University of Edinburgh
Division of Informatics

Correlated Optical Flow for Crowd Scenes

4th Year Project Report
Artificial Intelligence and Software Engineering

Salafeen Khan

March 4, 2005

Abstract: This paper describes a set of techniques for a framework applied to an automated surveillance system. The paper implements a system designed to detect dense anomalous flow in crowd scenes [5]. This allows modelling of crowds and automatic detection of abnormal events in these crowds. The system utilises robust statistical techniques for background estimation and to classify abnormal events in crowds.

Acknowledgements

I would like to thank Gillian Hayes and Ernesto Andrade fore their tireless help during the process of development of this dissertation. I would also like to thank Bob Fisher whose idea was inspiration for this project, and acknowledge the resources and help from the Institute of Perception And Behaviour (I.P.A.B.), in Edinburgh University.

Contents

1. Introduction

Increasing interest has gone into discovering better automated analysis of surveillance footage. The increased use of surveillance technology has necessitated the need for its automation, as the increase in data becomes too large to analyse by humans. Movement and tracking in dense crowds is one area that has need for a automated process. We cannot perform robust visual tracking of people in a dense crowd due to the continual occlusion covering of a pixel regions being tracked. One way of overcoming this is to use optical flow. Optical flow is the estimation of motion of correlated pixel regions in two images, where motion vectors are calculated to show the magnitude and direction of the movement of these regions. The flow of vectors in an image can be classified to detect regions of anomalous flow. The anomalous flow can be categorised as abnormal movements in a crowd. This paper aims to describe a robust system that can detect anomalous flow in dense crowd scenes, which can be utilised as a practical method for crowd scene surveillance.

1.1 Motivation

Techniques for tracking a person in a crowd do exist [7], however detection of the behaviour of an entire crowd has not been thoroughly examined. The main aim of this paper is to develop a framework that enables the detection of crowd movements utilising optical flow estimation. Real-time optical flow estimation is a useful technique in detecting motion and its main area of application has been in robotics [16] and navigation, but it has not been currently applied to crowd movement classification. This paper will show how abnormal events in crowds can be detected by analysing the patterns of flow through a successive stream of images.

The purpose of the framework is eventually to have a system that can detect and classify regions abnormal behaviour in a scene. There are several applications for such frameworks, including surveillance of crowds leaving large buildings and detecting, for instance, when someone has fallen over, or when a fight occurs in a moving crowd.

1.2 Overview

In the process of developing the framework for this system certain components have been identified. The description of these components and their place within the framework is given in chapter 3. Chapter 2 lays out the background and research work done in this paper for the development of the framework. The chapters following the background work describe the actual framework starting with the background estimation methods used in the system, in chapters 4 and 5. The description of the optical flow detection and theory is in chapter 7. Then the part of the framework that is involved in classifying the flow is described in chapter 8. The interface developed for the use of this framework can be seen in the Appendix A.

2. Background work

This chapter describes previous work in the area of tracking in image sequences, and other background related work done in this project. This includes the how the datasets used for this framework were captured, use of additional software to this framework as well the platform used to develop it.

2.1 Previous work

Various papers on frameworks in the area of tracking and surveillance have come out relatively recently. The papers employ various different methods in background estimation and other methods in terms of tracking. Most of these papers involve statistical and probabilistic methods for tracking regions in a scene.

2.2 Framework

Haritaoglu et al [7] looks at surveillance using the approach of studying complex interactions between people in closed circuit television environments. This work is similar to the type of environment this framework would be applied to. The paper describes the W^4 framework and its component parts in background removal, probabilistic tracking and behavioural monitoring of people. However the W^4 framework concentrates more on identifying and classifying behaviour of individuals, whereas the attempt by the system proposed in this paper concentrates on analysing and classifying crowd movements as a whole. The W^4 framework is capable of tracking less crowded areas and regions where occlusion of various parts of the human would not occur. Again the framework in this paper is designed to work with dense crowds were occlusion is common, but does not deal with the complex interactions between people. The assumption used in Haritaoglu et al's' [7] paper is that most crowds that the W^4 framework would be applied to, would not be too dense, so as to allow tracking and segmentation methods to work.

Haritaoglu et al's [7] basic model for background removal is the basis for many other papers, including this one, that are implementing similar automated surveillance frameworks. This paper presents a similar model for background estimation in chapter 4. The system uses a normal distribution probability model for background estimation. The technique is explained in chapter 4.

Stauffer et al [4] tracked moving objects in a scene, as well as monitoring the volume of observed traffic in a scene. Like the framework presented here this system is designed to work in an outside environment. Statistical pattern analysis was described in this paper and applied to traffic scenes. The system monitors change in behaviour over long periods of time and is only applied to traffic scenes.

The system consisted of robust tracking detection, such as the optical flow detection section of this framework (see chapter 7). For background removal, an on-line mixture model is used, which simply consisted of a multiple Gaussian model for the pixel intensities over time (see chapter 4). The system then finds and models regions of connected components to help identify object shapes. Finally Kalman filtering and probabilistic methods are applied to predict motion in a scene. Once motion has been detected the system does not actually classify the motion based on predetermined ideas, but actually learns the pattern of behaviour probabilistically. This involves the use of hidden Markov models and the use of probability mass functions [1].

Kang et al [10], concentrated on automated surveillance in a specific indoor environment, namely airport terminals. This was a system that would be able to identify the same object in one camera and the next. The background estimation method used is designed to work as a fast method in an indoor environment, typically only 5 frames are being used to model the background for the system. This is probably sufficient for an indoor environment that would have very little noise compared to an outdoor one. Tracking of objects then consisted of looking at ratios between size and frame and then using information for point to point identification of an object in multiple cameras.

Maurin et al [13],[12] looks at crowded traffic scene surveillance. They closely follow the work that is presented in this paper. However simple statistical and segmenting techniques were used in the application for classifying and identifying abnormal events. Maurin et al uses the same background estimation technique as Haritaoglu et al's [7], by applying a single Gaussian model to a pixel intensity value through time. Motion is extracted by applying optical flow to crowd scenes, and modelling the object regions for tracking purposes.

In both related papers Maurin et al [13],[12] states that Black's and Anandan's [2] method of fast transforms using pyramids to detect motion is too complex to implement in real-time (see chapter 7). The same algorithm is applied for this framework. Although not optimal for online calculations of flow, the software provided by Black [2] does achieve nominal results when compared to the images in a reasonable amount of time.

To detect flow the paper uses sum of absolute differences (S.A.D) method to look at small regions to detect flow spatially. The objects were tracked using a Kalman

[1]pmf

filter to monitor state information. A Kalman filter simply attributes probabilities to particular separated regions in the image to predict their movement. The system did not however attempt to segment regions that became occluded.

Siebel and Maybanks [19],[18] cover the ADVISOR (Annotated Digital Video for Intelligent Surveillance and Optimised Retrieval), an actively working system used in the Barcelona Metro, and present a description for fusing together multiple tracking techniques to identify people in scenes. The paper on robust people tracking [19] involved identifying blobs by segmenting the heads and hands, using colour to segment these parts. Our framework only uses grey-scale intensity images based on dense crowds scenes, so segmenting this information is not necessary. The paper on ADVISOR [18] describes a complete framework, made up of capture, motion detection, frame to frame tracking, crowd monitoring, behaviour detecting, Human Computer Interface and archiving. Although the papers [19],[18] describe surveillance frameworks much like the one presented here, not much analytical data of the system is provided.

Crowd analysis with optical flow can be seen in Reisman et al [15]. Its application is based on a non-static background for pedestrian identification. The problem presented in this paper is far more simple to implement, where the background will largely remain static (with the exception of anomalies), but has to cope with classifying different types of crowd movement and their identification. The paper describes a system that looks at probability distribution functions (p.d.f) of inward motion i.e. motion going from left to right or right to left, to detect motion of pedestrian crowds from cars.

Systems that use background subtraction give good results for tracking but fail to track multiple objects over a certain threshold [11],[7]. Maurin et al [12] can also detect movements in crowds in real-time by applying optical flow estimation, however there is no application of classifying motion of a particular event.

This work uses part of the framework described by S. Gnaneshwar [5]. The work presented here is a continuation of her work, and also its practical application in detecting anomalous flow.

2.3 Background removal

For the purposes of the development of the framework specific papers describing background estimation techniques were analysed. The following papers can be categorised into two main groups, using a single Gaussian distribution or using multiple Gaussians to model background pixel intensity. There are also some papers that propose alternative methods of background estimation such as Ridder et al [17], which uses a Kalman filtered model to predict whether a pixel would be foreground or background.

Dar-Shyang et al [8], used a K-Gaussian technique model with an additional Bayesian method to decide whether the pixel is background or foreground. However no clear benefits are shown for video surveillance methods, as to why Bayesian decision making is optimal over thresholding (see chapter 4).

Dufaux et al [3] used background segmentation techniques using a single Gaussian, but then using motion estimates to robustly detect background pixels. The paper calls the procedure mosaicking, and a similar method is produced for the framework in chapter 7 and 4. The system used a parametric motion estimate to compensate for overall motion in the camera.

Another paper that looks at alternative background removal methods is Gordan et al [6]. The paper uses K-Gaussian estimation, but applies this to colour images as well as depth information. This is unlike the background estimation procedure presented in this paper, which is applied over grey-scale intensity images in chapter 4. Although the results of the method are fast and it has good detection rates, the method requires specialist sensors to gain depth and colour information, which would not be applicable in typical surveillance environments.

2.4 Optical flow modeling

This section describes papers in the development of optical flow estimation. Some present new techniques, while others try to build on the model of the flow estimation to gain information about the scene.

In Ohtas paper [14] a new optical flow technique is presented, which takes into account noise in the image. The system uses Eigen-values of the vectors and computes intensity derivatives to increase the accuracy of flow information. The covariance matrix of the noise in the data is estimated and the distance from the current vector and the covariance is measured and compared to a threshold.

Yacoob and Black [1] presents techniques for parameterised optical flow to help tracking information in an image. They use eigen-values of the flow data to produce various tracking and identification methods. The eigen-values are a robust t way of calculating image information of a particular object. However on-line calculation of eigen-spaces is not described in this paper.

2.5 Summary

Here we present the summary of all the papers read for the background work for this dissertation. The table describes the relationship of this framework to all the previous and current work available.

The motion modeling papers are not included here. The papers generally present techniques rather then directly transferable frameworks for automated surveillance.

Paper :	Environment	Background Estimation	Type of Classification
Dar shyang st all	-	K-Gaussian	-
Dufaux et al	-	Single normal distribution	
Heriatagu et al	outdoor	Single Gaussian	Individuals
Kang et al	indoor	simple technique	Individuals
Khan	outdoor	single Gaussian/simple	Crowds
Mausin et al	outdoor	Single Gaussian	Crowds
Reisman et al	-	-	Crowds
Ridder et al	-	Kalman filtered model	
Seibal et al	-	-	Individual
Stausffer et al	outdoor	Multiple Gaussian	Crowds

2.6 Image capturing and flow detection

The datasets used in this project try to follow C.C.T.V. type footage from real surveillance cameras as closely as possible. The project was unable to obtain datasets that were actually from C.C.T.V. cameras, because of data protection act issues. Instead the datasets used were from raw digital camera footage. The frames decompiled from the video footage were set to grey-scale, as apposed to using colour ranges of Red Green and Blue (RGB). Grey-scale consists of values 0-255 intensity image, where 0 is white and 255 is the darkest black However to keep as much data as possible it was decided to keep the original frame-rate when it was captured. This helped retain data integrity to allow for later manipulation of the images for testing.

A Panasonic NV-DX110 with 24x zoom DV camera was used to capture video sequences of crowd movements. The camera was held in a static position, while simulated crowd movements were being recorded. These consists of crowds of about 10 people moving a across the scene. Simulated opposing flow was generated by people moving across the scene in a different direction to the majority crowd. he different types of crowd movement were simulated to generate datasets that closely followed different types of anomalous crowd movement. These crowd

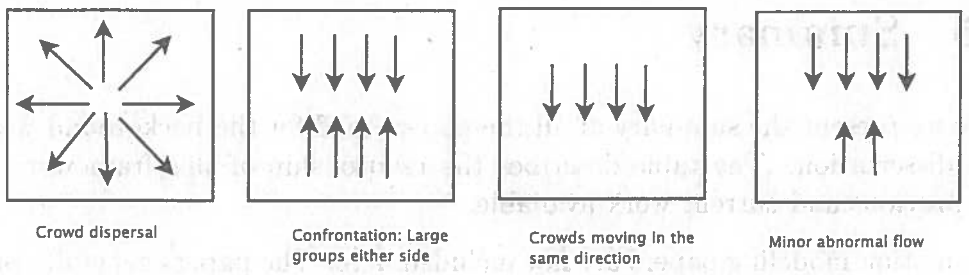
Crowd dispersal
Confrontation: Large groups either side
Crowds moving in the same direction
Minor abnormal flow

Figure 2.1: This figure describes the plan view of simulated crowd movements taken by the camera as datasets used for the framework

movements consisted of a group of 10 people moving across the scene in various directions, which are described in figure (2.1). The intention was for these datasets to represent various types of abnormal flow for the system to classify. Datasets were also simulated using graphics packages for testing purposes of that framework which is described in the results section in the following chapters.

The position of the camera was held above the height of the crowd at about 5.2 metres. This was done to reduce occlusion of people in the scene (see fig(2.2). The zoom was not adjusted at any time and was set to 10 x normal to remove unwanted areas in the scene. The camera was held on a tripod as statically as possible, however analysis of the datasets has shown that movement due to wind was present in the video stream. This was however compensated for in the classification section of the framework (see chapter 8). The video was streamed into Windows Media Video (WMV) 320x240 high resolution on a Windows PC and the video file was then sequenced into images using open source media software Zwei-Stein (see Appendix C). The images were first converted to bitmaps (bmp) to retain as much information as possible. The images were then converted from bitmaps to grey-scale intensity (portable grey map) PGM files. Use of formats such as jpeg was not used due to loss of information through compression. This compression would alter large regions of grey-scale intensities in a 320x240 image. This would in turn affect the background estimation and flow detection results as they use the assumptions that the intensity values remain static unless movement or foreground pixels are present. These images were then converted into the data format that the platform uses; i.e. Matlab data format files (see section 2.7).

Flow detection in the framework described in this paper (chapter 7) uses an algorithm devised by M. J. Black (see [2]). The optical flow detection software uses P.G.M. images and to reduce time in changing compression format the entire framework uses this format for processing purposes. The parameters given to the algorithm determine how much precision in the measurements of optical flow is given (details of and analysis given in chapter 7).

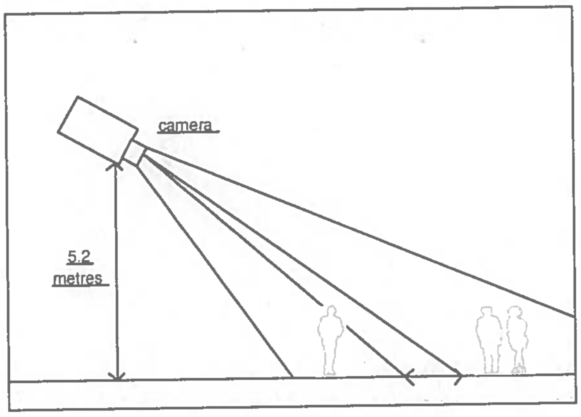

Figure 2.2: Camera was placed 5.2 metres above the subjects and crowd movement was monitored.

2.7 Platform

The framework uses the Matlab (see fig(C.1) in Appendix C) environment for analysis and implementation. The framework was run on the Red Hat 9 operating system on a Pentium 4 desktop PC. The advantages of using the Matlab environment were that it provides a built-in image processing library and fast and optimised calculations of large matrices. We are dealing with statistical pattern matching with these matrices, and Matlab (see fig(C.1))also provides a Statistical library. Matlab treats the datasets of images as matrices of pixels to be manipulated and converts them into a format to allow for this manipulation.

Although the Matlab environment is user friendly and has many built in features for processing, it has some disadvantages as well. The Matlab environment uses a lot of memory when dealing with large datasets of images and this can slow down the process of calculations especially when they concern each pixel. These conclusions can be found in chapter 9 and in the Results section 8.3 of chapter 8.

[illegible]

2.7 Platform

The implementation uses Matlab (see ref [illegible]) [illegible] for the purposes of implementation. The framework was run on the [illegible] 9 operating system on a Pentium 4 [illegible] PC. The advantages of using the Matlab environment were that it provides a built-in image processing library and fast and optimised calculations of large matrices. For the dealing with [illegible] signals and pattern matching with these matrices, and Matlab (see [illegible]) provides a statistical library. Matlab treats the datasets of images as matrices of pixels to be manipulated and converts them to a format to allow for efficient calculation.

Although the Matlab environment is user friendly and has an excess of [illegible] for processing it has some disadvantages as well. The Matlab [illegible] uses a lot of memory when dealing with large datasets of images and this can slow down the process of calculations especially when they need to be re-used. These considerations can be found in chapter 3 and in the Results section 5.2 of chapter 5.

3. Description of framework

The framework can be separated into 4 distinct parts. The first part consists of background estimation, allowing the system to separate foreground regions from background. The detection of the foreground regions is discussed in chapter 4 and improvements to the background detection can be found in chapter 5. The red highlighted part of the framework diagram fig(3.1), shows the whole background estimation module of the framework. This includes Morphological filtering which improves background estimation by removing errors in chapter 6. Motion estimation from the optical flow module, also plays a role in the background process as we will see in chapter 5. After morphological filtering is done we improve further background removal removing the static (non-moving) foreground parts of the image. The framework then produces a binary image, locating all the foreground objects and identifying the background.

The next part of the framework is concerned with the detection of motion[1], between two successive images in a sequence. This part of the framework, is called the optical flow (see fig(3.1)) module and is explained in chapter 7. Optical flow detection is not dependant on any of the background estimation results. So it can detect flow and use this information to improve background estimation results.

Once both background detection and the corresponding flow vectors are identified, the framework can then use this information to classify the flow of the image regions. The classification of the motion vectors produced by the optical flow detection, is the third section of the framework and it is treated in chapter 8.

The final section in the pipe-lined diagram is the display of classification results. This part of the framework is described in Appendix A.

[1]using the images provided

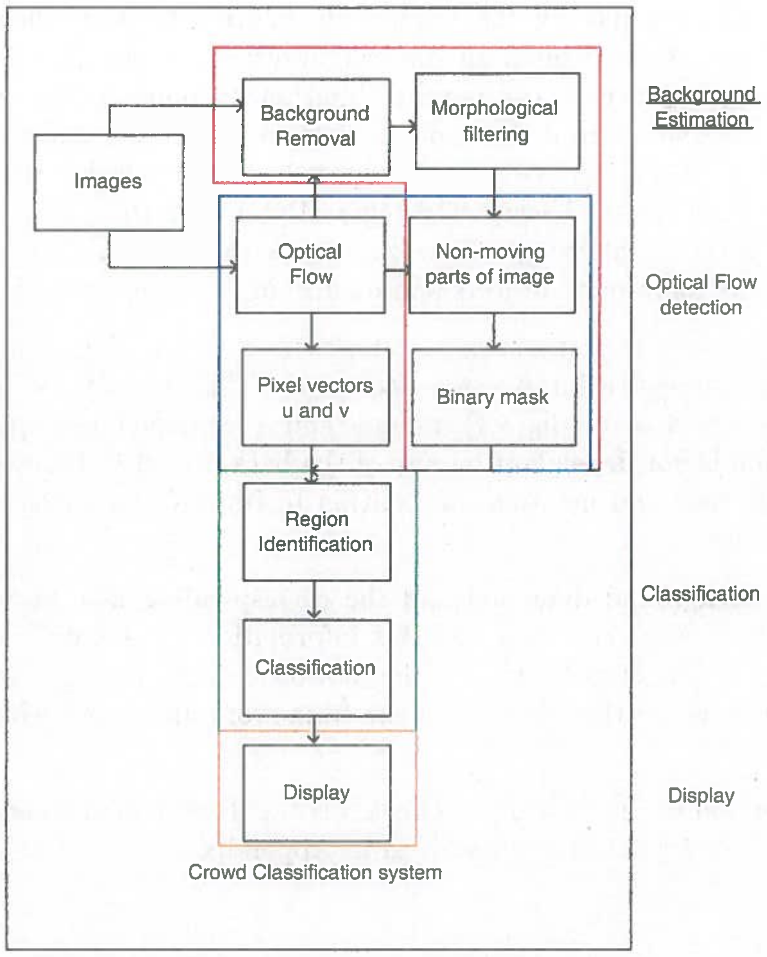

Figure 3.1: Pipelined flow diagram of framework

4. Statistical background estimation

Before we can look at the movement of crowds in a scene we have to define how we are going to locate crowds within an image. This chapter presents two methods for detecting and locating foreground pixels, using statistical analysis. Foreground pixels are assumed to consist of the objects, that we are interested in. The hope is that the image background will remain static and the foreground pixel regions in the image will be the only things moving (why this is important will be explained later). Both methods use similar statistical analysis to detect background from foregrounds. However the way they estimate the background median is different.

The images that we are dealing with consist of grey-scale intensity values. This means that the image can be treated as a matrix of values, where each value consists of a pixel intensity in the image. Each image consists of $320 * 240$ pixels, which all have a value from 0-255 as described in section 2.6 in chapter 2. Treating the image matrix as a dataset allows us to use these statistical methods to identify patterns in the image. However we cannot simply identify background pixel intensities from foreground ones with a single image alone. Using several frames in an image sequence, allows us to model each pixel intensity value through time. This is generally basis for most techniques in background estimation [7][9]. So taking each pixels intensity value in time, we can then go on to look for patterns within the sequence of images, the techniques are presented in sections 4.2 and 4.1 of chapter 4.

There is also another constraint to the process of background estimation. That is in a long sequence of images, the assumption that the background will remain static and the background illumination will be constant can be broken (as we are dealing with outside environments). So there is a need for this framework to be able to deal with these changes in conditions, and hence be adaptable. Both methods presented here are developed to sample a small range sequenced of images to detect an adaptive model for the pixel intensities.

4.1 Adaptive background model

Sampling a number of images we can identify the frequency of a particular pixel position, and see what intensity values are most frequent in that sample. A depiction of what an idealised sample might look like is shown in fig(4.1). This however is not necessarily what a pixel intensity distribution may behave like over the sample. The hope is that we can deal with pixel distribution over a sequence and relate it to a normal/Gaussian distribution. The idea behind this model is that the number of pixel intensities over a sample of a fixed size will correlate to a central mean/median value. If we sample this type of distribution and then go on to compare it to a new pixel intensity we can estimate how closely it follows this model. The assumption made by using this distribution is that the background median corresponds to the average background intensity value. So the hope is that the most frequently occurring pixel is of the background intensity.

Dealing with video sequences allows consecutive image frames to be stored to build the sample model of pixel distributions (see fig(4.3)). With this data the median of the sample is needed so that we can identify the most prominent pixel. The median is defined in equation 4.1. Let us take an ordered set of intensity values X and say $Y_1 = min_j X_j, Y_2, Y_3, ...Y_N = max_j X_j$, where Y_1 to Y_N, where N represents the number of frames/images in the sample. So essentially we have a bunch of ordered intensity values using a range from 0-255. The median of this set of values is defined by

$$\bar{x}_{previ,j} = \begin{cases} Y_{(N+2)/2} & \text{if N is } odd \\ \frac{1}{2}(Y_{(N+1)/2} + Y_{1+(N)/2}) & \text{if N is } even. \end{cases} \tag{4.1}$$

The median $\bar{x}_{previ,j}$ is simply the middle value in the distribution.

The median is chosen as a value in this sample distribution, because it is robust statistically compared to values such as the mean. Robust being defined by the fact that the median is not affected by large variances in a sample that are not frequent in the distribution i.e. there would be no change in the median value unless there is a more frequently occurring intensity in a distribution. The need for a robust estimation of the data is explained later. Variance of a particular pixel intensity is given by equation 4.2.

$$var = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2 \tag{4.2}$$

Variance is essentially the distance of one pixel intensity from the mean intensity. The median is also acts much like the mode of the data, because it usually is the most frequently occurring value.

Once the median is calculated it is important to know the variance of the sample data. Because we want the model of the data to fit a normal distribution, the variance defines the width of this distribution. If we look back at the idealised model of pixel intensities in fig(4.1), we can use this model to classify whether a pixel belongs to this particular distribution. If say a new pixel intensity is

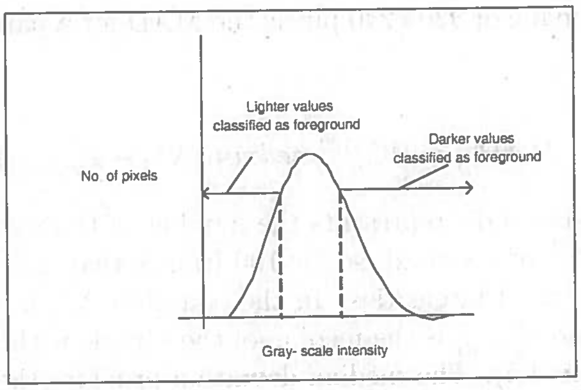
Lighter values classified as foreground
Darker values classified as foreground
No of pixels
Gray- scale intensity

Figure 4.1: This is an idealised distribution of pixel intensity over time

identified for that position in an image, we can tell whether it is part of the distribution if it's intensity value lies somewhere out of the range of the variance of the median. So pixels that are either too dark or light in intensity compared to the median are classified as not part of the current distribution. As we stated before the assumption is that the most frequently occurring pixel in a sample should be the background. So when a pixel is not part of a sample distribution we can classify it as a foreground pixel, rather than a background.

Going back to the point made in the beginning of this chapter, that foreground pixels are the moving objects in the scene is an allows us to assume that the background will remain largely static within the sample. And so the foreground pixels will be of a differing intensity value than the background and will cover a pixel for short period of time relative to the background pixel.

To detect the variance of a background pixel from around its median we use Median Absolute Deviation (MAD). We take a sample of of pixel intensities through time t as they are sequenced from the video. These initial images act as the sample, in which we identify the background median intensity. For this framework 100 frames were used as an sufficient number of frames to work with. This sufficient enough hopefully to contain large frequencies of background pixels. Going back to the other point made, that there was a need for a statistically robust values for the centre of the distribution (the median), because of the window size of the sequenced images. If we look at the ideal distribution in fig(4.1) we can see that we do not sample the entire distribution, and select a narrow region around the median. This is represented by the window. The window being the 100 images analysed by the framework. Within this window of a restricted size (due to computational memory restrictions), the sample can be affected by intensities it encounters that do not correlate with the majority of the viewed intensities. That is why the median should be robust enough to handle these values unlike the mean.

For an $i*j$ image made of $320*240$ pixels the MAD for a particular pixel is given by equation 4.3.

$$MAD_{i,j}^{t} = W_{t=1}^{t=100} median(|X_{i,j}^{t} - \bar{x}_{previ,j}^{t}|) \tag{4.3}$$

W is the window size and t represents the number of frames through time. Each frame runs at $1/25^{th}$ of a second, so for 100 frames that is 4 seconds to view the distribution of the pixel intensities. In the equation $X_{i,j}^{t}$ is all the pixels in the sample at time t and $\bar{x}_{previ,j}^{t}$ is the median of the sample without the current pixel ($X_{curri,j}$in equation 4.5). The median deviation provides the average amount of distance the sample of pixel intensities is from the median intensity. By applying equation 4.4

$$\sigma_{i,j}^{t-1} = W_{t=1}^{t=100} MAD_{i,j}^{t} * \beta \tag{4.4}$$

we can find the variance σ. Where $\beta = 1.4826$ This allows us to look at 1 standard deviation from the median. This width allows us to classify pixels as foreground or background using the following equation 4.5.

$$\frac{|X_{curri,j}^{t} - \bar{x}_{previ,j}^{t-1}|}{\sigma_{i,j}^{t-1}} < \lambda \tag{4.5}$$

By normalising the difference between the current pixel and the estimated median of the sample data at $t-1$, that does not have the current pixel in its sample. λ is the threshold for classification of the data and is set to 0.5. As we are dealing absolute values λ is always positive. In section 4.3 we explore changes in the threshold value.

When implementing the adaptive approach to background estimation we looked at a sliding window implementation. This meant that because of the limit of 100 frames to sample the pixels, the framework needed update the sample continually with the current pixel. This is by adding frames in a Last In First Out (L.I.F.O.) order when removing frames, and then update the data structure with the current frame. This allowed the framework to have a continually updated model for the background. It also allowed for the system to accommodate for any changes in the background such as latent motion from the camera. A description of the algorithm used for the buffered images in the sliding window is given in fig(4.2).

4.2 Fixed background model

A variation of the adaptive background model is described here. The impetus for devising a modified version of the adaptive method was from the use of small

```
Proc BufferWindow(counter,image)
          global X; % X is initialised as X = repmat[0,[240,320,100]]
if (counter > 100)

          foreach n = 1:99
                    X(:,:,n) = X(:,:,n+1); % shift all images along one
                                           % to buffer new image
          endfor
          foreach i = 1:240
                    foreach j = 1:240
                              X(i,j,100) = image(i,j);     % adds new value to the end of the
                                                           % window
                    endfor
          endfor
endif
else
          foreach i2 = 1:240
                    foreach j2 = 1:320
                              X(i,j,counter) = image(i,j);
                    endfor
          endfor
endelse
```

Figure 4.2: Description of the buffer window method for storing images

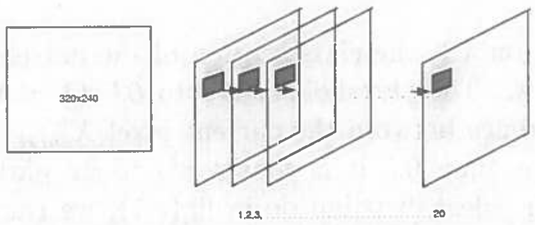

Figure 4.3: Pixel position through several frames, the median is found in this list

datasets that did not contain more than 80 frames. Half of these frames contained foreground pixels in their sequence. This meant that the adaptive approach would not have enough frames to produce a model in which it could identify reliably the background, this is discussed in the results, in section 4.3. To resolve this problem, an implementation of the adaptive method was given a fixed model of the background, the reasons for this is described in this section.

To gain a fixed model of the background, images that contained just background pixels where there no foreground regions were identified offline. An automated approach is presented in chapter 5 and described in chapter 7. The model of background pixels contained 30 frames and is depicted in fig(??). Using this fixed model the system is able to identify what the background median is for this sample.

The procedure in the fixed background method is much like the adaptive one presented in section 4.1. Both procedures use the background median and identify the variance with the sample of pixels from incoming frames. However in the adaptive method it is assumed that it can identify the background median from the sample it has received, whereas in this method we know what the background median is. Essentially this is fixing the background so that it is not reliant on the received samples. This allows the procedure to run on small datasets where most pixels are foreground, most of the time through the sequence.

The median of the background is defined as $\bar{x}_{backi,j}$, with this value we proceed to find the Medium Absolute Deviation. Equations 4.6 and 4.7, show the modified definitions of the MAD function and the thresholding of the current pixel.

$$MAD^t_{i,j} = W^{t=100}_{t=1} median|X^t_{i,j} - \bar{x}_{backi,j}| \tag{4.6}$$

$$\frac{|X^t_{curri,j} - \bar{x}_{backi,j}|}{\sigma^{t-1}_{i,j}} < \lambda \tag{4.7}$$

As described in section 4.1, the classification of the current pixel $X^t_{curri,j}$ is decided on the value λ. The threshold is set to 0.5 ($\lambda = 0.5$), which means if the normalised difference between the current pixel $X^t_{curri,j}$ and the background median $\bar{x}_{backi,j}$ is less than 0.5 it is considered to be part of the background. Referring back to the ideal distribution in fig(4.1), we can see that not only is the window size is restricting what we see of the distribution, the median deviation also restricts the size of values in intensity that are considered background. Once we have found what side of the decision boundary $pixel_{i,j}$ is on (foreground or background), we can then build a binary mask for the ij^{th} position (0 being background and 1 being foreground). Examples of these masks can be seen in section 4.3.

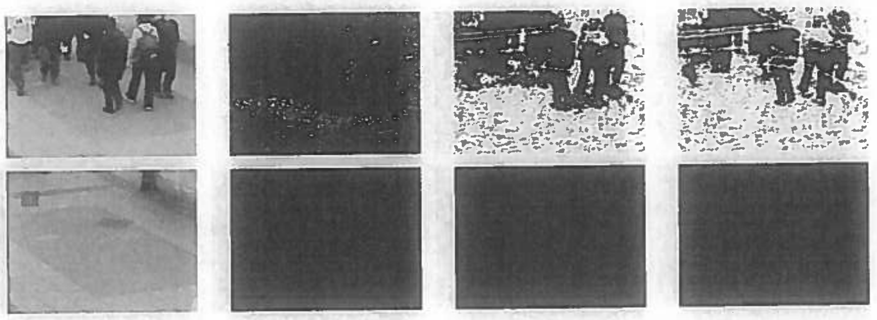

Figure 4.4: This depicts the tests carried out using the adaptive model over a synthetic sequence and raw images from the camera. Each process was tested with different λ thresholds, 0.5, 10, and 20

4.3 Results

The testing of the two statistical methods involved using two types of datasets. One set being raw footage from the datasets captured, as described in chapter 3. These real datasets were chosen to test the capability of the background estimation algorithms to cope with large dense flow. The other dataset being synthetic images that use background images captured, but have moving regions that are close to the background intensity values. The later datasets are used to detect the sensitivity of each background method.

Performance analysis was restricted, because of the use of the Matlab framework. This inhibited any analysis of the speed of the algorithms when it came to training. Even though coding was optimised for Matlab, dealing with large datasets and storage meant the speed of the algorithm was cancelled out. So instead offline preprocessing was done to speed up the testing of the algorithms to achieve robust analysis of background.

We present in this section examples of where the system may go wrong. We on purposely used dense crowd footage to see if the methods are capable of learning the background model within these sequences.

In fig(4.4) and (4.5) we see the image that should be detected over model of 100 previous sequenced images. Next to each image is the binary masked generated using the thresholds (λ) of 0.5, 10 and 20. The reason for changing the threshold was so that we could see, whether or not the size of variances from the median of the background to the median sample pixel is large (as described in section 4.1).

Both methods produced less than nominal results when compared to the actual images they were supposed to classify. With the synthetic images that contained $30 * 30$ simulated regions that were close to the background threshold both methods failed to detect these regions.

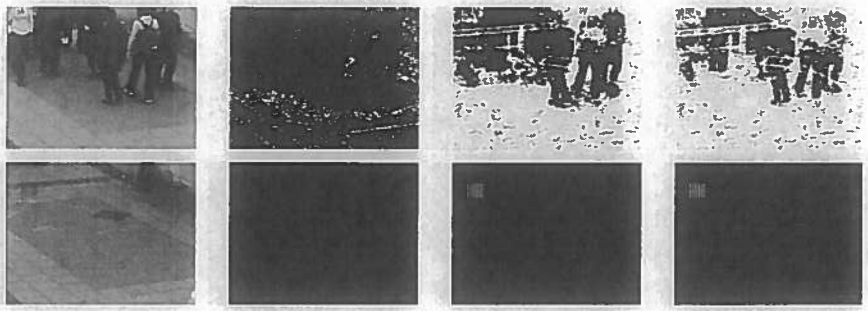
Figure 4.5: This depicts the tests carried out using the fixed background model over a synthetic sequence and raw images from the camera. Each process was tested with different λ thresholds, 0.5, 10, and 20

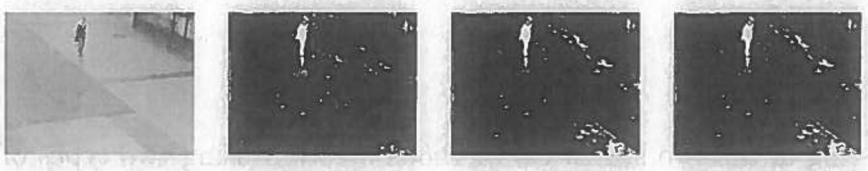
Figure 4.6: This shows the sample test results of a sparse dataset on the adaptive approach. Each process was tested with different λ thresholds, 0.5, 10, and 20

Both methods do not seem to be able to deal with dense crowd scenes over long periods of time. The use of 100 samples from a sequence was clearly not enough for the system to identify the background median in the adaptive method. There was expected improvement in the fixed model as the background median is already identified, however this was not the case.

Given that the adaptive approach did not achieve reliable results over a crowded image, we investigated its ability over a sparse image in fig(4.6). There was a modification of the results of this process, where foreground regions were identified as the white regions, unlike over the previous test set.

In the results for both methods there seems to be a lot of outliers identified. This is not all solely attributed to noise, however the assumption the adaptive model may have been able to adapt to the movement in the datasets, due to camera motion. As mentioned earlier in section 4.1 the use of the median absolute deviation which is more robust to outliers in a dataset than a method that uses averages, this means the systems susceptibility to change in the sequence is hampered and so will not adapt to changes in position the camera mat assume.

Analysis of the variance between the fixed model of a background pixel and the sample distribution of new pixels containing identified foreground regions is depicted in fig(4.7). The figure demonstrates a sample of pixel intensities that contain foreground pixels can lie either side of the background distribution. In the statistical model however

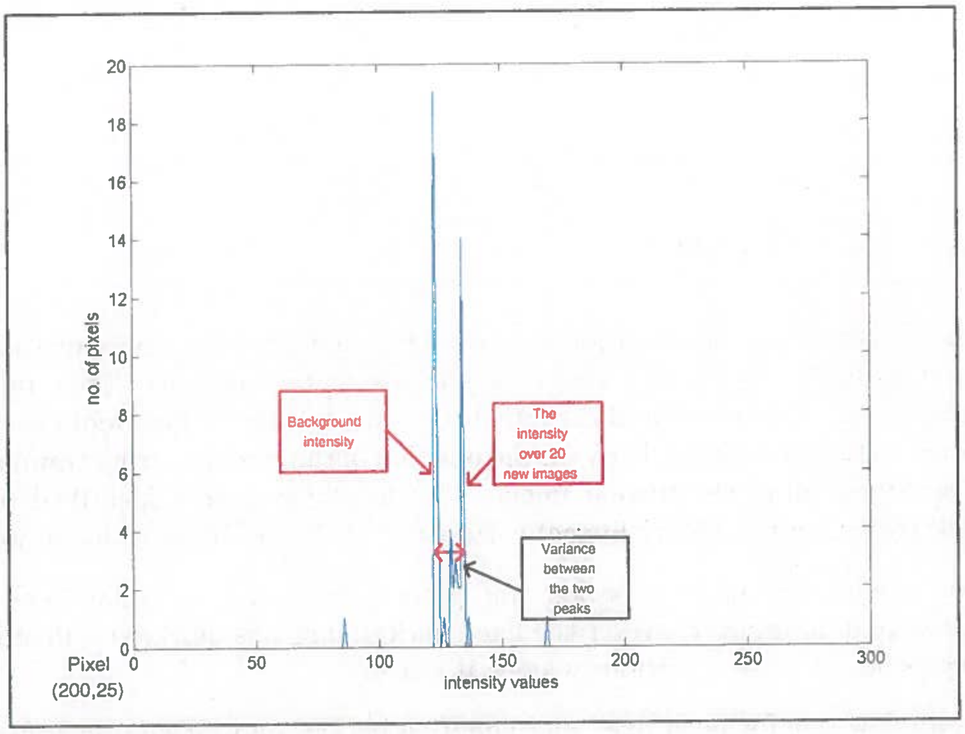

Figure 4.7: This shows two distributions on top of each other, one is a sample of new image pixels and the other is of the background pixel distribution, this is done to demonstrate the difference between the sizes of the pixel distributions.

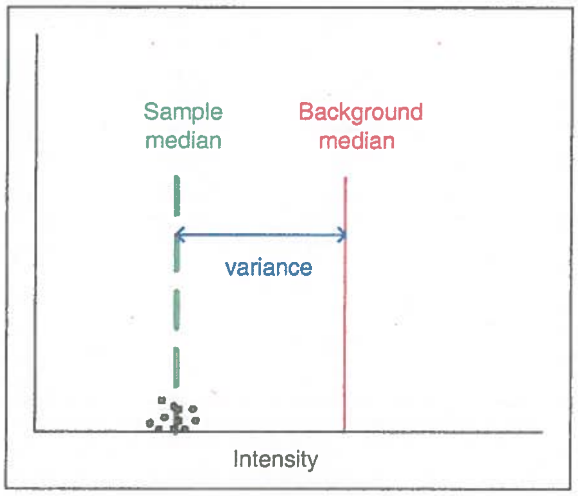

Figure 4.8: This shows how a fixed median is effects the variance of the data

4.4 Conclusion

Both statistical procedures produced similar results for the raw video images. This was mainly because of the dense crowds in the sequence, their range of motion during the sample and the latent movement in the camera footage. There appears to be a correlation between the position of the crowds during training and the classification of the current image. The large foreground identified regions (black regions in the binary image) correspond to the position of the crowds.

Both the adaptive and fixed background procedures produced similar results over a small sized datasets. However the fixed background was quicker to identify the background, because it obviously knew it already.

The window size for both fixed and adaptive background estimation techniques were not sufficient in size to retain information about the background intensity. Because the foreground objects i.e. the crowds, moved relatively slowly across the scene and the fact that large crowds covered pixels for more than 4 seconds meant that the system was unable to identify background intensities.

The conclusion from the fixed method results show that the assumption of fixing the background median to the data is flawed. The reason for this is the variance between the sampled image pixel intensities and the fixed background median. This relationship is depicted in fig(4.8). Because the background pixel is in a fixed position, the variance between itself and the median of the sample data can become large. As the large variance is used to normalise the difference between background pixel and the current pixel (in section 4.1), the value would become smaller the threshold ($\lambda = 0.5$) and so would be classified wrongly as a background pixel.

However the fixed median method attempted to detect the variance between the sample of pixel values from the window of 100 images. The conclusion from the results showed that the technique was not able to produce correct background estimation over a small sample. This was attributed to the fact the sample pixels could have been too far from the background median and the estimated variance generated from the Median absolute deviation normalised by 0.67285, would have been too large see fig(4.8). This would then effect the estimation of the current pixel, where it would have a greater probability in lying in the range of the estimated variance of the background.

The adaptive procedure for background estimation was also very limited in its ability to produce correct results over small datasets. The limitations for the procedure meant that the background had to be viewed frequently enough so that the correct median can be applied. However this is dependant on a large window size of images to detect the background value for a distribution of intensities for a particular pixel. Also the assumption relies on the window size being large enough to compensate for the dense crowds that will move across the scene. However if the sequence contains foreground pixels that occlude the background pixels for most of the sample in the window then the system will have problems identifying correctly the background. For this system to work correctly it would have to be able to anticipate the size of the window required to observe sufficient background pixels to model the background correctly.

Both methods require large window sizes for background estimation to work correctly. However these large sequences would certainly effect the performance of the framework with the need to store large amounts of data. The adaptive model would achieve correct results if the window size can accommodate for large crowd scenes that could last for a long time. The large window size also means that even though the system is an on-line adaptive method, building an accurate model would take time. The results of the statistical background estimation were not sufficient to estimate correctly the position of foreground pixels. Too much noise was left in the binary image of background and foreground pixels. It is clear that although these procedures of working with sparse sequences, in dense crowd applications they fail, due to the limit of the window size. This led to the development of a simpler and more adaptive approach to modeling the background, that would be able to deal (see chapter 5).

5. Simple background subtraction with flow

The previous methods in chapter 4 used intensity statistics to deal with background removal. This is generally the case inmost procedures and frameworks as presented in chapter 3. Alternatives to the single distribution idea as presented in chapter 4, is to use a K-Gaussian estimation model. However the problem with the previous method was the size of the number of images viewed to create the model of the pixel intensity distribution. We are also restricted by the Matlab framework, as we use large datasets and many more calculations the speed of training the system takes too much time, regardless of any optimisation. In this chapter present a simple background estimation procedure, which incorporates motion detection to help build its model.

First we will look at the background estimation model for the system. Much like the method presented in section 4.2 in chapter 4 a trained fixed model of the background median is needed to identify foreground pixels. We will see later how we can identify the model for the background.

Assuming that we have the median intensity of 30 frames of background images. We can then identify when a foreground pixel is currently in that pixel position by comparing the equivalent background median for that pixel. This is done by normalising the current pixel for a particular position by the background median. The equation for the normalisation of the current pixel is given by equation 5.1.

$$\gamma = \frac{x_{i,j}}{\bar{x}_{backi,j}} \tag{5.1}$$

γ provides the information of how proportionate the current pixel is to the background median. A decision boundary is then defined for the value of ρ.

$$\rho = \begin{cases} if\ 0.4 > \gamma < 1.3, & = 0 \\ else, & = 1 \end{cases} \tag{5.2}$$

The simplicity of the background estimation allows for fast processing of images to identify foreground regions. We are merely looking at the amount of difference or variance of the current pixel from the background, by comparing intensity values.

To identify background pixel values, motion estimation is incorporated into this part of the framework. As the frameworks motion estimation is not dependant

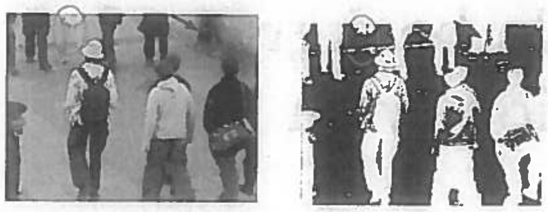

Figure 5.1: This shows the results on one dataset of crowded images. Notice that a person is missing from the image, and some background shadows appear.

on the identification of foreground pixels (fig(3.1)) by the background removal process we can treat both parts of the system as separate modules that deal with the images. This is important because it means that both modules (background removal and Optical flow) can be used together to help background estimation.

Building the background model is simply done by identifying when there is no or little motion in the scene (Further explanation can be seen in chapter 7). This information is provided by the optical flow part of the framework. Once an image has been identified as having no motion it is used as part of the background model.

This in many ways is much like the adaptive method shown in section 4.1 in chapter 4. Instead of the median changing relative to the sample, as the window moves through time to gain an accurate estimation, the simple method identifies this median value straight away, by using motion information. The procedure is also faster as it does not have to deal with large samples of data.

This adaptation of using motion estimates could have been applied to the fixed background model, where samples of data just consisted of background pixels. However it is computational faster to apply limits to the variance of a current pixel over a background pixel than directly calculate the variance for each pixel and threshold it that way.

5.1 Results

The background subtraction achieved results that had less noise present in the image, compared to the statistical methods presented in the previous chapters. In fig(5.1) we can see the results achieved by the background model that was estimated over 30 frames. The sequence of background images proceeded before foreground were identified. The system correctly identifies background images to train its background model with.

The background estimation method wrongly classified the shadows of the background objects in the scene. However the task for the framework is to monitor crowd movements and not to identify individual people. So the effects of detecting shadows should be negligible to identifying abnormal flow. Detailed discussion on the use of flow data for background estimation is given in chapter 7.

5.2 Conclusion

The results show significant improvement over the previous methods. Firstly we looked at the different datasets that the background estimation is applied over. There is still noise in the data, however there is a stark improvement from the statistical estimation in dense crowd scenes.

It was found that not all foreground regions corresponded to the thresholding applied to the background median. From fig(5.1) we can see there is a loss of foreground data. This due to the range of the threshold used being too large to identify very similar grey-scale values, of the foreground objects compared to the background. Possible improvements to the background estimation would be to apply M.A.D., of this background estimation method. We can then find the variance of the background model in the sample of 30 frames.

Because of the use of motion to train the background model problems could arise from motion being detected in sequences in the very beginning of that sequence. The framework would not be able to identify any foreground regions without a background model and so it would have to wait until the motion dissipated in the image. The adaptive model in comparison, in chapter 4, would be able to be applied over these images to build a model. But it would still take time to train that model to be accurate.

There is limitations however to the updating of the background model, which reduces its sensitivity to motion compared to the online adaptive method in section 4.1 in chapter 4. The assumption for the application of this method in a framework is that most cameras that are applied in surveillance situations should have negligible camera motion, so that the background will remain static.

The method improved on the other statistical techniques by performing faster[1] and accuracy of detection. However it must be mentioned that calculations are required to pre-process and identify when there is no motion, which takes time.

We have been looking at background estimation at the pixel level, and simply have been comparing the pixel to itself through time. In the next chapter we look at improving background estimation results by using morphological filtering.

[1]however we are not considering speed in this analysis of the framework

6. Morphological Filtering

Once we have achieved some estimation of whether a pixel is part of the background or foreground, we can then look at the image as a whole. The background detection algorithm produces a binary image version of a grey-scale intensity image. The hope is that the binary image [1], will divide the image up into its foreground and background components. For morphological filtering we do not look at the each pixel alone, but treat parts of the image as connected components in a plane i.e. large foreground regions that are not separated by background.

The binary image produced invariably contains noise. The noise usually consists of pixels that have been wrongly classified[2] as. These outliers in the image are usually very small and consist of a few pixels. The outliers should be negligible compared to the size of the objects that need to be tracked in an image i.e. the people in the crowd. And so the removal of these outliers would not effect the results of the background estimation too much.

The implementation of morphological filtering employed the use of in-built Matlab functions of erosion and dilation. We first identify the small unconnected regions in an image and then erode the outlying pixels from the image. Erosion simply is replacing foreground unconnected regions with background values. Parameters can be set to define the size of the removed image regions as well as the amount of times it is iterated over the image. For this framework a disk of radius 3 pixels was used to remove parts of the image.

For the dilations the same parameters were applied over the edges of foreground and background regions in the image. This was done because at the edges of foreground objects there can be a loss of information. Dilations help improve the accuracy of background estimation by expanding these foreground regions. The number of dilations and erosions done to an image is set to 3 each for this implementation.

6.1 Results

Morphologically filtering was applied to all of the background estimation algorithm outputs to improve their results. However it is unlikely to improve the results of dense flow images for the fixed and adaptive background estimators, because of the amount of noise present in these images.

[1] consists of values of 0's and 1's

[2] foreground(1) in the background(0) regions and vice versa for the foreground regions

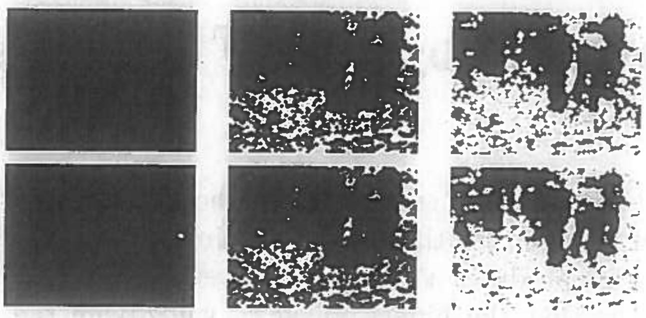

Figure 6.1: This is the morphological improvement of the background regions from the fixed and adaptive methods.

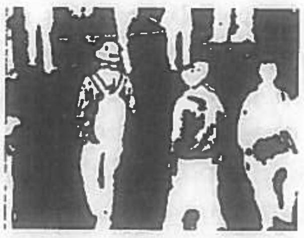

Figure 6.2: This is the results of the simple background subtraction method, after morphological filtering. The number of erosions did not remove much from the image.

In fig(6.1) we take a look at the morphological improvements to the binary image outputs from the statistical background methods.

In fig(6.2) we can see the use of morphological filtering over the binary image. However it failed to remove the shadows from the image and some of the outlying data. The erosion dilation sequence was not very long and therefore much of the original estimate remains. This led to the need of retaining the foreground information and we want from the crowd, but removing areas such as shadows of background objects identified in the scene. We took the step to integrate the morphological filtering with the optical flow estimation of motion. This would remove hopefully static objects from the scene i.e. the shadows of objects.

Fig (6.3) shows an image before filtering and after for the simple background subtraction method.

Using the simple background subtraction technique, we can see the improvement of the reduction of noise in the image. However it is identified that regions that are not part of the foreground have now become larger and more prominent in the binary data. As we are using background estimation to find foreground regions, this false classification can effect results in the classification procedure

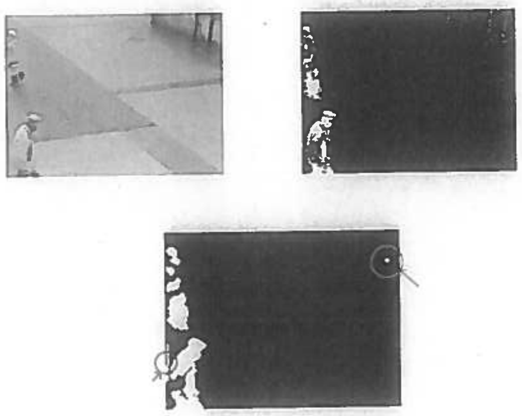

Figure 6.3: This contains the original frame with the background estimation and then the filtered version: Notice that some outliers are identified and dilated wrongly

(see chapter 8). A possible alternative to running the morphological filtering again over the image, is to use the motion estimation results for the images[3]. Again we are using the assumption that foreground pixels will be the only part of the image that is in motion the rest remains largely static.

[3]This can be seen in chapter 7

7. Optical flow

This chapter outlines the use of a optical flow algorithm, used to detect moving pixel regions over time. The software for optical flow calculations was provided by M.J. Black, from the Institute of Vision at Brown University [2]. Firstly we will describe how Black's software works and then how the software is integrated into the framework. And finally the chapter describes the use of optical flow detection in background estimation.

Let us first define the components of a vector of motion. This system produces a set of predictions for each pixel in an image. The predictions are in the form of a vector. Each vector is made up of two components, one value in the horizontal direction and the other in the vertical direction. These components are made up of the velocities that move in their corresponding directions. We will refer to these components as u and v [1]. From these components we can tell the direction and the magnitude of the velocity of a pixel. This is given by the equation in 7.1.

$$\theta = tan^{-1}\frac{|v|}{|u|}, \delta = \sqrt{u^2 + v^2} \tag{7.1}$$

7.1 Optical flow framework

In chapter 3 we gave a description of where the optical flow detection fits into the framework. Here we describe how the optical flow detection estimates the amount of motion in two successive images.

Optical flow is simply the measurement of motion in a scene. We look at a single pixel intensity in one intensity image I_n and try to calculate whether there is any movement of this pixel in a second intensity image I_{n+1}. Measurement of the motion is represented by vectors that show the speed and direction of each pixel. These vectors have two components that correspond to the horizontal and vertical velocities. The velocity is based on the proportion of pixel distances

[1]horizontal and vertical respectively

Figure 7.1: Angles in the framework are in radians

moved per frame. The frame rate for the datasets used in this framework are 25 frames per second.

The optical flow software used in this framework utilises robust statistics to detect flow in two images. The software takes two successive images in time and estimates the movement based on the pixel intensity translations. These estimates for translation are restricted one of the three main constraints involved in optical flow estimation. The constraint on the motion estimation are the intensity and spatial constraint. Further description of the constraints is discussed later.

The initial estimation of flow is based on the rate of change of intensity values through time. The rate of change of the intensity from one image to another is identified with the corresponding regions in a second image, so that an initial estimate of flow can be produced. However there are constraints that need to be verified before a robust estimation of flow can be found.

Earlier we described there were constraints to motion estimation. These are essentially a set of assumptions that allow us to model flow using intensity values, however these constraints can be broken. Robust statistical methods can identify when these constraints are broken and hence allow refinement of the model of flow[2]. The constraints are:

- **Spatial Constraint** Regions of pixels should have some spatial coherence to each other. This means that the same intensity values seen in one image should be around the same region in the next image, or slightly changed.
- **Intensity Constraint** Intensity values should not change abruptly. This assumption relies on the fact that most objects' grey-scale intensity values should stay around about the same, varying very little. But this is dependant on lighting conditions.
- **Temporal Constraint** Successive translations of pixels should not be too large, i.e. the frame-rate is sufficient to detect small motion in the images. The idea here is that the system is not expecting large jumps in motion of particular regions in the image.

In fig(7.2) we can see how the robust estimation framework is used. Using the two successive intensity images I_n and I_{n+1} and checking them whether any constraints are violated. Predictions are then made by the for those two images, when another set of intensity images are used the model is refined from the previous prediction and so can identify the motion vectors more accurately.

The system however does not treat each pixel in time by themselves, it actually divides the images up into regions. These regions a groups of intensity values

[2]from the initial estimation

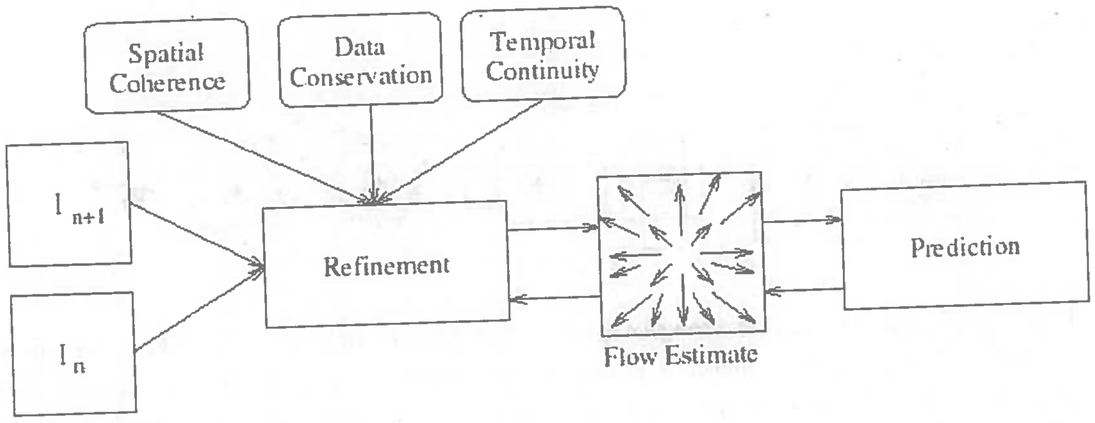

Figure 7.2: This is a flow diagram of the robust estimation framework used in Black's software.

that are similar to each other (see fig(7.3)). These regions parameters can be defined is explained in the next section.

7.2 Warping and Gaussian pyramids

One of the strategic components in Blacks[3] optical flow detection is the use of Gaussian pyramids. The pyramids are representations of intensity images reduced to different levels of resolution. The pyramid is depicted in fig(7.4), and shows the transition of an image going from coarse to fine levels of resolution. The top most level of this 4 level pyramid is the one with the lowest resolution. The reduction in resolution is implemented by a Gaussian distribution placed over the image intensities.

The Gaussian pyramid is used to detect motion boundaries, affine motion and remove noisy data from the estimates of motion. The procedure works by taking two successive intensity images (I_n and I_{n+1}) in a sequence, reducing the resolution of the initial image I_n and then using mathematical transforms to warp the initial image into the next successive frame I_{n+1}. The warping of the region is depicted in fig(7.3). Vectors of motion are calculated from this transform at each level in the pyramid. This allows Blacks software to robustly detect affine motion within a scene. The number of levels in the pyramid is specified by the parameters, but cannot reduce an image beyond its limits i.e. a reduction of an entire image to one pixel.

[3]This chapter does not attempt to explain the full nature of the software used to calculate optical flow, instead basic ideas and premises are expressed here. (for more detailed description

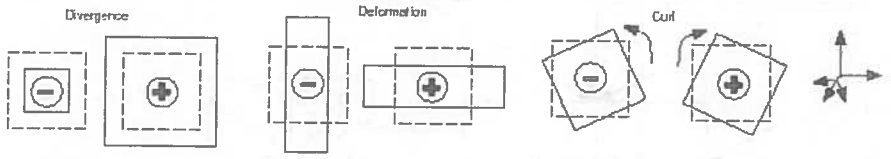

Figure 7.3: These regions are specified by the framework spatially. They are used to detect the correct motion if a region in another image is similar in size and position, but is transformed into another shape then modeling this transformation can help flow estimation

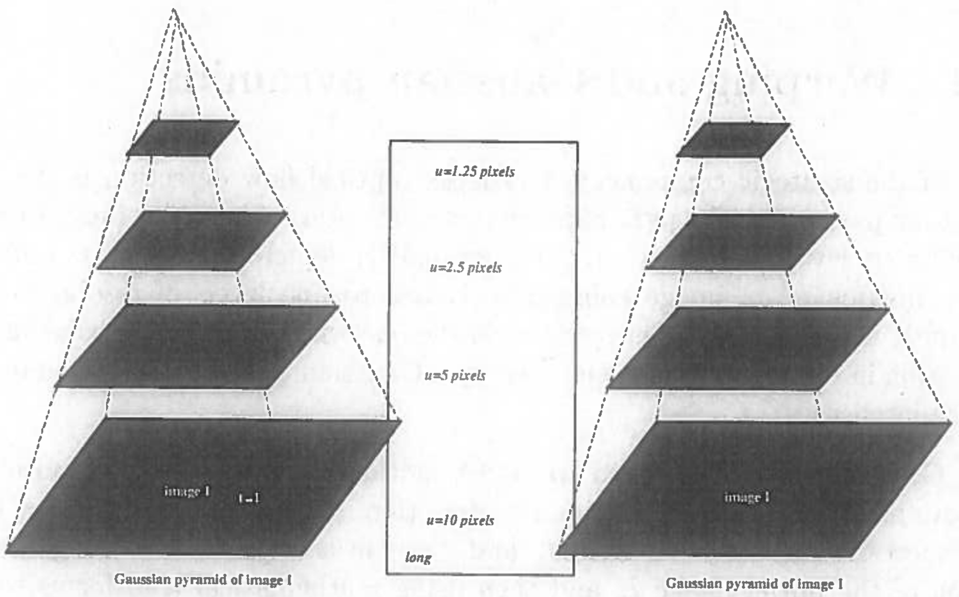

Figure 7.4: This show a 4 level Gaussian pyramid used over an image to calculate flow. The value u shows the projection of pixels through the levels of the pyramid. So u is 10 pixels at the bottom layer, when Gaussian reduction is applied over the 10 pixels 4 times it becomes 1.25 pixels.

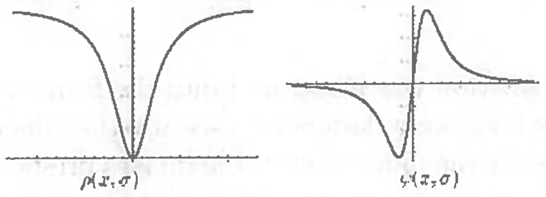

Figure 7.5: Both functions, Loretzian and over relaxation are used to remove outlying predictions of motion from motion boundaries

7.3 Implementation

This section outlines how the optical flow detection software was integrated into the framework. The algorithm is invoked as part of the framework and is passed a set of parameters. General parameters have been defined for the optical flow estimation software, which are general enough to be used on most images (see Black [2]). These parameters are defined here:

$$-l110.0 - l21.0 - S110.0 - S21.0 - s110.0 - s20.05$$

$$-nx320 - ny240 - f0.5 - F0 - w1.995 - iters20 - end.pgm - skip15'$$

The full description of each parameter set is given in Appendix B. The first set of parameters $-l1$ and $-l2$ set the limit of how many pixels need to be viewed for the spatial and intensity constraint respectively. The S parameters define the size of the region in which the error function depicted in fig(7.5), is viewed by the software to remove the outliers at the motion boundaries. The function is called the Loretzian function (left graph in fig(7.5)). These parameters along with the $-f$,$-F$ $-w$ and $iters$ define the amount of noise present in the motion estimate of an image. These values are optimal and are recommended values for the application of the over-relaxation function (defined in Black [2]). It was decided to keep these values fixed when calling Black's software in the framework.

One parameter that can be justifiably changed for experimentation purposes is the number of pyramid levels used to process the image. These should directly relate to the accuracy of the optical flow prediction [2]. However more transforms to levels in the pyramid will, it is assumed effect the performance and speed of application of the optical flow software in the framework.

Once the output of the components of each vector is depicted in fig(7.7). The figure shows the representation of the outputs as a mask consisting of the vertical

see [2]).

(v) and horizontal (u) components, which are utilised by the framework (see chapter8) .

The optical flow algorithm was integrated into the framework and tested on raw data images. These tests were designed to see whether the optical flow algorithm could identify correctly the movement in the images presented by this framework. In [2] the optical flow detection is tested thoroughly to see whether it can identify correct flow estimation regardless of when assumptions are broken.

7.4 Optical flow and background estimation

As described in chapter 4 the results from the background estimation section of the framework wrongly classified regions of the image as foreground pixels that should have been identified as background pixels. The background estimation seemed unable to identify background regions that consisted of shadows and essentially varied in intensity. As shown in figure (6.3), even after morphological filtering, outliers still remained in the image. As the framework was mainly interested in regions of the image that were moving, it was decided that the optical flow estimation could help in refining the background estimation. The assumption was that the background would largely remain static compared to the moving foreground, which the whole framework is interested in. Using the vector masks produced by the optical flow detection algorithm, regions of pixels with flow vectors that are small or close to zero were ignored by the framework, and the binary mask provided by the background estimation was updated.

In chapter 5 we described the application of optical flow analysis with background detection. As successive images are pulled into the framework optical flow analysis is used immediately to detect whether, there is motion in the scene. These images, which are classified in having minimal motion are used as the background model for the background subtraction.
This is done by finding the maximum magnitude of the vector in a image. Each pixel in an image has a corresponding vector to it. In fig(7.10) we can see the maximum magnitudes of a sequence of images. The first 10 frames of the sequence of magnitudes were identified consisting of just background images. This allowed the identification of an appropriate threshold for detecting the presence of motion. It was decided to put the threshold of the of the maximum magnitude at 0.1. The magnitude of the vector was calculated as

$$\delta_{i,j} = \sqrt{u_{i,j}^2 + v_{i,j}^2} \tag{7.2}$$

where $u_{i,j}$ and $v_{i,j}$ are the horizontal and vertical components of the vector at

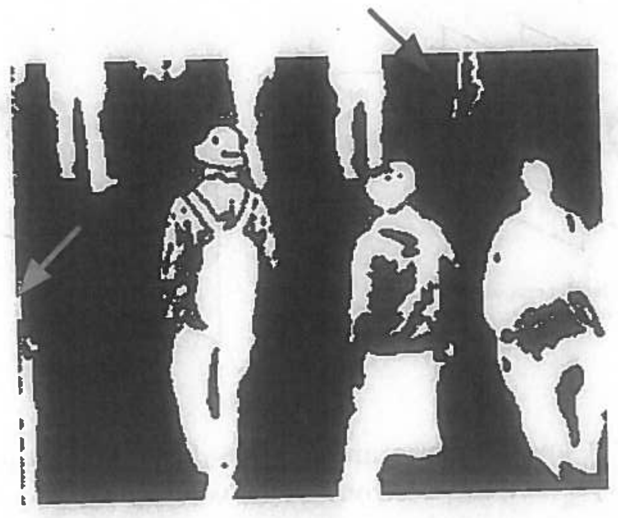

Figure 7.6: An example of the improved image with morphological filtering. However some foreground regions have been removed due to their lack of motion

position i, j in the image. Using this threshold the image is classified as either background or foreground.

However for reasons explained in section 7.5, the optical flow estimation does not always produce reliable results. The optical flow may identify images wrongly for background modeling. This was resolved by the simple background removal method by remembering what the previous maximum magnitude was. If the maximum magnitude of the previous motion estimate of the entire image was greater than the threshold, and the next estimate is zero, we ignore the frame as background. This is because there was motion in the previous image.

In fig(7.6) we see some improvement in the morphological process by removing shadows of background objects that are static. This was the original inspiration for the use of motion in filtering background images. However in [3], the use of motion to help in background estimation is used.

7.5 Evaluation of optical flow

The evaluation criterion for the optical flow software was to identify whether it could cope with the datasets that will be used in this framework. First we will discuss the frequency of motion detection in the sequences of images and then we will talk about the change in the number Gaussian pyramids used.

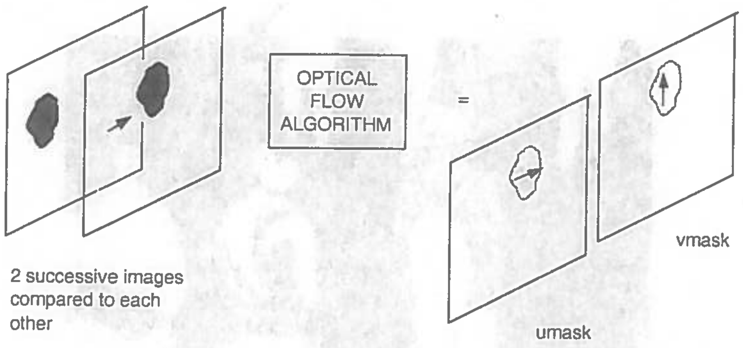

Figure 7.7: The u and v masks are generated from two successive images in memory. They represent the horizontal and vertical components of the vectors respectively.

The framework passes the set of parameters through a Matlab function called 'system'. This allows us to pass a string to the Linux kernel and then ultimately execute files. A list of Matlab functions used for this framework is given in the Appendix chapter C.

Breaks in the generated model of motion in an image sequence occurs periodically in the datasets used for this framework. You can see from fig(7.10) that depicts the mean of the magnitude in a sequence, that there is breaks in the motion detection[4]. This was problem for the classification model of the framework as it was dependant on the model of motion not to be broken (see chapter 8). But this resolved later in the classification chapter.

The optical flow algorithm was applied over the datasets seen in fig(7.9). Apart from the breaks in the motion estimates, the system accurately detected the synthetic sequences made. There was no loss of flow in the synthetic sequences where a region of 30∗30 pixels travels at one pixel per fame. So there maybe issues in the amount of motion present in the raw data sequences that were captured for this framework. Further explanation of the reasons is given in the next section.

The change of the Gaussian parameters produced very little change in the detection of maximum magnitudes and angles. This is because the use of Gaussian transform is best applied to finding affine motion, i.e. very small motion. When we look at the smaller motion we find bigger variances in the detection rate. This can be seen in fig(7.8).

When attempting to measure the magnitude of the vectors at each pixel very little change was noticed between the different number of pyramid levels. However the

[4]values set to zero

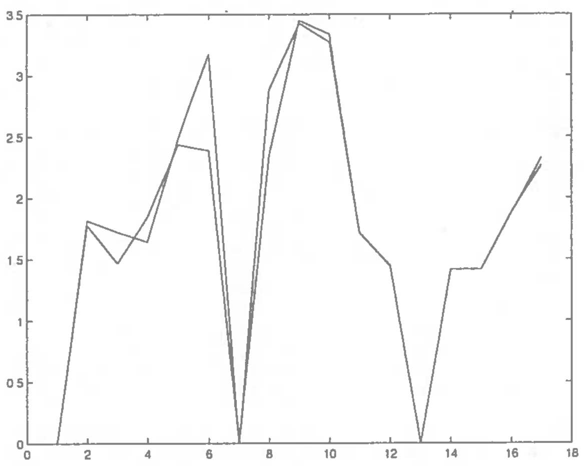

Figure 7.8: Shows the minimum magnitude over 18 frames with a change of pyramid levels 2 to 4.

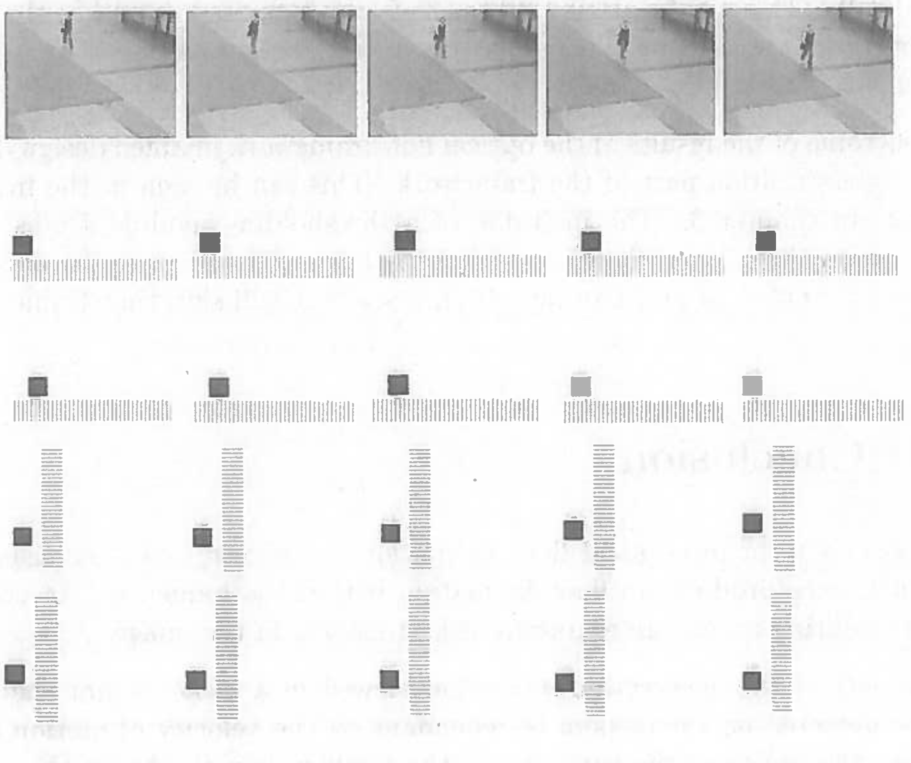

Figure 7.9: These are the datasets used to test the optical flow system. The first set is 60 images from crowd scene data. Four different data sets are tested. The synthetic images contain predicted motion the foreground regions travel at 1 pixel per frame.

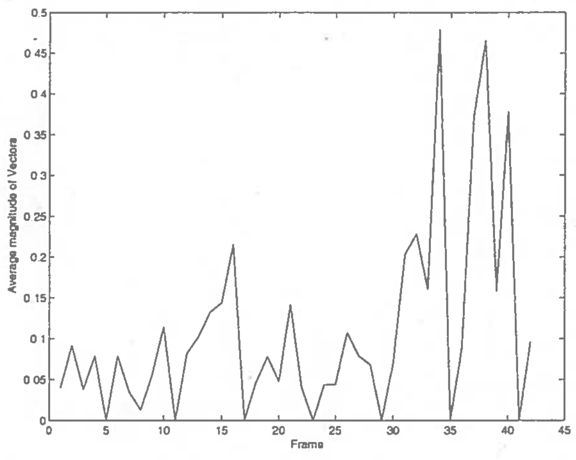

Figure 7.10: Shows the average magnitude of the vectors for pyramid of 4 layers. Notice that there is periodically no motion detected in the image. Motion in images does not behave this way, and is the effect of the software.

results do illustrate the system was unable to provide motion vectors in the image periodically. This maybe attributed to the frame-rate used as well as the amount of motion that is observed from that distance. Further explanation is given in section 7.6.

The outcome of the results of the optical flow framework justified design decisions for the classification part of the framework. This can be seen in the framework diagram in chapter 3. The inclusion of a thresholding module act as a buffer from results that do not fit the model of the data. Simply when the optical flow software is unable to find motion within a scene it will skip that frame from its model.

7.6 Conclusion

The sensitivity of the optical flow estimation has shown to be inconsistent for certain frames produce no flow estimation within the sequence. As no flow is detected during these frames due to lack of motion in the image.

One aspect of the observation of motion viewed in a scene is that the amount motion detected on the camera is dependant on the velocity of motion and the distance the are from the camera i.e. the further they the faster they have to move for motion to be detected in two consecutive frames.

The image taken by a digital camera is projected across a chip called a Charged Coupled Device (C.C.D.) (see fig(7.13)). If we look at the horizontal direction of flow we can estimate the amount of motion required for the system to be able

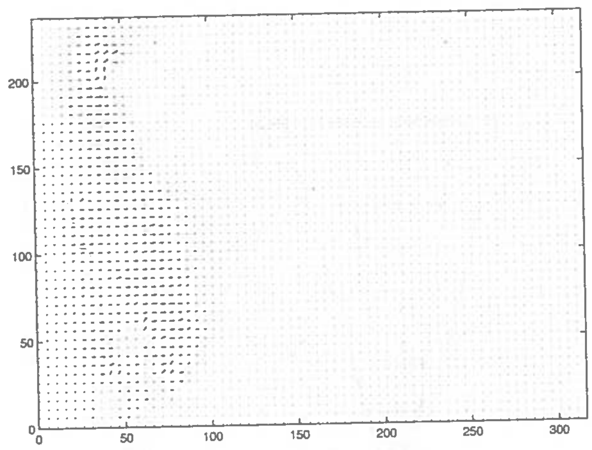

Figure 7.11: This is an example of sub-sampled vectors of an image from the algorithm. This shows every 5th pixel for the rows and columns.

to detect flow in each frame. One pixel in the horizontal direction of the chip is $0.20mm$ long ($582/114mm$). We take a velocity of one pixel per frame, where the frame rate is 25 frames per second (so $1/25_{th}$ of a second). To get the velocity required we do $v = \frac{distance}{time}$, and so the speed across the C.C.D. to see the required motion per frame is $0.05ms^{-1}$, this is accounting for magnification of the image ($*10$).

In fig(7.11) we show an example of the flow of vectors estimated in a scene. These vectors are sub-sampled, but gives an impression of the motion vectors in an image. If we divide up these vectors into their horizontal and vertical velocities these are the images/masks that are the output of this software.

In fig(7.12) we see the relationship between the distance from the camera and the magnitude of velocity in the horizontal direction. The average walking speed of a human is stated to be $1.2ms^{-1}$ (US M.U.T.C.D.). For motion to be detected across the C.C.D. chip from about 70 metres away [5] requires a speed of $3.5ms_{-1}$ to see motion in every frame.

This is possibly why the optical flow model breaks down given the amount of observed motion across a scene.

[5]the approximate distance from the camera to the observed scene in section 2.6 of chapter 2

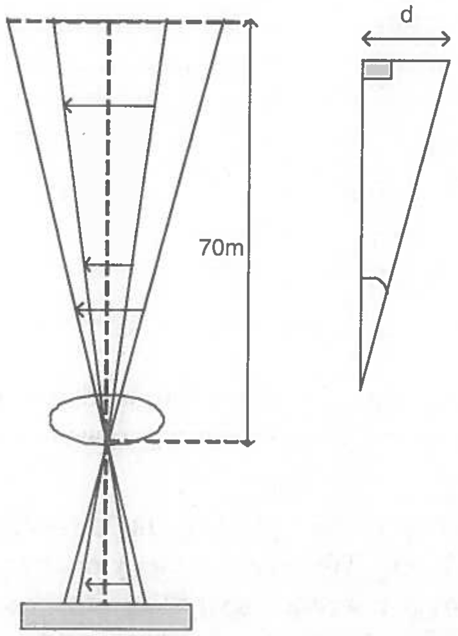

Figure 7.12: This depicts the relationship between the speed of movement seen by the camera and what is projected on the digital cameras cell. This show how the proportion of movement in reality relates to the amount of flow depicted in the pixels.

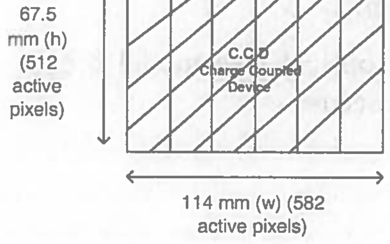
67.5 mm (h) (512 active pixels)
C.C.D Charge Coupled Device
114 mm (w) (582 active pixels)

Figure 7.13: The Charged Coupled Device is a photo sensitive chip that is used to digitise light onto film

8. Classification

From the previous chapters we have learnt that the background estimation (see chapter 4) technique, provides us with an image that has had an threshold applied to it, a binary image is then produced. The image defines where foreground and background regions lie in a frame. From the optical flow chapter (chapter 7) we receive two masks that make up the horizontal (u) and vertical (v) velocities of the vector at each pixel. In this chapter we will describe how the binary image is used in conjunction with the motion estimation results. In fig(8.1) we see how the classification module of the framework stores several frames of the motion estimation of each component of a vector in the image. The frames as well as the binary mask goes into the classification module of the framework.

What we are interested in, is looking at classifying regions of abnormal flow. The first task would be to define what is normal in the sequence of images. For this system we identify the normal motion of a sequence of images, by recording this motion over time. We can then find from this information the average motion in images through time. With this we can look at how the new motion in an image compares to the average. However the information we receive from the optical flow software is at pixel by pixel level. Computing all vectors of motion for every pixel in the image will be too computational expensive. In the next section we present a way of reducing this cost.

8.1 Region identification

We have to be able to sample the vector space of an image so that we can find the particular regions of motion we are interested in. Because the components of the vectors in the horizontal and vertical values are not independent of each other, we do not treat them independently of each other. Instead we calculate the magnitude and angles of each vector using these components, so we get an image of magnitude that $320 * 240$ elements and we do the same to the angles. These are calculated by the equation 7.1 in chapter 7. Examples of the angle and magnitude images are given in figures 8.2 and 8.3.

From the information of magnitudes in an image we can locate regions of motion in the currently analysed image. However there maybe noise in the data and it would desirable to remove this so we can locate the clusters of motion regions. The noise is removed from the magnitude image by projecting the binary mask[1], over the magnitude. Only the foreground regions are needed so we create a new

[1]Provided by the background subtraction method

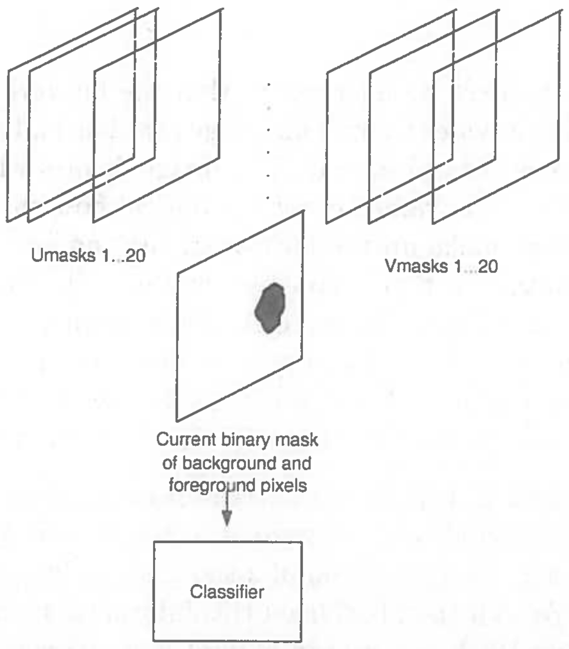

Figure 8.1: The framework takes several u and v masks of the components of the vectors through time and uses it with the current background estimate from the morphological filtering section of the framework. This information is used by the classifier to identify the current flow of the image and classify it

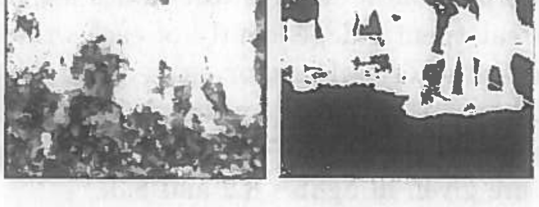

Figure 8.2: This shows the image magnitude intensities next to the actual binary image of foreground and background. The shades of grey shows the the sizes of the magnitudes of vectors in the image.

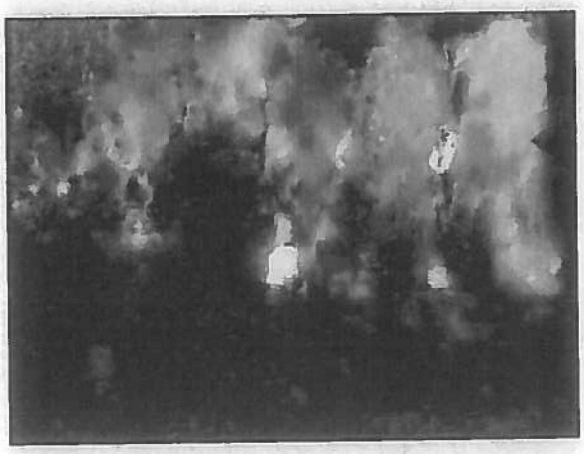

Figure 8.3: The image of angle intensities

```
Proc Kmeans( Wimage, K, max_iter)
        Vw = reshape(Wimage',1,length(Wimage(:))); % reshape 320*240 angle image into
                                                          % one dimensional vector
        r = randperm(Vw,2);                        % find a random point for the initial K-means
        m(:,1:K) = X(1,r(1:K));
        mode = m;
foreach iterations = 1:max_iter
        dist = sqdist(X, m);
        [small]=min(dist',2); % this finds the closest points to the current mean
        foreach n = 1:K
                if length(find(small==k)) > 0
                        m(1,K) = mean(find(small==k))
                endif
        endfor
        if mean(sum(m-mold).^2) < 0.001; break; end mold = m;
endfor

[centres, cov] = LocateCentres(small,K); % locates the position of the centres in the image
```

Figure 8.4: K-means clustering description

image of the magnitude by setting the parts of the magnitude image to zero, if they are not part of the foreground. With this restricted image of the magnitude we can locate the different regions clustered motion.

A good way of detecting clusters of data in an image is to use the K-means algorithm. This is a simple algorithm that locates the central means of data clusters. It is a proven technique in image processing to finding correlations in data[?]. The K-means algorithm is defined in fig(8.4). In fig(8.5) we show the above process diagrammatically.

The K-means algorithm detects clusters of data points and finds the maximum of the clusters in the data. Depending on the value of K we can define the number of means to find within the image. The K-means algorithm works by finding clusters of data of similar variance and iterates until finds the global mean for that cluster. The algorithm initially samples points in the data at random and then tries to reduce the error from the points to the nearest mean, the error is defined in equation 8.1.

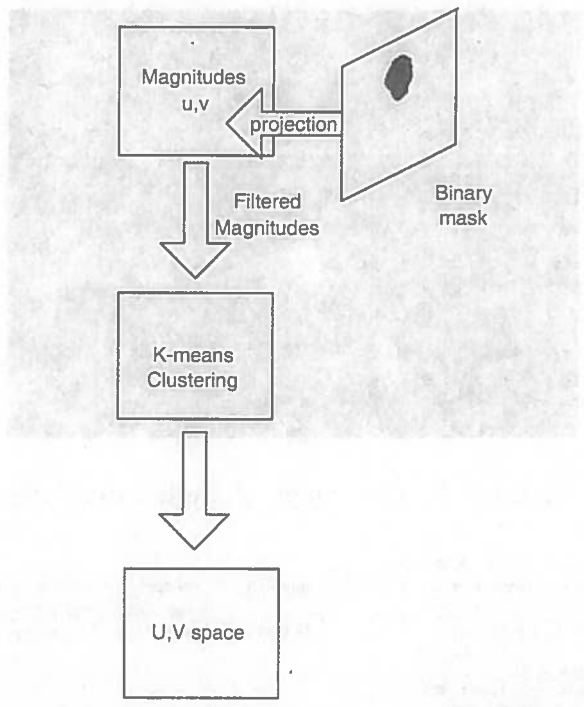

Figure 8.5: This shows how the masks are projected over each other to produce the information about direction

$$Error_c = \sum_{i=1}^{n_c} d^2 \tag{8.1}$$

Where d^2 is the Euclidean squared distance from point i to a mean of a cluster. In equation 8.1 we see how the error is reduced by the K-means algorithm to iterate closer to the mean clusters in the image. We iterate 20 times for each image of the magnitudes.

We also calculate the within sum squares error, which essentially means we sum all the error for every K to get some global value, this is seen in equation:

$$E = \sum_{i}^{K} Error_c \tag{8.2}$$

$Error_c$ being the error from one particular cluster. For the framework we use 5 for the value of K. The K-means algorithm produces coordinates for the each cluster in the magnitude image. This allows us to locate regions of large motion in the image. The K-means algorithm provides us with these locations and so we then sample from this data information about its motion.

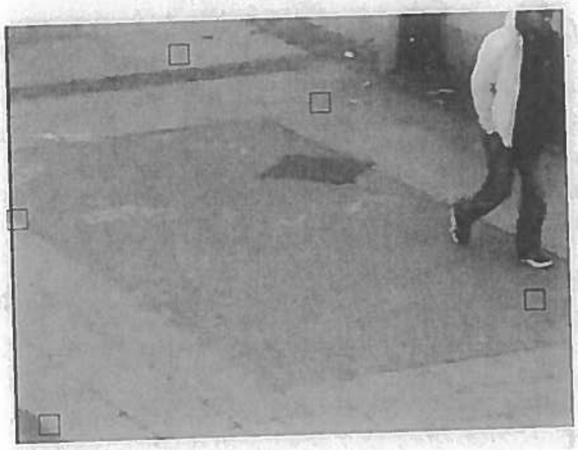

Figure 8.6: Bounded boxes in initialised K-means position 20 * 20

8.2 Pixel by pixel classification

Once we have identified the locations of the magnitudes in the image we can then sample the regions from the motion vector information. The framework uses the position from the K-means algorithm to locate regions of motion. We then apply bounding boxes around these points (see fig(8.6)). The boxes are of size 20 * 20

In these boxes where motion is occurring we look at the average motion of a pixel through time. This is done by averaging the motion information from the optical flow software. So we take an entire set of vector components in the horizontal (u) and vertical (v) positions. The comparison used for the framework was Mahalabonis distance. The equation of which is given in 8.5.

$$\Omega_{uv} = \frac{1}{n-1}\sum_{p=1}^{n}(u_p - \bar{u})(v_p - \bar{v}) \tag{8.3}$$

$$\alpha = \begin{pmatrix} u \\ v \end{pmatrix} \tag{8.4}$$

$$\Lambda = (\alpha - \bar{\alpha})^{\tau}\Omega_{i,j}(\alpha - \bar{\alpha}) \tag{8.5}$$

Where $\bar{u}$ and $\bar{v}$ are the mean of u and v The value Ω_{uv} shows one entry of the covariance matrix for the u and v components. A description of the applied algorithm is seen in fig(8.7).

This gives us the distance in a two-dimensions in which both u and v reside in the image. We use a threshold over Λ to identify whether a pixel is part of the

```
Proc Classification( BWImage,U,V)
        new   a = [];
        global U;
        global V;
foreach Bwimage =1
        store(U,V)
        alpha = ([U - mean(U)] , [V - mean(V)]);

        lambda = alpha'*Covariance(U,V)*alpha % mahalabonis distance
        if lambda > 6
                a(i,j) = 1;
        else
                a(i,j) = 0;
        endif
endfor
```

Figure 8.7: Pixel by Pixel Classification Description

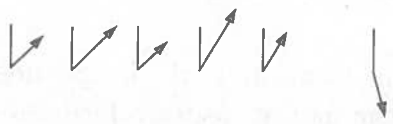

Figure 8.8: This shows the observed pixel classified on the basis of past observations.

the distribution of the previous vectors. A diagrammatic representation of this is given by fig(8.11). As u and v represent velocities for a pixel there position can be plotted in a 2 dimensional space. Using Mahalabonis distance we treat the distribution over the image as a probability distribution function (p.d.f). in fig(8.8) we see how a distribution of vectors is compared to an image.

Choosing an appropriate threshold for the classification of Mahalabonis distance led to the analysis of these values in an image. In Shilpa [5], the value for the threshold is set to 6. However there was no justification for this value, this to an investigation of an appropriate threshold. We took 50 images of the u and v components and calculated the Mahalabonis distance for the mean, maximum and minimum u,v values. The data is presented in fig(8.9), and depicts the Λ values over a single pixel. It was discovered that the behaviour of changes over different pixel position as can be seen from the mean in fig(C.4). From a sample of selected distributions for a 50 images, the most acceptable value for the threshold was 5.0. As seen in the next section, the classification of the sample tests sets gave favourable results for the size of the threshold.

Once we know the whether a pixel is moving in the same direction as previous other pixels have done. The pixels that we process all are within the boundaries of the bounding boxes. So we process a region of $20 * 20$ pixels and find, which

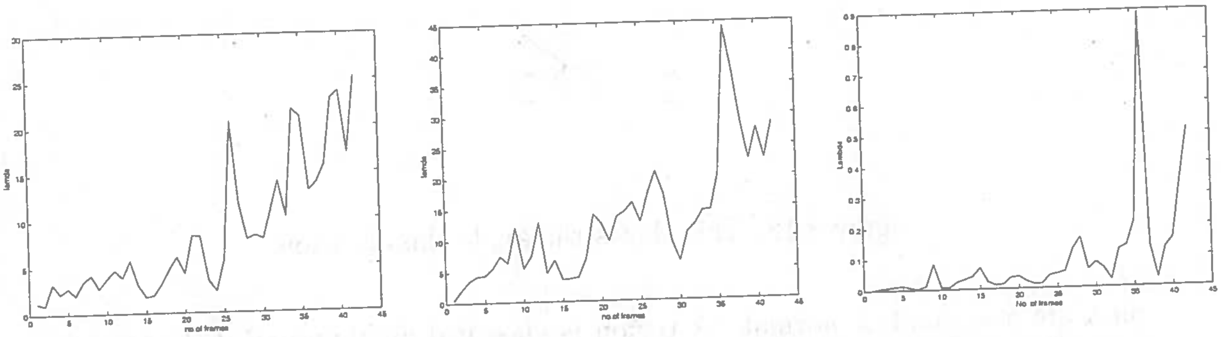

Figure 8.9: This shows the min, max and mean *Lamda* values through a sequence of 52 frame, removing the 0 velocity motion results. They are all over the same pixel.

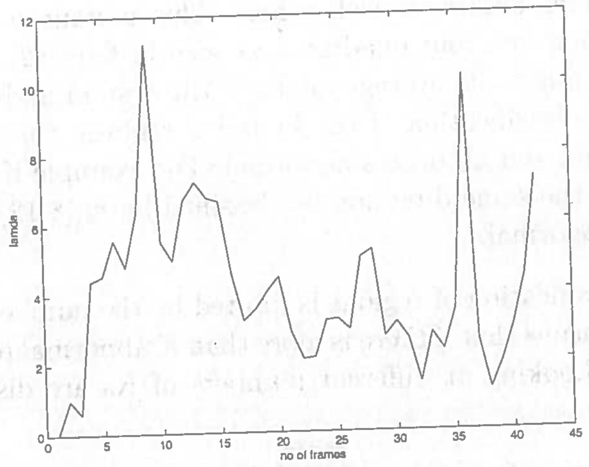

Figure 8.10: A pixel of known occlusion by another pixel in the other direction. This helps define the threshold for *Lambda*

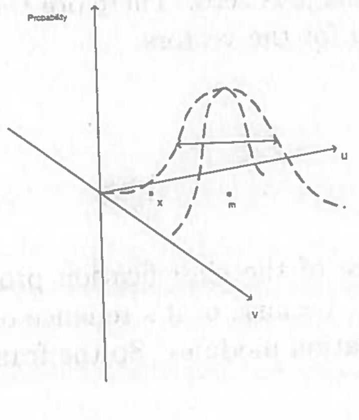

Figure 8.11: Depicting the distribution of the velocity of vectors in a 2 dimensional plane.

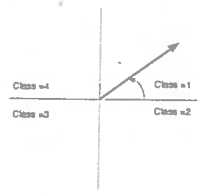

Figure 8.12: This shows the angle classification

ones are abnormal or normal. A region is classified as abnormal if it contains more than 50% abnormal pixels to normal. These boxes are labelled red for abnormal and green for normal. For each box we can display statistics of average angle and velocity for each bounded box. We can further add robustness to the classification by sampling the average angle from each bounded region and compare the regions angles to each other. This is done by dividing the angle space of each region into four quadrants as seen in fig(8.12). Each bounded box is classified according to its average angle. If the system finds one direction more prominent in the classification of the bounded regions, then it will classify this direction as normal and all others abnormal. For example if 3 out of 5 bounded boxes were going the same direction as classified by fig(8.12), the other two will be classified as abnormal.

Although the classification of regions is limited by the number K in the K-means clustering. This means that if there is more than K abnormal regions, the classifier will miss these. Looking at different numbers of Ks are discussed in the next section.

Some issues had arisen from the analysis of the flow estimation in chapter 7. The flow estimation algorithm failed periodically to detect flow and so the motion estimates through time would be broken if we kept these values as part of the model. This is resolved by simply ignoring any frames of vectors whose maximum magnitude for the entire image is zero. Therefore these frames are not added to the estimation of the mean for the vectors.

8.3 Results

Evaluating the performance of the classification procedure by itself cannot be treated as separate module, because of it's reliance on the output of the optical flow and background estimation modules. So the framework had to be tested in its entirety.

However because of the restrictions of performance in the Matlab framework, building the classification model for the pixel by pixel method had to be done offline. As we are not directly measuring the speed of performance of the algorithms this is decided to be a tolerable restriction. In this evaluation we want to see how well the classification method can identify abnormal events when monitoring crowds.

First we test the optimal number of iterations over a single image to locate the clusters accurately. The number of K was fixed at 5

Iterations	max $\bar{\omega}$	max sqdist Error	min sqdist error
20	1.5708	1.405	0.0587
40	1.3299	1.3507	0.0521
60	1.3340	1.3507	0.1552
80	1.5552	1.3507	0.0482
100	1.3340	1.3507	0.0534

Testing for the most optimal number iterations were performed over the training dataset. Each iteration set i.e. 40, 60,80, was examined five times and the average of the results were taken. This was done, because of the random sampling of the data, so an average of the test sets was needed to normalise all the data. It was discovered the most optimal number of iterations was 40 given lowest squared error distance from the mean, on average.

To measure the classification over each pixel we created a synthetic dataset. We took one background image from an actual dataset and then applied a moving pixel regions via a image editing software (see Appendix C). The pixel regions are shown in fig(8.13), they consisted of of a region of the size $40 * 40$ pixels, more than one object was present. Motion was simulated in the scenes by moving the regions across the image at one pixel per frame. With these datasets we train the classifier, with 30 frames of these moving regions. We then create another set of images of pixels that move in abnormal directions. From this we know how many pixels should be identified as abnormal.

Tests are carried over these datasets using different values for number Ks. We then identify the error in the amount of background identified in each bounded region.

The tests chosen for the framework were based upon the percentage of background pixels to foreground in the opposition. We then look at the number of bounded regions in the image that are classified as abnormal and normal.

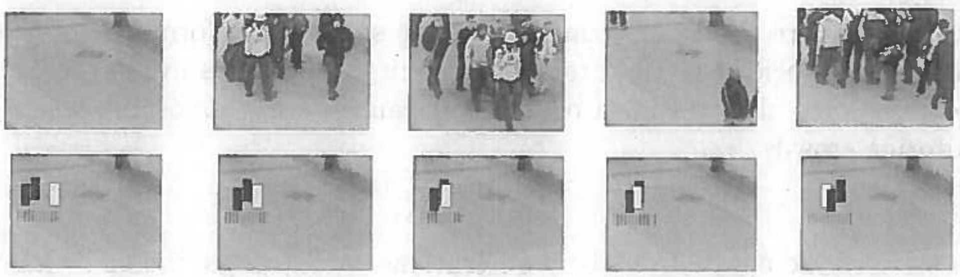
Figure 8.13: This shows the dense crowd dataset compared to the synthetic datasets.

K	Error in K bounded regions %	No.of abnormal	No.of normal
1	5	0	1
2	10	0	2
3	13	1	2
4	23	1	3
5	20	2	3
6	20	3	3
7	28	3	4
8	33	3	5
9	37	3	6
10	49	3	7

Reasons why we get errors might be due to the fact the system uses clustering on the motion, and so the centres of the motion magnitudes does not correspond to the actual place where the object lies in the image. And so we identify background in the bounded region. This gets proportionately higher when K is increased in number. The total number of abnormal and normal regions in the synthetic datasets is 3 each. The system identifies the correct number of abnormal regions, however as K is increased the number of normal regions are identified. As the number of false classifications increases so does the percentage of the error for all K bounded regions.
From these results, the importance of identifying the correct amount of K for an image is exemplified. We discuss possibilities of improvement to this in the next section.

In fig(8.13) we have an example of dense crowd regions over

8.4 Conclusion

Testing of the classification module validated indirectly the entire framework. Concluding from the results we found out that it would desirable to identify the

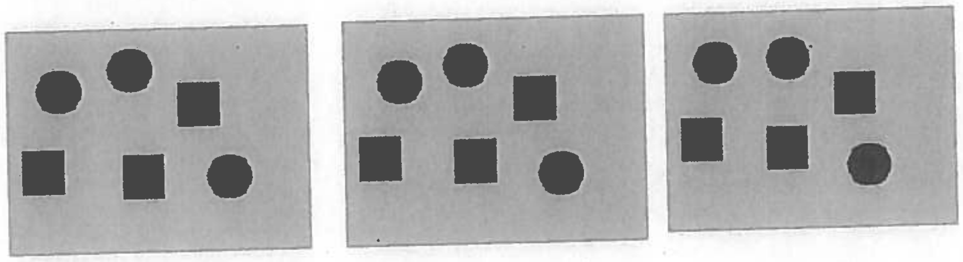
Figure 8.14: The datasets used for testing the accuracy of detection

number of appropriate K values for the framework to classify objects correctly. A possible way of identifying an appropriate number, is to detect separated foreground regions. But as the framework deals with dense motion, where occlusion of foreground objects is regular, this might not be possible. Having too many clusters analysed that are not part of the motion we are interested causes a problem for identifying abnormal regions.

Another avenue of improvement for the system is to have some form of adaptive approach to identifying a value for the threshold for Λ. This being the classification value for identifying abnormal or normal motion, by applying Mahalabonis distance.

9. Conclusion

We have discussed varying implementations for creating a robust framework for an automated surveillance system. To recap, we have looked at:

- Adaptive window thresholding and discovered that the method when applied with a model of 100 frames is not sufficient to build a good enough representation of the background model for a single pixel
- Things were not improved with the use of a fixed background method.
- An alternative method was used by applying motion estimation to identify when there static background scenes the simple background subtraction method could identify images that were a clean model for the background intensity.
- For the optical flow framework we discovered that the system cannot always identify motion correctly due to the framerate and the rate of change of objects at a certain distance. We also found ways of improving the the morphological filtered binary image with the optical flow framework.
- The motion also helped improve background estimation results.
- for the classification procedure we found a way of identifying abnormal regions by first identifying the greatest regions of motion (via K-means) and then identifying each pixels orientation relative to its past motion.

However there were problems that were identified during the development of this framework. One the most prominent was the implementation of the framework in Matlab. Matlab was identified as an ideal environment for the development of image processing framework. However when it came to running and analysing results the environment was not very reliable when dealing with large datasets. Processing time was very large, even when the code was optimised for speed.

Future development of the framework is given in the final section of this paper.

9.1 Future work

There are many avenues for development of the framework in the future. Namely in the area of classification of the anomalous flow. Greater application of learning techniques such as neural networks could be applied to identify more complex dynamics of the motion of objects. Rather than simply classify whether an abnormal event is happening in a scene.

One possible solution to learning is the application is to apply basis functions over the data. The technique developed in this paper used the image of angles of each vector at time t. Using an adaptive basis model we could go further through time by compressing 100 or so images, by applying Principle Component Analysis P.C.A over a number successive frames to identify an underling pattern within the range of images. For instance if there is constant motion across a pathway we can detect it by looking at the P.C.A'd image over time for angle and magnitudes vectors. Due to processing limitations of Matlab this idea was not developed for this framework, because Principle Component Analysis over a large datatset of images takes too long to process.

The idea to the authors knowledge is original as most applications of basis classification (P.C.A) are applied to vectors not angles and has been used to parameterise classification over speech and walking gate recognition (see Yacoob et al [1]).

It would be also desirable for a on-line adaptive framework to be developed and not one that requires offline processing to achieve fast implementation.

Bibliography

[1] Yacoob M. Jepson A. Paramerterised modeling and recognition of activities. *Computer Vision and Image Understanding*, 73(18):232–250, Febuary 1999.

[2] J. M. Black and P. Anandan. The robust estimation of multiple motions: Piecewise-smooth flow fields. *Computer Vision and Image Understanding*, 63(1):75–104, 1996.

[3] F. Dufaux and F. Moscheni. Background mosaicking for low bit rate video coding, 1996.

[4] Stauffer C. Grimson E.W. Learning patterns of activity using real-time tracking. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 22(8):747–757, 2000.

[5] S Gnaneshwar. Dense anomalous flow in crowd scenes, 2004. Msc Dissertation.

[6] G. Gordon, T. Darrell, M. Harville, and J. Woodfill. Background estimation and removal based on range and color. pages 459–464.

[7] Haritaoglu I., Harwood D., and Davis L.S. W4: Real-time surveillance of people and their activities. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):809–821, 2000.

[8] Dar-Shyang Lee Jonathan. *A Bayesian Framework For Gaussian Mixture Background Modeling.*

[9] Elgammal A. Harwood D. Davis L. Non-parametric model for background subtraction. *ECCV*, 1(1):192–198, 2003.

[10] Kang S. Paik J. Abidi B.R. Shelton D. Mitckes M. Abidi M.A. Gate-to-gate automated video tracking and location. *The FAA's 3rd International Aviation and Security Technology Symposium*, 1(1):139–144, 2001.

[11] O Masoud. Tracking and analysis of articulated motion with an apllication to human motion. 2000.

[12] B. Maurin and O. Papanikolopoulos N.P. Masoud. Monitoring crowded traffic scenes. *IEEE 5th International Conference on Intelligent Transportation Systems*, 1(1):19–24, 2002.

[13] Maurin B. Masoud O. Papanikolopoulos N. Camera surveillance of crowded traffic scenes. 2002.

[14] Naoya Ohta. Optical flow detection using a general noise model for gradient constraint. In *Computer Analysis of Images and Patterns*, pages 669–676, 1997.

[15] Reisman P., Mano O., Avidan S., and Shashua A. Monitoring crowded traffic scenes. *IEEE Intelligent Vehicles Symposium*, 1(1):66–71, 2004.

[16] Kawamoto K. Yamada D. Imiya A. Klette R. Navigation using optical flow fields: An application of dominant plane detection. 2002.

[17] C. Ridder, O. Munkelt, and H. Kirchner. Adaptive background estimation and foreground detection using kalman filtering, 1995.

[18] Siebel N. Maybank S. The advisor visual surveillance system. 2003.

[19] Siebel N.T. Maybank S. Fusion of multiple tracking algorithms for robust people tracking. 2002.

Appendix A. Display

This chapter explains the work done in the system to display the data and classification of images that is produced by the framework.

The framework was developed on Matlab 6 Release 13. The implementation of the simple interface for the framework was based on GUIDE the Matlab development environment for displaying and handling figures.

The following figures present the display and the various views of the images being processed. Using Matlab event handler we could display the figures of the Matlab functions in the GUI.

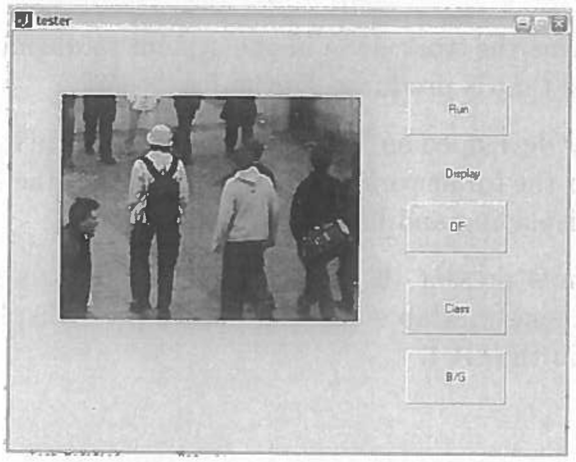

Figure A.1: The view of just the image

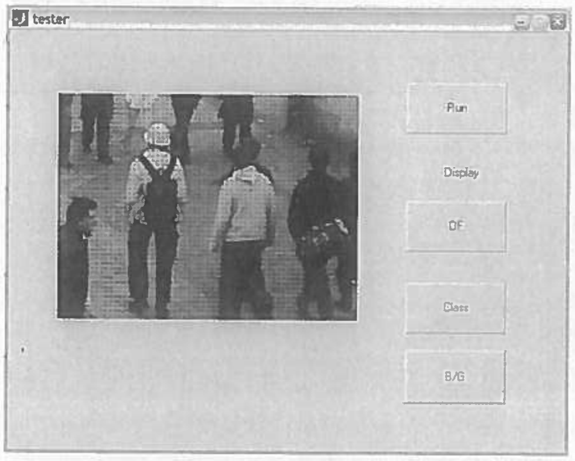

Figure A.2: The flow overlaid on top of the image

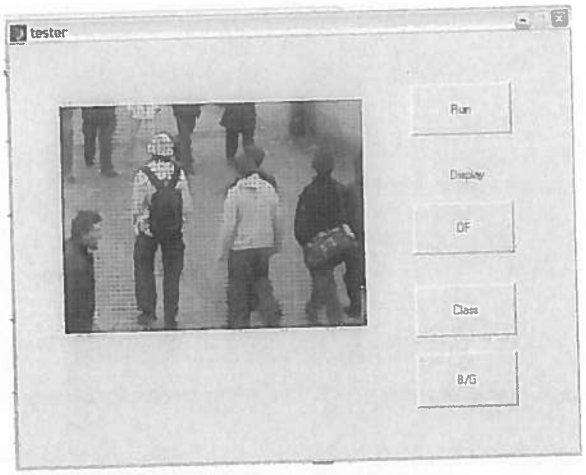

Figure A.3: Classification regions identified, these are just plots of the K-means identified points in the image, they do not represent the bound regions as described in chapter 8

Appendix B. Parameters

Most of the parameters that were provided A list of the parameters passed to Blacks [2] algorithm are described here.

- first-frame and second-frame are integer frame numbers (assuming that your image sequences is numbered).
- max and min pyramid really could be a single number (max - min) which says how many layers you want in the pyramid. So for a three level pyramid, you would have max = 4, min = 2, for example.
- -l1, -l2, -l3: These weight the data, spatial, and temporal terms respectively. Float. Usually l1=l2=l3=1.0.
- -S1, -S2, -S3: These are the INITIAL scale parameters for the robust estimator (for the data, spatial, and temporal terms respectively). These should be chosen large enough to ensure that the objective function is convex.
- -s1, -s2, -s3: These are the FINAL scale parameters for the robust estimator (for the data, spatial, and temporal terms respectively). These should be chosen to reflect what is considered an outlier.
- -f: A float that indicates the rate that S2, S2, and S3 are decreased each frame. So S1 at time t is S1 * "f" at t-1.
- -nx, -ny: These are the x and y dimensions of the images (integer).
- -F: If 1, then the images are prefiltered with a Laplacian, otherwise they are not.
- -w: The "overrelaxation" parameter ($0 < w < 2$). Typically $w = 1.995$.
- -iters: The number of iterations of relaxation per frame. Usually between 1 and 20 for most sequences. If you have just a two image sequence this will probably be much higher.
- -end: If your input files don't look like <in-path> <frame> but have a tail, then this is the value of end. So, if end is specified the input path is <in-path> <frame> <end>.

Appendix C. Software

The original input of the datasets were in Windows Media Video, which was streamed from The Panasonic NV-DX110 camera. The software used to convert the datasets into files to be used in the framework, namely the Portable grey-map (PGM) format used by the optical flow estimation algorithm.

- Windows movie centre editor was used to stream the file into a digital format (WMV)
- Zwei-Stein is used to divide the media video file into Bitmap .bmp images
- To convert several 100 frames of .bmp images into Portable grey-map format Batch it! Ultra was used for windows.
- The Matlab framework provides with image processing library was used for the development of this framework.

Here is an example of the Matlab environment.

Matlab functions used.

Matlab function :	Definition
bwlabel	Label connected regions of binary images
imread	Read in image
imwrite	write image
cov	calculate covariance matrix
repmat	create an representation matrix
reshape	reshape a matrix dimensions
mean	mean of a vector
max	maximium of a vector
min	min of a vector
var	Calculate variance
system	send commands to operating system

```
Using Toolbox Path Cache.  Type "help toolbox_path_cache" for more info.

To get started, select "MATLAB Help" from the Help menu.

>>
```

```
All Files           File Type      Last Modified
helloworld.c        C Source file  25-Oct-2004 10:12 AM
helloworld.h        C Header file  25-Oct-2004 10:12 AM
helloworld.m        M-file         25-Oct-2004 10:40 AM
helloworld_main.c   C Source file  25-Oct-2004 10:12 AM
[illegible].csv     CSV File       13-Jun-2005 04:13 PM
```

Figure C.1: Picture of the Matlab environment used for the development of the framework

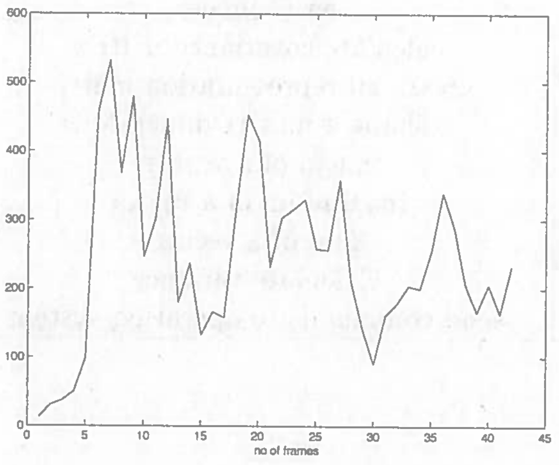

Figure C.2: This displays the mean vector and finds the Mahalabonis distance from the maximum in one image.

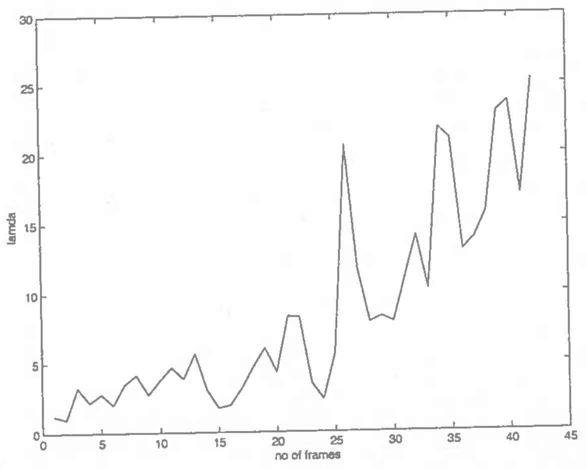

Figure C.3: This displays the minimum vector and finds the Mahalabonis distance from the maximum in one image.

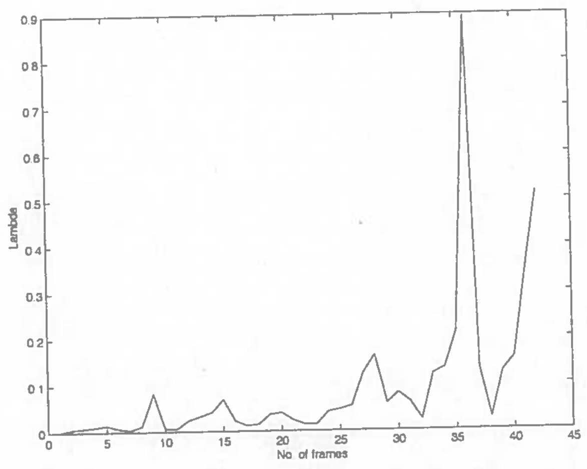

Figure C.4: This displays the maximum vector and finds the Mahalabonis distance from the maximum in one image.