(f+4, f+5, f+6)$.

Local palettes are created by modelling the spatiotemporal volume associated with each 3-frame path $\mathcal{P}_1^i$ as a GMM (Gaussian Mixture Model). Because the neighbourhoods associated with each vector in $\mathcal{P}_1^i$ is variable, a fixed number of pixel samples (5043) are drawn randomly from these regions and that set is used for modelling. This allows a balance in computational load for GMM parameter estimation. The p.d.f. of the colour samples $\mathbf{c}_i$ generated from $\mathcal{P}_1^i$ is therefore as follows.

$$f^i{}_{K_i}(\mathbf{c}_i) = \sum_{j=1}^{K_i} \Pi_j^i \, N\left(\mathbf{c}_i; \theta_j^i\right) \tag{3.1}$$

Where θ_j^i consists of a mean μ_j^i, covariance $\mathbf{R}_j^i$, mixing weight Π_j^i for each of K_i components of the GMM. These parameters are estimated via Expectation Maximisation using an implementation by Bouman [22], based on ideas from Redner [98] and the number of mixture components K_i is estimated using the minimum description length (MDL) criteria of Rissanen [101]. The mean components μ_j^i, $\{j = 1, 2, \ldots, K_i\}$ of this GMM constitute the local palette for the tracked region along path $\mathcal{P}_1^i$. Each track yields K_i mixture components $\left(\mu_1^i, \mathbf{R}_1^i\right), \left(\mu_2^i, \mathbf{R}_2^i\right), \ldots, \left(\mu_{K_i}^i, \mathbf{R}_{K_i}^i\right)$.

The global palette is then extracted from the local palettes by MS clustering ([5 2 2] kernel bandwidth (see section 4.1.3 for definition) for $[Y\ U\ V]$) of all the local palettes. Each of the resulting G clusters then contains a set of associated mixture components $\left(\mu_j^g, \mathbf{R}_j^g\right)$: $j = \{1, 2, \ldots, K_g\}$, where $g = \{1, 2, \ldots, G\}$, and K_g is the number local palettes in each cluster. This MS step in effect shows which of the local palettes are similar. Hence each of these clusters

can be represented by a single Gaussian with parameters $(\mu_g, \mathbf{R}_g)$ estimated as follows.

$$\mu_g = \frac{1}{K_g}\sum_{j=1}^{K_g}\mu_j^g, \quad \mathbf{R}_g = \frac{1}{K_g}\sum_{j=1}^{K_g}\left\{\mathbf{R}_j^g + \mu_j^g\mu_j^{gT}\right\} - \mu_g{\mu_g}^T \tag{3.2}$$

These G models now form the global palette. A global palette colour g is then defined as having parameters $\theta_g = (\mu_g, \mathbf{R}_g)$.

Consider that a pixel site (x, y) has an observed colour vector ξ. An ML estimate for the quantised colour g to be assigned to that site is generated by choosing g to maximize the log energy $e_g(\xi)$ as follows.

$$e_g(\xi) = -ln\left\{(2\pi)^{3/2}\left|\mathbf{R}_g\right|^{1/2}\right\} - 0.5\left(\xi - \mu_g\right)^T{\mathbf{R}_g}^{-1}\left(\xi - \mu_g\right) \tag{3.3}$$

Using all the global palette colours can produce very sparse colour vectors $\mathbf{c}$, since only a few of these colours occur in each local region. To further quantise this space, colours that are not used often from this palette can be discarded, and the retained colour space is used for the final quantised global palette. To do this, each spatiotemporal region associated with a path $\mathcal{P}_1^i$ votes for a single colour g_i. That colour g_i is the colour that best models the entire patch region, i.e. g_i minimises the sum of $e_g(\xi)$ over the whole patch. After all the paths are processed in this way, all colours which have no votes are discarded.

We will report later our dense and sparse segmentation results for three sequence called *Triniman*, *Artbeats-SP128* and *Calendar and Mobile* sequences. The *Triniman* sequence is a dynamic outdoor scene recorded with a hand-held camcorder. The *Artbeats-SP128* sequence is a scene from an American football game. The camera is static for this sequence. *Calendar and Mobile* is a well known sequence which will be used extensively through out this thesis for reporting our results.

Fig. 3.4 shows examples of quantised frames for these three sequences. The number of colours in the global palettes for the *Triniman*, *Artbeats-SP128* and *Calendar and Mobile* sequences are 241, 181 and 127 respectively.

3.2.2 Tracking

It is now required to extend the seed paths into long term tracks using the feature vectors associated with each interest point. An adaptation of the Viterbi algorithm [43] proposed by Pitie [90] is used here. Figure 3.3 shows an example of a typical initial Viterbi trellis at this stage. The states in the Viterbi algorithm at each frame are all the feature vectors of the detected interest points which include those that are already associated with paths. Note that, it is required to extend the tracks both forward and backward temporally. This bi-directional temporal extension is achieved by first extending all tracks forward in time, and then reversing the Viterbi trellis and extend the tracks forward again. The discussion that follows is therefore applicable to both directions.

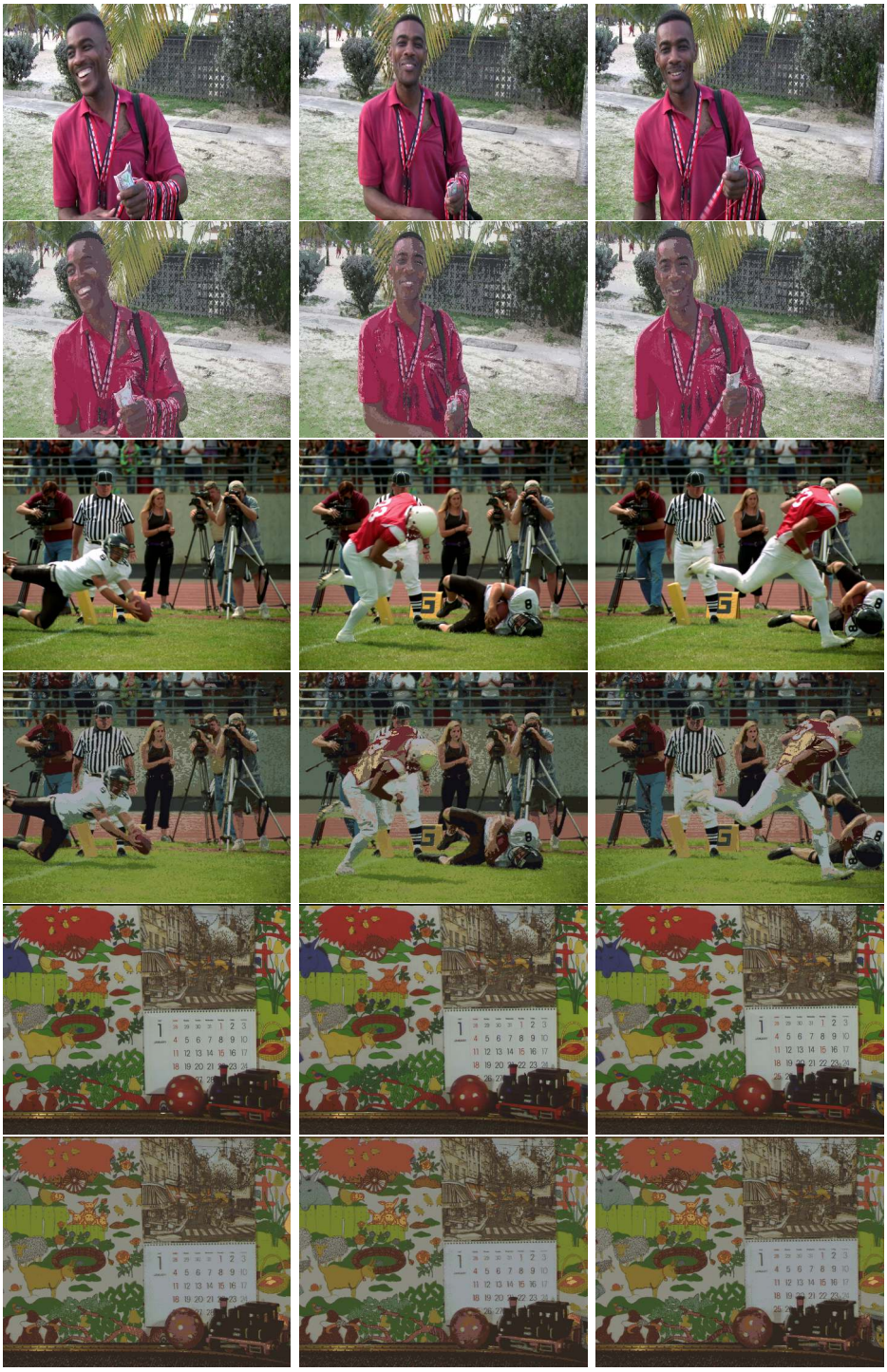

Figure 3.4: *Examples of quantised frames from the* ***Triniman****,* ***Artbeats-SP128*** *and* ***Calendar and Mobile*** *sequences.* ***Top and second rows****: Frames 7, 57 and 91 from the Triniman sequence, where the original and quantised versions of these frames are shown in the* ***top*** *and* ***second*** *rows respectively.* ***Third and fourth rows****: Frames 20, 62 and 95 from the Artbeats-SP128 sequence, where the original and quantised versions of these frames are shown in the* ***third*** *and* ***fourth*** *rows respectively.* ***Fifth and bottom rows****: Frames 5, 13 and 23 from the Calendar and Mobile sequence, where the original and quantised versions of these frames are shown in the* ***fifth*** *and* ***bottom*** *rows respectively.*

The Viterbi algorithm solves the problem of finding the best paths (in a minimum energy sense) connecting the states in the trellis i.e. solving our point tracking problem. The energy associated with each state (or point) in each frame depends both on the feature vector itself as well as an energy associated with the transition between states in consecutive frames. This framework is well understood and more information can be found here [43]. The essential requirement however is to define a transition energy between the last state (point) of the track $\mathbf{v}_\tau^i$, and the N_u candidate states $\mathbf{v}_{\tau+1}^u : u = \{1, 2, \ldots, N_u\}$, in image $\tau + 1$. Here N_u is the number of candidate states in the next frame $\tau + 1$. Only feature vectors that are at the start of a track or not associated with any track are considered to be candidates in image $\tau + 1$. This transition energy is defined as $\phi_{\tau,\tau+1}^i\left(\mathbf{v}_{\tau+1}^u\right)$, and is expressed below.

$$\phi_{\tau,\tau+1}^i\left(\mathbf{v}_{\tau+1}^u\right) = \lambda M\left(\mathbf{v}_{\tau+1}^u;\ \mathbf{v}_{\tau-2,\tau-1,\tau}^i\right) + O\left(\mathbf{v}_{\tau+1}^u;\ \mathbf{v}_{\tau-2,\tau-1,\tau}^i\right) \tag{3.4}$$

$M\left(\mathbf{v}_{\tau+1}^u;\ \mathbf{v}_{\tau-2,\tau-1,\tau}^i\right)$ is a motion smoothness constraint, and $O\left(\mathbf{v}_{\tau+1}^u;\ \mathbf{v}_{\tau-2,\tau-1,\tau}^i\right)$ is dependent on the colour, scale, and descriptor information of the track. λ is a weighting between these energy terms.

3.2.3 Motion smoothness

When estimating future state spatial locations, it is assumed that tracks in very close proximity to each other change their speeds at the same rate. Hence some spatial smoothness constraint is important. For a track $\mathcal{P}_f^i$, with the feature vector at the end of the track being $\mathbf{v}_\tau^i$, the last three feature vectors $\left\{\mathbf{v}_{\tau-2}^i, \mathbf{v}_{\tau-1}^i, \mathbf{v}_\tau^i\right\}$ are used to predict the next possible feature vector in image $\tau + 1$ to be added to the track. The motion constraint $M\left(\mathbf{v}_{\tau+1}^c;\ \mathbf{v}_{\tau-2,\tau-1,\tau}^i\right)$ in (3.4) penalizes candidate spatial locations $\mathbf{x}_{\tau+1}^u$ according to their distance from the predicted spatial location $\tilde{\mathbf{x}}_{\tau+1}^i$ of the track $\mathcal{P}_f^i$ in image $\tau + 1$. The predicted spatial location $\tilde{\mathbf{x}}_{\tau+1}^i$, is estimated using the last known location $\mathbf{x}_\tau^i$, and displacement $\mathbf{x}_\tau^i - \mathbf{x}_{\tau-1}^i$ of track $\mathcal{P}_f^i$, as follows

$$\tilde{\mathbf{x}}_{\tau+1}^i \quad = \quad \mathbf{x}_\tau^i + \mathbf{A}_m^i\left(\mathbf{x}_\tau^i - \mathbf{x}_{\tau-1}^i\right) + \eta \tag{3.5}$$

Here $\mathbf{A}_m^i = \begin{pmatrix} \alpha_x^m & 0 \\ 0 & \alpha_y^m \end{pmatrix}$ is a displacement magnitude gain matrix, and $\eta \sim \mathcal{N}(0, \mathbf{R}_i)$. The displacement gain matrix and noise covariance are estimated by using the known seed paths up to the current frame.

A candidate $\mathbf{A}_j$ is generated for each path $\mathcal{P}_f^j : j = \{1, 2, \ldots, J\}$ that exist over the images $\tau - 2, \tau - 1, \tau$, where J is the number of such paths. A weighted least square estimate is used to estimate α_x^j and α_y^j, which incorporates path information from neighbouring paths as follows.

$$\alpha_x^j = \frac{\sum_k \left\{w_{(j,k)}^2 |x_{\tau-2}^k - x_{\tau-1}^k||x_{\tau-1}^k - x_\tau^k|\right\}}{\sum_k \left\{w_{(j,k)}^2 |x_{\tau-2}^k - x_{\tau-1}^k|^2\right\}},\ \alpha_y^j = \frac{\sum_k \left\{w_{(j,k)}^2 |y_{\tau-2}^k - y_{\tau-1}^k||y_{\tau-1}^k - y_\tau^k|\right\}}{\sum_k \left\{w_{(j,k)}^2 |y_{\tau-2}^k - y_{\tau-1}^k|^2\right\}} \tag{3.6}$$

Where $w_{(j,k)} = \frac{1}{\sqrt{2\pi}\sigma_s} exp\left(-0.5\frac{\delta^T\delta}{2\sigma_s^2}\right)$, $\delta = \left(\mathbf{x}_{\tau-2}^k + \mathbf{x}_{\tau-1}^k + \mathbf{x}_\tau^k\right)/3 - \left(\mathbf{x}_{\tau-2}^j + \mathbf{x}_{\tau-1}^j + \mathbf{x}_\tau^j\right)/3$, and $k = \{1, 2, \ldots, J\}$. A constant spatial variance $\sigma_s = 50$ is used.

Every track is now associated with a particular A_m^i by choosing that A_j from amongst the candidate set that minimises the prediction error defined as follows.

$$\epsilon_j = |\mathbf{x}_{\tau-1}^j - \mathbf{x}_\tau^j| - \mathbf{A}_j|\mathbf{x}_{\tau-2}^j - \mathbf{x}_{\tau-1}^j| \tag{3.7}$$

The covariance matrix of those prediction errors is used as an estimate for $\mathbf{R}_i$. Hence the motion smoothness constraint can be written as

$$M\left(\mathbf{v}_{\tau+1}^u;\ \mathbf{v}_{\tau-2,\tau-1,\tau}^i\right) = -ln\left\{(2\pi)\left|\mathbf{R}_i\right|^{1/2}\right\} - 0.5\left(\mathbf{x}_{\tau+1}^u - \tilde{\mathbf{x}}_{\tau+1}^i\right)^T \mathbf{R}_i{}^{-1}\left(\mathbf{x}_{\tau+1}^u - \tilde{\mathbf{x}}_{\tau+1}^i\right) \tag{3.8}$$

We are more confident of our motion energy term when a track is moving at a constant speed. Therefore, the motion constraint energy weighting λ in (3.4) is made to be inversely proportional the maximum of the displacement gains α_x^m and α_y^m in both spatial dimensions.

3.2.4 Appearance constraint

The energy associated with the feature vector itself is an appearance energy that enforces the constraint that the appearance of the region in the next frame should be similar to the appearance of the regions along the previous frames in a path. By modelling the scale, SIFT descriptor and colour distributions of the region as a Gaussian with variance $\mathbf{R}_O$ and mean μ_0, the appearance energy is therefore as follows.

$$O\left(\mathbf{v}_{\tau+1}^c;\ \mathbf{v}_{\tau-2,\tau-1,\tau}^i\right) = -ln\left\{(2\pi)^{M/2}\left|\mathbf{R}_O\right|^{1/2}\right\} - 0.5\left(\kappa_c - \mu_O\right)^T \mathbf{R}_O{}^{-1}\left(\kappa_c - \mu_O\right) \tag{3.9}$$

$$\mathbf{Q} = \begin{pmatrix} \mathbf{s}_{\tau-2} & \mathbf{s}_{\tau-1} & \mathbf{s}_\tau \\ \mathbf{g}_{\tau-2} & \mathbf{g}_{\tau-1} & \mathbf{g}_\tau \\ \mathbf{c}_{\tau-2} & \mathbf{c}_{\tau-1} & \mathbf{c}_\tau \end{pmatrix},\ \mu_O = \frac{1}{3}\begin{pmatrix} \mathbf{s}_{\tau-2}+\mathbf{s}_{\tau-1}+\mathbf{s}_\tau \\ \mathbf{g}_{\tau-2}+\mathbf{g}_{\tau-1}+\mathbf{g}_\tau \\ \mathbf{c}_{\tau-2}+\mathbf{c}_{\tau-1}+\mathbf{c}_\tau \end{pmatrix},\ \mathbf{R}_O = \frac{1}{3}\mathbf{Q}\mathbf{Q}^T - \mu_O\mu_O{}^T$$

Where the parameters $\mathbf{R}_O$ and mean μ_0 are measured using the values of the feature vector along the path in the three previous frames, and $\kappa_u = \left(\mathbf{s}_{\tau+1}^v\ \mathbf{g}_{\tau+1}^v\ \mathbf{c}_{\tau+1}^v\right)$, is the scale, SIFT descriptor, and colour vector for the candidate state $\mathbf{v}_{\tau+1}^u$. The off-diagonal elements of the $M \times M$ covariance matrix $\mathbf{R}_O$ are set to zero, assuming independence amongst the rows of $\mathbf{Q}$.

3.3 Viterbi SIFT Tracks

Recall that we estimate seed tracks by performing Mean shift clustering of the feature descriptors over three frame windows. These seed tracks are then extended using the Viterbi algorithm, where transition energies are based on motion and appearance constraints.

Figs 3.5, 3.6 and 3.7 show examples of the Viterbi tracks for the *Triniman*, *Artbeats-SP128* and *Calendar and Mobile* sequences respectively. Here each track has a different colour, and the spatial location at the current frame is indicated with a 'dot'. The motion history of a track is also shown as a line that extends from its 'dot'. Note that this motion history is only shown over 5 frames for clearer illustrations.

In general it may be observed that for all three sequences, image regions with relatively low textures do not contain any tracks. Recall that we are using SIFT features, and these features can not be extracted from low texture image regions. Note that instead of SIFT, our tracking framework could easily use Maximally Stable Extremal Region (MSER) [44] features in order to produce tracks in low texture image regions.

The SIFT features are detected at various image scale. The scale of a feature determines the pixels in its neighbourhood of influence. Some 'large' scale features may be influenced by image material corresponding to both the foreground and background respectively. Hence the centers (x, y) of these features may be located in the background, but the motion of the track it belongs to is mainly influenced by the foreground. Examples of these tracks that follow the motion of the foreground, but are located in the background are highlighted with the white ovals in figs. 3.5, 3.6 and 3.7.

For the *Artbeats-SP128* sequence the camera is static, hence the tracks corresponding to the background (fig. 3.5) do not move away from their initial spatial positions (no motion history). However the football players in the foreground have several non-stationary tracks associated with them. The *bottom* row of fig. 3.5 shows a zoom on these tracks in the foreground image regions. Here the tracks follow the local motion in the image plane of the foreground objects.

The *Triniman* sequence was recorded with a hand-held camera. Here the background tracks (fig. 3.6) are influenced by the motion of the camera. The tracks for the face, upper body and left arm of the actor in the foreground are influenced by the respective local motion of these body parts.

The camera in the well known *Calendar and Mobile* is panning from left to right in the image plane. The background tracks here (fig. 3.7) are also influenced by the motion of the camera. A zoom on some of the tracks corresponding to the foreground are shown in the *bottom* row of fig. 3.7. It may be observed that there are tracks that follow the motions of the toy train, calendar and ball.

3.4 Performance Evaluation

The performance of our Viterbi tracker is compared to a standard KLT tracker implementation [18] derived from the work of [55, 113]. For the KLT tracker, the maximum number of features to track must be specified. Therefore, we use the total number of tracks in the first frame of the Viterbi tracker to specify the maximum number of tracks for the KLT tracker. So both trackers start with the same number of tracked features. Only the tracks from the first frame of each sequence are used for analysis since they potentially exist over the entire sequence. Comparing these tracks can reveal how tracking errors are propagated through time.

One natural sequence and three synthetic sequences of known affine transformations are used for testing, and three frames of each are shown in figure 3.8.

Two criteria are used to assess tracking performance in the synthetic sequences. The syn-

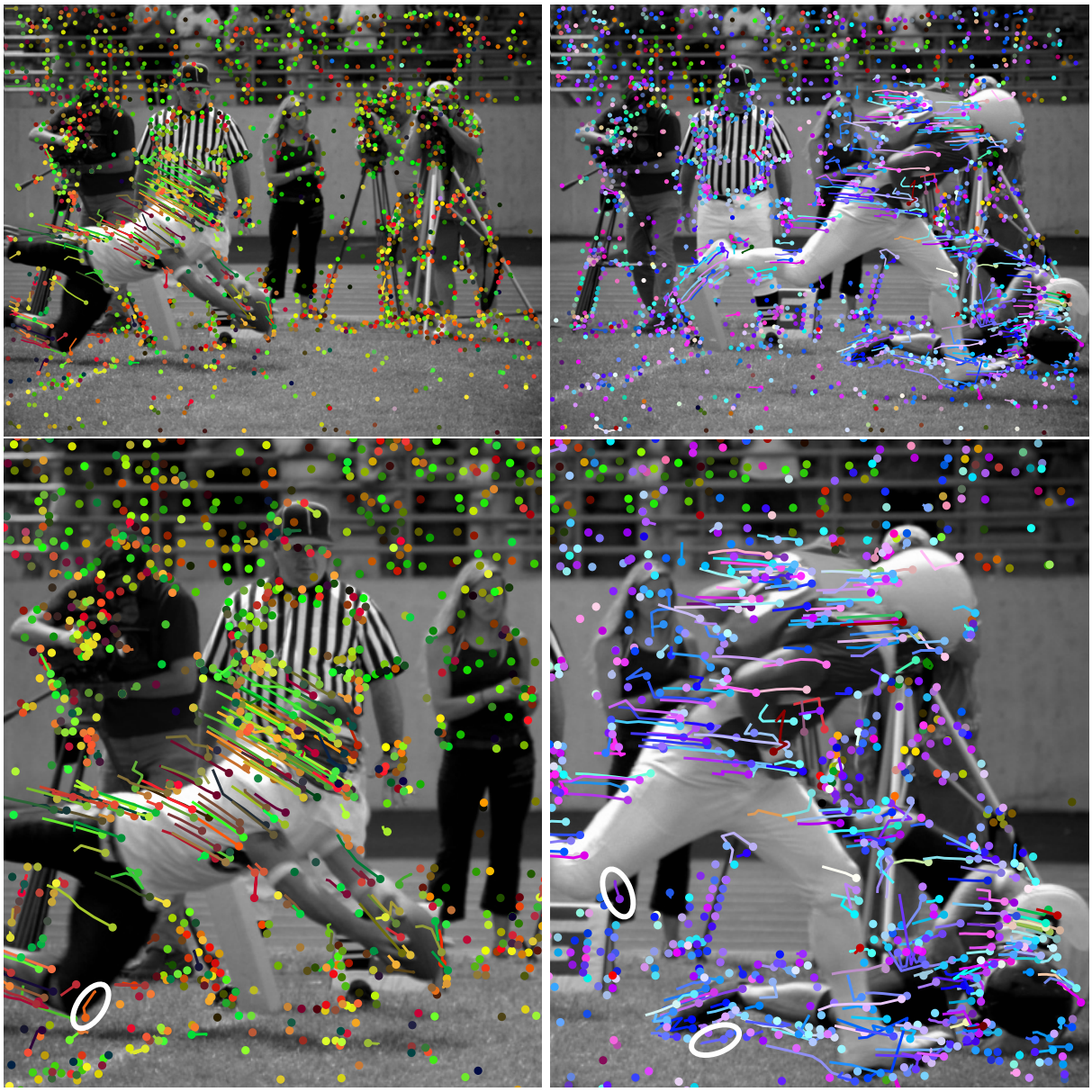

Figure 3.5: *The feature tracks generated by the* ***Viterbi*** *tracker for frames 18 and 91 of the* ***Artbeats-SP128*** *sequence.* ***Top row****: All the tracks for both frames are shown.* ***Bottom row****: A zoom on the tracks corresponding to the football players in the foreground motion. The spatial locations of the tracks at frame f are indicated with coloured dots, where each track has a different colour. The motion history of the tracks are shown as the lines that extend from these dots. The* ***white ovals*** *highlight tracks for which their motion is determined by a foreground object, but the points along this track are spatial located in the background.*

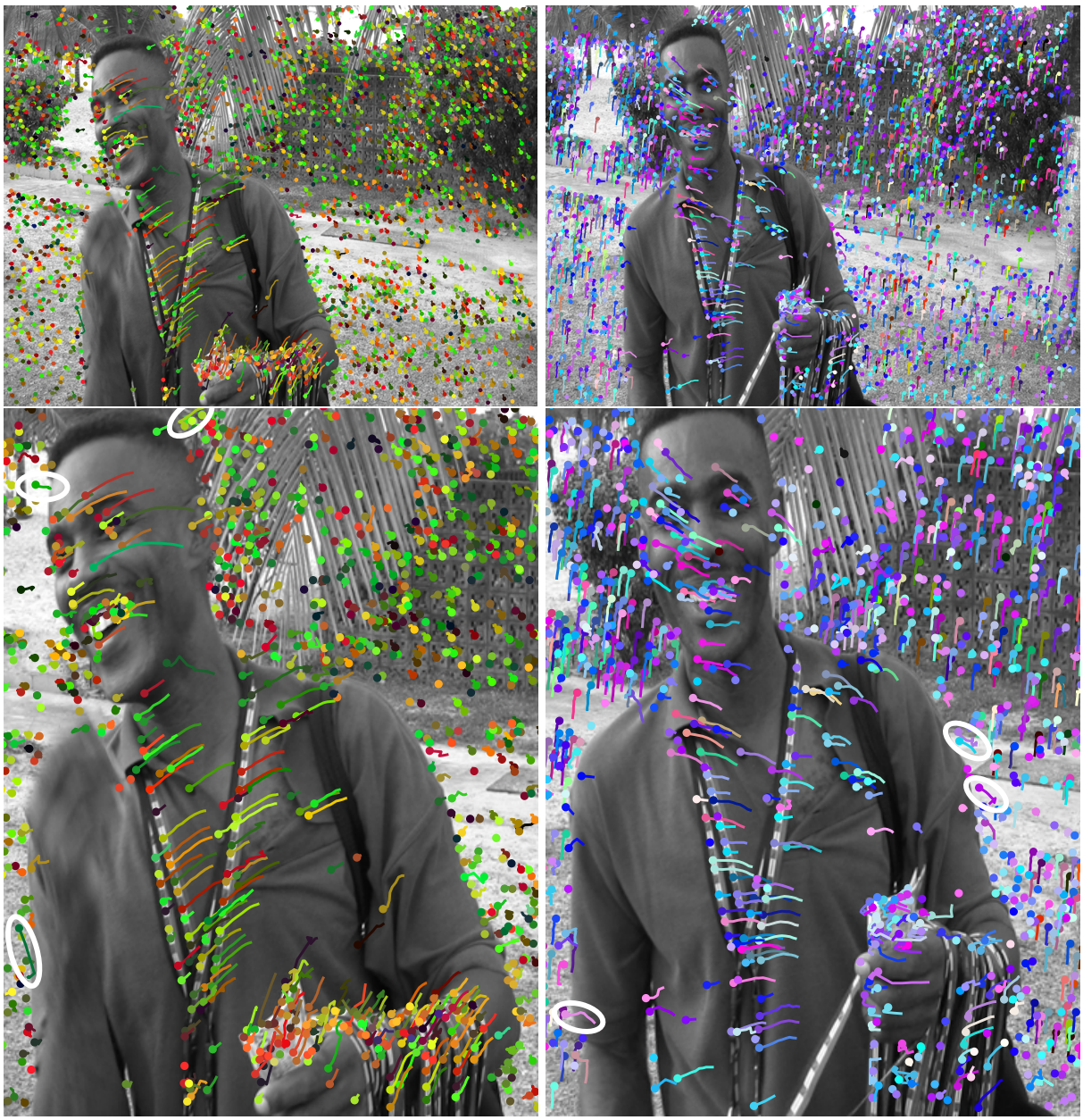

Figure 3.6: *The feature tracks generated by the* ***Viterbi*** *tracker for frames 15 and 88 of the* ***Triniman*** *sequence.* ***Top row****: All the tracks for both frames are shown.* ***Bottom row****: A zoom on some of the tracks corresponding to the actor in the foreground. The spatial locations of the tracks at frame f are indicated with coloured dots, where each track has a different colour. The motion history of the tracks are shown as the lines that extend from these dots. The* ***white ovals*** *highlight tracks for which their motion is determined by a foreground object, but the points along this track are spatial located in the background.*

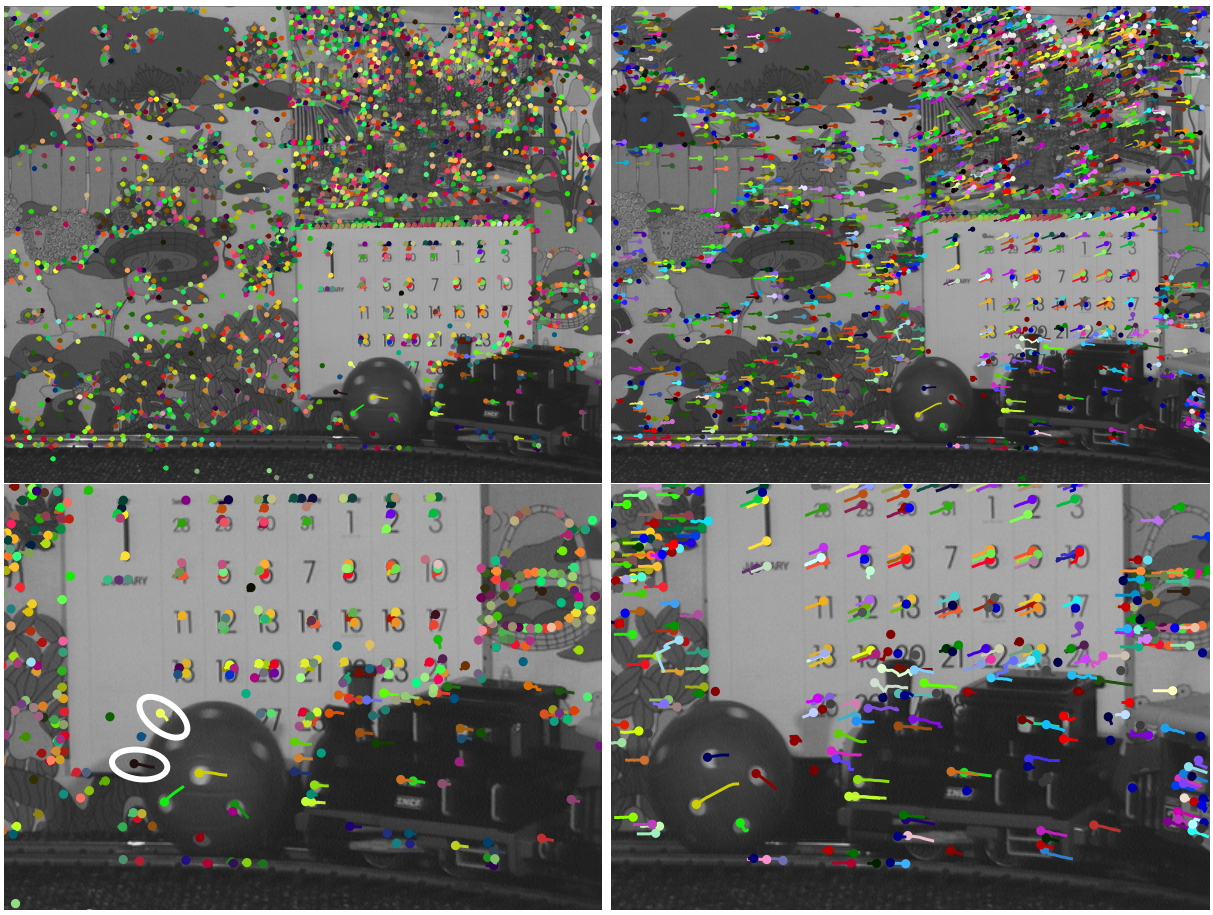

Figure 3.7: *The feature tracks generated by the* ***Viterbi*** *tracker for frames 5 and 25 of the* ***Calendar and Mobile*** *sequence.* ***Top row****: All the tracks for both frames are shown.* ***Bottom row****: A zoom on some of the tracks corresponding to the ball, train and calendar in the foreground. The spatial locations of the tracks at frame f are indicated with coloured dots, where each track has a different colour. The motion history of the tracks are shown as the lines that extend from these dots. The* ***white ovals*** *highlight tracks for which their motion is determined by a foreground object, but the points along this track are spatial located in the background.*

thetic sequences are generated using known affine transformations; zoom, rotation and translation. Then the spatial positions of the tracks $\mathbf{x}_f^i$, reported by both trackers are used to calculate a least squares estimate for the affine transformation parameters. The errors in estimation of these parameters provides one measure of the usefulness of the tracks. The second criterion used is to compare the actual expected spatial locations of the tracks at frame f given their starting locations $\mathbf{x}_1^i$, and the locations reported by the trackers $\mathbf{x}_f^i$. This is referred as the end point error. The mean percentage end point error over all the tracks at frame f is defined as ϵ_f^p.

The parameters used for the synthetic sequences (zoom z_f, rotational θ_f, and translational

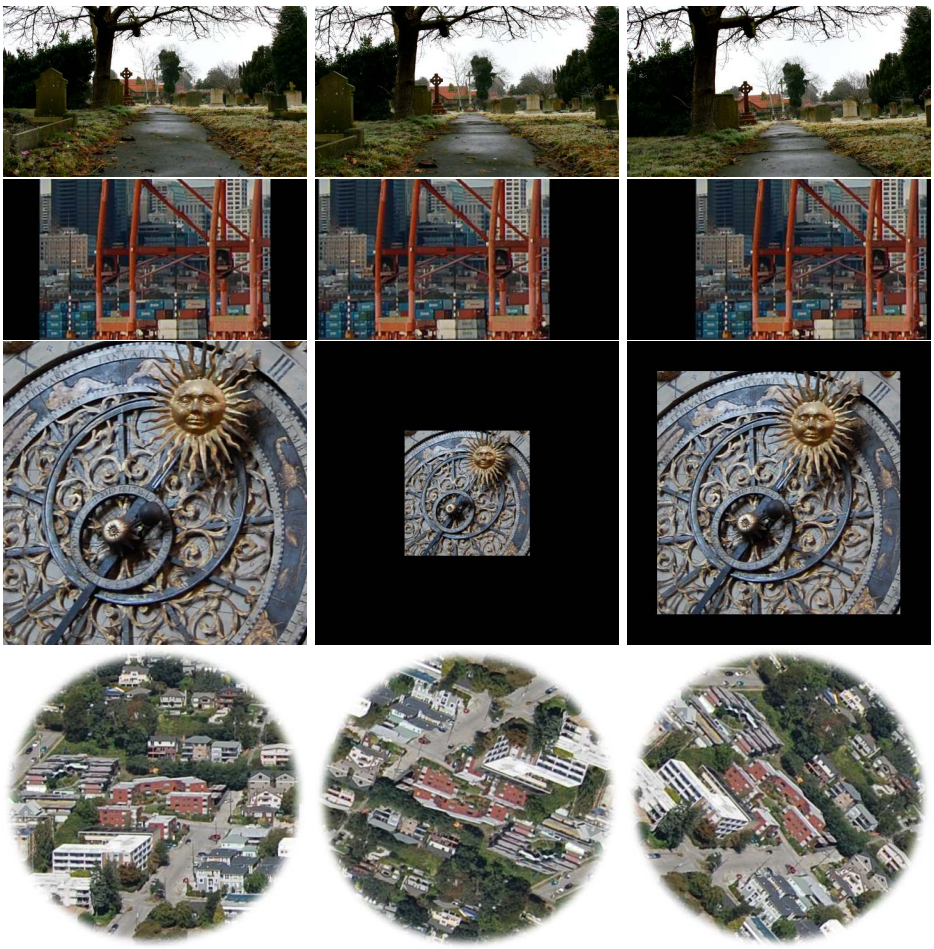

Figure 3.8: *The sequences used to compare the performance of the KLT and Viterbi trackers.* ***Top row****: Frames 1, 11 and 19 in the* ***graveyard*** *(*1920×1080*) sequence. This sequence is 19 frames of a natural outdoor scene.* ***Second row****: Frames 1, 46 and 208 in the* ***Dock*** *(*416×220*) sequence.* ***Third row****: Frames 1, 115 and 202 in the* ***clock*** *(*475×475*) sequence.* ***Bottom row****: Frames 1, 112 and 218 in the* ***house*** *(*419×419*) sequence. The* ***clock****,* ***house*** *and* ***dock*** *sequences are each 251 frames synthesized with known Affine transformations.*

t_f) vary with time and are given below.

$$z_f = 0.5\left\{1 + z_{min} + (1 - z_{min})\, cos\left(\omega \Delta t\,(f-1)\right)\right\} \tag{3.10}$$

$$\theta_f = (f-1)\,\omega\Delta t, \quad t_f = \begin{pmatrix} sin(\omega\Delta t(f-1)) & 0 \\ 0 & 0 \end{pmatrix} \mathbf{d}_{max} \tag{3.11}$$

Where $z_{min} = 0.4$ is the minimum zoom level, $\mathbf{d}_{max} = (50\ 0)^T$ is the maximum translation, $\Delta t = 40ms$, and $\omega = 0.6283 rad/s$.

Each sequence exhibits just one effect i.e. zoom for the *clock*, rotation for the *house* and translation for the *dock* sequence. The corresponding least squares estimates are $\hat{z}_f$, $\hat{\theta}_f$, $\hat{t}_f$ and

	Parameter		End point	
	KLT	Viterbi	KLT	Viterbi
mean	0.79	0.04	2.22	0.77
std	0.28	0.01	3.24	0.32

Table 3.1: *The overall parameter estimation and end point percentage errors for all three synthetic sequence.*

are derived in the usual manner using all the estimated feature point locations $\mathbf{x}_1^i$ and $\mathbf{x}_f^i$ in frames 1 and f respectively. These estimates are given below.

$$\hat{\theta}_f = atan\left\{\frac{\sum_i \det\left([\mathbf{x}_1^i\ \mathbf{x}_f^i]\right)}{\sum_i \left({\mathbf{x}_1^i}^T \mathbf{x}_f^i\right)}\right\}, \tag{3.12}$$

$$\hat{z}_f = \frac{\sum_i \left({\mathbf{x}_1^i}^T \mathbf{x}_f^i\right)}{\sum_i \left({\mathbf{x}_1^i}^T \mathbf{x}_1^i\right)}, \tag{3.13}$$

$$\hat{t}_f = \frac{1}{P}\sum_{i=1}^{P}\left(\mathbf{x}_f^i - \mathbf{x}_1^i\right) \tag{3.14}$$

Where $\det(\mathbf{A})$ is the determinant of the matrix $\mathbf{A}$, and P is the number of tracks that exist at frame f.

The percentage errors in the estimation of the zoom, rotation, and translation parameters at frame f are defined as $\epsilon_f^{z,z_{min}}$, ϵ_f^{θ}, and ϵ_f^{t} respectively.

3.4.1 Results and Discussion

Figure 3.9 shows performance for both trackers using the synthetic sequences. The left column plots the percentage error in estimating the global motion parameters for both trackers. In red is the KLT while in black is our Viterbi tracker. The right column shows the corresponding percentage end point error for each tracker. An overall summary of the performance of the trackers for all three sequences in shown in table 3.1. These results show that the Viterbi tracker produces much more accurate tracks in general. Lower end point errors are achieved, with relatively smaller standard deviations for these errors. This indicates that the Viterbi tracks are more dependable locally. Considering the rotational sequence (house), the standard deviations of the end point errors for the KLT tracker increase with time. This indicates that there is significant track drift for this tracker when the sequence undergoes rotational transformation.

The zoom sequence is the most challenging to track because of the significant loss of local features due to scale changes with time. In our experiment the clock sequence was zoomed from 100% to 40% and back to 100%. For this sequence, the KLT tracker had a mean end

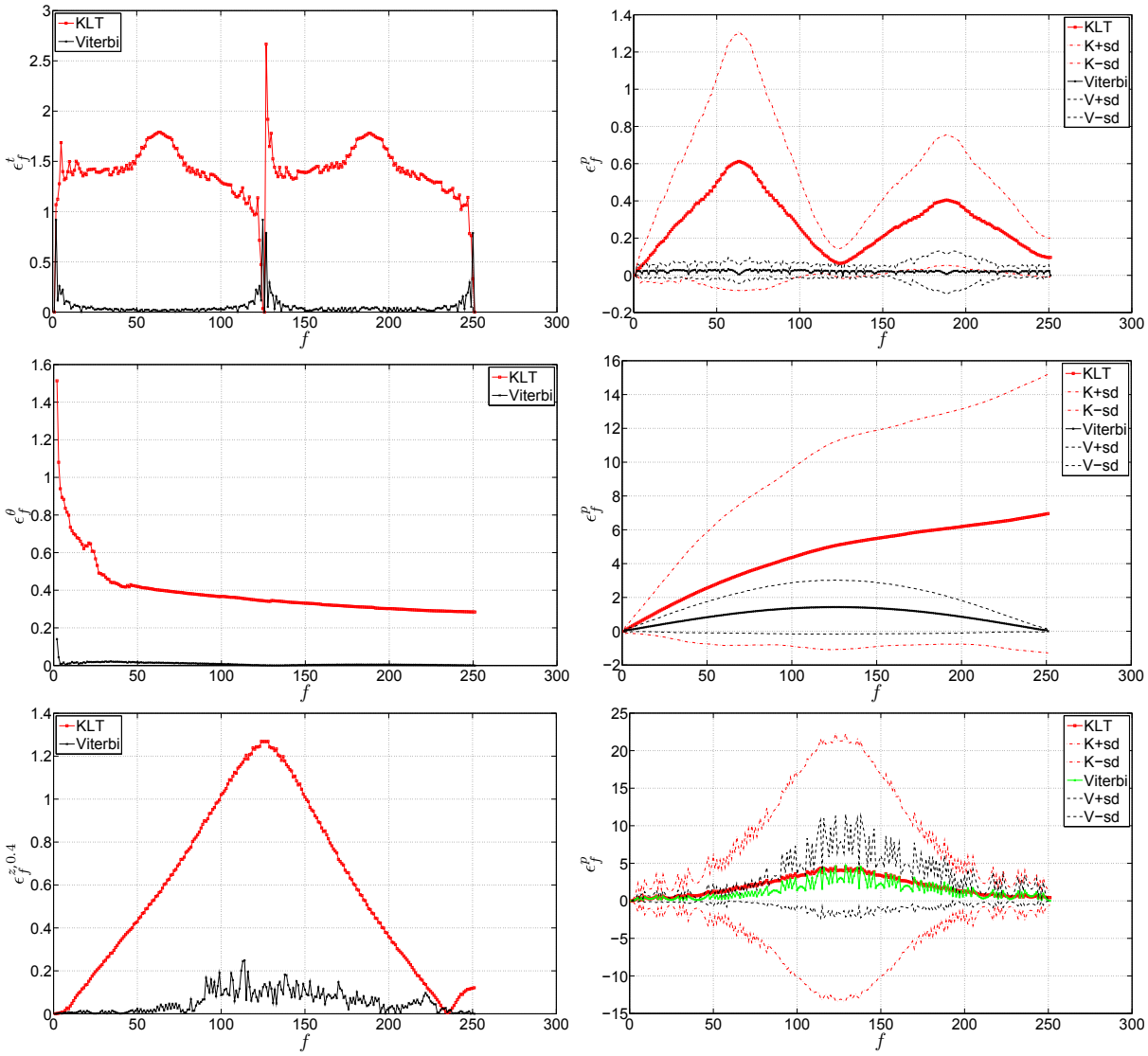

Figure 3.9: *The **left column** shows the global performances of the trackers for the translational (top), rotational (middle), and zoom (bottom) sequences. The corresponding local performances (endpoint error) for these sequences is shown in the **right column**, where $\epsilon_f^p \pm$ one standard deviation is shown as K+sd and K-sd, V+sd and V-sd, for the KLT and Viterbi trackers respectively.*

point error of 1.92% with a std (standard deviation) of 0.82%. The corresponding error for the Viterbi tracker was 1.38% with a std of 0.08%. This is a substantial improvement. In addition, the average length of the tracks for each sequence shown in figure 3.8 using the KLT tracker was 10, 68, 251, and 95 respectively, while for the Viterbi tracker it was 9, 33, 132, and 179 respectively. Even though the KLT tracker maintains much longer tracks for the zoom (*clock*) and rotation (*house*) sequences, the parameter and end point errors are relatively large

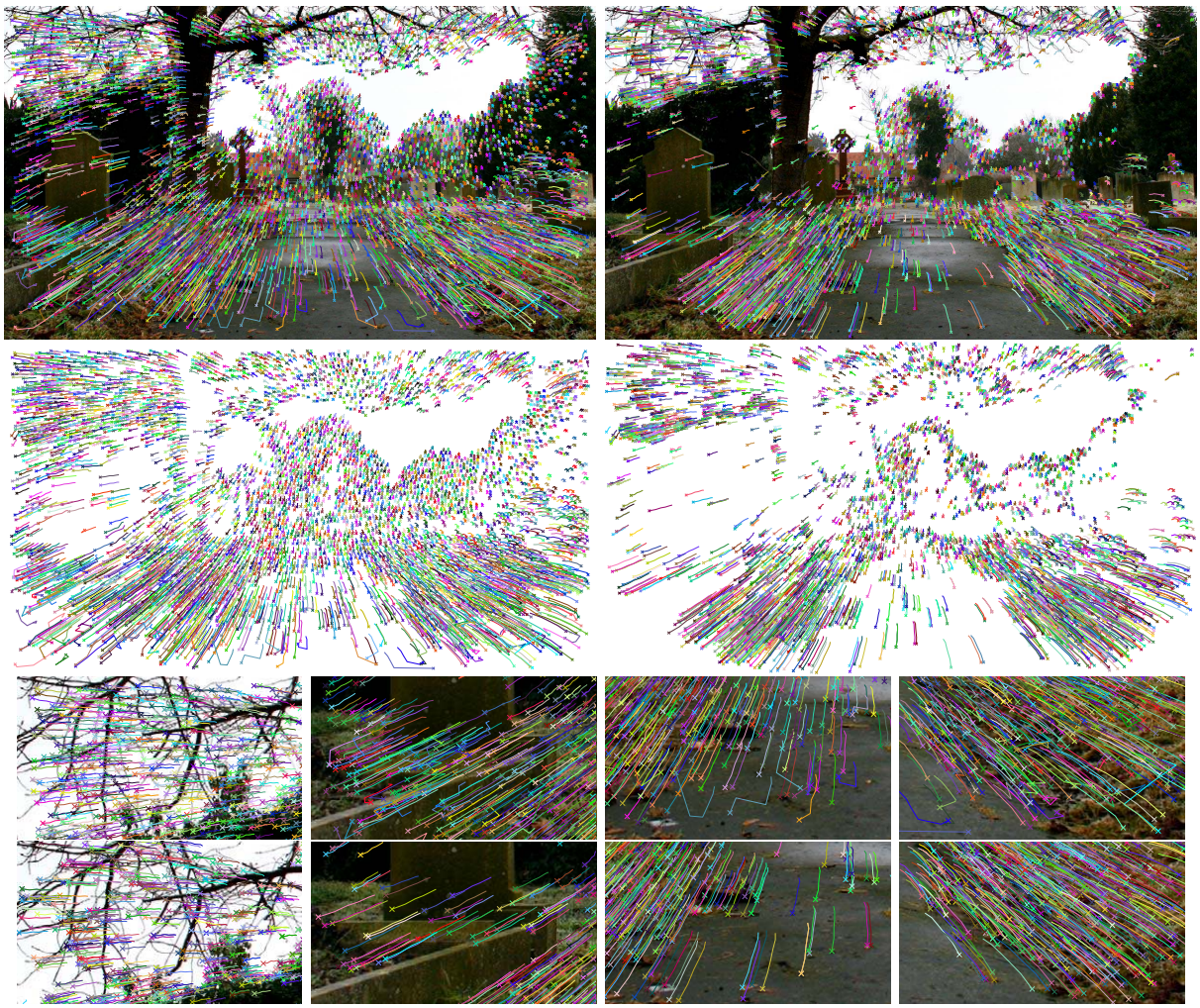

Figure 3.10: ***Top and second rows**: The tracks at frame 6 in the **graveyard** sequence produced by the **KLT** (left) and **Viterbi** (right) trackers. The tracks are superimposed on a white background in the **second** row. **Third and bottom rows**: Selected cropped image regions from the images in the **top** row. The **third and bottom** rows are the tracks produced by the KLT and Viterbi trackers respectively. **Column 1**: Trees - strong corner features. **Column 2**: Tomb - edges in direction of motion and flat image regions. **Column 3**: Pavement - mostly flat image region. **Column 4**: Grass - repetitive texture.*

compared to the Viterbi tracker. In the case of the zoom sequence, where there are rapid local image deformations, the average track lengths are expected to be shorter.

Figure 3.10 shows tracks estimated for the real sequence *graveyard*. It shows the Viterbi tracker in general does well in maintaining accurate tracks in problematic areas such as the *pavement* and *tomb* image regions. The KLT tracker produces more tracks in these areas, but it is obvious that some of them are incorrect. In case of the tomb, the KLT tracker made

errors probably because some of the edges are almost in the direction of motion, introducing an aperture effect. For the pavement, the KLT tracker selected some very poor features to track and eventually lost them on the smooth surface. Birchfield [19] identified that repetitive textures can cause the tracking of a feature to be distracted by similar nearby texture patterns when using a standard KLT tracker. This phenomenon is observed in the grass area for the KLT tracker but not for the Viterbi tracker. The Viterbi tracker produces more tracks for the grass since more interest points can be found this highly textured region. Where there are very strong corner features in an image region, like the tree region (first column of fig. 3.10), the performance of both trackers is roughly the same.

3.5 Summary

A new algorithm has been presented for tracking feature points over time. It is deterministic and exploits the use of a novel feature selection process based on temporal pre-selection and colour quantisation. Note that although we tracked SIFT feature in the experiments reported, our framework can be adapted to track other local features.

The performance of our framework when tracking SIFT features is more reliable than [18]. It is encouraging that the system presented does not suffer from the drift issues as much as the prediction/correction process of the KLT tracker. We expect this is due to the use of candidate tracking points instead of continuous tracking. Future work will consider what aspects of this process bring to the tracking problem. By changing the feature point detector to a Harris corner detector for instance, we would be able to assess the impact of the detector with respect to the KLT. Also unlike SIFT features, corner features are more localized in the image plane. That is, corner correspond to small scale image regions, while SIFT features may correspond to large scale image regions. Recall that we discussed this issue with SIFT features in section 3.3.

It is important to recognise that the Viterbi process itself is little more than a KLT with a restriction to search at candidate points only. Thus the advantages of this process only express when the candidates are unique and considerably fewer than the corners that might be used in the KLT. In the results shown here there were on average 1747 feature per frame in the KLT as compared to 1574 in our system. We shall explore these issues in future work.

4

Sparse Trajectory Segmentation

Given trajectories generated by tracking feature points (e.g. as discussed in chapter 3). This chapter addresses the task of labelling these trajectories according to their 2D image motion. We define this process as *sparse trajectory segmentation*. Here, trajectories with the same label are identified as having similar motion. A group of trajectories with the same label will be referred to as a bundle. The major challenges of trajectory segmentation identified in previous work are determining the number of bundles automatically for an arbitrary sequence, coping with non-rigid motions, and also implementing effective spatial constraints on the trajectories in each bundle.

The number of bundles that are required from the segmentation depends on the application the user has in mind. Applications such as surveillance and human gesture recognition generally require the number of bundles to be roughly the same as the number of objects in the scene. Consider however the application of assigning object labels to the pixels in a video (*dense* segmentation) by using trajectory bundles. Here, the dense segmentation process relies on having the motion of the bundles accurately describing the local motion of the objects in a scene. Therefore, the number of bundles required for this application is dependent on the motion and nature of the objects. Fig. 4.1 illustrates the type of segmentation required for this particular application. Non-rigid objects usually require several bundles to accurately describe their local motions.

As discussed previously (Chapter 2), the sparse trajectory segmentation problem has received some treatment by Fradet [46] and Pundlik [94]. However, there are no attempts in previous work to enforce spatial constraints on the trajectories in each bundle. The trajectories an object

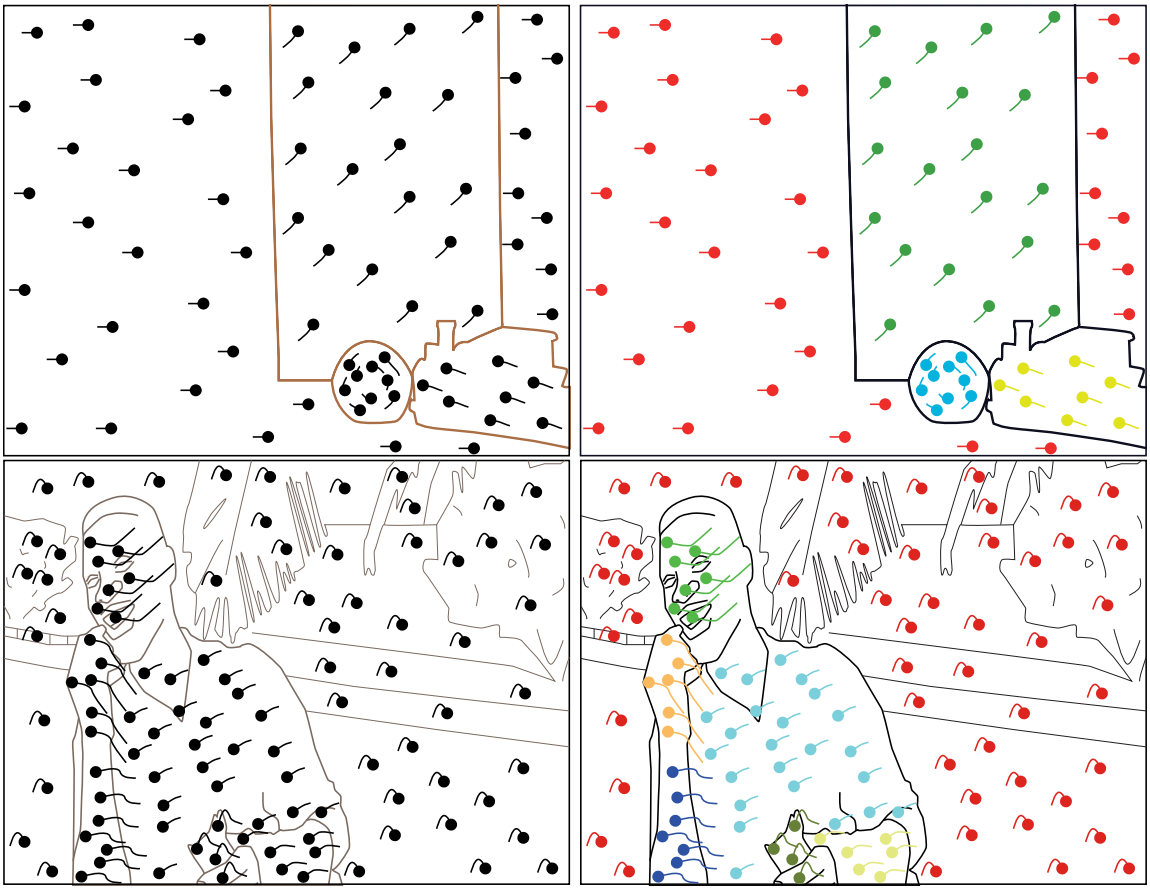

Figure 4.1: *The trajectories in the* ***left column*** *are segmented into the bundles in the* ***right column****. The dot-line represents a trajectory showing the spatial location at the current frame (dots) and the motion history of this trajectory. Each bundle is labelled with a different colour, and contains trajectories that all obey a specific motion model. The* ***top row*** *shows a simple case where the number of bundles and objects is the same due to the geometry, rigidity and motion of the objects. The* ***bottom row*** *shows that non-rigid motion can generate several bundles for each object, where each bundle describes some form of local motion.*

creates throughout a sequence are quite obviously constrained by their spatial location relative to this object. These constraints have been modelled as Markov Random Fields (MRF) in previous work [34, 40, 48, 64, 79–81] when doing pixel based segmentations. However, no attempts have been made at applying the MRF model to trajectory segmentation. The critical issue here is how to define the neighbourhood structure for sparse trajectories of varying lengths.

The proposed trajectory segmentation technique improves on previous work by,

- Introducing spatial constraints on the trajectories in the bundles using a 3D MRF.
- Automatically determining the number of bundles that best represents the local motion of the objects in a scene.

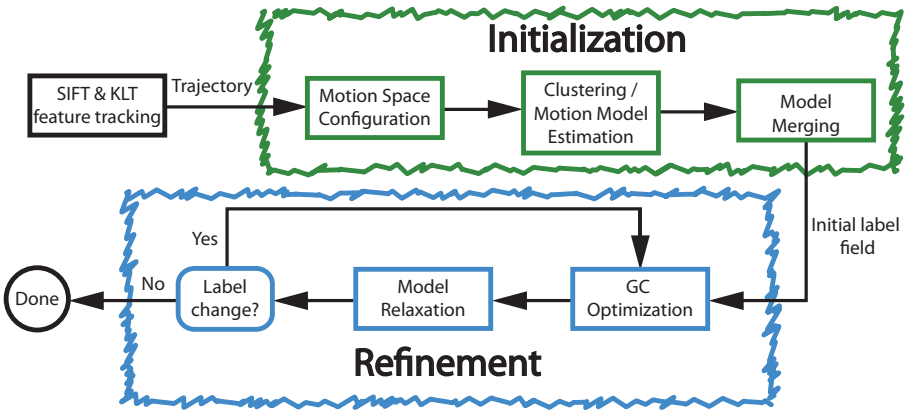

Figure 4.2: *The block diagram for the proposed trajectory segmentation technique. The main stages of* ***initialization*** *and* ***refinement*** *of the trajectory label field are highlighted in green and blue respectively.*

- Improving the segmentation of sequences with non-rigid motions.

Fig. 4.2 show a block diagram for our *sparse* segmentation technique, with the main *initialization* and *refinement* stages highlighted. The *initialization* stage uses ideas similar to previous work to generate an initial guess for the labels of the trajectories. The trajectory motion data is first expressed in a feature space that facilitates for the data to be clustered with Mean Shift [29]. We have found that clustering with MS provides robust and reliable trajectory bundles. To avoid over segmentation the initial motion bundles are modelled with Affine transformations and those with similar models are merged.

The techniques used in the *initialization* step are more effective on sequences with rigid object motions. Hence our *refinement* step is geared towards improving the segmentation results for sequences with non-rigid motions. Fig. 4.2 shows that this subsequent *refinement* stage is iterative. We first construct a Bayesian framework (GC Optimization step) for labelling trajectories given initialization from the previous step. Then the new label field is used to update our knowledge about the motion models in each bundle (Model Relaxation). These two steps iterate until there is no further label changes.

The Bayesian framework combines a motion likelihood with a spatial smoothness prior for trajectory labels based on a 3D MRF. The motion likelihood has two parts. One part models the 'average' motion of a bundle. The other models the sparseness of trajectories in each bundle. The α-*expansion* Graphcut algorithm [26] is used to solve for the MAP estimate of the label field.

In the model relaxation step (fig. 4.2), the new label field is examined to estimate refined models for the bundles. In this step the merging of the bundles is again attempted as their motions may be more coherent. Also a *local region constraint* is applied to the bundles to ensure that each one exclusively represents a single local image region. For example, a bundle cannot

contain trajectories for foreground and background regions - even though they may have similar motion. This requirement is violated in the initial MS clustering step. The lack of spatial constraints in the MS algorithm allows trajectories belonging to spatially separate regions with similar motion to be placed in a single bundle. Those errors introduced in MS clustering are actively corrected with this *local region constraint*.

In this chapter we will discuss the *initialization* stage, while chapter 5 discusses the *refinement* stage (fig. 4.1).

4.1 Initialization: Motion Space Configuration and Clustering

The first problem to be solved in any segmentation technique is to estimate the number of labels to be assigned. In this application, that implies the estimation of the number of motion models in the sequence. A typical approach is to assume a certain number of models, but in this work we address the situation of an arbitrary sequence. For such a sequence no assumptions can be made about the number of trajectory bundles and the nature of their motions. However, previous work has shown that technique based on clustering provides a reliable way of making an initial guess of the segmentation. The Mean Shift algorithm is a convenient non-parametric technique for clustering data points that does not require knowledge of the number of models. Once each motion trajectory is expressed in an appropriate feature space, the Mean Shift algorithm provides both an estimate of the number of motion objects (clusters) and an initial segmentation.

The design of the feature vector is important and here we use a combination of motion basis functions and average velocity measurements. The feature vector has two components - one that take advantage of previous work in identifying trajectories of similar motion, and the other that improves the overall robustness of the clustering.

Fig. 4.3 shows the notation associated with each trajectory. Given a trajectory $\mathcal{X}_t$ that starts and ends at frames a_t and b_t respectively, and has spatial coordinates (x_f^t, y_f^t) at frame f, it has

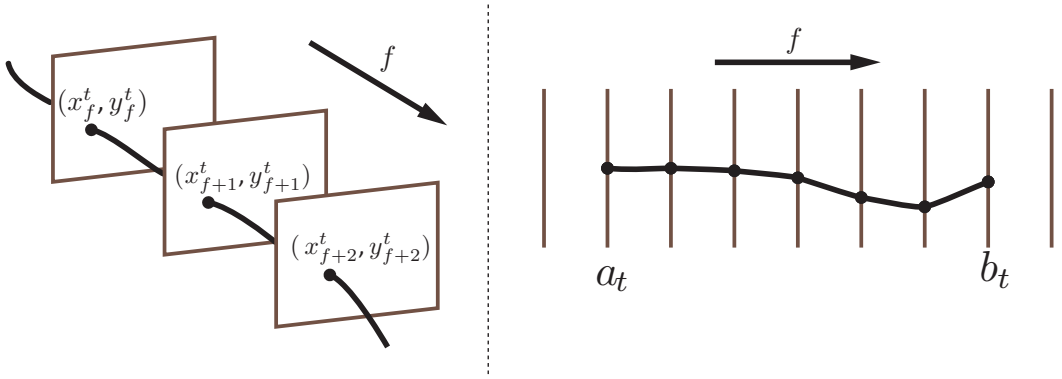

Figure 4.3: *The notation associated with trajectory $\mathcal{X}_t$ for f being the index of frames in a sequence. The spatial locations (x_f^t, y_f^t) of the points along the trajectory are shown in the* ***left*** *illustration for a monotonically increasing sequence of frames. The illustration on the* ***right*** *shows the start and end frames a_t and b_t respectively for trajectory $\mathcal{X}_t$.*

a feature vector $\mathbf{m}_t$ as follows,

$$\mathbf{m}_t = \begin{pmatrix} \mathbf{p_t} \\ \Delta_x^t \\ \Delta_y^t \\ \sigma_x^t \\ \sigma_y^t \\ a_t \end{pmatrix} \tag{4.1}$$

$$\Delta_x^t = \left(\frac{1}{b_t - a_t + 1}\right) \sum_{f=a_t}^{b_t} \left(x_f^t - x_{a_t}^t\right)$$

$$\sigma_x^{t\,2} = \left(\frac{1}{b_t - a_t + 1}\right) \sum_{f=a_t}^{b_t} \left(x_f^t - x_{a_t}^t\right)^2 - \Delta_x^{t\,2}$$

The statistics Δ_y^t and σ_y^t are similarly defined with the coordinates y_f^t.

The projection vector $\mathbf{p}_t$ of length $2F$, is derived from projecting the trajectory data into an orthogonal vector space, where F is the number of frames in the sequence. This orthogonal space allows the clustering process to use a conservative bandwidth to group trajectories with similar motion. The components Δ_x^t,Δ_y^t and σ_x^t,σ_y^t in the feature vector are the mean and standard deviation of the displacements in both directions. They are included in the vector to provide a reasonable segmentation when the projection vector $\mathbf{p}_t$ is not very informative. The starting frame a_t of the trajectory included in the feature vector encourages the clustering process to form bundles with trajectories of roughly the same starts.

4.1.1 Projection Vector Space

The projection vector $\mathbf{p}_t$ for the t^{th} trajectory is based on a space derived from previous work by Costeira *et al.* [31]. Costeira has shown that point trajectory data can be partitioned into groups of similar motion. This is done by manipulating the trajectory data projected into an orthogonal vector space. Their work is based on ideas proposed by Tomasi for solving the structure from motion problem [114]. Tomasi identified that the *measurement matrix* of trajectory data $\mathbf{W} \in \mathbb{R}^{2F}$ given below is highly rank deficient.

$$\mathbf{W} \;=\; \begin{bmatrix}\mathbf{X}\\ \hline \mathbf{Y}\end{bmatrix} \;=\; \left[\begin{array}{ccc} x_1^1 & \dots & x_1^T \\ \vdots & \ddots & \vdots \\ x_F^1 & \dots & x_F^T \\ \hline y_1^1 & \dots & y_1^T \\ \vdots & \ddots & \vdots \\ y_F^1 & \dots & y_F^T \end{array}\right] \tag{4.2}$$

Where T is the number of trajectories in the sequence.

As a result of this rank deficiency, trajectories with similar motion exist in a low dimensional manifold (2,3, or 4 dimensions) of the full $\mathbb{R}^{2F}$ space of $\mathbf{W}$. Locating these manifolds provides a reasonable solution to the trajectory segmentation. The dimensions they occupy can be identified from the factorization $\mathbf{W} \;=\; \mathbf{U}\mathbf{D}\mathbf{V}^T$ by performing singular value decomposition (SVD). However, to make the factorization less sensitive to possibly large trajectory coordinates (x_f^t, y_f^t), Tomasi factorizes a normalized version of $\mathbf{W}$, called the *registered measurement matrix* $\widetilde{\mathbf{W}}$. Factorizing the matrix $\widetilde{\mathbf{W}}$ below provides basis vectors in $\widetilde{\mathbf{U}}$ for projecting the trajectory data into an orthogonal vector space, since $\widetilde{\mathbf{U}}^T\widetilde{\mathbf{U}} \;=\; \mathbf{I}$.

$$\widetilde{\mathbf{W}} \;=\; \begin{bmatrix}\widetilde{\mathbf{X}}\\ \hline \widetilde{\mathbf{Y}}\end{bmatrix} \;=\; \left[\begin{array}{ccc} x_1^1 - u_1 & \dots & x_1^T - u_1 \\ \vdots & \ddots & \vdots \\ x_F^1 - u_F & \dots & x_F^T - u_F \\ \hline y_1^1 - v_1 & \dots & y_1^T - v_1 \\ \vdots & \ddots & \vdots \\ y_F^1 - v_F & \dots & y_F^T - v_F \end{array}\right] \;=\; \widetilde{\mathbf{U}}\widetilde{\mathbf{D}}\widetilde{\mathbf{V}}^T \tag{4.3}$$

$$\text{Where } u_f \;=\; \frac{1}{T}\sum_{t=1}^{T} x_f^t \quad , \quad v_f \;=\; \frac{1}{T}\sum_{t=1}^{T} y_f^t.$$

The projection of $\widetilde{\mathbf{W}}$ into the vector space defined by $\widetilde{\mathbf{U}}$ is $\mathbf{P} \in \mathbb{R}^{2F\times T}$,

$$\mathbf{P} \;=\; \widetilde{\mathbf{U}}^T\widetilde{\mathbf{W}} \tag{4.4}$$

It would be desirable that a number of trajectories undergoing the same translational motion, would be mapped to the same point in this space. Unfortunately with $\widetilde{\mathbf{W}}$ defined as above, this

is not the case and in fact those trajectories are mapped to a spread of points. This is discussed more clearly in the next section.

For this work we propose a modification to $\widetilde{\mathbf{W}}$ that allows all trajectories undergoing the same image translation to be projected to a single point in the orthogonal space. This modification facilitates for a smaller MS bandwidth to be used in clustering trajectories with similar translational motions. The modified $\widetilde{\mathbf{W}}$ is given as $\hat{\mathbf{W}}$ below,

$$\hat{\mathbf{W}} = \begin{bmatrix} \hat{\mathbf{X}} \\ \hline \hat{\mathbf{Y}} \end{bmatrix} = \left[\begin{array}{ccc} x_1^1 - u_1^1 & \dots & x_1^T - u_1^T \\ \vdots & \ddots & \vdots \\ x_F^1 - u_F^1 & \dots & x_F^T - u_F^T \\ \hline y_1^1 - v_1^1 & \dots & y_1^T - v_1^T \\ \vdots & \ddots & \vdots \\ y_F^1 - v_F^1 & \dots & y_F^T - v_F^T \end{array}\right] \tag{4.5}$$

$$\text{For } (u_f^t, v_f^t) = \begin{cases} (x_{a_t}^t, y_{a_t}^t), & \text{if } f \geq a_t \\ (0,0) & \text{if } f < a_t \end{cases}$$

Where trajectory $\mathcal{X}_t$ starts at frame $f = a_t$.

The fundamental difference between the $\widetilde{\mathbf{W}}$ of Tomasi and the one in this work is how the origins of the coordinate system are selected. Tomasi uses the center of the frames as the origin for all trajectories, while this work uses separate origins for each trajectory (the starting coordinates).

4.1.2 The Projection Vector

The $2F$ dimensions of the projection space defined by the basis $\widetilde{\mathbf{U}}$ outlined in the previous section is dependent on the length of the sequence. The trajectories however have varying lengths in a sequence due to occlusion or tracking failures. These partial trajectories result in the matrix $\widetilde{\mathbf{W}}$ containing empty entries for the frame in which those trajectories are not tracked. In order for varying length trajectories to all have projection vectors of the same length $2F$, the missing entries in the matrix $\hat{\mathbf{W}}$ must be addressed. To solve the structure from motion problem, Tomasi grows a full matrix from this initial sub matrix using an iterative procedure. However, for segmentation this step is not required. Instead, we generate the basis matrix $\hat{\mathbf{U}}$ using the measurement matrix obtained from using only the longest trajectories in a sequence. By only depending on the longest trajectories to define the projection space, the missing entries can be set to zeros. The projection matrix $\hat{\mathbf{P}}$ of the trajectory data is given below, where the longest trajectories define the orthogonal space.

$$\hat{\mathbf{P}} = \begin{bmatrix} \mathbf{p}_1 & \cdots & \mathbf{p}_T \end{bmatrix} = \hat{\mathbf{U}}^T \hat{\mathbf{W}} \tag{4.6}$$

Where the column vector $\mathbf{p}_t$ is the projection vector of trajectory $\mathcal{X}_t$. The basis $\hat{\mathbf{U}}^T$ is obtained from doing SVD on a matrix derived from $\hat{\mathbf{W}}$ that has only the columns for trajectories that

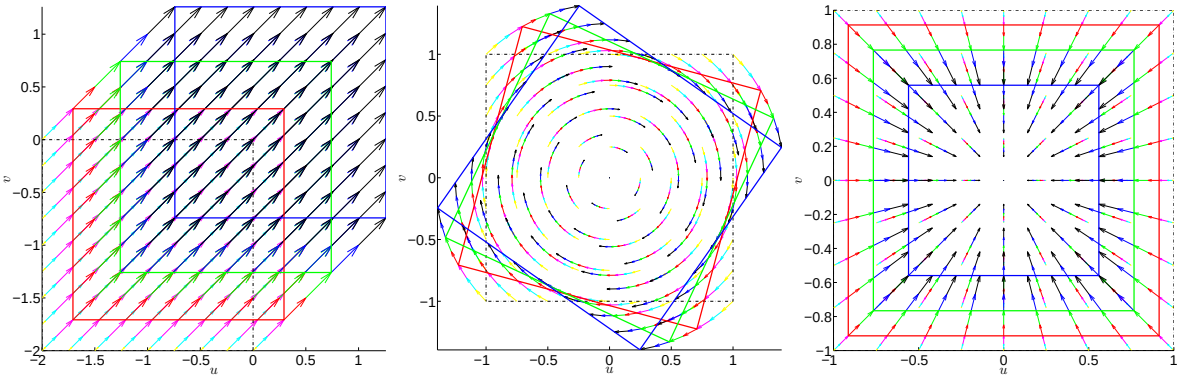

Figure 4.4: *The* 9×9 *mesh grid of points is translated (left), rotated (center), and zoomed (right) for 8 frames. Frames 4, 6, and 8 are indicated by the* ***red***, ***green***, *and* ***blue*** *boxes. The starting frame is indicated by the broken* ***black*** *box.*

exist over the entire sequence. When a trajectory is not tracked at a frame the coordinates are set to zero in the matrix $\hat{\mathbf{W}}$ at that frame.

As a result of projecting with the basis $\hat{\mathbf{U}}^T$, a complete trajectory that exists over the entire sequence has a maximum of 4 non-zero entries in the projection matrix $\hat{\mathbf{P}}$. Ideally, if this trajectory was created by translating a point for F frames, the maximum non-zeros entries would be 1. Similarly, rotating and zooming a point for F frames, the maximum non-zero entries would be 2 and 3 respectively. This remarkable result is discussed by Tron [117].

Partial/incomplete trajectories are projected into the orthogonal space by setting unknown frame coordinates to zero. Subsequently for a partial trajectory, the maximum non-zero entries in the projection matrix $\hat{\mathbf{P}}$ can exceed 4, the maximum for a complete trajectory. However, partial trajectories with similar image motion will reside in a low dimensional manifold. For example, if these partial trajectories were created by rotating a set of points for F frames, the shape of the manifold would be a planar surface.

To demonstrate how the matrix $\hat{\mathbf{W}}$, and partial/incomplete trajectories affect the projection matrix $\hat{\mathbf{P}}$, synthetic trajectory data is generated using a 9×9 grid of points that is translated, rotated and zoomed for 8 frames. Fig. 4.4 shows the synthesized trajectories with frames 4 and 6 highlighted for later illustrating the manifolds of incomplete trajectories. The projection of incomplete trajectories is simulated by creating two groups of partial trajectories from the full 8-frame trajectories. One group is created using the 8-frame trajectory data up to frame 4 only, and the matrix $\widetilde{\mathbf{W}}_4$ is derived from this data, with the rows for the trajectory coordinates at frames 5-8 all zeros. This simulates the scenario of having partial trajectories for frames 1-4 only, with frames 5-8 missing. Likewise, another group is formed by using the 8-frame trajectory data up to frame 6 only, providing the matrix $\hat{\mathbf{W}}_6$. The basis $\hat{\mathbf{U}}^T$ is obtained from the matrix $\hat{\mathbf{W}}$ derived from the the full 8-frame trajectory data. The projection matrices of the 4-frame, 6-frame and 8-frame trajectory data are $\hat{\mathbf{P}}_4 = \hat{\mathbf{U}}^T\hat{\mathbf{W}}_4$, $\hat{\mathbf{P}}_6 = \hat{\mathbf{U}}^T\hat{\mathbf{W}}_6$, and $\hat{\mathbf{P}} = \hat{\mathbf{U}}^T\hat{\mathbf{W}}$

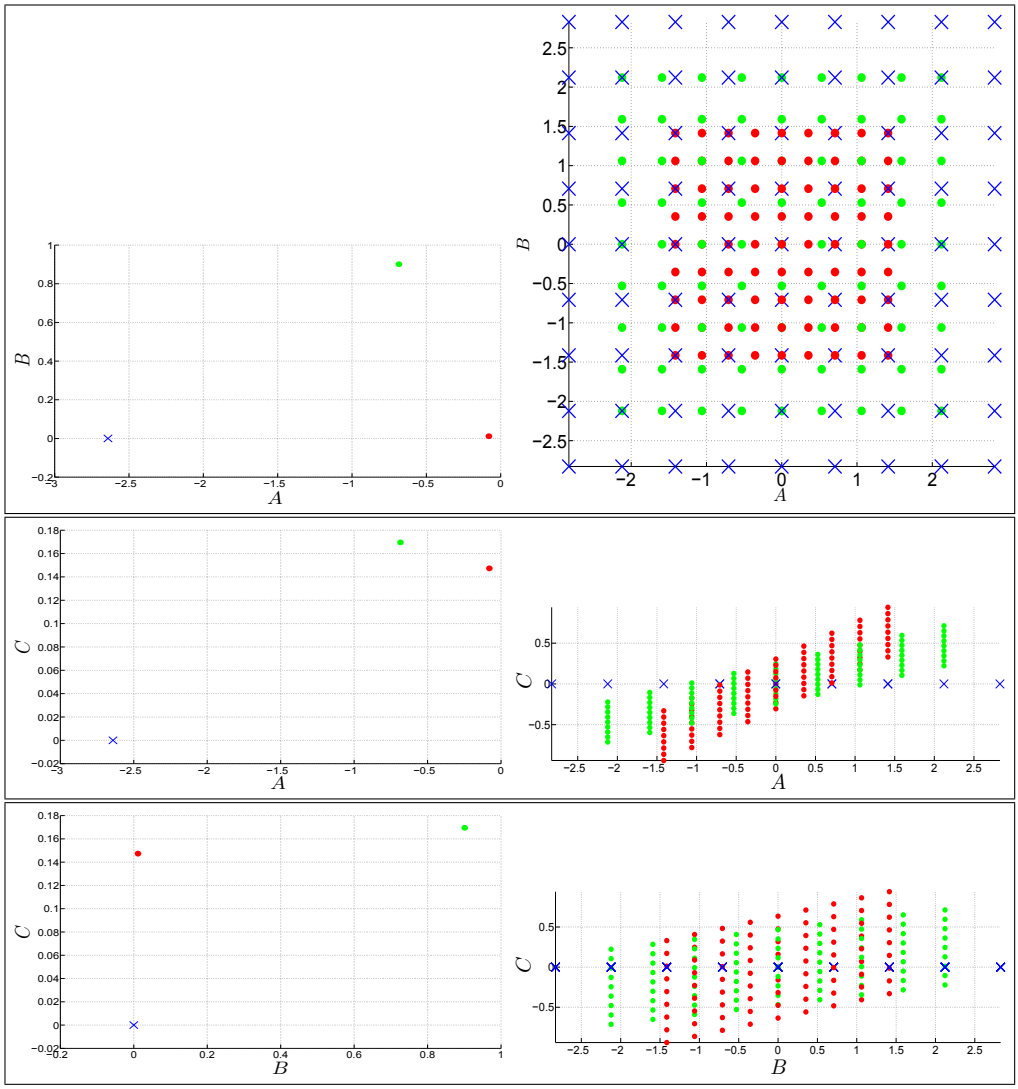

Figure 4.5: *The projection of the trajectory data for the translation sequence, where only the first three dimensions are illustrated. The* ***red*** *and* ***green dots*** *indicate the projection of the 4-frame and 6-frame trajectories respectively, while the* ***crosses*** *indicate the 8-frame trajectories.* ***Row 1-3****: Dimensions A vs B, A vs C, B vs C of the projection matrices respectively.* ***Left column****: The projection obtained using our modified matrix $\hat{\mathbf{W}}$. Here, all trajectories with the same translation are projected to a single point.* ***Right column****: The projection obtained using the matrix $\widetilde{\mathbf{W}}$ of Tomasi. Note that using this matrix, the spatial locations of the trajectories influence their projections.*

respectively.

Denoting the first four elements of the projection vector $\mathbf{p}_t$ as $\breve{\mathbf{p}}_t = [A\ B\ C\ D]^T$ we plot in fig. 4.5 three slices of the space for the *translation* sequence. The idea here is to show how various trajectories map into the space. The top graph plots A vs B, middle A vs C and bottom

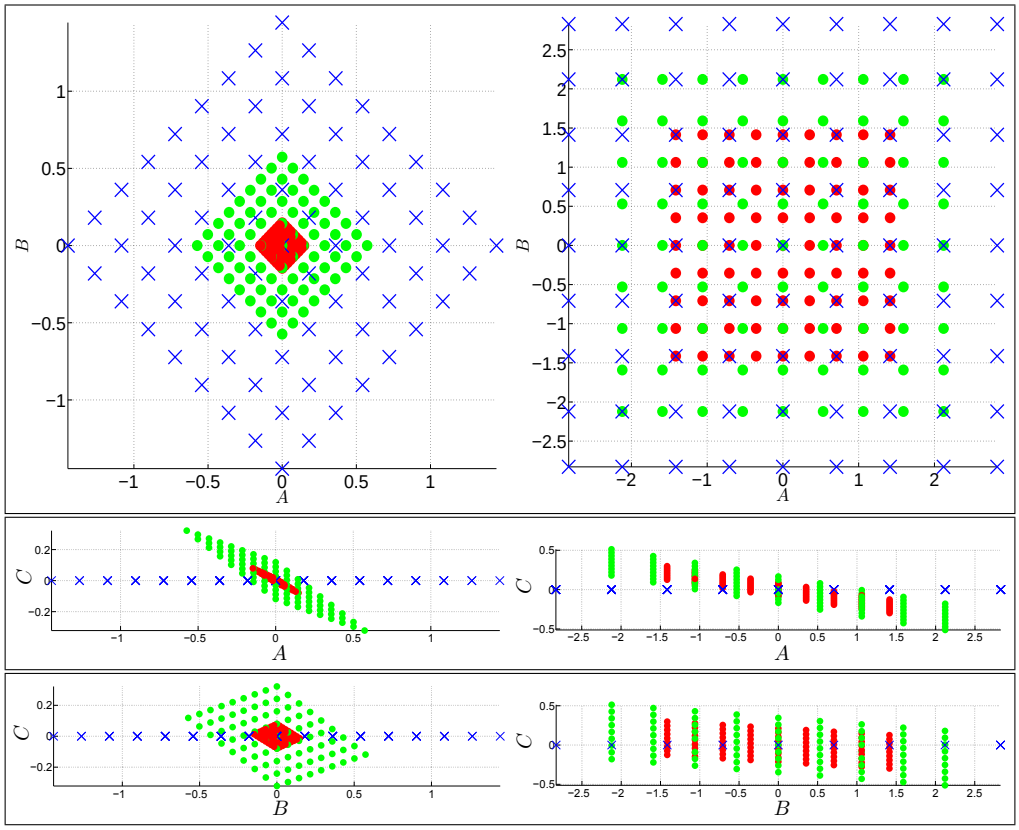

Figure 4.6: The projection of the trajectory data for the rotation sequence, where only the first three dimensions are illustrated. The red and green *dots* indicate the projection of the 4-frame and 6-frame trajectories respectively, while the *crosses* indicate the 8-frame trajectories. *Row 1-3*: Dimensions A vs B, A vs C, B vs C of the projection matrices respectively. *Left column*: The projection obtained using our modified matrix $\hat{\mathbf{W}}$. Here, all trajectories with the same translation are projected to a single point. *Right column*: The projection obtained using the matrix $\widetilde{\mathbf{W}}$ of Tomasi. Note that using this matrix, the spatial locations of the trajectories influence their projections.

B vs C. The left graphs are the projections using our modified $\hat{\mathbf{W}}$, while the right graphs are for the original $\widetilde{\mathbf{W}}$ of Tomasi.

Unlike Tomasi's formulation, our projections lead to the complete trajectories for the *translation* sequence mapping onto a point [A 0 0 0] and is shown as a *blue* '$\times$' (left of fig. 4.5) . The trajectories map to a spread of points for Tomasi's approach. This is as expected.

The two incomplete trajectory sets $\hat{\mathbf{W}}_4$ and $\hat{\mathbf{W}}_6$ map onto different points indicated by *red* and *green* 'dots' respectively using our approach. Nevertheless these points are nearby and would be clustered together if required.

Even though all three groups have the same motion up to frame 4, in reality there is no guarantee they would continue to move in a similar way for the duration of the sequence. So,

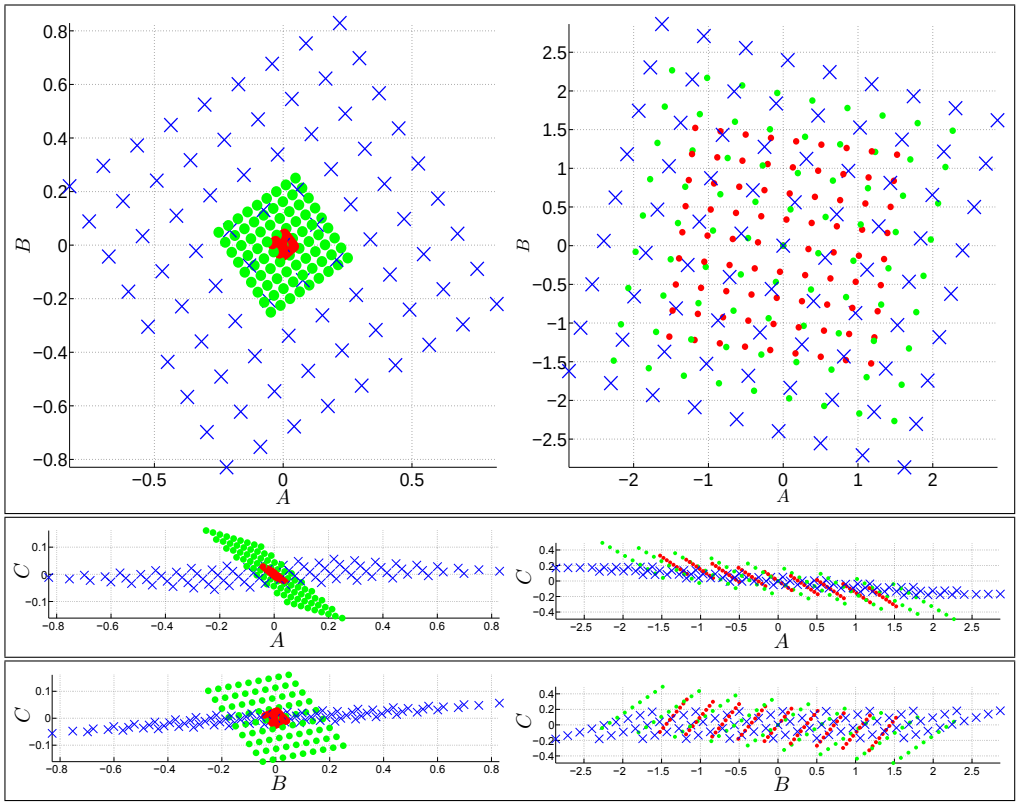

Figure 4.7: The projection of the trajectory data for the zoom sequence, where only the first three dimensions are illustrated. The red and green *dots* indicate the projection of the 4-frame and 6-frame trajectories respectively, while the *crosses* indicate the 8-frame trajectories. *Row 1-3*: Dimensions A vs B, A vs C, B vs C of the projection matrices respectively. *Left column*: The projection obtained using our modified matrix $\hat{\mathbf{W}}$. Here, all trajectories with the same translation are projected to a single point. *Right column*: The projection obtained using the matrix $\widetilde{\mathbf{W}}$ of Tomasi. Note that using this matrix, the spatial locations of the trajectories influence their projections.

it is desirable for partial trajectories to be clustered into separate bundles, and they may later be merged to longer bundles with spatial constraints in place. This is a crucial precaution in segmenting sequences shot with a moving camera. If the foreground objects temporally become static, the motion of the camera would generate similar trajectories on the foreground and background objects. The spatial constraints on the bundles would in this case help to correctly associate the partial trajectories to the respective foreground and background bundles.

The projections for the *rotation* and *zoom* sequences are shown in fig. 4.6 and fig. 4.7. Unlike the projections for the *translation* sequence using our modified $\hat{\mathbf{W}}$, the spatial location of the trajectories influence their projections. When zooming and rotating, the partial trajectories are projected into manifolds that are planar surfaces. Note that using our modified $\hat{\mathbf{W}}$, the feature point spread is less than that of Tomasi's $\widetilde{\mathbf{W}}$. These ideas influence on bandwidth selection is

discussed next.

4.1.3 Bandwidth Selection for Clustering

In Mean shift (MS) clustering the points in the d-dimensional feature space are treated as an empirical probability distribution. Given D data points $\mathbf{d}_i \in \mathbb{R}^d$, the multivariate kernel density estimate $p(\mathbf{d})$ using a Gaussian kernel is given below.

$$p(\mathbf{d}) = \frac{1}{Dh^d} \sum_{i=1}^{D} \left(\frac{\mathbf{d} - \mathbf{d}_i}{h} \right)^2 \tag{4.7}$$

Where h is termed the clustering *bandwidth*, which defines the radius of the Gaussian kernel.

When applied to our problem, the MS bandwidth controls the size of the trajectory bundles, and the quality of the segmentation. If a relatively small bandwidth is used significant over segmentation may occur. However, using too large a bandwidth may cause trajectories with obviously different image motion to be placed in the same bundle. Fig. 4.6 and fig. 4.7 show that the bandwidth required for clustering trajectories of similar motion varies according to their temporal lengths. The manifolds for longer trajectories (8-frame) are less dense than those for the shorter ones (4-frame and 6-frame). What is meant by the density of the projection here, is the closeness of trajectories with similar motion in the vector space. Therefore, to cluster a significant amount of the trajectories in the manifold into a bundle, a larger bandwidth must be used for longer trajectories. Since the bandwidth for clustering depends on the length of the trajectories, they are grouped according to these lengths. The MS clustering is then applied to each group of trajectories with the same temporal lengths. The bandwidth used for each group depends on the member trajectories. Separating the trajectories based on their lengths for clustering is a important step. This is because clustering all the trajectories at once using a large bandwidth for the longer trajectories, would result in a poor clustering of the shorter trajectories. The shorter trajectories require a smaller bandwidth for distinguishing the motion of the bundles.

The bandwidth for each group of trajectories with the same length is estimated from the feature vectors in eq. 4.1. The g^{th} group forms the feature matrix $\mathbf{M}_g$ given below. Clustering the columns of the feature matrix provide the segmentation for this group.

$$\mathbf{M}_g = \begin{bmatrix} \mathbf{m}_\alpha & \cdots & \mathbf{m}_\beta \end{bmatrix} = \begin{bmatrix} \begin{pmatrix} \mathbf{p}_\alpha \\ \Delta_x^\alpha \\ \Delta_y^\alpha \\ \sigma_x^\alpha \\ \sigma_y^\alpha \\ a_\alpha \end{pmatrix} & \cdots & \begin{pmatrix} \mathbf{p}_\beta \\ \Delta_x^\beta \\ \Delta_y^\beta \\ \sigma_x^\beta \\ \sigma_y^\beta \\ a_\beta \end{pmatrix} \end{bmatrix} \tag{4.8}$$

Where $\{\alpha : \beta\}$ are the set of indices for the trajectories in the group.

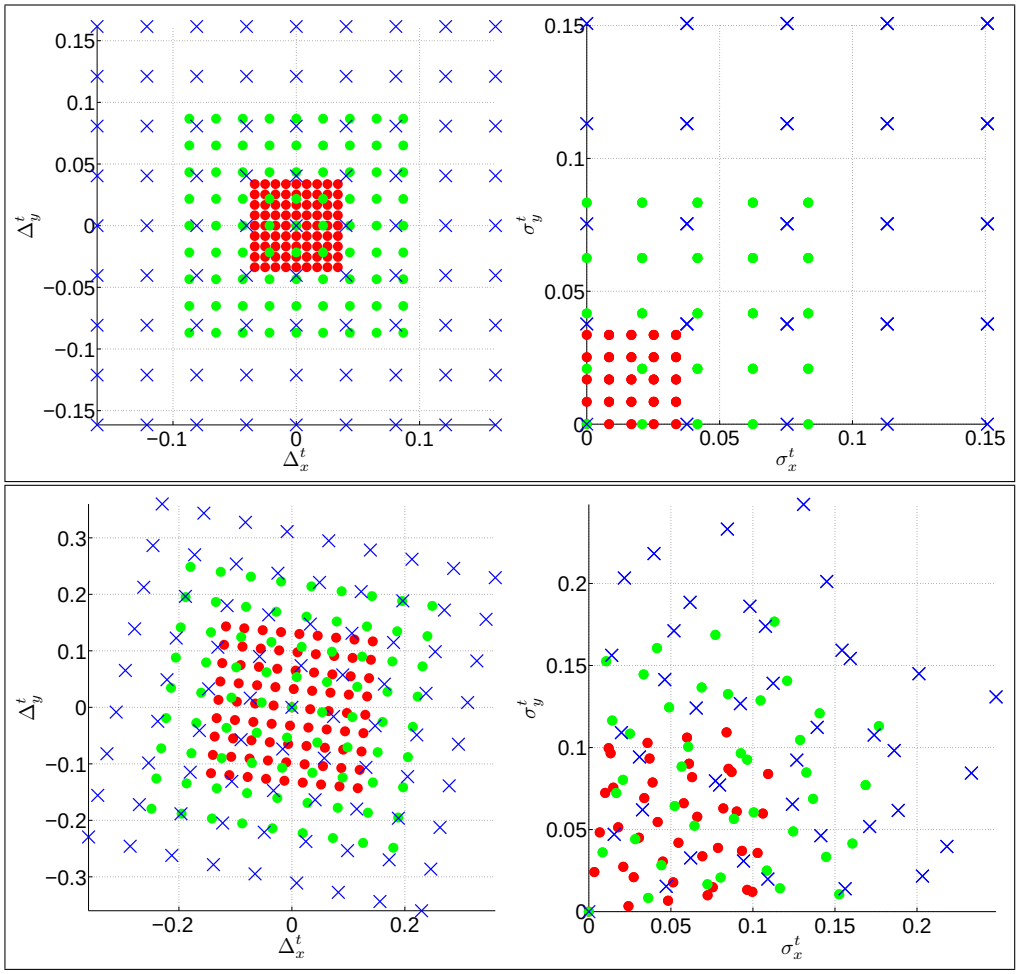

Figure 4.8: *The means Δ_x^t, Δ_y^t and standard deviations σ_x^t, σ_y^t of the trajectory displacements for the zoom and rotation synthetic sequences in fig. 4.4.* ***Top row****: Plots for the zoom sequence.* ***Bottom row****: Plots for the rotation sequence. The red and green* ***dots*** *indicate the plots of the 4-frame and 6-frame trajectories respectively, while the* ***crosses*** *indicate the 8-frame trajectories.*

The bandwidths for the rows of the feature matrix $\mathbf{M}_g$ collate into the bandwidth vector $\mathbf{b}_g$. The last element of the vector $\mathbf{b}_g$ corresponds to bandwidth for the starts of the trajectories a_t. To allow bundles (clusters) to contain trajectories that begin within ± 2 frames of each other, the bandwidth here is fixed to $\sqrt{8}$. The bandwidths for rows $1 : 2F$ of the feature matrix depend on the density of the trajectory projections $[\mathbf{p}_\alpha \ \cdots \ \mathbf{p}_\beta]$ in the orthogonal space. The standard deviation along these rows is directly proportional to the density of the projections.

The density of the means and standard deviations of the trajectory displacements are illustrated in fig. 4.8, for the rotation and zoom sequences in fig. 4.4. Here, it can be observed that the bandwidth required for the means Δ_x^t, Δ_y^t and standard deviations σ_x^t, σ_y^t of the trajectory displacements varies in a similar way to the projection vectors. That is, the densities of the Δ_x^t vs. Δ_y^t and σ_x^t vs. σ_y^t plots depend on the length of the trajectories. Therefore, the bandwidth

for rows $1 : 2F + 4$ of the feature matrix obeys the relationship below for the vector $\tilde{\mathbf{b}}_g$ being the standard deviation of the rows in the feature matrix $\mathbf{M}_g$.

$$\mathbf{b}_g(1 : 2F + 4) \propto \tilde{\mathbf{b}}_g(1 : 2F + 4) \tag{4.9}$$

$$\text{for } \tilde{\mathbf{b}}_g(1 : 2F + 4) = \text{STD}\left(\left[\begin{pmatrix} \mathbf{p}_\alpha \\ \Delta_x^\alpha \\ \Delta_y^\alpha \\ \sigma_x^\alpha \\ \sigma_y^\alpha \end{pmatrix} \quad \cdots \quad \begin{pmatrix} \mathbf{p}_\beta \\ \Delta_x^\alpha \\ \Delta_y^\alpha \\ \sigma_x^\alpha \\ \sigma_y^\alpha \end{pmatrix}\right]\right)$$

Where $\text{STD}(\mathbf{A})$ is the standard deviation along the rows of the matrix $\mathbf{A}$.

The bandwidth $\mathbf{b}_g$ used for clustering the gth group of trajectories is approximated iteratively using the standard deviation vector $\tilde{\mathbf{b}}_g$ as an initial guess. For every iteration a MS segmentation is performed until a convergence criteria is reached. The segmentation at the convergence point is the final segmentation for the group. The first of the two criteria for convergence is that the segmentation must remain constant for 5 consecutive iterations. The second criterion occurs when 90% of the trajectories in the group have been clustered to a bundle with at least 3 trajectories. This criteria encourages each bundle to have at least 3 trajectories allowing the 6 parameters of the 2D Affine motion model to be estimated with least squares.

The bandwidths corresponding to the first 4 rows of the feature matrix are kept constant for each iteration. These rows in the feature matrix contain the majority of the trajectory motion information, so we are careful not to use a large bandwidth for these rows, which would lead to a poor segmentation. Recall that the SVD factorization orders the vectors in the basis matrix $\hat{\mathbf{U}}$ according to the their eigen values. The first 4 rows of the projection correspond to the 4 largest eigen values. Keeping the bandwidths for these rows to a conservative constant prevents the merging of distinct trajectory bundles in the MS segmentation.

Dimensions 5 to $2F$ of the projection in the feature matrix correspond to significantly smaller eigen values, which makes them less informative for clustering the trajectories. Since less precaution can be taken in estimating the bandwidths for rows $5 : 2F+4$, they are linearly increased for each iteration. Linearly increasing these bandwidths reduces over segmentation caused by the sensitivity to the noise in the trajectory data of the SVD factorization process. The bandwidth

vector $\mathbf{b}_g^i$ at iteration i is given as,

$$\mathbf{b}_g^i = \begin{pmatrix} k_1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & k_2 & 0 & 0 & \cdots & 0 \\ 0 & 0 & k_3 & 0 & \cdots & 0 \\ 0 & 0 & 0 & k_4 & & \vdots \\ \vdots & 0 & 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & 0 & 0 & k_{2F+5} \end{pmatrix} \tilde{\mathbf{b}}_g = \mathbf{K}\tilde{\mathbf{b}}_g \tag{4.10}$$

$$\text{for } k_n = 0.25i,\ n = \{5 : 2F + 4\}$$

Where the matrix $\mathbf{K}$ is a diagonal matrix with the proportionality constants for the relationship defined in eq. 4.9. For all experiments reported, $k_1 = k_2 = k_3 = k_4 = 0.25$, and the entry for the starts of the trajectories, $k_{2F+5} = 1$, where $\tilde{\mathbf{b}}_g(2F + 5) = \sqrt{8}$.

4.1.3.1 Example Using Calendar and Mobile Sequence

The *Calendar and Mobile* sequence is used to demonstrate the clustering for a group of trajectories with the same length. This sequence contains four objects with distinct motions - the background, a calendar, a toy train, and a ball. There are trajectories for the background, the calendar and the toy train that exist over the entire 25 frames of the sequence, as shown in the left column of fig. 4.9. The motion and texture of the ball however prevented it from having any 25-frame trajectories. The segmentation for this group of trajectories (25-frame group) is shown in the right column of fig. 4.9. This segmentation took 34 iterations to cluster 1510 trajectories into 145 bundles. There were 6 bundles with 3 or more trajectory, accounting for 1366 of all the trajectories of length 25 frames. It may be observed that there are trajectory bundles associated with the background, calendar, and train validating the credibility of the segmentation.

Figs. 4.10 and 4.11 show the clustering of the feature matrix $\mathbf{M}_g$ derived from the 25-frame group of trajectories for the *Calendar and Mobile* sequence. The clusters are coloured similarly to the corresponding trajectory bundles in fig. 4.9. For example, red and green for the background and calendar trajectories respectively. In these figures all the trajectory feature matrix entries are shown on the *right* so it can be seen how outliers from the main trajectory bundles are isolated in the MS segmentation process. Outlier trajectories arise from feature tracking errors, and the motion of some local regions of the objects. The local motion of an object can create trajectories with motions significantly different from the global motion of that object. There are not enough of these local trajectories to form a trajectory bundle with more than 3 members, so they become isolated in the segmentation. For example, the wheels of the toy train have a harmonic motion that differs from the global translational motion of the train. The conservative bandwidth in the first 4 rows of the feature matrix prevents outlier trajectories from being

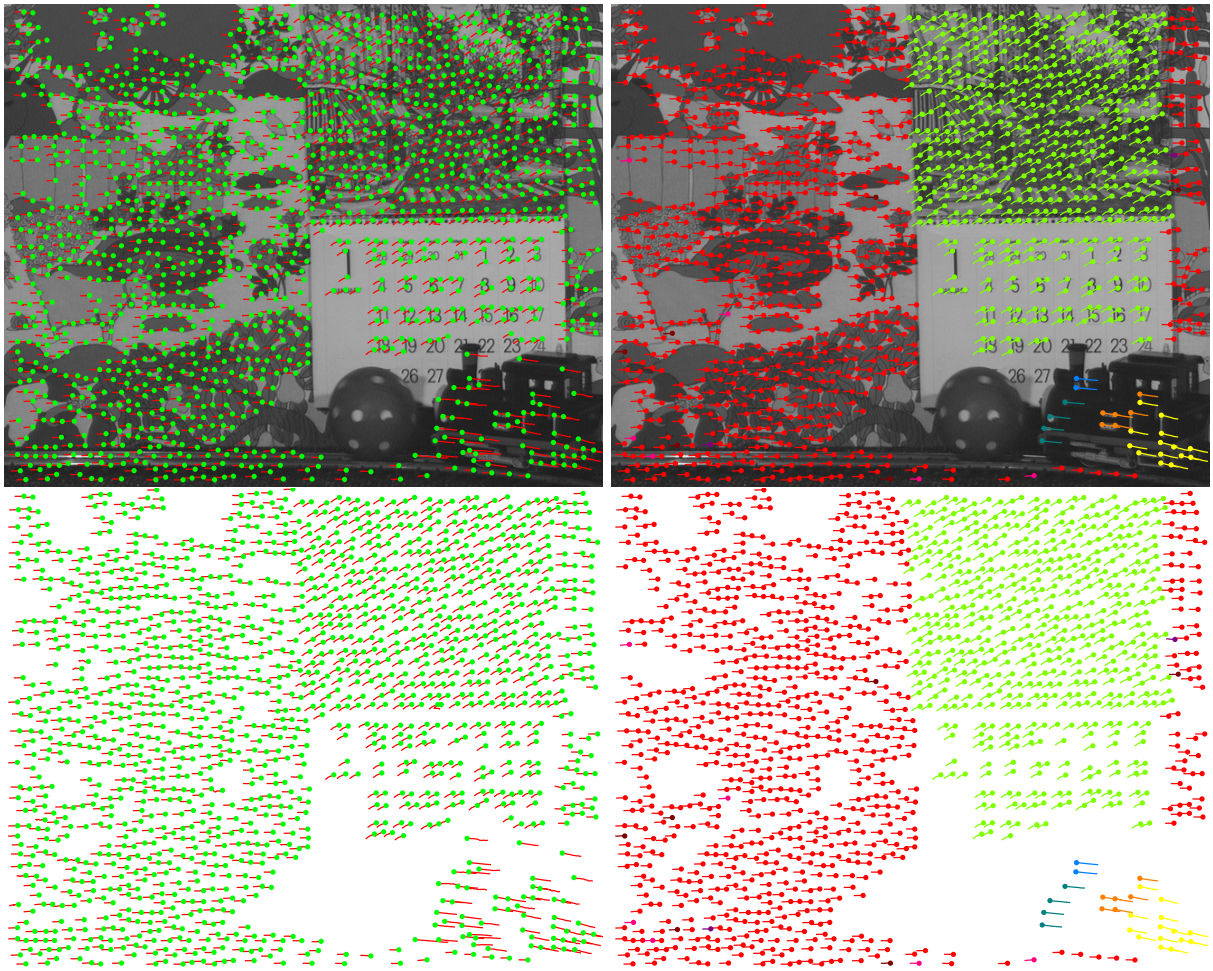

Figure 4.9: *The segmentation of the 25-frame group of trajectories for the* ***Calendar and Mobile*** *sequence.* ***Row 1****: The trajectories superimposed on frame 12, with the history the trajectories shown.* ***Left*** *- all the trajectories of length 25 frames,* ***right*** *- the segmentation of these trajectories, where bundles having a single trajectory not shown for clarity.* ***Row 2****: The trajectories in* ***row 1*** *superimposed on a white background. The segmented trajectory bundles are illustrated with different colours.*

associated with the main trajectory bundles (background, calendar, and train). They may be later be associated with a main bundle, or placed in a new bundle after the *refinement* stage.

4.2 Initialization: Motion Model Estimation and Merging

The final steps in the *initialization* stage involve modelling the motion of the trajectory bundles obtained from MS clustering, and merging those with similar motion. The MS clustering process primarily creates bundles of trajectories with similar translational motion. For other forms of 2D image motion, such as rotation and zoom, the trajectories undergoing these motions can

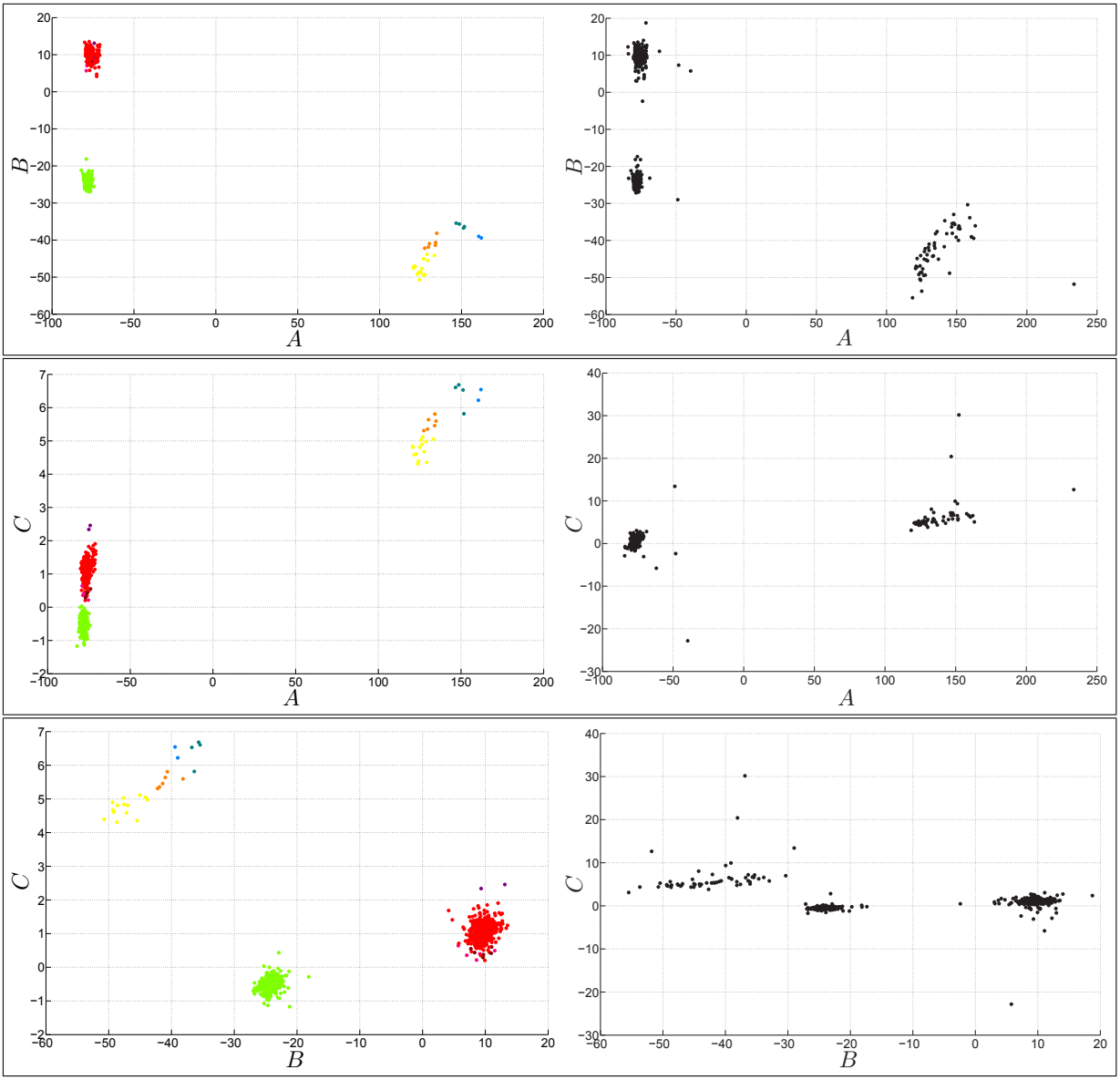

Figure 4.10: ***Left column***: *The MS clustering of the projection vectors for the 25-frame group of trajectories for the* ***Calendar and Mobile*** *sequence.* ***Right column***: *The projections for all the 25-frame trajectories.* ***Row 1-3***: *Dimensions A vs B, A vs C, B vs C of the feature vectors.*

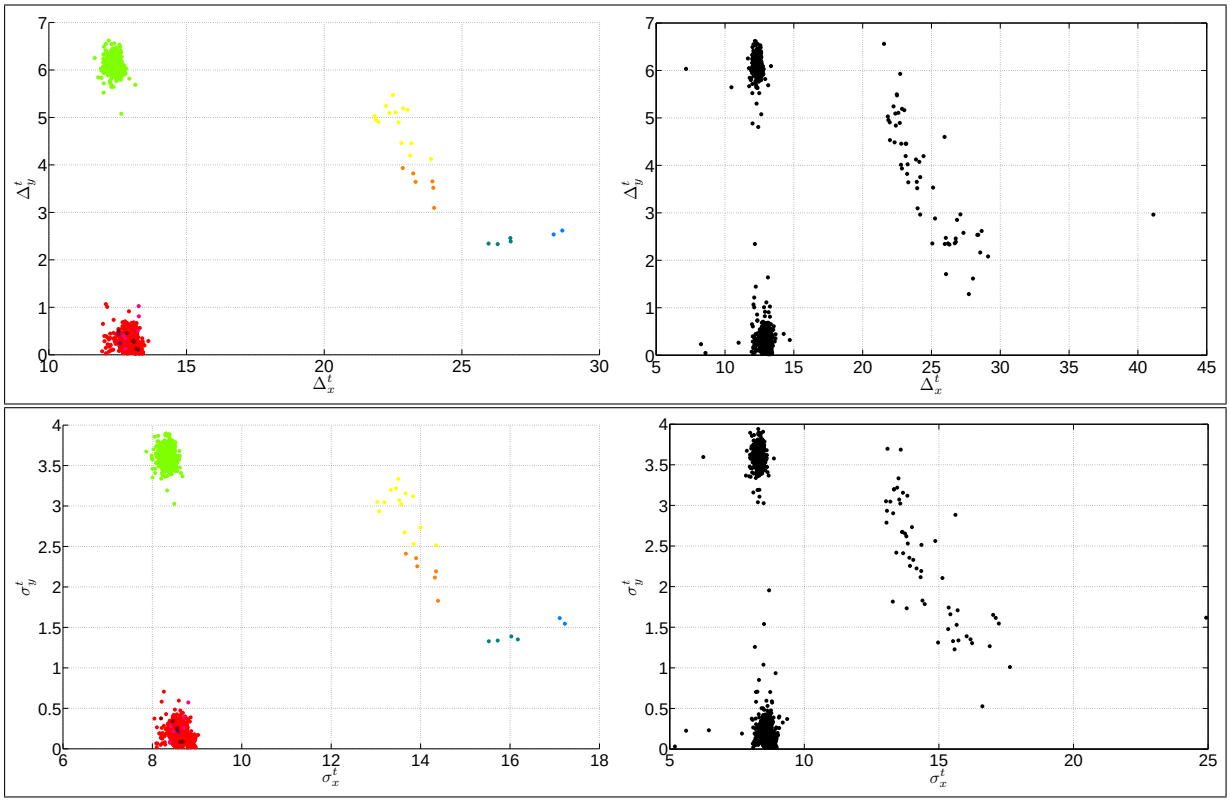

Figure 4.11: ***Left column***: *The MS clustering of the means* Δ_x^t *(top row) and standard deviations* σ_x^t *(bottom row) of the trajectory displacements for the 25-frame group of trajectories for the* ***Calendar and Mobile*** *sequence.* ***Right column***: *The means and standard deviations for all the 25-frame trajectories.*

occupy a manifold that may extend significantly across the feature space. The conservative clustering bandwidth will cause these trajectories to be placed into several sub-bundles, with the motion of each sub-bundle being approximately translational. The MS clustering process encourages these sub-bundles to have at least 3 trajectories, so their motions can be modelled with a 6-parameter Affine model. Using these motion models, the sub-bundles with similar 2D motion may be merged into a single bundle.

The merging of the bundles using the motion models also aims to combine bundles with different trajectory lengths. For clustering, the trajectories are separated into groups according to their lengths, where a different bandwidth is applied to each group. This grouping of the trajectories is a critical step for clustering as discussed previously. However, it introduces an unwanted level of segmentation that is reduced by merging the bundles based on their motion models.

The bundle merging step involves assessing the motion similarity amongst all the bundles using a defined metric. All the bundles that have been deemed to have similar motion are pooled together. Each bundle votes for another bundle in the pool that it wants to merge with, and all

bundles with at least one vote are merged into a new bundle. The merging of the new bundles and the previously unmerged bundles is iteratively attempted until there are no more mergers.

4.2.1 The Motion Model

The motion of the bundles are modelled using a set of 2D Affine transformations. These transformations represent the relationship between the spatial locations of the trajectories across adjacent frame pairs. That is, given the spatial locations (x_f^t, y_f^t) of the trajectories at frame f, a linear transformation $\mathbf{A}_{f,f+1}^n$ gives the locations (x_{f+1}^t, y_{f+1}^t) in the next frame $f+1$. The transformation $\mathbf{A}_{f,f+1}^n$ from frame f to $f+1$ for the nth trajectory bundle defines this transformation for the member trajectories. Given the set of trajectories in the nth bundle is $\mathcal{B}_n$. The transformation between frames f and $f+1$ for a member trajectory $\mathcal{X}_\nu$, where $\nu \in \mathcal{B}_n$, is given below.

$$\begin{pmatrix} x_{f+1}^\nu \\ y_{f+1}^\nu \\ 1 \end{pmatrix} = \left[\begin{array}{ccc} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ \hdashline 0 & 0 & 1 \end{array}\right] \begin{pmatrix} x_f^\nu \\ y_f^\nu \\ 1 \end{pmatrix} = \mathbf{A}_{f,f+1}^n \begin{pmatrix} x_f^\nu \\ y_f^\nu \\ 1 \end{pmatrix} \tag{4.11}$$

Using a least squares fit, the transformation $\mathbf{A}_{f,f+1}^n$ is estimated from the trajectories in the bundle that exist at both frames f and $f+1$. Given trajectory $\mathcal{X}_k$ is one of these K trajectories in bundle n that exists for both frames f and $f+1$, the transformation $\mathbf{A}_{f,f+1}^n$ is estimated as follow.

$$\mathbf{A}_{f,f+1}^n = \begin{cases} \left[\begin{array}{c:c} \widetilde{\mathbf{X}}\mathbf{X}^T\left(\mathbf{X}\mathbf{X}^T\right)^{-1} & \\ \hdashline 0 \quad 0 & 1 \end{array}\right], & \text{if } K \geq 3 \\ \left[\begin{array}{cc:c} 1 & 0 & \frac{1}{K}\sum_k \left(x_{f+1}^k - x_f^k\right) \\ 0 & 1 & \frac{1}{K}\sum_k \left(y_{f+1}^k - y_f^k\right) \\ \hdashline 0 & 0 & 1 \end{array}\right], & \text{if } K < 3 \end{cases} \tag{4.12}$$

$$\text{for } \mathbf{X} = \begin{bmatrix} x_f^\alpha & \cdots & x_f^\beta \\ y_f^\alpha & \cdots & y_f^\beta \\ 1 & \cdots & 1 \end{bmatrix}, \widetilde{\mathbf{X}} = \begin{bmatrix} x_{f+1}^\alpha & \cdots & x_{f+1}^\beta \\ y_{f+1}^\alpha & \cdots & y_{f+1}^\beta \end{bmatrix}$$

Where $k \in \{\alpha : \beta\}$ are the set of indices for the trajectories in the nth bundle, that exist at both frames f and $f+1$.

In eq. 4.12 above, the transformation $\mathbf{A}_{f,f+1}^n$ is estimated with least squares once the bundle has 3 or more trajectories ($K \geq 3$) for the frame interval f to $f+1$. This is the minimum number of trajectories required for the rank of the 3×3 matrix $\mathbf{X}\mathbf{X}^T$ to be 3, so it can be

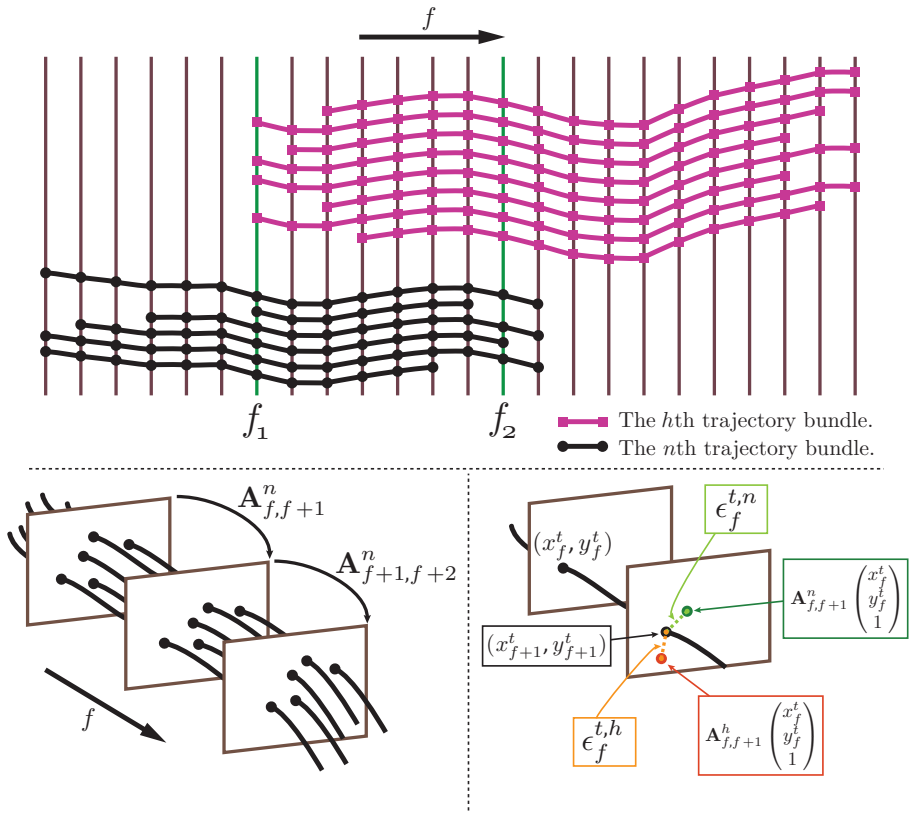

Figure 4.12: *The notation used for discussing the nth and hth trajectory bundles with similar motion.* ***Top****: The trajectories in both bundles are shown, where the average errors are evaluated in the frames ranging from f_1 to f_2.* ***Bottom left****: The transformations between adjacent frame pairs for a monotonically increasing frame index f.* ***Bottom right****: The errors for the tth trajectory evaluated with the transformations for both bundles.*

inverted providing the solution for the 6 parameters of the transformation matrix. The 2×3 matrix $\widetilde{\mathbf{X}}\mathbf{X}^T\left(\mathbf{X}\mathbf{X}^T\right)^{-1}$ are the first two rows of the transformation $\mathbf{A}^n_{f,f+1}$. When there are less than 3 trajectories ($K < 3$), the transformation is restricted to be translational, since the matrix $\mathbf{X}\mathbf{X}^T$ is not invertible.

4.2.2 Similarity Metric for Model Merging

To facilitate merging, some metric of similarity between bundles must be estimated. Trajectory bundles with similar image motion will have similar motion model transformations. Given the nth and hth bundles are similar in motion, fig. 4.12 illustrates both bundles and the notation that will be used to discuss them. The transformations for the nth bundle should be able to predict the spatial locations of the trajectories in the hth bundle and vice versa. Small prediction errors naturally allude to the similarity of the bundles. Here prediction errors combined into a measurement vector along a motion path would help to build a similarity metric.

4.2.3 Prediction Error

The prediction error $\epsilon_f^{v,n}$ at frame f for a trajectory $\mathcal{X}_v$ in the hth bundle with respect to the nth trajectory bundle is given below.

$$\epsilon_f^{v,n} = \left\| \mathbf{A}_{f,f+1}^{n} \begin{pmatrix} x_f^v \\ y_f^v \\ 1 \end{pmatrix} - \begin{pmatrix} x_{f+1}^v \\ y_{f+1}^v \\ 1 \end{pmatrix} \right\| \tag{4.13}$$

Where $\|.\|$ is the $l2$-norm.

By changing the transformation $\mathbf{A}_{f,f+1}^{n}$ in eq. 4.13 above to the corresponding transformation $\mathbf{A}_{f,f+1}^{h}$ for the hth bundle, the reference error $\epsilon_f^{v,h}$ for the trajectory is obtained. The reference error is defined as the error of the trajectory with respect to the bundle it belongs to. If the nth and hth bundles are similar in motion, then the error with respect to the other bundle $\epsilon_f^{v,n}$ is approximately equal to the reference error $\epsilon_f^{v,h}$. Also in the reverse case of a trajectory $\mathcal{X}_k$ in the nth bundle with the reference error being $\epsilon_f^{k,n}$, the error similarity condition is $\epsilon_f^{k,h} \approx \epsilon_f^{k,n}$. Note that we can only usefully calculate these errors where the trajectories overlap in time. Hence $\epsilon_f^{v,n}$ is calculated for frames f_1 to f_2 only, i.e. the overlap duration. See fig. 4.12.

The transformations $\mathbf{A}_{f,f+1}^{n}$ and $\mathbf{A}_{f,f+1}^{h}$ may have been estimated with noisy or insufficient trajectories causing these errors to significantly differ. Insufficient trajectories ($K < 3$) in one bundle will automatically cause the transformations to be translational for this bundle, while the other may be fully Affine. The transformation model differences here can cause the error similarity condition to fail. To be more robust to these scenarios, another transformation $\mathbf{A}_{f,f+1}^{\Omega}$ is estimated using the trajectories from the union of both bundles. Using this transformation the error conditions $\epsilon_f^{k,\Omega} \approx \epsilon_f^{k,n}$, and $\epsilon_f^{v,\Omega} \approx \epsilon_f^{v,h}$ provide additional information on the similarity of the bundles.

The transformation $\mathbf{A}_{f,f+1}^{\Omega}$ is estimated from the trajectories of both bundles using a weighted least squares solution. Consider that the nth and hth bundles have K_n and K_h trajectories that exist at both frames f and $f+1$. If K_n is much greater than K_h then a normal least square solution would bias the combined transformation $\mathbf{A}_{f,f+1}^{\Omega}$ towards $\mathbf{A}_{f,f+1}^{n}$, and vice versa. By weighting the trajectories from both bundles, the smaller contributing bundle will have a greater influence on the combined transformation, unlike the situation where a normal least square solution is utilized.

There are $K_\Omega = K_n + K_h$ trajectories from both bundles available for estimating the combined transformation, where trajectory $\mathcal{X}_j$ is one of these K_Ω trajectories. The transformation $\mathbf{A}_{f,f+1}^{\Omega}$

is estimated as follows,

$$\mathbf{A}^{\Omega}_{f,f+1} = \begin{cases} \left[\begin{array}{c|c} \widetilde{\mathbf{X}}\mathbf{X}^T\left(\mathbf{X}\mathbf{X}^T\right)^{-1} & \\ \hline 0 \quad 0 & 1 \end{array}\right], & \text{if } K_\Omega \geq 3 \\ \left[\begin{array}{cc|c} 1 & 0 & \sum_j w_j\left(x^j_{f+1} - x^j_f\right) \\ 0 & 1 & \sum_j w_j\left(y^j_{f+1} - y^j_f\right) \\ \hline 0 & 0 & 1 \end{array}\right], & \text{if } K_\Omega < 3 \end{cases} \tag{4.14}$$

$$\text{For } \mathbf{X} = \begin{bmatrix} w_\alpha x^\alpha_f & \cdots & w_\beta x^\beta_f \\ w_\alpha y^\alpha_f & \cdots & w_\beta y^\beta_f \\ 1 & \cdots & 1 \end{bmatrix}, \widetilde{\mathbf{X}} = \begin{bmatrix} w_\alpha x^\alpha_{f+1} & \cdots & w_\beta x^\beta_{f+1} \\ w_\alpha y^\alpha_{f+1} & \cdots & w_\beta y^\beta_{f+1} \end{bmatrix}$$

$$\text{The weight } w_j = \begin{cases} \frac{1}{2K_n}, & \text{if trajectory} \mathcal{X}_j \text{ is from the } n\text{th bundle.} \\ \frac{1}{2K_h}, & \text{if trajectory} \mathcal{X}_j \text{ is from the } h\text{th bundle.} \end{cases} \tag{4.15}$$

Where $j \in \{\alpha : \beta\}$ are the set of indices for the K_Ω trajectories from both bundles, that exist in the frame interval f to $f+1$.

4.2.4 The Similarity Matrix

We are now in a position to assemble errors from various points along the trajectory bundle paths. Recall that we seek to test the merging of two bundles n and h. And further recall that $\epsilon^{\cdot,\cdot}_f$ is only measured for frames f_1 to f_2. As illustrated in fig. 4.13, at each of these frames there are 4 cross errors $\epsilon^{k,h}_f$,$\epsilon^{v,n}_f$,$\epsilon^{v,\Omega}_f$,$\epsilon^{k,\Omega}_f$ (black arrows) and 2 reference errors $\epsilon^{k,n}_f$,$\epsilon^{v,h}_f$ (coloured arrows) generated using the three motion models.

At each time stamp we use the average error $\bar{\epsilon}^{\cdot,\cdot}_f$ as an indication of similarity over all trajectories in a bundle. For the trajectories $\mathcal{X}_v$ in the hth bundle, the average error over all these trajectories with respect to the nth bundle is $\bar{\epsilon}^{v,n}_f$. Given that the motion of the bundles n and h are the same $\bar{\epsilon}^{v,n}_f \approx \bar{\epsilon}^{v,h}_f$.

For both bundles having trajectories between frames f_1 and f_2, a similarity matrix $\mathbf{S}_{n,h}$ is created using the average trajectory errors for these frames. Note that the trajectory error $\epsilon^{t,n}_f$ at frame f in eq. 4.13 requires that this trajectory exists at frame $f+1$. Therefore frame f_2 here is one frame before the temporal overlap of the bundles end, as illustrated in the *bottom right* of fig. 4.12. The $L \times 4$ similarity matrix $\mathbf{S}_{n,h}$ for the nth and hth bundles is given below,

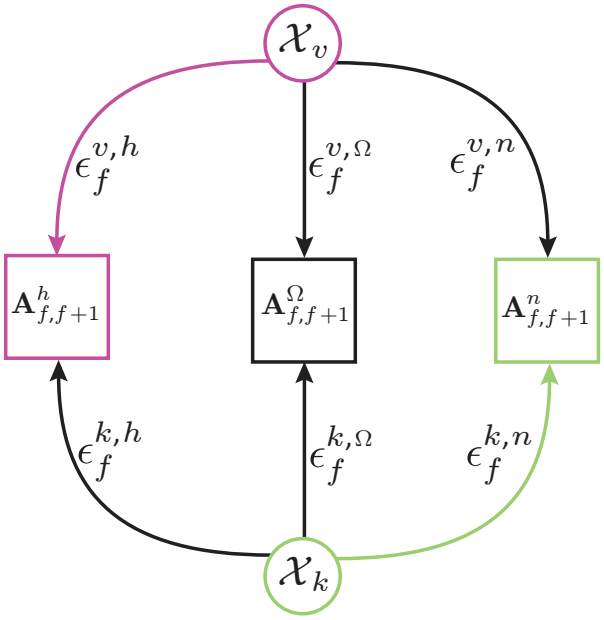

Figure 4.13: *Summary of the prediction errors associated with the trajectories in bundles n and h. Trajectory $\mathcal{X}_k$ and $\mathcal{X}_v$ belong to the bundles n and h respectively, and are indicated with the circles. The squares represent the three motion models $\mathbf{A}_{f,f+1}^{h}$, $\mathbf{A}_{f,f+1}^{\Omega}$ and $\mathbf{A}_{f,f+1}^{n}$ used to generate the errors. The prediction errors $\epsilon_f^{\cdot,\cdot}$ for the trajectories with respects to the motion models are the arrow labels. The reference prediction errors are the coloured arrows.*

where $L = f_2 - f_1 + 1$.

$$\mathbf{S}_{n,h} = \begin{bmatrix} \mathbf{S}_{n,h}(1,1) & \mathbf{S}_{n,h}(1,2) & \cdots & \mathbf{S}_{n,h}(1,4) \\ \mathbf{S}_{n,h}(2,1) & \mathbf{S}_{n,h}(2,2) & \cdots & \mathbf{S}_{n,h}(2,4) \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{S}_{n,h}(L,1) & \mathbf{S}_{n,h}(L,2) & \cdots & \mathbf{S}_{n,h}(L,4) \end{bmatrix} = \mathbf{E}_o - \mathbf{E}_r \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \tag{4.16}$$

$$\mathbf{E}_o = \begin{bmatrix} \bar{\epsilon}_{f_1}^{v,n} & \bar{\epsilon}_{f_1}^{k,h} & \bar{\epsilon}_{f_1}^{v,\Omega} & \bar{\epsilon}_{f_1}^{k,\Omega} \\ \vdots & \vdots & \vdots & \vdots \\ \bar{\epsilon}_{f_2}^{v,n} & \bar{\epsilon}_{f_2}^{k,h} & \bar{\epsilon}_{f_2}^{v,\Omega} & \bar{\epsilon}_{f_2}^{k,\Omega} \end{bmatrix}, \mathbf{E}_r = \begin{bmatrix} \bar{\epsilon}_{f_1}^{k,n} & \bar{\epsilon}_{f_1}^{v,h} \\ \vdots & \vdots \\ \bar{\epsilon}_{f_2}^{k,n} & \bar{\epsilon}_{f_2}^{v,h} \end{bmatrix}$$

Given the nth and hth bundles having member trajectories $\mathcal{X}_k$ and $\mathcal{X}_v$. The matrix $\mathbf{E}_r$ is the concatenation of the average reference errors over the frames f_1 to f_2. Also the matrix $\mathbf{E}_o$ is the concatenation of the average errors for using the cross transformations.

The design of the similarity matrix $\mathbf{S}_{n,h}$ makes it approximately equal to the null matrix if both bundles are similar in motion over all the frames f_1 to f_2. That is, the elements of the similarity matrix $\mathbf{S}_{n,h}(i,j) \approx 0$. By finding the percentage of these elements $\mathbf{S}_{n,h}(i,j)$ that are less than a threshold s_τ, we derive a single scalar metric that describes the similarity of the

bundles. The similarity metric $\varsigma_{n,h}$ for the nth and hth bundle is defined as,

$$\varsigma_{n,h} = \frac{1}{4L} \sum_{i=1}^{L} \sum_{j=1}^{4} (\mathbf{S}_{n,h}(i,j) \leq s_\tau) \tag{4.17}$$

The threshold s_τ controls what the acceptable average error differences should be for similar bundles. For all sequences reported this threshold is set to a constant of $s_\tau = 0.3$.

Fig. 4.14 shows how the threshold s_τ influences the merging of the trajectory bundles for the *Calendar and Mobile* sequence. The unmerged trajectories bundles at frame 12 in this sequence are shown in the *top left*. The illustrations clockwise from the *top right* show the merged bundles using the thresholds $s_\tau = 0.5, 0.3, 0.1$ respectively. It may be here observed that using the threshold $s_\tau = 0.3$ (bottom right) leads to the best merging result. For $s_\tau = 0.5$ (top right) the calendar and background bundles are undesirably merged together. Also for $s_\tau = 0.3$ (bottom left) there is a significant amount of over segmentation for the trajectories corresponding to the toy train.

When the similarity metric $\varsigma_{n,h} = 1$ there is maximum confidence that the bundles are very similar and they should be merged. However, $\varsigma_{n,h} = 0$ means the bundles have absolutely different motions. If both bundles do not have any trajectories that overlap in time, then the similarity metric $\varsigma_{n,h}$ is set to zero. Hence they are not merged.

Long-term bundles with absolutely different image motion can have similar motion over a short window of time (1-3 frames). They would have a high metric $\varsigma_{n,h}$ if the similarity analysis coincided with this short window of time. To prevent high metrics being awarded to these bundles there is a restriction on the analysis window size. So, the metric $\varsigma_{n,h} = 0$ if the overlap period is less than 4 frames ($f_2 - f_1 + 1 < 4$).

4.2.5 Bundle Merge Pools

The nth and hth bundles are deemed similar if the metric $\varsigma_{n,h}$ is greater than a threshold $\varsigma_\tau = 0.8$. This means that 80% of the measurements along the trajectory overlap duration satisfy the error similarity constraint. All pairs of bundles are tested in this way.

The nth bundle may have several merge candidate bundles that satisfy $\varsigma_{n,\cdot} > \varsigma_\tau$. This requires a method for deciding which candidates should be a part of the final merged bundle. There are situations that commonly arise where not all the candidate are suitable for the final bundle. For example, if a bundle is partially similar to two bundles that are absolutely different, caution must be taken in the merging of these bundles. A decision must be taken about how to merge these bundles while keeping the bundles with no similarity separate. A robust solution to this problem is to make the candidate bundles vote for each other. A candidate bundle h_1 votes for bundle h_2 if $\varsigma_{h_1,h_2} > \varsigma_\tau$.

Fig. 4.15 shows examples of the similarity metrics $\varsigma_{n,h}$ for 5 trajectory bundles. Since $\varsigma_{n,h} = \varsigma_{h,n}$ only the metrics $\varsigma_{n,h}$ are shown, where $n < h$. In this example we are attempting to merge bundles 2, 3 and 4 to bundle 1. Hence bundles 2, 3 and 4 are defined as the candidate bundles

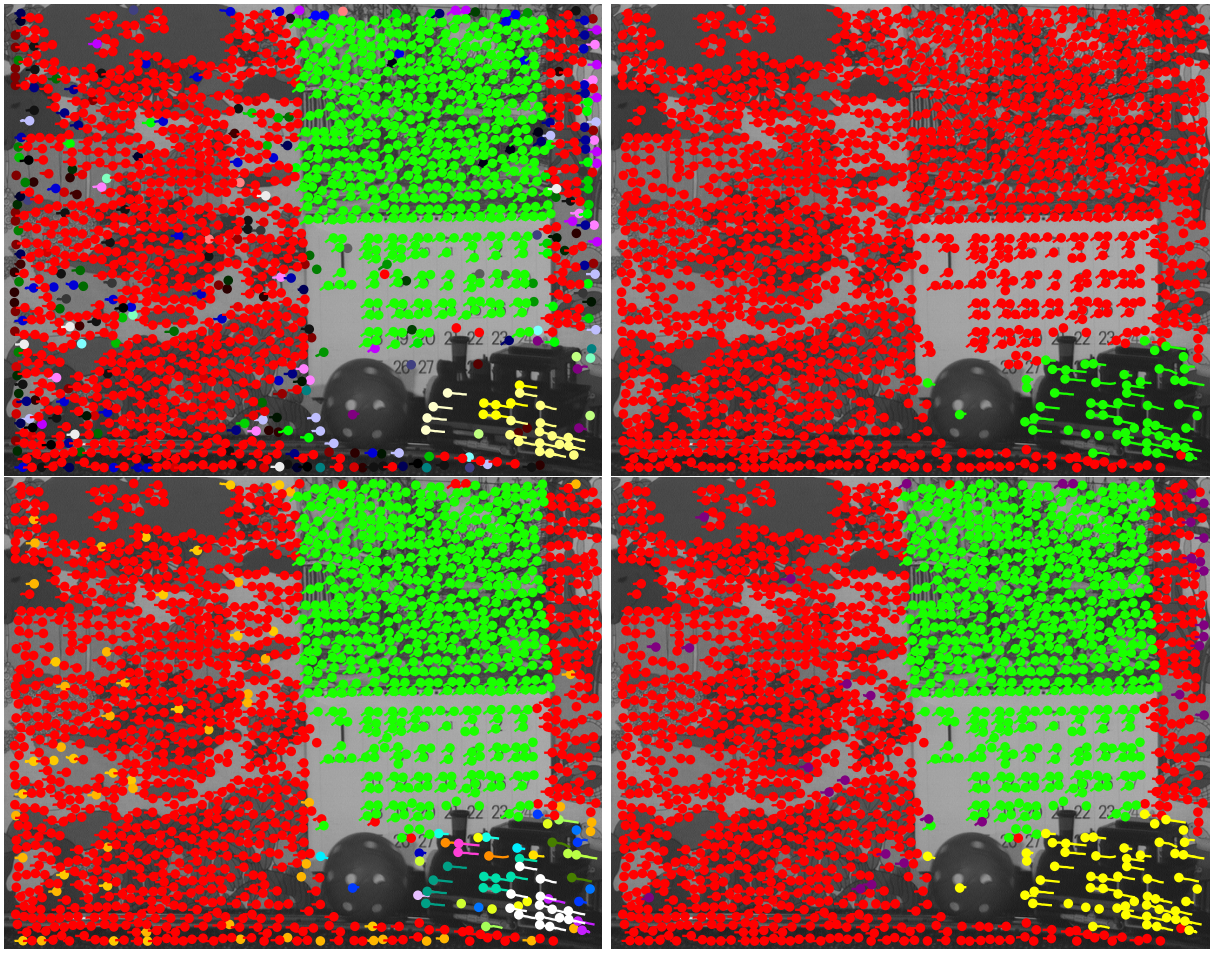

Figure 4.14: *The merging of the trajectory bundles at frame 12 in the* ***Calendar and Mobile*** *sequence.* ***Top left****: The bundles after Mean Shift clustering, this is the input to the merging step.* ***Clockwise from top right****: The merging result using the thresholds on the elements of the similarity matrix* $s_\tau = 0.5, 0.3, 0.1$ *respectively.*

for bundle 1, since $\varsigma_{1,2} > \varsigma_\tau$, $\varsigma_{1,3} > \varsigma_\tau$ and $\varsigma_{1,4} > \varsigma_\tau$. These metrics are highlighted in *green* in fig. 4.15. Note that bundle 5 is not a candidate for bundle 1 since $\varsigma_{1,5} < \varsigma_\tau$. All metrics where $\varsigma_{n,h} < \varsigma_\tau$ are highlighted in *red*.

The candidate bundles 2 and 4 both vote for each other since $\varsigma_{2,4} = \varsigma_{4,2} > \varsigma_\tau$. However bundle 3 does not receive a vote from either bundle 2 or 4 because $\varsigma_{2,3} < \varsigma_\tau$ and $\varsigma_{2,4} < \varsigma_\tau$. Hence only candidate bundles 2 and 4 are merged with bundle 1.

The final merged bundles along with the unmerged bundle at the current iteration are then considered for merging at the next iteration. When there are no more mergers the bundle merging step is complete.

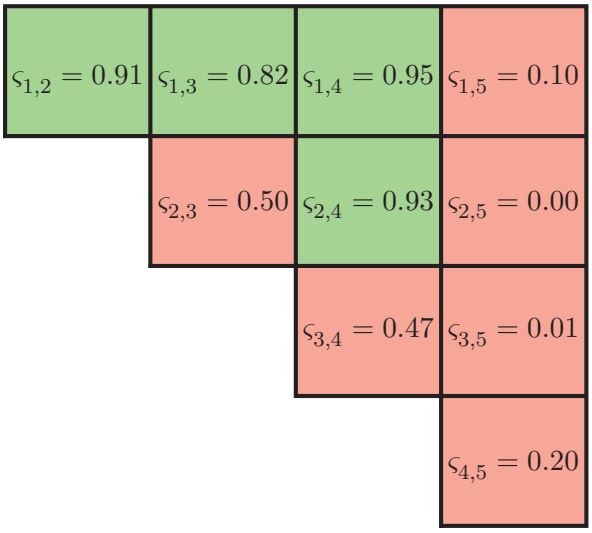

Figure 4.15: *An example of the similarity metrics $\varsigma_{n,h}$ among 5 bundles. Since $\varsigma_{n,h} = \varsigma_{h,n}$ only the metrics $\varsigma_{n,h}$ are shown, where $n < h$. The metrics for which $\varsigma_{n,h} > \varsigma_\tau$* are coloured **green**, otherwise they are coloured **red**.

4.3 Summary

This chapter introduced the general framework for our *sparse trajectory segmentation* technique. The segmentation task is broken down into two stages; an *initialization* and a *refinement* stage. The *initialization* stage is used to obtain a rough estimate of the trajectory segmentation, which is subsequently improved in the *refinement* stage. The *refinement* stage introduces spatial and temporal constraints on the trajectory bundles. This chapter discuss the *initialization* stage in detail, while the details of the *refinement* stage are exposed in chapter 5.

In the *initialization* stage the trajectory data is expressed in a new feature space that allows trajectories with similar motion to be clustered. We perform trajectory clustering in this space using the Mean shift [29] algorithm. To reduce over segmentation introduced from the clustering step, we then proceed to merge the trajectory bundles. We model the 2D motion of the trajectory bundles using Affine transformations, and merge bundles that have similar motion models.

5

Sparse Trajectory Segmentation: Model Refinement with Spatial Smoothness

The label field obtained from the *initialization* stage discussed in the previous chapter provides a reasonable guess of trajectory segmentation. However, in this initial segmentation there are no spatial constraints on the trajectories in each bundle. Hence fig. 5.1 shows an example from the *Calendar and Mobile* sequence where the lack of these spatial constraints prevent the trajectories in the bundles from being confined to specific image regions. For example, the area occupied by the calendar is an image region where it is desired to have only green trajectories. In this region there are unwanted background (red) and train (yellow) trajectories. Also we have unwanted green trajectories in the background. The *refinement* stage discussed in this chapter, introduces spatial constraints to make the bundles represent specific image regions.

The first step in the *refinement* stage involves relabelling the trajectories according to a posterior defined on the label field. This posterior distribution uses a MRF prior for enforcing spatial smoothness on the trajectory labels. The refined label field is then obtained as the MAP estimate generated using the α-*expansion* Graphcut optimization algorithm [135].

A *Model relaxation* step follows the GC optimization. In this step another attempt is made to merge the trajectory bundles, as their motions are more coherent. The framework for merging is as outlined in section 4.2, with a change in the similarity metric. The new similarity metric here is based on the separation of the trajectories in the bundles. This metric unlike the one defined in section 4.2, facilitates for the merging of bundles generated by the motion of non-rigid objects.

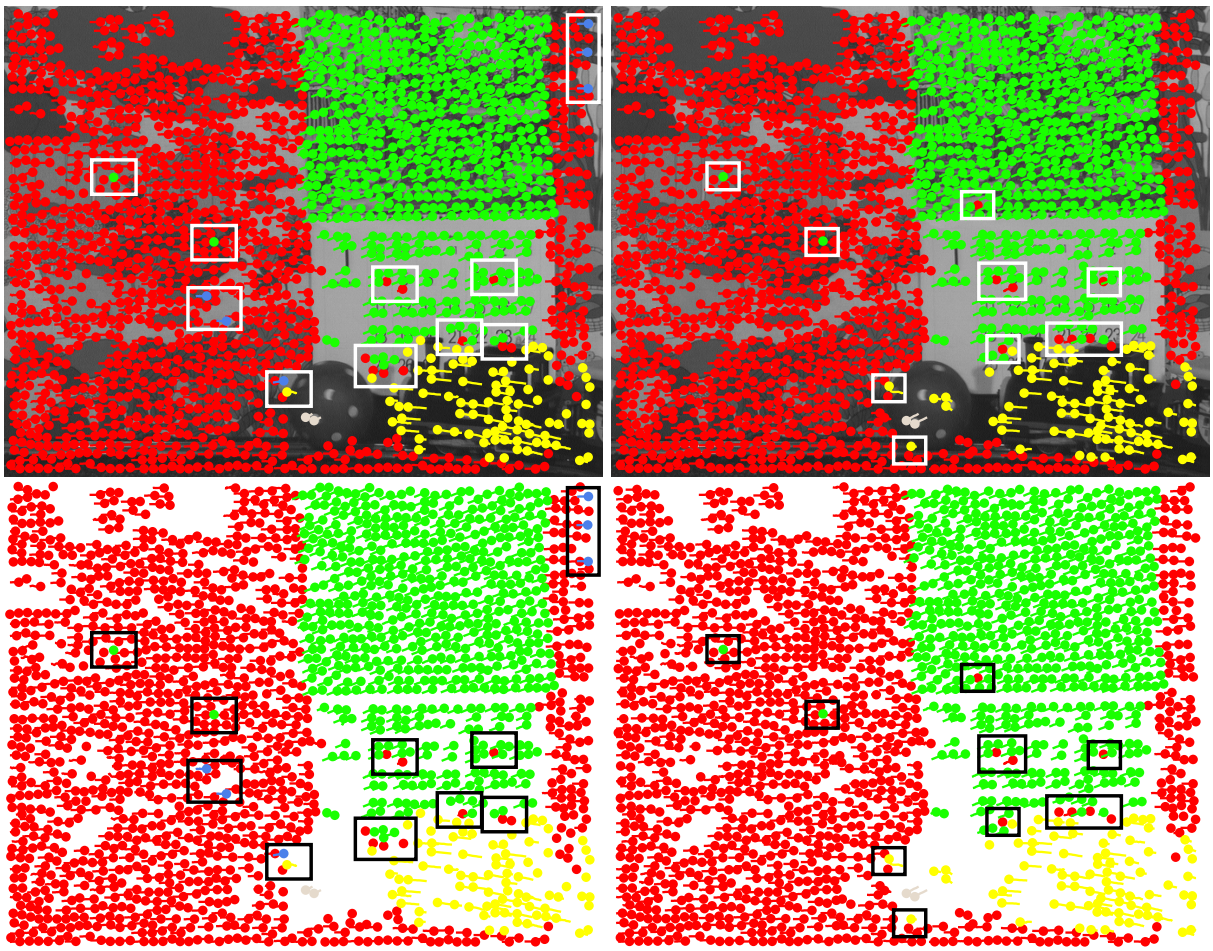

Figure 5.1: *The effect of no spatial constraints in the* ***initialization*** *stage are demonstrated with frames 22 (left) and 25 (right) of the* ***Calendar and Mobile*** *sequence.* ***Top row****: The white boxes highlight trajectories that lie outside the desired image regions of their corresponding bundles. Here all trajectories with the same colour belong to a particular bundle.* ***Bottom row****: The illustration in the top row displayed on a white background for clarity.*

The MRF prior included in the posterior distribution design mentioned previously encourages neighbouring trajectories to have the same label. The illustration on the left of fig. 5.2 shows a set of trajectories that all belong to the same bundle. The MRF constraint correctly allows these trajectories to have the same label in spatially separated groups. Each group is outlined with a closed contour. Note that the groups are separated spatially by trajectories belonging to other bundles. That is, there are trajectories belonging to other bundles that exist in the gaps between these groups.

Even though having all these spatially separated groups of trajectories belonging to the same bundle does not violate the MRF smoothness constraint, we require that every group be a separate bundle. By doing this, we can fulfill the requirement that a bundle represents a single

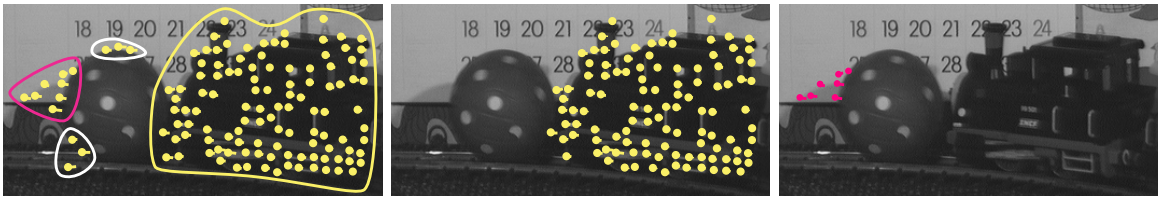

Figure 5.2: *The **local region constraint** algorithm uses Delaunay triangulations to find networks of connected trajectories in the bundle on the left. The bundles in the center and right are the two sets of connected trajectories obtained from this analysis. These two set of trajectories are the new bundles generated by the algorithm.*

image region.

We use a novel algorithm (*local region constraint* algorithm) that performs Delaunay triangulations at every frame to identify each group of connected trajectories. The algorithm then forms a new bundle from each group identified with more than three trajectories. The middle and right illustrations in fig. 5.2 show the two groups of trajectories that are used to form new bundles from the initial bundle on the left.

The merging of bundles and the formation of new bundles for new image regions in the *Model relaxation* step updates the knowledge of the motion models. Therefore these models are then reestimated and we iterate the GC optimization and Model relaxation steps until there are no changes in the label field.

5.1 Graphcut Optimization

As outlined in the previous section, the GC optimization step relabels the trajectories by generating the MAP estimate for a distribution defined on the label field. Given there are N labels in the initial label field, corresponding to the number of trajectory bundles. The tth trajectory has a label $\mathcal{L}_t \in \{1 : N\}$ according to the bundle it belongs to. The posterior distribution $p(\mathcal{L}_t|.)$ for the trajectory labels is given below.

$$p(\mathcal{L}_t|\mathcal{X}, \mathcal{L}_{\sim t}) \propto p_x(\mathcal{X}_t|\mathcal{L}_t)p_s(\mathcal{L}_t|\mathcal{L}_{\sim t}) \tag{5.1}$$

Where $\mathcal{L}_{\sim t}$ are the neighbouring labels of trajectory $\mathcal{X}_t$ that has the label $\mathcal{L}_t$, $p_x(\mathcal{X}_t|.)$ and $p_s(\mathcal{L}_t|.)$ are the likelihood and MRF prior distributions respectively. The likelihood distributions for the N possible labels are dependent on the member trajectories in the bundles. The design of the likelihood is motivated by observations about the behaviour of trajectories in a bundle. Firstly, trajectories tend to follow the average motion in a bundle in some sense. Secondly, the spread of the trajectories in the image plane tends to be consistent from frame to frame. This point about the consistent spread of the trajectories in each bundle is illustrated in fig. 5.3 using four trajectory bundles from the *Calendar and Mobile* sequence. It may be observed that the texture

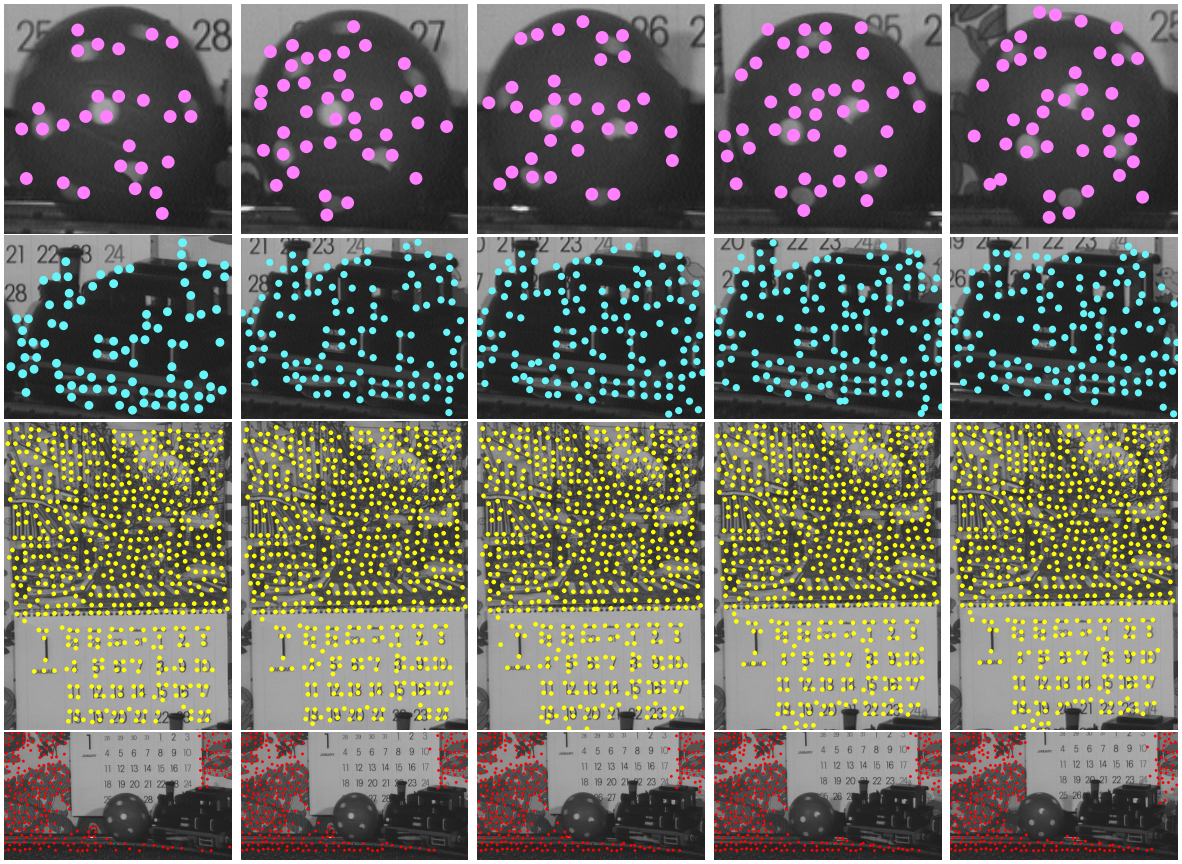

Figure 5.3: *Illustrated in each row are the trajectory bundles for the four independently moving objects in the Calendar and Mobile sequence. Each row shows the endpoints (dots) of the trajectories in the bundles for frames 1,5,10,15 and 20. For every bundle, the spread of the trajectories in the image plane tends to be consistent from frame to frame.*

of the objects in the scene determines the spatial separation of the trajectories. And since the texture of the objects changes slowly from frame to frame the spacing between the trajectories in each bundle is approximately constant. The *top row* shows the trajectories for the ball. Even though the ball is rotating the relative separation between a pair of these trajectories over their paths is roughly constant.

As before, Affine transformation models $\mathbf{A}^n_{f,f+1}$ for the bundles are used to generate a prediction error for the spatial locations of the trajectories. Here the bundle label with the minimum prediction error for a trajectory provides the maximum motion likelihood.

5.1.1 Likelihood Distribution

In this section the design of the likelihood distribution $p_x(\mathcal{X}_t|.)$ is discussed. Consider that the nth trajectory bundle has the corresponding likelihood $p_x(\mathcal{X}_t|\mathcal{L}_t = n)$. As illustrated in the *top row* of fig. 5.4 the likelihood of a trajectory $\mathcal{X}_t$ belonging to a bundle can only be assessed if

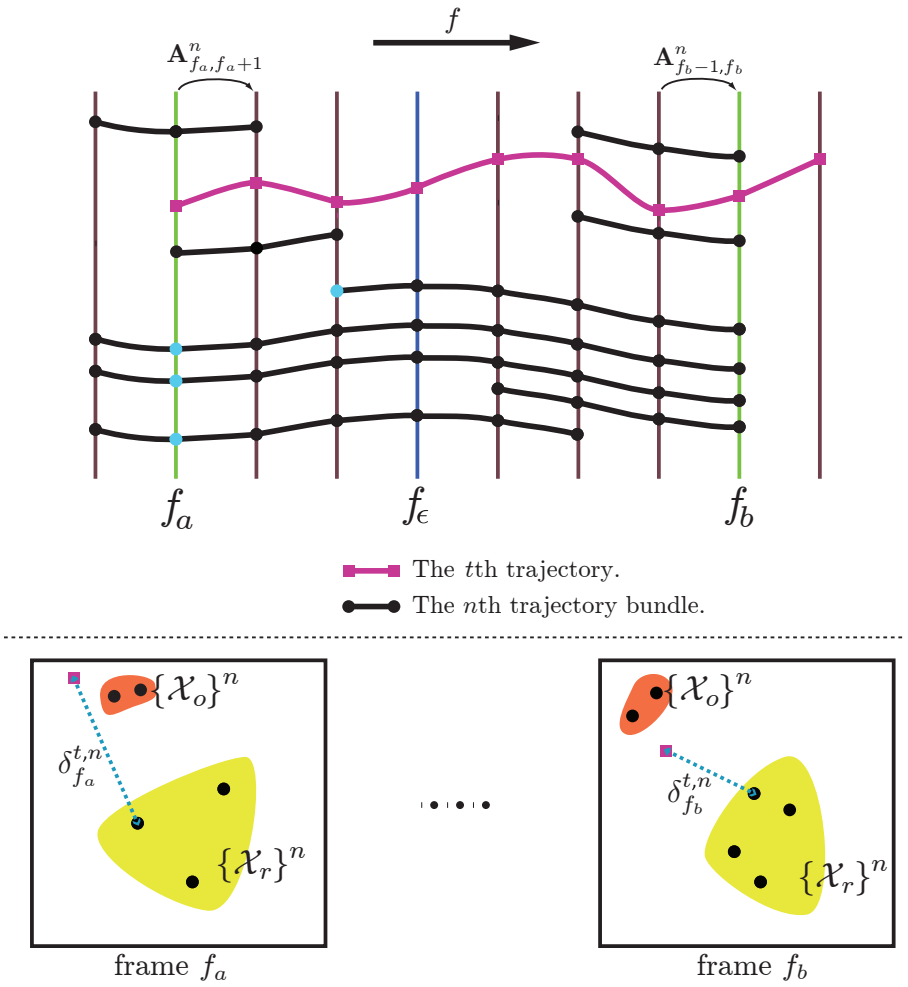

Figure 5.4: *The notation used to discuss the derivation of the likelihood distribution. The* ***top row*** *shows the temporal overlap between the tth trajectory and the nth bundle that starts at f_a and ends at f_b. Also indicated with the* ***blue*** *dots are the starting points used to estimate the motion model error at frame f_ϵ. The* ***bottom row*** *shows the minimum Euclidean distances $\delta^{t,n}_f$ of the tth trajectory to the* ***reference*** *trajectories $\{\mathcal{X}_r\}^n$ in the nth bundle, where the frame index $f \in \{f_a : f_b\}$. The trajectories $\{\mathcal{X}_o\}^n$ in the bundle n are ignored in estimating these distances as they do not confine to the object (yellow image regions) that bundle n represents.*

they both overlap in time. That is, there are frames f_a to f_b for which the trajectory and the bundle both exist. If they do not overlap the likelihood $p_x(\mathcal{X}_t|\mathcal{L}_t = n)$ is zero.

Consider that the set of trajectories in the nth bundle is $\{\mathcal{X}\}^n$, and also the set of Affine transformations for this bundle is $\mathcal{A}_n$. The likelihood of trajectory $\mathcal{X}_t$ belonging to the nth bundle is given as,

$$p_x(\mathcal{X}_t|\mathcal{L}_t = n) = p_m(\mathcal{X}_t|\mathcal{A}_n)p_e(\mathcal{X}_t|\{\mathcal{X}\}^n) \tag{5.2}$$

Where the distribution $p_m(\mathcal{X}_t|.)$ is dependent on how well the average motion of the bundle

describes the motion of the trajectory. The spatial spread distribution $p_e(\mathcal{X}_t|.)$ models the closeness in the image plane of the trajectory $\mathcal{X}_t$ to all the trajectories in the bundle.

5.1.2 Motion Likelihood Term

The likelihood term $p_m(\mathcal{X}_t|.)$ defined previously in eq. 5.2 is a Gaussian distribution of the prediction error for the tth trajectory with respect to the nth bundle. This prediction error ϵ_t^n is defined as,

$$\epsilon_t^n = \sum_{f=f_a+1}^{f_b} \left\| \mathbf{A}_{f_a,f}^n \begin{pmatrix} x_{f_a}^t \\ y_{f_a}^t \\ 1 \end{pmatrix} - \begin{pmatrix} x_f^t \\ y_f^t \\ 1 \end{pmatrix} \right\| \tag{5.3}$$

$$\text{Where } \mathbf{A}_{f_a,f}^n = \prod_{k=f-1}^{f_a} \mathbf{A}_{k,k+1}^n \tag{5.4}$$

The Affine transformation $\mathbf{A}_{f_a,f}^n$ above is the product of the chain of transformations between the start frame f_a and the current frame f. Using this transformation the current spatial location of the trajectory at f is estimated given the starting point $(x_{f_a}^t, y_{f_a}^t)$. The prediction error is the sum of all the estimation errors at the frames $f_a + 1$ to f_b.

The prediction error ϵ_t^n is influenced by two features of trajectories. As desired it is influenced by the motion difference between the trajectory and the Affine motion models. Secondly the model estimation process itself is prone to errors. In other words, the trajectory motion may not be well approximated by Affine motion in the first place. In addition the parameter estimation process itself is inherently noisy.

In order to negate the influence of these effects due to the motion modelling process, we subtract an expected (mean) error from the prediction error ϵ_t^n for the trajectory. This mean error is based on the motion model errors and is estimated from the trajectory in the nth bundle that exist between frames f_a to f_b (fig. 5.4).

Since the prediction error for the tth trajectory is the sum of the error contributions from several frames $f_a + 1$ to f_b, the counterbalancing error is the sum of mean motion model errors for these frames. A mean motion model error $\mu_{f_\epsilon}^n$ is estimated at every frame $f_\epsilon \in \{f_a + 1 : f_b\}$ using the trajectories in the nth bundle. This error $\mu_{f_\epsilon}^n$ accounts for the errors accumulated from the motion transformations going from frame f_a to the current frame f_ϵ. One of these frames f_ϵ is highlighted in the *top row* of fig. 5.4. Given that the kth trajectory starts before frame f_ϵ and is a member of the nth bundle, where there are K of these trajectories, the mean motion

error $\mu^n_{f_\epsilon}$ is given as,

$$\mu^n_{f_\epsilon} = \frac{1}{K}\sum_k [\hat{\epsilon}^n_k] \tag{5.5}$$

$$\text{Where } \hat{\epsilon}^n_k = \left\| \mathbf{A}^n_{f_s,f_\epsilon} \begin{pmatrix} x^k_{f_s} \\ y^k_{f_s} \\ 1 \end{pmatrix} - \begin{pmatrix} x^k_{f_\epsilon} \\ y^k_{f_\epsilon} \\ 1 \end{pmatrix} \right\|,$$

$$f_s = \max(f_a, a_k), \quad \mathbf{A}^n_{f_s,f_\epsilon} = \prod_{j=f_\epsilon-1}^{f_s} \mathbf{A}^n_{j,j+1}$$

The frame f_s above corresponds to the frame at which the starting point for the kth trajectory is defined. The starting points of the trajectories that contribute towards $\mu^n_{f_\epsilon}$ are indicated with *blue* dots in fig. 5.4. Note here that we are using trajectories with varying starting points to estimate the motion error at frame f_ϵ. Since the analysis is being done in the temporal frame window f_a to f_b, if a trajectory starts outside this window the starting point used is the spatial location at frame f_a. Using the actual start a_k of the trajectory would involve the use of motion model transformations outside of this window, which do not influence the prediction error for the tth trajectory. The focus here is only on the errors accumulated due to the model transformations within the analysis window.

In a sense the sum of the mean motion errors $\mu^n_{f_\epsilon}$ over the frames $f_a + 1$ to f_b is a standard error for a bundle introduced by the shortcomings in the motion modelling process itself. Hence this is in effect the mean of the observed error ϵ^n_t for a particular trajectory. We can think of ϵ^n_t to be distributed according to a Gaussian with non-zero mean and variance. But to simplify the discussion we incorporate this mean and variance as a normalising step that follows.

The error difference $\tilde{\epsilon}^n_t$ between the prediction error and the mean error for the tth trajectory with respect to the nth bundle is now defined as,

$$\tilde{\epsilon}^n_t = \epsilon^n_t - \sum_{f_\epsilon=f_a+1}^{f_b} \left[\mu^n_{f_\epsilon}\right] \tag{5.6}$$

Where the prediction error ϵ^n_t was previously given in eq. 5.3.

The confidence we have that $\tilde{\epsilon}^n_t$ is mainly influenced by the difference in motion between the tth trajectory and the motion models for the nth bundle is dependent on the variance in the motion model errors. Given the variables previously defined for the mean motion model error, the standard deviation σ^n_t of the error difference $\tilde{\epsilon}^n_t$ is approximated using the trajectories in the

nth bundle as,

$$[\sigma_t^n]^2 = \sum_{f_\epsilon=f_a+1}^{f_b} \left[\hat{\sigma}_{f_\epsilon}^n\right] \tag{5.7}$$

$$\text{Where } \hat{\sigma}_{f_\epsilon}^n = \frac{1}{K}\sum_k [\hat{\epsilon}_k^n]^2 - \left[\mu_{f_\epsilon}^n\right]^2$$

The standard deviation σ_t^n approaches zero when the motion of the trajectories in the nth bundle are well described by the motion model for that bundle. A low σ_t^n means that there is a high confidence in the motion models.

Hence the motion likelihood term $p_m(\mathcal{X}_t|\mathcal{A}_n)$ given the motion models $\mathcal{A}_n$ for the nth bundle is a Gaussian distribution of the error difference $\tilde{\epsilon}_t^n$ with a mean of zero and standard deviation of σ_t^n. The distribution for $p_m(\mathcal{X}_t|.)$ is given below.

$$\boxed{p_m(\mathcal{X}_t|\mathcal{A}_n) = \frac{1}{\sigma_t^n\sqrt{2\pi}} \exp\left\{-0.5\left[\frac{\tilde{\epsilon}_t^n}{\sigma_t^n}\right]^2\right\}} \tag{5.8}$$

5.1.3 Spatial Spread Likelihood Term

As expressed in eq. 5.2 the likelihood of a trajectory $\mathcal{X}_t$ belonging to bundle n is also proportional to the spatial closeness of $\mathcal{X}_t$ to all the trajectories in this bundle. Here we define a *reference* trajectory as a trajectory in bundle n for which the Euclidean distances of trajectory $\mathcal{X}_t$ is compared. We define 'closeness' as related to the minimum Euclidean distance $\delta_f^{t,n}$ of the points along the t^{th} trajectory to the points of the *reference* trajectories in bundle n. Note that not all the trajectories in bundle n are used for estimating these distances, and we will discuss why later. The *bottom row* of fig. 5.4 shows $\delta_f^{t,n}$ between $\mathcal{X}_t$ and the *reference* trajectories in yellow.

The reference trajectories in a bundle are confined geometrically in the image plane by the object that this bundle represents. Fig. 5.4 shows that there could be other trajectories $\{\mathcal{X}_o\}^n$ which are outside this object simply because of the lack of spatial constraints in *initialization*.

We need some mechanism therefore to select the reference trajectory set $\{\mathcal{X}_r\}^n$ at each iteration for each bundle n. This is discussed next.

5.1.4 Choosing Reference Trajectories

The *top row* of fig. 5.5 shows the trajectory bundles at frame 24 for the background and calendar image regions in the *Calendar and Mobile* sequence after the *initialization* stage. The closed yellow contours outline the regions where the reference trajectories $\{\mathcal{X}_r\}^n$ in these bundles are roughly desired to be. However because of the lack of spatial constraints in the *initialization* stage as previously discussed, some trajectories lie outside of these desired regions, and they are highlighted with white rectangular boxes. These outside trajectories $\{\mathcal{X}_o\}^n$ (see bottom row of

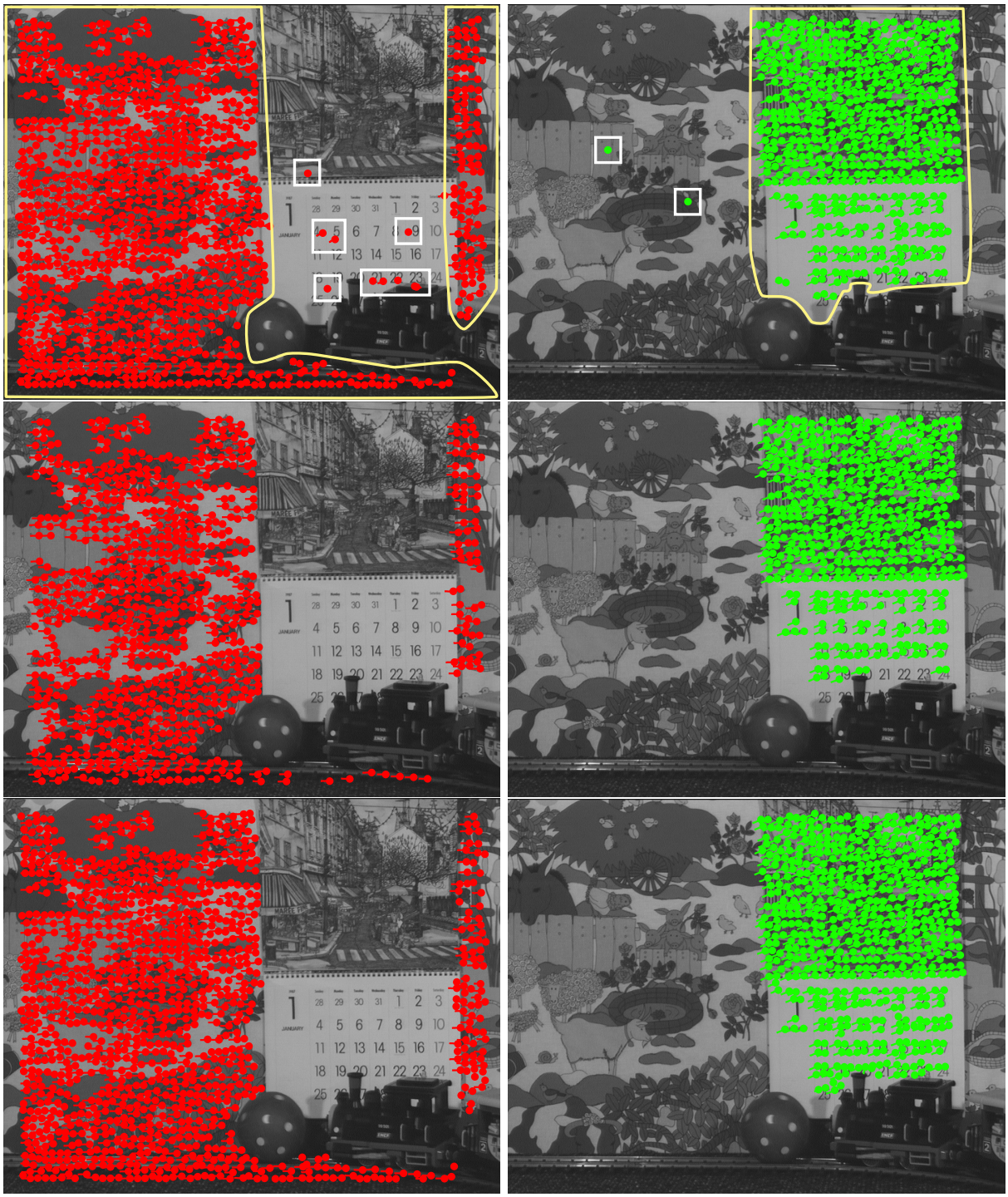

Figure 5.5: *The selection of the spatial reference trajectories $\mathcal{X}_r$ for the background and calendar bundles.* ***Top row****: The trajectories in these bundles after the* ***initialization*** *stage. The desired reference trajectories are outlined with the closed yellow contours. Outside trajectories $\{\mathcal{X}_o\}^n$ are highlighted with white boxes.* ***Middle and bottom rows****: The reference trajectories for the first and second iterations of the* ***refinement*** *stage.*

fig. 5.4, where outside trajectories are in orange regions) are not confined to the geometrical shape of the corresponding objects.

Trajectories $\{\mathcal{X}_o\}^n$ tend to be of shorter duration than $\{\mathcal{X}_r\}^n$. In this example the trajectories $\{\mathcal{X}_o\}^n$ were only 3 frames in duration (fig. 5.5). Hence for the 1^{st} iteration of *refinement*, $\{\mathcal{X}_r\}^n$ is created by selecting the longest trajectories in the bundle. In the example of the 25-frame sequence (*Calendar and Mobile*) in fig. 5.5, we select only trajectories of length 25 frames, as the reference set.

The result of configuring $\{\mathcal{X}_r\}^n$ in this way is shown in the *middle row* of fig. 5.5. As it can be seen all the 'outside' trajectories have been removed and the remaining *reference* trajectories $\{\mathcal{X}_r\}^n$ are better related to the object extent.

However, after iteration 1 of *GC optimization*, spatial constraints will have been applied and it is sensible to incorporate more trajectories in the reference set. In this case we select all trajectories that are best described by the average motion in a bundle. The idea here is that if the motion of trajectory $\mathcal{X}_q$ is better described by a bundle other than the one it belongs to it is not a good candidate for a reference trajectory.

Hence trajectory $\mathcal{X}_q$ is selected as a reference trajectory $\mathcal{X}_r$ for bundle n if the motion likelihood satisfies the condition $p_m(\mathcal{X}_q|\mathcal{A}_n) > p_m(\mathcal{X}_q|\mathcal{A}_h)$, where $\mathcal{A}_h$ are the set of motion models for the other bundles. Using this strategy, the reference trajectories selected for the background and calendar bundles are shown in the *bottom row* of fig. 5.5. It can be observed that these reference trajectories describe the geometry of the objects well. Also there are more reference trajectories here than those obtained from the longest trajectory strategy for the 1^{st} iteration. With more reference trajectories there is a higher possibility that trajectory $\mathcal{X}_t$ will have closer reference trajectories to it.

5.1.5 Spatial Spread Distribution

With the reference trajectories $\mathcal{X}_r$ for bundle n selected as described in the previous section, we can now define the spatial spread distribution $p_e(\mathcal{X}_t|.)$. This distribution is assumed to be a Gaussian distribution of the average distance in the image plane of trajectory $\mathcal{X}_t$ to the reference trajectories $\{\mathcal{X}_r\}^n$.

For the overlapping frames $f \in \{f_a : f_b\}$ (fig. 5.4) of trajectory $\mathcal{X}_t$ and the nth bundle, the tth trajectory has at each frame a minimum distance $\delta_f^{t,n}$ to the reference trajectories $\{\mathcal{X}_r\}^n$ in the nth bundle. Since there is a distance $\delta_f^{t,n}$ for every frame, the average of these minimum distances represents the distance of the trajectory to the bundle. The average distance of the tth trajectory to the nth bundle δ_t^n is therefore defined as,

$$\delta_t^n = \frac{1}{f_b - f_a + 1} \sum_{f=f_a}^{f_b} \left[\delta_f^{t,n}\right] \tag{5.9}$$

The parameters of the Gaussian distribution $p_e(\mathcal{X}_t|.)$ are estimated from the reference trajectories $\{\mathcal{X}_r\}^n$. Consider that there are R reference trajectories. For every reference trajectory

$\mathcal{X}_r$ we calculate the average distance $\tilde{\delta}_r^n$ to the other reference trajectories. Here the distance $\tilde{\delta}_r^n$ is obtained in a similar way to the distance δ_t^n calculated in eq. 5.9 for trajectory $\mathcal{X}_t$. Hence with these average distances for the reference trajectories, the mean $\bar{\delta}_n$ and standard deviation σ_n of the distribution $p_e(\mathcal{X}_t|.)$ are estimated as follows.

$$\bar{\delta}_n = \frac{1}{R}\sum_r \left[\tilde{\delta}_r^n\right] \tag{5.10}$$

$$\sigma_n = \frac{1}{R}\sum_r \left[\tilde{\delta}_r^n\right]^2 - \left[\bar{\delta}_n\right]^2 \tag{5.11}$$

The mean $\bar{\delta}_n$ and standard deviation σ_n above are global parameters that relate to the overall separation of the trajectories in bundle n. However, the separation between the trajectories in the bundle can vary according to local image texture. Therefore by design of the spatial spread likelihood, the probability $p_e(\mathcal{X}_t|.)$ only decreases as the distance δ_t^n of trajectory $\mathcal{X}_t$ moves away from the mean separation $\bar{\delta}_n$ ($\delta_t^n > \bar{\delta}_n$). A trajectory is not penalized for being too close the reference trajectories. Here 'too close' means that $\delta_t^n \leq \bar{\delta}_n$.

Hence the spatial spread likelihood $p_e(\mathcal{X}_t|.)$ is given as,

$$\boxed{p_e(\mathcal{X}_t|\{\mathcal{X}\}^n) = \begin{cases} \frac{1}{\sigma_n\sqrt{2\pi}} \exp\left\{-0.5\left[\frac{\delta_t^n - \bar{\delta}_n}{\sigma_n}\right]^2\right\}, & \text{if } \delta_t^n > \bar{\delta}_n \\ \frac{1}{\sigma_n\sqrt{2\pi}}, & \text{if } \delta_t^n \leq \bar{\delta}_n \end{cases}} \tag{5.12}$$

5.1.6 The Prior

This section addresses the MRF prior $p_s(\mathcal{L}_t|\mathcal{L}_{\sim t})$ used in the posterior distribution to enforce smoothness on the label field. The use of a MRF prior to enforce spatial smoothness in 2D segmentation at the pixel scale is well established [33, 68, 69]. However there has not been any attempts at using a MRF for the segmentation of sparse trajectories. The major challenge here specifying a neighbourhood system for the sparse trajectories.

The conventional use of a MRF to enforce label smoothness requires that a neighbourhood be defined for each trajectory $\mathcal{X}_t$. Here the label of trajectory $\mathcal{X}_t$ is influenced only by the labels of the trajectories in this neighbourhood. Consider that trajectory $\mathcal{X}_t$ has a label $\mathcal{L}_t$, the prior on the trajectory belonging to bundle n given the labels $\mathcal{L}_{\sim t}$ in the neighbourhood, is defined as $p_s(\mathcal{L}_t = n|\mathcal{L}_{\sim t})$.

Recall that there are N trajectory bundles. This prior for the nth bundle $p_s(\mathcal{L}_t = n|.)$ is modelled as a Gibbs distribution given below.

$$p_s(\mathcal{L}_t = n|\mathcal{L}_{\sim t}) = \frac{1}{Z}\exp\left\{-\lambda_{\mathcal{L}}\sum_c (n \neq \mathcal{L}_c)v_t^c\right\} \tag{5.13}$$

Where Z above is the usual normalization constant. Also c is the index of a trajectory $\mathcal{X}_c$ in the neighbourhood of trajectory $\mathcal{X}_t$, and the set of all these trajectories is $\{\mathcal{X}_c\}^t$. The weight

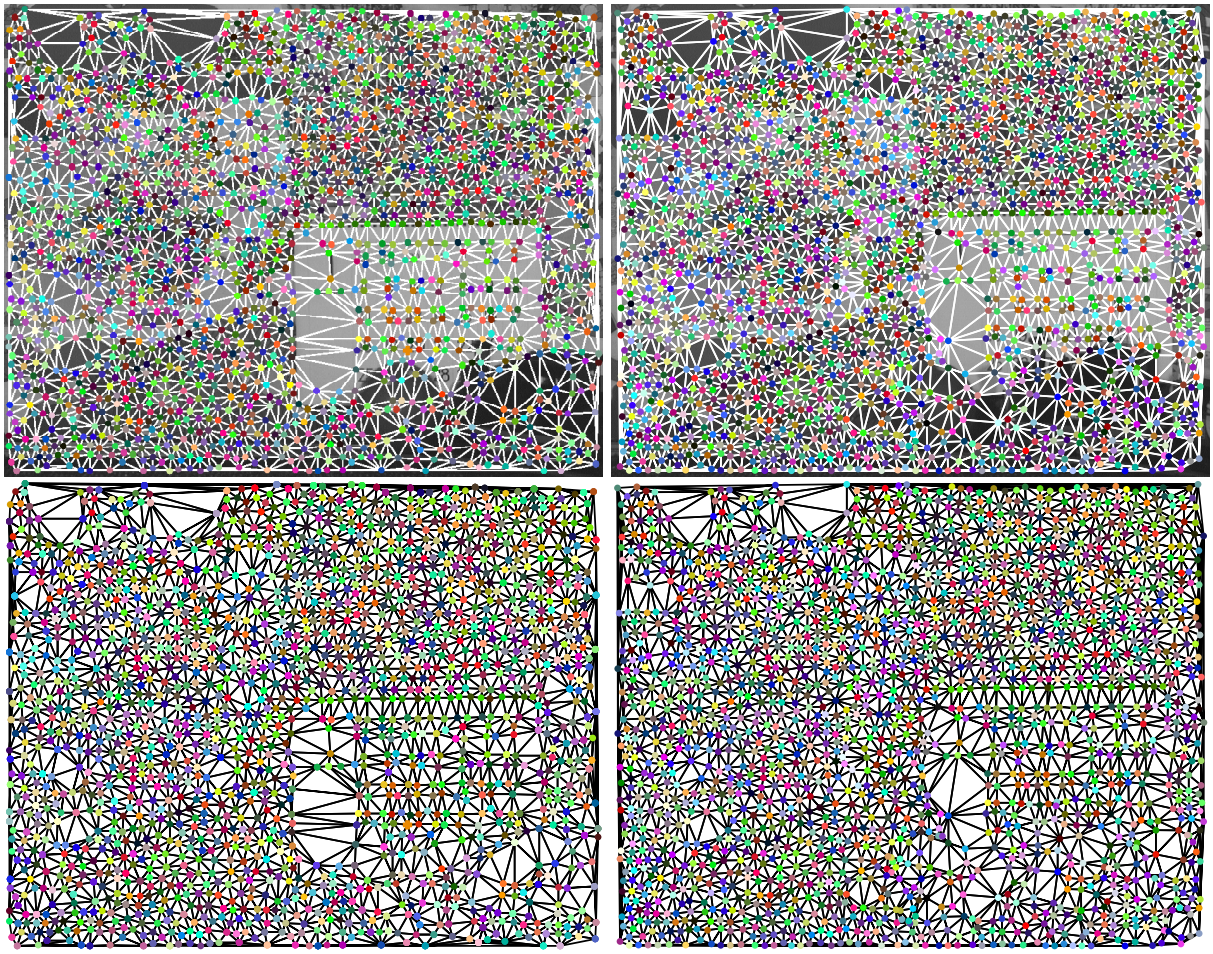

Figure 5.6: *The Delaunay triangulations for frames 1 (left) and 25 (right) of the* ***Calendar and Mobile*** *sequence. The* ***top row*** *shows the triangulation superimposed on the frames themselves, while in the* ***bottom row*** *they are superimposed on a white background.*

v_t^c determines how much trajectory $\mathcal{X}_t$ is influenced by the neighbouring trajectory $\mathcal{X}_c$. The constant $\lambda_{\mathcal{L}}$ controls the weight this prior has in the MAP solution. For all experiments it was found that setting the constant $\lambda_{\mathcal{L}} = 0.02$ gave suitable results. The design of the weight v_t^c will be discussed later in this chapter.

The next section discusses how the neighbourhood trajectories $\{\mathcal{X}_c\}^t$ are obtained.

5.1.7 Trajectory Neighbourhoods

Unlike image pixels which conform to a convenient rectangular lattice sthetructure, the points along trajectories do not conform to a regular geometric structure. We therefore perform Delaunay triangulations [83] at every frame in a sequence to define a neighbourhood structure for the trajectories. Fig. 5.6 shows the Delaunay triangulations for frames 1 and 25 of the *Calendar and Mobile* sequence. At every frame over the duration of the trajectory $\mathcal{X}_t$ it may be connected to

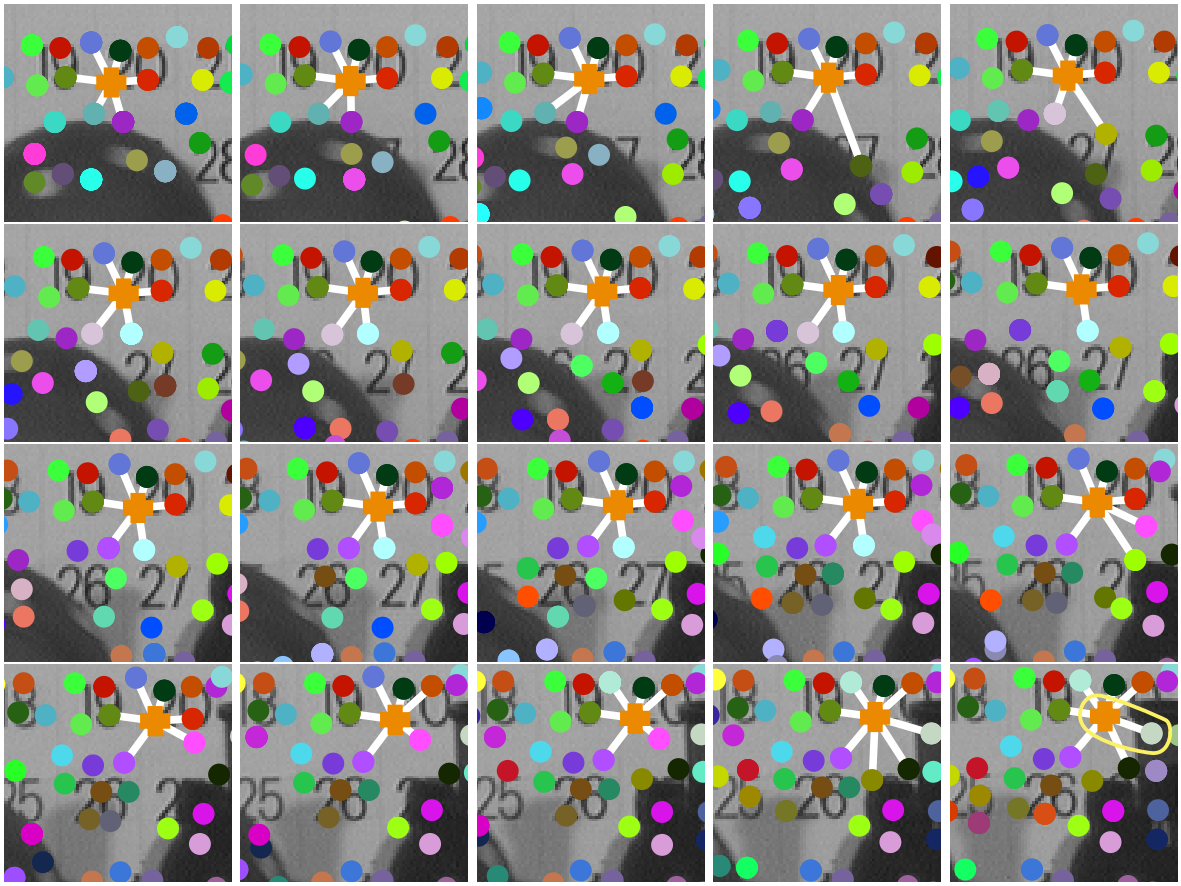

Figure 5.7: *The neighbourhood structure for trajectory $\mathcal{X}_t$ (orange plus) on the calendar, where the duration of this trajectory is 20 frames. The endpoints (**dots**) of each trajectory close to $\mathcal{X}_t$ are illustrated as having the same colour from frame to frame. The neighbours of trajectory $\mathcal{X}_t$ are connected to it with **white lines**. The **top row** shows the neighbourhood for frame 1-5, while the **bottom row** corresponds to frames 16-20. The other rows show the intermediate frame 6-15 in the order previously outlined.*

various trajectories as the triangulations involving this trajectory change from frame to frame. Therefore, the trajectories $\{\mathcal{X}_c\}^t$ in the neighbourhood of trajectory $\mathcal{X}_t$ are defined as all the trajectories connected to $\mathcal{X}_t$ via all these Delaunay triangulations.

An example of the neighbourhood for a trajectory $\mathcal{X}_t$ of length 20 frames is shown in fig. 5.7. The points at each frame along this trajectory $\mathcal{X}_t$ are indicated with an *orange* '+', while the other trajectories close by are indicated with different coloured *dots*. Note that at each frame shown in a different picture, the trajectories connected to $\mathcal{X}_t$ vary as they appear and disappear.

Trajectory $\mathcal{X}_t$ in fig. 5.7 is located on the calendar which determines the motion of $\mathcal{X}_t$. Note that not all of the neighbours of trajectory $\mathcal{X}_t$ are trajectories from the calendar. For example, at frame 20 (*bottom right* picture of fig. 5.7) the trajectory $\mathcal{X}_t$ has a neighbouring trajectory from the train, and this is highlighted with the yellow closed contour. We require that trajectories

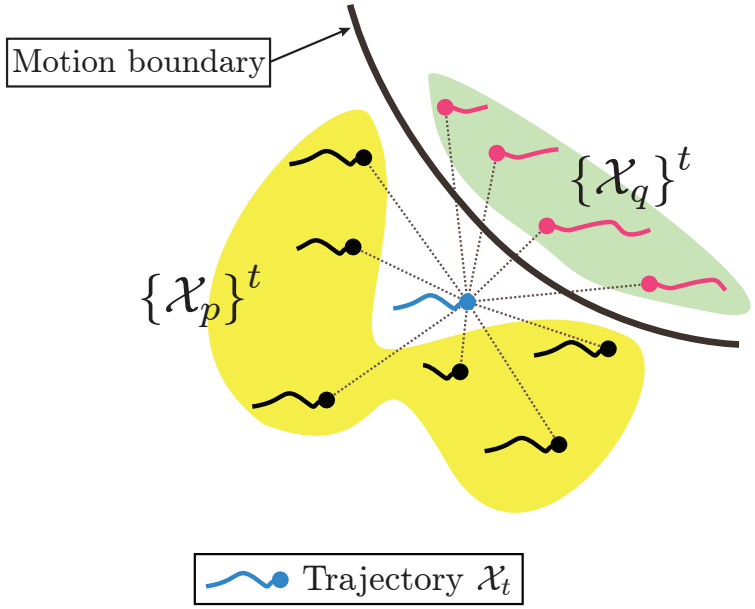

Figure 5.8: *The neighbourhood for trajectory $\mathcal{X}_t$. All the trajectories in the neighbourhood that are moving in the general direction of $\mathcal{X}_t$ form the trajectory set $\{\mathcal{X}_p\}^t$, and they are outlined in yellow. Only these trajectories influence the prior defined on the label $\mathcal{L}_t$ for trajectory $\mathcal{X}_t$. The trajectories moving in a different direction to $\mathcal{X}_t$ are the members of the set $\{\mathcal{X}_q\}^t$, and they are outlined in green. These trajectories are considered to lie across a motion boundary, therefore they have no influence on the prior for $\mathcal{L}_t$.*

which lie across a motion boundary do not influence each other.

The next section discusses how the trajectories in the neighbourhood $\{\mathcal{X}_c\}^t$ that have a different motion to trajectory $\mathcal{X}_t$ are identified.

5.1.7.1 Trajectory Motion Boundaries

Trajectories in the neighbourhood of trajectory $\mathcal{X}_t$ moving in a different direction to $\mathcal{X}_t$ should not influence the prior for the label $\mathcal{L}_t$. Fig. 5.8 illustrates the idea of trajectory $\mathcal{X}_t$ having neighbours that lie across a motion boundary. The neighbouring trajectories $\{\mathcal{X}_q\}^t$ in the green outline are moving in a different direction to trajectory $\mathcal{X}_t$. Here we require that the label $\mathcal{L}_t$ be only influenced by the trajectories $\{\mathcal{X}_p\}^t$ that are in the same object as $\mathcal{X}_t$, and they are outlined in yellow.

The overall direction of a neighbouring trajectory $\mathcal{X}_c$ relative to $\mathcal{X}_t$ is determined from the angles between the frame to frame displacements of both trajectories. Consider that both trajectories overlap temporally between frames f_1 to $f_2 + 1$, as shown in the *top row* of fig. 5.9. For trajectory $\mathcal{X}_t$ the displacement vector along the path between frames f and $f + 1$ is defined

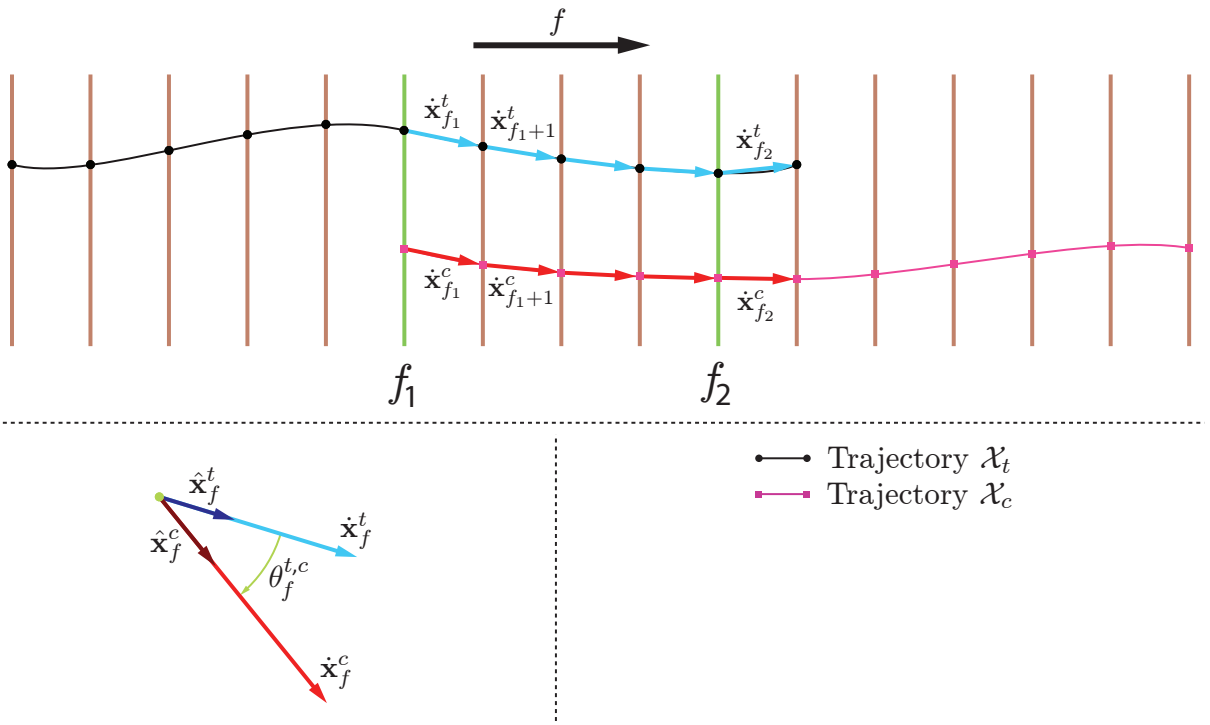

Figure 5.9: *The temporal overlap of trajectory $\mathcal{X}_t$ with a neighbouring trajectory $\mathcal{X}_c$.* ***Top row****: These trajectories overlap from frames f_1 to f_2+1, and the displacement vectors between adjacent frames are $\dot{\mathbf{x}}^t_f$ and $\dot{\mathbf{x}}^c_f$ respectively.* ***Bottom row****: The angle $\theta^{t,c}_f$ between the displacement vectors at frame f. The vectors of unit length along these displacement vectors are $\hat{\mathbf{x}}^t_f$ and $\hat{\mathbf{x}}^c_f$.*

as $\dot{\mathbf{x}}^t_f$. This displacement vector $\dot{\mathbf{x}}^t_f$ is given below.

$$\dot{\mathbf{x}}^t_f = \begin{pmatrix} x^t_{f+1} \\ y^t_{f+1} \end{pmatrix} - \begin{pmatrix} x^t_f \\ y^t_f \end{pmatrix} \tag{5.14}$$

$$\hat{\mathbf{x}}^t_f = \frac{\dot{\mathbf{x}}^t_f}{\left\|\dot{\mathbf{x}}^t_f\right\|}$$

The vector $\hat{\mathbf{x}}^t_f$ is a unit vector in the direction of $\dot{\mathbf{x}}^t_f$. The displacement $\hat{\mathbf{x}}^c_f$ and unit vectors $\dot{\mathbf{x}}^c_f$ are similarly defined for the neighbouring trajectory $\mathcal{X}_c$.

The dot product of the unit vectors $\hat{\mathbf{x}}^t_f$ and $\hat{\mathbf{x}}^c_f$ for both trajectories produces the cosine of the angle $\theta^{t,c}_f$ between the displacement vectors at frame f. This angle $\theta^{t,c}_f$ is shown in the *bottom row* of fig. 5.9. The trajectories $\mathcal{X}_t$ and $\mathcal{X}_c$ are considered to be moving in the same directions at frame f if one of the following two conditions are satisfied.

- The angle $\theta^{t,c}_f$ between the displacement vectors is less than a threshold $\theta_T = 60°$.
- The displacement vectors $\dot{\mathbf{x}}^t_f = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ and $\dot{\mathbf{x}}^c_f = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$.

The percentage of frames over the range f_1 to f_2 for which both trajectories $\mathcal{X}_t$ and $\mathcal{X}_c$ are moving in the same direction according to the conditions above is now defined as $\zeta_{t,c}$.

Recall that the trajectories $\{\mathcal{X}_p\}^t$ are the only trajectories in the neighbourhood of $\mathcal{X}_t$ that we require to influence the prior on the label $\mathcal{L}_t$ (see fig. 5.8). The metric $\zeta_{t,c}$ calculated for the neighbouring trajectory $\mathcal{X}_c$ with respect to trajectory $\mathcal{X}_t$ is used to decide whether $\mathcal{X}_c$ a member of the set $\{\mathcal{X}_p\}^t$. We require $\zeta_{t,c}$ to be greater than 60%, for trajectory $\mathcal{X}_c$ to be included in the set $\{\mathcal{X}_p\}^t$. That is, trajectory $\mathcal{X}_t$ and $\mathcal{X}_c$ are deemed to be moving in the same direction for at least 60% of the frame in their temporal overlap f_1 to $f_2 + 1$.

The next section discusses how the influence of the trajectories $\{\mathcal{X}_p\}^t$ is quantified.

5.1.7.2 The Influence of Neighbouring Trajectories

Recall that $\{\mathcal{X}_c\}^t$ is the set of all the trajectories in the neighbourhood of trajectory $\mathcal{X}_t$. Further recall that the influence of each neighbouring trajectory $\mathcal{X}_c$ on the prior $p_s(\mathcal{L}_t = n|.)$ is controlled by the weight v_t^c in eq. 5.13.

The previous section discussed how the trajectories $\{\mathcal{X}_c\}^t$ in the neighbourhood are partitioned into two sets $\{\mathcal{X}_p\}^t$ and $\{\mathcal{X}_q\}^t$. At this stage it is established that the trajectories $\{\mathcal{X}_q\}^t$ do not influence the label of trajectory $\mathcal{X}_t$. Therefore for these trajectory the weight v_t^q is set to zero.

The set of neighbouring trajectories $\{\mathcal{X}_p\}^t$ are required to influence the prior for label $\mathcal{L}_t$. We make the influence of a trajectory $\mathcal{X}_p$ in this set proportional to the average Euclidean separation of $\mathcal{X}_p$ to trajectory $\mathcal{X}_t$ in the image plane. Trajectories in $\{\mathcal{X}_p\}^t$ that are 'closer' to trajectory $\mathcal{X}_t$ have a greater influence on the prior for $\mathcal{L}_t$.

Hence the weight v_t^p for a trajectory in $\{\mathcal{X}_p\}^t$ is given below.

$$v_t^p = \exp\left\{-\left[\frac{\bar{\delta}_{t,p}}{D_e}\right]^2\right\} \tag{5.15}$$

$$\bar{\delta}_{t,p} = \frac{1}{f_2 - f_1 + 1}\sum_{f=f_1}^{f_2}\left[\sqrt{(x_f^t - x_f^p)^2 + (y_f^t - y_f^p)^2}\right]$$

Where the normalizing factor above $D_e = \sqrt{200}$ for all experiments reported. This allows trajectories within 200 pixels of $\mathcal{X}_t$ to have great influence.

5.2 Local Region Constraint

This section discusses how each trajectory bundle is constrained to represent a single coherent image region. The spatial smoothness enforced on the trajectory labels by using a MRF prior previously discussed, correctly allows neighbouring trajectories to have the same label. However, the MRF does not offer any control over where these neighbouring trajectories bearing the same

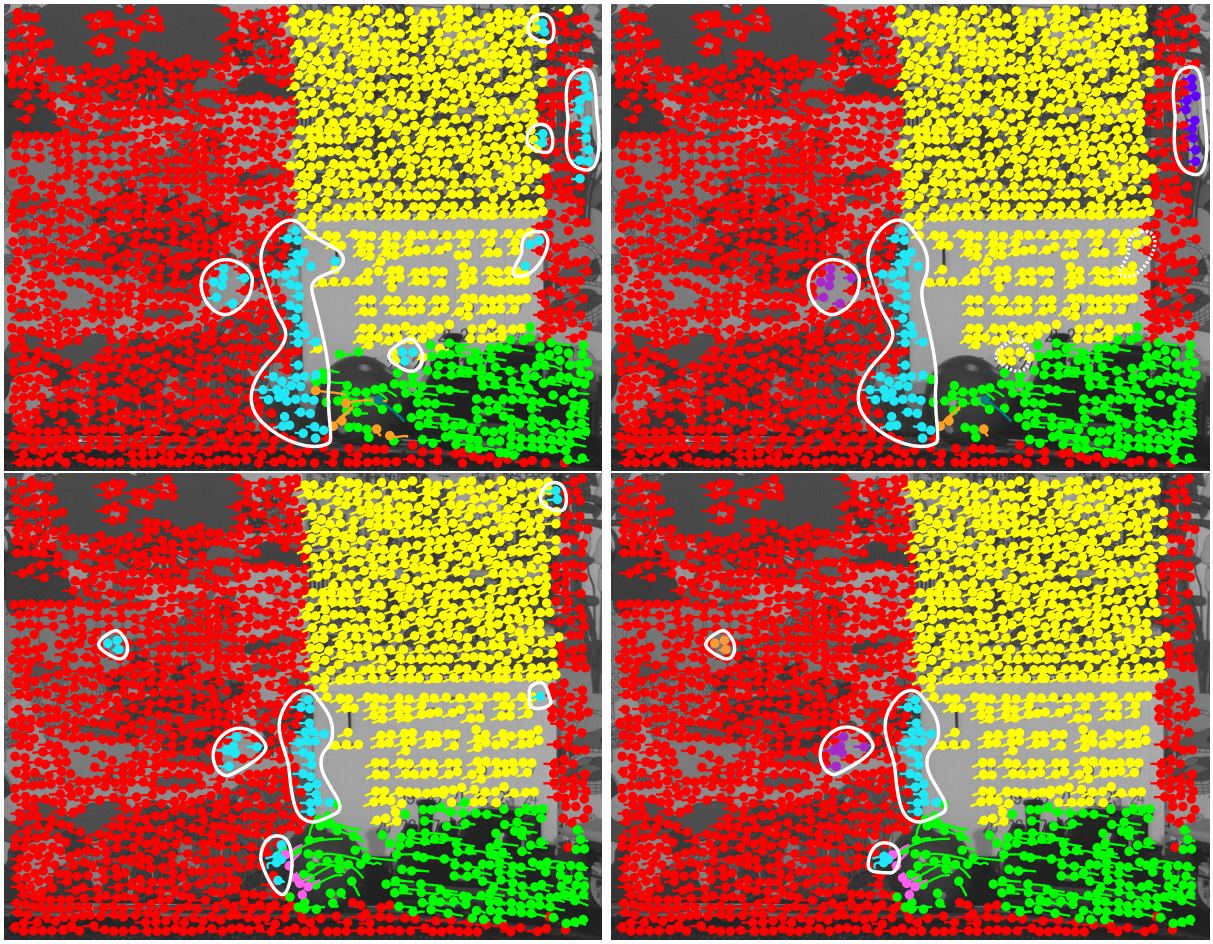

Figure 5.10: *The trajectory labels for frames 12 (**top row**) and 25 (**bottom row**) of the **Calendar and Mobile** sequence. **Left column**: The trajectory label field after solving the MAP estimate of the posterior distribution for the 1^{st} iteration of **refinement**. The white closed contours highlight the trajectories in the **cyan** bundle. The closed contours represent the sub-bundles of trajectories that all belong to the **cyan** bundle, and are spatially separated in the image plane. **Right column**: The results of the **local region constraint** algorithm on the label field. The refined **cyan** bundle as well as the new bundles formed from it and outlined with white closed contours. The trajectories in the new bundles are uniquely coloured. Two of these new bundles are subsequently merged with the calendar bundle in frame 12 (top), and they are outlined with broken white contours.*

label reside in the image plane. We require that all the trajectories belonging to each bundle reside in a single confined image region.

The *left column* of fig. 5.10 shows the trajectory bundles after solving for the MAP estimate (*GC optimization*) for the 1^{st} iteration of the *Calendar and Mobile* sequence. Here frames 12 (top) and 25 (bottom) are shown, and the trajectories for the *cyan* bundle are outlined with white closed contours. These closed contours highlight the spatially separated groups of trajectories

belonging to the *cyan* bundle. We define each group as a sub-bundle. The trajectories in each sub-bundle are all connected via their neighbourhoods. These neighbourhoods are derived from Delaunay triangulations of the points along the trajectories as previously discussed.

We can observe that for the example of the *cyan* bundle on the left of fig. 5.10 that the sub-bundles do not all represent a specific image region. Therefore, we decided that the trajectories in the largest sub-bundle should represent the original bundle. Also, the smaller sub-bundles with more than three trajectories should form new bundles. The algorithm for fulfilling this task is defined as the *local region constraint* algorithm.

The results of the *local region constraint* on the bundles at frame 12 (top) and 25(bottom) are shown in the *right column* of fig. 5.10. Here the *cyan* bundle along with the new bundles formed are outlined with white contours. The trajectories in the new bundles are highlighted with different colour. Note that two of the sub-bundles (for frame 20) from the original *cyan* bundle of the left have been subsequently merged with the calendar bundle in yellow. These sub-bundles are highlighted with broken white contours on frame 12 (top).

As can be seen for the two sub-bundles merged to the calendar bundle, splitting the trajectories into sub-bundles makes the average motion of the bundles more coherent. The merging of bundles is then better facilitated.

The next section discusses how image regions are sparsely represented by trajectory bundles.

5.2.1 Object Representation with Trajectory Bundles

As discussed in the previous section, the trajectories in a bundle after the *GC optimization* step, can occupy image regions corresponding to different objects. We require that the trajectories in each bundle occupy an image region corresponding to a single object. A mechanism is therefore required to identify the image regions a bundle occupies.

The points along the trajectories generated by an object are spatially constrained according to the relative location of the object in the image plane. The top right illustration in fig 5.11 shows the spatiotemporal volume of pixels in an object as they move through time. The trajectories of the object shown top left, are constrained to occupy this spatiotemporal volume. The nature of this spatial constraint causes these trajectories to have an important property of forming a 'closed 3D network' of trajectory points.

The bottom illustration of fig. 5.11 will be used to explain the concept of a 'closed 3D network' of trajectory points. Consider that a bundle has two sets of trajectories (sub-bundles) that are separated in space and time. In fig. 5.11 these sub-bundle are identified as the hth (black) and kth (green) sub-bundles, and they are separated by a spatiotemporal gap. This gap between the sub-bundles is occupied by trajectories from other bundles.

The points (*dots*) along the trajectories in each sub-bundle are connected at each frame by the defined neighbourhood obtained from the Delaunay triangulations as previously discussed. These neighbourhood connections are defined as **spatial links**, and are indicated with *cyan* lines

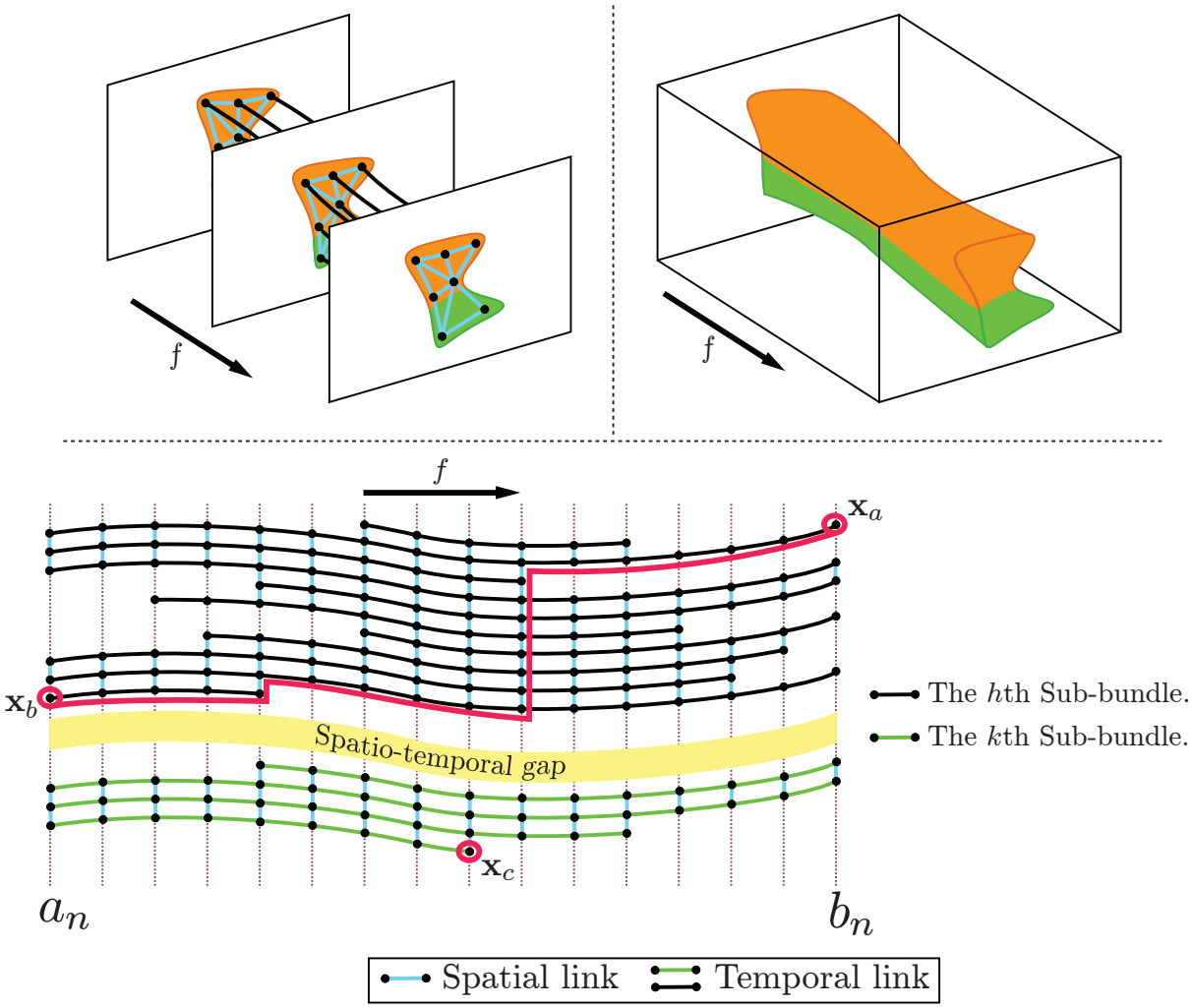

Figure 5.11: ***Top right**: The spatiotemporal volume the pixels of an object occupy as they move through a sequence. **Top left**: The spatiotemporal volume dictated by the movement of an object constrains the trajectories of that object to reside within this volume. **Bottom**: Examples of two sub-bundles h and k which both belong to the same bundle. The sub-bundles are separated in space and time by a spatiotemporal gap that contains trajectories from other bundles. The points $\mathbf{x}_a$ and $\mathbf{x}_b$ both belong to trajectories in sub-bundle h. These points are connected by a set of links (spatial and temporal) outlined with the **pink** line. The point $\mathbf{x}_c$ in sub-bundle k however has no links to connect it to any point in sub-bundle h. Therefore, the sub-bundles are defined as isolated 3D networks of trajectory points.*

between the trajectory points in fig. 5.11. The trajectories themselves provide links between the points from frame to frame, and these links are defined as **temporal links**.

Considering the analogy of the points along the trajectories (*dots*) as nodes in a communication network, the spatial and temporal links are the channels by which these nodes communicate. Now it may be observed that the design of the sub-bundles shown in fig. 5.11 causes each sub-bundle to be an isolated/closed 3D network of points. That is, for two points $\mathbf{x}_a$ and $\mathbf{x}_b$ that belong to sub-bundle h, there is a set of links (spatial and temporal) that allow these points to be connected. An example of a set of links that connect $\mathbf{x}_a$ to $\mathbf{x}_b$ are outlined with the *pink* line. However, for points $\mathbf{x}_a$ and $\mathbf{x}_c$ that reside on different sub-bundle there are no links that allow these points to be connected.

Therefore, a sub-bundle is defined as a closed 3D network of trajectory points. The wire-frame structure implicitly inferred by the links connecting the trajectory points is an approximation of a single image region as it moves through time.

The next section discusses how trajectory sub-bundles are identified for each bundle.

5.2.2 Locating Trajectory Sub-bundles

The previous section established the idea that a trajectory sub-bundle is a closed 3D network of trajectory points (bottom row of fig. 5.11). The network for a sub-bundle is termed 'closed' since only trajectory points in the sub-bundle can be connected to each other. Furthermore there are no links that exist to connect two points from separate sub-bundles.

We take advantage of the closed 3D network of each sub-bundle to identify them. The idea here is if a trajectory point $\mathbf{x}_r$ is chosen at random and given a unique 'message' which is then propagated over the network, only the points in a single sub-bundle will receive the 'message'. This sub-bundle will be the sub-bundle that the random point $\mathbf{x}_r$ belongs to. The other sub-bundles are discovered in a similar way by choosing another random point $\mathbf{x}_r$ that has not yet been associated with a sub-bundle.

Sub-bundles with more than three trajectories are used as previously mentioned to form new bundles. The sub-bundle with the most trajectories is assigned the label of the bundle it was derived from.

Appendix C gives the details of how the trajectories in a bundle are assigned to a particular sub-bundle. This may seem like a simple task, however designing an efficient algorithm requires some through.

5.3 Trajectory Bundle Merging

We merge trajectory bundles in order to keep the number of bundles to a minimum, which reduces modelling redundancy and improves computational efficiency. In section 4.2 (*initialization* stage) we presented a framework for merging trajectory bundles by assessing motion

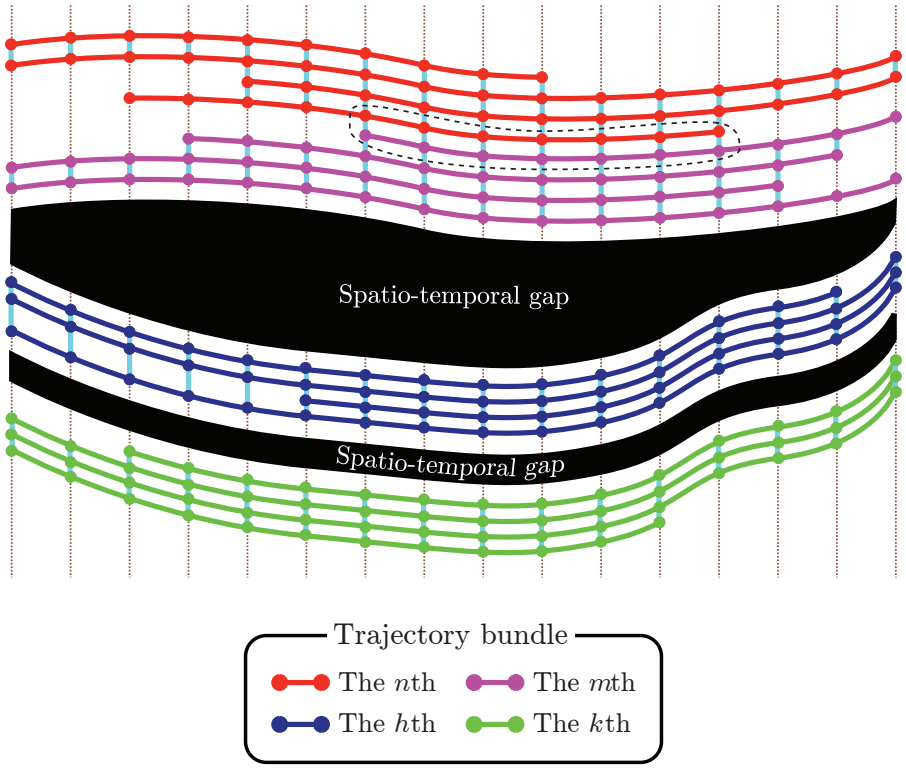

Figure 5.12: *Examples of which trajectory bundles are considered for merging. The nth (red) and mth (purple) trajectory bundles are considered for merging since their combined bundle forms a closed 3D network of points. The spatial links (cyan lines) highlighted with the broken black closed contour facilitate 'communication' between the points in both bundle n and m. However bundles h (blue) and k (green) are not considered for merging since both bundles are separated by a spatiotemporal gap.*

similarity based on Affine models. We used Affine motion models during the *initialization* stage to facilitate the merging of trajectory bundles corresponding to rigidly moving objects. However this strategy was not effective for merging trajectory bundles corresponding to non-rigid objects. Hence an alternative merging strategy is used during the *refinement* stage to facilitate the merging of these bundles. We attempt to merge these bundles in the *refinement* stage after spatiotemporal constraints on the bundles have been applied using the Bayesian framework.

The trajectory bundle merging strategy used during the *refinement* stage is defined as *spatial* merging. First, we consider merging the nth and mth bundles if the trajectories in both bundles will reside in a single coherent spatiotemporal volume. That is, the merged trajectory bundle should represent a single image region throughout a sequence.

As previously discussed we assess whether a bundle represent a single image region by identifying its sub-bundles. Recall that trajectory bundles with multiple sub-bundles do not represent a single image region, so each sub-bundle is used to form a new bundle. Hence the idea is to

consider the merging of bundles n and m if they can be combined into a single bundle. We then merge these bundles if a second criterion based on the spatial separation of the trajectories in the merged bundle is met. We will discuss this second criterion later.

Recall that the points along the trajectories are connected in some way by **temporal** and **spatial** links as previously discussed. The trajectories themselves provide the **temporal** links, while the **spatial** links are obtained from Delaunay triangulations of the points along the trajectories. Fig. 5.12 shows examples of which trajectory bundles are considered for merging. Here bundles n (red) and m (purple) are considered for merging since there are **spatial** links (cyan lines) that connect both bundles. These links are highlighted with the broken black closed contour. Hence the bundle obtained from combining bundles n and m forms a single sub-bundle which is a closed 3D network of trajectory points. However, bundles h (blue) and k (green) are not considered for merging since a spatiotemporal gap resides between both bundles. That is, there are no *spatial* links that connect both bundles.

5.3.1 Trajectory Separation

Given bundles n and m are considered for merging based on their combined bundle forming a close 3D network as previously discussed, we then proceed to the second step in the *spatial* merging process. In this second step we assess how the Euclidean distances in the image plane between the trajectories in both bundles change over a sequence. The idea is that we merge both bundles if their trajectories maintain roughly constant distances to each other over a sequence. This approach facilitates the merger of trajectory bundles corresponding to non-rigid objects. We have observed from real world sequences that two non-rigid trajectory bundles with similar motion usually maintain roughly constant Euclidean distances in the image plane amongst the trajectories in both bundles.

Consider that trajectory $\mathcal{X}_r$ is one of the T_n trajectories belonging to bundle n, where $r \in \{1 : T_n\}$. Similarly, the mth bundle has T_m trajectories where $\mathcal{X}_s$ being one of these trajectories. We evaluate how the Euclidean distances (spatial separations) between the trajectories $\mathcal{X}_r$ and $\mathcal{X}_s$ change over a sequence in order to derive metrics for determining if bundles n and m should be merged. For each trajectory $\mathcal{X}_r$ in bundle n we evaluate how its spatial separations with respect to all the trajectories in bundle m are changing in time. As an simple example, the *top* row of fig. 5.13 shows the current trajectory $\mathcal{X}_r$ in bundle n that is being evaluated with respect to the four trajectories in bundle m. Here trajectory $\mathcal{X}_r$ and the four trajectories in bundle m are represented as *red* and *purple* curved lines respectively extending across a set of frames (dotted vertical lines) in a sequence. For clarity of later discussions we define two of the trajectories in bundle m as $\mathcal{X}_s$ and $\mathcal{X}_u$, where the spatial locations along these trajectories are indicated with *green* and *blue* dots respectively.

We define $\mathcal{F}_{r,s} = [a_{r,s} : b_{r,s}]$ as the set of frames for which both trajectories $\mathcal{X}_r$ and $\mathcal{X}_s$ exist in time, where $a_{r,s}$ and $b_{r,s}$ are the first and last frames in this range respectively. The frames $a_{r,s}$

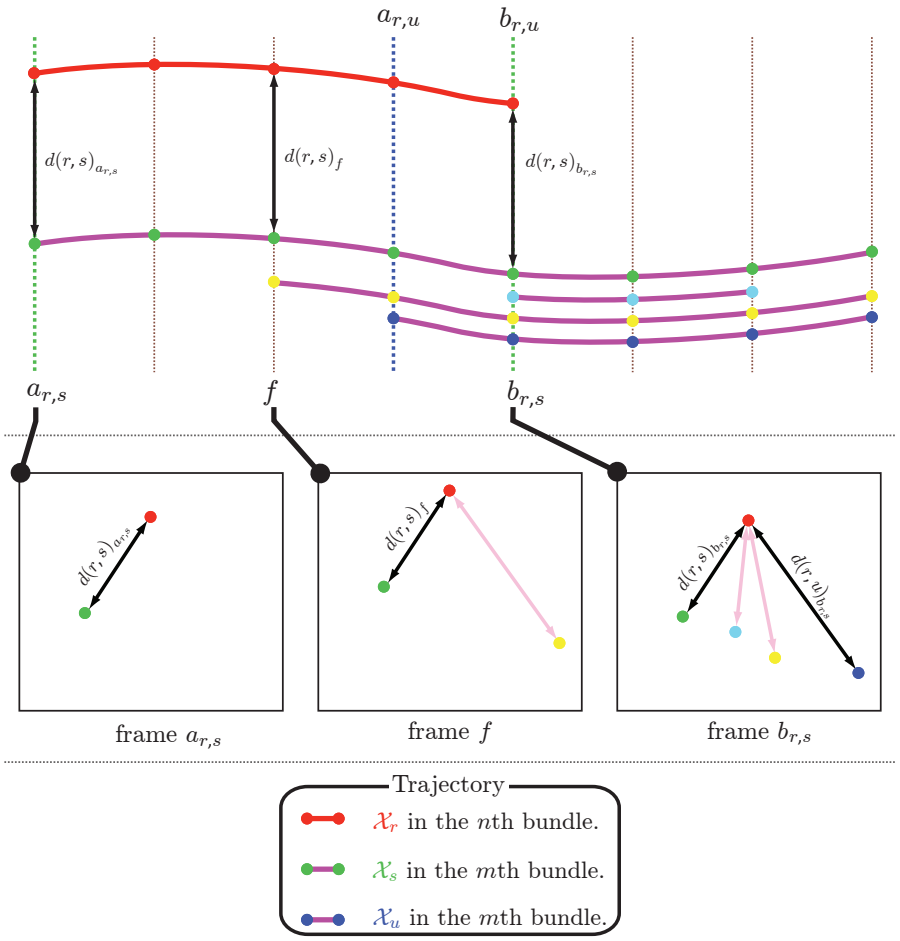

Figure 5.13: *Notations for the second step in the* ***spatial*** *merging process.* ***Top row****: The trajectory $\mathcal{X}_r$ (red line) is a trajectory in bundle n for which the Euclidean distances $d(r,s)_f$ are evaluated with respect to the trajectories in bundle m (purple lines). Both trajectories $\mathcal{X}_s$ and $\mathcal{X}_u$ belong to bundle m, and their spatial locations at each frame are indicated with* ***green*** *and* ***yellow*** *'dots' respectively. The start frame $a_{r,s}$ and end frame $b_{r,s}$ for which trajectories $\mathcal{X}_r$ (red line) and $\mathcal{X}_s$ (green 'dots') overlap temporally are the labelled green dotted vertical lines. Similarly the start $a_{r,u}$ and end $a_{r,u}$ frames are labelled for the overlap of trajectory $\mathcal{X}_r$ and $\mathcal{X}_u$. The Euclidean distances $d(r,s)_f$ between trajectories $\mathcal{X}_r$ and $\mathcal{X}_s$ are indicated with* ***black arrows****.* ***Second row****: The Euclidean distances in the image plane between the points along trajectory $\mathcal{X}_r$ (****red*** *'dots') and the various points along the trajectories in bundle m (green, cyan, yellow and blue 'dots'). These distances are show for frames $a_{r,s}$, f and $b_{r,s}$.*

and $b_{r,s}$ are the labelled *green* dotted vertical lines in fig. 5.13. For a frame $f \in \mathcal{F}_{r,s}$, we define $d(r,s)_f$ given below as the Euclidean distance between trajectories $\mathcal{X}_r$ and $\mathcal{X}_s$ at this frame.

$$d(r,s)_f = \left\| \begin{pmatrix} x_f^r \\ y_f^r \end{pmatrix} - \begin{pmatrix} x_f^s \\ y_f^s \end{pmatrix} \right\| \tag{5.16}$$

We define $\hat{d}(r,s)_f$ below as the absolute change in the Euclidean distances between trajectories $\mathcal{X}_r$ and $\mathcal{X}_s$ at frame f, given the current distance $d(r,s)_f$ and next distance $d(r,s)_{f+1}$.

$$\hat{d}(r,s)_f = \left| d(r,s)_f - d(r,s)_{f+1} \right|, \quad \text{for } f \in \{a_{r,s} : (b_{r,s}-1)\} \tag{5.17}$$

The idea is that if the distances between trajectories $\mathcal{X}_r$ and $\mathcal{X}_s$ stay constant over a sequence, then $\sum_{f=a_{r,s}}^{b_{r,s}-1} \hat{d}(r,s)_f \approx 0$. Hence we derive metrics for merging based on the mean and standard deviation of the changes in the distances $\hat{d}(r,s)_f$ between trajectory $\mathcal{X}_r$ and the trajectories in bundle m. This mean $\bar{d}_r$ along with the standard deviation ς_r for trajectory $\mathcal{X}_r$ in bundle n are given below.

$$\bar{d}_r = \frac{1}{Z} \sum_{o \in \mathcal{O}_r} \sum_{f=a_{r,o}}^{b_{r,o}-1} \hat{d}(r,o)_f \tag{5.18}$$

$$\varsigma_r = \left(\frac{1}{Z} \sum_{o \in \mathcal{O}_r} \sum_{f=a_{r,o}}^{b_{r,o}-1} \left[\hat{d}(r,o)_f \right]^2 \right) - \left[\bar{d}_r \right]^2 \tag{5.19}$$

Where $\mathcal{O}_r$ is the set of trajectories in bundle m that overlap temporally with trajectory $\mathcal{X}_r$ for more than two frames. As an example, all the trajectories in bundle m indicated with *purple* lines in fig. 5.13 belong to the set $\mathcal{O}_r$ except for the trajectory with *cyan* 'dots'. This trajectory does not overlap temporally with trajectory $\mathcal{X}_r$ (*red* line) for more than two frame so no distance changes can be estimated here.

We define trajectory $\mathcal{X}_o$ as a trajectory in this set $\mathcal{O}_r$, i.e. $o \in \mathcal{O}_r$. Also, Z given below is the number of distance changes $\hat{d}(r,o)_f$ estimated for trajectory $\mathcal{X}_r$ with respect to the all the trajectories $\mathcal{X}_o$.

$$Z = \sum_{o \in \mathcal{O}_r} \sum_{f=a_{r,o}}^{b_{r,o}-1} (1) \tag{5.20}$$

5.3.1.1 Spatial Merging Metrics

We merge bundles n and m based on the percentage of the trajectories $\mathcal{X}_r$ in bundle n that have means $\bar{d}_r$ and standard deviations ς_r below respective thresholds. We define $D_{n,m}$ as the percentage of trajectories $\mathcal{X}_r$ with means $\bar{d}_r$ less than a threshold d^τ. Similarly, $R_{n,m}$ is the percentage of trajectories with standard deviations ς_r less than a threshold ς^τ. For all

experiment reported later the thresholds d^{τ} and ς^{τ} are set to $0.5pels$ and $1.0pels^2$ respectively, unless specified otherwise.

Given that both $D_{n,m}$ and $R_{n,m}$ are high, this result indicates that the general consensus of the trajectories is that the Euclidean distances between bundle n and m stay roughly constant over a sequence. We define thresholds for the metrics $D_{n,m}$ and $R_{n,m}$ as D^{τ} and R^{τ} respectively. We make these thresholds D^{τ} and R^{τ} be 80% and 70% respectively for all experiments reported later, unless specified otherwise.

We always estimate the means and standard deviations of the changes in Euclidean distances for the bundle with the most trajectories. Recall that the number of trajectories in bundles n and m are R and S respectively. We define the bundle with the most trajectories as the *reference* bundle. Hence bundle n is the *reference* bundle if $R > S$, otherwise bundle m is the *reference* bundle. Using these means and standard deviations for the trajectories in the *reference* bundle provide more robustness in estimating the merging metrics $D_{n,m}$ and $R_{n,m}$ considering that there might be outlier trajectories in the bundles. Here an outlier trajectory is defined as a trajectory that does not always follow the average motion of the bundle it belongs to. Since the merging metrics are estimated through a trajectory voting process, outliers will have a smaller effect on the these metrics when there are a lot more reliable trajectories in the *reference* bundle.

In the previous discussion we assumed to bundle n had more trajectories than bundle m. However, if bundle m is the *reference* bundle then we estimate the merging metrics $D_{n,m}$ and $R_{n,m}$ using the estimated means $\bar{d}_s$ and standard deviations ς_s for the trajectories $\mathcal{X}_s$ in bundle m. The procedure for estimating $\bar{d}_s$ and ς_s is the same as previously discussed for $\bar{d}_r$ and ς_r, however note that in this case we are estimating the changes in Euclidean distances for the trajectories $\mathcal{X}_s$ in bundle m with respect to the trajectories in bundle n.

5.4 Graphcut Solution

The α-*expansion* Graphcut algorithm [135] is used to solve for the MAP estimate of the trajectory bundle labels $\mathcal{L}_t$. The posterior distribution for the trajectory labels $\mathcal{L}_t$ discussed previously is repeated below.

$$p(\mathcal{L}_t|\mathcal{X}, \mathcal{L}_{\sim t}) \propto p_x(\mathcal{X}_t|\mathcal{L}_t)p_s(\mathcal{L}_t|\mathcal{L}_{\sim t}) \tag{5.21}$$

Details of the adaptation of the Bayesian problem in eq. 5.21 above to this Graphcut solution are exposed in appendix A. There are no major issues to raise as far as the Graphcut techniques are concerned.

5.5 Summary

This chapter discusses the *refinement* stage in our *sparse trajectory segmentation* technique. This stage follows the *initialization* stage discussed in chapter 4. In the *refinement* stage a Bayesian

framework is used to enforce spatial and temporal smoothness on the trajectory bundle obtained from the *initialization* stage.

In the *refinement* stage we utilize an algorithm we define as the *local region constraint* algorithm to ensure that each trajectory bundle represent a single coherent image region. Also in this stage we include a trajectory merging strategy that is effective for merging trajectory bundles corresponding to non-rigid objects.

6

Sparse Trajectory Segmentation Performance Evaluation

In this chapter the performance of the proposed algorithm is compared to previous methods. Recall from the review in chapter 2 that sparse trajectory segmentation algorithms may be categorized based on whether they provide a 2D or 3D segmentation.

A 3D segmentation approach aims to group all the trajectories belonging to a particular object in 3D space. Each group of trajectories provided by a 3D segmentation is defined as a 3D trajectory bundle.

A 2D segmentation approach like our proposed technique, groups trajectories of coherent 2D image motion. To distinguish between the results of a 2D and 3D segmentation we define a group of trajectories labelled as having similar 2D image motion as a 2D trajectory bundle.

The illustration in fig. 6.1 will be used to demonstrate the difference between the segmentation results for both the 2D and 3D approaches. An important distinction between both approaches is the level of segmentation produced. Understanding the nature of both segmentation results is necessary for establishing some grounds for comparing them.

The *top* row of fig. 6.1 shows a truck at two different frames in a motion sequence. Some approximately planar surfaces on this truck are uniquely highlighted for illustration purposes in red, green, cyan, purple, and yellow. Here matching surfaces across both frames are coloured in a similar manner.

A 3D segmentation algorithm is required to identify all the trajectories generated by the truck as belonging to a single object. The spatial locations of these trajectories are shown as

Figure 6.1: ***Top row**: A truck shown at frames 1 (left) and 24 (right) in a motion sequence from the Hopkins dataset [117]. Some roughly planar surfaces on the truck are highlighted in red, green, yellow, purple and cyan. The matching surfaces across both frames are coloured in a similar manner. **Bottom left**: The desired labels for a 3D segmentation algorithm. The spatial locations of the trajectories for the background and truck at the current frame are indicated with **red** and **yellow** 'dots' respectively. **Bottom right**: The trajectory segmentation result produced by our proposed technique. Each trajectory bundle shown in a different colour roughly represents a single surface of the truck.*

yellow 'dots' in the *bottom left* illustration of fig. 6.1. Note here that the 3D segmentation task becomes more challenging when the object of interest is subjected to significant perspective distortions.

The 2D motion in the image plane of each surface of the truck is dependent on the relative perspective of the surface with respect to the camera. The red (side) and cyan (front) surfaces for example will have relatively different 2D image motions due to their orientations and distances from the camera.

A 2D segmentation algorithm groups the trajectories of the truck according to the planar

surface they belong to. The *bottom right* of fig. 6.1 shows the segmentation produced by our proposed technique. Note that each trajectory bundle roughly lies along a single planar surface.

We compared our 2D segmentation technique with five state-of-the-art 3D segmentation approaches using the Hopkins dataset [117]. This dataset is a collection of 155 motion sequences supplied with ground truth information. Our proposed algorithm significantly outperforms these 3D approaches as will be shown later in this chapter.

The only two published 2D trajectory segmentation approaches are by Fradet [46] and Pundlik [95]. These approaches were discussed in chapter 2 (review chapter). Fradet [46] proposed an approach that uses clustering and Affine motion modelling. Pundlik models the motion of the trajectories with Affine transformations and finds the optimal number of trajectory bundles using a region growing strategy.

Fradet [46] has provided a quantitative analysis of his results using the Hopkins dataset. However, Fradet only provides quantitative analysis on 51 sequences of the 155 sequences in the dataset. For these 51 sequences we have lower trajectory misclassification errors. We also visually compared our segmentation with that of Fradet for 4 sequences presented by the author. These sequences do not have any ground truth information for a quantitative analysis.

Pundlik [95] provides no quantitative analysis of his results, so a visual comparison of the respective segmentation results is done for 3 sequences presented by the author.

We now move on to discuss the comparisons in more detail.

6.1 Quantitative Analysis on the Hopkins Dataset

The performances of several 3D segmentation algorithms [59,97,119,132] have been traditionally assessed using the Hopkins dataset [117]. The Hopkins dataset includes 104 indoor checkerboard sequences, 38 outdoor traffic sequences, and 13 articulated/non-rigid sequences. Each sequence has two or three independently moving objects.

There are 155 sequences $\mathcal{S}_h$ in the Hopkins dataset, i.e. $h = 1 : 155$. Each sequence is F_h frames long. A ground truth trajectory set accompanies each sequence. These trajectories are all of length F_h frames, i.e. there are no incomplete/partial trajectories. As an example the *top two* rows of fig. 6.2 show three sequences from the Hopkins dataset. The *top* row shows the starting frames for sample sequences in the checkerboard (left), traffic (middle), and articulated (right) motion sequence categories. The *second* row shows the end frames for these sequences, where $F_h = 24, 22, 60$ from left to right respectively. The ground truth trajectories are superimposed on both the first and end frames.

The task of a 3D segmentation algorithm is to assign a label $\mathcal{L}_t \in \{1 : N_h\}$ to these trajectories, where N_h is the number of independently moving objects N_h in sequence $\mathcal{S}_h$. The number of objects is either 2 or 3 for the Hopkins dataset. For the example sequences in fig. 6.2 the number of objects are $N_h = 3, 2, 2$ from left to right respectively.

The number of 2D trajectory bundles estimated by our technique for sequence $\mathcal{S}_h$ may not

Figure 6.2: *Three sample motion sequences from the Hopkins dataset. A indoor checkerboard (left), outdoor traffic (middle), and articulated/non-rigid (right) sequence are shown.* ***Top two rows****: The first and last frames in these sequences respectively. The ground truth trajectory bundles are coloured red, yellow, and green in the first two rows.* ***Third row****: The 2D segmentations provided by our proposed technique for the example sequences.* ***Bottom row****: The segmentation results provided by one of the 3D segmentation algorithms being evaluated. All misclassified trajectories are outlined with white boxes.*

be equal to the number of objects N_h. As previously mentioned the number of 2D trajectory bundles generated in an object depends the orientation and distance of this object as it moves relative to the camera. The 2D trajectory bundles for the example sequences in fig. 6.2 are shown

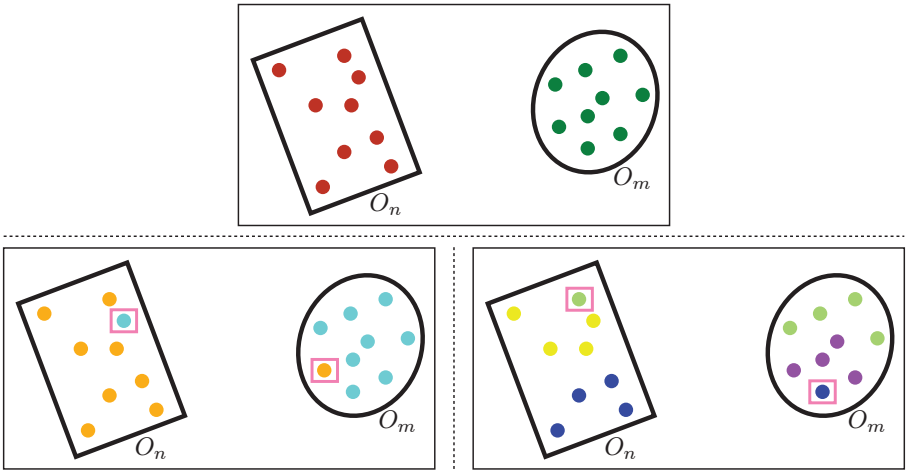

Figure 6.3: *A demonstration of how misclassified trajectories are identified.* ***Top row****: The ground truth segmentation of the trajectories for objects O_n and O_m. The trajectory endpoints are red and green dots for both objects respectively.* ***Bottom left****: An example 3D segmentation of the trajectories for both objects. The orange and cyan bundles represent objects O_n and O_m respectively.* ***Bottom right****: An example 2D segmentation of the object trajectories. The yellow and blue bundles represent object O_n, whilst the purple and light green bundles represent object O_m. Misclassified trajectories are outlined with pink boxes.*

in the *third* row. Here the number of bundles are 4, 17 and 7 for the sequences from left to right respectively, where every bundle has a different colour. Note that unlike 3D segmentation techniques, our 2D segmentation technique does not require a user to specify the number of objects N_h in a sequence.

The number of 3D trajectory bundles is always equal to the number of objects N_h. All 3D segmentation algorithms are told in advance how many objects are in the sequence. This is a major issue with these algorithms which prevent them from being used on an arbitrary sequence where N_h is unknown.

We compare the result of a 3D segmentation approach with our 2D segmentation by considering the trajectory misclassification rate. A trajectory $\mathcal{X}_t$ in bundle n is misclassified if it belongs to an object O_m when the majority of the trajectories in the bundle belong to object O_n. Fig. 6.3 demonstrates how misclassified trajectories are identified.

The *top* row of fig. 6.3 shows the ground truth trajectory labels for two objects O_n and O_m. The trajectory endpoints are red and green dots for both objects respectively. The *bottom left* illustration shows an example 3D segmentation of the trajectories for these objects. Here the two 3D trajectory bundles are orange and cyan. The majority of the trajectories for object O_n are in the orange bundle, so this bundle is deemed to represent object O_n. In a similar manner object O_m is represented by the cyan bundle. The misclassified trajectories are outlined with

pink boxes. These trajectories belong to an object that is not represented by the bundle it belongs to.

The *bottom right* illustration shows an example 2D segmentation of the trajectories for the objects O_n and O_m. The number of bundles here is not equal to the number of objects. However, each bundle represents the object for which it has the most trajectories. For example, The blue and yellow bundles both represent object O_n, whilst the purple and light green bundles represent object O_m. The misclassified trajectories here are also outlined with pink boxes.

The percentage misclassification rate M_h for sequence S_h is now defined as follow,

$$M_h = \frac{\#\text{misclassified trajectories}}{T_h} \tag{6.1}$$

Where T_h is the number of trajectories in sequence S_h.

The *third* row of fig. 6.2 shows the 2D segmentation of our technique for the example sequences in the figure. The misclassified trajectories are outlined with white boxes. The percentage misclassification rate for these sequences are $M_h = 0.4\%, 0.0\%, 6.1\%$ from left to right. The *fourth* row shows the results provided by one of the 3D segmentation algorithms. Again misclassified trajectories are outlined with white boxes. The percentage misclassification rate for the example sequences are $M_h = 0.6\%, 0.0\%, 6.1\%$ from left to right respectively.

This section has illustrated a few results and how they are measured. We now go on to present a complete analysis comparing with five 3D segmentation techniques and the technique of Fradet [46].

6.1.1 Misclassification Results

We have compared the proposed algorithm with the five 3D segmentation approaches. These approaches are titled Multistage Learning (MSL) [59],Generalized Principal Component Analysis (GPCA) [118–121], Random Sample Consensus(RANSAC) [42, 115], Local Subspace Affinity (LSA) [132] and Agglomerative Lossy Compression (ALC) [97, 133].

In general most 3D segmentation algorithms do not perform well when the number of moving object N_h in a sequence is more than two. Therefore the misclassification rates are reported for sequences with two and three moving objects separately. Table 6.1 shows the number of sequences (N_h), the mean number of trajectories ($\bar{T}_h$) and frames ($\bar{F}_h$) for the sequences with two or three moving objects. There are 120 and 35 sequences with two and three moving objects respectively.

Recall that the sequences in the Hopkins dataset are categorized into checkerboard, traffic, and articulated/non-rigid sequences. Each category has different motion sequence characteristics. The objects in the checkerboard sequences are physically controlled to have either rotational, translational or stationary motion. However for the traffic and articulated sequences the motions of the objects are not controlled.

Since the three categories in the Hopkins dataset all have different motion characteristics, the misclassification rates are reported for each category. For our technique the number of

	$N_h = 2$			$N_h = 3$		
	#Seq.	$\bar{T}_h$	$\bar{F}_h$	#Seq.	$\bar{T}_h$	$\bar{F}_h$
Checkerboard	78	291	28	26	437	28
Traffic	31	241	30	7	332	31
Articulated	11	155	40	2	122	31
All	120	266	30	35	398	29

Table 6.1: *The number of sequences (#Seq.), the mean number of trajectories ($\bar{T}_h$) and frames $\bar{F}_h$ for the three categories in the Hopkins dataset. These statistics are shown for sequences with two ($N_h = 2$) and three ($N_h = 3$) moving objects.*

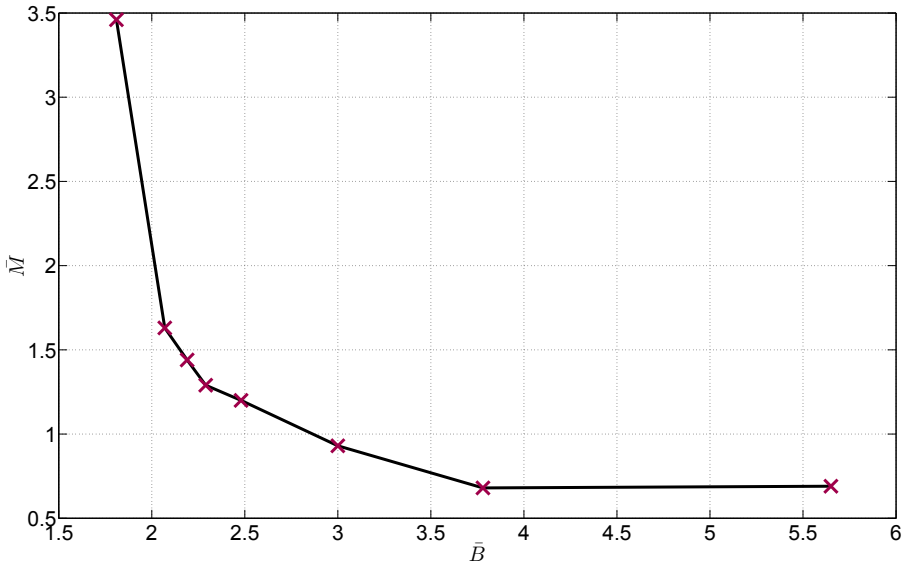

Figure 6.4: *The mean bundle ratios $\bar{B}$ versus the mean misclassification rates $\bar{M}$ for the 7 experiments outlined in table 6.2.*

bundles b_h produced per moving object for sequence $\mathcal{S}_h$ influences the misclassification rate M_h. Consider that B_h is the ratio of the number of bundles b_h to the number of moving objects N_h defined as follows.

$$B_h = \frac{b_h}{N_h} \tag{6.2}$$

It is reasonable to expect that increasing the ratio B_h (more bundles per object) will cause the misclassification rate M_h to decrease. Therefore we conducted 8 experiments on all 155 sequences where the parameters in the *refinement* stage of our technique were varied to control the ratio B_h of bundles per moving object.

Table 6.2 shows the parameters used for the 8 experiments along with the corresponding

	Initialization		*Refinement*			
	Affine Mer.	*MS Band.*	*Separation Mer.*			
Experiment	$s_{n,m}$	k_1, k_2, k_3, k_4	d^τ, D^τ	ς^τ, R^τ	$\bar{B}$	$\bar{M}$
0	0.001	0.5,0.5,0.5,0.5	0,∞	0,∞	5.65	0.69
1	**0.001**	**0.5,0.5,0.5,0.5**	**0.5,0.80**	**1.0,0.70**	**3.78**	**0.68**
2	0.001	0.5,0.5,0.5,0.5	1.0,0.70	2.0,0.50	3.00	0.93
3	0.001	0.5,0.5,0.5,0.5	1.0,0.60	2.0,0.40	2.48	1.20
4	0.001	0.5,0.5,0.5,0.5	1.0,0.60	3.0,0.40	2.29	1.29
5	0.001	0.5,0.5,0.5,0.5	1.0,0.50	3.0,0.30	2.19	1.44
6	0.001	0.5,0.5,0.5,0.5	1.0,0.40	3.0,0.20	2.07	1.63
7	0.001	0.5,0.5,0.5,0.5	1.2,0.35	5.0,0.15	1.81	3.46

Table 6.2: *The parameters for 8 experiments conducted for the proposed algorithm on the 155 sequences in the Hopkins dataset. The parameters for the Affine merging $s_{n,m}$ (Affine Mer.) and the Mean shift clustering bandwidth k_1, k_2, k_3, k_4 (MS Band.) in the initialization stage were kept constant over all experiments. The trajectory separation merging parameters d^τ, D^τ, ς^τ and R^τ (Separation Mer.) in the refinement stage were varied so the bundle ratio B_h (6^{th} column)decreases monotonically from experiment 1-7. In experiment 0 the trajectory bundles are not merged in the* ***refinement*** *stage. The mean misclassification rate $\bar{M}$ for each experiment is shown in the end column. The results* ***Experiment*** *1* are used for comparison with other segmentation methods. The row for this experiment is in bold text.

mean bundle ratios $\bar{B}$ and mean misclassification rates $\bar{M}$ over all 155 sequences. Note that experiment 0 acts as a control where no merging of the trajectory bundles is allowed in the *refinement* stage. The parameters for the Affine merging $s_{n,m}$ (Affine Mer.) and the Mean shift clustering bandwidth k_1, k_2, k_3, k_4 (MS Band.) in the *initialization* stage were kept constant over all experiments. However the trajectory separation merging parameters d^τ, D^τ, ς^τ and R^τ (Separation Mer.) in the *refinement* stage were varied to control the bundle ratio B_h.

For experiments 1-7 the trajectory separation merging thresholds d^τ, D^τ, ς^τ and R^τ are relaxed monotonically. That is, *experiment* 1 has the 'tightest' thresholds (0.5,0.8,1.0,0.70) to deter the merging of the trajectory bundles, so this experiment has the highest bundle ratio $\bar{B} = 3.78$. The 7^{th} experiment however has the 'loosest' thresholds (1.2,0.35,5.0,0.15) to encourage more merging of the trajectory bundles. Hence this experiment has the lowest bundle ratio $\bar{B} = 1.81$.

The mean bundle ratios $\bar{B}$ versus the mean misclassification rates $\bar{M}$ for the 8 experiments are plotted in fig.6.4. As expected the mean misclassification rate $\bar{M}$ decreases as the mean number of bundles per moving object ($\bar{B}$) increases. The plot in fig. 6.4 suggests that we can not expect much more improvement in the mean misclassification rate $\bar{M}$ by increasing the

Checkerboard	Proposed	MSL	GPCA	RANSAC	LSA	ALC
Average	**0.56%**	2.13%	5.80%	6.05%	2.14%	2.24%
Median	0.00%	0.00%	1.03%	1.84%	0.27%	0.00%
Traffic	Proposed	MSL	GPCA	RANSAC	LSA	ALC
Average	**0.05%**	0.74%	8.01%	8.15%	9.51%	1.04%
Median	0.00%	0.00%	0.80%	0.52%	5.66%	0.00%
Articulated	Proposed	MSL	GPCA	RANSAC	LSA	ALC
Average	4.10%	**1.90%**	9.78%	10.17%	8.77%	6.69%
Median	0.00%	0.00%	5.56%	6.35%	5.33%	0.00%
All	Proposed	MSL	GPCA	RANSAC	LSA	ALC
Average	**0.75%**	1.75%	6.74%	6.97%	4.65%	2.34%
Median	0.00%	0.00%	1.32%	1.84%	0.89%	0.00%

Table 6.3: *Misclassification rates for sequences with two object motions* $N_h = 2$.

bundle ratio $\bar{B}$ beyond 3.78. Therefore the results of *experiment* 1 are used later in this chapter for the comparison with the other segmentation approaches. This experiment has the lowest mean misclassification rate $\bar{M} = 0.68$, and is highlighted in bold text in table 6.2.

The next section reports the misclassification rates for the sequences with two moving objects.

6.1.2 Misclassification for Sequences with Two Moving Objects

For sequences with two moving objects ($N_h = 2$), our technique has the overall lowest mean percentage misclassification rate of 0.75% as presented in the *bottom* row (All) of table 6.3. The MSL algorithm provides the next best mean misclassification rate of 1.75%.

In table 6.3 the best misclassification average rates for the checkerboard, traffic, and articulated sequence categories are in bold text. The proposed algorithm has the lowest misclassification rates for the checkerboard (0.56%) and traffic (0.05%) sequences. However the MSL algorithm provides a lower rate for the articulated sequences. There are 11 articulated sequences with two moving objects. Our technique has a misclassification rate of 28.57% for one of these sequences called *two cranes* (discussed later). If this sequence is excluded when calculating the mean misclassification rate, the proposed and MSL algorithms would have mean rates of 1.66% and 2.09% respectively. Hence we would have a lower mean rate.

We can obtain a mean misclassification rate of zero for the *two cranes* sequence if a smaller Mean shift clustering bandwidth is used during the *initialization* stage. The *top* row of fig. 6.5 shows the first (left) and last (last) frames for the *two cranes* sequence. The ground truth trajectory bundles for both objects (cranes) are each shown as red and yellow dots.

Both cranes have trajectories that are approximately stationary in motion. Also the motion

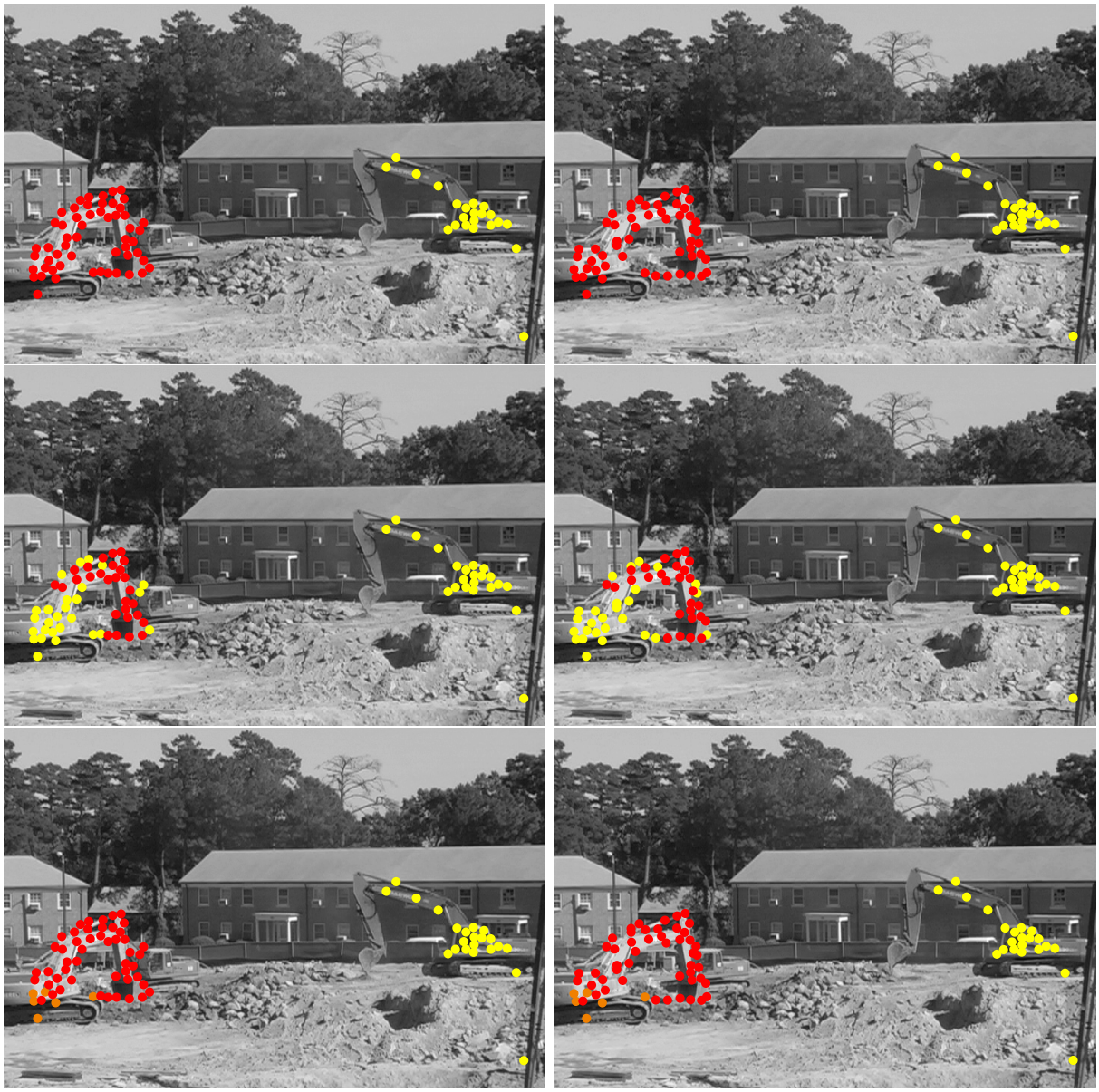

Figure 6.5: *The first (left) and last (right) frames for the* ***two cranes*** *sequence in the Hopkins dataset.* ***Top row****: The ground truth trajectories for both cranes are shown as red and yellow dots.* ***Middle row****: The segmentation achieved by the proposed algorithm using the general MS clustering bandwidth parameters specified for all the sequences in the experiment.* ***Bottom row****: The final segmentation achieved using half of the general MS clustering bandwidth in the initialization stage. Here the misclassification rate is zero, as the red, orange, and yellow bundles all represent a single crane.*

differences between both cranes are very subtle due to them being far away from the camera. When using the general clustering parameters stipulated for all the sequences, our technique

forms a bundle with some of the stationary trajectories corresponding to both cranes. The *middle* row of fig. 6.5 shows the segmentation when using these general MS clustering parameters for *experiment* 1. Here the yellow bundle contains roughly stationary trajectories corresponding to both cranes. The misclassification rate for this segmentation is 28.57%.

Our technique can differentiate between the subtle motions of the cranes by using a smaller MS clustering bandwidth during the *initialization* stage. By using half of the general clustering bandwidth ($(k_1, k_2, k_3, k_4) = (0.25, 0.25, 0.25, 0.25)$) in *initialization* a misclassification rate of zero is achieved. The bottom row of fig. 6.5 shows the segmentation result of our technique when using this smaller MS clustering bandwidth. The 3 bundles coloured red, orange and yellow all represent a single crane.

The graph at the *top* of fig. 6.6 shows a histogram of the percentage misclassification rates M_h for all 120 sequences with two moving objects. The *y-axis* shows the percentage of the 120 sequences for which a particular range of misclassification rates occur. For example, our technique (proposed) has a misclassification rate M_h between 0 and 5 for 95% (114 sequences) for the 120 sequences. In this range 95 of the 114 sequences have misclassification rates of exactly zero.

The GPCA, RANSAC, LSA, and ALC segmentation approaches unlike the proposed and MSL algorithms have significant misclassification rates in the histogram bins centered on 5%-45%.

6.1.3 Misclassification for Sequences with Three Moving Objects

For the 35 sequences with three moving objects ($N_h = 3$), our technique has the overall lowest mean percentage misclassification rate of 0.43% as presented in the *bottom* row (All) of table 6.4. The MSL algorithm provides the next best mean misclassification rate of 6.35%.

In table 6.4 the best misclassification average rates for the checkerboard, traffic, and articulated sequence categories are in bold text. Our technique has the lowest misclassification rates for the all three categories; checkerboard (0.56%),traffic (0.05%) and articulated (2.66%).

The 3D segmentation approaches have much higher mean misclassification rates for the sequences with three moving objects compared to the sequences with two moving objects. These approaches generally try to fit motion models to trajectory points in some derived feature space. For sequences with just two independently moving objects, the trajectory points usually reside in two distinct non-overlapping clusters/bundles in the feature space.

However, increasing the number of moving objects leads to the trajectory feature space becoming congested. This congestion causes the trajectory bundles for the objects to overlap in the feature space, which makes it difficult to find meaningful segmentation boundaries between the bundles. Hence the quality of 3D segmentation approaches deteriorate as the number of moving object N_h is increased.

Recall that the *initialization* stage of our technique uses clustering in an orthogonal feature

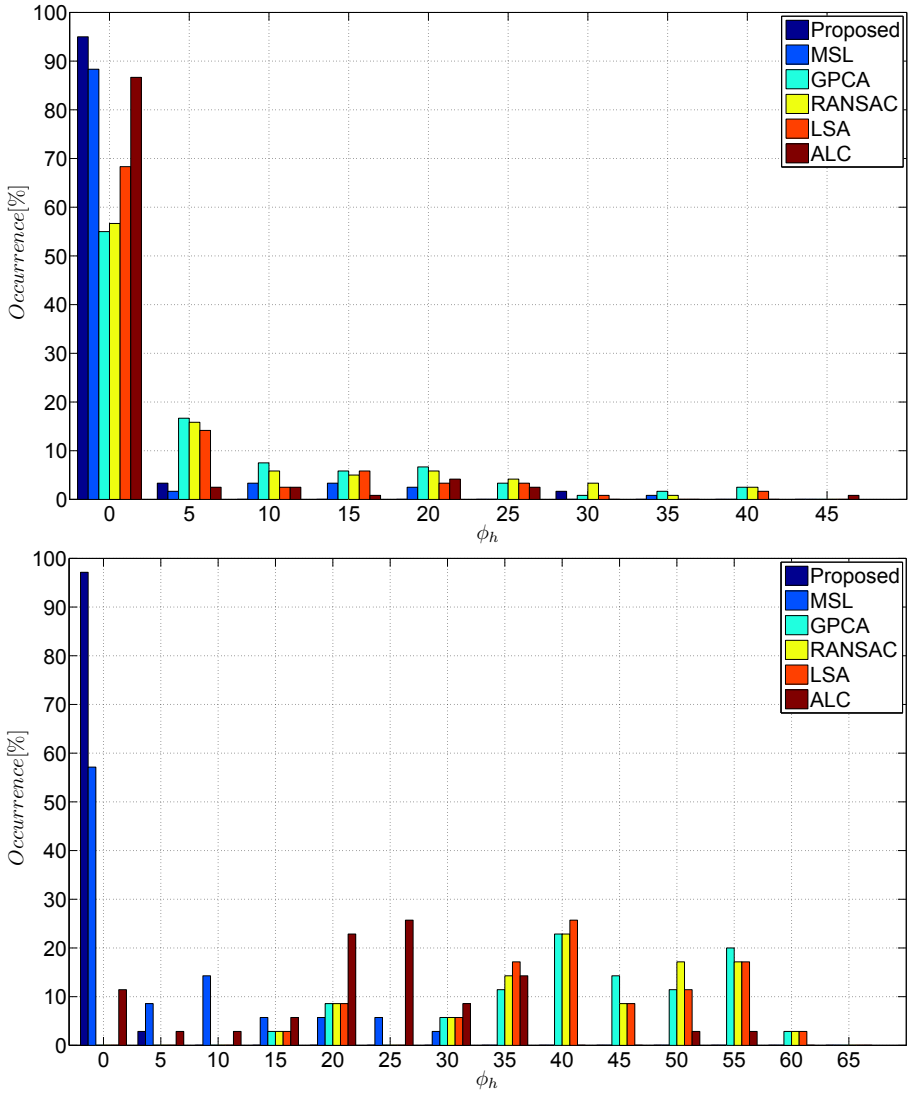

Figure 6.6: *Misclassification rates for the proposed, MSL, GPCA, RANSAC, LSA and ALC segmentation approaches.* ***Top row****: The histogram of the misclassification rates M_h for the sequences in the Hopkins dataset with two moving objects.* ***Bottom row****: The histogram of the misclassification rates M_h for the sequences in the Hopkins dataset with three moving objects.*

space to find an initial set of trajectory bundles. Here the idea is similar to the 3D segmentation approaches where we seek to identify trajectories of a particular object in a derived feature space. We however have the advantage of using spatial information later on in the *refinement* stage. The *refinement* stage uses 2D motion and spatial information to introduce important constraints that significantly improves the final segmentation.

The graph at the *bottom* of fig. 6.6 shows a histogram of the percentage misclassification rates M_h for all 35 sequences with three moving objects. Our algorithm has a misclassification rate

Checkerboard	Proposed	MSL	GPCA	RANSAC	LSA	ALC
Average	**0.30%**	6.95%	44.89%	44.50%	43.21%	27.64%
Median	0.00%	1.48%	44.30%	43.87%	42.27%	25.58%
Traffic	Proposed	MSL	GPCA	RANSAC	LSA	ALC
Average	**0.26%**	3.17%	29.28%	29.28%	29.28%	10.18%
Median	0.00%	0.34%	21.48%	21.48%	21.48%	6.36%
Articulated	Proposed	MSL	GPCA	RANSAC	LSA	ALC
Average	**2.66%**	9.71%	47.41%	45.74%	47.08%	12.04%
Median	2.66%	9.71%	47.41%	45.74%	47.08%	12.04%
All	Proposed	MSL	GPCA	RANSAC	LSA	ALC
Average	**0.43%**	6.35%	41.91%	41.52%	40.64%	23.26%
Median	0.00%	1.33%	42.48%	42.26%	39.56%	23.40%

Table 6.4: *Misclassification rates for sequences with three object motions* $N_h = 3$.

M_h between 0 and 5% for 97.1% (34 sequences) for the 35 sequences. In this range 20 of the 34 sequences have misclassification rates of exactly zero. All the 3D segmentation technique with the exception of MSL, have poor misclassification rates for these sequences with three motions. We have more than twice the amount of misclassification rates between 0 and 5% compared to the MSL technique. Hence we have a better overall mean rate of 0.43% compared to 6.35% for MSL.

6.1.4 The Highest Misclassification Rate

The highest misclassification rate produced by our technique was 29.7% for a checkerboard sequence called *2RT3RCR_g23*. The first and last frames for this sequence are shown in the *top* row of fig. 6.7. The two moving objects in this sequence are a checkered beam and the collective background objects moving according to the camera motion. The spatial locations of the trajectory for the checkered beam and background are shown as *green* and red dots respectively (top row of fig. 6.7).

For the *2RT3RCR_g23* sequence the checkered beam moves very slowly relative to the motion of camera. Hence the 2D motion of the trajectories for the beam are close to that of the background trajectories. The resulting segmentation for this sequence using the parameters for *experiment* 1 mentioned previously (table 6.2) is shown in the *third* row of fig. 6.7. Here the misclassification rate for this segmentation is 29.7%.

Since the 2D motion of the beam and background are quite close, the initial (MS clustering) segmentation of our technique created trajectory bundles that contained trajectories corresponding to both the beam and the background. The largest of these bundles is shown on the right

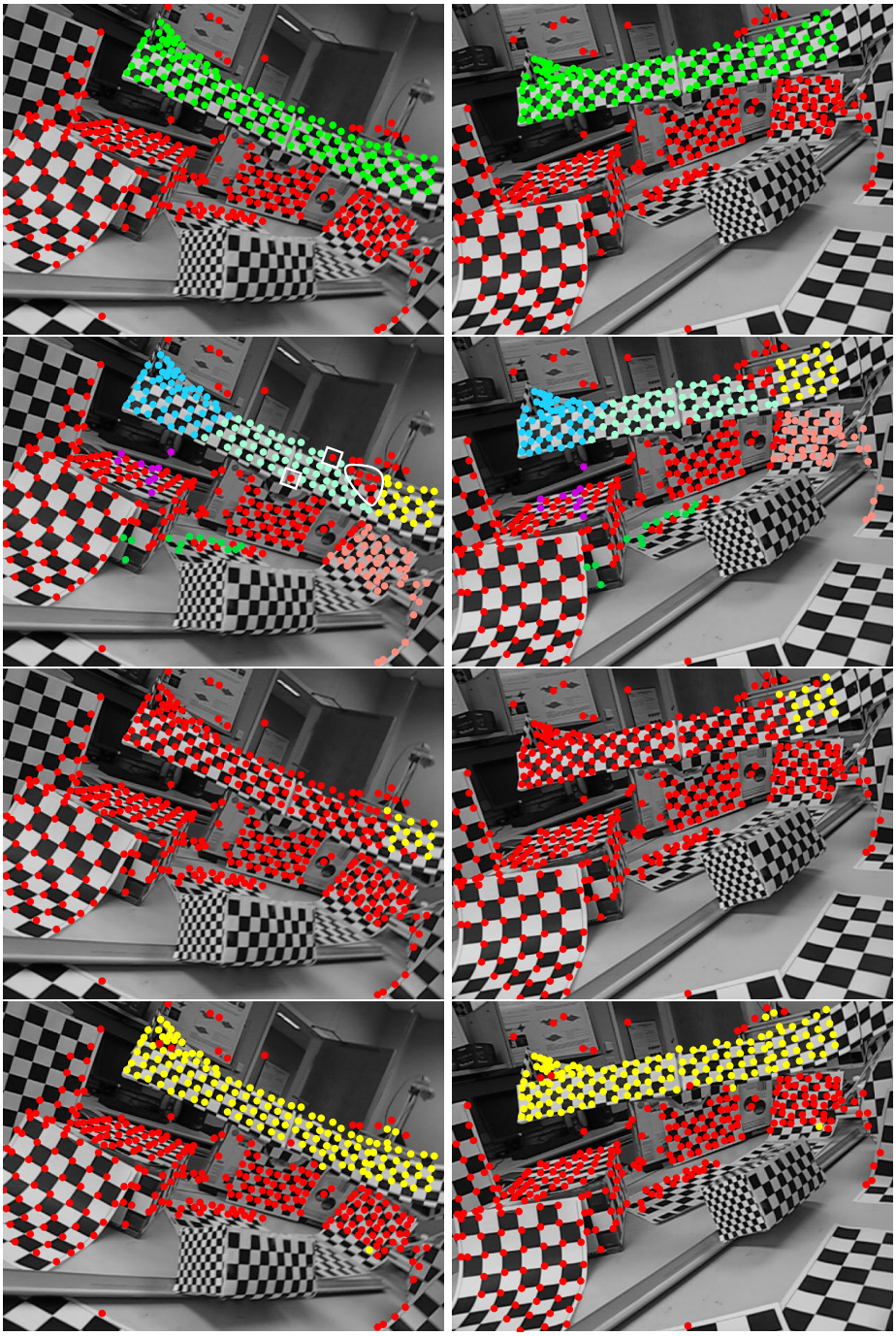

Figure 6.7: *The first (left) and last(right) frames for the* ***2RT3RCR_g23*** *sequence containing two moving objects; a checkered beam and the collective background objects moving according to the motion of the camera.* ***Top row****: The ground truth trajectory endpoints for the checkered beam and background shown as green and red dots respectively.* ***Second row****: The final segmentation of the proposed algorithm using a small Mean shift clustering bandwidth in the* ***initialization*** *stage. Each trajectory bundle is shown with a different colour. The misclassified trajectories are outlined with white closed contours.* ***Third row****: The segmentation of the proposed algorithm using the parameters in* ***experiment*** *1.* The two bundles here are in red and yellow. **Bottom row**: The segmentation result of the LSA algorithm for this sequence.

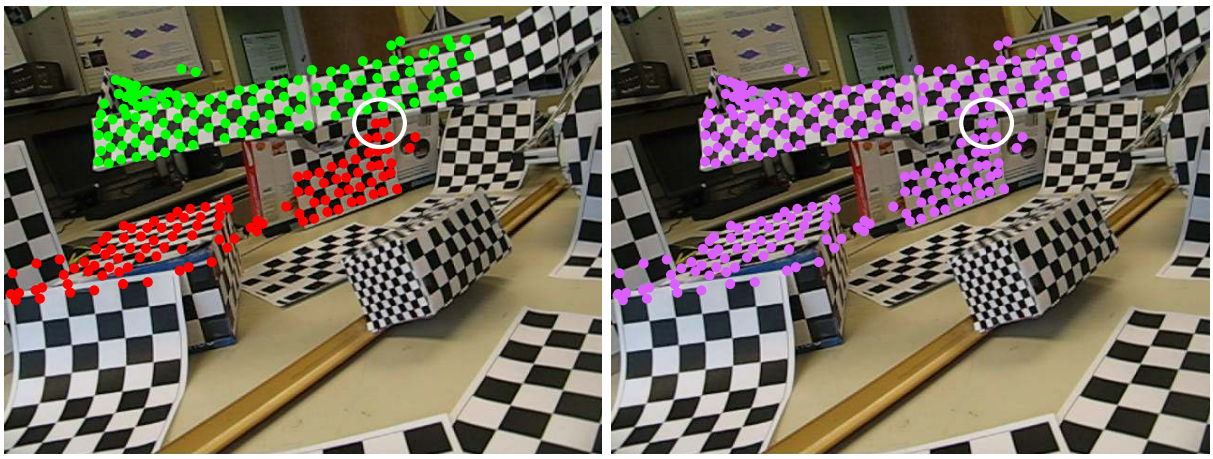

Figure 6.8: ***Right**: The trajectory bundle containing the majority of the trajectories for the checkered beam in sequence **2RT3RCR_g23**. This bundle was produced from Mean shift clustering using the parameters of **experiment** 1. The bundle also contains trajectories corresponding to the background. **Left**: The ground truth trajectory endpoint labels for the purple bundle on the right. Here the trajectory endpoints for the beam and background are shown as green and red dots respectively.*

of fig. 6.8, and the ground truth labels for the trajectories in this bundle are shown on the left. The beam and background trajectory points are shown as *green* and *red* 'dots' respectively.

The majority of the trajectories for the beam resides in the purple bundle in fig. 6.8. Since this bundle contains a significant amount of background trajectories, the average 2D motion of this bundle is highly influenced by these background trajectories. Hence there are no 2D motion models that best describe the motion of the beam exclusively, which results in the poor segmentation obtained.

6.1.5 Improving the Misclassification Rate

We will discuss now how the segmentation for the *2RT3RCR_g23* sequence may be improved by creating better initial trajectory bundles in the Mean shift clustering step. The *second* row of fig. 6.7 shows the improved segmentation for this sequence, where the misclassification rate is now 2.9% (old rate was 29.7%). The misclassified trajectories are outlined with white closed contours. Also show in the *bottom* row of the figure is the best result obtained from the 3D segmentation approaches. This result was provided by the LSA algoritm and the misclassification rate here is 1.7%.

Recall that every trajectory $\mathcal{X}_t$ is given a feature vector $\mathbf{p_t}$ in the *initialization* stage. The feature vectors are clustered using Mean shift (MS) to obtain the initial set of trajectory bundles. The MS clustering bandwidth here influences how much the 2D motion of the trajectories in a bundle can vary with respect to each other. Hence a 'small' bandwidth causes the resulting bun-

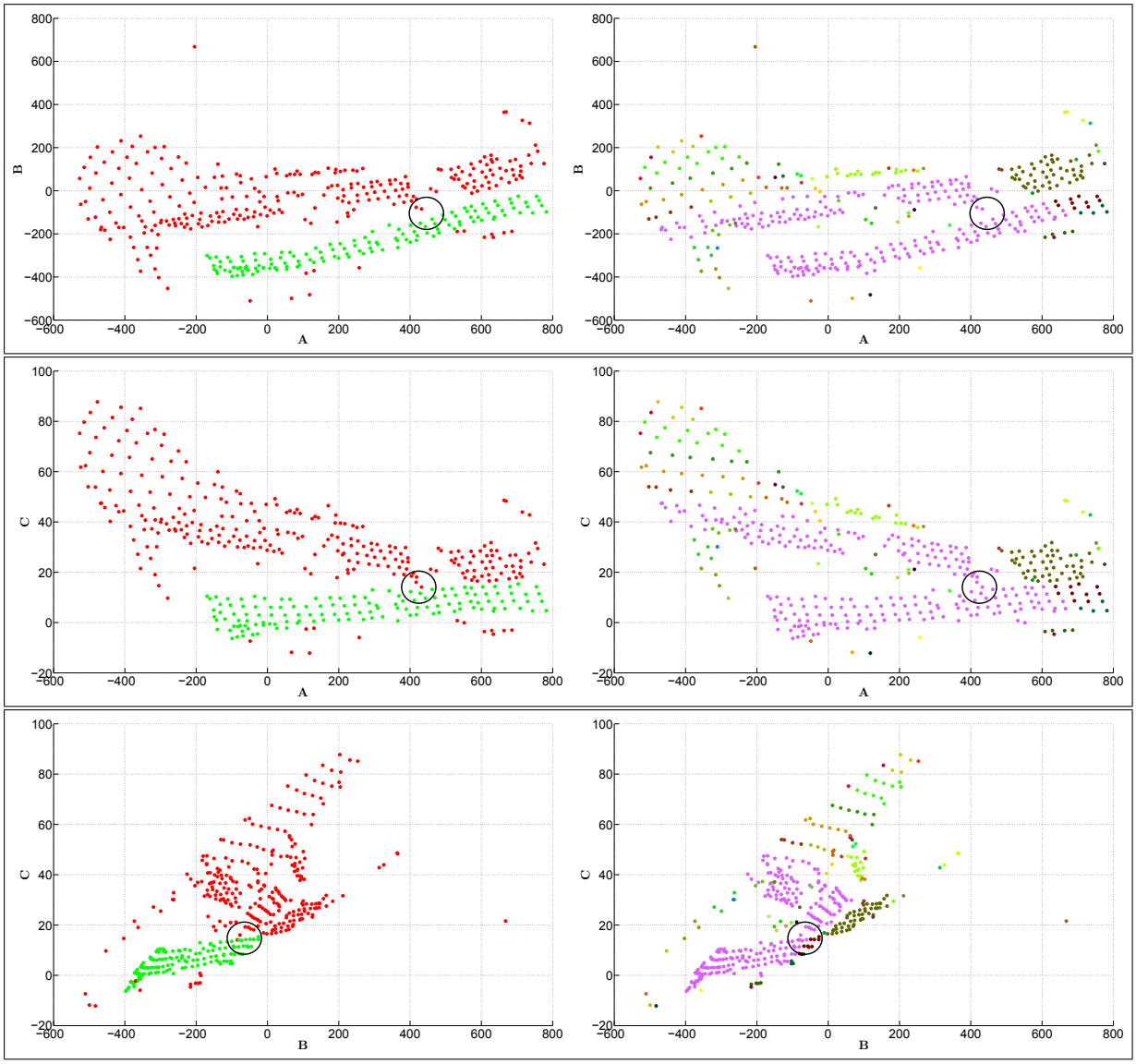

Figure 6.9: *Considering the first three elements of the feature vector* $\mathbf{p_t}$ *for trajectory* $\mathcal{X}_t$ *are A, B, and C. The slices of the feature vector space A versus B (**top row**), A versus C (**middle row**), and B versus C (**bottom row**) for the trajectories in the 2RT3RCR_g23 sequence. **Left column**: The ground truth segmentation of the trajectory feature points, where the beam and background points are shown as green and red dots respectively. **Right column**: The segmentation of the Mean shift clustering step in the proposed algorithm. Here each trajectory bundle is shown in a different colour. The closest feature points for the beam and background are highlighted with black circles.*

dles to have low variations in the 2D motion of the trajectories in it, whilst a 'large' bandwidth caters for more variations in motion.

Consider that the first three elements of the feature vector $\mathbf{p_t}$ for trajectory $\mathcal{X}_t$ are A, B, and C. Fig. 6.9 shows the slices of the feature vector space A versus B (top row), A versus C (middle row), and B versus C (bottom row) for the trajectories in the *2RT3RCR_g23* sequence. The ground truth segmentation of the feature point are shown in the plots on the left, where the beam and background points are in green and red respectively. The segmentation provided by the Mean shift clustering process is shown in the right column. Here the feature points for each trajectory bundle is shown in a different colour.

The purple feature points in fig. 6.9 correspond to the trajectory bundle in fig. 6.8. This purple bundle contained the majority of the trajectories for the beam. It can be observed in fig. 6.9 that this bundle contains feature points for the beam and background that are very close the each other in the feature space. These feature points are highlighted with the black circles in each plot. Now because of the MS clustering bandwidth used the inclusion of these beam and background feature points into the same bundle is unavoidable.

Hence, if a smaller MS clustering bandwidth is used we can over segment the feature points providing more bundles that represent the beam. These over segmented bundles can later be merged in the *refinement* stage. Here the over segmentation guarantees that each bundle exclusively represents either the beam or the background.

The improved segmentation of the proposed algorithm shown in the *second* row of fig. 6.7 was obtained by over segmenting the feature points using half of the bandwidth stipulated for *experiment* 1. That is, $(k_1, k_2, k_3, k_4) = (0.25, 0.25, 0.25, 0.25)$, where these parameters are listed in table 6.2.

6.2 Comparison with Other 2D Algorithms: J Linkage

As discussed previously in 2009 Fradet [46] had presented a trajectory segmentation process that exploits a robust clustering algorithm called 'J Linkage'.

Fradet reports the performance of his technique on the *traffic* and *articulated* sequences in the Hopkins dataset. Table 6.5 shows the misclassification rates our algorithm, along with the MSL, GPCA, RANSAC, LSA, ALC, and Fradet's algorithms. The misclassification rates for the approach of Fradet shown in the last column of the table were taken directly from his paper [46]. Note here that the misclassification rates for the MSL, GPCA, RANSAC, LSA and ALC approaches were generated by using the implementation provided by the respective authors. No implementation of the approach proposed by Fradet was available.

Fradet did not define the misclassification rate metric he used to report his result. Therefore we assume Fradet uses the same misclassification rate M_h metric as used by many previous authors. This misclassification rate M_h metric is repeated below.

$$M_h = \frac{\#\text{misclassified trajectories}}{T_h} \tag{6.3}$$

Where T_h is the number of trajectories in sequence S_h.

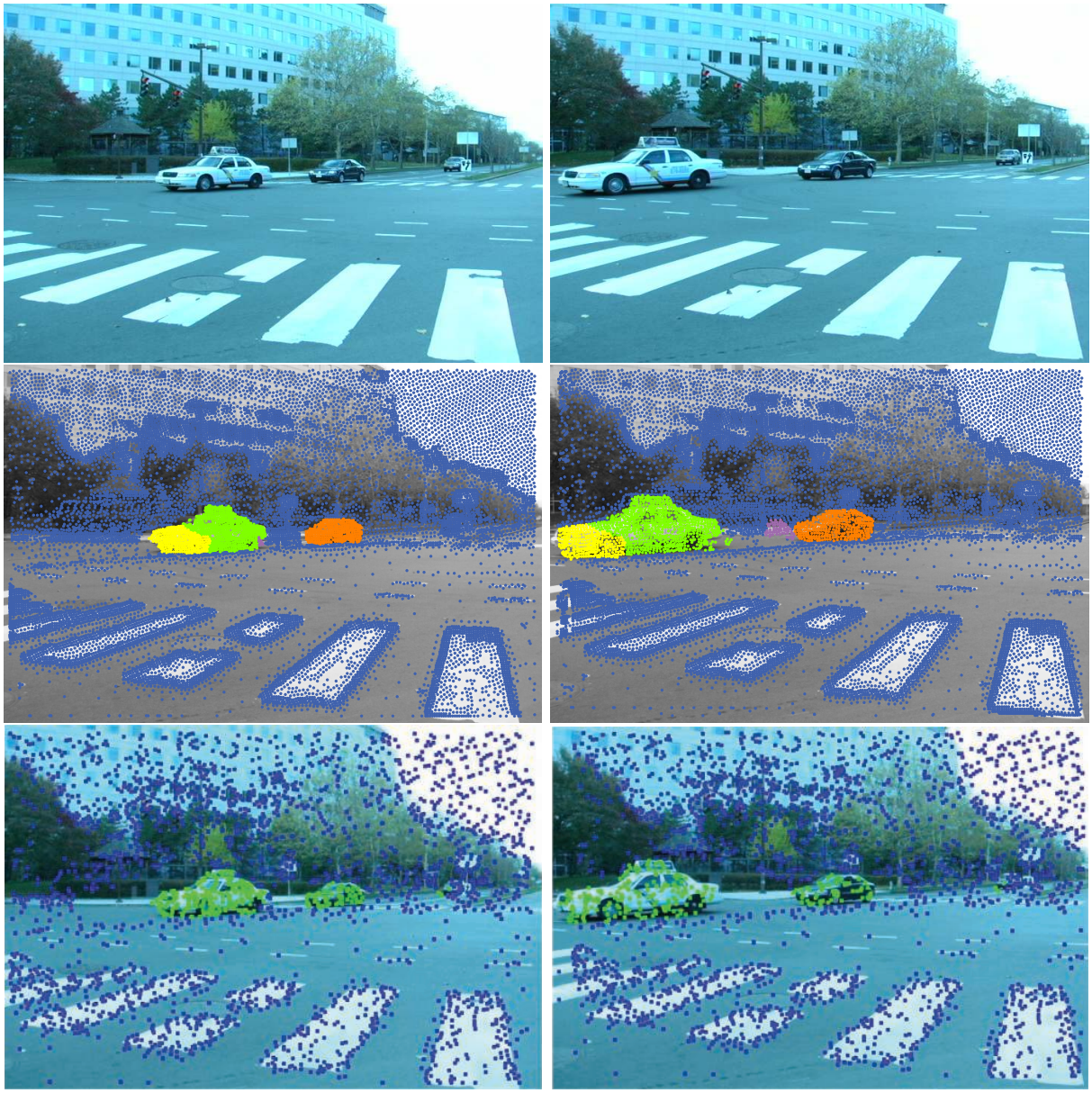

Figure 6.10: *The* ***Cars*** *sequence of 25 frames taken from the dataset provided by Sand [104].* ***Top row****: Frames 1 (left) and 25 (right) for this sequence, where the two cars of interest are moving from right to left in the image plane.* ***Middle row****: Our segmentation of the 'particle' trajectories accompanying the sequence in the dataset. There are four trajectory bundles produced in this segmentation. The blue and purple bundles correspond to the background trajectories exclusively. The yellow and green bundles contain trajectories for car #1, while the orange bundle contains trajectories for car #2. Note here that all the trajectories in the dataset provided for this sequence are used in this segmentation.* ***Bottom row****: The segmentation of Fradet [46] for this sequence, where two bundles are produced. The blue and green bundles correspond to the trajectories of the background and cars respectively. Not all the trajectories in the dataset are used for this segmentation.*

Traffic $[N_h = 2]$	Proposed	MSL	GPCA	RANSAC	LSA	ALC	Fradet [46]
Average	**0.05%**	0.74%	8.01%	8.15%	9.51%	1.04%	1.92%
Median	0.00%	0.00%	0.80%	0.52%	5.66%	0.00%	0.01%
Traffic $[N_h = 3]$	Proposed	MSL	GPCA	RANSAC	LSA	ALC	Fradet [46]
Average	**0.26%**	3.17%	29.28%	29.28%	29.28%	10.18%	4.89%
Median	0.00%	0.34%	21.48%	21.48%	21.48%	6.36%	0.19%
Articulated $[N_h = 2]$	Proposed	MSL	GPCA	RANSAC	LSA	ALC	Fradet [46]
Average	4.10%	**1.90%**	9.78%	10.17%	8.77%	6.69%	5.38%
Median	0.00%	0.00%	5.56%	6.35%	5.33%	0.00%	0.02%
Articulated $[N_h = 3]$	Proposed	MSL	GPCA	RANSAC	LSA	ALC	Fradet [46]
Average	**2.66%**	9.71%	47.41%	45.74%	47.08%	12.04%	20.41%
Median	2.66%	9.71%	47.41%	45.74%	47.08%	12.04%	20.41%
All	Proposed	MSL	GPCA	RANSAC	LSA	ALC	Fradet [46]
Average	**1.06%**	1.68%	12.86%	12.96%	13.53%	3.93%	3.80%

Table 6.5: *Comparison with the work of Fradet [46] using the* ***traffic*** *and* ***articulated*** *sequences in the Hopkins [117] dataset. The average and median misclassification rates are reported in the following categories:* ***Traffic*** *with 2 ($N_h = 2$) and 3 ($N_h = 3$) moving objects, and* ***Articulated*** *with 2 and 3 moving objects. The* ***All*** *category shows the average misclassification rates for all the traffic and articulated sequences.*

Tron [117] in his review of 3D segmentation approaches uses this misclassification rate metric M_h to report his results. Now Fradet [46] reports the same misclassification rates as Tron [117] for the MSL, LSA, and ALC approaches. Hence our assumption that Fradet is using the same misclassification rate metric for comparing his results is justified.

The *bottom* row of table 6.5 shows that our algorithm has an overall misclassification rate of 1.06% while Fradet [46] has an overall rate of 3.80%. Also in each category *traffic* $[N_h = 2]$ (0.05%), *traffic* $[N_h = 3]$ (0.26%),*articulated* $[N_h = 2]$ (4.10%), *articulated* $[N_h = 3]$ (2.66%) we have the lowest misclassification rates compared to Fradet [46].

6.2.1 Spatial Integrity

By visually observing the segmentation results of Fradet [46] it can be concluded that his approach suffers from a lack of spatial constraints on the trajectories in each trajectory bundle. For example this lack of spatial constraints on the trajectory bundles may be observed from the segmentation by Fradet for the *Cars* [104] sequence shown in the *bottom* row of fig. 6.10.

The first and last frames of the *Cars* sequence are shown in the *top* row of fig. 6.10. This sequence along with an accompanying set of 'particle' trajectories are included in a dataset

provided by Sand [104]. The 'particle' trajectories are similar to those produced by a KLT tracker. That is, each 'particle' trajectory is a set of tracked image points over consecutive frames. Unlike KLT trajectories however, the points along these 'particle' trajectories are more densely packed in the image plane.

There are two trajectory bundles produced by the algorithm of Fradet for the *Cars* sequence. The points for the trajectories in both bundle are coloured in blue and green respectively (See the *bottom* row of fig. 6.10). The green bundle contains trajectories corresponding to two cars that are spatially separated in the image plane, and both cars also have different image motions.

The *middle* row shows our segmentation of the *Cars* sequence, where four trajectory bundles are produced. The purple and blue bundles contain trajectories that exclusively correspond to the background. The yellow and green bundles contain the trajectories for car #1, while the orange bundle contains the trajectories for car #2. Here the spatial constraints in our technique allow us to differentiate between both cars unlike the approach of Fradet [46].

We also visually compared the segmentations of the *carmap* and *coastguard* sequence reported by Fradet [46]. The *carmap* sequence is 36 frames of a car driving behind a road sign. The car is occluded at several frames in this sequence. Frames 1,12, and 34 for this sequence are shown in the *top* row of fig. 6.11 from left to right respectively. The *coastguard* sequence is 82 frames of two boats passing each other travelling in opposing directions. Frames 1,41, and 81 for this sequence are shown in the *top* row of fig. 6.12.

As previously mentioned the segmentation provided by Fradet does not have any spatial constraints on the trajectories in each bundle. This is again evident in the segmentations for the *carmap* and *coastguard* sequences, where these segmentations are shown in the *bottom* rows of fig. 6.11 and fig. 6.12 respectively. The segmentations produced by our technique on both sequences are shown in the *middle* rows of fig. 6.11 and fig. 6.12 respectively. Here it may be observed that all our trajectory bundles are confined to a specific image region, due to the spatial constraints imposed on the bundles.

For the *carmap* sequence we produce more trajectory bundles as the parameters of our technique are tuned so we are more sensitive to different image motions that arise due to the perspective of the camera.

6.2.2 Using all Trajectories

It may also be noted from fig. 6.10 that Fradet [46] does not use all of the 'particle' trajectories included in the dataset for the *Cars* sequence. Fradet uses only the trajectories that are more than 14 frames long. Fradet stated in his paper [46] that short trajectories are problematic to correctly cluster with his approach. We however use all of the trajectories provided in the dataset to produce our segmentation.

The *Hand* sequence also included in the dataset provided by Sand [104] is now used to demonstrates how important it is not to discard 'short' trajectories. The first (left) and last

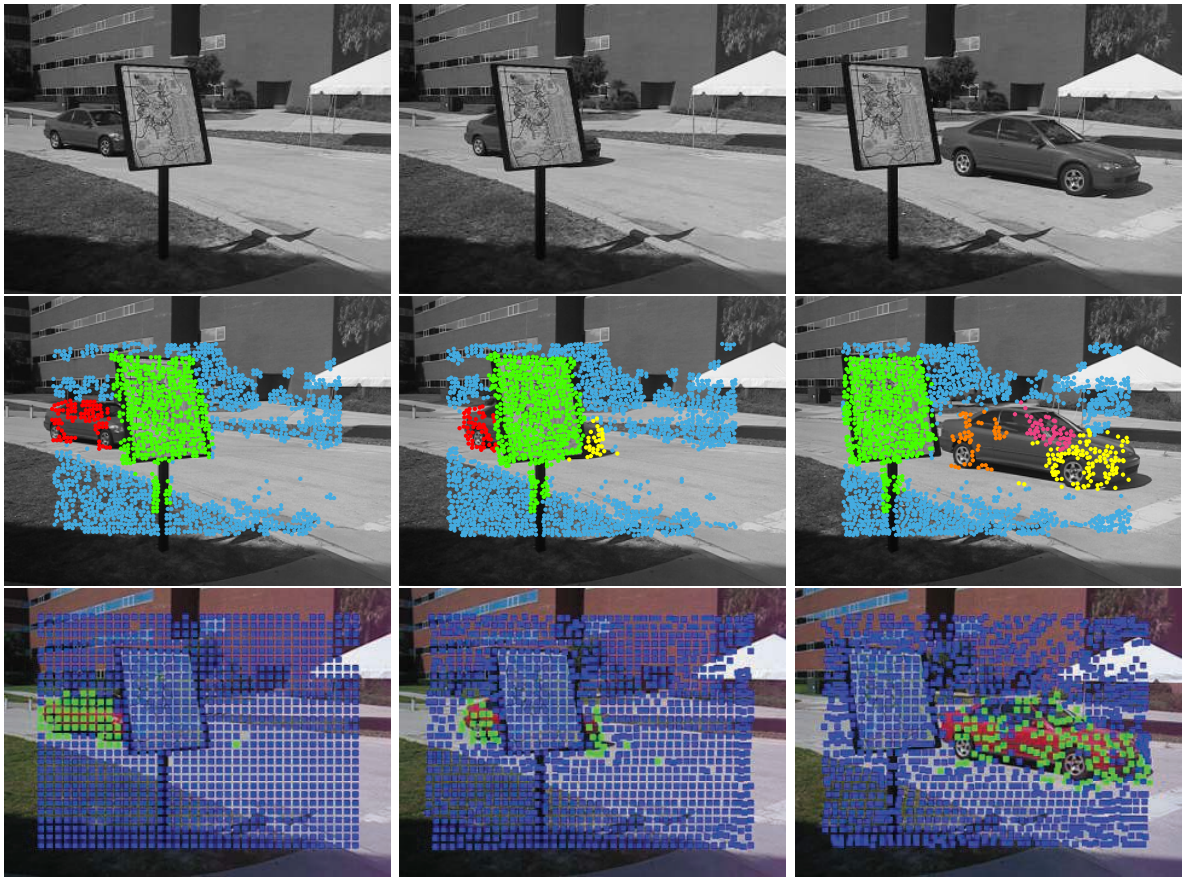

Figure 6.11: ***Top row**: Frames 1 (left), 12 (center), and 34 (right) in the **carmap** sequence. **Middle row**: Our segmentation for this sequence where six trajectory bundles are produced. The green bundle for the sign is a result of the image motion of the sign being different from the background. Here the motion difference is caused by the relative orientation and distance of the sign to the camera. Pundlik [95] has a separate bundle for the sign as well. **Bottom row**: The segmentation for this sequence produced by Fradet [46]. The blue and green bundles here contain the trajectories for the background and car respectively. Note that there are obvious spatial errors in this segmentations.*

(right) frames in this sequence are shown in the *top* row of fig. 6.13. For the Hand sequence Fradet [46] uses the trajectories that are at least 20 frames long.

The segmentation of Fradet is shown in the *bottom* row of fig. 6.13, where two trajectory bundles are produced. The green and blue bundles correspond to the trajectories of the background and hand respectively. Note here that the misclassification errors at the tips of the fingers are due to poor 'particle' trajectories and not the approach of Fradet. Some of the trajectories at the tips of the fingers have significant tracking errors. At some frames the points for these trajectories reside on the fingers, while at other frames they reside much further away from the

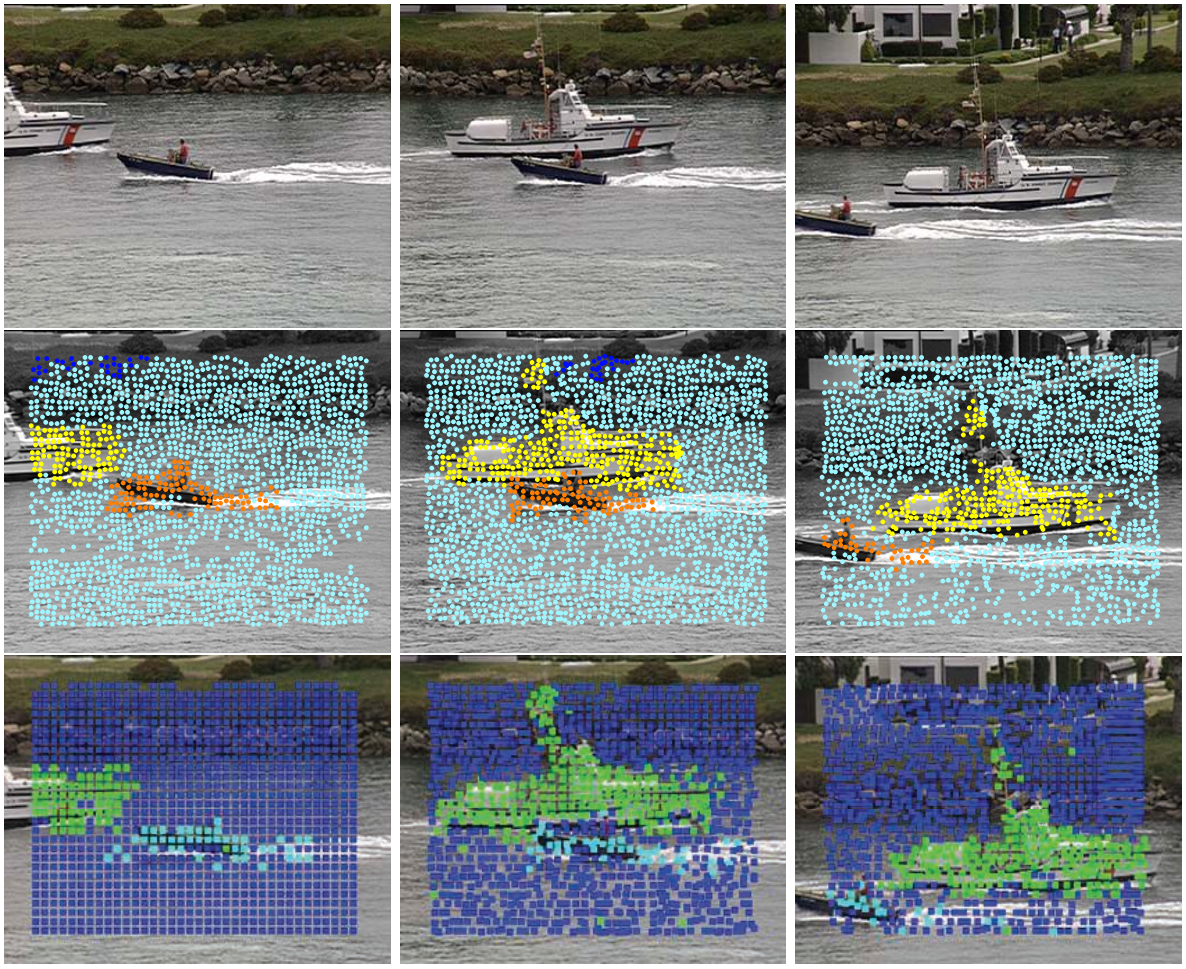

Figure 6.12: ***Top row**: Frames 1 (left),41 (center), and 81 (right) in the **coastguard** sequence. **Middle row**: Our segmentation for this sequence where four trajectory bundles are produced. The cyan and dark blue bundles contain the background trajectories, while the trajectories for boat #1 and #2 are in the yellow and orange bundles respectively. **Bottom row**: The segmentation for this sequence produced by Fradet [46]. The dark blue bundle here contains the trajectories for the background, while the trajectories for boat #1 and #2 are in the green and cyan bundles respectively. Note that there are obvious spatial errors in these segmentations.*

fingers in the background regions.

The *second* and *third* rows of fig. 6.13 show our segmentation for the *Hand* sequence. The bundles for which the majority of the trajectories in it correspond to the hand are shown in the *second* row. The *third* row in a similar sense shows the trajectory bundles for the background. Note here that the bundles either exclusively represent the background or the hand except for the *pink* bundle for the fingers. The pink bundle contains a substantial amount of the trajectories with significant tracking errors previously mentioned.

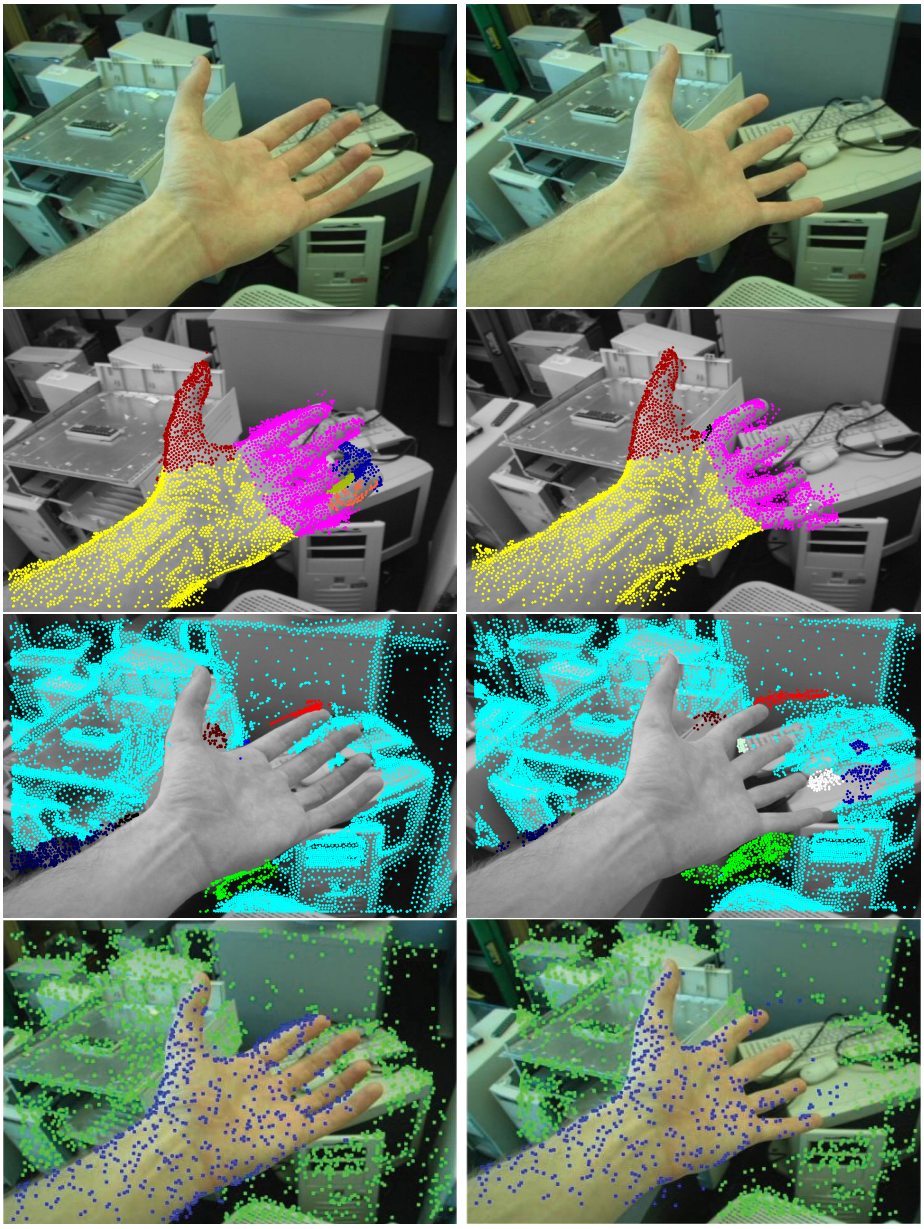

Figure 6.13: *The* ***Hand*** *sequence of 36 frames taken from the dataset provided by Sand [104].* ***Top row****: Frames 1 (left) and 36 (right) for this sequence, where an arm is moving randomly in the image plane while the fingers are flexed.* ***Second and third rows****: The trajectory bundles corresponding to the arm and background respectively.* ***Bottom row****: The segmentation of Fradet [46] for this sequence, where two bundles are produced. The green and blue bundles correspond to the trajectories of the background and hand respectively. Not all the trajectories in the dataset are used for this segmentation.*

Unlike the approach of Fradet, we used all of the sequence trajectories which allow us to differentiate between the global motion of the arm, and the local motion of the fingers. There are usually very 'short' trajectories in the image regions corresponding to non-rigid objects like the fingers. Important bundles are formed from these 'short' trajectories. These bundles describe the local non-rigid motion of the fingers. As an example, the maroon bundle describes the image motion of the thumb.

6.3 Comparison with Other 2D Algorithms: Region Growing

Another alternative 2D trajectory segmentation technique was presented by Pundlik [94] in 2006. As discussed before this method discovers trajectory bundles using a region growing strategy.

Pundlik [95] provides no quantitative analysis of his segmentation approach. Also he provides very limited proof that his approach works on sequences. He only shows the segmentation for a single frame from four sequences. Three of these sequences are called *freethrow*, *carmap*, and *calendar and mobile*. The *top* row of fig. 6.14 shows frames 5, 8, and 25 for each of these three sequences.

The *freethrow* sequence is 20 frames of a basketball player taking a free throw shot. Frames 1,12, and 20 in this sequence are shown in the *top* row of fig. 6.15. The *carmap* sequence has been introduced in the previous section, where frames 1, 12, and 34 are shown in the *top* row of fig. 6.11. Recall that Fradet [46] provides a segmentation for this sequence. The *calendar and mobile* sequence is commonly used to report motion segmentation results and has been discussed previously as well.

The *bottom* row of fig. 6.14 show the single frame segmentation provided by Pundlik [95] for the three sequences. We also show in the *middle* row of fig. 6.14 our segmentation for these frames. Note that Pundlik uses a KLT feature tracker to generate his trajectories, so we used only KLT trajectories for these three sequences as well.

6.3.1 Non-rigid Objects

From the frames provided by Pundlik it may be observed that his segmentation is spatially smooth unlike the segmentation of Fradet [46]. That is, all the trajectory bundles represent a single coherent image region. Achieving spatially smooth segmentations for the *carmap* and *calendar and mobile* sequences is much easier than for the *freethrow* sequence. The objects in the *carmap* and *calendar and mobile* sequences are rigid and well behaved. The image motion of these objects can be approximately well with 2D Affine models. However, the basketball player in the *freethrow* sequence is non-rigid, therefore the image motion for each part of his body may differ throughout the sequence.

Pundlik [95] states that for the *freethrow* sequence his segmentation approach only identified two trajectory bundles. The points for the trajectories in these bundles are shown as red dots

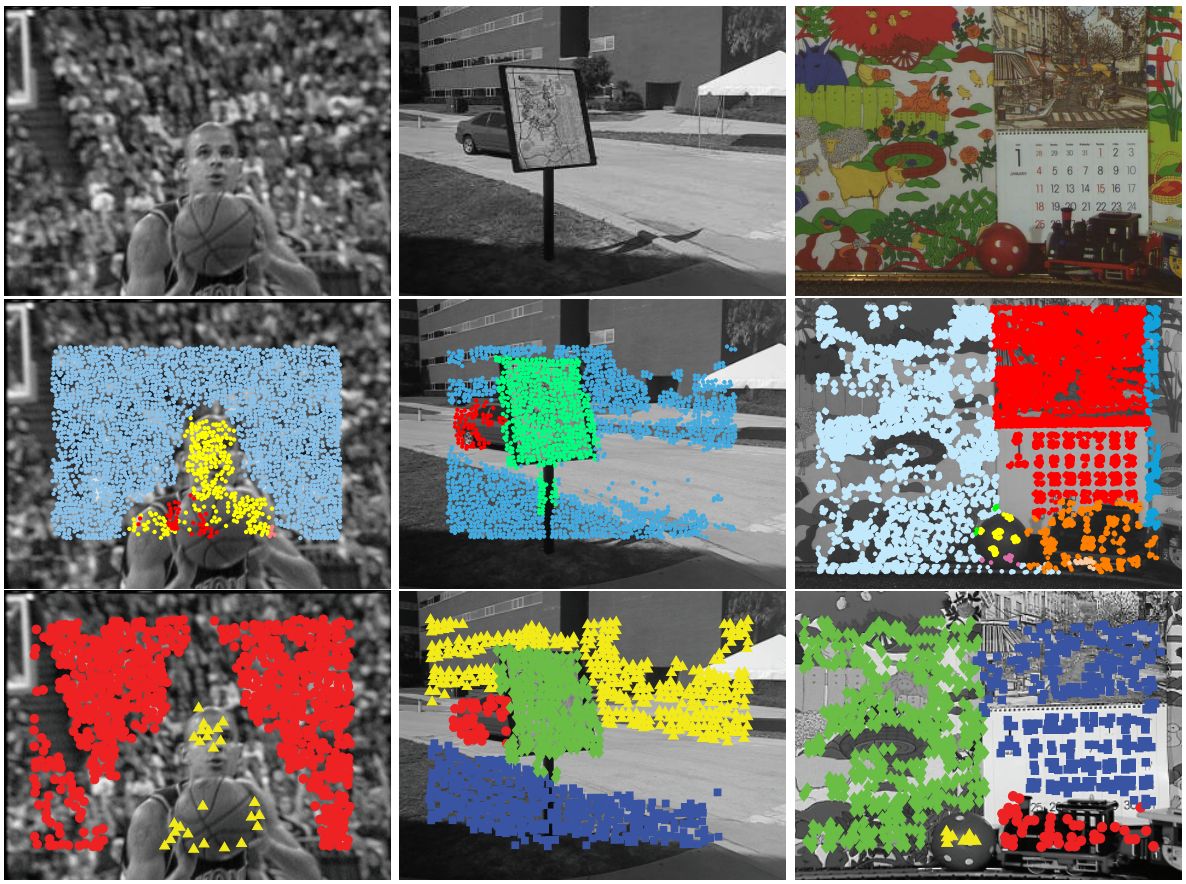

Figure 6.14: *The three sequences used for comparing the our segmentation with the segmentation of Pundlik [95].* ***Top row****: Frames 5 (left), 8 (center), and 25 (right) for the* ***freethrow****,* ***carmap****, and* ***calendar and mobile*** *sequences respectively.* ***Middle and bottom rows****: Our segmentation and the segmentation of Pundlik [95] respectively for the frames in the top row.*

and yellow triangles in the *bottom left* of fig. 6.14. However our segmentation over the entire 20 frames of the sequence (fig. 6.15) suggests that more than two trajectory bundles are required to fully describe the motion of the non-rigid basketball player.

The *bottom* row of fig. 6.15 shows our segmentation for frames 1,12, and 20 in the *freethrow* sequence. Here each bundle has a different colour. In general, the arms, chest, and face of the basketball player each has a bundle associated with it. These parts of the body move in different ways in the image plane throughout the sequence. At some instances of time however the motion of some these body parts may be described with a single trajectory bundle. As an example, in frame 1 (*bottom left* of fig. 6.15) to 5 (*middle left* of fig. 6.14) the head and shoulder regions of the player dip causing these regions to have similar image motion. Hence the yellow trajectory bundle describes the motion of both regions.

We have concluded that Pundlik discarded a lot of the trajectories associated with the bas-

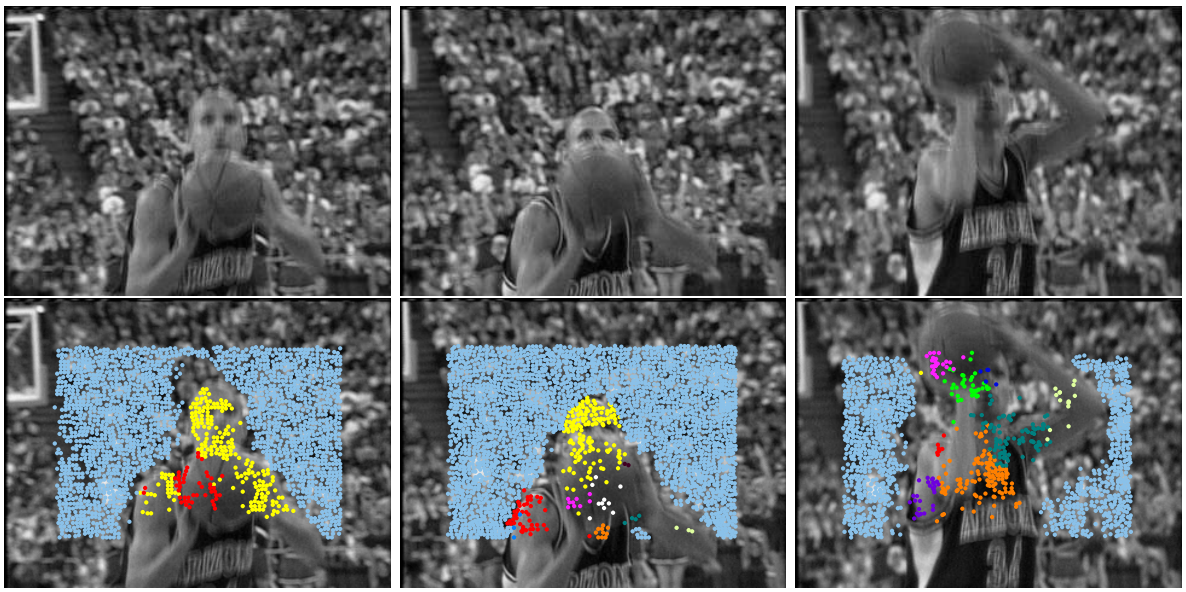

Figure 6.15: *The **freethrow** sequence of a basketball player taking a free throw shot. **Top row**: Frames 1 (left), 12 (center), and 20 (right) of this sequence from left to right. **Bottom row**: Our segmentation for the frames in the top row. The points for the trajectories are shown as dots. Each trajectory bundle has a different colour.*

ketball player in order to avoid the complexities involved in providing a reasonable segmentation for the player. Possibly he selected trajectories that had a minimum temporal length, which would automatically discard the majority of the trajectories corresponding to the player. The background image regions that are occluded at some instance throughout the sequence are void of trajectories, which supports our idea of Pundlik using the 'longest' trajectories in the sequence. Background trajectories that are occluded are usually very 'short' in duration because they are terminated prematurely. By observing our segmentation (*bottom left* of fig. 6.15) and the segmentation of Pundlik (*bottom left* of fig. 6.15) the regions where trajectories are discarded can be identified.

Recall that Fradet [46] as well discards trajectories based on temporal durations. Recall also that the consequence of discarding 'short' trajectories is that the motion of non-rigid objects can not by described fully. This is also the case for the segmentation of Pundlik [95] for the basketball player in the *freethrow* sequence.

6.3.2 Rigid Objects

For the *carmap* and *calendar and mobile* sequence where the objects are well behaved our segmentation and the segmentation of Pundlik are quite similar. See *middle and bottom* rows of fig. 6.14. Note here we used a different version of the *calendar and mobile* sequence from the one used by Pundlik. Our version had a spatial resolution of 720×576 pels, while the version that

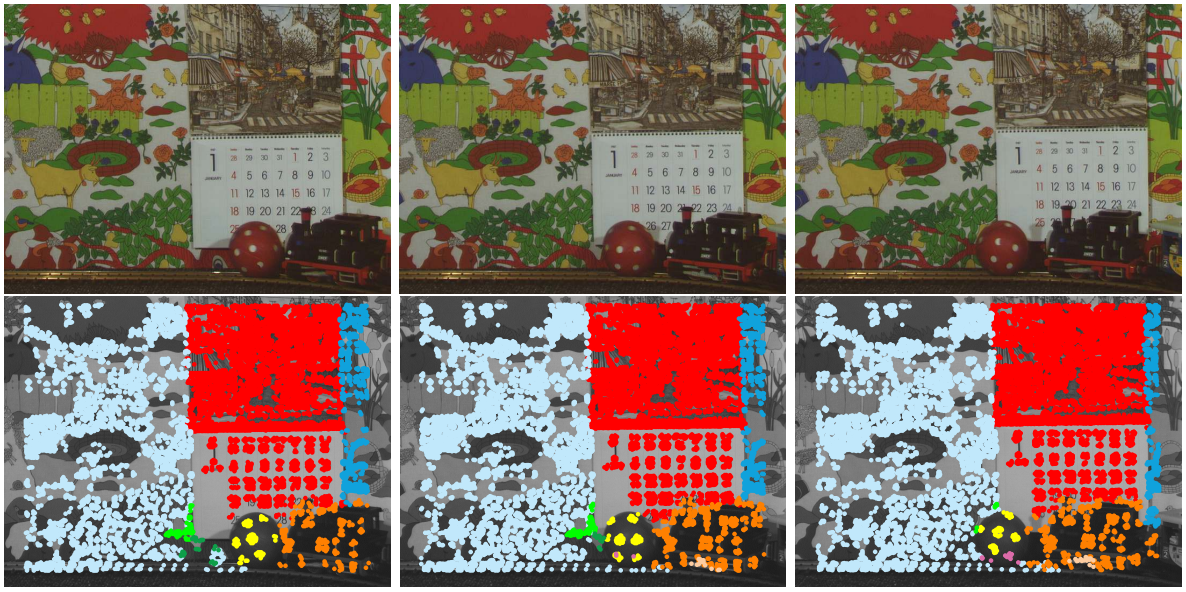

Figure 6.16: *The* ***calendar and mobile*** *sequence contains a moving calendar,ball, and toy train with the camera panning from left to right.* ***Top row****: Frames 1,12, and 25 in this sequence from left to right.* ***Bottom row****: Our segmentation for the corresponding frames in the top row.*

Pundlik made publicly available was 320×240 pels. This public version provided by Pundlik had a significantly amount of noise and compression artifacts so we did not use it. The general motion characteristics our version of the *calendar and mobile* sequence are the same as the version of Pundlik. Hence we can reasonably compare our segmentation with the segmentation of Pundlik for this sequence.

For the *calendar and mobile* sequence our segmentation produced 10 trajectory bundles. The *bottom* row of fig. 6.16 shows our segmentation for frames 1,12, and 25 in this sequence. Note that all the bundles either exclusively represent the background, calendar, ball or the train. The dark and light green bundles contain background trajectories generated due to the shadow casted by the moving ball. The orange bundle contains the trajectories corresponding to the train. The shadow that the train casts on the background has roughly the same image motion as the train itself. Therefore there are trajectories associated with this shadow that understandably included in the orange bundle.

Again Pundlik only segments the 'longest' trajectories in the *calendar and mobile* sequence, so he avoids segmenting the trajectories associated with the shadows. These shadow trajectories are of a relatively short temporal duration.

Our segmentations for frames 1, 41 and 81 of the *carmap* sequence are shown in fig. 6.11. For this sequence Pundlik can not just use the 'longest' trajectories because he would lose important trajectories for the moving car. Recall that the car is occluded by the road sign at several frames

in the sequence. Hence there are a significant number of 'short' trajectories associated with the car. Pundlik seems to use all the trajectories for this sequence to avoid obtaining a poor segmentation for the car. See our segmentation and the segmentation of Pundlik in the *center* and *bottom center* of fig. 6.14 respectively. Here the density of his trajectory points in the image plane for Pundlik's segmentation is similar to ours.

For the *carmap* segmentation in fig. 6.11 both our segmentation and the segmentation of Pundlik is roughly the same. The only difference is he has two blue and yellow trajectory bundles for the background while we have only a cyan bundle. This result is understandable as our algorithm would feasibly merge his yellow and blue bundles into one.

6.3.3 Summary

As far as alternative 2D processes are concerned our technique is overall better than the techniques of Fradet [46] and Pundlik [95]. Our main advantage over Fradet seems to derive from our spatial smoothness constraints. This means that there is good scope for incorporating these constraints into the approach of Fradet as a second step. Pundlik unfortunately does not provide enough information to make a definitive statement, but our performance on sequences with non-rigid objects is clearly better.

6.4 Real World Sequences

We will now discuss the trajectory segmentation of real world sequences. These sequences usually contain objects with significant amounts of non-rigid motions. For example, people performing articulated activities. Non-rigid motions along with occlusions give rise to the generation of incomplete trajectories. We define incomplete trajectories as trajectories that do not exist over the entire duration of a sequence from which they are generated.

Unfortunately the current 3D segmentation techniques discussed previously, can not be used to segment real world sequences that contain trajectories that do not exist over the entire duration (imcomplete trajectories) of these sequences. These techniques require the spatial locations for each trajectory at every frame in a sequence. Since incomplete trajectories do not exist at every frame, this requirement can not be fulfilled. Also 3D segmentation techniques proposed to date must be told in advance how many moving objects are in a sequence. This may be an issue when using an arbitrary sequence where the number of moving objects is unknown. Consider the case also where a sequence contains objects with perpetual or random motions, such as swaying tree branches, ocean waves, etc. For sequences with these characteristics even a human would find it difficult to identify the exact number of moving objects.

Recall that our technique does a better job of segmenting incomplete trajectories compared to the 2D approaches of Fradet [46] and Pundlik [95]. Our approach forms trajectory bundles with incomplete trajectories, while Fradet [46] and Pundlik [95] usually discards them. Our

Figure 6.17: ***Top row**: Frames 3 (left),43 (center) and 98 (right) in the **Triniman** sequence. **Second row**: Our segmentation for the frames in the top row. Here the trajectory points for the background bundle are cyan dots. All the other trajectory bundles represent the actor in the foreground. **Third and bottom rows**: A zoom on the trajectory bundles corresponding to the actor. The third row shows the points of the trajectories at the current frames, while the bottom row shows the motion history of each trajectory up to the current frames. For clarity only the motion history over the last 5 frames is shown for each trajectory.*

Figure 6.18: ***Top row**: Frames 3 (left), 57 (center) and 98 (right) in the **Artbeats-SP128** sequence. This sequence is a scene from an American football game. The foreground objects here are player #1 and #2 in red and white jerseys respectively. **Second row**: Our segmentation for the frames in the top row. Here the trajectory points for the background bundles are cyan and blue dots. All the other trajectory bundles represent player #1 and #2 in the foreground. **Third and bottom rows**: A zoom on the trajectory bundles corresponding to the players. The third row shows the points of the trajectories at the current frames, while the bottom row shows the motion history of each trajectory up to the current frames. For clarity only the motion history over the last 5 frames is shown for each trajectory.*

	Resol.	F_h	$\bar{L}$	T_h	#Bund.	M_h
Triniman	720×570	99	13	31501	10	0.32%
Artbeats-SP128	720×576	99	16	21735	37	1.66%

Table 6.6: *Tabulation for the number of frames (F_h), trajectories (T_h) and bundles (#Bund.) for the* ***Triniman*** *and* ***Artbeats-SP128*** *sequences. The average duration $\bar{L}$ in frames of the trajectories and the misclassification rate (M_h) for each sequence is shown in the* ***fourth*** *and* ***last*** *columns respectively. The spatial resolutions (Resol.) in pels for each sequence are shown in the* ***second*** *column.*

bundles with incomplete trajectories describe the local motion of non-rigid objects, which is important for obtaining a thorough description of the motions in real world sequences.

The performance of our technique was evaluated using two real world sequences called *Triniman* and *Artbeats-SP128*. Frames 3, 43 and 98 in the *Triniman* sequence are shown in the *top* row of fig. 6.17. This sequence is a dynamic outdoor scene recorded with a hand-held camcorder. The challenge in producing a reasonable segmentation for *Triniman* is that there are several non-rigid motions throughout this sequence. In the background there are swaying tree branches and the foreground actor himself is non-rigid.

Frames 3,57 and 98 in the *Artbeats-SP128* sequence are shown in the *top* row of fig. 6.18. This sequence is a scene from an American football game. The foreground objects here are player #1 and #2 (fig. 6.18) in red and white jerseys respectively. Both players are moving and deforming very quickly from left to right in the image plane. Player #1 partially occludes player #2 from frames 73-99. The challenges here are coping with the non-rigid motions and maintaining exclusive trajectory bundles for both players.

The next sections discuss the performances of our technique on both sequences in more detail.

6.4.1 Real World Segmentation

We will now discuss our segmentation for the *Triniman* and *Artbeats-SP128* sequences. Table 6.6 shows the number of trajectories (T_h) and frames (F_h) for both sequences. Also shown in this table is the number of trajectory bundles (#Bund.) our technique produces for both sequences. The *Artbeats-SP128* sequence has more rapid non-rigid motion compared to the *Triniman* sequence. Hence the *Artbeats-SP128* sequence has 37 trajectory bundles whereas the *Triniman* sequence has 10 bundles.

Our segmentation for the *Triniman* sequence is shown in the *second* row of fig. 6.17. Here all the background trajectories are in the cyan bundle, while all the other bundles contain the trajectories for the actor in the foreground. See the zoom on the actor in the *third* and *bottom* rows of fig. 6.17. The image motion of the head, torso and lower arms of the actor are all different

throughout the sequence. Hence each of these body parts has a trajectory bundle associated with it. For example, the yellow bundle describes the motion of the head. The motion difference between the various parts of the body can be observed from the illustration in the *bottom* row of fig. 6.17 where the motion history of trajectories in the bundles are shown.

The segmentation for the *Artbeats-SP128* sequence is shown in the *second* row of fig. 6.18. The background trajectory bundles are blue and cyan dots, while all the other bundles represent the players in the foreground. Note here that the camera is static for this sequence unlike the *Triniman* sequence. Hence the cyan and blue trajectory points are roughly in the same place over the entire sequence when looking at the motion history of the trajectories (*bottom* row of fig. 6.18).

See the zoom on the players in the *third* and *bottom* rows of fig. 6.17. It may be observed that the foreground bundles either exclusively represent player #1 or #2. The legs of player #1 are the fastest moving objects in the sequence, and their image motions are well described by various trajectory bundles.

6.4.2 Misclassification Rate for Real World Sequences

We evaluated the misclassification rates for the *Triniman* and *Artbeats-SP128* sequence using manually drawn ground truth mattes. Some of these ground truth mattes are shown in fig. 6.19. These mattes label every pixel as belonging to a particular object. For example, the mattes for the *Artbeats-SP128* sequence (*bottom* row of fig. 6.19) label each pixel as either background (green), player #1 (blue), or player #2 (red).

Recall that the nth trajectory bundle represents object $\mathcal{O}_m$ if the majority of the trajectories in bundle n correspond to object $\mathcal{O}_m$. Fig. 6.20 will be used to demonstrate how the objects the bundles represent are identified. The trajectory bundles at frame 8 in the Triniman sequence at shown in the *top left* of fig. 6.20. The two objects of interest in this sequence are the actor and background which are coloured red and green respectively in the ground truth matte in the *top right* of fig. 6.20. The actor is represented by the yellow, red, and light green bundles, since majority the points for the trajectories in these bundles reside in the region of the matte corresponding to the actor. In a similar sense the cyan bundle represents the background region of the image. The bundles for the actor and background are shown in the *bottom* row of fig. 6.20.

A trajectory $\mathcal{X}_t$ in bundle n is defined as misclassified if all the points (x_f^t, y_f^t) along this trajectory reside outside the matte region for the object that bundle n represents. The *top row* of fig. 6.21 shows the misclassified trajectory points at frame 8 in the *Triniman* sequence. The illustrations on the *left* and *right* of fig. 6.21 (top row) show the trajectories in the bundles that represent the actor and background respectively. The misclassified trajectory points are shown as black dots in both cases. Note here that the misclassified trajectories generally reside close to boundaries between the various object regions.

The misclassification rate for the *Triniman* and *Artbeats-SP128* sequences are 0.32% and

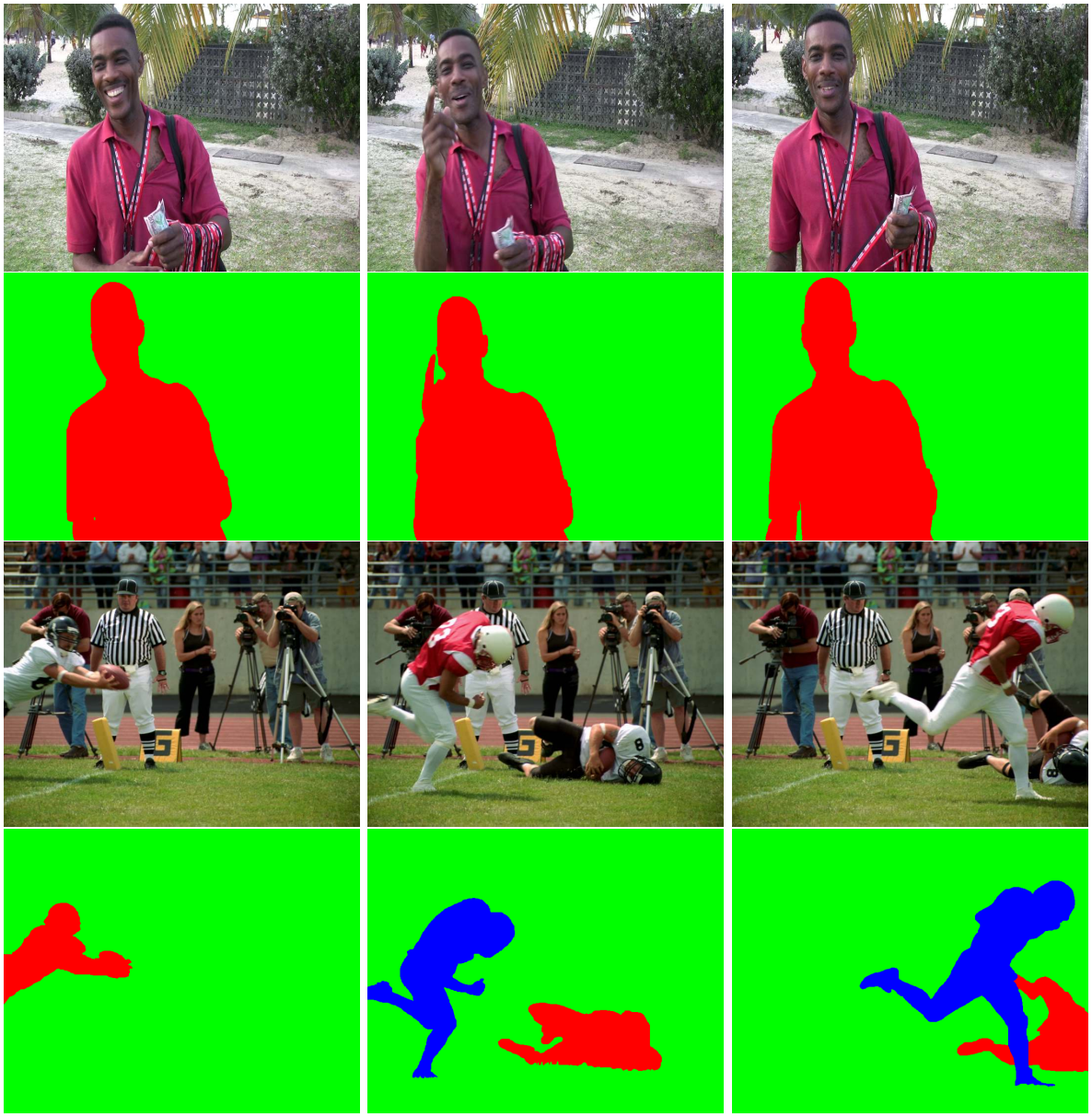

Figure 6.19: *Ground truth mattes for the* ***Triniman*** *and* ***Artbeats-SP128*** *sequences.* ***Top row****: Frames 3 (left) ,43 (center) and 98 (right) in the* ***Triniman*** *sequence.* ***Second row****: Ground truth mattes for the frames in the top row. The actor and background are coloured red and green respectively.* ***Third row****: Frames 3 (left),57 (center) and 98 (right) in the* ***Artbeats-SP128*** *sequence.* ***Bottom row****: Ground truth mattes for the frames in the third row. The background, player #1 and #2 are coloured green, blue, and red respectively.*

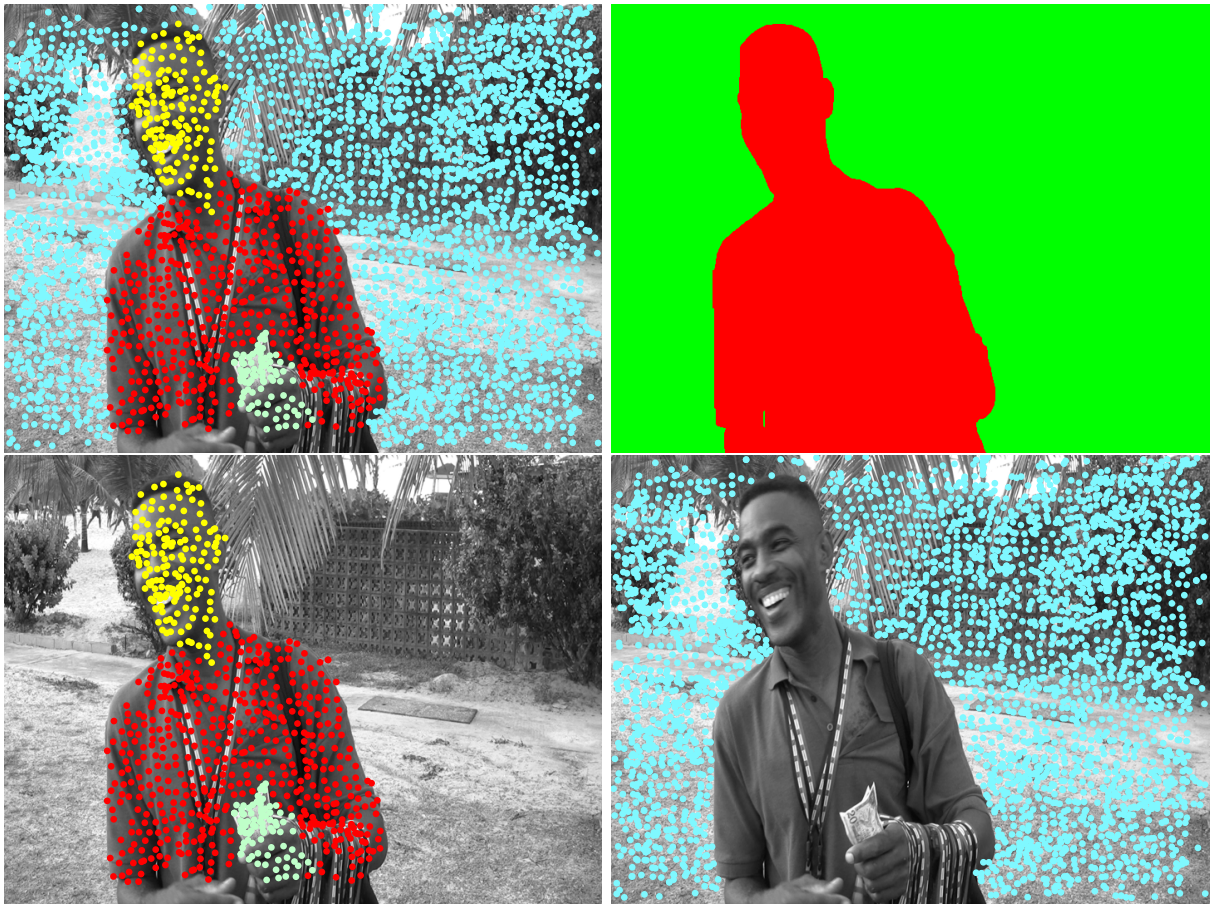

Figure 6.20: ***Top left**: Our segmentation for frame 8 in the **Triniman** sequence. **Top right**: The ground truth matte for frame 8. **Bottom left**: The trajectory bundles that represent the actor in the foreground. The points for the trajectories in these bundles are yellow, red and light green dots. **Bottom right**: The trajectory bundle that represents the background. The points for the trajectories in this bundle are cyan dots.*

1.66% respectively. These low misclassification rates confirm that our technique does create trajectory bundles that exclusively represent a single object.

In general the majority of the misclassified trajectories are about 10 pixels outside the image regions they are required to reside in. The *middle* row of fig. 6.21 shows how the misclassified trajectories change as the matte for the actor in *Triniman* at frame 8 is dilated by 0 (left) ,4 (middle) and 6 (right) pixels respectively. For the dilations of 0, 4 and 6 pixels the number of misclassified trajectories for the actor are 3,1, and 0 respectively. Similarly, the dilation of the background matte by 0 (left), 4 (middle) and 6 (right) pixels in shown in the *bottom* row of fig. 6.21. Here the number of misclassified trajectories for the background are 1,1, and 0 respectively.

Fig. 6.22 plots the percentage misclassification rate M_h versus the dilation d in pixels applied

to object regions for both the *Triniman* and *Artbeats-SP128* sequences. The plot for the these sequences are in green and black respectively. It may be observed from these plots that more than 50% of the misclassified trajectories are no more than 3 pixels away from their respective object regions. These misclassified trajectories usually occur when the edge of an object $\mathcal{O}_m$ is tracked but the points for the trajectories fall just outside the image region for object $\mathcal{O}_m$. See the *top left* illustration in fig. 6.21 where the three misclassified trajectories (black dots) are examples of trajectories tracking an edge feature.

6.4.3 Summary

This chapter has analysed the performance of our technique in some detail. A wide variety of experiments has shown that we are able to perform sparse segmentation better than several previous techniques. It is interesting that our process seems robust to motion blur and difficult motion in general. This is most likely due to the long term information used.

The main issue outstanding is that of computational complexity. We do not consider this here but further comments are left for the final chapter.

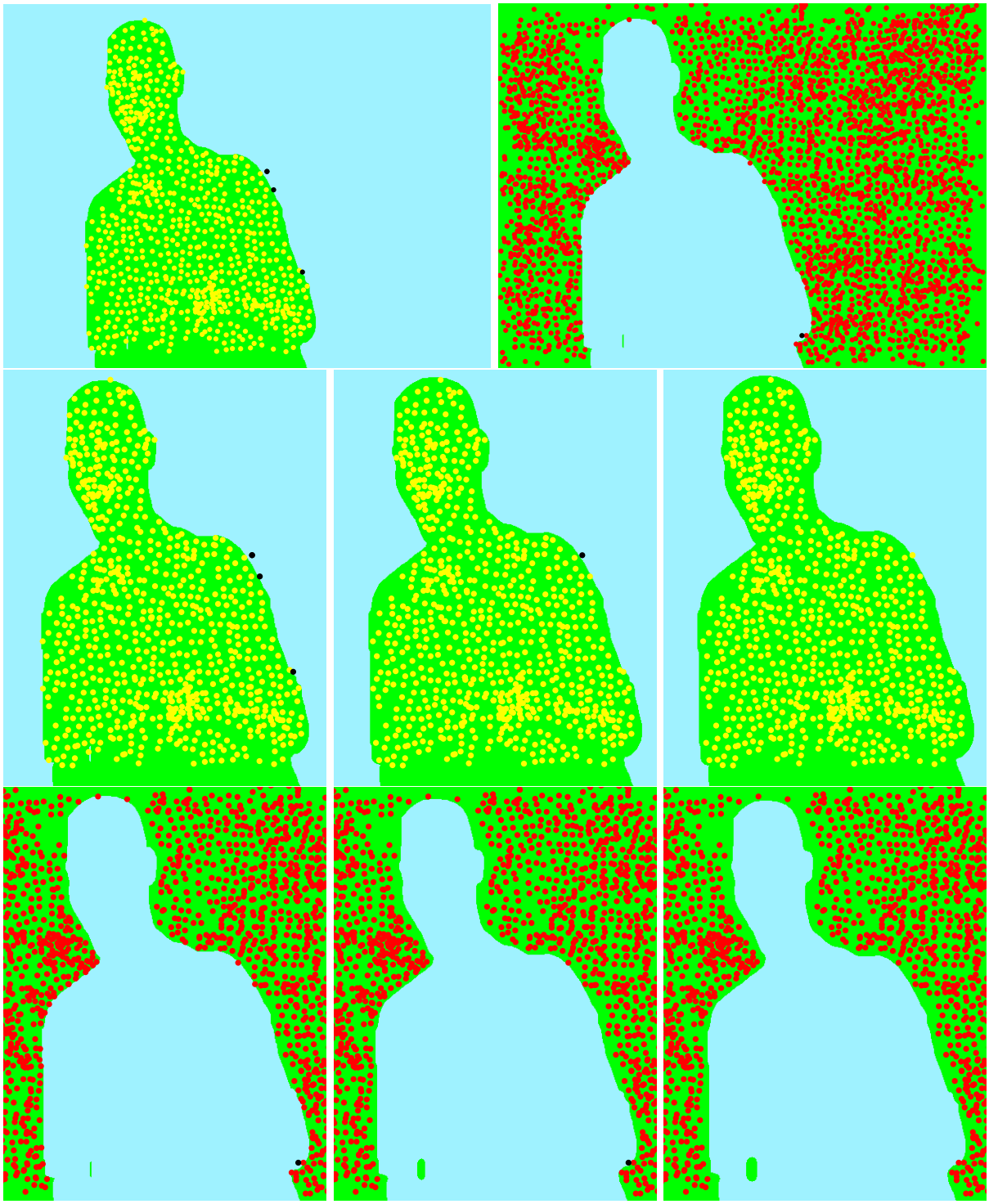

Figure 6.21: *The trajectory points (dots) at frame 8 in the* ***Triniman*** *sequence, where misclassified points are shown as black dots.* ***Top left****: The trajectory points in the bundles that represent the actor. The image region of the actor is coloured green and the trajectory points that reside inside this region are yellow dots.* ***Top right****: The trajectory points for the bundle that represent the background. The trajectory points here that reside inside the background region are shown as red dots.* ***Middle and bottom rows****: The dilation of the actor and background image regions respectively. From left to right the dilations applied are 0,4 and 6 pixels respectively. As the object regions are dilated the number of misclassified trajectories (black dots) decreases.*

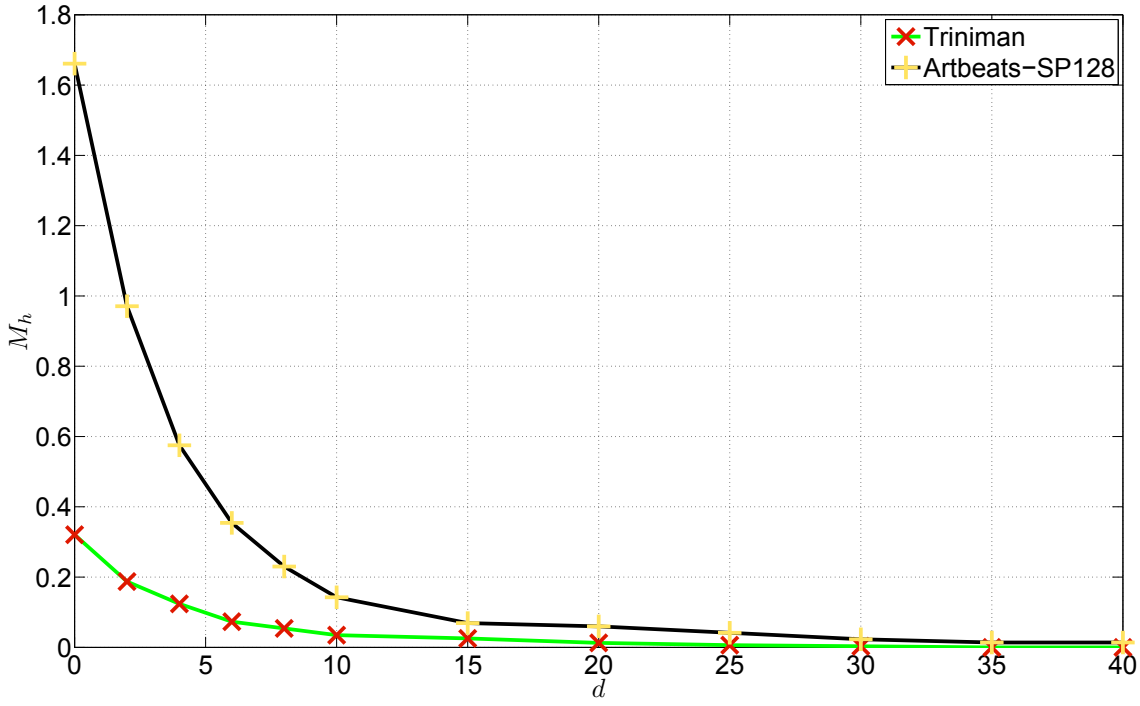

Figure 6.22: *The plots of the percentage misclassification rates M_h versus the dilation d in pixels applied to object regions for both the Triniman and Artbeats-SP128 sequences. The plot for the both sequences are in green and black respectively.*

7

Dense Pixel Segmentation

Consider that a sequence of F frames has O objects we would like to segment. The task of a dense pixel segmentation algorithm is to assign object labels $\mathcal{L}_f(\mathbf{s})$ to the pixel sites $\mathbf{s} = (x, y)$ at every frame $f = \{1 : F\}$ in the sequence.

Fig. 7.1 shows our aspirations for the resulting dense segmentations for frames 3 and 98 from the *Triniman* and *Artbeats-SP128* sequences respectively. Note that for both sequences the number of objects are $O = 2$ and $O = 3$ respectively. The object label fields $\mathcal{L}_f$ for the frames in fig. 7.1 are shown in the right column. The pixel object labels are $\mathcal{L}_f(\mathbf{s}) = \{1 : 2\}$ (green and red) and $\mathcal{L}_f(\mathbf{s}) = \{1 : 3\}$ (green, red and blue) for both sequences respectively.

There have been several previous dense pixel segmentation techniques [57,64,127]. The main challenge for these techniques is establishing temporal consistency in the object label field $\mathcal{L}_f(\mathbf{s})$ from frame to frame in a sequence. Unlike these previous approaches, we achieve temporal consistency by using long term trajectory information to guide our dense segmentation process.

Recall that we presented a sparse trajectory segmentation algorithm in chapter 4 and 5. Our dense segmentation algorithm uses the trajectory bundles obtained from the sparse trajectory segmentation step to estimate motion and appearance likelihoods for the O objects. These likelihoods along with a 2D Markov Random Field (MRF) prior are used to generate a maximum a posteriori estimate of the object labels $\mathcal{L}_f(\mathbf{s})$ in a sequence. The posterior distribution for the object labels $\mathcal{L}_f(\mathbf{s})$ at frame f given the pixel neighbourhoods $\mathcal{L}_f(\sim \mathbf{s})$, the feature point trajectories ($\mathcal{X}$) and image data ($\mathbf{I}$) is given below.

$$p(\mathcal{L}_f(\mathbf{s})|\mathcal{X}, \mathbf{I}, \mathcal{L}_f(\sim \mathbf{s})) \propto p_x(\mathcal{X}(f, \mathbf{s})|\mathcal{L}_f(\mathbf{s}))p_i(\mathbf{I}_f(\mathbf{s})|\mathcal{L}_f(\mathbf{s}))p_s(\mathcal{L}_f(\mathbf{s})|\mathcal{L}_f(\sim \mathbf{s})) \qquad (7.1)$$

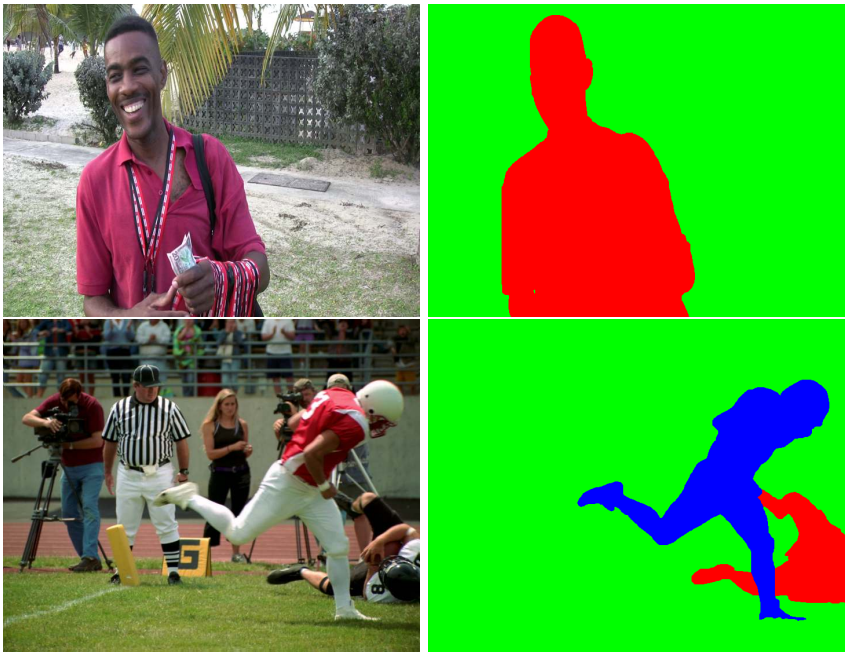

Figure 7.1: *Examples of the pixel labels required from a dense segmentation process.* ***Top row****: Frame 3 from the* ***Triniman*** *sequence shown on the left, with the object labels for the pixels in this frame shown on the right. Here the pixel labels for the two objects of interest are coloured red (foreground actor) and green (background) respectively.* ***Bottom row****: Frame 98 from the* ***Artbeats-SP128*** *sequence shown on the left, with the object labels for the pixels in this frame shown on the right. The objects of interest here are the background, player #1 and #2. The pixels for each object are coloured green, blue, and red respectively.*

Where $p_x(\mathcal{X}(f,\mathbf{s})|.)$ is the likelihood of the pixel site s at frame f following the motion $\mathcal{X}(f,\mathbf{s})$ defined by a particular object label $\mathcal{L}_f(\mathbf{s})$. The appearance likelihood $p_i(\mathbf{I}_f(\mathbf{s})|.)$ compares the observed colour $\mathbf{I}_f(\mathbf{s})$ at pixel site $(f,\mathbf{s})$ with a colour model for object $\mathcal{L}_f(\mathbf{s})$. Label smoothness is injected through the MRF prior $p_s(.)$.

The α-*expansion* Graphcut algorithm [26] is used to solve for the MAP estimate of the object label field $\mathcal{L}_f(\mathbf{s})$. The adaptation of the Bayesian problem in eq. 7.1 above to this Graphcut solution will be discussed later.

We have two versions of our dense segmentation algorithm that are discussed in detail later. One version is a fully automatic segmentation process, .i.e. no user intervention is required. To produce this automatic segmentation we assume that every trajectory bundle represent a single object. These bundles are obtained from our sparse trajectory segmentation process. Fig. 7.3 shows examples of the segmentation our automatic dense segmentation algorithm produces for the *Triniman* sequence. Frames 8, 28 and 97 are shown in the *left column*, and our corresponding segmentations for these frames are shown in the *middle column*. Note here that each trajectory

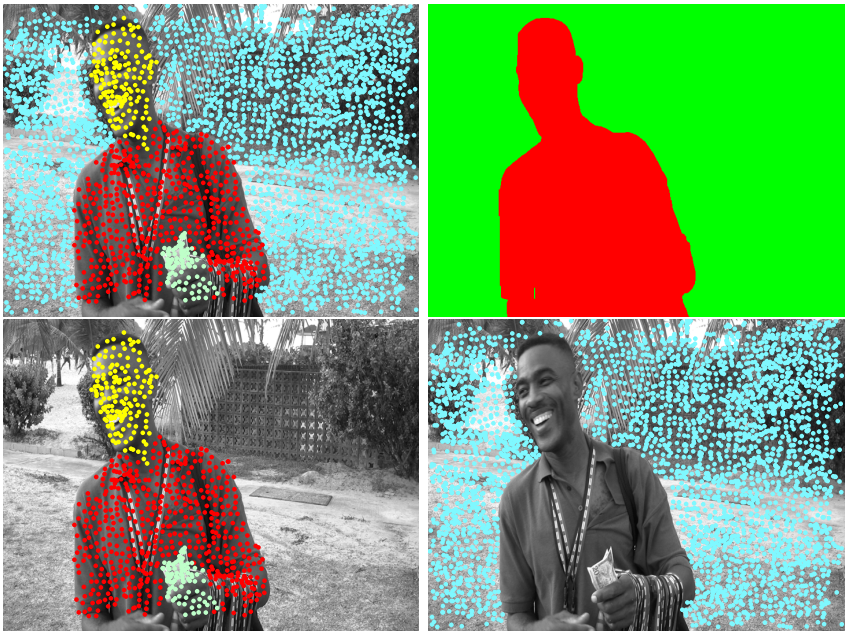

Figure 7.2: ***Top left****: Our sparse trajectory segmentation for frame 8 in the* ***Triniman*** *sequence.* ***Top right****: The user defined matte for frame 8.* ***Bottom left****: The trajectory bundles that represent the actor in the foreground. The points for the trajectories in these bundles are yellow, red and light green dots.* ***Bottom right****: The trajectory bundle that represents the background. The points for the trajectories in this bundle are cyan dots.*

bundle shown in the *right column* represents a single object in the dense segmentation (middle column).

The other version of our dense segmentation algorithm is a semi-automatic segmentation process. In this process we use user segmented mattes at roughly every 10^{th} frame in a sequence to estimate the appearance likelihoods $p_i(\mathbf{I}_f(\mathbf{s})|.)$ for the objects. We also determine from the user defined mattes the trajectory bundles associated with each object. The trajectory bundles for a particular object are used to estimate motion models that describe the image motion of that object. These motion models are then used to evaluate the motion likelihood $p_x(\mathcal{X}(f,\mathbf{s})|.)$ of the objects.

A user defined matte for frame 8 in the *Triniman* sequence is shown in the *top right* of fig. 7.2. The trajectory bundles that exist this frame are also shown in the *top left* of fig. 7.2. The objects for this sequence are the actor and the background. Both objects are coloured in *red* and *green* respectively in the user defined matte. Consider this user defined matte at frame f as a reference object label field $\mathcal{M}_f(\mathbf{s})$, i.e. $\mathcal{L}_f(\mathbf{s}) = \mathcal{M}_f(\mathbf{s})$ at frame f. The pixel site $\mathbf{s}_o$ resides in the image region corresponding to object o if $\mathcal{M}_f(\mathbf{s}_o) == o$. We derive appearance models for object o by modelling the colour information from the pixel sites $\mathbf{s}_o$ for this object.

The nth trajectory bundle represents object o if the majority of the trajectories in bundle

Figure 7.3: *Examples of the segmentation produced by our automatic dense segmentation algorithm for the* ***Triniman*** *sequence.* ***Left Column****: Frames 8 (top), 28 (center), and 97 (bottom) from this sequence.* ***Middle column****: Our dense segmentation for the corresponding frames on the left.* ***Right column****: Our sparse trajectory segmentation of the trajectories that exist at the corresponding frames in the left column. Here the trajectory bundles and their corresponding image regions in the middle column are coloured in a similar manner. Each trajectory bundle is assumed to represent a single object in our automatic dense segmentation.*

n resides in the image region defined for object o. Hence if the nth bundle represents object o, this bundle is used to model the image motion of object o. As an example, the yellow, red and light green bundles in the *bottom left* of fig. 7.2 are used to model the motion of the actor in the *Triniman* sequence. In a similar sense the background image motion is modelled by the cyan trajectory bundle (*bottom right* of fig. 7.2).

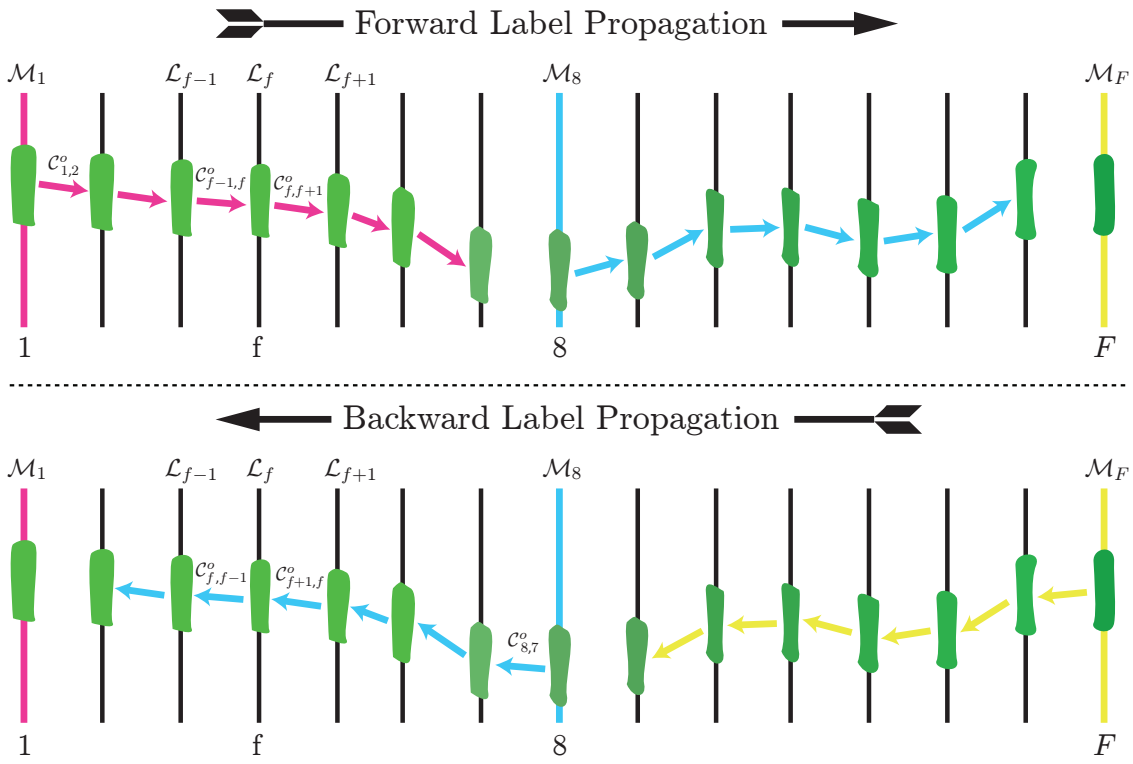

Figure 7.4: *The estimation of the appearance models for the objects in a sequence using* ***forward*** *and* ***backward*** *object label propagation steps.* ***Top row****: The propagation of the appearance models* $\mathcal{C}^o_{f-1,f}$ *for object* o *(green blob) in the* ***forward*** *direction from frame* $1:F$*.* ***Bottom row****: The propagation of the appearance models* $\mathcal{C}^o_{f+1,f}$ *for object* o *(green blob) in the* ***backward*** *direction from frame* $F:1$*. The appearance information is propagated from the* ***key*** *frames (****coloured lines****) to the* ***intermediate*** *frames (****black lines****). The* ***arrows*** *having the same colour as the key frames indicate which* ***key*** *frame is being propagated to the subsequent* ***intermediate*** *frames.*

7.1 Dense Segmentation Framework

The frameworks for both our semi-automatic and automatic dense segmentation algorithms are the same. We have a set of 'reference' frames $\mathcal{M}_f$ from which we estimate the appearance likelihoods for the objects. We define these 'reference' frames $\mathcal{M}_f$ as *key* frames. For the semi-automatic dense segmentation these *key* frames $\mathcal{M}_f$ are user defined mattes. For the automatic process the *key* frames are automatically derived from the trajectory bundles in a process described later (See section 7.1.2). The *top row* of fig. 7.4 shows an illustration of a sequence of F frames that contains three *key* frames. These *key* frames (coloured lines) are $\mathcal{M}_1$, $\mathcal{M}_8$, and $\mathcal{M}_F$ located at frames $f = \{1, 8 \text{ and } F\}$ respectively.

Now the frames between the *key* frames are defined as *intermediate* frames. In the *top row* of

fig. 7.4 these *intermediate* frames (black lines) are frames $f = \{2, \ldots, 7, 9, \ldots, F-1\}$. The general task now is to propagate the object labels from the *key* frames into the *intermediate* frames. This involves generating the MAP estimate for the object label fields $\mathcal{L}_f$ for the *intermediate* frames f. The posterior distribution used to estimate these object labels $\mathcal{L}_f$ was presented in the previous section and repeated as follows.

$$p(\mathcal{L}_f(\mathbf{s})|\mathcal{X}, \mathbf{I}, \mathcal{L}_f(\sim \mathbf{s})) \propto p_x(\mathcal{X}(f, \mathbf{s})|\mathcal{L}_f(\mathbf{s}))p_i(\mathbf{I}_f(\mathbf{s})|\mathcal{L}_f(\mathbf{s}))p_s(\mathcal{L}_f(\mathbf{s})|\mathcal{L}_f(\sim \mathbf{s})) \tag{7.2}$$

When segmenting an *intermediate* frame f it is reasonable to gather appearance information from the closest *key* frames to this frame f. We are using the closest *key* frames on both sides of the *intermediate* frame f to gather appearance information. This helps to disambiguate image occlusions. The pair of *key* frames on both sides of the *intermediate* frame f are defined as *border key* frames. As an example, for the *intermediate* frame at frames $f = \{9, \ldots, F-1\}$ (see fig. 7.4) their *border key* frames are $\mathcal{M}_8$ (cyan line) and $\mathcal{M}_F$ (yellow line).

In order to propagate appearance information from both *border key* frames to the *intermediate* frame f, we perform forward and backward label propagation steps. For the forward propagation step the frames in a sequence are processed in the frame order 1 to F. That is, frame $f-1$ is segmented before frame f can be segmented. Hence the segmentation at frame f is dependent on the segmentation at frame $f-1$. The object label field $\mathcal{L}_{f-1}$ estimated at frame $f-1$ is used to generate appearance models for frame f. These appearance models are used to evaluate the appearance likelihoods $p_i(\mathbf{I}_f(\mathbf{s})|.)$ for frame f.

For the backward propagation step the processing is from frame F to 1. Here the segmentation at frame f is dependent on the segmentation at frame $f+1$. The object label field $\mathcal{L}_{f+1}$ estimated at frame $f+1$ is used to generate appearance models for frame f.

We now go on to discuss aspects of the design of the various probability distribution functions (pdfs) first before presenting the actual segmentation process.

7.1.1 Propagation of Appearance Models

Consider that object o has an appearance model $\mathcal{C}^o_{\alpha,\beta}$ that is obtained from propagating appearance information from frame α to frame β. Here we use a collection of colour samples and their spatial locations to construct the appearance model $\mathcal{C}^o_{\alpha,\beta}$ for object o. These samples are obtained from the segmentation for object o at frame α. See section 7.1.7 for more details on this appearance model.

For the forward label propagation step these appearance models are propagated from frame $\alpha = f-1$ to frame $\beta = f$. Therefore the appearance models in this propagation step are of the form $\mathcal{C}^o_{f-1,f}$. The *pink* arrows in the *top row* of fig. 7.4 indicate the direction of propagation of the appearance models from the first (pink line $\mathcal{M}_1$) *key* frame to the *intermediate* frames $f = \{2, \ldots, 7\}$. Note here that the appearance models $\mathcal{C}^o_{1,2}$, $\mathcal{C}^o_{f-1,f}$, and $\mathcal{C}^o_{f,f+1}$ are labelled in this illustration.

For the backward propagation step the frames in a sequence are segmented in the frame order F to 1. Here the segmentation at frame f is dependent on the segmentation at frame $f+1$. Hence in the backward propagation step the appearance models $\mathcal{C}^o_{f+1,f}$ for the objects are propagated from frame $f+1$ to frame f. The *cyan* arrows in the *bottom row* of fig. 7.4 indicate the direction of propagation of these appearance models from the *key* frame $\mathcal{M}_8$ (cyan line) to the *intermediate* frames $f = \{7, \ldots, 2\}$. Note here that the appearance models $\mathcal{C}^o_{8,7}$, $\mathcal{C}^o_{f+1,f}$, and $\mathcal{C}^o_{f,f-1}$ are labelled in this illustration.

7.1.1.1 Appearance from Forward and Backward Segmentations

The forward and backward label propagation steps discussed previously are performed to obtain appearance models for the objects in a sequence. The appearance likelihood term $p_i(\mathbf{I}_f(\mathbf{s})|.)$ in eq. 7.2 is different for both propagation steps. For the forward step we only have the object appearance models $\mathcal{C}_{f-1,f}$ propagated from frame $f-1$ to frame f when estimating the labels $\mathcal{L}_f$. Therefore the appearance likelihoods here $p_i(\mathbf{I}_f(\mathbf{s})|.)$ are only dependent on these estimated appearance models $\mathcal{C}_{f-1,f}$. Similarly for the backward step we only have the object appearance models $\mathcal{C}_{f+1,f}$ propagated from frame $f+1$ to frame f when estimating the labels $\mathcal{L}_f$. Hence the appearance likelihoods $p_i(\mathbf{I_f}(\mathbf{s})|.)$ in this step are only dependent on the estimated appearance models $\mathcal{C}_{f+1,f}$.

After the appearance models $\mathcal{C}_{f-1,f}$ and $\mathcal{C}_{f,f+1}$ have been estimated from the forward and backward propagation steps respectively, we can proceed to doing the final segmentation. For this final segmentation the appearance likelihood $p_i(\mathbf{I}_f(\mathbf{s})|.)$ uses both appearance models $\mathcal{C}_{f-1,f}$ and $\mathcal{C}_{f,f+1}$. The various forms of the appearance likelihood $p_i(\mathbf{I}_f(\mathbf{s})|.)$ used for the propagation steps and the final segmentation are summarized in eq. 7.28 below.

$$p_i(\mathbf{I}_f(\mathbf{s})|\mathcal{L}_f(\mathbf{s})) = \begin{cases} p(\mathbf{I}_f(\mathbf{s})|\mathcal{L}_f(\mathbf{s}), \mathcal{C}_{f-1,f}(\mathbf{s})), & \text{forward propagation} \\ p(\mathbf{I}_f(\mathbf{s})|\mathcal{L}_f(\mathbf{s}), \mathcal{C}_{f+1,f}(\mathbf{s})), & \text{backward propagation} \\ p(\mathbf{I}_f(\mathbf{s})|\mathcal{L}_f(\mathbf{s}), \mathcal{C}_{f+1,f}(\mathbf{s}), \mathcal{C}_{f-1,f}(\mathbf{s})), & \text{final segmentation} \end{cases} \tag{7.3}$$

Note that the motion likelihoods $p_x(\mathcal{X}(f, \mathbf{s})|.)$ and the prior distributions $p_s(.)$ do not change for both label propagation steps and the final segmentation.

7.1.2 Selecting Key Frames for Automatic Segmentation

For the semi-automatic dense segmentation the *key* frames are the user segmented mattes. However for the automatic dense segmentation, we generate estimated *key* frames according to the starts and ends of the trajectory bundles in a sequence. When a trajectory bundle corresponding to object o starts at frame f, the object label o is a new label introduced at this frame. Hence this label must be included in the list of possible labels that can be assigned in the object label field $\mathcal{L}_f$. Therefore a new *key* frame is required at this point. Similarly when a

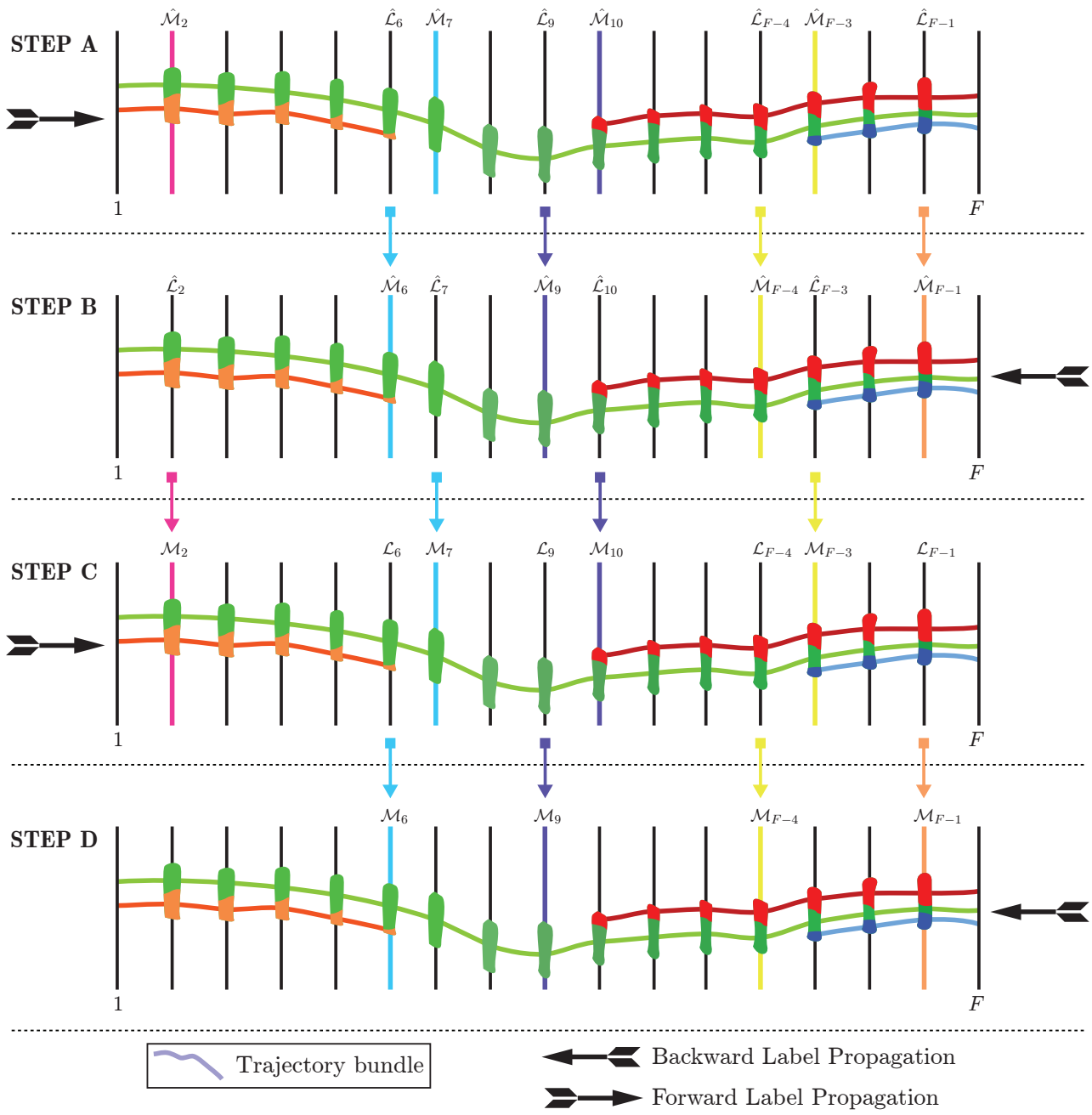

Figure 7.5: *The generation of* ***key*** *frames for the automatic dense segmentation process, where these* ***key*** *frames are refined from* ***STEP A*** *to* ***STEP D****.* ***STEP A*** *and* ***STEP C*** *are* ***forward*** *propagation steps in which the frames are segmented in the order* $1:F$*.* ***STEP B*** *and* ***STEP D*** *are* ***backward*** *propagation steps in which the frames are segmented in the order* $F:1$*. The trajectory bundles are represented by the* ***curved coloured*** *lines that extend across the frames. In the* ***forward*** *propagation steps (****STEP A*** *and* ***STEP C****) frame* f *is selected as a* ***key*** *frame (****coloured lines****) if a trajectory bundle starts at that frame* f*, or a bundle ended in the previous frame* $f-1$*. In the* ***backward*** *propagation steps (****STEP B*** *and* ***STEP D****) frame* f *is selected as a* ***key*** *frame (****coloured lines****) if a trajectory bundle starts at that frame* f*, or a bundle ended in the previous frame* $f+1$*. The coloured* ***arrows*** *indicate how the key frames* $(\mathcal{M}_f, \hat{\mathcal{M}}_f)$ *are selected from the segmentations* $(\mathcal{L}_f, \hat{\mathcal{L}}_f)$ *at the previous propagation steps.*

trajectory bundle ends, we must remove the object label corresponding to this bundle from the list of possible labels.

The *top row* of fig. 7.5 (**STEP A**) demonstrates the idea of how *key* frames are selected in the automatic dense segmentation process. Here the estimated *key* frames (coloured lines) are labelled $\hat{\mathcal{M}}_f$. The trajectory bundles are represented by the coloured curved lines extending across the frames. As an example shown in the *top row* of fig. 7.5, frame 7 (cyan line) is selected as a *key* frame ($\hat{\mathcal{M}}_7$) because the *orange* trajectory bundle ends at frame 6. From frames 7 to 9 the only object label is *green*. At frame 10 (purple line) the *red* trajectory bundle starts, therefore another *key* frame ($\hat{\mathcal{M}}_{10}$) is required here.

The *key* frames ($\hat{\mathcal{M}}_f$) in **STEP A** (fig. 7.5) are for the *forward* propagation step. When doing the *backward* propagation in **STEP B** the *key* frames here are shifted to the left by one frame, as shown in the *second row* of fig. 7.5. Note that the *key* frames for the *backward* propagation (**STEP B**) correspond to *intermediate* frames in the *forward* propagation step. For these *intermediate* frames we have at this stage estimated object label fields $\hat{\mathcal{L}}_f$ from **STEP A**. For example in fig. 7.5 the object labels $\hat{\mathcal{L}}_6$ at frame 6 obtained from **STEP A** is used as the *key* frame $\hat{\mathcal{M}}_6$ in **STEP B**. The *cyan* arrow indicates this association. The *key* frames $\hat{\mathcal{M}}_9$,$\hat{\mathcal{M}}_{F-4}$, and $\hat{\mathcal{M}}_{F-1}$ for **STEP B** are selected in a similar way.

The *key* frames in **STEP A** are at the top of the estimation chain, since the *key* frames in **STEP B** are derived from them. Hence we must obtain initial approximations to the *key* frames in **STEP A** only. These estimated *key* frames for **STEP A** (fig. 7.5) are generated using motion information and the geodesic distances [10,111] of the pixel sites to the spatial locations of trajectories (discussed later). As a result of using motion information we cannot generate *key* frames for the first (1) and last (F) frames in a sequence. For these frames there are no previous and next frames respectively which are required to do motion estimation. Hence the first *key* frames in the *forward* (**STEP A**) and *backward* (**STEP B**) propagation steps are at frame 2 and $F-1$ respectively. See the *top two rows* of fig. 7.5.

7.1.3 Refining Key Frames for Automatic Segmentation

As mentioned in the previous section, the initial *key* frames ($\hat{\mathcal{M}}_f$) for the *forward* propagation (**STEP A**) illustrated in the *top row* of fig. 7.5 are estimated naively using only motion information and geodesic distances (details are given later). Here the estimation of these initial *key* frames does not take into account the 'average' appearance of the objects throughout the sequence. Note here that the *key* frames for **STEP A** are *intermediate* frames in **STEP B**. Hence we can obtain new *key* frames for the *forward* propagation step from the corresponding labelled frames $\hat{\mathcal{L}}_f$ in **STEP B**.

The new *forward* propagation step that uses the estimated *key* frames from **STEP B** is labelled '**STEP C**' in *third row* of fig. 7.5. The coloured *arrows* indicate the *intermediate* frames $\hat{\mathcal{L}}_f$ in **STEP B** that have been used as *key* frames in **STEP C**. Note here that the *key*

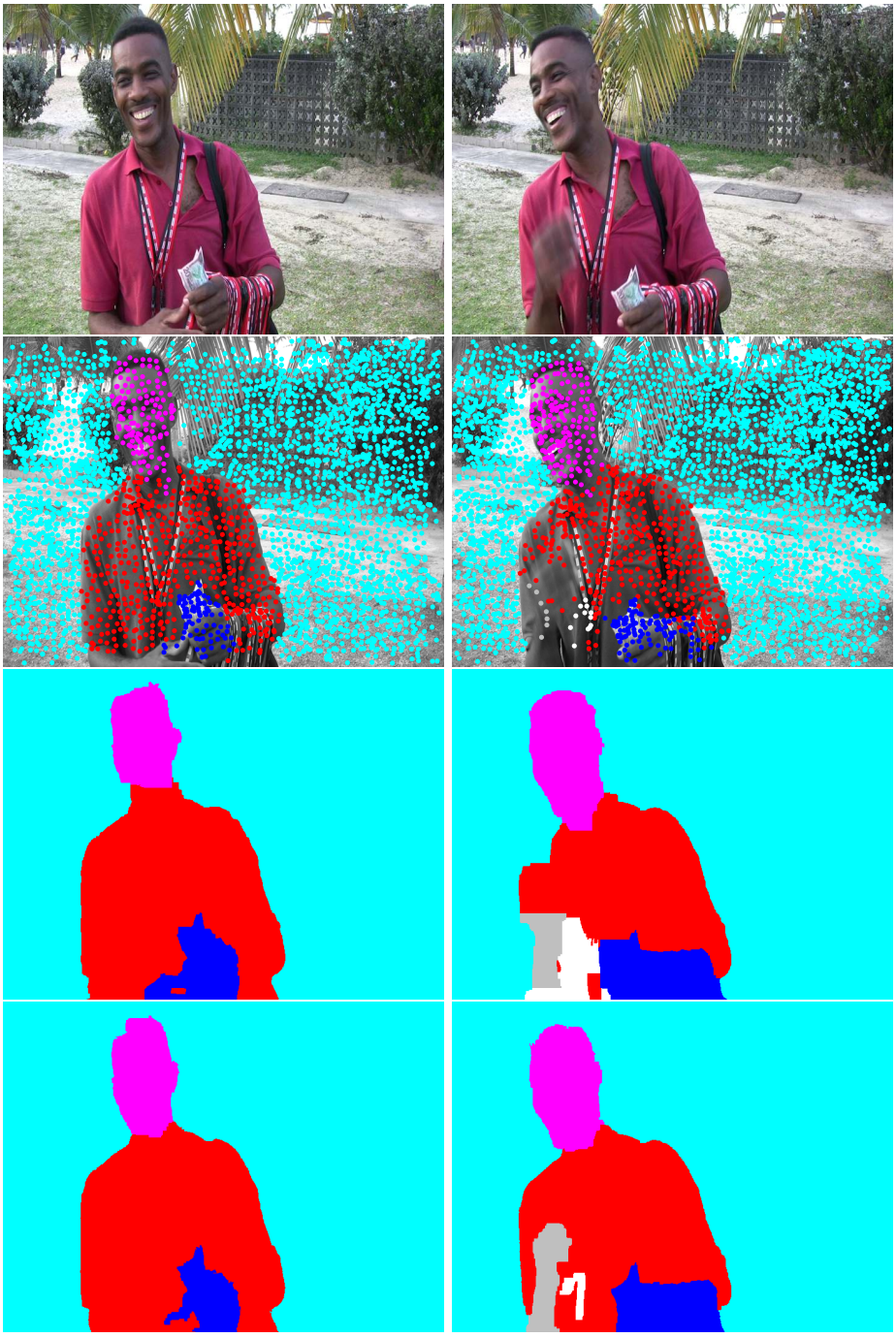

Figure 7.6: ***Key*** *frames for the automatic dense segmentation process.* ***Top row****: Frames 2 (left) and 13 (right) in the* ***Triniman*** *sequence.* ***Second row****: Our sparse trajectory segmentation for the frame in the top row. Here the points for the trajectories in the bundles are shown as coloured* ***dots****.* ***Third row****: The* ***key*** *frames generated from motion information and geodesic distances that are used in* ***STEP A*** *(See fig. 7.5).* ***Bottom row****: The refined* ***key*** *frames that are used in the final* ***forward*** *propagation in* ***STEP C****. The trajectory bundles in the second row and their corresponding image regions in the* ***key*** *frames are coloured in a similar manner.*

frames for both *forward* propagation steps **STEP A** and **STEP C** occur at the same frame positions.

The *third and bottom rows* in fig. 7.6 show examples of the *key* frames in **STEP A** and **STEP C** respectively for the *Triniman* sequence. The *key* frames at frames 2 and 13 are shown in the left and right columns respectively. It may be observed that the *key* frames for **STEP C** (*bottom row*) have better defined object boundaries than those for **STEP A** (*third row*).

Recall that the initial *key* frames for **STEP A** are generated using motion information and geodesic distances. The *second row* of fig. 7.6 shows the sparse trajectory segmentation for frames 2 and 13 from left to right respectively. The spatial locations of the trajectories directly influence the *key* frames generated for **STEP A**. It may be observed that the object boundaries are not well defined in these *key* frames. Hence it is reasonable to replace these 'rough' *key* frames in the final *forward* propagation done in **STEP C**. The *key* frames in **STEP C** have been influenced by the 'average' appearance of the objects during the *backward* propagation in **STEP B**. Therefore the object boundaries in these *key* frames for **STEP C** mimic the natural boundaries between the objects in the image. See the images for frames 2 (left) and 13 (right) in the *top row* of fig. 7.6.

The final *backward* propagation step that follows **STEP C** is labelled '**STEP D**' in the *bottom row* of fig. 7.6. Recall that the aim of the *forward* and *background* propagation steps in the segmentation framework is to obtain the appearance models $\mathcal{C}_{f-1,f}$ (forward) and $\mathcal{C}_{f+1,f}$ (backward) for doing the final segmentation. These appearance models are obtained from the final *forward* and *backward* propagation steps **STEP C** and **STEP D** respectively.

7.1.4 Key Frame Estimation via Geodesic Distances

The *geodesic distance* [111] between pixel sites s_1 and s_2 is defined as the minimum cost $g(s_1, s_2)$ between these sites given below.

$$g(s_1, s_2) = \min_{\mathbf{\Gamma} \in \mathcal{P}_{s_1,s_2}} \int_0^1 \sqrt{\|\mathbf{\Gamma}'(k)\|^2 + \gamma(\nabla \mathbf{I}(k) \cdot \mathbf{u})^2}\, \mathrm{d}k, \tag{7.4}$$

$$\text{for } \mathbf{\Gamma}'(k) = \frac{\partial \mathbf{\Gamma}(k)}{\partial k}, \quad \mathbf{u} = \frac{\mathbf{\Gamma}'(k)}{\|\mathbf{\Gamma}'(k)\|}$$

Given $\mathcal{P}_{s_1,s_2}$ is the set of all paths between the points s_1 and s_2, where $\mathbf{\Gamma}(k)$ is one such path parametrized by $k \in [0, 1]$. The weight γ controls the contributions of the grayscale image gradients $\nabla \mathbf{I}$ and the spatial distances $\mathbf{\Gamma}'$.

The idea behind using geodesic distances [111] to estimate the initial *key* frames for **STEP A** (fig. 7.5) is that we want to assess the 'closeness' of the pixel sites to the trajectories in some way that considers the topology of the objects in an image. By topology we mean the strength of the edge features and their spatial locations relative to the points of the trajectories.

Fig. 7.7 will be used to discuss how the geodesic distance $g(\mathbf{s}_1, \mathbf{s}_2)$ varies depending on the relative locations of the pixel sites $\mathbf{s}_1$ and $\mathbf{s}_2$ in an image. Here fig. 7.7 shows a cartoonized

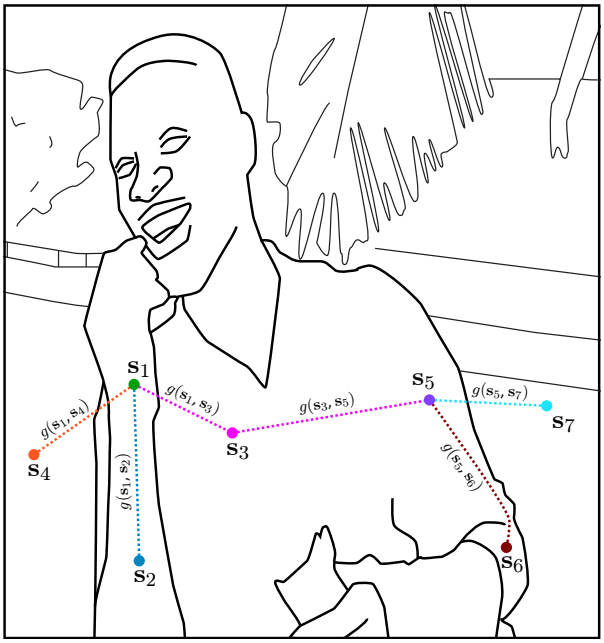

Figure 7.7: *A cartoonized version of frame 19 in the* ***Triniman*** *sequence. The edges in this frame are represented by bold* ***black lines****. Seven pixel sites* $\mathbf{s}_1, \ldots, \mathbf{s}_7$ *are shown as coloured dots, and the paths between them are broken coloured lines. The paths are labelled with the geodesic distances* $g(.,.)$*.*

version of frame 19 in the *Triniman* sequence. The edges in this frame are represented by bold *black lines*. Seven pixel sites $\mathbf{s}_1, \ldots, \mathbf{s}_7$ are shown as coloured 'dots' in the illustration, and the paths between them are broken coloured lines.

The geodesic distance $g(\mathbf{s}_1, \mathbf{s}_2)$ between sites $\mathbf{s}_1$ (green) and $\mathbf{s}_2$ (blue) is less than the geodesic distance $g(\mathbf{s}_1, \mathbf{s}_4)$ between sites $\mathbf{s}_1$ and $\mathbf{s}_4$ (orange) (fig. 7.7). This is because there are no edges between $\mathbf{s}_1$ and $\mathbf{s}_2$, whilst there is an edge between $\mathbf{s}_1$ and $\mathbf{s}_4$. Note that the Euclidean distance between $\mathbf{s}_1$ and $\mathbf{s}_4$ is less than that of $\mathbf{s}_1$ and $\mathbf{s}_2$, however $g(\mathbf{s}_1, \mathbf{s}_2) < g(\mathbf{s}_1, \mathbf{s}_4)$. Also shown in fig. 7.7 are more examples of geodesic distance inequalities: $g(\mathbf{s}_1, \mathbf{s}_2) < g(\mathbf{s}_1, \mathbf{s}_3)$, $g(\mathbf{s}_3, \mathbf{s}_5) < g(\mathbf{s}_5, \mathbf{s}_7)$ and $g(\mathbf{s}_5, \mathbf{s}_6) < g(\mathbf{s}_5, \mathbf{s}_7)$.

The geodesic distance of the pixel site s at frame f to object o is defined as $G_f^o(\mathbf{s})$ given below.

$$G_f^o(\mathbf{s}) = \min_{\mathbf{s}_f^t}(g(\mathbf{s}, \mathbf{s}_f^t)) \tag{7.5}$$

Where $\mathbf{s}_f^t = (x_f^t, y_f^t)$ are points along the trajectories at frame f that are in a trajectory bundle corresponding to object o.

Fig. 7.8 shows examples of the geodesic distance fields G_f^o for four objects at frame 2 in the *Triniman* sequence. The geodesic distance fields G_f^o are shown in the *middle and bottom rows* of fig. 7.8, where the points of the trajectories in the respective bundles are shown as yellow

dots. It can be observed that the geodesic distances change rapidly when the edges in the image (*top left* of fig. 7.8) are crossed. Hence geodesic distances provide a good approximation of the image regions associated with each object given that they are separated by distinct edges.

7.1.4.1 Estimating Key Frames for Automatic Segmentation

For the automatic dense segmentation we have to estimate initial *key* frames for the first *forward* propagation step as mentioned previously. Recall that these *key* frames ($\hat{\mathcal{M}}_f$) are generated using motion information and geodesic distances. The posterior distribution for these estimated *key* frame $\hat{\mathcal{M}}_f$ is given as follows.

$$p(\hat{\mathcal{M}}_f(\mathbf{s})|\mathcal{X}, \mathbf{I}, \hat{\mathcal{M}}_f(\sim \mathbf{s})) \propto p_x(f, \mathbf{s}|\hat{\mathcal{M}}_f(\mathbf{s}), \mathcal{X}) p_g(\mathbf{I}_f(\mathbf{s})|\hat{\mathcal{M}}_f(\mathbf{s})) p_s(\hat{\mathcal{M}}_f(\mathbf{s})|\hat{\mathcal{M}}_f(\sim \mathbf{s})) \tag{7.6}$$

Where $p_x(f, \mathbf{s}|.)$ is the likelihood of the pixel site s at frame f following the motion model defined by a particular object label $\hat{\mathcal{M}}_f(\mathbf{s})$. The geodesic distance likelihood $p_g(\mathbf{I}_f(\mathbf{s})|.)$ compares the geodesic distance of pixel site s to the trajectories locations in the bundle corresponding to object label $\hat{\mathcal{M}}_f(\mathbf{s})$. Label smoothness is injected through the MRF prior $p_s(.)$.

The motion likelihood distribution $p_x(f, \mathbf{s}|\hat{\mathcal{M}}_\mathbf{f}(\mathbf{s}))$ used here is the same as the motion likelihood $p_x(f, \mathbf{s}|\mathcal{L}_\mathbf{f}(\mathbf{s}))$ for the general framework previously defined in eq. 7.1. Note that in this case we are only using a different notation for the object label field $\hat{\mathcal{M}}_f(\mathbf{s})$ instead of $\mathcal{L}_f(\mathbf{s})$. Also the prior distribution $p_s(.)$ is the same as the one for the general framework.

7.1.4.2 Geodesic Distance Likelihood

We previously defined the geodesic distance of a pixel site s to object o as $G_f^o(\mathbf{s})$ (equ. 7.5). See fig. 7.8 for examples of the geodesic distance fields G_f^o for four objects in frame 2 of the *Triniman* sequence.

The geodesic likelihood $p_g(\mathbf{I}_f(\mathbf{s})|\hat{\mathcal{M}}_f(\mathbf{s}))$ for pixel site s with respect to the object label $\hat{\mathcal{M}}_f(\mathbf{s}) = \mathbf{o}$ is a Gaussian distribution of the geodesic distance $G_f^o(\mathbf{s})$. This distribution is given below.

$$\boxed{p_g(\mathbf{I}_f(\mathbf{s})|\hat{\mathcal{M}}_f(\mathbf{s})) = \frac{1}{\sigma_f(\mathbf{s})\sqrt{2\pi}} \exp\left(-0.5\left[\frac{G_f^o(\mathbf{s}) - \bar{G}_f(\mathbf{s})}{\sigma_f(\mathbf{s})}\right]^2\right)} \tag{7.7}$$

Where $\bar{G}_f(\mathbf{s})$ and $\sigma_f(\mathbf{s})$ are the mean and standard deviation of the distribution respectively. These parameters $\bar{G}_f(\mathbf{s})$ and $\sigma_f(\mathbf{s})$ are defined as follows.

$$\bar{G}_f(\mathbf{s}) = \min_o \left(G_f^o(\mathbf{s})\right), \ [\sigma_f(\mathbf{s})]^2 = \frac{1}{9}\sum_{\mathbf{p}\in\mathcal{N}_\mathbf{s}} \left[\bar{G}_f(\mathbf{p})\right]^2 \tag{7.8}$$

Where $\mathcal{N}_\mathbf{s}$ is the 3×3 neighbourhood of pixel site $\mathbf{s}$ including $\mathbf{s}$.

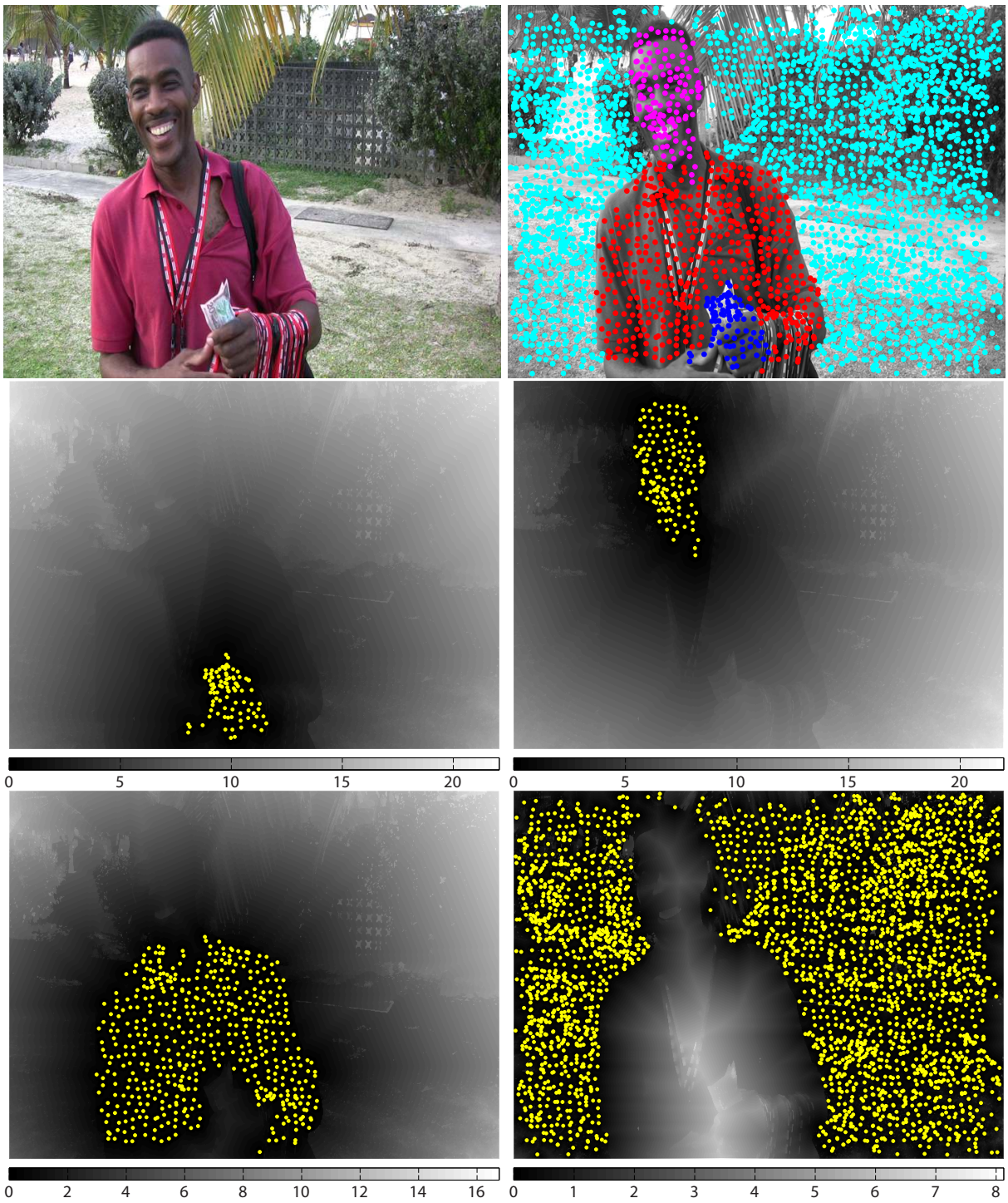

Figure 7.8: *The geodesic distances to the objects in frame 2 of the* ***Triniman*** *sequence.* ***Top row****: Frame 2 (left) in the* ***Triniman*** *sequence, and our sparse trajectory segmentation (right) for this frame. The points for the trajectories shown as coloured dots. The four trajectory bundles are coloured* ***cyan*** *(background),* ***pink*** *(face),* ***red*** *(body) and* ***blue*** *(hand) respectively.* ***Clockwise from middle left****: The geodesic distance to the hand, face, background and body objects respectively. The respective points of the trajectories are superimposed on the geodesic distance fields as* ***yellow*** *dots. The values for the geodesic distances are represented in gray scale, where* ***black*** *and* ***white*** *are minimum and maximum values respectively.*

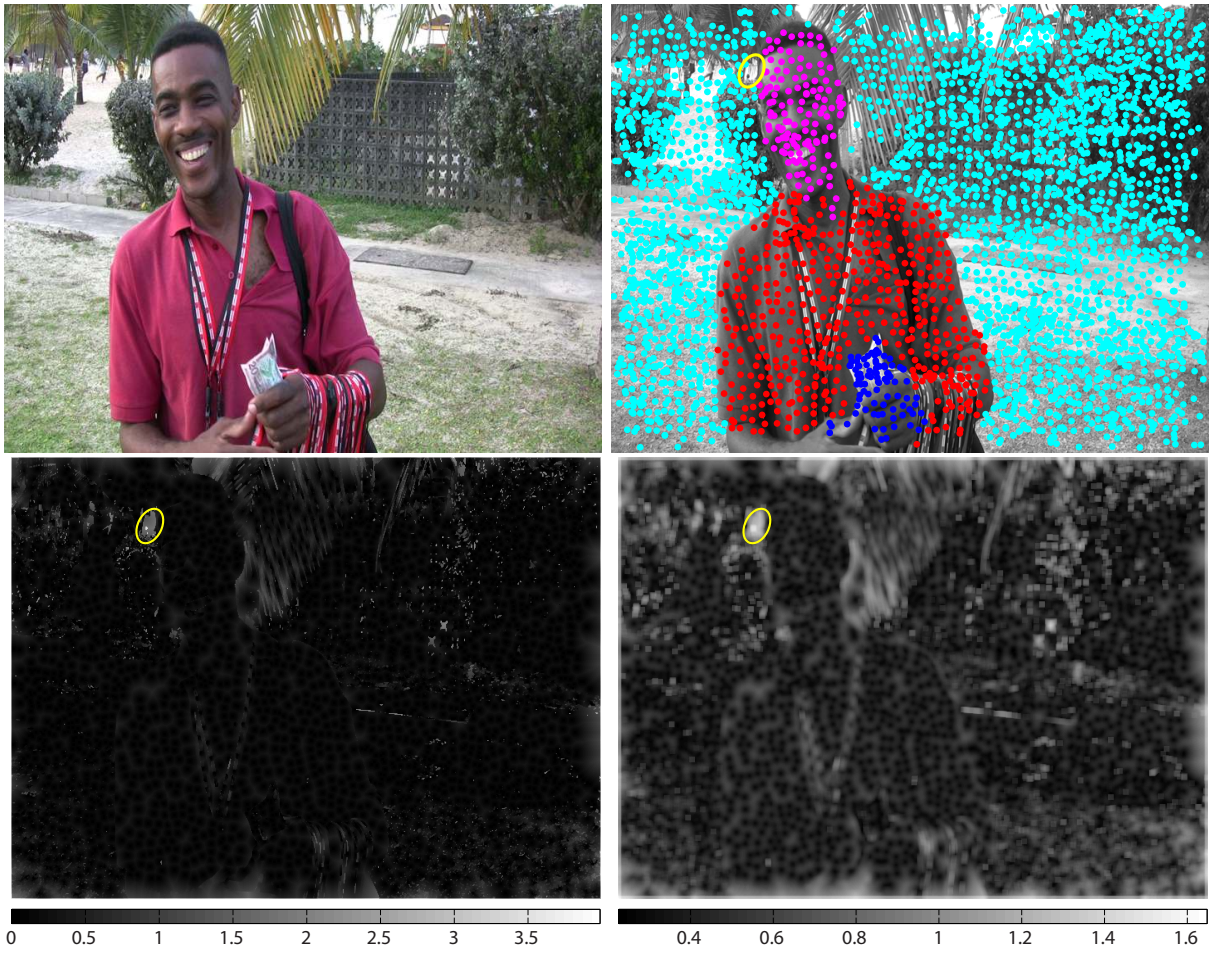

Figure 7.9: *The parameters for the geodesic distance likelihood at frame 2 in the* ***Triniman*** *sequence.* ***Top row****: Frame 2 (left) in the* ***Triniman*** *sequence, and our sparse segmentation (right) for this frame. The points for the trajectories are shown as coloured dots. The* ***yellow*** *oval highlights an image region bounded by strong edges. There are no trajectory points inside this region.* ***Bottom row****: The mean $\bar{G}_f$ (left) and standard deviations σ_f (right) fields for the geodesic distance likelihood at frame 2. The values for these parameter fields are represented in gray scale, where* ***black*** *and* ***white*** *are minimum and maximum values respectively.*

The examples of the mean $\bar{G}_f$ (left) and standard deviation σ_f (right) fields for frame 2 in the *Triniman* sequence shown in the *bottom row* of fig. 7.9 will be used to explain the choice of the parameters in eq. 7.7. In these parameter fields the numeric values are represented on a gray scale, where *black* and *white* are the minimum and maximum values respectively.

The *top row* of the fig. 7.9 shows frame 2 (left) in the *Triniman* sequence and our sparse trajectory segmentation (right) for this frame. Now the *yellow* oval superimposed on the sparse segmentation (top right) highlights an image region bounded by strong edges. Note also that there are no trajectory points inside this image region. Hence the minimum geodesic distances

$\min_o \left(G_f^o(\mathbf{s})\right)$ of the pixels in this region are relatively large (i.e. $\min_o \left(G_f^o(\mathbf{s})\right) >> 0$). This region is highlighted in the mean $\bar{G}_f = \min_o \left(G_f^o(\mathbf{s})\right)$ field (bottom left) also using a *yellow* oval. It may be observed that the minimum geodesic distances for the pixels in this region are larger than the majority of the pixels in the field. Therefore we vary the parameters $\bar{G}_f(\mathbf{s})$ and $\sigma_f(\mathbf{s})$ of the geodesic likelihood $p_g(\mathbf{I}_f(\mathbf{s})|.)$ according to the minimum geodesic distance at site $\mathbf{s}$.

It is reasonable to center the distribution $p_g(\mathbf{I}_f(\mathbf{s})|.)$ on the minimum geodesic $\min_o \left(G_f^o(\mathbf{s})\right)$ so that we are guaranteed that the object with smallest geodesic distance to site $\mathbf{s}$ will have the largest likelihood. We have a low confidence in labelling site $\mathbf{s}$ when the minimum geodesic distance at this site is relatively large. This indicates that there are not many trajectory points close to site $\mathbf{s}$. Hence we designed the standard deviation $\sigma_f(\mathbf{s})$ at site $\mathbf{s}$ to be proportional to the square of the minimum geodesic distance in the 3×3 neighbourhood of site $\mathbf{s}$ (See eq. 7.8). The *yellow* oval in *bottom right* of fig. 7.9 highlights the standard deviations $\sigma_f(\mathbf{s})$ for the pixels in the mean field (bottom left) that are also outlined with the *yellow* oval. Here the standard deviations are large (white) for the pixels with large minimum geodesic distances.

7.1.5 Motion Likelihood

The motion likelihood $p_x(\mathcal{X}(f, \mathbf{s})|.)$ measures how well the motion models corresponding to the object labels $\mathcal{L}_f(\mathbf{s})$ describe the motion of pixel site $\mathbf{s}$ at frame f. This measurement is based on displaced frame differences (DFDs) with respect to the current frame f. The DFD at pixel site $\mathbf{s}$ in frame f is simply a difference between that pixel and its motion compensated counterpart in some other frame. The idea is that a low DFD given a particular motion model is indicative of the presence of an object moving with that motion. In our work we measure DFDs (with respect to a single motion model) over $\pm k$ frames from the current frame. The *top row* of fig. 7.10 shows the frames $\mathbf{I}_{f-k}, \ldots, \mathbf{I}_{f-1}$ and $\mathbf{I}_{f+1}, \ldots, \mathbf{I}_{f+k}$ centered around the current frames $\mathbf{I}_f$ that are used to estimate the DFDs for frame f. These frames $\mathbf{I}_{f-k}, \ldots, \mathbf{I}_{f-1}$ and $\mathbf{I}_{f+1}, \ldots, \mathbf{I}_{f+k}$ are defined as the *window* frames for the current frame $\mathbf{I}_f$.

The problem is that an object will contain several different motion bundles. Hence to associate a single DFD measurement at a site with a particular object means combining the DFD information from all the motion (trajectory) bundles in that object in some way. We will describe later exactly how DFDs over several frames are generated. For now we assume that for an object label $\mathcal{L}_f(\mathbf{s}) = o$, we have several measurements of DFD based on each motion (trajectory) bundle 'n' inside object o. This DFD is defined as $\Lambda_f^n(\mathbf{s})$. To combine these DFDs to select one measurement representation of the object, we simply take the minimum and hence,

$$\delta_f^o(\mathbf{s}) = \min_{n \in \mathcal{B}_o} \Lambda_f^n(\mathbf{s}) \tag{7.9}$$

Where $\delta_f^o(\mathbf{s})$ is our new combined measurement, and $\mathcal{B}_o$ is the set of trajectory (motion) bundles associated with object o.

Hence we can define our motion likelihood $p_x(\mathcal{X}(f, \mathbf{s})|.)$ for site $(f, \mathbf{s})$ with respect to object

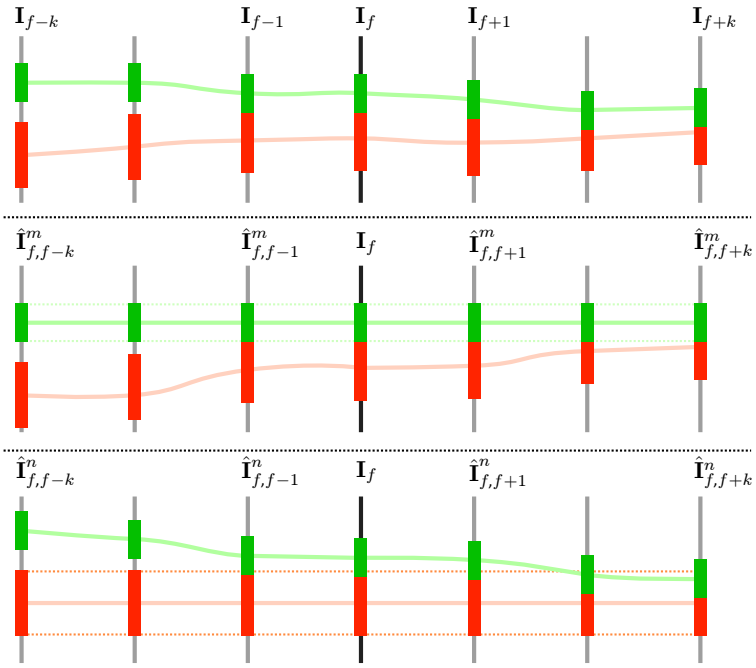

Figure 7.10: *A sequence with a **red** and a **green** object that are represented by similarly colour trajectory bundles. The trajectory bundles are illustrated as the coloured lines that extend across the frames. **Top row**: The **window** frames $\mathbf{I}_{f,j}$ are used to estimate the DFD for the current frame $\mathbf{I}_f$, where $j = \{f-k, \ldots, f-1, f+1, \ldots, f+k\}$. **Middle and bottom row**: The motion compensated **window** frames $\hat{\mathbf{I}}^m_{f,j}$ and $\hat{\mathbf{I}}^n_{f,j}$ obtained from compensating with respect to the motion of the **green** and **red** trajectory bundles respectively.*

o as a Gaussian distribution of the DFD $\delta^o_f(\mathbf{s})$. This distribution is given below.

$$p_x(\mathcal{X}(f,\mathbf{s})|\mathcal{L}_f(\mathbf{s})) = \frac{1}{\sigma^\delta_f(\mathbf{s})\sqrt{2\pi}} \exp\left(-0.5\left[\frac{\delta^o_f(\mathbf{s}) - \bar{\delta}_f(\mathbf{s})}{\sigma^\delta_f(\mathbf{s})}\right]^2\right) \tag{7.10}$$

Where $\bar{\delta}_f(\mathbf{s})$ and $\sigma^\delta_f(\mathbf{s})$ are the mean and standard deviation of the distribution respectively. These parameters $\bar{\delta}_f(\mathbf{s})$ and $\sigma^\delta_f(\mathbf{s})$ are defined as follows.

$$\bar{\delta}_f(\mathbf{s}) = \min_o \left(\delta^o_f(\mathbf{s})\right), \quad \left[\sigma^\delta_f(\mathbf{s})\right]^2 = \frac{1}{9}\sum_{\mathbf{p}\in\mathcal{N}_\mathbf{s}} \left[\bar{\delta}_f(\mathbf{p})\right]^2 \tag{7.11}$$

Where $\mathcal{N}_\mathbf{s}$ is the 3×3 neighbourhood of pixel site $\mathbf{s}$ including $\mathbf{s}$.

Note that the parameters $\bar{\delta}_f(\mathbf{s})$ and $\sigma^\delta_f(\mathbf{s})$ in eq. 7.11 are defined in a similar manner to the parameters for the geodesic distance likelihood $\bar{G}_f(\mathbf{s})$ and $\sigma_f(\mathbf{s})$ in section 7.1.4.2. The reasoning here is the same. We center the distribution $p_x(\mathcal{X}(f,\mathbf{s})|.)$ on the minimum DFD $\min_o \left(\delta^o_f(\mathbf{s})\right)$ over all the objects. Also the standard deviation $\sigma^\delta_f(\mathbf{s})$ is proportional the square of the minimum DFD for the sites in the 3×3 neighbourhood of site $\mathbf{s}$.

7.1.6 DFDs for the Trajectory Bundles

Recall from chapter 4 that the motion models for the nth trajectory bundle is a set of 2D affine transformations. These transformations represent the relationship between the spatial locations of the trajectories across adjacent frame pairs. That is, given the spatial locations (x_f^t, y_f^t) of the trajectories at frame f, a linear transformation $\mathbf{A}_{f,f+1}^n$ gives the locations (x_{f+1}^t, y_{f+1}^t) in the next frame $f+1$.

In order to estimate the DFD $\Lambda_f^n(\mathbf{s})$ for bundle n we must motion compensate the *window* frames using the motion models for this bundle. Given the motion models $\mathbf{A}_{f,f+1}^n$ for bundle n, the motion compensated version of a *window* frame $\mathbf{I}_j$ with respect to the current frame f is defined as $\hat{\mathbf{I}}_{f,j}^n$. Here $j = \{f-k, \ldots, f-1, f+1, \ldots, f+k\}$ is the index of the *window* frames. The motion compensated pixel $\hat{\mathbf{I}}_{f,j}^n(\mathbf{s})$ is calculated from $\mathbf{A}_{f,f+1}^n$ as shown below.

$$\begin{aligned} \hat{\mathbf{I}}_{f,j}^n(\mathbf{s}) &= \mathbf{I}_j(\mathbf{s}_c), \\ \text{for } \mathbf{s}_c &= \mathbf{A}_{f,j}^n \begin{pmatrix} \mathbf{s} \\ 1 \end{pmatrix} \end{aligned} \tag{7.12}$$

Where $\mathbf{A}_{f,j}^n$ is the transformation between the current frame f and the *window* frame j. Note that the pixel site $\mathbf{s} = (x, y)^T$ in eq. 7.12 above.

The transformation $\mathbf{A}_{f,j}^n$ is given as follows where $f \neq j$.

$$\mathbf{A}_{f,j}^n = \begin{cases} \left(\prod_{h=j}^{f-1} \mathbf{A}_{h,h+1}^n \right)^{-1}, & j < f \\ \left(\prod_{h=f}^{j-1} \mathbf{A}_{h,h+1}^n \right)^{-1}, & j > f \end{cases} \tag{7.13}$$

The *middle and bottom* row of fig. 7.10 show an illustration of the motion compensated *window* frames $\hat{\mathbf{I}}_{f,j}^m$ and $\hat{\mathbf{I}}_{f,j}^n$ for the mth and nth trajectory bundles respectively. The nth trajectory bundle is represented by a *red* line that extends across the frames. This bundle represents the object illustrated with *red* bars. Note that the non-motion compensated frames $\mathbf{I}_j$ are shown in the *top* row of fig. 7.10.

It may be observed that the *red* object is aligned across the motion compensated frames $\hat{\mathbf{I}}_{f,j}^n$ as shown in the *third* row of fig. 7.10. Here the *red* object exists in the current frame at the pixel sites $(f, \mathbf{s}) = (f, (x, y))$, and these sites correspond to the pixel sites $(j, \mathbf{s}) = (j, (x, y))$ in the jth *window* frame. Similarly the *middle* row of fig. 7.10 shows the motion compensated *window* frames $\hat{\mathbf{I}}_{f,j}^m$ for the mth trajectory bundle. This trajectory bundle is represented by a *green* line, and the object associated with this bundle is a *green* bar.

Examples of motion compensated *window* frames $\hat{\mathbf{I}}_{f,j}^n$ for the *Triniman* sequence are shown in fig. 7.11. The *top* row shows the current frame $\mathbf{I}_{20}$ with square spatial markers coloured in *yellow* (background), *cyan* (face), *green* (teeth) and *red* (body) depending on their locations

Figure 7.11: *Motion compensated frames in the* ***Triniman*** *sequence.* ***Top row****: The current frame* $\mathbf{I}_{20}$ *with square spatial markers coloured in yellow (background), cyan (face), green (teeth) and red (body) depending on the features they correspond to in the image.* ***Middle row****: The motion compensated frames* $\hat{\mathbf{I}}^{n}_{20,17}$ *(left) and* $\hat{\mathbf{I}}^{n}_{20,23}$ *(right), where these frames are compensated with respect to the background trajectory bundle.* ***Bottom row****: The motion compensated frames* $\hat{\mathbf{I}}^{n}_{20,17}$ *(left) and* $\hat{\mathbf{I}}^{n}_{20,23}$ *(right), where these frames are compensated with respect to the trajectory bundle corresponding to the face of the actor.*

in the image. The *middle* row shows the motion compensated frames $\hat{\mathbf{I}}^n_{20,17}$ (left) and $\hat{\mathbf{I}}^n_{20,23}$ (right), where these frames are compensated with respect to the background trajectory bundle. It may be observed that the corresponding image features inside the *yellow* (background) spatial markers are roughly the same in both motion compensated frames and the current frame $\mathbf{I}_{20}$ (top row). That is, the background features are aligned across the *window* frames and the current frame. Similarly, the *bottom* row of fig. 7.11 shows the motion compensated frames $\hat{\mathbf{I}}^n_{20,17}$ (left) and $\hat{\mathbf{I}}^n_{20,23}$ (right), where these frames are compensated with respect to the trajectory bundle corresponding to the face of the actor. The features in the *cyan* and *green* spatial markers are aligned in both these motion compensated frames and the current frame.

We now have $2k$ motion compensated *window* frames $\hat{\mathbf{I}}^n_{f,j}$ available to estimate the DFD Λ^n_f for current frame f. The displaced frame differences between these $2k$ *window* frames $\hat{\mathbf{I}}^n_{f,j}$ and the current frame $\mathbf{I}_f$ produce $2k$ DFDs $\Delta^n_{f,j}$. The DFD $\Delta^n_{f,j}(\mathbf{s})$ between the site $\mathbf{s}$ at the current frame f and the motion compensated site in the j *window* frame is given below.

$$\Delta^n_{f,j}(\mathbf{s}) = |I_f(\mathbf{s}) - \hat{I}^n_{f,j}(\mathbf{s})| \tag{7.14}$$

Where I_f and $\hat{I}^n_{f,j}$ are the grayscale versions of $\mathbf{I}_f$ and $\hat{\mathbf{I}}^n_{f,j}$ respectively.

The task at this stage is to combine these $2k$ DFDs $\Delta^n_{f,j}(\mathbf{s})$ into a single DFD $\Lambda^n_f(\mathbf{s})$ for the nth trajectory bundle. The next section discusses how this is done.

7.1.6.1 Combining Window Frame DFDs

The DFD $\Lambda^n_f(\mathbf{s})$ at frame f for the nth trajectory bundle is a weighted sum of the $2k$ DFDs $\Delta^n_{f,j}(\mathbf{s})$ obtained from the *window* frames. This DFD $\Lambda^n_f(\mathbf{s})$ is given below.

$$\Lambda^n_f(\mathbf{s}) = \frac{\sum_{j=f-k}^{f+k} \left[w^n_{f,j}(\mathbf{s})\Delta^n_{f,j}(\mathbf{s})\right]}{\sum_{j=f-k}^{f+k} \left[w^n_{f,j}(\mathbf{s})\right]},\ j \neq f \tag{7.15}$$

Where the weight for the DFD $\Delta^n_{f,j}(\mathbf{s})$ is $w^n_{f,j} \in \{0, 1\}$.

The weight $w^n_{f,j}$ for the DFD $\Delta^n_{f,j}(\mathbf{s})$ is zero if pixel site $\mathbf{s}$ is deemed to be occluded at *window* frame j. The weight $w^n_{f,j} = 1$ if no occlusion occurs at $(j, \mathbf{s})$. Here we assume that the DFD $\Delta^n_{f,j}(\mathbf{s})$ is 'high' when an occlusion occurs at the site $(j, \mathbf{s})$. We therefore use an auxiliary optimization process to label the DFDs $\Delta^n_{f,j}(\mathbf{s})$ at site $(j, \mathbf{s})$ as being generated from occlusion ('high' DFD) or not ('low' DFD). We define the occlusion label for the DFD $\Delta^n_{f,j}(\mathbf{s})$ as $\mathcal{D}^n_{f,j}(\mathbf{s})$, where $\mathcal{D}^n_{f,j}(\mathbf{s}) = 0$ and $\mathcal{D}^n_{f,j}(\mathbf{s}) = 1$ signify that $\Delta^n_{f,j}(\mathbf{s})$ has a 'low' and a 'high' DFD value respectively.

The *bottom* row of fig. 7.12 shows the DFDs $\Delta^n_{f,f-k}, \ldots, \Delta^n_{f,f-1}$ and $\Delta^n_{f,f+1}, \ldots, \Delta^n_{f,f+k}$ on the left and right of the current frame $\mathbf{I}_f$ respectively. For each DFD $\Delta^n_{f,j}$, low and high DFD values are coloured *cyan* and *brown* respectively. The pixel sites in the *window* frames

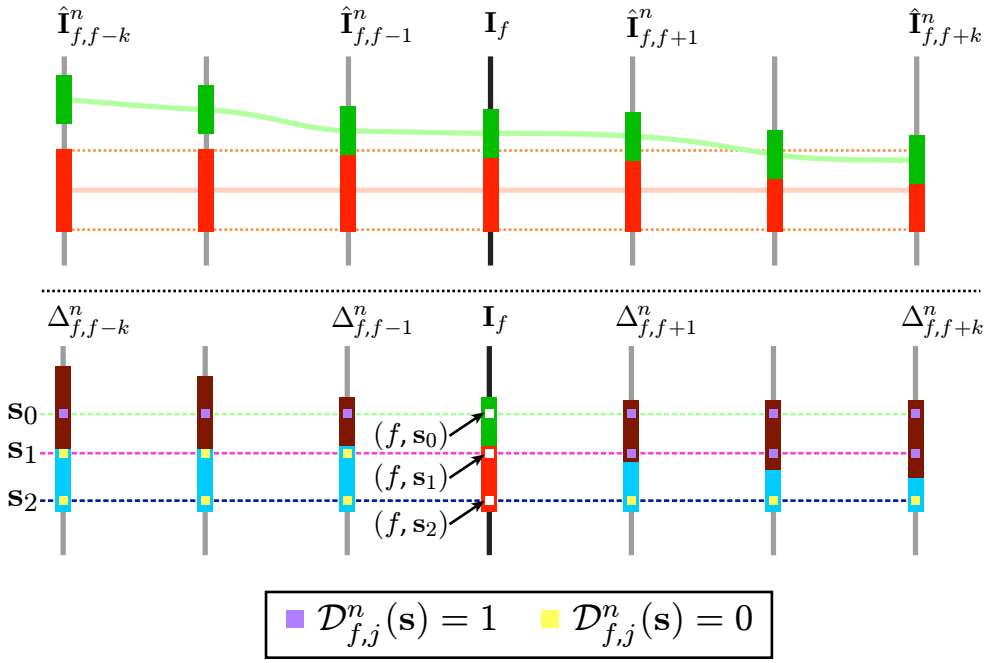

Figure 7.12: *A sequence with a **red** and a **green** object that are represented by similarly colour trajectory bundles. The trajectory bundles are illustrated as the coloured lines that extend across the frames. **Top row**: The motion compensated **window** frames $\hat{\mathbf{I}}^n_{f,j}$ obtained from compensating with respect to the **red** trajectory bundles, where $j = \{f-k, \ldots, f-1, f+1, \ldots, f+k\}$. The corresponding non-motion compensated frames $\mathbf{I}_j$ are shown in **top row** of fig. 7.10. **Bottom row**: The **window** DFDs $\Delta^n_{f,j}$ for the motion compensated frames $\hat{\mathbf{I}}^n_{f,j}$ in the top row. Here 'high' and 'low' DFD values are coloured in **brown** and **cyan** respectively. The pixel site $\mathbf{s} = (f, \mathbf{s}_0), (f, \mathbf{s}_1)$ and $(f, \mathbf{s}_2)$ are **white** squares in the current frame $\mathbf{I}_f$, and their corresponding sites in the **window** frames are indicated with broken coloured lines. The occlusion labels for the DFDs $\mathcal{D}^n_{f,j} = 0$ and $\mathcal{D}^n_{f,j} = 1$ are indicated with **yellow** and **purple** squares respectively.*

corresponding to the sites $(f, \mathbf{s}_0)$, $(f, \mathbf{s}_1)$ and $(f, \mathbf{s}_2)$ at the current frame $\mathbf{I}_f$ are indicated with the *light green*, *pink* and *blue* dotted lines respectively. These corresponding sites $(j, \mathbf{s}_0)$, $(j, \mathbf{s}_1)$ and $(j, \mathbf{s}_2)$ in the *window* frames have DFD labels $\mathcal{D}^n_{f,j}(\mathbf{s}) = 0$ and $\mathcal{D}^n_{f,j}(\mathbf{s}) = 1$ that are indicated with *yellow* and *purple* squares respectively.

For the illustration in the *bottom* row of fig. 7.12 we would like to encourage the pixel site $(f, \mathbf{s}_1)$ and $(f, \mathbf{s}_2)$ to take the label (*red*) for bundle n. These sites belong to the *red* object that bundle n represents at the current frame $\mathbf{I}_f$. Therefore we require the DFDs $\Lambda^n_f(\mathbf{s}_1)$ and $\Lambda^n_f(\mathbf{s}_2)$ to be as 'low' as possible. So in this case we give the 'low' $(\mathcal{D}^n_{f,j}(\mathbf{s}) = 0)$ and 'high' $(\mathcal{D}^n_{f,j}(\mathbf{s}) = 1)$ DFDs the weights $w^n_{f,j} = 1$ and $w^n_{f,j} = 0$ respectively.

However we want to discourage the pixel site $(f, \mathbf{s}_0)$ from obtaining the *red* object label.

Hence we require the DFD $\Lambda_f^n(\mathbf{s}_0)$ to be as 'high' as possible. In this case we want the final DFD Λ_f^n to be influenced by the 'high' DFDs ($\mathcal{D}_{f,j}^n(\mathbf{s}) = 1$) only. Hence the 'low' ($\mathcal{D}_{f,j}^n(\mathbf{s}) = 0$) and 'high' ($\mathcal{D}_{f,j}^n(\mathbf{s}) = 1$) DFDs are given the weights $w_{f,j}^n = 0$ and $w_{f,j}^n = 1$ respectively.

The idea here is that we only encourage the site $(f, \mathbf{s})$ to take the label for bundle n if either the closest forward ($\mathcal{D}_{f,f+1}^n(\mathbf{s}) = 0$) or backward ($\mathcal{D}_{f,f-1}^n(\mathbf{s}) = 0$) DFD is a 'low' value. Sites that have 'high' DFDs in both directions are discouraged from taking the label of bundle n. A decision must be made whether a pixel site $(f, \mathbf{s})$ should be encouraged or discouraged from taking the label for a particular bundle. We define a discouragement label $d_f^n(\mathbf{s})$ for pixel site $(f, \mathbf{s})$, where $d_f^n(\mathbf{s}) = 0$ and $d_f^n(\mathbf{s}) = 1$ mean that the site must be encouraged and discouraged respectively. Note that for the examples in the *bottom* row of fig. 7.12, the discouragement labels are $d_f^n(\mathbf{s}_1) = 0$, $d_f^n(\mathbf{s}_2) = 0$ and $d_f^n(\mathbf{s}_0) = 1$. The weights $w_{f,j}^n$ for the DFDs $\Delta_{f,j}^n(\mathbf{s})$ are determined from the discouragement label $d_f^n(\mathbf{s})$ as follows.

$$w_{f,j}^n = \begin{cases} 1, & \textbf{if } d_f^n(\mathbf{s}) = \mathcal{D}_{f,j}^n(\mathbf{s}) \\ 0, & \textbf{else} \end{cases} \tag{7.16}$$

The discouragement labels $d_f^n(\mathbf{s})$ are estimated only from the DFD labels $\mathcal{D}_{f,f+1}(\mathbf{s})$ and $\mathcal{D}_{f,f-1}(\mathbf{s})$. These labels $d_f^n(\mathbf{s})$ are defined as follows.

$$d_f^n(\mathbf{s}) = \mathcal{D}_{f,f+1}(\mathbf{s}) \times \mathcal{D}_{f,f-1}(\mathbf{s}) \tag{7.17}$$

The next section discusses how we derive the occlusion labels $\mathcal{D}_{f,j}(\mathbf{s})$ for the DFDs. Complete details are found in appendix D.

7.1.6.2 The DFD Occlusion Labels

We enforce spatial and temporal smoothness on the occlusion labels $\mathcal{D}_{f,j}(\mathbf{s})$ for the DFDs by generating a MAP estimate for these labels. Consider that $\mathbf{D}_f^n(\mathbf{s})$ is a label vector formed from the concatenation of the DFD labels $\mathcal{D}_{f,j}(\mathbf{s})$ where $\mathbf{D}_f^n(\mathbf{s})$ is defined as follows.

$$\mathbf{D}_f^n(\mathbf{s}) = \left[\mathcal{D}_{f,f-k}^n(\mathbf{s}), \ldots, \mathcal{D}_{f,f-1}^n(\mathbf{s}), \mathcal{D}_{f,f+1}^n(\mathbf{s}), \ldots, \mathcal{D}_{f,f+k}^n(\mathbf{s})\right] \tag{7.18}$$

As an example the occlusion label vectors for the sites $(f, \mathbf{s}_0)$, $(f, \mathbf{s}_1)$ and $(f, \mathbf{s}_2)$ in the *bottom* row of fig. 7.12 are $\mathbf{D}_f^n(\mathbf{s}_0) = [1\ 1\ 1\ 1\ 1\ 1]$, $\mathbf{D}_f^n(\mathbf{s}_1) = [0\ 0\ 0\ 1\ 1\ 1]$ and $\mathbf{D}_f^n(\mathbf{s}_2) = [0\ 0\ 0\ 0\ 0\ 0]$ respectively.

We define the posterior distribution for the occlusion vector label $\mathbf{D}_f^n(\mathbf{s})$ given the DFDs $\Delta_{f,j}^n(\mathbf{s})$ at the *window* frames and the neighbouring labels $\mathbf{D}_f^n(\sim \mathbf{s})$ as follows.

$$p(\mathbf{D}_f^n(\mathbf{s})|\Delta_{f,j}^n(\mathbf{s}), \mathbf{D}_f^n(\sim \mathbf{s})) = p_d(\Delta_{f,j}^n(\mathbf{s})|\mathbf{D}_f^n(\mathbf{s}))p_l(\mathbf{D}_f^n(\mathbf{s})|\mathbf{D}_f^n(\sim \mathbf{s})) \tag{7.19}$$

Where $p_d(\Delta_{f,j}^n(\mathbf{s})|.)$ is the likelihood that the pixel site $(f, \mathbf{s})$ takes the occlusion vector label $\mathbf{D}_f^n(\mathbf{s})$. Label smoothness is injected through the MRF prior $p_l(.)$.

The design of the distributions for prior and likelihood are exposed in detail in appendix D. The MAP estimation is found using a Graphcut technique. Note however, that the number of possible vectors labels $\mathbf{D}_f^n(\mathbf{s})$ is very large (2^{2k}). This would be a prohibitive number of labels in the Graphcut process. Hence we determine a smaller set of candidate labels by extracting local conditional models from a maximum likelihood examination in patches over the image. These candidate $\hat{\mathbf{D}}_f^{n,c}$ are then used as the labels in the Graphcut process.

At the end of this process, we are able to create the discourage labels defined in eq. 7.17. We are also able to determine the weights $w_{f,j}^n$ in eq. 7.16 for creating the DFD measurement in the motion likelihood $p_x(\mathcal{X}(f,\mathbf{s})|.)$.

7.1.7 Appearance Likelihood

Colour information is traditionally modelled as Gaussian mixture models (GMMs). However when the colour samples being modelled follow a non-Gaussian distribution, GMMs are not be very informative. Hence, in order to handle arbitrary distributions, we introduce a crude but novel colour model for the design of the appearance likelihood $p_i(\mathbf{I}_f(\mathbf{s})|.)$.

Consider that the colour at pixel site $(f,\mathbf{s})$ is $\mathbf{I}_f(\mathbf{s}) = [R_f(\mathbf{s})\ G_f(\mathbf{s})\ B_f(\mathbf{s})]$, where $R_f(\mathbf{s}), G_f(\mathbf{s})$ and $B_f(\mathbf{s})$ are the values for the red, green and blue channels in the RGB colour space. This colour space is divided into 13 rank ordered partitions. A partition is defined in terms of the sorting of the colour components, e.g. $R_f(\mathbf{s}) > G_f(\mathbf{s}) > B_f(\mathbf{s})$ defines partition 1, $R_f(\mathbf{s}) > G_f(\mathbf{s}) = B_f(\mathbf{s})$ defines partition 2 and so on. We define $\mathcal{P}_f(\mathbf{s})$ as the colour space partition label for pixel $(f,\mathbf{s})$. This partition label $\mathcal{P}_f(\mathbf{s})$ for site $(f,\mathbf{s})$ given its quantized RGB channel values $R_f(\mathbf{s}), G_f(\mathbf{s})$ and $B_f(\mathbf{s})$ is defined as follows.

$$\mathcal{P}_f(\mathbf{s}) = \begin{cases} 1, & R_f(\mathbf{s}) > G_f(\mathbf{s}) > B_f(\mathbf{s}) \\ 2, & R_f(\mathbf{s}) > G_f(\mathbf{s}) = B_f(\mathbf{s}) \\ 3, & R_f(\mathbf{s}) > B_f(\mathbf{s}) > G_f(\mathbf{s}) \\ 4, & G_f(\mathbf{s}) > R_f(\mathbf{s}) > B_f(\mathbf{s}) \\ 5, & G_f(\mathbf{s}) > R_f(\mathbf{s}) = B_f(\mathbf{s}) \\ 6, & G_f(\mathbf{s}) > B_f(\mathbf{s}) > R_f(\mathbf{s}) \\ 7, & B_f(\mathbf{s}) > R_f(\mathbf{s}) > G_f(\mathbf{s}) \\ 8, & B_f(\mathbf{s}) > R_f(\mathbf{s}) = B_f(\mathbf{s}) \\ 9, & B_f(\mathbf{s}) > G_f(\mathbf{s}) > R_f(\mathbf{s}) \\ 10, & R_f(\mathbf{s}) = G_f(\mathbf{s}) = B_f(\mathbf{s}) \\ 11, & R_f(\mathbf{s}) = G_f(\mathbf{s}) > B_f(\mathbf{s}) \\ 12, & R_f(\mathbf{s}) = B_f(\mathbf{s}) > G_f(\mathbf{s}) \\ 13, & G_f(\mathbf{s}) = B_f(\mathbf{s}) > R_f(\mathbf{s}) \end{cases} \tag{7.20}$$

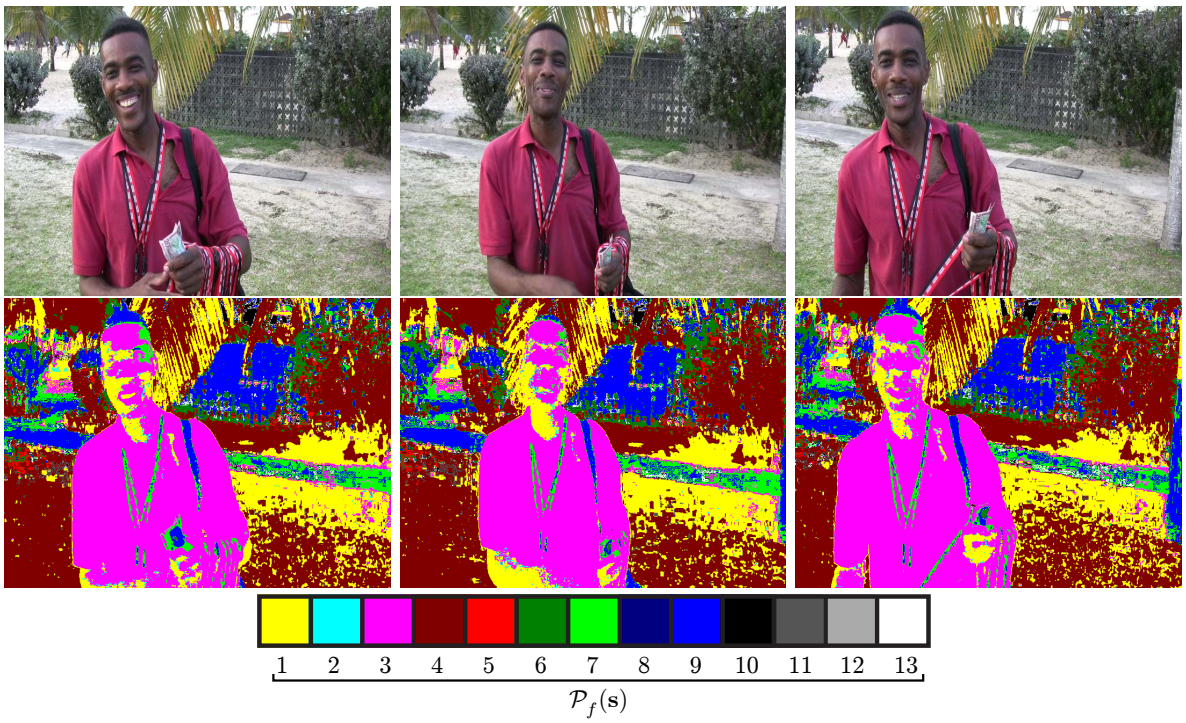

Figure 7.13: ***Top row**: Frames 1, 54 and 99 in the Triniman sequence. **Middle row**: The colour partition labels $\mathcal{P}_f$ for the frames in the top row. **Bottom row**: The **legend** which shows the colours that represent the partition labels $\mathcal{P}_f(\mathbf{s}) = \{1 : 13\}$.*

Where each colour channel is quantized to 256 levels (8-bits), i.e. $R_f(\mathbf{s}) \in \{0 : 255\}$ etc. The inequalities in (7.20) above compare the quantized levels of the three channels in order to decide the partition memberships. The pixels for an object are therefore clustered according to their membership in these 13 partitions.

The *middle* row of fig. 7.13 shows the partition labels $\mathcal{P}_f$ for frames 1 (left), 54 (middle) and 99 (right) for the *Triniman* sequence. Here frame 1, 54 and 99 are shown in the *top* row of fig. 7.13. The *legend* in the *bottom* row indicates the colours that represent the partition label $\mathcal{P}_f(\mathbf{s}) = \{1 : 13\}$. Note that our partitioning of the RGB colour space provides an implicit segmentation for each frame. It may be observed that the majority of the colours for the actor belong to the colour space partition $\mathcal{P}_f(\mathbf{s}) = 3$, $(R_f(\mathbf{s}) > B_f(\mathbf{s}) > G_f(\mathbf{s}))$. This partition label is coloured *pink* in the *bottom* row of fig. 7.13. Here the colours for the actor have higher values in the red channels compared to the other colour channels. For the background, the majority of the colours belong to the *maroon red* colour partition $(\mathcal{P}_f(\mathbf{s}) = 4)$. The colours for the background have higher values in the *green* channel compared to the other colour channels.

The next section discusses how the colour space partitions are used in the design of the appearance likelihood.

7.1.7.1 Appearance Likelihood based on Partitioned Colour Space

The appearance likelihoods $p_i(\mathbf{I}_f(\mathbf{s})|.)$ for the current frame f are derived directly from the previously segmented frame β. For the *forward* and *backward* label propagation steps, frame β is frame $f-1$ and $f+1$ respectively.

The appearance likelihood $p_i(\mathbf{I}_f(\mathbf{s})|.)$ can be factored into two components as follows.

$$p_i(\mathbf{I}_f(\mathbf{s})|\mathcal{L}_f(\mathbf{s})) = p_c(\mathbf{I}_f(\mathbf{s})|\mathcal{L}_f(\mathbf{s}))p_r(\mathbf{I}_f(\mathbf{s})|\mathcal{L}_f(\mathbf{s})) \tag{7.21}$$

Where $p_c(\mathbf{I}_f(\mathbf{s})|.)$ is the likelihood of the colour $\mathbf{I}_f(\mathbf{s})$ at pixel site $\mathbf{s}$ in frame f belonging to the object with label $\mathcal{L}_f(\mathbf{s})$. Also, $p_r(\mathbf{I}_f(\mathbf{s})|.)$ is the likelihood of the site $(f, \mathbf{s})$ belonging to the object labelled $\mathcal{L}_f(\mathbf{s})$ given the spatial locations of the colours for object $\mathcal{L}_f(\mathbf{s})$. We can consider the likelihood component $p_c(\mathbf{I}_f(\mathbf{s})|.)$ as a global constraint in the RGB colour space on the allowable colours for an object $\mathcal{L}_f(\mathbf{s})$. However the likelihood component $p_r(\mathbf{I}_f(\mathbf{s})|.)$ introduces local geometric constraints in the image plane which dictate where certain colours are expected for a particular object.

Each appearance likelihood is built from colour appearance models derived from previous and next frames. This allows the likelihood to depend on the previous as well as next *key* frames eventually.

We use the segmentation in frame β to define the RGB space component of the colour model in the current frame. Hence the colour space appearance model for frame f depends on the estimated colour in frame $\beta = f-1$, in the *forward* label propagation step. Similarly in frame f depends on the estimated labels in frame $\beta = f+1$, in the *backward* label propagation step.

The other model ($p_r(\mathbf{I}_f(\mathbf{s})|.)$) is sensitive to position and for that we need to propagate colour partition labels (in fig. 7.14) in frame $f-1$ (forward) and $f+1$ (backward) into frame f. Fig. 7.14 shows two sets of trajectories corresponding to a *red* labelled object. To propagate the label at site $\tilde{\mathbf{s}}_o$ into the current frame f we simply use the motion of the *cyan* trajectory. This process applied to all sites in frame β yields a set of partition labels $\hat{\mathcal{P}}_{\beta,f}(\mathbf{s})$ in frame f corresponding to object o.

In fig. 7.14 if the *red* object was labelled $\mathcal{L}_\beta(\tilde{\mathbf{s}}_o) = o$ then $\hat{\mathcal{P}}_{\beta,f}(\mathbf{s})$ would ideally cover all of the red object in frame f. Of course that is not the case and there are likely to be holes in the motion compensated label field $\hat{\mathcal{P}}_{\beta,f}(\mathbf{s})$. These issues are resolved in the design of the model discussed next.

The next section discusses how the propagated label field and the colour samples for the various objects are used to derive appearance models.

7.1.7.2 Appearance Models

Recall that we gather appearance information by doing *forward* and *backward* object label propagation steps as previously discussed. For the *forward* and *backward* steps we estimate the appearance models $\mathcal{C}_{f-1,f}(\mathbf{s})$ and $\mathcal{C}_{f+1,f}(\mathbf{s})$ respectively. Here $\mathcal{C}_{f-1,f}(\mathbf{s})$ represents appearance

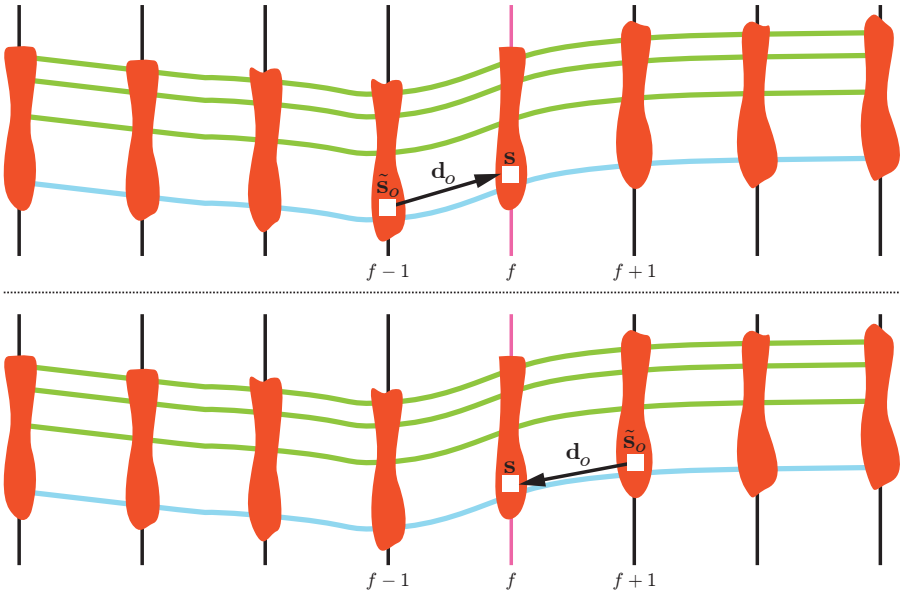

Figure 7.14: *The notation used for discussing the propagation of the colour partition labels from frame β to the current frame f for object o. Here object o is represented by the non-rigid* ***orange*** *blob that is changing its shape from frame to frame. The trajectories associated with this object are shown as the curved lines that extend across the frames. The pixel site $\hat{\mathbf{s}}_o$ (white square) belongs to object o in frame β, and this site corresponds to site s (white square) in the current frame f. The motion vector between these sites is $\delta\hat{\mathbf{s}}_o$, which is represented by the* ***arrow****. The 'closest' trajectory $\mathcal{X}_p$ to site $\hat{\mathbf{s}}_o$ in frame β is the* ***cyan*** *trajectory. This trajectory $\mathcal{X}_p$ is used to estimate the motion vector $\mathbf{d}_o$.* ***Top row****: In the* ***forward*** *label propagation step, frame β is frame $f-1$. The motion vector $\delta\hat{\mathbf{s}}_o$ is directed from frame $f-1$ to the current frame f.* ***Bottom row****: In the* ***backward*** *label propagation step, frame β is frame $f+1$. Here the motion vector $\mathbf{d}_o$ is directed from frame $f+1$ to the current frame f.*

information propagated from frame $f-1$ to the current frame f, while $\mathcal{C}_{f+1,f}(\mathbf{s})$ represents appearance propagated from frame $f+1$ to frame f.

We define $\mathcal{C}^o_{\beta,f}(\mathbf{s})$ as the appearance model for pixel site $(f,\mathbf{s})$ with respect to the object label $\mathcal{L}_f(\mathbf{s}) = o$. This model has two components, one is used in the likelihood term $p_c(\mathbf{I}_f(\mathbf{s})|.)$ that considers the similarity between the colour $\mathbf{I}_f(\mathbf{s})$ at site $(f,\mathbf{s})$, and the colours of object o. The other component of $\mathcal{C}^o_{\beta,f}(\mathbf{s})$ is used in the likelihood term $p_p(\mathbf{I}_f(\mathbf{s})|.)$. This component is based on the nearest pixel site $(f,\hat{\mathbf{s}}_o)$ to site $(f,\mathbf{s})$ in the image plane, that has the same colour partition label as $(f,\mathbf{s})$. Here we are using the propagated colour partition label field $\hat{\mathcal{P}}^o_{f,\beta}$ (discussed in previous section) for object o to find the site $(f,\hat{\mathbf{s}}_o)$.

For the colour $\mathbf{I}_f(\mathbf{s})$ at site $(f,\mathbf{s})$, the nearest neighbour colour to $\mathbf{I}_f(\mathbf{s})$ is defined as $\mathbf{C}^o_\beta(\mathbf{s})$, where $\mathbf{C}^o_\beta(\mathbf{s})$ is a colour sample $\mathbf{I}_\beta(\tilde{\mathbf{s}}_o)$ belonging to object o in frame β. We constrain this nearest neighbour search by requiring that the colour partition label for the closest colour $\mathbf{C}^o_\beta(\mathbf{s})$

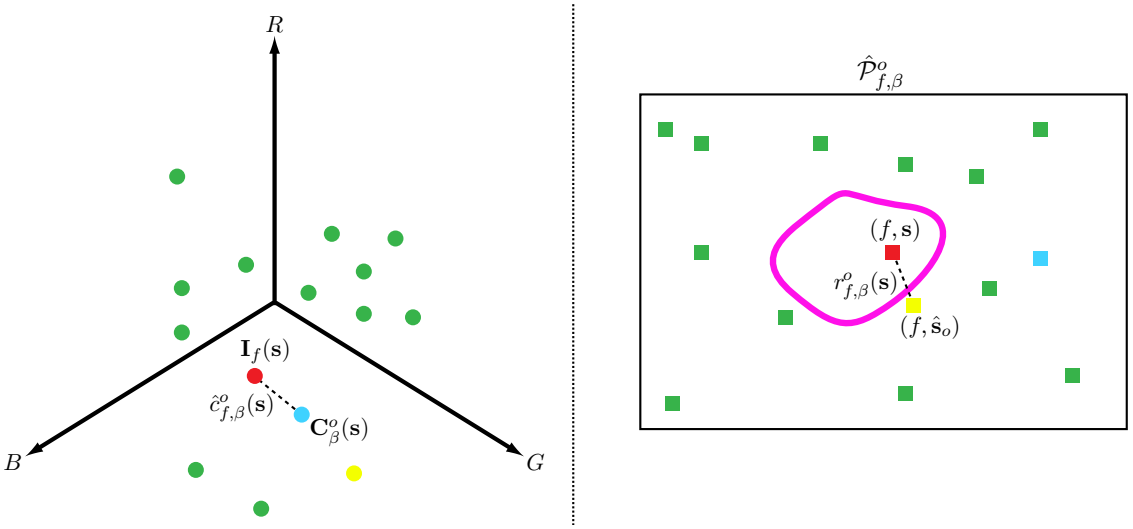

Figure 7.15: *An illustration of the two components* $\hat{c}^o_{f,\beta}(\mathbf{s})$ *and* $r^o_{f,\beta}(\mathbf{s})$ *of the appearance model* $\mathcal{C}^o_{\beta,f}(\mathbf{s})$ *for pixel site* $(f,\mathbf{s})$*, with respect to object* o*.* ***Left****: The colour* $\mathbf{I}_f(\mathbf{s})$ *at site* $(f,\mathbf{s})$ *is indicated with a* ***red*** *dot, while the other coloured dots represent colour samples for object* o*. The* ***cyan*** *dot indicates the colour sample* $\mathbf{C}^o_\beta(\mathbf{s})$*, which is the closest to* $I_f(\mathbf{s})$ *in the RGB colour space. The Euclidean distance between* $I_f(\mathbf{s})$ *and* $\mathbf{C}^o_\beta(\mathbf{s})$ *is defined as* $\hat{c}^o_{f,\beta}(\mathbf{s})$*. This distance is illustrated as the dotted line.* ***Right****: The site* $(f,\mathbf{s})$ *in the colour partition label field* $\hat{\mathcal{P}}^o_{f,\beta}$ *is indicated with a* ***red*** *square, while the other sites in this field having the partition label* $\mathcal{P}_f(\mathbf{s})$ *are indicated by the* ***green***, ***cyan*** *and* ***yellow*** *squares. Here the partition label field* $\hat{\mathcal{P}}^o_{f,\beta}$ *is propagated from frame* β *to the current frame* f *using the trajectory for object* o*. Note that the* ***pink*** *closed contour simulates an object boundary. The site* $(f,\hat{\mathbf{s}}_o)$ *(yellow square) is the closest point to site* $(f,\mathbf{s})$ *in the field* $\hat{\mathcal{P}}^o_{f,\beta} = \mathcal{P}_f(\mathbf{s})$*. The Euclidean distance between sites* $(f,\mathbf{s})$ *and* $(f,\hat{\mathbf{s}}_o)$ *is defined as* $r^o_{f,\beta}(\mathbf{s})$*. This distance is illustrated as the dotted line.*

be the same as the partition label for site $(f,\mathbf{s})$.

The illustration on the *left* of fig. 7.15 shows the colour $\mathbf{I}_f(\mathbf{s})$ for site $(f,\mathbf{s})$ and the closest colour $\mathbf{C}^o_\beta(\mathbf{s})$ sample from object o as *red* and *cyan* dots respectively in RGB colour space. The *green* and *yellow* coloured dots represent other colour samples for object o that have the same partition label as site $(f,\mathbf{s})$. The Euclidean distance between $\mathbf{I}_f(\mathbf{s})$ and $\mathbf{C}^o_\beta(\mathbf{s})$ is defined as $\hat{c}^o_{f,\beta}(\mathbf{s})$. This distance $\hat{c}^o_{f,\beta}(\mathbf{s})$ is shown as the dotted line in the illustration on the *left* of fig. 7.15. We now define the first component of the appearance model $\mathcal{C}^o_{\beta,f}(\mathbf{s})$ for pixel site $(f,\mathbf{s})$ as the distance $\hat{c}^o_{f,\beta}(\mathbf{s})$ which is given as follows.

$$\hat{c}^o_{f,\beta}(\mathbf{s}) = \left\|\mathbf{I}_f(\mathbf{s}) - \mathbf{C}^o_\beta(\mathbf{s})\right\| \tag{7.22}$$

The illustration on the *right* of fig. 7.15 shows the propagated colour partition label field $\hat{\mathcal{P}}^o_{f,\beta}$ for object o. The *red* square is spatially located at site $(f,\mathbf{s})$, and the other sites in the field $\hat{\mathcal{P}}^o_{f,\beta}$

that have the partition label $\mathcal{P}_f(\mathbf{s})$ are indicated with *green*, *cyan* and *yellow* squares. Recall that the partition label for site $(f, \mathbf{s})$ is $\mathcal{P}_f(\mathbf{s})$. We are interested in finding the closest point to site $(f, \mathbf{s})$ for which $\hat{\mathcal{P}}^o_{f,\beta} = \mathcal{P}_f(\mathbf{s})$. This closest point $(f, \hat{\mathbf{s}}_o)$ is indicated on the *right* of fig. 7.15 as a *yellow* square. The Euclidean distance between site $(f, \mathbf{s})$ and the closest point $(f, \hat{\mathbf{s}}_o)$ is defined as $r^o_{f,\beta}(\mathbf{s})$. We now define the second component of the appearance model $\mathcal{C}^o_{\beta,f}(\mathbf{s})$ for pixel site $(f, \mathbf{s})$ as the distance $r^o_{f,\beta}(\mathbf{s})$.

Note that the colour sample $\mathbf{C}^o_\beta(\mathbf{s})$ that is the closest to $\mathbf{I}_f(\mathbf{s})$ in the colour space may not provide the smallest partition label distance $r^o_{f,\beta}(\mathbf{s})$. The colour samples on the *left* of fig. 7.15 and their corresponding site in the partition label field $\hat{\mathcal{P}}^o_{f,\beta}$ on the *right* are coloured in a similar manner to emphasize this idea. The *cyan* colour sample is the closest to $\mathbf{I}_f(\mathbf{s})$ in the colour space, while in the partition label field the *yellow* site is the closest to site $(f, \mathbf{s})$.

The next section discusses how the appearance models $\mathcal{C}^o_{\beta,f}(\mathbf{s})$ are utilized in the design of appearance likelihood distributions.

7.1.7.3 Appearance Likelihood Distribution

Recall that the appearance likelihood $p_i(\mathbf{I}_f(\mathbf{s})|.)$ takes different forms for the *forward* and *backward* propagation, as well as for the final segmentation. These different forms of the appearance likelihood $p_i(\mathbf{I}_f(\mathbf{s})|.)$ which were previously discussed are repeated below.

$$p_i(\mathbf{I}_f(\mathbf{s})|\mathcal{L}_f(\mathbf{s})) = \begin{cases} p(\mathbf{I}_f(\mathbf{s})|\mathcal{L}_f(\mathbf{s}), \mathcal{C}_{f-1,f}(\mathbf{s})), & \text{forward propagation} \\ p(\mathbf{I}_f(\mathbf{s})|\mathcal{L}_f(\mathbf{s}), \mathcal{C}_{f+1,f}(\mathbf{s})), & \text{backward propagation} \\ p(\mathbf{I}_f(\mathbf{s})|\mathcal{L}_f(\mathbf{s}), \mathcal{C}_{f+1,f}(\mathbf{s}), \mathcal{C}_{f-1,f}(\mathbf{s})), & \text{final segmentation} \end{cases} \tag{7.23}$$

We introduced the two components of the appearance likelihood $p_i(\mathbf{I}_f(\mathbf{s})|.)$ in eq. 7.21 as $p_c(\mathbf{I}_f(\mathbf{s})|.)$ and $p_p(\mathbf{I}_f(\mathbf{s})|.)$.

The likelihood term $p_c(\mathbf{I}_f(\mathbf{s})|.)$ for pixel site $(f, \mathbf{s})$ is the product of weighted Gaussian distributions of the colour distances $\hat{c}^o_{f,f-1}(\mathbf{s})$ and $\hat{c}^o_{f,f+1}(\mathbf{s})$. These colour distances $\hat{c}^o_{f,f-1}(\mathbf{s})$ and $\hat{c}^o_{f,f+1}(\mathbf{s})$ are components of the appearance models $\mathcal{C}^o_{f-1,f}(\mathbf{s})$ and $\mathcal{C}^o_{f+1,f}(\mathbf{s})$ for site $(f, \mathbf{s})$ obtained from the *forward* and *backward* label propagation steps respectively. The distribution $p_c(\mathbf{I}_f(\mathbf{s})|.)$ is given below.

$$p_c(\mathbf{I}_f(\mathbf{s})|\mathcal{L}_f(\mathbf{s})) = \frac{1}{S} \times \exp\left\{ -0.5 \left(\lambda^{\mathcal{F}}_f \left[\frac{\hat{c}^o_{f,f-1}(\mathbf{s}) - \bar{c}_{f,f-1}(\mathbf{s})}{\sigma^c_{f,f-1}(\mathbf{s})} \right]^2 + \lambda^{\mathcal{B}}_f \left[\frac{\hat{c}^o_{f,f+1}(\mathbf{s}) - \bar{c}_{f,f+1}(\mathbf{s})}{\sigma^c_{f,f+1}(\mathbf{s})} \right]^2 \right) \right\} \tag{7.24}$$

Where $\lambda^{\mathcal{F}}_f$ and $\lambda^{\mathcal{B}}_f$ are weights on the *forward* and *backward* appearance models respectively at the current frame f. Recall that we only have the appearance models $\mathcal{C}^o_{f-1,f}(\mathbf{s})$ and $\mathcal{C}^o_{f+1,f}(\mathbf{s})$ available for the *forward* and *backward* propagation steps respectively. Hence we use the weights $(\lambda^{\mathcal{F}}_f = 1, \lambda^{\mathcal{B}}_f = 0)$ and $(\lambda^{\mathcal{F}}_f = 0, \lambda^{\mathcal{B}}_f = 1)$ for both steps respectively when evaluating the appearance likelihoods $p_i(\mathbf{I}_f(\mathbf{s})|.)$.

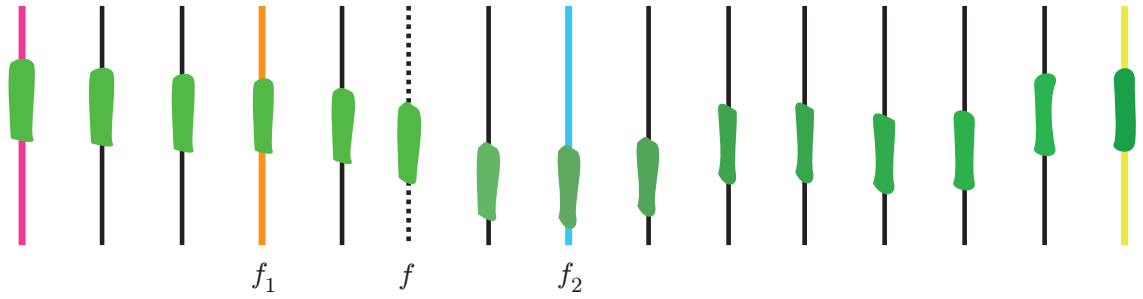

Figure 7.16: *An illustration of the* ***border key*** *frames for the current frame f. Here* ***key*** *frames are indicated with coloured lines.* ***Border key*** *frames are the closest key frames on both sides of the current frame f. For the current frame f (dotted line), the* ***border key*** *frames are f_1 (****orange****) and f_2 (****cyan****).*

The parameters for the *forward* and *backward* Gaussian distributions are $(\bar{c}_{f,f-1}(\mathbf{s}), \sigma^c_{f,f-1}(\mathbf{s}))$ and $(\bar{c}_{f,f+1}(\mathbf{s}), \sigma^c_{f,f+1}(\mathbf{s}))$ respectively. The derivation of these parameters is discussed later. The weight S is the usual normalizing constant, which takes the forms outlined below for the *forward* and *backward* label propagation steps and the final segmentation.

$$S = \begin{cases} \sqrt{2\pi}\sigma^c_{f,f-1}(\mathbf{s}), & \text{forward propagation} \\ \sqrt{2\pi}\sigma^c_{f,f+1}(\mathbf{s}), & \text{backward propagation} \\ 2\pi\sigma^c_{f,f-1}(\mathbf{s})\sigma^c_{f,f+1}(\mathbf{s}), & \text{final segmentation} \end{cases} \tag{7.25}$$

For the final segmentation we have both appearance models $\mathcal{C}^o_{f-1,f}(\mathbf{s})$ and $\mathcal{C}^o_{f+1,f}(\mathbf{s})$ available. In this case we vary the weights $\lambda^{\mathcal{F}}_f$ and $\lambda^{\mathcal{B}}_f$ according to how far away the current frame f is from the *border key* frames. Recall that the *border key* frames are the closest *key* frames on both sides of the current frame f. The illustration in fig. 7.16 shows the *border key* frames for the current frame 'f' that are labelled 'f_1' (orange) and 'f_2' (cyan). Given that f_1 and f_2 are the *border key* frames for the current frame f, the weights $\lambda^{\mathcal{F}}_f$ and $\lambda^{\mathcal{B}}_f$ are defined as follows.

$$\begin{aligned} \lambda^{\mathcal{F}}_f &= \frac{f_2 - f}{f_2 - f_1}, \\ \lambda^{\mathcal{B}}_f &= 1 - \lambda^{\mathcal{F}}_f \end{aligned} \tag{7.26}$$

The second likelihood term $p_r(\mathbf{I}_f(\mathbf{s})|.)$ (equ. 7.21) is defined in a similar way to $p_c(\mathbf{I}_f(\mathbf{s})|.)$. This distribution is the product of weighted Gaussian distributions of the partition label distances $r^o_{f,f-1}(\mathbf{s})$ and $r^o_{f,f+1}(\mathbf{s})$. These partition label distances are components of the appearance models $\mathcal{C}^o_{f-1,f}(\mathbf{s})$ and $\mathcal{C}^o_{f+1,f}(\mathbf{s})$ for site $(f, \mathbf{s})$, which were discussed in the previous section. The distribution $p_r(\mathbf{I}_f(\mathbf{s})|.)$ is given below.

$$p_r(\mathbf{I}_f(\mathbf{s})|\mathcal{L}_f(\mathbf{s})) = \frac{1}{U}\times$$
$$\exp\left\{-0.5\left(\lambda_f^{\mathcal{F}}\left[\frac{r^o_{f,f-1}(\mathbf{s}) - \bar{r}_{f,f-1}(\mathbf{s})}{\sigma^r_{f,f-1}(\mathbf{s})}\right]^2 + \lambda_f^{\mathcal{B}}\left[\frac{r^o_{f,f+1}(\mathbf{s}) - \bar{r}_{f,f+1}(\mathbf{s})}{\sigma^r_{f,f+1}(\mathbf{s})}\right]^2\right)\right\} \tag{7.27}$$

Where the parameters for the *forward* and *backward* distributions are $(\bar{r}_{f,f-1}(\mathbf{s}), \sigma^r_{f,f-1}(\mathbf{s}))$ and $(\bar{r}_{f,f+1}(\mathbf{s}), \sigma^r_{f,f+1}(\mathbf{s}))$ respectively. The weight U is the usual normalizing constant, which takes the forms outlined below for the *forward* and *backward* label propagation steps and the final segmentation.

$$U = \begin{cases} \sqrt{2\pi}\sigma^r_{f,f-1}(\mathbf{s}), & \text{forward propagation} \\ \sqrt{2\pi}\sigma^r_{f,f+1}(\mathbf{s}), & \text{backward propagation} \\ 2\pi\sigma^r_{f,f-1}(\mathbf{s})\sigma^r_{f,f+1}(\mathbf{s}), & \text{final segmentation} \end{cases} \tag{7.28}$$

The next section outlines the derivations of the parameters for the appearance likelihood distribution.

7.1.7.4 Appearance Likelihood Distribution Parameters

The parameters for the appearance likelihood are defined as follows.

$$\begin{aligned}
\bar{c}_{f,f-1}(\mathbf{s}) &= \min_o\left(\hat{c}^o_{f,f-1}(\mathbf{s})\right), \ \left[\sigma^c_{f,f-1}(\mathbf{s})\right]^2 = \frac{1}{9}\sum_{\mathbf{m}\in\mathcal{N}_\mathbf{s}}\left[\bar{c}_{f,f-1}(\mathbf{m})\right]^2 \\
\bar{c}_{f,f+1}(\mathbf{s}) &= \min_o\left(\hat{c}^o_{f,f+1}(\mathbf{s})\right), \ \left[\sigma^c_{f,f+1}(\mathbf{s})\right]^2 = \frac{1}{9}\sum_{\mathbf{m}\in\mathcal{N}_\mathbf{s}}\left[\bar{c}_{f,f+1}(\mathbf{m})\right]^2 \\
\bar{r}_{f,f-1}(\mathbf{s}) &= \min_o\left(r^o_{f,f-1}(\mathbf{s})\right), \ \left[\sigma^r_{f,f-1}(\mathbf{s})\right]^2 = \frac{1}{9}\sum_{\mathbf{m}\in\mathcal{N}_\mathbf{s}}\left[\bar{r}_{f,f-1}(\mathbf{m})\right]^2 \\
\bar{r}_{f,f+1}(\mathbf{s}) &= \min_o\left(r^o_{f,f+1}(\mathbf{s})\right), \ \left[\sigma^r_{f,f+1}(\mathbf{s})\right]^2 = \frac{1}{9}\sum_{\mathbf{m}\in\mathcal{N}_\mathbf{s}}\left[\bar{r}_{f,f+1}(\mathbf{m})\right]^2
\end{aligned} \tag{7.29}$$

Where $\mathcal{N}_\mathbf{s}$ is the 3×3 neighbourhood of pixel site $\mathbf{s}$ including $\mathbf{s}$.

Note that these parameters in eq. 7.29 are defined in a similar manner to the parameters for the geodesic distance likelihood in section 7.1.4.2. The reasoning here is the same. We center the distributions $p_c(\mathbf{I}_f(\mathbf{s})|.)$ and $p_r(\mathbf{I}_f(\mathbf{s})|.)$ on the minimum colour and partition label distances respectively. Also the standard deviations are proportional the square of the minimum distances for the sites in the 3×3 neighbourhood of site $\mathbf{s}$.

7.1.8 The Prior

The prior $p_s(\mathcal{L}_f(\mathbf{s})|\mathcal{L}_f(\sim \mathbf{s}))$ at site $(f, \mathbf{s})$ for the object label $\mathcal{L}_f(\mathbf{s})$ given the neighbouring labels $\mathcal{L}_f(\sim \mathbf{s})$ is a Gibbs distribution defined as follows.

$$p_s(\mathcal{L}_f(\mathbf{s})|\mathcal{L}_f(\sim \mathbf{s})) = \frac{1}{Z} \exp\left\{-\lambda_L \sum_{c \in \mathcal{N}_s} [\mathcal{L}_f(\mathbf{s}) \neq \mathcal{L}_f(\mathbf{c})]\, G(\mathbf{s}, \mathbf{c})\right\} \tag{7.30}$$

The neighbourhood of pixel site $(f, \mathbf{s})$ is defined as $\mathcal{N}_s$, where $\mathcal{N}_s$ is the set of pixel sites surrounding $(f, \mathbf{s})$ in a 4-connected structure. The weight Z above in eq. 7.30 is the usual normalization constant. The constant λ_D controls the weight the prior has in the MAP solution. For all experiments it was found that setting $\lambda_L = 0.01$ gave suitable results. The weight $G(\mathbf{s}, \mathbf{c})$ determines how much the label $\mathcal{L}_f(\mathbf{s})$ for site $(f, \mathbf{s})$ is influenced by the neighbouring label $\mathcal{L}_f(\mathbf{c})$ for site $(f, \mathbf{c})$. We designed the weight $G(\mathbf{s}, \mathbf{c})$ to allow neighbouring pixels to influence each other if they have similar intensities. This weight $G(\mathbf{s}, \mathbf{c})$ is defined as follows.

$$G(\mathbf{s}, \mathbf{c}) = \begin{cases} 1 - \frac{\delta I_f(\mathbf{s},\mathbf{c})}{\lambda_I}, & \textbf{if } \delta I_f(\mathbf{s}, \mathbf{c}) < \lambda_I \\ 0, & \textbf{else} \end{cases} \tag{7.31}$$

$$for\ \delta I_f(\mathbf{s}, \mathbf{c}) = |I_f(\mathbf{s}) - I_f(\mathbf{c})|$$

Where $I_f(\mathbf{s})$ is the gray scale version of the colour $\mathbf{I}_f(\mathbf{s})$ at site $(f, \mathbf{s})$. The constant λ_I is the intensity difference above which no neighbouring influence is required. For all experiment reported later we set $\lambda_I = 20$.

7.2 Graphcut Solution

The α-*expansion* Graphcut algorithm [135] is used to solve for the MAP estimate of the object label field $\mathcal{L}_f$ for the current frame f. The posterior distribution for the object labels $\mathcal{L}_f(\mathbf{s})$ discussed previously is repeated below.

$$p(\mathcal{L}_f(\mathbf{s})|\mathcal{X}, \mathbf{I}, \mathcal{L}_f(\sim \mathbf{s})) \propto p_x(\mathcal{X}(f, \mathbf{s})|\mathcal{L}_f(\mathbf{s}))p_i(\mathbf{I}_f(\mathbf{s})|\mathcal{L}_f(\mathbf{s}))p_s(\mathcal{L}_f(\mathbf{s})|\mathcal{L}_f(\sim \mathbf{s})) \tag{7.32}$$

Details of the adaptation of the Bayesian problem in eq. 7.32 above to this Graphcut solution are exposed in appendix B. There are no major issues to raise as far as the Graphcut techniques are concerned.

7.3 Summary

In this chapter we presented two *dense pixel segmentation* techniques. One technique is semi-automatic, while the other segments a sequence automatically. However both techniques utilize

the same general segmentation framework. The major difference between both techniques is that a user supplies a set of segmented reference mattes called *key* frames for the semi-automatic process, while we estimate these *key* frames in the automatic process. These *key* frames for the automatic segmentation process are estimated using motion and geodesic distance [10, 111] information.

The general segmentation framework uses the trajectory bundles generated from our *sparse trajectory segmentation* (chapter 4) to introduce long term motion and spatial information into the dense segmentation process. These bundles are used to estimate motion models over several frames that are used in our likelihood designs. Also the trajectories in the bundles are used to propagate appearance information. For the automatic segmentation process, the spatial locations of the trajectories are used to estimate geodesic distances which are then used to generate *key* frames.

8

Dense Segmentation Performance Evaluation

In this chapter the performances of both our automatic (*Geodesic*) and semi-automatic (*Semi-Auto*) dense segmentation technique, along with three previous method [12,62,100] are compared using the *Triniman*, *Artbeats-SP128*, and *Calendar and Mobile* sequences. The nature of these sequences was previously discussed in chapter 6 where we presented the sparse trajectory segmentation for all three sequences.

Recall that *Triniman* and *Artbeats-SP128* are real world sequences, that contain significant amounts of non-rigid object motions. The *top and middle* rows of fig. 8.1 show frames 7,57,91 and 20,62,95 for the *Triniman* and *Artbeats-SP128* sequences respectively. For both sequences there are one and two non-rigid foreground objects respectively.

The *Triniman* sequence is 99 frames (720×540 pels) of a dynamic outdoor scene recorded with a hand-held camcorder. The challenge in producing a reasonable segmentation for *Triniman* is that there are several non-rigid motions throughout this sequence. In the background there are swaying tree branches and the foreground actor himself is non-rigid. Also we have to cope with the global motion of the camera as well.

The *Artbeats-SP128* sequence is 99 frames (720×576 pels) of a scene from an American football game. The foreground objects here are player #1 and #2 (fig 8.1) in red and white jerseys respectively. Both players are moving and deforming very quickly from left to right in the image plane. Player #1 partially occludes player #2 from frames 73-99. The challenges here are coping with the non-rigid motions and segmenting both foreground objects (players #1 and #2) separately. For this sequence the camera is static, hence there is no global camera motion. However the people in the background are moving, and this introduces another challenge in

Figure 8.1: *Example frames for the three sequences used to assess the performance of our dense segmentation techniques.* ***Top row****: Frames 7, 57 and 91 in the* ***Triniman*** *sequence.* ***Middle row****: Frames 20, 62 and 95 in the* ***Artbeats-SP128*** *sequence.* ***Bottom row****: Frames 5, 13 and 23 in the* ***Calendar and Mobile*** *sequence.*

producing a reasonable segmentation.

The *bottom* row of fig 8.1 shows frames 5,13 and 23 in the well known *Calendar and Mobile* sequence. This sequence is 25 frames (720×576 pels) of a scene with four rigidly moving objects. The four objects are the background, along with three foreground objects; a calendar, a toy train, and a ball. For this sequence the camera is panning from left to right. There are several shadows casted by the foreground objects which are challenging to segment correctly.

The results of the our techniques on the three sequences are compared to that of a movie compositing software package called *NUKE* developed by The Foundry [2]. The motion segmentation algorithm of *NUKE* in the MotionMatte plugin is based on a traditional *unsupervised* dense motion segmentation algorithm using two motion models [62]. Each of these motion models describe the motion of the background and foreground objects respectively.

Figure 8.2: *Examples of the ground truth mattes for* ***Triniman***, ***Artbeats-SP128*** *and* ***Calendar and Mobile*** *sequences.* ***Top row****: Frames 43, 57 and 23 from the three sequences respectively.* ***Bottom row****: The ground truth mattes for the frames in the top row.*

We also compared our *semi-automatic* technique with two state-of-the-art *supervised* dense segmentation techniques called *FeatureCut* and *SnapCut* proposed by Ring [100] and Bai *et al.* [12] respectively. Both these techniques are discussed in the review done in chapter 2.

We have manually segmented ground truth mattes for all three sequences. The *bottom* row of fig. 8.2 shows examples of these mattes for the *Triniman* (left), *Artbeats-SP128* (center) and *Calendar and Mobile* (right) sequences. The frames these mattes correspond to are shown in the *top* row of fig. 8.2.

We will refer to our automatic and semi-automatic dense segmentation approaches as *Geodesic* and *Semi-Auto* respectively. Using the ground truth mattes, we compared the recall and false alarm rates for both our approaches along with the *NUKE* segmentations. We also compared our *semi-automatic* technique with *FeatureCut* and *SnapCut* using recall and false alarm rate.

It will be demonstrated later in this chapter that our *Semi-Auto* segmentation technique provides the best overall recall and false alarm rates for the three sequences. Also our *Geodesic* segmentation technique has better rates than the *NUKE* technique.

In the next section we will outline how we compare the segmentations of our *Geodesic*, *Semi-Auto* and the *NUKE* technique. Comparisons with *FeatureCut* and *SnapCut* are done later in this chapter.

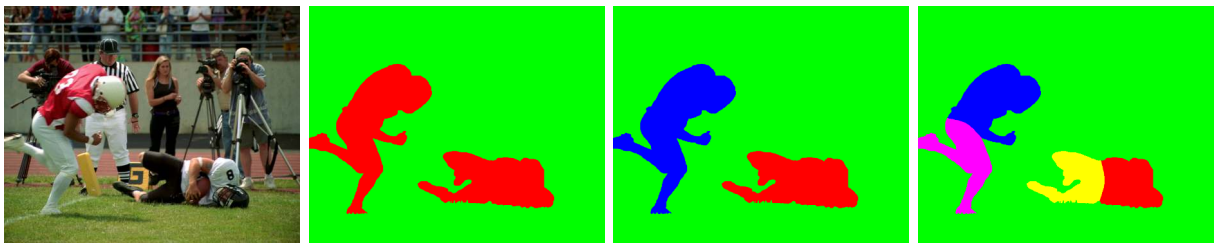

Figure 8.3: *An illustration of different levels of segmentation.* ***First column****: Frame* 57 *in the Triniman sequence.* ***Second column****: A* ***foreground/background*** *level segmentation of frame* 57*, where the background and foreground are labelled* ***green*** *and* ***red*** *respectively.* ***Third column****: A* ***object*** *level segmentation of frame* 57*, where each foreground object is labelled separately. That is, the background, player #1 and #2 are labelled* ***green****,* ***red*** *and* ***blue*** *respectively.* ***End column****: A* ***image region*** *level segmentation of frame* 57*, where various image regions for a particular object are labelled separately. Here the lower and upper body of player #1 are labelled* ***yellow*** *and* ***red*** *respectively. Also the lower and upper body of player #2 are labelled* ***pink*** *and* ***blue*** *respectively.*

8.1 Levels of Segmentation

Our *Semi-Auto* segmentation approach segments a sequences according to the object descriptions supplied in the user defined *key* frames. Fig. 8.3 shows examples of three user defined mattes that dictate different levels of segmentation. The matte in the *second* column specifies a **foreground/background** segmentation, where the foreground and background objects are coloured in *red* and *green* respectively (two labels). However, the matte in the *third* column requires that the two objects in the foreground be labelled *red* and *blue*. Hence a pixel site can be labelled *green* (background), *red* (player #1) or *blue* (player #2). This desired segmentation is defined as an **object** level segmentation. The user defined matte in the *last* column of fig. 8.3 specifies that various image regions corresponding to a particular object should be labelled separately. As an example, the lower and upper body of player #2 are labelled *pink* and *blue* respectively. This desired segmentation is defined as an **image region** level segmentation.

For all the three sequences we conducted **object** level segmentation using our *Semi-Auto* segmentation approach. Recall that the *Geodesic* segmentation approach labels the pixels in a sequence as belonging to a specific trajectory bundle. These bundles are obtained from our sparse trajectory segmentation process. Here each bundle represents a specific image region, hence the *Geodesic* segmentation approach produces **image region** level segmentations.

However, NUKE produces a **foreground/background** level segmentation. Hence, in order to compare the segmentation of all three algorithms, we must first express them as **foreground/background** level segmentations. With this level of segmentation we can then proceed to comparing foreground recall and false alarm rates using our ground truth segmentations.

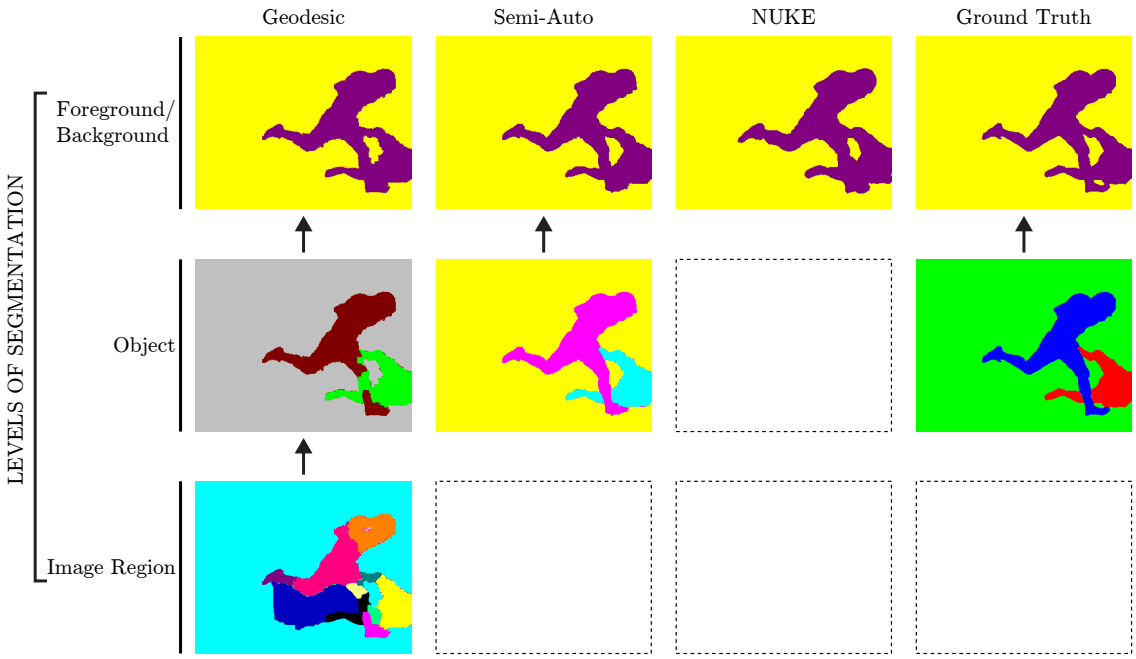

Figure 8.4: *An illustration using an example frame from the* ***Artbeats-SP128*** *sequence showing how* ***foreground/background*** *level segmentations are determined for our* ***Semi-Auto*** *and* ***Geodesic*** *segmentation techniques. We use the ground truth matte to express the* ***object*** *(middle row) and* ***image region*** *(bottom row) level segmentations as* ***foreground/background*** *level segmentations (top row). The ground truth segmentation is supplied as an* ***object*** *level segmentation which is shown in the* ***center*** *of the* ***last*** *column. Here the background, player #1 and #2 are coloured in* ***green***, ***red*** *and* ***blue*** *respectively. All the labels that are not* ***green*** *constitute the foreground. Hence the we obtain the* ***foreground/background*** *level segmentation for the ground truth, which is shown in the* ***top right*** *picture. We simply label a segment from the* ***object*** *or* ***image region*** *level segmentation as foreground if the majority of the pixels for this segment correspond to the foreground region in the ground truth. The* ***arrows*** *indicate the transitions between the various levels of segmentation. Each column shows the segmentations for the* ***Geodesic***, ***Semi-Auto*** *and* ***NUKE*** *approaches as well as the supplied* ***Ground Truth***.

In the next section we will discuss how we determine the **foreground/background** level segmentation for our *Semi-Auto* and *Geodesic* segmentation approaches.

8.1.1 Foreground/Background Segmentation

Fig. 8.4 illustrates the different levels of segmentation for the three techniques using an example frame from the *Artbeats-SP128* sequence. The ground truth segmentation shown in the *center* of the *last* column is supplied as an **object** level segmentation. Here the background, player #1 and #2 are coloured in *green*, *red* and *blue* respectively. All the objects that are not *green* consti-

tute the foreground. Hence we can reexpress the ground truth as a **foreground/background** segmentation which is shown in the *top right* of fig. 8.4.

We use the **foreground/background** level ground truth segmentation (*top right* of fig. 8.4) to assess the **foreground/background** level segmentation for the *Semi-Auto* and *Geodesic* approaches. We simply label a segment from the **object** (*Semi-Auto*) or **image region** (*Geodesic*) level segmentation as foreground if the majority of the pixels for this segment corresponds to the foreground region in the ground truth. The arrows in fig. 8.4 indicate the transitions between the levels of segmentation for the *Geodesic*, *Semi-Auto* and ground truth segmentations. The required **foreground/background** level segmentations are shown in the *top* row of fig. 8.4.

8.2 Segmentation of the Three Sequences

In general our *Semi-Auto* and *Geodesic* approaches produce good segmentations for all three sequences. However, the *NUKE* approach only produces similar segmentations for the *Artbeats-SP128* sequence. The quality of the *NUKE* segmentations seems to deteriorate whenever there is some sort of global camera motion. With global motion present, the task of detecting foreground motion becomes more difficult. Our segmentation techniques do not have these shortcomings.

In the next three sections we will discuss the segmentation for the three sequences.

8.2.1 Triniman

Fig. 8.5 shows the segmentations for frames 7, 57 and 91 in the *Triniman* sequence. These frames are shown in the *top* row. The *second, third and bottom* rows show the segmentations produced by our *Semi-Auto*, *Geodesic* and the *NUKE* approaches respectively. Recall that the segmentations produced by the *Semi-Auto*, *Geodesic* and *NUKE* approaches are **object**, **image region** and **foreground/background** level segmentations respectively as mentioned previously. Note in fig. 8.5 the different levels of segmentation produced by these three techniques. For our *Semi-Auto* segmentation approach we used 12 user segmented mattes and the remaining 87 frames in this sequence were estimated in the segmentation process.

It may be observed that our *Semi-Auto* approach produces the best segmentation, followed by our *Geodesic* approach. Both of these approaches produce temporally consistent segmentations. However, the NUKE approach produces a poorer quality segmentation which is not temporally consistent. The swaying tree branches are incorrectly identified as foreground.

Fig. 8.6 shows the equivalent **foreground/background** level segmentations for the segmentations shown in fig. 8.5 for the *Triniman* sequence. Here the foregrounds are extracted using the corresponding **foreground/background** level segmentations and the backgrounds are shaded in green. This level of segmentation is used to compare the segmentations of the three techniques quantitatively, as previously discussed. This quantitative analyses for all three sequences are presented later.

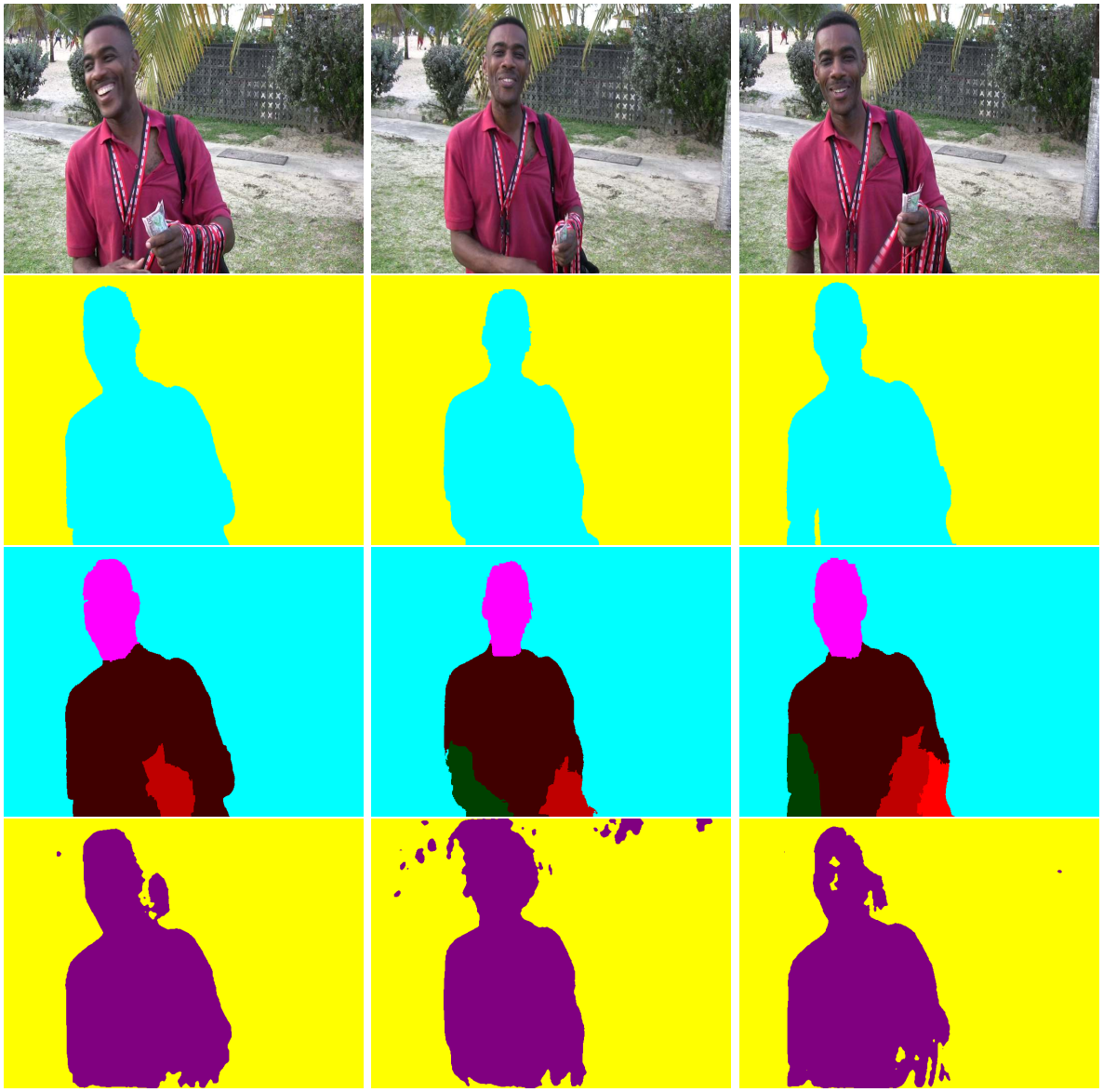

Figure 8.5: *The segmentations for three frames in the* ***Triniman*** *sequence.* ***Top row****: Frames* 7 *(left),* 57 *(center) and* 91 *(right) in this sequence.* ***Second row****: The* ***object*** *level segmentations for the frames in the top row produced by our* ***Semi-Auto*** *approach. Since there is only one foreground object, this segmentation is the same as a* ***foreground/background*** *segmentation. The background and actor are coloured* ***yellow*** *and* ***cyan*** *respectively.* ***Third row****: The* ***image region*** *level segmentations for the frames in the top row produced by our* ***Geodesic*** *approach. The background is coloured in* ***cyan****. The various image regions corresponding the actor are coloured* ***pink*** *(face),* ***green*** *(right arm),* ***maroon red*** *(torso),* ***red*** *(upper left hand), and* ***orange*** *(lower left arm).* ***Bottom row****: The* ***foreground/background*** *level segmentations of the frames in the top row produced by* ***NUKE****. The background and foreground are coloured* ***yellow*** *and* ***purple*** *respectively.*

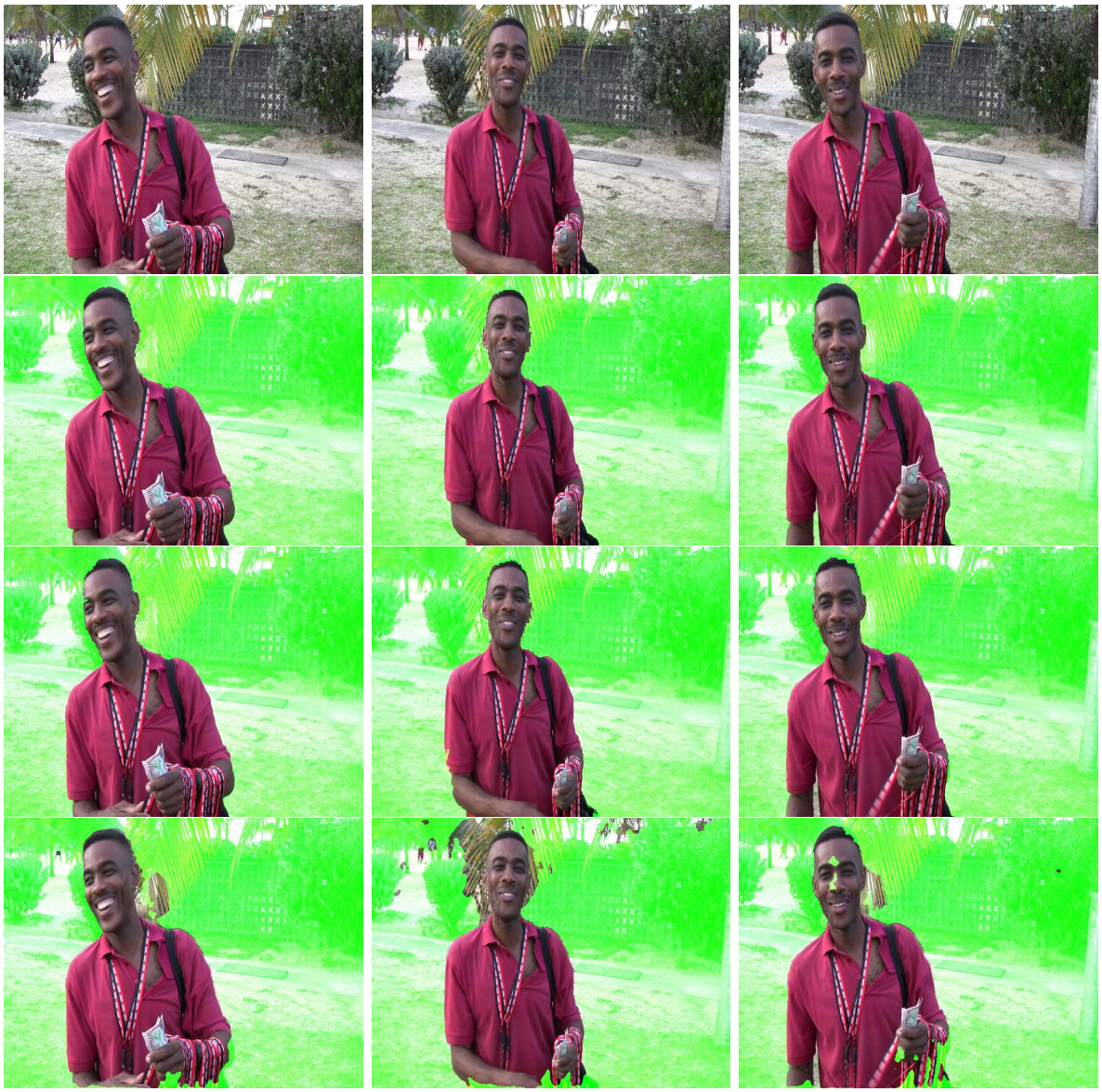

Figure 8.6: *The* ***foreground/background*** *level segmentations for three frames in the* ***Triniman*** *sequence. The foregrounds are extracted using the respective segmentations and the backgrounds are shaded in green.* ***Top row****: Frames* 7, 57 *and* 91 *in this sequence.* ***Second, third and bottom rows****: The segmentations for the frames in the top row produced by our* ***Semi-Auto****,* ***Geodesic*** *and the* ***NUKE*** *approaches respectively.*

8.2.2 Artbeats-SP128

Fig. 8.7 shows the segmentations for frames 20, 62 and 95 in the *Artbeats-SP128* sequence. The equivalent **foreground/background** level segmentations are shown in fig. 8.8. The *second, third and bottom* rows in both fig. 8.7 and fig. 8.8 show the segmentations produced by the *Semi-Auto*, *Geodesic* and *NUKE* approaches respectively. For the *Semi-Auto* segmentation approach we used 18 user segmented mattes and the remaining 81 frames in this sequence were estimated in the segmentation process.

Again our *Semi-Auto* approach provides the best segmentation for the *Artbeats-SP128* sequence. The *NUKE* segmentations are slightly better then those for the *Geodesic* approach. Here the *Geodesic* approach suffers from a lack of image texture on the legs of both players (players are wearing plain tights). Hence there are not much feature point trajectories on the legs of the players, which are required to produce a good segmentation using our *Geodesic* approach. Also the static camera for the *Artbeats-SP128* sequence means that *NUKE* does not have to do any background motion compensation. Therefore the task of detecting foreground motion is simplified in this case.

8.2.3 Calendar and Mobile

Fig. 8.9 shows the segmentations for frames 5, 13 and 23 in the *Calendar and Mobile* sequence. The equivalent **foreground/background** level segmentations are shown in fig. 8.10. The *second, third and bottom* rows in both figs. 8.9 and 8.10 show the segmentations produced by our *Semi-Auto*, *Geodesic* and the *NUKE* approaches respectively. For our *Semi-Auto* segmentation approach we used 6 user segmented mattes and the remaining 19 frames in this sequence were estimated in the segmentation process.

Our *Semi-Auto* technique also produces the best segmentations for this sequence, followed by our *Geodesic* technique. The *NUKE* segmentation approach generally fails to differentiate between the subtle motions of the calendar and the background. Hence the majority of the upper half of the calendar is labelled as being part of the background. In both our *Semi-Auto* and *Geodesic* approaches, we benefit from using appearance information and the spatial locations of the trajectories in order to produce the desired segmentations for this sequence.

8.3 Quantitative Analysis

By expressing the segmentations produced by our *Semi-Auto* and *Geodesic* approaches as **foreground/background** level segmentations we can then proceed to comparing these segmentations with those produced by *NUKE*. Here we use the usual quantitative measures of foreground recall and false alarm rates as the basis for our comparisons. The recall and false alarm rates

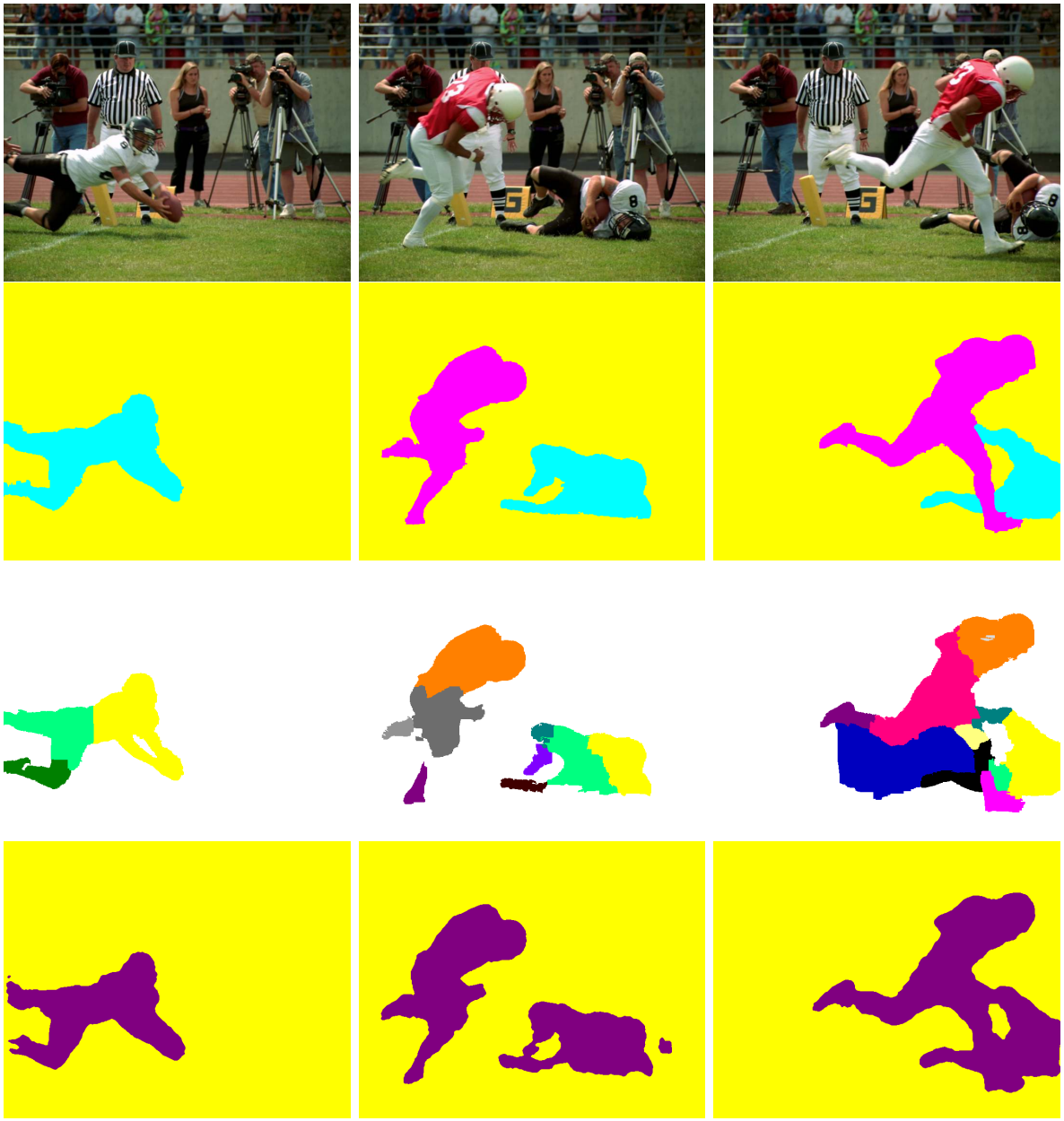

Figure 8.7: *The segmentations for three frames in the* ***Artbeats-SP128*** *sequence.* ***Top row****: Frames* 20, 62 *and* 95 *in this sequence.* ***Second row****: The* ***object*** *level segmentations for the frames in the top row produced by our* ***Semi-Auto*** *approach. The background, player #1 and #2 are coloured* ***yellow****,* ***cyan*** *and* ***pink*** *respectively.* ***Third row****: The* ***image region*** *level segmentations for the frames in the top row produced by our* ***Geodesic*** *approach. The background is coloured in* ***white*** *and* ***dark blue*** *(in frame on the right). The various image regions corresponding the players are in different colours, where corresponding regions are coloured similarly across the frames.* ***Bottom row****: The* ***foreground/background*** *level segmentations of the frames in the top row produced by* ***NUKE****. The background and foreground are coloured* ***yellow*** *and* ***purple*** *respectively.*

Figure 8.8: *The **foreground/background** level segmentations for three frames in the **Artbeats-SP128** sequence. The foregrounds are extracted using the respective segmentations and the backgrounds are shaded in green. **Top row**: Frames 20, 62 and 95 in this sequence. **Second, third and bottom rows**: The segmentations for the frames in the top row produced by our **Semi-Auto**, **Geodesic** and the **NUKE** approaches respectively.*

Figure 8.9: *The segmentations for three frames in the* ***Calendar and Mobile*** *sequence.* ***Top row****: Frames* 5, 13 *and* 23 *in this sequence.* ***Second row****: The* ***object*** *level segmentations for the frames in the top row produced by our* ***Semi-Auto*** *approach. The background, toy train, calendar and ball are coloured* ***yellow****,* ***cyan****,* ***pink*** *and* ***maroon red*** *respectively.* ***Third row****: The* ***image region*** *level segmentations for the frames in the top row produced by our* ***Geodesic*** *approach. The background, toy train, calendar and ball are coloured* ***cyan****,* ***pink****,* ***green*** *and* ***brown*** *respectively.* ***Bottom row****: The* ***foreground/background*** *level segmentations of the frames in the top row produced by* ***NUKE****. The background and foreground are coloured* ***yellow*** *and* ***purple*** *respectively.*

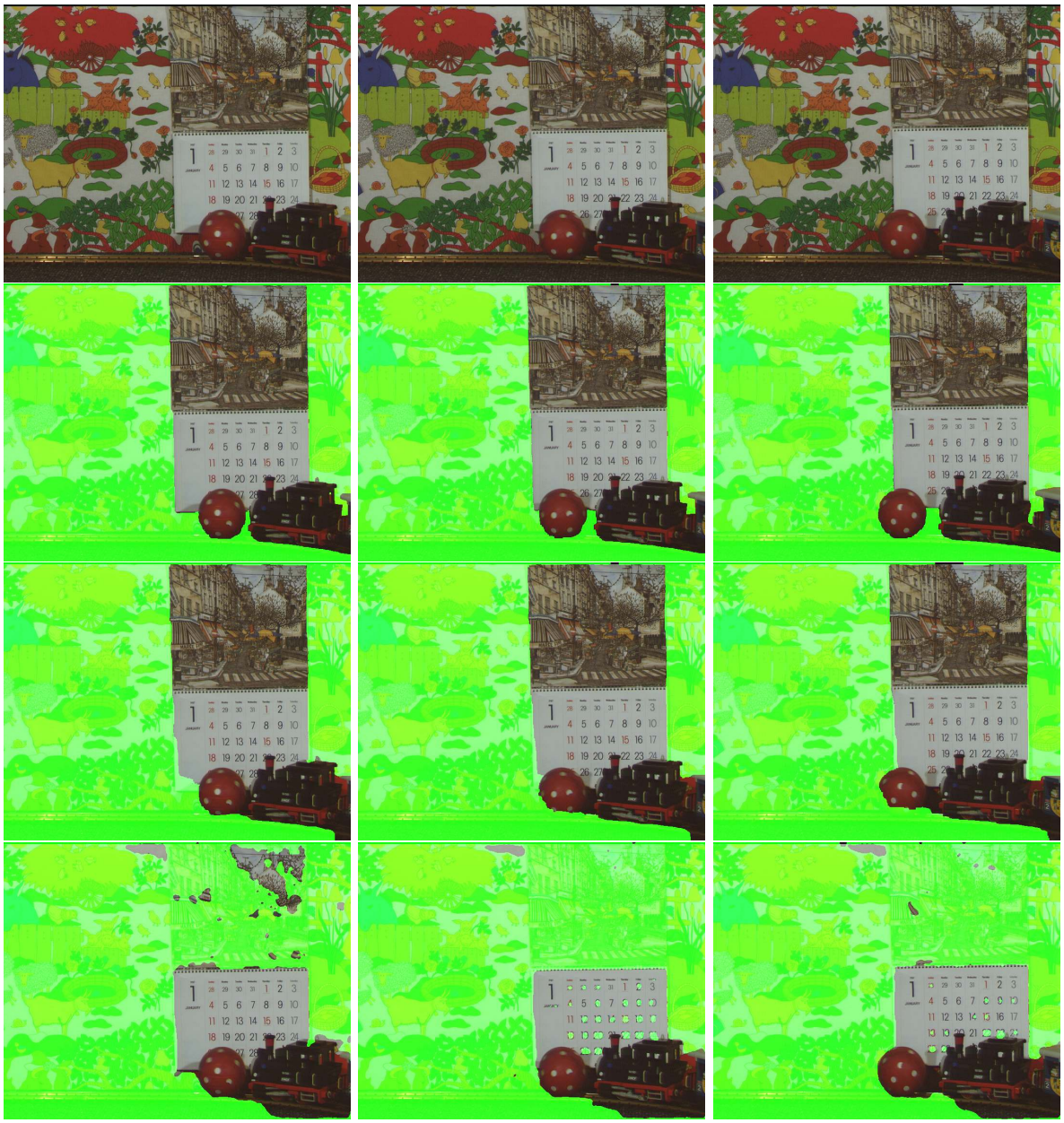

Figure 8.10: *The **foreground/background** level segmentations for three frames in the **Calendar and Mobile** sequence. The foregrounds are extracted using the respective segmentations and the backgrounds are shaded in green. **Top row**: Frames 5, 13 and 23 in this sequence. **Second, third and bottom rows**: The segmentations for the frames in the top row produced by our **Semi-Auto**, **Geodesic** and the **NUKE** approaches respectively.*

		Mattes	
Sequence	#Frames	#User Supplied	#Estimated
Triniman	99	12	87
Artbeats-SP128	99	18	81
Calendar and Mobile	25	6	19
All	223	36	187

Table 8.1: *The number of frames (#Frames), user supplied mattes (#User Supplied) and estimated mattes (#Estimated) for the* ***Triniman****,* ***Artbeats-SP128*** *and* ***Calendar and Mobile*** *sequences.*

are defined as follows.

$$\text{Recall} = \frac{\#\text{correct foreground pixels}}{\#\text{foreground pixels}} \times 100\%, \tag{8.1}$$

$$\text{False alarm} = \frac{\#\text{incorrect foreground pixels}}{\#\text{background pixels}} \times 100\%$$

Where the foreground and background pixels are identified using the ground truth segmentations. A high recall rate with a corresponding low false alarm rate is indicative of a good segmentation.

Recall that we do *forward* and *backward* label propagation steps in both our *Semi-Auto* and *Geodesic* segmentations approaches in order to gather appearance information. The *forward* and *backward* steps involve processing a sequence from frame $1 : F$ and $F : 1$ respectively. Hence we obtain two sets of segmented frames in this process, i.e. for each frame f we have a *forward* and *backward* segmentation. Both these segmentations for frame f are combined in some way (described in chapter 7) to produce the final segmentation for this frame.

We define the *forward* and *backward* segmentation processes for our *Semi-Auto* approach as *Semi-Auto(F)* and *Semi-Auto(B)* respectively. Also the *forward* and *backward* segmentation processes for our *Geodesic* approach are defined as *Geodesic(F)* and *Geodesic(B)* respectively. We will show later that our final *Semi-Auto* segmentations are improved by utilizing both the *Semi-Auto(F)* and *Semi-Auto(B)* segmentations. Since the general framework is the same, it is reasonable to assume that our final *Geodesic* segmentations are also improved by utilizing both the *Geodesic(F)* and *Geodesic(B)* segmentations. To avoid redundancy we only consider the case for the *Semi-Auto* approach only.

The qualities of our *Semi-Auto* segmentations are obviously related to the number of user defined mattes used in this process. We will demonstrate later that we can reduce the number of user supplied mattes without significantly compromising the integrity of our segmentations. Table 8.1 summaries the number of user supplied and estimated mattes for the three sequences. For the 223 frames in all three sequences 83.86% are estimated in the *Semi-Auto* segmentation

Triniman	Semi-Auto(F)	Semi-Auto(B)	Semi-Auto	Geodesic	NUKE
Average	99.65%	99.62%	**99.69**%	98.60%	91.66%
Median	99.68%	99.70%	99.69%	98.81%	94.16%
Artbeats SP128	Semi-Auto(F)	Semi-Auto(B)	Semi-Auto	Geodesic	NUKE
Average	97.56%	97.86%	**98.09**%	90.97%	95.80%
Median	97.98%	98.15%	98.33%	92.00%	96.90%
Calendar & Mobile	Semi-Auto(F)	Semi-Auto(B)	Semi-Auto	Geodesic	NUKE
Average	99.36%	99.42%	**99.53**%	97.34%	52.25%
Median	99.39%	99.48%	99.54%	97.38%	50.83%
All	Semi-Auto(F)	Semi-Auto(B)	Semi-Auto	Geodesic	NUKE
Average	98.86%	98.97%	**99.10**%	95.64%	79.90%

Table 8.2: *The average and median recall rates for the* ***Triniman****,* ***Artbeats-SP128*** *and* ***Calendar and Mobile*** *sequences. The segmentation approaches shown are* ***Semi-Auto(F)****,* ***Semi-Auto(B)****,* ***Semi-Auto****,* ***Geodesic*** *and NUKE. The* ***bottom row*** *shows the average rates over all the three sequences.*

process. We will show later in this chapter that we can increase the number of estimated mattes to 95.07% and still maintain better recall and false alarm rates than both *NUKE* and our *Geodesic* approaches.

The next two sections will discuss the recall and false alarm rates for the three segmentation techniques. Note that the results for our *Semi-Auto* approach are based on us estimating 83.86% (See the breakdown in table 8.1) of the mattes for the three sequences.

8.3.1 Recall Rates

Table 8.2 show the recall rates for the *Semi-Auto(F)*, *Semi-Auto(B)*, *Semi-Auto*, *Geodesic* and *NUKE* segmentations. Here the average and median rates over all the frames in each of the three sequences are shown.

Our *Semi-Auto* approach has the best recall rates for the *Triniman* (99.69%), *Artbeats-SP128* (98.09%) and *Calendar and Mobile* (99.53%) sequences. The *Geodesic* approach has better recall rates for the *Triniman* and *Calendar and Mobile* sequences compared to *NUKE*. However *NUKE* has a better recall rate for the *Artbeats-SP128* sequence. As previously mentioned, the *Geodesic* approach produces poorer segmentations for this sequence because of the lack of image texture on the legs of the players.

It may be observed from the recall rates in table 8.2 that the overall final *Semi-Auto* segmentations are always better than either the *Semi-Auto(F)* or *Semi-Auto(B)* segmentations. Hence doing *forward* and *backward* segmentation steps improves the recall rates for our final

Triniman	Semi-Auto(F)	Semi-Auto(B)	Semi-Auto	Geodesic	NUKE
Average	0.25%	0.21%	0.21%	**0.17**%	12.48%
Median	0.23%	0.20%	0.19%	0.14%	4.49%
Artbeats SP128	Semi-Auto(F)	Semi-Auto(B)	Semi-Auto	Geodesic	NUKE
Average	0.68%	0.69%	**0.60**%	0.61%	1.53%
Median	0.74%	0.73%	0.63%	0.57%	1.67%
Calendar & Mobile	Semi-Auto(F)	Semi-Auto(B)	Semi-Auto	Geodesic	NUKE
Average	0.34%	0.40%	**0.31**%	1.17%	4.26%
Median	0.34%	0.40%	0.31%	1.17%	4.00%
All	Semi-Auto(F)	Semi-Auto(B)	Semi-Auto	Geodesic	NUKE
Average	0.42%	0.43%	0.37%	0.65%	6.09%

Table 8.3: *The average and median false alarm rates for the* ***Triniman****,* ***Artbeats-SP128*** *and* ***Calendar and Mobile*** *sequences. The segmentation approaches shown are* ***Semi-Auto(F)****,* ***Semi-Auto(B)****,* ***Semi-Auto****,* ***Geodesic*** *and NUKE. The* ***bottom row*** *shows the average rates over all the three sequences.*

segmentations.

Recall that for the *Calendar and Mobile* sequence, *NUKE* consistently failed to segment the majority of the calendar as foreground (See fig. 8.10). Hence this approach has a relatively low average recall rate of 52.25%.

Overall, considering all three sequence our *Semi-Auto* approach has the best recall rate (99.10%), followed by our *Geodesic* approach (95.64%). The overall recall rate for *NUKE* is 79.90%.

8.3.2 False Alarm Rates

Table 8.3 show the false rates for the *Semi-Auto(F)*, *Semi-Auto(B)*, *Semi-Auto*, *Geodesic* and *NUKE* segmentations. Our *Semi-Auto* approach has the best false alarm for the *Artbeats-SP128* (0.60%) and *Calendar and Mobile* (0.31%) sequences. Our *Geodesic* approach has a slightly better rate (0.17%) than the *Semi-Auto* approach (0.21%) for the *Triniman* sequence. Both approaches are by far much better than *NUKE* in term of false alarm rates.

For the *Triniman* sequence, *NUKE* has problems differentiating between the motions of the actor (foreground) and the swaying tree branches in the background (See fig. 8.6). Hence the tree branches are consistently being classified as foreground, which is reflected in the high false alarm rate of 12.48%. Both our segmentation approaches correctly identify that the tree branches are not apart of the foreground, and this is confirmed by the corresponding low false alarm rates (0.21% and 0.17%).

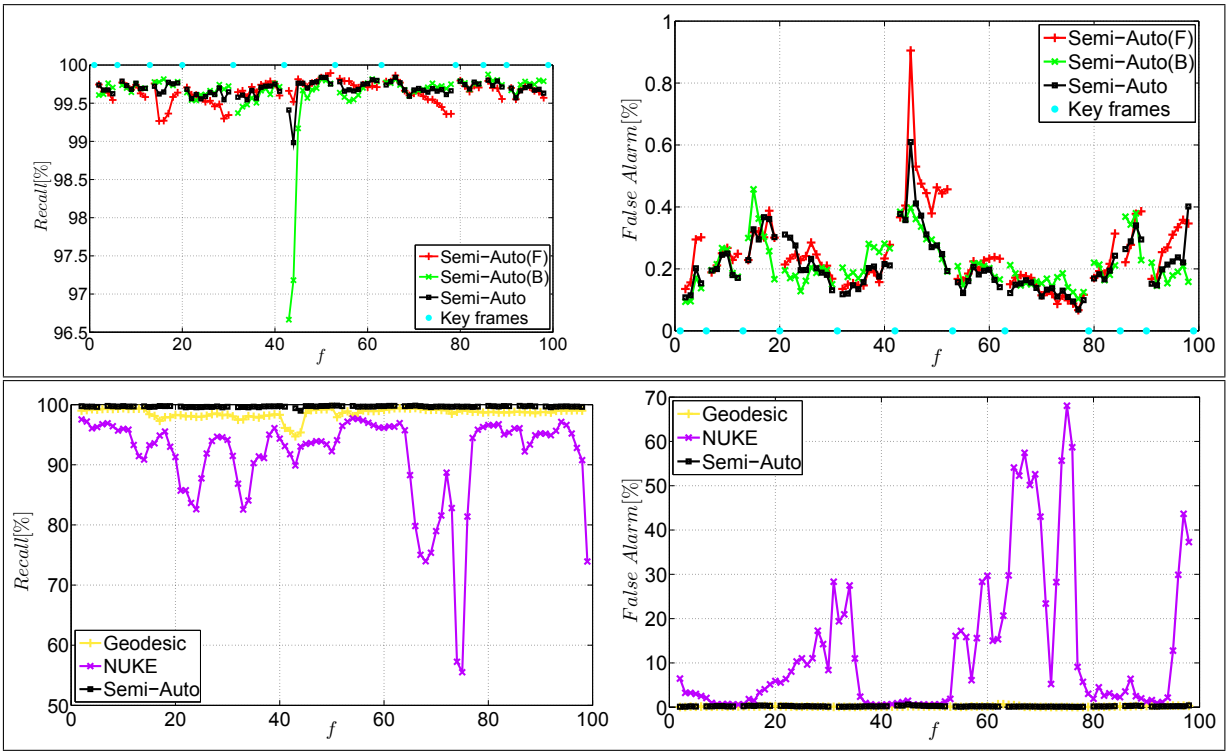

Figure 8.11: *Plots of the recall (Recall[%]) and false alarm (False Alarm[%]) rates per frame (f) for the* ***Triniman*** *sequence.* ***Top row****: The recall (left) and false alarm (right) rates for the* ***Semi-Auto(F)*** *(red),* ***Semi-Auto(B)*** *(green) and* ***Semi-Auto*** *(black) segmentations. The frame locations of the* ***key*** *frames are indicated with* ***cyan*** *dots.* ***Bottom row****: The recall (left) and false alarm (right) rates for the* ***Geodesic*** *(yellow),* ***NUKE*** *(purple plots) and* ***Semi-Auto*** *(black) segmentations.*

The false alarm rates in table 8.3 suggest that the final *Semi-Auto* segmentations are always better than either the *Semi-Auto(F)* or *Semi-Auto(B)* segmentations. Again, as suggested by the recall rates in the previous section, doing *forward* and *backward* segmentation steps improves our final segmentations. Hence we can conclude that both segmentation steps contribute towards a better final segmentation.

Overall, considering all three sequence our *Semi-Auto* approach has the best false alarm rate (0.37%), followed by our *Geodesic* approach (0.65%). The overall recall rate for *NUKE* is 6.09%.

8.3.3 Recall and False Alarm Rates per Frame

Recall and false alarm rates for each frame in the three sequences, allows us to see how the frame locations of the user supplied mattes affect these rates. Also we can assess the temporal consistency in term of the qualities of the segmentations for the three segmentation algorithms.

Figs. 8.11, 8.12 and 8.13 show plots of the recall and false alarm rates for the *Triniman*,

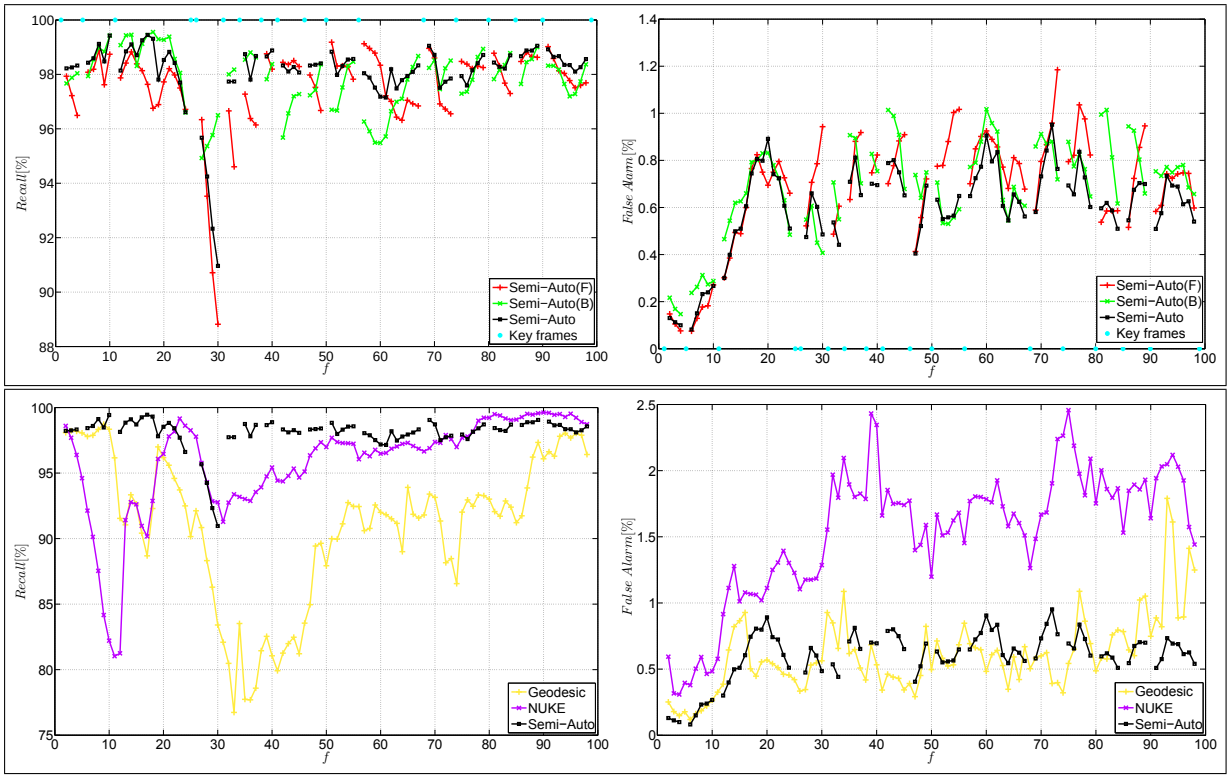

Figure 8.12: *Plots of the recall (Recall[%]) and false alarm (False Alarm[%]) rates per frame (f) for the* ***Artbeats-SP128*** *sequence.* ***Top row****: The recall (left) and false alarm (right) rates for the* ***Semi-Auto(F)*** *(red),* ***Semi-Auto(B)*** *(green) and* ***Semi-Auto*** *(black) segmentations. The frame locations of the* ***key*** *frames are indicated with* ***cyan*** *dots.* ***Bottom row****: The recall (left) and false alarm (right) rates for the* ***Geodesic*** *(yellow),* ***NUKE*** *(purple plots) and* ***Semi-Auto*** *(black) segmentations.*

Artbeats-SP128 and *Calendar and Mobile* sequences respectively. The *top* rows of these figures show plots of the rates for the *Semi-Auto(B)* (green), *Semi-Auto(F)* (red) and *Semi-Auto* (black) segmentations. Here the recall rates (Recall[%]) versus frame index f are on the *left*, and the false alarm rates (False Alarm[%]) versus frame index f are on the *right*. The frame positions of the user supplied mattes (Key frames) are indicated with *cyan* dots. The *bottom* rows of figs. 8.11, 8.12 and 8.13 show plots of the rates for the *Geodesic* (yellow), *NUKE* (purple) and *Semi-Auto* (black) segmentations. Here the recall and false alarm plots are shown on the *left* and *right* respectively of the *bottom* rows. These plots are separated from those in the *top* rows for clarity of illustration.

8.3.3.1 Semi-Auto Segmentation

In our *Semi-Auto* segmentation approach, we expect poorer recall and false alarm rates due to the accumulation of segmentation errors, as we move away from the *key* frames (user supplied mattes). Recall that for the *forward* (*Semi-Auto(F)*) and *backward* (*Semi-Auto(B)*) segmentation steps the frames are processed in the order $1 : F$ and $F : 1$ respectively. Hence, for the *Semi-Auto(F)* segmentations we expect the rates for frame f to be poorer than those for $f - 1$ (given frame $f - 1$ is not a *key frame*). Also for the *Semi-Auto(B)* segmentations we expect the rates for frame f to be poorer than those for $f + 1$ (given frame $f + 1$ is not a *key frame*).

From the plot in the *top* rows of figs. 8.11, 8.12 and 8.13, we may observe our expectations of the recall and false alarm rates getting poorer for the *Semi-Auto(F)* (red plots) and *Semi-Auto(B)* (green plots) segmentations, as we move away from the *key* frames. Note that the segmentation at frame f is influenced only by the *key* frame (*cyan* dot) to the *left* and *right* of frame f, for the *Semi-Auto(F)* and *Semi-Auto(B)* segmentation steps respectively.

Recall that the final segmentations for our *Semi-Auto* approach are dependent on weighted combinations of the appearance models obtained from both the *Semi-Auto(F)* and *Semi-Auto(B)* segmentations. Here these combination weights are dependent on temporal distances from the *key* frames in the *forward* and *backward* directions. Hence, we can consider the final segmentations as combinations of the 'best' segmentations from both the *Semi-Auto(F)* and *Semi-Auto(B)* segmentations.

It may be observed in the *top* rows of figs. 8.11, 8.12 and 8.13 that the recall and false alarms for the *Semi-Auto* segmentations (black plots) are closer to those for the *Semi-Auto(F)* (red plots) and *Semi-Auto(B)* (green plots) segmentations depending on the temporal distances from the *key* frames (*cyan* dots). Effectively, by utilizing the 'best' segmentations from both the *forward* and *backward* segmentations we obtain better recall and false alarm rates for our final segmentations. These better rates are indicative of better segmentations.

8.3.3.2 Consistency of Segmentation

The *bottom* rows of figs. 8.11, 8.12 and 8.13 show the plots of the recall and false rates for the *Triniman*, *Artbeats-SP128* and *Calendar and Mobile* sequences respectively. Here temporally consistent segmentations for a sequence are reflected in these plots as roughly constant recall and false alarm rates from frame to frame.

The lack of temporal consistency in the segmentations for *NUKE* (purple plots) are reflected in the fluctuating recall and false alarm rates. Our *Semi-Auto* approach produced the most temporally consistent segmentations, which is reflected in the plots as relatively low fluctuations in the recall and false alarm rates from frame to frame. The *Geodesic* approach only produces temporally consistent segmentations for the *Triniman* and *Calendar and Mobile* sequences. For *Artbeats-SP128*, this approach suffers from a lack of image texture on the legs of the players as previously mentioned.

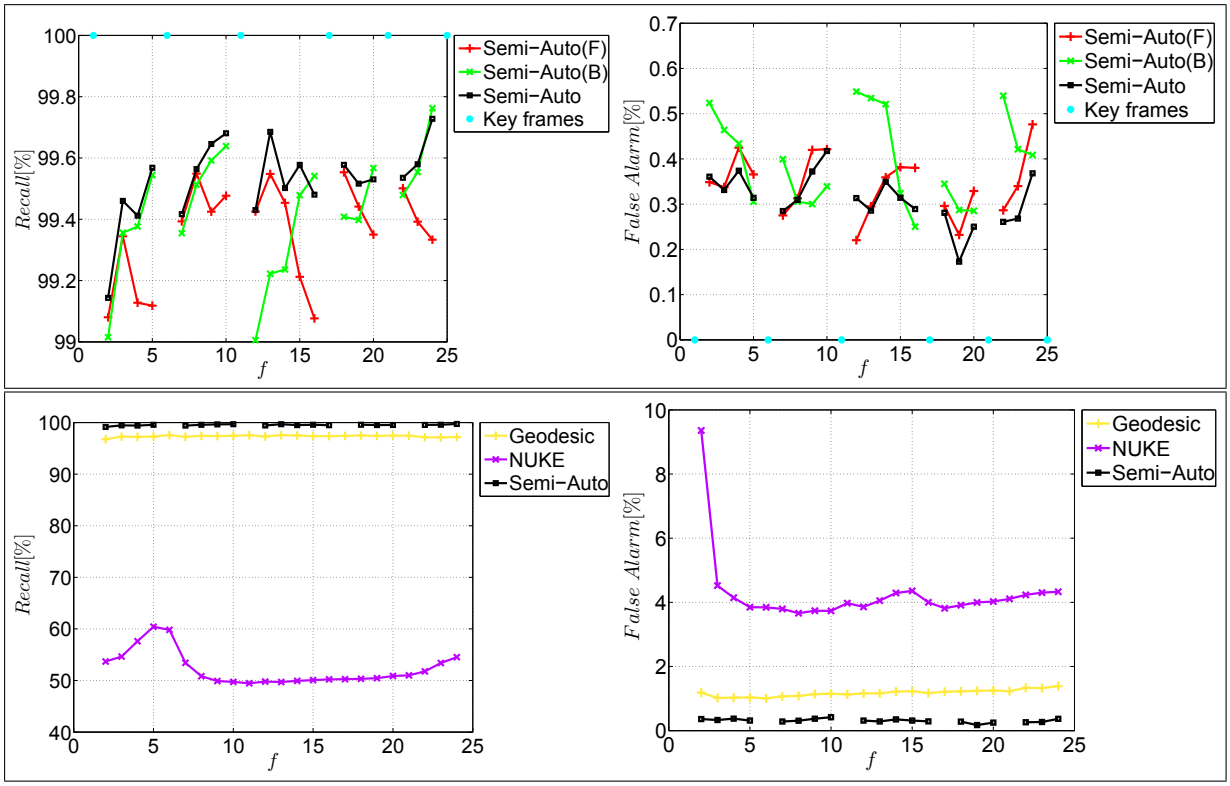

Figure 8.13: *Plots of the recall (Recall[%]) and false alarm (False Alarm[%]) rates per frame (f) for the* ***Calendar and Mobile*** *sequence.* ***Top row****: The recall (left) and false alarm (right) rates for the* ***Semi-Auto(F)*** *(red),* ***Semi-Auto(B)*** *(green) and* ***Semi-Auto*** *(black) segmentations. The frame locations of the* ***key*** *frames are indicated with* ***cyan*** *dots.* ***Bottom row****: The recall (left) and false alarm (right) rates for the* ***Geodesic*** *(yellow),* ***NUKE*** *(purple) and* ***Semi-Auto*** *(black) segmentations.*

From the plots in figs. 8.11, 8.12 and 8.13 it may be observed that our *Semi-Auto* (black plots) technique consistently produced high quality segmentation for every frame in all three sequences. These high quality segmentations correspond to high recall rates and low false alarm rates for every frame (very close to ground truth segmentations). Our *Geodesic* (yellow plots) approach produced the next most consistent segmentations in terms of quality.

Although our *Semi-Auto* approach can not be used for applications requiring fully automatic segmentations, this approach would appeal to the post-production industry. Currently in post-production houses an Artist manually draws several mattes in order to extract foreground objects. Our technique would significantly reduce to number of mattes that have to be drawn. There is also some merit in allowing the user the control the segmentation process. He can according to his desired standards, actively control the final segmentation.

Recall that we are estimating 83.86% of the mattes in the three sequences. In the next

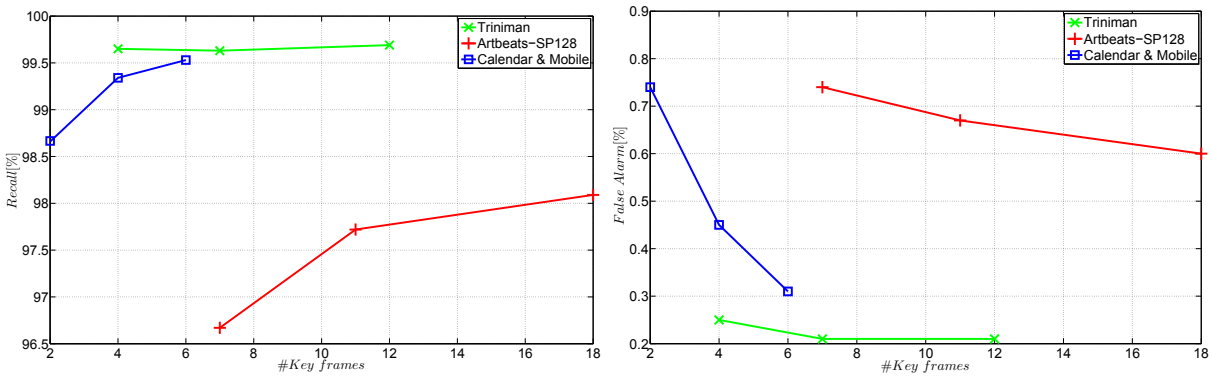

Figure 8.14: *The recall (left) and false alarm (right) rates versus the number of* ***key*** *frames (#Key frames) used in the* ***Semi-Auto*** *segmentations of the* ***Triniman*** *(green plots),* ***Artbeats-SP128*** *(red plots) and* ***Calendar and Mobile*** *(blue plots) sequences.*

section we will show that we can estimate 95.07% of these mattes in our *Semi-Auto* approach and still produce the best segmentations out of all three approaches.

8.4 User Supplied Mattes for Semi-Auto Segmentation

The number of user supplied mattes in our *Semi-Auto* segmentation technique understandably affects the recall and false alarm rates. We conducted three segmentation experiments for each sequence where we vary the number of *key* frames (user supplied mattes). In general a frame is chosen to be a *key* frame if at this frame a significant amount of image material becomes occluded or revealed. Also, the frames for which a new object enters or leaves a scene are selected as *key* frames.

Fig. 8.14 shows the plots of the average recall (left) and false alarm (right) rates for these three experiments. Here the plots for the *Triniman*, *Artbeats-SP128* and *Calendar and Mobile* sequences are coloured *green*, *red* and *blue* respectively.

For the *Triniman* sequence the number of *key* frames used for the three experiments are 4, 7 and 12. The recall and false alarm rates (fig. 8.14) for this sequence stay relatively constant for all three experiments. For the *Artbeats-SP128* and *Calendar and Mobile* sequences where the number of key frames are 7,11,18 and 2,4,6 respectively, the changes in their corresponding rates were within acceptable limits. That is, using the minimum number of *key* frames (7 and 2) for both sequences still produced quality segmentations.

For the *Triniman*, *Artbeats-SP128* and *Calendar and Mobile* sequences the minimum number of *key* frames used in a segmentation experiment were 4, 7 and 2 respectively. The average recall and false alarm rates for all three sequences using these minimum number of *key* frames are 98.33% and 0.58% respectively. Here these rates are better those for the automatic approaches (see tables 8.2 and 8.3); *Geodesic* (95.64%,0.65%) and *NUKE* (79.90%,6.09%).

Note that we estimated 95.07% of the mattes for the three sequences when using the minimum number of *key* frames.

8.5 Comparison with Other Matte Propagation Algorithms: FeatureCut

The *FeatureCut* segmentation algorithm proposed by Ring [100], uses SIFT features matches to propagate a collection of user defined mattes (*key* frames) to the unsegmented frames (*intermediate* frames) in a sequence. Here feature matches are used to identify corresponding image regions between a reference *key* frame and the current *intermediate* frame of interest. These *key* frames contain **foreground/background** labels, and the image region correspondences are used to propagation these labels to the *intermediate* frames. At each pixel site **s** in an *intermediate* frame f, votes are accumulated whether site **s** is foreground or background. The author then proceeds to generate a MAP estimate of the label field for *intermediate* frame f, using a Graphcut solution.

The author of the *FeatureCut* approach provides an implementation of this algorithm as a NUKE [2] plugin. In order to compare our *Semi-Auto* approach with *FeatureCut*, we used this plugin to segment the *Artbeats-SP128* sequence only. We did not attempt to segment any other sequences (*Triniman*, *Calendar and Mobile*) with the *FeatureCut* plugin because it was quite difficult to find the right combination of tweakable parameters (a total of 36 tweakable parameters) in order to produce a reasonable segmentation. Using the default parameter values produces very poor segmentations.

For our comparison with *FeatureCut* on the *Artbeats-SP128* sequence, we supplied both our *Semi-Auto* and the *FeatureCut* approaches with the same set of *key frames* (user defined mattes). The number of *key* frames used was 18, and they are located at frames 1,5,11,25,26,31,34,38,41,46,50, 56,68,74,80,85,90 and 99 (same as presented previously for comparison with *NUKE*).

Fig. 8.15 shows the segmentations produced by *FeatureCut* and our *Semi-Auto* approach for frames 12, 61 and 95 in the *Artbeats-SP128* sequence. Our segmentations are shown in the *second and third* rows, while those for *FeatureCut* are shown in the *fourth and bottom* rows. It may be observed that we produce better segmentations for this sequence compared to the *FeatureCut* approach. We will show later that we have better recall false alarm rates for this sequence.

8.5.1 Reliance on Feature Matches

The general approach of the *FeatureCut* technique is similar to our *Semi-Auto* segmentation technique. We both use feature point correspondences for propagating label information. However, unlike *FeatureCut*, our *Semi-Auto* approach utilizes appearance information. Recall that we model the colour and shape of the objects in a sequence. Also our sparse trajectory seg-

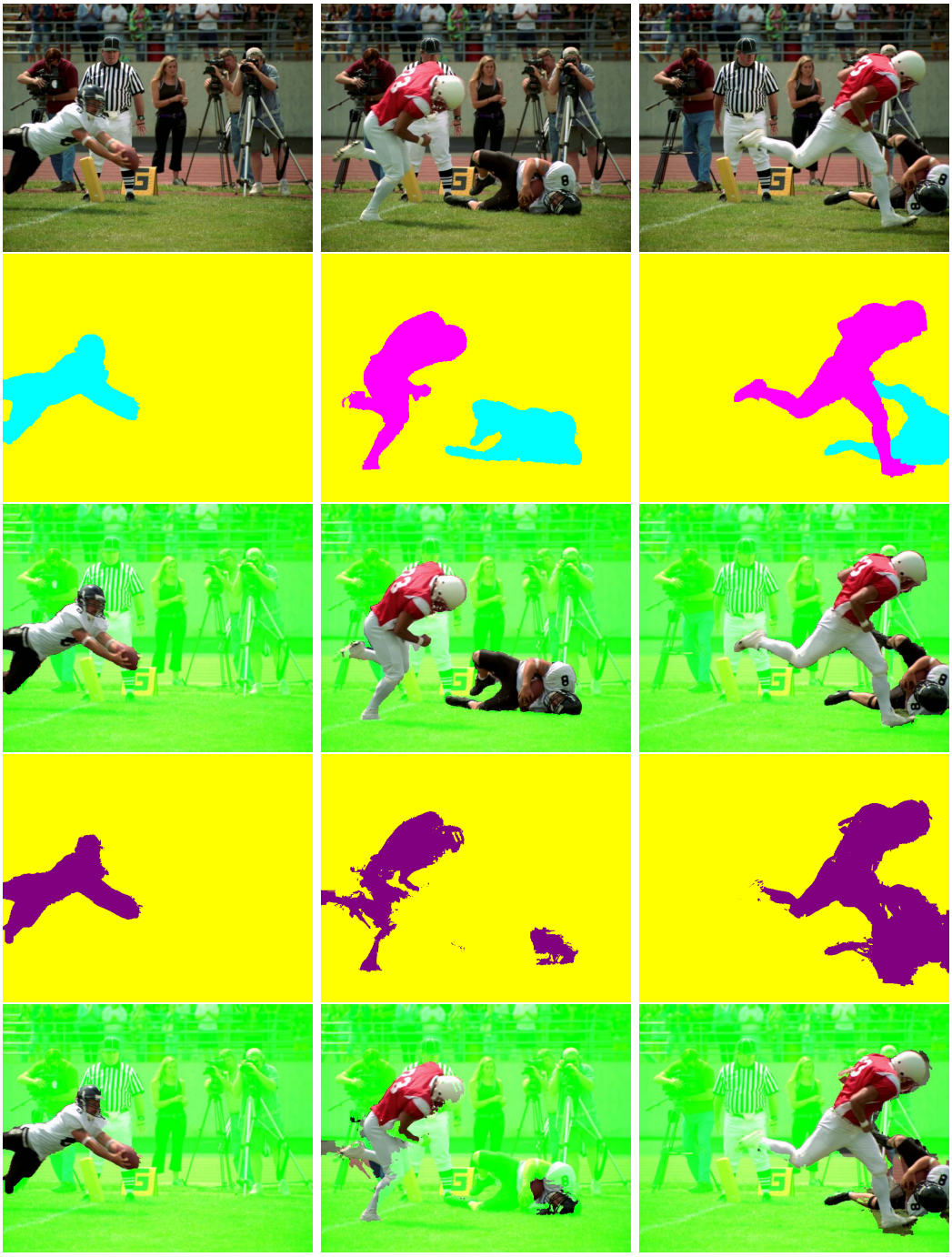

Figure 8.15: *The segmentations for three frames in the* ***Artbeats-SP128*** *sequence.* ***Top row****: Frames* 12, 61 *and* 95 *in this sequence.* ***Second row****: The* ***object*** *level segmentations produced by our* ***Semi-Auto*** *approach. The background, player #1 and #2 are coloured* ***yellow****,* ***cyan*** *and* ***pink*** *respectively.* ***Third and bottom rows****: The* ***foreground/background*** *level segmentations produced by our* ***Semi-Auto*** *and the* ***FeatureCut*** *approaches respectively. The foregrounds are extracted using the respective segmentations and the backgrounds are shaded in green.* ***Fourth row****: The* ***foreground/background*** *level segmentations produced by the* ***FeatureCut*** *algorithm. The background and foreground are coloured* ***yellow*** *and* ***purple*** *respectively.*

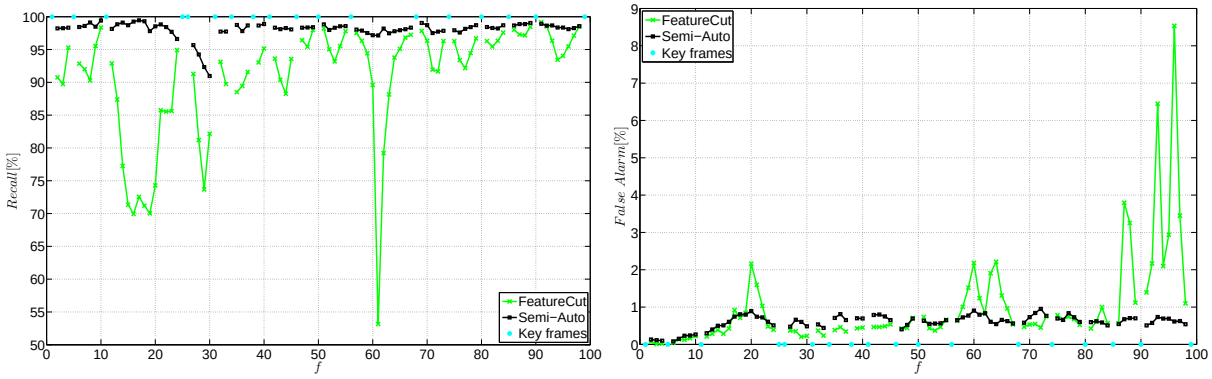

Figure 8.16: *Plots of the recall (Recall[%]) and false alarm (False Alarm[%]) rates per frame (f) for the* ***Artbeats-SP128*** *sequence. The recall (left) and false alarm (right) rates for the* ***FeatureCut*** *(green) and our* ***Semi-Auto*** *(black) segmentations. The frame locations of the* ***key*** *frames are indicated with* ***cyan*** *dots.*

mentation step guarantees that our feature point correspondences are temporally and spatially smooth. However, the equivalent SIFT feature matching step for *FeatureCut* that is used to generate feature correspondences is not temporally and spatially smooth.

Our utilization of appearance information and better feature correspondences in a robust framework allows us to produce more temporally consistent and accurate segmentations for the *Artbeats-SP128* sequence (fig. 8.15) compared to the *FeatureCut* approach. The main issue with the *FeatureCut* approach identified by the author is that the quality of the segmentations is highly dependent on the feature matches between frames. If the current frame to be segmented is temporally far away from any of the user supplied *key* frames, there will not be sufficient feature matches for doing reliable label propagations. Also there will not be any feature matches in low texture image regions.

The segmentation of frame 61 (last two rows of the *middle* column in fig. 8.15) in the *Artbeats-SP128* sequence is an example where *FeatureCut* fails because of unreliable feature matching. Here this approach failed to correctly segment *player* #1 because of a relatively low number of feature matches. Our *Semi-Auto* approach (second and third rows of the *middle* column in fig. 8.15) does not have these shortcomings. While *FeatureCut* uses only SIFT feature correspondences, we are using both KLT and SIFT feature correspondences. Hence we have more feature point correspondences available in our framework.

8.5.2 Recall and False Alarm Rates

Our average recall and false alarm rates for the *Artbeats-SP128* sequence are 98.09% and 0.60% respectively (see tables 8.3 and 8.2). The corresponding rates for *FeatureCut* are 90.93% and 0.99% respectively. Our recall and false alarm rates are better than those for *FeatureCut*, which

is indicative of our overall segmentations being better for this sequence.

Fig. 8.16 shows the recall (left) and false alarm (right) rates per frame for the *Artbeats-SP128* sequence. The plots for the *FeatureCut* and our *Semi-Auto* approach are coloured in *green* and *black* respectively.

The *FeatureCut* approach produces its lowest recall rate of 53.16% for the segmentation of frame 61. The segmentation for this frame is shown is the *middle* column of fig. 8.15. We discussed previously that this poor segmentation for frame 61 is the result of unreliable feature correspondences between the *key* frames and this frame.

The temporal inconsistency of the *FeatureCut* segmentations are reflected in the significant fluctuations in the recall and false alarm rates (fig. 8.16) over the sequence. The rates for our *Semi-Auto* approach are relatively constant over the entire sequence, which is indicative of temporal consistency in the segmentations produced.

8.6 Comparison with Other Matte Propagation Algorithms: SnapCut

The *SnapCut* segmentation algorithm proposed by Bai *et al.* [12] uses optical flow to propagate appearance information from a segmented frame $f-1$ to the current unsegmented frame f (*intermediate* frame). This appearance information consists of colour and shape models for the foreground image regions. These models are used in the design of a posterior distribution for the labels at the current frame f. The author generates a MAP estimate for the *foreground/background* labels for this frame using a Graphcut optimization solution.

For the *SnapCut* segmentation approach, the user fully segments the first frame (1^{st} *key* frame) in a sequence, and all subsequent frames are processed in the order 2 to F (where F is the number of frames in the sequence). The user is allowed to correct any segmentations errors generated for the current frame f, and by default these changes only affect the frames after f (*forward* propagation), i.e. frames $f+1,\ldots,F$. However the user can make the current frame f a *key* frame (reference segmentation), and allow previous *intermediate* frames (2 *to* $f-1$) to be influenced by this *key* frame (*backward* propagation). Recall that we defined an *intermediate* frame as a frame to be labelled in the segmentation process.

An implementation of the *SnapCut* algorithm is bundled into Adobe **After Effects CS5** [1] as a matting tool called *Roto Brush*. Fig. 8.17 shows a screenshot of **After Effects** with the *Roto Brush* tool active. In this screenshot, we are segmenting the *Calendar and Mobile* sequence, and the user supplied *key* frames (orange markers along the time line) are highlighted with *white* ovals. The frames in between these *key* frames are the *intermediate* frames, and the *SnapCut* algorithm is required to generate segmentations for these frames. The segmentation shown currently in fig. 8.17 is for the second frame in the *Calendar and Mobile* sequence.

To make comparisons of our *Semi-Auto* approach to the *SnapCut* approach we used the

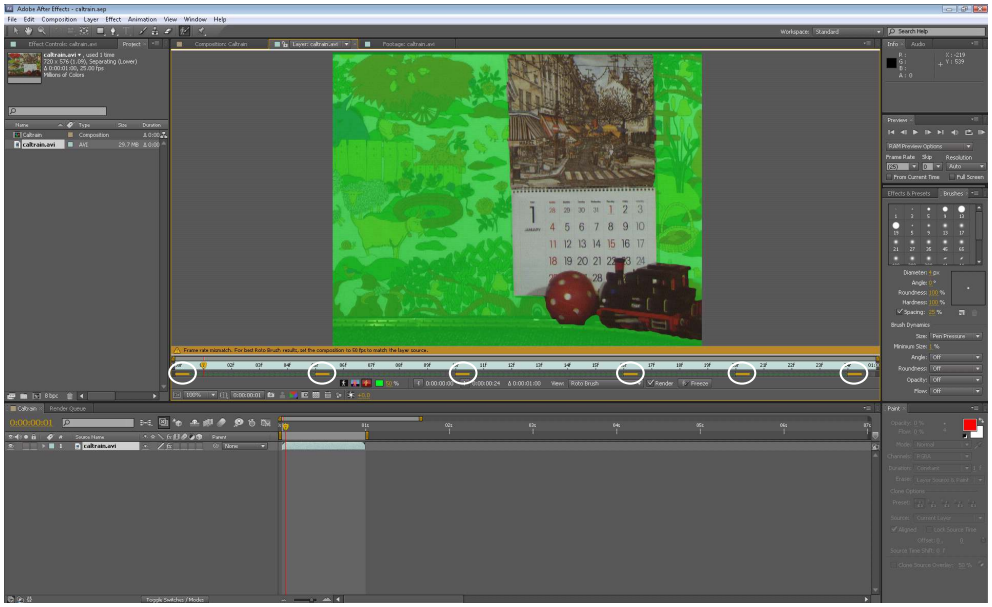

Figure 8.17: *A screenshot of the workspace in* ***Adobe After Effects CS5*** *when using the* ***Roto Brush*** *tool on the* ***Calendar and Mobile*** *sequence. The user defined* ***key*** *frames (orange markers on time line) are highlighted with* ***white*** *ovals.*

	#Key frames	Locations
Triniman	12	01 06 13 20 31 42 53 63 79 85 90 99
Artbeats-SP128	18	01 05 11 25 26 31 34 38 41 46 50 56 68 74 80 85 90 99
Calendar & Mobile	6	01 06 11 17 21 25

Table 8.4: *The number of* ***key*** *frames (#Key frames) and their locations for the* ***Triniman***, ***Artbeats-SP128*** *and* ***Calendar and Mobile*** *sequences.*

Triniman, *Artbeats-SP128* and *Calendar and Mobile* sequences. We supply both approaches with the same set of *key* frames for all three sequences. The locations of these *key* frames for each sequence is shown in the *last* column of table. 8.4. The second column shows the total number of *key* frames for each sequence.

We will show later that our *Semi-Auto* approach has better overall recall and false alarm rates for the three sequences, compared to the *SnapCut* approach. Our approach also produces more temporally consistent segmentations.

	Recall		False Alarm	
Triniman	*SnapCut*	*Semi-Auto*	*SnapCut*	*Semi-Auto*
Average	**99.76**%	99.69%	0.40%	**0.21**%
Median	99.78%	99.69%	0.26%	0.19%
Artbeats-SP128	*SnapCut*	*Semi-Auto*	*SnapCut*	*Semi-Auto*
Average	96.57%	**98.09**%	1.45%	**0.60**%
Median	97.59%	98.33%	1.12%	0.63%
Calendar & Mobile	*SnapCut*	*Semi-Auto*	*SnapCut*	*Semi-Auto*
Average	99.42%	**99.53**%	1.18%	**0.31**%
Median	99.50%	99.54%	1.14%	0.31%
All	*SnapCut*	*Semi-Auto*	*SnapCut*	*Semi-Auto*
Average	98.58%	**99.10**%	1.01%	**0.37**%

Table 8.5: *The average and median recall and false alarm rates for the* ***Triniman****,* ***Artbeats-SP128*** *and* ***Calendar and Mobile*** *sequences. The segmentation approaches shown are* ***SnapCut*** *and* ***Semi-Auto****. The* ***bottom row*** *shows the average rates over all the three sequences.*

8.6.1 Recall and False Alarm Rates

Table 8.5 shows the recall and false alarm rates for the segmentations produced by our *Semi-Auto* and the *SnapCut* approaches on the *Triniman*, *Artbeats-SP128* and *Calendar and Mobile* sequences. Considering all three sequence, our approach has the best overall recall (99.10%) and false alarm (0.37%) rates. The corresponding rates for *SnapCut* are 98.58% and 1.01% (bottom row of table 8.5).

The average recall rate (99.76%) for the segmentations of the *Triniman* sequence produced by *SnapCut* is fractionally better than our *Semi-Auto* approach (99.69%). However our false alarm rate (0.21%) for this sequence is better than the corresponding rate for *SnapCut* (0.40%). Hence we may conclude that both approaches produce roughly the same high quality segmentations for this sequence. We will visually compare the segmentations for both approaches later.

The *Artbeats-SP128* sequence is the most challenging of the three sequences. This sequence contains significantly faster moving objects (motion blur), and the foreground and background colours are not well separated in the colour space. Both our recall (98.09%) and false alarm (0.60%) rates are better than those for *SnapCut* (96.57%,1.45%).

In general, *SnapCut* produces good segmentations when the foreground and background have different colours. If this is the case, the *SnapCut* approach identifies that the foreground and background are well separated in the colour space, and assigns a higher weight to its colour likelihood compared to its shape (geometry) likelihood. For the *Triniman* sequence this is the case, where the foreground is predominantly red and brown, while the background is green.

However, for the *Artbeats-SP128* sequence, the colour likelihood is less reliable, so there is more dependence on the shape likelihood. This shape likelihood is directly dependent on inaccurate optical flow estimations. The design of the shape likelihood is based on the idea of generating an estimated shape for the current frame by propagating the previous segmentation via motion vectors obtained from optical flow. In most cases (especially for non-rigid objects), this estimated shape will differ significantly from the true foreground shape in the current frame. The author of the *SnapCut* approach has identified this issue and has given examples (using *Artbeats-SP128*) of this shape misalignment in his paper [12].

Hence, whenever the *SnapCut* approach places significantly more weight on its shape likelihood, the quality of the segmentations produced deteriorates. This is the case for the *Artbeats-SP128* sequence. Unlike the *SnapCut* approach which utilizes only appearance likelihoods (colour and shape), our *Semi-Auto* approach use both appearance and motion likelihoods. Hence we have more information available which allows us to produce higher quality segmentations. It will be shown later that our segmentations for the *Artbeats-SP128* sequence are visually better than those for *SnapCut*.

Our *Semi-Auto* approach performs better on the *Calendar and Mobile* sequence compared to *SnapCut*. The recall and false alarm rates for our approach are 99.53% and 0.31% respectively. The corresponding rates for *SnapCut* are 99.42% and 1.18% respectively. The significantly higher false alarm rate for *SnapCut* was due to the shadows casted by the ball and the toy train throughout the sequence were consistently being segmented as foreground. Our approach did not have these shortcomings.

8.6.2 Visual Comparison of Segmentations

Fig. 8.18 shows plots of the recall and false alarm rates per frame for the *Triniman*, *Artbeats-SP128* and *Calendar and Mobile* sequences. The plots for the *SnapCut* and our *Semi-Auto* approaches are coloured in *green* and *black* respectively. Using these plots we identify segmentations with relatively low recall or high false alarm rates for doing visual comparisons. As an example, the maximum false alarm rate (2.193%) for a segmentation produced *SnapCut* occurs at frame 52 in the *Triniman* sequence (*top right* in fig. 8.18). Hence we select the segmentations for this frame produced by our *Semi-Auto* and the *SnapCut* approaches for doing visual comparisons.

Table 8.6 shows the frames selected (second column) for visual comparisons in each of three sequences. Also shown are the recall and false alarm rates for the segmentations of these frames produced by our *Semi-Auto* and the *SnapCut* approaches. Here we select two frames for each sequence that have either relatively poor recall or false alarm rates. These poor rates are shown in *red* text. Also we select a third frame for which both segmentation approaches produce roughly the same corresponding rates. The rows in table 8.6 corresponding to these frames are in *green* text. See fig. 8.18 for more verification of the selection of the frames in table 8.6.

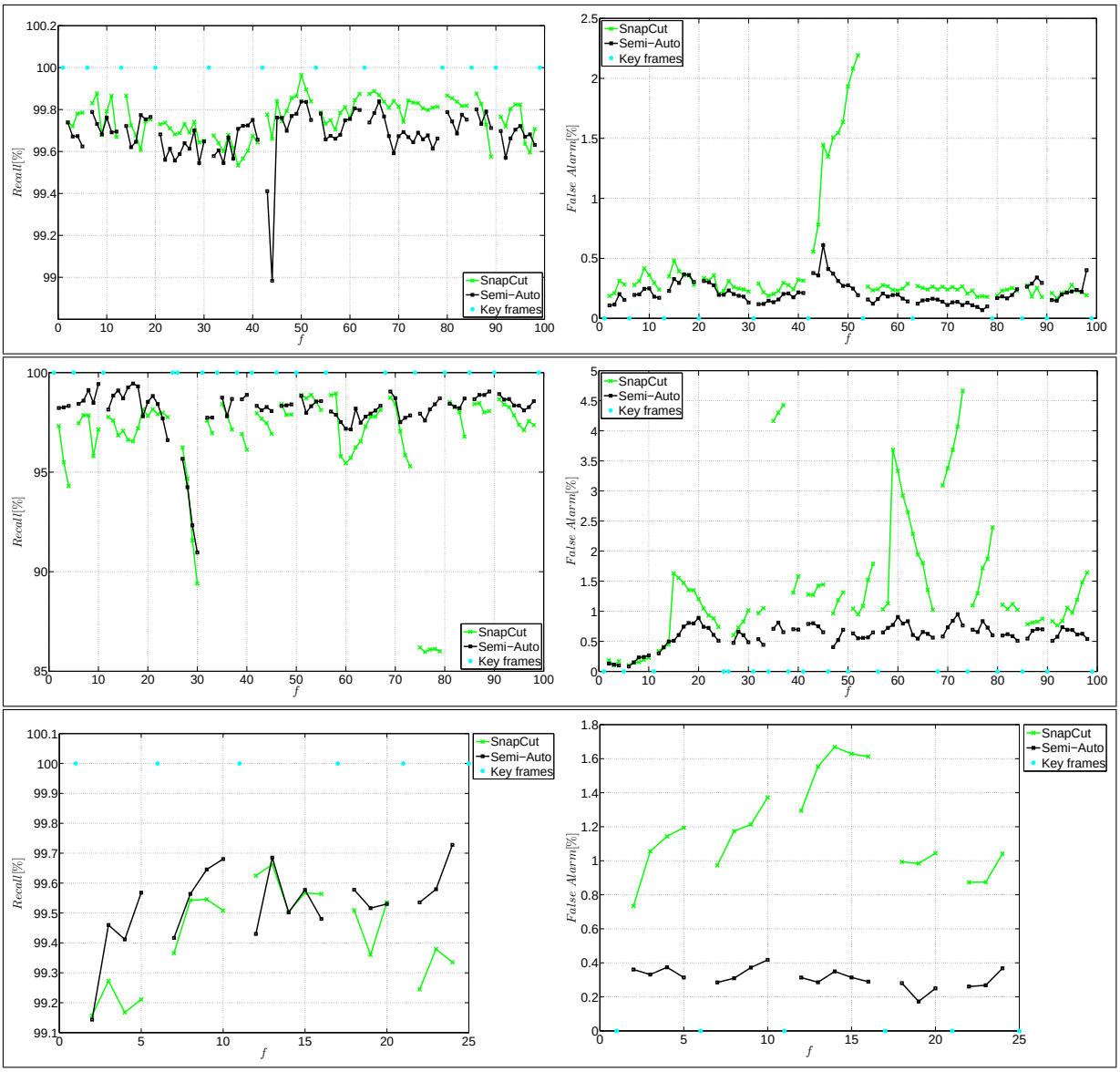

Figure 8.18: *Plots of the recall (Recall[%]) and false alarm (False Alarm[%]) rates per frame (f) for the* ***Triniman*** *(**top row**),* ***Artbeats-SP128*** *(**middle row**) and* ***Calendar and Mobile*** *(**bottom row**) sequences. The recall and false alarm rates are shown in the* ***left*** *and* ***right*** *columns respectively. The plots for the* ***SnapCut*** *and our* ***Semi-Auto*** *segmentations are coloured* ***green*** *and* ***black*** *respectively. The frame locations of the* ***key*** *frames are indicated with* ***cyan*** *dots.*

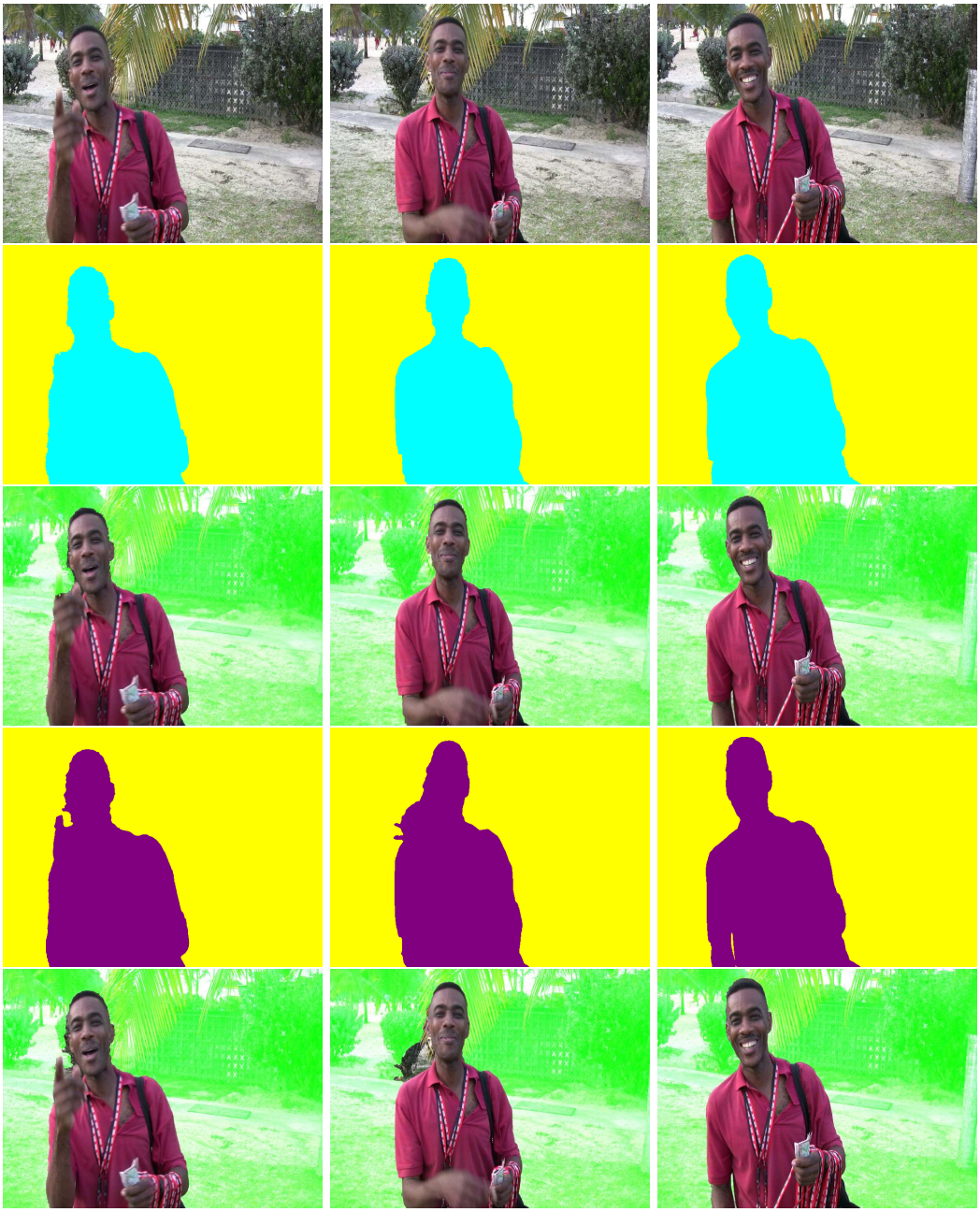

Figure 8.19: *The segmentations for three frames in the* ***Triniman*** *sequence.* ***Top row****: Frames 44, 52 and 86 in this sequence.* ***Second row****: The* ***object*** *level segmentations produced by our* ***Semi-Auto*** *approach. The background, player #1 and #2 are coloured* ***yellow****,* ***cyan*** *and* ***pink*** *respectively.* ***Third and bottom rows****: The* ***foreground/background*** *level segmentations produced by our* ***Semi-Auto*** *and the* ***SnapCut*** *approaches respectively. The foregrounds are extracted using the respective segmentations and the backgrounds are shaded in green.* ***Fourth row****: The* ***foreground/background*** *level segmentations produced by the* ***SnapCut*** *algorithm. The background and foreground are coloured* ***yellow*** *and* ***purple*** *respectively.*

		Recall		False Alarm	
	Frame	*SnapCut*	*Semi-Auto*	*SnapCut*	*Semi-Auto*
Triniman	44	99.66%	98.98%	0.78%	0.36%
	52	99.84%	98.75%	2.19%	0.19%
	86	99.88%	98.80%	0.27%	0.26%
Artbeats-SP128	30	89.40%	90.96%	1.02%	0.49%
	75	86.19%	97.94%	1.10%	0.69%
	91	98.66%	98.92%	0.84%	0.51%
Calendar & Mobile	5	99.21%	99.57%	1.19%	0.31%
	14	99.50%	99.50%	1.67%	0.35%
	24	99.73%	99.34%	1.04%	0.37%

Table 8.6: *The recall and false alarm rates of the frames selected for visual comparisons from the* ***Triniman****,* ***Artbeats-SP128*** *and* ***Calendar and Mobile*** *sequences. Relatively poor recall or false alarm rates are coloured in* ***red****. The frame selected in each sequence for which the corresponding rates for both our* ***Semi-Auto*** *and the* ***SnapCut*** *approaches are roughly same are coloured in* ***green****.*

8.6.2.1 Triniman

Fig. 8.19 shows the segmentations produced by the *SnapCut* and our *Semi-Auto* approach for frames 44, 52 and 86 in the *Triniman* sequence. Our segmentations are shown in the *second and third* rows, while those for *SnapCut* are shown in the *fourth and bottom* rows.

Our recall rate (98.98%) for frame 44 (left column) is less than that of *SnapCut* (99.66%) because we failed to segment the right index finger of the actor as foreground. This is an understandable result as this finger is relatively thin and it is moving very quickly (significant motion blur) in the image plane. Although *SnapCut* correctly labelled the finger as foreground, there is some of the background (on left of the face of the actor) which is incorrectly labelled as foreground. Hence *SnapCut* has a higher false alarm rate (0.78%) for the segmentation of this frame compared to our approach (0.36%).

Both approaches produce roughly the same segmentations for frame 86 shown in the last column of fig. 8.19. This frame represents an example of a frame where the foreground object is moving slowly in the image plane. Since the foreground and background colours are well separated in the colour space, both approaches have reliable appearance likelihoods in order to produce high quality segmentations.

Colour Model Design: The segmentation for frame 52 (middle column of fig. 8.19) reveals an issue with the *SnapCut* approach. This issue is the inability of *SnapCut* to correctly segment background regions that has been quickly revealed (unoccluded). That is, when a foreground ob-

Figure 8.20: *Frames 47 (left), 52 (middle) and 57 (right) in the* ***Triniman*** *sequence. In the frames 47 to 57, the actor in the foreground is moving his upper body quickly from left to right in the image plane.*

ject moves relatively quickly in the image plane, there is a significant amount of the background that becomes revealed. In this case where we have a fast moving foreground object, *SnapCut* can not discriminate well between new background colours and those for the foreground. This is because of the way that *SnapCut* models the colours of the foreground and background.

The *SnapCut* approach models only some of the colours in the foreground image region and a few background colours on the boundary of the foreground. Also, colour models are generated for a finite set of local image patches (rectangular windows of size 80×80 pels) along the contour of the foreground. In each local patch it is assumed that the foreground colours remain roughly the same from frame to frame. However the background colour are continuously changing as the foreground moves over the background. Hence these local colour models are quite naive as to which colours constitute the background, and this reduces the ability of this approach to discriminate between foreground and background.

The segmentation for frame 52 (*bottom two* rows of the middle column in fig. 8.19) produced by *SnapCut* suffers from this colour modelling issue. For this segmentation some of the background that was previously occluded by the foreground is incorrectly labelled as foreground. We show the 5^{th} frame before (47) and after (57) frame 52 in fig. 8.20, where we can observe the background regions that are being revealed as the actor moves in the foreground.

Unlike *SnapCut* which models only the colours along the contour of the foreground, we model all the colours of the foreground and background. Hence we can identify new background colour revealed in unoccluded image region if these colours occur somewhere else in the background. In a sense our colour models combine both local and global colour information. Recall also that we obtain colour models from *forward* and *background* segmentation steps. This provides us with more information for disambiguating occlusions. The motion likelihood in our approach as well contributes to a better final segmentation. Revealed background image regions in most cases follow the motion of the background.

The segmentation for frame 52 produced by our approach is shown in the *second and third* rows of the middle column in fig. 8.19. Unlike *SnapCut*, our approach successfully recovers the

occluded background regions in this frame.

8.6.2.2 Artbeats-SP128

Fig. 8.21 shows the segmentations produced by the *SnapCut* and our *Semi-Auto* approach for frames 30, 75 and 91 in the *Artbeats-SP128* sequence. Our segmentations are shown in the *second and third* rows, while those for *SnapCut* are shown in the *fourth and bottom* rows.

Both approaches have relatively low recall rates for the segmentations of frames 27 to 30 (fig. 8.18) because player #2 (object in foreground) starts to enter the scene in these frames. The motion and appearance of this new foreground object is changing rapidly as it enters the frame from the left of the image plane. Hence both approaches understandable struggle to correctly predict the motion and appearance of this new object.

The left column of fig. 8.21 shows the segmentations for frame 30 which has the lowest corresponding recall rates in the set of frames for which player #2 enters the scene. The recall rates for this frame produced by our *Semi-Auto* and the *SnapCut* approach are 90.96% and 89.40% respectively (see table 8.6). Here it may be observed that both approaches failed to segment as foreground the majority of the helmet for player #2. For player #1 which is completely in the scene at frame 30, both approaches correctly segmented this player as foreground.

The segmentation produced by *SnapCut* for frame 75 is shown in the *bottom two* rows of the *middle* column in fig. 8.21. Here the majority of the right leg of player #2 is incorrectly labelled background by this approach. *SnapCut* probably produced a poor segmentation for this frame because the colour of the pants for both player #2 and the referee in the background are quite similar (both wearing white pants). Recall that in this case since the colour likelihood corresponding to the player #2's legs is not very reliable, *SnapCut* places a higher weight on its shape likelihood. As discussed previously, a greater reliance on this shape likelihood usually leads to poor segmentations. Therefore the poor segmentation for the legs of player #2 could be directly related to an unreliable shape likelihood.

The segmentation produced by our *Semi-Auto* approach for frame 75 is shown in the *second and third* rows of the *middle* column in fig. 8.21. Unlike *SnapCut*, we segmented the right leg of player #2 correctly.

The *end* column of fig. 8.21 shows the segmentations for frame 91 produced by both approaches. Here both approaches produce roughly the same segmentations for this frame. However, *SnapCut* again segmented a small part of the referee's pants as foreground. The reasons here are similar to those for the segmentation of frame 75 previously mentioned.

8.6.2.3 Calendar and Mobile

Fig. 8.22 shows the segmentations produced by the *SnapCut* and our *Semi-Auto* approach for frames 5, 14 and 24 in the *Calendar and Mobile* sequence. Our segmentations are shown in the

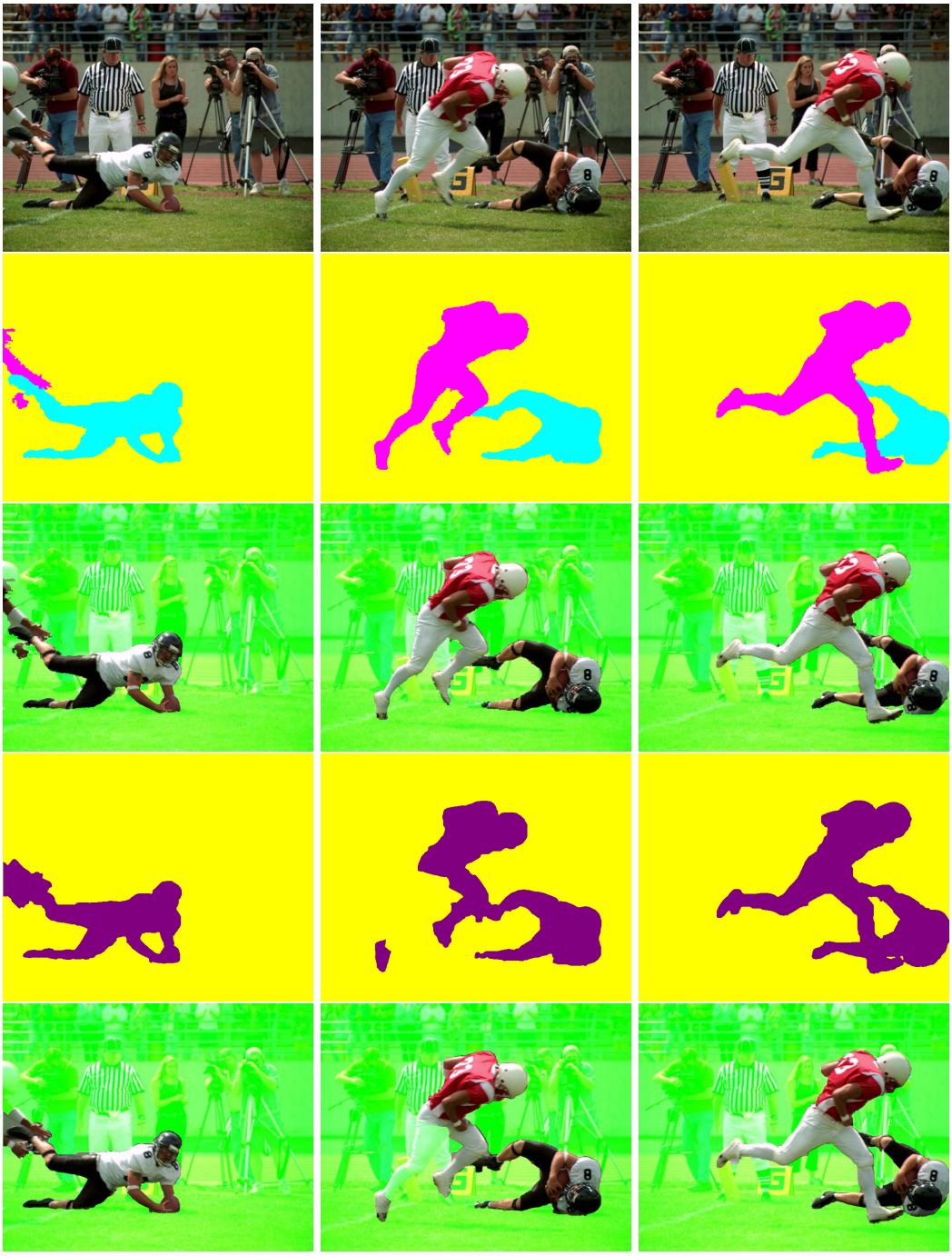

Figure 8.21: *The segmentations for three frames in the* ***Artbeats-SP128*** *sequence.* ***Top row****: Frames* 30, 75 *and* 91 *in this sequence.* ***Second row****: The* ***object*** *level segmentations produced by our* ***Semi-Auto*** *approach. The background, player #1 and #2 are coloured* ***yellow****,* ***cyan*** *and* ***pink*** *respectively.* ***Third and bottom rows****: The* ***foreground/background*** *level segmentations produced by our* ***Semi-Auto*** *and the* ***SnapCut*** *approaches respectively. The foregrounds are extracted using the respective segmentations and the backgrounds are shaded in green.* ***Fourth row****: The* ***foreground/background*** *level segmentations produced by the* ***SnapCut*** *algorithm. The background and foreground are coloured* ***yellow*** *and* ***purple*** *respectively.*

second and third rows, while those for *SnapCut* are shown in the *fourth and bottom* rows.

For frames 5 and 14 (*first and middle* columns), *SnapCut* produced segmentations with false alarm rates that are more than twice the rates corresponding to our approach (table 8.6). This is because in these frames *SnapCut* consistently segmented the gap between the ball and the toy train as foreground. Throughout the sequence, the train casted a shadow in this gap. Hence it is difficult to estimate a reliable colour likelihood in this darkened gap between the ball and the train.

Unlikely *SnapCut*, our approach correctly segmented the gap between the ball and train as background for frame 5 and 14. These segmentations are shown in the *second and third* rows of the *first two* columns in fig. 8.22. The segmentation of these frames demonstrate that our approach can produce robust segmentations in the presence of shadows.

For frame 24 (*end* column in fig. 8.22) the gap between the ball and the toy train is much wider then in frames 5 and 14 previously discussed. Hence *SnapCut* correctly labels this gap as background. However, the false alarm rate (1.04%) for the segmentation produced by *SnapCut* is greater than the corresponding rate for our approach (0.37%).

Overall, we produced better segmentation for all three frames shown in fig. 8.22 for the *Calendar and Mobile* sequence.

8.7 Summary

In this chapter we compared the performances of both our *semi-automatic* and *automatic* dense segmentation techniques with three previous techniques [12, 62, 100]. These previous technique are referred to as *NUKE* [62], *FeatureCut* [100] and *SnapCut*. Recall from chapter 2 that the *NUKE* approach is *unsupervised*, while the *FeatureCut* and *SnapCut* approaches are *supervised*. For these *supervised* techniques, a user is required to supplied fully segmented mattes for some of the frames in a sequence. However, for the *unsupervised* techniques no user assistance is required.

In order to perform quantitative comparisons, we manually generated ground truth segmentations for three sequences called *Triniman* (99 frames), *Artbeats-SP128* (99 frames) and *Calendar and Mobile* (25 frames). We compared both our *semi-automatic* and *automatic* techniques with *NUKE* by looking at recall and false alarm rates for the three sequences. Also we made visual comparisons for selected frames in each sequence. Both our techniques in general produced higher quality segmentations for the three sequences compared to *NUKE*.

We also compared our *supervised* (*semi-automatic*) technique with *FeatureCut* and *SnapCut* since these techniques are all *supervised*. Again, our technique out performed both these techniques.

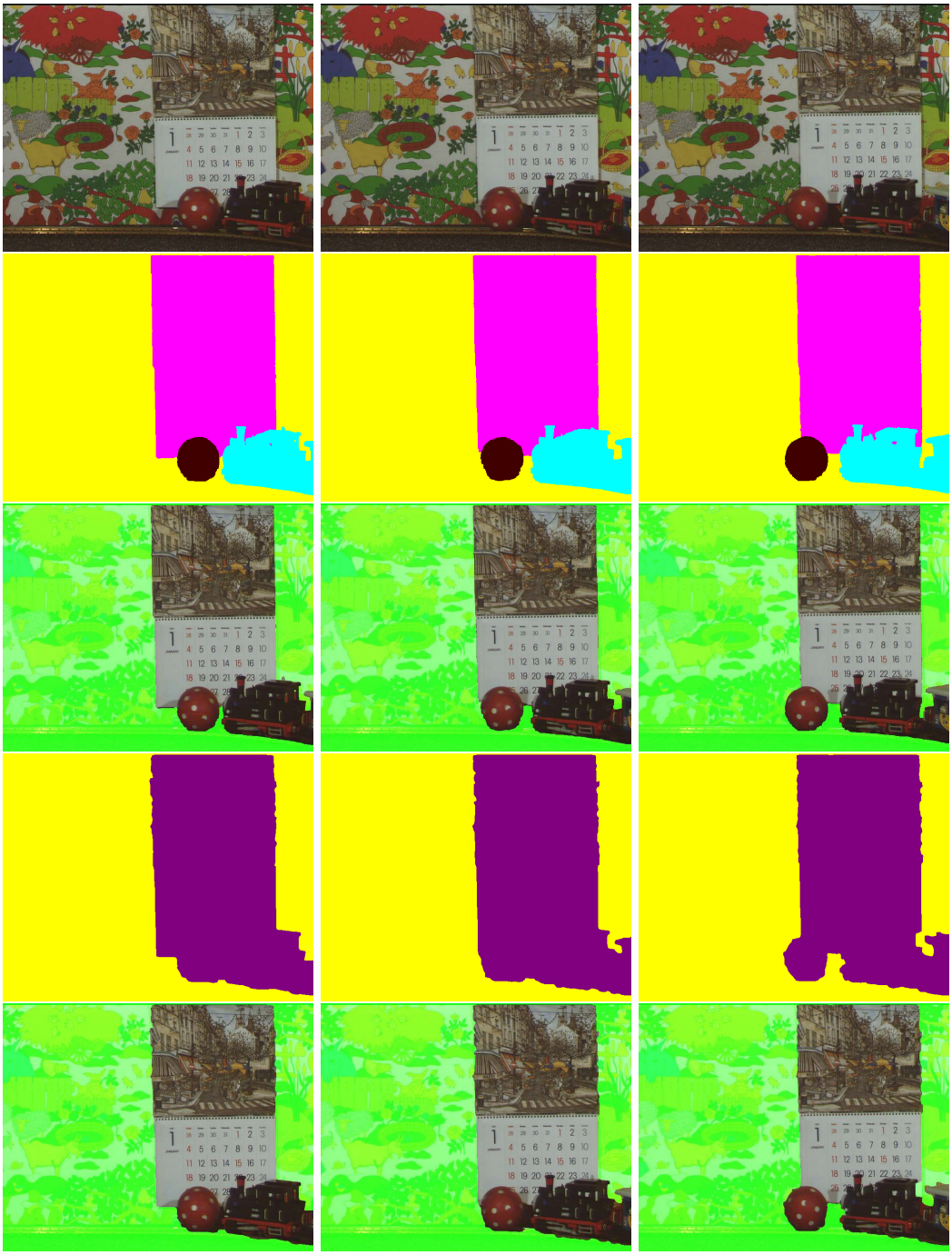

Figure 8.22: *The segmentations for three frames in the* ***Calendar and Mobile*** *sequence.* ***Top row****: Frames* 5, 14 *and* 24 *in this sequence.* ***Second row****: The* ***object*** *level segmentations produced by our* ***Semi-Auto*** *approach. The background, player #1 and #2 are coloured* ***yellow****,* ***cyan*** *and* ***pink*** *respectively.* ***Third and bottom rows****: The* ***foreground/background*** *level segmentations produced by our* ***Semi-Auto*** *and the* ***SnapCut*** *approaches respectively. The foregrounds are extracted using the respective segmentations and the backgrounds are shaded in green.* ***Fourth row****: The* ***foreground/background*** *level segmentations produced by the* ***SnapCut*** *algorithm. The background and foreground are coloured* ***yellow*** *and* ***purple*** *respectively.*

9 Conclusion

This thesis presents my contribution to the development of high quality *video object segmentation* techniques. We proposed a technique for sparse trajectory segmentation, where the feature point trajectories in a sequence are grouped according to their 2D image motions. We define a group of trajectories with similar 2D motion as a *trajectory bundle*. The trajectory bundles produced by our sparse segmentation technique in general represent a single coherent image region, as we enforce spatial and temporal smoothness on these bundles using a Bayesian framework along with a *local region constraint* algorithm. This *local region constraint* algorithm uses Delaunay triangulations to locate groups of connected trajectories in each bundle, and then forms new bundles from the disconnected groups identified. The local motions in a sequence are better described when each trajectory bundle represents a single image region moving through time. This is important for handling the motions of non-rigid objects.

We define a *dense pixel segmentation* as the task of labelling every pixel in a sequence as belonging to a particular object. The major challenge of dense segmentation techniques is producing segmentations that are spatially and temporally consistent over a sequence. An effective way of introducing spatiotemporal smoothness into a dense segmentation process is to utilize long term motion information provided by feature point trajectories. Hence our proposed *dense pixel segmentation* framework utilizes the trajectory bundles obtained from our sparse segmentation to influence the pixel segmentations. We concluded from the review of previous work in chapter 2 that utilizing long term feature point trajectory information leads to improved segmentations for video sequences. Although some techniques [129, 130] prior to our work used trajectories for estimating motion models, we are the first to use the spatial locations of these

trajectories to influence the *dense* pixel segmentation process. We use the spatial locations of the trajectories to propagate appearance information over a sequence.

Since we have long term motion models for the objects in a sequence obtained from the sparse trajectory segmentation step, we can estimate the motion of these objects over several frames. We therefore have a very informative motion likelihood, that considers object occlusions at the various frames in a sequence.

Estimating accurate motion models using feature point trajectories is an important step in our segmentation framework. Hence we propose a framework for tracking local image feature points using the Viterbi [90] algorithm. Our Viterbi tracking framework allows more accurate motion models to be estimated compared to a traditional KLT tracker [13].

Our general dense segmentation framework was modified so a sequence could be segmented automatically or semi-automatically. The semi-automatic technique uses user defined mattes (*key* frame) for some frames in a sequence to estimate appearance models that are then used for segmenting the remaining frames. Hence the user can specify exacted how a sequence should be segmented. However for the automatic segmentation technique no user information is utilized. We therefore estimate mattes for selected *key* frames in a sequence using geodesic distances [111]. Given these estimated *key* frames, we then use our general dense segmentation framework to segment the remaining frames in a sequence.

Our semi-automatic dense segmentation technique may be used for post production applications where high quality segmentations are required. Traditionally an artist would manually segment a sequence to ensure high quality segmentations. Our semi-automatic technique would reduce the number of mattes that would have to be manually segmented, which leads in an increase in productivity.

9.1 Comparisons with State-of-the-art Techniques

Note that we provide an accompanying DVD with videos of the results presented in this thesis. See appendix E for information on how this DVD is organised.

Our proposed *sparse* and *dense* segmentation techniques perform better than previous state-of-the-art techniques, which is proven with thorough quantitative analyses. We compared our *dense pixel segmentation* techniques with techniques implemented in commercial products. One of these techniques is *SnapCut* by Bai *et al.* [12], which is implemented in *Adobe After Effect CS5* [1] as a tool called *Roto Brush*. The other technique proposed by Kokaram [62] is implemented in a software package called *NUKE*. *NUKE* is the leading compositing software package used in the post production industry.

Our *sparse trajectory segmentation* technique produced the best performance on the Hopkins dataset [117] compared to the other state-of-the-art segmentation techniques considered. Providing a quantitative analysis using this dataset is the most popular way the authors of *sparse trajectory segmentation* techniques report their performances. We could not do a quantitative

analysis for the *Region Growing* technique proposed by Pundlik [94], so we did visual comparisons of both our segmentations. Our segmentations were shown to be better, especially for sequences with non-rigid articulated objects.

We also demonstrated that our *sparse* technique could use different types of feature point trajectories. For the majority of the experiments conducted we used KLT [55] and/or SIFT [73] feature point trajectories. However we used particle video trajectories provided by Sand [104] when we visually compared our segmentations with those of Fradet [46]. We did our visual comparisons based on two sequences, where one had roughly rigidly moving objects and the other had significant amounts of non-rigid motions. Our technique was able to improve on the foundational work of Fradet [46] by being able to estimate motion models that describe the motions of the non-rigid objects. Also, the spatiotemporal smoothness strategy included in our technique allowed us to produce better segmentations for sequences with rigidly moving objects compared to the technique of Fradet.

In chapter 3 we presented a new framework for tracking local features using a Viterbi [90] tracking strategy. This general framework allows any type of local image features to be tracked over a sequence, given they can be detected/extracted in every frame. We chose to track SIFT [73] features, and compare the trajectories generated with KLT [55] trajectories. These SIFT trajectories allowed more accurate motion models to be estimated for a sequence compared to KLT trajectories. Accurate motion models are important for generating informative motion likelihoods which are typically used video object segmentation techniques. The improved accuracy of the motion models estimated from SIFT [73] trajectories reflects the effectiveness of the proposed Viterbi tracking framework.

9.2 Future Work

For both our *sparse* and *dense* segmentation steps combined, the average overall computation time is 10 frames per hour in MATLAB. Future work will address developing more computationally efficient implementations of our *video object segmentation* techniques. Although we obtain better segmentations compared to previous technique we did not address the computational efficiency of our techniques. All our techniques were implemented in MATLAB, which made it difficult to assess the speed of these techniques.

We would also like to investigate how different types of trajectories influence the final *dense* segmentations. We usually used a mixture of KLT and SIFT trajectories in our segmentation framework. This allows us to cover the majority of the image plane with trajectories which leads to a better description of the local motion in a sequence. However, it would be interesting to know if only KLT or SIFT could be used. Although SIFT trajectories provide more accurate motion models, they are more computationally costly to generate compared to KLT trajectories. Hence, we would like to know what is the performance penalty for using only KLT trajectories.

9.2.1 User Interaction

In order for our segmentation techniques to be used in real world applications (such as post production) we would require an implementation with a graphical user interface (GUI). This interface should allow the user to specify *key* frames (fully segmented frames) and make corrections to both dense (pixel) and sparse (trajectory) segmentations produced by our techniques. Also incorporating an image segmentation technique such as Grabcut [102] to assist the user in generating *key* frames will be required in the segmentation pipeline. The idea is that utilizing an image segmentation technique reduces the workload of user as he simply has to specify foreground and background paint strokes.

The implementations of our techniques would have to be optimized so the GUI is always responsive to the user. The majority of the computational operations in our techniques are for likelihood estimations. Fortunately, we can reduce the computational time these operations require by executing them in parallel. Modern computers with multicore CPUs/GPUs provide a great platform for implementing these parallel execution strategies.

To make our segmentation system more useful, we would require some sort of performance self-assessment. That is, assess the quality of the segmentations generated and indicate to the user whether his assistance is required. This self-assessment may involve examining if the various likelihoods are informative in some way. For example, given the respective object likelihoods for a segmentation are similar, this may indicate that the resulting segmentation may contain errors.

9.2.2 Stereo 3D Video Object Segmentation

The current popularity in stereo 3D cinema has demanded video segmentation techniques that can handle stereo sequences. Video object segmentations are required for compositing and/or modifying stereo sequences. Segmenting stereo sequences is challenging because the segmentations must be consistent spatially, temporally and also across all camera views. Inconsistencies across camera views will be quite obvious as they break the 3D illusion.

In order for our current segmentation framework to be able to segment stereo sequences, we would have to incorporate more constraints by defining new likelihood terms. For example, a likelihood term based on the depth of objects would be an useful addition to our framework.

9.3 Final Remarks

Although we can perform semi-automatic segmentations for cinema post-production applications, there are other applications such as video compression, video object indexing and visual surveillance that require a fully automatic segmentation process. Therefore in order for automatic techniques to be successfully employed in these applications, common problems in video such as motion blurs, sudden illumination changes (camera flashes) and compression artifacts

must be explicitly addressed. A semi-automatic segmentation technique can also benefit from robust solutions that combat these problems. Even though a user can identify when the segmentation process fails and make the necessary corrections, by handling these problems properly in the segmentation framework the user would be required to make less corrections. This leads to an increase in productivity.

The use of long term trajectory information in our segmentation framework implicitly helps us to cope with relatively small amounts of motion blur and compression artifacts. However the tracking of sparse local features would fail for sequences with sudden illumination changes (such as camera flashes). These illumination changes can be considered to be temporal discontinuities. Hence our current segmentation framework would have to be modified to successfully handle these temporal discontinuities. That is, make correct associations of local feature information before and after these discontinuities. Directly tackling these problems common to video in order to improve the efficiency of the segmentation process can be explored in future work.

A

Graphcut Solution for MAP Estimation: Sparse Trajectory Segmentation

The α-*expansion* Graphcut algorithm [135] is used to solve for the MAP estimate of the trajectory bundle labels $\mathcal{L}_t$. The posterior distribution for the trajectory labels $\mathcal{L}_t$ discussed previously in chapter 5 is repeated below.

$$p(\mathcal{L}_t|\mathcal{X}, \mathcal{L}_{\sim t}) \propto p_x(\mathcal{X}_t|\mathcal{L}_t)p_s(\mathcal{L}_t|\mathcal{L}_{\sim t}) \tag{A.1}$$

The adaptation of the Bayesian problem in eq. A.1 above to this Graphcut solution will be discussed in this section.

The structure of the graph used to generate the MAP estimate for the trajectory bundle labels $\mathcal{L}_t$ is illustrated in the *top* row of fig. A.1. Note that for this illustrated example the number of bundle labels $N = 2$. Hence the bundle labels are $\mathcal{L}_t = 0$ (cyan) and $\mathcal{L}_t = 1$ (pink). The trajectories are nodes in a Graphcut problem, and they are represented by the coloured dots. To do a Graphcut optimization we must specify '*t-link*' and '*n-link*' weights for the nodes/trajectories. A *t-link* weight is the cost for assigning a node a particular label without considering the interactions of neighbouring labels. The *t-links* for assigning the labels $\mathcal{L}_t = 0$ and $\mathcal{L}_t = 1$ are represented by the *cyan* and *pink* arrows (fig. A.1) respectively. The cost we place on these *t-links* are derived from the motion likelihoods.

The cost c_t^n on the *t-link* for the trajectory $\mathcal{X}_t$ with respect to bundle n is defined as follows.

$$c_t^n = -\log\left[p_x(\mathcal{X}_t|\mathcal{L}_t = n)\right] \tag{A.2}$$

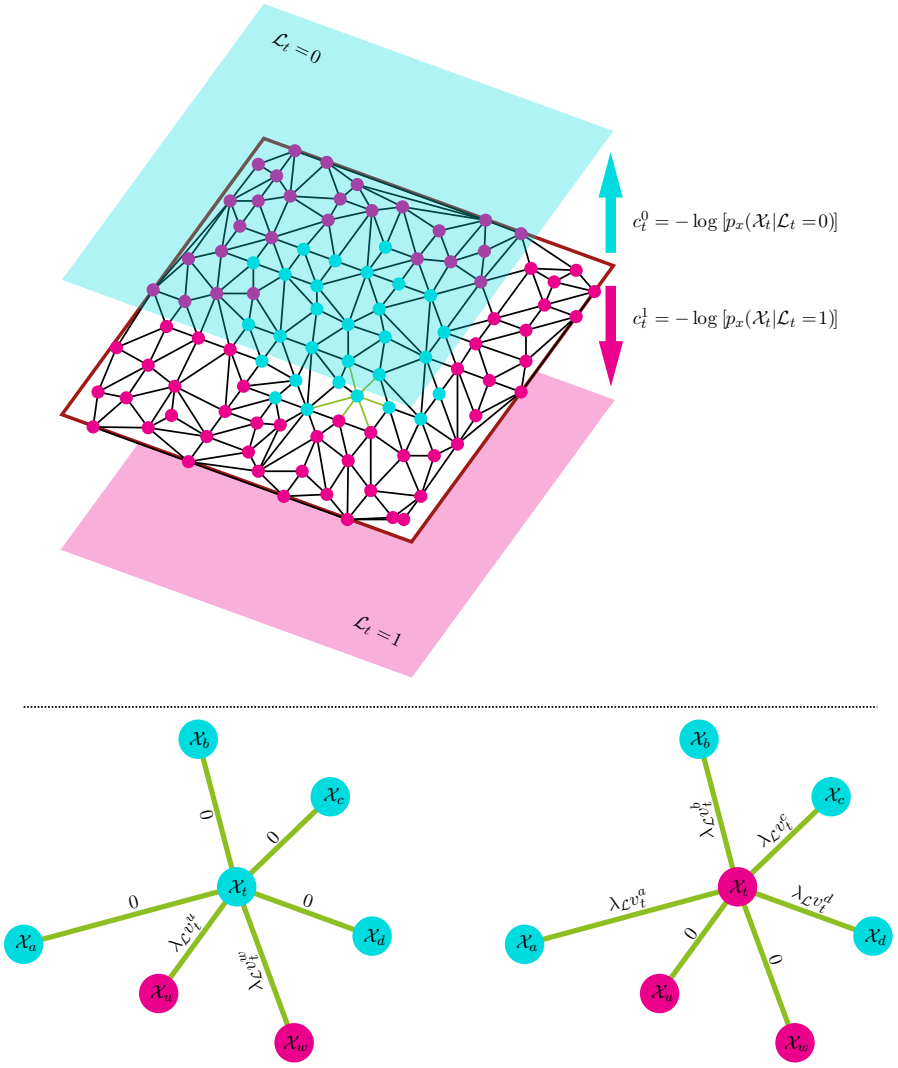

Figure A.1: *The Graphcut solution for generating the MAP estimate for the trajectory bundle labels $\mathcal{L}_t$.* ***Top row****: An example of a graph for two bundle labels $\mathcal{L}_t = 0$ (cyan) and $\mathcal{L}_t = 1$ (pink). The nodes/trajectories are represented by the coloured dots on the grid. The t-links costs c_t^n for assigning the labels $\mathcal{L}_t = 0$ and $\mathcal{L}_t = 1$ are represented by the* ***cyan*** *and* ***pink*** *arrows respectively. The n-links costs $n(\mathcal{X}_t, \mathcal{X}_r)$ are represented by the lines connecting the nodes/trajectories.* ***Bottom row****: The neighbourhood structure (obtained by Delaunay triangulations) for the current trajectory $\mathcal{X}_t$, where the neighbouring trajectories are $\mathcal{X}_a$, $\mathcal{X}_b$, $\mathcal{X}_c$, $\mathcal{X}_d$, $\mathcal{X}_u$ and $\mathcal{X}_w$. The* ***left*** *and* ***right*** *illustrations shows the various* ***n-link*** *costs $n(\mathcal{X}_t, \mathcal{X}_r)$ when the bundle label for trajectory $\mathcal{X}_t$ is $\mathcal{L}_t = 0$ (cyan dot) and $\mathcal{L}_t = 1$ (pink dot) respectively.*

Where $p_x(\mathcal{X}_t|\mathcal{L}_t = n)$ is the motion likelihood for trajectory $\mathcal{X}_t$ assuming the nth bundle label. The *t-link* costs in a Graphcut optimization must be greater than or equal to zero, hence c_t^n is the negative natural logarithm of the likelihood. Here a 'high' likelihood correspond to a 'low'

t-link cost c_t^n.

The *n-link* weights control the costs associated with the label configurations in the local trajectory neighbourhoods. Hence our *n-link* weights are based on the MRF smoothness prior $p_s(.)$. The *n-links* are represented in fig. A.1 by the lines connecting the nodes/pixel sites. The *bottom* row of fig. A.1 illustrates the Delaunay neighbourhood structure used in our Graphcut solution. The neighbours for the current trajectory $\mathcal{X}_t$ are trajectories $\mathcal{X}_a$, $\mathcal{X}_b$, $\mathcal{X}_c$, $\mathcal{X}_d$, $\mathcal{X}_u$ and $\mathcal{X}_w$. Recall from the definition of the prior distribution $p_s(.)$ in eq. 5.13 that there is a penalty of $\lambda_L v_t^c$ if the current label $\mathcal{L}_t$ differs from the neighbouring label $\mathcal{L}_c$. The *n-link* cost $\eta(\mathcal{X}_t, \mathcal{X}_c)$ between trajectory $\mathcal{X}_t$ and the neighbouring trajectory $\mathcal{X}_c$ is defined as follows.

$$\eta(\mathcal{X}_t, \mathcal{X}_c) = \begin{cases} \lambda_L v_t^c, & \textbf{if } \mathcal{L}_t \neq \mathcal{L}_c \\ 0, & \textbf{else} \end{cases} \tag{A.3}$$

The *left* and *right* of the *bottom* row of fig. A.1 illustrate the *n-link* costs $\eta(\mathcal{X}_t, \mathcal{X}_c)$ when the trajectory $\mathcal{X}_t$ has the bundle label $\mathcal{L}_t = 0$ and $\mathcal{L}_t = 1$ respectively. Here the neighbouring trajectory labels are $\mathcal{L}_a = 0$, $\mathcal{L}_b = 0$, $\mathcal{L}_c = 0$, $\mathcal{L}_d = 0$, $\mathcal{L}_u = 1$ and $\mathcal{L}_w = 1$.

B

Graphcut Solution for MAP Estimation: Dense Pixel Segmentation

The α-*expansion* Graphcut algorithm [135] is used to solve for the MAP estimate of the object label field $\mathcal{L}_f$ for the current frame f. The posterior distribution for the object labels $\mathcal{L}_f(\mathbf{s})$ discussed previously is repeated below.

$$p(\mathcal{L}_f(\mathbf{s})|\mathcal{X}, \mathbf{I}, \mathcal{L}_f(\sim \mathbf{s})) \propto p_x(\mathcal{X}(f, \mathbf{s})|\mathcal{L}_f(\mathbf{s}))p_i(\mathbf{I}_f(\mathbf{s})|\mathcal{L}_f(\mathbf{s}))p_s(\mathcal{L}_f(\mathbf{s})|\mathcal{L}_f(\sim \mathbf{s})) \tag{B.1}$$

The adaptation of the Bayesian problem in eq. B.1 above to this Graphcut solution will be discussed in this section.

The structure of the graph used to generate the MAP estimate for the label field $\mathcal{L}_f$ is illustrated in the *top* row of fig. B.1. Note that for this illustrated example the number of object labels $N = 2$. Hence the object labels are $\mathcal{L}_f(\mathbf{s}) = 1$ (cyan) and $\mathcal{L}_f(\mathbf{s}) = 0$ (pink). The pixel sites are nodes in a Graphcut problem, and they are represented by the coloured dots. To do a Graphcut optimization we must specify '*t-link*' and '*n-link*' weights for the nodes/pixel sites. A *t-link* weight is the cost for assigning a node a particular label without considering the interactions of neighbouring labels. The *t-links* for assigning the labels $\mathcal{L}_f(\mathbf{s}) = 1$ and $\mathcal{L}_f(\mathbf{s}) = 0$ are represented by the *cyan* and *pink* arrows respectively. The cost we place on these *t-links* are derived from the motion and appearance likelihoods.

The cost $t_f^o(\mathbf{s})$ on the *t-link* for the pixel site $(f, \mathbf{s})$ with respect to object o is defined as follows.

$$t_f^o(\mathbf{s}) = -\left\{log\left[p_x(\mathcal{X}(f, \mathbf{s})|\mathcal{L}_f(\mathbf{s}) = o)\right] + \log\left[p_i(\mathbf{I}_f(\mathbf{s})|\mathcal{L}_f(\mathbf{s}) = o)\right]\right\} \tag{B.2}$$

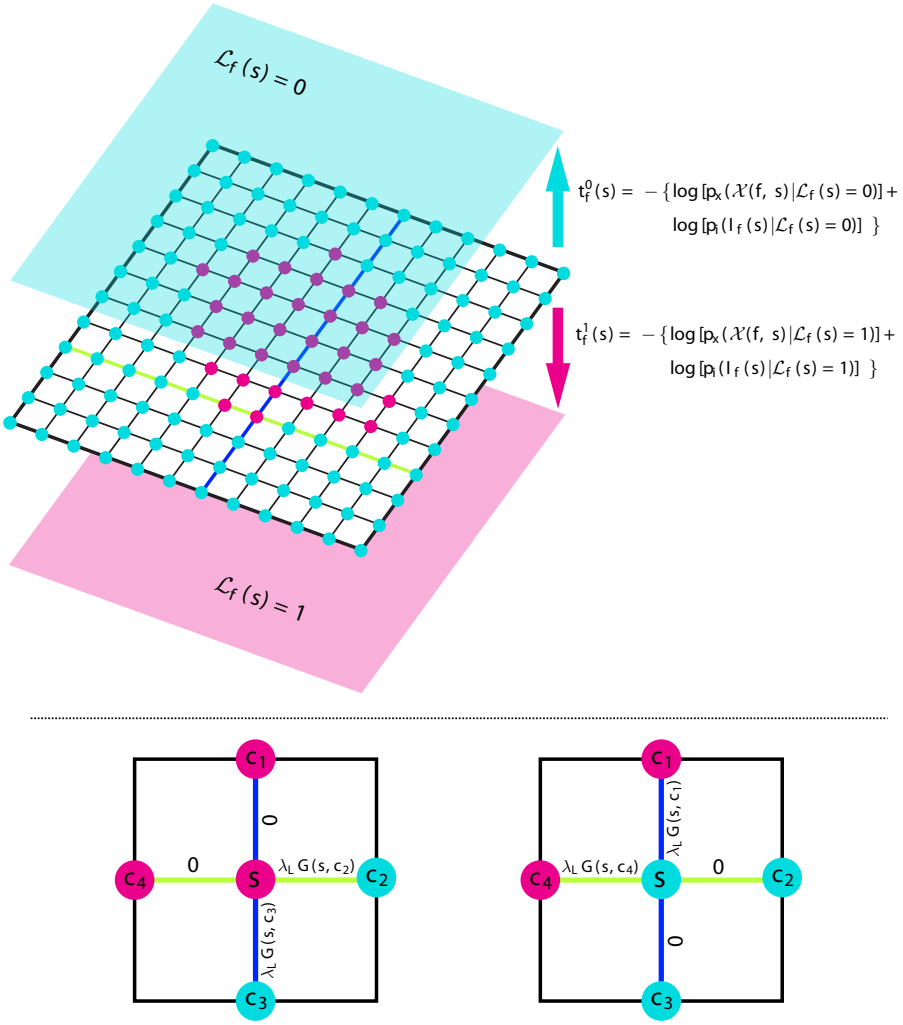

Figure B.1: *The Graphcut solution for generating the MAP estimate for the object label field* $\mathcal{L}_f$. ***Top row****: An example of a graph for two object labels* $\mathcal{L}_f(\mathbf{s}) = 0$ *(cyan) and* $\mathcal{L}_f(\mathbf{s}) = 1$ *(pink). The nodes/pixel sites are the coloured dots on the grid. The t-links costs* $t_f^o(\mathbf{s})$ *for assigning the labels* $\mathcal{L}_f(\mathbf{s}) = 0$ *and* $\mathcal{L}_f(\mathbf{s}) = 1$ *are represented by the* ***cyan*** *and* ***pink*** *arrows respectively. The n-links costs* $n_f(\mathbf{s}, \mathbf{c})$ *are represented by the lines connecting the nodes/pixel sites.* ***Bottom row****: The 4-connected neighbourhood structure for the current site* $\mathbf{s}$*, where the neighbouring sites are* $\mathbf{c}_1$*,* $\mathbf{c}_2$*,* $\mathbf{c}_3$ *and* $\mathbf{c}_4$*. The* ***left*** *and* ***right*** *illustrations shows the various* ***n-link*** *costs* $n_f(\mathbf{s}, \mathbf{c})$ *when the object label for site* $\mathbf{s}$ *is* $\mathcal{L}_f(\mathbf{s}) = 0$ *(cyan dot) and* $\mathcal{L}_f(\mathbf{s}) = 1$ *(pink dot) respectively.*

Where $p_x(\mathcal{X}(f, \mathbf{s})|\mathcal{L}_f(\mathbf{s}) = o)$ and $p_i(\mathbf{I}_f(\mathbf{s})|\mathcal{L}_f(\mathbf{s}) = o)$ are the motion and appearance likelihoods for site $(f, \mathbf{s})$ respectively. The *t-link* costs in a Graphcut optimization must be greater than or equal to zero, hence $t_f^o(\mathbf{s})$ is the negative natural logarithm of the likelihoods. Here 'high' likelihoods correspond to 'low' *t-link* costs $t_f^o(\mathbf{s})$.

The *n-link* weights control the costs associated with the label configurations in the local pixel neighbourhoods. Hence our *n-link* weights are based on the MRF smoothness prior $p_s(.)$. The *n-link* are represented in fig. B.1 by the lines connecting the nodes/pixel sites. The *bottom* row of fig. B.1 illustrates the 4-connected neighbourhood structure used in our Graphcut solution. The neighbours for the current site $\mathbf{s}$ are sites $\mathbf{c}_1$, $\mathbf{c}_2$, $\mathbf{c}_3$ and $\mathbf{c}_4$. Recall from the definition of the prior distribution $p_s(.)$ in eq. 7.30 that there is a penalty of $\lambda_L G(\mathbf{s}, \mathbf{c})$ if the current label $\mathcal{L}_f(\mathbf{s})$ differs from the neighbouring label $\mathcal{L}_f(\mathbf{c})$. The *n-link* cost $n_f(\mathbf{s}, \mathbf{c})$ between pixel site $(f, \mathbf{s})$ and the neighbouring site $(f, \mathbf{c})$ is defined as follows.

$$n_f(\mathbf{s}, \mathbf{c}) = \begin{cases} \lambda_L G(\mathbf{s}, \mathbf{c}), & \textbf{if } \mathcal{L}_f(\mathbf{s}) \neq \mathcal{L}_f(\mathbf{c}) \\ 0, & \textbf{else} \end{cases} \tag{B.3}$$

The *left* and *right* of the *bottom* row of fig. B.1 illustrate the *n-link* costs $n_f(\mathbf{s}, \mathbf{c})$ when the site $\mathbf{s}$ has the object label $\mathcal{L}_f(\mathbf{s}) = 0$ and $\mathcal{L}_f(\mathbf{s}) = 1$ respectively. Here the neighbouring object labels are $\mathcal{L}_f(\mathbf{c}_1) = 1$, $\mathcal{L}_f(\mathbf{c}_2) = 0$, $\mathcal{L}_f(\mathbf{c}_3) = 0$ and $\mathcal{L}_f(\mathbf{c}_4) = 1$.

C

Propagating Trajectory Sub-bundle Labels

Consider that there are S_n trajectory sub-bundles in bundle n. Every trajectory $\mathcal{X}_t$ in bundle n is given a label $\mathcal{S}_t \in \{0 : S_n\}$ according to the sub-bundle it belongs to. The *bottom row* of fig. C.1 shows 3 sub-bundles where their trajectories have sub-bundle labels $\mathcal{S}_t = 1$ (red), $\mathcal{S}_t = 3$ (purple), and $\mathcal{S}_t = 0$ (yellow).

Here a trajectory $\mathcal{X}_t$ has label $\mathcal{S}_t = 0$ if $\mathcal{X}_t$ is a member of a sub-bundle with less than three trajectories. The yellow trajectory in the *bottom row* of fig. C.1 has a label $\mathcal{S}_t = 0$ since it is the only trajectory in its sub-bundle. Trajectories with sub-bundle label $\mathcal{S}_t = 0$ are not used to form new bundles since they reside in a sub-bundle with less that three trajectories.

The trajectories are labelled in a forward and backward label propagation process. We first label the trajectories in the forward direction going from the starting frame of the bundle a_n to the end frame b_n as shown in the *top row* of fig. C.1. For this example, the sub-bundle labels at the starting frame a_n are $\mathcal{S}_t = 1$ (red) and $\mathcal{S}_t = 2$ (green), corresponding to the number of groups of connected trajectory points. These groups are outlined with black ellipses.

For a new sub-bundle (purple) discovered at frame f_s, the sub-bundle label is 3. This label is one more than the maximum label (2) up to frame f_s. All new sub-bundles discovered going through the frame sequence are assigned labels in this manner.

We require that at least two trajectory points are connected at the starting frame f_s of a sub-bundle for this sub-bundle to get a label. If this is not the case, the trajectories in the sub-bundle at frame f_s are labelled $\mathcal{S}_t = 0$ as previously discussed.

An example of a sub-bundle with less than two trajectories is highlighted with a purple closed contour in the *top row* of fig. C.1. This sub-bundle has a single trajectory which later

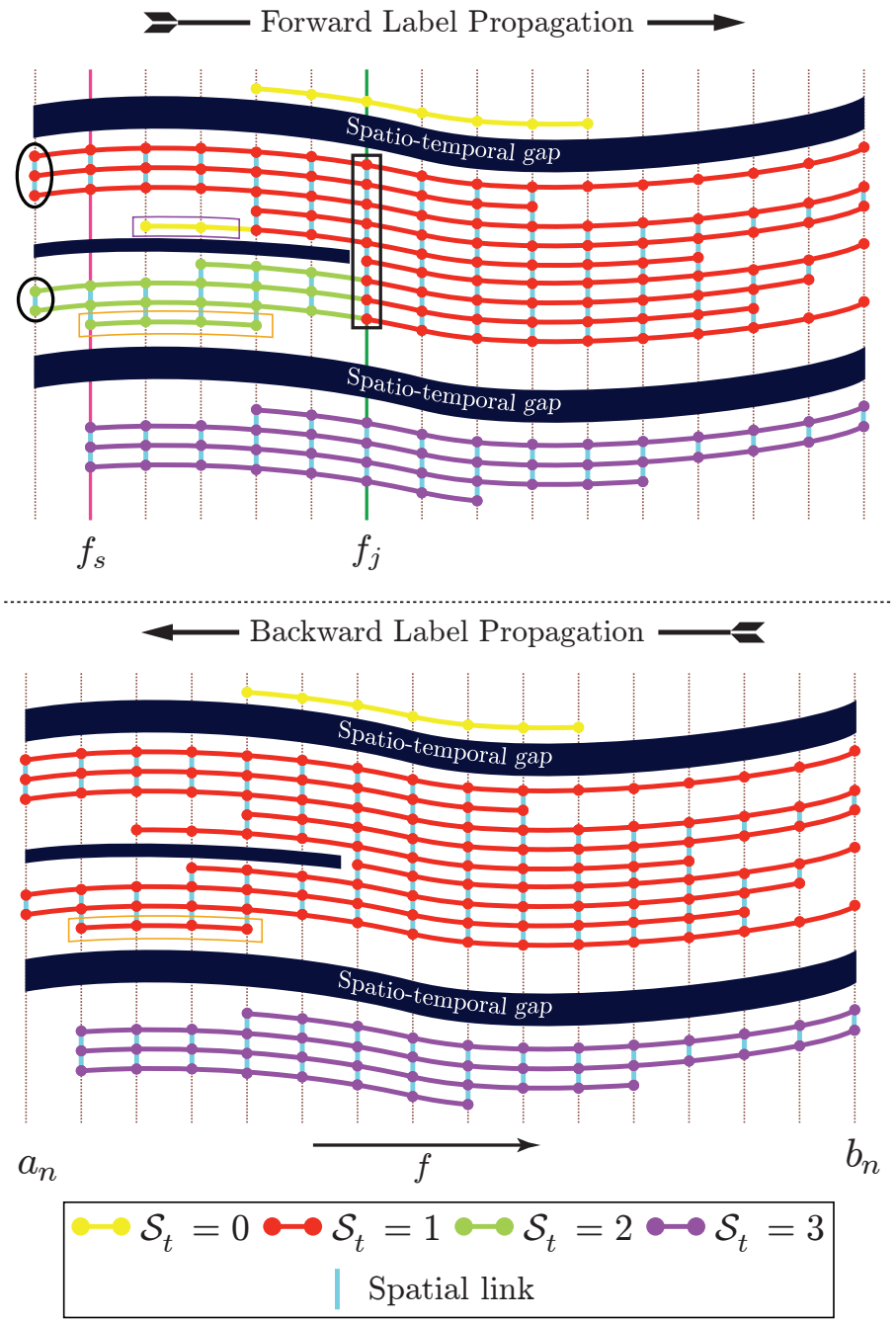

Figure C.1: *The propagation of the 3 sub-bundle labels to the trajectories in a particular bundle. For a trajectory $\mathcal{X}_t$ the sub-bundle labels is defined as $\mathcal{S}_t$.* ***Top row****: The propagation of the labels in the forward direction from the starting frame a_n to the end frame b_n. The groups of connected trajectory points at frame a_n determines the initial number of sub-bundles, and these groups are outlined with black ellipses. The 3^{rd} sub-bundle (purple) starts at frame f_s. At frame f_j the green and red sub-bundles are joined by the spatial links (**cyan** lines) and the connection is outlined with a black box.* ***Bottom row****: The propagation of the sub-bundle labels in the backward direction (frame b_n to a_n), performed after the forward propagation step. The trajectory outlined with the **orange** closed contour is correctly labelled as belonging to the red sub-bundle in this step.*

joins with sub-bundle 1 (red).

The sub-bundle labels are propagated from frame to frame by the temporal links of the trajectories. Note that at a junction frame like f_j the trajectory points with various sub-bundle labels (1,2) are all connected by the spatial links (outlined with black box). At this frame f_j, the two sub-bundles that emerged from the start frame a_n are now merged at f_j. Therefore the new label propagated by the trajectories in the merged sub-bundle is the minimum of the non-zero labels at the junction frame f_j. This minimum label is $\mathcal{S}_t = 1$ (red).

The labels discovered from the previous forward propagation are at this stage propagated backward as shown in the *bottom row* of fig. C.1. The purpose of this backward propagation step is to correctly assign sub-bundle labels to trajectories that ended before a junction frame. An example is outlined with an *orange* closed contour in fig. C.1.

The number of sub-bundles S_n in bundle n is the number of unique bundle labels that remain after the final backward label propagation step. For the example in fig. C.1 the number of bundles is 2, and the unique labels are $\mathcal{S}_t = 1$ and $\mathcal{S}_t = 3$. Each trajectory $\mathcal{X}_t$ in bundle n after the backward label propagation step has a label $\mathcal{S}_t$ according to the sub-bundle it belongs to.

Fig. C.2 shows four bundles at frame 12 in the *Calendar and Mobile* sequence, where each sub-bundle is indicated with a different colour. The background, calendar, and train bundles at this frame all have one sub-bundle (red trajectory points only). The background bundle (top left) has two sets of trajectories on the left and right of the calendar that are connected by spatial links at some frames in the sequence. The locations of these spatial links are generally in the image region outlined with the white contour.

The bundle shown in the *bottom left* of fig. C.2 contains trajectories corresponding to the background and calendar image regions. The 7 sub-bundles however all represent separate regions as required.

The next section discusses how the groups of connected trajectory points at each frame are identified.

C.1 Spatial Connections of Trajectory Points

The left illustration in the *top row* of fig. C.3 shows the trajectory points belonging to bundle n as *orange* dots, while the trajectory points for other bundles as *dark blue* dots. The spatial links (black lines) between these points are obtained from Delaunay triangulations.

The trajectory points (*orange*) in bundle n can be observed to form spatially separated groups of connected points. The connections between these points are defined by the spatial links (Delaunay triangulations). The right illustration in the *top row* shows each group of connected points with a different colour. Here, the yellow points signify isolated trajectory points. Isolated points are not considered as a group, so therefore for this example there are 3 groups of connected trajectory points.

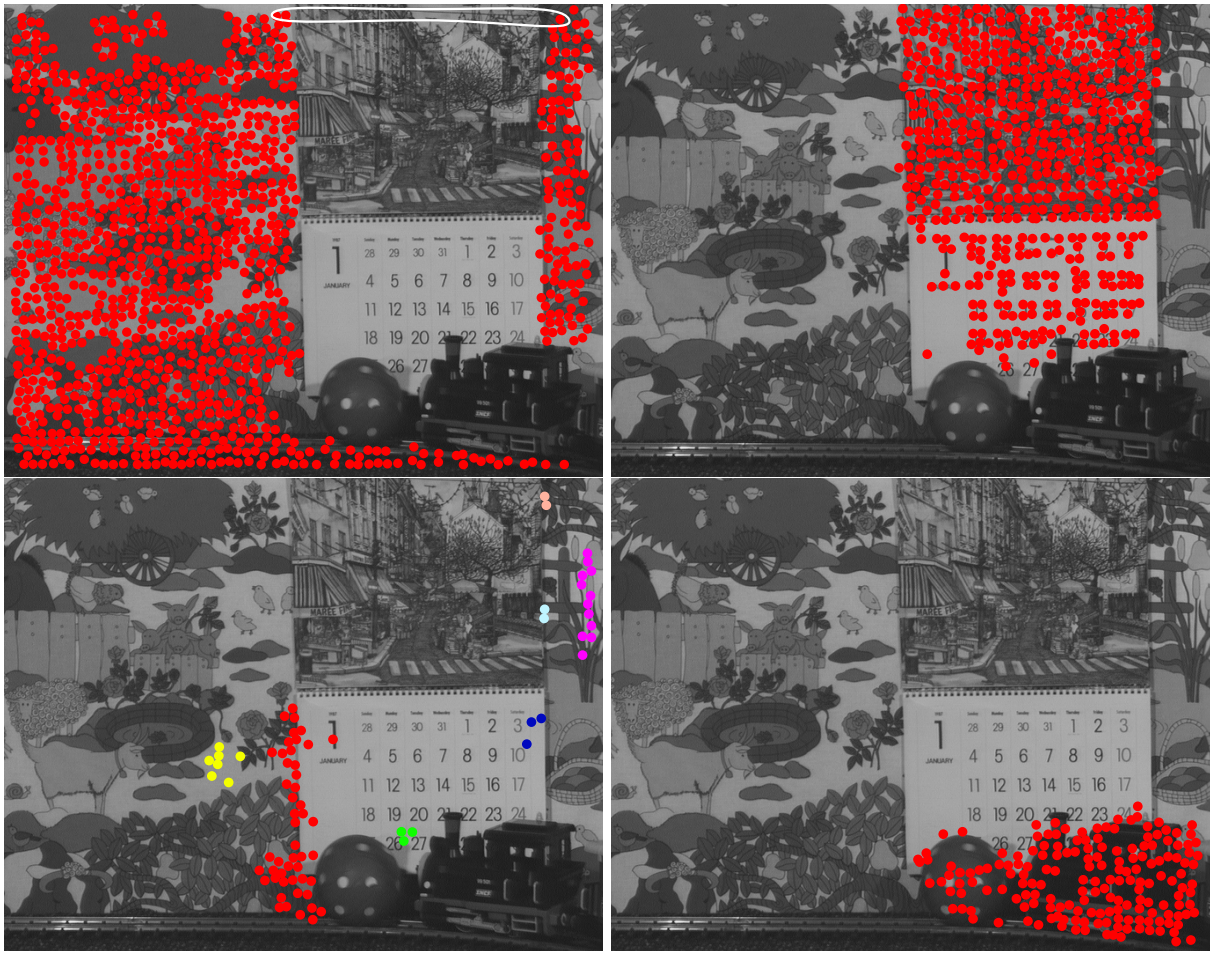

Figure C.2: *Four bundles that exist at frame 12 in the* ***Calendar and Mobile*** *sequence. All the trajectory points (dots) belonging to a particular sub-bundle share the same colour.* ***Top left****: The background bundle has only one sub-bundle even though it has two sets of spatially separated trajectories on both sides of the calendar. At some frames in the sequence both set of trajectories are connected by spatial links in the general location outlined by the white closed contour.* ***Bottom left****: This bundle has a mixture of trajectories in the background and calendar image regions. The 7 sub-bundles however represent separate image regions.* ***Right column****: The calendar (top) and train (bottom) bundles both have a single sub-bundle (red).*

Consider that there are C_f^n groups of connected trajectory points at frame f for the trajectories in bundle n. We require that trajectory $\mathcal{X}_t$ in bundle n be given a connection label $\mathcal{C}_f^t \in \{0 : C_f^n\}$ according to the group of connected points it belongs to. The connection label $\mathcal{C}_f^t = 0$ means that the trajectory point (x_f^t, y_f^t) is isolated as previously mentioned (yellow point in fig. C.3). Hence if trajectories $\mathcal{X}_a$ and $\mathcal{X}_b$ both belong to bundle n and have points at frame f, they are in the same group if $\mathcal{C}_f^a = \mathcal{C}_f^b \neq 0$.

In the example (fig. C.3) the trajectory points have connection labels $\mathcal{C}_f^t = 1$ (green), $\mathcal{C}_f^t = 2$

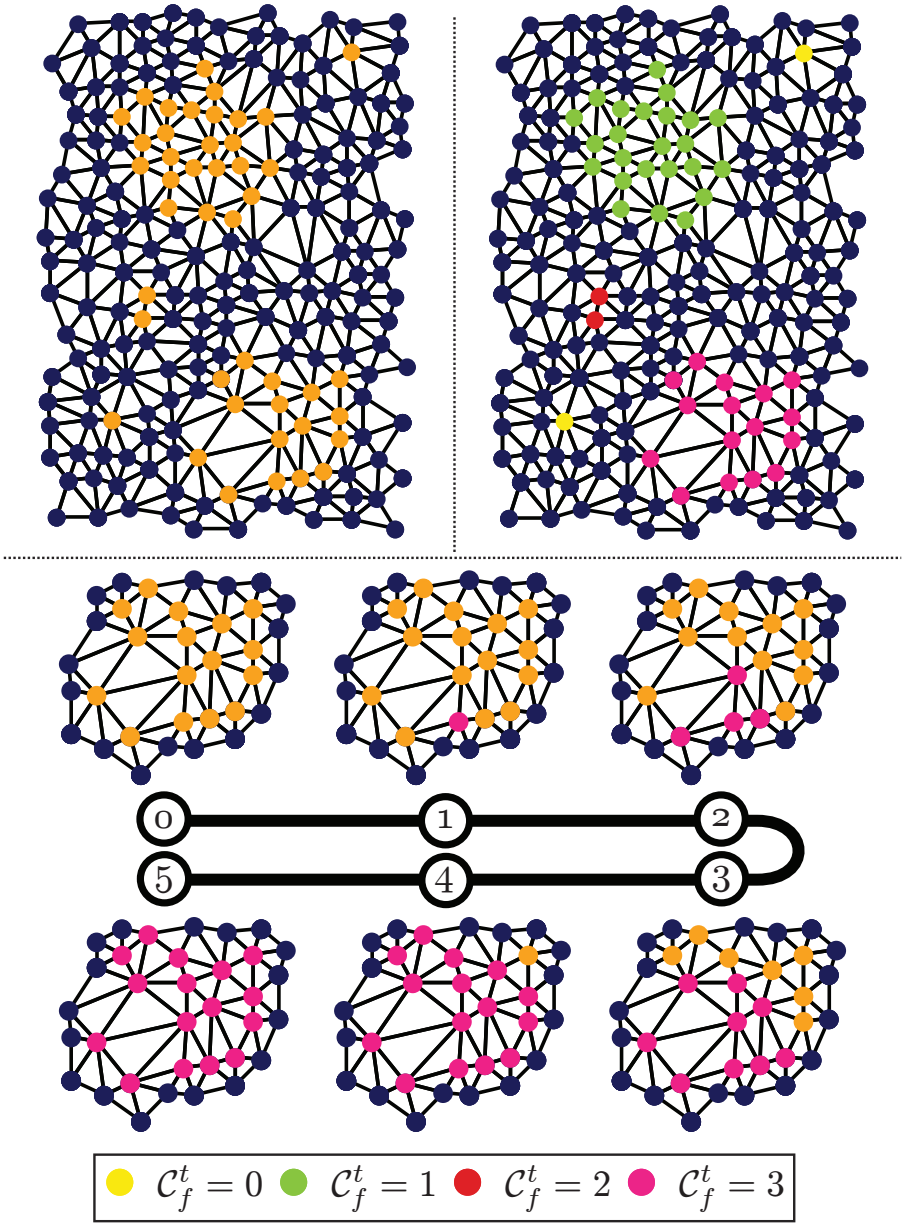

Figure C.3: ***Top left**: Examples of trajectory points in bundle n indicated by **orange** dots. The **dark blue** dots are trajectory points belonging to other bundles. The spatial links (black lines) between the points are obtained from Delaunay triangulations. **Top right**: The 3 groups of connected points in green, red and pink. A trajectory $\mathcal{X}_t$ has a connection label $\mathcal{C}_f^t$ at frame f, according to the group it belongs to. The yellow points ($\mathcal{C}_f^t = 0$) are isolated points that are not connected spatially to any other point in the bundle. **Bottom row**: The propagation of the connection label $\mathcal{C}_f^t = 3$ to the pink group of connected points. At step 1, a point is selected at random (pink point), and it passes the connection label to the other points in the group via the network of spatial links (steps 2-5).*

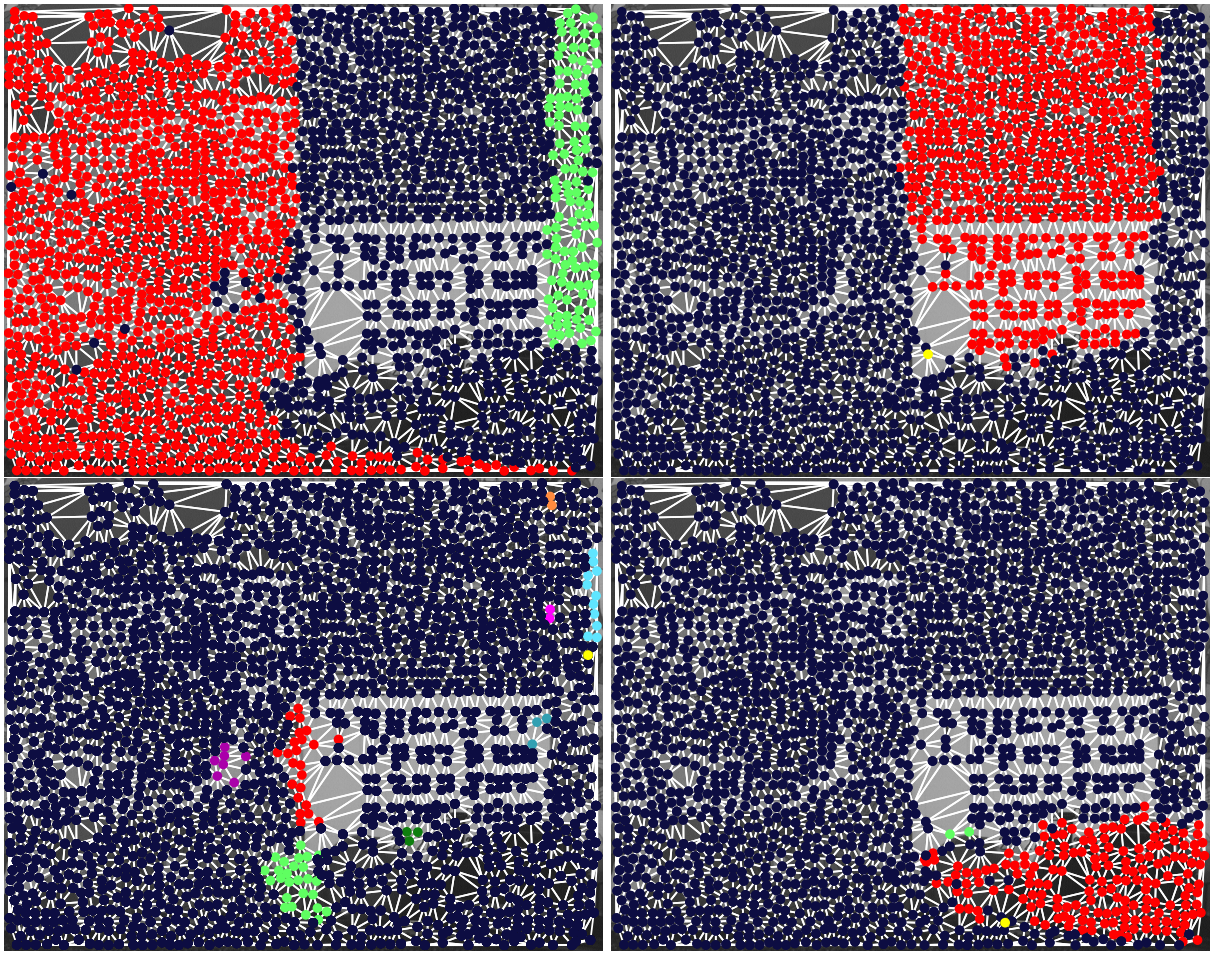

Figure C.4: *The connection labels for four bundles at frame 12 in the* ***Calendar and Mobile*** *sequence. The trajectory points that have connection labels* $\mathcal{C}_f^t = 0$ *(isolated point),* $\mathcal{C}_f^t = 1$*, and* $\mathcal{C}_f^t = 2$ *are coloured* ***yellow****,* ***red****, and* ***light green****.* ***Bottom left****: This bundle has a mixture of trajectory points corresponding to the background and calendar image regions. However the 8 groups of connected trajectories indicated by different colours represent separate image regions.*

(red) and $\mathcal{C}_f^t = 3$ (pink).

We locate a group of connected points by randomly selecting a point $\mathbf{x}_r$ from the set of trajectory points and iteratively propagate a unique 'message' through the network of spatial links. Here, only the points in the same group as $\mathbf{x}_r$ will receive the 'message'. We then assign all these points in the same group as $\mathbf{x}_r$ a connection label $\mathcal{C}_f^t = c + 1$, where c is the number of groups discovered so far. The other groups are discovered in a similar manner, by choosing a random point $\mathbf{x}_r$ from the remaining unlabelled points. Recall that if the point $\mathbf{x}_r$ is isolated it is given a connection label of zero. The *bottom row* of fig. C.3 demonstrates an example of a random point $\mathbf{x}_r$ being selected from the 3^{rd} group of connected points (pink). The point $\mathbf{x}_r$ is identified at step 1 as a *pink* dot, while the unlabelled points are *orange* dots. The label is

propagated to the other points in the group via the spatial links (neighbourhoods) in steps 2-5.

Fig. C.4 shows the groups of connected trajectory points at frame 12 for the four bundles in fig. C.2 (*Calendar and Mobile* sequence). In these illustrations trajectory points that have connection labels $\mathcal{C}_f^t = 0$ (isolated point), $\mathcal{C}_f^t = 1$, $\mathcal{C}_f^t = 2$ are coloured *yellow*, *red*, and *light green* respectively. The trajectory points belonging to other bundles are shown as *dark blue* points.

The background has two groups of connected points $\mathcal{C}_f^t = 1$ (red) and $\mathcal{C}_f^t = 2$ (light green) on both sides of the calendar as expected. The trajectories in both these groups however are members of a single sub-bundle as previously illustrated in fig. C.2. They are connected by a set of spatial links at some other frames.

D

Pixel Occlusion Labels

This appendix is a detailed discussion of ideas presented in chapter 7. See this chapter for the definitions of the various notations that will be subsequently used here.

We enforce spatial and temporal smoothness on the occlusion labels $\mathcal{D}_{f,j}(\mathbf{s})$ for the DFDs by generating a MAP estimate for these labels. Consider that $\mathbf{D}_f^n(\mathbf{s})$ is a label vector formed from the concatenation of the DFD labels $\mathcal{D}_{f,j}(\mathbf{s})$ where $\mathbf{D}_f^n(\mathbf{s})$ is defined as follows.

$$\mathbf{D}_f^n(\mathbf{s}) = \left[\mathcal{D}_{f,f-k}^n(\mathbf{s}), \ldots, \mathcal{D}_{f,f-1}^n(\mathbf{s}), \mathcal{D}_{f,f+1}^n(\mathbf{s}), \ldots, \mathcal{D}_{f,f+k}^n(\mathbf{s})\right] \tag{D.1}$$

As an example the occlusion label vectors for the sites $(f, \mathbf{s}_0)$, $(f, \mathbf{s}_1)$ and $(f, \mathbf{s}_2)$ in the *bottom* row of fig. 7.12 are $\mathbf{D}_f^n(\mathbf{s}_0) = [1\ 1\ 1\ 1\ 1\ 1]$, $\mathbf{D}_f^n(\mathbf{s}_1) = [0\ 0\ 0\ 1\ 1\ 1]$ and $\mathbf{D}_f^n(\mathbf{s}_2) = [0\ 0\ 0\ 0\ 0\ 0]$ respectively.

We define the posterior distribution for the occlusion vector label $\mathbf{D}_f^n(\mathbf{s})$ given the DFDs $\Delta_{f,j}^n(\mathbf{s})$ at the *window* frames and the neighbouring labels $\mathbf{D}_f^n(\sim \mathbf{s})$ as follows.

$$p(\mathbf{D}_f^n(\mathbf{s})|\Delta_{f,j}^n(\mathbf{s}), \mathbf{D}_f^n(\sim \mathbf{s})) = p_d(\Delta_{f,j}^n(\mathbf{s})|\mathbf{D}_f^n(\mathbf{s}))p_l(\mathbf{D}_f^n(\mathbf{s})|\mathbf{D}_f^n(\sim \mathbf{s})) \tag{D.2}$$

Where $p_d(\Delta_{f,j}^n(\mathbf{s})|.)$ is the likelihood that the pixel site $(f, \mathbf{s})$ takes the occlusion vector label $\mathbf{D}_f^n(\mathbf{s})$. Label smoothness is injected through the MRF prior $p_l(.)$.

Note that the number of possible vector labels $\mathbf{D}_f^n(\mathbf{s})$ is 2^{2k}, where $2k$ is the number of *window* frames. We reduce the number of possible labels to a smaller set of candidate labels $\hat{\mathbf{D}}_f^{n,c}$ in a selection process discussed later. Here $c = \{1 : C\}$ where $C < 2^{2k}$ is the number of candidate vector labels.

D.1 Occlusion Label Likelihood

The likelihood $p_d(\Delta^n_{f,j}(\mathbf{s})|.)$ involves evaluating the probability of the candidate vector label $\hat{\mathbf{D}}^{n,c}_f$ describing the observed DFDs for pixel site $(f, \mathbf{s})$. This likelihood is defined as follows.

$$p_d(\Delta^n_{f,j}(\mathbf{s})|\mathbf{D}^n_f(\mathbf{s})) = \prod_{j=f-k}^{f+k} p_e(\mathcal{D}^n_{f,j}(\mathbf{s}) = \hat{\mathcal{D}}^{n,c}_{f,j}|\Delta^n_{f,j}(\mathbf{s})),\ j \neq f \tag{D.3}$$

Here the likelihood $p_d(\Delta^n_{f,j}(\mathbf{s})|.)$ in eq. D.3 is dependent on a data driven distribution $p_e(\mathcal{D}^n_{f,j}(\mathbf{s}) = \hat{\mathcal{D}}^{n,c}_{f,j}|.)$. Since the candidate occlusion label $\hat{\mathcal{D}}^{n,c}_{f,j} \in \{0, 1\}$, the two forms of the prior $p_e(\mathcal{D}^n_{f,j}(\mathbf{s}) = \hat{\mathcal{D}}^{n,c}_{f,j}|.)$ are $p_e(\mathcal{D}^n_{f,j}(\mathbf{s}) = 0|.)$ and $p_e(\mathcal{D}^n_{f,j}(\mathbf{s}) = 1|.)$ for $\hat{\mathcal{D}}^{n,c}_{f,j} = 0$ and $\hat{\mathcal{D}}^{n,c}_{f,j} = 1$ respectively.

Hence the distribution $p_e(\mathcal{D}^n_{f,j}(\mathbf{s}) = \hat{\mathcal{D}}^{n,c}_{f,j}|.)$ for pixel site $(j, \mathbf{s})$ in *window* frame j, is a Gamma distribution of the DFD $\Delta^n_{f,j}(\mathbf{s})$ at this site.

$$p(\mathcal{D}^n_{f,j}(\mathbf{s}) = \hat{\mathcal{D}}^{n,c}_{f,j}|\Delta^n_{f,j}(\mathbf{s})) = \frac{\left[\Delta^n_{f,j}(\mathbf{s})\right]^{h-1} \exp\left[-\Delta^n_{f,j}(\mathbf{s})/\theta\right]}{\theta^{h-1}\Gamma(h)} \tag{D.4}$$

Where the parameters (θ, h) for the Gamma distribution in eq. D.4 are $(\theta^n_{f,0}, h^n_{f,0})$ and $(\theta^n_{f,1}, h^n_{f,1})$ for $p_e(\mathcal{D}^n_{f,j}(\mathbf{s}) = 0|.)$ and $p_e(\mathcal{D}^n_{f,j}(\mathbf{s}) = 1|.)$ respectively. From observation of DFDs for real sequences we found that the distribution is long tailed with heavier tails than a Gaussian. The estimation of the parameters $(\theta^n_{f,0}, h^n_{f,0})$ and $(\theta^n_{f,1}, h^n_{f,1})$ is discussed later.

Fig. D.1 will now be used to explain when maximum likelihood is achieved given the occlusion vector labels $\mathbf{D}^n_f(\mathbf{s})$. The illustration in fig. D.1 shows the DFDs $\Delta^n_{f,j}$ (top row) for the motion compensated *window* frames $\hat{\mathbf{I}}^n_{f,j}$ (middle row), where $j = \{f-k, \ldots, f-1, f+1, \ldots, f+k\}$. These frames are motion compensated with the nth trajectory bundle which represents the *red* object. The *green* object occludes the *red* object at frames $\{f-k, \ldots, f+1\}$. The *bottom* row shows a zoom on the current frame $\mathbf{I}_f$ with the pixel sites in the 10×10 grid indexed $x = \{1:10\}$ and $y = \{1:10\}$. Pixel sites $\mathbf{s} = (5, 6)$ and $\mathbf{s} = (5, 7)$ for the *red* object are occluded in the current frame $\mathbf{I}_f$. 'High' $(\mathcal{D}^n_{f,j}(\mathbf{s}) = 1)$ and 'low' $(\mathcal{D}^n_{f,j}(\mathbf{s}) = 0)$ DFDs in the *top* row are indicated with *purple* and *cyan* squares respectively.

In fig. D.1 the pixel sites $\mathbf{s} = (6, 4), (6, 5), (6, 6),$ and $(6, 7)$ have maximum likelihoods $p_d(\Delta^n_{f,j}(\mathbf{s})|.)$ when $\mathbf{D}^n_f(\mathbf{s}) = [0\ 0\ 0\ 0\ 0\ 0]$. For sites $\mathbf{s} = (5, 4)$ and $(5, 5)$ maximum likelihoods for these sites occur when $\mathbf{D}^n_f(\mathbf{s}) = [1\ 1\ 0\ 0\ 0\ 0]$ and $\mathbf{D}^n_f(\mathbf{s}) = [0\ 1\ 1\ 0\ 0\ 0]$ respectively. For all the other sites in the 10×10 grid we achieve maximum likelihood when $\mathbf{D}^n_f(\mathbf{s}) = [1\ 1\ 1\ 1\ 1\ 1]$.

The next section discusses how the candidate labels $\hat{\mathcal{D}}^{n,c}_{f,j}$ are selected.

D.1.1 Candidate Occlusion Labels

We use the maximum likelihood estimates of occlusion for all the $H \times W$ pixel sites $(f, \mathbf{s})$ to determine the candidate labels $\hat{\mathbf{D}}^{n,c}_f$. Here H and W are the height and width of the frame $\mathbf{I}_f$ respectively. We make the candidate vector labels $\hat{\mathbf{D}}^{n,c}_f$ simply be the most frequently occurring

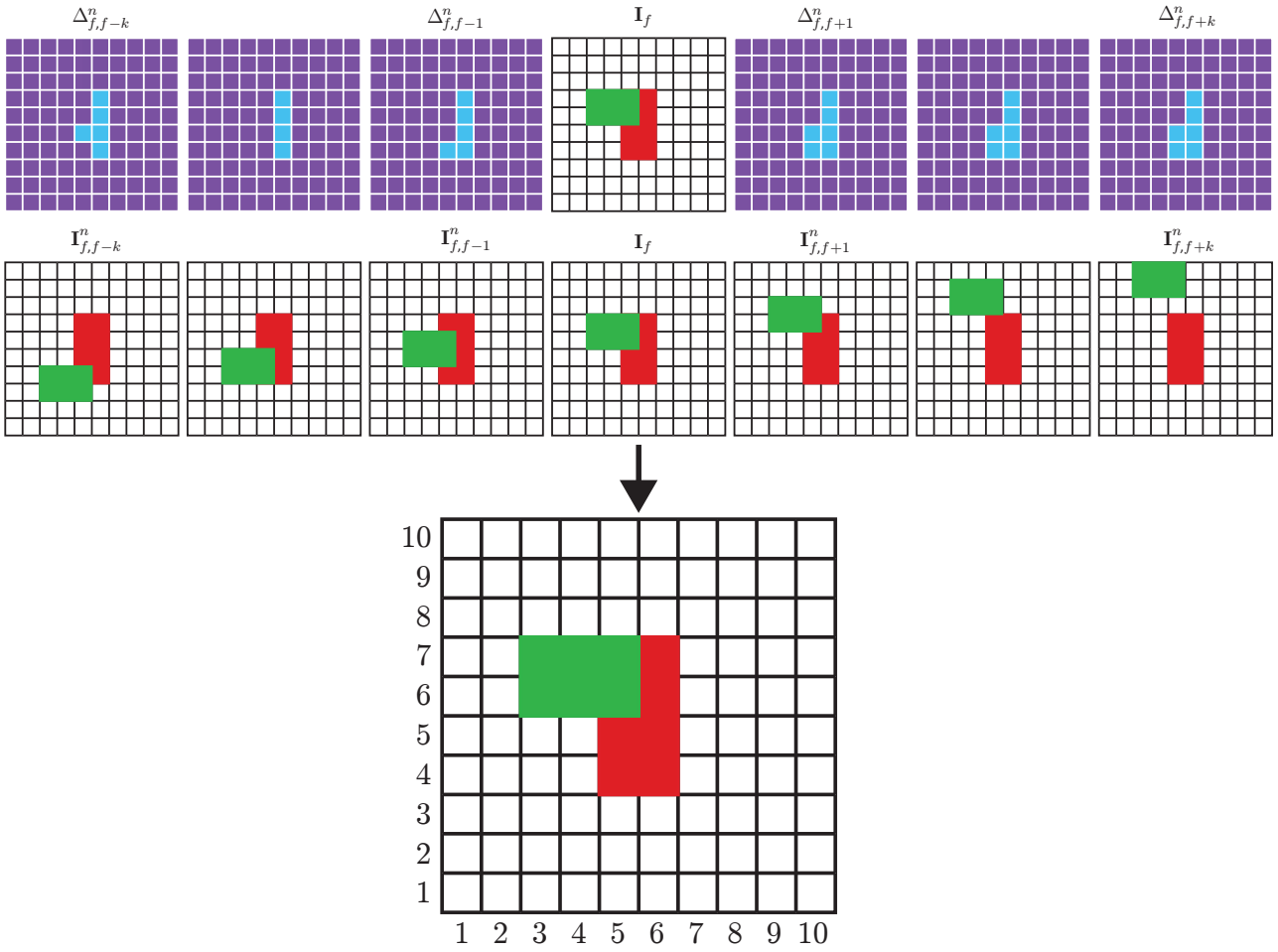

Figure D.1: *The DFDs* $\Delta^n_{f,j}$ ***(top row)*** *for the motion compensated window frames* $\hat{\mathbf{I}}^n_{f,j}$ ***(middle row)****, where* $j = \{f-k, \ldots, f-1, f+1, \ldots, f+k\}$*. These frames are motion compensated with the* n*th trajectory bundle which represents the* ***red*** *object. The* ***green*** *object occludes the* ***red*** *object at frames* $f-k, \ldots, f+1$*. 'High'* ($\mathcal{D}^n_{f,j}(\mathbf{s}) = 1$) *and 'low'* ($\mathcal{D}^n_{f,j}(\mathbf{s}) = 0$) *DFDs in the* ***top row*** *are indicated with* ***purple*** *and* ***cyan*** *squares respectively.* ***Bottom row****: A zoom on the current frame* $\mathbf{I}_f$ *with the pixel sites in the* 10×10 *grid indexed* $x = \{1:10\}$ *and* $y = \{1:10\}$*.*

maximum likelihood labels. Consider that the vector label $\tilde{\mathbf{D}}^n_f(\mathbf{s})$ provides maximum likelihood for the pixel site $(f, \mathbf{s})$. This vector label $\tilde{\mathbf{D}}^n_f(\mathbf{s})$ is given below.

$$\tilde{\mathbf{D}}^n_f(\mathbf{s}) = \underset{\mathbf{D}^n_f(\mathbf{s})}{\arg\max} \left(p_d(\Delta^n_{f,j}(\mathbf{s}) | \mathbf{D}^n_f(\mathbf{s})) \right) \tag{D.5}$$

The elements in the maximum likelihood vector label are defined as follows.

$$\tilde{\mathbf{D}}^n_f(\mathbf{s}) = \left[\tilde{\mathcal{D}}^n_{f,f-k}(\mathbf{s}), \ldots, \tilde{\mathcal{D}}^n_{f,f-1}(\mathbf{s}), \tilde{\mathcal{D}}^n_{f,f+1}(\mathbf{s}), \ldots, \tilde{\mathcal{D}}^n_{f,f+k}(\mathbf{s}) \right] \tag{D.6}$$

Where $\tilde{\mathcal{D}}^n_{f,j}(\mathbf{s})$ is the maximum likelihood label at site $(j, \mathbf{s})$ in the jth *window* frame, and $j = \{f-k, \ldots, f-1, f+1, \ldots, f+k\}$.

The value of the maximum likelihood label $\tilde{\mathcal{D}}^n_{f,j}(\mathbf{s})$ at site $(j,\mathbf{s})$ is determined from the distribution $p_e(\mathcal{D}^n_{f,j}(\mathbf{s}) = \hat{\mathcal{D}}^{n,c}_{f,j}|.)$ in eq. D.4 as follows.

$$\tilde{\mathcal{D}}^n_{f,j}(\mathbf{s}) = \begin{cases} 1, & \textbf{if } p_e(\mathcal{D}^n_{f,j}(\mathbf{s}) = 1|.) > p_e(\mathcal{D}^n_{f,j}(\mathbf{s}) = 0|.) \\ 0, & \textbf{else} \end{cases} \tag{D.7}$$

A maximum likelihood label $\tilde{\mathbf{D}}^n_f(\mathbf{s})$ is selected as a candidate label $\hat{\mathbf{D}}^{n,c}_f$ if this label $\tilde{\mathbf{D}}^n_f(\mathbf{s})$ occurs for more than 10 pixel sites. For $k = 3$ the number of candidate labels C is usually about 10, which is less than the $2^{2k} = 64$ possible labels.

The next section discusses how the parameters for the distribution $p_e(\mathcal{D}^n_{f,j}(\mathbf{s}) = \hat{\mathcal{D}}^{n,c}_{f,j}|.)$ are derived.

D.1.2 Parameters for the Gamma Distribution

As mentioned in a previous section, the parameters (θ, h) for the distribution $p_e(\mathcal{D}^n_{f,j}(\mathbf{s}) = \hat{\mathcal{D}}^{n,c}_{f,j}|.)$ in eq. D.4 are $(\theta^n_{f,0}, h^n_{f,0})$ and $(\theta^n_{f,1}, h^n_{f,1})$ for $p_e(\mathcal{D}^n_{f,j}(\mathbf{s}) = 0|.)$ and $p_e(\mathcal{D}^n_{f,j}(\mathbf{s}) = 1|.)$ respectively.

We first estimate the parameters $(\theta^n_{f,0}, h^n_{f,0})$ and then proceed to estimate $(\theta^n_{f,1}, h^n_{f,1})$. We found from observation of DFDs for real sequences that we can safely assume that a pixel site $(f,\mathbf{s})$ is not occluded at the corresponding sites $(j,\mathbf{s})$ if the variance and mean of the DFDs $\Delta^n_{f,j}(\mathbf{s})$ is low. Recall that $j = \{f-k, \ldots, f-1, f+1, \ldots, f+k\}$ is the index of the *window* frames for the current frame f. Hence a pixel site $(f,\mathbf{s})$ is automatically given a DFD occlusion vector label of all zeros ($\mathbf{D}^n_f(\mathbf{s}) = [0\ \ldots\ 0\ 0\ \ldots\ 0]$) if the variance $V^n_f(\mathbf{s})$ and mean $M^n_f(\mathbf{s})$ of the DFDs $\Delta^n_{f,j}(\mathbf{s})$ is below a threshold of 10 and 20 respectively. This mean $V^n_f(\mathbf{s})$ and variance $V^n_f(\mathbf{s})$ is given below.

$$M^n_f(\mathbf{s}) = \frac{1}{2k}\sum_{j=f-k}^{f+k}\left[\Delta^n_{f,j}(\mathbf{s})\right], \quad V^n_f(\mathbf{s}) = \frac{1}{2k}\sum_{j=f-k}^{f+k}\left[\Delta^n_{f,j}(\mathbf{s})\right]^2 - \left[M^n_f(\mathbf{s})\right]^2 \tag{D.8}$$

We now define $\tilde{\mathbf{s}}$ as a pixel site at frame f for which the variance $V^n_f(\mathbf{s}) < 10$ and mean $M^n_f(\mathbf{s}) < 20$. We make the reasonable assumption that the DFD $\Delta^n_{f,j}(\tilde{\mathbf{s}})$ is a sample drawn from the distribution $p_e(\mathcal{D}^n_{f,j}(\mathbf{s}) = 0|.)$. Hence the parameters $(\theta^n_{f,0}, h^n_{f,0})$ are obtained from fitting the model of the Gamma distribution to these DFD samples $\Delta^n_{f,j}(\tilde{\mathbf{s}})$.

The parameters $(\theta^n_{f,1}, h^n_{f,1})$ for $p_e(\mathcal{D}^n_{f,j}(\mathbf{s}) = 1|.)$ are estimated in a similar manner to $(\theta^n_{f,0}, h^n_{f,0})$. However we use the parameters $(\theta^n_{f,0}, h^n_{f,0})$ to select DFD samples for estimating $(\theta^n_{f,1}, h^n_{f,1})$. We now define $\hat{\mathbf{s}}$ as a pixel site at frame f for which the variance $V^n_f(\mathbf{s}) \geq 10$ and mean $M^n_f(\mathbf{s}) \geq 20$. The DFD $\Delta^n_{f,j}(\hat{\mathbf{s}})$ is selected as a sample to estimate the parameters $(\theta^n_{f,1}, h^n_{f,1})$ if $\Delta^n_{f,j}(\hat{\mathbf{s}}) > \phi^n_f$, where ϕ^n_f is a predetermined threshold. This threshold is selected so that $p_e(\mathcal{D}^n_{f,j}(\mathbf{s}) = 0|\Delta^n_{f,j}(\mathbf{s}) < \phi^n_f) = 0.95$, i.e. 95% of the 'low' DFDs are below the threshold ϕ^n_f. Using this threshold we can gather all the 'high' DFDs samples for estimating the parameters $(\theta^n_{f,1}, h^n_{f,1})$ for $p_e(\mathcal{D}^n_{f,j}(\mathbf{s}) = 1|.)$. The threshold ϕ^n_f is estimated from the parameters $(\theta^n_{f,0}, h^n_{f,0})$ as follows.

$$\phi^n_f = \theta^n_{f,0}\left(h^n_{f,0} + 2\sqrt{h^n_{f,0}}\right) \tag{D.9}$$

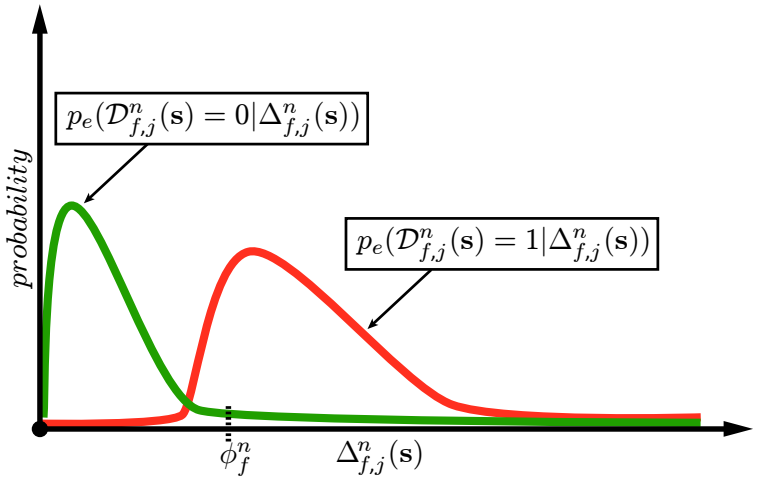

Figure D.2: *The typical shapes of the distributions* $p_e(\mathcal{D}^n_{f,j}(\mathbf{s}) = 0|\Delta^n_{f,j}(\mathbf{s}))$ *and* $p_e(\mathcal{D}^n_{f,j}(\mathbf{s}) = 1|\Delta^n_{f,j}(\mathbf{s}))$ *are shown as the* ***green*** *and* ***red*** *plots respectively. The threshold* ϕ^n_f *is also highlighted.*

Examples of the typical distributions $p_e(\mathcal{D}^n_{f,j}(\mathbf{s}) = 0|.)$ and $p_e(\mathcal{D}^n_{f,j}(\mathbf{s}) = 1|.)$ are shown as the *green* and *red* plots in fig. D.2 respectively. Also shown in fig. D.2 is the relative location of the threshold ϕ^n_f in eq. D.9 with respect to these distributions.

D.2 The Occlusion Label Prior

The prior $p_l(\mathbf{D}^n_f(\mathbf{s})|\mathbf{D}^n_f(\sim \mathbf{s}))$ at site $(f, \mathbf{s})$ for the occlusion label $\mathbf{D}^n_f(\mathbf{s})$ given the neighbouring labels $\mathbf{D}^n_f(\sim \mathbf{s})$ is a Gibbs distribution defined as follows.

$$p_l(\mathbf{D}^n_f(\mathbf{s})|\mathbf{D}^n_f(\sim \mathbf{s})) = \frac{1}{Z} \exp\left\{ -\frac{\lambda_D}{2k} \sum_{\mathbf{c}\in\mathcal{N}_\mathbf{s}} \left(\sum_{j=f-k}^{f+k} \left[\mathcal{D}^n_{f,j}(\mathbf{c}) \neq \mathcal{D}^n_{f,j}(\mathbf{s})\right] \right) \right\}, j \neq f \qquad \text{(D.10)}$$

The neighbourhood of pixel site $(f, \mathbf{s})$ is defined as $\mathcal{N}_s$, where $\mathcal{N}_s$ is the set of 3×3 pixel sites surrounding $(f, \mathbf{s})$. The weight Z above in eq. D.10 is the usual normalization constant. The constant λ_D controls the weight the prior has in the MAP solution. For all experiments it was found that setting $\lambda_D = 0.01$ gave suitable results.

The operation $\mathcal{D}^n_{f,j}(\mathbf{c}) \neq \mathcal{D}^n_{f,j}(\mathbf{s})$ in the prior distribution encourages neighbouring pixels $(f, \mathbf{s})$ and $(f, \mathbf{c})$ to have the same occlusion labels $\mathcal{D}^n_{f,j}(\mathbf{s})$ at the *window* frame sites $(j, \mathbf{s})$.

E

Demonstration DVD

Our demonstration DVD contains videos of the results presented in this thesis. The *top row* of fig. E.1 shows the main menu for this DVD. The links on this menu are:

- *Dense Pixel Segmentation* - Link to the *Dense Pixel Segmentation* menu shown in the bottom left of fig. E.1. We will discuss this menu later.
- *Sparse Trajectory Segmentation* - Link to the *Sparse Trajectory Segmentation* menu shown in the bottom right of fig. E.1. We will discuss this menu later.
- *Visual Surveillance* - Demonstrates the tracking of foreground objects in three sequences that were used for our visual surveillance application [15] discussed in section 2.5.4 (chapter 2). Each foreground object trajectory is shown in a different colour.
- *Viterbi Tracker* - The video results for chapter 3, which discusses our Viterbi tracking framework. The tracking of SIFT [73] features for the *Triniman*, *Artbeats-SP128*, *Calendar and Mobile* and *Graveyard* sequences are shown in this video. Also demonstrated are tracking comparisons with a standard KLT tracker [13] on the *Graveyard* sequence.
- *Bonus* - The story behind the *Triniman* sequence.

E.1 Dense Pixel Segmentation Menu

The *bottom left* of fig. E.1 shows this menu. On this menu there are links to the dense segmentation results for the *Triniman*, *Artbeats-SP128* and *Calendar and Mobile* sequences. These

Figure E.1: *The* ***main*** *(top row),* ***Dense Pixel Segmentation*** *(bottom left),* ***Sparse Trajectory Segmentation*** *(bottom right) menus on our demonstration DVD.*

results were presented in chapter 8. In this chapter we compared our proposed dense segmentation techniques with three other techniques; *NUKE* [62], *FeatureCut* [100] and *SnapCut* [12]. We proposed a semi-automatic and an automatic segmentation technique which we defined as *Semi-Auto* and *Geodesic* respectively. See chapter 8 for a detailed description of these techniques.

The results in the videos for the dense segmentations are labelled *NUKE*, *FeatureCut*, *SnapCut*, *Semi-Auto* and *Geodesic* according to the technique that generated them. Our *Semi-Auto* technique, along with the *FeatureCut* and *SnapCut* techniques are supervised. That is, for these techniques a user supplies segmentations for some of the frames in a sequence. These user defined frames are defined as *key* frames. In all videos demonstrated, *key* frames are marked with *black squares* in the *top right* corners of these frames. Fig. E.2 shows the *key* frame for frame 1 in the *Calendar and Mobile* sequence. The *left* of fig. E.2 shows the foreground extracted with the user *key* frame on the right. Note that both these illustrations have the *black squares* to indicate that they are *key* frames.

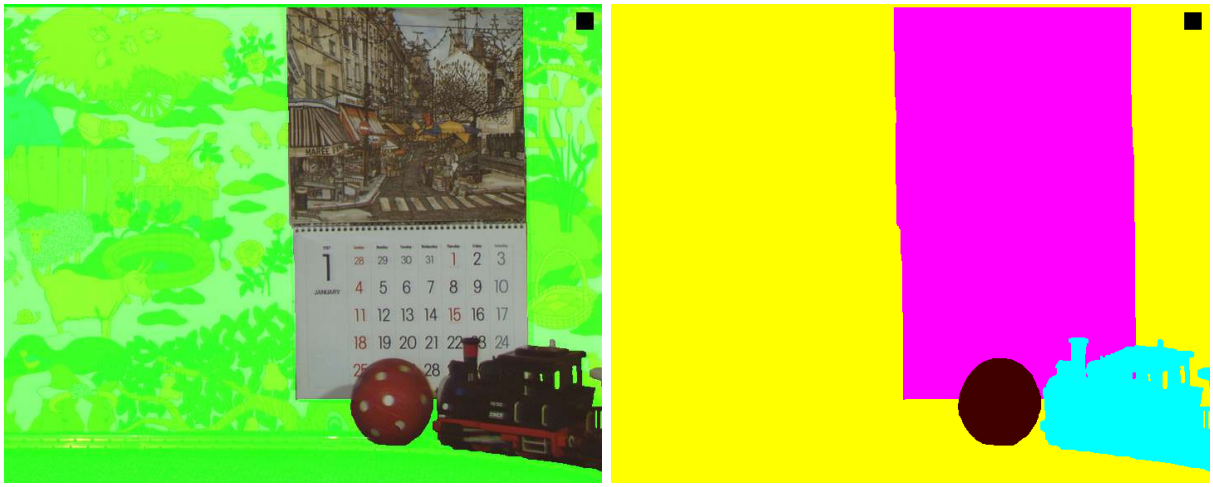

Figure E.2: *The* ***key*** *frame for frame 1 in the* ***Calendar and Mobile*** *sequence. The foreground on the left is extracted using the key frame on the right where foregound is any colour that is not yellow (bakcground). Both these representations are indicated as* ***key*** *frames by the* ***black squares*** *in the top right corners.*

E.1.1 Varying Key Frames for Semi-Auto Approach

This link on the *Dense Pixel Segmentation* menu shows the results when the number of *key* frames supplied to our *Semi-Auto* technique is varied for the *Triniman*, *Artbeats-SP128* and *Calendar and Mobile* sequences. Recall that these experiments were discussed in section 8.4 (chapter 8). Fig. E.3 demonstrates how we indicate the number of key frames (# key frames), the mean recall (RR) and false alarm (FA) rates for each sequence in our video demonstration.

E.2 Sparse Trajectory Segmentation

The *bottom right* of fig. E.1 shows this menu. On this menu there are links to the sparse segmentation results for the 11 sequences discussed throughout chapter 6. Also there is a link to a video showing our segmentations for the 155 sequences in the Hopkins dataset [117]. In this dataset ground truth segmentations are supplied so we can evaluate misclassification rates for the segmentation produced by various techniques. See chapter 6 for more details.

Our sparse trajectory segmentation technique produced the overall lowest misclassification rates for the sequences in the Hopkins dataset compared to the other techniques investigated. Out of the all the other techniques investigated the MSL [59] technique produced the lowest overall rates. Hence in the video for the Hopkins dataset we show the ground truth segmentations, our segmentations and the MSL segmentations only for each sequence. Note that this video is 22 minutes long. Fig. E.4 demonstrates how we indicate the segmentation technique ([Our] or [MSL]) for a sequence in the Hopkins dataset. The corresponding misclassification

Figure E.3: *The segmentation produced by our* ***Semi-Auto*** *technique for a frame in the* ***Calendar and Mobile*** *sequence where 6* ***key*** *frames are supplied to our technique. The text sumperimposed on the frame indicate the number of* ***key*** *frames (# key frames), the mean recall (RR) and false alarm (FA) rates for this sequence. The illustration on the right is a zoom on this text shown in the left frame.*

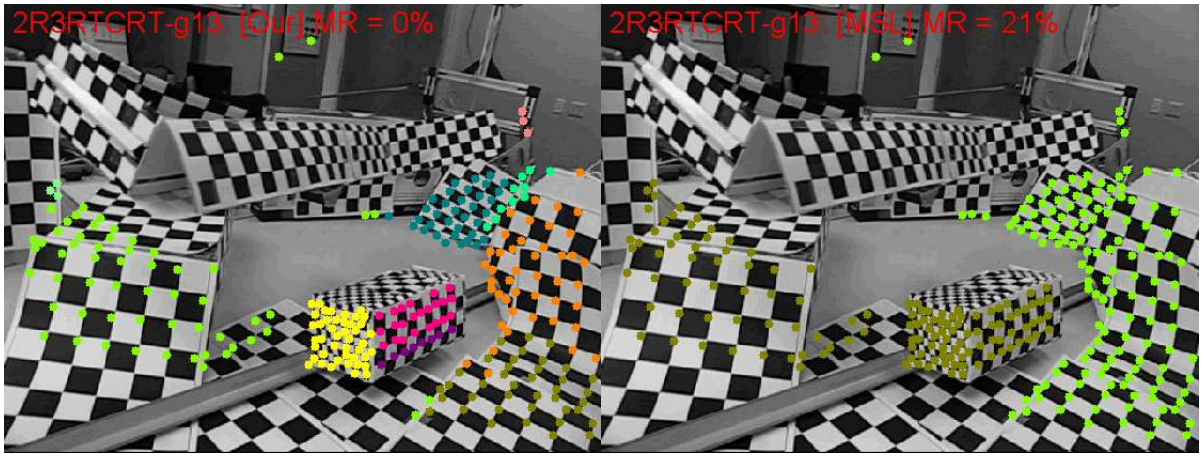

Figure E.4: ***Left****: Our sparse segmentation ([Our])for a frame in the* ***2R3RTCRT-g13*** *sequence.* ***Right****: The MSL sparse segmentation ([MSL])for a frame in the* ***2R3RTCRT-g13*** *sequence. The misclassification rates for our technique and the MSL technique are indicated as $MR = 0\%$ and $MR = 21\%$ respectively on the corresponding frames.*

rates are indicated as '$MR =$'.

Bibliography

[1] Adobe systems incorporated, after effects cs5. http://www.adobe.com/products/aftereffects/.

[2] The foundry, nuke 6.0, http://www.thefoundry.co.uk.

[3] E. H. Adelson and J. R. Bergen. Spatiotemporal energy models for the perception of motion. *j. Opt. Soc. Am. A.*, 2(2):284–299, 1985.

[4] Y. Altunbasak, P. E. Eren, and A. M. Tekalp. Region-based parametric motion segmentation using color information. *Graph. Models Image Process.*, 60(1):13–23, 1998. 288134.

[5] N. Anjum and A. Cavallaro. Unsupervised fuzzy clustering for trajectory analysis. In *Image Processing, 2007. ICIP 2007. IEEE International Conference on*, volume 3, pages III – 213–III – 216, 2007.

[6] N. Apostoloff and A. Fitzgibbon. Bayesian video matting using learnt image priors. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 1, pages I–407–I–414 Vol.1, 2004.

[7] E. Arnaud and E. Memin. Partial linear gaussian models for tracking in image sequences using sequential monte carlo methods. *International Journal of Computer Vision*, 74(1):75–102, 2007.

[8] S. Arseneau and J. R. Cooperstock. Real-time image segmentation for action recognition. In *Communications, Computers and Signal Processing, 1999 IEEE Pacific Rim Conference on*, pages 86–89, 1999.

[9] S. Ayer and H. S. Sawhney. Layered representation of motion video using robust maximum-likelihood estimation of mixture models and mdl encoding. In *Computer Vision, 1995. Proceedings., Fifth International Conference on*, pages 777–784, 1995.

[10] X. Bai and G. Sapiro. A geodesic framework for fast interactive image and video segmentation and matting. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8, Oct. 2007.

[11] X. Bai, J. Wang, and G. Sapiro. Dynamic color flow: A motion-adaptive color model for object segmentation in video, 2010.

[12] X. Bai, J. Wang, D. Simons, and G. Sapiro. Video snapcut: robust video object cutout using localized classifiers. In *ACM Transactions on Graphics,SIGGRAPH*, 2009. 1531376 1-11.

[13] S. Baker and I. Matthews. Lucas-kanade 20 years on: A unifying framework. *International Journal of Computer Vision*, 56:221–255, 2004.

[14] S. Baker, R. Szeliski, and P. Anandan. A layered approach to stereo reconstruction. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE Computer Society, 1998.

[15] G. Baugh and A. Kokaram. Feature-based object modelling for visual surveillance. In *Image Processing, 2008. ICIP 2008. 15th IEEE International Conference on*, pages 1352–1355, 2008.

[16] J. Bergen, P. Burt, R. Hingori, and S. Peleg. Computing two motions from three frames. In *Proceedings of the Third International Conference on Computer Vision*, pages 27–32, December 1990.

[17] J. Besag. On the statistical analysis of dirty pictures. *Journal of the Royal Statistical Society*, B-48:259–302, 1986.

[18] S. T. Birchfield. Klt: An implementation of the kanade-lucas-tomasi feature tracker, 2007.

[19] S. T. Birchfield and S. J. Pundlik. Joint tracking of features and edges. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–6, 2008.

[20] S. T. Birchfield and R. Sriram. Spatiograms versus histograms for region-based tracking. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, pages 1158–1163 vol. 2, 2005.

[21] A. Blake, C. Rother, M. Brown, P. Perez, and P. Torr. Interactive image segmentation using an adaptive gmmrf model. In *Computer Vision - ECCV 2004*, pages 428–441. 2004.

[22] C. A. Bouman. Cluster: An unsupervised algorithm for modeling Gaussian mixtures, April 1997.

[23] P. Bouthemy and E. Francois. Motion segmentation and qualitative dynamic scene analysis from an image sequence. *International Journal of Computer Vision*, 10(2):157–182, 1993.

[24] Y. Boykov and G. Funka-Lea. Graph cuts and efficient n-d image segmentation. *Int. J. Comput. Vision*, 70(2):109–131, 2006. 1147801.

[25] Y. Boykov, O. Veksler, and R. Zabih. Markov random fields with efficient approximations. In *Computer Vision and Pattern Recognition, 1998. Proceedings. 1998 IEEE Computer Society Conference on*, pages 648–655, 1998.

[26] Y. Y. Boykov and M. P. Jolly. Interactive graph cuts for optimal boundary & region segmentation of objects in n-d images. In *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, volume 1, pages 105–112 vol.1, 2001.

[27] Y.-Y. Chuang, A. Agarwala, B. Curless, D. H. Salesin, and R. Szeliski. Video matting of complex scenes. *ACM Transactions on Graphics*, 21(3):243–248, July 2002. Special issue: Proceedings of ACM SIGGRAPH 2002.

[28] D. Comaniciu and P. Meer. Robust analysis of feature spaces: color image segmentation. In *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, pages 750–755, 1997.

[29] D. Comaniciu and P. Meer. Mean shift: a robust approach toward feature space analysis. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(5):603–619, 2002.

[30] D. Corrigan, S. Robinson, and A. Kokaram. Video matting using motion extended grabcut. In *Visual Media Production (CVMP 2008), 5th European Conference on*, pages 1–9, 2008.

[31] J. Costeira and T. Kanade. A multi-body factorization method for motion analysis. In *Computer Vision,Proceedings of the 5th International Conference on*, pages 1071–1076, 1995.

[32] D. Cremers and S. Soatto. Variational space-time motion segmentation. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 886–893 vol.2, 2003.

[33] A. Criminisi, G. Cross, A. Blake, and V. Kolmogorov. Bilayer segmentation of live video. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 1, pages 53–60, 2006.

[34] G. Csurka and P. Bouthemy. Direct identification of moving objects and background from 2d motion models. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 1, pages 566–571 vol.1, 1999.

[35] T. Darrell and A. Pentland. Robust estimation of a multi-layered motion representation. In *Visual Motion, 1991., Proceedings of the IEEE Workshop on*, pages 173–178, 1991.

[36] J. Davis and A. Bobick. The representation and recognition of human movement using temporal templates. In *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, pages 928 –934, jun. 1997.

[37] D. Dementhon. Spatio-temporal segmentation of video by hierarchical mean shift analysis. In *Center for Automat. Res., U. of Md, College Park*, 2002.

[38] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

[39] F. Dufaux, F. Moscheni, and A. Lippman. Spatio-temporal segmentation based on motion and static segmentation. In *Image Processing, 1995. Proceedings., International Conference on*, volume 1, pages 306–309 vol.1, 1995.

[40] R. Dupont, O. Juan, and R. Keriven. Robust segmentation of hidden layers in video sequences. In *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*, volume 3, pages 75–78, 2006.

[41] P. F. Felzenszwalb and D. P. Huttenlocher. Efficient graph-based image segmentation. *Int. J. Comput. Vision*, 59(2):167–181, 2004. 981796.

[42] M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, 1981. 358692.

[43] J. Forney, G. D. The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278, 1973.

[44] P.-E. Forssén and D. Lowe. Shape descriptors for maximally stable extremal regions. In *IEEE International Conference on Computer Vision*, volume CFP07198-CDR, Rio de Janeiro, Brazil, October 2007. IEEE Computer Society.

[45] W. Forstner and E. Gulch. A fast operator for detection and precise location of distinct points, corners and centres of circular features. In *Intercommission Conference on Fast Processing of Photogrammetric Data*, pages 281–305, 1987.

[46] M. Fradet, P. Pérez, and P. Robert. Clustering point trajectories with various life-spans. In *Proc. Eur. Conf. on Visual Media Production (CVMP'09)*, London, UK, 2009.

[47] E. S. L. Gastal and M. M. Oliveira. Shared sampling for real-time alpha matting. *Computer Graphics Forum*, 29(2):575–584, 2010. Proceedings of Eurographics.

[48] S. Geman and D. Geman. Stochastic relaxation, gibbs distributions and the bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(6):721–741, 1984.

[49] D. B. Goldman, C. Gonterman, B. Curless, D. Salesin, and S. M. Seitz. Video object annotation, navigation, and composition. In *UIST '08: Proceedings of the 21st annual ACM symposium on User interface software and technology*, pages 3–12, New York, NY, USA, 2008. ACM.

[50] M. Grundmann, V. Kwatra, M. Han, and I. Essa. Efficient heirarchical graph-based video segmentation. In *Computer Vision and Pattern Recognition (CVPR)*, 2010.

[51] C. Gu and M. Lee. Tracking of multiple semantic video objects for internet applications. In *Visual Communications and Image Processing,SPIE*, volume 3653, pages 806–820, December 1999.

[52] P. Hillman, J. Hannah, and D. Renshaw. Alpha channel estimation in high resolution images and image sequences. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–1063–I–1068 vol.1, 2001.

[53] B. Horn and B. Schunck. Determining optical flow. *Artificial Intelligence*, 17:185–203, 1981.

[54] I.-S. Hsieh and K.-C. Fan. An adaptive clustering algorithm for color quantization. *Pattern Recognition Letters*, 21(4):337–346, 2000. doi: DOI: 10.1016/S0167-8655(99)00165-8.

[55] S. Jianbo and C. Tomasi. Good features to track. In *Computer Vision and Pattern Recognition,IEEE Computer Society Conference on*, pages 593–600, 1994.

[56] X. Jiangjian and M. Shah. Motion layer extraction in the presence of occlusion using graph cuts. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(10):1644–1659, 2005.

[57] X. Jiangjian and M. Shah. Motion layer extraction in the presence of occlusion using graph cuts. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(10):1644–1659, 2005.

[58] N. Jojic and B. J. Frey. Learning flexible sprites in video layers. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–199–I–206 vol.1, 2001.

[59] K. Kanatani and Y. Sugaya. Multi-stage optimization for multi-body motion segmentation. In *IEICE Transactions on Information and Systems*, volume E87-D, No. 7, pages 335–349, 2003.

[60] Q. Ke and T. Kanade. A subspace approach to layer extraction. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–255–I–262 vol.1, 2001.

[61] A. W. Klein, P.-P. J. Sloan, A. Finkelstein, and M. F. Cohen. Stylized video cubes. In *ACM SIGGRAPH Symposium on Computer Animation*, pages 15–22, July 2002.

[62] A. Kokaram, B. Collis, and S. Robinson. Practical motion based video matting. In *Visual Media Production, 2005. CVMP 2005. The 2nd IEE European Conference on*, pages 130–136, Nov. - 1 Dec. 2005.

[63] A. Kokaram and P. Delacourt. A new global motion estimation algorithm and its application to retrieval in sports events. In *Multimedia Signal Processing, 2001 IEEE Fourth Workshop on*, pages 251–256, 2001.

[64] M. P. Kumar, P. H. S. Torr, and A. Zisserman. Learning layered motion segmentations of video. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, volume 1, pages 33–40 Vol. 1, 2005.

[65] J. D. Lafferty, A. McCallum, and F. C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*. Morgan Kaufmann Publishers Inc., 2001.

[66] A. Levin, D. Lischinski, and Y. Weiss. Colorization using optimization. *ACM Trans. Graph.*, 23(3):689–694, 2004. 1015780.

[67] A. Levin, D. Lischinski, and Y. Weiss. A closed-form solution to natural image matting. *IEEE Trans. Pattern Anal. Mach. Intell.*, 30(2):228–242, 2008. 1340234.

[68] S. Li. Markov random field models in computer vision. In *Computer Vision ECCV '94*, pages 361–370. Springer, 1994. 10.1007/BFb0028368.

[69] S. Li. *Markov Random Field Modeling in Computer Vision*. Computer Science Workbench. Springer, 1995. ISBN-4-431-70145-1.

[70] Y. Li, J. Sun, and H.-y. Shum. Video object cut and paste. In *ACM Transactions on Graphics*, volume 24, pages 595–600, 2005.

[71] Y. Li, J. Sun, C.-K. Tang, and H.-Y. Shum. Lazy snapping. *ACM Trans. Graph.*, 23(3):303–308, 2004.

[72] Z. Liang and L. S. Davis. Closely coupled object detection and segmentation. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, volume 1, pages 454–461 Vol. 1, 2005.

[73] D. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 20:91–110, 2003.

[74] M. McGuire, W. Matusik, H. Pfister, J. F. Hughes, Fr #233, and d. Durand. Defocus video matting, 2005. 1073231 567-576.

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

[76] T. Mitsunaga, T. Yokoyama, and T. Totsuka. Autokey: human assisted key extraction. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 265–272, New York, NY, USA, 1995. ACM.

[77] J. Moehrmann and G. Heidemann. Automatic trajectory clustering for generating ground truth data sets. In *Proceedings of SPIE Visual Communications and Image Processing,VCIP*, 2010.

[78] F. Moscheni, S. Bhattacharjee, and M. Kunt. Spatio-temporal segmentation based on region merging. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(9):897–915, 1998.

[79] J. M. Odobez and P. Bouthemy. Detection of multiple moving objects using multiscale mrf with camera motion compensation. In *Image Processing, 1994. Proceedings. ICIP-94., IEEE International Conference*, volume 2, pages 257–261 vol.2, 1994.

[80] J. M. Odobez and P. Bouthemy. Mrf-based motion segmentation exploiting a 2d motion model robust estimation. In *Image Processing, 1995. Proceedings., International Conference on*, volume 3, pages 628–631 vol.3, 1995.

[81] J.-M. Odobez and P. Bouthemy. Direct incremental model-based image motion segmentation for video analysis. *Signal Process.*, 66(2):143–155, 1998. 284523.

[82] J. M. Odobez, P. Bouthemy, H. H. Li, S. Sun, and H. Derin. Separation of moving regions from background in an image sequence acquired with a mobile camera. In *Video Data Compression for Multimedia Computing*, pages 283–311. Kluwer Academic Publishers, 1997.

[83] D. O'Regan and A. Kokaram. Implicit spatial inference with sparse local features. In *Image Processing, 15th IEEE International Conference on*, pages 2388–2391, Oct. 2008.

[84] N. Papamarkos, A. E. Atsalakis, and C. P. Strouthopoulos. Adaptive color reduction. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 32(1):44–56, 2002.

[85] S. Paris. Edge-preserving smoothing and mean-shift segmentation of video streams. In *ECCV '08: Proceedings of the 10th European Conference on Computer Vision*, pages 460–473, Berlin, Heidelberg, 2008. Springer-Verlag.

[86] S. Paris and F. Durand. A topological approach to hierarchical segmentation using mean shift. In *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on*, pages 1–8, 2007.

[87] A. P. Pentland and T. Darrell. Cooperative robust estimation using layers of support. *T.R. 163, MIT Media Lab, Vision and Modeling Group*, 1991.

[88] A. P. Pentland and T. Darrell. Cooperative robust estimation using layers of support. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 17(5):474–487, 1995.

[89] C. Piciarelli, G. L. Foresti, and L. Snidara. Trajectory clustering and its applications for video surveillance. In *Advanced Video and Signal Based Surveillance, 2005. AVSS 2005. IEEE Conference on*, pages 40–45, 2005.

[90] F. Pitié, S. A. Berrani, A. Kokaram, and R. Dahyot. Off-line multiple object tracking using candidate selection and the viterbi algorithm. In *Image Processing, 2005. ICIP 2005. IEEE International Conference on*, volume 3, pages III–109–12, 2005.

[91] A. Pooransingh, C. A. Radix, and A. Kokaram. The path assigned mean shift algorithm: A new fast mean shift implementation for colour image segmentation. In *Image Processing, 2008. ICIP 2008. 15th IEEE International Conference on*, pages 597–600, 2008.

[92] J. Potter. Scene segmentation using motion information. In *Comput. Graphics Image Processing*, volume 6, pages 558–581, 1977.

[93] B. Price, B. Morse, and S. Cohen. Livecut: Learning-based interactive video segmentation by evaluation of multiple propagated cues. In *Proc. of ICCV International Workshop on Video-oriented Object and Event Classification (VOEC)*, Kyoto, Japan, 2009. IEEE Computer Society.

[94] S. Pundlik and S. Birchfield. Motion segmentation of sparse feature points at any speed. In *British Machine Vision Conference*, Edinburgh, Scotland, 2007.

[95] S. J. Pundlik and S. T. Birchfield. Real-time motion segmentation of sparse feature points at any speed. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 38(3):731–742, 2008.

[96] Y. Quan, A. Thangali, V. Ablavsky, and S. Sclaroff. Multiplicative kernels: Object detection, segmentation and pose estimation. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8, 2008.

[97] S. R. Rao, R. Tron, R. Vidal, and Y. Ma. Motion segmentation via robust subspace separation in the presence of outlying, incomplete, or corrupted trajectories. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, 0:1–8, 2008.

[98] R. A. Redner and H. F. Walker. Mixture densities, maximum likelihood and the em algorithm. *SIAM Review*, 26(2):195–239, 1984.

[99] C. Rhemann, C. Rother, W. Jue, M. Gelautz, P. Kohli, and P. Rott. A perceptually motivated online benchmark for image matting. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 1826–1833, 2009.

[100] D. Ring and A. Kokaram. Feature-cut: Video object segmentation through local feature correspondences. In *Proc. of ICCV International Workshop on Video-oriented Object and Event Classification (VOEC)*, Kyoto, Japan, September 2009. IEEE Computer Society.

[101] J. Rissanen. A universal prior for integers and estimation by minimum description length. *Annals of Statistics*, 11(2):416–431, 1983.

[102] C. Rother, V. Kolmogorov, and A. Blake. "grabcut": interactive foreground extraction using iterated graph cuts. *ACM Trans. Graph.*, 23(3):309–314, 2004. 1015720.

[103] M. A. Ruzon and C. Tomasi. Alpha estimation in natural images. In *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on*, volume 1, pages 18–25 vol.1, 2000.

[104] P. Sand and S. Teller. Particle video: Long-range motion estimation using point trajectories. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 2, pages 2195–2202, 2006.

[105] H. S. Sawhney and S. Ayer. Compact representations of videos through dominant and multiple motion estimation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 18(8):814–830, 1996.

[106] P. Scheunders. A comparison of clustering algorithms applied to color image quantization. *Pattern Recognition Letters*, 18(11-13):1379–1384, 1997. doi: DOI: 10.1016/S0167-8655(97)00116-5.

[107] H.-Y. Shum, J. Sun, S. Yamazaki, Y. Li, and C.-K. Tang. Pop-up light field: An interactive image-based modeling and rendering system. *ACM Trans. Graph.*, 23(2):143–162, 2004. 990005.

[108] J. Sivic, F. Schaffalitzky, and A. Zisserman. Object level grouping for video shots. *Int. J. Comput. Vision*, 67(2):189–210, 2006.

[109] P. Smith, T. Drummond, and R. Cipolla. Layered motion segmentation and depth ordering by tracking edges. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(4):479–494, 2004.

[110] W. B. Thompson. Combining motion and contrast for segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, pages 543–549, 1980.

[111] P. J. Toivanen. New geodesic distance transforms for gray-scale images. *Pattern Recognition Letters*, 17(5):437–450, 1996. doi: DOI: 10.1016/0167-8655(96)00010-4.

[112] R. Toldo and A. Fusiello. Robust multiple structures estimation with j-linkage. In *Proceedings of the 10th European Conference on Computer Vision: Part I*, Marseille, France, 2008. Springer-Verlag. 1478436 537-547.

[113] C. Tomasi and T. Kanade. Shape and motion from image streams: a factorization method - part 3 detection and tracking of point features. Technical report, Computer Science Department, April 1991.

[114] C. Tomasi and T. Kanade. Shape and motion from image streams: a factorization method - full report on the orthographic case. Technical report, International Journal of Computer Vision, 1992.

[115] P. H. S. Torr. Geometric motion segmentation and model selection. *Phil. Trans. Royal Society of London A*, 356:1321–1340, 1998.

[116] P. H. S. Torr, R. Szeliski, and P. Anandan. An integrated bayesian approach to layer extraction from image sequences. In *Computer Vision, The Proceedings of the 7th IEEE International Conference on*, volume 2, pages 983–990 vol.2, 1999.

[117] R. Tron and R. Vidal. A benchmark for the comparison of 3-d motion segmentation algorithms. In *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on*, pages 1–8, 2007.

[118] R. Vidal and R. Hartley. Motion segmentation with missing data using powerfactorization and gpca. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 2, pages II–310–II–316 Vol.2, June-2 July 2004.

[119] R. Vidal, R. Tron, and R. Hartley. Multiframe motion segmentation with missing data using powerfactorization and gpca. *International Journal of Computer Vision*, 79(1):85–105, 2008.

[120] R. Vidal, M. Yi, and S. Sastry. Generalized principal component analysis (gpca). In *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, volume 1, pages I–621–I–628 vol.1, 2003.

[121] R. Vidal, M. Yi, and S. Sastry. Generalized principal component analysis (gpca). *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(12):1945–1959, 2005.

[122] P. Villegas and X. Marichal. Perceptually-weighted evaluation criteria for segmentation masks in video sequences. *Image Processing, IEEE Transactions on*, 13(8):1092–1103, 2004.

[123] J. Wang, P. Bhat, R. A. Colburn, M. Agrawala, and M. F. Cohen. Interactive video cutout. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, pages 585–594, New York, NY, USA, 2005. ACM.

[124] J. Wang and M. F. Cohen. An iterative optimization approach for unified image segmentation and matting. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, volume 2, pages 936–943 Vol. 2, 2005.

[125] J. Wang and M. F. Cohen. Image and video matting: a survey. *Found. Trends. Comput. Graph. Vis.*, 3(2):97–175, 2007. 1391084.

[126] J. Wang, B. Thiesson, Y. Xu, and M. Cohen. Image and video segmentation by anisotropic kernel mean shift. In *European Conference on Computer Vision (ECCV)*, 2004.

[127] J. Y. A. Wang and E. H. Adelson. Representing moving images with layers. *Image Processing, IEEE Transactions on*, 3(5):625–638, 1994.

[128] Y. Weiss and E. H. Adelson. A unified mixture framework for motion segmentation: incorporating spatial coherence and estimating the number of models. In *Computer Vision and Pattern Recognition, 1996. Proceedings CVPR '96, 1996 IEEE Computer Society Conference on*, pages 321–326, 1996.

[129] J. Wills, S. Agarwal, and S. Belongie. What went where [motion segmentation]. In *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, volume 1, pages I–37–I–44 vol.1, 2003.

[130] J. Wills, S. Agarwal, and S. Belongie. A feature-based approach for dense segmentation and estimation of large disparity motion. *International Journal of Computer Vision*, 68(2):125–143, 2006.

[131] J. Xiao and M. Shah. Accurate motion layer segmentation and matting. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, pages 698–703 vol. 2, 2005.

[132] J. Yan and M. Pollefeys. A general framework for motion segmentation: Independent, articulated, rigid, non-rigid, degenerate and non-degenerate. In *Computer Vision ECCV 2006*, 2006.

[133] M. Yi, H. Derksen, H. Wei, and J. Wright. Segmentation of multivariate mixed data via lossy data coding and compression. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(9):1546–1562, 2007.

[134] C. Yung-Yu, B. Curless, D. H. Salesin, and R. Szeliski. A bayesian approach to digital matting. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings*

of the 2001 IEEE Computer Society Conference on, volume 2, pages II–264–II–271 vol.2, 2001.

[135] B. Yuri. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23:1222–1239, 2001.