Tracking People in Video Surveillance

- VGIS Master Thesis 2011-2012-

Benjamin Levesque & Vlad Nica

Project Report – VGIS 921

Aalborg University
School of Information and Communication Technology

LaTeX template by Jesper Kjær Nielsen <jkn@es.aau.dk>.

AALBORG UNIVERSITY

School of Information and Communication Technology
Selma Lagerlöfsvej 300
9220 Aalborg Øst, Denmark
www.en.sict.aau.dk

Title:
Tracking People in Video Surveillance

Theme:
Computer Vision

Project Period:
September 2011 - June 2012

Project Group:
VGIS 921

Participant(s):
Benjamin Levesque
Vlad Nica

Supervisor(s):
Preben Fihl
Thomas B. Moeslund

Copies: 3

Page Numbers: 89

Date of Completion:
May 30, 2012

Abstract:

We present a complete framework for finding and tracking people in a video surveillance context, for both indoor and outdoor environments, in a long-term video sequence.

We introduce a detection solution based on pixel and region detection principles. We use a *Gaussian Mixture Model* for modelling the background, combined with contour analysis to isolate the blobs. We ensure a robust detection of humans by applying a *Histogram of Oriented Gradients* detector confronted with a *Support Vector Machine*, previously trained for this purpose.

We keep track of detected people through the frames by associating a unique identification to each of them. The new detections are associated to these by using matching techniques, such as colour histogram comparison and position prediction, using a Kalman filter.

Preface

This report is the result of our master thesis, started in September 2011 and handed in May 2012. The master *Vision, Graphics and Interactive Systems* that we followed gave us the necessary knowledge in Computer Vision to tackle this problem correctly.

We thank our supervisors, Preben Fihl and Thomas B. Moeslund, from Aalborg University, for their numerous advices, their attention and interest to our work. We also thank all teachers, staff, and students from the Aalborg University, the Department of Electronic Systems and the School of Information and Communication Technology for welcoming us as exchange students in this Master of Science.

Aalborg University, May 30, 2012

Benjamin Levesque
<bleves11@student.aau.dk>

Vlad Nica
<vnica10@student.aau.dk>

Contents

List of Figures

Chapter 1

Introduction

1.1 Problem Description

With the evolution and growing of society, human behaviour in public places has became important to supervise. Since the police forces cannot always be physically present, the solution of video surveillance was introduced. The United Kingdom was the first to do so, after the IRA attacks in the early nineties. Since then, many countries like the USA, France, and several countries in South America are using these systems[1].

The limitation of these systems is obvious: the amount of data to handle is way too massive for human to process it. As an illustration, there are around 1.85 millions cameras in the UK, including almost 500,000 in London only [1]. Whether these data is processed locally (institutions with internal security like banks, prisons, universities, military facilities) or generally (public places like streets, highways), human fatigue problem still arises. Adding to this issue the human labour cost, the solution of manual processing becomes completely impossible.

Detractors also mention the burning issue of privacy, mainly because of the possibility of filing individuals according to their actions and moves.

Computer Vision is the branch of computer sciences that deals with everything related to images (acquiring, processing, analysing, understanding). Its recent advances permits a lot of different applications, which are widely used in industry (detecting drawbacks in production lines), medical imagery, computer interactions and, of course, video surveillance. When applied to this last purpose, systems usually have two main aspects: identifying a specific object and tracking it through the video.

Because most of the everyday interaction is with people, our project focuses on tracking humans in a video stream and analysing their path (e.g. Where can people walk? Where do they typically appear in a known scene?). Its purposes is to find robust methods applicable to several usages.

However detecting a human in a given scene is not an easy task as multiple variables may interfere. The appearance and the wide range of poses that a human can adopt are

[1]Surveillance systems are often denominated as CCTV, for Close-Circuit Television

the main concerns that should be taken into account when designing a human detection system.

Therefore, there is need for an automated system that would track humans and detect/analyse their movement.

1.2 Applications

The applications of human detection algorithms are very numerous: count how many people are inside a given area in the same time, generate statistics on pedestrian traffic, or recognize humans for interaction purposes in robotics or other. However, the applications for tracking is not that straightforward since its usage is not always relevant (for many applications, detection is enough). The first one that comes to mind is security issues. Indeed we can imagine a system that would facilitate human work by automatically extracting the path of all humans in a scene rapidly.

Another possible usage could be for human behaviour analysis. For instance, we can imagine a pedestrian aid for traffic: tracking the path taken by pedestrians in one scene during a long period (several weeks to a couple of months) would show behaviour and would help some decision making like "is a new pedestrian crossing needed at that specific point?".

1.3 Previous Work

Several researches have been led in the field of human detection. A method used for detecting people was introduced by Dalal and Triggs [2] who created a robust detection technique specially adapted to human detection, called the Histogram of Oriented Gradients. The method implies the usage of a sliding detection window on an input image, that would return values corresponding to edges or contours that together define a person. The respective values are then introduced inside a *Support Vector Machine* (SVM) and the retrieved detection will be classified as according to whether they are human or not. The algorithm is intensively used or improved in new applications.

One approach was developed by Breitenstein *et al.* [3] who introduced a complete detection-tracking system. The respective algorithm uses only a part from the *Histogram of Oriented Gradients* method presented earlier. After using the detection window, the algorithm will output a confidence map which will describe the probability of a person to be detected in each region. The high confidence detections will have a particle filter assigned for tracking. Particle filtering is a strong method mainly used for multiple-tracking purposes. The idea is to generate N particles and then estimate their position for the next prediction. The main problem in this type of system is the association step. The association is basically the way in which the system determines which detection matches which track. By doing this one can uniquely identify a specific person through the whole scene without loosing its ID in case of occlusion. In his system Breitenstein associates the tracks by using a *greedy* association algorithm. A scoring matrix is created for each track-detection pair. Then the pair with the highest score will be selected and

the rows and columns belonging to the tracks and detection will be deleted until there will not be any available pair left. In the end only the associations above a certain threshold will be considered a valid match to a target. However the system's problem is that, due to the fact both used algorithms (confidence-detection and particle filtering) are very computationally intensive, the performance of the system will be quite poor.

Another approach for a human detection-tracking systems was developed by Gavrila and Munder [4]. It implied the usage of a *shape-based* human detection on regions of interest. Using a large number of exemplars that described the human shape distribution, their system performed a template matching in combination with human texture based classification that would detect pedestrians. For tracking, their system used a simple $\alpha - \beta$ tracker that estimated the object state parameters. The association step in this case is done with the Hungarian method by using a cost matrix built from the similarity between the prediction of the tracks and the associated measurements. Every time an object appears in the scene it will have an associated track only if the object appears in m number of frames. The track it will be lost if the track was not detected in n number of frames.

A common characteristic of these systems is the clear identification of two problems, Detection and Tracking. The output of the detection is used in the tracking, independently of the chosen method. The same structure is used for the system described in this report.

1.4 Problem Formulation

The problem can be formulated as follow:

> How can we realize a robust tracking framework using simple devices and as few requirements as possible?

More precisely, *simple device* means that the system should work on any kind of camera, with no calibration and no specific hardware (e.g. laser rangefinder, infra-red camera); *few requirements* means that it intends to adapt to most kinds of situations and changes.

1.5 General Approach

The aim of this project is to obtain good results with simple and fast computations. For this a new approach of detection is introduced: it regroups two state of the art methods, segmentation detection (Mixture of Gaussian [5] with contour detection) and region detection (Histogram of Oriented Gradients)[2]. The tracking part uses features comparison and a Kalman filter.

Figure 1.1 represents the two parts of the system. In the **Detection** part (see chapeter 3), computer vision techniques are used to process the image obtained from a camera and find human blobs, using predicted position of previous blobs to enhance

detection. The **Tracking** part (see paragraph 4) will match those detected blobs with the previous ones using some particular features.

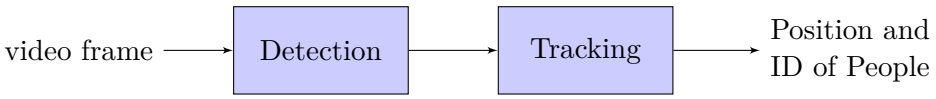


Figure 1.1: System representation in blocks

1.6 System Requirements

The system should be capable of working in the following conditions:

Long term application The system should work on a long video sequence, with a 640x480 resolution camera. It is important nowadays to have surveillance systems that work autonomously and detect or track a person during a long period of time.

Adaptability The system should adapt in an autonomous manner to any kind of environment (indoor or outdoor) changes. Such changes can be represented by repetitive movement in the scene (trees blown by wind, waves from water) or illumination changes. The system should work in daylight scenes, in the widest range of different illuminations as possible. This means that it should work in both cloudy and sunny scenes, indoor and outdoor.

Classification In every classification system, four states for a detection/classification couple can occur. Here are these states applied to the relevant possible classifications of this application.

	positive	**negative**
true	Human detected as human	Non-human detected as non-human
false	Non-human detected as human	Human detected as non-human

Table 1.1: Classification of possible results

Nb: False positive and false negative are respectively known in statistics as Type I and Type II errors.

Considering table 1.1, it has been decided to reduce as much as possible the false-positive rate (avoiding "false alarms"), and be more flexible on the false-negative rate (missing people sometimes).

Unique track identification The system should have an autonomous tracking algorithm that would assign an unique ID to a tracked person during the whole video capture. Moreover if the same person gets out of the scene and re-enters it, the ID should be reassigned to that person.

1.7 System Limitations

Shadows In all vision algorithms, the results mainly rely on the illumination conditions in the scene. Either the system is working in indoor environments, where illumination conditions are represented by artificial lighting, or with outdoor environments where natural lighting is involved, there will always be confounding problems given by the presence of shadow. The presence of shadow brings many disadvantages to the vision system as there is loss of information of the region covered by the shadow making the images more difficult to interpret. Also various algorithms like image matching, detection and tracking will not be able to output optimal results due to shadow presence. There have been many studies made regarding shadow removal methods. One of them can be seen in Sanjeev Kumar's paper on shadow removal [6]. The proposed algorithm firstly removes the unwanted salt and pepper noise from the frame by applying a mean filter. Afterwards the image will be split in three channels in order to get a better overview on the effect that shadows have in the three dimensions of colour. It has been observed that colours in the shadow regions will have a larger value than the average, while the regions that are not shadow will have colour values smaller than the average. Then a threshold piecewise function is created in order to determine which pixels are shadow and which non-shadow. By convolving the noise-free binary image with the original one shadow will be detected. Finally shadow is removed by the usage of energy functions.

However in this system the shadow removal problem was not treated at all due to the fact that more importance was given towards the detection and the tracking. Therefore it was established that the implemented system should work only on cloudy condition where lighting sources would not interfere with the used algorithms.

Separating two objects by their colour Another problem that represent a limitation to the proposed system is how can two objects be fully separated and assigned with an unique ID based on their colour. As the given system is meant to detect and track human beings, the problem tends to get more complex. It is known that most people tend to dress more or less the same below their waist (pants usually tend to have same colours: blue, black and brown) therefore assigning each person with an unique ID becomes a hard task to solve. On the other hand, the upper body can give relevant information regarding clothing colour. Although this case can give good outputs, there are some cases in which these outputs might not give relevant results whatsoever like a person wearing an opened blue jacket over a green shirt. In this case the frontal view will give one result while the side-view and the back will give another. Therefore the respective person will have *2* IDs and it would be really hard to track him/her.

There are numerous methods in which two objects can be separated according to their colour and all are based on analysing the colour histogram. Comparing two histograms is one of the approach and this can be seen later in section 2.1.5. Another approach in which this problem was attacked can be seen in [7] where Swain and Ballard introduce an algorithm called *Histogram Backprojection*. Their algorithm de-emphasize the colours from objects that are not of interest in order not to distract the algorithm.

Groups of People The segmentation process, and the classification that follows it gets way more complicated when people are close. Indeed the found blobs do not have the expected shape for a human, and the possible combinations are just too many to be trained the same way (with a Support Vector Machine for instance). A possible approach to solve this problem is the use of the particle filtering as introduced by Breitenstein *et al.* [3] and explained in the Previous Work section. This is a limitation to the system.

Chapter 2

Image & Video Processing

This chapter will briefly introduce methods and concepts from the Computer Vision science, which will be further used in the implementation and development of this project.

2.1 Image Processing operations

2.1.1 Image Processing

Image Processing is one of the most fundamental parts of computer vision science. It refers to various simple techniques that help analysing images in an easier manner, but it also contains high-level operations which can be used if, for example, a feature of high interest is needed. The methods used for processing an image are various ranging from pixel analysis to image recovery and recognition. However the most important and trivial method used is image segmentation.

Image segmentation represents a process in which the image is partitioned in segments (groups of pixels), which allows a simpler representation of the given image making it more meaningful and therefore easier to analyse. The process implies the usage of some high-level methods in which the input images can be tested for object recognition or scene understanding. Segmentation can be done in many ways starting from thersholding or edge detection to a more complex method such as model-based segmentation. For this system the segmentation process is represented by four fundamental and straightforward methods:

- Filtering
- Morphology
- Contour Detection
- Histogram Analysis

2.1.2 Filtering

Filtering an image is also a crucial step in analysing an image. Usually after applying some segmentation techniques such as thresholding, noise or some small artefacts are introduced in the image. A good way of eliminating these unwanted aspects is using some kind of filter which would remove the small isolated silhouette-pixels. The process is also known as blurring and can be done using different types of filters:

- Mean filter
- Median filter
- Gaussian filter

6	2	0
3	97	4
19	3	10

Table 2.1: A 3x3 window.

The **Mean filter** implies using a simple sliding-window of any shape over the given image and replacing the centre pixel of the sliding window with the average of all pixels inside the window. For example, *Table 2.1* shows a 3 x 3 window where the average value would be 16.

(a) Original **(b)** Mean Filter

Figure 2.1: Mean Filtering.

The **Median filter** uses the same sliding-window method like the previous filter does but it replaces the centre pixel of the window with the median of all the pixel values inside. Therefore, for the same kernel in *Table 2.1* the centre value will be set to 4. The method works really good when the given image is altered by "salt-and-pepper" noise[1]. This method is the most commonly used because of its ability of performing fast and preserving the edges in the filtered image. The bigger the kernel the more powerful the filter becomes as it can be seen in Figure 2.2.

[1]Salt-and-Pepper noise is the presence of isolated pixels which are either white - 0 value - or black - 255 value.

(a) Median Filter size 3x3 **(b)** Median Filter size 5x5 **(c)** Median Filter size 7x7

Figure 2.2: Median Filtering.

The **Gaussian filter** is one of the best and useful filters, though not the fastest. The Gaussian filter is a low-pass filter that reduces both noise and detail. The output image looks as if you are watching the original through a translucent screen. By using the Gaussian filter, the components with high frequencies are reduced, making it an important asset in the pre-processing step. The algorithm implies convolving each point from the given image with a Gaussian kernel like the one in 2.1.

$$\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix} \tag{2.1}$$

However the presented kernel is just one representation of how a Gaussian kernel looks like. It is called a "Gaussian hump" because of its specific centred shape [8]. The filter follows the formula 2.2:

$$G(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{2.2}$$

The value of the variance (σ) shows how much the image is blurred. Therefore Figure 2.3 the impact of sigma can be seen as it was increased progressively on a 3x3 kernel. The size of the kernel also matters when applying the Gaussian filter.

2.1.3 Dilation and Erosion

Dilation is a morphological operation which plays an important role in the pre-processing step of image analysis depending on the application. The goal of this procedure is to increase the dimensions of objects in a binary image. Therefore small unwanted holes are filled and objects merged. The strength of the dilation's effect is given by the size of the structuring element, so the bigger the size the bigger objects will get in the output image. The method is very useful when one wants to have clear view of the foreground objects in a given scene by enlarging the boundaries of the regions that contain white pixels. [9].

(a) kernel size = 3x3 **(b)** kernel size = 5x5 **(c)** kernel size = 11x11

Figure 2.3: Gauss Filtering. $\sigma = 5$

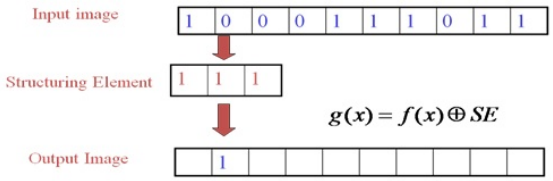


Figure 2.4: Dilation Algorithm. If one of the "1"s from the structuring element correspond with a "1" from the analysed input then the output will be **1** and **"0"**otherwise.

However sometimes dilation is not enough to have a good overview of all the objects in the image, as most of the times objects are being merged and therefore another operation should be used. Figure 2.5 shows how the number of iterations can influence the processed image. Its visible that at *5* iterations unwanted noise becomes an unwanted blob that may interfere with further processing.

(a) 1 iteration **(b)** 3 iterations **(c)** 5 iterations

Figure 2.5: Dilation.

Erosion is another operation in mathematical morphology used for pre-processing. In contrast to the dilation method, erosion will decrease the dimensions of objects enlarging the holes in the binary image. The method implies using a structuring element that can have various shapes and that draws conclusions whether the chosen shape can fit or miss the objects in the input image.[9]

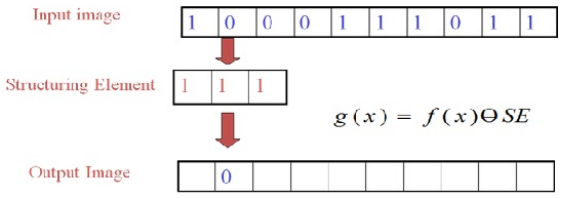


Figure 2.6: Erosion Algorithm. If all the "1"s from the structuring element correspond with the analysed input then the output will be **1** and **"0"**otherwise.

Figure 2.7 shows how the blobs progressively disappear as the number of iterations is increased. Depending on the application this operation can be useful when noise that wasn't removed with a smoothing filter was applied. Also the shape of the kernel has an important impact on the processed image as it will be seen later in section 2.1.4.

(a) 1 iteration **(b)** 3 iterations **(c)** 5 iterations

Figure 2.7: Erosion.

2.1.4 Opening / Closing

The combination of the previous morphological operations (dilation and erosion), form another class of operations called compound methods. The most common used for these methods are the Closing and Opening operations.

Closing operation is very helpful with dealing with problems that may occur while using the dilation technique. As stated earlier, the dilation operation has some drawbacks. As objects are increasing in size they tend to connect with other objects making it hard to analyse the image. The closing operation introduces a straightforward solution to this problem by applying an Erosion operation right after Dilation. This will often separate the old objects bringing them to a way more better segmented form. The method's results mainly depends on the chosen structuring element which varies in shape and size. Figure 2.8 shows the results with structuring elements (SE) of different shapes and sizes. As it can be seen after one iteration, the rectangular gave better results for the closing operation. [10]

However Figure 2.9 shows that by increasing the number of iterations the square

(a) Squared kernel. Size:3x3 **(b)** Rectangular kernel. Size:3x5 **(c)** Circlular kernel. Size:3x3

Figure 2.8: Effect of the Kernel shape on the image for the Closing operation.

structuring element and the rectangular one have the same result, while the circular one still leaves small holes inside the blob.

(a) Squared kernel **(b)** Rectangular kernel **(c)** Circlular kernel

Figure 2.9: Effect of the number of iteration on the image for the Closing Operation, compared to Figure 2.8. Number of iterations: 3.

Opening operation, in contrast is used to compensate the limitations of the erosion method. Decreasing the size that objects have in one scene may be useful when the main goal would be the noise removal. However this will make some of the relevant objects disappear. Opening operation solves the problem in the same straightforward manner as closing but this time the erosion operation will be followed by a dilation one. Therefore the form of the object will not be that altered. [10]

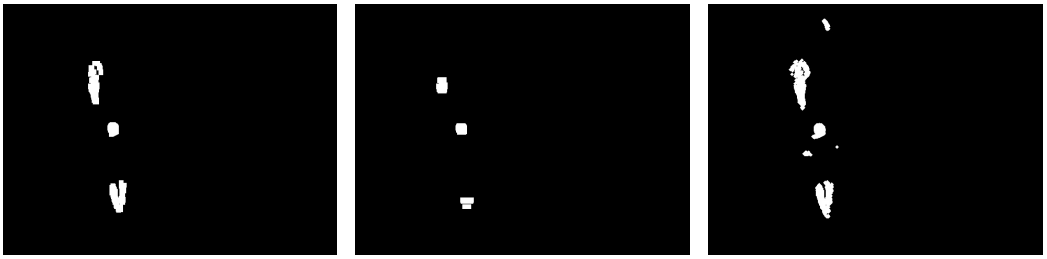
Figure 2.11 shows that contrary to the closing experiment where the square kernel and the rectangle one gave the same result at 3 iterations, the rectangle kernel alters the blobs and eliminates even the smaller blobs that were of interest. On the other hand the circular kernel gave the best results compared to the other two.

2.1.5 Histograms

When one needs to analyse images, video sequence or just particular objects inside, the most frequently used "tool" that provides relevant information regarding these objects is the *histogram*. Histograms can be used in various ways from representing the colour

(a) Squared kernel. Size:3x3 **(b)** Rectangular kernel. Size:3x5 **(c)** Circlular kernel. Size:3x3

Figure 2.10: Effect of the shape of the Kernel on the image for the Opening operation.



(a) Squared kernel. **(b)** Rectangular kernel. **(c)** Circlular kernel.

Figure 2.11: Effect of the number of iteration on the image for the Opening Operation, compared to Figure 2.10. Number of iterations: 3.

distribution of an object, to the distribution of probabilities regarding the location of an object in a scene.

In the computer vision domain histograms play an important role and they find their use in numerous applications. One application would be that histograms can be used to detect transitions that may occur in a video scene checking the changes from one frame to another regarding the edge and colour statistics. The reason for tracing the colour, corners or edges, is that they are important features for the object recognition process.

The main principle that a histogram uses is that it collects and the *counts* of the analysed feature and then organizes and stores them into a set of predefined bins Fig.2.12. Usually the histogram dimensions are fewer than the source data has. Fig.2.12 shows a two-dimensional distribution of points (upper left). A grid is imposed (upper right) and then date points are counted in each grid. The result is a 1-D histogram (lower right). [11]

An important application with the histograms is found in systems that work with matching processes. The comparison of two histograms becomes really useful when, for example, a tracking algorithm is invoked. In order to increase the chances so that the track would not be lost, histogram comparison really comes in handy. There are many ways in which histogram comparison can be done. Here are some of these methods [11]:

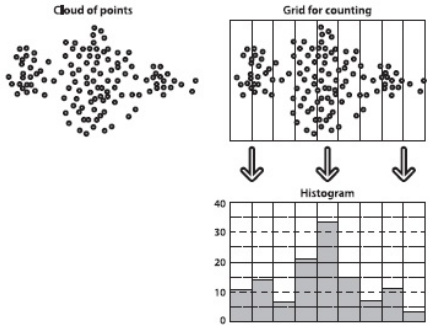


Figure 2.12: Histogram principle.

Source: [11]

Correlation The method is given by formula 2.3:

$$d_{correl}(H_1, H_2) = \frac{\sum H_1'(i) \cdot H_2'(i)}{\sqrt{\sum H_1'(i) \cdot H_2'(i)}} \tag{2.3}$$

Where:

$H_k'(i) = H_k(i) - (\frac{1}{N})\Sigma_j H_k(j)$ and N represents the number of bins in the histogram.

For this method the result of a higher score is better than a lower one. The perfect match is given by a value of *1* and a maximal mismatch is given by a value of *-1*.s

Chi-square method The method is used based on equation 2.4:

$$d_{chi-square}(H_1, H_2) = \sum \frac{(H_1(i) - H_2(i))^2}{H_1(i) + H_2(i)} \tag{2.4}$$

For this representation the lower the score the better the matching. A score of *0* represents a perfect match, and a total mismatch is unbounded (depending on the size of the histogram).

Intersection This method follows equation 2.5:

$$d_{intersection}(H_1, H_2) = \sum_i min(H_1(i), H_2(i)) \tag{2.5}$$

In this case a good score has a high value while a bad score has a small value. If the histograms ar normalized to 1 then a perfect score is *1* and a total mismatch is *0*.

Bhattacharyya distance The method uses the following equation 2.6:

$$d_{Bhattacharyya}(H_1, H_2) = \sqrt{1 - \sum \frac{\sqrt{H_1(i) \cdot H_2(i)}}{\sqrt{\sum H_1(i) \cdot \sum H_2(i)}}} \tag{2.6}$$

For the Bhattacharyya distance low scores represent good matches and high scores bad ones. A perfect match is given by a value of *0* and a maximal mismatch has the value of *1*.

Choosing the correct method is a trade-off between accuracy and speed. According to [11] the intersection method is fast but not that precise while the chi-square and Bhattacharyya methods offer more accurate matchings at lower speeds.

2.2 Object Detection and Classification

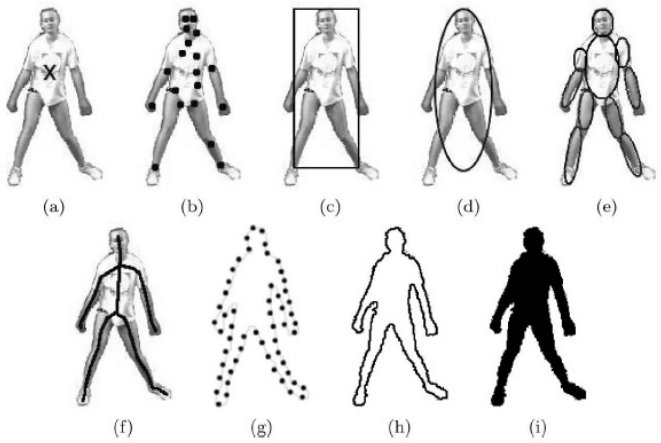


Figure 2.13: Object representations. (a) Centroid, (b) multiple points, (c) rectangular patch,(d) elliptical patch, (e) part-based multiple patches, (f) object skeleton,(g)complete object contour, (h) control points on object contour, (i) object silhouette.

Source: [11]

A major class of image/video processing theory is **object detection**. It implies analysing and scanning an image or a video frame and find the objects inside. There are various methods in which detection can be done depending on the application. The methods can be roughly separated in to classes : *feature-based* methods and *learning-based* methods.[11]

The feature-based methods imply detection an object in one scene by identifying some strong characteristics they might have such as: *edges, corners, colour, texture, contours etc.*. However if the object is more complex, learning-based methods are used. The algorithms are more complex and involve training samples from the object's environment. An example of such a method is the *Viola and Jones Rapid Object Detection* which use a boosted cascade of simple features. Therefore in the detection process an object will be defined as anything that is of interest or relevant for further analysis. Here follows a list of terms that are relevant when proceeding in doing object detection [11]:

1. Points. As it can be seen in Figure 2.13(a) an object can be represented as a

point usually the centroid (centre of mass), or a set of points Figure 2.13(b). This representation is relevant and also very helpful in tracking procedures

2. **Primitive geometric shapes** Objects can also be represented as primitive objects: rectangles, circles, ellipses etc.(Figure 2.13(c,d)). However these primitive rigid shapes are used for representing rigid objects, but are also very useful when tracking.

3. **Object Contour and Silhouette** An object can be very easy to detect and classify using its own boundaries which are also known as *contours* (Figure 2.13(g, h)). Also another relevant representation would be the inner contour which is often called *silhouette*.

4. **Articulated shape models** Articulated objects are composed of parts connected by joints. For example the human body is an articulated object with the limbs and head connected with the torso by joints. This representation uses kinematic models and each connected object can be represented individually by other shapes. For example in Figure 2.13(e), the human parts are described as ellipses.

5. **Skeletal models** They are used by applying medial axis to the object's silhouette.This type of representation seen in Figure 2.13(f) can be used on both rigid and articulated objects.

2.2.1 Blob Analysis

Before starting explaining what this chapter is all about few examples will be shown in order to get a full understating of it. One example can be seen in image below (Fig. 2.14), where an algorithm is needed to figure out how many small objects within a certain area are there in the scene. another

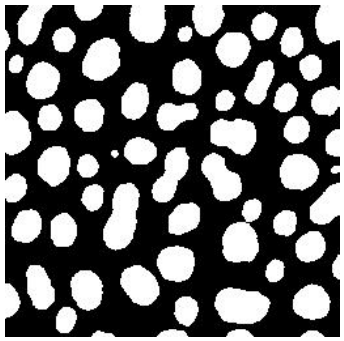


Figure 2.14: Binary image with shapes.

Source: [11]

Another example is presented in Fig. 2.15 where three persons can be seen. An algorithm is needed in order to determine the position of the persons in the image.

Figure 2.15: Binary image with three persons.

A solution for solving such a problem would be to separate the three silhouettes and evaluate them individually. The process is called BLOB extraction. The word stands for Binary Large OBject which basically is a large group of connected pixels in a binary image [10].

The chapter refers to blob analysis by *extracting* and *classifying* the blobs according to some specific features.

BLOB Extraction The extraction step stands mainly in separating different blobs from a given binary image. *Connectivity* determines whether two pixels are in fact neighbours and therefore connected. One of the most common methods in which blobs can be extracted is the **contours** method.

Contours. The contours in an image represent a list of connected pixels that draw a curve in the image. The main goal in tracing such a contour is to give some relevant information about the shapes that objects have in one scene. After extracting the contour of a certain pattern further analysis can be made and accurate conclusions can be drawn. The reason for getting a pattern's contour is that th process of feature extraction will not be that computational intensive, being more efficient if applied on a contour rather than the pattern itself. Some of the contour extraction algorithms can be shown below [12]:

1. **Square Tracing Algorithm** Given an input image with a certain pattern (a group of black pixels on a grid of whit pixels), the idea would be to get the pattern's contour. The algorithm is pretty straightforward: first of all the starting pixel needs to be located. This can be done by starting scanning from the bottom-left corner of the grid and going upwards and proceeding to the right scanning the following column until a black pixel is found. When the pixel is found it will be marked as the anchor. Now there are two situations:

   – every time you encounter a black pixel you turn left

   – every time you encounter a white pixel you turn right

   The Process is over when the anchor pixel is found again.The method is based on

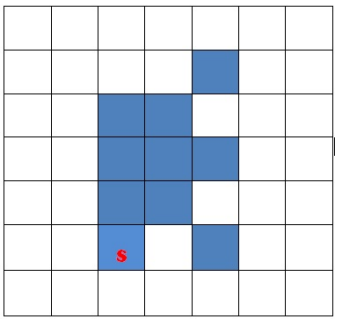


Figure 2.16: Square-Tracing Algorithm.

the "sense of direction", therefore the way you move (left or right) depends on the way that the current pixel was encountered.

2. **Moore-Neighbour Tracing Algorithm** Before starting explaining the algorithm the **Moore neighbourhood** of a pixel needs to be explained. The Moore neighbourhood of a pixel **P** is a set of 8 pixels (P1 to P8) that share a vertex or an edge with it as it can be seen in the figure below.

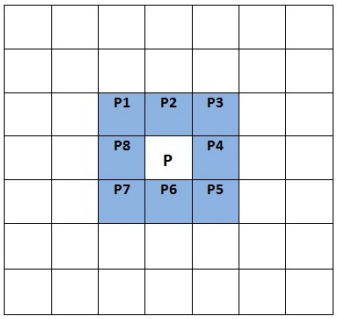


Figure 2.17: Moore Neighbourhood.

Considering the same 7x7 grid with a different pattern the algorithm for finding the pattern's contour will be as it follows. Firstly the anchor pixel needs to be found and this will be done in the same manner as mentioned earlier in the *Square tracing algorithm*. Once the anchor pixel is found the contour will be extracted going around the pattern in a clockwise direction. The idea of the algorithm is that every time a black pixel is hit, backtrack to the white pixel previously visited and start again. The algorithm finishes when you visit the anchor pixel a second time. A good example can be seen in the figure below Fig.2.18.

Another method that is quite used in computer vision science while extracting blobs, is the *connected-component analysis* [13]. The connected-components analysis is a graph theory algorithm that finds connected regions in a binary image. Therefore, given an input binary image, a graph is formed containing vertices and edges. The vertices

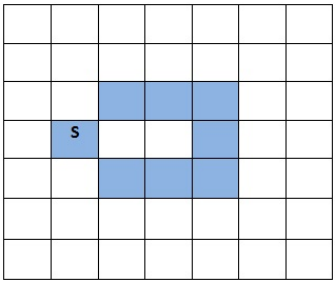
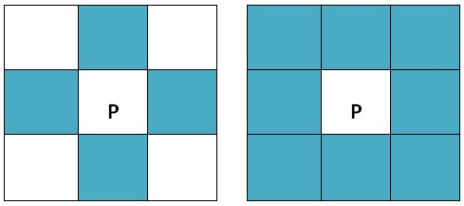


Figure 2.18: Moore-Neighbour Tracing Algorithm.

contain the relevant information about the respective pixel while the edges are represent connected "neighbours". Connectivity is given by how one pixel relates to its neighbours and it can have several types, the most commonly used being the *4-connected* or *8-connected* see Figure 2.19.



(a) 4-connected **(b)** 8-connected

Figure 2.19: Pixel connectivity.

The algorithm is as it follows. Given a binary input image (Figure 2.20) the first foreground pixel (white pixel) is found and all its non-background neighbour pixels. If none of its neighbours is labeled, then the current analysed pixel will be assigned with an unique label. The algorithm moves to the next pixel which has an already labeled neighbour, therefore it will be assigned with the same label as the one of its neighbour.

And if a pixel happens to have neighbours of different labels, then it will be assigned with the smallest label out of all of its neighbours Figure 2.21. After the image has been entirely scanned the labels are merged outputting the resulting blobs.

BLOB Features. Given the above mentioned examples another problem interferes after extracting blobs from an image. *Is it possible to classify a* ***blob*** *as unique or as belonging to a certain class?* The answer to this question is yes, if certain characteristics are taken into account. These characteristics are often called *features* and comparing them will give relevant information regarding the problem of blob comparison. The main idea is to reduce the blobs an transform them into some relevant numbers. Another important aspect would be to ignore the blobs that appear at the border of the image as there is no information whatsoever about the part that stands outside of the frame. There are numerous features that can be extracted when proceeding in doing

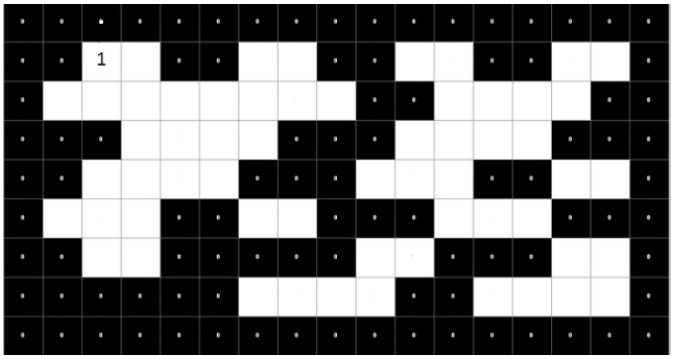


Figure 2.20: Binary image. First non-background pixel detected labeled with *1*

Source: [13]

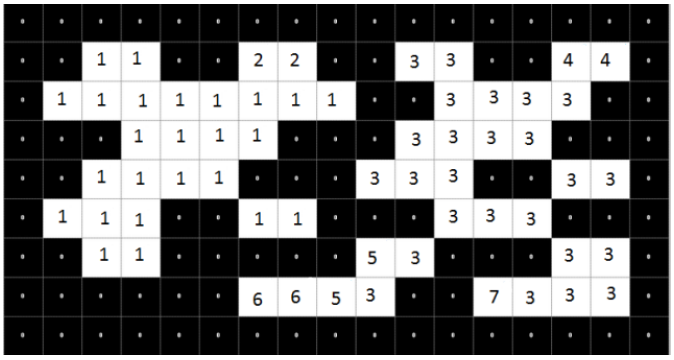


Figure 2.21: Connected-component analysis result

Source: [13]

blob analysis. Here follows some of the most fundamental ones that describe a blob:[10]

- **Area** The area of a blob represents the number of pixels of the entire blob. This feature is very useful when smaller blobs need to be eliminated from the image. For example, when trying to detect people in one scene it is obvious that a certain value will be set so that all the blobs with values smaller than the set one will be discarded.

- **Moments** Usually in computer vision there are certain applications that need some specific information regarding the pixels' intensities from a given image. The weighted average of these intensities are called moments [14] and are usually used as an input for dedicated functions that allows the interpretation o the image. Image moments can describe many things ranging from area of the image or some part of it, to information regrading the orientation. Because blob extraction basically implies getting the contours in order to retrieve blobs in one scene, one way of describing and comparing contours is using the moments. Computing the *moments*

that a contour has, is very valuable when contours from one scene need to be compared. The moment is basically a characteristic of the contour determined by integrating (or summing) over all the pixels that the respective contour has.The (p,q) moment of a contour is defined by the following formula [11]:

$$m_{p,q} = \sum_{i=1}^{n} I(x,y)x^p y^y \tag{2.7}$$

As it can be seen in formula 2.7 the order for x, and y is p respectively q. The summation si done over all n pixels of the contour. Therefore the moment $m_{0,0}$ will represent the length of the contour's pixels. The information that a contour gives is rather rudimentary but it is still an important asset when comparing two contours. In practice *normalised contours* are used, therefore if two contours have the same shape but different sizes the returned values for both contours' moments will be similar. A common usage for the contours would be to help in computing fundamental features that a contour has, for example the centroid:

$$x_{centre} = m_{1,0}/m_{0,0}$$
$$y_{centre} = m_{0,1}/m_{0,0}$$

– **Bounding Box** The bounding box represents the minimum rectangle that contain the blob. It is computed by scanning through all the blob's pixels and finding the minimum and maximum positions on both x and y axis. Having these four values $(x_{min}, x_{max}, y_{min}, y_{max})$ the width and height of the rectangle can be computed using the following formulas [10]:

$$width = x_{max} - x_{min}$$
$$height = y_{max} - y_{min}$$

The bounding rectangle is of major importance as it can be used as a ROI (region of interest) while doing detection/tracking procedures.

– **Bounding Box ratio** The bonding box ratio gives information about the blob's elongation determining whether the blob is long or short. It is computed as the height divided by the width as it can be seen in the following formula.[10]

$$ratio = \frac{height}{width} = \frac{y_{max} - y_{min}}{x_{max} - x_{min}} \tag{2.8}$$

– **Centre of bounding box** The centre of the bounding box is an approximation of the centre of mass and it is computed using the following formulas:[10]

$$x_{bb} = x_{min} + \frac{x_{max} - x_{min}}{2}$$
$$y_{bb} = y_{min} + \frac{y_{max} - y_{min}}{2}$$

– **Centre of mass** The centre of mass is the point on the blob that you could place your finger in order to balance it (for example all the blobs that humans create in one scene will most likely have their centre of mass around their belly button). The centre of mass is a the point whose x value is computed by summing all the x coordinates of all the pixels in the blob and then dividing by the total number of pixels. The process is similar for ty y value. The formulas are shown below : [10]

$$x_c = \frac{1}{N}\sum_{i=1}^{N} x_i \ , \ y_c = \frac{1}{N}\sum_{i=1}^{N} y_i$$

– **Perimeter** The perimeter represents the length of a blob,and it is computed by summing/counting all the pixels encountered along the contour of the blob.

2.2.2 Support Vector Machine

A Support Vector Machine (SVM) is a machine learning technique used for **classification** or regression[2]. It was introduced by Vladimir N. Vapnik and C. Cortes in 1995 [15]. In other words it is a technique that allows to organize data according classes (or category) in order to determine the class of another sample.

Definition In a n-dimensional space, an **hyperplane** is a subset of dimension $n-1$ which separates the space into two half spaces. It can be described as a set of points $\mathbf{x}$ such as

$$\mathbf{w} \cdot \mathbf{x} - b = 0 \tag{2.9}$$

where $\mathbf{w}$ is the normal vector to the hyperplane.

In the following, the algorithm uses a set $\mathcal{D}$ of n elements of class y as a training set.

$$\mathcal{D} = \{(\mathbf{x}_i, y_i) \mid \mathbf{x}_i \in \mathbb{R}^p,\ y_i \in \{-1, 1\}\}_{i=1}^{n} \tag{2.10}$$

The general idea of the **training** (or learning) part is to find a linear hyperplane (of dimension $p-1$, p being the number of classes) that separates each class of the training data (which classes are known, by definition). The **classification** process is to confront the unknown data to this hyperplane and decide to which class it belongs. Figure 2.22 represents that system.

As shown in Figure 2.23, there are infinite possible hyperplanes to classify the data. To optimize this classification, a hyperplane is needed that will maximize the distance to the nearest points on either side of the hyperplane. This distance is called the **margin**.

This is equivalent to find two hyperplanes separating the data with the maximum distance between them. These hyperplanes are respectively as such as: $\mathbf{w} \cdot \mathbf{x} - b = 1$ and $\mathbf{w} \cdot \mathbf{x} - b = -1$. The points belonging to one of these hyperplanes are called **support vectors**. These different items are explicated on Figure 2.24.

[2]Set of statistical techniques to study the relation of a variable to one or several others.

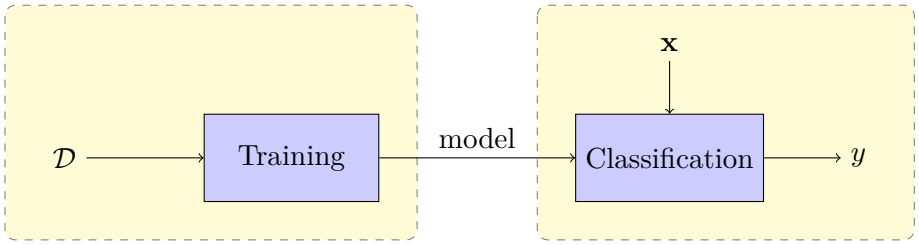


Figure 2.22: Description of Support Vector Machines. The first block, executed once, builds the model according to the training data. The second block, executed as many time as needed, classifies an input data according to the model.

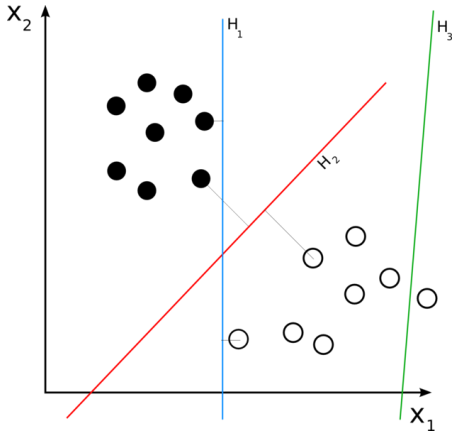


Figure 2.23: Hyperplanes separating data. H1 separates the data but is not optimal. H3 doesn't separate the classes. H2 is optimal. There are infinite hyperplanes like H1.

Source: [16]

To maximize the margin, which value is $\frac{2}{\|\mathbf{w}\|}$, $\|\mathbf{w}\|$ needs to be minimized, with

$$y_i(\mathbf{w} \cdot \mathbf{x_i} - b) \geq 1, \qquad 1 \leq i \leq n. \tag{2.11}$$

2.2.3 Histogram of Oriented Gradients

Histogram of Oriented Gradients, commonly called HOG, is a computer vision technique commonly used to detect all kind of objects. Unlike background subtraction algorithms, it has a region-based approach which makes it robust against geometric and photometric transformations.

The motivation for using this technique is mainly its robustness. Indeed, the training is computed with SVM on a database with hundreds of images. The algorithm is designed to recognize specific kinds of objects, which makes it a good choice for human detection purposes.

Compared to other algorithm, like ASM (Active Shape Model), which uses statistical

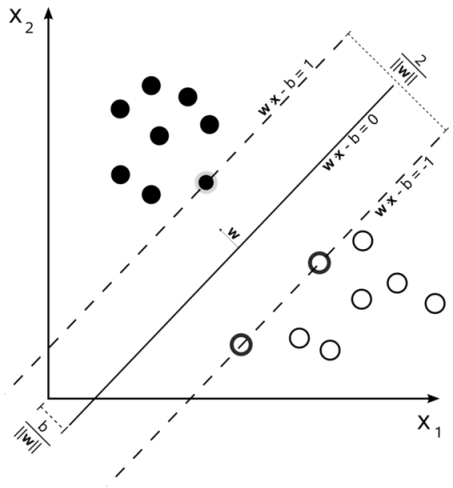


Figure 2.24: Description of Hyperplanes and Margins for Support Vector Machines. Plain line represent the maximum-margin hyperplane. Dashed lines represent the margins. Bold dots are the Support Vectors.

Source: [16]

models and deform the shape of an object to fit a reference image, HOG shows better result in both positive and negative matches (meaning that there are rate of true positive / true negative is better).

Figure 2.25: HOG Detection.

The algorithm was described by N. Dalal and B. Triggs in [2], from the French National Institute for Research in Computer Science and Control (INRIA) at the Conference of Computer Vision and Pattern Recognition (CVPR) of June 2005

It uses gradients magnitude and direction to describe all parts (or cells) of an image. It can be combined with a SVM detector for human detection purposes.

Figure 2.26 represents the different steps to compute the histogram of oriented gradients:

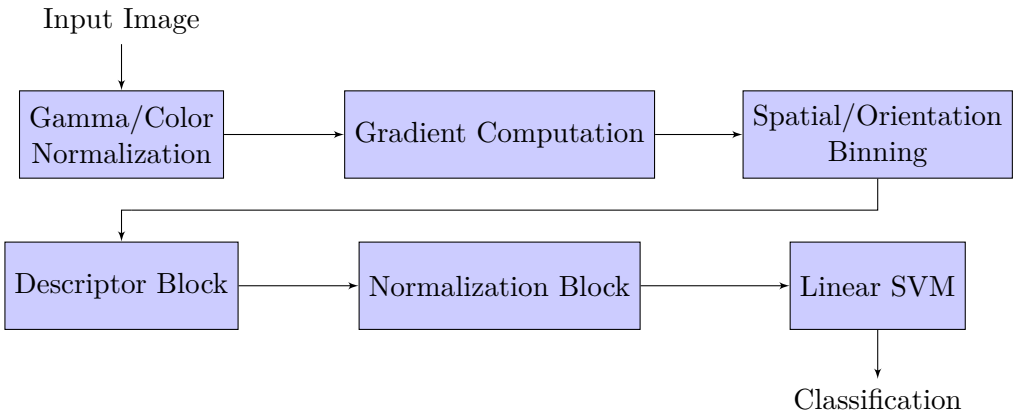


Figure 2.26: Block Diagram of HOG Algorithm.

Gamma/Colour Normalization The main purpose of the Gamma/Colour normalization block is to reduce the impact the light has on the given image (local shadowing and illumination variations). The used images should be in RGB or LAB colour spaces as the coloured images show better results than the gray-scale ones. The normalization itself doesn't improve performance significantly, probably because a similar job is done again later.

Gradient computation The importance of determining the gradients from one image is due to the fact they get information related to contours,silhouettes and sometimes texture. All these aspects are important when defining specific objects such as cars,animals or humans. The gradients are computed by applying some kind of derivative mask over the input image.

However it was showed in [2] that the most effective technique for gradient computation is the application of a 1-D, centred, point discrete derivative mask to filter the colour or intensity of the image. The following masks are used:

$$\begin{bmatrix} 1 & 0 & 1 \end{bmatrix} \text{ or } \begin{bmatrix} 1 & 0 & 1 \end{bmatrix}^T$$

This gives us the gradient at each point, which allows us to obtain the orientation and the magnitude, using the following equations (G_x and G_y and the x and y projections of the gradient). Th following formulas show the **magnitude** (Eq.2.12) and **orientation** (Eq.2.13):

$$G = \sqrt{G_x + G_y} \tag{2.12}$$

$$\Theta = \arctan \frac{G_y}{G_x} \tag{2.13}$$

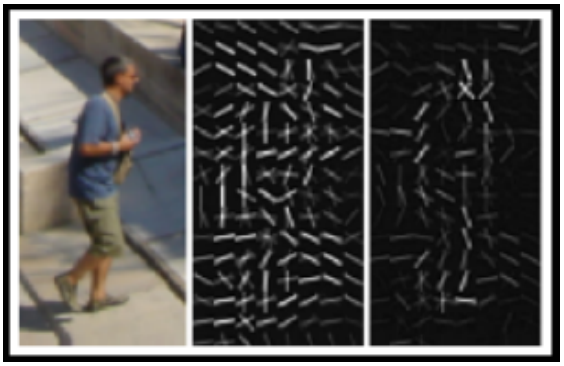


Figure 2.27: Right and middle image show computed gradients at each location, and the brightness is proportional to the gradient magnitude at that location

Source: `people.seas.harvard.edu/~ely/faceparts/serial.html`

Spatial/Orientation Binning. This is the part where the histogram of oriented gradients is built. Each pixel in the image holds a weighted vote for the orientation, based on the gradients computed in the previous step. The image is divided in small spatial regions called *cells* (see Figure 2.28), and for every cell the orientations of the gradients are accumulated in a 1D histogram with a predefined number of bins. Each pixel votes for one of the histogram's bins according to its orientation, and increases it with its magnitude.

These orientation bins are evenly spaced over $0° - 180°$ in case the gradient is "unsigned", or $0° - 360°$ in case the gradient is "signed". The "signed" gradient is of no use for human detection, but shows better results for other object detection. Usage shows that 9-class (9 bins) histograms are sufficient.

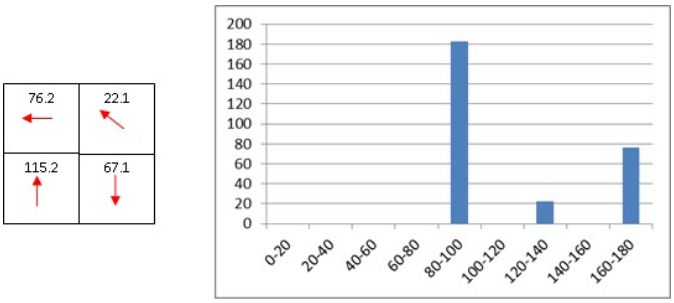


Figure 2.28: A Cell (with orientation and magnitude) and its Histogram.

Normalization and Descriptor Block. As mentioned earlier the normalization process helps in reducing the light variations or shadowing problems. In this step, cells are grouped into bigger units called blocks. Usually the blocks are overlapped and the cells are shared between the blocks and also normalized separately.

Two representations exist for this grouping: the **R-HOG**(rectangular) block and the **C-HOG** (circular) block.

The **R-HOG** block is the default descriptor used in [2]. The R-HOGs are similar to **SIFT** descriptors (Scale Invariant Feature Transform) but they don't align to their dominant orientation. The R-HOGs are computed over $m \times m$ grids (m being the number of cells in each block) of $n \times n$ pixel cells and β histogram bins. Figure 2.29 shows an R-HOG blocks composed of 3x3 cells of 6x6 pixels. However the best configuration is using a 2x2 cells of 8x8 pixels with 9 histogram bins (see [2]).

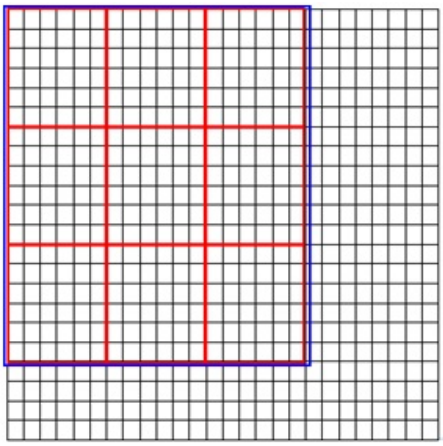


Figure 2.29: Representation of pixels (black), cells (red), and R-HOG block (blue).

There are three methods in which the blocks can be normalized where v is the unnormalised descriptor vector and e which is a normalisation constant that prevents the division by zero:

$$L2 - norm : f = \frac{v}{\sqrt{\|v\|_2^2 + e^2}} \tag{2.14}$$

$$L1 - norm : f = \frac{v}{\|v\|_1 + e^2} \tag{2.15}$$

$$L1 - SQ : f = \sqrt{\frac{v}{\|v\|_1 + e^2}} \tag{2.16}$$

SVM Classifier. Using the created descriptor with a trained Support Vector Machine (See section 2.2.2) classifier allows determining the presence of a specific object, such as a human being. This classifier is binary and looks for the optimal hyperplane which separates matching objects from non-matching ones.

2.3 Video Processing

2.3.1 Segmentation and detection

Video processing uses the same principles as image processing with slight modifications. All the operations are now done on a video sequence which is basically a sequence of images. While working on videos a new notion is introduced that of temporal information, therefore the segmentation and all the analysis will be dedicated towards the *moving* objects in one scene.

Like image processing the first step in analysing a video stream is segmentation. In order to detect and analyse a moving object in the scene it's fair to start with a background subtraction algorithm. This procedure will "discard" the background (which will consist of black pixels) leaving the scene only with the wanted relevant objects also known as foreground (represented by white pixels). However the method is not that straightforward as it encounters a lot of problems while applying it. In an ideal case the background is constant and if the scanned environment is for example a room. All the objects in the scene: walls, tables, chairs are static and, from one frame to another are supposed to stay the same. However as mentioned earlier in the report, illumination drastically changes this "static" aspect and therefore if sun enters the room the algorithm will consider this as "a change" in the scene and unnecessary objects will be detected as positives. If the canning is done outside the control on illumination is even more harder and more complex techniques are used such as background-modelling. Therefore the challenges that should be taken into account while modelling the background consist of handling some major problems:

- **Robustness** against changes in illumination
- **Adaptability** (avoid detecting non-stationary objects such as shadows cast by moving objects, rain snow or moving leaves, therefore adaptability implies the fact that the background should react quickly to sudden changes in the scene.

The most used methods for background subtraction are:

- **Frame Differencing**
- **Mixture of Gaussians** (MoG) [5]

Frame Differencing

It is one of the easiest method to apply when doing background subtraction. The main idea that stands behind this algorithm is that in a video sequence, the present frame will be subtracted from the next one. However a simple algorithm doesn't always imply a good efficiency, therefore the frame differencing method presents some major drawbacks. The first one would be that the algorithm will never detect a new static object introduced in the scene. So keeping the room example in mind, if a person enters the scene it will be detected until the point he/she stops. At that point the person

will disappear and will rapidly become part of the background. Another drawback that the algorithm presents can be observed when scanning an outdoors environment. The problem in this case would be that the algorithm has no control whatsoever on the scene and changes may occur at any moment (leaves moving, clouds affecting illumination). All these changes will create false detections. Also the algorithm is not that sensible to colours, therefore it might not always output a good segmentation result, as it can be seen in Figure 2.30.

Figure 2.30: Frame Differencing Algorithm.

Mixture of Gaussians

The main idea would be to find a compromise between the critical problems that need to be solved while doing background subtraction by modelling the background. One approach would be the Mixture of Gaussians (MOG) or Gauss Mixture Model (GMM) proposed by Stauffer and Grimson [5]. This method belongs to the statistical background modelling techniques. The main idea that stands beneath the GMM algorithm is that, instead of modelling all the pixel values as a particular type of distribution, a particular pixel value is modelled as a combination (mixture) of Gaussians. Knowing the specific parameters for each Gaussian (persistence and variance), it can be determined whether a Gaussian belongs to the background or not. Therefore pixel values that do not correspond to the background will be considered as foreground until there will be a Gaussian that will eventually include them in the background.

The system adapts well at lighting changes, repetitive motions that may occur in the scene or slow moving objects. Because the colour of slowly moving objects has a larger variance than the background, the process of integrating them into the background will take more time.

Algorithm. The method implies creating a model for each background pixel using a mixture of K Gaussian distributions (K usually being a number from 3 to 5). The main idea is that each Gaussian represents a colour from the given scene. The algorithm introduces some weight parameters that represent the amount of time a particular colour represented by a Gaussian is staying in the scene. After this, the algorithm forms an

idea of how the background looks like by an assumption that it contains a number of B highest most likely colours. Therefore, the colours that will stay longer in the scene will be chosen as the ones that form the background. An important aspect for this mixture of Gaussian method is that the system adapts to changes in illumination due to an update scheme. If a new pixel enters the scene, its value will be checked against the ones that already form the background model. The colours are checked according their corresponding cluster. The static colours tend to have tight clusters while moving colours have wide clusters because of the different reflection it has during the movement. If a matched is found after doing the checking, then the background model will be updated accordingly. On the other hand if no match is found, then the system will add a new Gaussian component to the scene [5].

The changing values of an individual pixel are referred to as a pixel process, and for each pixel there is a recent history of all its changes $X_j \dots X_t$ that is modelled by a mixture of K Gaussian distributions. The probability of observing the current value of one pixel is given by formula 2.17:

$$P(X_t) = \sum_{i=1}^{K} w_{i,t} * \eta(X_t, \mu_{i,t}, \Sigma_{i,t}) \tag{2.17}$$

Where:

- $w_{i,t}$ is an estimate of the weight showing the proportion of the data used for the i-th Gaussian ($G_{i,t}$) at time t.
- $\mu_{i,t}$ is the mean value of $G_{i,t}$.
- $\Sigma_{i,t}$ is the covariance matrix of $G_{i,t}$.
- η is the Gaussian probability density function given by the formula 2.18:

$$\eta(X_t, \mu, \sigma) = \frac{1}{(2\pi)^{\frac{n}{2}} * |\Sigma|^{\frac{1}{2}}} * e^{-\frac{1}{2}(X_t-\mu_t)^T \Sigma^{-1}(X_t-\mu_t)} \tag{2.18}$$

Choosing the number of Gaussians used (K) depends on the available memory but also on the computational power that the algorithm should use. Usually the values range between 3 and 5. Because of computational reasons, it was assumed that the RGB colour components are independent and have the same variances; therefore the covariance matrix of this form would be of this form (equation 2.19):

$$\Sigma_{i,t} = \sigma_{i,t} I \tag{2.19}$$

The foreground detection is as follows . In the beginning the designed system that is using K Gaussian distributions will be initialized with some pre-defined parameters such as the mean, variance (which is set to a high value), and weight (set to a lower value). Then, while observing a new pixel, in order to determine its type (whether it's foreground or background), the RGB vector is checked against the K Gaussians until a match is found. A match is considered to be a pixel value within a standard deviation of 2.5 from any of the background distributions [5].

Having the above mentioned initialization, the system is then updated after equation 2.20:

$$w_{k,t} = (1 - \alpha)w_{k,t-1} + \alpha M_{k,t} \tag{2.20}$$

Where:
- α is the learning rate;
- $M_{k,t}$ is 1 if a match is fond an 0 otherwise.

After this step the weights of the distribution are normalized and the remaining parameters (mean 2.21 and variance 2.22) are updated as well:

$$\mu_t = (1 - \rho)\mu_{t-1} + \rho X_t \tag{2.21}$$

$$\sigma_t^2 = (1 - \rho)\sigma_{t-1}^2 + \rho(X_t - \mu_t)^T(X_t - \mu t) \tag{2.22}$$

Where:

$$\rho = \alpha\eta(X_t|\mu_k, \sigma_k) \tag{2.23}$$

The main advantage of this updating method is that whenever changes are made in the scene (movement in general) the previous state of the background won't be discarded. In fact the original colour is kept in the mixture until it becomes the K-th probable and a new colour is observed. That means that if a moving object becomes stationary for a period of time enough to become part of the background, and then it moves again, the previous background still exists and has the same μ and σ^2 but the weight w will be lower eventually being re-incorporated into the background.

In order to determine whether the new pixel is corresponding to the background or foreground the K Gaussian distributions are sorted by the value of w/σ . An ordered list is therefore created which will contain, from top to bottom, the most probable backgrounds. After that a number of B distributions are chosen to model the background, where B is given by equation 2.24:

$$B = argmin_b(\sum_{1}^{b} w_k > T) \tag{2.24}$$

And T represents the minimum proportion of the pixel data that is needed to model the background. If T is small then the background is uni-modal. This means that only the most probable distribution is used. If T is chosen to have a higher value then the background is using a multi-modal distribution (because of moving objects that belong to the background such as leaves, a flag in the wind etc.). Therefore the background model will allow the background to save more separate colours that will be integrated.

The figure below (Fig. 2.31) shows the Gaussian Mixture Model algorithm done on the same video seen in the frame differencing algorithm:

Figure 2.31: Gauss Mixture Model Algorithm.

Figure 2.32: Gauss Mixture Model Limitations.Two persons in the scene are detected as one blob

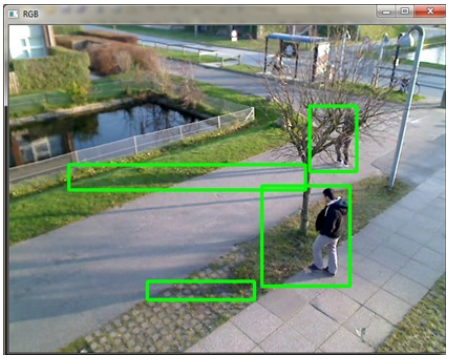


Figure 2.33: Gauss Mixture Model Limitations.Bad detection in a sunny scene.Shadows detected as positives

Algorithm Limitations. Also, as mentioned earlier in the introduction chapter, there are some limitations for using this algorithm. The shadow caused by the sun or other light sources will create unwanted blobs that will be detected as a human (Figure 2.33). Another problem rises when two or more passengers walk together the blobs that each other form will be connected (Figure 2.32); therefore that blob will need a thorough

analysis in order to determine how many persons are in that group or to establish whether or not that blob is indeed formed by a group of people.

2.3.2 Filtering/Tracking

A major part of the Tracking process (detailed in chapter 4) is the prediction. This part uses data filtering as used in electronic systems to estimate the position of the people.

The most commonly used method used for tracking is the *Kalman filter* described by R.E Kalman in 1960 [17] in as a solution for filtering linear data, and it has been developed ever since. What the filter actually does is to estimate the state of a linear dynamic system from a set of noisy measurements. The idea is that the filter smooths the measurements by weighting them against their predicted values by their variances.

The filter will output estimates regarding the unknown values and will also give information about the estimates' uncertainties.

Because it is quite hard to retrieve accurate results regarding precise measurements, it is fair to analyse the behaviour of the Kalman filter by it's gain. This gain is a function of relative certainty and current measurement estimate and it can be "modelled" in such manner so you can be able to "set" the system's performance. A high gain will focus more on the measurements and therefore will follow them more closely. Using a low gain, noise will be smoothed and the predictions will have more weight when followed. If the gain is set to **1** then the state estimate will be fully ignored. On the other hand if the gain is set to **0** the measurements will be the ignored ones.

The estimation process for the Kalman filter is recursive and can be run in real time. This recursive aspect means that the in order to compute the current estimate only the current measurement and the estimated previous step are needed.

The Kalman filter assumes that all the states and noise have a Gaussian distribution. The Kalman filter uses state vectors to analyse how the system's behaviour changes in time, and based on how the state output looks like the corresponding measurement will be predicted. Therefore the Kalman filter will describe the current state derived from the previous state according to the following formula [17]:

$$x_k = F_k x_{k-1} + B_k u_k + w_k \tag{2.25}$$

Where:

- x_k is the current state

- F_k transition model matrix applied to the previous state F_{k-1}

- B_k is the control input matrix (if you there is control on the analysed object) applied on the control vector u_k

- w_k is the process noise drawn from a zero mean multivariate normal distribution. $w_k \sim N(0, Q_k)$ where Q_k is the covariance matrix of the process noise.

At time k a measurement of the current state is made according to the following formula:

$$z_k = H_k x_x + v_k \tag{2.26}$$

Where:

- z_k is the measurement or observation

- H_k is the observation model matrix that maps the true state x_k in the measurement space.

- v_k is the observation noise. v_k $N(0, R_k)$ where R_k is the covariance matrix of the measurement noise.

The two distinct phases of the Kalman filter are the *Prediction* and the *Update*. The prediction step uses the previous state estimate in order to detect the current state estimate. this process is also known as the **a priori** state estimate because even though it determines the current state it doesn't contain measurement information about it. The update phase combines the prediction with the measurement in order to output an accurate estimate. This last estimate is also known as **a posteriori** state estimate.[17]

The prediction phase is represented by formulas of the predicted a priori state eq.2.27 and the predicted estimate covariance eq.2.28:

$$X_{k|k-1} = F_k X_{k-1|k-1} + B_{k-1} u_{k-1} \tag{2.27}$$

$$P_{k|k-1} = F_k P_{k-1|k-1} F_k^T + Q_{k-1} \tag{2.28}$$

The update phase is represented by the formulas of the Kalman filter gain eq. 2.29, the updated state estimate eq.2.30 and the updated estimate covariance matrix eq.2.31 [17] [18]:

$$K_k = P_{k|k-1} Hk^T (H_k P_{k|k-1} H_k^T + R_k)^{-1} \tag{2.29}$$

$$X_{k|k} = X_{k|k-1} + K_k (z_k - H_k X_{k|k-1}) \tag{2.30}$$

$$P_{k|k} = (I - K_k H_k) P_{k|k-1} \tag{2.31}$$

Chapter 3

Detection

3.1 Introduction

The previous chapters stated the problem formulation, system requirements and the description of all the theoretical concepts that were used in the system's implementation. The current chapter describes the first part of the algorithm, describing in details the **Human Detection** step. The video caption will be taken in an *outdoor/indoor* environment from a camera placed at a **certain height around 3 meters** parallel with the side-walk. After the capture is taken the system will determine the correct ***detections***.

Definition A correct **Detection** represents an extracted blob ,from the given scene, that has a certain set of features describing a human, as it will be seen further in the chapter.

3.2 Design

According to the system presented earlier in Chapter 1 this part will only handle the implementation in detail of the *Detection* block.

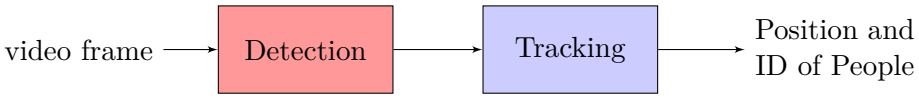


Figure 3.1: System representation in blocks

The detection part of the system's framework is simple and straightforward it implies the usage of to major blocks that output a robust fast human detection. As it can be seen in *Figure 3.2* the first step would be a ***segmentation*** approach which enables the blob detection after a background subtraction algorithm is applied. The block will output filtered blobs with some specific characteristics that will further help in the system's development. For example the bounding rectangle of a specific blob will be used a region of interest on which the ***Region Detection*** block will be used. The ***region***

based approach will validate whether the analysed blob is a human or not and if so the tracking process can continue, otherwise the blob will not be taken into account and therefore no tracking will be done on it.

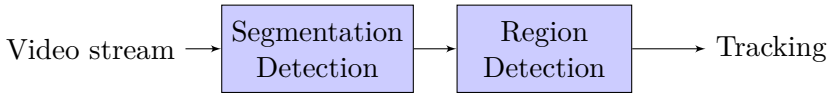


Figure 3.2: Detection process representation

3.2.1 Setup

As mentioned in section 1.4, the system is supposed to be as adaptive as possible. For this reason, the system has simple requirements:

Video Camera The Video Camera used is a simple 2 Mega Pixels web cam, manufactured by Logitech®, recording 15 frames per second. See Fig.3.3.

Figure 3.3: The Logitech® Pro 9000, used for recording the video sequences.

Source: `http://www.logitech.com`

Computer Personal laptops are being used for running the software. Typical configuration can be as follow:

- **Processor** Intel® Centrino® Core™ 2 Duo T8100 @ 2.10GHz.
- **RAM** 4.0 GB DDR2.
- **Operating System** Windows® 7 64bits.
- **Graphic Card** NVIDIA® GeForce™ 8400M GS.

3.2.2 Segmentation Detection

As it can be seen in *Figure 3.4* the detection part is realised in a straightforward way. Firstly a background subtraction algorithm is applied on the input video in order to fully separate the foreground from the background. The second step consists of the preprocessing algorithms which prepare the foreground image for the blob extraction step. The output of this sub-system will be some well defined human blobs.

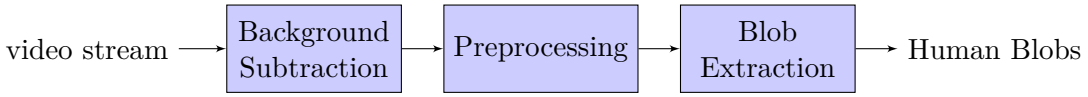


Figure 3.4: Segmentation detection representation

Background Subtraction

At the beginning of this part many trials have been made in order to gain a good and robust background subtraction algorithm (see 3.4.1). However as it can be seen in the *Segmentation and detection* subsection of *Chapter 2* the Gaussian Mixture Model (*GMM*)[5] gave the best results.

Figure *3.5* shows how the background subtraction algorithm works. As mentioned earlier in subsection *Segmentation and detection* in *Chapter 2* the method adapts very well to lighting changes and other movements belonging to the background. Each pixel will be modelled according to a mixture of *3* Gaussians. According to the pixel's parameters (weight and variance) it will be determined whether it belongs to the background or not. After this step the foreground objects can easily be detected. An important aspect for this algorithm stands in the update step as some of the foreground objects will stay longer in the scene therefore becoming part of the background and therefore a new model is needed as a base. The output will be processed by the preprocessing block in order to get a good detection.

The system works only on cloudy scenarios as sun creates unwanted shadows in the scene that would lead to bad detections.

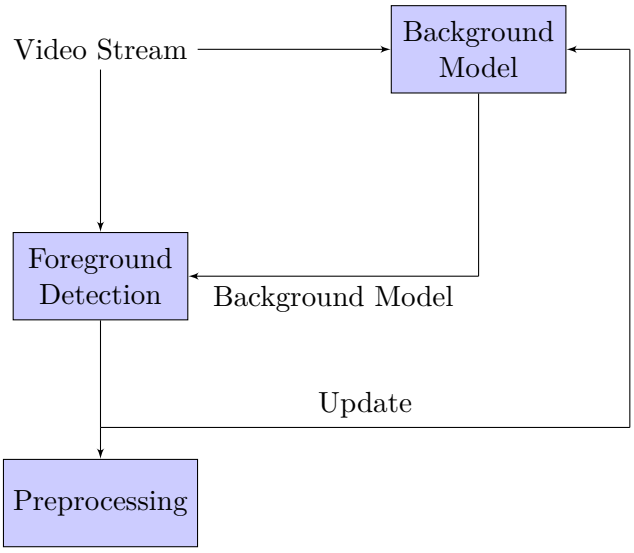


Figure 3.5: Background Subtraction Diagram

Preprocessing

The main goal of video processing is to obtain clear and relevant images that can be further processed to extract certain features that could describe an object. Various algorithms are used for processing the input frames such as: thresholding (to obtain the binary images), morphology algorithms (opening, closing, dilation and erosion) , smoothing(used for clearing the images from unwanted noise).

Figure 3.6: Preprocessing

The order of the processing algorithms is as it can be seen in *Figure 3.6*. Firstly a closing operation is used in order to fill the possible small holes inside a blob. After closing a Median filter is used (in order) followed by a Gaussian filter is used in order to clear the area of unwanted noise. Although a combination of the two is not that fast it does not alter the resulting blobs compared to the cases where the filter were used independently. Also combined with other operations it outputs the best results *Figure 2.3*. The last operation of the preprocessing block is an alternation of the dilation and erosion algorithms which proved to output well shaped blobs that could be further processed for determining whether they are human or not.

Blob Extraction

After the background subtraction and the preprocessing steps are done the system can finally determine whether certain blobs are human or not according to a certain set of features that will also help in the tracking framework. The way in which characteristics of the new processed blobs are extracted can be seen in *Figure 3.7*. Firstly the contours of the blobs are retrieved. This measure helps a lot in getting the actual features. As seen in subsection *Blob Analysis* of *Chapter 2* there are many ways in which constraints can be added to a blob. The first important feature used for detecting a human blob is the ***blob's area***. It has been determined that within a specific range of 1000 pixels a blob will be considered "*human*". The next important aspect that is taken into account is the bounding rectangle of the blob along with its own parameters (area, ratio). The bounding rectangle is used for the detection improvement as it represents the region of interest on which the *Histogram of Gradients* (HOG) algorithm will be applied to (see paragraph 3.2.3). The last feature extracted is the blob's centroid (centre of mass). The centroid is a crucial parameter as it will be used further in the tracking process. The *Kalman filter* will use this point as an input and all the measurements will be done according to it's coordinates (see paragraph 4.2.3).

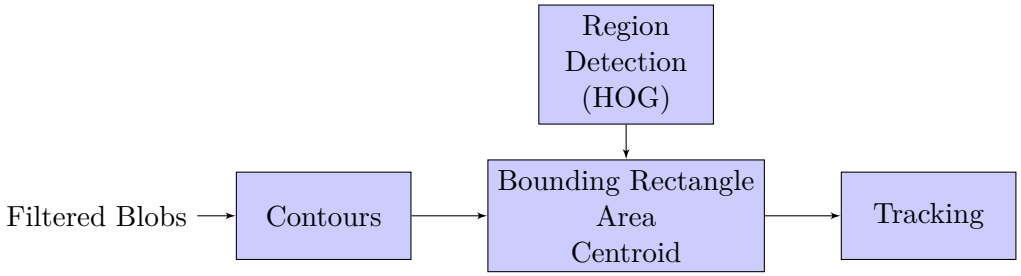


Figure 3.7: Blob extraction Framework

3.2.3 Region Detection

As mentioned in the previous chapter, the **Histogram of Oriented Gradients** [2] is a robust object detector, which is able to detect various types of shapes, ranging from people to animals or vehicles. However its main purpose is to detect human, which makes it even more fitting the needs of the system. The algorithm is a bit complex but rather intuitive: given an input image the HOG algorithm uses a 64x128 pixels detection window that is slid through the image in order to output a set of vectors which will be confronted to a *Support Vector Machine*(SVM) to determine whether the respective "scan" belongs to a human class or not.

The traditional usage of this algorithm is to run it on the entire image and scale the detection window to be able to find objects at different depth in the image. However, this process is an intensive computation and takes a lot of time. See paragraph 3.4.2 for performance explanations.

As can be seen in Figure 3.7 the proposed system will use the *HOG* in combination with segmentation detection to reduce the computation. After blobs are detected and filtered, their bounding box is used as a region of interest, and apply the *HOG* to this part of the image. This significantly increase the speed of the algorithm. The interest of adding the *HOG* to the detection is that it deals with the classification. This means that only humans will be found by this algorithm.

In conclusion, the blob analysis narrows down the work for the *HOG* to classify each blob as human or not.

3.3 Implementation

The system was implemented in the C# language using a wrapper for the *OpenCV* library called *EmguCV*. This library was used due to the fact that it provides the user with almost all the computer vision functions ranging from video/image processing to machine learning algorithms.

> NB: For clarity in the diagrams, the Properties (which are a specificity of C#) are simply represented as attributes with capital first letter. You can assume for all UML diagrams that in reality they have getters and setters.

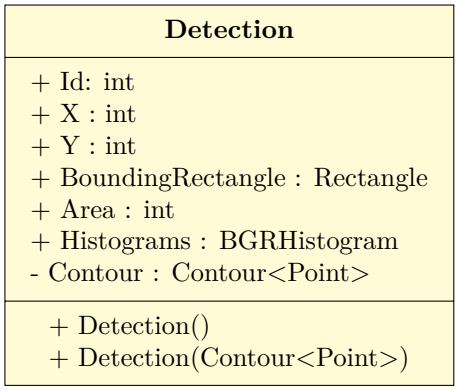


Figure 3.8: The Detection class diagram

As mentioned in the Design section the detection process takes as input the processed foreground mask provided by the Gaussian Mixture Model background subtraction, and it returns several features retrieved from detected blobs that are confirmed to be human. These features will later be fed inside the tracking block. As it can be seen in the *Detection* UML diagram in *Figure 3.8*, the class provides the following attributes: the centre of the detected blob represented in the diagram by the two integer coordinates *X* and *Y*. Each detection has an unique ID which is automatically incremented. Its longevity is always of 1 frame, since new detections are made at each frame independently form the previous. The class also provides the bounding rectangle of the detected blob which is of type *Rectangle*. Another feature given by this class is the area of the blob (of type *integer*) which will be used, according to a certain threshold, to select the blobs that are of interest. One of the most interesting features provided by this class is the colour histogram of each RGB blob (type *BGRHistogram*, see B.6) which will be useful to maintain the unique characteristic of the detection.

3.3.1 Segmentation Detection

The segmentation detection was implemented according to the UML diagram presented in *Figure 3.9*. As it can be seen in the diagram two *interfaces* were created for implementing the segmentation detection: **ICaptureManager** and **IFrameManager**. The **CaptureManager** deals with capturing the video input. Each frame of the respective video will be stored as an RGB image. It provides some of the video stream properties that can be displayed for information or used in some algorithms: the frame size, *FrameSize* and the number of frames per second (FPS) that the camera runs with, *FramePerSecond*. The *Progress* property has an informative purpose to show the progress of the video and is used in the user interface.

The **FrameManager** interface deals with every frame belonging to the video input. This interface takes an RGB image as input (the frame) and also the foreground mask which is a binary image obtained from applying the Gaussian Mixture Model background subtraction on the input video. This interface loads all the needed parameters

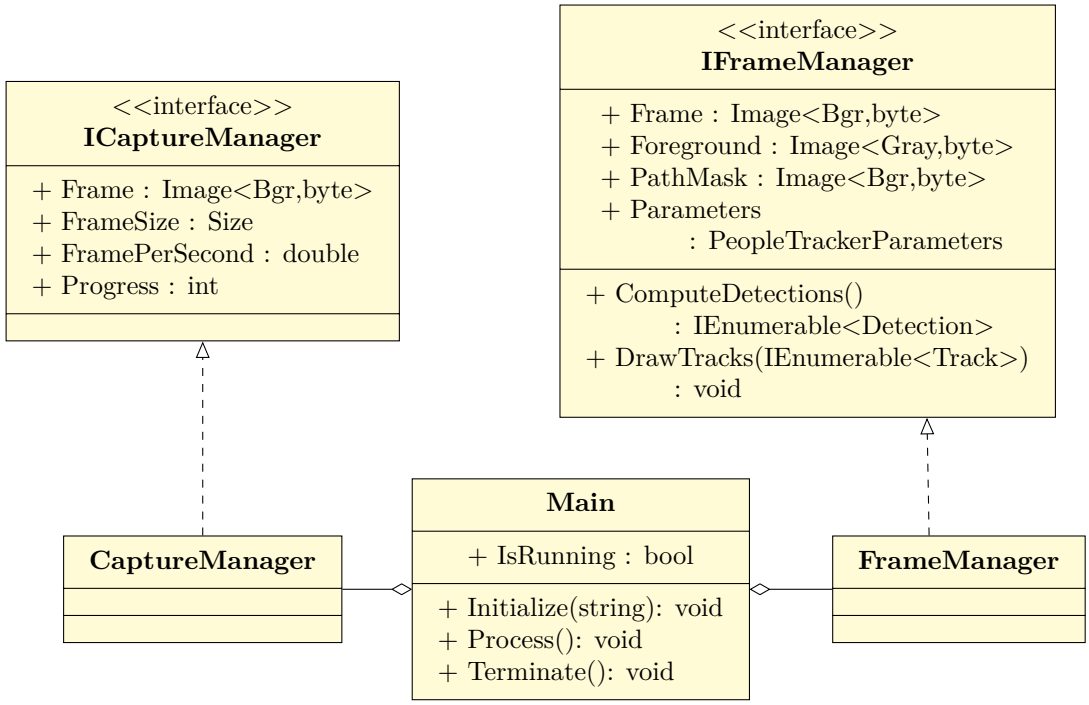


Figure 3.9: UML diagram of the segmentation-detection process.

for both background subtraction algorithm and preprocessing steps. Therefore after the foreground is extracted, several operations are applied in order to get a reasonable foreground image. The operations implementation have been implemented as in such way they can be reorganized (added, moved, modified or deleted) at runtime. For this to be possible there is an interface **IOperation** and a concrete realization of it for each operation (Gaussian Filter, Median Filter, Closing, Opening...). See Appendix **??** for more details on this implementation.

Each operation relates to a function in OpenCV and its equivalent in the wrapper, EmguCV.

Morphology Operations OpenCV function *cvMorphologyEx*, relating to *Image._MorphologyEx* in EmguCV, can apply morphology operations. A flag of type *CV_MORPH_OP* determines which operation will be executed: *CV_MOP_CLOSE* for the closing, *CV_MOP_OPEN* for the opening, and others not relevant. The Structuring Element of the operation is described by a *StructuringElementEx* structure, containing the size and shape of the element.

Gaussian and Median Filter Respective functions *Image._SmoothGaussian* and *Image.SmoothMedian* (*SmoothGaussian* and *SmoothMedian* in OpenCV. For each function the size of the kernel needs to be specified.

Dilation and Erosion The functions *Image.Dilate* and *Image.Erode* are used (respectively *cvDilate* and *cvErode* in OpenCV), with the number of iterations specified.

3.3.2 Region Detection

As mentioned in paragraph 3.2.3, the Histogram of Oriented Algorithm deals with the classification of humans, and uses a *Support Vector Machine* for that purpose. This *SVM* needs to be trained on a set of images; this system uses the data provided by the authors of the *HOG* algorithm [2], freely available [19]. It is trained on 2416 positive images cropped to 64x128 and centred on standing humans, and on 1218 negative images randomly sampled 6 times, for a total of 7308 negative windows. It is important to have a large number of negative images compared to positives because it prevents false positives detections [20], which is the most unwanted behaviour. However, with this configuration, false negative rate will increase too. To avoid that, a combination of HOG and blob detection is used again, but this time in a different way: blobs found after contour analysis are tracked, and a score for the number of positive hits of the *HOG* is kept. This allows to determine after just a few frames (which does not need to be big since false positive are rare) that a blob is really a person.

The used **SVM** implementation is the one recommended by the authors of the *HOG* [2]: *SVMLight* [21][22]. This implementation is separated in two programs: *svm_learn*, which is the learning module, and *svm_classify* which is the classification module. Only the learning module matters in this system, since the classification is done inside the *HOG* algorithm. The *svm_learn* utility is used in linear regression mode as suggested by [**?**]. This utility outputs a file with information like the number of features, the number of training data, and the number of vectors. Then the actual feature vectors are given. Example for 500 positive and 1200 negative image, with a 64x128 window size for the HOG: 1344 support vectors of 3780 features each.

The generated SVM model contains n support vectors of m features. For and $j \in m$, each line of the model is the represented as follow:

$$\alpha_i \ 1 : X_{i,1} \ \ 2 : X_{i,2} \ \ \ldots \ \ m : X_{i,m}$$

where $i \in n$, X_i is the *HOG* feature vector at line i, and α_i represents the class of X_i.

However, the HOG implementation given by OpenCV that is used in the system requires a single 3780 features vector. Here is how the n support vectors are transformed to a single one:

$$\sum_{i=1}^{n} \alpha_i X_i \tag{3.1}$$

where

OpenCV provides an implementation of the Histogram of Oriented Gradients, which is also available from the EmguCV wrapper, through the *HOGDescriptor* class. An object of this class can either be constructed from the default OpenCV parameters or from custom parameters (see paragraph Testing for more details on the parameters). Two functions are available for the actual computation of the HOG. The first one,

HOGDescriptor.Compute gets an array of *float* which is the feature vector of the input image, with no scaling or sliding. This function is used to generate the input values of the SVM. The other available function gets an array of *Rectangle*, representing the positions of the detected people in the input image. This function scales and slides the window, either with default parameters from OpenCV, either with specified parameters (see paragraph Testing for more details).

This last function is used for the actual detection and classification. It is noted that default parameters cannot output good results since they are designed to work for a whole image, and not a region of interest, meaning they are optimized to work in that case (difference of sliding and scaling mainly).

3.4 Testing

The most important thing while running and assessing a system, is the testing process. This section will be separated in two main subsections: *parameter testing* and *algorithm testing*. The parameter testing subsection will show why the applied methods (background subtraction, pre-processing) are using specific values for their parameters and will therefore justify the chosen values. The algorithm testing subsection will show how the overall functionality of the detection system and present some of the limitation and problems that occurred while running it.

3.4.1 Parameter testing

Gaussian Mixture Model The sequences used for testing are 640x480 images retrieved from an un-calibrated RGB normal webcam. The first used parameter ,as it can be seen in *Table 3.1*, is the window size which gives the learning rate ($\alpha = 1/windowsize$ where *window size* represents the default history measured in frames) which is set to a value of *0.001*. It had been observed that the lower α's value the more robust the algorithm. Increasing this learning rate means using fewer frames to learn therefore, for an α value of *0.01*, choosing a value of *100* for the *window size* will make the algorithm work a lot faster. However the background model will be rather unstable. In an outdoor environment illuminations changes occur frequently and also objects belonging to the background often move (tree leaves, the presence of water that creates waves) and using such a big learning rate will lead to unwanted foreground. Several tests have been done on the testing video and it has been shown that even by using the default value of *0.002* for α is not enough to obtain a reliable foreground as a static person from the scene will be lost quickly becoming part of the background.

The next parameter is the number of Gaussians needed to model the background. Testing multiple values showed that choosing a number of only two Gaussians will output a really bad foreground. The proposed system uses the default value of *5* Gaussians as an increased number (e.g. *7*) will not show any kind of difference in the foreground image. The background threshold represents the T value in formula *2.24* from the *Video Processing* section in *Chapter 2* representing the number of highest probable colours used

Parameter	Value
window size	1000
number of gaussians	5
background threshold	0.5
standard threshold	2.5
minimum area	15
initial weight	0.05
initial variance	30

Table 3.1: Gaussian Mixture Model Parameters

when modelling the background. The purposed algorithm used the default value of *0.7* for the threshold value as it gave the best results. The rest of the parameters' values were chosen according to their default values.

Preprocessing steps The preprocessing steps were tested empirically and the order in which they were applied was done in the same manner. It has been observed that applying on the foreground mask a closing operation before filtering will provide better blobs for the later detection. Using only the median filter (with a kernel size of *3*) showed a good noise removal but the alterations of the blob was the major drawback. However used in combination with a Gaussian filter with a kernel size of *5* the retrieved blobs aren't altered that much and noise is still removed. Then dilating the blobs two times, eroding them 5 times and then re-dilating them two times, will output optimal filtered blobs ready to be processed.

Histogram of oriented gradients Because most of the implementation of the entire system is mainly based on the *OpenCV* library, the algorithm is constrained by several in-built functions. The *Histogram of oriented gradients* method, is one of that exceptions. The used parameters can be seen in table 3.2.

Parameter	Value
block size	16 x 16
cell size	8 x 8
block stride	8 x 8
number of bins	9
window size	64 x 128
window stride	32 x 32
scale	1.04
hit threshold	0.4

Table 3.2: Histogram of Oriented Gradients Parameters

The block size represents the size of the *R-HOG* presented earlier in section 2.2.3 in the block normalization step. The respective block will consist of four cells each of 8x8 pixels. The block stride represents the overlap between the *R-HOG*s so that one cell can provide more values when blocks are normalized. As mentioned earlier in this paragraph the Histogram of Oriented Gradients algorithm used for this system is limited due to the library's implementation. Therefore the three above mentioned parameters are set according to the values from table 3.2. The respective values cannot be modified in any way as the algorithm won't function. Their values were chosen according to Dalal and Triggs tests in [2]. Also according to [2] the best chosen number of bins for the orientation and binning block from the *HOG*'s framework (see 2.2.3) was set to *9* as above values showed no difference whatsoever in the algorithm's performance. The window size represents the size of the detection window, and was set to 64x128 as recommended in [2]. Dalal and Triggs paper show various types of window sizes depending on the application and many others have been established. The detection window is slid on to the image and can be overlapped over the previous scanned area. This overlap factor can be set with the window stride value which in this case was chosen to 32x64 for it gave the best results for a 640x480 video sequence. The scaling factor determines how much the image and window size are scaled to match smaller objects. A value of 1.2 is optimal for this system because it implies a small number of scaling, thus computations.

The hit threshold represents the threshold for which a HOG descriptor is considered to be a "hit" (a true positive) representing a human. It is basically a distance from the HOG feature vector to the SVM plane see 2.2.2. It was shown the implemented system a value of *0.4* gives the best hits in one sequence.

3.4.2 Algorithm testing

For determining the robustness of the system, multiple videos were used as input in order to challenge it. This was done firstly to see where the theoretical notions, that stated that a particular method will be optimal for this application, will fail. Secondly, usage of multiple videos taken will definitely provide some answers regarding what is the "perfect scenario" for the implemented system.

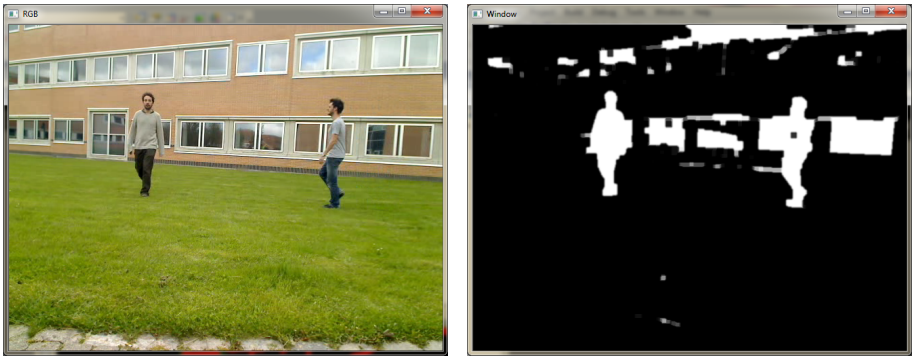
Figure 3.10 shows one of the scenarios where, when parameters presented earlier in 3.4.1 were used, the system still gave bad results. Illumination proved to have a major impact when background subtraction was performed and even though the *default history* (the frames needed to model the background) was chosen with a value of *1000*, the sudden illumination changes that occurred in the image output blobs that could not be classified as humans.

Although the system should have adapted (Figure 3.10 shows the beginning of the video) after several frames, the reflections in the windows constantly changed during the runtime outputting more bad results. Figure 3.11 shows that even with pre-processing the *human* blobs connect to the windows reflections making it impossible to detect the humans and to track them afterwards.

However under different conditions, were there were no window reflections or major

(a) Original RGB image **(b)** Foreground Mask

Figure 3.10: Illumination problems at optimal parameters.



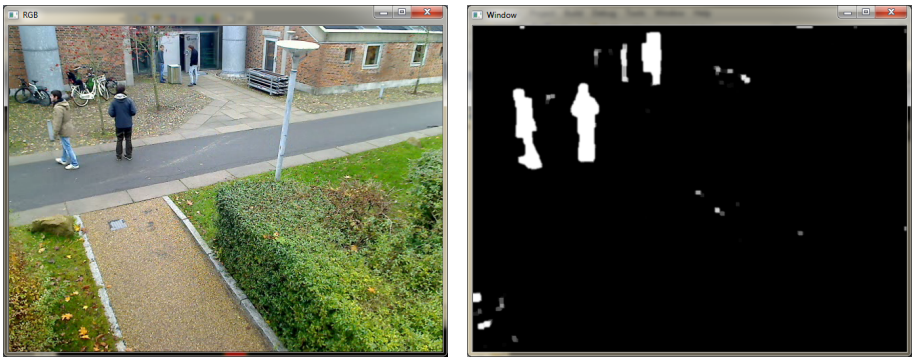
(a) Original RGB image **(b)** Foreground Mask

Figure 3.11: Illumination problems at optimal parameters.

illumination changes, the system worked perfectly with the same parameters as the ones used in the previously presented video. Besides some barely visible noise, Figure 3.12.shows how a totally cloudy environment provides good results that correspond with the system requirements.

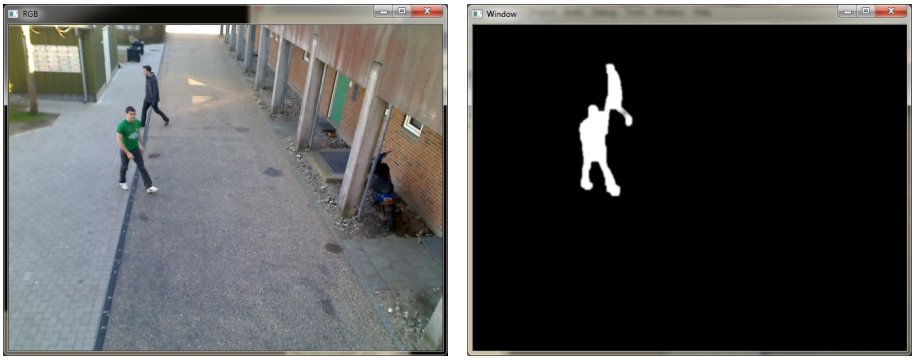
The best results were found on a video in which was taken in a totally shaded scene ,although the environment was sunny (Figure 3.13). It can be seen in the foreground mask Figure 3.13b ,that the algorithm is only retrieves the un-altered human blobs with no noise whatsoever.

Performance Figure 3.14 shows the performance of the implemented detection block. As it can be seen the algorithm runs faster than the *OpenCV*'s version, at a frame rate around 21 *FPS*. This shows that applying the ***HOG*** only on regions of interest will drastically improve the algorithm's performance compared to the one in which the whole image was scanned. The default HOG and the new algorithm were run using the same Support Vector Machine data.



(a) Original RGB image **(b)** Foreground Mask

Figure 3.12: Background subtraction testing. Totally cloudy environment



(a) Original RGB image **(b)** Foreground Mask

Figure 3.13: Background subtraction. Totally shaded environment

3.5 Conclusion

The chapter presents a custom approach to detect people in a 640x480 video sequence, taken with a commercial webcam. By using a background subtraction technique, foreground is separated from the background retrieving the relevant objects in the scanned scene. Preprocessing steps are then used in order to get good segmented blobs that will further represent the regions of interest (ROIs) for the actual detection step. The Histogram of Oriented Gradients (HOG, [2]) proved to be a useful tool when detecting humans. Moreover applying the HOG on the detected ROIs proved to be a faster approach than the original version proposed by Dalal and Triggs [2], implemented in OpenCV. The HOG features extracted from the ROIs are classified using a Support Vector Machine (SVM) trained with images from the INRIA pedestrian data set.

Training with a different dataset from the one given in *OpenCV* also proved to increase the system's robustness. Although the testing was done on a video sequence where only two people interact in the scene, compared to the *OpenCV* version, the

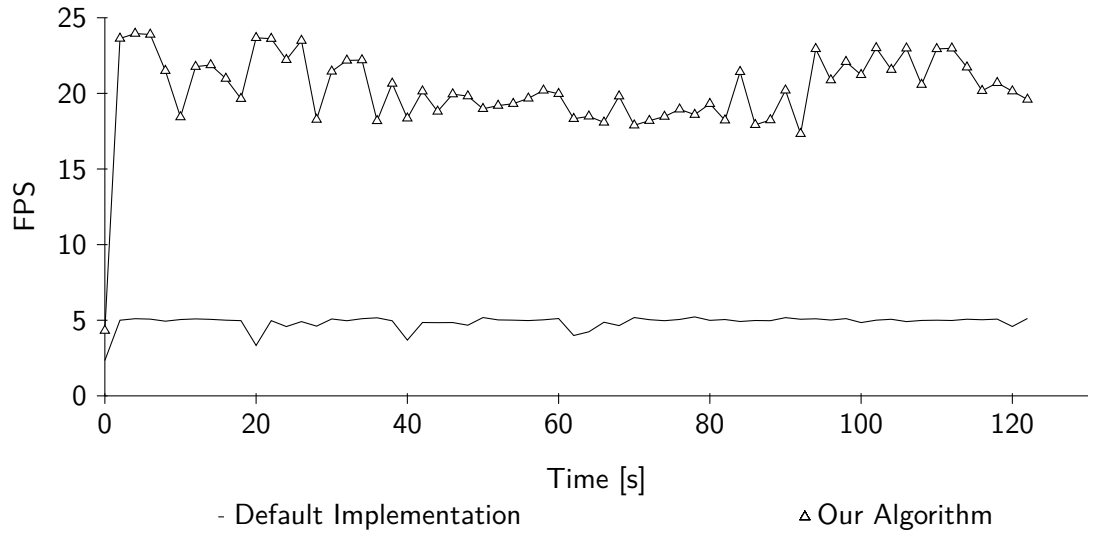


Figure 3.14: Comparison of the system to the OpenCV HOG default implementation.

persons were almost always detected (see 5.1).

Chapter 4

Tracking

4.1 Introduction

As the detection part was taken care of the tracking process can be enabled. As seen in the previous chapter *Detection* the detection block outputs trivial information used further for the tracking process.

In Detection, only the background model is time dependent, meaning that it is updated through frames; all the other processes can be process on a static image. Contrariwise, the Tracking process is completely time dependant, meaning that it has to know about previous states.

Definition A **Track** is the object that is assigned to a blob once and which is updated at each frame to follow it through time.

4.2 Design

Figure 4.1 is a reminder of the general organization of the system and the role of Tracking.

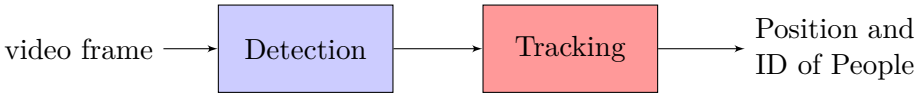


Figure 4.1: System representation in blocks

The input of this part of the system can be of three types:

- Detection of a new person.
- Detection of a person that was already there before.
- Detection of something that is not a person.

Indeed, even if the Detection includes a classification to ensure that it takes only human blobs, errors can still occurs.

The states are determined in the **Matching** part, and the system is updated accordingly. When a Track is lost, the **Prediction** is used to guess its next position.

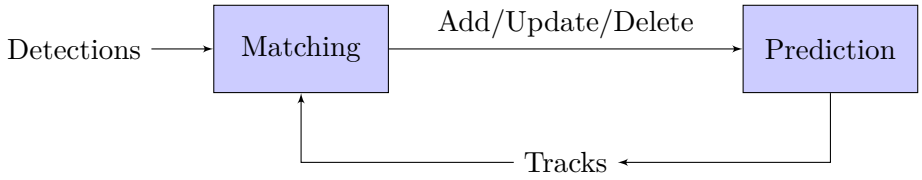


Figure 4.2: Tracking process

Figure 4.2 shows that the Tracks are kept through time and that decisions are made at every frame with a new set of detections. Prediction helps making decision about lost tracks and enhances the chances of matching the correct detection to a track.

4.2.1 Setup

There is no specific setup for this part, since this subsystem is entirely dependant of the output of the Detection subsystem. See 3.2.1 for the detection subsystem setup.

4.2.2 Matching

The Matching updates the set of Tracks by adding, deleting and updating Tracks. It is a complex decision making that is described later on.

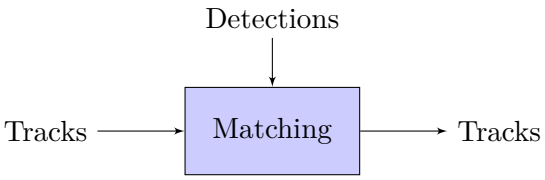


Figure 4.3: Tracking process

At each new frame, all existing Tracks and all the Detections and compared in order to find the optimal track/detection pair.

When associating a set of Detections to a set of Tracks, the following cases can occur:

- A single suitable Track is found for the Detection (common case). The Track is then updated with the data of the Detection.
- No Track is found for the Detection. The Detection is considered as new, and a new Track is associated.
- A Track is not matched by any Detection. In this case the Track is considered (temporarily) lost.

The decision making for deciding whether a Track and a Detection are matchable takes into account the different properties of each entity.

Features

The Matching process relies on different features, described hereafter. The choice of features and their correct usage is a major concern of this problem.

Distance The distance is computed from the estimated position of the Track (see subsection 4.2.3) and the gravity centre of the Detection. Doing this enhances the chances to get the correct association because direction and speed of the person are included, so the position can be accurately predicted, as soon as the direction does not change too much.

Histograms Another important feature that helps in matching a track from one frame to another is the colour histogram comparison. The algorithm uses the *Bhattacharyya* distance to determine whether one match state corresponds to its previous one. The smaller the distance the more certain it will be that the current track matches the previous state increasing the tracking robustness. This aspect will also help when, for example, two blobs will connect and separate afterwards. It will then be easier to associate which track belongs to which detection, by having the value of the *Bhattacharyya* distance.

Area Comparing the Area of the Track and the Detection allows to avoid major differences between the two. A threshold of Area variation is set to find merging blobs (or groups), and therefore avoiding to affect an ID of a person to a group.

Association

The next step is to associate a Track to a Detection according to the features presented above. This part is challenging because of the multiplicity of the features to optimize in order to find the best track / detection pair. Indeed, taking into account a single feature is not possible. For instance, if only the distance is taken into account, the matching will fail in a lot of crossing cases, the same applies to the histogram of colours (which was stated to be difficult to use, because of similarity of clothing among people).

The solution is to find a way to weight one and the other in order to make it as precise as possible. This weighting process is described in the Implementation part (see 4.3.1).

4.2.3 Prediction

The prediction part consists in the estimation of the next position of a Track according to its previous position and velocity. For that, the Kalman Filter [17] is used, as described in section 2.3.2. As it can be seen in diagram 4.4 the Kalman structure works

like a feedback control. The filter estimates the state and it gets a feedback(represented by noisy measurements) from the corrector.

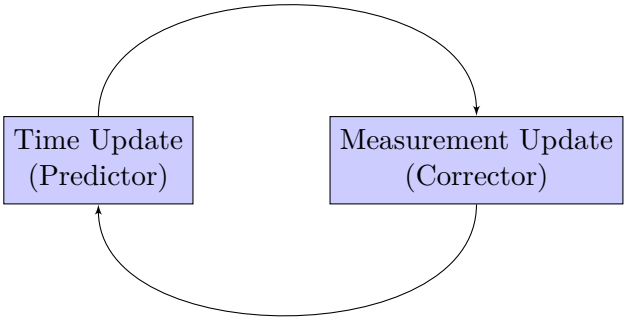


Figure 4.4: Kalman structure

In the proposed system the state is given by the matrix 4.1, where (x, y) represent the coordinates of the centre of mass retrieved from the detection block, and $(\dot{x}, \dot{y})$ represent its corresponding velocity.

$$S = \begin{pmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{pmatrix} \tag{4.1}$$

According to equation 2.27 in section 2.3.2 the first step of the "*predictor*" consists of determining a state estimate. As the state has only four variables the state-transition model matrix F will be a 4x4 matrix 4.2 [18]:

$$F = \begin{pmatrix} 1 & 0 & \Delta T & 0 \\ 0 & 1 & 0 & \Delta T \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.2}$$

The second step in the "*predictor*" consists of determining the error covariance matrix P (equation 2.31). For this system P is a 4x4 matrix which provides trivial information regarding how accurate the state estimates are. Basically it shows how the variables inside the state vector vary with respect to each other, therefore the matrix will hold the *differences* between the estimate state values and the actual ones. Usually the initial values of the covariance matrix are set by the user according to the certainty of the prediction. Bigger values on the P matrix diagonal represent more uncertainty regarding how S looks like at start-up while values closer to *0* represent the contrary.

A a major influence in the prediction step stands in the covariance matrix of the process noise Q from equation 2.31.Covariance Q is a measure that shows how the input state varies away from the initial transition. The matrix has the same dimension as P (see matrix 4.3) and its q value is also set by the user. Larger values represents

larger variance for the input state and the filter needs to be more adaptable to changes. Contrariwise if smaller values are used the output will be smoother but in conditions of massive changes the filter will not adapt.

$$Q = \begin{pmatrix} q & 0 & 0 & 0 \\ 0 & q & 0 & 0 \\ 0 & 0 & q & 0 \\ 0 & 0 & 0 & q \end{pmatrix} \tag{4.3}$$

The *corrector* block uses a measurement vector equation 2.26 to correct the state estimate from equation 2.27. Because the x and y are mapped to the measurement vector z and the derivative variables (velocities) from state S are not directly measured, the measurement matrix H will have the form of 4.4.

$$H = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \tag{4.4}$$

The trivial aspect of the *corrector* block is the Kalman gain K (equation 2.29) which measures "how much" the system updates state S. If K is small than the measurements are not trusted, otherwise the measurements will have more weight when running the process. Using this gain a new covariance matrix is computed, equation 2.31. However the gain is influenced ,like in the prediction case, by the covariance matrix of the measurement noise R and has the form of 4.5 If the r value is much grater than the value of q in the process noise covariance matrix, then K is small, otherwise K is big.

$$R = \begin{pmatrix} r & 0 \\ 0 & r \end{pmatrix} \tag{4.5}$$

4.3 Implementation

Similarly to chapter 3, the system implementation details are given here for C# implementation.

The **Track** class is described by the UML diagram in Figure 4.5 which presents the tracking system. It describes a rectangle (defined by location, height and width) where the tracked object was last detected. To ensure matching will be possible at next frame, it also saves some more information about this detection, like the location of the center of mass of the blob, the color histograms, and the area of the blob. A unique ID is assigned to each object and is kept through time thanks to the matching process. The Track structure also contains an *Age* attribute which represents how many times a track was found, or in other words how many time a detection has been matched to it. The *Lost Score* attribute represents the number of consecutive frames in which a Track was *not* matched to a detection. During the matching process, if this counter exceed a given threshold, the Track is considered completely lost, and is deleted to prevent memory loss and increase the computation speed. Finally, the velocity of the object is kept in the Track to be used in the Kalman Filtering. (see paragraph 4.2.3)

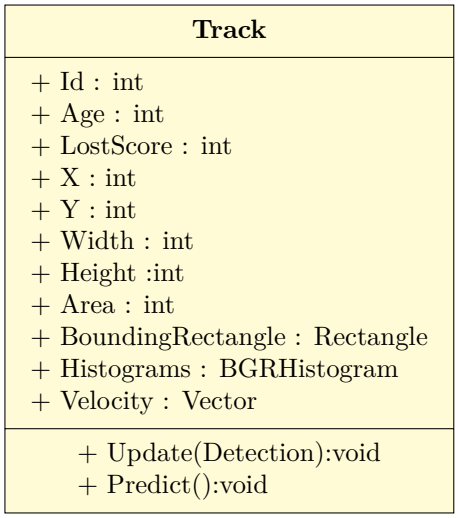


Figure 4.5: UML Representation of the Tracking system

Another attribute of the *Track* class is the *Histograms* which is of type *BGRHistogram*. The attribute has the role of storing the colour information for each detected blob, which will be further used in the matching part.

A better overview of the tracking system implementation can be seen in the appendix section B.9.

4.3.1 Matching

Features

The features introduced earlier are computed according to the following details.

Distance The distance between the estimated position of the Track - given by the Kalman filter - and the Gravity Center - given by the moments of the contour of the blob - is easy to compute:

$$distance(d,t) = \sqrt{(x_d - x_t)^2 + (y_d - y_t)^2} \tag{4.6}$$

Histogram The comparison of two histograms is implemented in OpenCV with the function *cvCompareHist*, and not re-implemented in EmguCV (thus accessed via the CvInvoke class). This method takes two *DenseHistogram*, which represents a multi-dimension histogram, and a comparison method (of type *HISTOGRAM_COMP_METHOD*). The system uses the Bhattacharyya distance, described in section 2.1.5, for which the comparison method is *CV_COMP_BHATTACHARYYA*. As the system uses RGB colored images, three dimensions are needed for the histogram. However, it was more convenient to implement a new class (*BGRHistogram*, see appendix B.6) containing

three one-dimension histogram. To compare two histograms, the following product is applied:

$$hist_distance(h_d, h_t) = \prod_{i=1}^{3}(1 + bhattacharyya(h_{d,i}, h_{t,i})) \tag{4.7}$$

This will give a result such as a perfect match will be 1, and the maximum difference will output 8 (since the Bhattacharyya outputs a result between 0 and 1).

Area The area ratio between a Track and a Detection is computed from their bounding box in the following way:

$$area\_ratio(d, t) = \frac{\max(\mathcal{A}_d, \mathcal{A}_t)}{\min(\mathcal{A}_d, \mathcal{A}_t)} \tag{4.8}$$

where t and d represent the bounding box of the track, respectively the detection. With this equation, a ratio is obtained, that will always be one or above, and which permits to find a threshold for maximum variation of size from a frame to the next.

Association

The association matter, which determines which finds the most appropriate track/detection pair, can be implemented following algorithm 4.1. The algorithm is of complexity $\mathcal{O}(n^2)$. The *Update* function, which belongs to the $Track$ class (see figure 4.5), allows the $Track$ to modify some of its value according to the *Detection* given as parameter (see algorithm 4.2).

Given the features described in the previous section, a score following next form could be computed:

$$score(d, t) = hist\_distance(d, t) \times (\beta + distance(d, t)) \tag{4.9}$$

This score, inspired from the work of Breitenstein *et al* [3] would give the 'closest' (in terms of features, not only distance) Track from each Detection and allow to match them.

However, this solution gave poor results for many cases and was dropped. The realized approach is less computational and more case dependant. It is based on a set of threshold values, found empirically, which isolate the possible matches by distance, then choose the closest acceptable colour histogram.

4.3.2 Prediction

The prediction part is given by the Kalman filter, which was implemented according to the UML diagram presented in Figure 4.6.

The implementation follows the exact steps presented in the design part (see 4.2). The *KalmanFilter* class takes as input the *State* attribute which is of type *Matrix*. As presented earlier the *State* matrix will consist of the centre coordinates (X and Y) of the detected person and its velocity (dx and dy). Also the class will take as input all the matrices presented in the Design section 4.2.

Algorithm 4.1 The matching algorithm

T : Set of all tracks
D : Set of all detections
$S(t, d) = 1000$: Array of all track/detection values ▷ Calculate all scores
for $t \in T$ **do**
  for $d \in D$ **do**
    $s \leftarrow Score(d, t)$
    if $s \geq \tau, s < S(d, t)$ **then**
      $S(d, t) \leftarrow s$
    end if
  end for
end for
for $t \in T$ **do** ▷ Update matched scores
  if $t \in S$ **then**
    $t.Update(d)$
    $D \leftarrow D - d$ ▷ Mark unmatched tracks as lost
  else
    $t.LostScore \leftarrow t.LostScore + 1$
  end if
end for ▷ Mark unmatched detections as new Tracks
for $d \in D$ **do**
  $T \leftarrow T + Track(d)$
end for

Algorithm 4.2 The Update function

function UPDATE($track, detection$)
  $\{track.X, track.Y\} \leftarrow \{detection.X, detection.Y\}$
  $track.Histogram \leftarrow detection.Histogram$
  $track.LostScore \leftarrow 0$
  $track.Age \leftarrow track.Age + 1$
  return $track$
end function

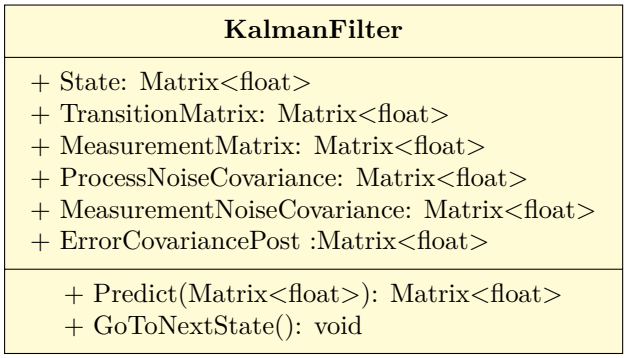


Figure 4.6: UML Representation of the Prediction step

4.4 Testing

Depending on the application, the Kalman filter can be adjusted by changing the q value from the covariance matrix Q 4.3 and also the r from the measurement covariance matrix R 4.5.

Several tests were made in order to see the impact of the above mentioned parameters. Figure 4.7 shows the results used on a single track where the chosen values were $q = 0.001$ and $r = 0.05$. As it can be seen if the q value is much more lower than the r one, more importance is given for the prediction and the corrected estimated values (blue graph) will "follow" the predicted ones (red graph). In this case the system is fast but not that robust as the filter is not that stable to sudden changes. The next figure 4.8, shows that increasing the value of r even more to 0.5, measurements will be weighted less and less.

On the other hand as the q value is increased, it can be seen in figures 4.9 and 4.10 that the measurements are given more importance and the estimated values (blue line) will "follow" the measurement ones (green line). In this case the system adapts better to sudden changes but the speed of the filter drastically decreases and the results will not be as smooth as in the previously presented case.

In Figure 4.11 it can be seen how the Kalman filter works when a detection is found. The parameter used are q=0.001 and r=0.05 therefore it can be seen that the results are more smooth when more importance is given to the predicted values. The filter works fast and it follows the human closely.

Figure 4.12 shows the results of the Kalman filter on the same video presented above, but the chosen parameters were: q=0.1 and r=0.05. In this case more importance is given to the measurement values. However, as it can be seen, the filter will run rather slow and it will not follow the detection as close as the previous case loosing the track more rapidly.

Matching As it was presented earlier in the *Design* section 4.2 the matching is dependent on three parameters: area ratio, distance of blobs from one frame to another and the histogram of the tracked blobs. Since getting an overview on the ratio and the

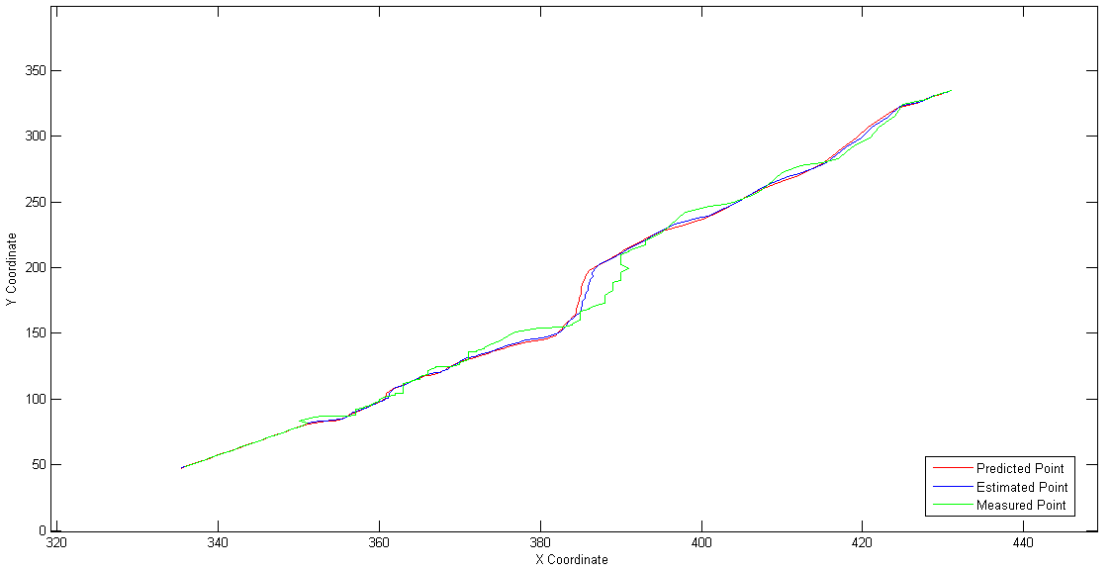


Figure 4.7: Impact of q and r on the Kalman filter. q=0.001, r=0.05

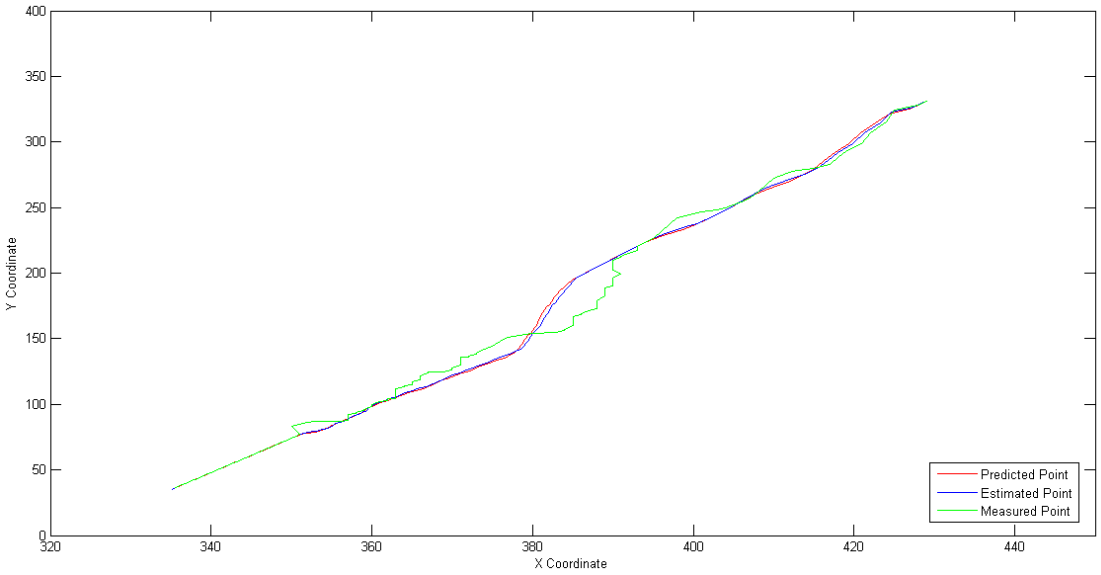


Figure 4.8: Impact of q and r on the Kalman filter. q=0.001, r=0.5

distance was fairly intuitive and visible, the colour comparison was the actual problem in this case. Therefore several tests were made on some videos to challenge the algorithm and determine some solid threshold values when comparison is done.

For the video seen in Figure 4.13 , the results weren't too satisfactory as both persons in the scene were wearing clothing of the same colour. Moreover, the lighting conditions alter the colours making them darker, and this often known as the "contre-jour" effect.

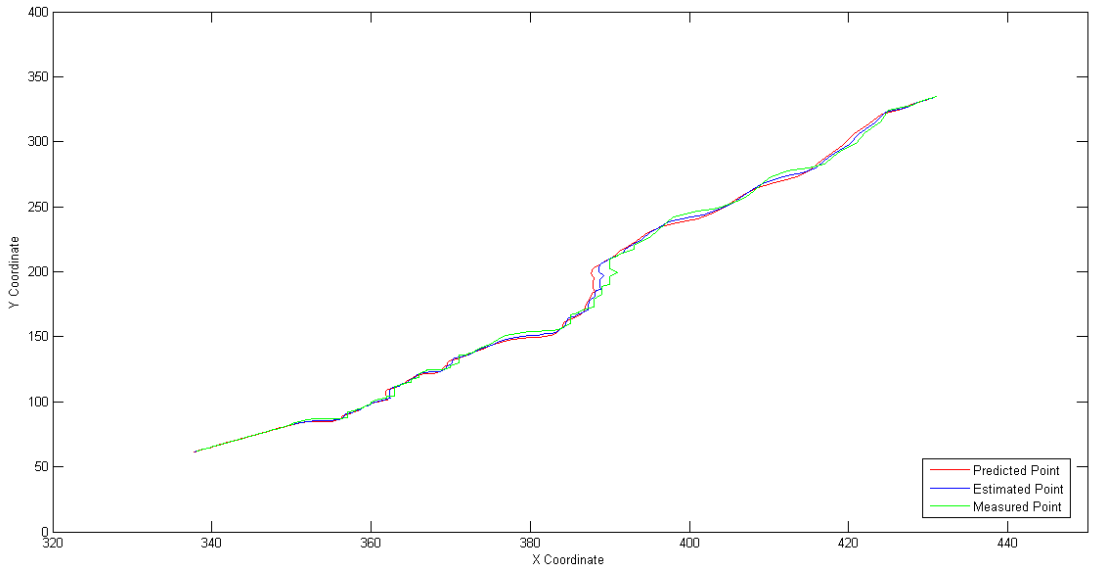


Figure 4.9: Impact of q and r on the Kalman filter. q=0.01, r=0.05

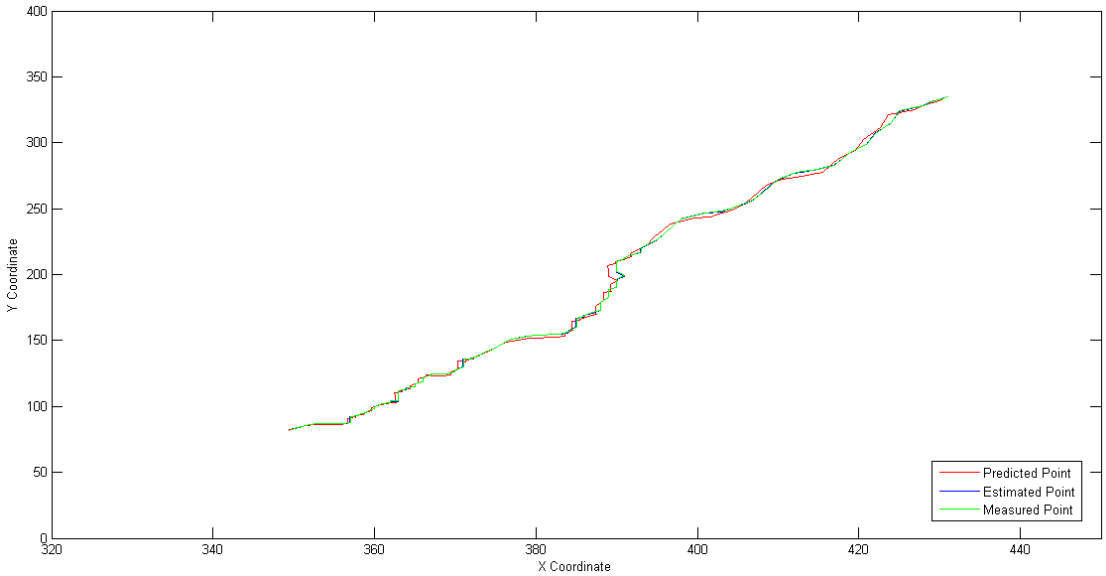
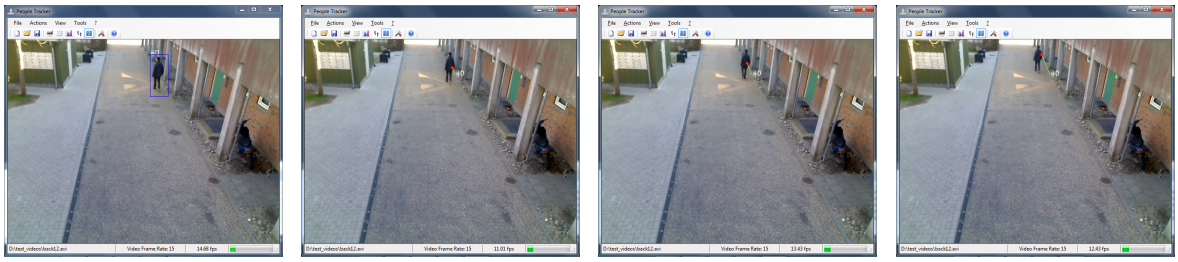


Figure 4.10: Impact of q and r on the Kalman filter. q=1, r=0.05

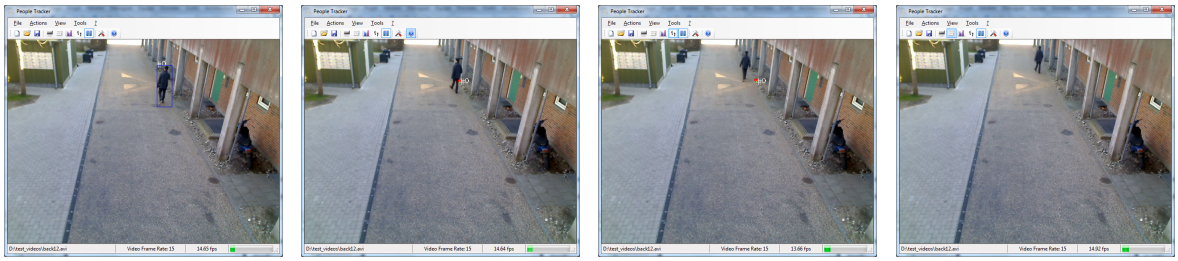
As the idea of matching is to reassign, after an occlusion, the lost ID of the track, the problem in this case is that all the blobs will have their colour data in the same range. This will make the histogram comparison algorithm irrelevant for the matching step.

When tested on a video where the persons were dressed completely different, the results were slightly better regarding the histogram comparison. However ,this is quite a limitation, as one cannot predict whether two persons will or will not dress the same.



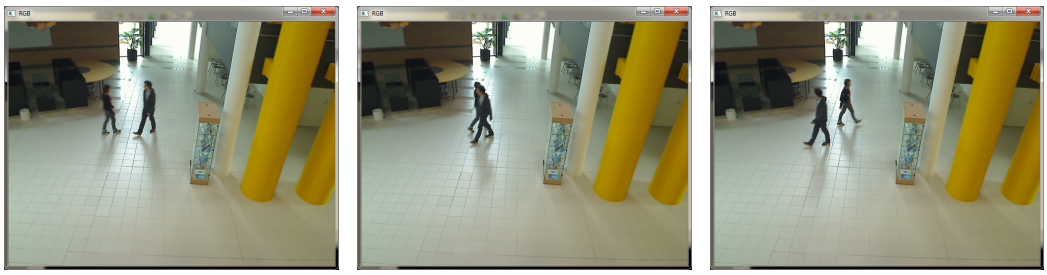
(a) (b) (c) (d)

Figure 4.11: Kalman filter q=0.001 and r=0.05.



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

Figure 4.12: Kalman filter q=0.1 and r=0.05.



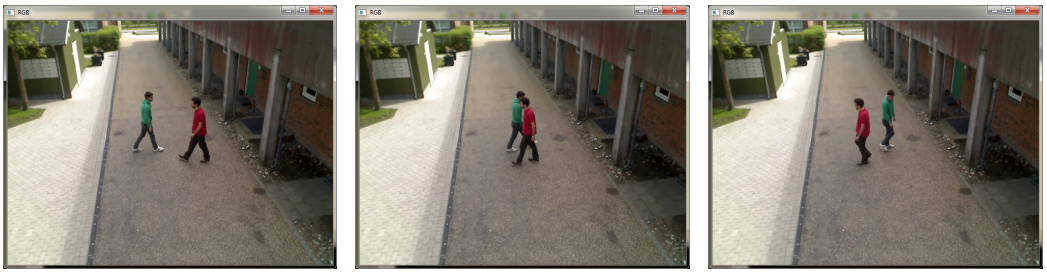
(a) (b) (c)

Figure 4.13: Bad video for histogram comparison.

Figure 4.14 shows an ideal case where a person dressed in red meets another one dressed in green. Here the histogram comparison' s results are better.

4.5 Conclusion

The chapter presents a classical approach for a tracking framework by using the Kalman filter [17] and colour histogram comparison. The main idea of this part of the system is to keep track of the different human detections that were found by the detection



(a) (b) (c)

Figure 4.14: Good video for histogram comparison.

block. It ensures that in simple cases and more complex ones, like when people cross each other, the tracking IDs will be reassigned to the right person. It also prevents a person to be confused with a group of people (people merging) using the area ratio. The three used features (area, distance, histograms) represent a fairly simple and fast, though robust, set of attributes to be used in the matching step. This aspect performance will be demonstrated in Chapter 5.

As the Kalman filter works with predicted and measured values in order to get the estimation, improving the accuracy of the tracker is in fact a trade-off choosing between having a slow but adaptable system or a fast but not that adaptable one. The second case was chosen as more weight was given to the predictions in the Kalman filter, resulting with a fast prediction but with a few errors in some cases. Therefore the results were more smooth and suited the application better (see section 4.4).

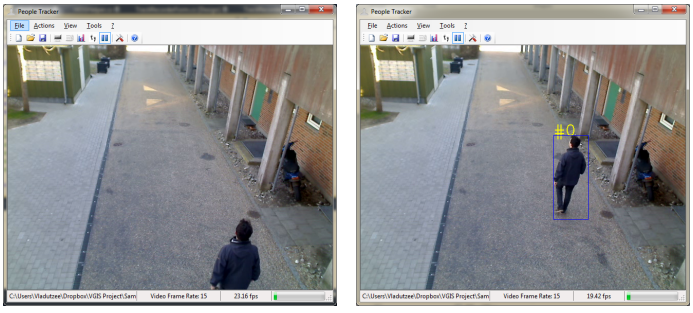
Chapter 5

Closure

Given the presented system, this chapter is meant to show final results of the entire framework. A discussion will be made based on these results and conclusions will be drawn in contrast to the imposed requirements stated at the beginning of the report (1.6). Furthermore the chapter will conclude with some future work aspects that will show how the system could be improved .

5.1 Results

The proposed system works on 640x480 resolution videos taken with an ordinary webcam. Its final results are slightly different from the expected ones but still demonstrate a good detection-tracking framework. As it can be seen in Figure 5.1 the detection and tracking is done in an area of interest where the whole human blob appears. That means that the system will not perform at the upper and lower border of the frame. However when the full person enters the scene, they will be bounded by its own rectangle and a track (in this case *0*) will be assigned.

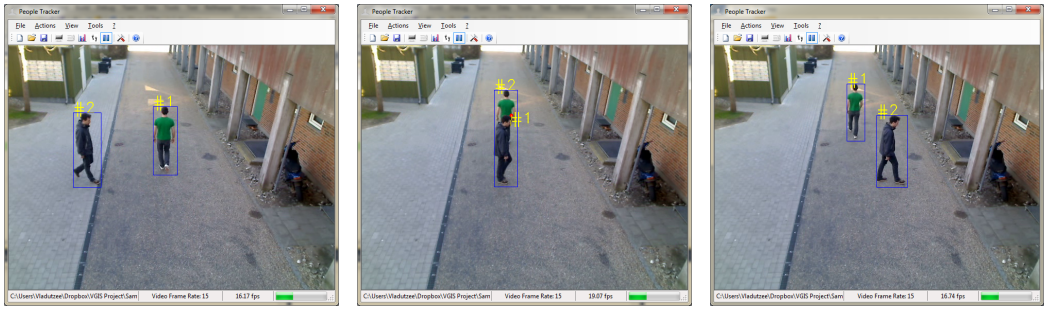


(a) **(b)**

Figure 5.1: Detection/Tracking done only when the entire human blob appears in the scene.

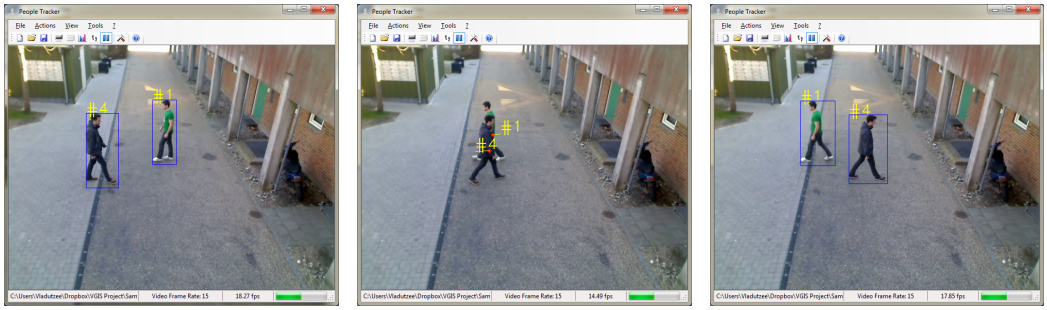
Given the chosen parameters presented in the testing sections of Chapter 3 and Chapter 4, the system presents good results in the video seen in figures 5.2, 5.3 and

5.4. The figures shows how two persons are detected and each have an assigned track. This is indeed a good result as the tracking block will only be active if, after background subtraction and pre-processing, a human is detected over the region of interest described by the blob. As it can be seen, the matching part works smooth as the track of each blob is re-assigned after the two persons intersect in different ways.



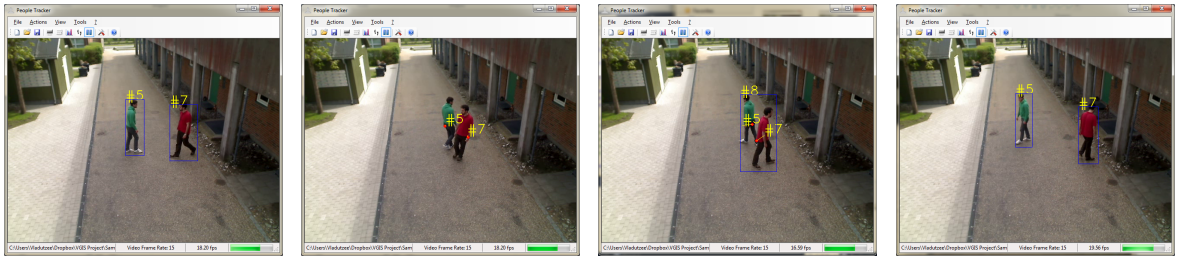
(a) **(b)** **(c)**

Figure 5.2: Simple intersection of two persons.



(a) **(b)** **(c)**

Figure 5.3: Simple intersection of two persons.



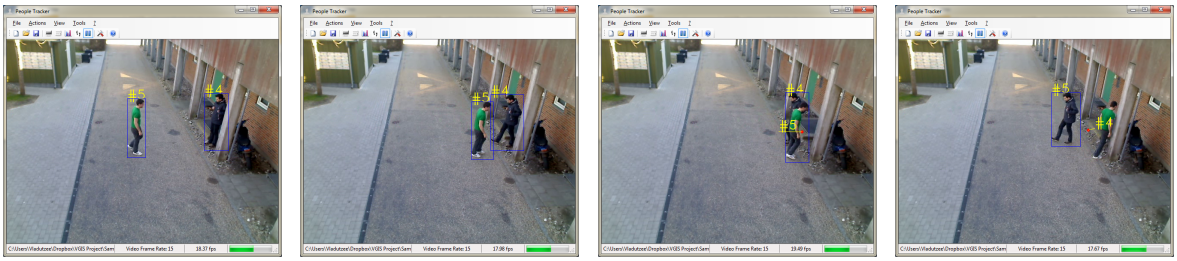
(a) **(b)** **(c)** **(d)**

Figure 5.4: Complex intersection of two persons.

However the same video sequence contains also few flaws regarding the performance of the system. The set of videos that was used was not "natural", meaning that they are not recordings of people walking freely, but scenes in which the movement and events were controlled. These videos were meant to challenge and push the system's performance in different situations (different types of intersections between persons in order to test the matching robustness, different lighting conditions to see how the system responds or adapts).

It should be mentioned that most videos in the set contain only interaction between two persons as it was considered that resolving this issue is the first step to get a clear overview about particular used methods, such as the Kalman filter or the Histograms of Oriented Gradients, where parameters needed to be tuned in order to gain an optimal result.

Figure 5.5 shows a case where the matching fails. As it can be seen the track of the person dressed in green (track *5*) is lost and assigned to the other person (track *4*). An explanation for this situation is that the Kalman filter does not have time to predict the future values of track 4. By the time prediction is done, the two persons will have intersected, thus the prediction fails. If the prediction fails then the matching part will be more difficult to be done and it is highly probable that it will fail. This kind of error is a limitation of the system but is also a very tricky situation.



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

Figure 5.5: Algorithm mismatch when two persons intersect.

The performance of the system is given by formula 5.1:

$$error\ rate = \frac{FN + FP}{N} \tag{5.1}$$

Where:
- FN are the *false negative* values
- FP are the *false positive* values
- N represents the total number of frames

The error was computed as it follows. Firstly a ground truth data[1] is created which illustrate the real position of the detected human. After that, the algorithm is run and

[1]Ground Truth Data is the *real* data measured by human. It is used to estimate the system's performance.

the *false positive* and *false negative* are obtained. Usually a value is considered to be good if it is within an acceptable distance from the one in the ground truth. If one value exists in the ground truth case but not in the real one, then it will be considered as false negative. On the other hand if one value is found in the real case but it does not exist in the ground truth one then it will be established as a false positive value.

The error rate was computed for several videos as it can be seen in Table 5.1.

Input	Number of frames	FP	FN	Error rate
Video1	0	11	158	6.9 %
Video2	8	33	157	26 %
Video3	7	16	147	15.6 %
Video4	1	31	192	16.6 %
Video5	1	40	200	25.5 %

Table 5.1: Error Rate

The data presented in table 5.1 is given by videos where the system is challenged a lot. *Video1* shows a crossing of two people walking on different directions. *Video2* shows two people intersecting in diagonal. *Video3* shows two people crossing in one direction (one walking to the right, the other to the left), but some noise is noticed due to the camera autofocus. In *Video4* two people meet (one direction crossing) and then turn back on their steps. Finally, *Video5* is an example of border problem.

The average error rate for these five challenging videos is **18.12%**.

The tracking system is demonstrated with figures 5.6 and 5.7. These data represent the comparison between the ground truth data and the system output. The used scenes were chosen for their interest, meaning that they show either good results (e.g. crossing where the matching is correctly done), or on the opposite scenes with mistakes (e.g. people not detected by the system).

5.2 Conclusion

An overall functionality of the proposed system was taken with respect to the imposed requirements from section 1.6. The system is able to run on long term videos. Even though all the testing samples have an average of 3 minutes each, the performance of the system on long video sequences is influenced by additional problems. As it could be seen in section 3.4 of the Detection chapter the adaptability of the system is highly dependent on the environment that the scanning is performed on. Testing showed that optimal results were obtained only for totally cloudy or totally shaded environments. It was seen in other video tests (see 3.4), that sudden changes of the light, mess up the foreground detection step, and not even the preprocessing framework can give a

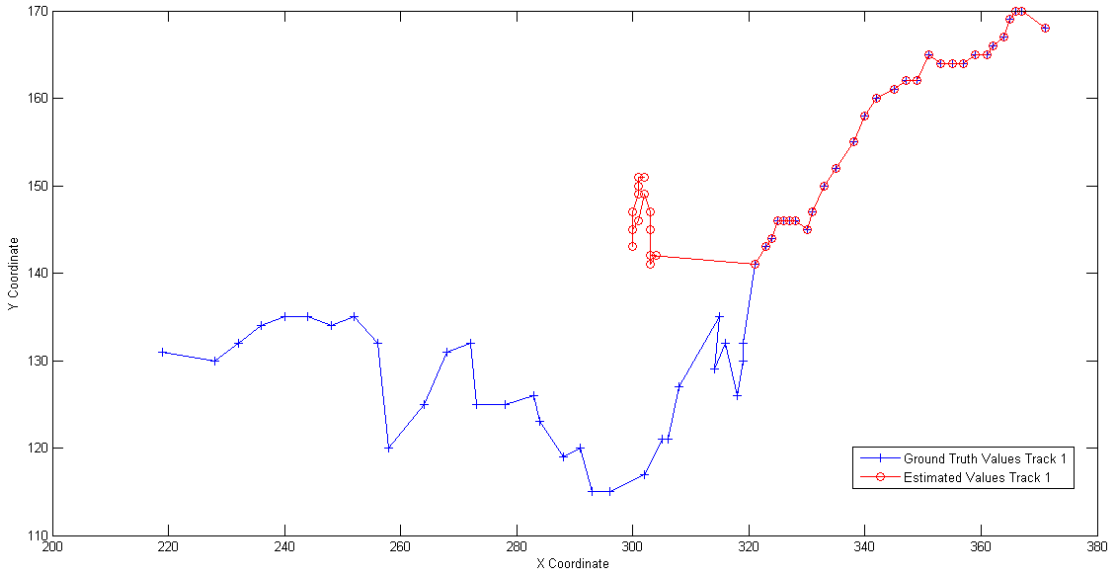


Figure 5.6: Comparison of ground truth and system result in a successful matching.

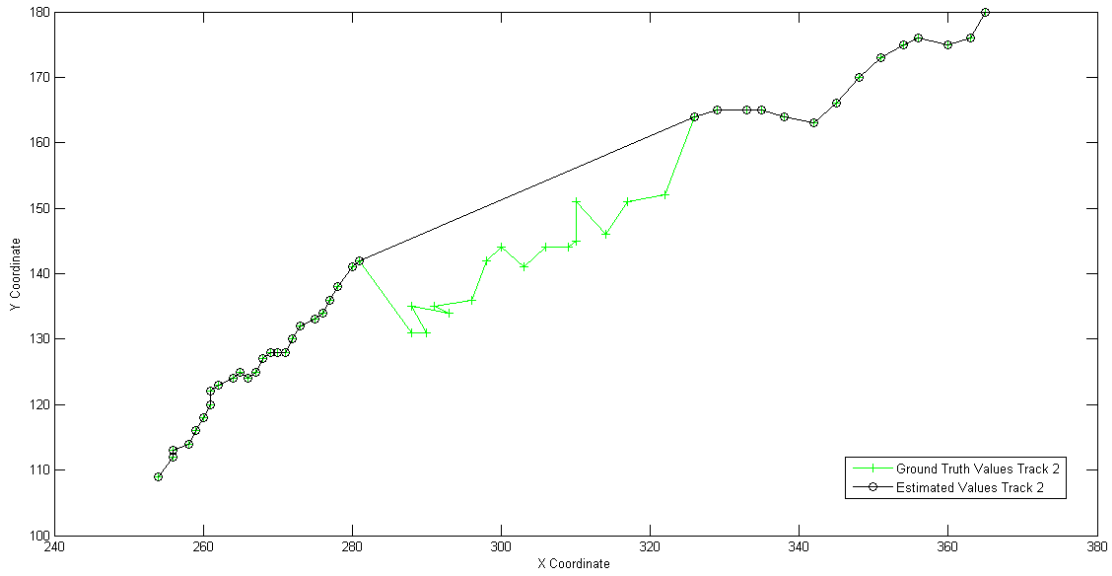


Figure 5.7: Comparison of ground truth and system result in a failed matching.

better segmentation result. However, shaded or cloudy scenes present no problem for the system to adapt on a long period of time.

The detection part runs smooth when a newly trained support vector machine (SVM) was used in order to classify the descriptor vectors provided by the Histogram of oriented gradients (HOG). It should be stated that the system doesn't handle groups (see 1.7). That means that each time two persons will intersect their resulting blob will not be taken into consideration when doing the detection.

Another relevant aspect of the system presented in the requirements section, is the unique ID assignment. It was seen earlier in this chapter that this part is done by solving a matching problem. When two human blobs split after a crossing they are reassigned. However due to clothing colour limitations and also lighting conditions, the matching will fail. It was shown that for a totally shaded environment where persons wear clothing of distinct colour, the matching gives good results as IDs are reassigned. Another issue that has been stated earlier is that the system doesn't handle the grouping problem. This will also interfere with the matching part as it can be seen in Figure 5.5 where the reassignment of IDs fails.

5.3 Perspective

The perspectives for future work on this system are mainly related to fixing some of the system's limitations.

Firstly, the grouping problems could be solved by detecting merging blobs and applying a different technique. The easiest one to apply would probably be head detection, that would give a reasonable idea of how many people are in that group.

The tracker could be improved by finding a better way of combining the used features. Indeed as mentioned in this report (section 4.2.2), the scoring should give a result such as a small score means a high probability of matching (relative to distance and histogram similarity). Emerging ideas were to use normalized values (for distance and histogram) and weight them according to a coefficient (that should be found by testing on every possible values to generate a phase diagram and give the optimal values). Another idea would be to use an exponential form to be tolerant on small differences but dismiss drastically the big ones.

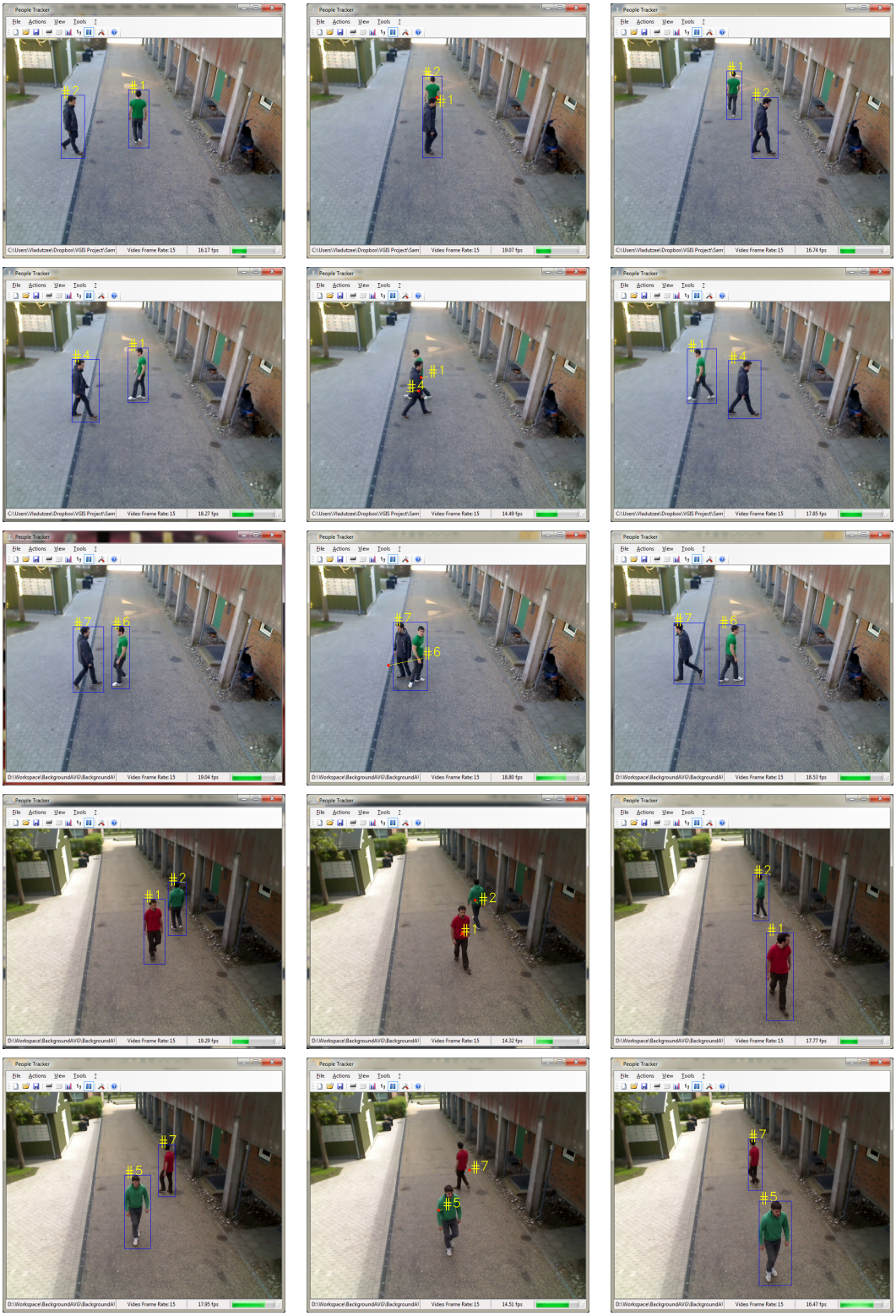


Figure 5.8: System results

Appendices

Appendix A

Application

In this appendix the different features of the final application are introduced.

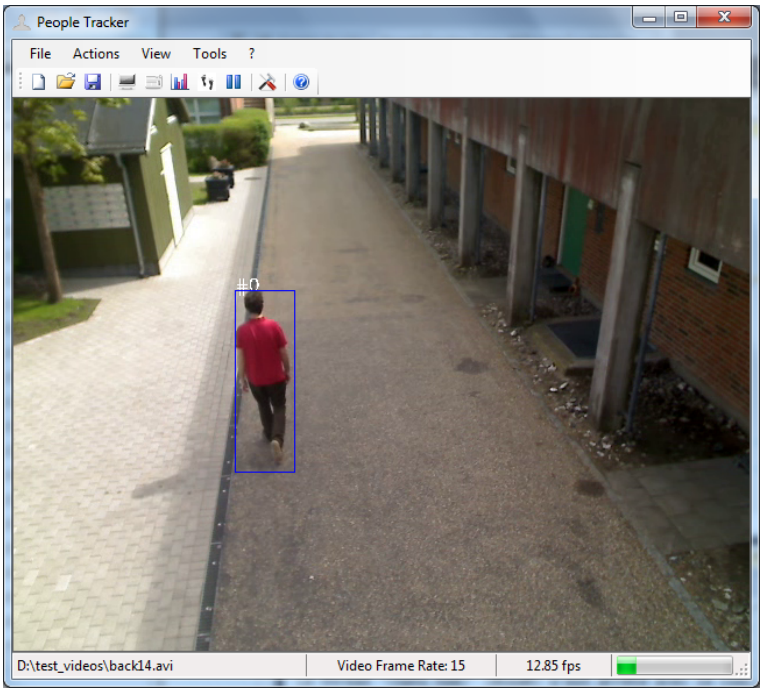


Figure A.1: The application interface

Figure A.1 shows the main window of the application. It displays the image with human tracks in blue.

The main window displays, in addition to the video with tracks, the running file (bottom left corner), the video frame rate (bottom middle left), the program frame rate (bottom middle right) and the progress of the video (bottom right corner).

A.1 Features

Load video stream. Loading a video can either be done by doing drag & drop from the OS explorer to the software, or using the browse dialog. The last 10 videos are also kept in registry to facilitate the access. It is possible to directly use a live video stream from a webcam. Video can be paused at any time use space bar or "Pause" button. Opening a new video clears the memory and reinitializes the system.

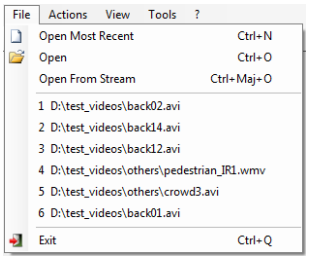


Figure A.2: Opening a video.

Views The interface offers some information panels, as can be seen in figure A.3, like a Foreground mask. Some other information on the selected track (displayed in red) can be obtained in the property panel (internal ID, Age, Area, Position...), and in the histogram panel (one histogram for each channel R, G, B).

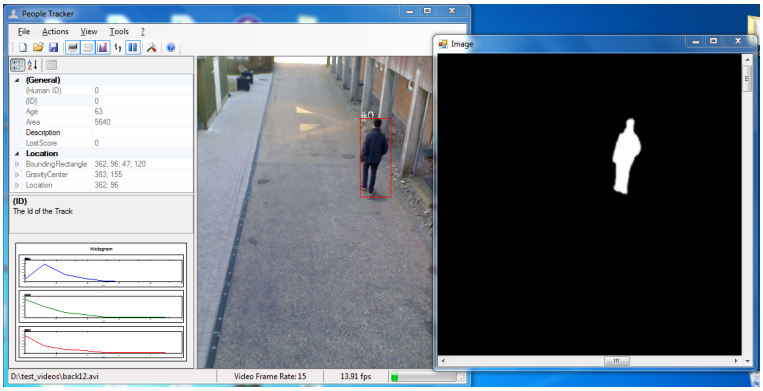
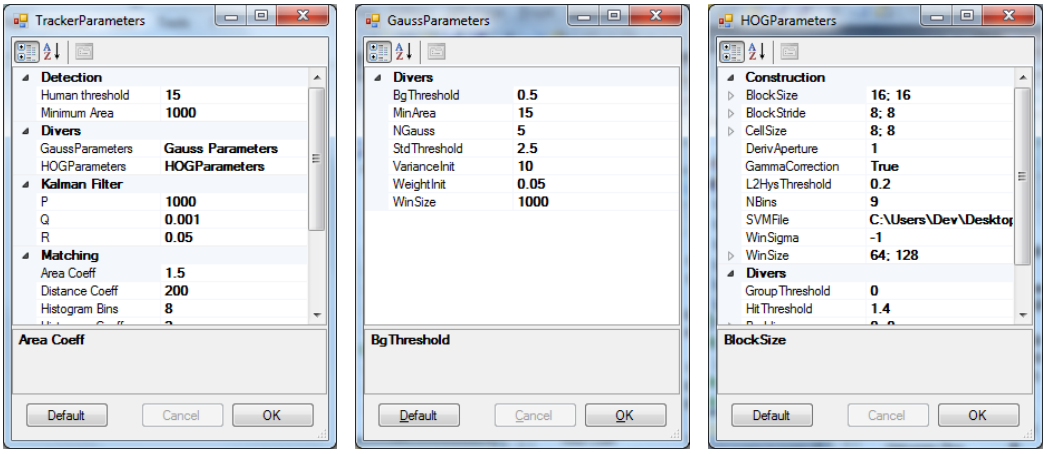


Figure A.3: The different available panel views.

Parameters The most interesting and useful feature of the application is the customizability. Indeed, every single parameter used by the algorithms are customizable. Figure A.4 shows the editors used to modify these parameters. The update is live, meaning that the parameters can be changed while the system is running (opening a parameter editor actually pauses the system to avoid conflicts during computation). However, for some parameters (HOG parameters for instance) modification is highly unstable and might

make the system crash. Indeed it has been decided to leave the values completely free so the user can try everything that crosses his mind.



(a) General Parameters **(b)** MoG Parameters **(c)** HOG Parameters

Figure A.4: Parameters Editors

To conclude with the customizability of the system, the preprocessing operations editor allows the user to completely reorganize, edit, add or delete some operations to fit his needs. Figure A.5 shows the editor.

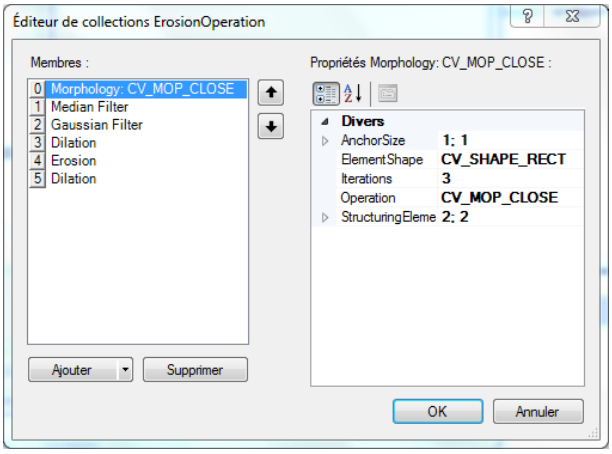


Figure A.5: Parameters Editors

A.2 Utilities

Finally the software also provides two utilities, that can be run using a command line option. (see previous point).

Histogram Comparison Utility Using the same functions as the main application, this software allows to compare two images by comparing their histograms of colour.

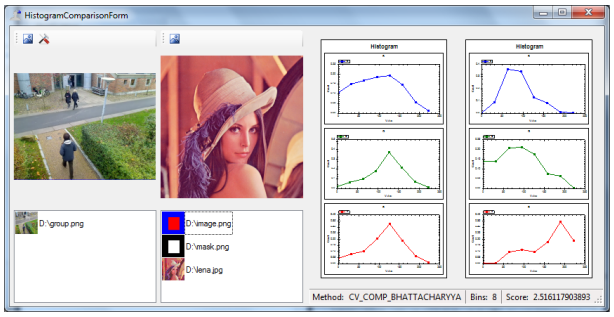


Figure A.6: Histogram Comparator Utility

Manual Tracking Utility With the same methods (detection, classification, matching) and the same parameters, this software allows to record the *real* position of the tracked humans. Indeed, it allows to pause and correct the recorded position of each track. We are then able to compare those *ideal* values to the one obtained by the system.

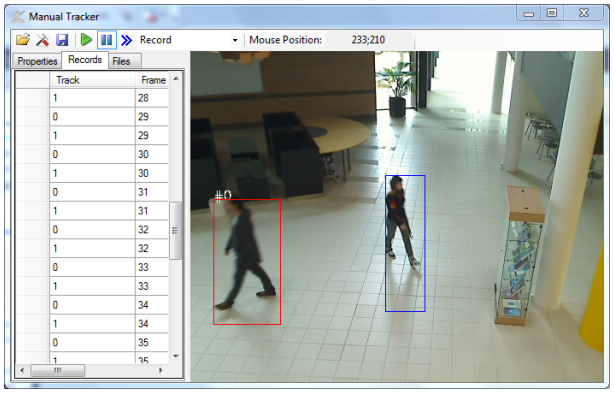


Figure A.7: Manual Tracker Utility

A.3 Command Line

The application provides a command line interface:

```
-h, --help                  Show this message and exit
-f, --filename=FILENAME     Loads the video FILENAME.
-c, --config=FILENAME       Loads the configuration file FILENAME.
    --histogram             Launch Histogram Comparison utility.
-m, --manual                Launch Manual Track Recorder utility
```

Appendix B

Implementation Details

B.1 General presentation

The organization of the project, as seen in figure B.1 clearly separates different parts of the program. The **Core** contains the main processing like image acquisition, preprocessing, blobs extraction...(see chapter 3), and deals with the tracking (see chapter 4). The **UI** is the Graphical User Interface which presents the video to the user and provides some features that can modify the Core behaviour. The **HOG** contains some tools for the Histogram of Oriented Gradients computation. The Utils part are functionalities which are useful for several parts of the system.

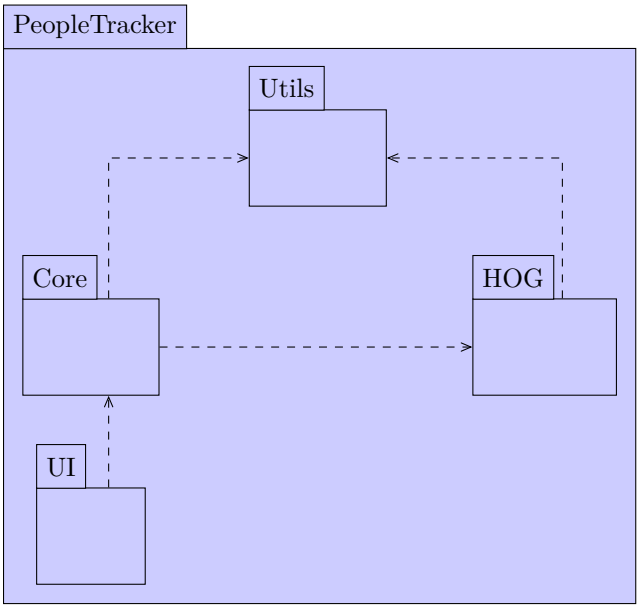


Figure B.1: Solution Description

B.2 Choices

In this paragraph we explain the choices that were made for the development process.

C# .NET The system was developed in C# using Microsoft® .NET framework 4. Indeed this high level approach suited both the group members because it enhances the organization of the project, facilitates the sharing

EmguCV Library We are using an OpenCV wrapper for C# called EmguCV. This library provides high level structures and classes for most OpenCV functions, as well as a low level access to the all functions (basically for those which were not implemented) through the *CvInvoke* class. According to the EmguCV documentation [23]:

> The *CvInvoke* class provides a way to directly invoke OpenCV function within .NET languages. Each method in this class corresponds to a function in OpenCV of the same name.

GIT We used the distributed Source Code Management (SCM) system Git. The main reason of this choice is the distributed aspect, which allows each developer to handle its own part without messing with the main development repository. The use of an SCM in this project appears essential for keeping previous versions of our code and to simplify the integration of the other's modifications.

B.3 Structures

This appendix introduces the main structures used in the system. Classes from EmguCV library are not detailed here. For more information on this library, see its documentation [23].

All structures presented in this appendix are part of the namespace **PeopleTracker. Core. Structure**.

B.4 Detection

B.4.1 Properties

.NET classes

- System.Drawing.Point
  According to documentation, it "represents an ordered pair of integer x- and y-coordinates that defines a point in a two-dimensional plane."

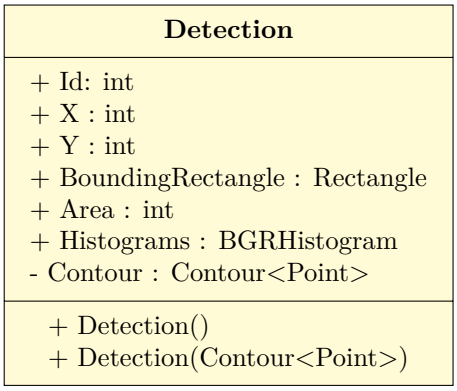


Figure B.2: The Detection class diagram

EmguCV classes

- Emgu.CV.Contour<T>
  This class is a set of points (*T* is *Point* or *PointF*, which is a *Point* with float values) used to describe a blob. It can be used to find the **moments** of the shape, though give the gravity center of the blob. It also provides the bounding box and the area.

Custom classes

- PeopleTracker.Core.Structure.BGRHistogram
  See paragraph B.6 for definition.

B.4.2 Operations

- Constructor
  The constructor either builds an empty *Detection*, either makes one from a *Contour*.

B.5 Track

B.5.1 Properties

.NET classes

- System.Drawing.Rectangle
  According to documentation, it "stores a set of four integers that represent the location and size of a rectangle."

EmguCV classes There is no undefined EmguCV class in *Track*.

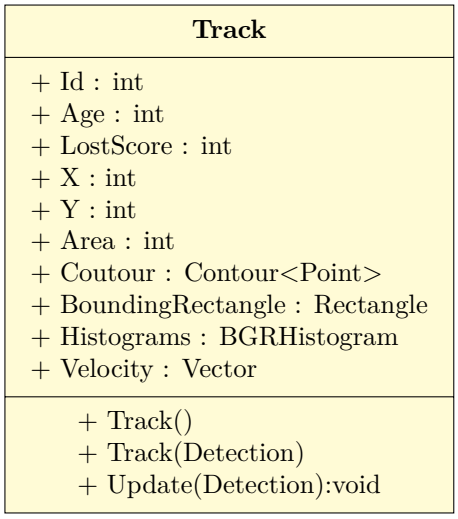


Figure B.3: The Track class diagram

Custom classes

- PeopleTracker.Core.Structure.Vector
  This class, which will not be further detailed, defines a mathematical 2 dimensions Vector with a set of methods (dot product, sum, ...) and properties (magnitude,...)

B.5.2 Operations

- Constructor
  The constructors either create an empty *Track*, or create one from an associated *Detection*.

- Update
  This function allows to modify the *Track* with the *Detection* it has been matched with.

B.6 BGRHistogram

This class is just a helper for the EmguCV **DenseHistogram**, to adapt the fact that we are keeping a histogram for each of the 3 channels of the image (Blue, Green and Red). The values are normalized to 1 so the number of pixels does not matter.

B.6.1 Properties

.NET classes We do not use .NET classes BGRHistogram.

Histogram
+ Histograms : List<DenseHistogram>
+ BGRHistogram(Image<Bgr, byte>, Image<Gray, byte>, int) + CompareTo(BGRHistogram, HISTOGRAM_COMP_METHOD): double

Figure B.4: The BGRHistogram class diagram

EmguCV classes

- Emgu.CV.DenseHistogram
  It is a "Uniform Multi-dimensional Dense Histogram", according to the documentation. Although, we are only using one dimension for our purposes, since the multi dimension is not flexible enough.
- Emgu.CV.CvEnum.HISTOGRAM_COMP_METHOD
  This enumeration is used to specify the Histogram Comparison method to apply among these:
  - *CV_COMP_CORREL*: Correlation
  - *CV_COMP_CHISQR*: Chi-Square
  - *CV_COMP_INTERSECT*: Intersection
  - *CV_COMP_BHATTACHARYYA*: Bhattacharyya distance.

Custom classes There are no custom structures in this class.

B.6.2 Operations

- Constructor
  The constructor takes an image, an optional gray scale mask, and a number of bins. The mask allows to build the histogram on a blob, getting rid of the background.
- CompareTo
  The *CompareTo* method uses the OpenCV *cvCompareHist* method on the three B, G, and R channels and returns the sum of the three comparisons. See section 2.1.5 for details.

B.7 PeopleTrackerParameters

This class regroups all the parameters used in the algorithm. This class gives the system great testing possibilities. See sections 3.4 and 4.4 to understand the choices of the parameters.

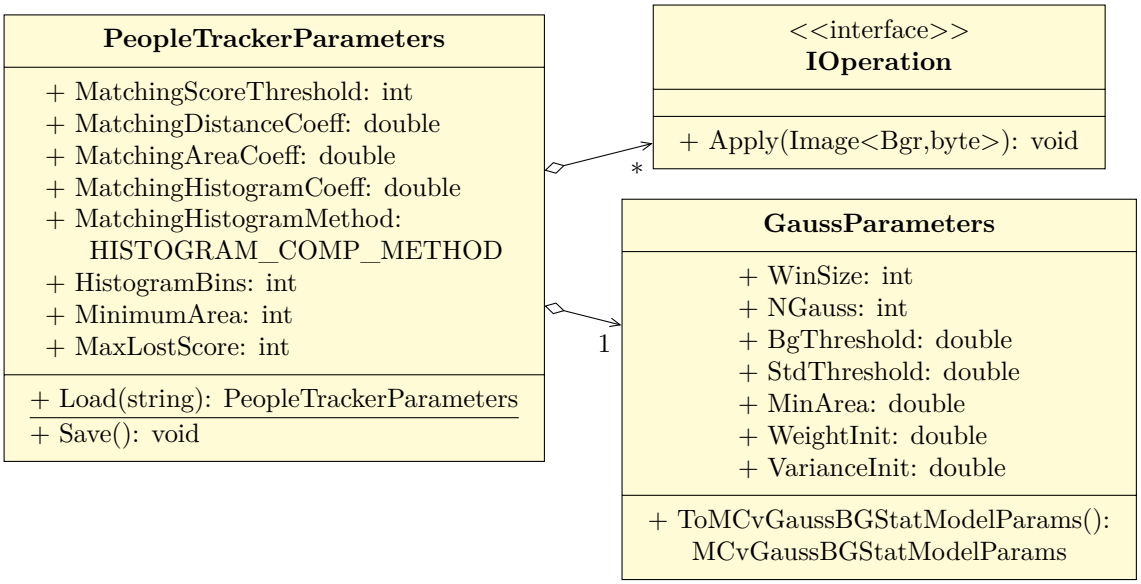


Figure B.5: The PeopleTrackerParameters class diagram

B.7.1 Properties

B.7.2 Operations

B.8 Preprocessing Operations

These structures represent the operations applied to the foreground in order to find the contours of the blobs. It basically wraps EmguCV functions in classes implementing the same interface, in order to add modularity.

The details for these classes have already been given in section **??**. Diagram B.6 shows the modularity of these classes. It gives the convenience introduced in Appendix A concerning the possibility of modifying the applied operations at runtime.

B.9 Tracking

Figure B.7 shows the Tracking system organization.

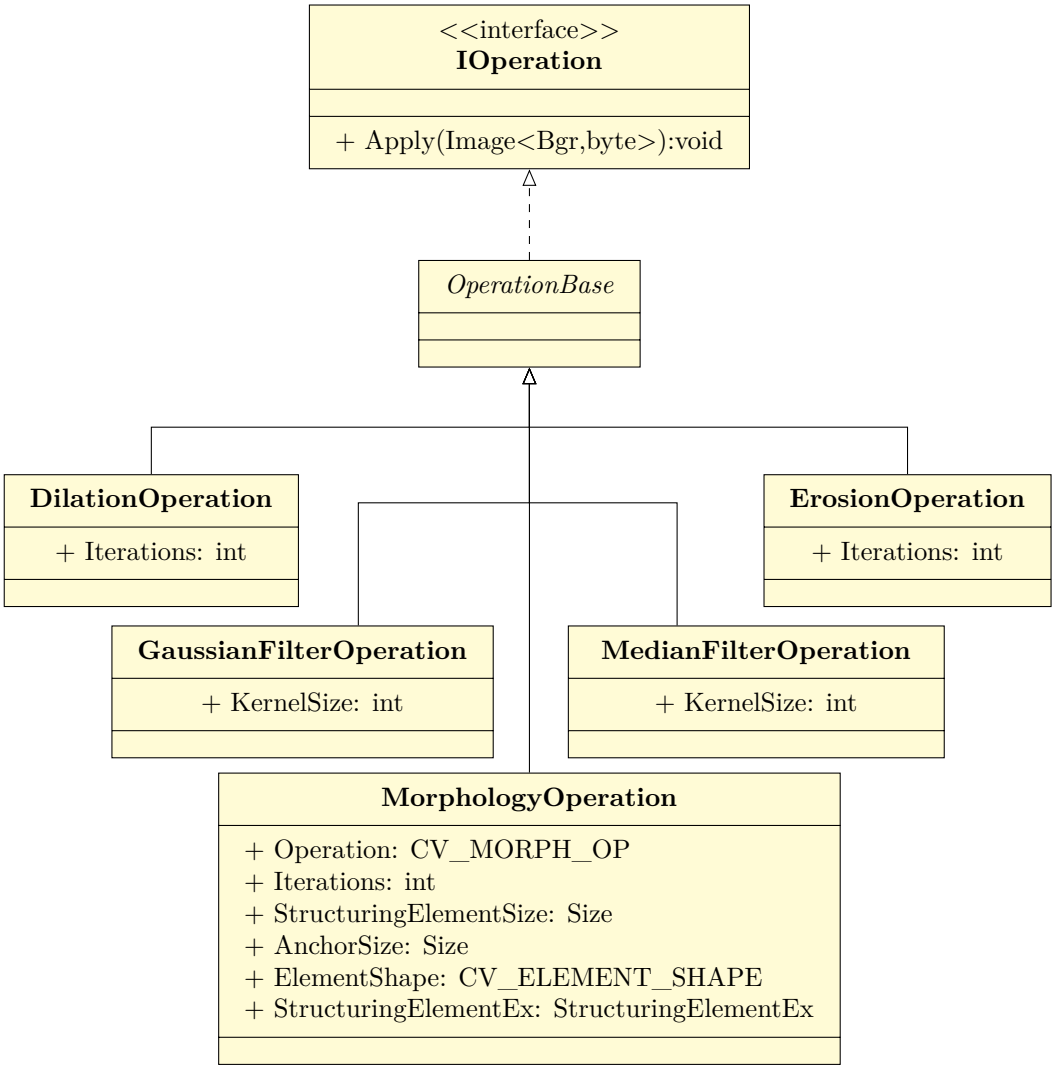


Figure B.6: Operation class diagram

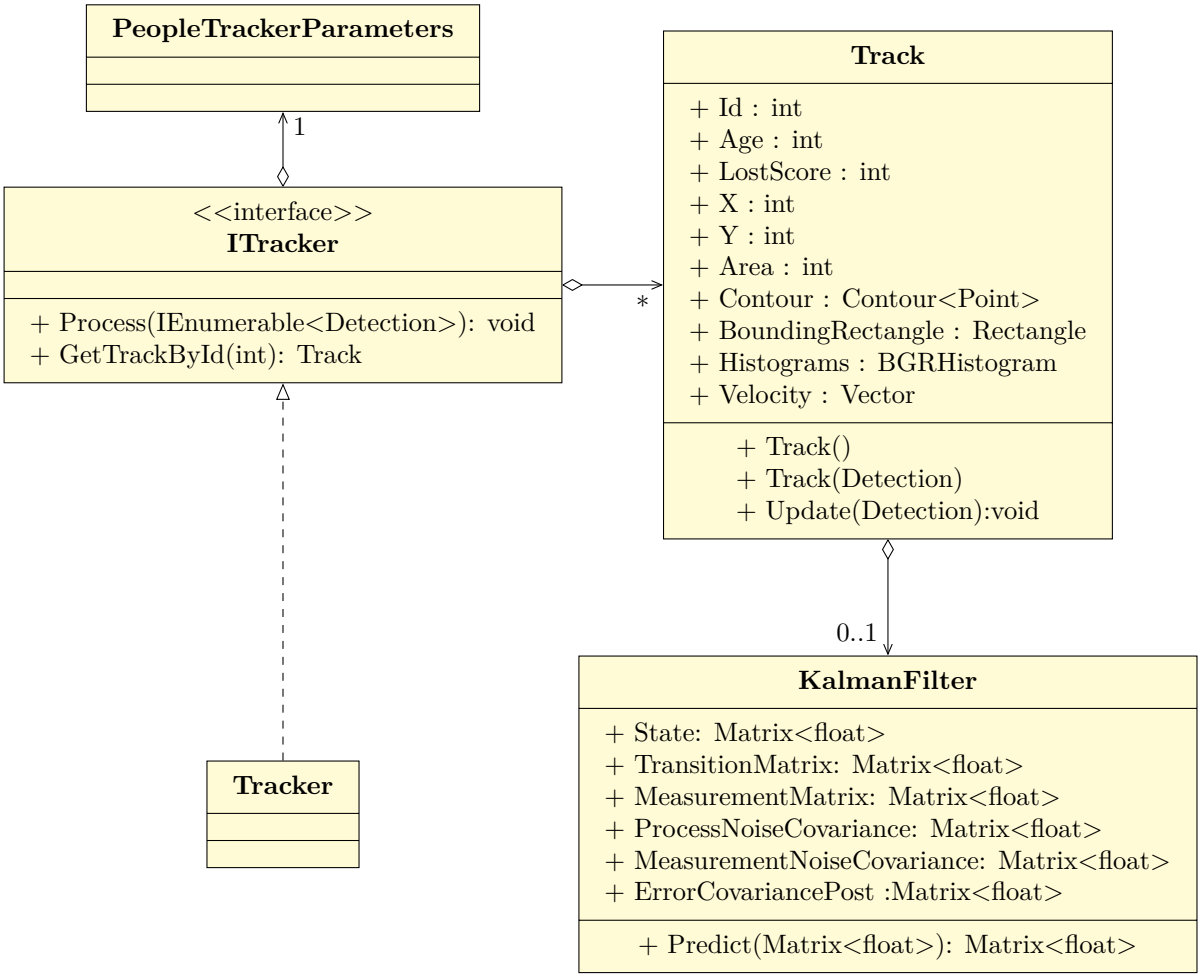


Figure B.7: UML Representation of the Tracking system

Appendix C

Sources

Enclosed is a DVD-ROM containing the following.

- "VGIS 1021 - Mater Thesis - 2012.pdf": this Thesis Report.
- "setup.exe" - installer of the software.
- "Sample Videos": some videos to be used with the software.
- "References": some of the articles used for this master thesis.
- "Binaries":
  - "PeopleTracker.exe": Binary files of the main program
  - "HOG.exe": utility to extract HOG features from a set of images
  - "SVM.exe": utility to train a SVM model (to be used with output of HOG). Nb: this software uses SVMLight[22].
- "Sources":
  - "PeopleTracker": sources of the main program and the different utilities
  - "packages": all the *NuGet* packages for the sources, to be used with *Visual Studio* plugin *NuGet Package Repository*. Nb: put these files in a local nuget repository or in the *packages* of the solution.

Bibliography

[1] Wikipedia, "Closed-Circuit Television — Wikipedia, the Free Encyclopedia." `http://en.wikipedia.org/wiki/Closed-circuit_television`, 2002. [Online; accessed 22-February-2012].

[2] N. Dalal and W. Triggs, "Histograms of oriented gradients for human detection," *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition CVPR05*, vol. 1, no. 3, pp. 886–893, 2004.

[3] M. D. Breitenstein, F. Reichlin, B. Leibe, E. Koller-Meier, and L. Van Gool, "Online multi-person tracking-by-detection from a single, uncalibrated camera.," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. October, pp. 1–14, 2010.

[4] D. M. Gavrila and S. Munder, "Multi-cue pedestrian detection and tracking from a moving vehicle," *International Journal of Computer Vision*, vol. 73, no. 1, pp. 41–59, 2006.

[5] C. Stauffer and W. E. L. Grimson, "Adaptive background mixture models for real-time tracking," *Proceedings 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Cat No PR00149*, vol. 2, no. c, pp. 246–252, 1999.

[6] S. Kumar, "Shadow detection and removal in colour images using matlab," *International Journal of Engineering Science*, vol. 2, no. 9, pp. 4482–4486, 2010.

[7] M. J. Swain and D. H. Ballard, "Color indexing," *International Journal of Computer Vision*, vol. 7, no. 1, pp. 11–32, 1991.

[8] A. W. R. Fisher, S. Perkins and E. Wolfart., "Gaussian smoothing." `http://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm`. [Online; accessed 26-April-2012].

[9] T. B. Moeslund, *Image and Video Processing*, vol. 2. Aalborg University, 2009.

[10] T. B. Moeslund, *Introduction to video and image processing*. Springer, 2012.

[11] G. Bradski and A. Kaehler, *Learning OpenCV*. O'Reilly Media Inc., 2008.

[12] A. Ghuneim, "Contour Tracing Algorithms." http://www.imageprocessingplace.com/. [Online; accessed 17-May-2012].

[13] Wikipedia, "Connected-Component Labeling — Wikipedia, the Free Encyclopedia." http://en.wikipedia.org/wiki/Connected-component_labeling, 2002. [Online; accessed 14-May-2012].

[14] Wikipedia, "Image Moment — Wikipedia, the Free Encyclopedia." http://en.wikipedia.org/w/index.php?title=Image_moment&oldid=491600617, 2012. [Online; accessed 14-May-2012].

[15] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[16] Wikipedia, "Support Vector Machine — Wikipedia, the Free Encyclopedia." http://en.wikipedia.org/wiki/Support_vector_machine, 2002. [Online; accessed 15-May-2012].

[17] R. E. Kalman, "A new approach to linear filtering and prediction problems 1," *Journal Of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.

[18] E. T. B. Dereje Woldemedhin Kifle, "Tracking with the kalman filter." http://mmlab.didi.unitn.it/wiki/index.php?title=Tracking_with_the_Kalman_Filter&action=edit. [Online; accessed 14-April-2012].

[19] N. Dalal, "Inria person dataset." http://pascal.inrialpes.fr/data/human/, 2005. [Online; accessed 23-May-2012].

[20] C. Yildiz, "An implementation on histogram of oriented gradients for human detection,"

[21] T. Joachims, "Making large-scale svm learning practical," *Advances in Kernel Methods Support Vector Learning*, pp. 169–184, 1999.

[22] T. Joachim, "Svmlight. support vector machine." http://svmlight.joachims.org/. [Online; accessed 23-May-2012].

[23] EmguCV, "EmguCV Tutorial." http://www.emgu.com/wiki/index.php/Tutorial, 2008. [Online; accessed 27-April-2012].

[24] H. Shahid, K. Khan, and W. A. Qazi, *Using modified mixture of gaussians for background modeling in video surveillance*, vol. 3, pp. 155–159. Ieee, 2008.

[25] J.-M. Pelletier, "A Simple OpenCV Tutorial." http://jmpelletier.com/category/tutorials/, 2009. [Online; accessed 7-May-2012].

[26] T. B. Moeslund, A. Hilton, and V. Krüger, "A survey of advances in vision-based human motion capture and analysis," *Computer Vision and Image Understanding*, vol. 104, no. 2-3, pp. 90–126, 2006.

[27] R. Duda, P. Hart, and D. Stork, *Pattern classification*. Pattern Classification and Scene Analysis: Pattern Classification, Wiley, 2001.

[28] P. KaewTraKulPong and R. Bowden, *An improved adaptive background mixture model for real-time tracking with shadow detection*, vol. 1, pp. 1–5. Citeseer, 2001.