Thermal and colour data fusion for people detection and tracking

by

Pierre Joubert

Thesis presented in partial fulfilment of the requirements for the degree of Master of Science in Applied Mathematics in the Faculty of Science at Stellenbosch University

Department of Mathematical Sciences,
Stellenbosch University,
Private Bag X1, Matieland 7602, South Africa.

Supervisor: Dr W.H. Brink

March 2014

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Pierre Joubert
Date: March 2014

Abstract

In this thesis we approach the problem of tracking multiple people individually in a video sequence. Automatic object detection and tracking is non-trivial as humans have complex and mostly unpredictable movements, and there are sensor noise and measurement uncertainties present. We consider traditional object detection methods and decide to use thermal data for the detection step. This choice is supported by the robustness of thermal data compared to colour data in unfavourable lighting conditions and in surveillance applications. A drawback of using thermal data is that we lose colour information, since the sensor interprets the heat emission of the body rather than visible light. We incorporate a colour sensor which is used to build features for each detected object. These features are used to help determine correspondences in detected objects over time.

A problem with traditional blob detection algorithms, which typically consist of background subtraction followed by connected-component labelling, is that objects can appear to split or merge, or disappear in a few frames. We decide to add 'dummy' blobs in an effort to counteract these problems. We refrain from making any hard decisions with respect to the blob correspondences over time, and rather let the system decide which correspondences are more probable. Furthermore, we find that the traditional Markovian approach of determining correspondences between detected blobs in the current time step and only the previous time step can lead to unwanted behaviour. We rather consider a sequence of time steps and optimize the tracking across them. We build a composite correspondence model and weigh each correspondence according to similarity (correlation) in object features. All possible tracks are determined through this model and a likelihood is calculated for each. Using the best scoring tracks we then label all the detections and use this labelling as measurement input for a tracking filter.

We find that the window tracking approach shows promise even though the data we us for testing is of poor quality and noisy. The system struggles with cluttered scenes and when a lot of dummy nodes are present. Nonetheless our findings act as a proof of concept and we discuss a few future improvements that can be considered.

Opsomming

In hierdie tesis benader ons die probleem om verskeie mense individueel in ’n video-opname op te spoor en te volg. Outomatiese voorwerp-opsporing en -volging is nie-triviaal, want mense het komplekse en meestal onvoorspelbare bewegings, en daar is sensor-ruis en metingonsekerhede teenwoordig. Ons neem tradisionele voorwerp-opsporing metodes in ag en besluit om termiese data te gebruik vir die opsporingstap. Hierdie keuse word ondersteun deur die robuustheid van termiese data in vergelyking met kleur data in ongunstige lig-kondisies en in sekuriteitstoepassings. Die nadeel van die gebruik van termiese data is dat ons kleur inligting verloor, aangesien die sensor die hitte vrystelling van die liggaam interpreteer, eerder as sigbare lig. Ons inkorporeer ’n kleur-sensor wat gebruik word om die kenmerke van elke gevolgde voorwerp te bou. Hierdie kenmerke word gebruik om te help om ooreenkomste tussen opgespoorde voorwerpe te bepaal met die verloop van tyd.

’n Probleem met die tradisionele voorwerp-opsporing algoritmes, wat tipies bestaan uit agtergrond-aftrekking gevolg deur komponent-etikettering, is dat dit kan voorkom asof voorwerpe verdeel of saamsmelt, of verdwyn in ’n paar rame. Ons besluit om ‘flous’-voorwerpe by te voeg in ’n poging om hierdie probleme teen te werk. Ons weerhou om enige konkrete besluite oor opgespoorde voorwerpe se ooreenkomste met die verloop van tyd te maak, en laat die stelsel eerder toe om te besluit watter ooreenkomste meer waarskynlik is. Verder vind ons dat die tradisionele Markoviaanse benadering vir die bepaling van ooreenkomste tussen opgespoorde voorwerpe in die huidige tydstap en die vorige een kan lei tot ongewenste gedrag. Ons oorweeg eerder ’n reeks van tydstappe, of ’n venster, en optimeer die volg van voorwerpe oor hulle. Ons bou ’n saamgestelde ooreenstemmingsmodel en weeg elke ooreenstemming volgens die ooreenkoms (korrelasie) tussen voorwerpe se kenmerke. Alle moontlike spore word deur hierdie model bepaal en ’n waarskynlikheid word bereken vir elkeen. Die spore met die beste tellings word gebruik om al die opsporings te nommeer, en hierdie etikettering word gebruik as meting-inset vir ’n volgingsfilter.

Ons vind dat die venster-volg benadering belowend vaar selfs al is die invoerdata in ons toetse van swak gehalte en ruiserig. Die stelsel sukkel met besige tonele en wanneer baie flous-voorwerpe teenwoordig is. Tog dien ons bevindinge as ’n bewys van konsep en ons bespreek ’n paar verbeterings wat in die toekoms oorweeg kan word.

Acknowledgements

I would like to thank the LEDGER University Research Programme for funding this work, my supervisor Dr Willie Brink for going the extra mile when it was needed, my girlfriend Karma for all the unconditional love and support and my parents who gave me the opportunity to pursue further studies.

Contents

Chapter 1

Introduction

The problem of automatic object detection and tracking in digital video has been studied extensively in recent years, and significant advances have been made due to developments in computing power and technology. The appeal for research in this field stems from the broad range of applications, such as in security and surveillance [27, 15, 11], in production lines [15, 44, 1] and for people counting and behaviour analysis [31, 42].

Surveillance footage is readily available due to the high demand for security network implementations, but because of the sheer size of these networks the use of cost effective sensors such as closed-circuit television (CCTV) cameras are most common. Unfortunately, they usually provide poor quality (noisy) data, which makes the task of automatic object detection and tracking more difficult.

Tracking people is non-trivial even with good quality data, as humans have complex and mostly unpredictable movements and change appearance over time as they move. Moreover, the quality of object detection in video is directly related to the quality of the background model used or developed. This can be challenging, as the background can be busy. These factors coupled with noisy data can, and will at some point, lead to erroneous detections. It is then up to the tracking part of the system to sort out such errors.

A typical solution for automatic object detection and tracking consists of the following steps:

- background modelling,
- foreground detection,
- establishing the correspondences between newly detected blobs and those currently being tracked,
- updating the states of tracked objects through the use of some filter.

Before we proceed with brief overviews of these steps, we first discuss the data our system is designed for.

1.1 The use of thermal and colour data

Thermal sensors differ from colour sensors as they detect longer wavelengths that fall in the infrared band. These wavelengths correspond to the thermal radiation emitted by objects, and a thermal sensor is therefore a good candidate to use for people detection. One shortcoming colour (or visible light) sensors have with detection is that they discern only between colours. This implies that if a person has an item of clothing having the same colour as the background, it will be wrongfully identified as background. This is why we opt to use thermal data for the detection, as it discerns between heat emitted by objects of interest and the (presumably) cooler background. This enables us to detect objects equally well at night time, in hazy lighting conditions and sometimes even when objects are visually occluded. This is especially useful for security and surveillance implementations, as is shown in Figure 1.1 where thermal and colour data are compared for the same scenes.

A significant shortcoming of using thermal data is that all the detected objects may have more-or-less the same appearance. We therefore use colour data for object classification rather than object detection as it is more informative than thermal data. The idea is to detect blobs in the thermal image, and map their locations to the colour image. This enables us to

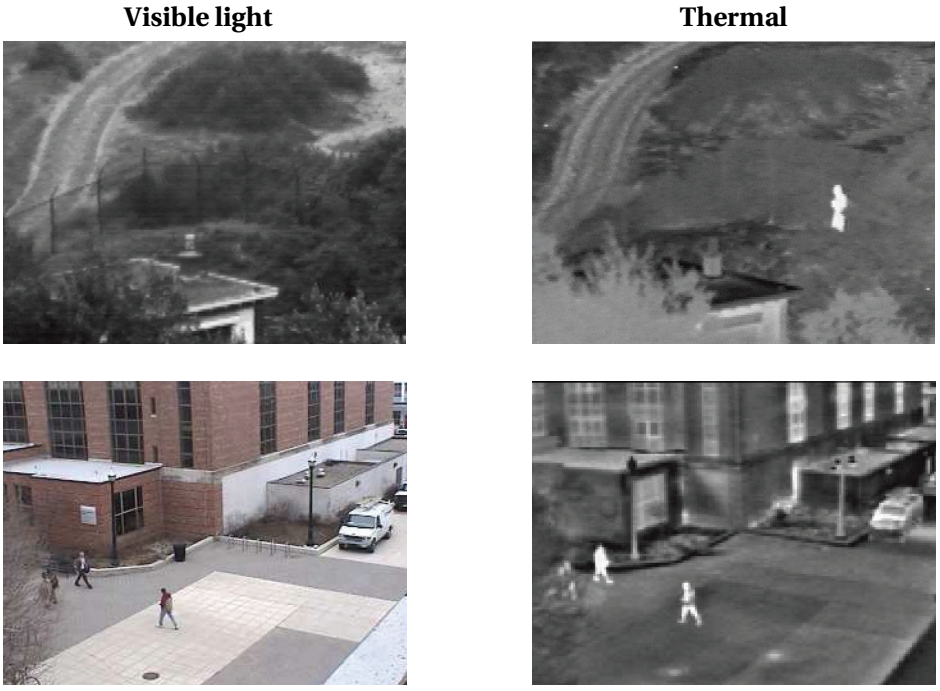


Figure 1.1: *Different scenarios indicating the superior object detection capabilities of thermal data compared to colour. The left column shows the visible light image compared to the corresponding thermal image (right column) of the same scene. (images from [50])*

determine colour features for each blob. Having features for each blob enables us to compare and determine how similar different blobs (over time) are, so that they can be tracked.

1.2 Overview of tracking filters

Target tracking originated in the 1950s and 1960s where the need arose to track blips in radio detection and ranging (RADAR) signals. From this spurred numerous different algorithms, of which the Kalman filter endured to become the most popular.

The filters we consider in this thesis are all based on similar principles and their respective equations are divided into two groups, namely the prediction step and the measurement step. Which filter we use to track objects over time depends on how we can model the expected noise, the relationship of the object states and measurements over time, and the way in which states are expected to change over time. These issues will be discussed in Chapter 3.

The object states that are to be tracked can be the centre of a group of pixels, their outline (or silhouette), four points defining a bounding box, or in fact anything that describes the location and/or shape of that object. The state vector can also be expanded to incorporate velocities of these attributes.

A tracking filter operates by predicting an estimate $\hat{\mathbf{x}}_t$ for the current state at time t, based on the previous state $\mathbf{x}_{t-1}$. This estimate is then updated by some measurement $\mathbf{z}_t$ of the object to produce an optimized state $\mathbf{x}_t$. This two-step process is illustrated in Figure 1.2, where a previous state, its prediction, a new measurement and the resulting update are shown.

There can be uncertainties and noise present in the prediction model and measurements of the object state. The filter should account for such uncertainties, and prior knowledge of

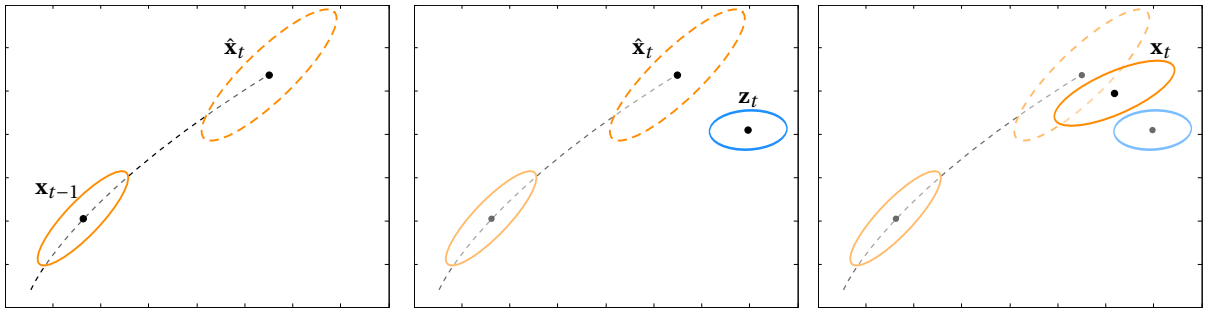


Figure 1.2: *The basic methodology used by the tracking filters we consider in this thesis. Figure (a) shows the previous state $\boldsymbol{x}_{t-1}$ with its projection $\hat{\boldsymbol{x}}_t$, together with their corresponding error ellipses (indicating the size of uncertainty). The projection is used as an estimate for the state at the current time step. After receiving a measurement $\boldsymbol{z}_t$, shown in (b), the state is updated to $\boldsymbol{x}_t$, as shown in (c).*

the expected behaviour of the system can result in different attributes being trusted more, or less.

The accuracy of the tracker hinges on the accuracy of the data it receives, and is therefore directly dependent on the quality of the measurements made. If the object is partially detected or wrongfully corresponded it could cause the tracker to become unstable. We will therefore give particular focus on the object detection and correspondence problem. Detection is discussed next.

1.3 Overview of object detection

The first step in object detection is background modelling. This is arguably the most important step in object detection, as an inaccurate background model will result in inaccurate foreground detection. The background model should ideally enable detection through busy scenes, incorporate static objects and be able to handle changing lighting conditions as well as repetitive motion such as sensor noise.

We will discuss the implementation of a few background subtraction methods in Chapter 4. They range from basic methods such as frame differencing and running Gaussian averaging to more advanced methods such as Gaussian mixture models.

Having built a background model we detect foreground pixels by comparing a new image frame pixel-wise with the background model. If any pixel in the new frame differs sufficiently from the corresponding background model pixel it is considered to be foreground. It can be beneficial to perform some morphological operations to clean up and smooth the detected foreground pixels. We may then group neighbouring pixels together by means of connected-component labelling. This results in individually detected foreground blobs to be tracked over time.

Object detection can be performed on video data, and for this thesis we are adopting an image fusion approach that combines colour and thermal imagery and uses the strengths of each in an attempt to produce a more robust detection and tracking system.

1.4 Overview of object correspondence over time

In order to track multiple objects it is imperative to find correspondences between objects detected at the current time step and those detected and tracked at previous time steps. It is easier to interpret the behaviour of the correspondences between blobs over time by representing them in the form of a graph (or network). We may represent every detected blob at each time step as a vertex and an edge connects two vertices in consecutive time steps if there is sufficient evidence that they are detections of the same object.

When determining correspondences over time we may, and most probably will, encounter certain unwanted detection errors. These can be due to an inaccurate background model, sensor noise, occlusions or close proximity of different objects. Two of the detection errors we consider in this thesis are splits and merges, and their occurrence in thermal detection is shown in Figure 1.3. It is these kinds of scenarios that we aim to smooth out and fix in this thesis.

1.4.1 Using dummy nodes

We approach this problem of incorrect detections by introducing, what we consider might be, the correct detection. This is done by adding dummy nodes at the current time step. When a split is detected, that is one blob in the previous time step is adjacent to two or more blobs in the current time step, we add a split-dummy that essentially ignores this split. This dummy is the superposition of all the newly detected blobs involved in the split. It is evident that this addition would be beneficial to the tracking system if we consider frames 2 and 3 in Figure 1.3 (a).

Similar reasoning follows for a merge, where two or more blobs in the previous time step are adjacent to a single blob at the current time step. Here we add a merge-dummy at the current time step for each of the blobs in the previous time step involved in the merge to counteract that merge and once again separate them. The benefit of doing so is evident when viewing frames 2 and 3 in Figure 1.3 (b). The addition of these dummy nodes is illustrated graphically in Figure 1.4.

Such newly added dummies are then handled just like any other detection. We calculate fea-

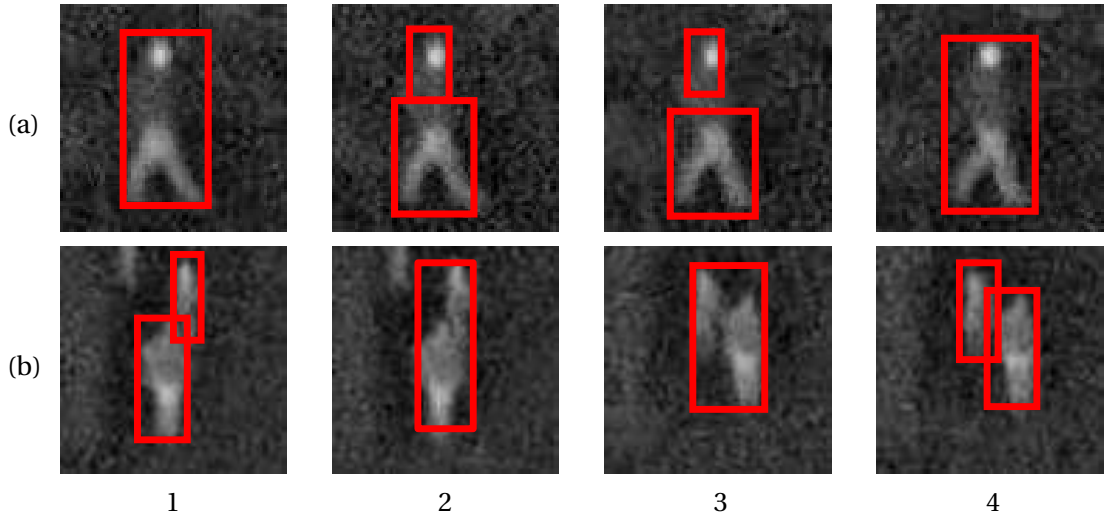
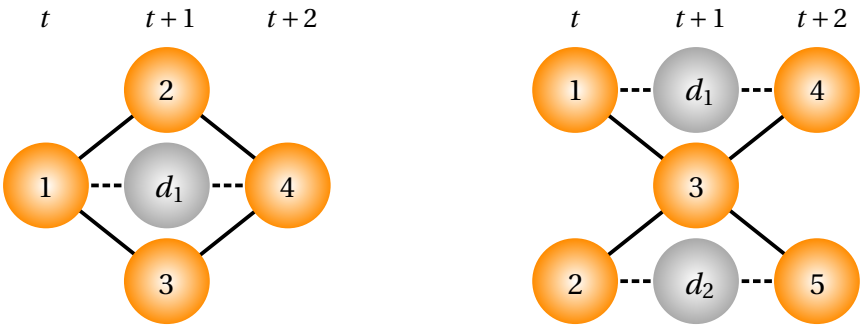


Figure 1.3: *Examples of two typical detection sequences in thermal data. The sequence in (a) shows a split where a person is walking from left to right. The sequence in (b) shows a merge as two people walk past one another, the one from below and the other from the top.*



(a) split (b) merge

Figure 1.4: *Graphical representation of the addition of dummy nodes to counteract incorrect blob detection we may encounter over time. In (a) vertex 1 splits and we add a split-dummy, while in (b) vertices 1 and 2 merge and we add a merge-dummy for each of them.*

tures (using the colour data) and determine correspondences to detected blobs in the previous time step. The usefulness of adding dummy blobs is that we allow the system to choose which correspondences are more likely: the original split or merge, or the newly added dummies. How the system decides which correspondences are most likely is the basis of our window tracking scheme, outlined next.

1.4.2 Window tracking

Object tracking over time is traditionally done in a Markovian fashion, by considering only the current and previous time steps, which makes the handling of splits and merges very difficult. We propose an approach where, instead of considering only the current and previous detections, we rather look at a sequence of time steps.

This window tracking scheme works by scanning ahead, detecting blobs at each time step in the window, adding dummy blobs where necessary and determining correspondences as before. We then consider the entire window and build a composite multipartite graph, and enumerate all possible tracks through it. Every edge in the composite graph is weighed according to a similarity measure between the feature vectors of the adjacent blobs. These weights are used to calculate a likelihood for each track. We then use the best scoring tracks to determine the correspondences to be used as measurements for each object's tracking filter.

1.5 Our approach

The main focus of this thesis is the handling of the above mentioned erroneous detections. We use an image fusion approach by incorporating the best attributes of thermal and colour data which, when combined, should result in a more accurate system than when either one is used separately. We use the thermal data for object detection, as it is particularly well-suited

for the task of detecting people. One drawback of using thermal data is that we lose visual information. We therefore use the colour data to build a feature vector for each detected blob which is used in calculating similarity measures between blobs over time.

We begin by building a background model for the thermal data. Foreground pixels are determined by using a background subtraction method. Background subtraction leads to clustered foreground pixels which, with some morphological operations and connected-component labelling, can be grouped together to form foreground blobs. We determine all possible correspondences between current blobs and those detected at previous time steps.

We then introduce dummy blobs in an attempt to correct any possibly wrong correspondences arising from splits and merges. We let the system decide which collection of correspondences is more probable over a window of previous frames. With this window tracking method we incorporate similarity measures between the colour feature vectors of detected and dummy blobs as an additional classification component to aid the decision making. The similarities essentially weigh the correspondences.

We formulate a set of rules which are used to determine all the possible tracks through the window of frames. Using the similarity measures we formulate a likelihood function that is used to score each possibility. The correspondences resulting from the track with the highest likelihood are then used as measurements for the tracking filter to update the states of the objects. The progression of our system is summarized in Figure 1.5.

1.6 Related work

When researching different tracking algorithms we found that they can either fall into distinct categories of techniques used, or make use of a combination of techniques. Single camera tracking can be divided roughly into three categories, namely discrete tracking, contour tracking and region-based tracking. All of these techniques follow the principles similar to those given at the start of the chapter: build a background model, detect foreground, establish correspondences between detected features over time and update the states of the tracked objects through the use of a filter.

Discrete tracking algorithms represent objects as discrete features such single points, sets of points, lines or edges. There exist many edge detection algorithms, such as Canny [7], Nalwa [36], Iverson [20] and Bergholm [4]. Discrete features have been used successfully in many systems [13, 28, 19], but a drawback is that these approaches discard potentially useful information such as colour and texture.

Contour trackers are developed to include colour and texture information. Intensity-based methods focus on the change within a window surrounding a point of interest [35]. Here a point of interest (POI) is any point where the surrounding image region changes in two di-

mensions, such as corners which can be identified using a Harris corner detector [16]. These POIs can be expanded to also include points within image regions containing distinctive texture patterns. One of the most popular contour feature extractors is the scale invariant feature transform (SIFT) by Lowe [29].

Both discrete and contour trackers consider only a single point or the deviation about a single point. Region-based trackers rather consider features that describe a larger image region. These regions are commonly referred to as blobs, as the connected foreground pixels form clusters that roughly define targets. When considering blobs the use of colour histograms and histograms of oriented gradients (HOG) are popular [10, 56, 48, 38], and consider the region occupied by the detected blob. The use of normalized histograms is also popular due to their simplicity, effectiveness and efficiency as they are invariant to scale and translation, and we also make use of this approach.

All these methods involve a single visible light sensor. Other multi-sensor tracking algorithms exist [1, 39, 25, 53], where multiple sensors are calibrated and synchronized, and the resulting detections fused to produce a final tracking result. The idea of fusing multiple sensors gives rise to the concept of using more than one type of sensor and fusing the results of them. The reader is referred to a recent work edited by Ukimura [50] for detailed discussions on image fusion. These principles can be used in fusing detection and tracking data, such as combining radar and infrared [33], range and doppler data [12] and visible light and infrared [41].

1.7 Thesis outline

This thesis starts by covering some essential background theory. As we are working with more than one sensor we begin by discussing sensor calibration in Chapter 2. We also consider a method of inferring a relationship between sensors when it is not possible to calibrate them. We use this relationship in mapping the locations of detected blobs in the thermal image to the colour image.

Next we introduce the different tracking filters applicable to our implementation in Chapter 3. Each tracked object has a filter that optimally combines a prediction of its previous state and a newly acquired measurement. To be able to detect (measure the location of) objects we first need to build a background model, and this is described in Chapter 4.

In determining which blobs correspond to which over time, we initially enforce adapted versions of the locality and parent constraints of Masoud and Papanikolopoulos [31]. We adapt the parent constraint in Chapter 5 by incorporating the use of feature vectors. If a split or merge is detected we add dummy nodes to the graph. By doing so we force the “correct” detection into the graph. Each edge in the graph is weighed by a correlation coefficient between the feature vectors of adjacent vertices.

We show that the traditional Markovian approach to object correspondence struggles to handle splits and merges and we introduce our window tracking scheme in Chapter 6. It works by considering a sequence of frames and building a composite graph that incorporates all the detections, added dummies and correspondences. We then determine all possible tracks through the composite graph and we formulate a likelihood function to score each of these. The best scoring track is used as input for the measurement update of the object's associated filter.

Experimental results are given in Chapter 7 and conclusions and future work are discussed in Chapter 8.

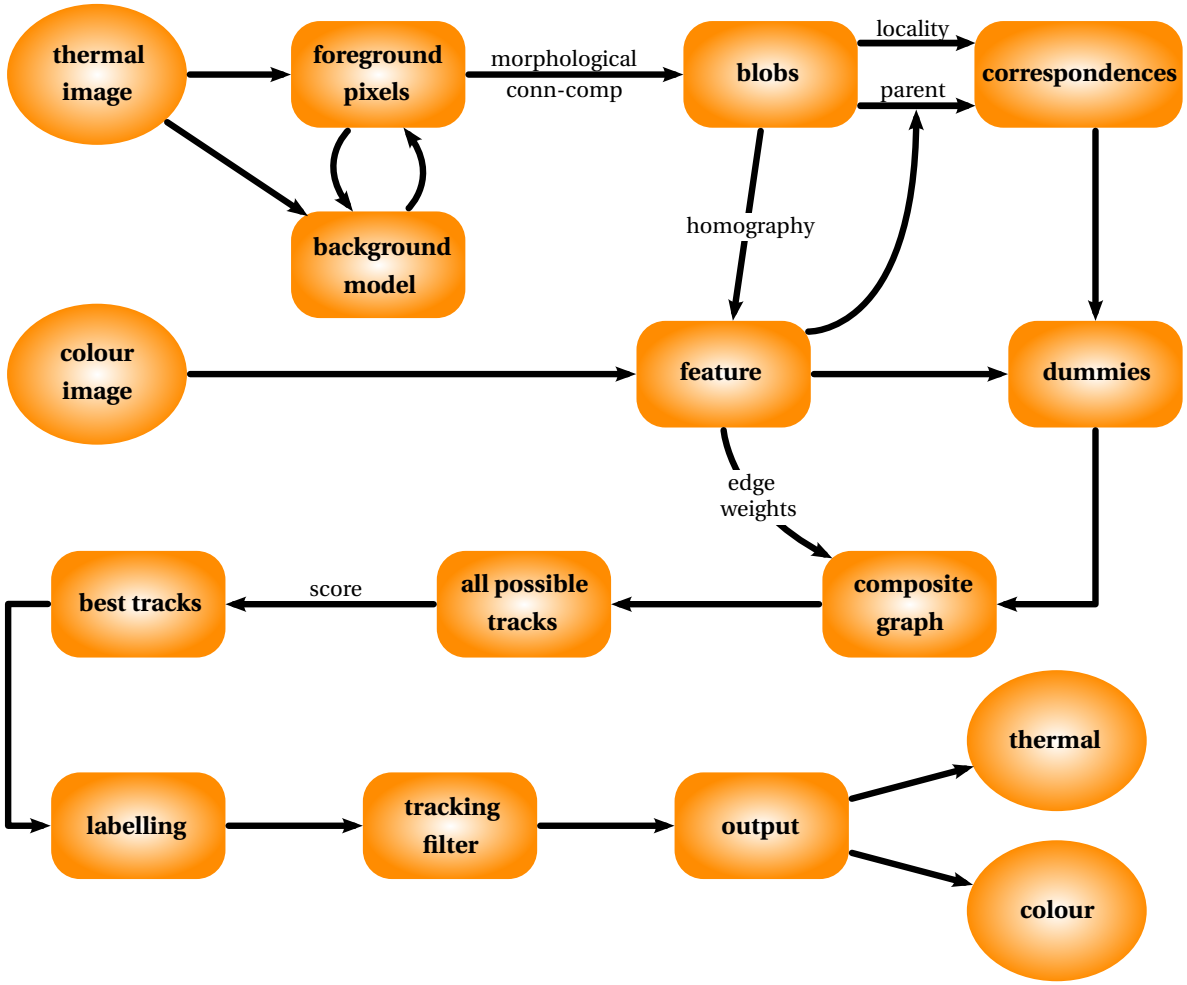


Figure 1.5: *Flowchart representation of the steps involved in our tracking system.*

Chapter 2

Sensor calibration

We use thermal and colour data to produce a more accurate detection and tracking system, where the thermal data is used for detection and the colour data for building features. We therefore assume to have access to a synchronized thermal and colour camera pair, observing the same scene, and we need a method to calculate a mapping of the location of each detection from the thermal image to the colour image. This can be achieved by calibrating the sensors.

Calibrating a single sensor provides useful intrinsic information, such as the focal length, the principal point and the image centre. Calibrating two cameras yields extrinsic information, that is the 3-dimensional relationship between the sensors. The intrinsic and extrinsic parameters place epipolar geometric constraints on correspondences between the two images. However, for our application where people are expected to remain on a ground plane (or a small set of planes) it is possible to infer a relationship between the two sensors without fully calibrating the cameras, as will be shown in Section 2.2. The reader is referred to the literature [17, 46, 22] for more in-depth discussions on camera calibration.

2.1 Stereo camera calibration

If we consider the thermal camera and colour camera as a stereo pair, we can perform standard stereo calibration on them. The widely used pinhole camera model assumes that all light rays project through a single point onto the image plane. This point is called the camera centre, and the image plane is located a certain focal length distance behind it. Equivalently, we may consider the image plane to be in front of the camera centre, as illustrated in Figure 2.1 (a).

A point $\mathbf{X} = (X, Y, Z)^T$ in 3D world coordinates is projected to a point $\mathbf{x} = (u, v)^T$ in 2D image coordinates. Using *homogeneous coordinates* we define this mapping as

$$\mathbf{x} = \mathbf{PX}, \tag{2.1.1}$$

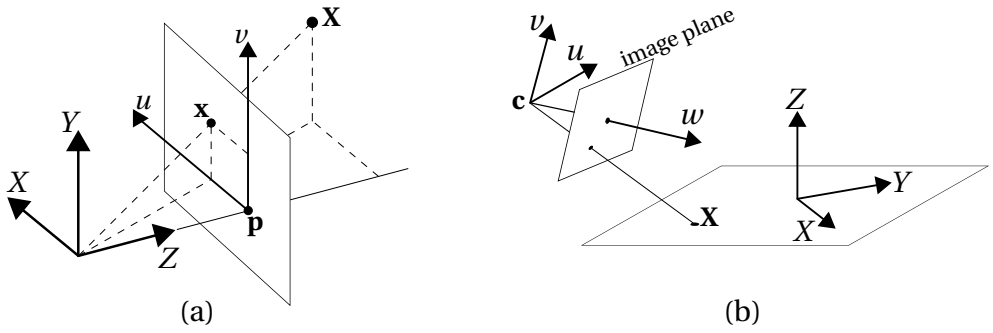


Figure 2.1: *The pinhole camera model, in (a), shows a point* $\boldsymbol{X}$ *in world coordinates projected onto point* $\boldsymbol{x}$ *on the image plane. The XYZ world coordinate system is related to the uvw camera coordinate system by a rotation and translation, as illustrated in (b).*

where $\mathbf{P}$ is the 3×4 *camera matrix* [17]. This matrix allows for a possible offset in world coordinates and camera coordinates, as shown in Figure 2.1 (b)

In our system we have two cameras, and therefore two camera matrices of the form

$$\mathbf{P}_1 = \mathbf{K}_1[\mathbf{I}|\mathbf{0}], \tag{2.1.2}$$

$$\mathbf{P}_2 = \mathbf{K}_2\mathbf{R}[\mathbf{I}|-\mathbf{c}]. \tag{2.1.3}$$

Here $\mathbf{I}$ is the 3×3 identity matrix, $\mathbf{K}_1$ and $\mathbf{K}_2$ are the so-called *calibration matrices* of the two cameras that include the various intrinsic parameters, and $\mathbf{R}$ and $\mathbf{c}$ give the orientation and position of the second camera relative to the first. If these two camera matrices are known, we can project any point in world coordinates onto the two image planes.

The calibration of such a stereo rig involves the determination of $\mathbf{P}_1$ and $\mathbf{P}_2$. This is typically done offline, before the system is deployed, and can be achieved easily with existing software [5, 54].

We want to be able to map a detected point in the thermal image to its corresponding point in the colour image. If we have access to all the calibration parameters, we can restrict this search to a single *epipolar line* in the colour image [17]. Even though the search space is restricted to a line, the inherently ambiguous nature of colour data makes the search difficult and results unreliable. Moreover, we may not have access to and control over the camera system, which can make an offline calibration procedure impossible.

Therefore, instead of attempting to find complete calibration parameters that still do not guarantee error-free mapping between the two image planes, we make some assumptions about the scene and how people can move in it.

2.2 Plane-to-plane homographies

A homography, or projective transformation, is an invertible mapping $h : \mathbb{P}^2 \mapsto \mathbb{P}^2$ such that three points $\{\mathbf{u}, \mathbf{v}, \mathbf{w}\}$ lie on a straight line if and only if $\{h(\mathbf{u}), h(\mathbf{v}), h(\mathbf{w})\}$ also lie on a straight line. It is known [17] that the mapping h is a projective transformation if and only if there exists a nonsingular, homogeneous 3×3 matrix $\mathbf{H}$ such that for any point $\mathbf{x} \in \mathbb{P}^2$ it is true that

$$h(\mathbf{x}) = \mathbf{H}\mathbf{x}. \tag{2.2.1}$$

Equation (2.2.1) allows us to write a projective transformation that maps the homogeneous coordinates $\mathbf{x} = (x, y, 1)^T$ to the homogeneous coordinates $\mathbf{x}' = (x', y', 1)^T$ in terms of matrix multiplication as

$$\begin{pmatrix} x'_1 \\ x'_2 \\ x'_3 \end{pmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}, \tag{2.2.2}$$

or more compactly as $\mathbf{x}' = \mathbf{H}\mathbf{x}$, such that $x' = x'_1 / x'_3$ and $y' = x'_2 / x'_3$.

It should be clear that, under the pinhole camera model, any straight line in world coordinates will project to a straight line in the image. So, when two cameras observe the same scene, straight lines in one image will correspond to straight lines in the other image. Moreover, if we consider points on a plane in world coordinates we realize that their projections onto the two image planes must also be related according to a fixed homography.

It is reasonable to assume, since we are tracking people, that people's movements are restricted to a planar surface. Furthermore, we know that there exists a homography that maps points detected in the thermal image, known to lie on that plane, to corresponding points in the colour image. If we can find that homography, i.e. the 3×3 matrix in (2.2.2), we can use it to map the location of a detected object in the thermal image to the colour image.

In order to calculate the homography we choose correspondences of the form $(x, y, 1) \leftrightarrow (x', y', 1)$. Here (x, y) indicates the pixel coordinates of a point in the thermal image, and (x', y') its correspondence in the colour image. We need only a small number of such correspondences, and can select them manually (using for example strong corners visible in both images). The formula in (2.2.2) can be written in inhomogeneous form as

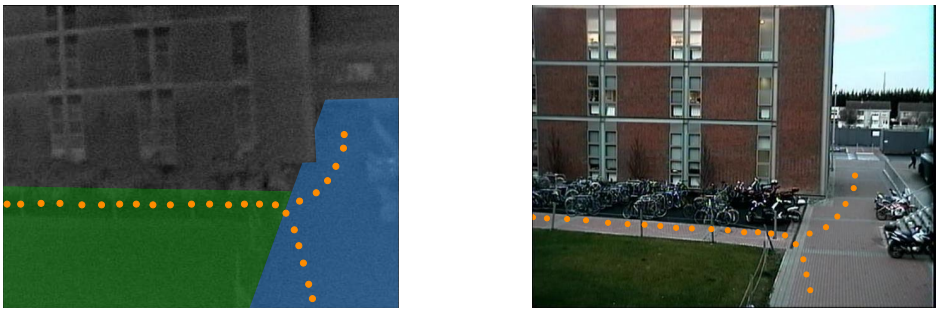
$$x' = \frac{x'_1}{x'_3} = \frac{h_{11}x + h_{12}y + h_{13}}{h_{31}x + h_{32}y + h_{33}}, \tag{2.2.3}$$

$$y' = \frac{x'_2}{x'_3} = \frac{h_{21}x + h_{22}y + h_{23}}{h_{31}x + h_{32}y + h_{33}}, \tag{2.2.4}$$

which lead to two equations, both linear in the unknown entries of $\mathbf{H}$:

$$x'\left(h_{31}x + h_{32}y + h_{33}\right) = h_{11}x + h_{12}y + h_{13}, \tag{2.2.5}$$

$$y'\left(h_{31}x + h_{32}y + h_{33}\right) = h_{21}x + h_{22}y + h_{23}. \tag{2.2.6}$$



(a) thermal (b) colour

Figure 2.2: *The result of calculating a planar mapping from the thermal to the colour image. As the walking surface is not flat we segmented it into two different planar polygons and calculated a homography for each. Points chosen in (a), spanning both surfaces, are mapped to the corresponding points shown in image (b).*

Equations (2.2.5) and (2.2.6) can be combined and written in terms of matrix multiplication as

$$\begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -x'x & -x'y & -x' \\ 0 & 0 & 0 & x & y & 1 & -y'x & -y'y & -y' \end{bmatrix} \begin{pmatrix} h_{11} \\ h_{12} \\ \vdots \\ h_{33} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}. \tag{2.2.7}$$

The mapping h is a projective transformation, which implies that $\mathbf{H}$ has eight degrees of freedom [17]. We therefore need four point correspondences $\mathbf{x}_i \leftrightarrow \mathbf{x}'_i$ to solve for $\mathbf{H}$, as each point correspondence provides two equations. It is important to note that no three of the point correspondences should be colinear. It is also possible, and usually advised, to use more than four point correspondences and solve the overdetermined system in a least-squares sense. This is done to minimize the measurement error introduced with manual point correspondence selection.

With four point correspondences we get the system of equations $\mathbf{Ah} = \mathbf{0}$, that is

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x'_1x_1 & -x'_1y_1 & -x'_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y'_1x_1 & -y'_1y_1 & -y'_1 \\ \vdots & & & & & \vdots & & & \vdots \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x'_4x_4 & -x'_4y_4 & -x'_4 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -y'_4x_4 & -y'_4y_4 & -y'_4 \end{bmatrix} \begin{pmatrix} h_{11} \\ h_{12} \\ \vdots \\ h_{32} \\ h_{33} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix}, \tag{2.2.8}$$

such that $\mathbf{A}$ is an 8×9 matrix, $\mathbf{h}$ is a 9×1 vector and $\mathbf{0}$ is a 8×1 vector. The vector $\mathbf{h}$ is therefore a vector in the null space of $\mathbf{A}$. One way of calculating $\mathbf{h}$ is to compute the SVD decomposition of $\mathbf{A}$, as $\mathbf{A} = \mathbf{U\Sigma V}^T$. The last column of $\mathbf{V}$ will be the basis for the null space of $\mathbf{A}$, and we can therefore choose $\mathbf{h}$ as the last column of $\mathbf{V}$. Having calculated $\mathbf{h}$ it is then possible to calculate the homography $\mathbf{H}$ by reshaping the 9×1 vector $\mathbf{h}$ into a 3×3 matrix $\mathbf{H}$.

It is important to note that not all scenes will be perfectly flat. One extension of this plane-to-

plane homography approach is to segment the walking surfaces into various planar regions and calculate a homography for each. Figure 2.2 shows an example of the dataset we use for experimentation [8] with a selection of points in the thermal image that are mapped to the corresponding points in the colour image with the use of homographies. Here we use two homographies, as indicated by the two different shadings in the first image.

Chapter 3

Tracking filters

When performing object detection and tracking in video, a key step is to implement a tracking filter. Data from a surveillance camera is usually noisy and of poor quality, which implies that there can be uncertainties associated with measurements of the objects. A filter should take these measurement uncertainties into consideration and combine them with some prediction of how objects move, to arrive at an optimal track. The type of filter implemented depends on the application. More specifically, it depends on how we model the noise that is expected, the relationship between measurements and states that are tracked (the *measurement* model) as well as the prediction of how states change over time (the *prediction* model).

If the noise can be modelled as additive zero-mean Gaussian, and the measurement and prediction models are linear, we can implement a *Kalman filter*. If the models are nonlinear but still have Gaussian noise, we can use an *extended Kalman filter*. If the models are nonlinear and the noise cannot be described adequately by Gaussian distributions, we can implement a *particle filter*.

In this chapter we discuss the three filters mentioned above. We refrain from deriving all the equations from basic principles as it is beyond the scope of this thesis. For this the reader is referred to the literature [24, 52, 3].

3.1 The Kalman filter

The Kalman filter is a recursive solution for the discrete data linear filtering problem [24]. It uses a *prediction-correction* approach by projecting the current state as estimation for the next time step, and then using (noisy) measurements as correction to update the state vector. The equations of the Kalman filter can therefore be separated into two groups, namely *time update* and *measurement update* equations. The time update equations are responsible for projecting the current state and error covariance estimates forward in time to obtain *a priori* estimates for the next time step. The measurement update equations are responsible for incorporating the new measurement into the *a priori* estimate to obtain an improved *a*

posteriori estimate.

The filter combines estimates of consecutive states $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N \mid \mathbf{x}_t \in \mathbb{R}^n\}$ of the tracked object together with corresponding measurements $\{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_N \mid \mathbf{z}_t \in \mathbb{R}^m\}$ to produce an optimized track. The filter assumes a Markov relation, as it uses only the previous state with the current measurement to determine a new estimate.

The state of the process, or rather how the filter expects the system to behave over time, is expressed as a linear difference equation

$$\mathbf{x}_t = \mathbf{A}\mathbf{x}_{t-1} + \mathbf{B}\mathbf{u}_{t-1} + \mathbf{w}_{t-1}, \tag{3.1.1}$$

and we assume the measurement relates to the state according to

$$\mathbf{z}_t = \mathbf{H}\mathbf{x}_t + \mathbf{v}_t. \tag{3.1.2}$$

The $n \times n$ matrix $\mathbf{A}$ in (3.1.1) relates the state at the previous time step $t-1$ to the current one at step t, the $n \times l$ matrix $\mathbf{B}$ relates the optional external input $\mathbf{u} \in \mathbb{R}^l$ to the state $\mathbf{x}$ and the $m \times n$ matrix $\mathbf{H}$ in (3.1.2) relates the state $\mathbf{x}_t$ to the measurement $\mathbf{z}_t$.

The variables $\mathbf{w}_{t-1}$ and $\mathbf{v}_t$ represent the process and measurement noise respectively. They are assumed to be independent and zero-mean Gaussian with covariances $\mathbf{Q}$ and $\mathbf{R}$, so

$$\mathbf{w}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}), \tag{3.1.3}$$

$$\mathbf{v}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{R}). \tag{3.1.4}$$

Note that the $n \times n$ process noise covariance $\mathbf{Q}$, the $m \times m$ measurement noise covariance $\mathbf{R}$, as well as the relation matrices $\mathbf{A}$ and $\mathbf{B}$ in (3.1.1) and $\mathbf{H}$ in (3.1.2), can change over time.

Next we define $\hat{\mathbf{x}}_t^- \in \mathbb{R}^n$ to be the *a priori* state estimate at step t given knowledge prior to step t, and $\hat{\mathbf{x}}_t \in \mathbb{R}^n$ to be the *a posteriori* state estimate at step t given the measurement $\mathbf{z}_t$. The *a priori* and *a posteriori* error estimates are defined as

$$\mathbf{e}_t^- = \mathbf{x}_t - \hat{\mathbf{x}}_t^-, \tag{3.1.5}$$

$$\mathbf{e}_t = \mathbf{x}_t - \hat{\mathbf{x}}_t, \tag{3.1.6}$$

with covariance estimates

$$\mathbf{P}_t^- = E\left[\mathbf{e}_t^- \mathbf{e}_t^{-T}\right], \tag{3.1.7}$$

$$\mathbf{P}_t = E\left[\mathbf{e}_t \mathbf{e}_t^T\right], \tag{3.1.8}$$

where $E[\xi]$ is the expected value of ξ. It can be shown [24] that the error covariance update equation in the time update step can be formulated as

$$\mathbf{P}_t^- = \mathbf{A}\mathbf{P}_{t-1}\mathbf{A}^T + \mathbf{Q}. \tag{3.1.9}$$

The goal of the measurement update step is to find an equation that computes an *a posteriori* state estimate $\hat{\mathbf{x}}_k$ as a linear combination of the *a priori* estimate $\hat{\mathbf{x}}_k^-$ and a weighted difference between the measurement $\mathbf{z}_t$ and the measurement prediction $\mathbf{H}\hat{\mathbf{x}}_t^-$, as

$$\hat{\mathbf{x}}_t = \hat{\mathbf{x}}_t^- + \mathbf{K}_t\left(\mathbf{z}_t - \mathbf{H}\hat{\mathbf{x}}_t^-\right). \tag{3.1.10}$$

The $n \times n$ matrix $\mathbf{K}_t$ is known as the Kalman gain factor and minimizes the *a posteriori* error covariance (3.1.8). This minimum error covariance is given by

$$\mathbf{P}_t = (\mathbf{I} - \mathbf{K}_t\mathbf{H})\mathbf{P}_t^-. \tag{3.1.11}$$

It can be shown [32, 6] that the gain factor $\mathbf{K}_t$ can be expressed as

$$\mathbf{K}_t = \mathbf{P}_t^-\mathbf{H}^T\left(\mathbf{H}\mathbf{P}_t^-\mathbf{H}^T + \mathbf{R}\right)^{-1}. \tag{3.1.12}$$

A way to interpret the effect of the gain factor is that as the measurement error covariance $\mathbf{R}$ decreases, the measurement $\mathbf{z}_t$ is trusted more and more while the predicted measurement $\mathbf{H}\hat{\mathbf{x}}_t^-$ is trusted less and less. On the other hand, as the *a priori* error estimate $\mathbf{P}_t^-$ decreases, the measurement $\mathbf{z}_t$ is trusted less and less while the predicted measurement $\mathbf{H}\hat{\mathbf{x}}_t^-$ is trusted more and more.

All the equations in the time and measurement update steps are summarized in Table 3.1.

Kalman filter equations	
Time update	$\hat{\mathbf{x}}_t^- = \mathbf{A}\hat{\mathbf{x}}_{t-1} + \mathbf{B}\mathbf{u}_{t-1} + \mathbf{w}_{t-1}$ $\mathbf{P}_t^- = \mathbf{A}\mathbf{P}_{t-1}\mathbf{A}^T + \mathbf{Q}$
Measurement update	$\mathbf{K}_t = \mathbf{P}_t^-\mathbf{H}^T(\mathbf{H}\mathbf{P}_t^-\mathbf{H}^T + \mathbf{R})^{-1}$ $\hat{\mathbf{x}}_t = \hat{\mathbf{x}}_t^- + \mathbf{K}_t(\mathbf{z}_t - \mathbf{H}\hat{\mathbf{x}}_t^-)$ $\mathbf{P}_t = (\mathbf{I} - \mathbf{K}_t\mathbf{H})\mathbf{P}_t^-$

Table 3.1: *Summary of equations used for each step in the Kalman filter.*

3.2 The extended Kalman filter

As mentioned in Section 3.1, the Kalman filter addresses the general problem of estimating the state $\mathbf{x} \in \mathbb{R}^n$ of a discrete-time controlled process that is governed by a linear difference equation. However, in many cases it might be necessary to formulate a nonlinear relationship. It is possible to adapt the Kalman filter to linearize about a current mean and covariance, and this adaptation produces the extended Kalman filter (EKF).

We can linearize the estimation around the current estimate by using the partial derivatives of the process and measurement functions. This can be thought of as a Taylor series approximation about the current estimate. Again we have a state vector $\mathbf{x} \in \mathbb{R}^n$, but the process is now governed by some *nonlinear* difference equation

$$\mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1}, \mathbf{w}_{t-1}), \tag{3.2.1}$$

and the measurement $\mathbf{z} \in \mathbb{R}^m$ is governed by

$$\mathbf{z}_t = h(\mathbf{x}_t, \mathbf{v}_t). \tag{3.2.2}$$

The random variables $\mathbf{w}_t$ and $\mathbf{v}_t$ again represent the process and measurement noise, as in (3.1.1) and (3.1.2). The nonlinear function f in the difference equation (3.2.1) relates the state at the previous time step $t-1$ to the state at the current time step t. It includes as parameters an optional external function $\mathbf{u}_{t-1}$ and the process noise $\mathbf{w}_{t-1}$. The nonlinear function h in the measurement equation (3.2.2) relates the state $\mathbf{x}_t$ to the measurement $\mathbf{z}_t$ and takes as additional input the measurement noise $\mathbf{v}_t$.

In order to estimate a process with nonlinear difference and measurement equations, we formulate new difference and measurement equations that linearize (3.2.1) and (3.2.2) about an estimate as

$$\mathbf{x}_t \approx \tilde{\mathbf{x}}_t + \mathbf{A}(\mathbf{x}_{t-1} - \hat{\mathbf{x}}_{t-1}) + \mathbf{W}\mathbf{w}_{t-1}, \tag{3.2.3}$$

$$\mathbf{z}_t \approx \tilde{\mathbf{z}}_t + \mathbf{H}(\mathbf{x}_t - \tilde{\mathbf{x}}_t) + \mathbf{V}\mathbf{v}_t, \tag{3.2.4}$$

where

- $\mathbf{x}_t$ and $\mathbf{z}_t$ are the state and measurement vectors,
- $\tilde{\mathbf{x}}_t$ and $\tilde{\mathbf{z}}_t$ are the predicted state and measurement vectors calculated by means of (3.2.1) and (3.2.2),
- $\hat{\mathbf{x}}_t$ is an *a posteriori* estimate of the state at step t,
- the variables $\mathbf{w}_{t-1}$ and $\mathbf{v}_t$ represent the process and measurement noise, as in (3.1.1) and (3.1.2),
- $\mathbf{A}$ is the Jacobian matrix containing partial derivatives of f with respect to $\mathbf{x}$:

$$A_{i,j} = \frac{\partial f_i}{\partial x_j}(\hat{\mathbf{x}}_{t-1}, \mathbf{u}_{t-1}, \mathbf{0}),$$

- $\mathbf{W}$ is the Jacobian matrix containing partial derivatives of f with respect to $\mathbf{w}$:

$$W_{i,j} = \frac{\partial f_i}{\partial w_j}(\hat{\mathbf{x}}_{t-1}, \mathbf{u}_{t-1}, \mathbf{0}),$$

- **H** is the Jacobian matrix containing partial derivatives of h with respect to $\mathbf{x}$:

$$H_{i,j} = \frac{\partial h_i}{\partial x_j}(\tilde{\mathbf{x}}_t, \mathbf{0}),$$

- and **V** is the Jacobian matrix containing partial derivatives of h with respect to $\mathbf{v}$:

$$V_{i,j} = \frac{\partial h_i}{\partial v_j}(\tilde{\mathbf{x}}_t, \mathbf{0}).$$

We also define the prediction state and measurement residuals as

$$\tilde{\mathbf{e}}_{\mathbf{x}_t} = \mathbf{x}_t - \tilde{\mathbf{x}}_t, \tag{3.2.5}$$
$$\tilde{\mathbf{e}}_{\mathbf{z}_t} = \mathbf{z}_t - \tilde{\mathbf{z}}_t. \tag{3.2.6}$$

We are trying to estimate the current state vector $\mathbf{x}_t$ in (3.2.5), which we do not yet have. We do, on the other hand, have access to the measurement $\mathbf{z}_t$ in (3.2.6) which we use to estimate $\mathbf{x}_t$. Using (3.2.5) and (3.2.6) we can write governing equations for an *error process* as

$$\tilde{\mathbf{e}}_{\mathbf{x}_t} \approx \mathbf{A}(\mathbf{x}_{t-1} - \hat{\mathbf{x}}_{t-1}) + \boldsymbol{\epsilon}_t, \tag{3.2.7}$$
$$\tilde{\mathbf{e}}_{\mathbf{z}_t} \approx \mathbf{H}\tilde{\mathbf{e}}_{\mathbf{x}_t} + \boldsymbol{\eta}_t, \tag{3.2.8}$$

where $\boldsymbol{\epsilon}_t$ and $\boldsymbol{\eta}_t$ are independent zero-mean Gaussian variables having covariance matrices $\mathbf{WQW}^T$ and $\mathbf{VRV}^T$. Here **W** and **V** are the Jacobian matrices defined above, and **Q** and **R** are the covariances defined in (3.1.3) and (3.1.4) respectively. The distributions of these random variables are

$$\tilde{\mathbf{e}}_{\mathbf{x}_t} \sim \mathcal{N}\left(\mathbf{0}, E\left[\tilde{\mathbf{e}}_{\mathbf{x}_t}\mathbf{e}_{\mathbf{x}_t}^T\right]\right), \tag{3.2.9}$$
$$\boldsymbol{\epsilon}_t \sim \mathcal{N}\left(\mathbf{0}, \mathbf{W}\mathbf{Q}_t\mathbf{W}^T\right), \tag{3.2.10}$$
$$\boldsymbol{\eta}_t \sim \mathcal{N}\left(\mathbf{0}, \mathbf{V}\mathbf{R}_t\mathbf{V}^T\right), \tag{3.2.11}$$

where $E[\xi]$ is the expected value of ξ. Notice that (3.2.7) and (3.2.8) are linear, and that they closely resemble the difference and measurement equations (3.1.1) and (3.1.2) of the Kalman filter. This motivates us to use the measurement residual $\tilde{\mathbf{e}}_{\mathbf{z}_t}$ in (3.2.6). All we need is an estimate of the prediction error $\tilde{\mathbf{e}}_{\mathbf{x}_t}$ given by (3.2.7). This can be achieved by using the prediction step of a separate Kalman filter. This prediction $\hat{\mathbf{e}}_t$ of $\tilde{\mathbf{e}}_{\mathbf{x}_t}$ can be used along with (3.2.5) to obtain the *a posteriori* state estimate for the original nonlinear process as

$$\hat{\mathbf{x}}_t = \tilde{\mathbf{x}}_t + \hat{\mathbf{e}}_t. \tag{3.2.12}$$

If we assume a perfect prediction, i.e. $\hat{\mathbf{e}}_t = \mathbf{0}$, the Kalman filter prediction used to estimate $\hat{\mathbf{e}}_t$ becomes

$$\hat{\mathbf{e}}_t = \mathbf{K}_t\tilde{\mathbf{e}}_{\mathbf{z}_t}. \tag{3.2.13}$$

By substituting the prediction (3.2.13) back into the state estimate (3.2.12) and making use of the measurement residual (3.2.6) we get

$$\begin{aligned}\hat{\mathbf{x}}_t &= \tilde{\mathbf{x}}_t + \mathbf{K}_t\tilde{\mathbf{e}}_{\mathbf{z}_t} \\ &= \tilde{\mathbf{x}}_t + \mathbf{K}_t(\mathbf{z}_t - \tilde{\mathbf{z}}_t),\end{aligned} \tag{3.2.14}$$

where we have removed the need to implement the Kalman prediction. Equation (3.2.14) can now be used as the final measurement update equation in the extended Kalman filter. The equations used for the time and measurement update steps are summarized in Table 3.2.

Extended Kalman filter equations	
Time update	$\hat{\mathbf{x}}_t^- = f(\hat{\mathbf{x}}_{t-1}, \mathbf{u}_{t-1}, \mathbf{0})$ $\mathbf{P}_t^- = \mathbf{A}_t\mathbf{P}_{t-1}\mathbf{A}_t^T + \mathbf{W}_t\mathbf{Q}_{t-1}\mathbf{W}_t^T$
Measurement update	$\mathbf{K}_t = \mathbf{P}_t^-\mathbf{H}_t^T\left(\mathbf{H}_t\mathbf{P}_t^-\mathbf{H}_t^T + \mathbf{V}_t\mathbf{R}_t\mathbf{V}_t^T\right)^{-1}$ $\hat{\mathbf{x}}_t = \hat{\mathbf{x}}_t^- + \mathbf{K}_t\left(\mathbf{z}_t - h(\hat{\mathbf{x}}_t^-, \mathbf{0})\right)$ $\mathbf{P}_t = (\mathbf{I} - \mathbf{K}_t\mathbf{H}_t)\mathbf{P}_t^-$

Table 3.2: *Summary of equations used for each step in the extended Kalman filter.*

3.3 The particle filter

The particle filter is, among the filters discussed in this chapter, the most universal as it can handle nonlinear models and non-Gaussian noise. The derivations of equations become more and more extensive and complex as the filter becomes more general, as seen with the extended Kalman filter. We will therefore rather explain the particle filter's application and reasoning and present the appropriate equations. For an in-depth analysis of the derivation and expansion of the particle filter, the reader is referred to the literature [3, 51].

The particle filter can be thought of as a hidden Markov model (HMM), where the state of the system $\mathbf{x}_i$ is unobservable, but the corresponding measurements $\mathbf{z}_i$ related to the states can be observed, as shown in Figure 3.1. The approach is to represent the posterior density as a set of random independent samples, or particles. As the number of particles go to infinity, the model approaches an optimal Bayesian estimate.

For a general discrete-time model, the state of the system is governed by

$$\mathbf{x}_t = f_t(\mathbf{x}_{t-1}, \mathbf{w}_{t-1}), \tag{3.3.1}$$

where $\mathbf{x}_t$ is the state vector, $\mathbf{w}_{t-1}$ is the process noise and f_t is a possibly nonlinear, time dependent function that relates the state vector at the previous time step $t-1$ to the current

time t. The state vector $\mathbf{x}_t$ is assumed to be unobservable and the only information about it is obtained through noisy measurements, governed by

$$\mathbf{z}_t = h_t(\mathbf{x}_t, \mathbf{v}_t). \tag{3.3.2}$$

Here $\mathbf{v}_t$ is the measurement noise and h_t is a possibly nonlinear, time dependent function that relates the state vector to its measurement at the current time step t.

The particle filter follows the same ideology as the previous filters as it combines a prediction step with a measurement update step. Our goal is to estimate the current unknown state $\mathbf{x}_t$ based on all of the noisy measurements $\mathbf{z}_{1:t} = \{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_t\}$ up to that point. Alternatively, we can formulate this problem in a Bayesian setting, as the computation of the posterior distribution $p(\mathbf{x}_t|\mathbf{z}_{1:t})$. The calculation of this distribution can be separated into two recursive steps, namely the prediction and update step. The prediction step entails the computation of

$$p(\mathbf{x}_t|\mathbf{z}_{1:t-1}) = \int p(\mathbf{x}_t|\mathbf{x}_{t-1})\, p(\mathbf{x}_{t-1}|\mathbf{z}_{1:t-1}) \mathrm{d}\mathbf{x}_{t-1}, \tag{3.3.3}$$

where $p(\mathbf{x}_{t-1}|\mathbf{z}_{1:t-1})$ is assumed to be known from the previous time step and $p(\mathbf{x}_t|\mathbf{x}_{t-1})$ is given in (3.3.1). The distribution $p(\mathbf{x}_t|\mathbf{z}_{1:t-1})$ can be thought of as a prior over $\mathbf{x}_t$ before the most recent measurement $\mathbf{z}_t$ is received. This prior is updated with the new measurement $\mathbf{z}_t$, in the update step, using Bayes' rule. Thus the posterior over $\mathbf{x}_t$ is obtained as

$$p(\mathbf{x}_t|\mathbf{z}_{1:t}) = \frac{p(\mathbf{z}_t|\mathbf{x}_t)\, p(\mathbf{x}_t|\mathbf{z}_{1:t-1})}{p(\mathbf{z}_t|\mathbf{z}_{1:t-1})}, \tag{3.3.4}$$

with

$$p(\mathbf{z}_t|\mathbf{z}_{1:t-1}) = \int p(\mathbf{z}_t|\mathbf{x}_t)\, p(\mathbf{x}_t|\mathbf{z}_{1:t-1}) \mathrm{d}\mathbf{x}_t. \tag{3.3.5}$$

The factor $p(\mathbf{x}_t|\mathbf{z}_{1:t-1})$ is calculated in the prediction equation (3.3.3) and $p(\mathbf{z}_t|\mathbf{x}_t)$ is given in (3.3.2). The problem is that the equations for the prediction and update steps can generally not be calculated analytically, especially in the case of nonlinear and non-Gaussian models, and we therefore have to use approximation methods such as Monte Carlo sampling.

We represent the posterior density as a set of N samples, $\{x^1, x^2, \ldots, x^N\}$, drawn from the target distribution $p(x)$. This distribution can be approximated by a proposal distribution $q(x)$ when it is difficult to sample directly from the target distribution itself. As the number of samples increases the model approaches an optimal estimation. This however leads to uneven partitioning, as regions with high density will lead to many particles. This can be addressed by performing *importance sampling*. It accounts for the discrepancy between the target and proposal distributions by weighing each sample x^i by

$$w_i \propto \frac{\pi(x^i)}{q(x^i)}, \tag{3.3.6}$$

where $\pi(x)$ is a known function that is proportional to $p(x)$.

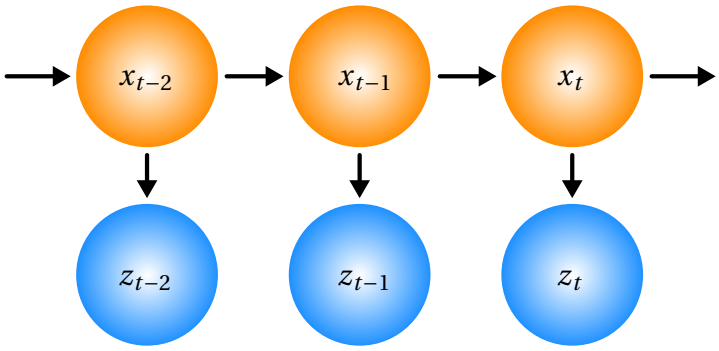


Figure 3.1: *Graphical representation of the nature of the particle filter, where a hidden Markov model is assumed. The state vector $\boldsymbol{x}_t$ is unknown and can only be estimated by the previous observations $\boldsymbol{z}_{1:t}$.*

The idea with sequential importance sampling (SIS) is to update the particles and their corresponding weights such that they would approximate the posterior distribution at the next time step. To do this we first assume that the proposal distribution at time t can be factorized as

$$q(\mathbf{x}_{0:t}|\mathbf{z}_{1:t}) = q(\mathbf{x}_t|\mathbf{x}_{0:t-1},\mathbf{z}_{1:t})\,q(\mathbf{x}_{0:t-1}|\mathbf{z}_{1:t-1}), \tag{3.3.7}$$

so that we can simply update each particle $\mathbf{x}^i_{0:t-1}$ at time $t-1$ with a new state $\mathbf{x}^i_t$ at time t sampled from $q(\mathbf{x}_t|\mathbf{x}_{0:t-1},\mathbf{z}_{1:t})$. According to importance sampling, we update the weights of the particles at time t according to

$$w^i_t \propto w^i_{t-1}\frac{p(\mathbf{z}_t|\mathbf{x}^i_t)\,p(\mathbf{x}^i_t|\mathbf{x}^i_{t-1})}{q(\mathbf{x}^i_t|\mathbf{x}^i_{0:t-1},\mathbf{z}_{1:t})}. \tag{3.3.8}$$

The weights $\{w^1_t, w^2_t, \ldots, w^N_t\}$ are then normalized to sum to 1. If we further assume the process to behave like a Markov chain, i.e. that the current state depends only on the previous state, we can assume that $q(\mathbf{x}_t|\mathbf{x}_{0:t-1},\mathbf{z}_{1:t}) = q(\mathbf{x}_t|\mathbf{x}_{t-1},\mathbf{z}_t)$. We then only have to generate the particle at the next time step from the distribution $q(\mathbf{x}_t|\mathbf{x}^i_{t-1},\mathbf{z}_t)$. This leads to the simplified update equations

$$\mathbf{x}^i_t \sim q(\mathbf{x}_t|\mathbf{x}^i_{t-1},\mathbf{z}_t), \tag{3.3.9}$$

$$w^i_t \propto \frac{p(\mathbf{z}_t|\mathbf{x}^i_t)\,p(\mathbf{x}^i_t|\mathbf{x}^i_{t-1})}{q(\mathbf{x}^i_t|\mathbf{x}^i_{t-1},\mathbf{z}_t)}\,w^i_{t-1}. \tag{3.3.10}$$

The time and measurement update equations discussed in this chapter enable us to successfully track an object over time, through a video sequence. The choice of the filter depends on the situation, but it is clear that each of them requires a measurement of the object. In Chapter 4 we discuss how we go about detecting objects in a video sequence.

It is of course possible to detect more than one object at a particular time step and, in order to track those, it will be imperative to correctly correspond detected objects to those already being tracked. This issue is addressed in Chapter 5.

Chapter 4

Object detection

The task of automatic people detection is far from trivial as a person's movements are complex and mostly unpredictable. Inexpensive hardware typically employed for surveillance produces noisy input data, which makes the task even more difficult. Moreover, the background can be challenging: tiles, bricks and reflections are present in indoor situations, and trees and shadows in outdoor scenes. These entities can also change over time. The background can lead to occlusions, such as a tree or a fence that is between the sensor and the object of interest. All of these factors have to be taken into consideration when building a background model. This is important as the background model governs the success of the foreground detection step.

A robust foreground detector is a key factor in assisting the correspondence step, of determining which detections relate to which over time. As mentioned in Chapter 3, an implemented tracking filter relies on the correspondences between detected objects. The accuracy of these relationships relies on the accuracy of the detections, which in turn relies on the quality of the background model.

4.1 Background modelling

Background modelling and subtraction is the first step in object detection. The idea is to subtract a background frame or model from the current frame. This can be done pixel-wise when we work with static cameras. The deviation in pixel values, which can be colour vectors or single intensity values, is used to determine which pixels are classified as foreground. Ideally, the background model must

- enable the detection of movement through busy areas;
- incorporate an object that has permanently stopped;
- adapt to sudden or gradual lighting changes;

- and handle repetitive motion, such as sensor noise.

There exist different background subtraction methods, varying from fairly basic to more complex. We start the discussion with two basic methods: *frame differencing* and *running Gaussian averaging*. They are simple to implement and computationally inexpensive, but rarely work well in practice. They do however provide a basis from which more sophisticated methods can be developed, such as those based on *Gaussian mixture models*.

4.1.1 Frame differencing

The most basic method to perform background subtraction is to simply subtract the current frame from the previous frame, or from the average image taken over a number of frames. The average image can be trained offline or consist of some average of the previous n frames. These can possibly include foreground objects, which should be accounted for.

We define $I_t(x,y)$ as the pixel value in image frame I at time step t with coordinates (x,y). If we consider only the current and the previous frame we can calculate the foreground pixels in the current frame according to

$$\left|I_t(x,y) - I_{t-1}(x,y)\right| > T, \tag{4.1.1}$$

where T is a chosen threshold.

If we rather want to incorporate an average image for background subtraction, we can build the background image B_t at time t with a running average as

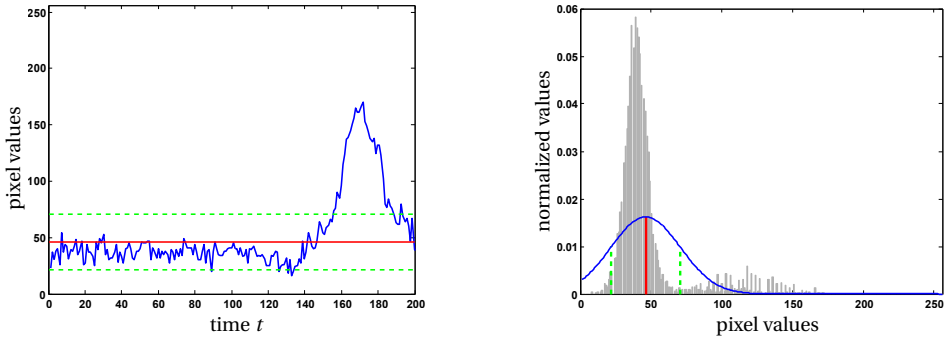
$$B_t = \alpha I_{t-1}(x,y) + (1-\alpha)B_{t-1}(x,y), \tag{4.1.2}$$

where $\alpha \in [0,1]$ is the so-called *learning rate*. The learning rate determines how quickly new pixels are incorporated into the background image. Ghosting appears as $\alpha \to 1$, due to past foreground pixels being (wrongfully) added to the background image. If $\alpha \to 0$, on the other hand, the method does not include new background pixels quick enough and may classify them as foreground pixels. An example of this is a newly parked car.

As the background image is built up of the previous n images, it is beneficial to include only the classified background pixels, of the previous frames, in the background image. This selectivity can be incorporated into the running average as

$$B_t(x,y) = \begin{cases} \alpha I_{t-1}(x,y) + (1-\alpha)B_{t-1}(x,y), & \text{if } I_{t-1}(x,y) \text{ is background,} \\ B_{t-1}(x,y), & \text{if } I_{t-1}(x,y) \text{ is foreground.} \end{cases} \tag{4.1.3}$$

The limitations of frame differencing are that it does not provide an explicit method to determine the threshold, it struggles against multimodal background distributions and it is sensitive to different frame rates.



(a) (b)

Figure 4.1: *Fitting a single Gaussian PDF to a pixel's intensity values, shown in (a), over time. The histogram of intensity values is multimodal, as shown in (b), which causes the fitting of a single Gaussian to be inaccurate. This shortcoming encourages the use of GMMs. The red line in both figures (a) and (b) indicates the mean μ, and the green dashed lines indicate the standard deviation.*

4.1.2 Running Gaussian average

The running Gaussian average method works by fitting a Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$ over the histogram of the history of each pixel, as illustrated in Figure 4.1. This fit is then used as the background probability density function (PDF) for a particular pixel. It would be expensive to recalculate the PDF for all the pixels in every new frame, so we compute a running average instead as

$$\mu_t(x,y) = \alpha I_t(x,y) + (1-\alpha)\mu_{t-1}(x,y), \tag{4.1.4}$$

$$\sigma_t^2(x,y) = \alpha \left(I_t(x,y) - \mu_t(x,y)\right)^2 + (1-\alpha)\sigma_{t-1}^2(x,y), \tag{4.1.5}$$

where $\mu_t(x,y)$ is the mean and $\sigma_t^2(x,y)$ is the variance of the PDF associated with pixel (x,y).

The foreground pixels are then determined as $\left|I_t(x,y) - \mu_t(x,y)\right| > T$, where the threshold T is typically chosen as some constant multiple of $\sigma_t(x,y)$. This means that, since we know the behaviour of each pixel, we check how much each pixel in the new frame differs from the corresponding mean, and if it differs sufficiently (to the point where it becomes highly unlikely to be described by the pixel's PDF) we classify it as foreground.

This method suffers in the same conditions as with frame differencing: when the background is multimodal and hence cannot be modelled as a single Gaussian PDF. This shortcoming leads to the use of Gaussian mixture models, described next.

4.1.3 Gaussian mixture model

Gaussian mixture models (GMMs) are among the most enduring and widely used statistical models [21]. Such a model considers different data "sources", in the form of single Gaussian PDFs from which data can be sampled, and weighs these sources in order to determine a pixel's origin. GMMs can also handle changes in lighting conditions well, which is a signif-

icant shortcoming of the previous methods. Different surfaces in a scene typically produce different data signatures, each corresponding to a different Gaussian PDF. It is clear that the data shown in Figure 4.1 is multimodal and a fit using GMMs would be more accurate than using a single Gaussian.

As before we define $\mathbf{x}_t = I_t(x, y)$ as the value of a pixel (x, y) at time step t. We define the pixel value as a vector, for generality, to account for the possibility of it corresponding to the red, green and blue intensity values of a colour pixel (or any other colour scheme). The discussion below is still applicable to 1-dimensional pixel vectors as found in greyscale or thermal images.

A d-dimensional Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ has probability density function

$$\eta(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{\frac{d}{2}} |\boldsymbol{\Sigma}^{\frac{1}{2}}|} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})}, \tag{4.1.6}$$

and in our case d would be the dimension of the pixel vector $\mathbf{x}$ as mentioned above.

We model the history of a pixel's values by a weighted mixture of K Gaussian distributions, so that we can calculate the probability of observing the current pixel intensity value as

$$p(\mathbf{x}_t) = \sum_{k=1}^{K} \omega_{k,t} \eta(\mathbf{x}_t; \boldsymbol{\mu}_{k,t}, \boldsymbol{\Sigma}_{k,t}), \tag{4.1.7}$$

where K is the total number of distributions, $\omega_{k,t}$ is the weight, $\boldsymbol{\mu}_{k,t}$ the mean and $\boldsymbol{\Sigma}_{k,t}$ the covariance matrix of the k-th Gaussian in the mixture at time t. The mixture weights must satisfy the constraint $\sum_{k=1}^{K} \omega_{k,t} = 1$.

The number of distributions used is determined by the availability of system memory and computation power, but usually three to five suffice. To further improve computation speed, and to simplify the update rules to follow, we define the covariance matrix as

$$\boldsymbol{\Sigma}_{k,t} = \sigma_{k,t}^2 \mathbf{I}_d, \tag{4.1.8}$$

where $\sigma_{k,t}^2$ is the variance for distribution k at time t and $\mathbf{I}_d$ is the $d \times d$ identity matrix. With this we assume that the d variables are all independent and have the same variances. It might be a gross oversimplification, but this assumption simplifies calculations dramatically at the expense of some model accuracy.

It is clear from (4.1.7) that the distribution of the pixel's intensity values in the current frame is modelled as a weighted mixture of Gaussians. Every new pixel value, $\mathbf{x}_t$, is checked against the existing K Gaussian distributions until a match is found. It is considered to match a distribution if it lies sufficiently close, usually within 2.5 standard deviations [21], to the mean of that distribution.

If the current pixel value does not match any of the K distributions, the least probable distribution is replaced with one that has the pixel value as mean. This new distribution will initially have a high variance and low prior weight. The prior weights of the K distributions at time t can be updated as

$$\omega_{k,t} = \alpha M_{k,t} + (1-\alpha)\omega_{k,t-1}, \tag{4.1.9}$$

where $\alpha \in [0,1]$ is the learning rate and $M_{k,t}$ is equal to 1 for the matched distribution(s), and 0 for those that did not match. This lets the system initially ignore the newly added distribution(s) with high variance. After the update the weights are normalized so that they sum to 1.

The parameters of the unmatched distributions remain unchanged, but the parameters of the distributions that match the new observation are updated as

$$\boldsymbol{\mu}_{k,t} = \rho \mathbf{x}_t + (1-\rho)\boldsymbol{\mu}_{k,t-1}, \tag{4.1.10}$$

$$\sigma^2_{k,t} = \rho(\mathbf{x}_{k,t} - \boldsymbol{\mu}_{k,t})^T(\mathbf{x}_{k,t} - \boldsymbol{\mu}_{k,t}) + (1-\rho)\sigma^2_{k,t-1}, \tag{4.1.11}$$

where

$$\rho = \alpha\eta(\mathbf{x}_{k,t}; \boldsymbol{\mu}_{k,t-1}, \boldsymbol{\Sigma}_{k,t-1}). \tag{4.1.12}$$

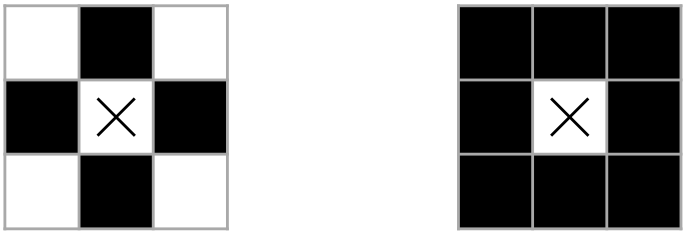
It is clear that only matched data is used to update the model. This implies that the existing background is not destroyed if additional data is included in the model. The original background pixel values remain in the mixture until they become the K-th most probable match, after which they get replaced. This is particularly useful in handling sudden and slow lighting changes as well as slow moving or temporarily static objects which otherwise would be (wrongfully) incorporated into the background. The rate of inclusion depends on the chosen value for α.

4.2 Foreground detection and clustering

With a background model at hand, we use background subtraction to classify foreground pixels in the current frame. This gives us a collection of unlabelled foreground pixels. These pixels can be clustered together using *connected-component labelling*, but it can be beneficial to perform some pre-processing on the classified pixels first. The idea is that such operations may

- remove isolated foreground pixels;
- reclassify background pixels if they are surrounded by foreground pixels;
- smooth the boundaries of foreground clusters.

The above can be achieved by applying *morphological* operations on the detected foreground pixels.



4-neighbouring 8-neighbouring

Figure 4.2: *Comparison between 4- and 8-neighbouring structure elements. The 4-neighbouring element considers pixels directly above, below, to the left and right of the pixel to be neighbours, whereas the 8-neighbouring element considers any surrounding pixels to be neighbours. "X" marks the origin of the elements.*

4.2.1 Morphological operations

There exist many morphological operations, but the two principal ones are *erotion* and *dilation*. Dilation expands objects, thus potentially filling in small holes and connecting disjoint clusters. Erosion shrinks objects by etching away their boundaries.

In our case we use binary images as input for morphological operations, although it is possible to perform greyscale morphology. A binary image is an image composed of only "active" and/or "inactive" pixels. Inactive pixels have a value of zero, while active pixels have an integer value greater than zero (usually 1).

Let E be the integer grid $\mathbb{Z}^2$, with $A \subseteq E$ the original binary image and $B \subseteq E$ the binary structuring element used to perform the morphology. B can be any simple, pre-defined shape, such as

- a 3×3, 4-neighbour element, represented as a cross:

$$B = \{(-1;0),(0;0),(1;0),(0;-1),(0;1)\};$$

- a 3×3, 8-neighbour element, represented as a square:

$$B = \{(-1;-1),(0;-1),(1;-1),(-1;0),(0;0),(1;0),(-1;1),(0;1),(1;1)\}.$$

Depictions of these can be seen in Figure 4.2. The morphological operation is similar to convolution, except that the mask (B in this case), only works on the image if its centre is on an active pixel. A few useful operations will be explained next.

4.2.1.1 Erosion

Erosion of the binary image A is the result of subtracting the structuring element B around the edges of A, and is defined as

$$A \ominus B = \{\mathbf{z} | \mathbf{z} = \mathbf{a} \cdot \mathbf{b}, \quad \exists (\mathbf{a} \in A, \mathbf{b} \in B)\}, \tag{4.2.1}$$

where $\mathbf{a} \cdot \mathbf{b}$ represents the logical "and" operation between vectors **a** and **b**.

4.2.1.2 Dilation

Dilation of the binary image A is the result of adding the structuring element B around the edges of A, and is defined as

$$A \oplus B = \left\{ \mathbf{z} | \mathbf{z} = \mathbf{a} \cdot \mathbf{b}, \quad \forall (\mathbf{a} \in A, \mathbf{b} \in B) \right\}, \tag{4.2.2}$$

where $\mathbf{a} \cdot \mathbf{b}$ once again represents the logical "and" operation between **a** and **b**. Erosion and dilation operators can be combined to form more complex operators, the most useful of these being *opening* and *closing*.

4.2.1.3 Opening and closing

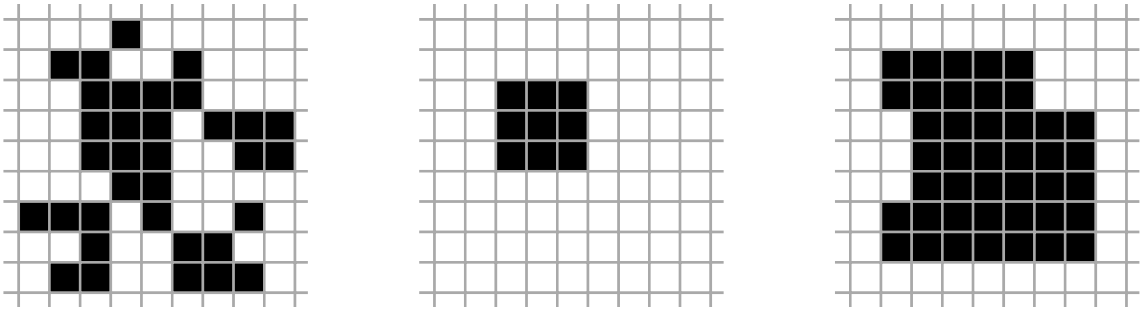
The opening and closing operators are constructed by combining the erosion and dilation operators. Opening consists of an erosion followed by a dilation whereas closing consists of a dilation followed by an erosion. That is,

$$A \circ B = (A \ominus B) \oplus B, \tag{4.2.3}$$

$$A \bullet B = (A \oplus B) \ominus B. \tag{4.2.4}$$

Even though opening and closing consist of the same operators, their results are significantly different from one another, as seen in Figure 4.3.

Opening can be used to remove unwanted foreground pixels, as it minimizes the original size of the cluster by firstly eroding the boundary and then fleshing the remainder out again. The result is a cluster of smaller size, which can be ignored by thresholding. Opening removes isolated foreground pixels completely, as the cluster's size is smaller than the structuring element B. Opening also smooths out foreground clusters as it removes unwanted spurious 1-pixel "tails", as seen in Figure 4.3 (b).



(a) original (b) opening (c) closing

Figure 4.3: *Different results when using the opening (b) and closing (c) operators on the input image (a). Even though opening and closing both use dilation and erosion, the results are significantly different.*

Closing, on the other hand, can be used to fill small gaps in and around foreground clusters. This is achieved by maximizing the original cluster size, by fleshing it out with dilation and then eroding the boundary. The erosion results in smoothed clusters as seen in Figure 4.3 (c).

Once the foreground pixels have been processed in this manner, we can classify the resulting clusters uniquely which enables us to eventually track consecutive objects over time.

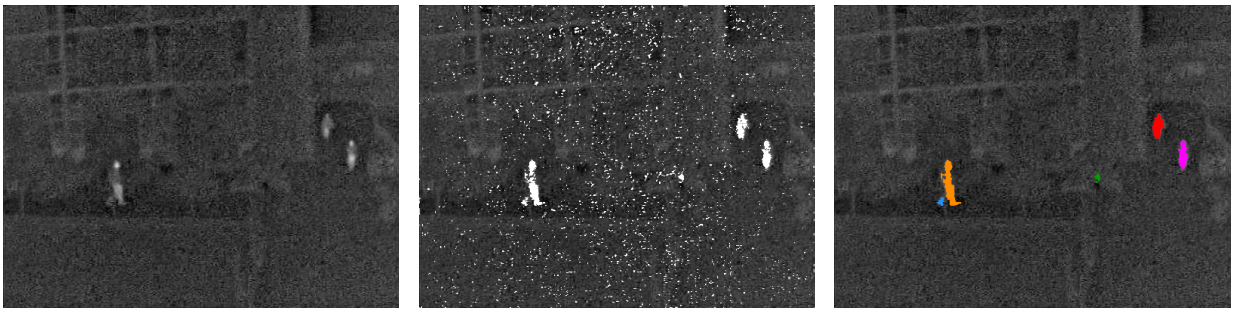
4.2.2 Connected-component labelling

Connected-component labelling classifies neighbouring pixels with similar values together, and is outlined in Algorithm 1. It is common to use a binary image as input, but the algorithm can handle other data such as greyscale or colour images.

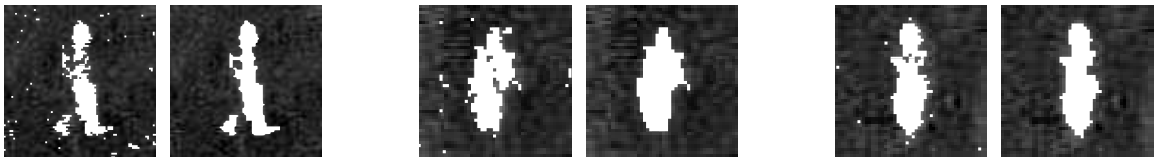
The first pass of the algorithm scans the image pixel-wise from top to bottom, left to right, and identifies neighbouring pixels with similar values. Here pixels are considered to be neighbours if they are 8-connected, that is, every pixel that has the coordinates $(x \pm 1, y)$, $(x, y \pm 1)$, $(x \pm 1, y \pm 1)$ or $(x \pm 1, y \mp 1)$ are connected to the pixel (x, y). The algorithm assigns temporary labels to all the foreground pixels. Note that it is possible to label different pixels in the same blob (cluster of foreground pixels) with different labels. This is due to the top-down, left-right nature of the algorithm where it compares the current pixel (x, y) only with the pixels to the top, top-left and left of it. All the different labels for each blob are recorded

Algorithm 1 Two pass connected-component labelling

```
1:  procedure ConnCompLabelling(I)                          ▷ I is the input image
2:      [m, n] = size(I)                                    ▷ m = height, n = width
3:      Labels = []
4:      Equivalence_classes = []

5:      for i = 1 : m do                                    ▷ First pass
6:          for j = 1 : n do
7:              if I(i, j) is foreground then
8:                  neighbours = all foreground pixels that are 8-connected to pixel (i, j)
9:                  if neighbours.label is empty then
10:                     Assign new label l_k to pixel (i, j)
11:                     Add l_k to Labels
12:                 else
13:                     Find l_m = min(neighbours.label)
14:                     Assign l_m to pixel (i, j)
15:                     Add all neighbours.label to an entry in Equivalence_classes

16:     for i = 1 : m do                                    ▷ Second pass
17:         for j = 1 : n do
18:             if I(i, j) is foreground then
19:                 if (i, j)'s label is in Equivalence_classes then
20:                     Relabel pixel with smallest label in its Equivalence_classes entry
21:                     Update Labels

22:     return Labels
```



(a) input image (b) foreground pixels (c) labelled blobs



object 1 object 2 object 3

Figure 4.4: *Results of performing morphological clean-up and connected-component labelling on detected foreground pixels. The input image is shown in (a), the detected foreground in (b) and the labelled blobs in (c) where every colour corresponds to a label. The effects of the morphological operations are shown in the bottom row as pre- and post-processed pairs.*

as equivalence labels. Any possible misclassifications are fixed in the second pass where the algorithm assigns unique labels to all the equivalence classes.

Figure 4.4 shows an example of an input thermal image, followed by the detected foreground pixels and finally the labelled blobs extracted with morphological clean-up and connected-component labelling.

With this in place we are able to detect separate objects in a video frame. This is done at every time step, and is the first component in our object tracking system. The issue we encounter now is that, although we have detected blobs at time $t-1$ and t, we still need a way to determine which of these detections relate to which, over time. We deal with this issue in the next chapter.

Chapter 5

Correspondence

The correspondence problem in automatic object detection and tracking is a significant focus of this thesis, and we will discuss our complete approach in Chapter 6. For the moment we assume to have foreground blobs from the previous time step $t-1$ with newly acquired blobs at time t. The immediate problem here is to determine which blobs at the current time step relate to which from the previous time step. The solution can be fairly straightforward for uncluttered scenarios where a small number of well separated blobs are present, but additional robustness is required for scenes with more complex activity.

We formulate this correspondence problem by using a graph theoretic approach. We represent each blob as a vertex in a graph, and if there is evidence that a blob from time step $t-1$ corresponds to a blob at time step t we connect them with an edge. The graph G_t at time t can be classified as a directed bipartite graph, with vertex set $V_{t-1} \cup V_t$ and edge set E_t, directed (chronologically) from time step $t-1$ to time t. This means that all the vertices are divided into two disjoint vertex sets, namely V_{t-1} at time $t-1$ and V_t at time t, such that every edge connects a vertex in V_{t-1} to a vertex in V_t. When drawing such graphs we will neglect arrows that indicate direction of edges, as it is clear that the graph progresses from one time step to the next. We may also switch between using the terms "vertex", "node" and "blob", depending on the context, but it should be clear that they imply the same thing.

In this chapter we discuss a number of constraints that assist in determining putative correspondences from one time step to the next. These constraints are adaptations of those presented by Masoud and Papanikolopoulos [31]. The first is the *locality constraint* which only considers, as candidates, blobs at the current time step that are sufficiently close to blobs in the previous time step. This is a fair assumption if we track objects in video with a high frame rate. Secondly we incorporate the *parent constraint* which simply states that from one time step to the next, a blob cannot be part of a split and a merge detection simultaneously. Before detailing these constraints, we first discuss the terms *split*, *merge*, *death* and *birth* as well as their graphical representations.

5.1 Graphical representation

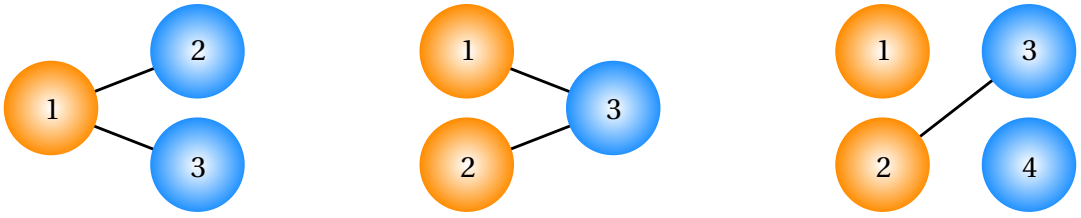
It is easier to grasp and interpret the behaviour of the correspondences between blobs over time by representing them graphically. As mentioned, we can encounter splits, merges, deaths and births from one time step to the next. We will cover splits, merges and deaths, and how we handle them, in more detail in Chapter 6, but for now it suffices to know that these scenarios exist and how we interpret them graphically.

In most cases the occurrence of these detections is unwanted. They can occur due to noise in the measurement data or from occlusions or misclassifications. Occlusions occur when a static object (such as a tree) comes between the camera and the object. Misclassifications can occur in colour data if, say, a person's shirt is close to having the same colour as the background he is moving over. This would result in the shirt being classified as background which would split the person being tracked into two separate blobs.

The correspondence issues of interest are shown in Figure 5.1, and can be classified as

- a *split*, where a vertex in V_{t-1} is adjacent to two or more vertices in V_t;
- a *merge*, where a vertex in V_t is adjacent to two or more vertices in V_{t-1};
- a *death*, where a vertex in V_{t-1} is not adjacent to any vertices in V_t;
- a *birth*, where a vertex in V_t is not adjacent to any vertices in V_{t-1}.

Even though we can classify these events, it is important to note that we consider it impossible for an object to actually split or merge with another. When a split is detected it can only be due to more than one object being detected first as a single blob, say $v_1 \in V_{t-1}$, which then separates at some point into disconnected blobs $v_2, v_3 \in V_t$. This should in fact result in either the edge (v_1, v_2) with v_3 being born, or the edge (v_1, v_3) with v_2 being born. When a merge is detected, say $v_1, v_2 \in V_{t-1}$ merge with $v_3 \in V_t$, it can be due to one person walking in front of another, thereby causing an occlusion. This situation should result in the edge (v_1, v_3) with v_2 dying, or the edge (v_2, v_3) with v_1 dying.



(a) split (b) merge (c) death and birth

Figure 5.1: *Graphical representation of four issues we may encounter in correspondences from one time step to the next. In (a) vertex 1 splits, in (b) vertices 1 and 2 merge, in (c) vertex 1 dies and vertex 4 is born.*

Next we discuss the primary constraints used to create an initial graph G_t. These constraints can lead to splits and merges, as we shall see, and we describe our way of handling them in Chapter 6.

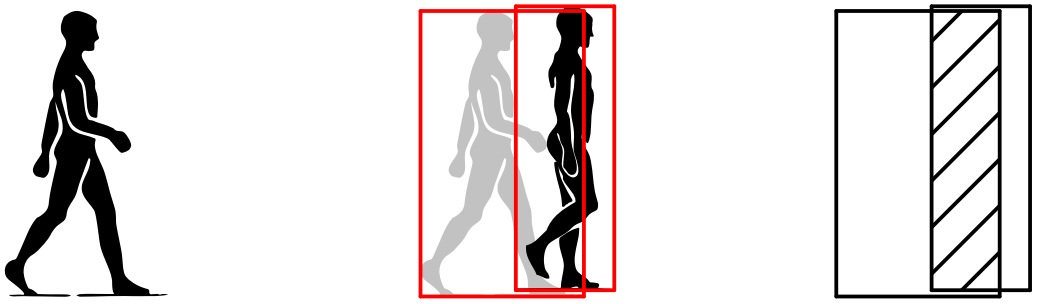
5.2 Locality constraint

As we are dealing with object detection in video it is reasonable to assume that the position and size of an object do not change greatly from one frame, or time step, to the next. This assumption is especially valid in the detection of slow moving objects, such as people, with the use of high frame rate sensors. The locality constraint considers the bounding boxes of detected blobs at consecutive time steps.

Let $A(v)$ denote the area (in pixels) of the bounding box of blob v, and consider two blobs $v_i \in V_{t-1}$ and $v_j \in V_t$. The locality constraint stipulates that the edge (v_i, v_j) is added to the graph G_t if and only if the *bounding box overlap area* of v_i and v_j is at least as large as α times $\min\{A(v_i), A(v_j)\}$. Here α is a confidence factor typically chosen as 0.5 [31] and the concept of bounding box overlap area is clarified in Figure 5.2.

This constraint offers a quick way for determining putative correspondences between blobs over time. It can however yield unwanted results in more complex scenes, as it works purely on the location of the detections. It might wrongfully detect correspondences between different objects if they were to move past one another.

Nevertheless, the locality constraint is cheap and can be effective in most cases. We can implement an additional constraint, which acts as a safety net for cases where splits and merges detected by the locality constraint intertwine to the point where they become difficult (if not impossible) to resolve. This is called the *parent constraint*, and is discussed next.



(a) (b) (c)

Figure 5.2: *Graphical representation of the implementation of the locality constraint. A detection at time $t-1$ is shown in (a), and (b) shows the same object detected at time t superimposed over the previous detection, with the corresponding bounding boxes shown in red. The bounding box overlap area is shown in (c) as a shaded area.*

5.3 Parent constraint

The parent constraint requires that a blob cannot be part of both a split and a merge from one time step to the next, as illustrated in Figure 5.3. Stated differently, for each connected component of G_t there may not be more than one vertex of degree more than 1. To satisfy this constraint we have to devise a way of choosing, in the violating cases, which edges to keep and which to remove.

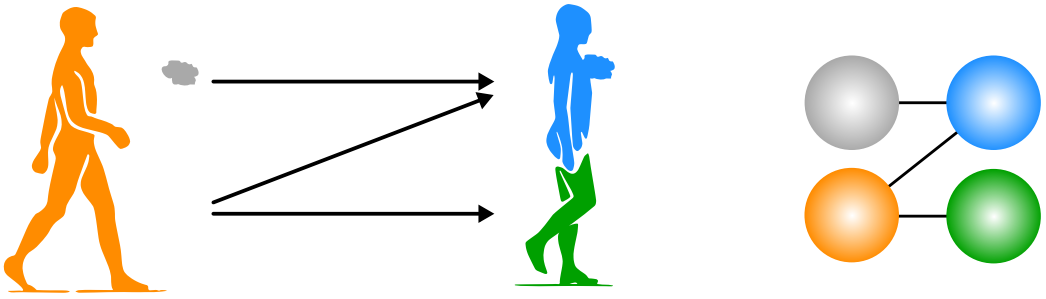
Let us assume that we have, after enforcing the locality constraint, a violating graph G_t with vertex set $V_{t-1} \cup V_t$ and edge set E_t relating vertices in V_{t-1} to V_t. We can then systematically remove an edge, or set of edges, until we have a valid graph G_t^i induced by edge set E_t^i. Here the graph is considered to be valid if it satisfies both the locality and parent contraint.

Following the formulation of Massoud and Papanikolopoulos [31] we can calculate a score for each of the induced valid graphs. We first define the neighbours of a vertex $v \in V_{t-1}$ as all the vertices adjacent to it in V_t induced by the edge set E_t^i, and denote it by $N_t^i(v)$. Furthermore, we define the area of the neighbouring vertices of v as the sum of their areas, and denote it by S_t^i. This enables us to formulate a cost function as

$$C\left(G_t^i\right) = \sum_v \frac{\left|A(v) - S_t^i\right|}{\max\left(A(v), S_t^i\right)}, \tag{5.3.1}$$

where $A(v)$ is the area of blob v in pixels. The cost function is set up in such a way that it penalizes graphs in which blobs change significantly in size over time, which is uncharacteristic in the detection of people. We compare all valid graphs G_t^i and choose the corresponding edge set E_t^i of the graph with the lowest cost.

A major issue with this cost function approach is that it can become computationally expensive if there are many edge combinations to evaluate. Another issue is that it does not



(a) (b)

Figure 5.3: *Graphical representation of the occurrence of a situation violating the parent constraint. A detection from time step $t-1$ to time step t is shown in (a), with detections coloured according to the resulting graph shown in (b). It is clear that the parent constraint is violated, as the orange blob at time $t-1$ splits to blue and green at time t while the grey and orange blobs at $t-1$ merge to blue at time t.*

consider the *appearance* of different blobs, which would be available from the corresponding colour image, but rather only considers the bounding box areas. We alter this method by considering feature vectors of vertices in the violating graph. We can calculate a similarity measure between connected vertices and use that as a measure to cheaply compare vertices. We discuss our adapted enforcement of the parent constraint in Section 5.6.

5.4 Feature vectors

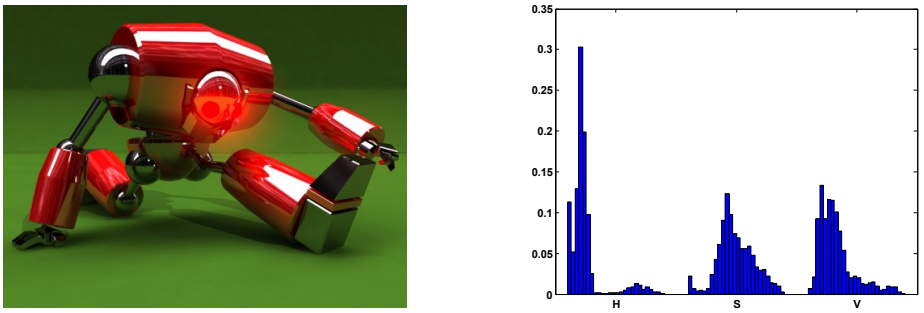
A problem we encounter with blob detection is that all detections are similar in appearance since they are all regions of positive values in the binary foreground image. To help classify the detected blobs we build a feature vector for each. A feature vector can contain a number of attributes describing an object's appearance and shape, such as colour, contour, texture, edge and intensity.

When building a feature vector it is beneficial to consider the bounding box around the blob, and not just the actual foreground pixels, for a number of reasons. Firstly, we know that there can be measurement noise present, which can result in either over- or under-detection. This simply means that the detected blob can be smaller or larger than the actual object of interest. By taking the bounding box we accommodate such noisy detections. Secondly, the bounding box also includes some background data. This is especially beneficial for differentiating between objects moving over different backgrounds. It is clear that colour is a useful attribute for differentiating between different people, and will be described next.

5.4.1 Colour-based feature

There are different approaches to building feature vectors. Firstly we have to consider which representation of the pixel values we are going to use. We can deconstruct a pixel's colour into its red, green and blue (RGB) components, but this representation is heavily influenced by changes in lighting conditions [9] that in turn affect shadows and reflections cast by objects. It therefore makes sense to use a different colour representation, such as the hue, saturation and intensity (HSV) representation, instead. Hue is the degree to which a colour differs from the unique hues (namely red, green, blue and yellow). Saturation is defined as the degree to which a colour's value differs from grey relative to its own brightness. The intensity value simply indicates how bright the hue and saturation are. This composition results in HSV being invariant to high intensity white lights, ambient light and surface lighting change [23]. This is due to the effects of these changes being mostly present in the intensity component. Handling the intensity values separately, or even discarding them completely, allows us to produce a more robust colour feature.

As mentioned, we build the colour feature from the bounding box surrounding a detected blob. We can also apply a kernel function to weigh the pixels in the bounding box, if we assume that the pixels in the centre are more important than those at the edges. We may



(a) (b)

Figure 5.4: *Building a feature vector using the HSV representation of a colour image. The input image is shown in (a) and (b) shows the corresponding HSV histograms.*

define a kernel function as

$$k(\mu) = \begin{cases} 1-\mu^2, & \text{if } \mu < 1, \\ 0, & \text{otherwise,} \end{cases} \tag{5.4.1}$$

where μ is the normalized distance of a pixel to the centre of the bounding box. Kernels such as Epanechnikov, quartic (biweight), tricube (triweight) or Gaussian could also be used [38]. By scaling the distance of each pixel to the centre of the bounding box, given by

$$\hat{d} = \sqrt{\left(\frac{H_x}{2} - x\right)^2 + \left(\frac{H_y}{2} - y\right)^2}, \tag{5.4.2}$$

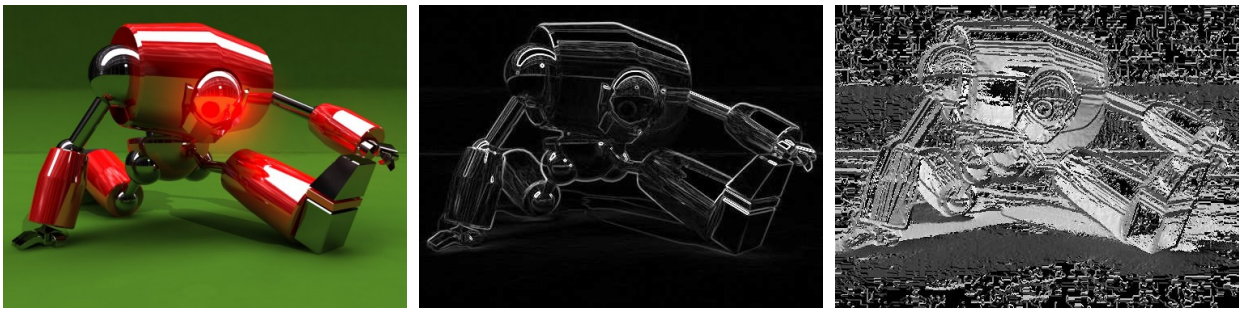
with the circular radius $r = \sqrt{\frac{1}{4}\left(H_x^2 + H_y^2\right)}$, the kernel function assigns the largest weights to pixels at the centre. Here H_x and H_y represent the width and height of the bounding box respectively. Importance weighting of this type also assists in the handling of partial occlusions near the edges of the bounding box.

The feature vector is built by linearly concatenating the histograms of each of the deconstructed colour components, weighted according to the kernel function. The advantage of using the histograms is that it can handle objects with changing size and orientation, as the histograms are normalized and we assume that the colour representation of detected blobs remain more or less the same. Figure 5.4 gives an example of an HSV colour feature vector.

Up to now we have considered only colour. It is possible to add further distinctiveness by incorporating the object shape data to the feature. The shape can be modelled by adding a histogram of orientated gradients (HOG) descriptor, as described next.

5.4.2 Histogram of oriented gradients (HOG) feature

The histogram of oriented gradients (HOG) has been shown to be a successful human detector [10]. HOG operates on localized image regions, and is designed to be invariant to changes in object size and lighting conditions. However, the object's orientation has to remain relatively constant, as HOG uses the orientation of pixel gradients.



(a) (b) (c)

Figure 5.5: *Visual representation of the calculations leading to the building of a HOG feature descriptor. The input image is shown in (a), its maximum gradient magnitudes are shown in (b) and the final weighted orientations are shown in (c).*

The idea of a HOG feature is to describe appearance and shape by a distribution of edge directions. This is achieved by dividing the image into smaller regions, called cells, and building a HOG for the pixels in each cell. The combination of all these histograms forms the HOG feature. It is possible to improve the accuracy of this feature to handle changing lighting conditions by determining an intensity value measured across a larger region of the image, called a block, and using this value to normalize the histograms in all the cells.

It is redundant to perform any preprocessing, such as colour or gamma normalization, as Dalal and Triggs [10] indicate that the descriptor normalization essentially achieves the same result. The first step is to calculate the gradient values for each pixel in a cell. This is done by convolving with the 1-dimensional kernel masks

$$\mathbf{k}_x = [-1\,0\,1], \tag{5.4.3}$$

$$\mathbf{k}_y = [-1\,0\,1]^T, \tag{5.4.4}$$

which results in the horizontal and vertical gradients respectively. It is possible to use different masks, such as the 3×3 Sobel masks or diagonal masks, or to perform Gaussian smoothing before convolving, but all of these result in poorer performance [10] as they induce some form of loss of the available shape data.

When working with colour data we apply the kernels in (5.4.3) and (5.4.4) to each component separately, so that we have for each pixel an x- and y-gradient. These gradient components are then used to calculate the orientation and magnitude of each pixel, as

$$O(x,y) = \tan^{-1}\left(\frac{g_y}{g_x}\right), \tag{5.4.5}$$

$$M(x,y) = \sqrt{(g_x)^2 + (g_y)^2}, \tag{5.4.6}$$

where g_x and g_y represent the x- and y-gradients of pixel (x,y). An example of an image with its magnitude and orientation matrices are shown in Figure 5.5.

We reduce these calculations to a single plane by choosing the largest magnitude $M(x, y)$ for each pixel (x, y) across its colour components. The corresponding orientation $O(x, y)$ is then used and weighed with this magnitude. The weighted orientations in each cell are then binned into uniformly spaced intervals. This results in the HOG feature descriptor, which can now be used separately from the colour feature, or in addition to it, to form a single feature vector for a detected object.

Now that we have feature vectors, we need a method of comparison. This can be done by calculating some similarity measure between two features.

5.5 Similarity measures

It is necessary to be able to compare the features of different blobs and determine how similar they are. The issue we encounter is that the locality and parent constraints alone are insufficient, as they consider only the locations of detected blobs. We need a way of comparing the appearance and shape of the blobs, so that the occurrence of splits and merges can be handled more successfully.

One way of comparing feature vectors is by using a *distance metric*. The most commonly used distance metric is the length of the straight line segment connecting two points, and is called the *Euclidean distance*. It is calculated as

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_N - y_N)^2}, \tag{5.5.1}$$

where $\mathbf{x}$ and $\mathbf{y}$ are both n-dimensional vectors in Euclidean space.

A different distance metric we can use is the *Manhattan distance* [26] between two points. It takes into account the absolute differences between the Cartesian coordinates of two points. The idea is derived, as implied by its name, by finding the shortest path between two points on a fixed grid, as a taxicab would in the streets of Manhattan. For n-dimensional vectors we have

$$d(\mathbf{x}, \mathbf{y}) = |x_1 - y_1| + |x_2 - y_2| + \cdots + |x_N - y_N|. \tag{5.5.2}$$

A shortcoming of using such distance metrics is that their interpretation can be problematic, particularly in high dimensions. A small Euclidean distance does not necessarily imply a good similarity measure or high probability that the measurements are similar. We will see in Chapter 6 that it is useful to have a metric that is probabilistically interpretable, which is not possible with a Euclidean distance as it is unbounded.

The *Mahalanobis distance* [30] is used to identify and gauge the similarity between two sample sets, where the one is known. It differs from Euclidean distance metrics as it takes into account the correlations of the data set and is scale invariant. If we have an unknown vector

$\mathbf{x} = (x_1, \cdots, x_N)^T$ we can formulate the Mahalanobis distance between $\mathbf{x}$ and a known group of values with mean $\boldsymbol{\mu} = (\mu_1, \cdots, \mu_N)^T$ and covariance matrix $\boldsymbol{\Sigma}$ as

$$d(\mathbf{x}) = \sqrt{(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu})}. \tag{5.5.3}$$

It is interesting to note that if $\boldsymbol{\Sigma}$ is the identity matrix, (5.5.3) reduces to the Euclidean distance in (5.5.1). If $\boldsymbol{\Sigma}$ is diagonal the Mahalanobis distance results in a *normalized Euclidean distance* given by

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{N} \frac{(x_i - y_i)^2}{\sigma_i^2}}, \tag{5.5.4}$$

where σ_i is the standard deviation of x_i and y_i over the sample set.

If the two data sets we wish to compare are expressible as probability distributions we can use the *Bhattacharyya distance* [43]. It can be used to determine the relative similarity between two samples and is considered to be more reliable than the Mahalanobis distance. This is due to the Mahalanobis distance being a special case of the Bhattacharyya distance if the standard deviation of both classes are the same. The Bhattacharyya distance between two discrete probability distributions $\mathbf{x}$ and $\mathbf{y}$ is defined as

$$d(\mathbf{x}, \mathbf{y}) = -\ln\left(\beta(\mathbf{x}, \mathbf{y})\right), \tag{5.5.5}$$

where

$$\beta(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{N} \sqrt{x_i y_i} \tag{5.5.6}$$

is the Bhattacharyya coefficient.

Lastly, it is possible to calculate the *correlation* between variables, and use that as a similarity measure. There are different correlation coefficients available, of which Pearson's coefficient is the most widely used [40]. It is only sensitive to a linear relationship between variables. Other correlation coefficients, that are more sensitive to nonlinear relationships, have been developed to be more robust than Pearson's coefficient [14, 2]. However, we found Pearson's coefficient to be adequate for our purposes.

The Pearson correlation coefficient, also known as "Pearson's r", is a dimensionless index, which is invariant to linear transformations of either variable [40], and is formulated as

$$r = \frac{\sum_{i=1}^{N} \left[\left(\mathbf{x}_i - \bar{\mathbf{x}}\right)\left(\mathbf{y}_i - \bar{\mathbf{y}}\right)\right]}{\left[\sum_{i=1}^{N} \left(\mathbf{x}_i - \bar{\mathbf{x}}\right)^2 \sum_{i=1}^{N} \left(\mathbf{y}_i - \bar{\mathbf{y}}\right)^2\right]^{1/2}}. \tag{5.5.7}$$

Equation (5.5.7) centres the two random variables $\mathbf{x}$ and $\mathbf{y}$ by subtracting their corresponding means $\bar{\mathbf{x}}$ and $\bar{\mathbf{y}}$. The sum of cross-products of the centred variables is then accumulated. This sum is then divided by the product of the sums of centred variables to ensure the scales of both variables have the same units.

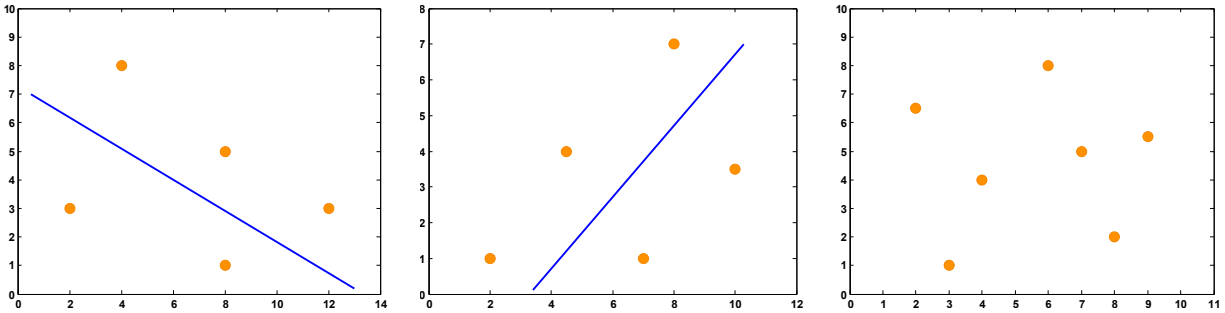


Figure 5.6: *Graphical representation of Pearson's r for random variables* $\boldsymbol{x}$ *and* $\boldsymbol{y}$*, where* $\boldsymbol{x}$ *lies on the x-axis and* $\boldsymbol{y}$ *on the y-axis. The linear correlation between these variables is calculated as the goodness of fit between the blue line and the data.*

Pearson's r is widely used in mathematics and science, and has subsequently become known as the "chi-squared test" of goodness of fit [37]. It measures the degree of linear dependence between two variables and results in values $r \in [-1, 1]$. A positive correlation implies a linear fitting with a positive gradient, whereas a negative correlation implies a fitting with a negative gradient. A zero correlation implies that no linear fitting to the data is possible. A result of $r = \pm 1$ is achieved if the linear fit agrees with the data completely and without any points missed. Graphical interpretation of Pearson's r is shown in Figure 5.6.

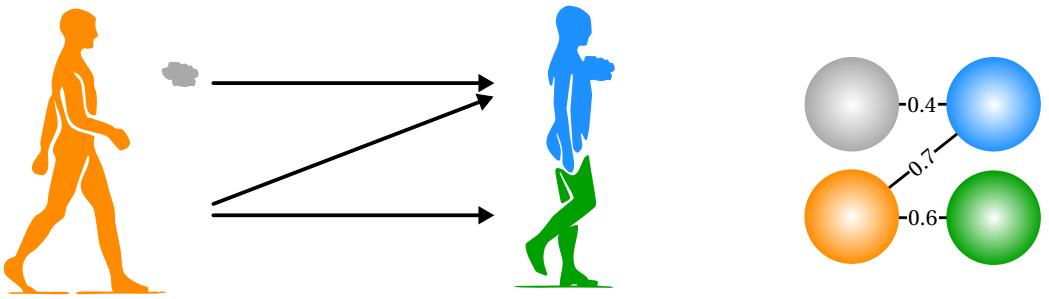
We are now able to calculate a tangible measure between features of different blobs. Moreover, Pearson's r provides magnitudes between 0 and 1, which we will show is especially useful as a similarity measure in calculating the likelihood of blobs corresponding over time.

5.6 Enforcing the parent constraint with features

The parent constraint requires that a blob cannot be part of a split and a merge from one time step to the next, as mentioned in Section 5.3. The proposed method of Massoud and Papanikolopoulos [31] requires the calculation of all the edge combinations resulting in a valid graph from time $t-1$ to time t. They then score all the valid graphs in order to pick the best edge combination. This method can become computationally expensive when there are many edge combinations to evaluate.

We propose an alternative method: instead of calculating a score for all of the possible edge permutations, we rather weigh each edge and use this as a comparison measure. This weighting is done by calculating the correlation coefficient between the feature vectors of adjacent vertices. Pearson's r calculates a probabilistic measure between two variables and is therefore a suitable metric to use.

Consider the example shown in Figure 5.7 with the edges of the graph in (b) weighted according to correlation magnitudes. To fix the parent constraint we systematically remove the



(a) (b)

Figure 5.7: *Graphical interpretation of our revised approach to fixing a violated parent constraint. A detection from time step $t-1$ to time step t is shown in (a), with detections coloured according to the resulting graph shown in (b). The edges in (b) are weighed according to the correlation coefficient between adjacent vertices.*

edge with the smallest weight until the graph is valid. In our example, we would remove the edge between the grey and blue nodes, with weight 0.4. This results in a graph that satisfies the parent constraint, as there is only one vertex with degree more than one.

It is clear that even with a lot of edges present in the violating graph, the calculation of each edge weight is far less expensive than calculating all possible permutations and for each of those a corresponding cost. We do note that this method removes edges in a greedy manner and may therefore result in a suboptimal configuration. However, we find that in most cases the removal of a single edge rectifies the violating graph. Since this approach reduces computational complexity significantly, we decide to use it.

This covers all the theory necessary to calculate correspondences between detected blobs from one time step to the next. In the next chapter we introduce our improvement over the Markovian approach (where correspondences are considered only between time steps $t-1$ and t) used thus far, with our *window tracking scheme*.

Chapter 6

Window tracking scheme

We now have all the theoretical tools necessary to detect and track an object in a video sequence: we are able to build a background model (Section 4.1), interpret the foreground as individual blobs (Section 4.2), determine putative correspondences between blobs over time by using their locations (Sections 5.2 and 5.6), and calculate correlation magnitudes between such corresponding blobs (Section 5.5).

A main concern during the blob detection step is the possibility for errors such as those listed in Section 5.1 (notably splits and merges). These errors can occur when parts of the object and the background are similar in appearance. This can result in a person being detected as multiple blobs, or multiple people being detected as one blob. The correspondences graph would then interpret this as a split or a merge, respectively, which is incorrect. These errors can occur quite frequently when colour data is used, but it is also possible for them to occur when using thermal data. An example of a split would be when a person walks past a hot air vent, engine or exhaust pipe, or if a person is wearing a backpack which occludes his heat emission.

Up until now we have considered only a Markovian approach to calculating the correspondences between blobs, as we determine correspondences only from time $t-1$ to time t. We will show the usefulness of rather considering a broader range of detections in Section 6.2. This is done to introduce more information into the system in order to fix correspondence errors. The first step, however, is to explain the introduction of *dummy nodes* at each time step where a split, merge or death is detected.

6.1 Dummy nodes

As mentioned in Chapter 5, we have detected blobs from the previous time step $t-1$ and newly acquired blobs at time t, and some constraints place possible correspondences between them. These correspondences are expressed in the form of a directed bipartite graph G_t with vertex set $V_{t-1} \cup V_t$ and edge set E_t at time t. For clarity we may refer to all the ver-

tices V_{t-1} in the previous time step as the *parents* and all the vertices V_t in the current time step as the *children*.

It is important to note that we consider it impossible for an object to actually split or merge. When a split is detected, say blob $v_1 \in V_{t-1}$ is connected to $v_2, v_3 \in V_t$, it can be interpreted as edge (v_1, v_2) with v_3 being born, or as edge (v_1, v_3) with v_2 being born. A similar argument holds for a merge. If blobs $v_1, v_2 \in V_{t-1}$ are connected to blob $v_3 \in V_t$, then we can have edge (v_1, v_3) with v_2 dying, or edge (v_2, v_3) with v_1 dying. This observation springs from the fact that the only logical way a split can occur is when two objects enter the scene together, be detected as one, and then at a later point in time separates (and thus a new blob is born). The only way a merge can occur is when two objects intersect (from the camera's point of view), and are detected as a single blob. This leads to the one occluding the other, and hence one of the blobs in the previous time step dies.

We handle splits, merges and deaths by adding dummy nodes to compensate for these situations. This forces what we consider to be the "correct" scenario as a possibility into the graph. In doing so we refrain from making hard choices and we let the system determine the most likely outcome. We need location measurements for the newly added dummies as we need to build feature vectors for each.

6.1.1 Split-dummies

Whenever a split occurs in the graph we add a split-dummy to the graph at the current time step. A bounding box for such a dummy is calculated by concatenating the bounding boxes of all the children involved in the split. This bounding box is then used to calculate a feature vector, as in Section 5.4. An example is shown in Figure 6.1 (a) where the bounding boxes of the blobs are shown in red and that of the dummy is shown in grey. This example shows the progression over three time steps, as it justifies the inclusion of the dummy. Figure 6.1 also shows all the possible, viable edge combinations that can be considered when calculating the most likely outcome. It should be clear in this case that out of all the possible outcomes, the one involving the dummy node is favoured most.

6.1.2 Merge-dummies

A detected merge is handled a bit differently, as it consists of more parents than children. For each parent involved in the merge, we add a merge-dummy to the graph at the current time step and connect it to its parent, as shown in Figure 6.2 (b). The problem we have with these merge-dummies is that, unlike the split-dummies, we do not have any concrete bounding box measurements to work with from which to calculate their features. This is due to the fact that the child involved in the merge is in fact an occlusion, or at least a partial occlusion of one or more of the parents. We can therefore not use the same approach as with split-dummies. We are however busy tracking these blobs over time, which implies that we have

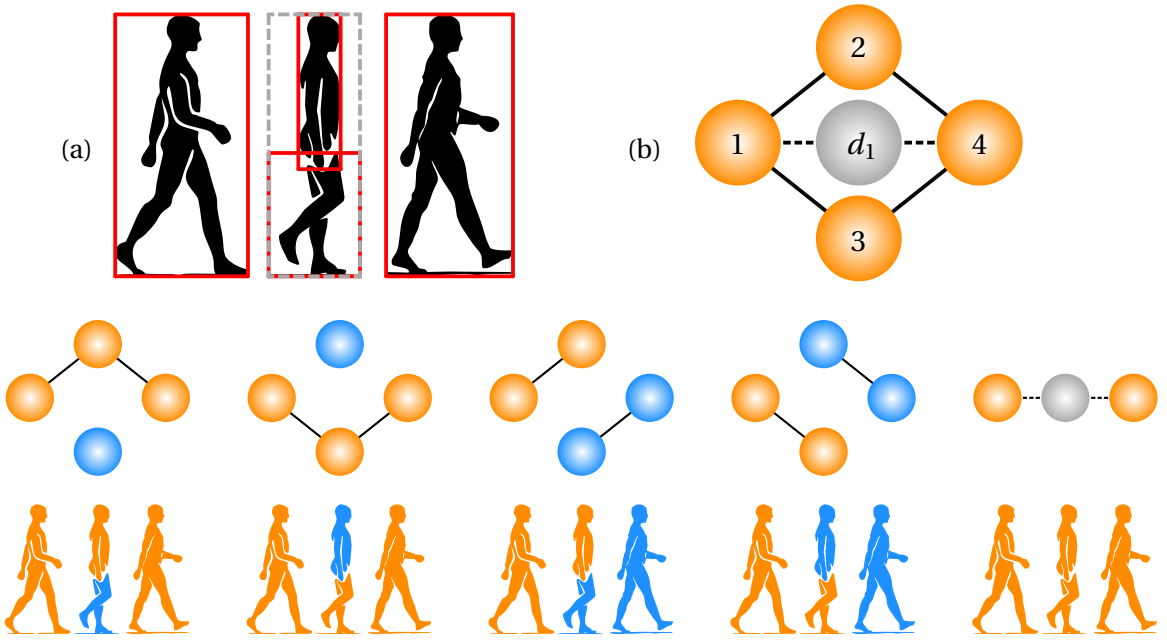


Figure 6.1: *A detected split is shown in (a) over three time steps, with bounding boxes shown in red and the added dummy's bounding box in grey. This detection results in the graph shown in (b), where the added dummy node is visible. This adapted graph results in five possible subgraphs, each having a different set of edges. The correspondences induced by these different edge sets are indicated below them.*

some kind of filter implemented.

As mentioned in Chapter 3, such a filter consists of a prediction followed by a measurement update step. We can therefore use the prediction of the state (location) of the blob at the previous time step as an estimate for the blob at the current time step. Using this approach enables us to calculate an estimate of each of the parents' bounding boxes, at the current time step, which can then be used to calculate feature vectors for each of the merge-dummies.

This is illustrated in Figure 6.2 where once again we have detections over three time steps indicating a merge. The bounding boxes of the detected blobs are shown in red with the predicted bounding boxes for the merge-dummies shown in grey. It is clear that out of all the possible outcomes shown in Figure 6.2, the one involving the dummy nodes is favoured most.

6.1.3 Death-dummies

When a death is detected, that is to say a blob in V_{t-1} is not connected to any blob in V_t, we can assume that it might have been missed in the current time step. This can occur because of an occlusion or noise in the data. We therefore add a death-dummy to the graph at the current time step, and connect it with that blob. Once again we need a measurement for the

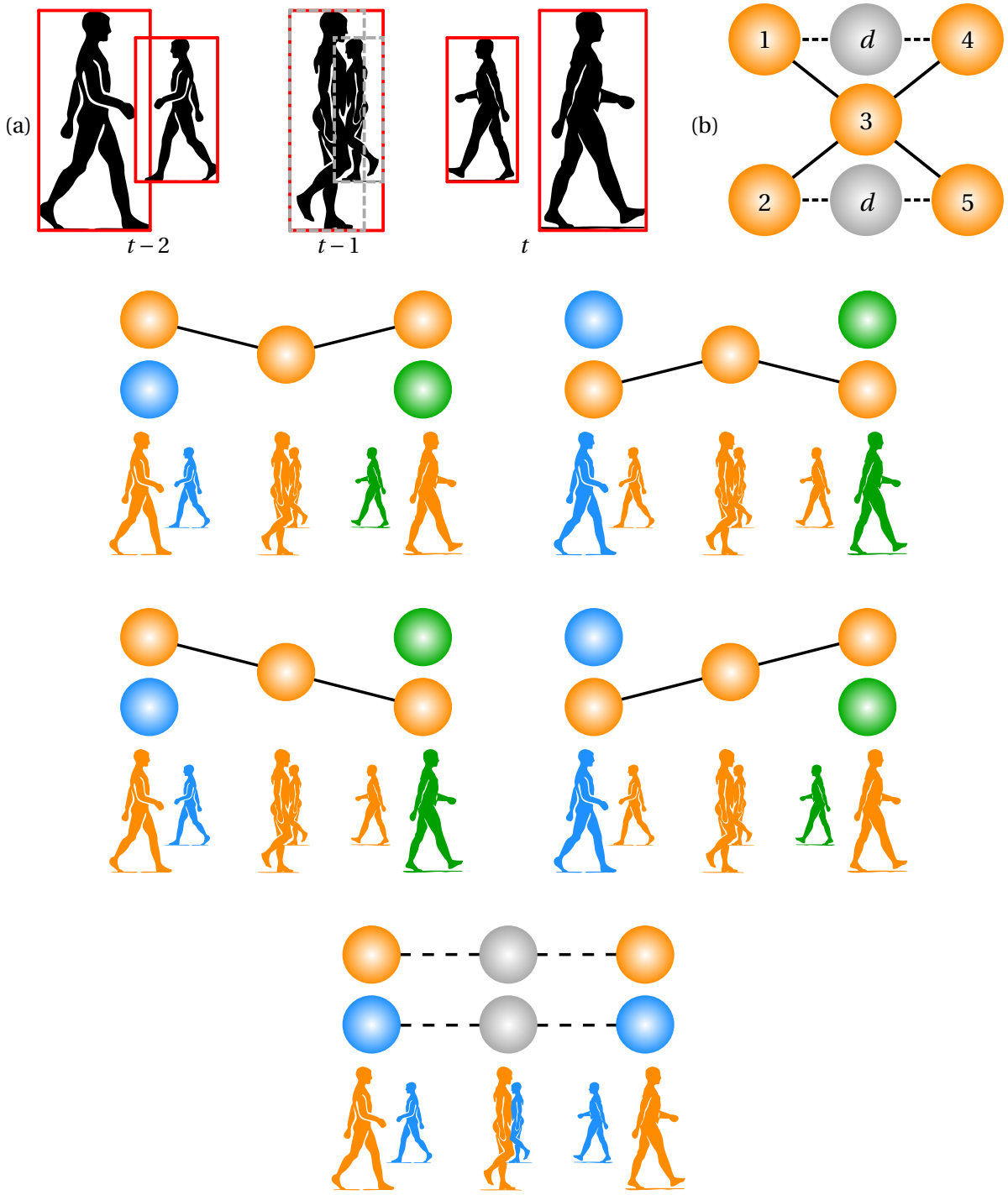


Figure 6.2: *A detected merge shown in (a) with the corresponding bounding boxes shown in red. This detection results in the graph shown in (b), where the added dummy nodes are visible. This results in seven possible subgraphs, each with a different set of edges inducing the different labellings shown.*

dummy, so that we can calculate its feature vector. We can use the projection of the bounding box of the blob in the previous time step as an estimate for the death-dummy in the current time step, as was done with merge-dummies.

6.1.4 Dummy overview

The way in which we introduce dummy nodes into the graph has the additional feature that even if the detections were correct, and we add the extra dummies, the system can still decide to ignore them if the correlation coefficients deem them to be improbable. If we therefore add a dummy which is not supposed to be there, it should not influence the results as we move to the next time step.

The addition of dummy nodes becomes a bit tricky once we encounter splits, merges and deaths involving other dummy nodes, as these additions can explode in numbers if we do not include a few extra rules. Without such rules we may, for example, inadvertently add merge-dummies corresponding to newly added split-dummies or vice versa. It is necessary to introduce a new formulation to express these rules clearly. We denote the parent nodes as $v_{t-1}^i \in V_{t-1}$ and the children nodes as $v_t^j \in V_t$. This allows us to summarize all the rules as follows:

- if v_{t-1} splits to $v_t^1, v_t^2, \cdots, v_t^k$,
  add a split-dummy σ_t and edge (v_{t-1}, σ_t);
- if $v_{t-1}^1, v_{t-1}^2, \cdots, v_{t-1}^k$ merge to v_t,
  add merge-dummies $\mu_t^1, \mu_t^2, \cdots, \mu_t^k$ and edges $\left(v_{t-1}^i, \mu_t^i\right)$ for $i = \{1, 2, \cdots, k\}$;
- if v_{t-1} dies,
  add a death-dummy δ_t and edge (v_{t-1}, δ_t);
- if $v_{t-1}^1, v_{t-1}^2, \cdots, v_{t-1}^k$ and $\mu_{t-1}^1, \mu_{t-1}^2, \cdots, \mu_{t-1}^l$ merge to v_t,
  add merge-dummies $\mu_t^1, \mu_t^2, \cdots, \mu_t^l$ and edges $\left(\mu_{t-1}^i, \mu_t^i\right)$ for $i = \{1, 2, \cdots, l\}$;
- if any dummy d_{t-1} (that is a split-, merge- or death-dummy) splits to $v_t^1, v_t^2, \cdots, v_t^k$,
  add a split-dummy σ_t and edge (d_{t-1}, σ_t);
- if any dummy d_{t-1} dies,
  add a death-dummy δ_t and edge (d_{t-1}, δ_t).

We purposefully omitted a rule from the above list, as it does not involve the addition of any dummy nodes. Rather, it suppresses the addition of merge-dummies where at least one split-dummy is present in a merge. This rule can be formulated as:

- if $v_{t-1}^1, v_{t-1}^2, \cdots, v_{t-1}^k$ and $\sigma_{t-1}^1, \sigma_{t-1}^2, \cdots, \sigma_{t-1}^l$ merge to v_t, add nothing.

The small examples in Figures 6.1 and 6.2 demonstrate the usefulness of adding dummy nodes to assist the correspondence problem. The problem however is that the system only considers the correspondences between blobs from the previous to the current time step when making decisions. If for some reason the system chooses the wrong permutation of edges at this time step, that error will propagate to the next step and could cause the tracker to fail. As an example, consider the situation in Figure 6.1. If we consider only the correspondences from the first to the second time step (from left to right in the figure), it can happen that the edge linking the body to the split torso appears to be more likely at that stage than the edge connecting the split-dummy. However, by considering a larger window of time steps, there should be enough evidence that the track involving the split-dummy is in fact the most probable.

Another issue we may encounter is that splits and merges can span over multiple time steps before meeting up or separating again. This leads to the conceptualization of our *window tracking* scheme, in which we consider the behaviour of the system over multiple time steps, instead as just two, to effectively smooth the tracking results.

6.2 Window tracking

We are now able to handle splits, merges and deaths by adding dummy nodes, as described in the previous section. The Markovian way of choosing the best permutation of edges, that only considers edges from the previous time step to the current one, can lead to wrong decisions being made in the short term. This approach can be greatly improved by considering a broader range of time steps and then optimizing across them all. We plan to do this by building a composite graph spanning this broader view, and then weighing each edge according to the correlation coefficient between its vertices. We then determine all possible tracks through this graph and calculate a score for each using the edge weights. These scores can then be compared to determine the most probable collection of tracks for that window.

Up until now we have calculated the correspondence graph G_t relating blobs at time t to blobs at time $t-1$. We build a composite graph, which spans a window of n time steps, by scanning ahead with the detector and concatenating the graphs at each time step together to form the composite graph $G_{t:t+n}$. This composite graph now has vertex set $\bigcup V_i$ and edge set $\bigcup E_i$ for $i \in \{t, t+1, \cdots, t+n\}$. Every edge in the graph is weighted by the magnitude of the correlation between its two adjacent feature vectors.

Next we need to calculate all valid tracks through the composite graph. Let us first consider the scenario where $G_{t:t+n}$ only consists of a single connected component. If this component does not contain split- or merge-dummies it corresponds to a clear path without ambiguity. If the connected component has split- or merge-dummies, we need to calculate all possible

tracks through the graph (as was done for the examples in Figures 6.1 and 6.2). These tracks can possibly be disconnected, as the inclusion of dummies may imply that some blobs are occluded.

In order to find all the possible tracks we traverse the graph from time t to time $t+n$. Edges that do not form part of a split or a merge are always included in every possibility. That is to say, we do not break tracks unnecessarily.

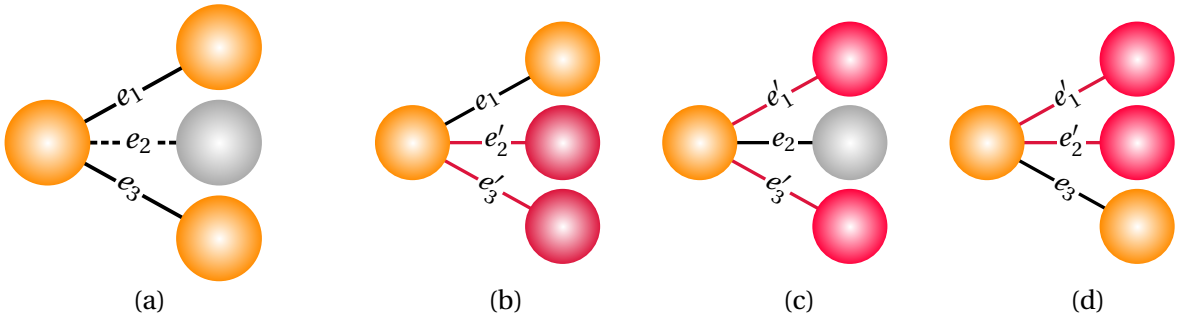
When a split is encountered we include, as separate possibilities, every edge involved in the split. If such an edge connects a node at time k to a normal node (not a dummy) at time $k+1$, we furthermore include all the other normal nodes at time $k+1$ involved in the split as they now coincide with new blobs being born. If the edge to be added connects a normal node with the split's dummy node, we do not add the other (normal) nodes since the split-dummy effectively supersedes those nodes from being born.

When a merge is encountered we include, as separate possibilities, every edge involved in the merge. If any such edge connects a normal node with its corresponding merge-dummy, we immediately also include the edges connecting the other normal nodes with their merge-dummies. Thus we always choose between either one node going to the detected merge (and the others dying) or all nodes going to their respective merge-dummies.

These possible tracks are enumerated and extended recursively, until we get to the last time step of the window. This enumeration is fairly straightforward for small examples, as seen in Figures 6.1 and 6.2, but the total number of tracks increases with increased graph complexity and the number of dummy nodes present. A larger split detection is shown in Figure 6.4 where this increase in the number of possible tracks is evident.

Having determined all possible tracks through $G_{t:t+n}$ we need a method to calculate a score for each of them. We use the Pearson correlation between connected blobs (calculated from their feature vectors) as edge weights in $G_{t:t+n}$. Since such a value is between -1 and 1, with 0 indicating no correlation and ± 1 indicating perfect correlation, we regard its magnitude as an indication of the probability that the correspondence is correct. It is known [49] that the strength of the correlation is not dependent on the direction (or the sign) of the correlation. This implies that a correlation measure of $p = 0.8$ and $p = -0.8$ are equal in the degree of association between the measured variables.

We can now calculate a joint probability that all the edges in a possible track are correct, but we must also take into account the probabilities of those edges that are excluded. This means that we have to calculate the joint probability of traversing the edges contained in the track and of **not** traversing the edges excluded from that track. If we do not take these complement events into account, we are essentially marginalizing over the possibilities of including and excluding those missing edges.



(a) (b) (c) (d)

Figure 6.3: *A detected split is shown in (a) with the added dummy node in grey. When calculating the score for each track, in this case just a single edge, we have to calculate the joint probability of including edge e_1 as well as excluding edges e_2 and e_3, as in (b). The same reasoning holds for choosing e_2 in (c) and in choosing e_3 in (d).*

Let us explain this concept with a simple example. Say we have a detection, as in Figure 6.3 (a), where vertex $v_1 \in V_{t-1}$ splits to $v_2, v_3 \in V_t$. We add a dummy at time t and label the edges as shown. In calculating a score for including e_1, as in Figure 6.3 (b), we take into account its weight as well as the complement weights of e_2 and e_3. The same reasoning holds for the other edge inclusions shown.

Suppose a particular track segments the edge set E of the original graph into a set of included edges $E_{\text{incl}} = \{e_1, e_2, \cdots, e_k\}$ and a set of excluded edges $E_{\text{excl}} = \{e'_1, e'_2, \cdots, e'_l\}$. Each of these edges e_i has a probability of being included in the correct track, which we write as $p(e_i)$. The probability of excluding an edge is $1 - p(e'_j)$. The joint probability associated with the track is then

$$\left(\prod_{i=1}^{k} p(e_i)\right)\left(\prod_{j=1}^{l}\left(1 - p(e'_j)\right)\right), \tag{6.2.1}$$

where we assume statistical independence between the edges.

We may now consider all the possible segmentations of E into two subsets, calculate a joint probability for each using the above formula, and pick the segmentation that is found to be most probable. There are 2^{k+l} possible segmentations, where $k+l$ is the number of edges in the original graph, but we need only consider those tracks deemed physically possible by the procedure discussed above. This makes the task of finding the most likely solution tractable.

A problem with this formulation of the track score is that it multiplies probability values (between 0 and 1) together. This inadvertently leads to very small scores which can be troublesome to compare reliably. We know that multiplications turn to additions in log-space [47], so by taking the log of the joint probability in (6.2.1) of a track we get

$$\left(\sum_{i=1}^{k} \log\left(p(e_i)\right)\right) + \left(\sum_{j=1}^{l} \log\left(1 - p(e'_j)\right)\right). \tag{6.2.2}$$

Calculating the track likelihood is therefore a matter of adding the logs of included edge weights and the logs of the complements of the excluded edge weights together. Figure 6.4 shows all the valid tracks through an example graph, with the edge weights as indicated, and the score for each possible track underneath it. It is noteworthy that even though the first edge (top left) has $p = 0.7$, which would make it the best choice with a Markovian approach, the tracks containing it only score -14.2 and -13.4, which are lower than the best score of -12.3. This best scoring track is the one containing the dummies. The example indicates that our window tracking scheme has the potential to work in practice.

6.3 Labelling

The final step is to label the blobs according to the best scoring tracks through each component of the graph. We have the composite graph $G_{t:t+n}$ containing both labelled and unlabelled blobs, and we assume that all labels assigned up to the current time t is correct. We calculate all the valid tracks through each component of the graph and determine the best scoring track for each component. We scan through all the vertices at time t, and

- if a vertex has a label and is part of the best track we propagate its label forward by relabelling all the vertices on that track;
- if a vertex does not have a label, but now lies on the best track we assign a new label and propagate it forward by relabelling all the vertices on the track;
- and we remove the labels from all the vertices in the entire graph $G_{t:t+n}$ that do not lie on the best track.

With all the blobs in the current window labelled we can send this information to a tracking filter to update the states of each object. We would therefore have a slight delay between the detector and the tracker, as the detector scans ahead n time steps and the tracker only outputs at time t. An alternative would be to output the labels at the end of the window under consideration, i.e. at time $t+n$. However, since those blobs have only just entered the system, we expect them to be less stable than those at the beginning of the window.

6.4 Synopsis

We now have all the tools and theory covered to implement our automatic object detection and tracking system, which uses both thermal and colour video data to optimize the tracking. We choose to combine the best qualities of thermal and colour data to produce a system which should be more robust than one that uses only thermal data or only colour data.

We have a **background subtraction** algorithm implemented which updates the background model over time. This update should include newly static objects, such as a newly parked

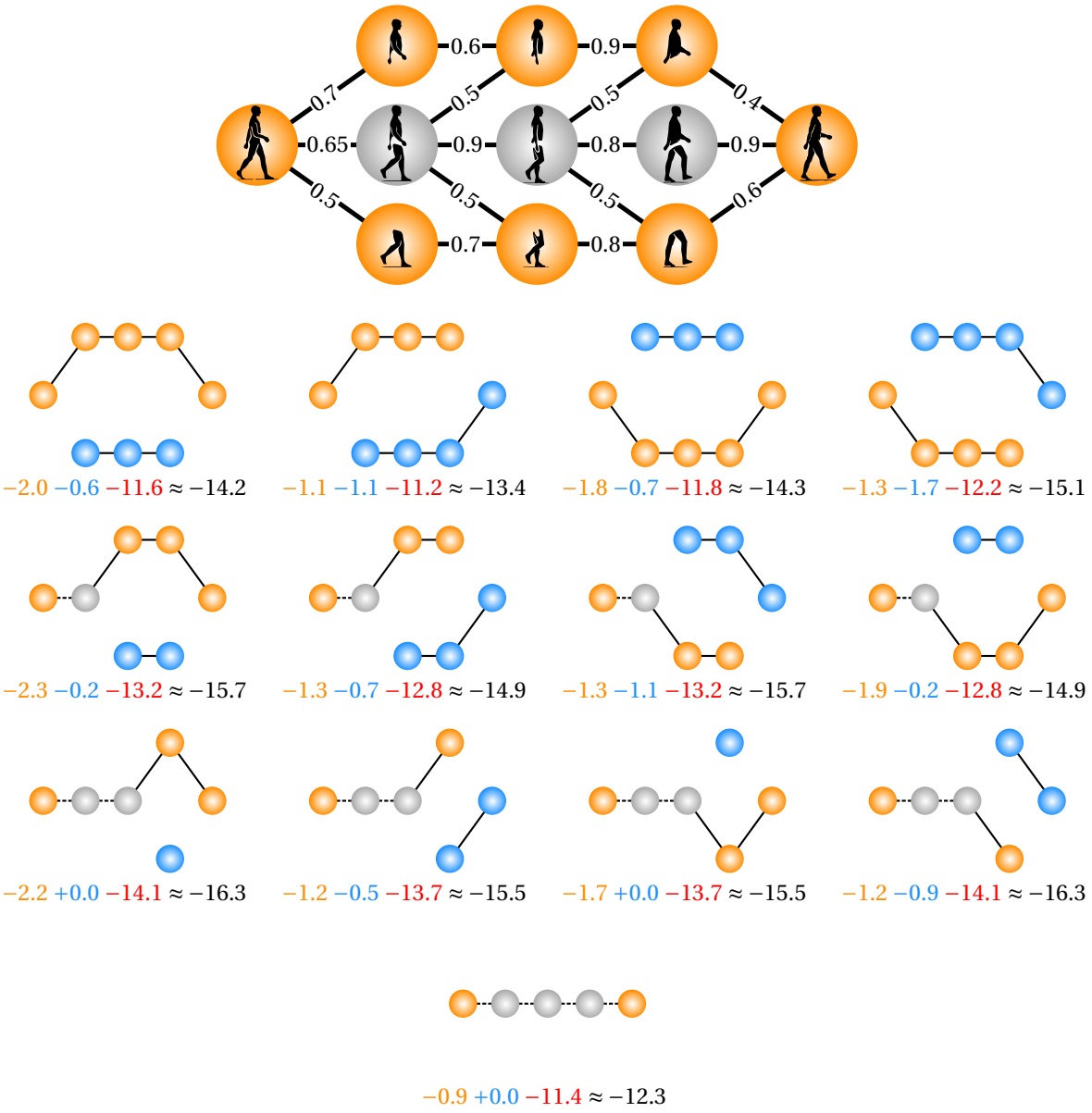


Figure 6.4: *The composite graph induced by a split over multiple time steps is shown with grey blobs indicating dummy nodes and edges weighted as shown. When choosing a track involving a dummy, we ignore edges incident with the normal blobs at that time step, and vice versa. This leads to all the valid tracks for the given graph as shown. The score for each track is shown, where the orange score corresponds to the track through the orange nodes, the blue score corresponds to the track through the blue nodes and the red score corresponds to the complement weights of excluded edges.*

car, into the background as it is no longer an object of interest. Performing morphological clean-up and using **connected-component labelling** allows us to interpret the foreground pixels as localized clusters, or blobs.

Considering consecutive detections from one time step to the next, we determine the **cor-**

respondences between blobs. This is done by enforcing the **locality** and **parent** constraints. These correspondences are expressed in the form of a bipartite graph with all the detected blobs at the previous time step in a vertex subset and those detected at the current time step in another. Vertices are connected with an edge if there is evidence that they may correspond to each other.

If a split or a merge is detected we assume that the detection might have been incorrect, by arguing that it is impossible for a person to actually split or merge, and we add **dummy nodes**. This forces a possible "correct" option into the graph, but we refrain from making any hard decisions and let the system determine the most likely edges. For this we calculate **feature vectors** for each detected blob and added dummy. This is done by mapping the location of each detected blob to the colour image. The colour data is then used to build the feature vector, as there is very little distinctive information in the thermal image.

We find that approaching the tracking in a Markovian fashion can lead to bad decisions in the short term, which can propagate through time and cause the tracker to fail. We rather develop a **window tracking** scheme, which works by scanning ahead with the detector and building up a composite graph of all the detected blobs and their correspondences. We calculate all valid tracks through the composite graph and calculate a score for each. This score is determined by calculating the edge weights between vertices as the correlation between their feature vectors. We can thus determine the best scoring tracks through each connected component of the composite graph.

These best tracks are then used to **label** the blobs connected by them. This enables the system to update long term correspondences based on new detection data, as it is possible to alter the choice of edges in the next time step. We use the labelling as input for each of the tracked objects' **tracking filter** and display the output at the current time step.

In the next chapter we discuss our implementation and experimental results.

Chapter 7

Experimental results

For our implementation we use thermal and colour datasets made available by Conaire et al. [8]. The data was recorded using a Raytheon Thermal IR-2000B and a Panasonic WV-CP470 in a beam-splitter configuration, using Pilkington K-glass as the beam-splitter medium. The datasets are surveillance sequences of pedestrians walking outside during day time and a few frames can be seen in Figure 7.1. Our goal is to accurately detect and uniquely track objects over time in these sequences. The first step is to investigate the behaviour of the thermal data, as we need to train a background model for the initial detection step.

7.1 Background model

We recorded the values of a background pixel from the thermal video over time and found its behaviour to be almost perfectly described by a single Gaussian PDF with $\mu \approx 40.1$ and $\sigma^2 \approx 131.2$, as shown in the top row of images in Figure 7.2. This behaviour is consistent across the entire image, and we can therefore use a single Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$ and the running Gaussian average method (Section 4.1.2) to model each background pixel. We build an average and a variance image by selecting a subsequence of the dataset that is free of moving objects. Using frames without objects is beneficial, but it can also be done with frames including objects as the update equations should eventually phase out any wrongfully included foreground data. The bottom row of images in Figure 7.2 shows the thermal background frame, and the mean and variance images calculated after the model was built.

7.2 Object detection

7.2.1 Foreground detection

With an appropriate background model in place it is possible to determine which pixels from a new frame do not fit the model and must be considered foreground. The foreground pixels

Figure 7.1: *A few frames taken form the datasets we use for our experiments.*

are determined by thresholding over the difference between the pixel value in the frame to that of the background model, as

$$I(x,y) - \mu(x,y) > k\sqrt{\sigma^2(x,y)}, \tag{7.2.1}$$

where $\mu(x,y)$ and $\sigma^2(x,y)$ are the mean and variance of pixel (x,y) in the background model. In our experiments we found that $k = 1.5$ is sufficient. We ignore connected clusters whose areas are smaller than 50 pixels, and perform morphological closing (Section 4.2.1.3) on the remaining foreground pixels. This leads to foreground blobs which we can track over time. At this point it is convenient to calculate a feature vector for each blob, as it is used in enforcing the parent constraint during the initial determination of correspondences, as well as in the final window tracking scheme.

7.2.2 Feature vectors

Since we do not have access to any calibration parameters, and we need a way to relate points in the thermal and colour images, we calculate a plane-to-plane homography (Section 2.2) that maps points in the thermal image to points in the colour image. We divide the walking surface in the scene into two planar surfaces, one for the inclined walkway and one for

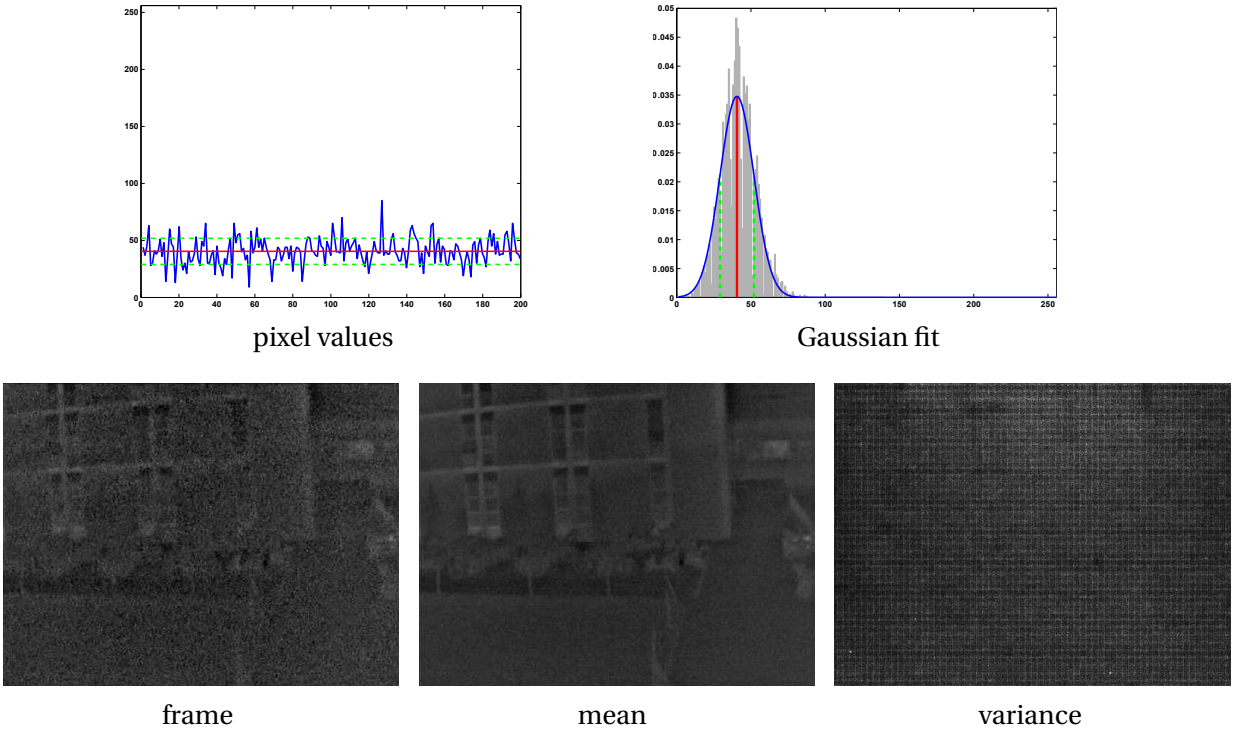


Figure 7.2: *Using a single Gaussian we are able to build a background model for the thermal data. The top images show the actual values of a pixel over time and the corresponding histogram. Here the red line indicates the mean and the green dashed lines the standard deviation. The bottom row of images shows a thermal frame, the mean background image and the background variance after the model has been built using the equations from Section 4.1.2.*

pathway that runs vertically across the right side of the images, and calculate a homography for each. The first two images in Figure 7.3 show some thermal detections, with the surfaces used to calculate separate homographies shown, and the corresponding colour mappings. We also show the detected objects with their corresponding features. In building a feature vector we only include the hue, saturation and HOG histograms, as the intensity values are too sensitive to lighting changes. For increased execution speed and some smoothing we use a bin size of 10, which results in a feature vector of length $26+26+36=88$.

7.3 Object tracking

Now that we have detected blobs we can calculate their correspondences over time. This is done by implementing the locality and parent constraints (Chapter 5), which we base on the work of Masoud and Papanikolopoulos [31]. The parent constraint is adapted to rather use similarity measures between the feature vectors of adjacent objects as this lowers computation time. These constraints determine an initial correspondence between detected blobs from one time step to the next. We know that we can encounter erroneous correspondences in terms of splits and merges and we handle them by incorporating dummy nodes and implementing our window tracking scheme. We first explain our implementation of tracking

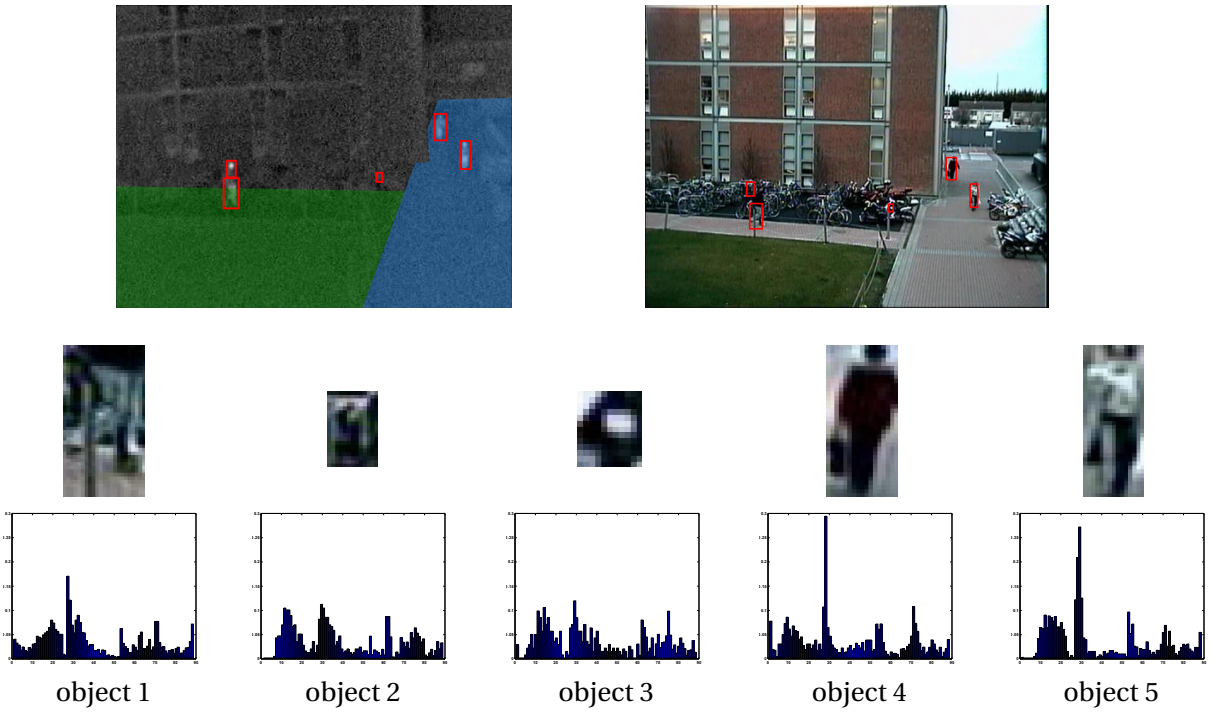


Figure 7.3: *Once we have detected blobs we build a feature vector for each. The top two images show the detection in the thermal image and the mapped locations in the colour image. The two planar surfaces used to calculate homographies are shown in the thermal image. All the detections and corresponding features are shown underneath.*

filters used to track every detected object.

7.3.1 Tracking filter

For our implementation we found the Kalman filter to suffice. We may model the state of an object as

$$\mathbf{x}_t = \left(x, y, w, h, a, \dot{x}, \dot{y}\right)^T, \tag{7.3.1}$$

where (x, y, w, h) corresponds to the top-left coordinates, width and height of the bounding box, and we include the positional change $(\dot{x}, \dot{y})$. The assumption here is that the shape of the bounding box should remain more or less constant, such that any measured deviation is explained by noise, while the bounding box itself moves with constant velocity. We include the area a of a blob in the state as this is used in the locality constraint.

The Kalman filter assumes that the state of an object changes linearly over time. However, we find that the top-left position of an object's bounding box can change in a nonlinear manner, with periodic variation caused by swinging limbs especially when a person moves horizontally across the image. For this reason we redefine (x, y) to be the centre of the bounding box, which should remain more stable and within the bounds of our constant velocity assumption.

Furthermore, we do not consider any external driving force input and, since we do not have any prior knowledge on the covariance between the various state variables, we assume the noise and measurement covariance matrices to be diagonal and constant.

The Kalman filter enables us to track a detected blob over time. In order to track multiple objects, we instantiate multiple filters. This approach does require reliable correspondences between the blobs detected over time, and we proceed with a discussion on how we implement that.

7.3.2 Dummy nodes

We are tracking people in a video sequence and we find it reasonable to assume that a person cannot split, merge or disappear from one time step to the next. We handle such potential errors by adding dummy nodes (Section 6.1).

Figure 7.4 shows a split and merge detection (in the thermal data) from one time step to the next, as well as the added dummies and how they are mapped to the corresponding colour frames. It is important to note in the split sequence that the mapping of the torso to the colour image is inaccurate. This is due to the nature of our mapping strategy: the homography maps points on a planar surface in the one image to a planar surface in the other. The torso is therefore believed to be located on the walking surface, from the camera's point of view. It is evident that the mapping of the split-dummy is more accurate as it is in fact on the planar surface considered by the homography. Such inaccurate mappings can work in our favour as we will be calculating correlation coefficients between adjacent blobs, and if the bounding box in the colour image is inaccurate it lowers the probability of an unwanted split to be part of the optimal solution.

Tracking results on a longer sequence are shown in Figure 7.5, on selected frames, where we consider only the previous and current time steps (that is, the complete system without window tracking). Normal detections are shown in red and the dummies in white. It is interesting to note that when considering a long range of detections, such as this, where a split stays separated for long enough the system interprets it as correct and stops adding split-dummies. It is possible to incorporate a parameter which forces the system to continue adding split-dummies, which effectively acts as a delay for breaking the split, if we have prior knowledge that this kind of detections can be present. For this example we did not expect such long splits, and had the split-delay parameter set to 4 time steps.

Another set of tracking results are shown in Figure 7.6 where two people intersect from the camera's point of view, causing a long merge and split detection. Here we increased the merge-delay parameter to 20 time steps, or roughly 1 second, and increased the split-delay to 10 time steps. It is noteworthy that these delay parameters give the desired result, especially when the two people intersect and merge.

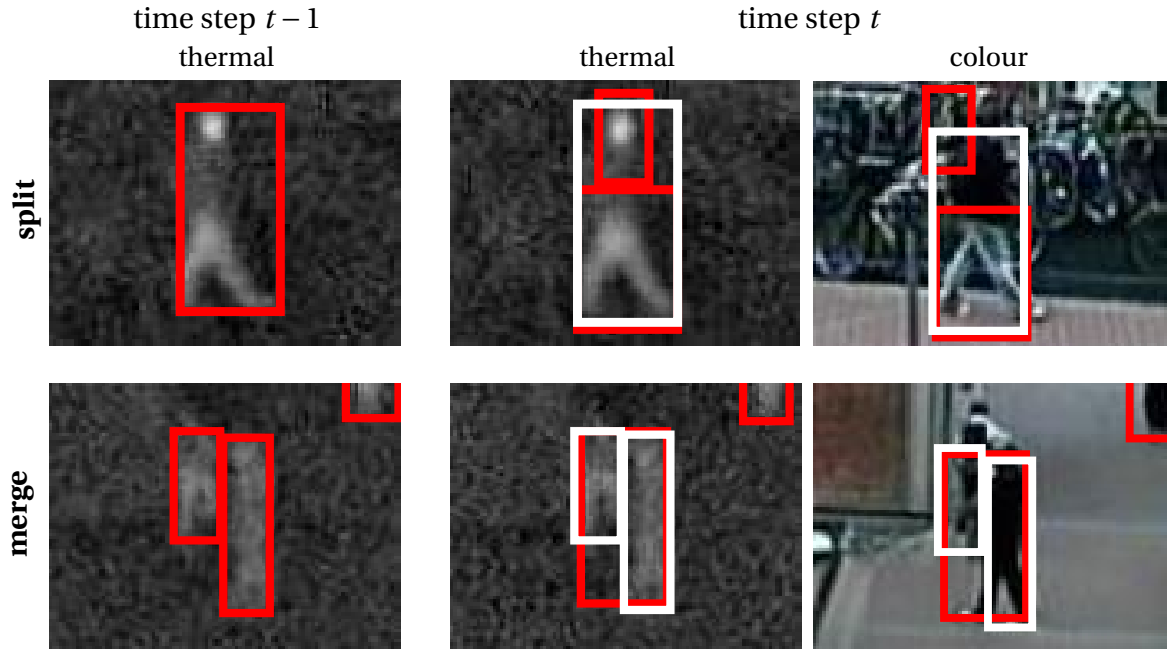


Figure 7.4: *An example of the addition of dummy nodes. The top images show a detected split and the bottom images show a merge from time $t-1$ to time t. The detected blobs are shown in red while the added dummy nodes are shown in white. It is clear that the dummies act as a reversal of the erroneous split and merge.*

It would be ideal to keep these delay parameters as large as possible to include the added dummies in the graph for the entire duration of the erroneous detection. It is however not practical to do so, as the processing speed of the system suffers greatly from it. This is due to the size of the composite graph, and the number of possible tracks, increasing with the addition of each dummy. We therefore keep these parameters down to a modest 5 frames.

These results show that dummy nodes add value to the system, as it is clear that the addition of dummy nodes includes all the relevant detections needed. This should be beneficial for our window tracking scheme, discussed next.

7.4 Window tracking

We consider all possible correspondences and build a composite graph $G_{t:t+n}$ of the next n time steps. This is done by scanning ahead and at each time step determining correspondences and adding dummies as needed. This composite graph allows us to consider the behaviour of detected blobs over time rather than just relating the previous time step to the current one.

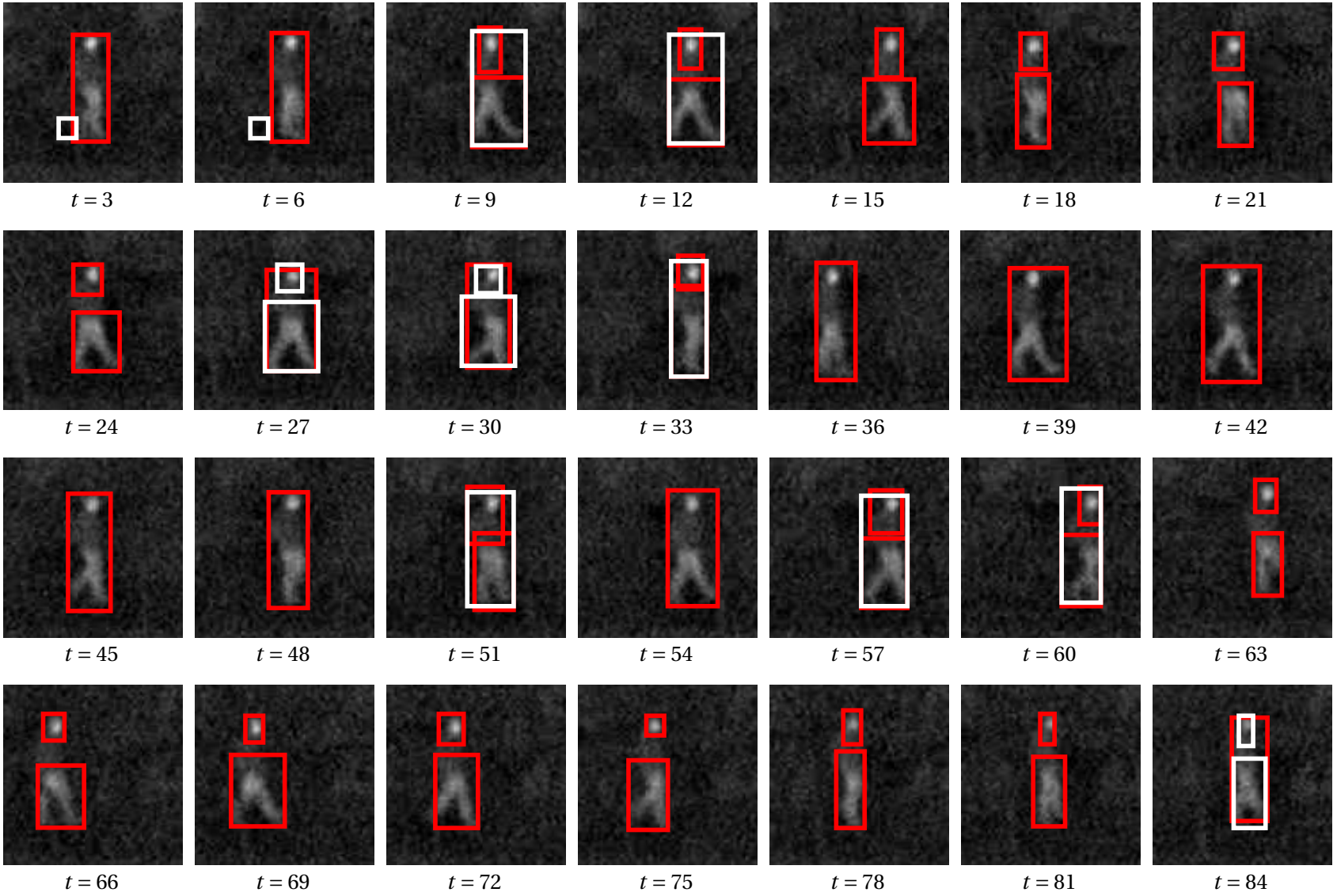


Figure 7.5: *Markovian tracking results from "sequence 1" with normal detections shown in red and dummies shown in white.*

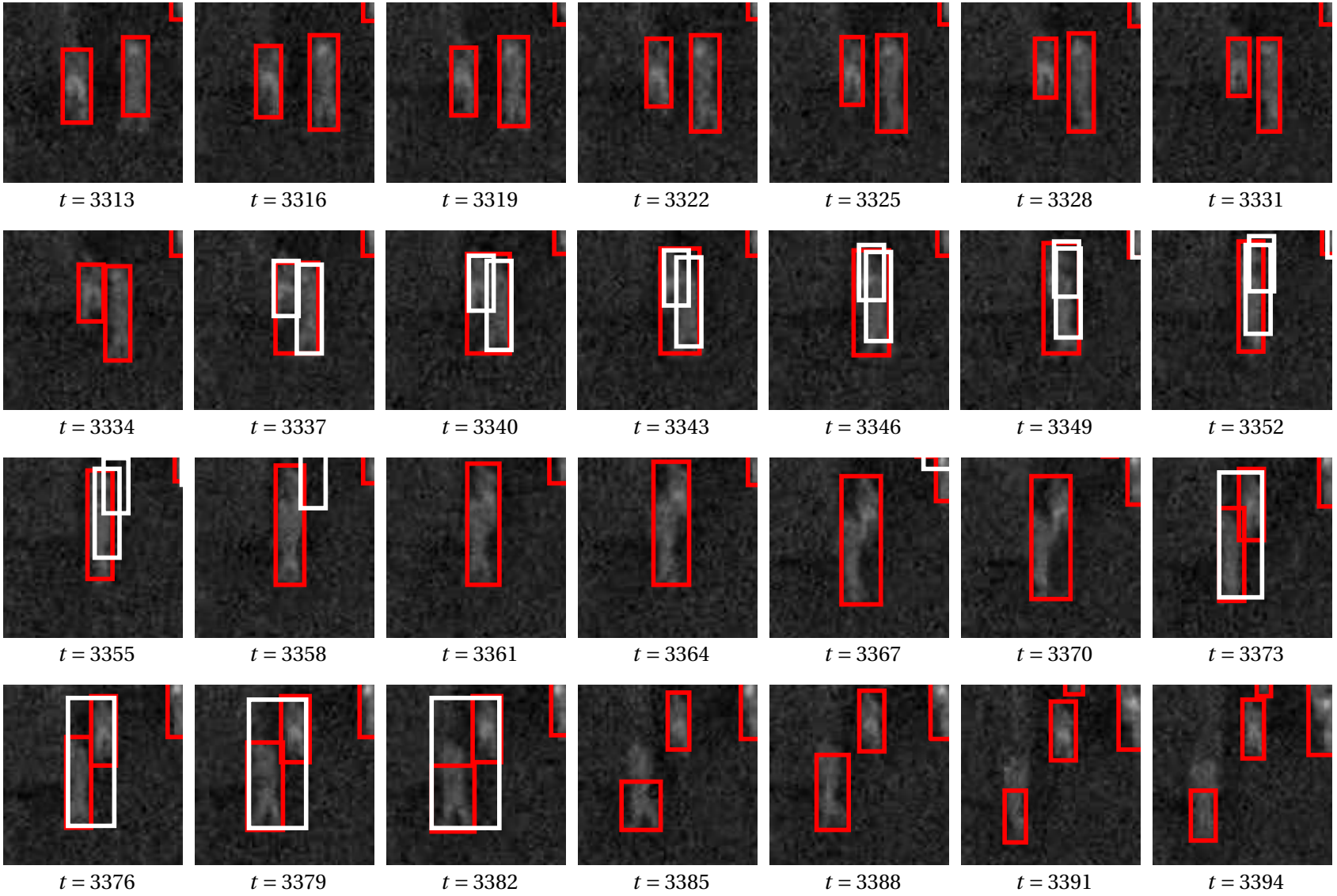


Figure 7.6: *Markovian tracking results from "sequence 2" with normal detections shown in red and dummies shown in white.*

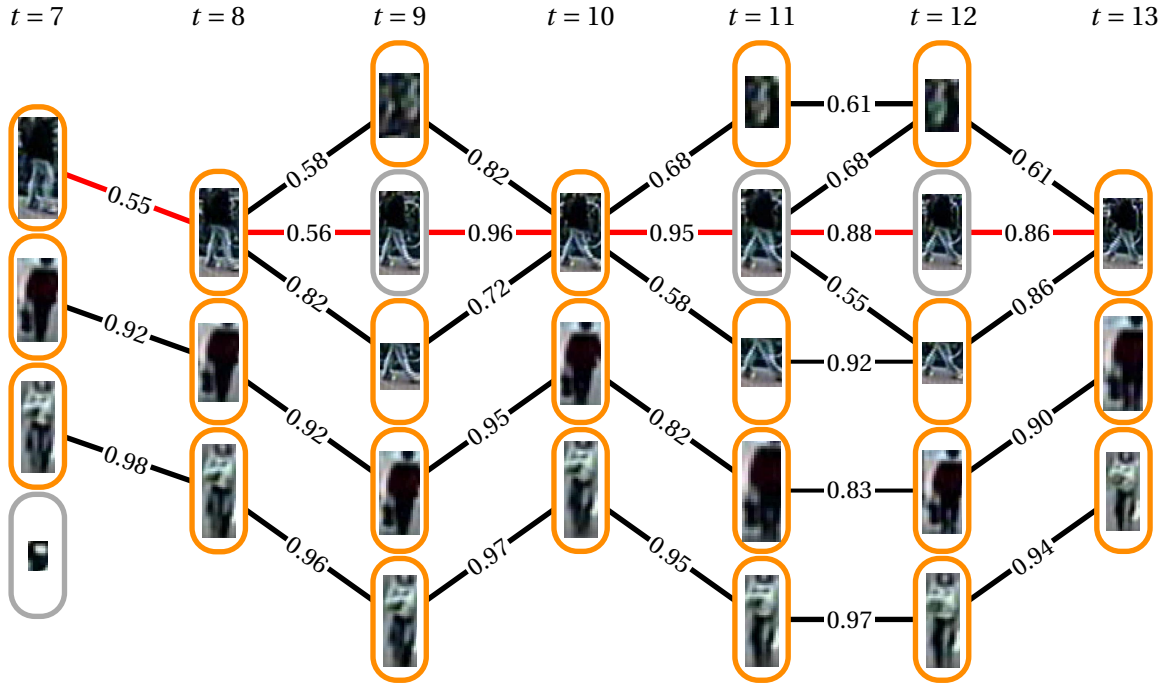


Figure 7.7: *The composite graph of a series of detections involving splits. The correlation coefficients between adjacent nodes are shown, and dummies are outlined in grey. It is interesting to look at the best path of the first component shown in red: when considering only two consecutive time steps it would have been more probable to exclude dummy correspondences, but in the whole window they are included.*

Next we calculate all possible tracks through each component in $G_{t:t+n}$ (as explained in Chapter 6), taking into account dummy blobs and all the different combinations of edges created by splits and merges. We then calculate a likelihood for each track, by incorporating the weights of included edges as well as the complement weights of excluded edges (as explained in Section 6.2).

Figure 7.7 shows the composite graph $G_{7:13}$ of a sequence of 7 time steps. This graph consists of three components with the first including two splits. The best track through the first component is also shown in red and includes the dummy nodes, as we had expected. It is quite interesting and insightful to look at the weights of the individual edges involved in these two splits. It is clear that errors would have been made if the system considered only the correspondences from one time step to the next, instead of optimizing over the entire sequence of time steps. This example very clearly illuminates the benefit of our window tracking scheme.

Using the most likely track for each component we relabel all the nodes in the composite graph. This relabelling is done by considering all the vertices at time t. If a vertex has a label and lies on a best track, we propagate its label forward and relabel all vertices on the track. If it does not have a label, but now lies on a best track, we assign a new label to it and propagate that label forward. We remove all labels from vertices which do not lie on a best track. The

resulting labelling is then used to assign a measurement to each blob's tracking filter at the current time step t.

The final state for each tracked object (at the current time step t) can now be displayed. We load the next frame and repeat all the steps mentioned, moving the window forward by one time step and recalculating optimal correspondences.

Figure 7.8 shows the progression of our window tracking and labelling method, over a number of time steps. This example is a continuation of the first component in Figure 7.7. We also simplify the representation by excluding the colour data of the objects. Here the new labels at every increment are shown in red. It is interesting to note that the window tracking scheme performs as we would expect with the first two splits, but in the third and fourth split it favours the tracks excluding the split-dummies. This could be due to some inaccuracy in mapping the bounding boxes to the colour image.

The results of the optimized state for each tracked object are displayed at the current time step (the start of the window). This induces a small delay in tracking results as the tracker first has to scan ahead n frames.

A comparison between the different detection and tracking methods discussed thus far is shown, as applied to a detected split, in Figure 7.9. Here the top row shows the raw tracking results using only the locality and parent constraints, the middle row shows the addition of dummy nodes (in white) and the bottom row shows the results of the window tracking scheme. The tracks that favour correspondences to the person's legs have not reached the front of the window yet (as seen in the last window in Figure 7.8), and so the displayed results are still that of the split-dummy.

Figure 7.10 shows the detection of a merge using the same layout as in Figure 7.9. It is interesting to see that the most basic tracking system, involving just the locality and parent constraints, actually produces very accurate results for this example. This is due to our ambiguous handling of detected splits and merges, which results in our system interpreting the merge between frames 5 and 8 as well as the split between frames 26 and 43 as possibly erroneous. The tracker includes the merge- and split-dummies for a few frames and then realizes it should rather track the actual detection(s). The window tracker fails in this example, most possibly due to an inaccurate mapping to the colour image which in turn affects the usefulness of the calculated edge weights. It could also be due to the strong correlation scores we see between consecutive splitted blobs (adjacent leg detections can have a larger correlation between them than the correlation between the full detection and the legs for instance) and merged blobs.

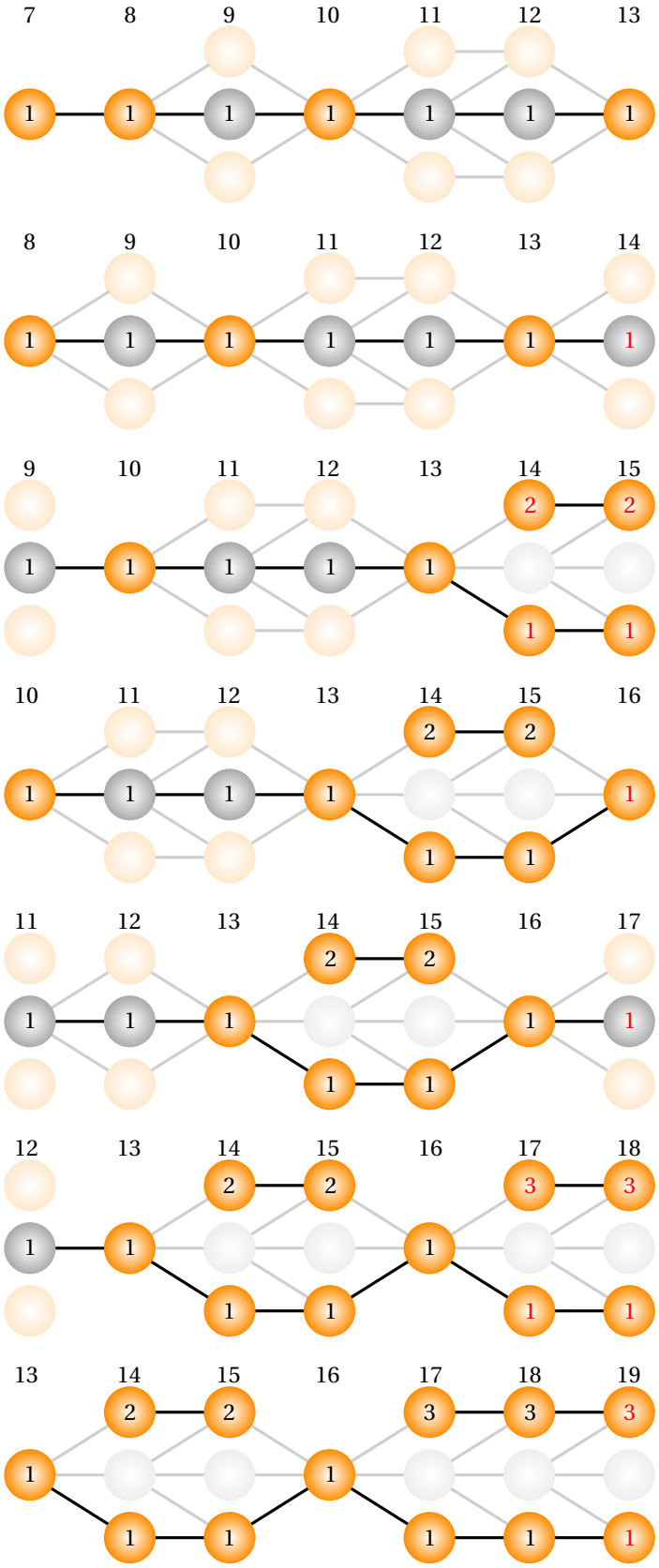


Figure 7.8: *The best tracks for a sequence of composite graphs are shown for each new time step. Here blobs are labelled according to the best scoring tracks, and new labels are shown in red.*

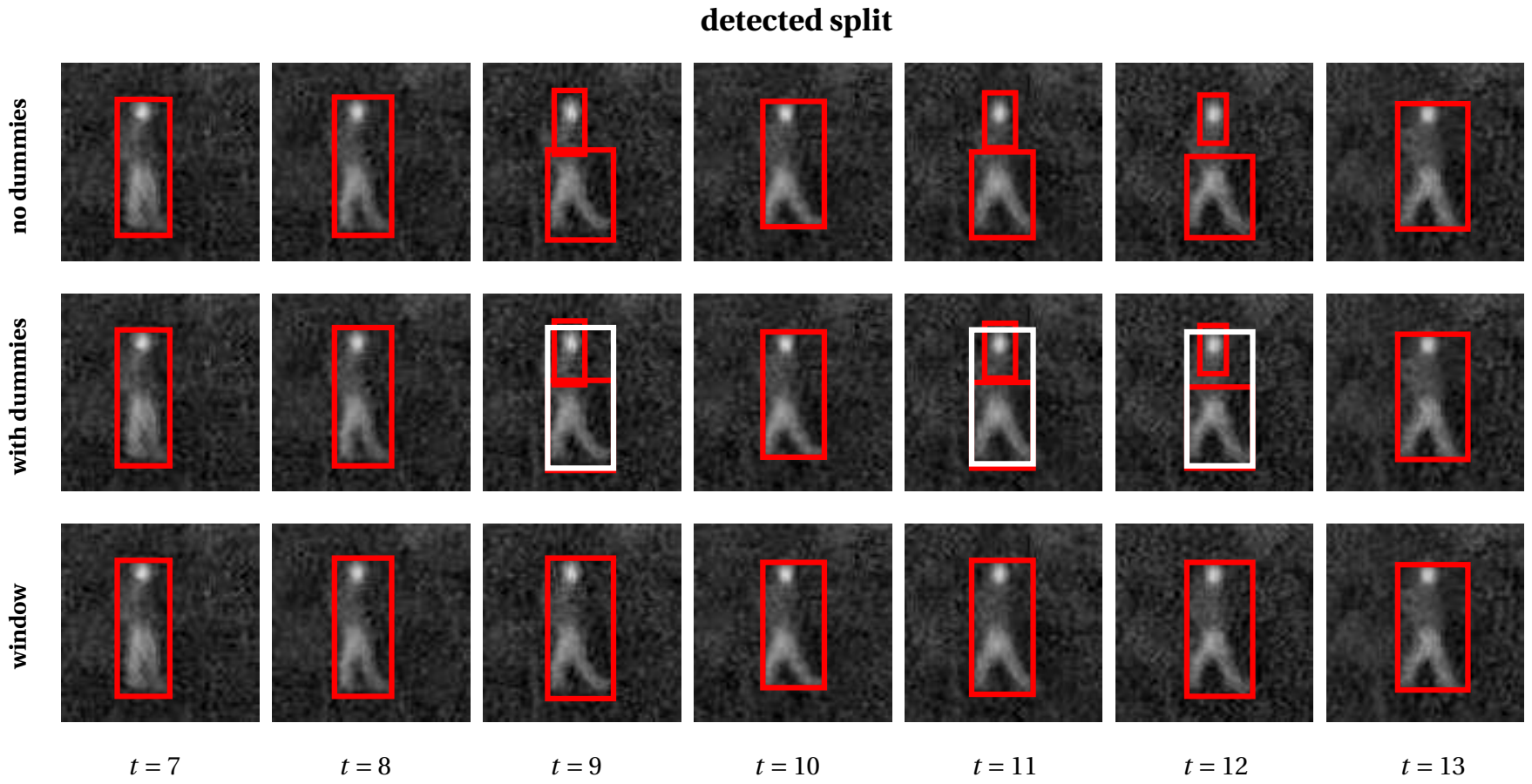


Figure 7.9: *Comparison of different methods we can use to track a split over time. The implementation of only the locality and parent constraint is shown in the top row, the addition of dummy blobs (in white) is shown in the middle row and using our window tracking scheme is shown in the bottom row. The window tracking only displays its results at the current time step (at the beginning of the window).*

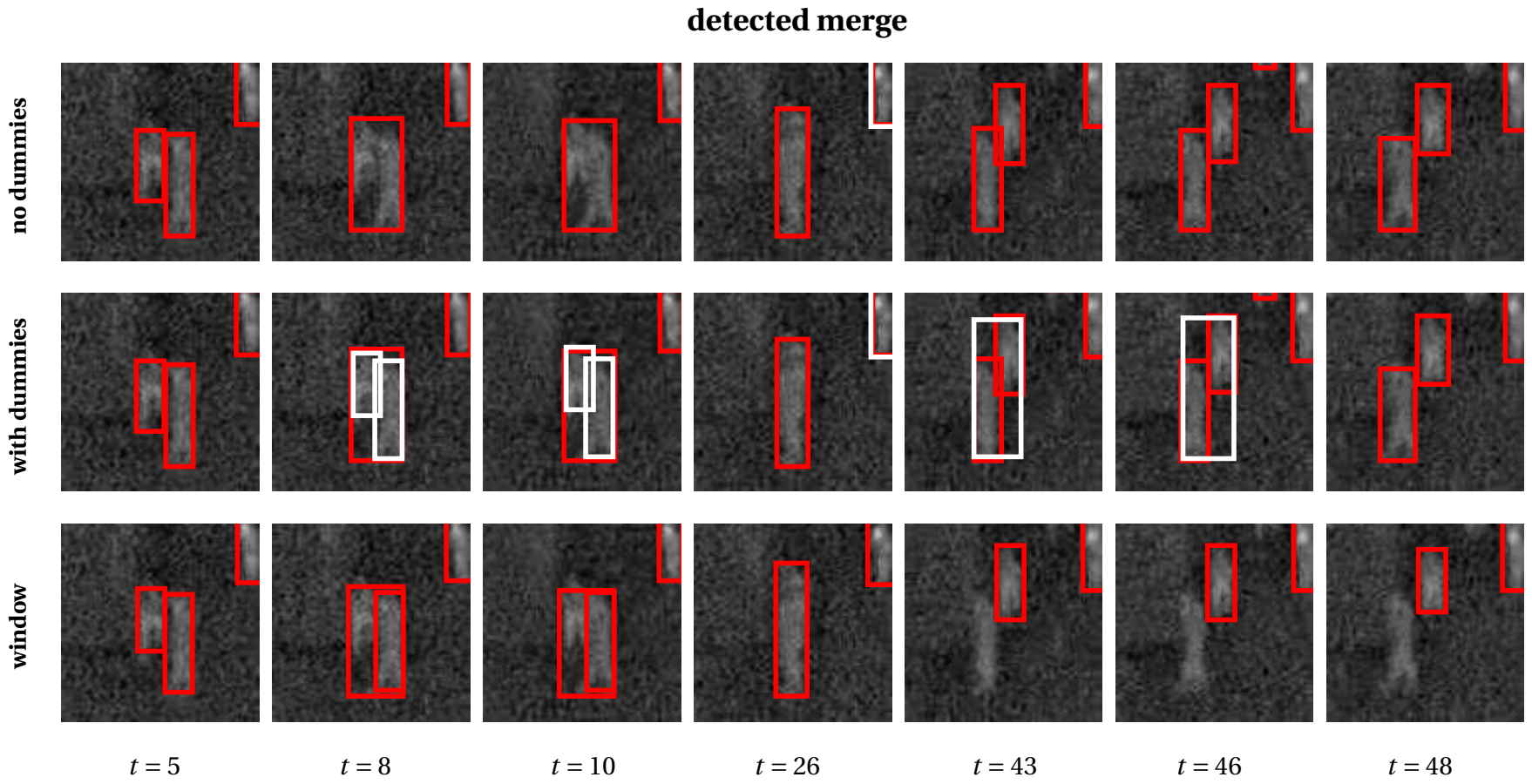


Figure 7.10: *Comparison of different methods we can use to track a merge over time. The implementation of only the locality and parent constraint is shown in the top row, the addition of dummy blobs (in white) is shown in the middle row and using our window tracking scheme is shown in the bottom row. We only display the chosen correspondences in the window tracking scheme, at the beginning of the window, although all the detections, as seen in the middle row, are still present.*

Chapter 8

Conclusions and future work

The main focus of this thesis was to track multiple objects over time through a video sequence. For this we use thermal and colour data, and argued that both have useful but distinct qualities. The thermal data is used for detection and initial correspondence, while the colour data is used for calculating object features to aid the final decision on correspondence. We focussed on handling detection errors, namely splits and merges, and proposed an alternative to the traditional Markovian tracking approach that considers only the previous time step and the current one.

In this chapter we make some concluding remarks and discuss some ideas for future work.

8.1 Conclusions

We have discussed the traditional approach to object detection and tracking in Chapter 1. This included a review of different tracking algorithms used in practice (Section 1.6) as well as an overview of the essential components needed for tracking objects over time: object detection, correspondences and implementing a tracking filter (Sections 1.2-1.4).

After that we discussed and supported our approach of using both thermal and colour data in building our tracking system and showed that certain potential errors, namely splits, merges and deaths, can appear when relating detected blobs at the current time step to those of the previous time step. We demonstrated the usefulness and potential of including dummy nodes to counteract these issues, and forcing what we consider might be the correct solution into consideration, in Section 6.1.

Finally we introduced our window tracking scheme as an alternative to the traditional Markovian tracking approach. It entails the calculation of a composite graph over a window of time steps. This graph incorporates probability measures as edge weights between corresponded blobs. We then explained the methodology involved in determining all possible tracks through this graph in Section 6.2, which can be compared by calculating a likelihood

score for each. The best scoring tracks are then used as measurement input for every object's tracking filter.

The concept of adding dummy nodes and rather considering a window of detections and correspondences originated from the fact that we did not want the system to make any hard decisions in an effort to alleviate the above mentioned issues. We decided to rather add additional information, in the form of these dummy nodes, into the graph, not knowing if the additions are in fact correct or not. This allowed the system to decide which configuration over the entire window is most probable.

It is difficult to quantitatively assess the accuracy of our developed system as there was no ground truth data available for the datasets we used. We can however interpret the final results qualitatively.

We have shown that our detection and tracking algorithms perform well, as they can handle detected splits and merges successfully. This was shown in Figures 7.5 and 7.6 where we implemented the basic tracker on two sequences. We also showed a detailed sequence of iterations involved with our proposed window tracking scheme in Figures 7.7 and 7.8, and those results are promising.

Furthermore, we saw in Figure 7.9 that the window tracker handles the detection of splits quite well. We found that when the scene becomes more cluttered, which can cause complex erroneous detections and correspondences, the system begins to struggle. It still detects foreground blobs sufficiently well, but fails in terms of correct correspondences. This is especially clear when viewing the results in Figure 7.10, where the objects are far away from the camera and occupy a small number of pixels. A main cause for the incorrect correspondences could be an inaccurate mapping of the detected bounding box in the thermal frame to the colour frame. It could however also be due to poor quality data. Both of these causes would result in inaccurate features which could result in meaningless edge weights.

8.2 Future work

This thesis stands as a proof of concept of our window tracking scheme and preliminary results are promising. There are however some areas on which we can improve.

Firstly, we found that there are inconsistencies with regards to the calculated correlations, especially with cluttered scenes and low quality data. This could be improved with the implementation of a more accurate method to map points from thermal data to the colour data, as this would result in more accurate and tangible features. This could also be rectified with a better sensor system, with higher quality sensors that can be calibrated. We also found that our system struggles with cluttered scenes, which can be due to many detections, or due to noisy data resulting in multiple segmented objects in close proximity.

Future work may include a more in-depth investigation into how all the valid tracks through a composite graph are determined, and the possibility of including some heuristics to speed up the process and make it more robust.

We may also consider the use of other object features, and other probabilistic measures to compare them. In this thesis we used fairly basic colour and edge features to describe every detected blob, and a simple linear correlation to measure similarity between feature vectors. These can be replaced with more sophisticated techniques, which may improve the performance of the system.

Finally, we hypothesize that the implementation of a global classifier for each detected blob could result in not only a more accurate tracker, but also result in an algorithm that can track objects uniquely. This would imply that if a person were to leave the scene and return later, or even the next day, that the system could be able to classify that as the same person. This has numerous applications in data mining, such as behaviour analysis [18, 34] and people counting [45, 55].

List of References

[1] H. Aghajan and A. Cavallaro. *Multi-camera Networks: Principles and Applications*. Academic Press, 2009.

[2] A. Aitken. *Statistical Mathematics*, volume 2. Oliver and Boyd Edinburgh, 1957.

[3] M. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2):174–188, 2002.

[4] F. Bergholm. Edge focusing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(6):726–741, 1987.

[5] J. Bougeut. Camera calibration toolbox for Matlab, `http://www.vision.caltech.edu/bouguetj/calib/_doc/`.

[6] R. Brown and P. Hwang. *Introduction to Random Signals and Applied Kalman Filtering*. John Wiley & Sons, 1992.

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

[8] C. Conaire, N. O'Connor, and A. Smeaton. Thermo-visual feature fusion for object tracking using multiple spatiogram trackers. *Machine Vision and Applications*, 19(5-6):483–494, 2008.

[9] R. Cucchiara, C. Grana, M. Piccardi, A. Prati, and S. Sirotti. Improving shadow suppression in moving object detection with HSV color information. In *IEEE International Conference on Intelligent Transportation Systems*, pages 334–339, 2001.

[10] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pages 886–893, 2005.

[11] Y. Dedeoglu. *Moving object detection, tracking and classification for smart video surveillance*. PhD thesis, Bilkent University, 2004.

[12] R. Deming, J. Schindler, and L. Perlovsky. Multi-target/multi-sensor tracking using only range and doppler measurements. *IEEE Transactions on Aerospace and Electronic Systems*, 45(2):593–611, 2009.

[13] R. Deriche and O. Faugeras. Tracking line segments. *Image and Vision Computing*, 8(4):261–270, 1990.

[14] C. Dietrich. *Uncertainty, Calibration and Probability: the Statistics of Scientific and Industrial Measurement*. CRC Press, 1991.

[15] A. Hampapur, L. Brown, J. Connell, A. Ekin, N. Haas, M. Lu, H. Merkl, and S. Pankanti. Smart video surveillance: exploring the concept of multiscale spatiotemporal tracking. *IEEE Transactions on Signal Processing*, 22(2):38–51, 2005.

[16] C. Harris and M. Stephens. A combined corner and edge detector. In *Alvey Vision Conference*, volume 15, page 50, 1988.

[17] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*, volume 2. Cambridge University Press, 2000.

[18] W. Hu, T. Tan, L. Wang, and S. Maybank. A survey on visual surveillance of object motion and behaviors. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 34(3):334–352, 2004.

[19] D. Huttenlocher, J. Noh, and W. Rucklidge. Tracking non-rigid objects in complex scenes. In *IEEE International Conference on Computer Vision*, pages 93–101, 1993.

[20] L. Iverson and S. Zucker. Logical/linear operators for image curves. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(10):982–996, 1995.

[21] N. Joubert. *Background modelling and subtraction for object detection in video*. Final year project, Stellenbosch University, 2009.

[22] P. Joubert and W. Brink. Scene reconstruction from uncontrolled motion using a low cost 3D sensor. *22nd Annual Symposium of the Pattern Recognition Association of South Africa*, pages 13–18, 2011.

[23] P. Kakumanu, S. Makrogiannis, and N. Bourbakis. A survey of skin-color modeling and detection methods. *Pattern Recognition*, 40(3):1106–1122, 2007.

[24] R. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35–45, 1960.

[25] T. Kanade, R. Collins, A. Lipton, P. Burt, and L. Wixson. Advances in cooperative multi-sensor video surveillance. In *Proceedings of DARPA Image Understanding Workshop*, volume 1, page 2, 1998.

[26] E. Krause. *Taxicab Geometry: An Adventure in Non-Euclidean Geometry*. Dover Publications, 1987.

[27] L. Lee, R. Romano, and G. Stein. Introduction to the special section on video surveillance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):745, 2000.

[28] D. Lowe. Fitting parameterized three-dimensional models to images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(5):441–450, 1991.

[29] D. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[30] P. Mahalanobis. On the generalized distance in statistics. *Proceedings of the National Institute of Sciences (Calcutta)*, 2:49–55, 1936.

[31] O. Masoud and N. Papanikolopoulos. A novel method for tracking and counting pedestrians in real-time using a single camera. *IEEE Transactions on Vehicular Technology*, 50(5):1267–1278, 2001.

[32] P. Maybeck, J. D'Azzo, et al. Stochastic models, estimating, and control. *New Directions in Effective Management*, 8:12–14, 1982.

[33] R. Mobus and U. Kolbe. Multi-target multi-object tracking, sensor fusion of radar and infrared. In *IEEE Intelligent Vehicles Symposium*, pages 732–737, 2004.

[34] T. Moeslund, A. Hilton, and V. Krüger. A survey of advances in vision-based human motion capture and analysis. *Computer Vision and Image Understanding*, 104(2):90–126, 2006.

[35] H. Moravec. Obstacle avoidance and navigation in the real world by a seeing robot rover. Technical report, DTIC Document, 1980.

[36] V. Nalwa and T. Binford. On detecting edges. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (6):699–714, 1986.

[37] R. Plackett. Karl Pearson and the chi-squared test. *International Statistical Review*, pages 59–72, 1983.

[38] E. Pretorius. *An adaptive feature-based tracking system*. MSc thesis, Stellenbosch University, 2008.

[39] B. Rao, H. Durrant-Whyte, and J. Sheen. A fully decentralized multi-sensor system for tracking and surveillance. *The International Journal of Robotics Research*, 12(1):20–44, 1993.

[40] J. Rodgers and W. Nicewander. Thirteen ways to look at the correlation coefficient. *The American Statistician*, 42(1):59–66, 1988.

[41] K. Romeo, P. Schwering, and M. Breuers. Multi-sensor track fusion. In *Proceedings of SPIE*, volume 4310, page 443, 2000.

[42] M. Rossi and A. Bozzoli. Tracking and counting moving people. In *IEEE International Conference in Image Processing*, volume 3, pages 212–216, 1994.

[43] G. Sfikas, C. Constantinopoulos, A. Likas, and N. Galatsanos. An analytic distance metric for Gaussian mixture models with application in image retrieval. In *Artificial Neural Networks: Formal Models and Their Applications*, pages 835–840. Springer, 2005.

[44] W. Shuan, A. Haizhou, and H. Kezhong. Difference image based multiple motion targets detection and tracking. *Journal of Image and Graphics*, 6, 1999.

[45] O. Sidla, Y. Lypetskyy, N. Brandle, and S. Seer. Pedestrian detection and tracking for counting applications in crowded situations. In *IEEE International Conference on Video and Signal Based Surveillance*, page 70, 2006.

[46] F. Singels. *Real-time stereo reconstruction using hierachical DP and LULU filtering*. MSc thesis, Stellenbosch University, 2010.

[47] J. Stewart. *Single Variable Calculus: Early Transcendentals*. Brooks/Cole Publishing Company, 2011.

[48] F. Suard, A. Rakotomamonjy, A. Bensrhair, and A. Broggi. Pedestrian detection using infrared images and histograms of oriented gradients. In *IEEE Intelligent Vehicles Symposium*, pages 206–212, 2006.

[49] R. Taylor. Interpretation of the correlation coefficient: a basic review. *Journal of Diagnostic Medical Sonography*, 6(1):35–39, 1990.

[50] O. Ukimura, editor. *Image Fusion*. InTech, 2011.

[51] R. van der Merwe, A. Doucet, N. de Freitas, and E. Wan. The unscented particle filter. In *Conference on Neural Information Processing Systems (NIPS)*, pages 584–590, 2000.

[52] G. Welch and G. Bishop. *An introduction to the Kalman filter*. Techniocal Report TR 95–041, University of North Carolina, 2006.

[53] N. Xiong and P. Svensson. Multi-sensor management for information fusion: issues and approaches. *Information Fusion*, 3(2):163–186, 2002.

[54] W. Yin, Y. Luo, and S. Li. Camera calibration based on OpenCV. *Computer Engineering and Design*, 1:063, 2007.

[55] X. Zhang and G. Sexton. A new method for pedestrian counting. In *Image Processing and Its Applications*, pages 208–212. Fifth International Institution of Engineering and Technology (IET) Conference, 1995.

[56] Q. Zhu, M. Yeh, K. Cheng, and S. Avidan. Fast human detection using a cascade of histograms of oriented gradients. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, pages 1491–1498, 2006.