DUBLIN CITY UNIVERSITY
SCHOOL OF ELECTRONIC ENGINEERING

Speeding Up Adaboost Object Detection With Motion Segmentation and Haar Feature Acceleration

A thesis submitted for the award of

M.Eng.

in Electronic Engineering at
Dublin City University

by
Radha Krishna. R

Supervised by
Prof. Noel O'Connor

August 19, 2009

DECLARATION

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of M.Eng. in Electronic Engineering is entirely my own work, that I have exercised reasonable care to ensure that the work is original, and does not to the best of my knowledge breach any law of copyright, and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

Signed _____

Radha Krishna. R ID.No:56108885
sun.radha@gmail.com
August 19, 2009

To my family

I dedicate this work to my mother Prema Vasantha,
father Satnarayana Murthy and brother Ravi Krishna
for believing in me and encouraging me to pursue higher studies.

Acknowledgements

Thanks to my supervisor, Prof. Noel E O'Connor, for giving me the opportunity to work in the research group and more importantly for his encouragement and good guidance. I also thank Chanyul Kim, Kealan McCusker, Ciaran O Conaire and Philip Kelly for many interesting discussions and for giving me very valuable advice. I would also like to thank Deirdre Sheridan for support in making travel arrangements for me.

I am also grateful for the financial support I have received from the Informatics Commercialisation initiative of Enterprise Ireland and Science Foundation Ireland.

I was also fortunate enough during my period of study to have an opportunity to visit Prof. Dr-Ing. Walter Stechele's research group Institute for Integrated systems at Technical University Munich. I would like to thank those I worked with there for making my time a rewarding experience. Many thanks to Walter Stechele for inviting me.

I would also like to thank all my friends at Center for Digital Video Processing for making my stay very memorable.

I am also grateful to my family, who always have encouraged my interest to study further, even when it has meant living thousands of miles away from home. Many thanks to all my friends back home in India for the good time I had during my last visit. I would also like to express my gratitude to Chanyul and his family for the affection and friendliness, a special thanks for showing me some beautiful places in and around Ireland.

Radha Krishna. R
sun.radha@gmail.com
Dublin, August 19, 2009

Abstract

A key challenge in a surveillance system is the object detection task. Object detection in general is a non-trivial problem. A sub-problem within the broader context of object detection which many researchers focus on is face detection. Numerous techniques have been proposed for face detection. One of the better performing algorithms is proposed by Viola et. al. This algorithm is based on Adaboost and uses Haar features to detect objects. The main reason for its popularity is very low false positive rates and the fact that the classifier network can be trained for any detection task. The use of Haar basis functions to represent key object features is the key to its success. The basis functions are organized as a network to form a strong classifier. To detect objects, this technique divides each input image into non-overlapping sub-windows and the strong classifier is applied to each sub-window to detect the presence of an object. The process is repeated at multiple scales of the input image to detect objects of various sizes.

In this thesis we propose an object detection system that uses object segmentation as a preprocessing step. We use Mixture of Gaussians (MoG) proposed by Stauffer et.al. for object segmentation. One key advantage with using segmentation to extract image regions of interest is that it reduces the number of search windows sent to detection task, thereby reducing the computational complexity and the execution time. Moreover, owing to the computational complexity of both the segmentation and detection algorithms we used in the system, we propose hardware architectures for accelerating key computationally intensive blocks. In this thesis we propose hardware architecture for MoG and also for a key compute intensive block within the adaboost algorithm corresponding to the Haar feature computation.

Contents

Acknowledgements iii

Abstract iv

1 Introduction 1
- 1.1 Motivation . . . 2
- 1.2 Objective . . . 5
- 1.3 Overview . . . 6

2 Background Modeling and Moving Object Segmentation 8
- 2.1 Introduction . . . 8
- 2.2 Background Modeling . . . 9
  - 2.2.1 Mixture Of Gaussian . . . 10
  - 2.2.2 Post Processing . . . 12
    - 2.2.2.1 Morphology . . . 12
    - 2.2.2.2 Connected Component Labeling . . . 14
  - 2.2.3 Discussion . . . 16
- 2.3 Segmentation Quality . . . 17
- 2.4 Conclusion . . . 20

3 Introduction to AdaBoost Based Object Detection **25**
3.1 Introduction . . . 25
3.2 Face Detection Techniques . . . 26
3.3 Haar Features and Integral Image . . . 27
3.3.1 Haar Basis Functions . . . 27
3.3.2 Integral Image . . . 29
3.4 Training with Adaboost . . . 30
3.4.1 Haar Basis set as Classifiers . . . 31
3.4.2 Boosted Cascade of Classifiers . . . 32
3.4.3 Discussion . . . 33
3.5 Open CV Haar Cascade . . . 34
3.6 Conclusion . . . 35

4 Feature Element For Adaboost Object Detection **39**
4.1 Introduction . . . 39
4.2 MoG Hardware Architecture . . . 40
4.3 Boosting Hardware Architecture . . . 43
4.4 Feature Element based Architecture . . . 46
4.4.1 Hardware architecture . . . 48
4.4.2 Grouping multiple FEs . . . 49
4.5 Conclusion . . . 53

5 Evaluation and Results **54**
5.1 Introduction . . . 54
5.2 Performance Evaluation . . . 55
5.2.1 Detection Accuracy . . . 60
5.2.2 Execution Speed . . . 61
5.3 Hardware Platform . . . 62
5.3.1 OCP Interface . . . 63
5.3.2 Programmer's Model . . . 65
5.4 Results . . . 67
5.5 Conclusion . . . 69

6 Conclusion and Future Work **70**

6.1 Conclusion 70

6.2 Future Work 71

References **79**

List of Figures

1.1 Block Digram of the proposed system 6

2.1 Foreground Segmentation with Uni-Model and Fixed Threshold . 11
2.2 Foreground Segmentation with Mixture of Gaussian 12
2.3 Block Digram of the proposed system 13
2.4 Window mask for 8-way CCL with center pixel e 15
2.5 Typical output at various stages for a sample test sequence. . . . 15
2.6 Moving object segmentation and face detection. 16
2.7 3×3 Structuring Element used for Morphology Operations . . . 18
2.8 MoG applied to a sample Tree test sequence 19
2.9 MoG applied to a sample Bootstrap test sequence 20
2.10 Segmentation quality Comparison for Bootstrap Sequence 21
2.11 Segmentation quality Comparison for Tree Sequence. 22
2.12 Segmentation Quality for Different MoG Models 22
2.13 Segmentation Quality for Different MoG Models 23
2.14 Segmentation Quality Plots with Different Learning Rate. 23
2.15 Segmentation quality with Different Learning Rate 24

3.1 Haar Basis Functions - Lienhart *et. al.* 28
3.2 Haar Features applied to a sub-window - Viola *et.al.* 28
3.3 Integral Image - Viola *et. al.* 29
3.4 Feature Computation - Viola *et. al.* 29
3.5 Face Training Example- The face dataset can be obtained from cbcl (2000). 32

3.6 Classifier Cascade - Viola *et. al.* . . . 33
3.7 Scaling the image to detect faces at various size . . . 33
3.8 Open CV Haar cascade applied to detect faces and pedestrians . 35
3.9 Open CV Haar cascade applied to detect faces . . . 36
3.10 Open CV Feature Database . . . 37
3.11 Open CV Haar Basis Representation - *where* $\Sigma(x_n, y_n)$ *denotes sum of pixels in the rectangular region covered by* (x_n, y_n)*, width and height* . . . 38

4.1 Block Diagram of Mixture of Gaussian. . . . 41
4.2 Internal architecture of Match Model block . . . 41
4.3 Variance update block . . . 42
4.4 Internal architecture of Mean update process . . . 43
4.5 Internal architecture of weight update process . . . 43
4.6 Internal architecture of Sort Model block . . . 44
4.7 Number of Stages for Typical Face Detection . . . 45
4.8 Haar-like feature evaluations . . . 45
4.9 Block Diagram of Face Detection . . . 46
4.10 Classifier Engine using 16 Classifiers . . . 47
4.11 Network-On-Chip Based Face Detection . . . 48
4.12 Proposed Feature Element . . . 48
4.13 Configuration Register with pixel address in clock wise direction. 49
4.14 Internal Architecture of FE . . . 50
4.15 Configuration RAM Data . . . 51
4.16 A Reconfigurable Stage formed by using 16 FEs . . . 52

5.1 Face detection applied to Corridor test sequence- Haar feature window size 24 × 24 and image resolution 640 × 480 . . . 57
5.2 Upper body detection applied to Corridor test sequence - Haar feature window size 22 × 18 . . . 58
5.3 Upper body detection applied to CAVIAR test sequence - Haar feature window size 22 × 18 and image resolution 384 × 288 . . . 59
5.4 Hardware Software Integration Illustration . . . 63
5.5 Integration Platform with Virtual Components . . . 65

5.6 MoG Virtual Component 66

List of Tables

3.1 Summary of face detection techniques 27

5.1 Object detection performance. FS = Full Search detection, MS = Background suppression and detection 60
5.2 Detection accuracy after excluding small objects 61
5.3 Execution time for test sequences. 62
5.4 OCP Bus Signals 64
5.5 MCmd Encoding 64
5.6 SResp Encoding. 65
5.7 MoG OCP interface Registers. 66
5.8 Stage Control and Configuration Registers 67
5.9 MoG Hardware Resource Usage - Synthesized for XC2VP30 . . . 68
5.10 Resource Usage for Reconfigurable FE - Xilinx XC2VP30. 68
5.11 Resource Usage for 16 FE Stage - Xilinx XC2VP30 68

Chapter 1

Introduction

Video surveillance and security systems are crucial for crime prevention, and to protect critical infrastructure. Current generation surveillance systems unlike closed circuit TV (CCTV) systems have an important advantage that they can use the existing internet infrastructure, thus making it much easier to deploy video surveillance equipment in a cost effective way. On the other hand the emergence of visual sensor networks as a promising way to deal with multiple nodes poses many research challenges. One of the key challenges among others is the ability to manage the huge information flow coming from each imaging node (visual sensor). Transmitting image/video data to a central station for storage/analysis is not efficient when considering a large number of these camera nodes. In order to make effective use of the visual surveillance system it is of key importance to equip each sensing node with autonomous sensing capabilities i.e. push intelligence into the device. One major drawback with most existing surveillance systems is the lack of the ability to process information locally and only send important information to the central station. The amount of data generated by visual sensor networks is huge and the fact that this data can only be used for later analysis and cannot be used to detect events on-line and in real time restricts these systems from realizing their full potential. One way to make these systems autonomous is to use some of the computer vision and image

understanding techniques developed over the last few years to detect and analyse events in real-time. Another motivation is the fact that the amount of data transmitted across the network would make human inspection and assessment of events in the monitored area very difficult.

The choice of algorithms or techniques that a sensing node should support for autonomous analysis strictly depends on where the node is deployed. For instance, for access control applications face detection and recognition may be sufficient. On the other hand a public place requires pedestrian detection and tracking. It is of vital importance that smart cameras with adequate capabilities are deployed to make effective use of the surveillance and monitoring infrastructure. The key aspects apart from detection performance that need to be considered when implementing an object detection algorithm in a constrained device such as an embedded smart camera is the algorithm's computational complexity and the amount of parallelism in the algorithm. Parallelism in the algorithm enables efficient hardware implementation needed for real-time detection as the majority of detection algorithms are compute and data intensive. This thesis focus on one such important aspect of a smart camera that is the object detection component. Object detection research is very wide and the details presented in this thesis are intentionally restricted in scope to consider only an extremely popular approach.

1.1 Motivation

Moving object segmentation and detection is an important research challenge in computer vision. Some of the key applications that can benefit from such systems include security and surveillance, robotics, human computer interaction, environment monitoring and advanced driver assistance in the automotive industry. Object segmentation and detection deal with two different aspects of computer vision. Firstly, segmentation deals with extracting image regions of interest such as objects that are not part of the background or in general moving objects. On the other hand, detection deals with identifying the object type for example face, human, car, animal etc. Moreover, video surveillance in the last few years has become crucial at various places for general monitoring and crime prevention. However, the volume of digital surveillance cameras deployed makes human

inspection and analysis increasingly difficult. Consequently, there has been an increased effort to build computer vision enabled cameras (smart camera) for security and surveillance, where the camera can carry out autonomous analysis of the captured scene to detect events of interest. Object detection and classification hold the key to many other high level applications such as face recognition, human computer interaction, security and tracking among others.

Object detection in general is a non-trivial problem. In a typical surveillance system the main goal is to detect and classify objects for example faces, humans, animals, vehicles etc. For effective object detection the key is to identify a set of unique features that distinguish the object from other objects. Apart from the features, the training and detection methodology also contribute greatly to detection performance. There are multiple challenges within the broader context of object detection. For instance for face detection, which many researchers focus on, the goal is to detect faces whilst overcoming some of the following problems (as mentioned in Yang et al. (2002)):

Pose: The position of face relative to camera can vary, resulting in partial or total occlusion of facial features (e.g. profile, upside-down, 45 degree).

Presence of other structural components: Facial features such as beards, mustache and glasses etc.

Change in facial expression: The appearance of a face is directly affected by facial expression.

Occlusion: partial occlusion of face by other objects.

Image Orientation: In plane rotation of faces.

Imaging conditions: Change in lighting etc.

Face detection in general can be classified as a sub problem within the broader area of object detection. A general consensus that has emerged is that some of the detection techniques developed for face detection can be used in a broader context of object detection Yang et al. (2002). While numerous methods have been proposed to detect faces, the ones based on learning have demonstrated

good detection performance, for instance Eigen faces Turk and Pentland (1991), Kirby and Sirovich (1990), neural network techniques Sung and Poggio (1998), Feraud and Bernier (1998), Rowley et al. (1998) and techniques that use a support vector machine (SVM) Osuna et al. (1997), and finally the approach of Viola and Jones (2004) that is described in detail below as it is the approach adapted in this thesis. For a detailed review of different face detection techniques and their detection performance the reader is referred to Yang et al. (2002) and Hjelmas and Low (2001). Many of these techniques have been successfully applied to other pattern recognition problems such as optical character recognition, object recognition and autonomous robot driving. For example, Papageorgiou et al. (1998) applied SVM based techniques for object detection. Lucas (1993) used neural networks for optical character recognition (OCR).

Adaboost based face detection proposed by Viola *et. al.* has proved to be one of the best techniques because of its low computational complexity (unlike other techniques) containing only simple arithmetic operations such as addition, subtraction and comparison. Because of the algorithm's better performance it is implemented in both Intel IPP (2008) and Open CV (2008). Also the high amount of parallelism present in the algorithm and the fact that it can be used in a broader context of object detection makes this algorithm very suitable for embedded smart cameras. The key idea proposed by Viola *et.al.* is the use of Haar like features to represent important distinguishing features of objects combined with adaboost based training. For instance, for face detection a set of positive and negative face samples are used to train the face detector. The output of the training process is a strong classifier with a set of Haar features. In the detection phase, this technique divides each input image into non-overlapping sub-windows and the strong classifier is applied to each sub-window to detect the presence of a face. The process is repeated at multiple scales of the input image to detect faces of various sizes. The same training and detection methodology can be used to detect objects of other classes provided the training process uses training samples from the target object class (see Chapter 13 on Machine learning Bradski and Kaehler (2008)). The authors in Miyamoto et al. (2006) and Snow et al. (2003) successfully applied this technique to pedestrian detection.

As mentioned earlier it is very important to make visual sensing nodes more

intelligent (smart) by giving them the capability to carry out autonomous real-time analysis of the captured scene. Object detection is one among several other potential computer vision and image understanding techniques that a smart cameras can benefit from. Adaboost based face detection proposed by Viola *et. al.* uses machine learning techniques to train a network of weak classifiers for a specific detection task and has gained much popularity due to its high detection rate and low false positive rate. Despite its low computational complexity this algorithm cannot achieve real-time performance on a typical embedded RISC platform Hiromoto et al. (2009). For instance the Open CV implementation of full frame search face detection on a 640×480 image takes around 400 ms on a Pentium Core 2 Duo 2.4 GHz machine. However, in a surveillance application a full frame search is not always necessary and a significant amount speed-up could be achieved by only sending image region of interest to the detection stage. The region of interest can be obtained by some form of moving object segmentation technique where regions of motion are extracted and later sent to the detection stage. This could significantly reduce the number of sub-windows processed by the detector thus increasing the detection speed.

Moving object segmentation is a crucial step in many detection and tracking application. Motion segmentation aims to accurately detect moving objects from a temporal image sequences. However moving object segmentation is a non-trivial problem and many techniques have been proposed. One approach to robust moving object segmentation which we focus on in this work is background modeling. In this technique, a model of the background scene is built and each incoming frame is compared against the background model to identify foreground objects. Many algorithms of varying complexity have been proposed that provide varying segmentation quality Cheung and Kamath (2005), Pless (2005) and Oliver et al. (2000) (to name just a few). However, one of the popular algorithms is the mixture of Gaussian (MoG) proposed by Stauffer and Grimson (1999). This technique is successfully used in surveillance type applications Shahid et al. (2008), Liao et al. (2006). In this algorithm Steuffar *et.al.* model each pixel using multiple Gaussian probability distribution functions (PDFs). This algorithm performs well in dynamic outdoor environments where there are numerous moving objects for example trees, persons etc. The Mixture of Gaussian approach can be treated

as a pixel process where each incoming pixel is matched against all the models to decide if the pixel is part of background or the foreground.

1.2 Objective

As mentioned earlier, the use of motion segmentation is an effective way to speed-up adaboost detection in a surveillance application. In this thesis we propose an object detection system by combining one of the popular background modeling and object detection techniques, that is the MoG and the adaboost respectively. The system starts by forming a robust model of the background and each subsequent image is compared with this background to extract the foreground pixels. Many typical object segmentation techniques usually generate noise pixels and false positive pixels due to change in lighting conditions. Some form of post processing is needed to remove these unwanted foreground pixels for better detection performance. We use morphological operations and connected component labeling in the post processing stage. Figure 1.1 shows the block diagram of the proposed system. The output from the post processing stage is sent on to the detection stage. One key advantage with adaboost is that it can be used for any detection task provided the training data belongs to the target object class. As mentioned earlier the adaboost training and detection process is implemented in Open CV (2008) and comes with a number of pre-trained Haar classifiers for frontal face, profile face, upper body and full body. The Open CV default Haar classifiers come in an .XML database file (more on this in Chapter 3) and we use these pre-trained database to evaluate our object detection system.

Another key challenge with the proposed system is that, the MoG and the Haar feature computation for object detection are both compute and data intensive. In order to achieve real-time performance it is essential to accelerate these compute intensive blocks in hardware. In this work we build a hardware accelerator for MoG and for the Haar feature computation. The key building block for Haar feature computation in this work is the Haar Feature Element (FE). To enable multiple object detections the FE is designed in a way that, it can be configured to form any function from the Haar basis set Lienhart and Maydt (2002). Moreover, these FEs can be grouped to form a generic configurable clas-

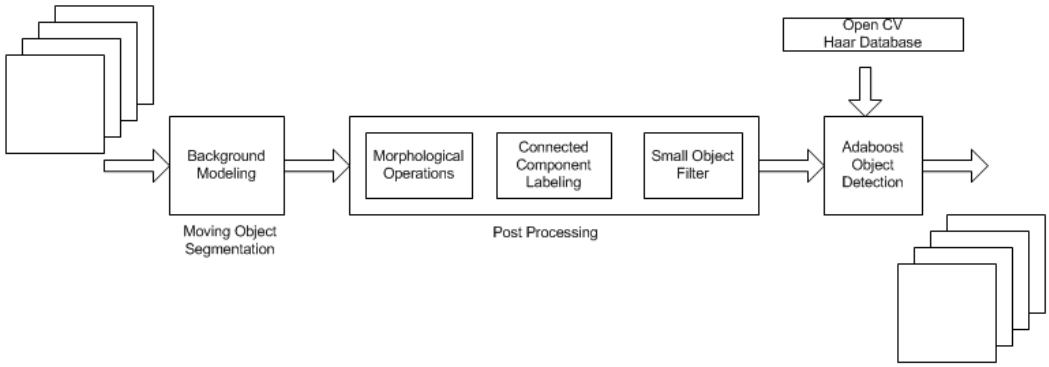

Figure 1.1: Block Digram of the proposed system

sifier stage. With the proposed FE based approach and grouping multiple FEs the adaboost cascade can become more flexible. The FE based approach has an advantage that it is configurable and can be used to different detection tasks.

1.3 Overview

In Chapter 2 details of background modeling and post processing techniques used in the system are presented. Chapter 3 focuses on the algorithmic issues of adaboost based object detection. Adaboost training process, the key to this detection technique is explained in Section 3.4. The Haar basis set and the concept of integral image that enables fast Haar feature computation is explained in Section 3.3. Hardware implementation issues are presented in Chapter 4. In Section 4.3 details of existing hardware architectures for Adaboost are presented followed by details of the reconfigurable FE in Section 4.4. In Chapter 5 hardware implementation results of MoG, Feature Element and details of Xilinx XUP board used for evaluating FE are presented. Finally Chapter 6 presents conclusions and outlines some future work.

Chapter 2

Background Modeling and Moving Object Segmentation

2.1 Introduction

Smart cameras are becoming increasingly popular in surveillance systems for detecting humans and vehicles in dynamic scenes. The task is to push intelligence and adaptability, to the device itself for autonomous analysis of the captured image. One of the first stages in many video/image analysis tasks for surveillance applications is object segmentation. Moving object segmentation is one of the most important research areas in computer vision. Over the last few years researchers have proposed many techniques for object segmentation. These techniques range from very simple frame-differencing with an adaptive threshold Rosin and Ioannidis (2003) to background subtraction Piccardi (2004) and complex feature based segmentation Zhu et al. (2005). However in this work we are concerned with background modeling based approaches. In background subtraction, a robust model of the background scene is built and each incoming frame is subtracted from this background model, and an adaptive threshold technique is used to identify foreground pixels. However background modeling is a complex problem, mainly due to changes in lighting conditions and small moving objects among others. The rest of the chapter presents a brief description of various challenges encountered in object segmentation and then the details of the mixture of

Gaussian approach which we use in this work are presented.

2.2 Background Modeling

An important element in many surveillance applications is the moving object segmentation module. Background subtraction is the most popular technique used for moving object segmentation used successfully in Sato and Aggarwal (2004), Ran et al. (2005) and Snidaro et al. (2005). The task of this module is to identify foreground pixels and ignore background pixels. The foreground pixels are later processed by complex detection, identification and tracking modules. However robust background modeling is not always practical due to the following reasons as mentioned in Toyama et al. (1999b):

Moved object: A background object for example a chair or phone is moved and should not be considered part of the foreground.

Time of day: Gradual illumination changes occur in both indoor and outdoor scenes and the background modeling technique should adapt to these changes.

Light switch: Sudden changes in lighting conditions could effect the scene and the technique should quickly adapt to these sudden lighting changes.

Waving trees: Constant periodic motion of the background objects can corrupt the background model.

Camouflage: A foreground object's pixels can sometimes have the same value/pattern as the background object making the object difficult to detect.

Bootstrapping: A background modelling technique builds a model of the background from training images that are free of foreground objects. However for some environments it is not practical to obtain object free training images, for example public places.

Foreground aperture: When a homogeneously colored object moves, changes in interior pixels cannot be detected. Thus, the entire object may not appear as foreground but only parts thereof.

Sleeping person: When a foreground object stops moving it is hard to distinguish the motionless foreground object from other background objects.

Walking person: When an object initially part of the background starts moving, both the object itself and newly revealed parts of the image appear as foreground.

Shadows: Moving objects often cast shadows and can result in identifying the shadow regions as foreground.

In the last few years many background modeling techniques were proposed to handle various scene types. For a detailed review of various techniques the reader is referred to Piccardi (2004). A popular approach among these methods due to its relatively low computational cost is the uni-model background. This technique is more suitable for scenes that are not very dynamic (e.g. indoor scenes). In this technique a background model is formed by averaging N frames without any foreground objects. Then each subsequent incoming frame's pixels are declared as foreground if,

$$|Im_t(x, y) - Bg(x, y)| > Th \tag{2.1}$$

where Im_t is current image, Bg is the background image and Th is a threshold. This technique is not robust in the sense that it cannot adapt to changes in lighting conditions, bootstrapping and moving background objects. Figure 2.1 shows typical foreground obtained using this technique, and clearly shows the lack of ability to handle sudden change in light and moving background objects. The test sequences light switch, bootstrap and waving tree are used by Toyama et al. (1999b) to evaluate different background modeling techniques and can be downloaded form Toyama et al. (1999a). The result shown in the figure are after some post-processing steps to remove noise. The details of post-processing are discussed in detail in Section 2.2.2.

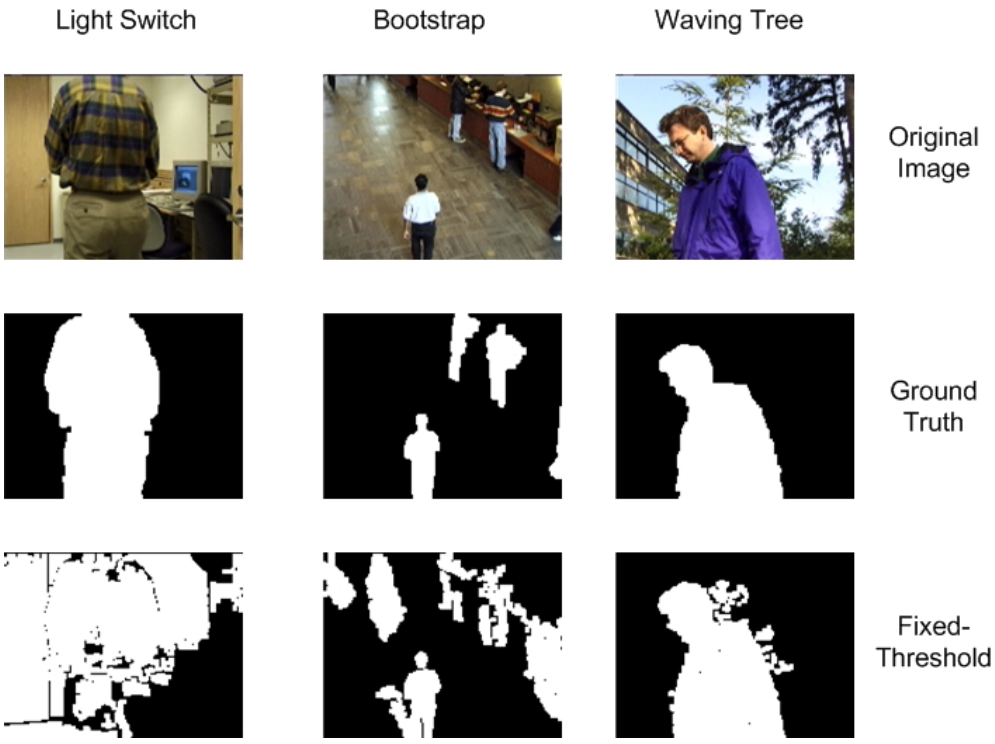

Figure 2.1: Foreground Segmentation with Uni-Model and Fixed Threshold

2.2.1 Mixture Of Gaussian

The technique discussed above uses a uni-model background and a fixed threshold to identity foreground pixels and is suitable for static indoor scenes with no change in lighting and moving objects. However the majority of outdoor scenes in surveillance applications are very dynamic so one alternative to deal with multi-modal backgrounds is the mixture of Gaussians as proposed in Stauffer and Grimson (1999). This algorithm performs well when multiple objects, which are part of the background and appear at the same pixel locations e.g. *waving trees, moved chairs etc.* To cope with multiple background objects each pixel is modeled with multiple Gaussian probability distribution functions (PDFs). The number of Gaussians is determined by the available resources and the required accuracy. Following Stauffer and Grimson (1999), many researchers proposed modifications to the basic technique to improve robustness and reduce update complexity. The Mixture of Gaussians (MoG) approach can be treated as a pixel

process where each incoming pixel is matched against the existing K models. A match is determined by Equation (2.2). If a match is found the respective model's mean-μ_i, variance- σ_i and weight-w_i are updated as in Equations (2.3), (2.4) and (2.5) respectively Wang and Suter (2005).

$$|x_t - \mu_{i,t-1}| < 2.5\sigma_{i,t-1} \tag{2.2}$$

$$\mu_{i,t} = \mu_{i,t-1} + M_i\alpha(x_t - \mu_{i,t-1}) \tag{2.3}$$
$$\sigma^2_{i,t} = \sigma^2_{i,t-1} + M_i\alpha((x_{i,t} - \mu_t)^T(x_{i,t} - \mu_t) - \sigma^2_{i,t-1}) \tag{2.4}$$
$$w_{i,t} = w_{i,t-1} + \alpha(M_i - w_{i,t-1}) \tag{2.5}$$

where α is the learning rate, M_i is 1 when the new pixel matches the i^{th} model, otherwise 0. Following the update process, the models are sorted according to their new weights. Figure 2.2 shows typical output obtained from MoG approach.

2.2.2 Post Processing

Foreground regions obtained by many of the techniques described earlier need some form of post processing to remove spurious noise pixels, false positives and remove small objects. In a typical object segmentation system the output image generated by the background suppression process is further processed by a series of post processing stages such as morphological opening and closing Gonzalez and Woods (2002). Following the morphological operations a connected component labeling (CCL) step is used to remove small objects. The gray areas in Figure 2.3 show a block diagram of a typical system containing object segmentation and post processing steps.

2.2.2.1 Morphology

Morphology operations (MOs) are a well known and effective image processing tools, used in image filtering, segmentation and pattern recognition among others. MOs are a set of window based operations for shape analysis based on set

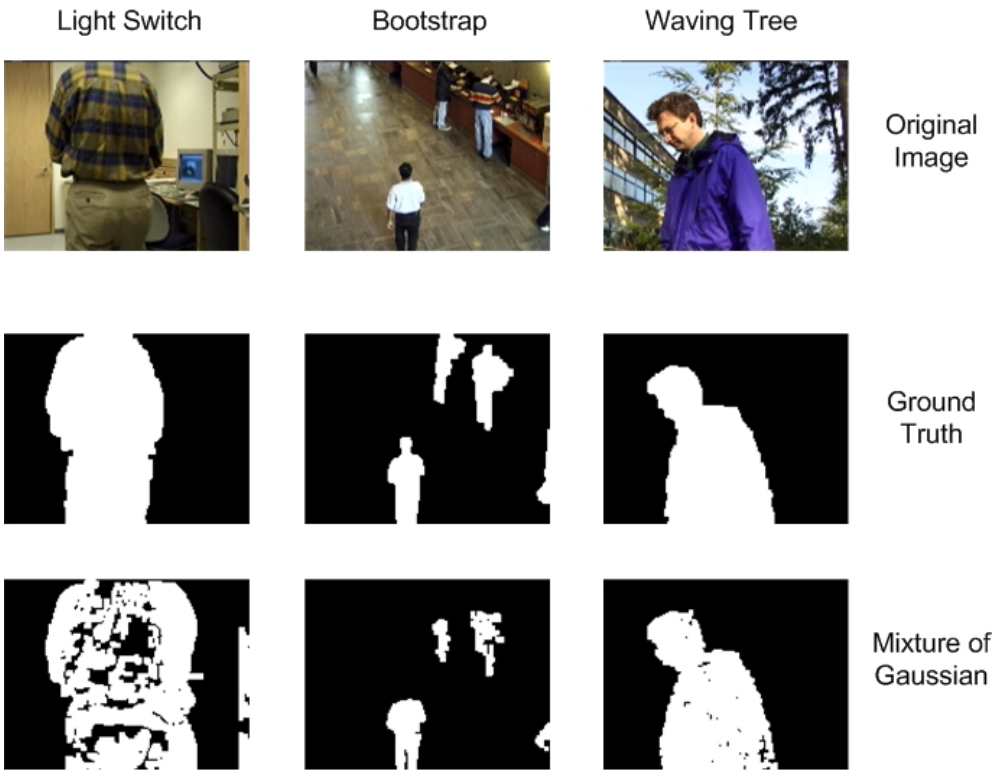

Figure 2.2: Foreground Segmentation with Mixture of Gaussian - Wallflower test dataset and image resolution 160 × 120

theory (intersection, union, inclusion, complement etc.) Gonzalez and Woods (2002). The transformed image has fewer details, but its main characteristics are preserved. Once the image is transformed, measurements can be made to give a quantitative analysis of the image. Morphology operation are based on a structuring element (SE), characterized by shape, size and origin. Images are transformed by centering the SE on each pixel and replacing it with a new value. MOs were first developed for binary images and later extended for gray scale images. In this section details of two basic morphology operations *erosion and dilation* are presented.

Binary dilation of an image A with a SE B is defined as $\psi_\delta = A \oplus B = \{z|(B)_z \cap A \neq \phi\}$. In simple terms binary dilation is where the center pixel of the SE in the input image is replaced with 1 if any of the neighboring 4 or 8 (depends on the SE used) pixels are 1, otherwise the center pixel is set to zero.

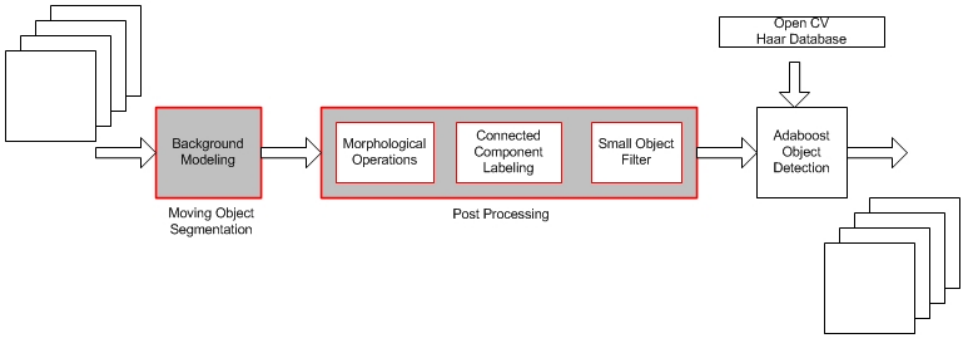

Figure 2.3: System Block Diagram

Binary erosion is the opposite of dilation, which is defined as $\psi_\delta = A \ominus B = \{z|(B)_z \subseteq A\}$. Erosion is where the center pixel in the SE in the input image is replaced with 1 if all of the neighboring 4 or 8 (depends on SE used) pixels are 1, otherwise is set to zero.

Morphological concepts can be extended to gray scale images, given an image A of size $N \times N$ and a SE B of size $2M+1 \times 2M+1$; gray scale dilation is defined as

$$\psi_{(x,y)} = A \oplus B = Max_{i,j}[A(x-i, y-j) + B(i,j)].$$

Similarly gray scale erosion is defined as

$$\psi_{(x,y)} = A \ominus B = Min_{i,j}[A(x-i, y-j) - B(i,j)].$$

for all i, j, such that $-M \leq i, j \leq M$, with $0 \leq x, y \leq N-1$.

Several other MOs can be formed by combining erosion, dilation and by varying the SE; for example opening, closing, thickening and thinning among others; refer to Tickle et al. (2007) and Gonzalez and Woods (2002) for different morphology operations. Opening and closing are duals of each other with respect to function complement and reflection. Opening smooths contours of an object, breaks narrow connections and eliminates protrusions while closing also soothes but fuses narrow gaps and eliminates small holes. Opening an image A by a SE B is defined as:

$$A \circ B = (A \ominus B) \oplus B$$

Thus, opening is erosion of A by B, followed by dilation by B. Similarly closing is defined as:

$$A \bullet B = (A \oplus B) \ominus B$$

2.2.2.2 Connected Component Labeling

Identifying connected pixels is a fundamental concept used for extracting objects in images. In connected component labeling all connected pixels are given unique labels. These are two common ways of defining connectedness in a 2D image, 4-way or 8-way connected. Connected component labeling (CCL) is an important tool in image analysis and pattern recognition and also the most time consuming. There are many algorithms in literature for CCL, however the one which is of importance in the context of this thesis is multi-pass technique, where provisional labels are assigned to an object pixel while scanning in forward (left to right and top to bottom) and backward directions. The scan process is repeated and provisional labels are updated in each pass until there is no change in the image.

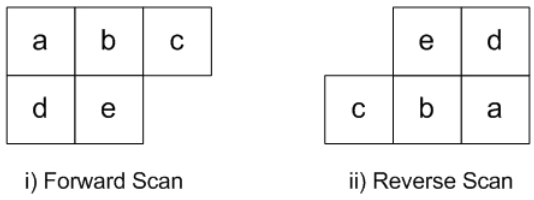

Figure 2.4: Window mask for 8-way CCL with center pixel e
Suzuki et al. (2003)

Although CCL can be performed for gray scale images, here we only consider binary images for the sake of simplicity. Consider a binary image I, where $I[x, y] = 0$ denote background pixels and $I[x, y] = 1$ denote foreground pixels.

Eight way connected component labeling is performed by following a forward raster scan order, using a mask shown in figure 2.4 with mask center at pixel **e**. Provisional labels are assigned to each new pixel as follows:

$$L_e \leftarrow \begin{cases} 0 & I[e] = 0 \\ l, l \leftarrow (l+1) & I[i] = 0; \forall i \in (a,b,c,d) \\ L_{min}[i]_{\forall i \in (a,b,c,d) | I[i]=1} & otherwise \end{cases} \tag{2.6}$$

Where $L[e]$ is the transformed image and e is current pixel in window. If $e = 0$, $L[e]$ remains 0. If $e = 1$ and all neighboring pixels are 0 i.e. background, a provisional label l is assigned to $L[e]$ and l is incremented by one. Otherwise, $L[e]$ is assigned the minimum of the provisional labels in the window mask. In subsequent scans both forward and reverse, labels of pixels are replaced by minimum labels in the mask window as: $L[e] \leftarrow L_{min}[i], i \in (a,b,c,d)$. The window mask in figure 2.4.ii is used for the reverse scan with mask center at pixel **e**. The output image from morphology operations is processed by the connected component labeler and once connected pixels are labeled a small object filter is used to remove objects regions that are too small to consider for further processing. Figure 2.5 shows the output at various stages in the system for a sample test sequence. As mentioned earlier output from background suppression module typically contains unwanted noise pixels and by applying post-processing a significant reduction in noise pixels can be obtained.

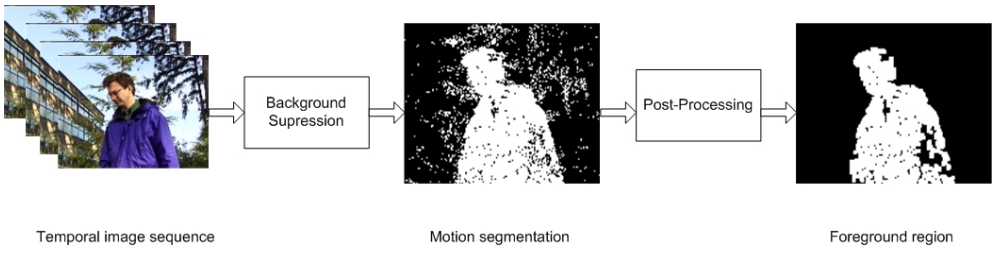

Figure 2.5: Typical output at various stages for a sample test sequence

In this work background suppression, morphology and connected component labeling play an important role in properly segmenting moving regions. A subwindow is extracted based on the bounding box containing the foreground object

and sent to the detection stage as shown in the Figure 2.6. A significant increase in the detection performance can be obtained in this case because the detection stage instead of processing the whole frame to detect faces now processes only regions of motion.

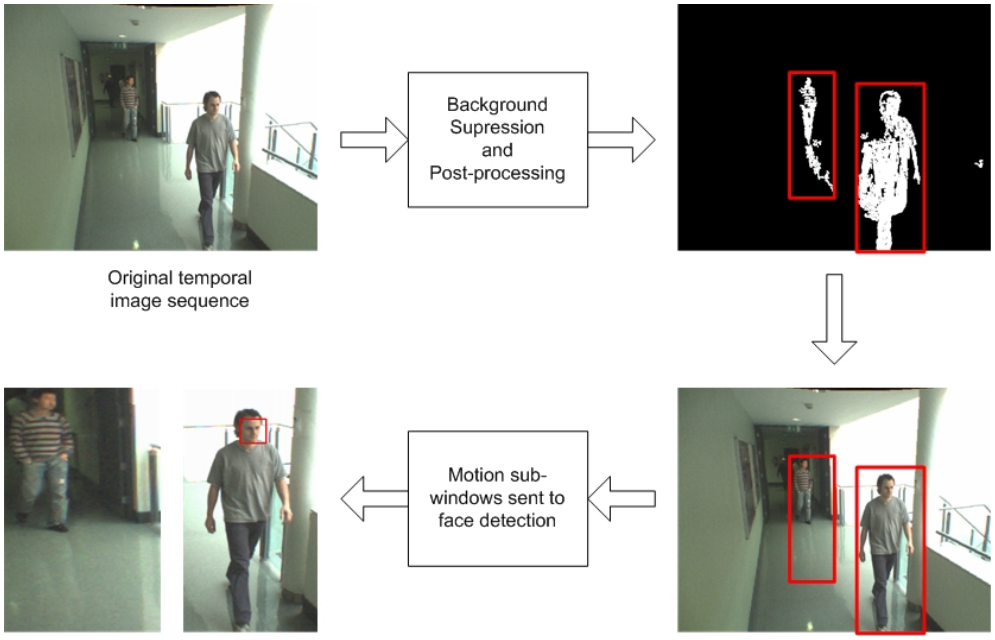

Figure 2.6: Moving object segmentation and face detection

2.2.3 Discussion

Among other background modeling techniques such as Eigen background Li et al. (2006), Codebooks Kim et al. (2004) and non parametric models Elgammal et al. (2000) we choose to use MoG because of its relatively low computational complexity and reasonably good segmentation performance. Online update of Eigen background for instance requires matrix multiplication which is compute intensive, also the codebooks algorithm proposed by Kim *et.al.* although arguably providing better performance is both compute and data intensive. The MoG used in this work gives good segmentation quality and the background update process which is very important for dynamic scenes is simple as explained in Section 2.2.1. Furthermore the fact that MoG is a pixel process makes it highly

parallel and suitable for hardware acceleration as detailed in Chapter 4.

2.3 Segmentation Quality

In this section we compare segmentation accuracy of MoG approach with the uni-model fixed threshold approach. We use the initial 50 frames to construct the background in case of uni-model approach and a fixed threshold of 30 is used. The optimal threshold is chosen based on the visual inspection of segmentation accuracy for various test sequences used in this work. For MoG a learning rate of 0.01 is used to update each model's parameters. The MoG system maintains 3 models of which the first two models represent the background and the third models the time varying foreground. In this work we maintain only 3 models for simplicity and much better segmentation accuracy can be obtained by maintaining more background models, however at the cost of increased computational complexity. The Wallflower Toyama et al. (1999a) test sequences we use here are representative of typical surveillance environments and outdoor scenes. Moreover in the post-processing stage we use morphology operations with a structuring element as shown in Figure 2.7 and we consider foreground objects with pixel count less than 200 as too small for further processing. The threshold set for small object filter mainly depends on the resolution of the image and the average size of noise regions observed in the image. A detailed analysis for choosing the optimal threshold for small object filter is outside the scope of this work. The optimal threshold for small object filter in this work is chosen based on visual inspection of segmentation result for various test sequences used in this thesis. However for the adaboost object detection system that we will describe in Chapters 3 and 5 the minimum window size depends on the target object and the Open CV database used. For instance for faces the Open CV default window size is 24×24. To apply Haar features we calculate integral image on a sub-window of size 25×25 and thus the minimum threshold for small objects is 625 pixels. Similarly for upper body the default window size is 22×18 and the minimum size of segmented regions is 437 pixels.

Segmentation quality is the amount of accuracy with which an algorithm separates the foreground. To validate the segmentation quality of MoG, we compared

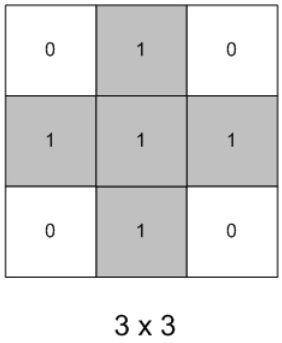

Figure 2.7: 3 × 3 Structuring Element used for Morphology Operations

the segmentation results with hand segmented ground truth using the Rand index as introduced in Rand (1971) as the segmentation quality measure. The ground truth is generated manually for the tree and bootstrap test sequences from Toyama et al. (1999a). Rand index is a measure of similarity between two datasets and is given by $\frac{a+b}{a+b+c+d}$ where $a+b$ are the number of agreements and $c+d$ are the number of disagreements between the two datasets. Figure 2.8 and 2.9 shows the sample test sequence, hand segmented ground truth and the output from the MoG technique.

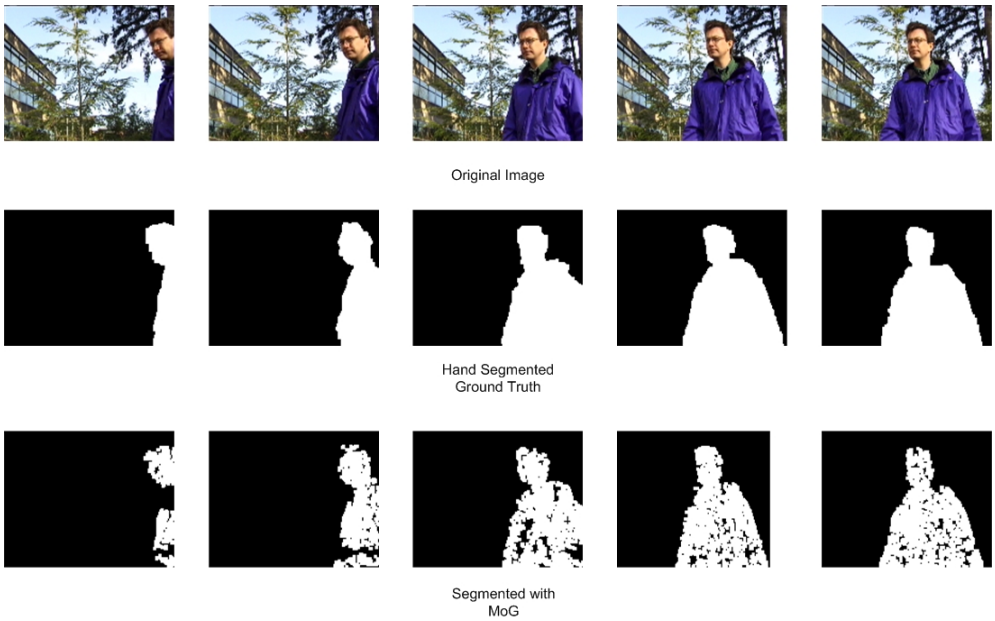

Figure 2.8: MoG applied to a sample Tree test sequence

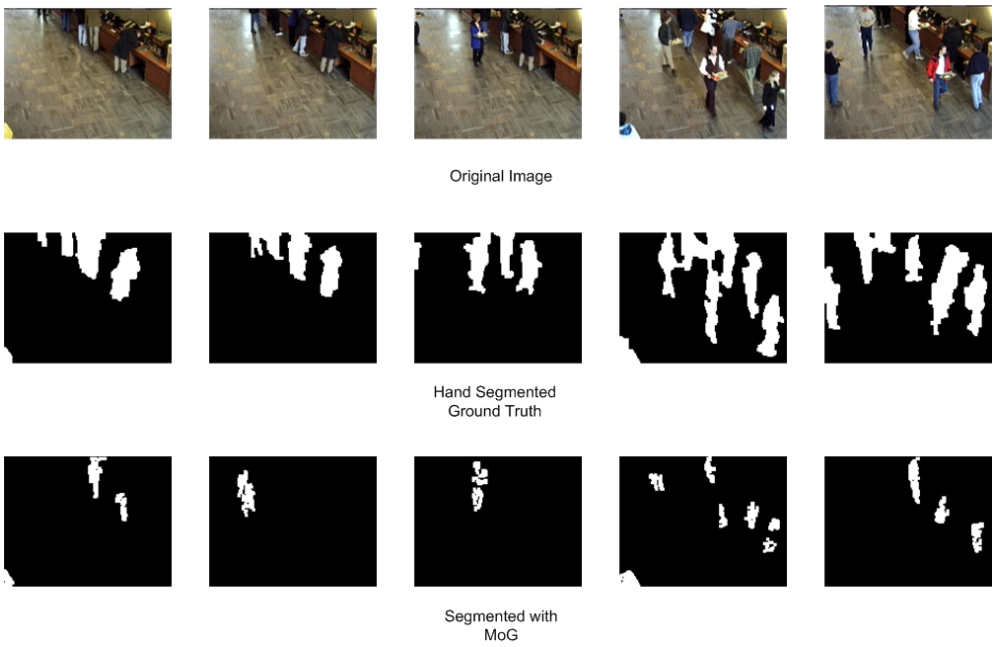

Figure 2.9: MoG applied to a sample Bootstrap test sequence

Plots in Figures 2.10 and 2.11 show segmentation quality achieved with uni-model fixed threshold and MoG approach for two sample test sequences. The post processing stage parameters used are the same for both tests, that is we use morphological opening (dilation followed by erosion) with a 3×3 structuring element as shown in 2.7 and a small object filter with threshold set to 200 pixels. In our test we use a total of 2000 images from bootstrap sequence and 260 images from tree sequence. For bootstrap test sequence we generated ground truth for images from 200 to 2000 with 25 images apart. In case of tree test sequence we generate ground truth from images 243 to 259 i.e. when the foreground object (person) appears in the scene. To compute the Rand index segmentation quality we compare the hand generated ground truth images with segmented images from both fixed threshold and MoG for both test sequences. Higher Rand index values indicate better segmentation quality and the plots in Figure 2.10 and 2.11 show that MoG approach performs better than uni-model with fixed threshold for both the test sequences.

However more importantly the performance of MoG algorithm relies on various parameters such as the learning rate and the number of background models

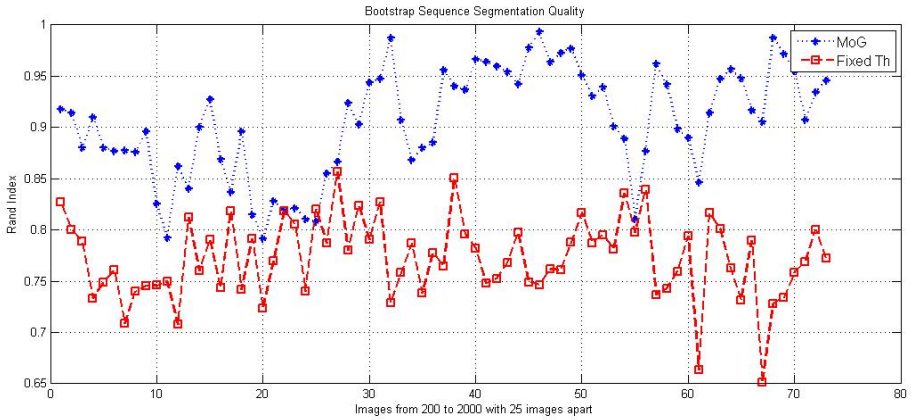

Figure 2.10: Segmentation quality Comparison for Bootstrap Sequence

maintained. Although a higher number of background models give better segmentation quality, the choice of number of background models depends on the required accuracy, available resources and the amount of complexity in the scene. For instance outdoor scenes with several moving objects may require more background models on the otherhand for indoor scenes it is sufficient to maintain fewer background models. Apart from this the learning rate is also an important parameter, for instance a very high learning rate may quickly adapt to changes in the scene but will also push foreground objects into the background there by corrupting the background models. A very low learning rate on the otherhand will be very slow to adapt to changes in the background and this results in background objects appearing as foreground for a long time.

Figures 2.12 and 2.13 show segmentation quality plots for two test sequences with different number of background models. The plots compare MoG approach with 3, 4 and 5 background models in each case. For the test sequences used there is no significant difference in segmentation quality. At frame 255 the MoG with 3 background models show poor quality relative to 4 and 5 models however the drop is insignificant. Although there is no variation in segmentation quality with increasing the number of background models maintained for the test cases used here, this may change for other test cases. That is in case of complex scenes better segmentation quality can be achieved by maintaining more background models. The choice of number of background models is a tradeoff between computational

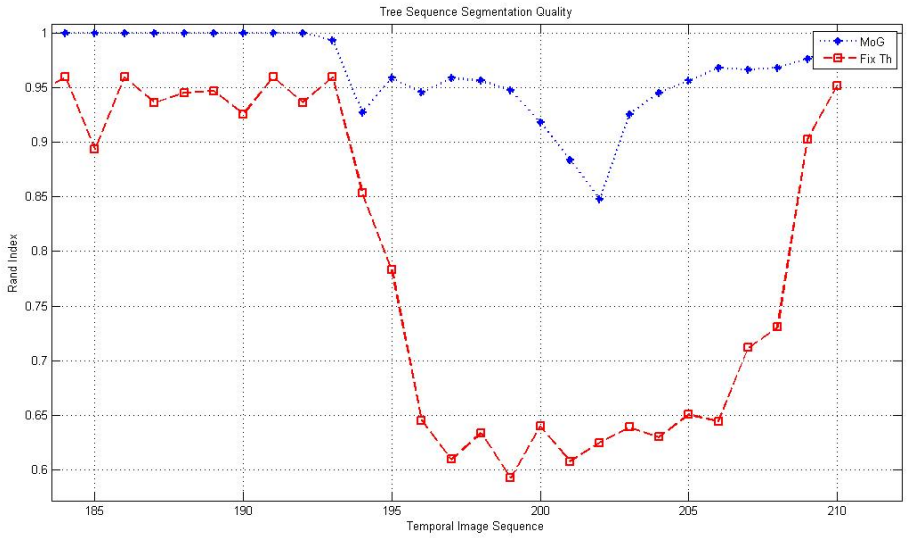

Figure 2.11: Segmentation quality Comparison for Tree Sequence

complexity, speed and required segmentation quality. In this work we maintain 3 models as it achieves reasonably good segmentation quality.

Secondly Figures in 2.14 and 2.15 show segmentation quality plots for test sequences with different learning rates. We use a learning rates of 0.05, 0.01 and 0.005 to test the performance of the MoG approach. In Figure 2.14 the segmentation quality for learning rate 0.005 is poor initially due to the fact that it is very slow to adapt to changes in the background. On the other hand a learning rate of 0.05 shows very poor performance during frames 245-260 because it adapts too quickly to changes in the background. The learning rate of 0.01 adapts reasonably quickly to changes in the background and achieves better segmentation quality. In Figure 2.15 on the otherhand a very low learning rate of 0.05 achieves relatively very poor quality initially and is slow to adapt. Again the choice of learning rate is a trade off between the required update speed and the segmentation quality. In our work we choose a learning rate of 0.01 to maintain reasonable update speed and segmentation quality.

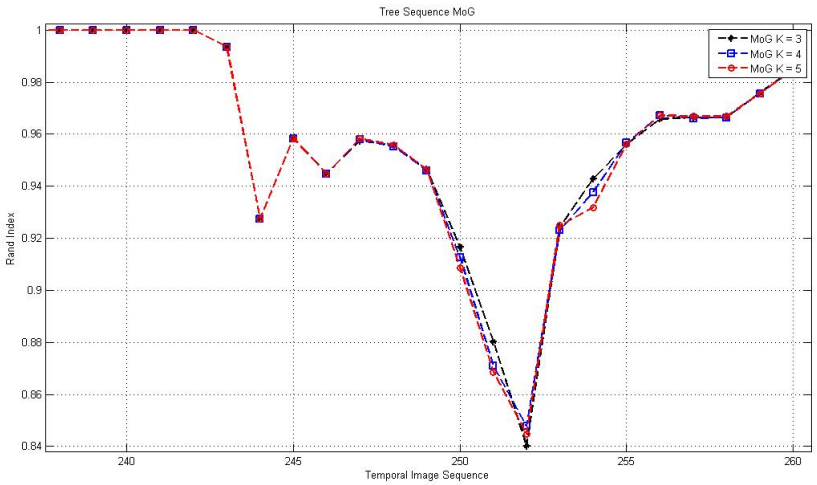

Figure 2.12: Segmentation Quality for Different MoG Models- Tree Sequence

2.4 Conclusion

A key bottle neck when implementing face detection or an object detection algorithm in a surveillance system is the enormous amount of time consumed for full frame search. An effective way to reduce the computational time in such systems is by focusing only on the motion regions. Clearly a background suppression based motion segmentation technique can be used effectively to extract sub-windows. These sub-windows can be later processed to detect any objects of interest. In this work we use mixture of Gaussians as a pre-processing step for object detection. The key advantages with MoG is that it is adaptive in the sense that the background is updated with every incoming frame and also the fact that it is suitable for hardware acceleration. Apart from this the post-processing steps play another key role in properly segmenting the foreground regions. In this work we use a set of morphological operations and a small object filter to remove unwanted pixels and objects that are too small for further processing. The output from this stage is a bounding box containing foreground objects that are sent to the detection phase.

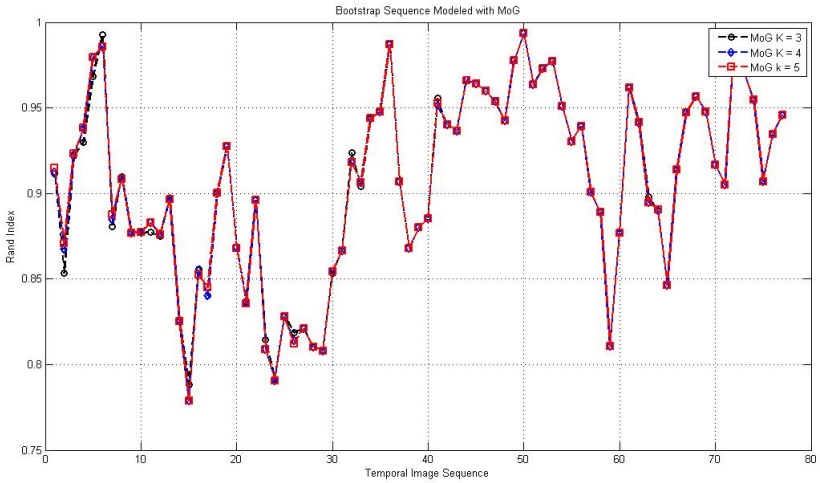

Figure 2.13: Segmentation Quality for Different MoG Models- Bootstrap Sequence

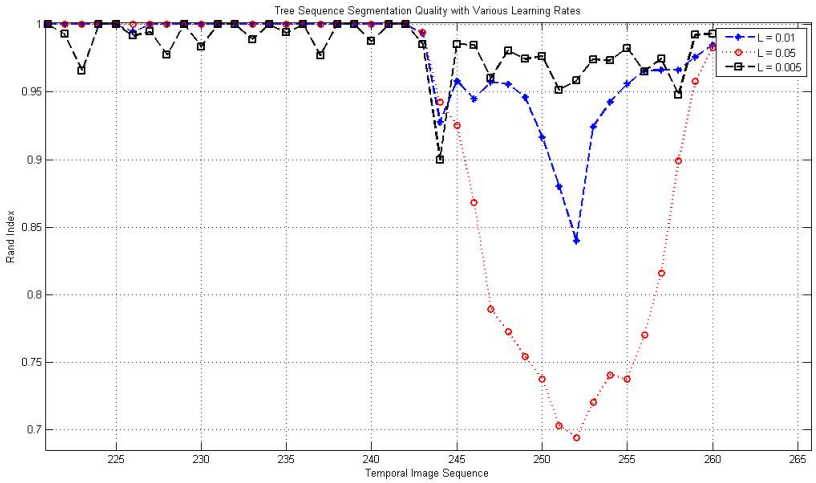

Figure 2.14: Segmentation Quality with Different Learning Rate - Tree Sequence

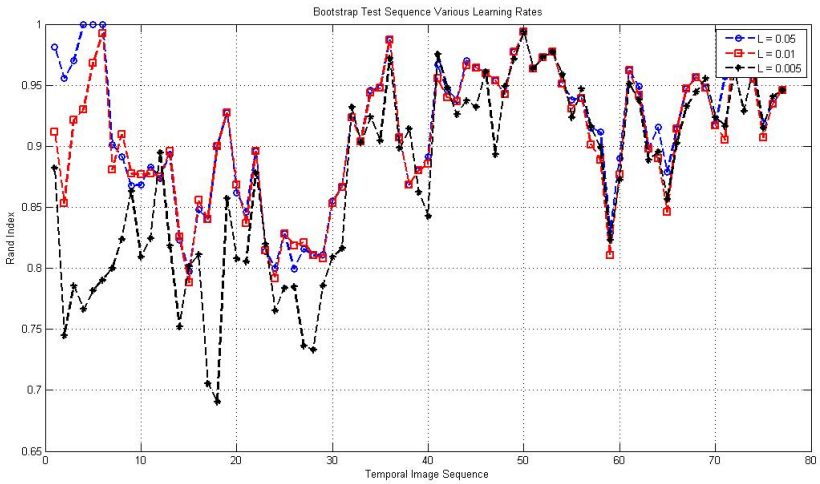

Figure 2.15: Segmentation quality with Different Learning Rate - Bootstrap Sequence

Chapter 3

Introduction to AdaBoost Based Object Detection

3.1 Introduction

One of the most popular and widely used techniques for object detection tasks was proposed by Viola and Jones for detecting faces. The technique uses machine learning to train a network of weak classifiers/detectors for a specific detection task. One reason for the technique's popularity is its very low false positive rate Hiromoto et al. (2007), and also the fact that the detector can be trained for different object classes Lienhart et al. (2002), Lienhart and Maydt (2002), Miyamoto et al. (2006) and Snow et al. (2003). The technique uses Adaptive Boosting formulated by Yoav Freund and Robert Schapire Freund and Schapire (1995), a popular machine learning technique for selecting a set of better performing weak classifiers from a pool of over complete weak classifiers. A weak classifier in simple terms is a decision rule that classifies a test sample as either a positive or a negative sample. A weighted linear combination of these weak classifiers forms a strong classifier with improved detection performance. The boosting process has two stages corresponding to training and detection. In the training stage a very large set of labeled samples is used to identify the better performing weak classifiers, and a strong classifier network is constructed by a weighted linear combination of these weak classifiers. The output of the train-

ing stage is a trained classifier network that can be used in the detection phase to classify samples as positive or negative. For example a typical classification network trained for detecting frontal faces can have nearly 22 stages with 9-200 features per stage Hiromoto et al. (2007). In the detection phase these features are applied to each sub-window to make a decision. The rest of this chapter presents how Haar-like features are effectively used as weak classifiers and the adaboost training process.

3.2 Face Detection Techniques

In this section we present a brief review of the various face detection techniques. Face detection research is very broad and this section is intentionally restricted in scope to provide a very high level overview. The majority of the face detection techniques can be broadly classified in to the following categories Yang et al. (2002):

Knowledge based methods encode human knowledge of what constitutes a face in a rule. The rule typically captures the relationship between different facial features.

Feature invariant approaches find key structural features that are invariant to changes in view point, pose or lighting conditions. These features are later used to localize faces.

Template matching methods have been successfully applied to various pattern recognition problems. In template matching, several face images are used to describe the face as a whole. The correlation between an input image and the stored template images is used for detection.

Appearance based methods in contrast to template matching, try to capture the representative variability of facial features. This technique uses a set of training samples to identify distinct face features and the learned models are then used for detection.

Table 3.1 summaries various face detection techniques. The techniques have different complexity and detection performance. Appearance based methods have

proved to achieve better detection performance among other techniques. The majority of the appearance based methods are based on a learning algorithm on a training set. For instance, Rowley *et.al.* face detection is based on neural network and they use a large set of face images in the training process. Among all face detection methods that used neural networks, the most significant work is done by Rowley *et.al.* Yang et al. (2002). This technique has become an early standard in face detection for the majority of other face detection work. For instance Viola et. al. compare their adaboost based face detection against the neural network technique proposed by Rowley et. al. The adaboost based face detection which we focus on in this thesis has shown significant improvement in detection speed and accuracy as mentioned in Viola and Jones (2004).

Approach	Related work
Knowledge based	
	Multi resolution rule based method Yang and Huang (1994)
Feature based	
- Facial Features	Grouping of edges Leung et al. (1995) Yew and Cipolla (1997)
- Texture	Space Gray Level Dependence matrix of face pattern Dai and Nakano (1996)
- Skin Color	Mixture of Gaussian Yang and Waibel (1996) McKenna et al. (1998)
- Multiple Features	Integration of skin color, shape and size Kjeldsen and Kender (1996)
Template Matching	
- Predefined face templates	Shape template Craw et al. (1992)
- Deformable Templates	Active Shape model Lanitis et al. (1995)
Appearance based	
- Eigenface	Eigenvector decomposition and clustering Turk and Pentland (1991)
- Distribution-based	Gaussian distribution and multilayer prediction Sung and Poggio (1998)
- Neural Network	Ensemble of neural networks and arbitration schemes Rowley et al. (1998)
- Support Vector Machine SVM	SVM with polynomial kernel Osuna et al. (1997)
- Naive Bays Classifier	Joint statistics of local appearance and position Schneiderman and Kanade (1998)
- Hidden Markov Model	Higher order statistics with HMM Rajagopalan et al. (1998)
- Information Theoretical Approach	Kullback relative information Lew (1996), Colmenarez and Huang (1997)

Table 3.1: Summary of face detection techniques
Yang et al. (2002)

3.3 Haar Features and Integral Image

The amount of variation in objects (as mentioned in Section 1.1) makes the object detection task very complex. However, modeling a face by considering variations such as pose, expression, color, texture and lighting can be effectively achieved using a set of Haar like features as proposed by Viola *et. al.*. The use of these features as weak classifiers combined with a adaboost learning technique makes object detection practical. Haar like features proved to be more robust when

representing various features facilitating rapid object detection.

3.3.1 Haar Basis Functions

The Haar basis functions are a set of rectangular 2D features derived from the Haar wavelet, see Equation 3.1. The 2D Haar features are shown in Figure 3.1. There are three kinds of Haar features typically used for object detection: 2-rectangle, 3-rectangle and 4-rectangle features. In practical terms, the output of these features when applied to an image is the difference between the sum of pixels in the white region and the black region.

$$\Psi(x) = \begin{cases} 1 & if \ \ 0 \leq x \leq \frac{1}{2} \\ -1 & if \ \ \frac{1}{2} \leq x \leq 1 \\ 0 & otherwise \end{cases} \tag{3.1}$$

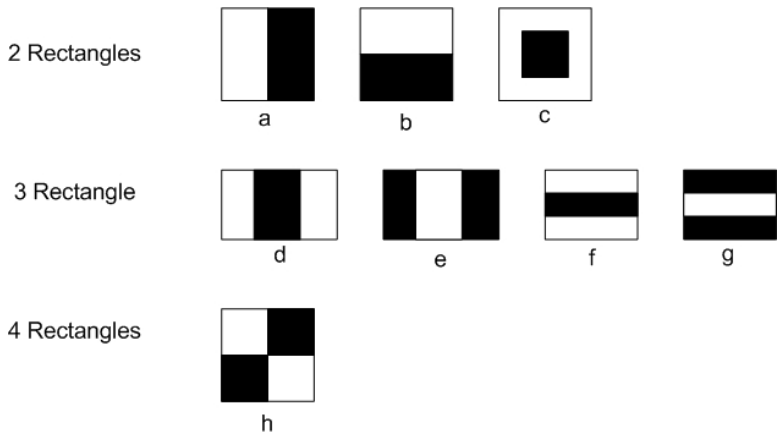

Figure 3.1: Haar Basis Functions - Lienhart *et. al.*

Starting with the basis set the detection algorithm applies scaled and shifted versions of these Haar features at various locations in the image sub-window to make a decision. Clearly this results in a large number of features. For example for a 24×24 sub-window the exhaustive set of features is quite large (approx. 10^5). Figure 3.2 shows how features are applied to a typical face sub-window. The task of the training process described in Section 3.4 is to select a sub-set of Haar

features that best separate the training data into positive and negative samples. The choice of features is important but not crucial for detector performance.

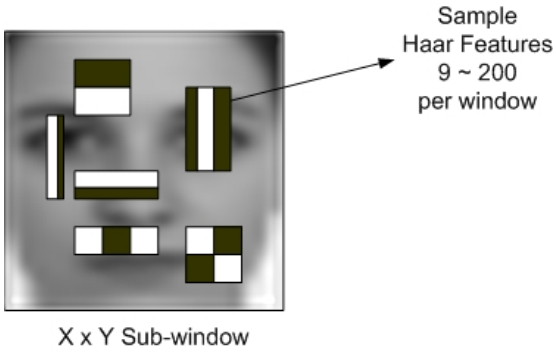

Figure 3.2: Haar Features applied to a sub-window - Viola *et.al.*

3.3.2 Integral Image

One of the other contributions made by Viola *et. al.* is the idea of an "integral image" for fast Haar feature computation. The idea is to pre-compute the sum of pixels to the top and to the left of each pixel as shown in Figure 3.3. After computing the integral image any feature value can be computed very easily, the following example shows one example feature computation:

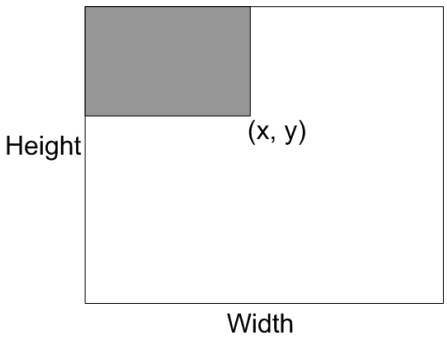

Figure 3.3: Integral Image - Viola *et. al.*

In figure 3.4 the sum of pixels within the rectangular region D can be computed with reference to integral image location 1, 2, 3 and 4. The value of the integral image at locations 1, 2, 3 and 4 are A, A+B, A+C and A+B+C+D respectively.

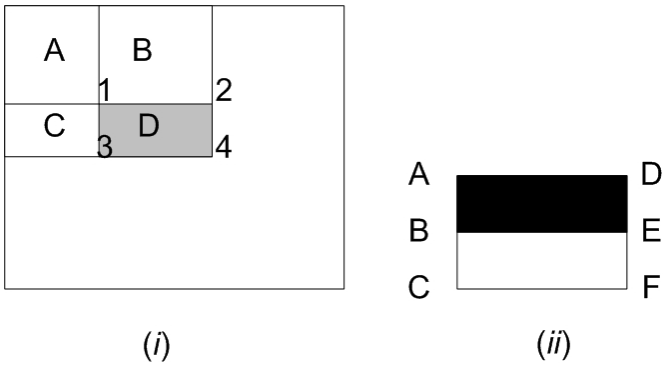

Figure 3.4: Feature Computation - Viola *et. al.*

The sum within D can be computed as 4+1-(2+3). Similarly the value of the feature shown in figure 3.4.*ii* can be computed as D+F-A-C+2(B-E), where A→F denote the values of integral images at respective locations. The key advantage with the integral image is that all Haar basis functions can be computed within a maximum of eight memory accesses, and with few arithmetic computations.

3.4 Training with Adaboost

The idea of boosting is to combine a set of simple "rules" or weak classifiers to form an ensemble such that the performance of a single ensemble member is improved i.e. "boosted" Meir and Rätsch (2003). For example given a family of weak classifiers and a set of training data consisting of positive and negative samples, the adaboost approach can be used to select a subset of weak classifiers and the classification function. Some of the key advantages of adaboost are Meynet (2003):

- **Adaboost requires no prior knowledge**, that is no information about the structure or features of the face are required when used for face detection. Given a set of training samples and weak classifiers the boosting process automatically chooses the optimal feature set and the classifier function.
- **The approach is adaptive** in the sense that misclassified samples are given higher weight in order to increase the discriminative power of the

classifier. As a result, easily classified samples are detected in the first iteration and have less weight and harder samples with higher weights are used to train the later iterations.

- **The theoretical training error converges to zero** as proved by Freund and Schapire (1995). The training for a set of positive and negative samples reaches zero after a finite number of iterations.

Given a feature set $\{h_1, h_2, h_3 \ldots\ldots\}$ and a set of training samples

$$T = \{(x_1, y_1), (x_2, y_2), (x_3, y_3) \ldots\ldots\}$$

where x_i is the training sample and y_i is a binary value of the sample class (1 is positive, 0 negative). A final boosted classifier network is formed from the subset of given features after an arbitrary number of iterations as shown in Equation 3.2.

$$if(x) = \sum_{t=1}^{T} \alpha_t h_t(x) \tag{3.2}$$

where α_t is the weight assigned to the t^{th} classifier, and h_t is the classifier decision. Adabost training is an iterative process and the accuracy of the final classifier function depends on the number of iterations and whether the training error converges to zero after finite number of iterations. A classifier is trained as follows by Viola *et. al.*:

- Given example images $(x_1, y_1), (x_2, y_2) \ldots, (x_n, y_n)$ where $y_i = [0, 1]$ for negative and positive samples respectively.
- Initialize weights $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ for $y_i = [0, 1]$ respectively, where m and l are the number of negative and positive samples respectively.
- For $t = 1, 2, \ldots, T$:
  - Normalize the weights $w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$
  - Select the best weak classifier with respect to the weighting error $\epsilon_t = min_{f,p,\Theta} \sum_i w_i \mid h(x_i, f, p, \Theta) - y_i$.

  - Define $h_t(x) = h(x, f_t, p_t, \Theta_t)$ where f_t, p_t and Θ_t are minimizes of ϵ. Where f_t is the feature, x is the image sample, p_t is the parity of inequality and Θ_t is a threshold associated with each feature.
  - Update weights: $w_{t+1,i} = w_{t,i}\beta_t^{1-\epsilon_i}$ where $\epsilon_i = 0$ if example x_i is classified correctly, $\epsilon_i = 1$ otherwise, and $\beta = \frac{\epsilon_t}{1-\epsilon_t}$.

- The final strong classifier is: $f(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha \\ 0 & Otherwise \end{cases}$ where $\alpha_t = log\frac{1}{\beta_t}$.

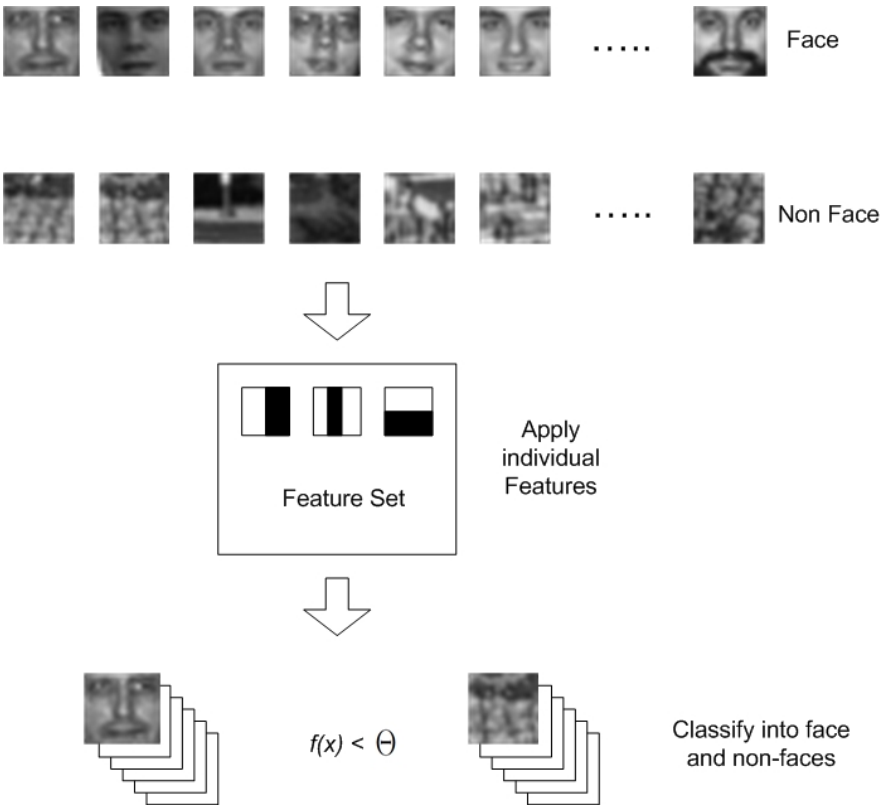

Figure 3.5: Face Training Example- The face dataset can be obtained from cbcl (2000)

3.4.1 Haar Basis set as Classifiers

The role of a weak classifier is to separate the given training set into positive and negative samples with more than 50% accuracy. Figure 3.5 shows an illustration

of the training process. To use Haar basis set as classifiers we compute the value of each feature i.e. compute the difference between the sum of pixels in black region and the white region. The feature value is then compared with a threshold to classify the samples. A weak classifier $h_t(x)$ consists of a Haar feature f_t, threshold Θ_t a left value and a right value. The optimal value of threshold Θ_t that best separates positive and negative samples with more than 50% accuracy is chosen by applying each feature to the given training set. For detailed description on calculating Θ_t the reader is referred to Chapter 3 of Meynet (2003).

One major disadvantage with the boosting process is the amount of time consumed in learning the classifier function. There is one weak classifier for each distinct feature/threshold and a considerable amount of time is required to evaluate all these feature combinations for obtaining an optimal classification function. This grows linearly with the increase in the training and feature set.

$$h_t(x) = \begin{cases} 1 & if \;\; p_t f_t(x) < p_t \Theta_t \\ 0 & otherwise \end{cases} \tag{3.3}$$

3.4.2 Boosted Cascade of Classifiers

To improve the overall performance of the face detection system Viola *et.al.* proposed to combine several of the classifiers in a cascade. According to Viola *et.al.* the detection performance of a single classifier with a limited number of features is very poor for a face detection system. The key idea in a multi-stage classifier as shown in Figure 3.6 is that the initial stages are less complex and can reject the majority of negative samples while detecting almost all positive samples. These positive samples are sent on to further complex classifier stages to achieve low false positive rate. It is out of the scope of this thesis to detail the cascade training process, the reader is referred to Viola and Jones (2004) for a detailed description of cascade training.

To detect faces at different scales a pre-processing stage down scales the input image starting from 1.25 times the original image and a step size of 1.5. Figure 3.7 illustrates this process.

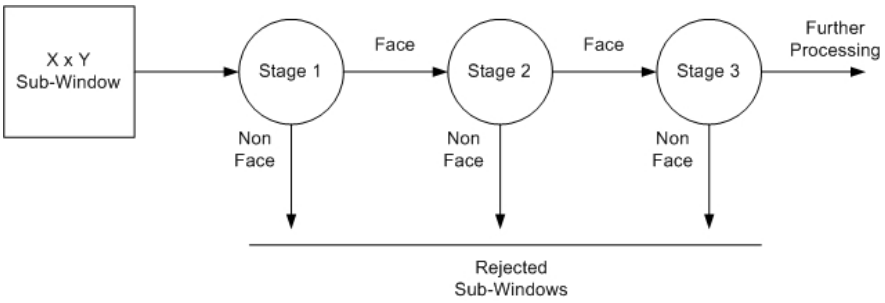

Figure 3.6: Classifier Cascade - Viola *et. al.*

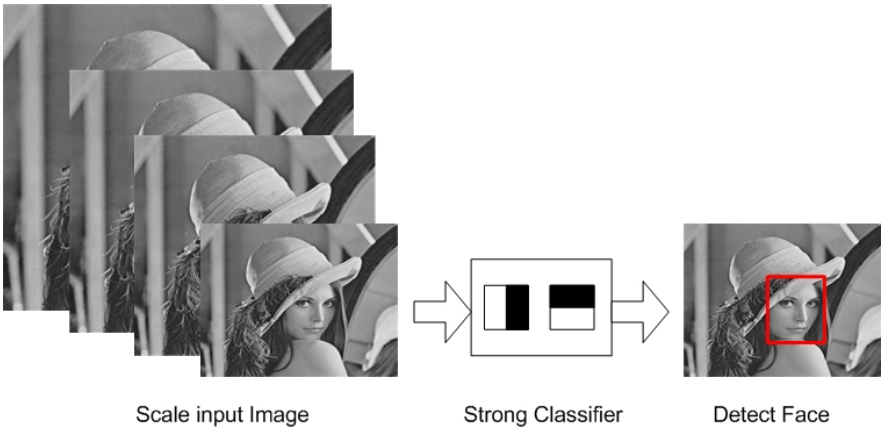

Figure 3.7: Scaling the image to detect faces at various size

3.4.3 Discussion

Face detection is an interesting research challenge and there are many solutions to solve this problem. Among various techniques proposed so far in the literature Viola *et.al.* present what is widely considered to be the best detector in terms of both computation time and detection performance. The authors trained a 38 stage cascade detector for frontal upright faces using 4916 hand labeled faces and 9544 non-faces gathered from the world wide web. The number of features in the first five layers are 1, 10, 25, 25 and 50 respectively. The number of features in later layers is much higher and the total number of features in all 38 layers is 6016. According to Viola *et.al.* their 38 stage cascade takes .067 seconds to scan a 384 $\times$ 288 image at various scales on 700 MHz Pentium III processor, which they claim is 15 times faster than one of the better performing neural network

based face detector proposed by Rowley et al. (1998).

3.5 Open CV Haar Cascade

The adaboost based detector discussed so far is implemented in Open CV, an open source library for image processing and computer vision. Open CV comes with sample database that are already trained to detect frontal face, full body and upper body. Figures 3.8 and 3.9 show Open CV Haar classifier cascade applied to detect faces and pedestrians. To detect other objects the classifier can be trained with a few hundred labeled positive and negative samples of a target object (e.g. person, car). The adaboost training process is highly time consuming and takes days to finish. The output from Open CV Open CV (2008) training process is a .XML file of the Haar features and the thresholds used in the various stages. Figure 3.10 shows the typical structure of the training file. The values within the fields *rects* (Figure 3.10) are *x, y, width, height, weight* of the rectangle feature. In Open CV Haar features are represented as either a two vertex feature or a three vertex feature as shown in Figure 3.11. The only difference is the weight assigned to each vertex. The output of each feature computation is compared with the threshold and the classifier output is decided as shown in Equation 3.4.

After the classifier is trained it can be applied to an image region of interest to detect the presence of a target object. Alternatively the classifier can be applied to the whole image at various scales to localize a target object.

To apply a feature to an image sub-window, the result of each vertex in the feature is computed and multiplied by its corresponding weight. Note that each vertex represents a weighted rectangular region due to the use of an integral image. The sum of pixels in the rectangular region are computed by reading four corner pixels from the integral image as detailed in section 3.3.2. The final feature value is computed by summing the result of all vertices. This feature value is used to decide the classification weight $h_t(x)$. The final sum for a stage is calculated by adding $h_t(x)$ (Equation 3.4) of all features in the stage. The final stage sum is compared with the stage threshold (shown in Figure 3.10.i), and if the final stage sum is less than stage threshold the sub-window is accepted as positive and sent on to later stages in the cascade, otherwise it is rejected.

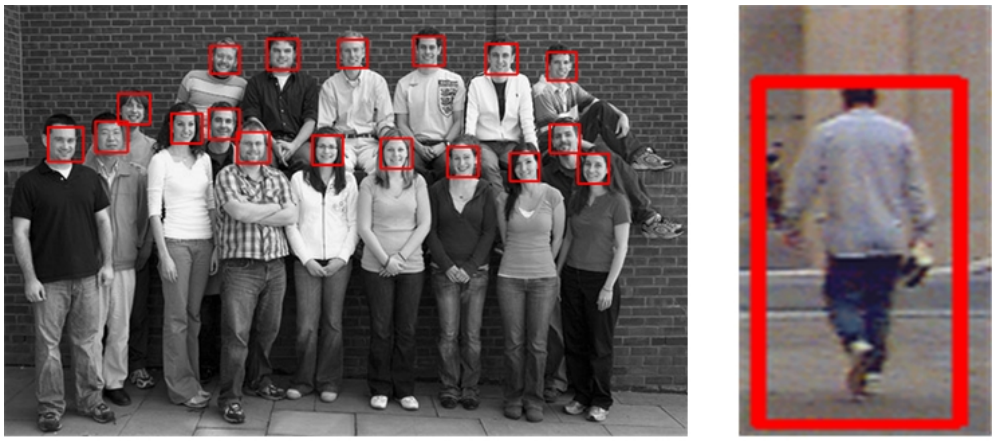

Figure 3.8: Open CV Haar cascade applied to detect faces and pedestrians

$$h_t(x) = \begin{cases} left\_val & if \;\; f_t(x) < threshold \\ right\_val & otherwise. \end{cases} \tag{3.4}$$

To compensate for illumination changes in images Viola *et.al.* use variance normalization on each sub-window *i.e.* they subtract the sub-window mean from each pixel and divide each pixel by the sub-window standard deviation. The standard deviation of an image sub-window can be computed by using a integral and square integral image as shown in Equation 3.5.

$$\sigma^2 = m^2 - \frac{1}{N}\sum x^2 \tag{3.5}$$

where σ is the standard deviation, m is the mean and x is the pixel value within the sub-window. The mean of the sub-window can be computed by using the integral image. The sum of squared pixels can be computed by using an integral image of a squared sub-window pixels. Normalization can be achieved by post multiplying the feature value with $\frac{1}{\sigma}$ rather than operating on individual pixels. Note that subtracting the mean from individual pixels has no significance on the feature value.

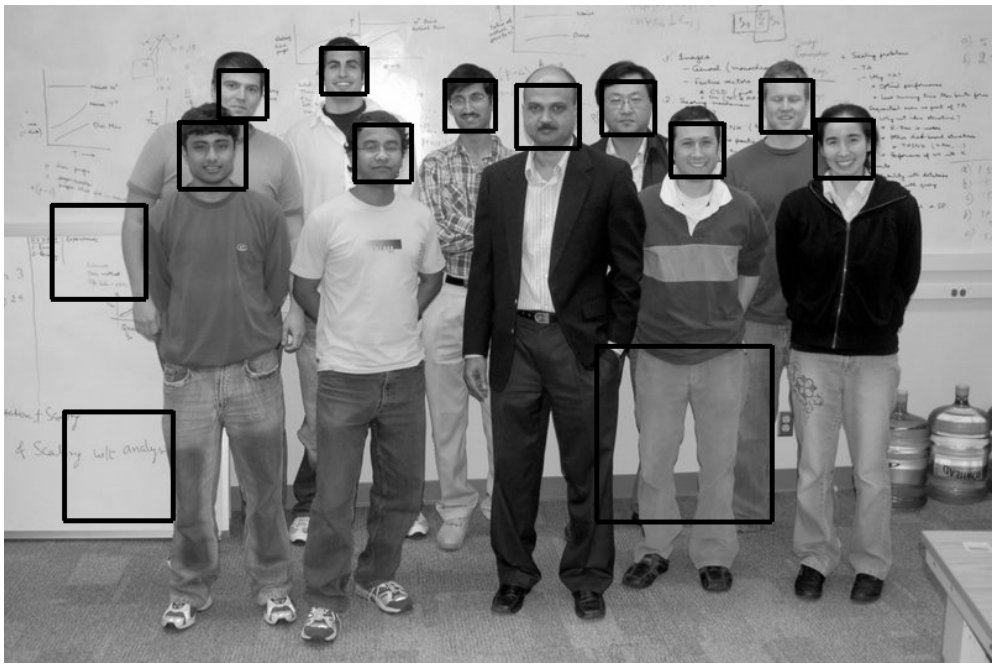

Figure 3.9: Open CV Haar cascade applied to detect faces

3.6 Conclusion

The use of Haar features and the adaboost based training process made fast face detection practical. One of the primary advantages of this algorithm is the high speed and detection accuracy compared to previous face detection techniques. Also the fact that this technique can be adapted to any detection task makes it suitable for generic object detection in surveillance environments. It is increasingly becoming important to detect objects in real-time in surveillance systems and a hardware architecture for adaboost which can be used in such a system would be highly attractive. Chapter 4 focuses on existing adaboost hardware architectures and the FE based architecture proposed in this thesis.

```
<!-- root node -->
      <feature>
       <rects>
        <_>4 0 7 6 -1.</_>
        <_>4 3 7 3 2.</_></rects>
       <tilted>0</tilted></feature>
      <threshold>5.9739998541772366e-003</threshold>
      <left_val>-0.8590919971466065</left_val>
      <right_val>0.8525559902191162</right_val></_></_></trees>
  <stage_threshold>-5.0425500869750977</stage_threshold>
```

i

```
   <rects>
    <_>5 6 8 10 -1.</_>
    <_>5 6 4 5 2.</_>
    <_>9 11 4 5 2.</_></rects>
   <tilted>0</tilted></feature>
  <threshold>-0.0129300002008677</threshold>
  <left_val>0.4075860083103180</left_val>
  <right_val>-0.5123450160026550</right_val></_></_>
```

ii

Figure 3.10: Open CV Feature Database
Open CV (2008)

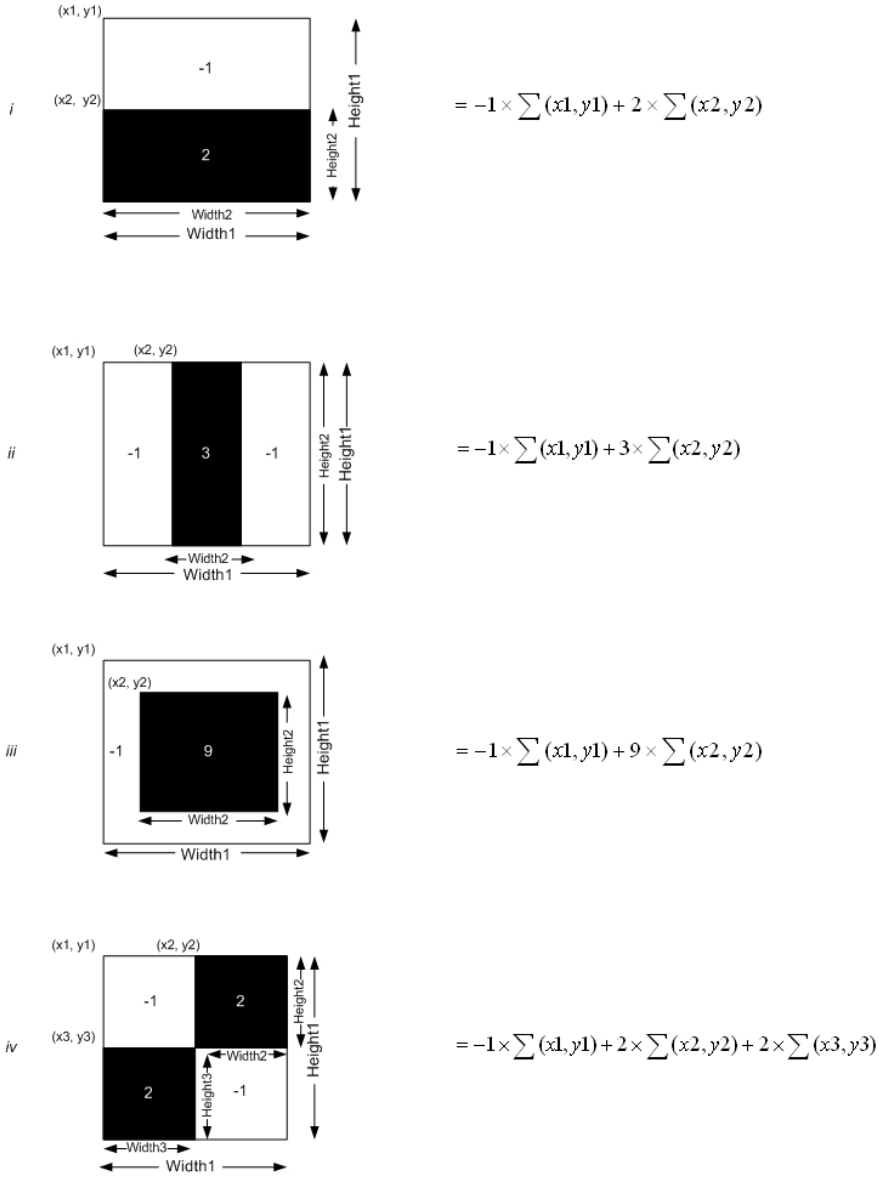

Figure 3.11: Open CV Haar Basis Representation - *where* $\Sigma(x_n, y_n)$ *denotes sum of pixels in the rectangular region covered by* (x_n, y_n)*, width and height* Open CV (2008)

Chapter 4

Feature Element For Adaboost Object Detection

4.1 Introduction

Computer Vision over the last few years has seen tremendous advancement in terms of algorithmic development and number of applications. This phenomenal development can be largely attributed to the availability of powerful computing platforms. However real time execution of computer vision algorithms can only be achieved by employing parallel execution of specific image processing tasks. For instance the object detection system proposed in this work when simulated in software takes around 300 ms to process a 640 $\times$ 480 size image on a Pentium Core 2 Duo 2.4GHz PC, which is approximately 3.5 frames per second. The key bottle neck in the system apart from the post processing blocks are the MoG and the Haar feature computation. The MoG is computationally intensive due to the fact that, it maintains 3 background models and each incoming pixel is compared to these background models to find a match. The background update process also adds complexity to the system. On the other hand the Haar feature computation occupies a significant amount of the execution time of object detection due to the fact that, to localize faces in a sub-window of size N$\times$N, the Haar features are applied to each sub-window of size 24$\times$24 within the image. This process is repeated multiple times, and each time the sub-window is scaled down to detect

faces of different sizes. As mentioned in the previous chapter there are nearly 30 stages with each stage containing anywhere between 9-200 features. The total number of sub-windows in the image is given by $\frac{N}{24} \times \frac{N}{24}$ and the computational complexity increases with the sub-window size and the target object size. Due to the complexity of the detection process it is hard to achieve real-time detection rates in embedded platforms such as RISC and DSP. A major problem when deploying complex vision algorithms in embedded smart cameras is the lack of necessary computational power.

The most effective way to achieve real-time performance is by accelerating some of the compute intensive blocks in the system. A key advantage with hardware acceleration is that we can exploit inherent parallelism in these sub-blocks thereby achieving higher detection performance. The most suitable hardware platform for vision application is the Field Programmable Gate Array (FPGA) as they are both fine grained (more flexible) and reconfigurable (adaptive) as mentioned in MacLean (2005) and McBader and Lee (2003). FPGAs are used in many compute intensive areas such as encryption/decryption, image processing and neural networks Anderson et al. (2005), Ratha and Jain (1999), Nair et al. (2005), and Zhang et al. (2007) . Other useful feature of FPGAs is the ability to customize the data path width depending on the required precision. In Baumgartner et al. (2007) performance benchmarks of DSP and FPGA implementation of three low level vision algorithms 1) Gaussian Pyramids 2) Bays filtering 3) Sobel edge detector are presented. Subsequently it is stated that DSPs perform faster than FPGAs however on a sub-function basis. From a system level perspective an FPGA is more advantageous due to the ability to exploit parallelism and the availability of hardware resources.

4.2 MoG Hardware Architecture

The hardware architecture for the MoG approach comprises of three modules i) Match Model ii) Update Model and iii) Sort Model. As described in Section 2.2.1 the match model determines if each incoming pixel is part of the background. Once a match is determined the update module updates the μ, σ and w of the model. If no match is found then the least probable model or the model with

lowest weight is replaced with the current pixel value, a high variance and a low initial weight. The final stage sorts the models according to their weights. Unlike Stauffer and Grimson (1999) where models are sorted according to their $\frac{w}{\sigma}$ ratio, we only use weights to sort the models in order to reduce computational complexity. Furthermore, we use 16 bits to represent the weights and variances, and the mean of each model is represented with 8 bits. Based on simulation results sixteen bits are sufficient to represent weight and variance of each model. Moreover the mean here denotes pixel mean and so we use eight bits to represent mean of each model. A learning rate of 0.01 is used to update each model's parameters as mentioned in Section 2.3. Figure 4.1 shows the block diagram of the MoG architecture. Inputs to the block are pixel values and their corresponding means, variances and weights. The proposed system maintains 3 models of which the first two models represent the background and the third models the time varying foreground. Incoming pixels that match either of the first two models are considered a part of background, otherwise they are in the foreground. As mentioned earlier in Section 2.3 the choice of number of background models is a trade off between required segmentation accuracy and computational complexity.

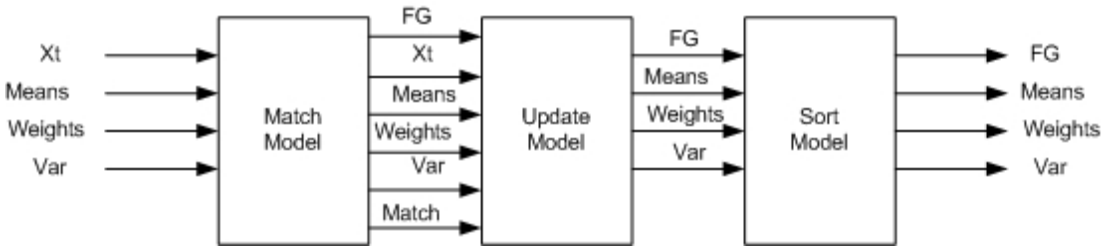

Figure 4.1: Block Diagram of Mixture of Gaussian

Figures 4.2, 4.3, 4.4, 4.5 and 4.6 show the hardware architecture we implemented for various blocks within the MoG algorithm. The match model block starts by computing the squared euclidean between each incoming pixel and the pixels of three different background models. A match is found by comparing the squared euclidean with the variance of each model. If the euclidean distance is less than three times the model's variance it is considered as a match. The matched background model's mean weight and variance are updated in a way that the system adapts to the changes that occur in the background. A multiplexer

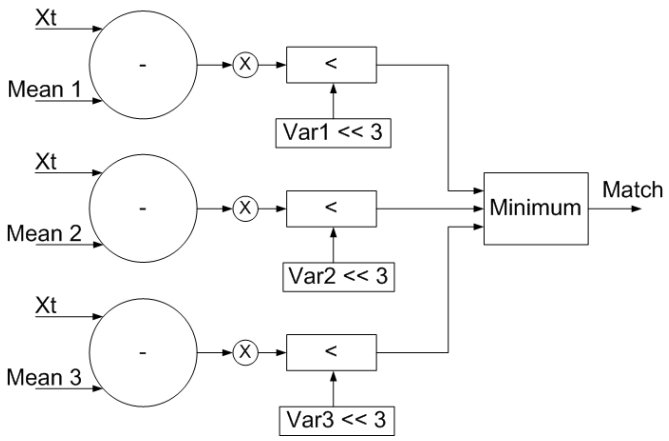

Figure 4.2: Internal architecture of Match Model block

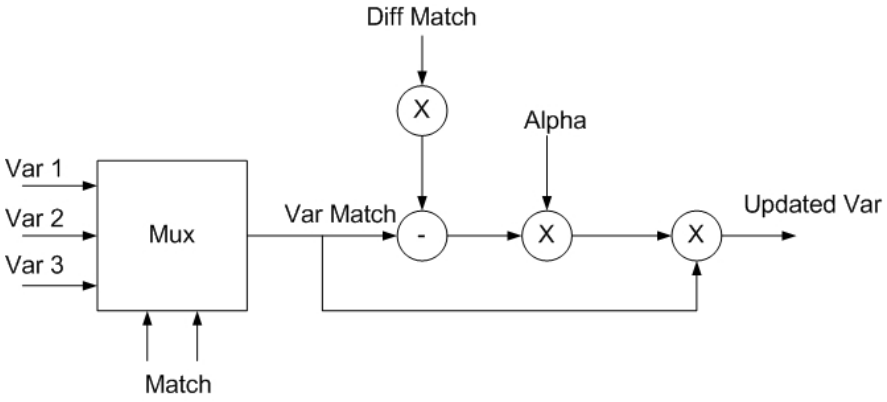

Figure 4.3: Variance update block

is used to select one of the three model's parameters based on the match. The update rate is controlled by the learning rate α. The final block sorts individual models so that the most probable background gets higher weight.

The match model block shown in figure 4.2 accepts an incoming pixel value along with mean pixel values of three background models that we maintain in this work. Square of the difference between incoming pixel and the means are compared against variance of each model (Equation 2.2). If this difference is greater than three times the variance of each model we consider it as a match. Output of this block is either "00", "01", "10" or "11". A value "11" signifies no match and values "00", "01", "10" signify a match corresponding to the respective

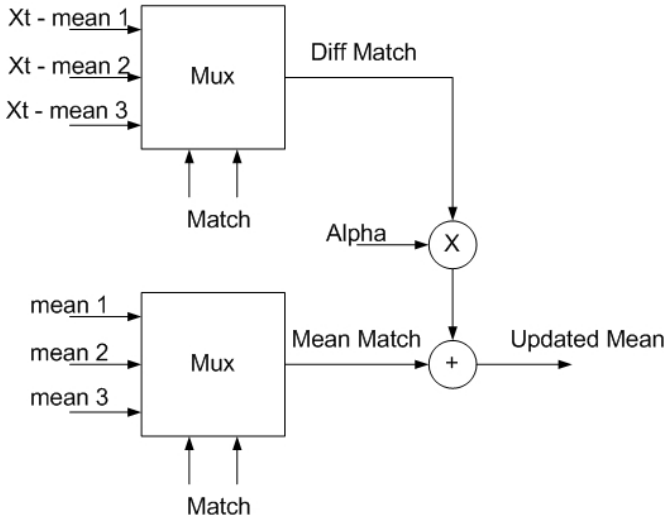

Figure 4.4: Internal architecture of Mean update process

model. Once a match is found the variance update block and mean update block shown in Figure 4.3 and Figure 4.4 update the respective mean and variance according to Equation 2.2. A multiplexer (mux) is used to select appropriate model's parameters for the update process and the mux is controlled by the output from the match model block. If no match is found the mean and variance of the last model are replaced with the incoming pixel value and a high variance (not shown here) respectively. Similarly the weight update block takes weights of three background models and a mux is used to select matched model weight. If no match is found the last model's weight is replaced with a small weight. Finally sort model block shown in Figure 4.6 is used to order the background models according to their weights. Input to this block are weights of three background models. Weights of individual models are compared and the result of comparison is used to swap respective model's mean and variance parameters.

The MoG hardware interfaces with a host processor with a Open Core Protocol (OCP) OcpIP (2006) Bus to load image data along with the background models. A set of control and configuration registers are used to control the operation of the module, details of which are presented in Chapter 5.

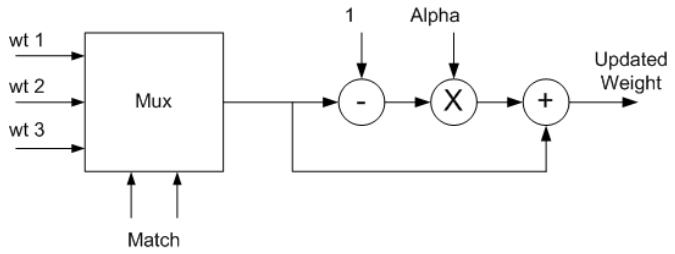

Figure 4.5: Internal architecture of weight update process

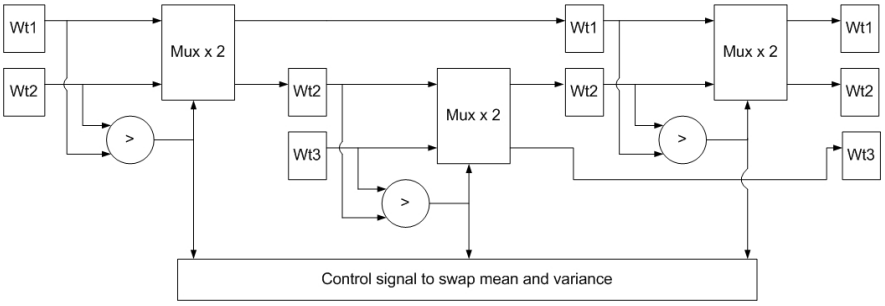

Figure 4.6: Internal architecture of Sort Model block

4.3 Boosting Hardware Architecture

A number of hardware architectures have been proposed for boosting based face detection. The majority of the architectures discussed in this section are hard-wired for a specific detection task (e.g. faces), i.e. the data access patterns for computing feature values if fixed. The type and number of features used for detecting an object in a sub-window is fixed for a specific object type and is decided during the training phase. To detect a different object a different feature/threshold set needs to be used and note that the number of features per stage may also vary.

The authors in Hiromoto et al. (2009) use serial processing for features in later stages but use parallel processing for initial stages as the number of sub-windows are huge for initial stages. Also the fact that the number of features used in initial stages is relatively small (Figure 4.7). The architecture is targeted for full frame search and is implemented on a Xilinx Virtex-5 FPGA (XC5VLX330-2). The design runs at 160MHz and achieves a frame rate of 30fps for 640×480 image. Figure 4.9 shows the block diagram of the architecture and Figure 4.8

shows feature computations for each sub-window. The sub-window is stored in a register array and the number of bits used for an integral image is 18 bits.

This architecture mainly focuses on real-time execution of full frame search localization. The main advantage of the cascade classifier is that majority of negative sub-windows are rejected by the initial stages. Consequently the number of sub-windows processed by later stages is far less compared to those processed by initial stages. So to attain faster performance this architecture uses parallel processing for the initial stages. In figure 4.9 the scaler and integral image generator, scale the incoming image and compute integral image respectively. The squared integral image block computes the square of each pixel and then computes the integral image. The squared integral image is used to normalize the image as explained in Section 3.5.

Stage number n Weak classifeirs	0 9	1 16	2 27	3 32	4 52	5 53	6 62
7 72	8 83	9 91	10 99	11 115	12 127	13 135	
14 136	15 137	16 159	17 155	18 169	19 196	20 197	
21 181	22 199	23 211	24 200	Total 2913			

Figure 4.7: Number of Stages for Typical Face Detection
Hiromoto et al. (2009)

The work presented in Gao and Lu (2008) has an approach similar to the architecture proposed in this thesis to compute the weak classifier function. A classifier engine forms the core for this architecture similar to a classifier stage discussed in section 4.4.2. The classifier engine comprises of 16 weak classifier functions that are computed in parallel on a FPGA. In order to effectively use the classifier engine Changjin Gao *et. al.* re-trained the Haar classifier network to use a multiple of 16 Haar features per stage. The re-training resulted in a 40

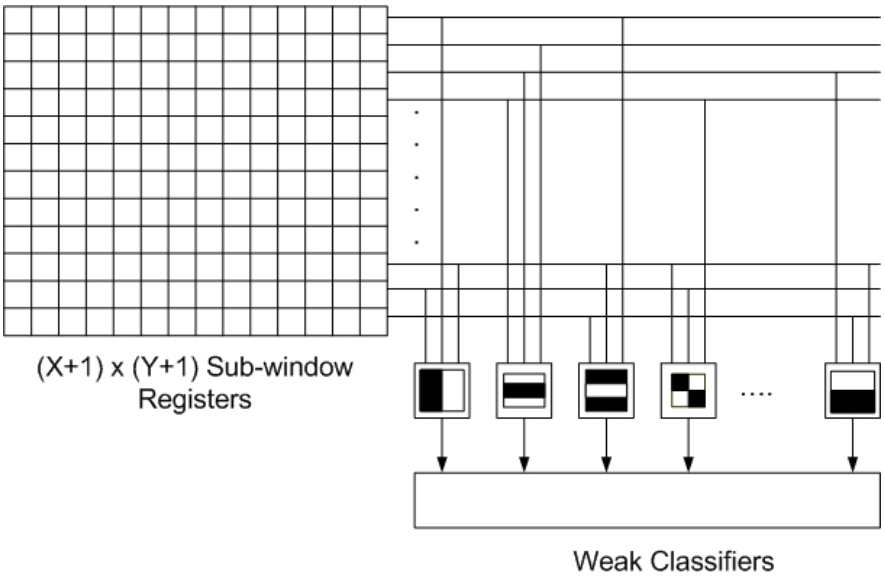

Figure 4.8: Haar-like feature evaluations
Hiromoto et al. (2009)

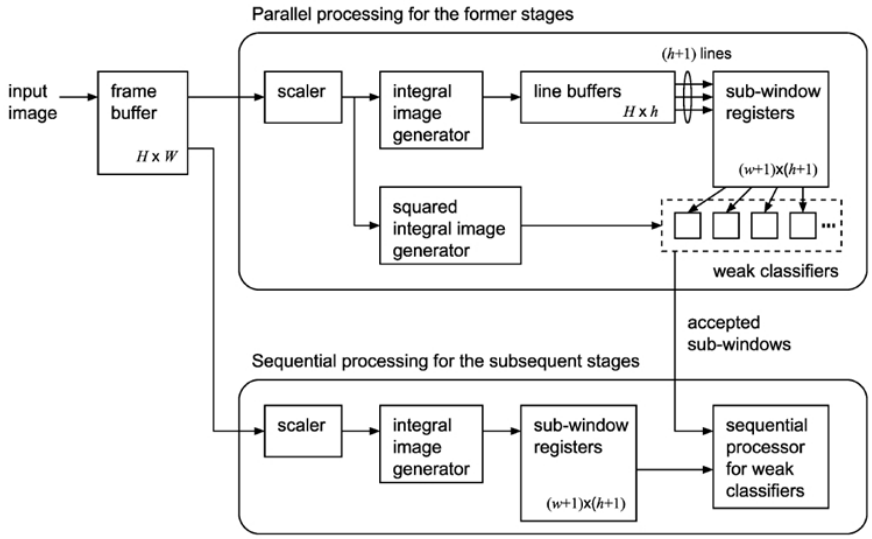

Figure 4.9: Block Diagram of Face Detection
Hiromoto et al. (2009)

stage classifier instead of a 22 stage classifier (Open CV) for frontal faces. The main advantage of using 16 features in the initial stage instead of fewer features is that more than 90% of non-face windows are dropped compared to only 50% in

the case of 22 stage classifier. Changjin Gao *et. al.* also show that the 40 stage classifier has a better detection rate and low false positive rate compared to a 22 stage classifier. Also the authors propose a more aggressive resource consuming classifier engine where classifier parameters are loaded in parallel to achieve higher frame rates. Figure 4.10 shows the aggressive classifier engine. Gao *at.al.* design the classifier function with reuse in mind and to detect a different object the design files need to be modified with new parameters and resynthesized, however manual editing of VHDL design files is a long and difficult task. A key advantage with the work proposed in this thesis is the FE based stage can be configured with parameters online thus saving valuable design modification and synthesis time.

Another architecture Lai et al. (2008) shown in Figure 4.11 used Network-On-Chip architecture to establish communication between different computational blocks. Various blocks used are: memory, image down scaling, integral image computation, feature database and classifier. This when implemented on a Xilinx Virtex II FPGA achieves a frame rate of 40fps for 320×240 image.

4.4 Feature Element based Architecture

In a surveillance system a full frame processing for face detection (or object detection) is not always necessary. The most important thing in such a system is the ability to detect various objects (e.g. face, car, person, animal etc.) with low complexity hardware. More importantly the hardware should be flexible in the sense that it should be reconfigurable to accomplish many detection tasks as the amount of hardware resources are highly constrained on a embedded device. The proposed configurable feature element based architecture makes it easy to reprogram the detection network for various detection tasks. With this approach it is not necessary to design hardware blocks for detecting different objects in effect saving massive effort needed in the design cycle.

The Haar classification function that Viola *et. al* use occupies more than 90% of computation time Gao and Lu (2008). To achieve real-time detection performance it is a good idea to accelerate feature computations in hardware. In this thesis we propose a generic feature element (FE) that can be configured

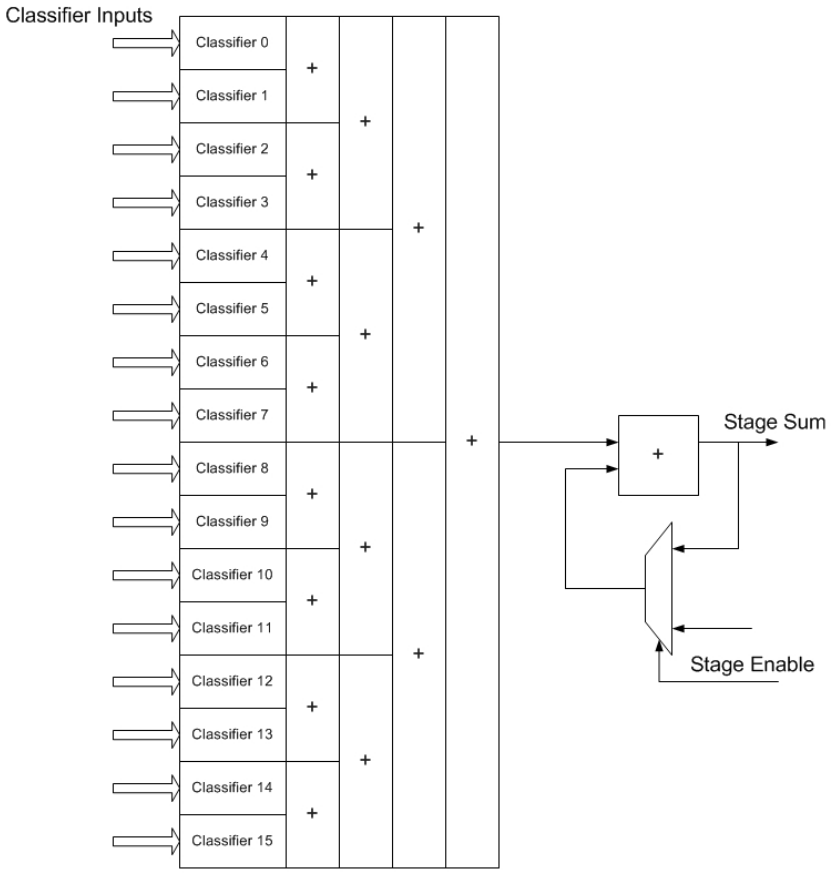

Figure 4.10: Classifier Engine using 16 Classifiers
Gao and Lu (2008)

to form any feature from the basis set as shown in Figure 3.11. The FEs are designed in a way that multiple FEs can be grouped to form a stage. Subsequent sections detail our FE and the operation of a stage formed by grouping 16 FEs, as motivated by Changjin Gao *et. al.* who also used 16.

4.4.1 Hardware architecture

A typical FE as shown in figure 4.12, comprises of a set of data and configuration registers and a computation part. There are a total of 16 registers that are used to load pixel data corresponding to each vertex. Also classifier parameters such

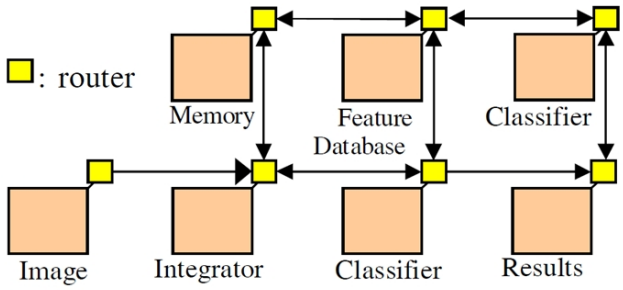

Figure 4.11: Network-On-Chip Based Face Detection
Lai et al. (2008)

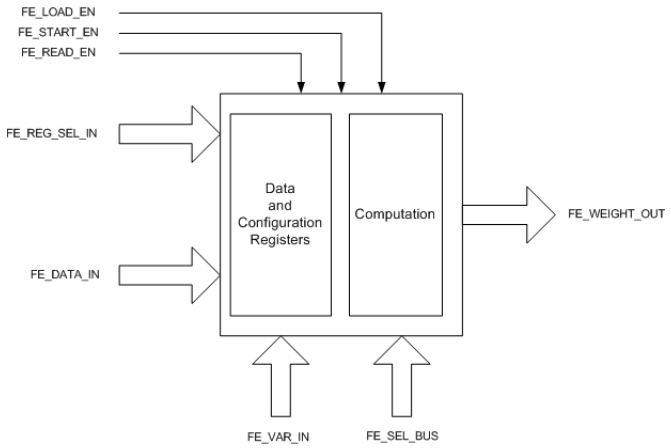

Figure 4.12: Proposed Feature Element

as *Threshold, Left_ val and Right_ val* are loaded into each feature element (FE). Figure 4.13 shows the configuration register in each FE. A basis function register controls the number of vertices used in a FE and also the weight assigned to the vertices 2 and 3. For instance a basis type 00 uses only two vertices and weights the second vertex by 2, basis type 01 also uses two vertices and weights the second vertex by 3, basis type 10 uses two vertices and weights second vertex by 9 and finally basis type 11 uses three vertices and weights second and third vertex by 2. Vertex one always uses a weight of -1. See Figure 3.11 for weights of feature rectangles. In our FE weights are applied by simple shift and add operations.

The top level classifier stage detailed in Section 4.4.2 controls loading of data and parameters into each FE. Signal FE_SEL_BUS is a 4-bit bus used to enable

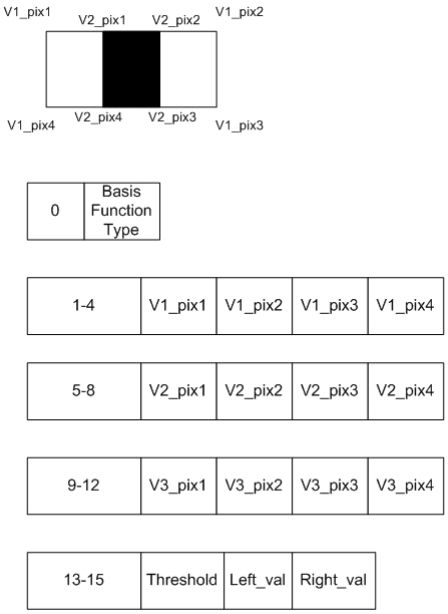

Figure 4.13: Configuration Register with pixel address in clock wise direction

a specific FE from a group of FEs in the stage. FE_REG_SEL_IN is also a 4-bit bus used to select configuration registers when loading data using FE_DATA_IN. Control signals FE_LOAD_EN, FE_START_EN and FE_READ_EN are used to load, to start computation and to read classifier weight respectively. Instead of post multiplying the inverse of each sub-window standard deviation to normalize the feature value as explained in section 3.5, we multiply standard deviation with the threshold value before comparison. The computation part typically comprises of arithmetic addition, subtraction, and a compare operation to determine the classifier output. The output of each FE is a classification weight. Figure 4.14 shows the internal architecture of each FE.

4.4.2 Grouping multiple FEs

Figure 4.16 shows a typical stage formed by grouping 16 FEs. The stage interfaces with a host processor with a Open Core Protocol (OCP) OcpIP (2006) Bus to load integral image and configuration data. To enable detection on various sub-window sizes we use a maximum of 1024×18 RAM to store sub-window

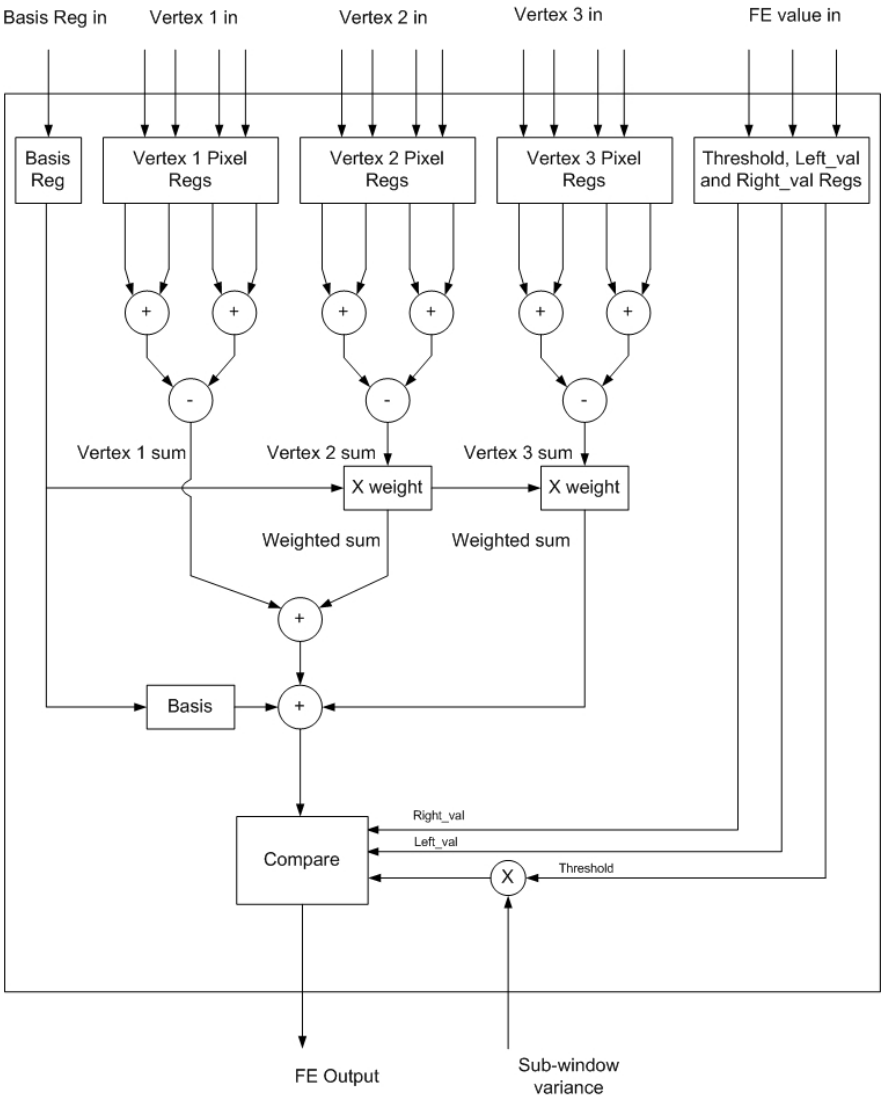

Figure 4.14: Internal Architecture of FE

pixels. We use 18 bits to represent integral image pixels. We use eighteen bits because of the fact that the maximum sub-window size in Open CV pre-trained database is 24 × 24 which is for frontal faces apart from other databases for upper body, full body etc. To represent a integral sub-window of size 24 × 24 we need 18 bits per pixel[1]. Moreover with 18 bits per-pixel we can represent integral image sub-windows of any size with a maximum size of 32 × 32 to enable generic

[1] $24 \times 24 \times 255 = 146880$ requires 18 bits

object detection. Apart from this a configuration RAM holds individual FE's configuration data. The configuration data RAM is organized as 256 × 16, note that each FE has 16 configurable parameters and hence 256 locations store configuration data for all 16 FEs in a stage. Figure 4.15 shows example configuration data for a FE. Locations 1-12 store pixel addresses for individual vertices of each feature. These pixel addresses are used to load pixel data from sub-window memory when loading a FE. A multiplexer is used to switch between configuration data and pixel data, note that basis type, threshold, left value and right value are loaded from configuration memory and pixel data from sub-window memory. An address generation unit generates addresses for configuration RAM, data RAM and the address of each FE that is being configured. A control and configuration unit generates the necessary control signals to enable both RAMs and individual FEs. A set of control registers that can be written to and read from the OCP bus are used to control the stage. Details of OCP interface, control registers and the stage validation process are detailed in Chapter 5.

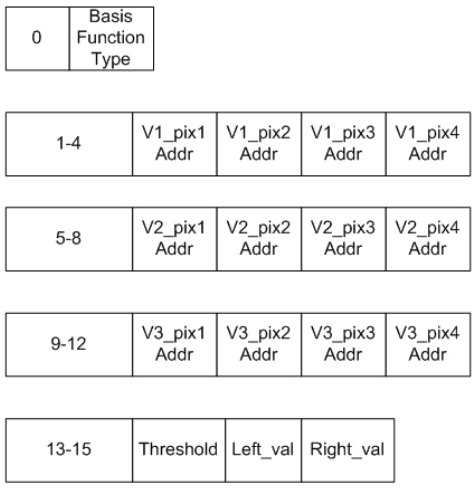

Figure 4.15: Configuration RAM Data

All FEs are configured sequentially and once a FE is fully configured, computation inside the FE is started by FE_START_EN signal. After all the FEs are configured we start the process of reading classification weights from each FE. A FE_READ_EN is used to enable reading from each FE. Classification weights

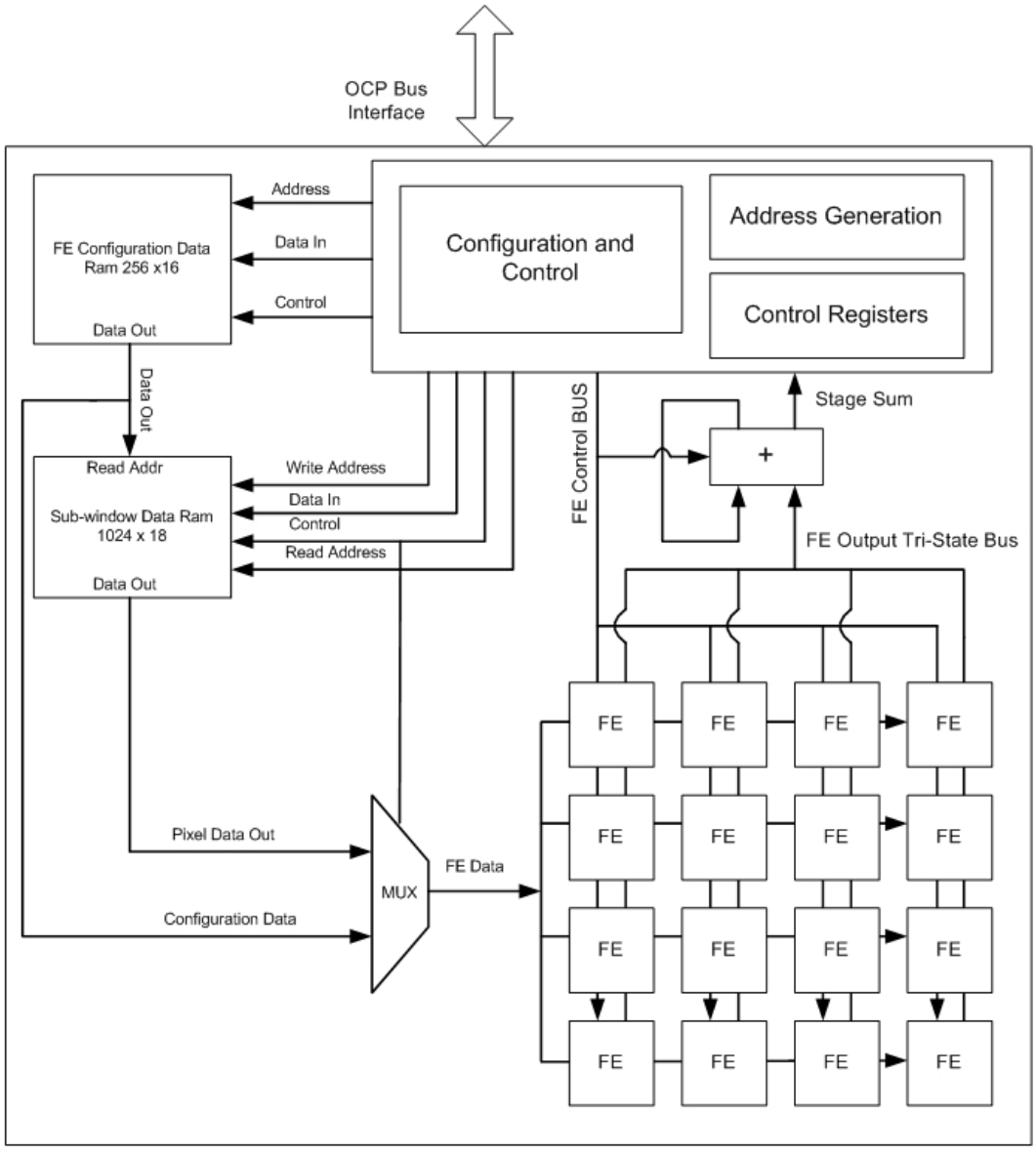

Figure 4.16: A Reconfigurable Stage formed by using 16 FEs

from all FEs are added inside the stage and are read by the host processor and compared against a stage threshold as described in Section 3.5.

Open CV uses floating point numbers to represent threshold, left value and right value. To reduce computational complexity in our design we use fixed point to represent these values. The choice of precision for fixed point representation for these values comes from Open CV pre-trained databases for frontal face, profile face and upper body. The maximum value for left and right values is between ± 7, so a precision of 4.12 representation is used for left value and right value. That is four bit signed number to represent integer part and twelve bits to represent the fraction part. The feature threshold is normalized and is always less than ± 1, so it is represented as 1.15 where 1 is used for sign and 15 bits are used to represent the fraction part. One major advantage with fixed point representation as opposed to floating point representation is the reduction in hardware complexity. Moreover the choice of precision is a trade of between detection accuracy, speed and hardware complexity. The choice to use 16 bits to represent the floating point parameters comes from the fact that this work targets an embedded devices with limited computational resources. To attain real-time speed it is essential to reduce computational complexity of the algorithm.

4.5 Conclusion

In this chapter we present details of the hardware architecture designed to accelerate the key compute intensive blocks in the proposed object detection system. Accelerating Haar feature computation plays an important role in achieving real-time detection performance. The Haar feature computation is accelerated with the help of a Feature Element. The main advantage of using a Feature Element based architecture is that we can dynamically load the configuration data along with the image sub-window for various object detection tasks without actually redesigning the underlying hardware, taking effective advantage of parallelism in the detection algorithm. The number of FEs in each stage is limited to 16 in this work but could be extended to higher values in the future.

Chapter 5

Evaluation and Results

5.1 Introduction

So far in this thesis we have introduced the reader to the importance of object detection and the concept of a smart camera. An important challenge when using object detection techniques is that, they are highly compute intensive and cannot achieve real-time performance in a resource constrained device such as an embedded smart camera. We present details of one of the better performing face detection algorithm that is the adaboost based face detection proposed by Viola et. al. Some of the key reasons for this algorithms popularity are that it is faster than the majority of face detection techniques and the fact that it can be trained to detect any object. This algorithm uses full frame search to detect objects of interest and thus takes more time to execute, however in a surveillance system it is not always necessary to perform a full frame search. A significant reduction in execution time can be achieved by only processing motion regions or image regions of interest to detect target objects. We propose to use background modeling, a popular moving object segmentation technique, as a pre-processing step to extract image regions of interest. These image sub-windows are later passed to the adaboost detector to detect objects of interest.

The primary goals of this thesis are two fold. One to speed-up the adaboost detection performance by extracting the motion regions using a background modeling technique. The second is to build hardware acceleration architectures for key

compute intensive blocks, that is, the background modeling and Haar feature computation. To facilitate generic object detection we implemented a configurable feature element and a configurable stage by grouping 16 FEs. A configurable stage is the core of our architecture and we use pre-trained frontal face and upper body databases from OpenCV in order to test it. Note that features in Open CV database are organized as 2-vertex and 3-vertex features. Our FE is designed around this which enables us to load four corner integral pixels represented by these vertices. The key components of the proposed system are:

i Build a model of the background.

ii Extract foreground pixels.

iii Post-processing to remove noise and small objects.

iv Send the segmented region to object detection.

  a Load Open CV Haar database for target object.

  b Compute integral image of the sub-window.

  c Compute square integral image.

  d Compute variance (see equation 3.5).

  e Apply features to sub-window.

  f Repeat steps (b) to (e) at different scales.

v Proceed to next frame and repeat steps (ii) to (v).

We implemented a software model of the proposed system in C in order to verify its operation. The rest of this chapter details important performance metrics such as the detection speed and accuracy for various test sequences. Following this we present hardware implementation details of both MoG and the Haar feature acceleration.

5.2 Performance Evaluation

The detection performance and speed of our implemented system is tested with 300 temporal images from two different test sequences for different objects (face and upper body). The test sequences used are corridor test sequence from Kelly (2008) and CAVIAR test sequence from CAVIAR (2000) with image resolutions 640×480 and 384×288 respectively. Manually hand segmented ground truth is generated for both test sequences for 300 frames. With hand segmented ground truth we segmented a total of 120 face regions and 393 upper body regions in the corridor test sequence and a total of 133 upper body regions in the CAVIAR test sequence. We use the Open CV implementation of Haar detection to test the system. The Open CV default sub-window size of 24×24 for frontal faces and 22×18 for upper body is used to apply the Haar features. The system's performance is evaluated for both speed and accuracy. The detection accuracy is measured with precision and recall values as given in Equations 5.1 and 5.2. The system is executed on a Pentium Core 2 Duo 2.4 GHz machine with 4 Giga Bytes RAM running Windows XP. Figure 5.1, 5.2 and 5.3 show typical detection results. Images in the ground truth row indicate hand annotated target image regions in a bounding box, similarly images in rows labelled Open CV full search and motion segmentation and detection indicate automatically detected target image regions with full search and the approach proposed in this thesis respectively.

$$Precision = \frac{\#No.\ of\ Correctly\ Detected}{\#No.\ Total\ Detected} \tag{5.1}$$

$$Recall = \frac{\#No.\ Correctly\ Detected}{\#No.\ Ground\ Truth} \tag{5.2}$$

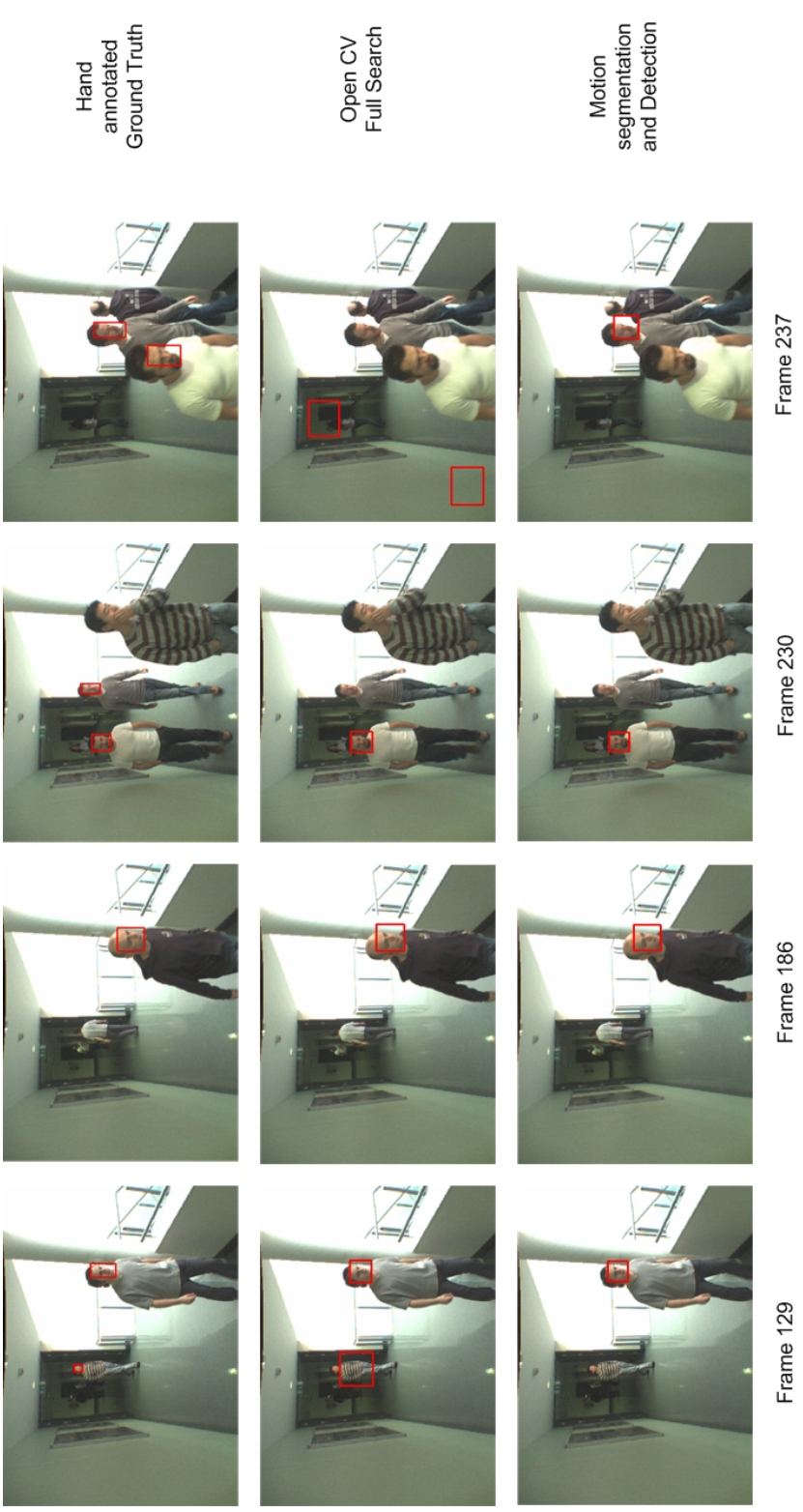

Figure 5.1: Face detection applied to Corridor test sequence- Haar feature window size 24×24 and image resolution 640×480

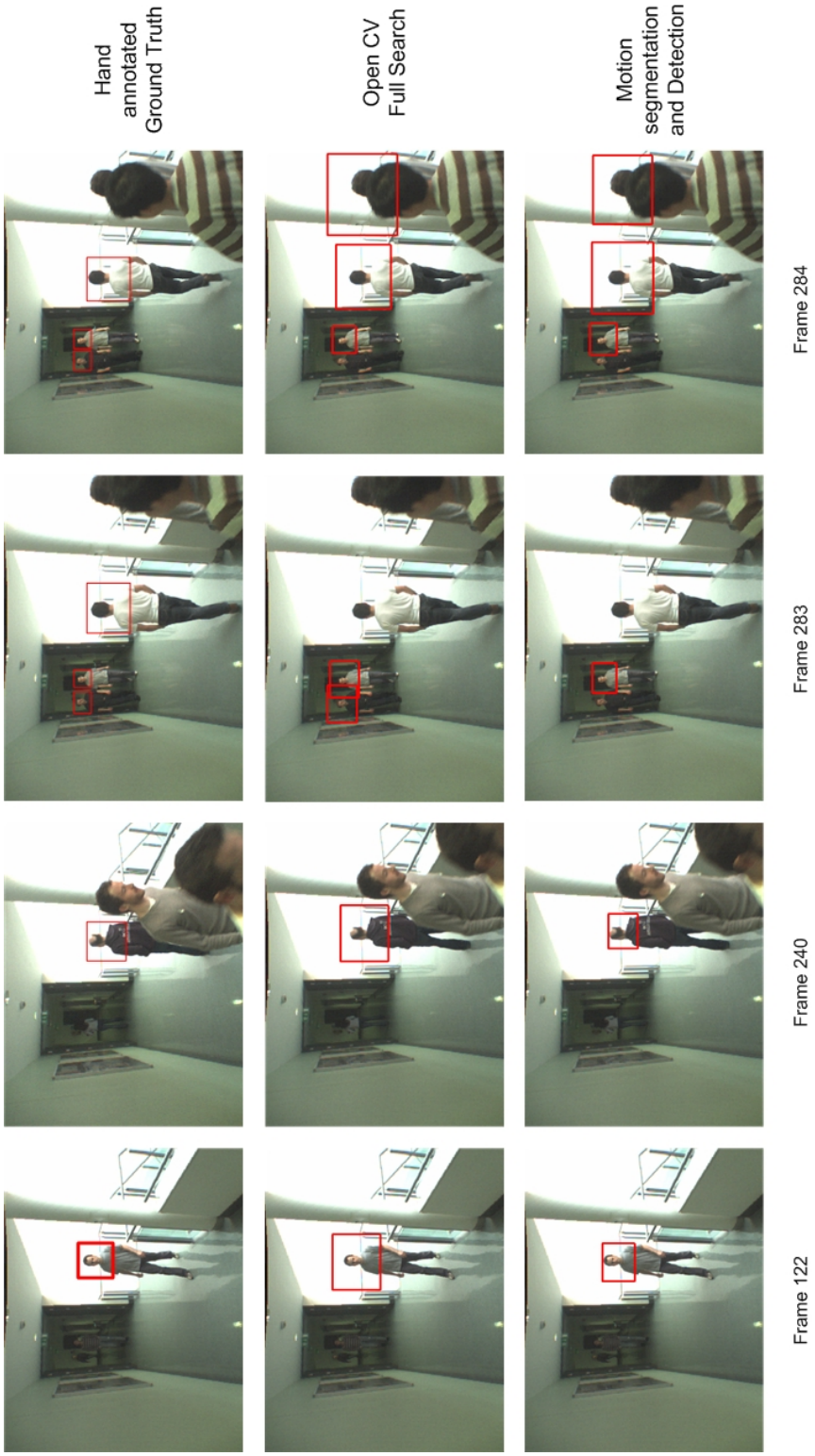

Figure 5.2: Upper body detection applied to Corridor test sequence - Haar feature window size 22×18

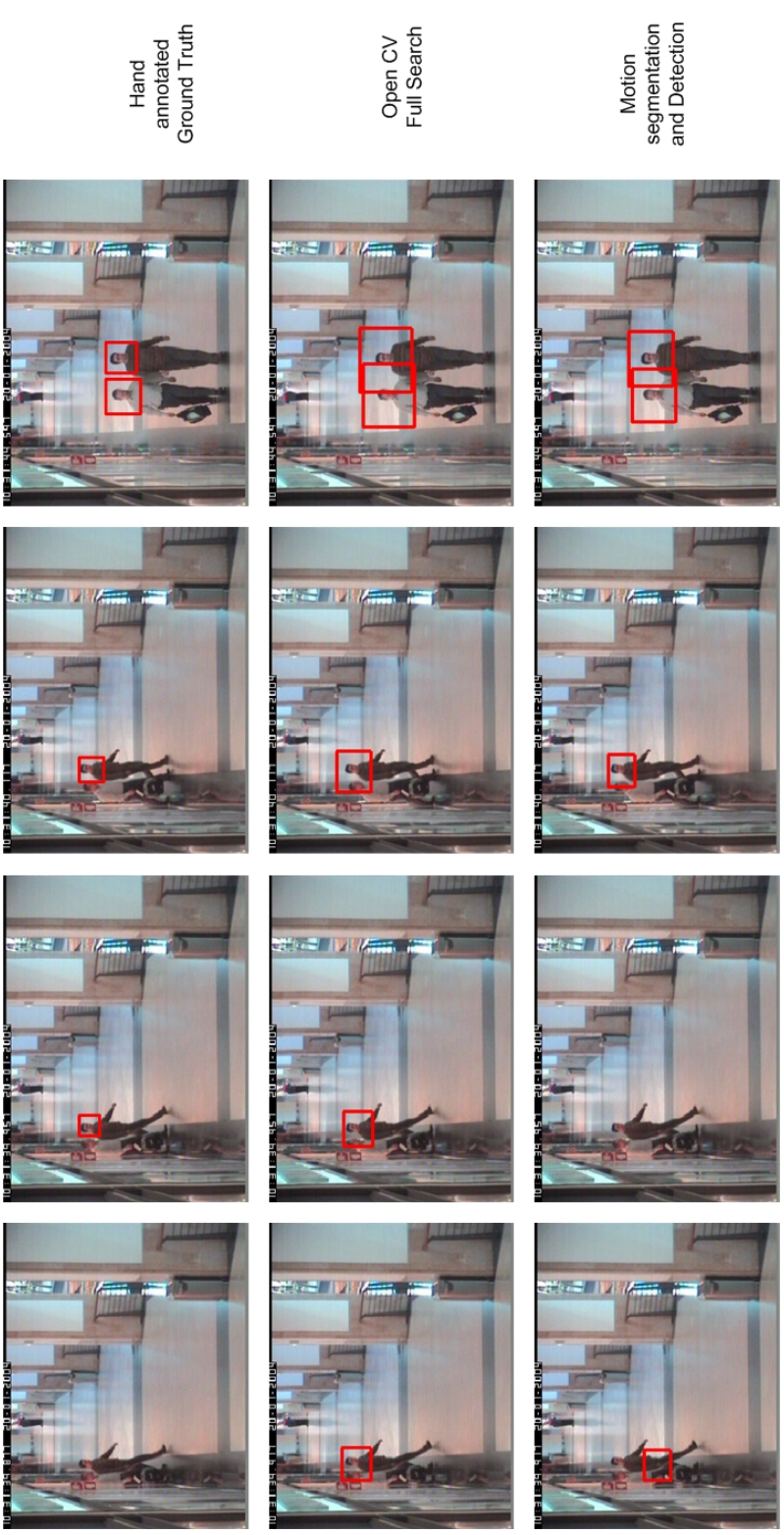

Figure 5.3: Upper body detection applied to CAVIAR test sequence - Haar feature window size 22×18 and image resolution 384×288

Test Sequence 300 frames	Ground Truth	True Positive		False Positive		Frame Rate - fps		Precision/Recall	
		FS	MS	FS	MS	FS	MS	FS	MS
Corridor- Face	120	57	41	48	2	0.9	2.5	0.54/0.48	0.95/0.34
Corridor - Upper body	393	144	44	133	15	0.7	2.0	0.51/0.37	0.74/0.11
CAVIAR- Upper body	133	133	45	42	1	1.1	4.2	0.76/1.0	0.98/0.34

Table 5.1: Object detection performance. FS = Full Search detection, MS = Background suppression and detection

5.2.1 Detection Accuracy

Table 5.1 and 5.2 show typical performance metrics for two test sequences. The ground truth column indicates number of hand segmented object regions that contain faces and upper body extracted from 300 frames from both the test sequences. The columns true positive, false positive and precision/recall indicate detection accuracy with both Open CV based full search (FS) and background suppression based motion segmentation (MS) implemented as a part of this work. The column frame rate compares the detection speed of both approaches.

The background suppression based detection achieves relatively higher detection speed and higher precision compared to Open CV full search as shown in Table 5.1. The increase in speed is mainly due to processing only image regions of interest segmented by the background modeling technique instead of the full frame. Similarly, for both corridor and CAVIAR test sequences there is an increase in the precision because of the fact that with the proposed motion segmentation based approach we process only fewer image regions instead of the full frame.

With the proposed approach there is a reduction in the recall values achieved for both test sequences. In the case of the corridor test sequence the reduction in recall values is slightly less compared to the CAVIAR test sequence. This is due to the fact that objects in the corridor sequence are relatively fast moving compared to objects in the CAVIAR test sequence. Inherent drawbacks within the background suppression techniques as mentioned in Chapter 2 make it difficult to detect homogeneously colored objects and slow moving objects. The main limitations in building a robust background model such as aperture, camouflage, sleeping person etc make it hard to generate precision foreground. Mainly with

Test Sequence 300 frames	Ground Truth	True Positive		False Positive		Precision/Recall	
		FS	MS	FS	MS	FS	MS
Corridor- Face	68	57	41	48	2	0.54/0.84	0.95/0.60
Corridor - Upper body	250	144	44	133	15	0.51/0.60	0.74/0.20
CAVIAR- Upper body	133	133	45	42	1	0.76/1.0	0.98/0.34

Table 5.2: Detection accuracy after excluding small objects

the MoG approach used in this thesis slow moving objects and objects that are homogeneously colored are hard to detect because the update process pushes these objects into the background. The detection accuracy greatly relies on segmented motion regions or regions of interest to achieve higher recall values. Another important reason for reduced recall values is that, the hand segmented ground truth in Table 5.1 deliberately considers all objects (faces and upper body) in the sequence even if they are too small or too far away from the camera to be detected by our approach. The motivation is to simulate ideal operation in the ground truth. Table 5.2 presents detection accuracy after excluding objects that are too small for both object detection and background suppression technique to identify.

In table 5.2 there is a significant improvement in recall values for faces in corridor test sequence after removing faces that are too small to detect. In the case of upper body in corridor test sequence the recall values show a small improvement. Unlike faces when we consider upper body we focus on a relatively large and homogeneously colored regions that are difficult to segment due to the inherent limitations in background modeling as mentioned earlier. Moreover we use the Open CV default database to evaluate the system. Better detection accuracy could possibly be obtained by retraining the Haar classifier cascade with robust dataset of the target object class.

5.2.2 Execution Speed

As mentioned earlier one of the important goals of the proposed work is to speed up the adaboost detection process in a typical surveillance application. The key idea proposed in this work is that to detect objects a full frame search is

Test Sequence	Total Frames	Execution Time. Sec	
		FS	MS
Corridor - Face	300	334	120
Corridor - Upper body	300	431	144
CAVIAR Upper body	300	267	70

Table 5.3: Execution time for test sequences

not always necessary and significant amount of speed up can be achieved by extracting motion regions and image regions of interest. These image regions of interest are processed by the object detection task to detect target objects. Table 5.3 shows the amount of time consumed to process temporal images from both test sequences by full search based detection approach and the proposed motion segmentation based detection approach. The column total frames indicate the number of temporal images used from each test sequence and the columns execution time indicates the amount of time consumed in seconds to process total number of frames by both approaches.

In case of corridor test sequence we process a total of 300 temporal images and there is a reduction of approximately 60% in the amount of time consumed for both faces and upper body. For the CAVIAR test sequence for upper body we process a total 300 temporal images and a reduction of approximately 73% in execution time is achieved.

5.3 Hardware Platform

In the previous section we presented details of performance evaluation of the proposed system in software. From the detection speed it is evident that object detection is a complex task and cannot achieve real-time speed. It is a good proposition to accelerate some of the key compute intensive blocks in hardware and exploit parallelism in the algorithm to speed-up the detection process. In this thesis we propose hardware architectures for two compute intensive blocks corresponding to MoG and the Haar feature computation. These hardware blocks are designed to work with the hardware integration platform built with-in the group. In this section we present details of the hardware integration platform

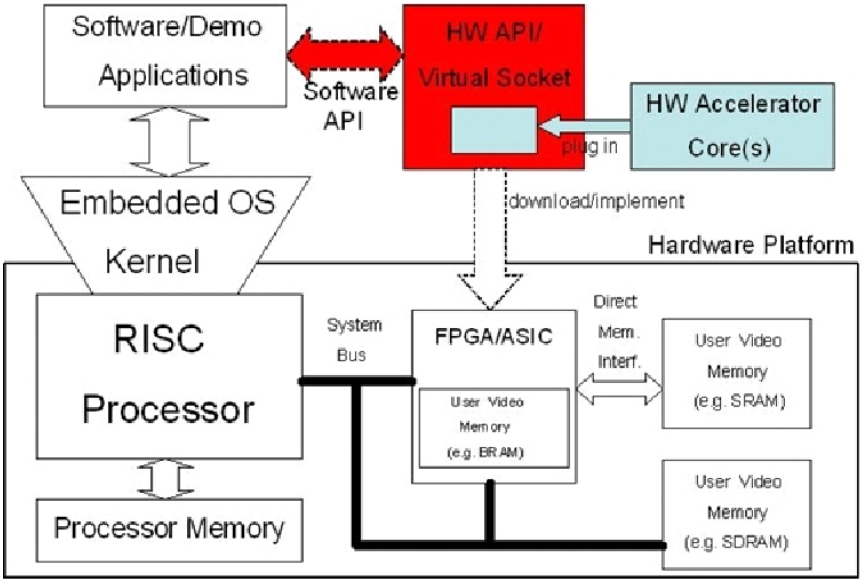

Figure 5.4: Hardware Software Integration Illustration

following which details of hardware implementation of MoG and Haar feature acceleration are presented.

The evaluation platform we use is a Xilinx XUP board, an advanced hardware platform that consists of a Virtex II Pro FPGA with an embedded PowerPC core. The PowerPC is tied to a standard IBM Processor Local Bus (PLB) interface and a PLB-OCP bridge converts processor signals to OCP Bus. Note that our configurable stage uses a OCP interface bus. One key reason for using OCP interface is to enable plug-n-play with different host processor platforms and to integrate several other hardware blocks onto the system, a concept of design reuse widely used for System-On-Chip design. The hardware integration platform we use here is a system developed within the group as part of a wider project. The application code runs on an Embedded PowerPC under the Linux operating system. All hardware accelerators (called virtual components) are connected to the system via the virtual socket interface. Figure 5.4 shows an illustration of the integration platform details of which are omitted for simplicity.

Name	Width	Driver	Function
Clk	1	varies	Clock Input
MReset	1	varies	Reset Input
MAddr	configurable	master	Transfer Address
MCmd	3	master	Transfer Command
MData	configurable	master	Write Data
MByteEn	configurable	master	Request Phase Byte Enables
SCmdAccept	1	slave	Slave Accepts Transfer
SData	configurable	slave	Read Data
SResp	2	slave	Transfer Response

Table 5.4: OCP Bus Signals

5.3.1 OCP Interface

The OCP standard defines a point-to-point interface between two communicating entities such as intellectual property (IP) cores and bus interface modules. One entity act as a master and the other as a slave and the master is responsible for issuing commands to send and receive data. The slave generally responds to master commands by presenting requested data or by accepting data from the master. In table 5.4 the list of OCP signals relevant to the work described here are presented. For detailed description of other OCP signals the reader is referred to OCP technical specification manual OcpIP (2006).

MAddr
The transfer address, master specifies the address of the slave resource targeted by the current transfer. Used to address several configuration, control and data registers in a slave.

MCmd
Transfer command issued by master to slave for a data transfer. Table 5.5 show list of commands used.

MData
OCP data word carries data to be written from master to slave, the length of MData bus is configurable and is not restricted to multiples of 8.

MCmd[2:0]	Command	Mnemonic	request
0 0 0	Idle	IDLE	none
0 0 1	Write	WR	write
0 1 0	Read	RD	read

Table 5.5: MCmd Encoding

SRespp[1:0]	Response	Mnemonic
0 0	No Response	NULL
0 1	Data valid / Accept	DVA
1 0	Request failed	FAIL
1 1	Response Error	ERR

Table 5.6: SResp Encoding

MByteEn
This signal facilitates partial data transfers over MData bus. Each byte in MData is represented by a bit in MByteEn and these bits indicate which bytes within the OCP data word are part of the current transfer.

SCmdAccept
Slave accepts data from master.

SData
Slave data bus carries data requested by master from slave.

SResp
Slave response to a request, signal data valid (DVA) indicates SData has valid data, table 5.6 shows list of responses used.

5.3.2 Programmer's Model

In this section we present details of configuration registers to aid the programing process. In the integration platform we use, each hardware module is termed as a virtual component (VC) and is given an unique identification (ID). Several of these VCs interface with the host processor system with a OCP bus as shown in Figure 5.5. To access different VC blocks attached to the same system bus the

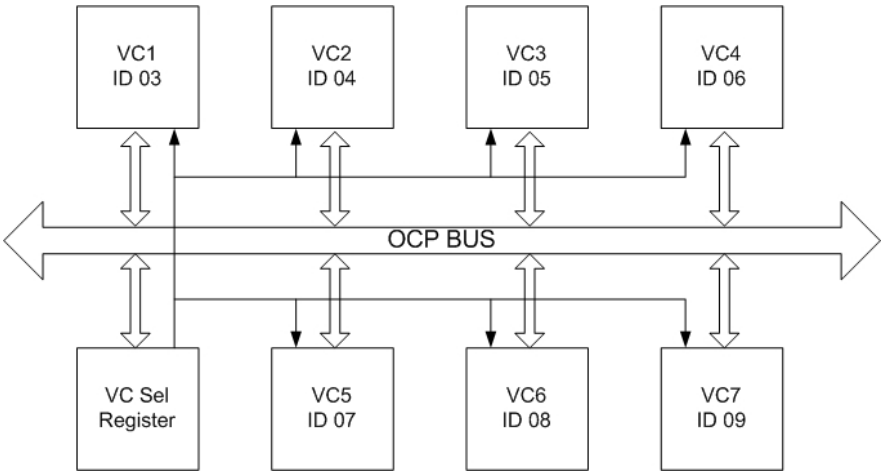

Figure 5.5: Integration Platform with Virtual Components

host processor system first writes the target virtual component ID to the VC select register block. All the subsequent data access performed will be between the host processor and the target VC. Moreover, all the data and configuration registers within each VC can be memory mapped to enable access from application code running on the processor. Reminder of this section presents memory mapping details for the MoG block and the Haar feature acceleration block.

In the MoG hardware block to make data IO simple we use two FIFOs of size 128 $\times$ 32 bits as shown in Figure 5.6. Image data along with background model data for each pixel is written into the IN_FIFO sequentially. The OUT_FIFO is used to read back foreground pixels and the updated background model parameters. A IN_FIFO full status register indicates if the FIFO is full while writing pixel data and similarly a OUT_FIFO status indicates if the FIFO is empty when reading back data. Table 5.7 shows the address map of internal data and status registers. Data written to IN_FIFO_AD is sent to the MoG module and after the data is processed within the MoG hardware block it is written to the OUT_FIFO. Data from OUT_FIFO can be read from the address OUT_FIFO_AD.

Table 5.8 shows the address map of configuration and control registers used to control the reconfigurable stage. The mapping makes it easy to read and write data from host processor and helps program the module. Any data written to Config RAM and Data RAM address locations will be written to the respective

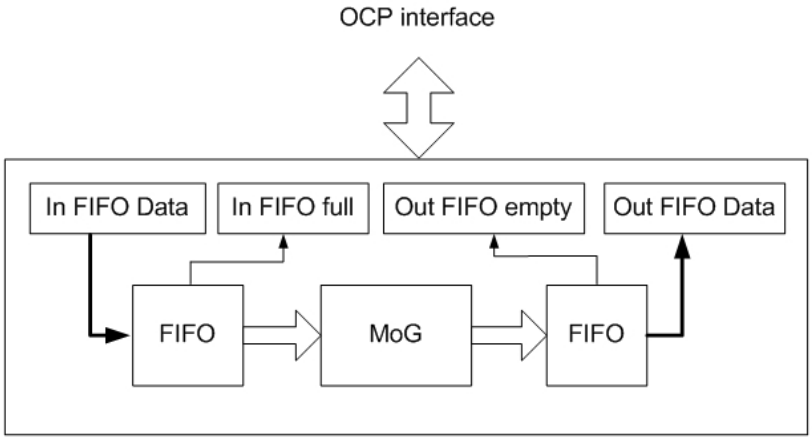

Figure 5.6: MoG Virtual Component

Register Name	Read/Write	Address	Function
IN_FIFO_AD	WR	0xCAC40200	Write data to MoG module
IN_FIFO_ST	RD	0xCAC40208	Indicate if FIFO full
OUT_FIFO_AD	RD	0xCAC40210	Out FIFO read address
OUT_FIFO_ST	RD	0xCAC40218	Indicate is FIFO is empty

Table 5.7: MoG OCP interface Registers

Register Name	Read/Write	Address	Function
Config RAM	WR	0xCAC40200	Write data to config RAM
Data RAM	WR	0xCAC40208	Write data to data RAM
Config Status	RD	0xCAC40210	Stage configuration status flag
Start Stage	WR	0xCAC40218	Start stage computation flag
Finish Stage	RD	0xCAC40220	Stage finish status flag
Stage Sum	RD	0xCAC40228	Final stage sum register
Sub-window variance	WR	0xCAC40230	Window variance register

Table 5.8: Stage Control and Configuration Registers

RAMs in the physical stage. A start stage flag is used to start the process of configuring individual FEs with feature and integral pixel data. Once data is loaded each FE starts computing the feature value. The config status flag indicates the completion of configuring all FEs in the stage. The finish stage flag indicates that all FEs have finished computing and the stage sum is ready to be read by the host processor. The stage sum location holds the final result from the stage.

5.4 Results

From the Section 5.2 it is clear that the proposed object detection system cannot achieve real-time execution in software. To achieve real-time detection speeds it is important that some of the compute intensive blocks are accelerated in hardware. As per the motivations presented earlier in Chapter 4 the most suitable hardware platform for vision applications is the Field Programmable Gate Array (FPGA). Moreover the cost of the FPGA and the amount of resources available on the FPGA have a significant impact on the overall system performance. Cost of the device is an important factor when considering inexpensive embedded smart cameras and this has a direct affect on the detection speeds. Here we consider a bigger and more expensive device which is the Xilinx Virtex II Pro however depending on the required system performance and the cost a much smaller device can also be used. Tables 5.9, 5.10 and 5.11 show the amount of hardware resources consumed by the MoG, a reconfigurable FE and a stage formed by grouping 16 FEs respectively.

The hardware resource usage for MoG presented in table 5.9 show that in total less than 5% of the resources are consumed. This is because here we target a bigger device from the Virtex II Pro family but a much smaller device can also be used. This however depends on other system modules, the required system performance and the cost constraints.

The proposed MoG hardware module runs at 223 MHz as reported by the Xilinx tools and can process a 640 $\times$ 480 image at 60 frames per second excluding bus interface delay. Various factors that need to be considered when choosing an FPGA for embedded smart camera are: cost, power consumption, required

Number of Slices	794	out of 13696	5%
Slice Flip-Flops	1190	out of 27392	4%
4 in-put LUTs	1074	out of 27392	3%
BRAMS	2	out of 136	1%
MULTs 18 x 18	8	out of 136	5%

Table 5.9: MoG Hardware Resource Usage - Synthesized for XC2VP30

Resource Type	used	Maximum Available	% Usage
Number of Slices	311	13696	2%
Number of Slice Flip Flops	488	27392	1%
Number of 4 input LUTs	317	27392	1%
Number of TBUFs	16	6848	0%
Number of MULT18X18s	1	136	0%

Table 5.10: Resource Usage for Reconfigurable FE - Xilinx XC2VP30

hardware resources depending on the number of hardware modules used and the required system performance.

Table 5.11 presents details of resource usage for the proposed stage and clearly the object detection task needs more hardware resources. As mentioned earlier to achieve real-time detection performance it is important that Haar-feature computation is accelerated in hardware.

The proposed configurable FE stage can run at 245 MHz as reported by the Xilinx tools and can compute the result of 16 Haar features in 2.5 μsec. The key advantage with using a FE based architecture is that we can exploit the parallelism within a stage and also the fact that multiple stages can run in parallel

Resource Type	used	Maximum Available	% Usage
Number of Slices	5197	13696	37%
Number of Slice Flip Flops	8289	27392	30%
Number of 4 input LUTs	5481	27392	20%
Number of TBUFs	256	6848	3%
Number of BRAMs	2	136	1%
Number of MULT18X18s	16	136	11%

Table 5.11: Resource Usage for 16 FE Stage - Xilinx XC2VP30

in effect achieving higher detection speeds. Running multiple stages in parallel however depends on the amount of available resources and various other design constraints mentioned earlier. A key idea proposed in this work is the design of a stand-alone and a flexible stage formed by grouping 16 FEs. The stage is flexible in the sense that it can be configured to form any stage in the adaboost cascade and target different object detection tasks.

5.5 Conclusion

In this chapter we presented details of performance evaluation of the proposed system in software. Some of the key advantages with using a motion segmentation technique as a pre-processing step to object detection is that a significant reduction in execution time, reduction in false positives can be achieved (thus increase in precision). However, there is a reduction in the recall values because of challenges involved in building a robust background suppression technique. As mentioned earlier the proposed object detection system contains many compute intensive tasks such as background modeling, post-processing, integral image computation, square integral image computation, image scaling and Haar feature computation for several cascade stages. A typical embedded smart camera cannot achieve real-time processing for object detection needed for surveillance applications. A good idea is to off load some of these compute intensive tasks to a hardware accelerator and the most suitable hardware acceleration platform for vision application is the Field Programmable Gate Array (FPGA). In this work we proposed FPGA accelerators for MoG and the Haar feature computation. Also instead of making Haar computation specific to detecting a single object (e.g. face) we designed a more generic and flexible architecture. The key building block in the proposed architecture is a feature element FE and a stage formed by grouping multiple FEs. The stage formed by grouping multiple FEs can be configured to form any stage in the adaboost classifier cascade and can be used to detect different objects. Also the use of OCP as a standard bus interface for both MoG and Haar stage, that makes it possible to build a multi stage cascade classifier in hardware which can be easily interfaced with a host processor.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

Smart cameras are becoming increasingly popular in surveillance systems for detecting humans, vehicles among others. The ultimate goal is to push intelligence and adaptability, to the device itself for autonomous analysis of the captured image. An important task in surveillance systems is the object detection task. Adaboost based object detection which we use in this thesis is one of the most popular techniques. There are several compute intensive blocks in the adaboost algorithm as outlined in the previous chapters and a full frame search for detecting target objects makes the algorithm highly time consuming. One way to speed up the detection process is by processing only image sub-regions or regions of interest instead of the whole frame. This makes good sense in a surveillance type application where a full frame search is not always necessary. Moreover a resource constrained device such as a smart camera can greatly benefit from the reduction in computation time.

One way extract image regions of interest is to use background modeling and moving object segmentation. Many background modeling techniques of varying complexity and segmentation performance were proposed. Among these various approaches we choose to use the Mixture of Gaussian approach because of its relatively low computational complexity and better segmentation performance. The MoG approach we use in this work can handle complex outdoor scenes and

is adaptive. Another major advantage with MoG is it is suitable for hardware implementation. Hardware acceleration is an important element because of the fact that the majority of image and video analysis algorithms are both data and compute intensive. When targeting embedded devices such as smart cameras for surveillance applications it is vital that the system/algorithm achieves real-time performance.

In addition to this a key compute intensive block in adaboost is the Haar feature computation and it occupies a significant amount of computation time. In this thesis we proposed an accelerator for Haar feature computation. To enable generic object detection we designed a flexible Haar feature element.

The use of motion segmentation as a pre-processing step to adaboost object detection has a significant effect on the detection speed as presented in previous chapter 5. When implementing object detection in surveillance systems it is essential to reduce the computation time by processing only motion regions or regions of interest to detect target objects. Although the proposed system shows good precision, recall performance greatly relies on robust background modeling and motion segmentation. Due to inherent limitations in building a robust background model, it is hard to segment slow moving objects, homogeneously colored objects and objects that are camouflaged.

Moreover to achieve real-time detection performance it is vital that some of the key compute intensive blocks are accelerated in hardware. We proposed hardware acceleration architectures for both MoG and Haar feature computation. To enable generic object detection we designed a flexible feature element that can be programed to form any basis function. As outlined earlier the adaboost detection process uses multiple Haar features in a stage and several of these stages are grouped in a cascade to improve the detection accuracy. With the FE based design we can group several of these FEs to form a stage and multiple stages could run in parallel making effective use of the parallelism in the algorithm.

Finally the use of OCP bus for both hardware blocks makes it easy to integrate with a host processor system. A set of configuration and control registers that are memory mapped enables application code running on the host processor to communicate with these blocks.

6.2 Future Work

The research work presented in this thesis deals with decreasing the computation time for adaboost based object detection. Overall the system has three important blocks, background modeling, post-processing and adaboost detection. Key aspects that contribute to overall system performance are as follows:

Precision of object segmentation.

Robust Haar cascade for adaboost detection.

Hardware acceleration for key compute intensive blocks.

An important block in the system is moving object segmentation. The detection accuracy and speed rely on precise object segmentation. As outlined in Chapter 2, inherent limitations in background modeling techniques makes it difficult to detect foreground regions (e.g. aperture, camouflage) and prevent background regions (e.g. shadows, reflections) from appearing in the foreground. Moving object segmentation is an ongoing research field and a more robust or a less compute intensive technique could greatly improve the system performance. Another important aspect is the adaboost cascade detector, where by significant reduction in false positives can be achieved by training the Haar cascade with a more robust data set from the target object class. Moreover, the adaboost training process is highly complex, time consuming and requires few hundred training samples. Efforts to reduce training complexity could greatly benefit surveillance applications. The proposed system is tested for faces and upper-body using pretrained Haar feature databases from Open CV. However the system could be used to detect other objects for example automobiles and this should be investigated. Finally, the optimal number of features per stage and the optimal number of stages in hardware to achieve real-time detection performance and the tradeoffs thereof requires significant research attention in the future. Also low complexity hardware for other compute intensive blocks such as morphology, connected component labeling, integral image computation and image scaling could be a good research direction.

Bibliography

Anderson, J. D., Lee, D. J., and Archibald, J. K. (2005). FPGA implementation of vision algorithms for small autonomous robots. In Casasent, D. P., Hall, E. L., and Röning, J., editors, *Intelligent Robots and Computer Vision XXIII: Algorithms, Techniques, and Active Vision. Edited by Casasent, David P.; Hall, Ernest L.; Röning, Juha. Proceedings of the SPIE, Volume 6006, pp. 401-411 (2005).*, volume 6006 of *Presented at the Society of Photo-Optical Instrumentation Engineers (SPIE) Conference*, pages 401–411. SPIE. 40

Baumgartner, D., Rossler, P., and Kubinger, W. (2007). Performance benchmark of DSP and FPGA implementations of low-level vision algorithms. In *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on*, pages 1–8. 40

Bradski, G. and Kaehler, A. (2008). *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly, Cambridge, MA. 4

CAVIAR (2000). http://homepages.inf.ed.ac.uk/rbf/CAVIARDATA1/. 55

cbcl (2000). MIT face dataset availabel at http://cbcl.mit.edu/software-datasets/FaceData2.html. Center for Biological and Computational Learning at MIT and MIT. viii, 32

Cheung, S.-C. S. and Kamath, C. (2005). Robust background subtraction with foreground validation for urban traffic video. *EURASIP J. Appl. Signal Process.*, 2005(1):2330–2340. 5

Colmenarez, A. and Huang, T. (1997). Face detection with information-based maximum discrimination. In *CVPR '97: Proceedings of the 1997 Conference on*

Computer Vision and Pattern Recognition (CVPR '97), page 782, Washington, DC, USA. IEEE Computer Society. 27

Craw, I., Tock, D., and Bennett, A. (1992). Finding face features. In *ECCV '92: Proceedings of the Second European Conference on Computer Vision*, pages 92–96, London, UK. Springer-Verlag. 27

Dai, Y. and Nakano, Y. (1996). Face-texture model-based on sgld and its application in face detection in a color scene. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(6):1007–1017. 27

Elgammal, A. M., Harwood, D., and Davis, L. S. (2000). Non-parametric model for background subtraction. In *ECCV '00: Proceedings of the 6th European Conference on Computer Vision-Part II*, pages 751–767, London, UK. Springer-Verlag. 16

Feraud, R. and Bernier, O. (1998). Ensemble and modular approaches for face detection: a comparison. In *NIPS '97: Proceedings of the 1997 conference on Advances in neural information processing systems 10*, pages 472–478, Cambridge, MA, USA. MIT Press. 3

Freund, Y. and Schapire, R. E. (1995). A decision-theoretic generalization of on-line learning and an application to boosting. In *EuroCOLT '95: Proceedings of the Second European Conference on Computational Learning Theory*, pages 23–37, London, UK. Springer-Verlag. 25, 30

Gao, C. and Lu, S.-L. (2008). Novel FPGA based haar classifier face detection algorithm acceleration. In *Field Programmable Logic and Applications, 2008. FPL 2008. International Conference on*, pages 373–378, Heidelberg. 44, 47

Gonzalez, R. and Woods, R. (2002). *Digital Image Processing, Second Edition*, chapter 9, pages 519–566. Number ISBN-81-203-2758-6. Prentice Hall. 12, 13, 14

Hiromoto, M., Nakahara, K., Sugano, H., Nakamura, Y., and Miyamoto, R. (2007). A specialized processor suitable for adaboost-based detection with haar-like features. *Embedded Computer Vision*, pages 1–8. 25

Hiromoto, M., Sugano, H., and Miyamoto, R. (2009). Partially parallel architecture for adaboost-based detection with haar-like features. *IEEE Transactions on Circuits and Systems for Video Technology*, 19(1):41–52. 4, 43, 45, 46

Hjelmas, E. and Low, B. K. (2001). Face detection: A survey. *Computer Vision and Image Understanding*, 83(3):236 – 274. 3

Intel IPP (2008). Integrated Performance Primitives (Intelő IPP). Intel. 4

Kelly, P. (2008). Corridor Scenario. http://www.cdvp.dcu.ie/datasets. 55

Kim, K., Chalidabhongse, T. H., Harwood, D., and Davis, L. (2004). Background modeling and subtraction by codebook construction. In *Image Processing, 2004. ICIP '04. 2004 International Conference on*, volume 5, pages 3061–3064. 16

Kirby, M. and Sirovich, L. (1990). Application of the karhunen-loeve procedure for the characterization of human faces. In *IEEE Trans. Pattern Anal. Mach. Intell.*, volume 12, pages 103–108, Washington, DC, USA. IEEE Computer Society. 3

Kjeldsen, R. and Kender, J. (1996). Finding skin in color images. In *FG '96: Proceedings of the 2nd International Conference on Automatic Face and Gesture Recognition (FG '96)*, page 312, Washington, DC, USA. IEEE Computer Society. 27

Lai, H.-C., Marculescu, R., Savvides, M., and Chen, T. (2008). Communication-aware face detection using noc architecture. In Gasteratos, A., Vincze, M., and Tsotsos, J. K., editors, *International Conference on Computer Vision Systems*, volume 5008 of *Lecture Notes in Computer Science*, pages 181–189. Springer. 46, 48

Lanitis, A., Taylor, C., and Cootes, T. (1995). Automatic face identification system using flexible appearance models. *Image and Vision Computing*, 13:393–401. 27

Leung, T. K., Burl, M. C., and Perona, P. (1995). Finding faces in cluttered scenes using random labeled graph matching. In *ICCV '95: Proceedings of the Fifth International Conference on Computer Vision*, page 637, Washington, DC, USA. IEEE Computer Society. 27

Lew, M. S. (1996). Information theoretic view-based and modular face detection. In *FG '96: Proceedings of the 2nd International Conference on Automatic Face and Gesture Recognition (FG '96)*, page 198, Washington, DC, USA. IEEE Computer Society. 27

Li, R., Chen, Y., and Zhang, X. (2006). Fast robust eigen-background updating for foreground detection. In *Image Processing, 2006 IEEE International Conference on*, pages 1833–1836. 16

Liao, M. H. Y., Chen, D.-Y., Sua, C.-W., and Tyan, H.-R. (2006). Real-time event detection and its application to surveillance systems. In *Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium on*, Island of Kos. 5

Lienhart, R., Kuranov, A., and Pisarevsky, V. (2002). Empirical analysis of detection cascades of boosted classifiers for rapid object detection. Technical report, Microprocessor Research Lab, Intel Labs. 25

Lienhart, R. and Maydt, J. (2002). An extended set of haar-like features for rapid object detection. In *Image Processing. 2002. Proceedings. 2002 International Conference on*, volume 1, pages 900–903. 6, 25

Lucas, S. (1993). Optical character recognition with hough transform based neuralnetworks. In *Hough Transforms, IEE Colloquium on*, London, UK. 3

MacLean, W. (2005). An Evaluation of the Suitability of FPGAs for Embedded Vision Systems. In *Computer Vision and Pattern Recognition, 2005 IEEE Computer Society Conference on*, volume 3, pages 131–131. 40

McBader, S. and Lee, P. (2003). An FPGA implementation of a flexible, parallel image processing architecture suitable for embedded vision systems. In *Parallel*

and Distributed Processing Symposium, 2003. Proceedings. International, page 5pp. 40

McKenna, S., Gong, S., and Raja, Y. (1998). Modelling facial colour and identity with gaussian mixtures. *Pattern Recognition Journal*, 31(12):1883–1892. 27

Meir, R. and Rätsch, G. (2003). An introduction to boosting and leveraging. *Advanced lectures on machine learning*, pages 118–183. 30

Meynet, J. (2003). Technical report - fast face detection using adaboost. Technical report, Signal Processing Institute, EPFL. 30, 31

Miyamoto, R., Sugano, H., Saito, H., Tsutsui, H., Ochi, H., Hatanaka, K., and Nakamura, Y. (2006). Pedestrian recognition in far-infrared images by combining boosting-based detection and skeleton-based stochastic tracking. In *1st IEEE Pacific-Rim Symp. Image Video Technologies*, pages 483–494. 4, 25

Nair, V., Laprise, P.-O., and Clark, J. J. (2005). An FPGA-based people detection system. *EURASIP Journal on Applied Signal Processing*, 2005(7):1047–1061. doi:10.1155/ASP.2005.1047. 40

OcpIP (2006). *Open Core Protocol Specification*. OCP IP. 42, 49, 63

Oliver, N. M., Rosario, B., and Pentland, A. P. (2000). A bayesian computer vision system for modeling human interactions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):831–843. 5

Open CV (2008). Open Computer Vision Library, available at http://sourceforge.net/projects/opencvlibrary/. 4, 5, 34, 37, 38

Osuna, E., Freund, R., and Girosit, F. (1997). Training support vector machines: An application to face detection. In *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, pages 130–136, San Juan. 3, 27

Papageorgiou, C. P., Oren, M., and Poggio, T. (1998). A general framework for object detection. In *Computer Vision, 1998. Sixth International Conference on*, pages 555–562, Bombay. 3

Piccardi, M. (2004). Background subtraction techniques: a review. In *Systems, Man and Cybernetics, 2004 IEEE International Conference on*, volume 4, pages 3099–3104vol.4. 8, 10

Pless, R. (2005). Spatio-temporal background models for outdoor surveillance. In *EURASIP J. Appl. Signal Process.*, volume 2005, pages 2281–2291, New York, NY, United States. Hindawi Publishing Corp. 5

Rajagopalan, A. N., Kumar, K. S., Karlekar, J., Manivasakan, R., Patil, M. M., Desai, U. B., Poonacha, P. G., and Chaudhuri, S. (1998). Finding faces in photographs. In *ICCV '98: Proceedings of the Sixth International Conference on Computer Vision*, page 640, Washington, DC, USA. IEEE Computer Society. 27

Ran, Y., Zheng, Q., Weiss, I., Davis, L., Abd Almageed, W., and Zhao, L. (2005). Pedestrian classification from moving platforms using cyclic motion pattern. pages II: 854–857. 9

Rand, W. M. (1971). Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 66(336):846–850. 17

Ratha, N. K. and Jain, A. K. (1999). Computer vision algorithms on reconfigurable logic arrays. *Transactions on Parallel and Distributed Systems*, 10(1):29–43. 40

Rosin, P. L. and Ioannidis, E. (2003). Evaluation of global image thresholding for change detection. *Pattern Recogn. Lett.*, 24(14):2345–2356. 8

Rowley, H. A., Baluja, S., and Kanade, T. (1998). Neural network-based face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(1):23–38. 3, 27, 34

Sato, K. and Aggarwal, J. K. (2004). Temporal spatio-velocity transform and its application to tracking and interaction. *Comput. Vis. Image Underst.*, 96(2):100–128. 9

Schneiderman, H. and Kanade, T. (1998). Probabilistic modeling of local appearance and spatial relationships for object recognition. In *CVPR '98: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, page 45, Washington, DC, USA. IEEE Computer Society. 27

Shahid, H., Khan, K., and Qazi, W. A. (2008). Using modified mixture of gaussians for background modeling in video surveillance. In *Advances in Space Technologies, 2008. ICAST 2008. 2nd International Conference on*, pages 155–159, Islamabad, Pakistan. 5

Snidaro, L., Micheloni, C., and Chiavedale, C. (2005). Video security for ambient intelligence. *IEEE Transaction on System, Man and Cybernetics - Part A*, 35(2):133–144. 9

Snow, D., Jones, M., and Viola, P. (2003). Detecting pedestrians using patterns of motion and appearance. In *IEEE International Conference on Computer Vision*, pages 734–741. 4, 25

Stauffer, C. and Grimson, W. (1999). Adaptive background mixture models for real-time tracking. In *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.*, volume 2. 5, 10, 40

Sung, K. K. and Poggio, T. (1998). Example-based learning for view-based human face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(1):39–51. 3, 27

Suzuki, K., Horiba, I., and Sugie, N. (2003). Linear-time connected-component labeling based on sequential local operations. *Computer Vision and Image Understanding*, 89(1):1–23. 15

Tickle, A, J., J, S, S., and Q, H, W. (2007). Development of morphological operators for field programmable gate arrays. *Journal of Physics: Conference Series*, 76. 14

Toyama, K., Krumm, J., Brumitt, B., and Meyers, B. (1999a). The test sequences are availabel at - http://research.microsoft.com/en-us/um/people/jckrumm/wallflower/testimages.htm. Microsoft.com. 10, 17

Toyama, K., Krumm, J., Brumitt, B., and Meyers, B. (1999b). Wallflower: principles and practice of background maintenance. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 1, pages 255–261vol.1. 9, 10

Turk, M. and Pentland, A. (1991). Eigenfaces for recognition. *J. Cognitive Neuroscience*, 3(1):71–86. 3, 27

Viola, P. and Jones, M. J. (2004). Robust real-time face detection. *Int. J. Comput. Vision*, 57(2):137–154. 3, 27, 32

Wang, H. and Suter, D. (2005). A re-evaluation of mixture of gaussian background modeling [video signal processing applications]. In *Acoustics, Speech, and Signal Processing, 2005. Proceedings. (ICASSP '05). IEEE International Conference on*, volume 2. 11

Yang, G. and Huang, T. S. (1994). Human face detection in a complex background. *IEEE Pattern Recogn*, 27(1):53–66. 27

Yang, J. and Waibel, A. (1996). A real-time face tracker. In *WACV '96: Proceedings of the 3rd IEEE Workshop on Applications of Computer Vision (WACV '96)*, page 142, Washington, DC, USA. IEEE Computer Society. 27

Yang, M.-H., Kriegman, D. J., and Ahuja, N. (2002). Detecting faces in images: a survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(1):34–58. 3, 26, 27

Yew, K. and Cipolla, R. (1997). Feature-based human face detection. *Image and Vision Computing*, 15(9):713–735. 27

Zhang, H., Xia, M., and Hu, G. (2007). A multiwindow partial buffering scheme for FPGA-based 2-d convolvers. *Circuits and Systems II: Express Briefs, IEEE Transactions on [see also Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on]*, 54(2):200–204. 40

Zhu, Q., Avidan, S., and Cheng, K.-T. (2005). Learning a sparse, corner-based representation for time-varying background modelling. In *Computer Vision,*

2005. ICCV 2005. Tenth IEEE International Conference on, volume 1, pages 678–685. 8