Statistical models of pedestrian behaviour in the Forum

Barbara Majecka

s0675480

Master of Science

Artificial Intelligence

School of Informatics

University of Edinburgh

2009

Abstract

This dissertation describes an MSc project for which the purpose was to develop a system that could be used for automated surveillance. The main novelty is the use of a vertical camera. The project investigates whether such a system can effectively detect moving objects, track their trajectories, and use these to recognise anomalous events.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Barbara Majecka*
s0675480)

Table of Contents

Chapter 1

Introduction

1.1 Motivation

Visual surveillance has become widely used in both criminology as well as in sociology. In 2005 the number of CCTV cameras had reached four million in the UK alone [4]. These cameras can be used to gather sociological and marketing information by observing the behaviour of people and their interactions with each other and with their surroundings. However, the main use of surveillance systems is to provide a safer environment through detection or even prevention of dangerous situations and crimes. The former consist of suicide attempts, accidents, and situations in which help is required (e.g. if a person has collapsed); the latter could involve assaults, fights, thefts or burglaries. The effectiveness of such cameras has been confirmed by the Police Research Group [1] which reports, for example, that after installation of CCTV cameras in Newcastle, the number of burglaries fell by 56%. According to the same source, CCTV also contributed to the decrease ! in criminal damage (34%) and non-motor vehicle theft (11%) in this city.

Unfortunately, in order to ensure good coverage of the public spaces which should be monitored, the surveillance systems often need a large number of cameras, reaching even a hundred [7]. Due to the fact that one operator can watch at most four screens at a time [2], successful surveillance requires a lot of human resources. To make things worse, only a small fraction of video footage can actually be watched because operators tend to focus only on some cameras; they take necessary breaks; and they simply suffer from boredom which decreases their level of attention [7]. All this makes the manual surveillance systems vulnerable to errors and omissions. Therefore it is beneficial to improve visual surveillance by automating some of the tasks performed by operators.

In particular, abnormal or suspicious behaviour should be detected and the operators' attention directed to it.

The robustness of surveillance systems is extremely important - they should not ignore any anomalous events. At the same time, the number of false alerts should be kept to a minimum because flooding operators with numerous false positives would have similar detrimental effects to the case of no automation at all. These issues have provoked extensive research in this field which has resulted in a large number of different techniques for representing video footage, modeling behaviour, training detection systems and detecting anomalous events (e.g. [3, 5, 6, 8, 9, 12, 14, 16]). All of these techniques aim to improve the performance of the surveillance systems but they still have to address the same basic problems caused by the nature of the raw scene images. Due to the side view of the cameras, these problems include the possibility of occlusions (caused by parts of the background as well as other people) and merging silh! ouettes of tracked people [3]. The side view cameras are also unable to provide a full coverage of the monitored space. This often makes it necessary to use complicated integration techniques in order to construct a complete representation of behaviours of tracked people [8]. These algorithms are prone to errors.

The problems described above can be avoided, and therefore more reliable systems can be developed, by using top view cameras, provided that these cameras can cover the whole monitored area. This coverage could be achieved either by mounting the cameras sufficiently high or by using cameras with wide-angle lenses. Despite its advantages, this idea has not been investigated. So far the only systems that used perpendicular cameras were dedicated to counting people and were not appropriate for detection of abnormal behaviour. Adaptation of these systems would be very difficult because they either cover a very small, specific area of the scene (e.g. entries and exits) [10] or use techniques which are not capable of analyzing human behaviour [15].

Nevertheless, a system based on a top view camera has to face and solve a set of challenges. The most important one is the lack of detail on the detected people – most of the time only their heads and shoulders will be visible. Because the movements of limbs are barely visible, algorithms that depend on local flow descriptors or that try to analyze behaviour on the basis of the exact actions performed (e.g. running across a street) will not be usable [12]. This imposes restrictions on the techniques that can be used. Therefore, the project investigates whether it is possible to effectively detect abnormal behaviour using a perpendicular camera. An appropriate system was developed and its robustness evaluated.

1.2 System objectives

The project included design, implementation and evaluation of a system for detection of abnormal behavior in the Informatics Forum entrance area. It used a camera mounted near the ceiling pointing vertically downwards towards the ground floor.

The system involves low-level image processing, which produces information about detected objects. This data is then used by another component to infer trajectories of the moving objects. The gathered trajectory information is fed into the final component, which builds a model that can be used to detect abnormal behaviours.

The system was implemented to preserve robustness under different environmental conditions, such as changes in lighting levels and in the number of people tracked simultaneously.

1.3 The Forum

The Forum is one of the University of Edinburgh buildings located on the Central Campus. It consists of three parts, one of which is allocated to the School of Informatics. The camera was mounted in this part of the building, on the ceiling, overlooking the ground floor. An example image obtained from the camera is shown in Figure 1.1. The image covers most of the main hall. The most significant features of the hall are the main entrance to the building, lifts, access to the Atrium, access to the second part of the hall, staircase, reception desk, and the four other exits. The hall is mainly used to get into and out of the building. But it is attached to the Atrium, which is sometimes used for events, and people will often gather in the hall before and after such an event.

Figure 1.1: The view from the camera. The arrows show exits.

Chapter 2

Background

As mentioned above, automated surveillance has been a very popular research field in recent years. A vast range of techniques have been developed for different parts of surveillance systems. In particular, techniques used to represent video footage can be categorized as based on trajectories, local motion descriptors, or scene-wide motion patterns (classification according to [13]).

Techniques that fall into the last category represent global changes in a scene and are used in cases where accurate tracking of individuals is not feasible [16]. This makes them less interesting for this project because large crowds will not be expected and the perpendicular view of the camera will allow each person to be treated separately.

Incorporating techniques based on local motion descriptors can provide more precise information about the types of actions performed by a given target (e.g. running or fighting [5]). This information, when combined with spatio-temporal data [12], can give better results in the detection of abnormal behaviour than when simple trajectories are used. However, local motion is recognised on the basis of the optic flow measured in a fixed window around a target. Due to the perpendicular view of the camera, as well as the small size of the targets (about 20 pixels wide), obtaining accurate descriptors could be very problematic and therefore unreliable.

For this reason, techniques based on trajectories are the most appropriate for this project. Among them, the simplest are those that form a geometric representation of the raw trajectory data. In [14], the full trajectories are approximated by cubic spline curves with seven control points. In this way, each trajectory is represented by the same number of attributes – the control points and the duration of the object's existence in the scene.

A slightly different approach was proposed in [9]. Here, the data for each trajectory

is converted into a sequence of 4-dimensional flow vectors composed of 2D position coordinates of the tracked object and its instantaneous velocity at that position. After gathering a large set of flow vectors, a clustering algorithm based on competitive neural networks is applied to obtain a finite set of prototype vectors.

Given the time scale of this project, only one of these two techniques could be investigated. The geometric representation of trajectories was chosen.

Chapter 3

System overview

The project consisted of four subtasks:

1. detection of moving objects
2. tracking objects
3. modeling normal behaviours and training a classifier
4. recognition of anomalous behaviours

Each of these subtasks was implemented as a separate component of the system.

3.1 Detection

The efficiency of the detection application was crucial because it determined the maximum capture rate of video footage that could be used by the whole system. The greater this rate, the more accurate the tracking could be, allowing the detection of anomalous behaviour to be more successful. In order to allow the frequent capture of live images and to minimise the amount of data stored, the detection and tracking processes were carried out separately.

The detection focussed only on the extraction of the basic information, including bounding boxes and colour histograms of the detected objects. At each new frame, this information was used to update an appropriate file. Greater efficiency of the detection process was achieved by implementing it as a multithreaded C++ application.

3.2 Tracking

The files written out by the detection process are used as input for the tracker to infer trajectories of each object. The tracker needs to deal with different scenarios, including merging silhouettes of people. Also, it has to cope with imperfect detections. The trajectories constructed contain sequences of position coordinates of a particular object together with times at which the coordinates were sampled:

$$\{(x_1, y_1, time_1), (x_2, y_2, time_2), ..., (x_N, y_N, time_N)\}$$

The full sequences have different lengths N due to the different lifetimes of objects within the scene.

3.3 Modeling and Recognition

The trajectories are approximated by cubic spline curves [14] and represented by the vectors of their control points. These vectors are then clustered, and normality of a new trajectory is assessed on the basis of its Mahalanobis distance to the nearest cluster. If this distance is larger than an experimentally determined threshold, then the trajectory is flagged as anomalous.

Chapter 4

Detector

4.1 Introduction

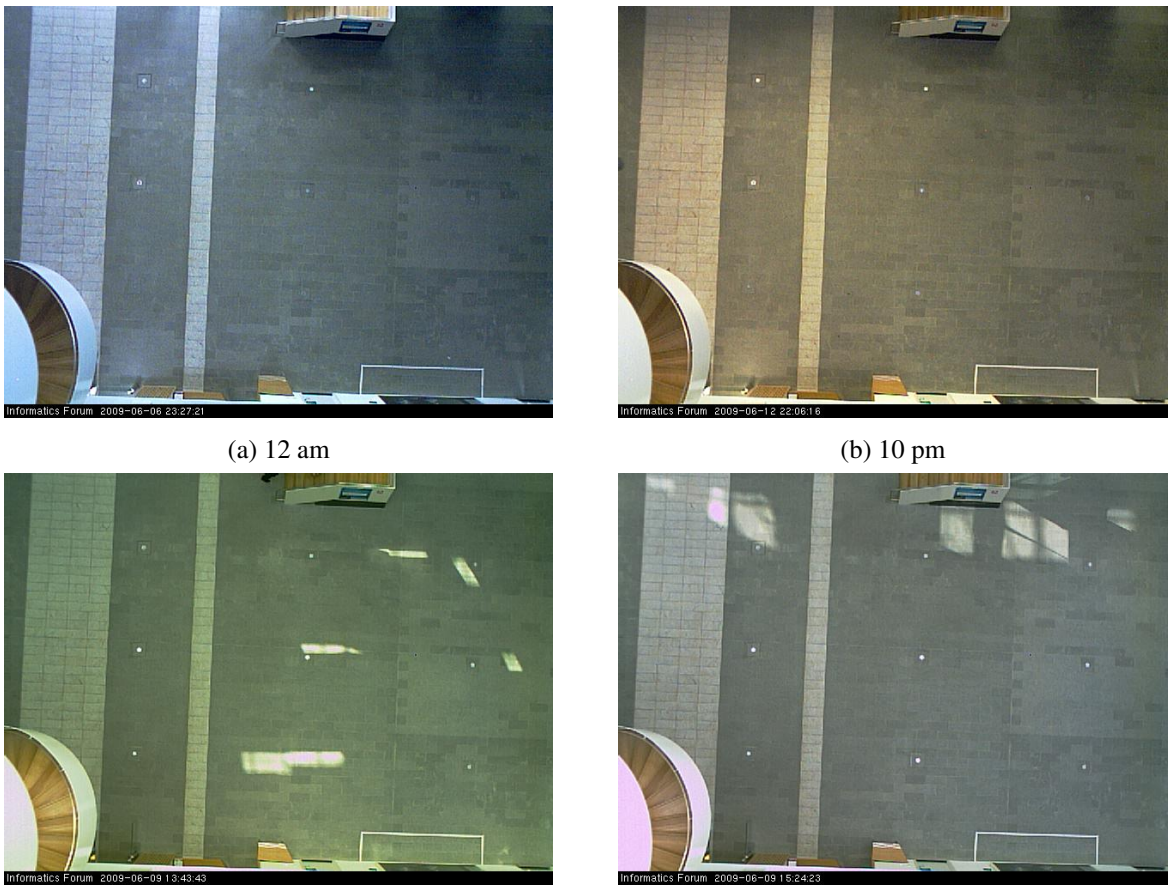
The purpose of the detector is to segment each image obtained from the camera into two sets of pixels: foreground and background. The foreground set should contain only those pixels which belong to tracked objects. Despite the fact that the camera is fixed in one position, the background is not completely constant throughout a day (Figure 4.1). The main changes to the background are caused by:

- reflections from the ground
- shadows
- changes in the ambient lighting

The reflections do not always appear in the same places and they strongly depend on the strength and position of the sun. Shadows in the top right corner are cast by the staircase and are more visible with artificial lighting. The overall brightness of the scene depends on many factors including weather, time of day, and use of artificial light sources and their positions. During the evening, an arbitrary combination of lights may be switched off, resulting in some parts of the scene being darker than others. All these factors make segmentation a non-trivial task, and I therefore experimented with a number of techniques in order to choose the most robust one.

4.2 Comparison of methods

The following techniques have been tested using a selection of four images (Figure 4.2) which represent examples of problematic situations: reflections, shadows, people



(a) 12 am (b) 10 pm

(c) 1 pm (d) 4 pm

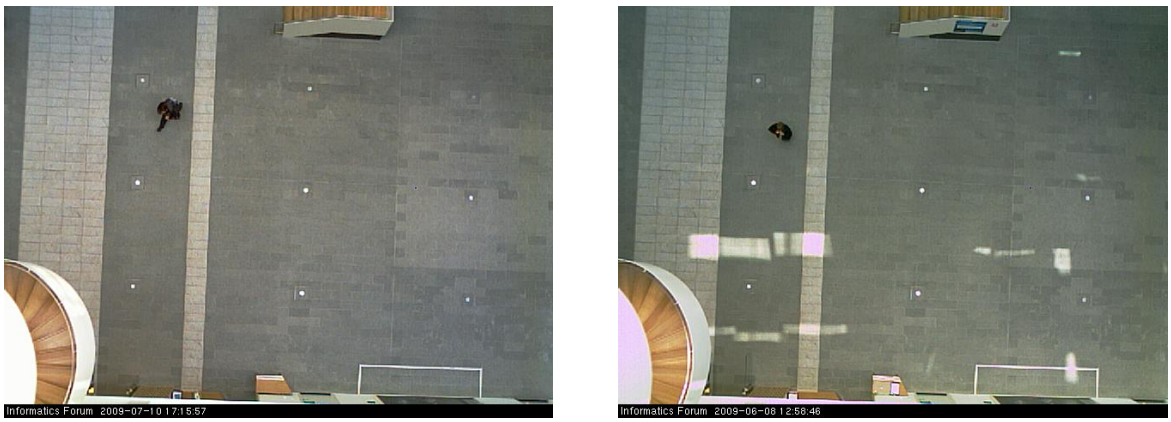
Figure 4.1: Four images of the background taken at different times.

wearing clothes in grey colours, artificial lighting and overall darkness.

4.2.1 Background subtraction

The first technique was a simple background subtraction and thresholding over all the colour channels (red, green and blue) to give three binary images. The overall segmentation was calculated by finding the pixel-wise conjunction of the three images. The background image (Figure 4.3a) was obtained by calculating the mean image from 50 background images obtained at different times of a day. The threshold was chosen dynamically by smoothing the histogram of pixel values after the subtraction and choosing the deepest valley between the highest peaks.

This technique performed very badly as shown in Figure 4.3b. Even when the threshold was chosen manually, the technique was not able to cope with either shadows or reflections, as shown in Figures 4.3c–4.3d.



(a) (b)

(c) (d)

Figure 4.2: Four test images with foreground objects and: reflections, grey clothing colours, shadows and difficult lighting conditions.

4.2.2 Constant background updating

The problem with the previous technique was the fact that the mean image of the background did not reflect the actual state of the background at a given time. The background image B_{RGB} can be updated as a weighted sum of itself and the most recently captured image I_{RGB}:

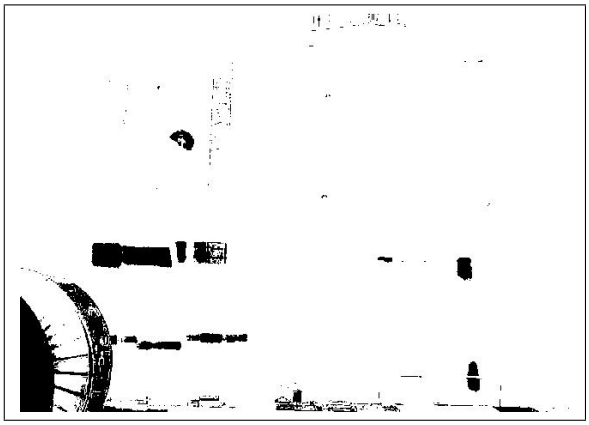
$$B_{RGB}(t) = (1-\alpha)\, B_{RGB}(t-1) \;+\; \alpha\, I_{RGB}(t) \tag{4.1}$$

This way, new objects which stay in the scene long enough become part of the background. This applies not only to physical objects like chairs or tables, but also to reflections on the ground and shadows. The technique also copes with changes in overall lighting conditions.

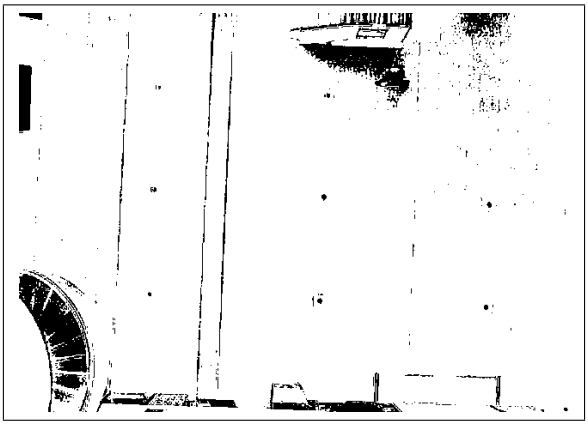
Unfortunately, this method has some problems too. In Edinburgh, natural lighting

(a) mean background

(b) normal subtraction of Fig.4.2c



(c) optimal subtraction of Fig.4.2b



(d) optimal subtraction of Fig.4.2d

Figure 4.3: Results of background subtraction using RGB coordinates. Optimal subtraction used a manually set threashold.

conditions can change very rapidly, and this can cause shadows and reflections to materialise quickly – within a second, for example. Therefore I initially set the α parameter to a high value: $\alpha = 0.02$. This dealt with changes very well but at the same time caused people to disappear if they stood in one position for only a couple of seconds (Figure 4.4). On the other hand, when the parameter value was very small, $\alpha = 0.002$, the reflections were classified as foreground and the algorithm was not able to cope with the lights being switched on or off.

The optimal parameter value could not be determined because it strongly depends on the time people should be expected to stand in one position. Because people tend to stop and chat on the ground floor I decided not to impose any limitation and therefore to disregard this technique.

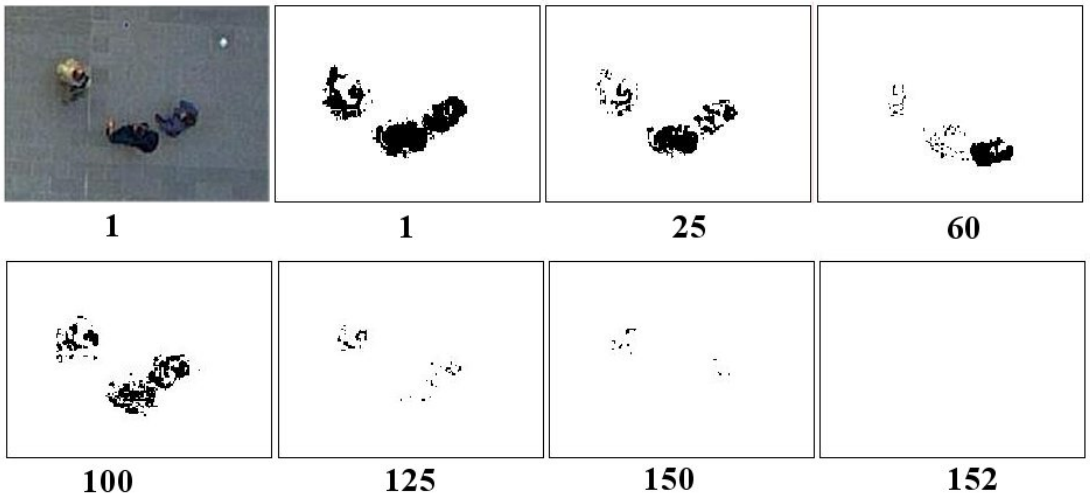


Figure 4.4: Results of background subtraction with constant background updating. Frame numbers are shown below each image. Approximately 10 frames were fetched per second. Here: $\alpha = 0.02$.

4.2.3 Background subtraction using chromaticity coordinates

This technique used chromaticity coordinates instead of RGB components. Each pixel was represented by two values calculated according to the following normalization formulae:

$$chrom_1 = \frac{Red}{Red + Green + Blue} \tag{4.2}$$

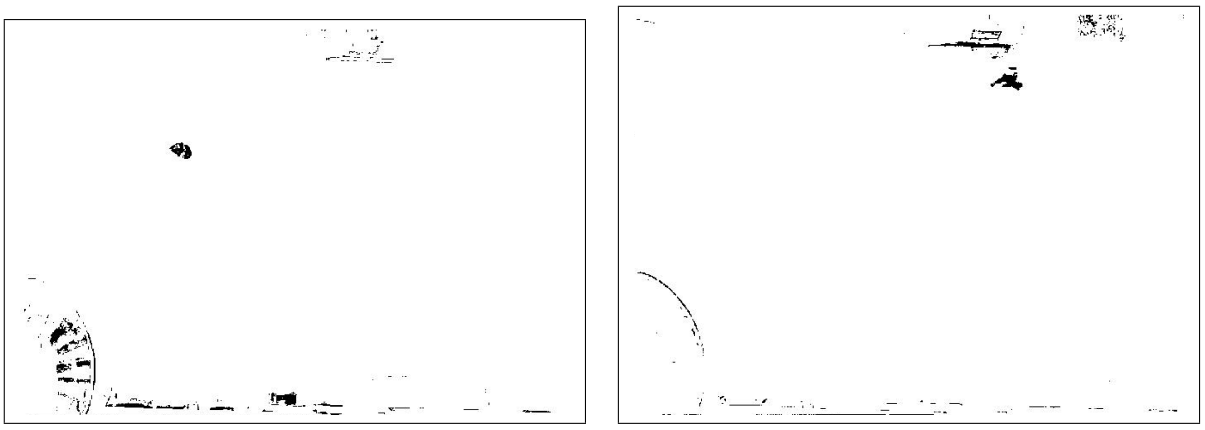
$$chrom_2 = \frac{Green}{Red + Green + Blue} \tag{4.3}$$

$$chrom_3 = \frac{Blue}{Red + Green + Blue} = 1 - chrom_1 - chrom_2 \tag{4.4}$$

The last equation shows that keeping all three values would be redundant. Chromaticity coordinates convey information about the pixel colour but not its intensity, therefore the same object in different lighting conditions should have pixels of similar chromaticity coordinates.

After representing a new image and the mean background image (Figure 4.3a) in chromaticity coordinates, computing the difference of these two, and then applying a manually chosen threshold, the result shows that reflections are no longer problematic (Figure 4.5a). Unfortunately, in the case of shadows, the result is far from satisfactory. This is particularly visible in the image taken during the night – there are visible detections in the top right corner, where the staircase casts a shadow (Figure 4.5b).

Moreover, the choice of threshold is very difficult, since dynamic thresholding is not possible (the histograms have only one distinctive peak), and a fixed threshold gives different results under different lighting conditions.

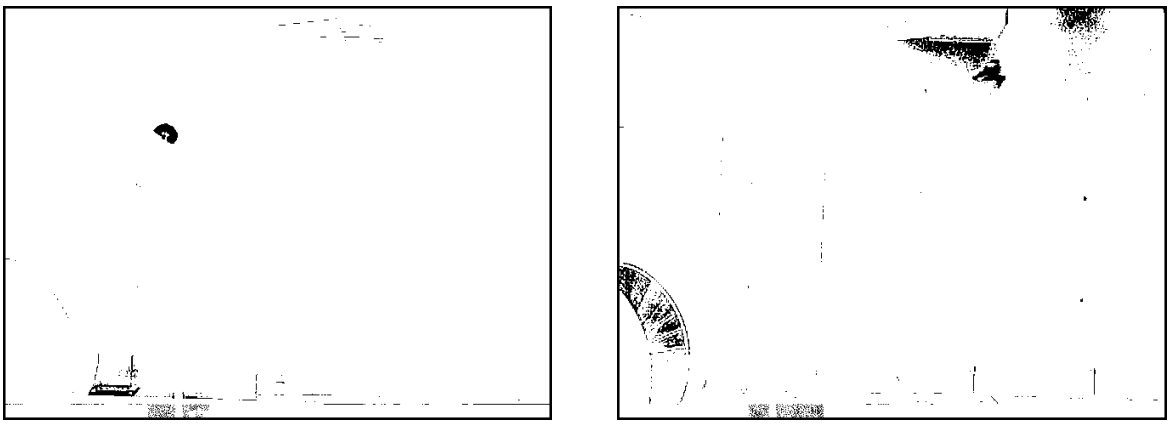


(a) optimal subtraction of Fig.4.2b (b) optimal subtraction of Fig.4.2d

Figure 4.5: Results of background subtraction using chromaticity coordinates. Optimal subtraction used a manually set threashold.

4.2.4 Background division using chromaticity coordinates

Similar results are obtained when a new image is divided by the background (Figure 4.6). The problem with the shadows still remains.



(a) optimal division of Fig.4.2b (b) optimal division of Fig.4.2d

Figure 4.6: Results of background division using chromaticity coordinates. Optimal division used a manually set threashold.

4.2.5 Principal Component Analysis

In order to take into account differences between the light distribution throughout the day I have built a model of the background image using the Principal Component Analysis ([13]). I gathered a set of 50 background images, some of which are presented in the Figure 4.1. I represented each image as a vector of length L equal to the number of pixels times number of chromaticity coordinates. From each vector, the mean vector was subtracted. The normalized vectors were then appended horizontally to each other to form an LxN matrix:

$$\mathbf{M} = [\vec{x}_1 - \vec{m},\ \vec{x}_2 - \vec{m}\ ,...,\ \vec{x}_N - \vec{m}] \tag{4.5}$$

where $\vec{m}$ is the mean vector and N is the number of background images (here: $N = 50$). In order to obtain the most significant principal components, the eigen-decomposition should be carried out on the covariance matrix $C = \mathbf{M}\mathbf{M}^T$ to obtain the eigenvectors ($\vec{v}_i$) together with their corresponding eigenvalues (λ_i):

$$\mathbf{C}\vec{v}_i = \lambda_i \vec{v}_i \tag{4.6}$$

However, the dimensions of the covariance matrix do not allow to do that in Matlab, therefore I used the technique suggested by [11]. This technique allowed to carry out the eigen-decomposition on the NxN matrix $\mathbf{M}^T\mathbf{M}$ giving eigenvalues:

$$\lambda'_i = \lambda_i \tag{4.7}$$

and eigenvectors:

$$\vec{u}_i = \mathbf{M}^T \vec{v}_i \tag{4.8}$$

This can be derived by multiplying both sides of the equation 4.6 by $\mathbf{M}^T$:

$$\mathbf{M}^T\mathbf{M}(\mathbf{M}^T\vec{v}_i) = \lambda_i(\mathbf{M}^T\vec{v}_i) \tag{4.9}$$

The required eigenvectors were then retrieved by using the following formula which ensures that eigenvectors have unit length:

$$v_i = \frac{\mathbf{M}\vec{u}_i}{|\mathbf{M}\vec{u}_i|} \tag{4.10}$$

The chosen K eigenvectors were appended to each other horizontally to form the matrix $\mathbf{V}$:

$$\mathbf{V} = [\vec{v}_1,\ \vec{v}_2, ...,\ \vec{v}_K] \tag{4.11}$$

which, together with the mean image vector $\vec{m}$ formed the model of the background.

The segmentation of a new image was then possible by projecting this image into the space determined by the matrix $\mathbf{V}$ and then reprojecting it into the original image space:

$$\overrightarrow{rep} = \mathbf{V}\mathbf{V}^T(\overrightarrow{image} - \vec{m}) \tag{4.12}$$

A binary image was then created by comparing the two representations of the image against a specified threshold. This way, parts of the image which could not be explained by the model, were classified as foreground:

$$(\overrightarrow{image} - \vec{m}) - \overrightarrow{rep} > threshold \tag{4.13}$$

This technique required to choose two parameters: the number of eigenvectors used in the model and the threshold. The former could not be too large because this would significantly increase the computation time of the whole detection algorithm. At the same time, the most significant eigenvectors show the directions of the greatest variations, so by choosing a number of eigenvectors, we can ensure that different directions of variability can be represented by the model. The Figure 4.7 shows the distribution of significance among the 50 principal components. The effect of the first three the most significant components is presented on the Figure 4.8. The first one changes the colours of the whole image and represents the changes which take place when the ambient light conditions change (Figure 4.8b). The second and the third one deal mainly with the shadow in the top right corner (Figure 4.8c - 4.8d).

For optimal results, I experimented with different numbers of eigenvectors K. The results are shown in the Figure 4.10. Here the thresholds were chosen to be $th_1 = 0.14$ and $th_2 = 0.2$ for the first and the second chromaticity coordinate accordingly. Similarily to the basic background subtraction, the binary image was composed as the disjunction of both binary images for the two chromaticity coordinates separately. An additional mask was applied to mask away the staircase (Figure 4.9). The thresholds were chosen experimentally. These experiments led to the conclusion that just one (the most significant) principal component is sufficient for a robust segmentation process. This decision saved a lot of computation time.

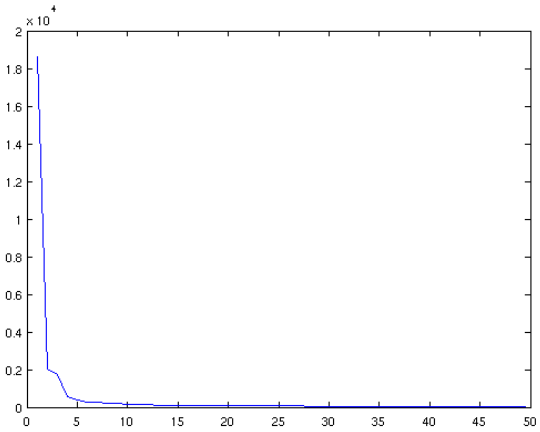
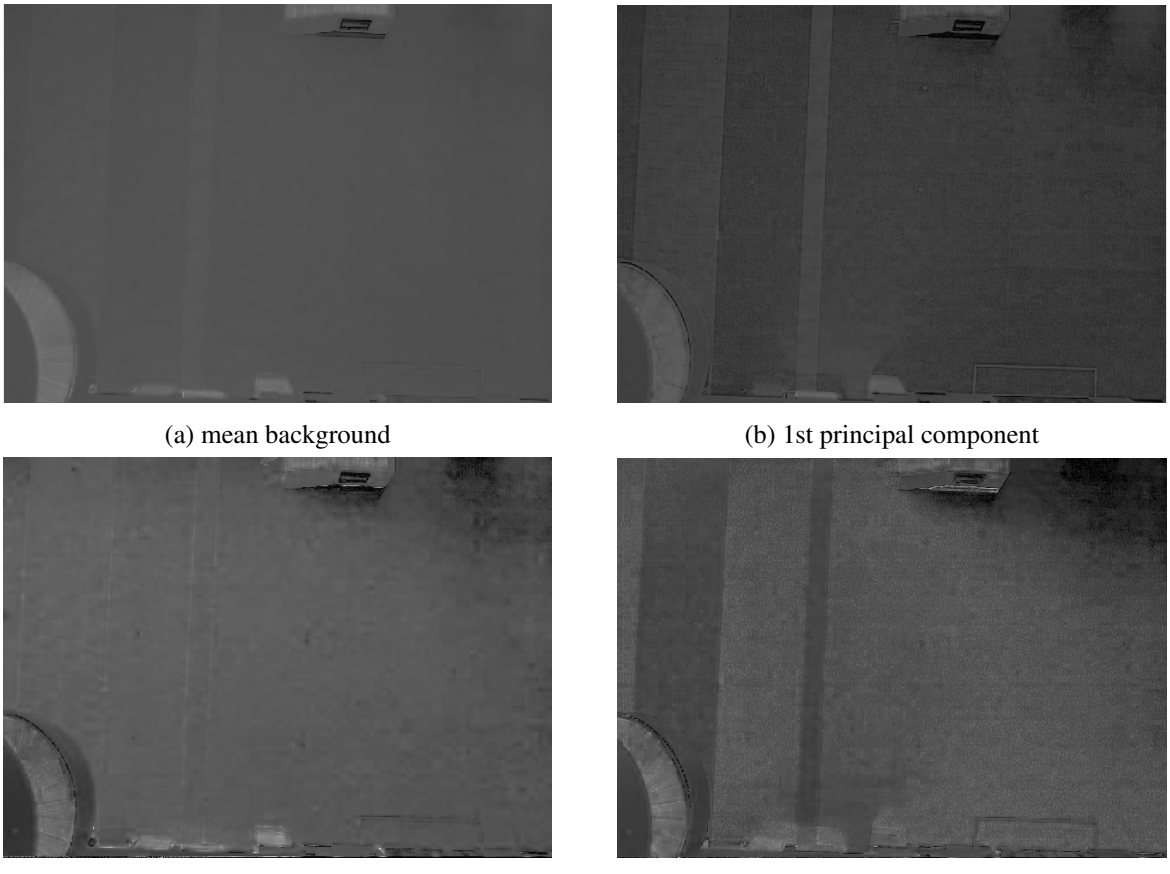


Figure 4.7: The eigenvalues which show the significance of 50 principal components.



(a) mean background

(b) 1st principal component

(c) 2nd principal component

(d) 3rd principal component

Figure 4.8: Result of varying the mean image in the direction specified by principal components. All the images are represented by the first chromaticity coordinate.

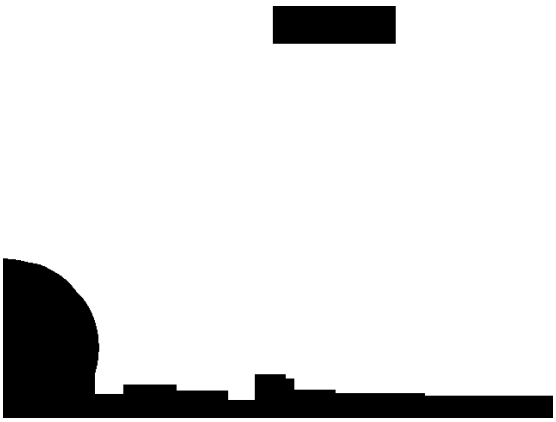


Figure 4.9: The binary mask used to avoid classification of the staircases as foreground.

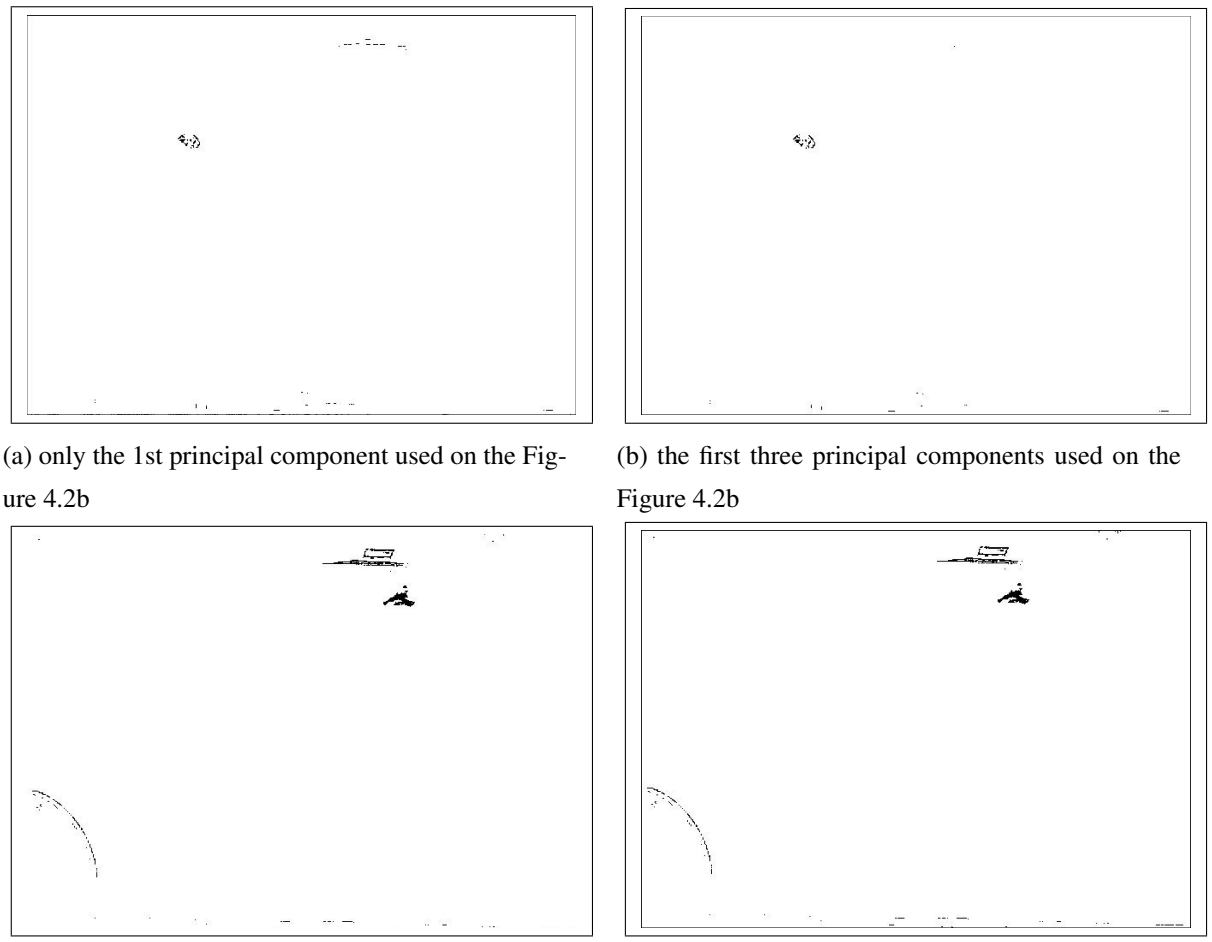
4.3 Implementation

The main steps in the detector's algorithm are:

1. Fetch a new image.
2. Obtain a binary image:
   (a) Segment using the background model.
   (b) Dilate and erode.
3. Label the image.
4. Append frame information to the output file.

4.3.1 Fetching a new image

In order to achieve maximum efficiency in the detector component, I implemented it in C++, and used a separate thread for fetching and processing images. The detector uses the libcURL library to fetch images directly into memory without first storing and then loading them from disk. This allows the fetching of images with a speed of nine frames per second. The ImageMagick library is used to decompress jpeg images into bitmaps, which are then preprocessed – the channels are separated – so that they are ready to be passed to the CImg library. CImg allows easy manipulation of images, but in the final version of the program this library was used only to contain the array of pixels: CImg methods are quite general, and I was able to make efficiency gains by writing code more specific to the needs of this project.



(a) only the 1st principal component used on the Figure 4.2b

(b) the first three principal components used on the Figure 4.2b

(c) only the 1st principal component used on the Figure 4.2d

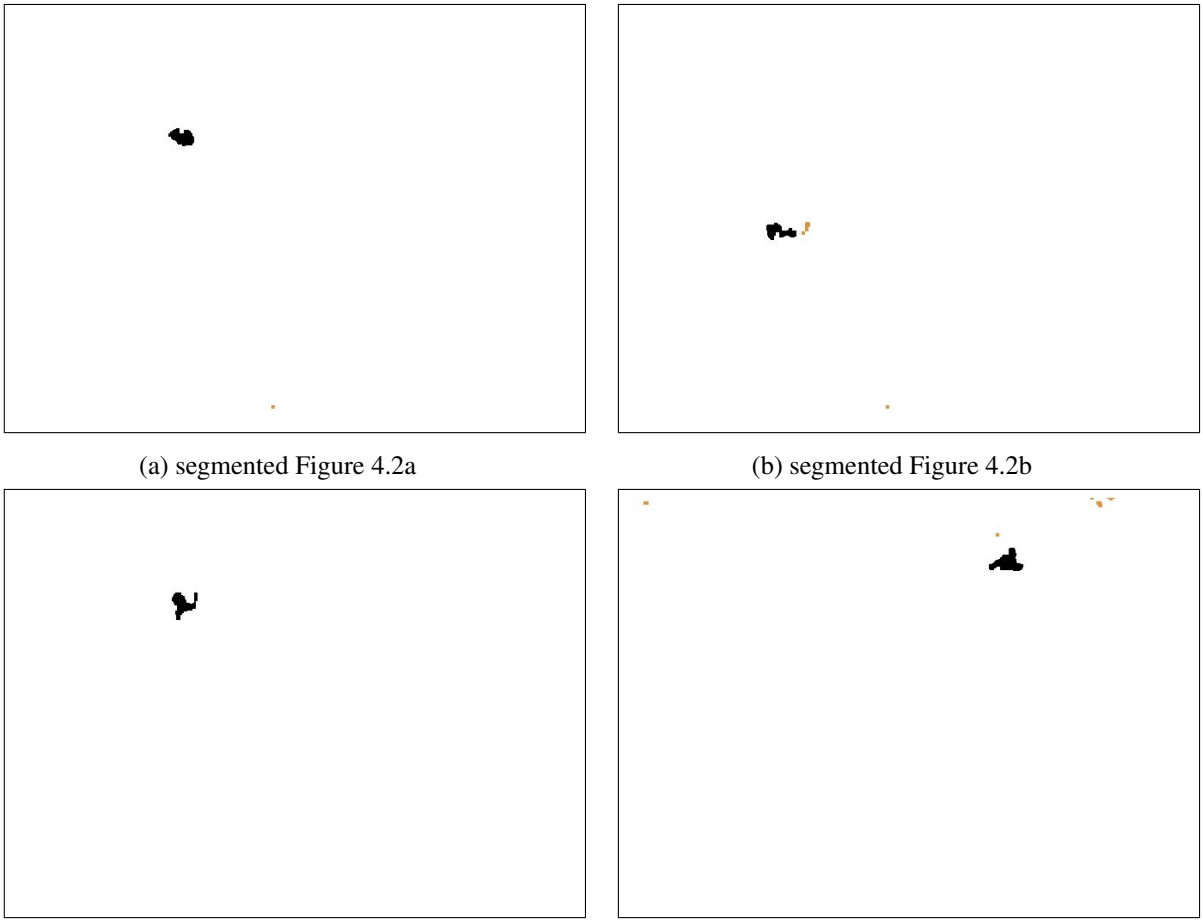
(d) the first three principal components used on the Figure 4.2d

Figure 4.10: The result of using different number of principal components during segmentation process.

4.3.2 Obtaining a binary image

The background model was built using Matlab and loaded into the C++ program using the matio library. The binary image produced after the segmentation process is then further cleaned up. For this purpose I implemented the morphological algorithms of dilation and erosion. These algorithms are ten times more efficient than those provided by the CImg library. I experimented with different combinations of erosion and dilation to achieve the best balance between removing all the noise from the image and merging blobs belonging to the same object. In the final version of the program I do not erode first because some foreground objects are detected as many separate pixels. I therefore perform dilation first which attempts to merge all the disjoint blobs which belong to the same object. If I had eroded first then objects which were detected as many loose

pixels could decrease in size or disappear altogether. The whole cleaning algorithm includes: a double dilation with 3x3 window; a single dilation with 2x2 window; and an erosion with 3x3 window. Example cleaned images are shown in Figure 4.11. This algorithm sometimes dilates noise pixels, leaving them in the resulting binary image. Blobs that do not exceed 75 pixels in size are deleted.



(a) segmented Figure 4.2a

(b) segmented Figure 4.2b

(c) segmented Figure 4.2c

(d) segmented Figure 4.2d

Figure 4.11: Binary images produced by the segmentation process. Blobs which are smaller than 75 pixels are shown in brown - they are not written out to files.

4.3.3 Labelling an image

A pixel belongs to a blob if it connects to the blob from one of eight directions. All the pixels which belong to the same blob are assigned the same value – the blob's identifier. This labelling process is implemented as a recursive algorithm. While the whole binary image is scanned, when an unlabelled foreground pixel is found, a recurvise subroutine

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

Figure 4.12: People used in the examples of colour histogram analysis.

is called which labels the pixel and calls itself for all the neighbours. The subroutines return if the current pixel either has already been labelled or belongs to the background.

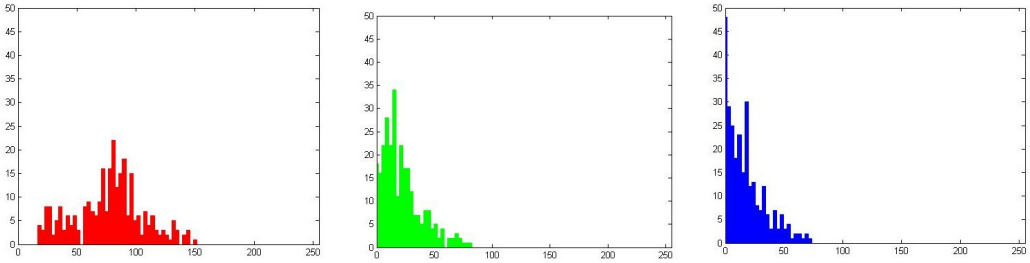
During the labelling process, these properties of the blobs are updated:

1. size
2. centroid position
3. bounding box:
   (a) top left corner position
   (b) width
   (c) height
4. colour histogram

4.3.3.1 Histograms

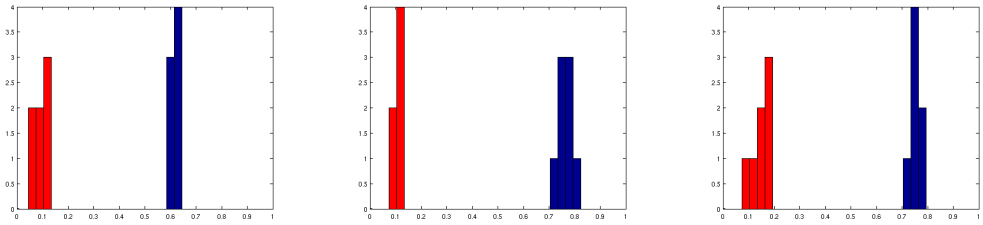
In order to produce the colour histograms, each colour channel (red, green and blue) is divided into four bins. This way the whole histogram is represented using $4*4*4=64$ bins. However, most pixels of the detected blobs fell into the lowest (the darkest) bins. Observations have shown that most people wear clothes which, from above and from a distance of 23 metres, appear to be very dark. Also, the detector copes particularly well with dark parts of foreground objects but sometimes omits their lighter parts (e.g. white trousers). Figure 4.13 shows distribution of pixel intensity values for one of the brighest people shown in Figure 4.12b. It shows that even the red component does not take the highest values and the whole mass of the histogram lies in the lower region.

Therefore I decided to divide the channels into uneven bins. In order to determine exact values for the bin sizes, I carried out a series of experiments. In each experiment, I obtained nine images of two people present in the same sequence of frames – person A and person B. For each of these images, I created a pair of colour histograms, one for each person. Then, for each histogram of person A, I calculated its Bhattacharrya distance to the histogram from the previous frame. (This distance was not computed



(a) red (b) green (c) blue

Figure 4.13: Histograms of the red, green and blue components of the person in Figure 4.12b.

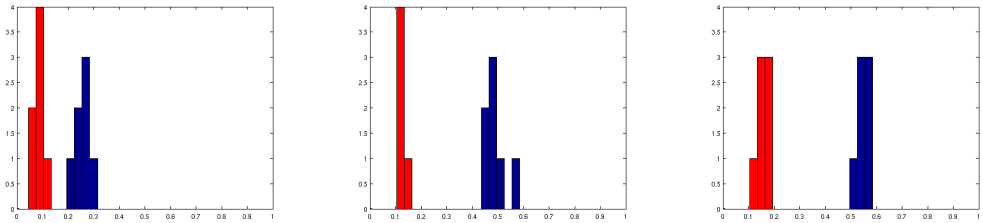


(a) bin sizes: 64, 64, 64, 64 (b) bin sizes: 20, 30, 40, 166 (c) bins: 10, 20, 30, 196

Figure 4.14: Comparison of Bhattacharrya distances between histograms of the same and different people. The former is shown in red and the latter in blue color. Here the comparison was carried out between people in Figure 4.12b and Figure 4.12c.

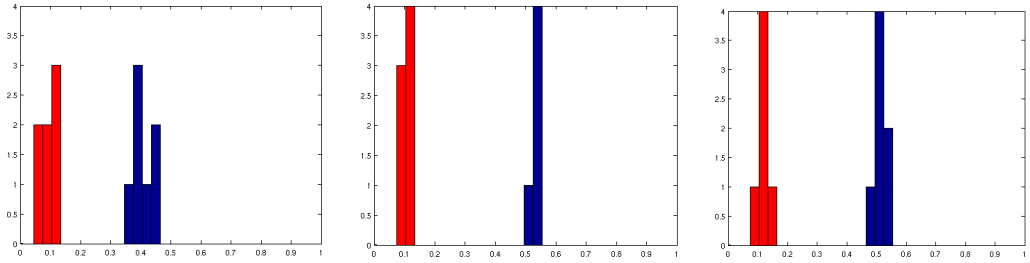
for the histogram from the first frame). Additionally, for each histogram of person B, I calculated its Bhattacharrya distance to the histogram of person A from the previous frame. To make it possible to distinguish between two people, the first group of distances should have small values (comparison of person A to itself) and the second group should have large values (comparison of person A to person B). I experimented with different pairs of people and different sizes of the bins. Some of my experiments are shown in Figures 4.14–4.15.

On the basis of the results obtained, I decided to assign the following sizes to the bins: 20, 30, 60 and 166. The bins for lower pixel intensity values should be much smaller because, as discussed above, dark blobs are much more common than brighter ones. At the same time, it was important to strike a balance between adapting the bins towards darker people and still being able to distinguish between the brighter ones.



(a) bin sizes: 64, 64, 64, 64 (b) bin sizes: 20, 30, 40, 166 (c) bins: 10, 20, 30, 196

Figure 4.15: Comparison of Bhattacharrya distances between histograms of the same and different people. The former is shown in red and the latter in blue color. Here the comparison was carried out between people in Figure 4.12d and Figure 4.12c.



(a) bin sizes: 64, 64, 64, 64 (b) bin sizes: 20, 30, 40, 166 (c) bins: 10, 20, 30, 196

Figure 4.16: Comparison of Bhattacharrya distances between histograms of the same and different people. The former is shown in red and the latter in blue color. Here the comparison was carried out between people in Figure 4.12b and Figure 4.12a.

Figure 4.17: Background frame fetched during the night when the lights were switched off.

4.3.4 Writing out frame information

The detector component produces files containing information about the detected blobs. These files are then used by the tracker component. To provide the latter with as much information as possible, I print out all the properties listed above. All data concerning blobs detected in the same frame are grouped together and preceeded by information about this frame: the frame number and time since the program was started. This information is printed only for those frames within which there were detections. Moreover, only those blobs which have size above 75 pixels are printed, which serves to eliminate noise.

4.3.5 Shell Script

A shell script was used to start the detector every day at 7am and stop it at 11pm. The detector did not run during nights because one eigenvector would not be sufficient for frames like the one presented in the Figure 4.17. The script also ensured that the program continued to run during the day: if it found that the program was not running, it would attempt to start it and report this event by email. This was necessary because the detector sometimes crashed due to a bug in the ImageMagick library.

4.4 Testing

Figure 4.11 shows the results obtained by the detection algorithm running on the images shown in Figure 4.2. Only the black blobs are written out to the file - the brown

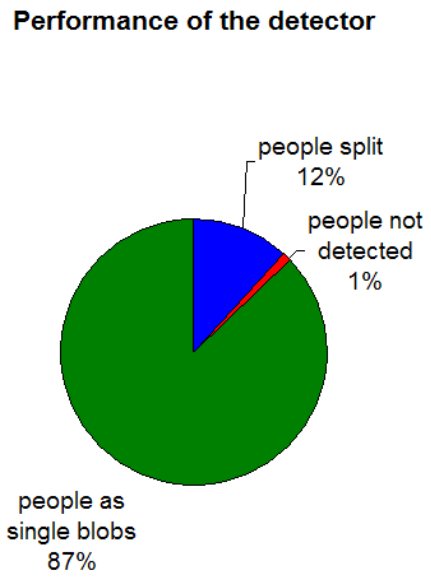


Figure 4.18: Probability that a person will be detected as one, many or none blobs.

blobs do not exceed 75 pixels in size.

I gathered performance statistics on the basis of 52 trajectories. Each person was observed for an average of 56 frames. The images were fetched during different lighting conditions and people wore clothes of different colours.

Of the 52 trajectories, only two contained frames where detection failed (no blobs were detected). In these two cases, the detection failed in 12% and 50% of all the frames in which the tracked person appeared. Both of these people wore grey and white clothing which is similar to the background.

There were no false positive detections in any of the 2916 test frames. In 30 of the 52 trajectories, people were always detected as single blobs. In the remaining 22 trajectories, people were detected as single blobs in 72% of frames.

I also carried out experiments to determine the conditions that cause blobs of separate people to merge. Among eight sets of trajectories of people walking closely together (878 frames), only two contained frames in which blobs were merged. Over these trajectories, 12% and 23% of blobs were merged, respectively. Merging was more common on the edges of frames, because there, people were seen from a wider angle and therefore occlusions could take place. The largest distance between the positions of two people which resulted in them merging was equal to 22 pixels (approximately 50cm) in the middle of the floor and 32 pixels (approximately 75cm) on the edge of the scene. However, the largest apparent distance between two people that were merged (as measured from the extrema of their bodies) was equal to 4 pixels

(approximately 9cm).

Chapter 5

Tracker

5.1 Specification

The files updated by the detector component are used as input to the tracking process to infer trajectories from detected objects. Each trajectory is represented by a sequence of centroid positions together with the time when the object was detected at this position. The time stamps are important because the positions are not uniformly spaced in time, and the speed with which the objects are moving is important for detection of abnormal behaviour. Also note that each object could have a trajectory of a different length due to the different lifetimes of objects within the scene.

The tracker component makes a distinction between the notions of a person and a blob:

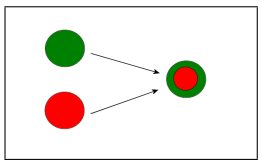
Person is a tracked real object that appeared in the scene.

Blob is an area of 8 point connectivity labelled as foreground by the detector.

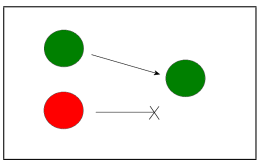
Each person has a unique identity. Deciding which person corresponds to each blob in a given frame is not a trivial task. There are several problems that the tracker has to overcome:

1. Several people could **merge** together when walking side by side, and therefore they could be represented by a single blob (Figure 5.1).

2. A group of people, represented in one frame as a single blob could **split** in the next frame producing multiple blobs (Figure 5.2).

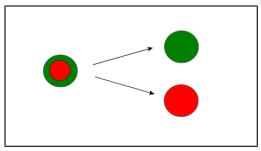


(a) Actual scenario: two people merge.

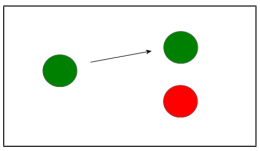


(b) Failed tracking: the red person is judged to have disappeared.

Figure 5.1: An example of two people merging. Circles represent blobs (which are inputs from the detector component). Colours represent people's identities. Arrows illustrate how peple move between frames.



(a) Actual scenario: two people, merged in the previous frame, now split into separate blobs.



(b) Failed tracking: the blob from the previous frame is assumed to represent only one person, while the red person in the current frame is judged to be new to the scene.

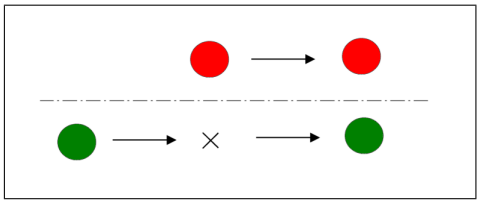
Figure 5.2: An example case of two people splitting.

3. For some people the detection process could fail completely causing the person to **disappear** for a few frames (Figure 5.3).

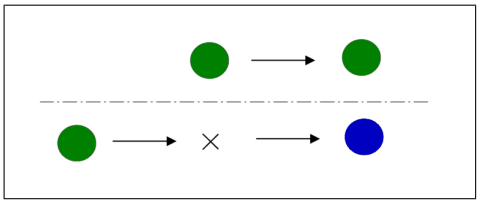
4. Some parts of a person's body may not be detected resulting in the person being represented by **several** disjoint **blobs** (Figure 5.4).

5.1.1 Merging and splitting

Due to the perpendicular positioning of the camera, the possibility of occlusions is minimised, but it is still present. As shown in the previous chapter, these situations are the most common on the edges of the scene. Therefore, it is possible that people could enter the scene merged (represented by a single blob) and then split into several blobs when they come closer to the centre of the floor. Such a situation is depicted in Figure 5.5. Figure 5.2b shows an undesirable tracking behaviour: the merged people are recognised as a single person, and a new blob that appears after the splitting is assigned a new identity with a trajectory starting in the middle of the floor. The tracker should not allow this; it should try to reason about where the new blob came from (Figure 5.2a).

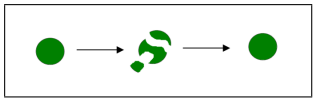


(a) Actual scenario.

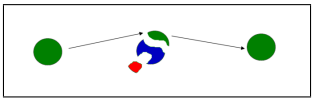


(b) Failed tracking: the blob that appears in the second frame is assigned the identity of the green person. The person who disappears in the second frame and reappears in the third is assigned a new identity.

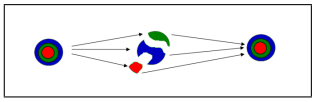
Figure 5.3: An example of a person disappearing and another person appearing at the same time. The dashed line represents a large distance between blobs.



(a) Actual scenario: a single person is detected as separate blobs in the second frame.



(b) Failed tracking: the second frame is assumed to contain three people, two of whom appeared only in that frame.



(c) Failed tracking: all three frames are assumed to contain three people. In the first and the third frames the people are merged together.

Figure 5.4: An example of one person being detected as several blobs.

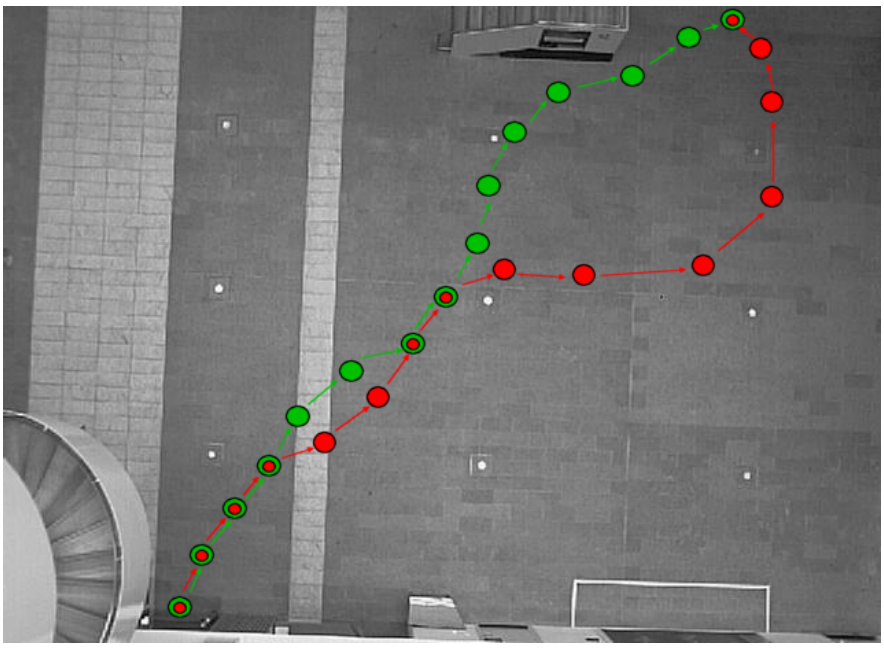


Figure 5.5: A simple scenario of people walking together. Two people enter the scene merged together. They split, merge, split again, walk far away from each other, and then merge again before leaving the scene.

Figure 5.5 shows also a situation in which people previously detected as separate blobs merge together to create a single blob. Their trajectories should be updated separately: new positions should be appended to both trajectories (Figure 5.1a), not just to one of them (Figure 5.1b). After they split again, their identities should be reassigned correctly (Figure 5.5).

5.1.2 Disappearing

The evaluation of the detector component showed that a person may not be detected in some of the frames in which the person was present. The tracker should be able to cope with this problem. In particular, it should not terminate a trajectory if the person has disappeared for just a few frames, and it should identify the person correctly on their reappearance. It should, however, terminate a trajectory if it decides that there is a high probability that the person has left the scene.

5.1.3 People as separate blobs

12% of people are represented by the detector as disjoint blobs. This causes several problems. If these blobs are recognised as separate people then the merging and split-

ting rules will be applied. This way, many redundant trajectories could be produced, and the component responsible for detecting abnormal behaviour could believe certain trajectories to be more common than they are. Only one among these trajectories should be preserved.

5.2 Design

The tracker keeps a list of all the people currently being tracked. It updates their trajectories one frame at a time by processing the file provided by the detector. Because one person can be represented by multiple blobs and one blob can represent multiple people, an M-to-N relationship has to be preserved in the program. Therefore, at every frame, a list of detected blobs is kept, and for each blob is stored a list of the people represented by it.

5.2.1 The core algorithm

The high level algorithm is illustrated by Pseudocode 1.

5.2.1.1 Step 1: Assigning people to blobs

The Pseudocode 2 illustrates the first step of the high level algorithm.

For each person still in the list of tracked people, the tracker tries to find the corresponding blobs in the current frame. It calculates the expected position of each person and the radius of the circle wherein the person could potentially be detected. Only those blobs that lie within this predicted area are taken into account. If there is more than one blob, their relative probabilities of representing the given person are calculated and the blob of the highest probability is chosen. The centroid position of the blob is then added to the trajectory of the person. However, if the predicted area does not contain any blobs, the person is assumed to disappear for this particular frame and their trajectory is not updated.

This algorithm allows for the assignment of multiple people to the same blob, thereby coping with merging scenarios. In this case, trajectories of all the merged people are extended by the centroid of the same blob. If, in fact, the people did not merge and the blob just happened to be the most probable for all of them, this mistake can still be corrected in the second step of the algorithm (subsection 4).

Pseudocode 1 The tracking algorithm (high level).

```
FOR EACH frame IN file
  blobs = frame.getBlobs()

  //-- step 1 --//
  FOR EACH person IN people
    updateCorrespondence(person, blobs)
  END

  //-- step 2 --//
  unassignedBlobs = blobs.getUnassignedBlobs()
  FOR EACH blob IN unassignedBlobs
    correctCorrespondence(blob, people)
  END

  //-- step 3 --//
  removeAndPrintPeopleWhoDisappeared()

  //-- step 4 --//
  updateHistograms()
END
```

Pseudocode 2 Step 1: Assigning people to blobs.

```
FOR EACH person IN people
   predictedPosition = person.getPredictedPosition()
   radius = person.getPredictedAreaRadius()

   //-- look for most probable blob --//
   FOR EACH blob IN blobs
      IF blob.isInArea(predictedPosition, radius)
         probabilities.add(blob, blob.getProbability(person))
      END
   END
   chosenBlob = getMostProbable(probabilities)

   //-- person -> blob --//
   IF (chosenBlob != NULL)
      assign(person, chosenBlob)
   END
END
```

5.2.1.1.1 Position prediction. Every time a new point is added to a trajectory, the person's instantaneous velocity **v** is calculated at that point. Its magnitude is estimated on the basis of the last two points in the trajectory:

$$|\mathbf{v}_n| = \frac{\left|\mathbf{pos}_n - \mathbf{pos}_{n-1}\right|}{time_n - time_{n-1}}$$

where n is the index of the point at which the velocity is calculated. The direction, on the other hand, is estimated using the last four points instead of just two:

$$\mathbf{v}_n = |\mathbf{v}_n| \frac{\mathbf{pos}_n - \mathbf{pos}_{n-3}}{|\mathbf{pos}_n - \mathbf{pos}_{n-3}|}$$

In this way, I minimise the error caused by the "jumping" centroid point. This error is especially noticeable when different parts of a person's body are detected at each frame causing their position, as described by the centre of mass of all their pixels, to change significantly. At the same time, by choosing a small number of previous points, I still allow for rapid changes in the trajectory. This is important for detection of behaviour abnormalities.

For the trajectory with just one point, the initial velocity is set to $\mathbf{0}$. If the trajectory has fewer points than four, the direction is determined using all the available points.

Having the instantaneous velocity of the last point in the trajectory, the expected position is estimated as follows:

$$\widehat{\mathbf{pos}}_n = \mathbf{pos}_{n-1} + \mathbf{v}_{n-1}\left(time_n - time_{n-1}\right)$$

The smaller is the time difference, the more accurate the prediction. After a longer time, the predictions could be inaccurate, and I therefore allow a person to disappear only for four frames (i.e. no more than 500ms).

5.2.1.1.2 Radius. The position prediction is not always accurate, therefore the prediction area should be large enough to allow for those errors, but at the same time small enough to avoid mistakes with other blobs. I chose the radius on the basis of the previous instantaneous velocity of the considered person (Pseudocode 3).

In order to estimate the optimal values of the parameters `MAX_SPEED`, `MIN_RADIUS` and `FACTOR`, I carried out a statistical analysis of the error between the predicted and the actual positions. The results are shown in subsection 5.2.1.1.3. I experimentally determined the maximum speed at which a person can walk: `MAX_SPEED` = 26 pixels per 100ms. Before the actual velocity of a person is known (i.e. when there is only one

Pseudocode 3 Choosing the radius.

```
IF ( trajectory.hasOnlyOnePoint() )
   radius = MAX_SPEED * time + DIAMETER
ELSE
   radius = velocity * time * FACTOR
   IF (radius < MIN_RADIUS)
      radius = MIN_RADIUS
   END
END
```

point in their trajectory), the maximum velocity is assumed. Additionally, the radius is increased by the diameter of a human blob due to the fact that the centroid can "jump". For simplification, `DIAMETER` is chosen to be 30 pixels. If the velocity of a person is known, the radius is chosen to be `FACTOR` = 6 times longer than the distance from the last position to the next predicted position. However, it should never be smaller than `MIN_RADIUS` = 30 pixels (the approximate diameter of a person). The last two values were chosen on the basis of the statistical analysis discussed in the subsection 5.2.1.1.3.

5.2.1.1.3 Correspondence probability. The Bhattacharrya distances from the previous chapter show that there is a clear distinction between the people based purely on their colour histograms. However, the detection quality of some people varies from frame to frame, e.g. some lighter parts of their clothing might not be detected in several frames. This causes their colour histograms to differ, which:

1. increases the Bhattacharrya distance between a person and the same person in the previous frame, and

2. raises the likelihood of confusing the person with another.

Also, some people might simply wear very similar clothes, which additionally increases the difficulty of distinguishing them. Therefore, I have avoided relying purely on the colour histograms $\mathbf{h}_{new}$, and have also used the error of position prediction err_{new} to estimate the probability of a blob representing a particular person P_i. Using Bayes' theorem:

$$p(P_i|\mathbf{h}_{new}, err_{new}) = \frac{p(\mathbf{h}_{new}, err_{new}|P_i)p(P_i)}{p(\mathbf{h}_{new}, err_{new})} \tag{5.1}$$

I assume that:

$$\forall i, j \quad p(P_i) = p(P_j) \tag{5.2}$$

$$\forall i, j \quad p(\mathbf{h}_i, err_i) = p(\mathbf{h}_j, err_j) \tag{5.3}$$

$$p(\mathbf{h}_{new}, err_{new}|P_i) = p(\mathbf{h}_{new}|P_i)p(err_{new}|P_i) \tag{5.4}$$

Prediction error. I define the error in position prediction as the distance between the predicted and the actual positions. I computed a histogram of error values for the position predictions of 25 people, which totalled 4059 samples. I chose the bins to have a width of one pixel. The numbers in each bin were divided by the number of samples; using this maximum likelihood approach I obtained a likelihood measure for each bin. I then tried to fit different continuous distributions to this discrete distribution. The results are shown in Figures 5.6a–5.6b. Since the error values are positive, the Gaussian distribution was fitted to the error distribution plus its reflection about zero. Then the values of the Gaussian were multiplied by two and shown only for positive error values. The multiplication was necessary to ensure that the distribution would still integrate to one after removing half of it. I computed the negative log likelihood for each fitted distribution. The exponential distribution proved to be the best fit:

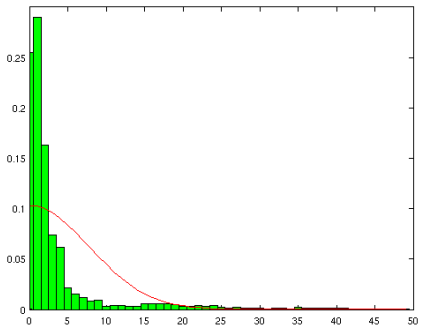
$$p(err_{new}|P_i) = \lambda e^{-\lambda err_{new}}$$

$$\lambda = 0.2762$$

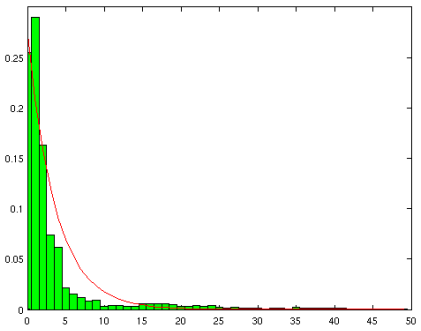
I also checked whether normalisation of the error with respect to the distance between the predicted and the actual position could improve the choice of distribution. Intuitively, the longer are the distances between detections, the greater are the errors in the predicted positions. However, as shown in Figure 5.7, the distribution looks more ragged and does not resemble well any standard distribution. This could be due to the fact that the error depends more on the "jumps" of the centroid (the detection of different fragments in each frame) rather than the distance.

Histograms. I approximate the probability distribution of the colour histograms for a particular person by the 64 dimensional Gaussian distribution:

$$p(\mathbf{h}_{new}) = \frac{1}{(2\pi)^{n/2}|\mathbf{K}_i|} e^{-\frac{1}{2}(\mathbf{h}_{new}-\overline{\mathbf{h}_i})^T \mathbf{K}_i^{-1} (\mathbf{h}_{new}-\overline{\mathbf{h}_i})}$$



(a) Gaussian fit: $NLL = 11.319 \times 10^3$



(b) Exponential fit ($\lambda = 0.2762$): $NLL = 9.340 \times 10^3$

Figure 5.6: The distribution of prediction errors.

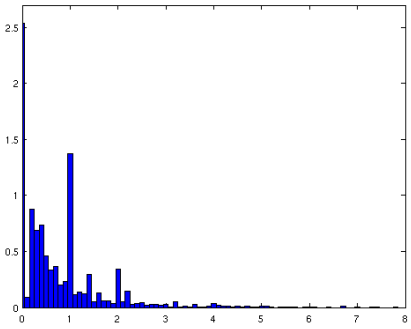


Figure 5.7: The distribution of prediction errors normalized by the traversed distance.

Each colour histogram is a vector of 64 elements. This means that in order to calculate the covariance matrix $\mathbf{K}_i$ it is necessary to gather at least 65 samples of colour histograms belonging to the particular person. This, however, would only be possible after a person has been detected in at least 65 frames, which would take at least 7 seconds. In order to compute the probability after just one frame, I assumed that the covariance matrix does not differ significantly for different people. I calculated a common covariance matrix by taking the mean of covariance matrices calculated individually for 100 people.

The validity of the assumption was then tested by the following experiment. For a person A (Figure 5.8a), I calculated the mean histogram and its covariance from a set of 91 detections. Then, I took a set of 50 detections of A, and 50 detections of another person B (Figure 5.8b), and calculated the probability that each of these histograms belonged to person A. I did this using the covariance matrix calculated for person A

(a) person A (b) person B

Figure 5.8: The two people used in the experiment of comparing histogram probabilities.

as well as using the general covariance matrix common to all people. None of the people used in the experiment were used to compute the general covariance matrix. Each colour histogram was normalised by dividing each bin by the number of pixels. The results are shown in the Figure 5.9.

5.2.1.2 Step 2: Assigning blobs to people

The second step of the tracking algorithm is illustrated by Pseudocode 4.

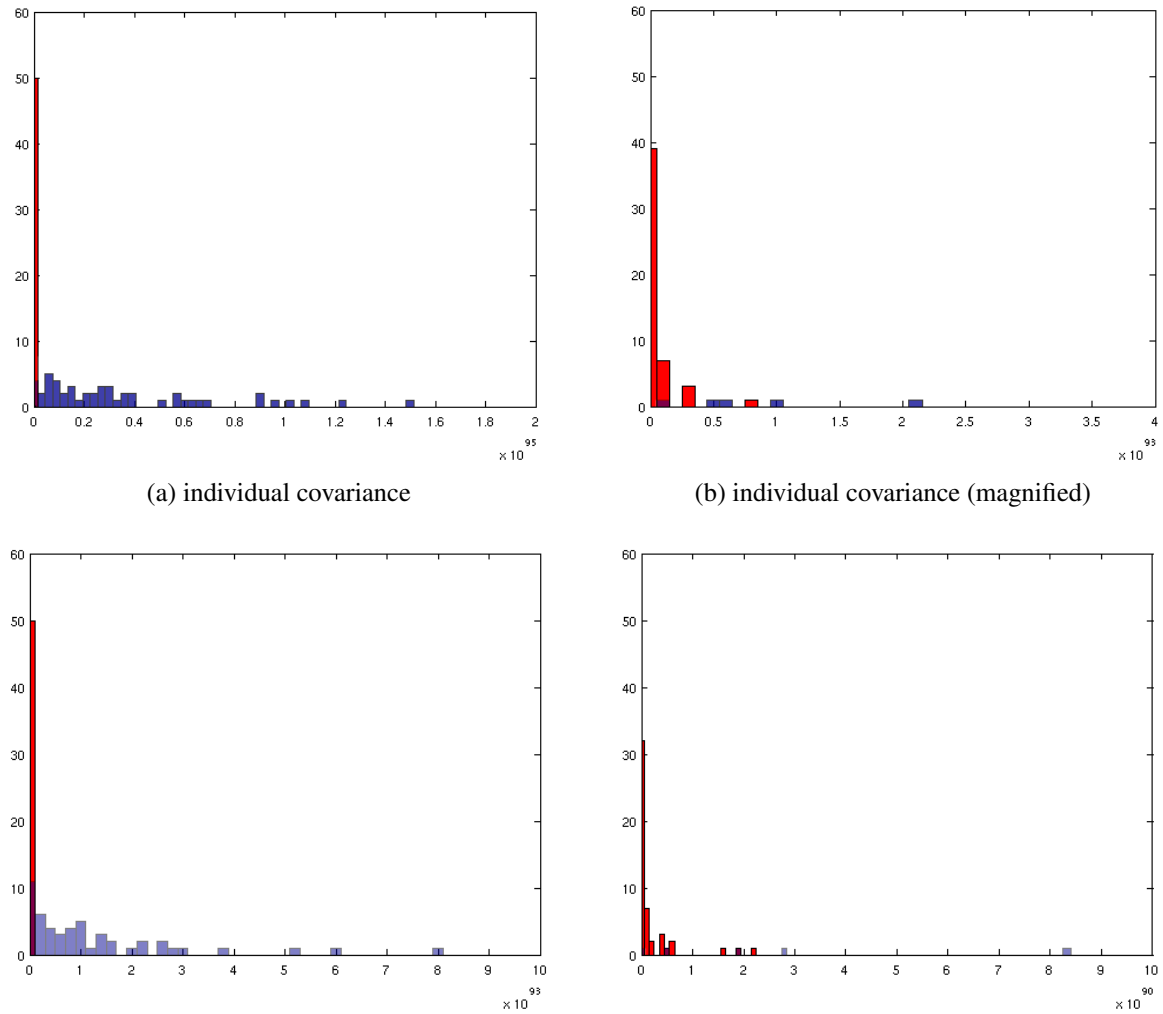
All the blobs which have not been assigned identities in the previous step are added to a list of remaining blobs. There are three possible reasons why a blob was left out:

1. It could be one of the products of the splitting of a group of people.
2. Its corresponding person could have been wrongly assigned to a blob already representing a different person.
3. It could represent a new person in the scene.

For each such blob I find the most probable corresponding identity. The same probability measure is applied as in subsection 5.2.1.1.3. If the blob does not lie within the prediction area of any person, then the blob is recognised as a new person which is then added to the list of people.

If such a person does exist, then it is checked whether this person was recognised by the previous step as merging with another person, i.e. whether the person was assigned to the same blob as another person. This would mean that there is a strong possibility of an incorrect assignment, which now should be corrected. Such a scenario is presented in Figure 5.10.

The last option deals with the case when the blob lies within the predicted area of a person who was assigned to a different blob, and no other person was assigned to the same blob. Then it is assumed that the remaining blob represents a person who was previously merged with the known person (Figure 5.11). A new person is created and



(a) individual covariance

(b) individual covariance (magnified)

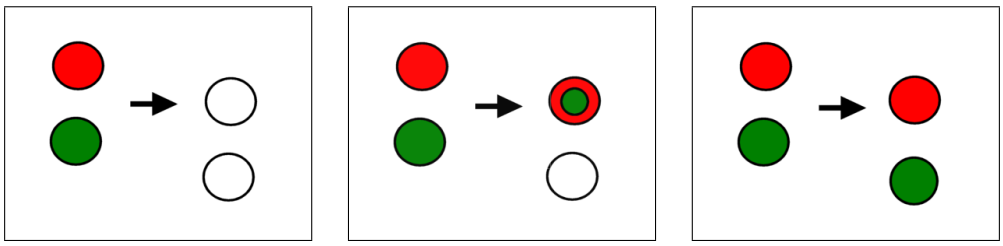
(c) general covariance

(d) general covariance (magnified)

Figure 5.9: The results of experiment of comparing probabilities of a histogram of person A belonging to the person A (shown in blue) and to the person B (shown in red). The blue bars are semi transparent to show the height of the red bars hidden underneath. The Figures 5.9a – 5.9b show the results when the individaully computed covariance matrix was used. The Figures 5.9c – 5.9d show the results when the covariance matrix was computed on the basis of the samples from 100 people.

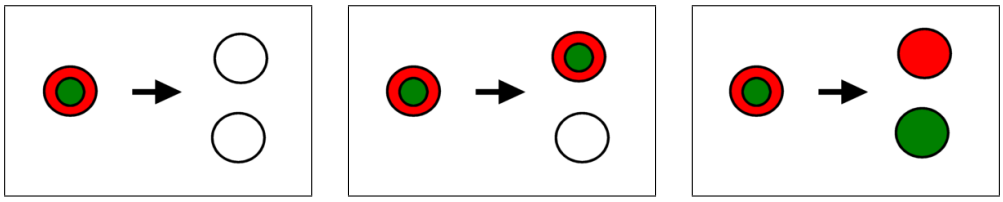
Pseudocode 4 Step 2: Assigning blobs to people.

```
remainingBlobs = blobs.getUnassignedBlobs()

FOR EACH blob IN remainingBlobs

   //-- look for most probable person --//
   FOR EACH person IN people
      predictedPosition = person.getPredictedPosition(interval)
      radius = person.getPredictedAreaRadius(interval)
      IF blob.isInArea(predictedPosition, radius)
         probabilities.add(person, blob.getProbability(person))
      END
   END

   chosenPerson = getMostProbable(probabilities)
   badAllocation = wasBadlyAllocated(chosenPerson)

   //-- blob -> person --//
   IF badAllocation
      correctBadAllocation(blob, chosenPerson)
   ELSE
      IF (chosenPerson != NULL)
         newPerson = chosenPerson.getCopy()
      ELSE
         newPerson = new Person()
      END
      assign(newPerson, blob)
   END
END
```



(a) input for step 1 (b) input for step 2 (c) step 2 completed

Figure 5.10: An example case of wrong correspondence made in the step 1.



(a) input for step 1 (b) input for step 2 (c) step 2 completed

Figure 5.11: An example case of not recognising splitting scenario by the step 1.

added to the list of people. Since both of these people walked together, they should have the same trajectories. Therefore the previous trajectory of the known person is copied to the new person and the current centroid position is added to the latter.

5.2.1.3 Removing People

A person can be removed from the list of people and their trajectory printed out to the output file on two conditions:

1. The person disappeared for five frames, or
2. The person disappeared for one frame and their expected position is now outside the scene.

The error in position prediction accumulates with every frame. Therefore, a balance had to be found between allowing a person to disappear and keeping the prediction area as small as possible. The number of frames during which a person is allowed to disappear was chosen experimentally. The prediction area is kept to a minimum to avoid problems with cases in which two people enter and leave the scene at the same time using the same entrance.

Before the trajectories are printed out to the file, the tracker tries to eliminate redundant trajectories. They appear when one person is detected as disjoint blobs.

If two trajectories started and ended at exactly the same position and time, and additionally the largest distance between corresponding points in these trajectories does not exceed a threshold of 100 pixels, then only one of them is printed out. The latter condition is used to make sure that all trajectories are printed out in the situations depicted in Figure 5.5.

5.2.1.4 Updating histograms

After the process of identity allocation is complete, I update the histograms $\mathbf{h}_i$ of those people who are not merged with others. This way, after the people split again, their individual histograms can be used for identity reassignment. The update takes the form of a weighted average:

$$\mathbf{h}_{i,n} = \frac{\alpha\,\mathbf{h}_{i,n-1} + \beta\,\mathbf{h}_{i,new}}{\alpha+\beta} \tag{5.5}$$

where $\alpha = 2$ and $\beta = 1$.

5.3 Implementation

The tracker was implemented in Java. The main structure of the program is illustrated by the class diagram shown in Figure 5.12. The general covariance matrix of colour histograms was built using Matlab and loaded into the Java program using the jmatio library.

5.4 Evaluation

The evaluation of the tracker was carried out on the basis of 15 different situations with multiple people present in the scene at the same time. The statistics gathered are presented in the Tables 5.1 - 5.3.

Additional evaluation was carried out in situations when there was only one person in the scene. For all of those cases single, full trajectories were produced.

The algorithm copes with redundant trajectories quite well. It has a very high effectiveness in preserving a full trajectory. The cases when the trajectories are split represent situations in which a person disappeared for more than four frames. The biggest problem caused identity reassignement. However, people usually walk alone and merging does not happen too often. And even though resolution of this problem

	% people	# people	# trajectories
full	95.12%	39	39
split	4.88%	2	6
total	100%	41	45

Table 5.1: The table shows the numbers of cases when a person's trajectory is split into fragments.

	# trajectories before	# trajectories after	% trajectories preserved
real	45	45	100%
redundant	34	2	6%
total	79	47	59%

Table 5.2: The table shows the number of redundant trajectories which were produced because a person was represented by several blobs.The table shows their number before and after the atomatic removal of similar trajectories.

# people	# merging and splitting	# correct identity reassignment	% failure rate
2	1	0	100%
3	4	3	25%
total	5	3	40%

Table 5.3: The table shows the failure rate of the identity assignment.

would be necessary for the developement of a commercial system, the tracker produced a vast majority of good trajectories with the sufficient robustness.

A set of example trajectories was plotted in Figure 5.14. They tend to have a form of straight lines in the center of the floor and take a curvy forms next to the lifts.

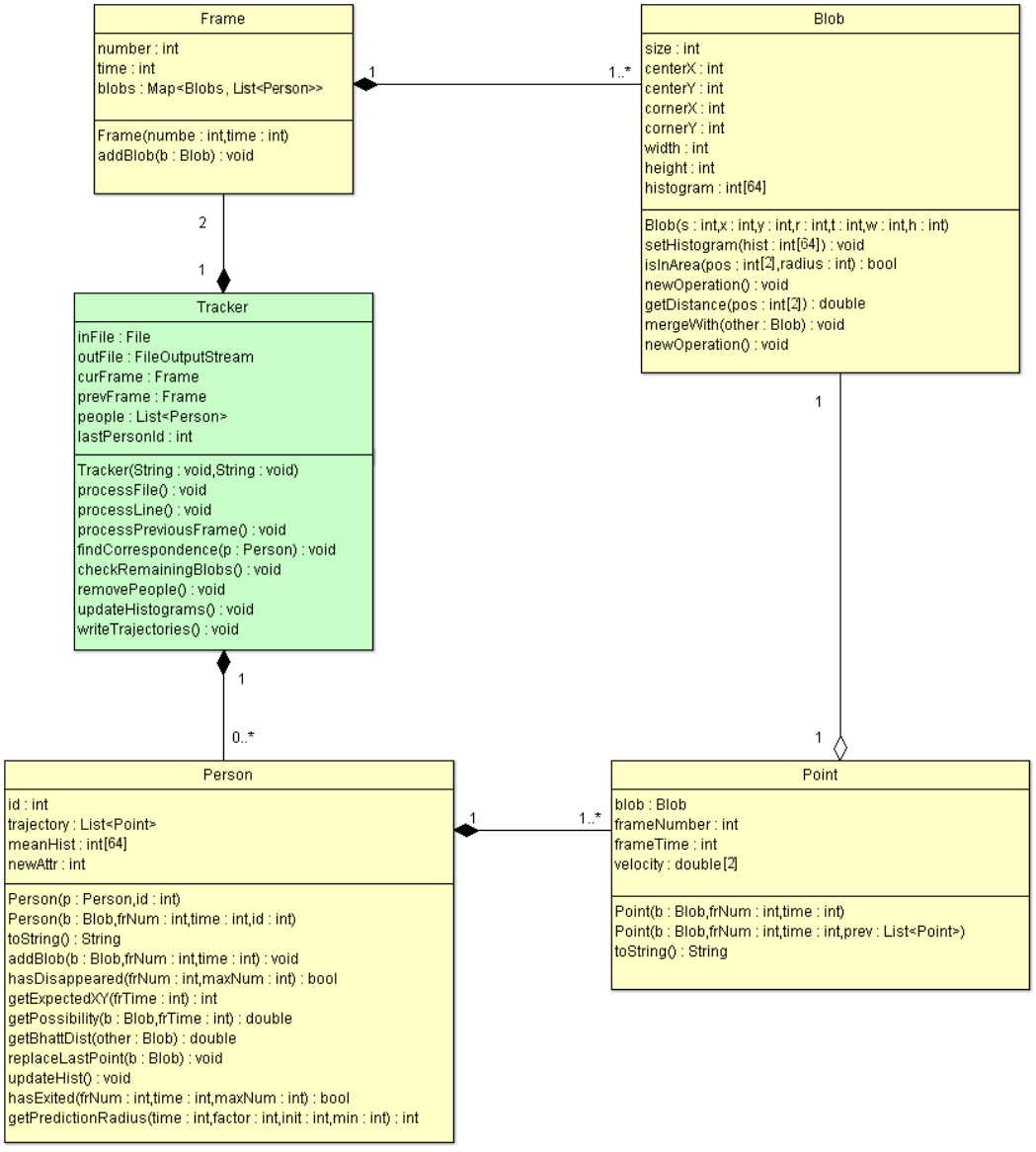


Figure 5.12: Class diagram of the Tracker component.

Figure 5.13: An example scenario. Three people walking closely together. Three other people present in the scene. The grey areas represent blobs. Each frame is represented by a different gray shade. Trajectories of each person are represented by different colours.

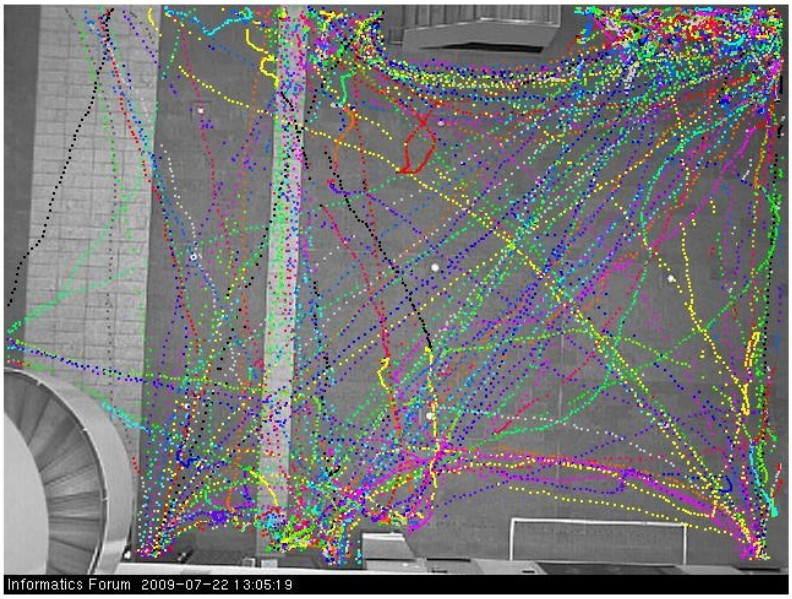


Figure 5.14: A set of trajectories. Some tracked objects were allocated the same colour due to the display purposes.

Chapter 6

Model of normal behaviour

6.1 Overview

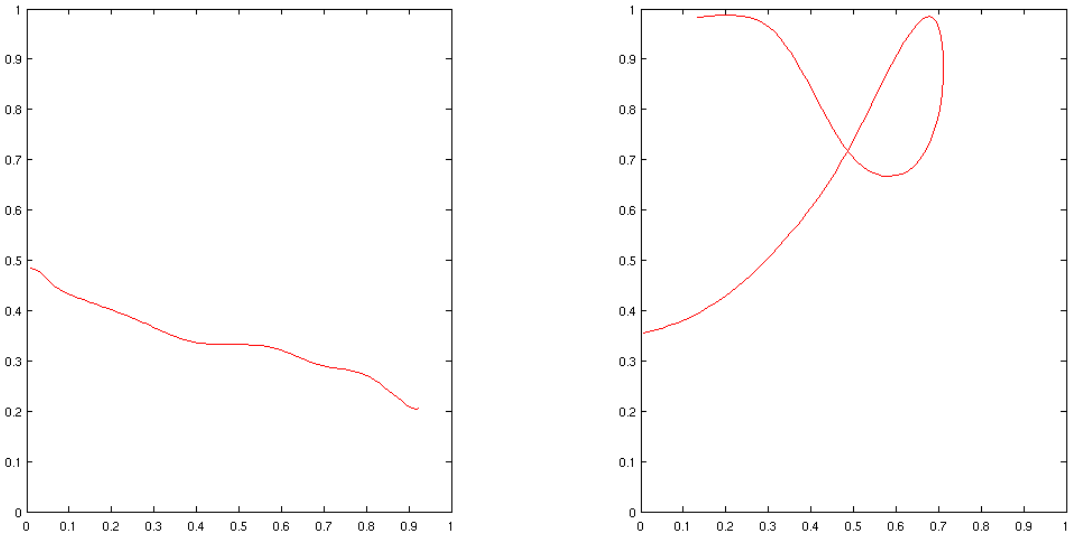
The purpose of the third component was to build a model of normal behaviour based on a set of collected trajectories, and then use this model to determine whether a new trajectory represents a potentially abnormal behaviour. The trajectories provided by the Tracker component were approximated by cubic spline curves and then divided into three parts:

1. **training dataset** was used for training the model (section 6.2.3);
2. **validation dataset** was used to choose the most appropriate parameters for the process of recognising anomalous trajectories (section 6.2.4);
3. **test dataset** was used to evaluate the robustness of the component (subsection 6.3).

A Gaussian mixture model was fitted to the trajectory data. Trajectories were then allocated a probability score which determined the likelihood of them being generated by the model. This was used to assess their normality.

Both the validation and evaluation processes required a manual judgement of trajectory normality. This assessment is subjective, and so it is difficult to develop a set of rules for making the decision. In most cases, a simple elimination technique was used – a trajectory was classified as normal if it met the following criteria:

- it represented an action with a clear goal (e.g. going from one exit to another)



(a) normal (b) abnormal

Figure 6.1: Example trajectories chosen for the evaluation.

- the goal was achieved in an efficient way (e.g. the trajectory was close to a straight line).

Note that these were rough guidelines only: there were normal cases that did not meet the second criterion. Two of the trajectories and their classifications are shown in Figures 6.1.

6.2 Building the model

The training dataset produced by the tracking component contained trajectories which represented both normal and abnormal behaviours. The trajectories of the former type could be selected manually. However, the model building requires a large number of such trajectories - at least a few thousand - making the purely manual approach impractical. Instead, each step of the model building also attempted to automatically remove those trajectories which were clearly abnormal.

The following subsections explain the steps in the model-building process.

6.2.1 Removing bad trajectories

The detector and tracking components are not 100% accurate, so the files produced by the tracker could contain bad trajectories. Some of these may be recognisably incorrect, such as those falling into the following classes:

- Trajectories which start or end outside the marginal area of the scene. The marginal area is shown in Figure 6.2. These trajectories could be produced if the detector did not detect a person in the initial or final frames. Also, the tracker could fail to notice that two people split and classify one of the resulting blobs as a new person in the scene. If they split outside the marginal area, the new person would appear out of nowhere.
- Trajectories shorter than 30 points. These could represent spurious detections.
- Trajectories which start and end in the area next to the lifts (Figure 6.2). They are produced by people waiting for a lift, therefore their shapes are characterised by a high level of randomness.

The preprocessing stage ensures that all these trajectories are not used when building the model by removing them from the training dataset.

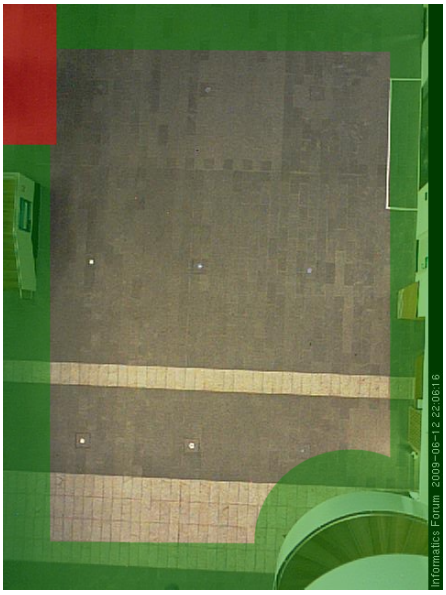


Figure 6.2: The marginal area (green) shows the region where trajectories have to start and end. The red area shows the region next to the lifts where the trajectories were removed.

6.2.2 Fitting splines

Trajectories have different numbers of points which makes it difficult to compare them. In order to represent them using the same number of attributes, each trajectory was approximated by a cubic spline curve with 11 control points. The fitting algorithm and its implementation were provided by Rowland R. Sillito [14].

Figures 6.3 – 6.4 show results of using different numbers of control points. All coordinates are normalised and can take values between zero and one (proportion of the length or width of the scene). Seven control points are sufficient to approximate a normal trajectory with high accuracy. However, abnormal behaviours tend to have more changes in path direction, and therefore more control points are needed to approximate them accurately. Figure 6.4 shows the results of fitting a spline to an abnormal trajectory using different numbers of control points. The approximation which uses only seven control points loses some of the trajectory features which make it appear abnormal. On the other hand, using too many control points can result in overfitting and make the resulting approximating splines sensitive to noise. This noise is most often caused by jumping centroids which is particularly visible in cases when people merge and split several times.

The representation using splines makes a distinction between two trajectories which have exactly the same positions but opposite starting points. Both of them have the same set of control points, however in a different order. This distinction can be important when determining whether a behaviour is abnormal, therefore it should be preserved.

Finally, I used the spatial approximation of the trajectories, i.e. the splines approximate the shapes and do not take into account the speed of particular fragments of the trajectory. Otherwise, 11 control points would not be sufficient to represent accurately the trajectory of a person who traversed a large distance, stopped and stayed in one position for some time, and then continued to one of the exits, after which they disappeared. Due to the fact that the person spent most of the time standing in one position, the spline fitting algorithm would focus on this part of its trajectory, which resulted in a poor representation of the actual path (Figure 6.5).

6.2.3 Gaussian mixture model

A Gaussian mixture model was fit to the data using the Netlab toolbox for Matlab. I set the initial number of components to $K = 200$ and the covariance structure of each

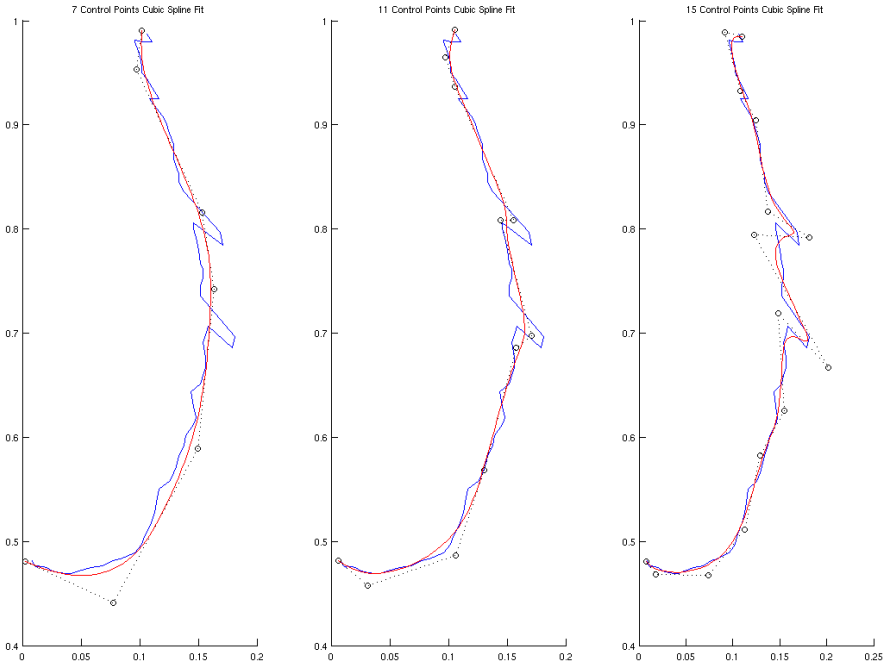


Figure 6.3: A cubic spline with 11 control points fitted to a normal trajectory. The irregularities in the trajectories are caused by 'jumping' centroids.

component to be diagonal. The centres of the model were initialised by 100 iterations of the K-means algorithm. Their priors were calculated using the maximum likelihood approach: they were set to the proportions of datapoints belonging to the clusters. The model was trained using the Expectation Maximisation (EM) algorithm for 100 cycles. The centres of the resulting mixture model are shown in Figure 6.6.

The model was fit to all 8382 trajectories including those which were in fact abnormal. Therefore, the model was not the best possible representation of normal behaviour. However, the abnormal trajectories tended to be assigned very low probabilities. Therefore I decided to use this initial model to preprocess my training dataset by removing from it all the trajectories which the model deemed to be highly unlikely. I removed 5% of the trajectories from the dataset which was then used to retrain the model. The number of components was chosen to be $K = 50$.

6.2.4 Recognition

The abnormality of a new trajectory is determined on the basis of the probability assigned to it by the Gaussian mixture model. When this value does not exceed a chosen

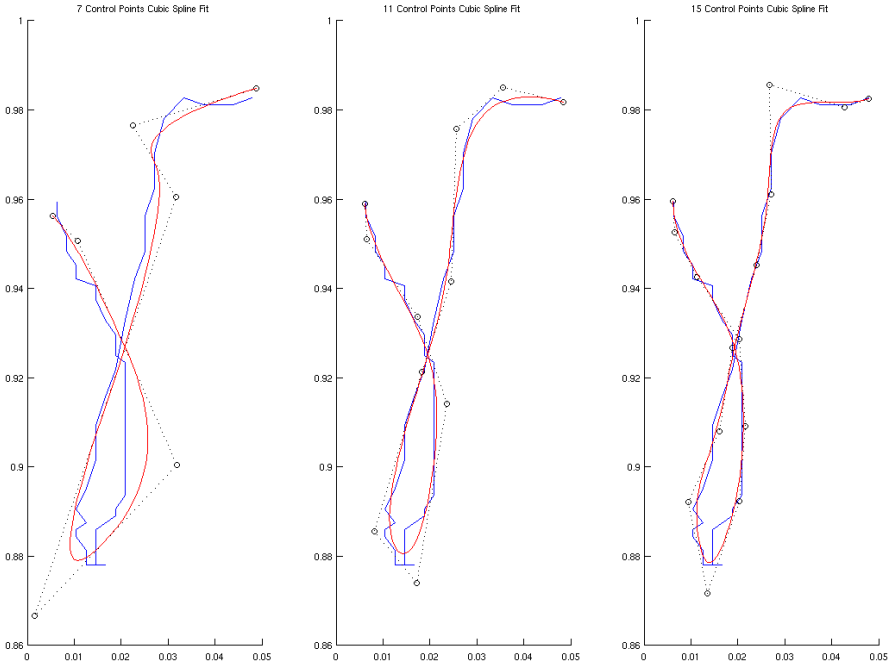


Figure 6.4: A cubic spline with 11 control points fitted to an abnormal trajectory.

	False	True
Positive	15% (8)	85% (42)
Negative	15% (18)	85% (102)

Table 6.1: Validation results of the Gaussian mixture model with 50 components and the threshold set to 10^9.

threshold then the trajectory is classified as abnormal (*positive*). The Receiver Operating Characteristic curve was plotted for different threshold values, as shown in Figure 6.8. The validation dataset contained 120 normal trajectories and 50 abnormal ones. The threshold which gives the optimal ratio of false positives to false negatives is: $threshold = 10^9$.

An appropriate balance between false positives and false negatives was required. Too many false positives could distract CCTV camera operators, reducing the utility of the program. On the other hand, it is very important to ensure that the program minimises failures to detect abnormal behaviours.

The classification performance on the validation set is shown in table 6.1.

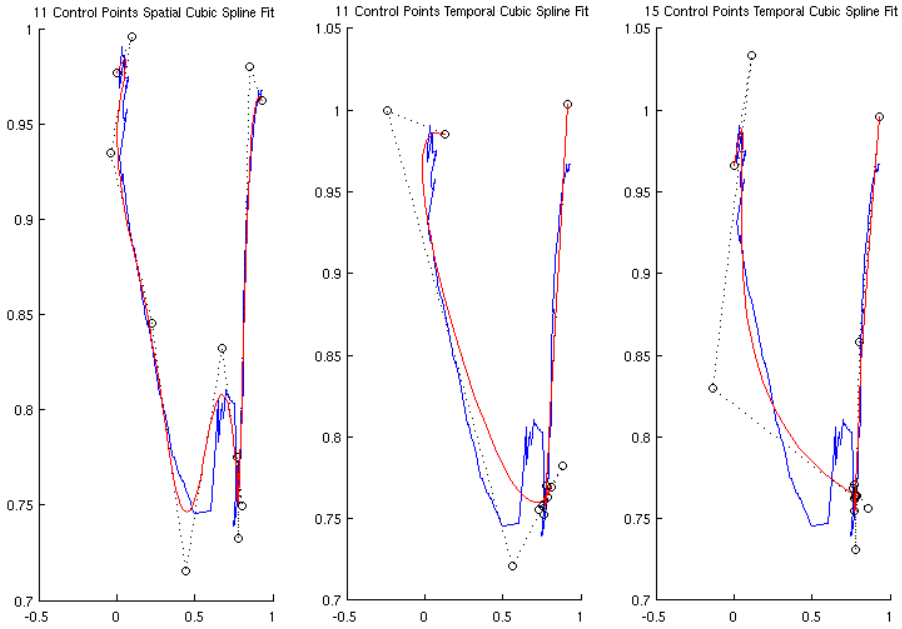


Figure 6.5: Comparison of spatial and temporal cubic spline fitting.

6.3 Evaluation

The evaluation set consisted of 50 abnormal trajectories and 120 normal ones. The results are presented in Figure 6.9. Some of the misclassified trajectories are shown in Figure 6.10. The Gaussian mixture model had problems with abnormal trajectories that were quite similar to normal ones (Figure 6.10a). Also, some trajectories that were classified as abnormal turned out to be combinations of normal ones. This is due to an occasional mistake in tracking: if a person leaves the scene and another person enters at the same time and in the same place, this new person is assigned the identity of the person leaving. This produces a trajectory which looks abnormal but is actually a concatenation of normal paths (Figure 6.10b).

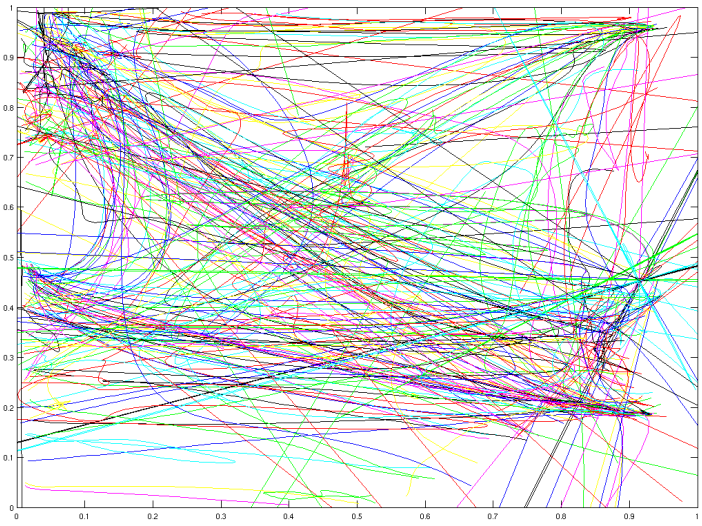


Figure 6.6: The centers of the Gaussian mixture model with 200 components.

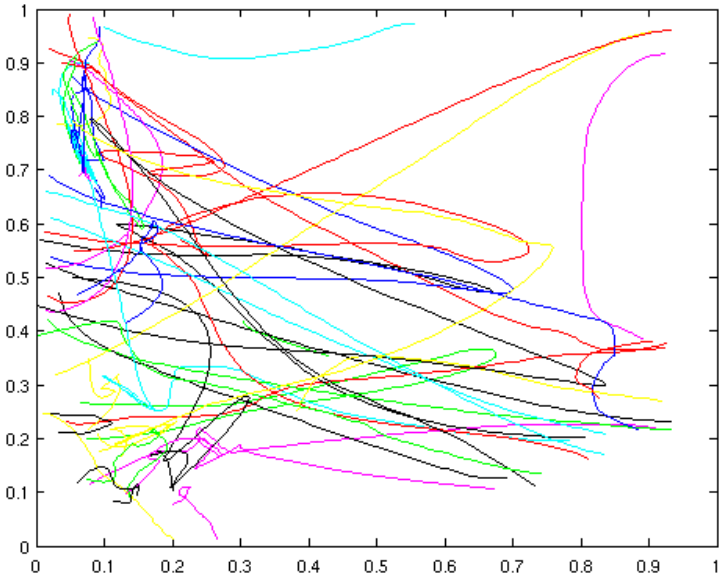


Figure 6.7: The centers of the Gaussian mixture model with 50 components.

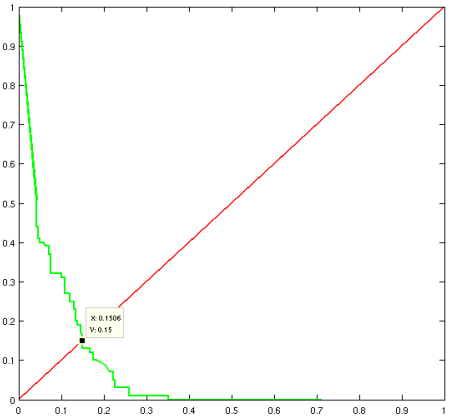
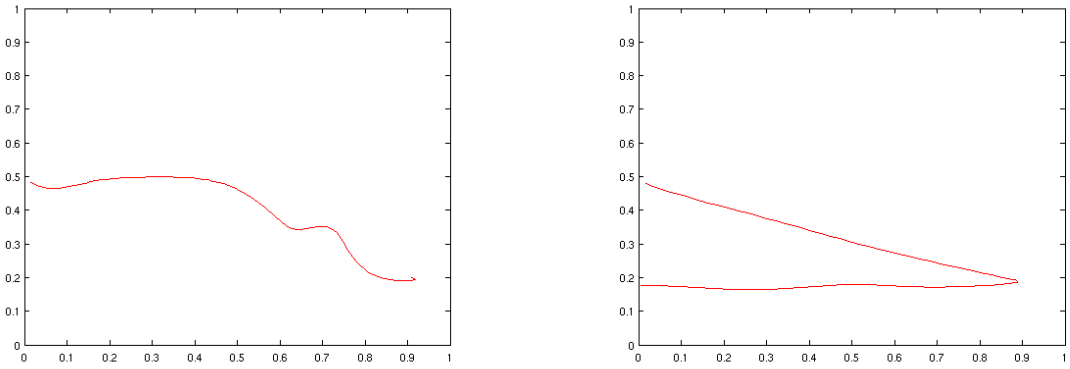


Figure 6.8: The Receiver Operating Characteristic curve for the Gaussian mixture model with 50 components. X axis: False Negative rate. Y axis: False Positive rate.

	False	True
Positive	15% (8)	85% (42)
Negative	10% (12)	90% (108)

Figure 6.9: Evaluation results of the Gaussian mixture model with 50 components and the threshold set to 10^9.



(a) false positive (b) false negative

Figure 6.10: Example misclassified trajectories.

Chapter 7

Conclusion and future work

The process of detecting people proved to be a non-trivial task. Difficult lighting conditions, as well as the need for an efficient algorithm, led me to investigate different techniques for image segmentation. The use of chromaticity coordinates plus Principal Components Analysis produced a detection quality sufficient for the purposes of this project. However, it would need to be improved if it were to be used in a commercial system. This is because poor detections, such as people represented by several blobs, had a negative impact on the performance of other components.

This problem could be avoided by applying an additional segmentation step, with a lower threshold, in the areas around the initial foreground detections. Other segmentation techniques could also be investigated. The Tracker could try to merge blobs which lie sufficiently close to each other. In order to do this, it could compute the probablity of a set of blobs composing a given person. This would require improvements in the representation of colour histograms. For example, the number of bins could be increased and their dimensions could be different along each colour channel.

Despite the limitations of the detector, the majority of the trajectories produced by the first two components represented real paths. The results for the detection of anomalous behaviour illustrate that a system with a perpendicular camera could be useful in a practical situation. It would be able to eliminate those trajectories that are clearly normal, and therefore point the CCTV operators to those cameras which show potentially anomalous behaviour.

The methods with which I experimented could be supplemented by additional components - for example, a module that could flag trajectories whose cubic spline approximations are not accurate. Such a trajectory could be produced, say, by an inebriated person. It was not possible to do this in the current system due to the fact that trajecto-

ries contained relatively rapid changes in the blobs' centroid positions, and these could be confused with genuinely erratic trajectories. Another possible extension would be to make use of a person's movement speed.

The current application would be most useful for cases in which statistical analysis of pedestrian behaviour is desirable - for example, for finding the most popular routes or checking which areas are the most visited.

Bibliography

[1] B. Brown, Police Research Group. CCTV in Town Centers: Three Case Studies, 1995.

[2] E. Wallace and C. Diffley. CCTV making it work - CCTV control room ergonomics 14-98, 1998.

[3] B. Bennett, D. Magee, A.G. Cohn, and D.C. Hogg. Using spatio-temporal continuity constraints to enhance visual tracking of moving objects. In *16th European Conference on Artificial Intelligence*, pages 922–926, 2004.

[4] Liberty CCTV, 2005. http://www.liberty-human-rights.org.uk/issues/3-privacy/32-cctv/index.shtml.

[5] A. Datta, M. Shah, and N. Da Vitoria Lobo. Person-on-person violence detection in video data. In *16th International Conference on Pattern Recognition*, volume 1, pages 433–438, 2002.

[6] H. Dee and D. Hogg. Detecting inexplicable behaviour. In *BMVC*, pages 477–486, 2004.

[7] H. Dee and S. Valestin. How close are we to solving the problem of automated visual surveillance? *Machine Vision and Applications*, 2007.

[8] S. Dockstader and A. Tekalp. Multiple camera fusion for multi-object tracking. In *IEEE Workshop on Multi-Object Tracking*, 2001.

[9] N. Johnson and D. Hogg. Learning the distribution of object trajectories for event recognition. *Image and Vision Computing*, 14:583–592, 1996.

[10] J.W. Kim, K.S. Choi, B.D. Choi, and S.J. Ko. Real-time vision-based people counting system for the security door. In *International Technical Conference On CircuitsSystems Computers and Communications.*, 2002.

[11] Y. Li. On incremental and robust subspace learning. In *Pattern Recognition*, pages 1509–1518, 2004.

[12] N. Robertson and I. Reid. A general method for human activity recognition in video. *Computer Vision and Image Understanding*, 104:232–248, 2006.

[13] R.R. Sillito. Phd thesis.

[14] R.R. Sillito and R.B. Fisher. Semi-supervised learning for anomalous trajectory detection. In *BMVC08*, 2008.

[15] T. Teixeira and A. Savvides. Lightweight people counting and localizing in indoor spaces using camera sensor nodes. *Distributed Smart Cameras*, pages 36–43, 2007.

[16] T. Xiang and S. Gong. Video behavior profiling for anomaly detection. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 30:893–908, 2008.