Automatic Incident Detection in Video Surveillance

PIERRE BÉGUERIE

MSc by Research in Pattern Analysis and Neural Networks

ASTON UNIVERSITY

August 2006

ASTON UNIVERSITY

Automatic Incident Detection in Video Surveillance

PIERRE BÉGUERIE

MSc by Research in Pattern Analysis and Neural Networks, 2006

Thesis Summary

A new generation of video based detection systems for use outside has recently been developed. They are intelligent detection systems which provide automatic surveillance through real-time video analysis and event detection. In doorway scenarios, the goal of such systems is to be able to automatically detect any incident occurring in front of a door.

This thesis introduces several tools useful to create an automatic incident detection system in video surveillance. Every event occurring in the scene filmed by the camera is first detected. Then, classification techniques will be described so that a robust human classifier can be constructed; assuming the humans are the only objects able to cause an incident. A probabilistic video tracking algorithm will then be presented. It will be used to track the detected human beings over time in order to estimate their trajectories and know if an alarm should be given or not by the system. The theoretical backgrounds to these tools will be explained. Some results will also be shown. Toy data will first be used to validate the different tools separately. Global results on real data will be displayed at the end of the thesis.

Keywords: Video Tracking, Motion Estimation, Event Detection, Shape Classification

Acknowledgements

I would like to thank Professor Ian Nabney for his more than weekly help during the whole project. I would also like to thank Doctor Yuan Yuan. Their thoughtful comments have always been extremely useful to me.

I would like to thank TVM for the time they spent installing and helping me understanding their softwares, for the very useful meetings we had, for the video data they provided us.

Thank you so much to everyone who spent and made this wonderful year in Birmingham with me. Thank you all.

Contents

List of Figures

Chapter 1

Introduction

Surveillance camera systems (such as CCTV) for use outside were first developed as a means of increasing security in banks. Today they have developed to the point where they are simple and inexpensive enough to be used in home security systems.

CCTV was developed partly in response to IRA bombings in the United Kingdom. Experiments in the UK during the 1970s and 1980s led to several larger trial programs in the early 1990s. These were deemed successful in the government report "CCTV: Looking Out For You", issued by the Home Office in 1994, and paved the way for a massive increase in the number of CCTV systems installed. Today, systems cover most town and city centres. The exact number of CCTV cameras in the UK is estimated around 4,000,000.

Nowadays, a new generation of video based detection systems (VBDS) has been developed. They are intelligent detection systems which provide automatic surveillance through real-time video analysis and event detection. Surveillance camera systems have become such a burning issue that the Home Office has developed its own imagery library for intelligent detection systems (i-LIDS) so that academics and system manufacturers can test the performance of their VBDS.

The aim of this project was to build an intelligent VBDS able to automatically detect the incidents occurring in front of a doorway. We will show you how we first detected any event occurring in the scene filmed by the surveillance camera system. We will then describe the classification methods we applied in order to only consider the events which can cause an incident (i.e. the human beings). Finally, we will explain how, once detected and classified, the human beings are tracked over time so that we can decide whether an event alarm should be given or not.

After presenting the application context in Chapter 2, the different methods we applied and used during the project will be described. Motion estimation algorithms

and some of their applications will be explained in Chapter 3. After that, mathematical models will be used to build a video tracking algorithm in Chapter 4. Some preliminary processes applied to the video data will then be presented in Chapter 5. Chapter 6 will deal with the detection algorithm we developed and the classification methods we used in order to construct a robust human classifier. Finally, some results on real video data will be shown in Chapter 7.

Chapter 2

Application Context

TransVisualMedia (TVM) is an innovative English company based in Birmingham. They are specialized in video based detection systems. Their already existing VBDS concentrates on doorway surveillance scenarios. Thanks to their system, any human being passing in or out of the door should trigger an alarm event. Any other event (vehicle driving in the street, human being walking through the scene, illumination change) should be neglected.

However, TVM was facing difficulties classifying events triggered off by human beings from the others. They also tend to struggle with changing environmental conditions (as they are dealing with outdoor applications). These variations generate a high number of false alarms, reducing the effectiveness of their algorithms. The goal of this project was to improve this already existing system.

2.1 TVM's algorithms

This section is not an exhaustive description of the whole already existing TVM's system. Some key ideas are just given so that the reader can understand the general concepts used in their algorithms.

An event is caused by an object moving in the scene. To be considered, such an object has to be larger than a minimum threshold size and smaller than a maximum threshold size. These limit sizes depend on the setting up of the system. They are initialized by hand. Doing so, any too small event is neglected. Too big events caused by vehicles for instance are not taken into account as well.

In order to focus on the regions of interest (i.e. the regions where events can occur), some masks can be applied to the video data. Such processes cannot be generalized to every camera, they are initialized by hand.

Cameras are not set up carefully for orientation. Indeed, they can be hidden behind

a curtain or a window. Masks can also be applied in order to cancel lighting issues caused by such orientation conditions. They also need to be initialized by hand.

To conclude, TVM's system is based on a minimum/maximum size thresholds system which reduces the number of events taken into account. Moreover, different masks can be applied locally in order to cancel problems to a particular camera.

2.2 The video data provided

TVM's algorithms are run over video sequences recorded by fixed cameras aiming at a doorway. The scene is filmed 24 hours a day.

Every ten minutes, a new avi file is generated. This will allow us to run the algorithm for every ten-minute video sequence separately. Actually, avi files are converted to motion JPEG format which contain a list of images extracted from the video (in JPEG format). Four frames are extracted every second. As a video sequence lasts ten minutes, a motion JPEG file contains around 2400 frames. These motion JPEG files are used as the inputs of the algorithms of the system.

Few examples of video frames provided are shown in Figure 2.1.

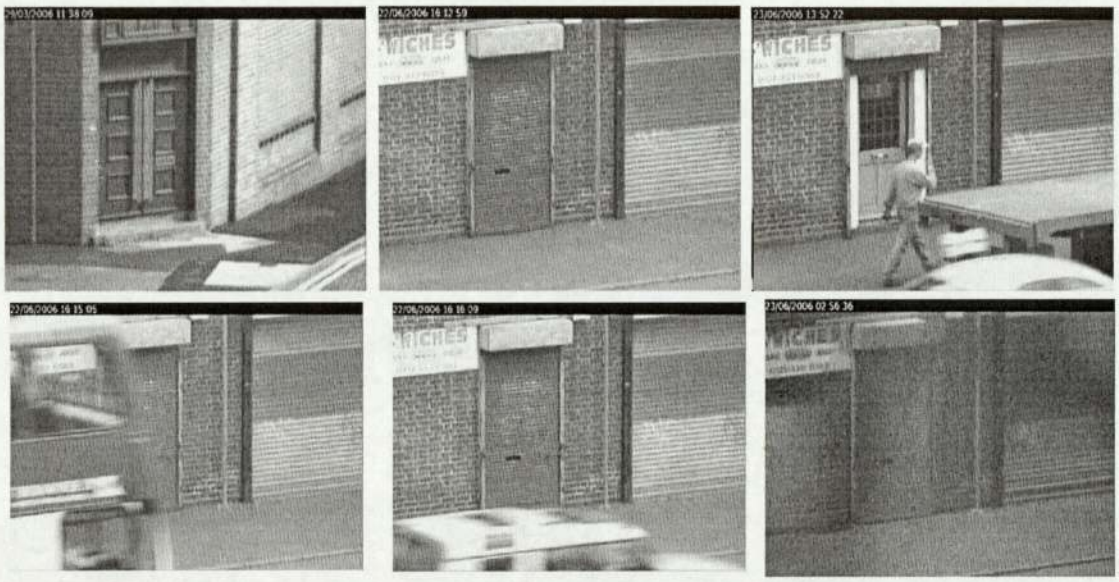

Figure 2.1: Examples of video data provided. *The camera is fixed, aiming at a doorway. Some events can occur in the scene (a man walking, a bus or a car driving). The scene is filmed 24 hours a day, during days and nights.*

2.3 Issues and Objectives

Such a system did not allow TVM to get sufficient results. The false positive rate was too high. Too many events were still taken into account. The purpose of this project was to improve the robustness of TVM's algorithms.

A robust human classifier needs to be constructed so that human beings or groups of human beings can be distinguished from any other events (cars, buses).

Illumination changes generated many false triggers, especially at night. Solutions need to be found.

Moreover, in order to generalize the system to every camera, the parameters of the system need to be adjusted in another way.

In the next chapter, we shall consider the problem of motion estimation. This first tool will be used to try to improve the robustness of TVM's system.

Chapter 3

Motion Estimation

Information from video sequences can be extracted using the intuitive idea of motion between frames. Pixels corresponding to an object on one frame move to form a corresponding image of the object on the subsequent one. To reduce this temporal redundancy, techniques such as motion estimation [1] are used.

3.1 Macro-Block Motion Estimation

Motion vectors defining displacement of regions are defined into the frame. Each scene is divided into non-overlapping macroblocks assumed to be composed of closely associated pixels moving the same way (distance and direction). Each macroblock defines a single motion vector. The process of determining the values of the motion vector for each frame is called macro-block motion estimation.

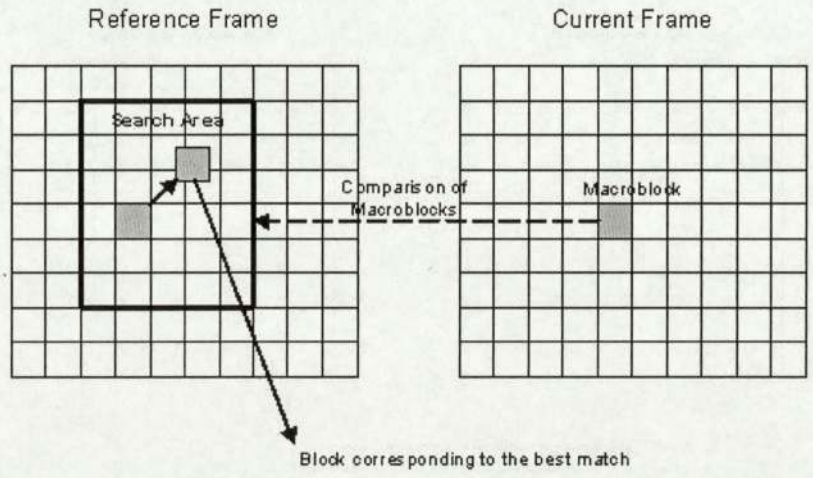

Figure 3.1: Macro-Block Motion Estimation.

In order to determine these motion vectors, three key ideas are needed. To make this

clear, let us define motion estimation in another way. Motion estimation can also be seen as the comparison of the macroblocks F of a current frame with the macroblocks of a search area G in a reference frame (Figure 3.1). The reference frame generally represents the previous frame as we are trying to estimate the motion of objects over time.

3.1.1 Search Area

We first need to define the search area in the reference frame - the region where we should find the corresponding block of a given macroblock of the current frame. The search area is usually simply defined by a parameter p, the search parameter that defines the size of the area. To give an example, in MPEG compression, $p = 6$ so that the search area surrounding the macroblock looks like the one shown on Figure 3.2.

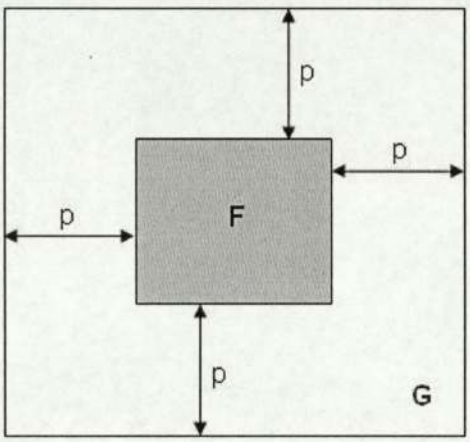

Figure 3.2: Search area. *The search parameter p defines the search area G surrounding the macroblock in the current frame F.*

3.1.2 Cost Function

Then, we have to introduce a cost function between macroblocks to be able to compare them. Some commonly used cost functions are : the mean-squared difference (MSD) and the mean-absolute difference (MAD).

$$MSD(x,y) \propto \sum_i \sum_j [F(i,j) - G(i+x, j+y)]^2 \tag{3.1}$$

$$MAD(x,y) \propto \sum_i \sum_j |F(i,j) - G(i+x, j+y)| \tag{3.2}$$

The smaller the value of the cost function, the better two blocks match.

3.1.3 Search Algorithm

Finally, we have to define an effective search algorithm that finds the best matching block in the search area efficiently - computing as few cost function values as possible.

Exhaustive Search Algorithm

The obvious algorithm one could implement is the exhaustive search algorithm that computes cost function values for every macroblock in the search area; this is very costly! Its complexity is of $O(p^2)$. Figure 3.3 illustrates such a search algorithm.

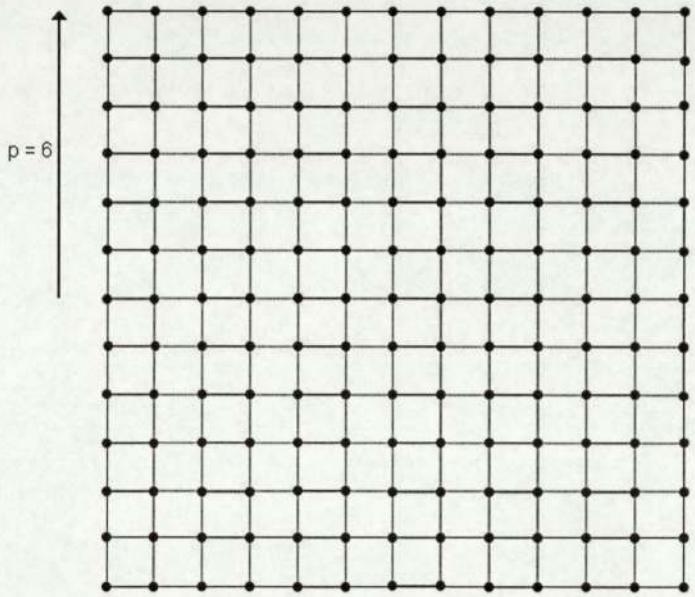

Figure 3.3: The exhaustive search algorithm.

Fast Search Algorithm

As soon as a search algorithm is not exhaustive, it becomes a fast search algorithm. Many popular techniques have been proposed in the literature [1]. They are faster than exhaustive search by computing fewer cost function values. Their complexity is of $O(\log p)$.

Figure 3.4 illustrates the Three-Step Search Algorithm, one of the most popular algorithms. It was implemented by Lee et al. in 1994. It works in three steps:

1. *Step 1:*
   The algorithm starts by computing the cost function value at the centre of the search area. A step size L is then defined by $L = p/2$. Cost function values are then calculated at the eight following points surrounding the initial

point: $(L,0),(L,L),(0,L),(-L,L),(-L,0),(-L,-L),(0,-L),(L,-L)$. These eight values are compared. The best one (i.e. the smallest one) is chosen to become the best matching point so far. This position represents the current best motion vector for the block in the search area. It becomes the center location of the next step.

2. *Step 2:*
   The step size is divided by 2: $L = L/2$. The cost function is then applied to the new eight surrounding locations this step size away from the current best position. As in step 1, the smallest value among these eight points and the current best point becomes the new best location.

3. *Step 3:*
   This process of calculating the eight neighbours of the current best position continues until the step size becomes smaller than 1. At this point, the current best point becomes the actual best point in the whole search area. It gives the selected motion vector for the block in the search area.

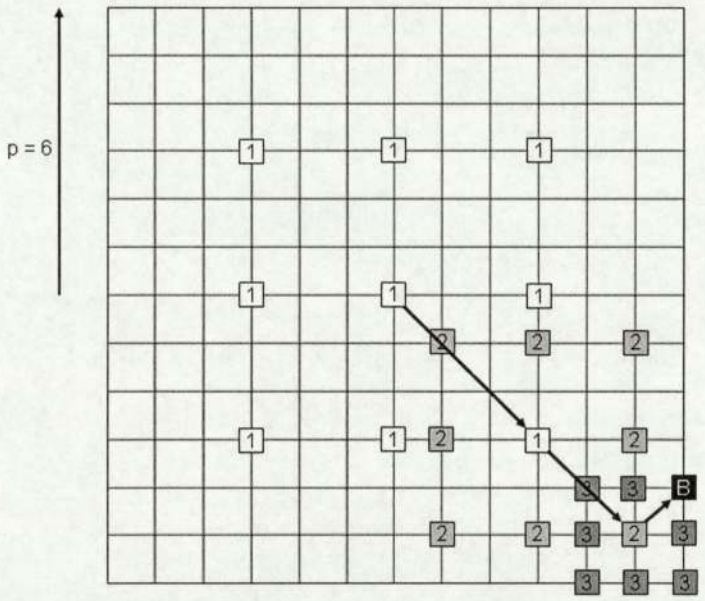

Figure 3.4: The three-step search algorithm. *The labels in the squares correspond to the iteration of the algorithm. The first step of the algorithm finds a new best position at the location (3,-3). Then, dividing the step size by two, the new best location after step 2 becomes (5,-5). Finally, the actual best location is reached at (6,-4) after step 3. (6,-4) is the obtained motion vector.*

As it computes fewer cost function values, this algorithm is more efficient than the exhaustive search algorithm. Other algorithms working in a similar way have

been proposed: for instance the two-dimensional logarithmic search algorithm and the conjugate direction search algorithm [1].

However, these quite old algorithms are based on a very strong assumption that cannot be verified in real data. The search area viewed by these motion estimation techniques is assumed to have one and only one minimum (for the cost function). New algorithms do not assume this anymore. One of these algorithms, Diamond Search [2] is particularly efficient.

3.1.4 Diamond Search Algorithm

Observations

This algorithm relies on two main observations. Firstly, in real data, the assumption made in the previous fast search algorithms of a unique global minimum is unlikely to hold.

The second observed fact is that motion vectors are very often enclosed in a circular support with a radius of two pixels, centred on the position of zero motion. Moreover, displacements are mainly horizontal and vertical. Hence, taking into account these two facts, a new search pattern and algorithm have been proposed by S. Zhu and K. K. Ma in 2000. The search pattern is composed of two diamond patterns: the Large Diamond Search Pattern (LDSP) and the Small Diamond Search Pattern (SDSP) (Figure 3.5).

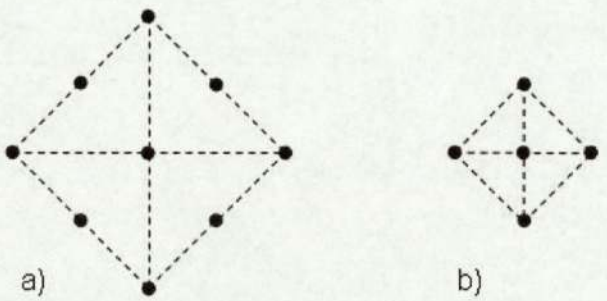

Figure 3.5: The diamond search patterns. *a) Large Diamond Search Pattern (LDSP), b) Small Diamond Search Pattern (SDSP).*

The Diamond Search Algorithm

The diamond search algorithm works in three steps:

1. *Step 1:*
   The initial LDSP is centered at the origin of the search area. The cost function is

then evaluated at the nine points of the LDSP. If the minimum value is achieved at the centre of the LDSP, then go to *Step 3*, otherwise, go to *Step 2*.

2. *Step 2:*
   The minimum point found in the previous step becomes the new centre point of the LDSP. The nine cost function values of the new LDSP are then evaluated. If the minimum is achieved at the centre position, go to *Step 3*, otherwise, recursively repeat this step.

3. *Step 3:*
   Switch the search pattern from LDSP to SDSP with the same centre. Cost function values are then evaluated at the five points of the SDSP. The minimum gives the actual final solution of the motion vector.

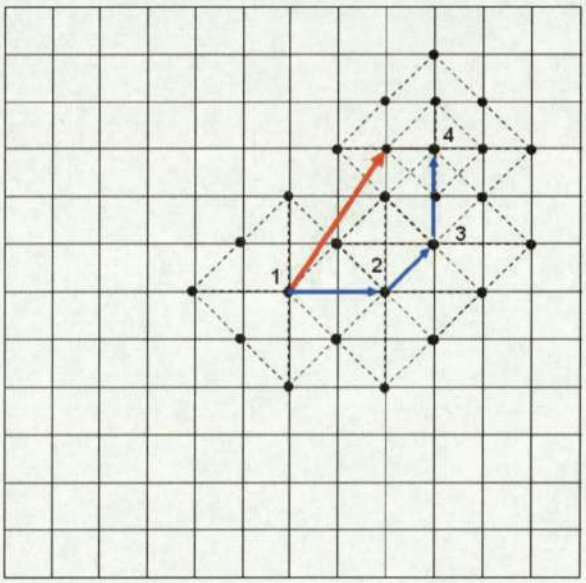

Figure 3.6: The diamond search algorithm. *The initial LDSP is centered at the origin of the search area. The first best point is found to be (2,0) (Step 1). LDSP are iteratively updated (labels corresponding to the number of iterations are displayed in the LDSP): the new minimum becoming the center of the next LDSP (Step 2). As soon as the minimum is reached at the centre location of the LDSP (at (3,3)), the search pattern is switched to SDSP (Step 3). The best point is found at (2,3). It defines the motion vector of the macroblock (red arrow).*

Figure 3.6 illustrates this algorithm. This technique is very efficient as very few cost function values are computed to find the final motion vector. Moreover, from one LDSP to another, many points are redundant and do not need to be computed twice.

This is the motion estimation search algorithm we have decided to implement. Figure 3.7 is an example of diamond search motion estimation on the video data provided.

Figure 3.7: Example of motion estimation. *From the initial frame (left), we reconstruct the current frame (centre) using diamond search motion estimation. The reconstructed frame is displayed on the right.*

The woman walking has moved from left to right. Starting from her left position in the previous frame, the position of the woman in the reconstructed frame is similar than the one in the current frame. The motion estimation algorithm has estimated well the motion of the human being. Moreover, for the macroblocks on the fringe of the image, the search algorithm can find the minimum in regions outside the initial image. That is why we note black blocks appearing in the reconstructed frame.

Run in Matlab, the algorithm of reconstruction of such a frame with macroblocks of 8 times 8 pixels lasts around 12 seconds. It should be much faster using other programing language.

3.2 Feature extraction using Motion Estimation

Once we have built the reconstructed frame using motion estimation, we want to be able to extract the moving objects of the scene.

3.2.1 Motion Frame

Each macroblock has moved along a certain motion vector. The faster the object moves, the longer its corresponding motion vectors are. Therefore, from the reconstructed motion estimation image, we build the motion frame which is the frame of the norms of the motion vectors. It is a grey scale image of $N * M$ pixels with N and M the number of macroblocks in a row and a column of the image respectively.

Figure 3.8 shows a motion frame obtained from a motion estimation reconstructed image (the one in Figure 3.7). When a macroblock has not moved, its norm is 0. Its corresponding pixel in the motion frame is black. The longer a motion vector is, the whiter its corresponding pixel in the motion frame becomes.

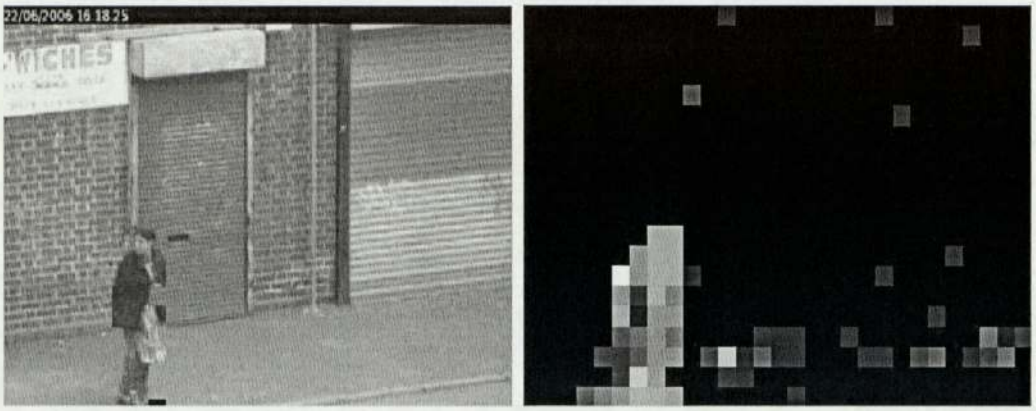

Figure 3.8: Example of motion frame. *From the motion estimation reconstructed frame (left), we build the motion frame (right). The moving areas become grey or even white while the static regions are black.*

3.2.2 Contours of the moving objects

Motion frames are quite noisy. We note on Figure 3.8 that some macroblocks from the background have moved although no moving object is in this area. This is due to little illumination changes occurring in real data. In order to get rid of this noise, two-dimensional median filter is applied to the motion frames to obtain smoother areas of motion. The median filter considers each pixel in the image and looks at its nearby eight neighbours. The pixel is then replaced by the median of those values. The median is calculated by first sorting all the pixel values from the surrounding neighbourhood into numerical order and then outputting by the middle value among the nine sorted value (ie the fifth value).

Contours are then drawn around the regions of motion detected [7]. The contours are the isolines of the filtered motion frame. The object boundary length must be greater than a minimum threshold in order to be considered. This removes small moving objects. Contours of the motion frame are finally converted from macroblock to pixel to fit the real data and actually surround the moving objects (Figure 3.9).

3.3 Experiments and Results

The images shown in section 3.2 suggest that diamond search is a good motion estimation algorithm for this data (Figure 3.7). Moreover, the feature extraction module using motion estimation has also given satisfactory results (Figure 3.9).

We want to test the motion estimation module on more examples to validate it and identify any drawbacks.

Figure 3.9: Contours of the moving objects. *The motion frame is filtered to get rid of the noise due to little illumination changes (left). Contours of the moving objects are then drawn (blue curve). They are finally converted from macroblock to pixel to fit the real data (right).*

Several examples are displayed from Figure 3.10 to Figure 3.14. They show six images. In the first row, the first two are the previous and the current frame respectively and the third image is the reconstructed frame using motion estimation. In the second row, the motion frame, the filtered motion frame and the current frame with the contours of the moving objects (blue curves) are displayed from left to right.

Motion estimation is a tool that works in video sequences in which motion is slow from frame to frame. As we are dealing with 4-frame per second video sequences, motion can be quite fast. For instance, a car or a bus driving on the street can appear completely in the scene from one frame to the subsequent one. If an object is on the previous frame, motion estimation should find the corresponding pixels to this object on the current image. In such fast motion cases, motion estimation search algorithm cannot find the corresponding pixels to the moving objects as it is just not present in the previous frame. The difference between two subsequent frames is too high. In fast motion cases, motion estimation cannot reconstruct properly the current image.

However, motion frames in such cases show that motion has been detected in the regions where the moving objects have appeared. Thus, feature extraction is possible. Contours surrounding the moving objects can be drawn (Figure 3.10 and Figure 3.11).

Other examples displayed in Figure 3.12 to Figure 3.14 demonstrate that big objects (bus, van, ...) passing through the scene can generate several areas of motions. This causes a redundancy in the feature extraction (Figure 3.12).

Moreover, large objects affect the whole illumination of the scene. False regions of

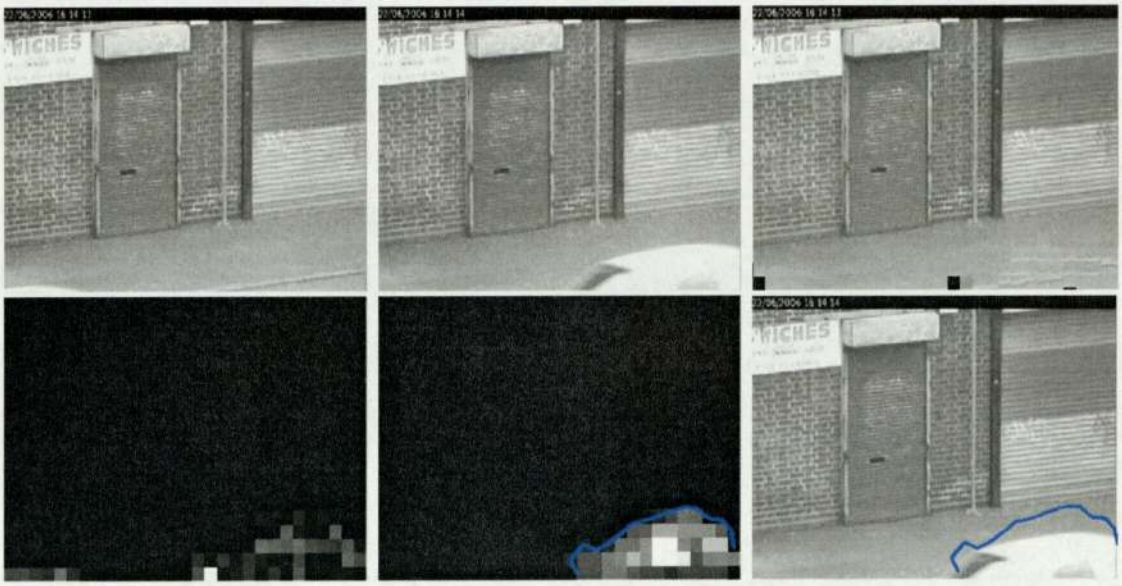

Figure 3.10: Motion estimation of a car. *From the previous frame to the current image, the whole car has appeared in the scene. Too many pixels have moved and motion estimation cannot work properly. The car cannot be on the reconstructed frame as it is not on the previous frame. However, the motion frame shows that motion has been detected in the area where the car should be. Thus, moving object extraction is possible.*

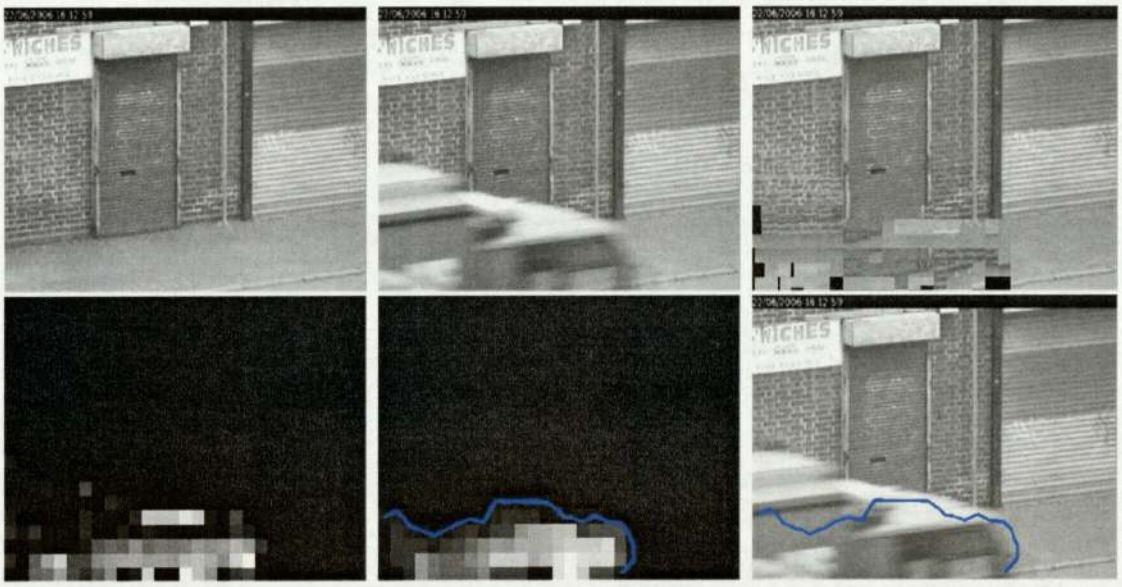

Figure 3.11: Motion estimation of a truck. *The whole truck has appeared in the scene from the previous to the current frame. Like in Figure 3.10, motion estimation reconstruction cannot work properly. However, the motion frame shows that motion has been detected in the area where the truck should be. Moving object extraction is possible.*

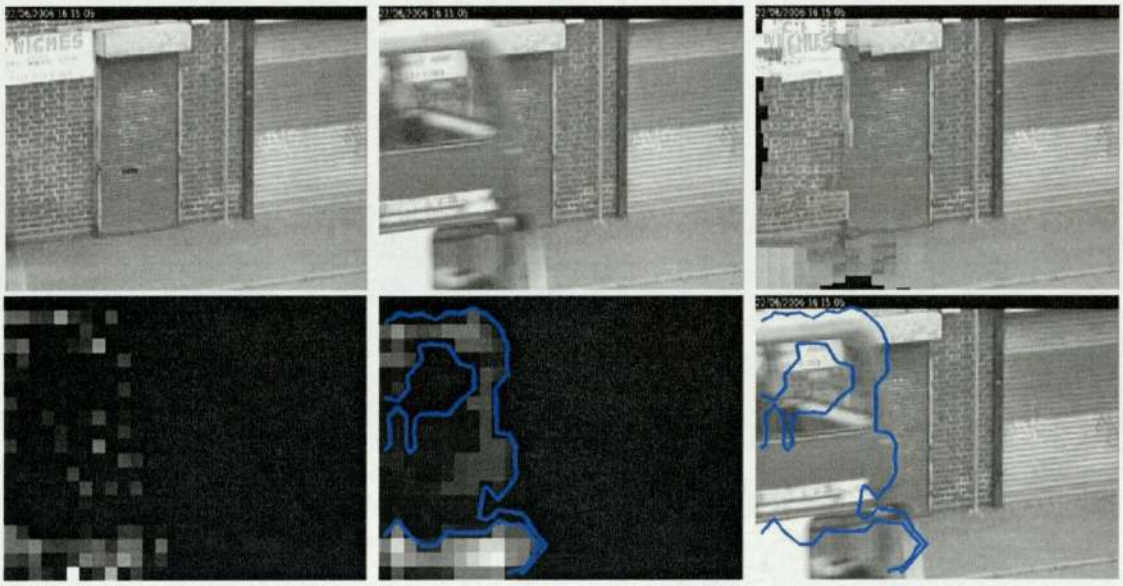

Figure 3.12: Motion estimation of a bus. *Motion has been detected in the region where the bus has appeared. However, several regions of motion are extracted from only one moving object.*

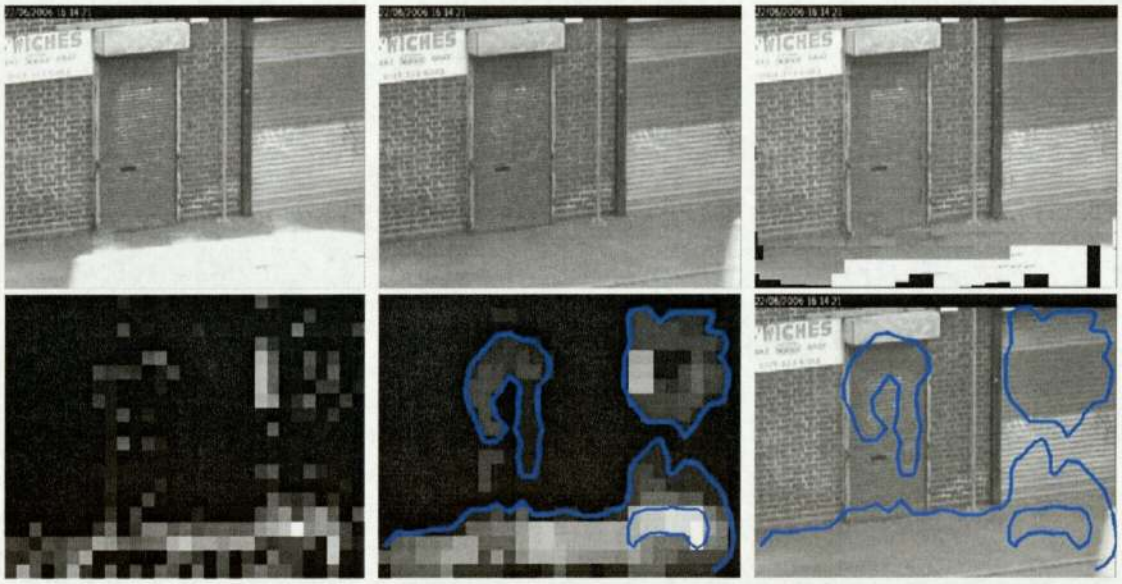

Figure 3.13: Motion estimation of a van. *Motion has been detected in the region where the van has moved. The van has caused an illumination change in the whole scene. From the previous to the current frame, the iron shutters have become darker; this has generated areas of motion as shown on the motion frame. Contours are drawn around these regions. Illumination changes can create false areas of motion.*

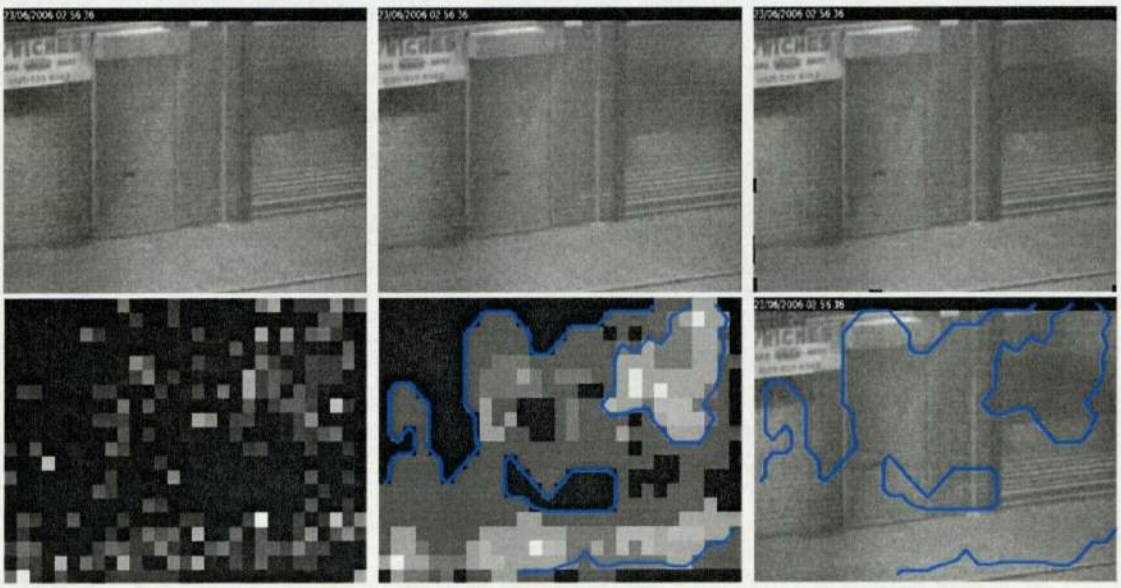

Figure 3.14: Motion estimation at night. *At night, illuminations changes are more likely to occur. They creates false areas of motion.*

motion can then be created (Figure 3.13). This phenomenon is emphasized at night (Figure 3.14).

In conclusion, motion estimation is an efficient tool to detect motion and extract moving objects in the scene. Even if the reconstruction of the image using diamond search motion estimation is just possible for slow motion objects (human walking), feature extraction is still feasible in fast motion cases. Contours can still be drawn around the moving objects. However, a few drawbacks have been enumerated. Redundancy in the feature extraction can be caused by big vehicles generating several areas of motion or illumination changes creating false regions of motion.

In the next chapter, we shall consider the problem of tracking moving objects in a cluttered background. Instead of simply looking for regions of change, a model-based approach will be described.

Chapter 4

Model-Based Vision

The computer vision system we want to build must be able to analyse object shape and motion in real time. To this end, model-based vision approaches [3,4] use mathematical models. Historically, the first mathematical models proposed to fit data, known as deformable models, combined mathematical geometry (splines) and the dynamics of elastic curves. They were known as active contours or snakes, or later, deformable templates, which added harder geometrical constraints to the shapes. On a dynamic point of view, fitting over time is called tracking.

4.1 Spline-space model

In order to construct curves $\mathbf{r}$ in the plane, parametric spline functions can be used:

$$\mathbf{r}(s) = (x(s), y(s)) \qquad s \in [0, L], \tag{4.1}$$

where L is the length of the curve and the coordinate functions x and y are spline functions of the curve parameter s:

$$x(s) = \mathbf{B}(s)^T \mathbf{Q}^x \qquad y(s) = \mathbf{B}(s)^T \mathbf{Q}^y, \tag{4.2}$$

with $\mathbf{B}$ the vector made of the B-spline basis functions [4] and $\mathbf{Q}^x$ and $\mathbf{Q}^y$ the x and y coordinates of the control points of the curve respectively.

Hence, one can define a *spline-space* as follows:

$$\mathbf{r}(s) = \mathbf{U}(s)\mathbf{Q}, \tag{4.3}$$

where

$$\mathbf{U(s)} = \begin{pmatrix} \mathbf{B}(s)^T & \mathbf{0} \\ \mathbf{0} & \mathbf{B}(s)^T \end{pmatrix} \qquad \mathbf{Q} = \begin{pmatrix} \mathbf{Q}^x \\ \mathbf{Q}^y \end{pmatrix}. \tag{4.4}$$

$\mathbf{Q}$ is called a spline-vector. Therefore, in a sline-space a curve $\mathbf{r}$ is defined thanks to a finite number of control points.
The length of $\mathbf{Q}$, N_Q, defines the dimension of the spline-space. This dimension is very high. Indeed, it is twice the number of control points of the curve (Figure 4.1).

4.2 Shape-space model

4.2.1 Definition

In order to speed up the algorithms run on the curves, one may want to define a lower dimensional space to express any curve in the plane. The idea is to no longer consider a curve as a set of control points but as a set of transformations allowed from an initial template curve.

Shape-space is a space parameterising the allowed deformation on the curve. Its dimension N_X is typically considerably smaller than that of a spline-space. A shape-vector $\mathbf{X}$ is a vector of length N_X.

One can mathematically define a spline-space vector $\mathbf{Q}$ from a shape-space vector $\mathbf{X}$ as follows:

$$\mathbf{Q} = \mathbf{W}\mathbf{X} + \mathbf{Q}_0, \tag{4.5}$$

where $\mathbf{W}$ is a $N_Q * N_X$ shape-matrix which creates a linear mapping from shape-space to spline-space and $\mathbf{Q}_0$ is a template curve against which shape variations are measured. $\mathbf{Q}_0$ is generally drawn by hand on the initial frame of the video sequence we are working with (Figure 4.1).

4.2.2 The Space of Euclidean similarities

The Euclidean similarities of a template curve $\mathbf{Q}_0$ (defined in spline-space) are the curves obtained from $\mathbf{Q}_0$ by any translations in the plane, any verical rotations or any scaling in the plane. They form a 4-dimensional shape space with $N_Q * 4$ shape-matrix

$$\mathbf{W} = \begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{Q}_0^x & -\mathbf{Q}_0^y \\ \mathbf{0} & \mathbf{1} & \mathbf{Q}_0^y & \mathbf{Q}_0^x \end{pmatrix} \tag{4.6}$$

where $\mathbf{0} = \begin{pmatrix} 0 & \dots & 0 \end{pmatrix}^T$ and $\mathbf{1} = \begin{pmatrix} 1 & \dots & 1 \end{pmatrix}^T$ are $\frac{N_Q}{2}$-vectors.

The first two columns of W cover the horizontal and vertical translations respectively. The third and fourth columns, obtained from the template curve, govern the rotation and scaling. By varying $\mathbf{X}$, we can define any curve obtained from the initial

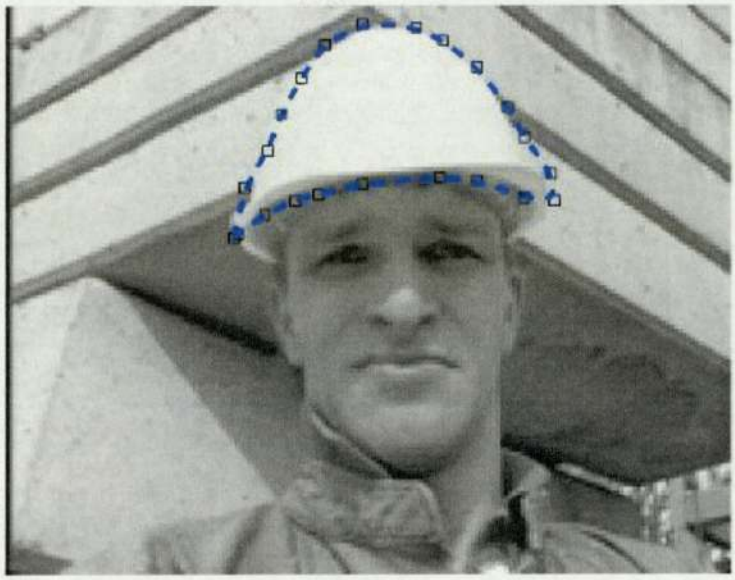

Figure 4.1: Initial template. *The black boxes represent the control points of the interpolated curve (dashed blue line).*

template curve $\mathbf{Q}_0$.
For instance, the initial curve is represented by $\mathbf{X} = \begin{pmatrix} 0 & 0 & 0 & 0 \end{pmatrix}^T$.
More generally, a curve obtained from the template curve by a translation of $\mathbf{T} = \begin{pmatrix} T_x & T_y \end{pmatrix}^T$, a rotation of angle θ and a magnification of scaling factor λ is represented by $\mathbf{X} = \begin{pmatrix} T_x & T_y & \lambda\cos\theta - 1 & \lambda\sin\theta \end{pmatrix}^T$.

More complex shape spaces can be defined. They allow more degrees of freedom to the curves. However, we have decided that translations, rotations and scaling should be enough to be able to characterize most of the human motions in the short range of movement in the application.

4.3 The L_2 - norm

The main point of defining such models is to compare spline curves - in spline-space - or shape-vectors - in shape-space. A definition of a norm in spline-space and shape-space is thus needed. This norm is called the L_2 - norm and is defined as follows:

$$L_2(\mathbf{Q}) = \|\mathbf{Q}\| = \sqrt{\mathbf{Q}^T U \mathbf{Q}} \tag{4.7}$$

$$L_2(\mathbf{X}) = \|\mathbf{X}\| = \sqrt{\mathbf{X}^T H \mathbf{X}} \tag{4.8}$$

where

$$U = \begin{pmatrix} B & \mathbf{0} \\ \mathbf{0} & B \end{pmatrix} \qquad B = \frac{1}{L}\int_0^L \mathbf{B}(s)\mathbf{B}(s)^T\, ds \qquad H = \mathbf{W}^T U \mathbf{W}. \tag{4.9}$$

U (with dimension $N_Q * N_Q$) is called the metric matrix for curves. It is defined in terms of the metric matrix for the B-spline B (with dimension $\frac{N_Q}{2} * \frac{N_Q}{2}$). H (with dimension $N_X * N_X$) is the metric matrix for shape-vectors obtained from spline-space [4].

Using all these newly defined spaces, one can now write a basic deterministic fitting algorithm.

4.4 Deterministic fitting

On the first frame of a video sequence, one can define the initial template curve $\mathbf{r}_0$, choosing by hand its control points $\mathbf{Q}_0$ and using B-spline interpolation as shown in Figure 4.1. This allows us to define the initial shape-vector $\mathbf{X}_0 = \begin{pmatrix} 0 & 0 & 0 & 0 \end{pmatrix}^T$.

Then, on the second frame of the video sequence, starting from the initial template, we would like to find the new position of the hat (Figure 4.2).

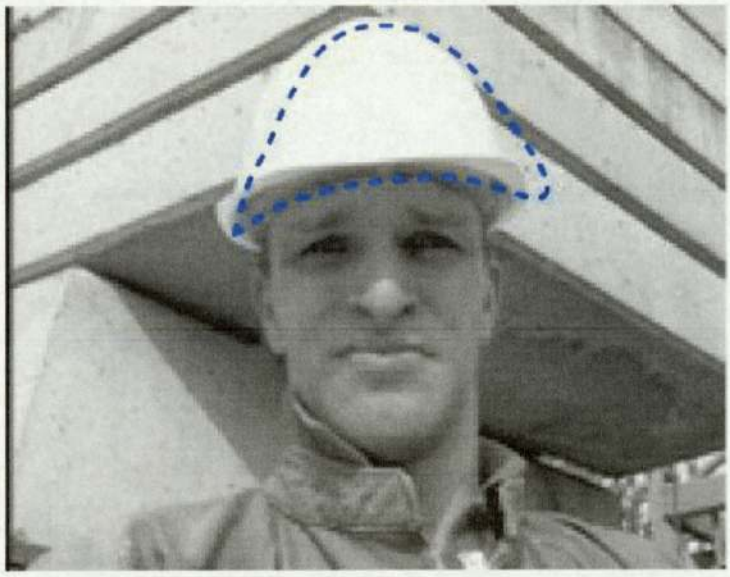

Figure 4.2: Next frame. *The hat has moved from its original position (the dash blue curve).*

4.4.1 Normal image processing

Image-filtering operations are applied along the initial curve $\mathbf{r}_0$. They enable to find the new position of the hat by emphasizing its edges. However, in order to build

an efficient algorithm and to avoid filtering across the entire image, we need to define a search region in which the image feature we are interested in is likely to lie. Image processing can then effectively be restricted to this search area. Forming such a search region by sweeping normal vectors of a chosen length along the initial curve is very efficient (Figure 4.3). Indeed, the search of the edges is done along one-dimensional small lines - as the object does not move far.

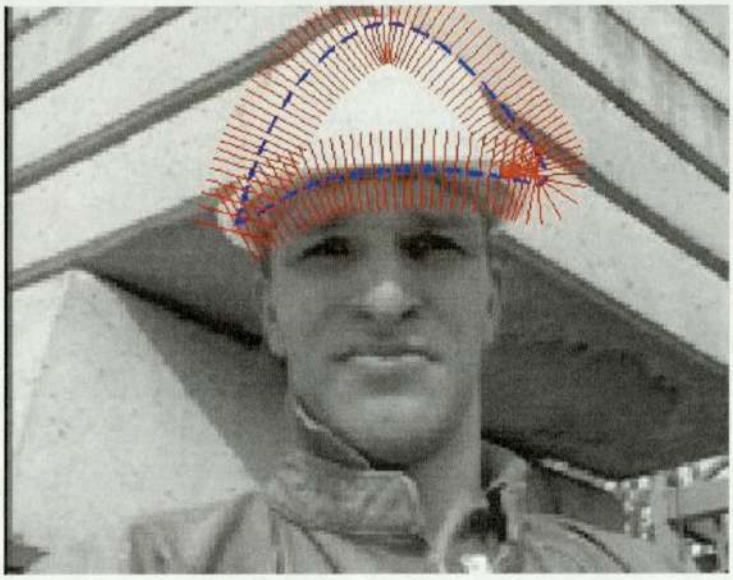

Figure 4.3: Search Region. *Normals along the initial curve enable to define an efficient search region to restrict the image processing.*

Features can then be extracted by performing one-dimensional image filtering along each of the sampled normals. If $s = \{s_1, ..., s_N\}$ is a sample of points lying on the initial curve $\mathbf{r}_0(s)$, this process will give a sequence of sampled points $\{r_f(s_1), ..., r_f(s_N)\}$ which defines the feature curve $\mathbf{r}_f(s)$.

The image intensity is computed along each normal. Then, using a discrete convolution product between the image intensity along the normal and a one-dimension edge detection operator, the feature point is located. It is given by the maximum of the convolution product function (Figure 4.4).

However, it is possible that more than one feature point is found on the normal (more than one maximum value). We assume that such cases are rare and that just one point - the global maximum - is then retained.

Such a process outputs an estimate of the feature curve $\mathbf{r}_f$ as shown in Figure 4.5.

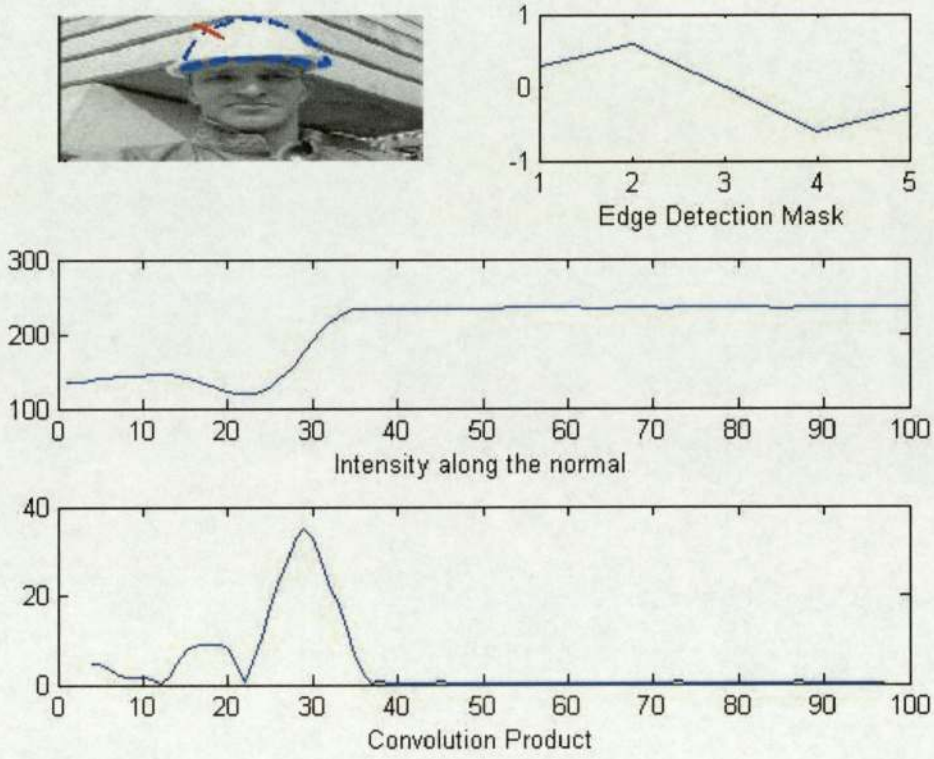

Figure 4.4: Normal image processing. *Normals are constructed at sample points along the initial curve (top left image). Using a convolution product between the one-dimensional edge detection mask (top right graph) and the intensity along the curve, the new position of the edge is given by the maximum of the convolution product.*

4.4.2 Regularisation techniques

Generally, measurements made from images are noisy, for instance due to cluttered backgrounds. We notice on Figure 4.5 that, on some normals, the feature points found by the image processing do not correspond to the edge of the hat but to an edge of the background. Regularisation techniques can procure higher tolerance to noise by biasing the fitted curve toward a mean shape $\overline{\mathbf{r}}(s)$ to a degree determined by a constant α. The minimisation problem can be expressed as

$$\min_{\mathbf{r}(s)} \alpha \|\mathbf{r} - \overline{\mathbf{r}}\|^2 + \|\mathbf{r} - \mathbf{r}_f\|^2 \tag{4.10}$$

The problem can be written more conveniently using shape space as

$$\min_{\mathbf{X}} \alpha \|\mathbf{X} - \overline{\mathbf{X}}\|^2 + \|\mathbf{Q} - \mathbf{Q}_f\|^2 \qquad \text{where} \quad \mathbf{Q} = \mathbf{W}\mathbf{X} + \mathbf{Q}_0. \tag{4.11}$$

$\mathbf{Q}_f$ is the representation of the feature curve in spline-space.

The goal of regularisation is to constrain the fitted curve toward the mean shape $\overline{\mathbf{r}}$. However, this is not satisfactory. Indeed, it may be desirable in practice for $\overline{\mathbf{r}}$ to

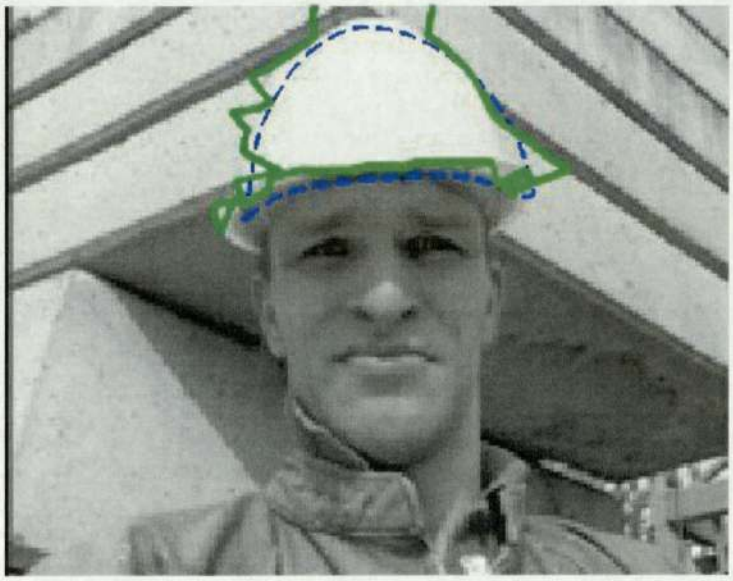

Figure 4.5: Feature curve. *The green feature curve is obtained from the dashed blue initial curve using normal image processing.*

influence the shape of the fitted curve but not its position and orientation. Thus, a better regulariser is needed. It uses a weight matrix $\mathbf{R}$ (positive semi-definite).

$$\min_{\mathbf{X}}(\mathbf{X}-\overline{\mathbf{X}})^T\mathbf{R}(\mathbf{X}-\overline{\mathbf{X}})+\|\mathbf{Q}-\mathbf{Q}_f\|^2 \qquad \text{where} \quad \mathbf{Q}=\mathbf{W}\mathbf{X}+\mathbf{Q}_0 \tag{4.12}$$

To achieve the desired invariance of the regulariser to translation and orientation transformations (the Euclidean similarities), $\mathbf{R}$ must be restricted by means of a projection $\mathbf{p}$ to operate over deformations outside the space of Euclidean similarities.

$$\mathbf{R}=\alpha\mathbf{p}^T H\mathbf{p} \tag{4.13}$$

The projection operator $\mathbf{p}$ can be expressed in term of the shape-matrix $\mathbf{W}$ of the shape-space used

$$\mathbf{p}=\mathbf{I}-(\mathbf{W}^+\mathbf{W})^2, \tag{4.14}$$

where $\mathbf{I}$ is the identity matrix and $\mathbf{W}^+$ is the pseudo-inverse matrix of $\mathbf{W}$.

4.4.3 Deterministic fitting algorithm

Combining the new measurements of the position of the hat (from the feature curve obtained using the normal image processing) with the regularisation techniques described above, one can write a basic recursive deterministic fitting algorithm.

Given an initial shape estimate $\overline{\mathbf{r}}(s)$ ($\overline{\mathbf{X}}$ in shape-space) with normals $\overline{\mathbf{n}}(s)$ and a regularisation weight matrix $\mathbf{R}$, the deterministic algorithm outputs a shape-vector

estimate $\hat{\mathbf{X}}$ of the fitted curve on the subsequent frame (Figure 4.6). A vector of aggregated observations $\mathbf{Z}$ and its associated statistical information matrix $\mathbf{S}$ are also computed (Algo 4.1).

The deterministic fitting algorithm (Algo 4.1)

1. Choose samples along the initial curve $\overline{\mathbf{r}}(s)$: $s = \{s_1, ..., s_N\}$.

2. For each sample s_i, apply the normal image-processing filter passing though $\overline{\mathbf{r}}(s_i)$. It outputs the corresponding feature position $\overline{\mathbf{r}}_f(s_i)$.

3. Initialisation of the aggregated observation vector and its associated statistical information matrix

$$\mathbf{Z}_0 = \mathbf{0}, \qquad \mathbf{S}_0 = \mathbf{0}.$$

4. Iterate, for $i = 1, ..., N$

   - *Normal displacement*

$$\begin{aligned} \nu_i &= (\mathbf{r}_f(s_i) - \overline{\mathbf{r}}(s_i)) \cdot \overline{\mathbf{n}}(s_i), \\ \mathbf{h}(s_i)^T &= \overline{\mathbf{n}}(s_i)^T \begin{pmatrix} \mathbf{B}(s_i) & \mathbf{0} \\ \mathbf{0} & \mathbf{B}(s_i) \end{pmatrix} \mathbf{W}. \end{aligned}$$

   - *Update the aggregated observation vector and its associated statistical matrix*

$$\begin{aligned} \mathbf{S}_i &= \mathbf{S}_{i-1} + \tfrac{1}{N}\mathbf{h}(s_i)\mathbf{h}(s_i)^T, \\ \mathbf{Z}_i &= \mathbf{Z}_{i-1} + \tfrac{1}{N}\mathbf{h}(s_i)\nu_i. \end{aligned}$$

5. Final aggregated observation vector and statistical matrix

$$\mathbf{Z} = \mathbf{Z}_N, \qquad \mathbf{S} = \mathbf{S}_N.$$

6. The fitted shape-vector is given by

$$\hat{\mathbf{X}} = \overline{\mathbf{X}} + (\mathbf{R} + \mathbf{S})^{-1}\mathbf{Z}.$$

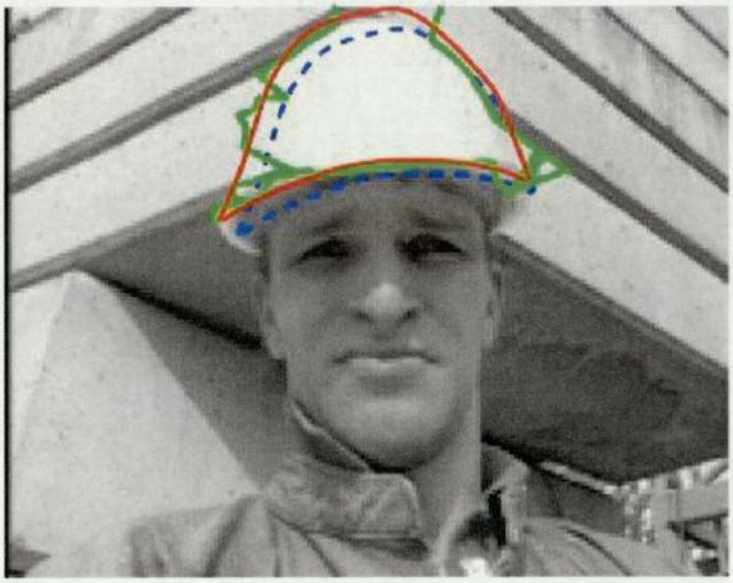

Figure 4.6: Fitted curve. *From the initial dashed blue curve, new measurements of the position of the hat are found using normal image processing (green feature curve). Then, using regularisation techniques, the fitted curve is found (red solid curve).*

4.4.4 Deterministic tracking

Fitting over time is called tracking. Using the previously presented deterministic fitting algorithm, one can obviously write down a first rudimentary tracking algorithm using the fitted curve at time t as the initial estimate of the fitting algorithm at time $t+1$.

Such an algorithm is quite efficient. However, in such a deterministic modelling, tracking becomes prone to divergence due to background clutter (Figure 4.7).

The figure shows 16 frames extracted regularly from a 80-frame video tracking (one every five frames). We note that the black fitted curves correspond well to the actual hat from the first frame to the tenth one (in the 50 first frames of the video sequence). However, after this, the right part of the hat falls into an edge of the background. The consequence is a deformation of the shape that becomes worse and worse until the end of the tracking.

4.5 Probabilistic modelling

In order to get rid of the drawbacks caused by the deterministic modelling previously considered, a new concept of active vision needs to be defined. It uses probabilistic modelling.

Figure 4.7: Deterministic tracking. *The deterministic tracking algorithm is run over a 80-frame video sequence. One frame out of five is displayed. The black curves represent the fitted curves. After the tenth image (the fiftieth frame of the actual video sequence), the fitted curve starts to diverge to the right due to background clutter. This is the main drawback of deterministic tracking.*

4.5.1 Probabilistic model of shape

In the deterministic point of view, a shape-vector $\hat{\mathbf{X}}$ was found to be the best solution to the fitting problem (Algo 4.1). In the probabilistic way of thinking, it is no longer a single value one has to find but a whole probability distribution whose mean will be the fitting solution [4,6].

Hence, one can define, using Bayes' rule, a probabilistic model of shape as follow:

$$P(\mathbf{X}|\mathbf{Z}) \propto P(\mathbf{Z}|\mathbf{X})P(\mathbf{X}), \tag{4.15}$$

where $P(\mathbf{X}|\mathbf{Z})$ is the posterior distribution (the distribution of $\mathbf{X}$ from fitting), $P(\mathbf{Z}|\mathbf{X})$ is the likelihood (the measurement of the observations $\mathbf{Z}$) and $P(\mathbf{X})$ is called the prior (the regularisation term).

4.5.2 Dynamic models

For dynamic models, probabilities need to depend on the previous states. The prior up to time k becomes $P(\mathbf{X}_k|\mathbf{X}_{1..k-1})$ for instance, where $\mathbf{X}_{1..k-1} = \{\mathbf{X}_i, i \in [0, k-1]\}$.

The Bayesian tracking problem is to recursively construct the posterior probability density function (pdf) $P(\mathbf{X}_k|\mathbf{Z}_{1..k})$ given measurements up to time k. In principle, this is done in two steps: prediction and update [6].

Assuming the required pdf $P(\mathbf{X}_{k-1}|\mathbf{Z}_{1..k-1})$ is available, using the Chapman-Kolmogorov equation, the prediction step gives the prior pdf of the state at time k as follows:

$$P(\mathbf{X}_k|\mathbf{Z}_{1..k-1}) = \int P(\mathbf{X}_k|\mathbf{X}_{k-1})P(\mathbf{X}_{k-1}|\mathbf{Z}_{1..k-1})\, dX_{k-1}. \tag{4.16}$$

Moreover, at time k, a measurement $\mathbf{Z}_k$ becomes available and this may be used to update the prior via Bayes' rule:

$$P(\mathbf{X}_k|\mathbf{Z}_{1..k}) = \frac{P(\mathbf{Z}_k|\mathbf{X}_k)P(\mathbf{X}_k|\mathbf{Z}_{1..k-1})}{P(\mathbf{Z}_k|\mathbf{Z}_{1..k-1})}, \tag{4.17}$$

where the normalising constant is:

$$P(\mathbf{Z}_k|\mathbf{Z}_{1..k-1}) = \int P(\mathbf{Z}_k|\mathbf{X}_k)P(\mathbf{X}_k|\mathbf{Z}_{1..k-1})\, dX_k. \tag{4.18}$$

In the update stage, the measurement $\mathbf{Z}_k$ is used to modify the prior density to obtain the required posterior density of the current state.

However, such a propagation of the posterior density cannot always be determined analytically. In those cases other methods such as particle filters [5,6] will be needed. If we assume that the posterior density is a Gaussian and that the state propagation is linear, then the exact optimal solution to the tracking problem is given by the Kalman filter.

4.5.3 Tracking using Kalman filters

First-order Auto-regressive processes

The evolution of the Gaussian density for the state of the tracked object is propagated thanks to the Kalman filter. Moreover, the propagation of the Gaussian prior distribution from time $t-1$ to time t is assumed to follow a first order auto-regressive process

$$\mathbf{X}(t) = \mathbf{A}\mathbf{X}(t-1) + \mathbf{D} + \mathbf{B}\mathbf{w}, \tag{4.19}$$

where $\mathbf{w}$ is randomly chosen from a distribution $N(0,1)$ and $\mathbf{D}$ is the mean displacement in each time step.

$\mathbf{A}$, $\mathbf{B}$ and $\mathbf{D}$ have been chosen to reflect translational motions in shape-space [4]:

$$\mathbf{A} = \mathbf{I}, \qquad \mathbf{B} = b_0\mathbf{H}^{-1/2}, \qquad \mathbf{D} = \mathbf{0} \tag{4.20}$$

with $b_0 = \gamma_0\tau_0^{3/2}$. γ_0 is the rate of growth of the moving object over time ($\gamma_0 = 35\ pixel.s^{-3/2}$). τ_0 is the interval of time between two consecutive frames of the video sequence ($\tau_0 = 0.25\ s$).

The prior distribution can be written

$$P(\mathbf{X}(t)|\mathbf{X}(t-1)) \propto \exp\{-\frac{1}{2}\|\mathbf{B}^{-1}(\mathbf{X}(t) - \mathbf{A}\mathbf{X}(t-1) - \mathbf{D})\|^2\}. \tag{4.21}$$

As we are dealing with Gaussian distributions, we just need to know their mean and covariance matrix to entirely define them. Then, since by definition $\hat{\mathbf{X}}(t) = E[\mathbf{X}(t)]$ and $\mathbf{P}(t) = Var[\mathbf{X}(t)]$, we can write down the mean-state and the covariance equations:

$$\hat{\mathbf{X}}(t) = \mathbf{A}\hat{\mathbf{X}}(t-1) + \mathbf{D}, \tag{4.22}$$

$$\mathbf{P}(t) = \mathbf{A}\mathbf{P}(t-1)\mathbf{A}^T + \mathbf{B}\mathbf{B}^T. \tag{4.23}$$

Tracking using first-order Kalman filter (Algo 4.2)

The propagation over one-step using first-order Kalman filter works in three steps [4]:

1. *Prediction step.*
   The mean-state and covariance equations (4.22 and 4.23) are used in this step to obtain the predicted state shape-vector $\tilde{\mathbf{X}}(t)$ and the predicted covariance matrix $\tilde{\mathbf{P}}(t)$.

   $$\tilde{\mathbf{X}}(\mathbf{t}) = \mathbf{A}\mathbf{X}(t-1) + \mathbf{D}, \qquad \tilde{\mathbf{P}}(t) = \mathbf{A}\mathbf{P}(t-1)\mathbf{A}^T + \mathbf{B}\mathbf{B}^T. \tag{4.24}$$

2. *New measurements.*
   New measurements are then found using the fitting algorithm previously presented (Algo 4.1) using $\tilde{\mathbf{X}}(t)$ and $\tilde{\mathbf{P}}(t)$ as inputs. The aggregated observation vector $\mathbf{Z}(t)$ and its corresponding statistical information matrix $\mathbf{S}(t)$ are obtained.

3. *Assimilation step.*
   The assimilation step is based on a Kalman gain $\mathbf{K}(t)$.

   $$\mathbf{K}(t) = \tilde{\mathbf{P}}(t)[\mathbf{S}(t)\tilde{\mathbf{P}}(t) + \mathbf{I}]^{-1} \tag{4.25}$$

   A new estimated value of the fitted shape-vector $\hat{\mathbf{X}}(t)$ and the new covariance matrix $\mathbf{P}(t)$ are computed thanks to this Kalman gain.

   $$\hat{\mathbf{X}}(t) = \tilde{\mathbf{X}}(t) + \mathbf{K}(t)\mathbf{Z}(t), \qquad \mathbf{P}(t) = \tilde{\mathbf{P}}(t) - \mathbf{K}(t)\mathbf{S}(t)\tilde{\mathbf{P}}(t). \tag{4.26}$$

   These two values will be used as the inputs of the prediction step of the propagation from time t to time $t+1$.

Experiments and Results

An example of probabilistic tracking using such a first-order Kalman propagation is presented in Figure 4.8. The algorithm is run on the same 80-frame video sequence as the deterministic tracking in Figure 4.7. One frame of every five is displayed. Black curves still represent the fitted curves. If we compare the results obtained from the deterministic algorithm in Figure 4.7 and this probabilistic result, we note that the fitted curve no longer diverges and falls into background edges after the tenth image. Such a probabilistic tracking using a first-order Kalman filter is more robust and less prone to divergence due to the background than deterministic tracking.

Figure 4.8: Probabilistic tracking using a Kalman filter. *The probabilistic tracking algorithm using a first-order Kalman filter is run over a 80-frame video sequence. One frame out of five is displayed. The black curves represent the fitted curves, which follow quite well the actual hat over the whole sequence. Such a probabilistic tracking is more robust than the deterministic one.*

To have in mind an idea of the performance of this algorithm, run in Matlab, such an 80-frame tracking lasts 52 seconds.

Chapter 5

Preliminary Image Processing

The image data provided is cluttered. Some preliminary image processes must be applied to the frames of the video sequences in order to simplify the data on which the algorithms are run.

5.1 Background Subtraction

Background subtraction is a very common tool to separate moving objects from their backgrounds. It suppresses background features to prevent them from distracting the fitting and tracking algorithms. Moreover, the background of the data provided is largely stationary. Therefore, this approach is suitable.

Let us call $\mathbf{Bck}(x, y)$ the image of the background and $\mathbf{I}(x, y)$ the current frame. Given a noise threshold σ, the background subtracted image $\mathbf{BS}(x, y)$ is constructed as follows [4,7]:

$$\mathbf{BS}(x,y) = \begin{cases} 0 & \text{if } |\mathbf{Bck}(x,y) - \mathbf{I}(x,y)| \leq \sigma \\ |\mathbf{Bck}(x,y) - \mathbf{I}(x,y)| & \text{otherwise} \end{cases} \tag{5.1}$$

Figure 5.1 shows three examples of background subtractions. The moving objects evolving on the scene (a bus, a car and a woman) are separated from the background as expected.

5.2 Dynamic background update

The camera films the scene 24 hours a day. The illumination changes as time passes during the day. Moreover, objects can stop in front of the door during a certain period of time (cars, buses). Such things affect the definition of the background which needs to be updated over time to be adapted to the current situation.

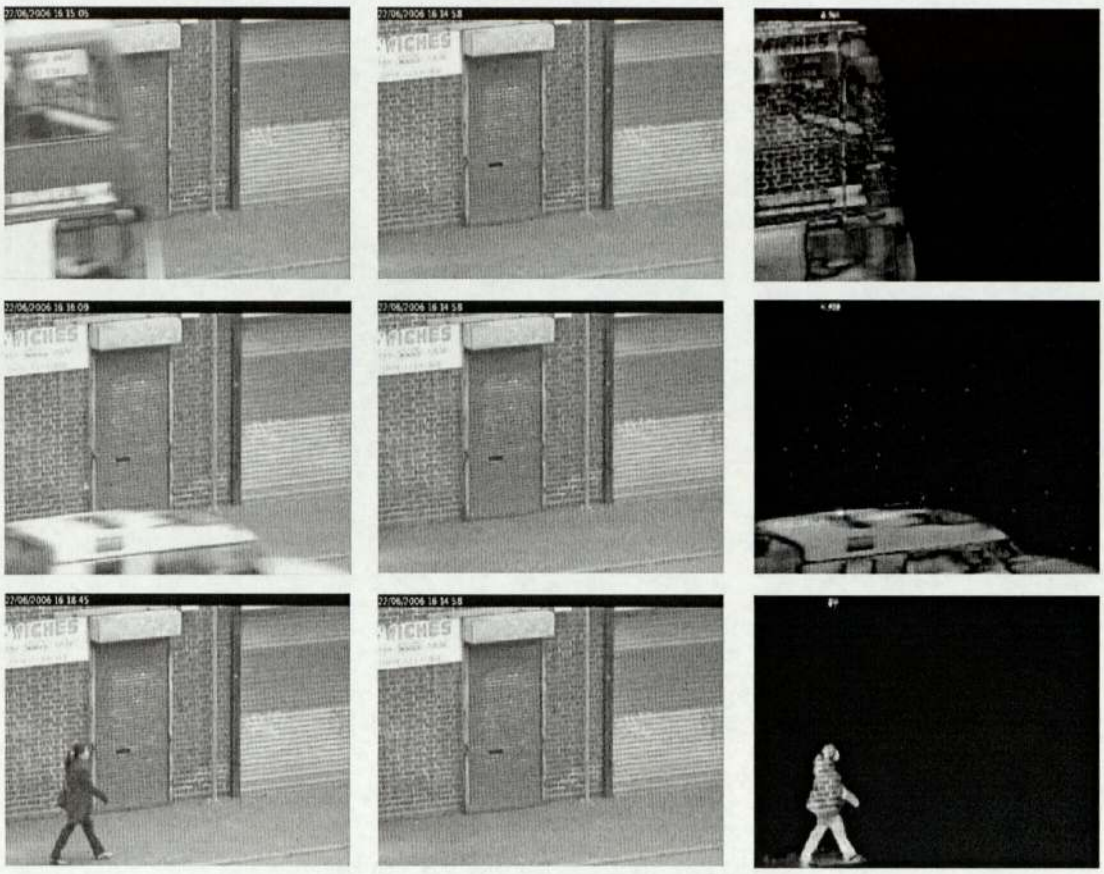

Figure 5.1: Background subtractions. *Three examples of background subtraction are shown. we have chosen the noise threshold to be $\sigma = 10$ which is around 5% of the maximum intensity value (256). The first column is a set of three images of the video in which events are occurring (a bus and a car driving, a woman walking). The second column shows the background of the scene. Images resulting from background subtraction between these two first columns are displayed in the third column. The moving objects are highlighted.*

A new background is defined as soon as two consecutive frames are identical enough. Effectively, we build the subtracted image between the two frames. When the number of non-zero pixels is lower than a minimum tolerance corresponding to the minimum size of detection for moving objects (see section 6.1.1), the latest frame becomes the new background.

Finally, to prevent from having to update the background at every stage, we define a minimum time step that represents the period of time starting from the latest background update during which the background cannot be updated. We have chosen this time step to be 10 seconds; i.e. 40 frames as we are dealing with 4-frame-per-second video sequences.

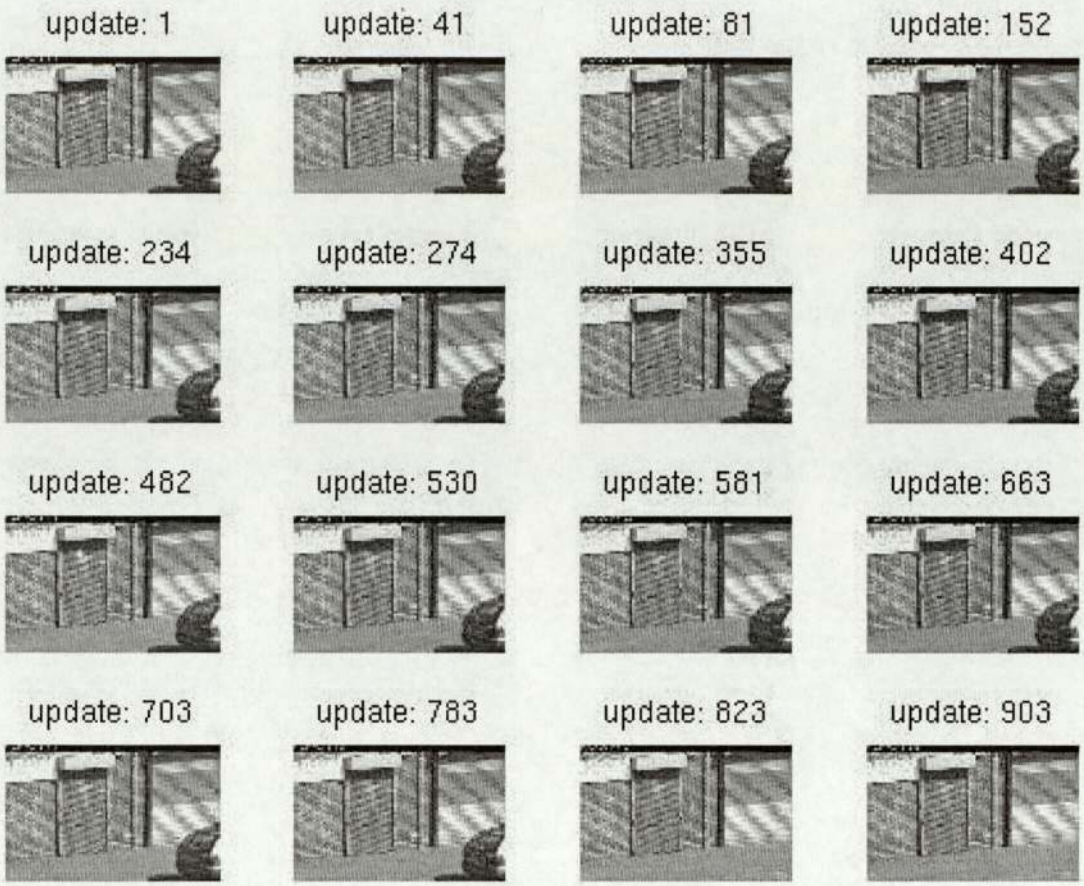

Figure 5.2: Dynamic background update. *The algorithm is run over a 1000-frame video sequence. The successive backgrounds are displayed. The numbers at the top of the images correspond to the times of update (in frames). The difference between two updates varies from 40 (the minimum time step) to more than 80, depending on the number of events occurring in the scene.*

Figure 5.2 illustrates the dynamic background update. The algorithm is run over a 1000-frame video sequence. The successive backgrounds are shown. The number at the top of each image corresponds to the time (in frames) of the background update. We note that the difference between two consecutive time updates is always greater than 40 (the minimum time step defined above). This difference can be quite large (up to 82 between the fourth and the fifth background). This is due to several events occurring in the scene in the same period of time.

These preliminary image processes were applied to the cluttered video data provided. They were useful to separate the moving objects from the background for instance. They will also be needed to detect and then classify the events occurring in the scene (Chapter 6).

Chapter 6

Detection and Classification of Events

The main goal of the project is first to be able to detect, among the continuous flow of video data, every event occurring in the scene, and then to classify whether these events have been triggered off by a human being or not.The objective is to detect human triggers only (i.e. improve the false positive rate).

6.1 Detection of Events

The first step to build such a human classifier is the detection of every event occurring on the video.

An event can be defined as a sudden change in the scene. It can be due to a person walking on the pavement, a bus or a car driving on the street, a door opening, an illumination change.

6.1.1 Minimum size detection

Using the previously described background subtraction and dynamic background update (Chapter 4) that define a model for the background of the scene at any time, one can easily highlight the sudden changes (the events) in the video sequence by just subtracting the current frame from its corresponding background.

Frames resulting from such an operation are mainly made up of black pixels except for the regions in which noticeable changes have been observed (see Figure 5.1). Hence, such an operation enables the detection of events.

We need to add more constraints and details to this basic idea to actually build an event detection algorithm.

Minimum size detection

The purpose of our detection module is to be able to register every relevant change in the video. Events creating too small changes, because they are occurring on the borders of the scene or because they are actually triggered off by too small objects such as a cat or a small dog, should not be taken into account.

To avoid those irrelevant events, we can use a minimum size threshold of detection. This threshold represents the number of pixels that are not equal to zero in the background subtracted frame. It represents the actual size of the detected event.

To have an idea of the size of an object on the video data provided (of resolution $240 * 320$ pixels), note that a moving human being generally affects more than 1500 pixels. However, this will depend on the distance from the camera. To make sure we do not miss any relevant trigger, we have decided to take a lower threshold of 500.

Window of interest

Relevant events are more likely to occur in the centre of the scene (which should be centered around the door). To support this, we define a window of interest on the video. This region is the only area in which an event will be detected.

To make the algorithm general, we have chosen to define the same window of interest for any scene: the vertically centered half shown by the region between the two dashed lines in Figure 6.1.

However, one can easily define by hand the region of interest for a particular scene in order to localise detection.

Figure 6.1: Window of interest.

The definition of such a window also enables the minimum size detection previously presented to work. Indeed, if we considered the whole scene as the window of interest, as soon as an object entered the scene, an event would be detected. Suppose that this event is coming from a border of the scene. When it is detected, we cannot see the whole of the object but just a tiny part of it (bigger then the minimum threshold). We do not have any idea of how big it really is. It also affects our knowledge of shape if only a small part of the object is detected.

Centering the window of interest enables to get rid of these border issues and to estimate the true size and the shape of the objects when they enter the detection area.

Redundancy and Slow motion

At this point, as soon as the size of the event occurring into the window of interest is larger than the minimum size of detection, a trigger is given. For instance, that means that a man (bigger than the minimum threshold) crossing the window of interest during 5 seconds creates around 5 times $4 = 20$ events. Thus, there is a huge redundancy among the detected events.

In order to avoid it, we no longer only check the size of the moving objects but we also take into account the variation of this size over time. Effectively, we store the size values for the past second - to allow slow moving objects to be detected and to let them enter the area of detection. Then, we check if the current value generates an increase greater than 100% compared with the previous values: i.e., if the size of the change has more than doubled over the past second. If and only if this is the case, then an event is detected. Basically, with such a system, an object is just detected on entering the window of interest because it affects a lot the size of change (an increase of more than 100%). However, once the object is fully into the region of interest, the size of change does not vary a lot anymore and no new event is detected. An example of such a detection algorithm is presented on Figure 6.2.

Four events are detected: first a car and a van entering the region of interest, then a human walking. Finally, as soon as a human is leaving the area of detection, a car is driving on the street. It is also detected.

We note that this algorithm allows multiple detection of events. Even if a moving object is already in the scene, if another object enters the region of interest, it is also detected.

However, one drawback of using background subtraction is that the background can affect the shape of the moving object. This can cause unexpected variations of its size, and thus some redundant events as shown in Figure 6.3.

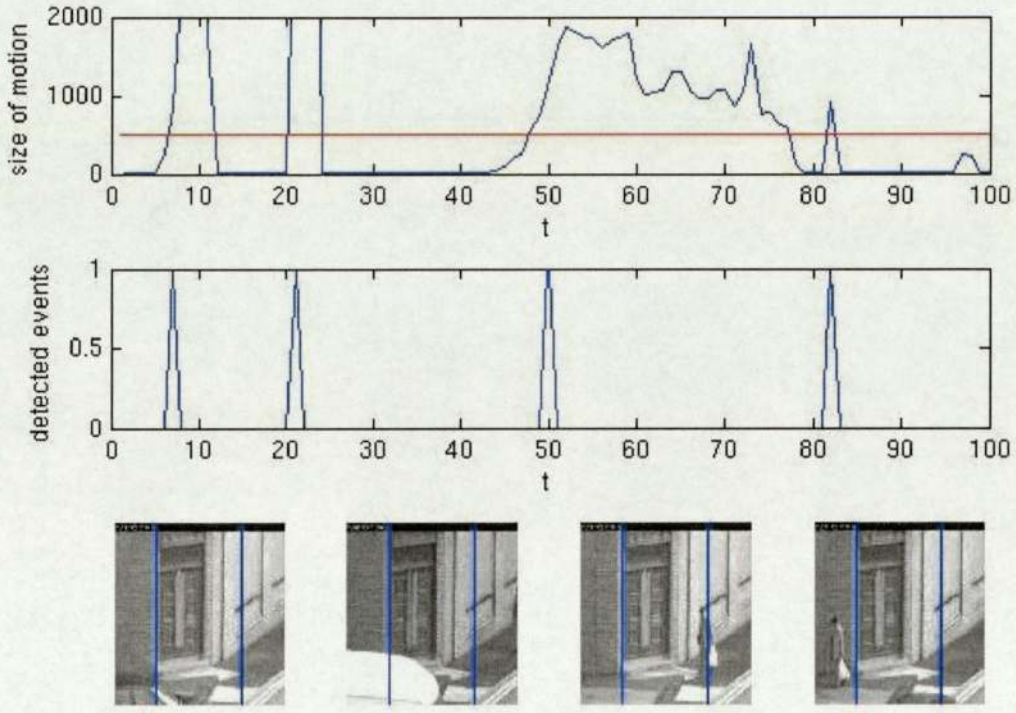

Figure 6.2: New detection algorithm to cancel detection redundancies. *The evolution of the number of non-zero pixels on the background subtracted frame is presented on the top graph. The red horizontal line represents the minimum size threshold. The values of 1 on the second graph represent the detected events. There are four, which are shown in the four images at the bottom.*

This figure shows the evolution of the size of motion and the actual detected events on the two top graphs as before. Moreover, the bottom pictures are key frames (first row) and their corresponding background subtracted frames (second row) taken from the video sequence.

The man enters the window of interest on the left image. A first event is detected. But, on seeing the background subtracted frames of the first two images, we note that the human shape is not clearly defined. As soon as the man walks in front of the white sign (third image), the contrast becomes higher. Thus, the shape becomes more visible. Its size increases and another event is unexpectedly detected because of the background. The contrast stays high until the man passes the door (fifth frame). Then, as we can see on the corresponding background subtracted frames, the shape becomes vaguer and its size decreases (top graph) until the man goes out of the window of interest (sixth and last image).

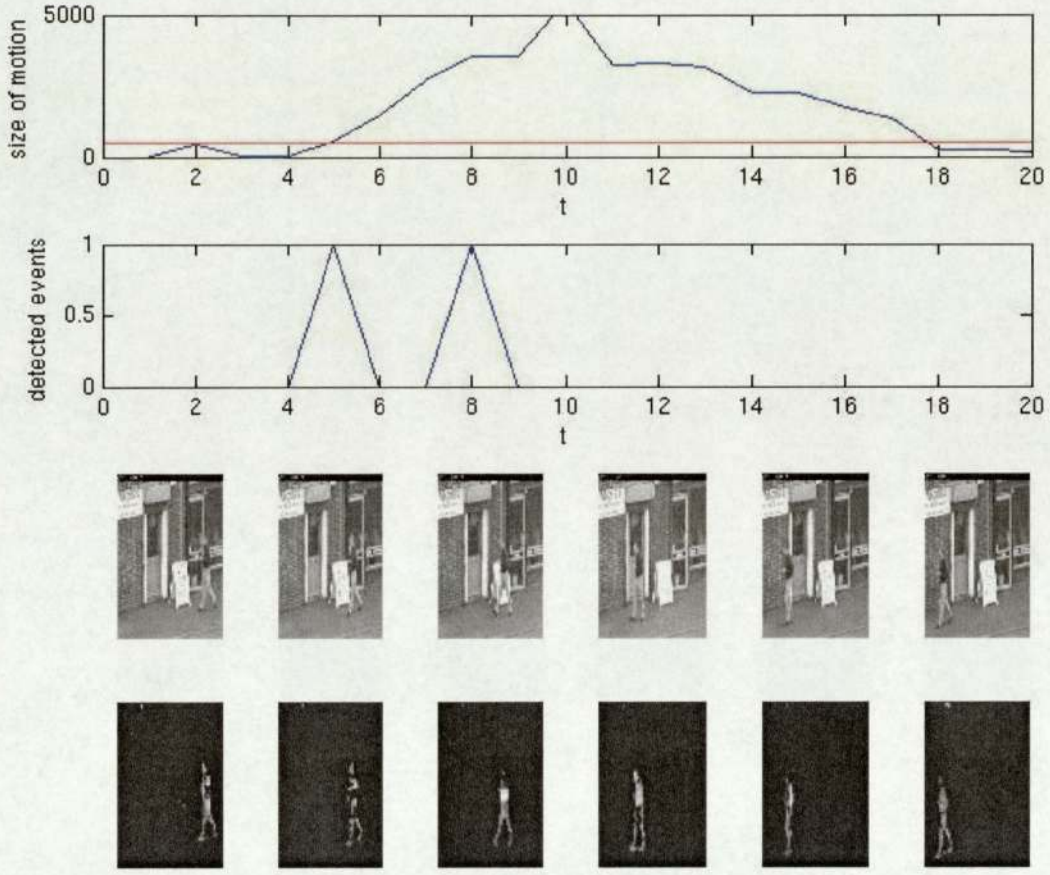

Figure 6.3: Redundancy due to the background. *The evolution of the number of non-zero pixels on the background subtracted frame is presented on the top graph. The values of 1 on the second graph represent the detected events. The bottom pictures are key frames (first row) and their corresponding background subtracted frames (second row) sampled from the video sequence. The background affects the apparent shape of the moving man while he is walking on the pavement. Hence, two triggers are given.*

6.1.2 Extraction of events using Motion Estimation

The detection step allows us to consider frames in which at least one event occurs. However, several events can occur at or almost at the same time and need to be distinguished by the detection algorithm. Indeed, if a human being walks through the scene while a bus is driving on the street, we must be able to detect both events, even if the bus is the bigger object of the two and thus the more likely to be detected.

Moreover, from the shape classification point of view (which will be explained in detail in section 6.2), if the detection algorithm outputs a single event in the person / bus situation, it will become extremely difficult to extract and take into account the shape of the human being as they are smaller than the bus. It will then become very

difficult to classify properly events triggered off by human beings from the others.

Hence, after having detected every event occurring on the scene, we need to extract them separately. This will allow us to deal with multiple events occurring at the same time and will enable the classification of events.

To do so, we can use feature extraction based on motion estimation as presented in section 3.3. As soon as some event is detected in a frame of the video sequence, we run the diamond motion estimation algorithm (see section 3.2) between this frame and the previous one. We can then extract different areas of motion corresponding to the different moving objects evolving on the scene. To be more precise, the feature extraction module gives the boundaries of the moving objects as outputs. From these contours, we define the areas of motion as the smallest rectangles surrounding the contours as shown on Figure 6.4. Non-rectangular areas of motion could have been considered, they are part of the future work.

We note that the motion algorithm detects the two moving objects: the man and the car. However, a third area of motion is detected in the window in the top right corner of the image. This is due to the change of illumination caused by the car. Such events were presented as drawbacks of the motion estimation algorithm in Chapter 3. They are quite common but will be easily discarded thanks to the classification step in section 6.2.

Finally, the event detection module is the combination of the minimum size algorithm and the event extraction using motion estimation. It allows us to detect and isolate every single event occurring in the video scene. Those detected events need now to be classified.

The Event Detection Algorithm (Algo 6.1)

1. Initialisation

   - *Initialisation of the previous and current frames*
     previousframe = NULL;
     currentframe = ReadFrame(Video, 1);
   - *History of the size of motion during 1 second*
     Hist = $[0, 0, 0, 0]$;

2. *While the video sequence is not finished*
   for (i = 2...NbFrame)

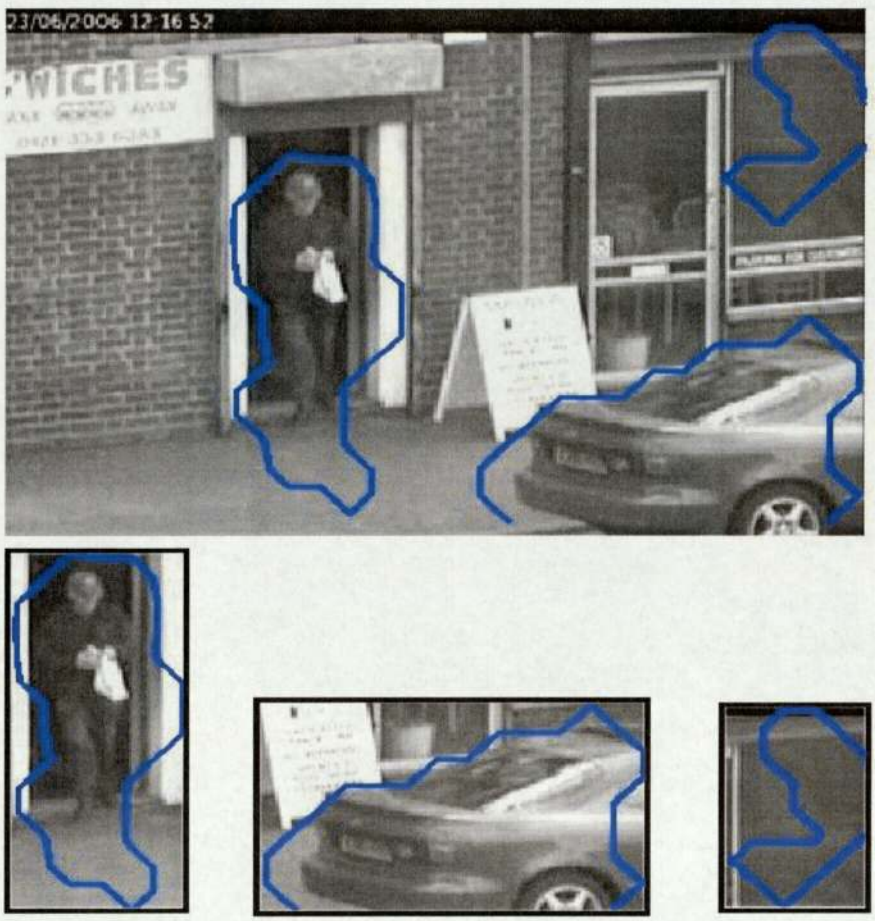

Figure 6.4: Event extraction using motion estimation. *Three rectangular areas of motion (bottom images) have been detected by the motion estimation feature extraction module. Blue lines represent the boundaries of moving object found by the motion estimation algorithm.*

- *Current and previous frame updates*
  previousframe = currentframe;
  currentframe = ReadFrame(Video, i);
- *Dynamic background update*
  UpdateBackground(previousframe, currentframe, σ, MinTimeStep);
- *Minimum size event detection avoiding redundancy* (6.1.1)
  DetectEvent(currentframe, MinThreshold, Hist);
- *Update the one-second historic* Hist
- *If an event has been detected* (6.1.2)
  - *Diamond search motion estimation*
    MotionEstimation(previousframe, currentframe);

- *Extraction of the events from the motion frame*
  BuildContours(MotionFrame);
- *Build rectangular areas of motion around the contours extracted*

end

end

Diamond search motion estimation takes around ten seconds to run. Thus, depending on the number of events detected during a ten-minute video sequence, the time of process of this event detection algorithm can vary a lot (from eight minutes to forty minutes).

Without the event extraction module (6.1.2), for a ten-minute video sequence, the minimum size event detection algorithm (6.1.1) lasts from five to six minutes, depending on the number of events detected.

6.2 Shape Classification

Once an event has been detected, we need to decide if it has been triggered by a human being, several human beings or by any other objects. We need to find a way to distinguish human beings from buses, cars, illumination changes, etc.

6.2.1 Edge Direction Histogram (EDH)

Edges in images constitute an important feature to represent their contents and shapes. Edge detection in an image can significantly filter out noise and useless information while preserving its important structural properties.

A. Jain , H. Zhang and A. Vailaya [8] introduced the edge direction histogram (EDH) in 1998. This method finds the image edges and groups them depending on their orientation.

After explaining how to construct such histograms, we will show how their feature can effectively represent the shape information of an object and be a useful tool to build a human classifier.

The algorithm for generating the EDH consists of four steps as follows [9]:

1. *Edge Detection*
   The Sobel operator is a well-known edge detection algorithm. It computes the

gradient image and generates two edge components at every pixel: a horizontal one (G_x) and a vertical one (G_y). Sobel convolution masks are presented in (6.1):

$$\mathbf{G}_X = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}. \qquad \mathbf{G}_Y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}. \tag{6.1}$$

The amplitude and edge orientation are then computed as follows:

$$amp_G = \sqrt{G_x^2 + G_y^2} \tag{6.2}$$

$$\Phi_G = \arctan(\frac{G_y}{G_x}) \tag{6.3}$$

2. *Finding prominent edges*
   This step extracts the prominent edges of the gradient image. They are extracted by comparing the edge amplitudes with a minimum threshold value T. We have chosen $T = 25$, which is approximatively 10% of the maximum intensity value (256). If the amplitude of the edge at a pixel is lower than T, then it is neglected.

3. *Edge orientation quantization*
   This step quantizes the prominent edges uniformly depending on their orientations into n segments equal to five degrees from 0 to 180: $\Phi_{G1}, ..., \Phi_{Gn}$.

4. *Computing elements of EDH*
   To finish and actually build the EDH, we just need to count the number of elements in every segment of quantization.

6.2.2 Examples of EDH

In order to justify how an EDH can be used as a tool to build a human classifier, we show some results for different detected events.

EDH on the real data

EDH can be directly applied to the rectangular areas extracted by the detection algorithm. Figure 6.5 and Figure 6.6 show several examples of EDHs generated from different detected events.

Each bar of the histogram represents a five-degree segment from 0 to 180. The first and the last ones correspond to the horizontal edges whereas the middle ones represent the vertical edges.

Figure 6.5 presents several EDHs of non-human events (a bus, a car, a van and an illumination change at night). As we are constructing the EDH directly from the real

data, edges from the background are also taken into account in the EDH. We note that for these events, edges are more likely to be horizontal than in any other direction. Moreover, the maximum of the EDH is reached for a value always higher than 200, up to 800 in the case of the bus. This is a measure of the size of the moving object.

On another hand, Figure 6.6 shows three EDHs of human beings walking in the scene. These three EDHs present a similar feature. Edges are more distributed over the whole range of directions. There are still more vertical and horizontal edges than in the other directions. Moreover, the size of the moving objects is generally smaller than in the previous cases.

EDH on the background subtracted data

In order to get rid of useless information contained in the background, EDH can also be applied to the background subtracted data.

EDHs are generated on the same set of examples (Figure 6.7 and Figure 6.8). The only difference is that the background is subtracted from the rectangular regions extracted by the event detection algorithm.

Vehicles (the bus, the van and the car) create EDHs in which horizontal edges are still predominant (Figure 6.7). However, as the background affects the shape of the moving objects, their edges can also be affected. That is why the second event in Figure 6.7 generates such a small EDH (its maximum is lower than 10). Illumination changes are almost completely canceled thanks to the background subtraction. The corresponding EDH is almost null.

On the other hand, human beings generate quite similar EDHs. They are more distributed over the whole directions. Their maximum is reached for the vertical edges - as a man is more likely to be upright. The maximum value is lower than 50.

EDH seems like a good tool to classify human beings from other events. Indeed, in both cases of EDH generated from the real data and from the background-subtracted data, EDHs of non-human events and EDHs of human beings are extremely different. Moreover, EDH generated by human beings present quite similar features.

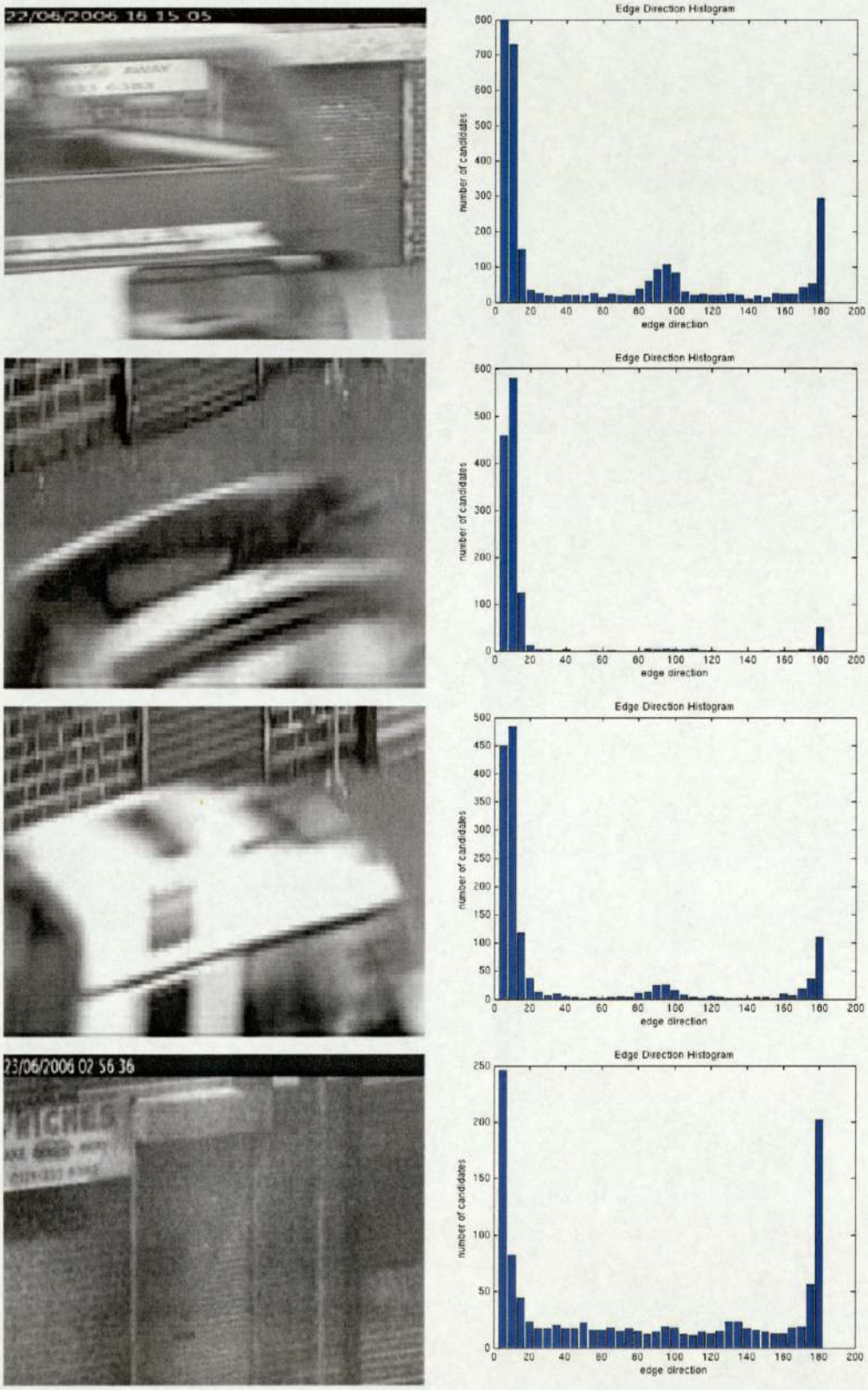

Figure 6.5: EDH applied on real data (1/2).

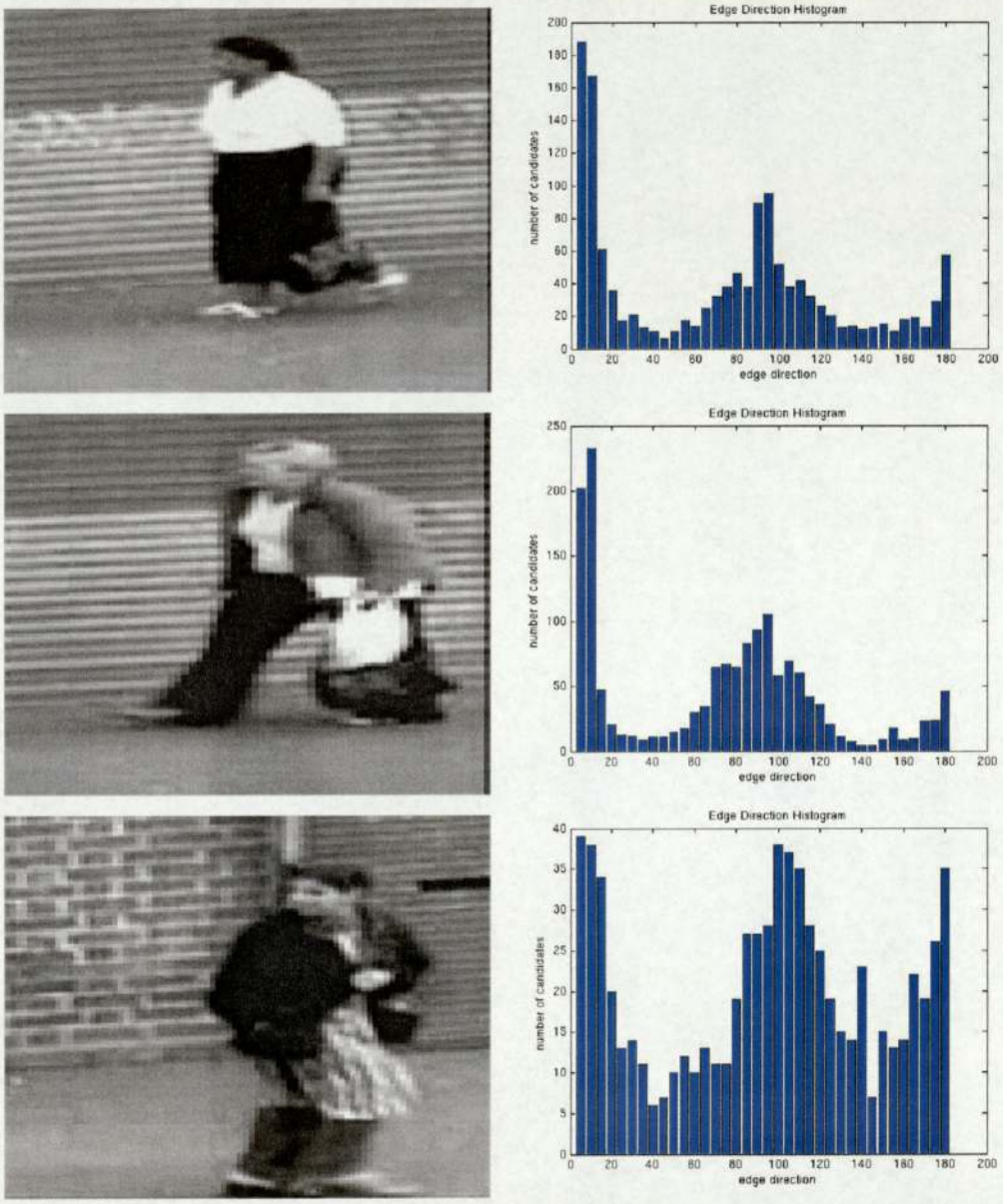

Figure 6.6: EDH applied on real data (2/2).

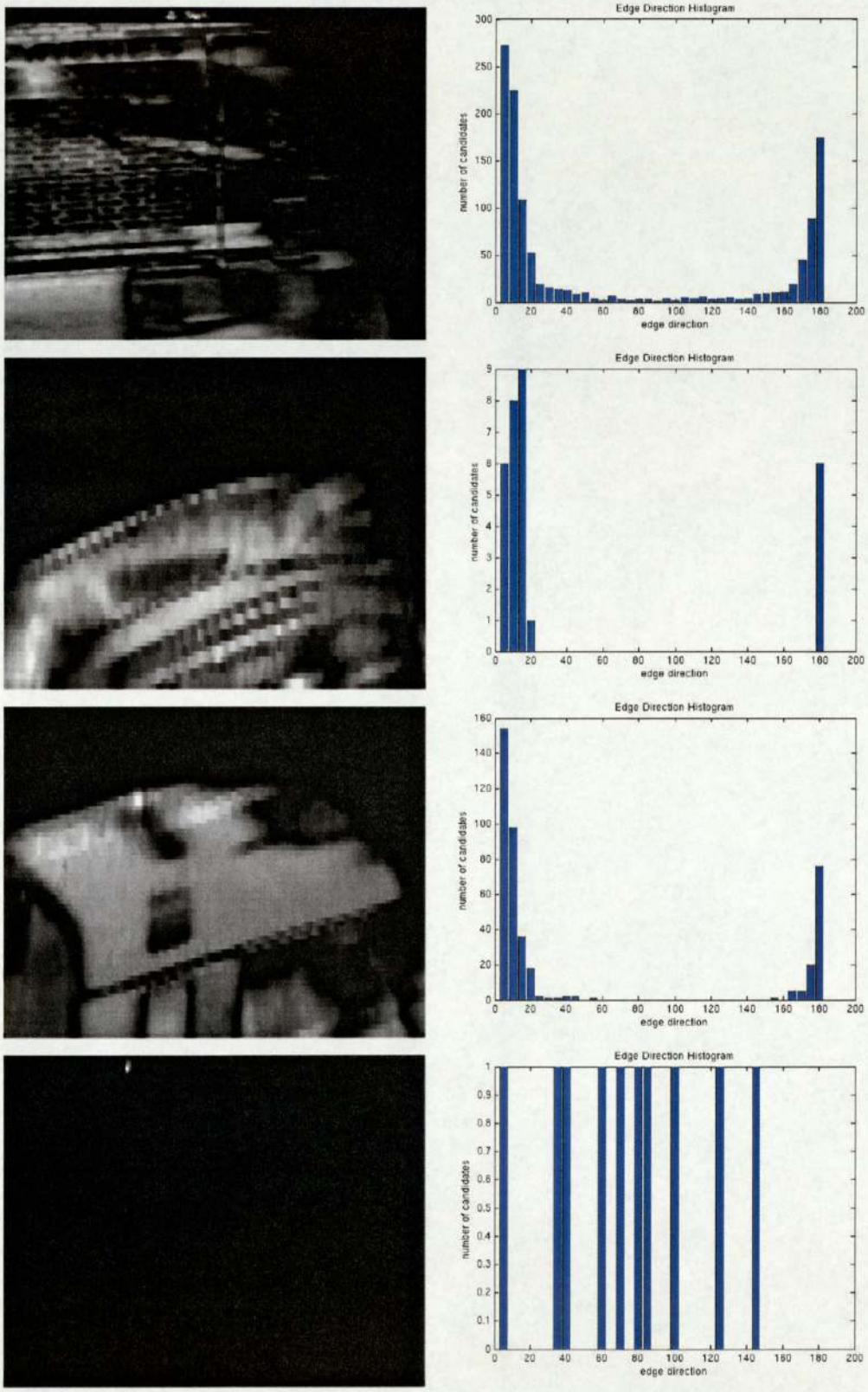

Figure 6.7: EDH applied on background subtracted data (1/2).

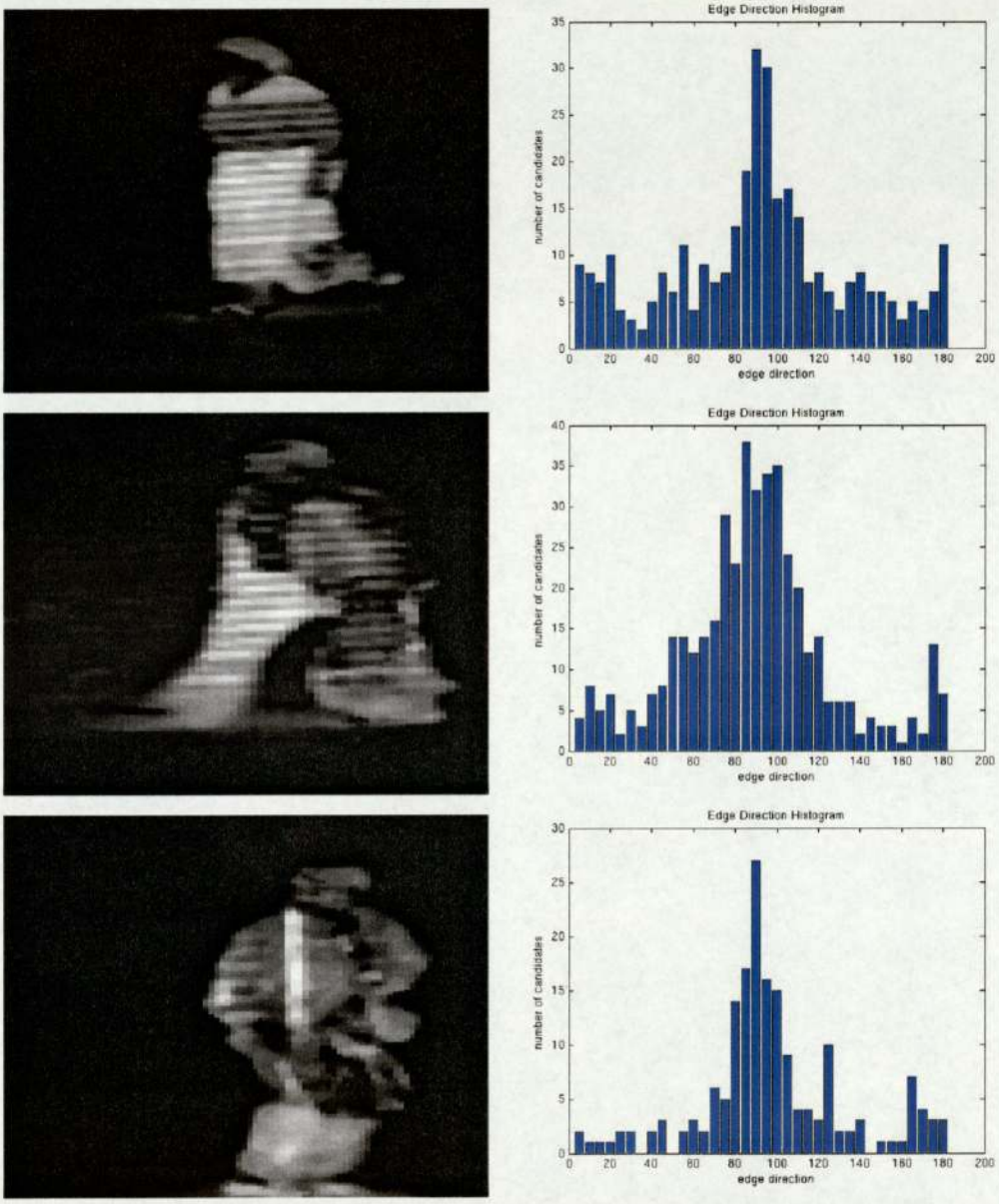

Figure 6.8: EDH applied on background subtracted data (2/2).

6.3 Experiments and Results

The EDH should be a useful tool to classify events triggered by human beings from any other. The examples displayed in section 6.2 were encouraging. We now need to test, on the video data provided, whether EDH may help us to build a human classifier or not.

As soon as an event is detected, we save the EDH generated. This 36-dimensional vector will be the input to our human classifier. One ten-minute video sequence was taken every four hours of a day to build a one-hour training set. Such a training set contains different illumination conditions depending on the time of the day. 584 events have been detected in the training set.

The test set was then built by taking nine ten-minute video sequences at different times of the day. 835 events have been detected in the test set. 39 have been triggered by human beings, 796 by non-human events.

From these data sets, we trained and built two linear classifiers: one for EDH generated directly from the real data and one for the EDH generated from the background subtracted data. We then tested them on the test set. The resulting confusion matrices are displayed in Figure 6.9 respectively on the left and on the right.

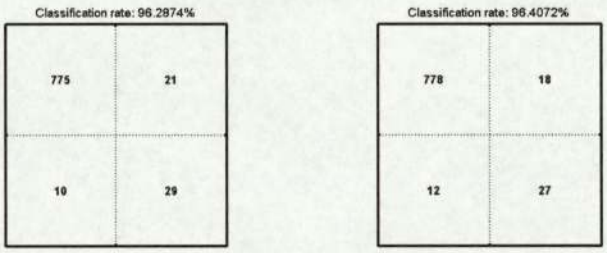

Figure 6.9: Confusion matrices of the linear human classifiers using the test set. *The first row represents the non-human events while the second one corresponds to the human beings detected. Left: Classifier on real data - Right: Classifier on background subtracted data.*

We note that even with such basic linear classifiers, the classification rates are quite high (around 96%). However, the human misclassification rate (bottom left number) is quite high in both cases: 25% and 30%.

These results are very encouraging. Further experiments will be made to try to build a more robust human classifier in Chapter 7.

Chapter 7

Experiments and Results

This chapter presents the experiments we have performed and the results we have obtained from the different tools described in the previous chapters. Motion estimation, model-based vision, detection and classification of events are applied on the real video data to try to solve the issues and achieve the objectives which were set at the beginning of the project.

7.1 Video tracking

Model-based vision uses mathematical models and probabilistic modelling that helped us to build a video tracking algorithm using first-order Kalman filter (Chapter 4). Some results on toy data were presented. We now need to build a video tracking algorithm working on the real data provided.

7.1.1 Video tracking on real data

A probabilistic tracking algorithm using a first-order Kalman filter was run on toy data at the end of Chapter 4. The tracking results obtained (Figure 4.8) were very encouraging. The hat was tracked quite well over time. Let us apply this algorithm directly to the cluttered data provided.

Figure 7.1 shows the results of such an operation. The initial template is generated by hand. The fitted curves do not track the man at all but fall into edges of the background. The video sequence is so cluttered that probabilistic tracking cannot work directly on it.

We note that the shape of the fitted curves stays the same during the tracking. However, their orientations and sizes vary too much.

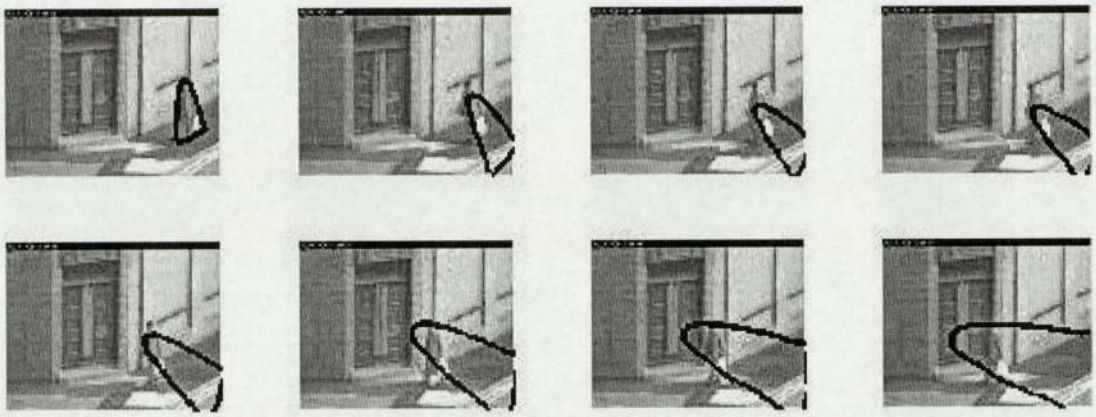

Figure 7.1: Probabilistic video tracking directly applied on real data. *Initialisation of the template surrounding the shape of the man is drawn by hand. The probabilistic tracking algorithm using Kalman filter is then run over a 25-frame video sequence. One frame out of three is displayed. The black curves correspond to the fitted curves.*

7.1.2 Video tracking on background-subtracted data

Our probabilistic video tracking algorithm cannot be run directly on the video sequences provided. Some preprocessing should be applied to simplify them. Let us use the background subtraction described in Chapter 5 and run the tracking algorithm on the background subtracted data.

Fitted curves no longer fall into edges of the background (Figure 7.2) - as it is subtracted. However, as the background affects the shape of the moving object, the fitted curves fall into edges created by the man himself. We do not track the whole man but a part of it.

In this case again, we note that the general shape of the fitted curve is still similar at all times. Orientation and size are still an issue.

7.1.3 Video tracking with constraints

The motion of a human being walking on the pavement is almost a translation. Moreover, we have found problems caused by the orientation and the size of the fitted curves in the previous tracking examples (Figure 7.1 and Figure 7.2).

The probabilistic tracking algorithm described in Chapter 4 (Algo 4.2) works in shape-space. At time t, the fitted curve is defined by an estimated shape-vector $\hat{\mathbf{X}}(t)$ and a covariance matrix $P(t)$. The shape space we have decided to use is the space of Euclidean similarities in which a shape-vector $\mathbf{X}$ is a four-dimensional vector

$$\mathbf{X} = \begin{pmatrix} x_1 & x_2 & x_3 & x_4 \end{pmatrix}^T \tag{7.1}$$

where x_1 and x_2 cover the translations of the curve while x_3 and x_4 govern its orienta-

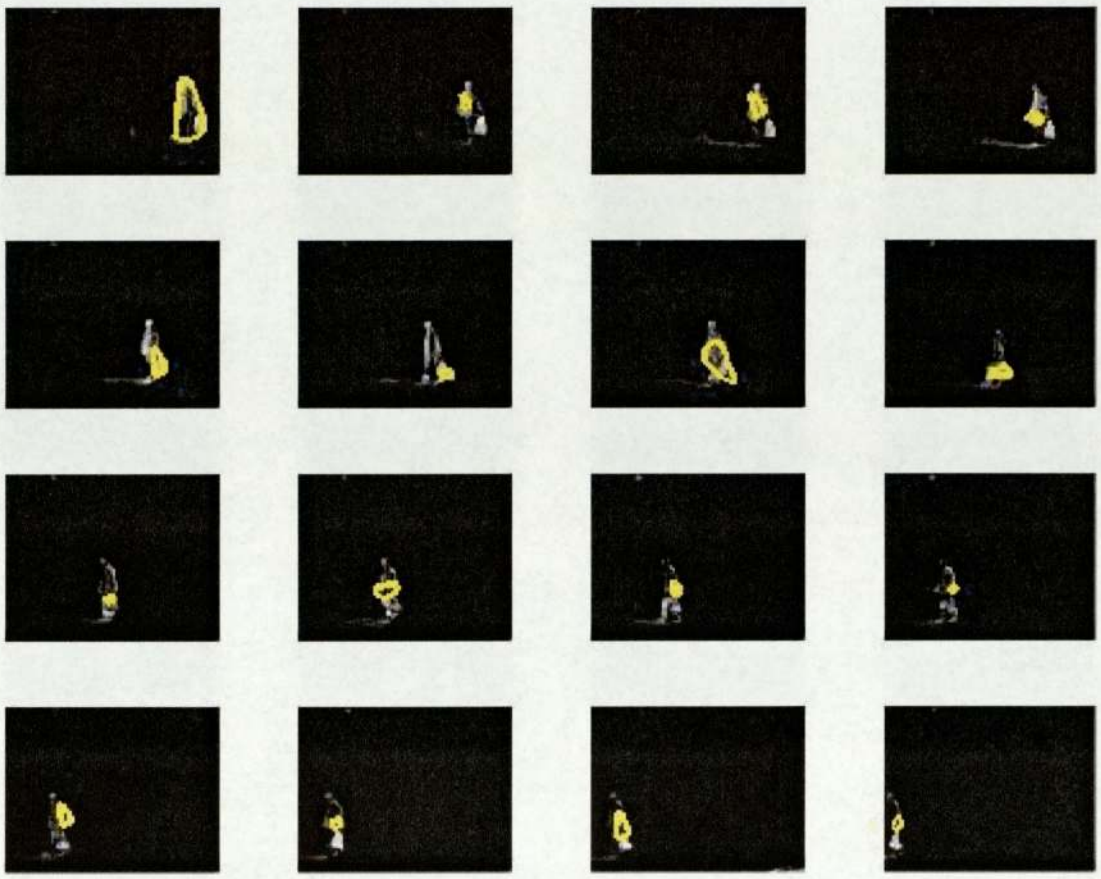

Figure 7.2: Probabilistic video tracking applied on background subtracted data. *Initialisation of the template surrounding the shape of the man is still made by hand. The probabilistic tracking algorithm is then run over a 50-frame background subtracted video sequence. One frame out of three is displayed. The yellow curves correspond to the fitted curves.*

tion and size (Section 4.2.2). If we add constraints on these last two components, the motion of the curve defined by such a shape-vector would be forced to be a translation. Therefore, we force the fitted curves to only have translation motion by adding constraints to the fitted shape-vectors. This probabilistic tracking algorithm with constraints is then run on the background subtracted data.

Figure 7.3 illustrates the success of such a video tracking with constraints. The fitted curves (black contours) track the human walking on the pavement during the whole sequence.

Assuming that human beings moves in translations, we have built a probabilistic video tracking algorithm that works on our cluttered video data. This assumption is acceptable provided that the principal motions are across the frame and not toward or away from the camera.

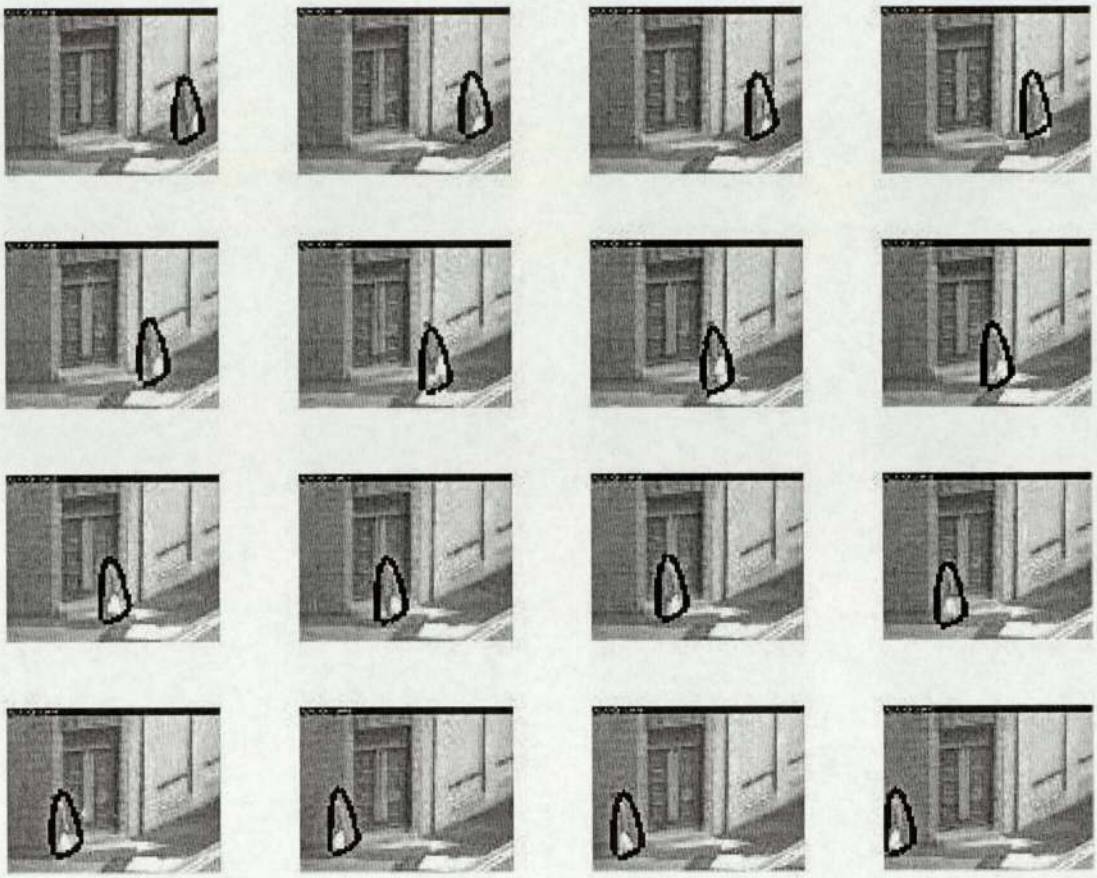

Figure 7.3: Probabilistic video tracking with constraints. *The probabilistic tracking algorithm is run over a 50-frame background subtracted video sequence. Moreover, we add constraints on the last two components of the fitted shape-vectors so that the motion of the fitted curves is a translation. One frame out of three is displayed on the real images (with the background). The black curves correspond to the fitted curves. Run in Matlab, this algorithm lasts 34 seconds.*

7.1.4 Automatic video tracking

The previous tracking algorithms were not completely automatic: the initial templates were drawn by hand around the person we wanted to track. To make tracking automatic, we have to find a way to generate these initial templates.

The person we want to track is a moving object. If we use a feature extraction module based on motion estimation as presented in section 3.3, we would be able to draw a boundary surrounding this person. This contour can be used as the initial template we need. It allows to initialise automatically the probabilistic video tracking algorithm.

The results of such an automatic video tracking algorithm is presented in Figure 7.4. After having initialized the template using motion estimation feature extraction

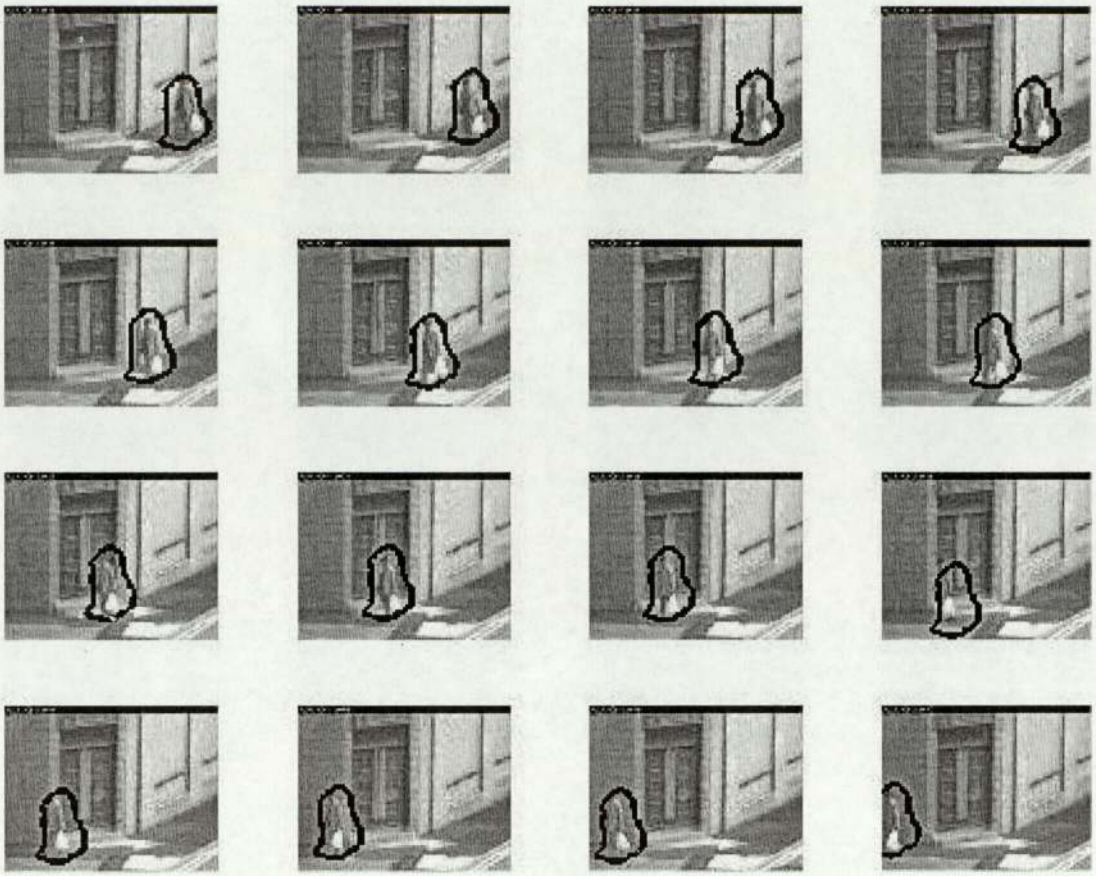

Figure 7.4: Automatic probabilistic video tracking. *Initialisation of the template surrounding the shape of the man is done using motion estimation feature extraction. The probabilistic tracking algorithm with constraints is then run over a 50-frame background subtracted video sequence. One frame of every three is displayed. The black curves correspond to the fitted curves.*

(first frame), the tracking algorithm is run over a 50-frame background-subtracted video sequence. Constraints are added so that the motion of the fitted curves is forced to be translational. Run in Matlab, this algorithm lasts 49 seconds.

We have built a probabilistic video tracking algorithm using constraints. It works on the cluttered video data provided. However, such an algorithm is quite slow. Indeed, tracking automatically a person over a 12-second video sequence lasts 49 seconds. It is too long to deal with real time tracking. However, time should be gained by using another programing language than Matlab.

This tracking algorithm is restricted to slow motion video tracking (human video tracking). Indeed, for fast motion objects, the tracking cannot work as the fitted curves are recursively obtained using normal image processing (section 4.4.1). The length of the normals is a measure of the motion speed. To make the tracking algorithm work

in fast motion cases, as we are dealing with 4-frame per second video sequences, the length of the normals should be longer than the size of the video frames!

7.2 Shape Classification

Chapter 6 allowed us to define an event detection algorithm (Algo 6.1). Once an event was detected, we needed to be able to classify it whether it has been triggered off by one or several human beings or not. EDH was presented as a useful tool to build such a human classifier (section 6.2). First basic but encouraging linear classification results have been presented in section 6.3. Two different classifiers were discussed: one built from EDH directly extracted from the real data and one using EDH generated from background subtracted data. Further experiments need to be performed to try to build a robust human classifier.

7.2.1 Visual results

In order to validate the use of EDH as a tool to build a human classifier, let us display some visual results. We use the same data sets as in section 6.3. The training set was built by taking one ten-minute video sequence every four hour during 24 hours. The test set was constructed by taking nine ten-minute sequences at different times of the day.

Classification results using NeuroScale

NeuroScale [10] enables to visualize a high dimensional data set into a two or three-dimensional space. The NeuroScale algorithm uses stress functions to learn a non-linear mapping from the high dimensional data space to the feature space. The dimension of the feature space is two or three so that the results can be visualized. The mappings in NeuroScale are based on neural network models, specifically radial basis functions.

We use such a probabilistic tool to visualize our classification results (Figure 7.5) on data sets obtained from EDH generated directly with the real data first (left), then from EDH created with the background-subtracted data (right).

We note that, in the case of EDH directly generated from the real video data, we cannot define at all a cluster of human events. Indeed, the red points are scattered in the whole two-dimensional feature space. On another hand, for EDH created from background-subtracted data, human events are more or less gathered in a clearly defined cluster. EDH generated from background subtracted data must be a more suitable tool to build a robust human classifier than EDH directly generated from the real data.

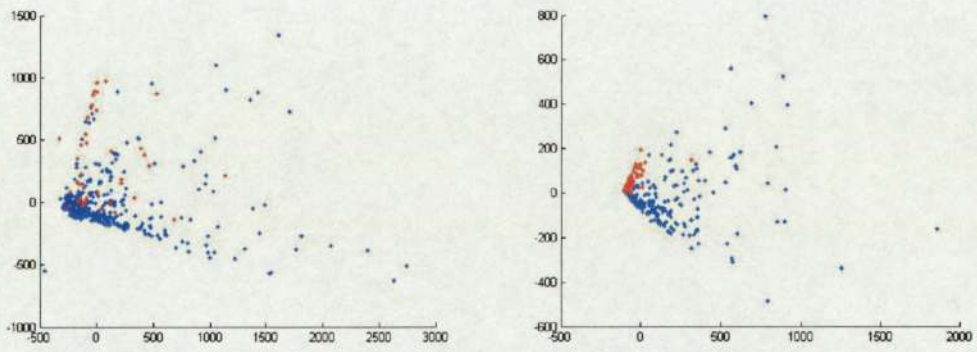

Figure 7.5: Visualisation with NeuroScale. *Training data from non-human events (blue points) and human events (red points) is projected into a two-dimensional plan by a NeuroScale model. The left and right plots are respectively obtained from EDH generated from real data and background-subtracted data.*

However, there still exist some overlapping regions between the human events and the non-human events.

Classification results using Partiview

In order to extract more information from the projection made by NeuroScale, we may want to know what events have been misclassified to be able to visualize them. To do so, the output of the NeuroScale algorithm is used as the input of Partiview. Partiview is a fast industrial-strength three-dimensional plotting tool written by Stuart Levy [11]. It turns out to be useful for seeing the output of machine learning algorithms (such as NeuroScale). An event is no longer represented by a point (blue or red, depending on the class it belongs to) but by its corresponding frame output by the event detection algorithm (Algo 6.1). An example of such a visualisation using background subtracted data is presented in Figure 7.6.

The classes of human events and non-human events can be displayed separately so that the misclassified events are extracted more easily (Figure 7.7).

In this example, one human event is clearly misclassified. Its corresponding feature point in the NeuroScale projection is standing far away from the other human beings, among the non-human events (Figure 7.6 and Figure 7.7). However, the principal overlap between the two classes is so dense that we cannot clearly define the whole misclassified events. Points are too close to each other. Mathematical methods will be used in section 7.2.3 to visualize properly the misclassified events.

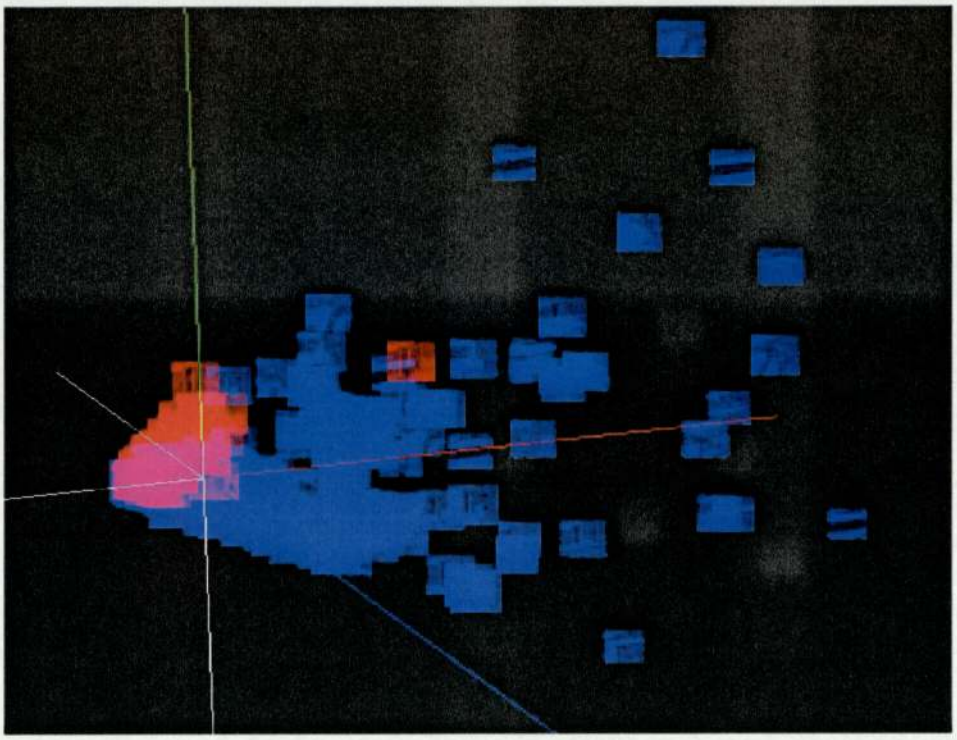

Figure 7.6: Visualisation with Partiview. *Training data from non-human events (blue) and human events (red) is projected into a three-dimensional space by a NeuroScale model. Then, using Partiview, the frames corresponding to the detected events are displayed.*

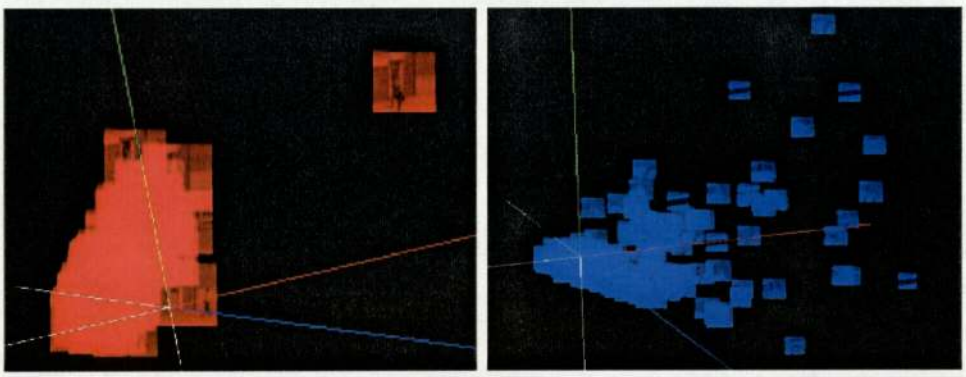

Figure 7.7: Visualisation of the classes with Partiview. *Separating the human class from the non-human class enables to visualize the clearly misclassified events. The red image at the top right corner of the human class image (left) is clearly misclassified.*

7.2.2 Non-linear classification

Basic linear human classifiers were presented in section 6.3. The results we obtained were very encouraging as we had good classification rates: around 96%. However, the human misclassification rates were greater than 25%. We need to build a more robust human classifier using non-linear classification in order to improve the false positive rate.

The multi-layer perceptron (MLP) is the most widely used neural network [12]. Assuming enough data is provided to estimate the network parameters, a MLP can model a non-linear decision boundary in a classification problem.

We used the same data sets as in section 6.3. Non-linear classification using MLP was run in both cases of EDH generated from real data and from background-subtracted data. The MLP was trained using the one-hour training set previously described (section 6.3). Evidence procedure [12] was used to determine the optimal weights and hyperparameters of the network. We then tested the non-linear classifiers with the ninety-minute test set. The resulting confusion matrices are displayed in Figure 7.8.

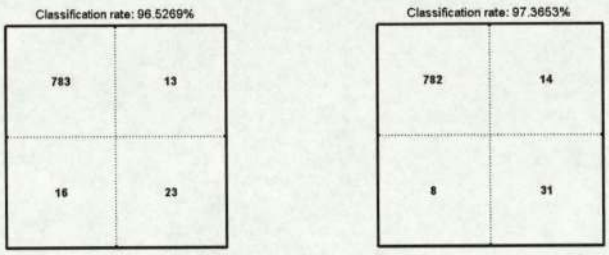

Figure 7.8: Confusion matrices of the non-linear human classifiers using MLP. *The first row represents the non-human events while the second one corresponds to the human beings detected. Left: Classifier on real data - Right: Classifier on background-subtracted data.*

The resulting classification rates are at least as high as for the linear classification described in section 6.3 (more than 96%). In the case of EDH generated from background-subtracted data, this rate is even greater than 97%. However, the human misclassification rate varies quite a lot between the two non-linear classifiers. For EDH created directly from real data, it is equal to 41% whereas for EDH generated from background-subtracted data it is 20%.

The non-linear classifier using background-subtracted data classifies the human beings from any other events in a much better way than the classifier using real data. It is the human classifier we have adopted to perform further experiments.

However, some misclassified events still remain. Let us display those events to understand why they have been classified wrongly. Figure 7.9 presents the eight human

beings classified as non-human events (the false negatives) whereas Figure 7.10 shows the fourteen non-human events classified as human beings (the false positives).

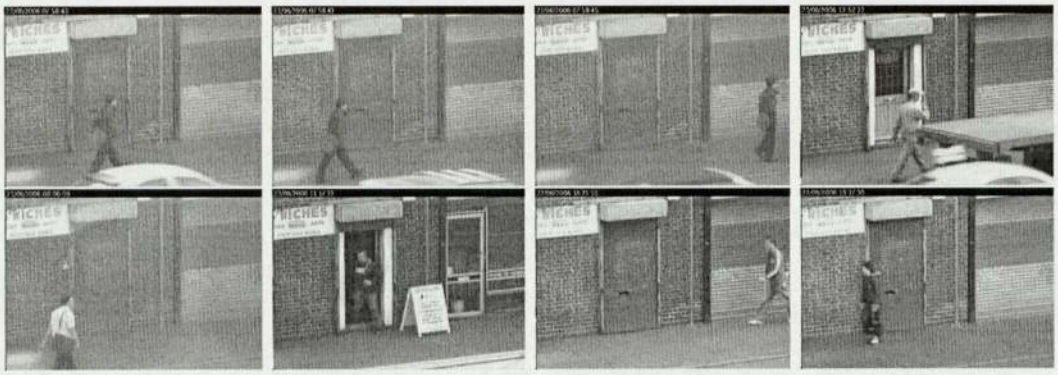

Figure 7.9: The false negatives *(i.e.: the human beings classified as non-human events)*

False negative triggers displayed in Figure 7.9 are caused by four different reasons. In the first row of Figure 7.9, the four detected men are always walking while a car is driving in the street. Two events are occurring at the same time. The moving object extraction module outputs a single area of motion. Since a car is bigger than a human being, edges created by the vehicle are predominant in the EDH generated. Thus, the humans walking are not considered. Events are misclassified. Secondly, in the first frame of the second row in Figure 7.9, a flash of light on the left side of the image affects the contours of the man walking. The man is misclassified as well. Then, on the next frame, the detected man is just going out of the door. Just a part of him is visible. It is not enough to classify well this man. Finally, on the last two frames in Figure 7.9, nothing seems to explain why those two walking human beings have been misclassified. If we display the background subtracted images corresponding to these two frames (Figure 7.11), we note that the background affects a lot the apparent shapes of the two human beings. That explains why they have been misclassified.

We discussed in section 6.1.1 how the background could affect the human shapes and detect several events for a single moving object at different time. Using this idea, we can try to find if the misclassified human beings have been well-classified at some point of the video by the non-linear classifier. Displaying the whole set of well-classified humans, three familiar human beings are extracted in Figure 7.12.

Finally, as some misclassified humans have been well-classified at another time in the video, the actual human misclassification rate decreases. If we no longer consider the three human beings in Figure 7.12 as misclassified events, this rate becomes lower than 13%.

On another hand, we also have to explain why the false positive triggers shown in

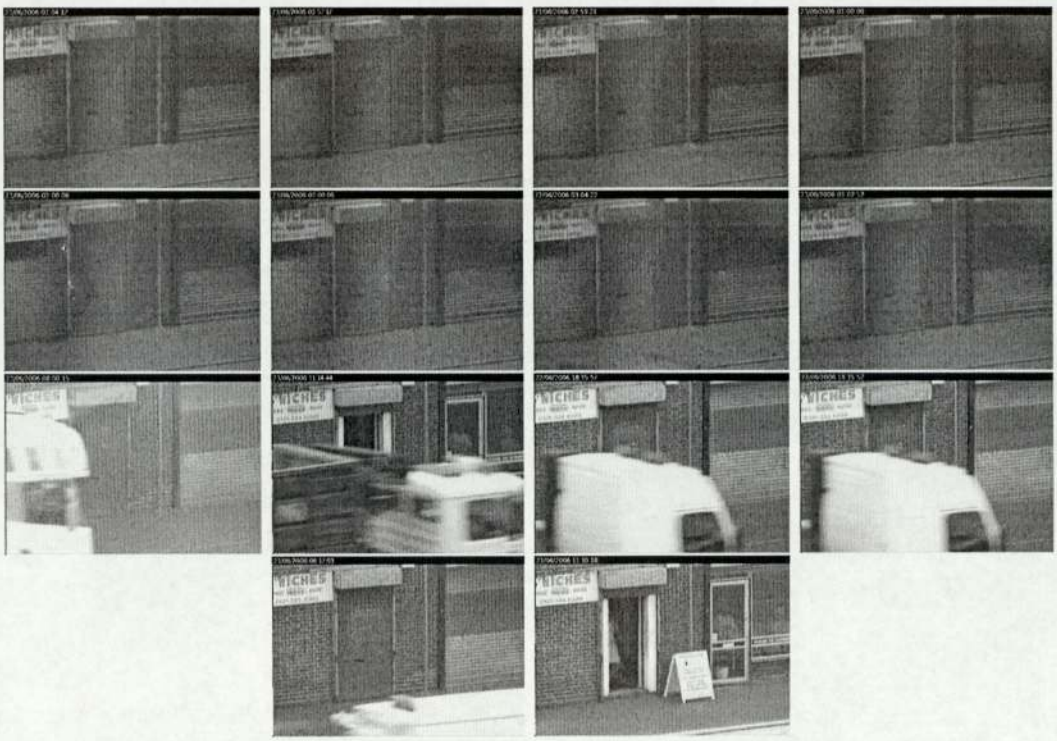

Figure 7.10: The false positives *(i.e.: the non-human events classified as human beings)*

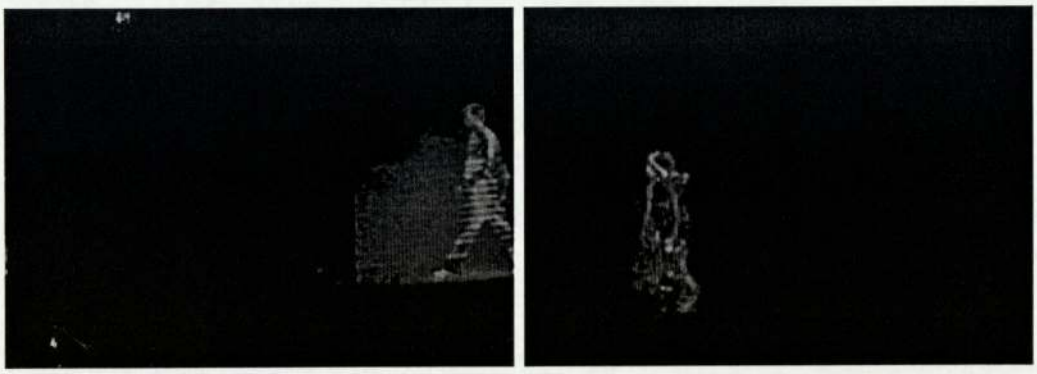

Figure 7.11: False negative triggers caused by the background which affects the apparent shapes of the humans.

Figure 7.12: False negative triggers correctly classified by the non-linear classifier at another time.

Figure 7.10 have been wrongly classified by the non-linear classifier. The eight first events (the two first rows in Figure 7.10) are caused by illumination changes occurring at night. Sometimes, some flashes of light are strong enough to create areas of motion whose edges are distributed in the whole range of directions. They can be classified as human beings. False positives caused by big moving objects are shown in the third row of Figure 7.10. They are classified as human beings. We presented in section 3.4 few drawbacks of feature extraction using motion estimation. Multiple areas of motion could be created by one big moving object. This can explain why a truck can be classified as a human being. In the images displayed in the third row of Figure 7.10, it is not the whole truck that is actually detected but a small part of it (Figure 7.13, left image), which is classified wrongly. Big moving objects can also generate wrong regions of motion by creating light changes in the scene. In the first image of the fourth row in Figure 7.10, an illumination change is detected in the iron shutter (Figure 7.13, central image). It is classified as a human being. Finally, the last image in Figure 7.10 is a man about to pass through the door (Figure 7.13, right image). It could have been considered as a human being.

Finally, using EDH generated from background-subtracted data, we have built a robust and efficient non-linear human classifier. A few misclassified events still remain though. They are caused by the feature extraction based on motion estimation module whose drawbacks have been discussed in section 3.4. The background can also affect the apparent shape of the moving objects and generates some misclassification.

7.2.3 Long training and test sets

The non-linear human classifier we have just presented in section 7.2.2 was trained and tested over longer training and test sets. One ten-minute video sequence was taken every hour during 24 hours to construct a four-hour training set. Such a training set contains different illumination conditions depending on the time of the day. 2156 events have been detected in the training set. A four-hour test set was built using the same

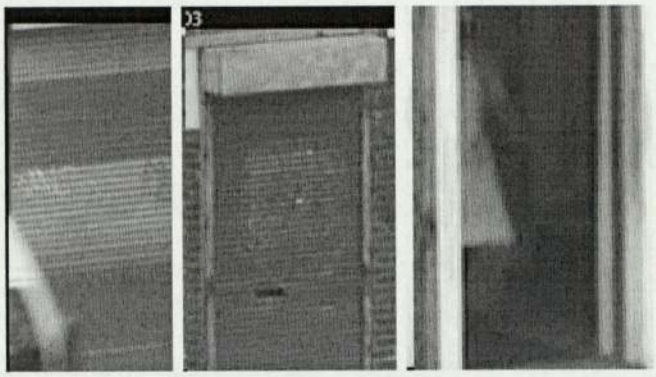

Figure 7.13: False positive triggers. *The left image is a small part of a truck. An illumination change is detected in the central image. A part of a man is shown in the right image. They are all wrongly classified as human beings.*

process. 1606 events have been detected in the test set. 72 were triggered by human beings, 1534 by any other events.

The resulting confusion matrix is displayed in Figure 7.14.

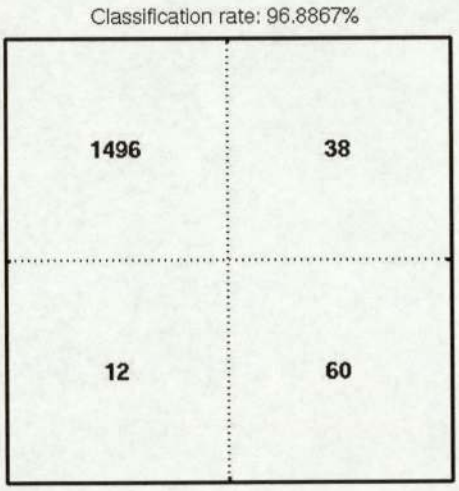

Figure 7.14: Confusion matrix of the non-linear human classifier tested on the long test set.

Even with longer data sets, the non-linear human classifier we have built in section 7.2.2 outputs very good results. The good classification rate is still high (almost 97%) whereas the human misclassification rate stays low (16%).

Chapter 8

Conclusion

8.1 Achievements

Some noticeable achievements have been made during this eight-month MSc project. We have developed an event detection algorithm. Thanks to it, any moving object, bigger than a minimum threshold size, triggered an alarm event on entering the central region of the scene (the region of interest). Thus, as soon as a big enough object was moving in the scene, we could detect it. Furthermore, in order to focus on the moving regions, feature extraction based on motion estimation was used to highlight the events occurring by drawing their boundaries. Every event occurring in the scene was more than detected. It was also highlighted. It allowed us to deal with multiple objects moving in the scene at the same time. However, a few drawbacks have been identified. Indeed, big moving objects could generate several areas of motion. Moreover, they could affect the lighting conditions of the whole scene and thus cause the detection of false events. This phenomenon was emphasized at night.

After being detected, each event had to be classified in two different classes. We have to know if it has been triggered by a human or several human beings or by any other kind of events (vehicle driving in the street, illumination change, etc). To do so, we have constructed a robust human classifier using multi-layer perceptrons (MLPs) and edge direction histograms (EDHs). EDH was built from the background-subtracted data corresponding to the detected event. On the contrary to vehicles or light changes which generated strongly vertical and horizontal EDHs, human beings created EDHs in which edges were more distributed in the whole range of directions. These ideas allowed us to build a quite robust non-linear human classifier. The results obtained were very good: the classification rate was almost 97% and the human misclassification rate was 16%.

Preliminary processes such as the background subtraction we used in the classification algorithm were applied to the data provided. Those processes were essential to

obtain good results from the cluttered video data. However, they were quite basic. Several parameters needed to be defined by hand: the noise threshold of the background subtraction (Section 4.1), the time step and the minimum tolerance of the dynamic background update (Section 4.2).

Such a human classifier does not exist in TVM's algorithms. It allowed us to have better results by decreasing the false positive rate of the system.

At this point, we have detected every event occurring in the scene. We have then classified them so that the events triggered by human being remained only (i.e. the objects able to cause an incident). Basically, every human beings passing through the scene generated an alarm event. This was not sufficient. Indeed, to cause an incident and trigger an alarm, a human being also needed to go in or out of the door. We had to know where the humans were going. We needed to track them over time to estimate their trajectories. That is why we finally used mathematical models to build a probabilistic video tracking algorithm using first-order Kalman filters. Preprocesses were applied to the data in order to make this algorithm work with the cluttered video data. Moreover, motion estimation was used to create an automatic tracking algorithm. This tracking was assumed to work for slow motion objects only (the human beings). As the only objects which needed to be tracked are the one classified as human beings, this assumption was suitable.

8.2 Future work

Future works still need to be done in order to actually build a video based detection system.

The main task will be to link the detection and the human classification algorithms to the video tracking. Doing so, every human walking through the scene will no longer be considered as an alarm event. The only incident remaining will be the human beings going in or out of the door as we will know where they are going thanks to the tracking algorithm.

Moreover, we just ran our algorithms in Matlab. Thus, they were quite long; especially the probabilistic tracking algorithm and the motion estimation search. Better results should be obtained by using other programming languages (Java for instance).

Other things will need to be improved in the algorithms we have already implemented. The basic preliminary processes (background subtraction and dynamic background update) will need to be changed. Indeed, a new definition of the background will need to be given so that it does not affect the apparent shape of the moving objects anymore. The feature extraction based on motion estimation will also need some improvements: the identified drawbacks should be solved. The regions of interest surrounding every event should no longer be rectangular so that the algorithms become more general. More complex shape-spaces may need to be used to allow more degrees of freedom to the tracked curves. Finally, some parameters which need to be initialized by hand still remain. They will need to be generalized so that the system can work for every camera.

Bibliography

[1] B. Furht, J. Greenberg and R. Westwater, *Motion estimation algorithms for video compression*, Kluwer Academic Publishers, 1997.

[2] S. Zhu and K.K. Ma, *A new diamond search algorithm for fast block matching motion estimation*, in Proceedings of Int. Conf. on Information, Communication and Signal Processing, pp. 292-296, Sept. 1997.

[3] A. Blake and A. Yuille, *Active Vision*, MIT Press, 1992.

[4] A. Blake and M. Isard, *Active Contours*, Springer, 1998.

[5] M. Isard and A. Blake, *ICondensation: Unifying low-level and high-level tracking in a stochastic framework*, Proc 5th European Conf. Computer Vision, Vol. 1 pp. 893-908, 1998.

[6] S. Arulampalam, S. Maskell, N. Gordon and T. Clapp, *A Tutorial on Particle Filters for On-line Non-linear/Non-Gaussian Bayesian Tracking*, IEEE Transactions on Signal Processing, Vol. 50, NO. 2, pp. 174-188, 2002.

[7] R.C. Gonzalez and R.E. Woods, *Digital Image Processing*, International Edition, pp. 519-642, 2002.

[8] A. Jain , H. Zhang and A. Vailaya, *On Image Classification: City vs. Landscape.*, In Workshop in Content-based Access to Image and Video Libraries, pp. 3-8, 1998.

[9] F. Mahmoudi, J. Shanbehzedeh, A-M. Eftekhari-Moghadam and H. Soltanian-Zadeh, *A new non-segmentation shape-based image indexing method*, IEEE International Conference on Acoustics, Speech, and Signal Processing, Proceedings (ICASSP '03), Vol. 3 pp. 17-20, 2003.

[10] D. Lowe, M. E. Tipping, *NeuroScale: Novel Topographic Feature Extraction using*

RBF Networks, Advances in Neural Information Processing Systems 9, pp. 543-549, 1997.

[11] Dinoj Surendran, Stuart Levy, *Visualizing High Dimensional Datasets Using Partiview*, infovis, p20, IEEE Symposium on Information Visualization (INFOVIS'04), 2004.

[12] C. M. Bishop, *Neural Networks for Pattern Recognition.*, Oxford University Press, 1995.