VLSI array architectures for navigation and human recognition in vision-guided embedded systems

Mohan Aruppukottai Rajarathinam

2011

Mohan, A. R. (2011). VLSI array architectures for navigation and human recognition in vision-guided embedded systems. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/43563>

<https://doi.org/10.32657/10356/43563>

VLSI Array Architectures for Navigation and Human Recognition in Vision-Guided Embedded Systems

A Thesis submitted to the Nanyang Technological University in fulfillment of the requirement for the degree of Doctor of Philosophy

by

Mohan Aruppukottai Rajarathinam

School of Computer Engineering
Nanyang Technological University

March 2011

Abstract

This thesis focuses on hardware solutions for vision-based navigation and human recognition. The application scenarios include mobile robotics, human-robot interaction, assistive technology for visually-challenged and video surveillance. Hardware accelerators are necessary for the navigation of robots and guiding systems for visually-challenged in a dynamic environment. They are also necessary for real-time identification of a human by a service robot or a visually-challenged for interaction. Single chip solutions with focus on reconfiguration, low area, and high speed are desirable for such applications. This thesis reports hardware-directed algorithms and array-type Very Large Scale Integrated (VLSI) architectures for navigation and face-based human recognition. The key contributions are listed below.

1. A hardware-efficient algorithm for navigation on a binary image of an environment and its VLSI architecture are presented in this thesis. The algorithm computes a distance map to identify the collision-free region and then constructs a breadth-first search tree to find a path in that region. The path obtained from a start point to the goal point is the shortest path in terms of the number of steps. The time-critical section of the algorithm is mapped onto a two-dimensional cellular architecture that consists of a locally interconnected array of identical processing elements. In view of this local interconnection and regular structure, the architecture can be operated at high speed and can be accommodated in a small chip area. The design has been implemented and evaluated on a Field Programmable Gate Array (FPGA).

The FPGA design is shown to be capable of processing images at video rate for real-time path planning in a dynamic environment.

2. The second contribution in the thesis is the development of VLSI architecture for face recognition. Face recognition is a natural and non-intrusive method of person identification. A neural network-based technique and its VLSI architecture for automatic recognition of faces are proposed in this thesis. The proposed method applies Principal Component Neural Network (PCNN) with generalized Hebbian learning for extracting eigenfaces from the enrolled face database. The face recognition system based on the proposed method demonstrates a recognition performance of more than 85% when evaluated on two benchmark face databases containing images with varying illumination and expression. Unlike the existing face recognition systems, the proposed approach not only recognizes the faces using computed eigenfaces, but also updates eigenfaces automatically whenever the enrolled face database changes.

A primary contribution from the architecture point of view is an optimized mapping of fine-grained systolized Signal Flow Graphs (SFGs) for each individual step of the algorithm onto a single reconfigurable linear systolic array by appropriate merging of the computations pertaining to different nodes of different SFGs. The architecture has the flexibility of processing face images and enrolled face database of any size and it is easily scalable for the number of eigenfaces to be computed. The proposed PCNN-based systolic face recognition system has been implemented and evaluated on an FPGA platform. The FPGA-based design for a reasonably large-sized enrolled face database can process more than 400 faces in a video image frame which is fast enough for video surveillance in busy public places and sensitive locations.

3. Detection of faces in an image is the primary step to face recognition, as only detected faces need to be processed for face recognition. In general, all possible sub-images in the given image or frame have to be analyzed to test whether it is a face or not. Also the sub-images can be of different sizes because the size of faces in the given image is not known beforehand, which again increases the search-space. As an eigenface-based algorithm has been proposed for recognition, using the same features for face detection as well would circumvent the computation of additional features separately for face detection and recognition.

In this thesis, an eigenface-based face detection method using boosted eigen features is proposed in which the eigenvectors obtained by principal component analysis (PCA) on a set of training images are used as features. The features are boosted using AdaBoost algorithm to perform face detection. A novel contribution is FFT-based computation of the distance map which facilitates hardware implementation and fast face detection. A suitable hardware mapping method is also discussed.

The VLSI architectures developed for navigation and recognition are arrays of simple processing elements with localized and reduced interconnection. They are modular and easily scalable for different image sizes and other parameters. High speed of operation and extensive concurrency in processing the pixels are achieved by these array-type VLSI designs.

Acknowledgements

First of all, I would like to express my sincere thanks to my supervisor Dr. Sudha Natarajan for providing invaluable guidance for my research work. She often provided me with useful suggestions and reference material and showed me the right direction whenever I was lost during my research. Her useful suggestions on technical writing have helped me in writing technical material without which this thesis would not have taken shape.

I would like to thank Dr. Pramod Kumar Meher for all the insightful discussions on the design techniques for the systolic architecture based face recognition. Initial discussions with Prof. Srikanthan offered me the motivation to pursue my research, for which I will always be thankful.

I convey my sincere thanks to Ms. Merilyn Yap, Ms. Nah and Mr. Chua Ngee Tat for all the administrative help they rendered during my 4 year stint at Center for High Performance Embedded Systems. I also thank fellow researchers at CHiPES for all the thought provoking discussions and useful suggestions.

I would like to thank Nanyang Technological University for providing me with scholarship to support my research and the university library for providing access to excellent research publications.

I wish to express my heartfelt thanks to all my friends who have always been there for me and provided me constant motivation to pursue my research. I would like to thank my brothers A R Karthikeyan who has been my mentor and guide and A R Sundresan who has always cheered me in all my endeavors.

ACKNOWLEDGEMENTS

Finally I wish to thank my parents who have supported and guided me throughout my life and encouraged my higher academic pursuits. I would like to dedicate this thesis to my family who have made me the person I am today.

Contents

Abstract	i
Acknowledgements	iv
1 Introduction	1
1.1 Motivation	1
1.1.1 Vision-Guided Embedded Systems	2
1.1.2 Need for Hardware-Directed Algorithms for Navigation and Recognition	3
1.2 Objectives	4
1.2.1 Cellular Architecture for Vision-Based Path Planning	4
1.2.2 Systolic Architecture for Face Recognition	4
1.2.3 Hardware-Friendly Algorithm for Face Detection	5
1.3 Organization of the Thesis	5
2 Background and Literature Review	7
2.1 Path Planning	7
2.1.1 Vision-Based Path Planning Algorithms	9
2.2 Hardware Architectures for Path Planning	13

CONTENTS

2.2.1	Visibility Graph-Based Architectures	13
2.2.2	Voronoi Diagram-Based Architectures	14
2.3	Overview of Face Recognition Algorithms	16
2.3.1	Holistic Approach	17
2.3.2	Feature-Based Approach	20
2.3.3	Hybrid Approach	22
2.3.4	Face Detection Algorithms	24
2.3.5	PCA for Face Recognition and Detection	25
2.4	Hardware Architectures for Face Recognition	28
2.5	Array Architectures	30
2.5.1	Systolic Architectures	31
2.5.2	Cellular Architectures	39
2.6	Conclusions	41
3	Path Planning on Binary Image of an Environment	42
3.1	Construction of Distance Map	43
3.2	Path Planning on the Distance Map	45
3.3	Proposed Algorithm	50
3.3.1	Complexity Analysis	53
3.3.2	Simulation Studies	54
3.4	Conclusions	57

4	Cellular Architecture for Path Planning	59
4.1	Proposed Cellular Architecture	59
4.1.1	Design of a Cell	61
4.1.2	Operation of Hardware	65
4.2	Implementation and Testing	66
4.2.1	FPGA Implementation	67
4.2.2	Testing of FPGA Design	70
4.2.3	Scalability of Design	72
4.3	Comparison Studies	76
4.4	Conclusions	78
5	Neural Network-Based Face Recognition Algorithm	79
5.1	System Description	80
5.1.1	Training Phase	82
5.1.2	Recognition Phase	83
5.1.3	Improving Recognition Performance	84
5.2	Performance Studies	86
5.2.1	Database Description	86
5.2.2	Performance Evaluation	87
5.2.3	Word Length Effects	90
5.3	Conclusions	94

6	Systolic Architecture for Face Recognition	95
6.1	Proposed Systolic Mapping of the Neural Algorithm	95
6.1.1	Computation of Neuron Outputs for Training Faces	96
6.1.2	Updating Weights	97
6.1.3	Computation of Projections of Training Faces onto Eigenfaces . .	99
6.1.4	Computation of Neuron Outputs for a New Face	101
6.1.5	Computation of Difference Between Projections	102
6.2	Architecture Design	104
6.2.1	Processing Element Design	105
6.2.2	Operation of Overall Architecture	113
6.3	Implementation and Comparisons	119
6.3.1	Hardware Implementation	119
6.3.2	Comparison Studies	126
6.4	Conclusions	130
7	FFT-Based Fast Face Detection Using Boosted Eigenfaces	131
7.1	Constructing Strong Classifier from PCA Features	132
7.1.1	Feature Extraction	132
7.1.2	Thresholding Based Weak Classifiers	133
7.1.3	Boosting	134
7.1.4	Experimental Results	136

CONTENTS

7.2	Extending the Classifier to Detect Faces in Images Using FFT	137
7.2.1	FFT Based Computation of Projection Values	138
7.2.2	FFT Based Computation of Distance Values	139
7.3	Hardware Mapping	142
7.4	Conclusions	145
8	Conclusions and Future Directions	146
8.1	Conclusions	146
8.2	Future Directions	149
	List of Publications	151
	Bibliography	153
	Appendix A ML403 FPGA Evaluation Platform	169

List of Figures

2.1	Collision free regions for navigators of different shapes	8
2.2	Path planning using Visibility Graph	10
2.3	Voronoi diagram of points	11
2.4	Cell Decomposition Method	12
2.5	Dependence Graph for a matrix-vector multiplication and description of a node	32
2.6	Projection of dependence graph onto signal flow graph	33
2.7	A 2-D cellular architecture	40
3.1	Neighborhood N of p	44
3.2	Illustration of the proposed path planning method	47
3.3	Computation of $(\Delta x, \Delta y)$ and δ for EDT	48
3.4	Path traced using two types of neighborhood	49
3.5	Results of path planning on a binary image	55
3.6	Path planning on a dynamic environment	56
4.1	A 4×4 cellular array.	60
4.2	Architecture of a cell	62

LIST OF FIGURES

4.3	Xilinx ISE simulation results for EDT computation	67
4.4	Xilinx ISE simulation results for <i>bfs</i> tree construction	68
4.5	Time taken for the construction of <i>bfs</i> tree in PC and FPGA for different image sizes.	69
4.6	ML403 board components and connections relevant to our experiment. . .	70
4.7	Scalability via processing of multiple pixels in a cell in pipeline	72
4.8	Xilinx ISE simulation results for EDT computation (scalable cell)	74
4.9	Xilinx ISE simulation results for <i>bfs</i> tree construction (scalable cell) . . .	75
5.1	PCNN-based face recognition system	82
5.2	Sub-images of face images	85
5.3	Sample face images from Yale and FRGC databases	85
5.4	Recognition performance for varying sizes of face images	89
5.5	Recognition performance for the division of training set	90
5.6	Performance of single-PCNN and multi-PCNN on Yale face database . .	91
5.7	Performance of single-PCNN and multi-PCNN on FRGC face database .	92
5.8	Word length effects on recognition rate	93
6.1	SFG for the computation of neuron outputs for a training face	98
6.2	SFGs for weights updating	100
6.3	SFG for the computation of projections of training faces onto eigenfaces.	101
6.4	SFG for the computation of neuron outputs for a new face.	102

LIST OF FIGURES

6.5	SFG for the computation of d_p	103
6.6	Proposed systolic architecture for face recognition	105
6.7	Processing element architecture	107
6.8	Circuit for control signal generation.	111
6.9	Combined control flow diagram for the j^{th} PE in training and recognition phases.	112
6.10	Scheduling of computations during the training phase.	115
6.11	Scheduling of computations during the recognition phase.	116
6.12	Performance of FPGA-based recognition system.	122
6.13	ML403 board components and connections relevant to our experiment. . .	123
7.1	Threshold selection	134
7.2	Sample images from the database	136
7.3	Selection of threshold θ_i	137
7.4	Detected faces	141
7.5	Dependency graph for integral image computation	143
7.6	Schedule vector and hyperplanes	143
7.7	Signal flow graph. ‘D’ represents unit delay	144
7.8	Processing element	144
A.1	ML403 snapshot	171
A.2	ML403 block diagram	172

LIST OF FIGURES

A.3	Block diagram of XPS UARTLite	176
A.4	LCD wiring on the ML403 board	176
A.5	Hardware netlist generation flow	178
A.6	Embedded application development flow	179

List of Tables

4.1	Value of <i>sel</i> for inputs to CMP-MUX module.	64
4.2	FPGA implementation results for image size of 40×40	68
4.3	Comparison with existing hardware solutions for path planning	76
5.1	Notations	81
6.1	Control signal generation	110
6.2	Activation of mode-control signals and presentation of input data	114
6.3	Resource utilization for the FPGA design for the training set of 1024 faces	124
6.4	Recognition performance of FPGA-based face recognition system with external RAM for large training sets	126
6.5	Comparison with existing architectures	129
A.1	Virtex-4 XC4VFX12 resources	173
A.2	LCD signal wiring on the ML403 board	177

CHAPTER 1

Introduction

VLSI Array processors have long been used for executing computationally intensive tasks exhibiting a substantial amount of data parallelism. In an embedded system, such array processors can be used to perform computationally intensive sections of the algorithm. Array processors in general are application specific and can only be used to perform the tasks they are designed for. However, with intelligent system design, they can be scalable and used to perform multiple tasks. Vision-guided systems are one particular domain in which array processors can be really helpful in performing some of their core computations.

1.1 Motivation

The development of embedded systems for vision-guided systems that are able to assist humans during their everyday tasks has become a popular research area over the last few years [1]. Potential application scenarios for such vision-guided systems include assistance for elderly and physically-challenged people [2][3], general service at shopping centers and to entertain visitors at exhibition and function halls by a service robot. Another important field of application is automatic surveillance to monitor wide area of interest which is otherwise difficult to cover with fixed sensors. A key requirement for such

vision-guided embedded systems is processing of image frames from a video at a desirable rate. This necessitates the development of systems with dedicated hardware modules for performing the computationally intensive operations. Important functionalities of a vision-guided system are the ability to navigate based on the information about the surroundings and to recognize the objects and humans in its surroundings. One method of human recognition is using faces. Face recognition [4] is a suitable method of person identification for a guidance system since it is a non-contact method and the subject can thus be unaware that recognition is being performed.

1.1.1 Vision-Guided Embedded Systems

To intelligently interact with the world, vision-guided systems need to perceive and interpret the environment around them and react appropriately. Vision has a versatile capability useful for many tasks in guiding systems that include autonomous navigation, object recognition and human interaction. A vision system analyses the content of the images captured by a camera to carry out these tasks. Vision-based analysis provides a rich set of information about the environment for the guiding system. Vision system enables the guiding system to understand the environment for navigation and to recognize the objects in the environment as well as to determine their locations and orientations. Vision also allows the guiding system to interact with unknown surroundings robustly and fast. This thesis mainly focuses on the vision-based navigation and human face recognition to be used in a guiding system. A typical example where vision-guided embedded system would be used is a mobile social robot [5]. Such mobile robots have been used to detect and follow people [6].

1.1.2 Need for Hardware-Directed Algorithms for Navigation and Recognition

Numerous algorithms have been proposed in the literature for path planning and human recognition problems. However, most of them are designed to run in software in a general purpose processor whereas the target applications for such algorithms need them to be run on embedded platforms. In the case of dynamic environments, high-speed planning and recomputation of paths are necessary to avoid collisions of robots with moving objects especially when new objects enter the environment suddenly or when moving objects change their predicted course. In such cases, the computational requirement exceeds even the computing power of present-day general-purpose processors that implement the path planning algorithm.

For a vision-guided system looking for interaction with a person, it is necessary to detect and recognize the person's face in the video arriving at a rate of nearly 30 frames per second. In the case of video surveillance systems, the recognition process involves comparison of given face with the faces in the enrolled database. The recognition time depends on the size of the enrolled database which can be as large as thousands of faces (a watch list for example) in security applications at busy places like airports.

Though some hardware architectures have been designed in the past, the potential of array type architectures for executing such algorithms have not been explored thoroughly. Embedded processors are far less powerful than the processors on desktop PCs due to power, cost and size considerations and require additional co-processors to perform specific computation-intensive tasks. Therefore, design of scalable hardware co-processors that will operate along with embedded processors for executing computation-intensive tasks is important. Programmable hardware devices such as FPGAs nowadays provide

advanced features and resources to allow rapid prototyping of System-on-Chips [7].

1.2 Objectives

The main focus of this thesis is to explore the possibility of designing locally connected and scalable array type VLSI architectures for path planning and human recognition using face as the modality. The objectives of this thesis are enumerated below.

1.2.1 Cellular Architecture for Vision-Based Path Planning

- To propose a parallel algorithm for path planning that would take the binary image of an environment, start and goal points as inputs and would provide the sequence of steps to trace the path.
- To design a cellular architecture that is capable of constructing the euclidean distance map as well as performing path planning using breadth first search.
- To verify the correctness of the designed architecture using Verilog simulation and implementing it on an FPGA.
- To design a scalable architecture in which each cell is capable of processing multiple pixels and verify the architecture by simulation.

1.2.2 Systolic Architecture for Face Recognition

- To study the performance of modular PCA based face recognition and identify the ideal number eigenfaces, sub-images and wordlength for optimum recognition performance.

- To propose a systolic array realization of the Principal Component Neural Network (PCNN) based face recognition system.
- To design a processing element capable of performing different computations at different phases.
- To verify the systolic architecture using Verilog simulation and implement it on an FPGA.

1.2.3 Hardware-Friendly Algorithm for Face Detection

- To propose a hardware-friendly eigenface-based face detection algorithm using boosted eigen features.
- To reduce the number of computations required by proposing an FFT-based face detection solution.

1.3 Organization of the Thesis

The next chapter presents the details of previous work on path planning and face recognition with emphasis on hardware schemes.

Chapter 3 details the method of path planning on the binary image of the environment. The algorithm and the simulation results are presented here. In Chapter 4, the cellular architecture that maps the path planning algorithm onto hardware is described. Its implementation on an FPGA platform is also presented.

Chapter 5 presents the eigenface-based recognition using neural network. Performance studies on benchmark databases are given. The systolic design for the neural

network-based face recognition system is described with signal flow graphs in Chapter 6. Its implementation on an FPGA platform to analyse the real-time performance of the system is also given.

The eigenface-based face detection algorithm and a suitable hardware mapping are presented in Chapter 7. Chapter 8 summarizes this thesis and discusses the future directions of research.

The appendix presents the details of the FPGA board ML403 used for our experiments.

CHAPTER 2

Background and Literature Review

This chapter presents a review of existing literature on algorithms and architectures for path planning and face recognition problems. Finally, the fundamentals of systolic and cellular architectures and their existing designs for a few applications are reviewed.

2.1 Path Planning

Path planning is a fundamental task for an autonomous navigator by which it guides itself through the environment on the basis of sensory information. The potential of computational vision for robotic navigation is enormous and vision-based path planning has been actively studied in the last decade [8]. In simple terms path planning can be defined as the art of deciding which route to take to reach a specified goal point given the information about the environment. Path planning can be done on *a priori* information about the location of the obstacles and the autonomous navigator. But in realistic situations, very often the environment changes as the navigator is navigating through the planned path. Therefore path planning is a continuous exercise to be performed dynamically as the navigator moves along the planned path.

In order to plan a path for a navigator, it is essential to define the space in which it is supposed to navigate. A robot navigating in a $2D$ plane can be represented using a set

of Cartesian coordinates (x, y) . If the robot is capable of rotating, then we would need a third parameter θ to represent its current orientation. Therefore a navigator is said to have k *degrees of freedom* if its current *configuration* can be represented using k real numbers. The set of all possible values for the k real numbers is called the *configuration space*. The configuration space consists of *free space* and *obstacles*. Free space coordinates represent the part of the configuration space that the mobile robot is allowed to take position at. Obstacles represent the set of coordinates that the robot should not move into in order to avoid collision.

The size of the navigator determines the working space (or *collision free region*) available for it to navigate along the configuration space. If it's a point, then all of the available free space represents the collision free region. In real situations, the navigator will have a shape and occupy a reasonable space. We have introduced the notation of d_{far} in Chapter 3 to denote the distance from the centre of rotation to the farthest point of the navigator. Figure 2.1 shows the collision free region represented as white space for a point sized and square shaped navigators.

Figure 2.1: (a) Collision free region for a point sized navigator. (b) Collision free region for a square shaped navigator

2.1.1 Vision-Based Path Planning Algorithms

A detailed study of various path planning algorithms is provided in this section. The vision-based navigation and path planning work has progressed on two separate fronts.

1. Vision-based navigation of indoor robots where the complete knowledge of the environment is available *a priori* [9],[10],[11],[12].
2. Vision-based navigation of outdoor robots where partial knowledge of the environment is only often available [13],[14],[15],[16].

The common methods employed for path planning when the knowledge of the environment is available *a priori* are:

1. Visibility graph method[17]
2. Voronoi diagrams method[18],[19]
3. Cell decomposition method [20][21]
4. Artificial potential field method [22],[23]

The visibility graph is a graph of intervisible locations, for a set of points and obstacles in the Euclidean plane. Each node in the graph represents a point location and each edge represents a visible link between them. Visibility graph for simple polygonal obstacles is shown in Figure 2.2. The path from ‘start’ to ‘goal’ is the shortest path on the visibility graph which is shown by the thick lines in Figure 2.2. The shortest path traced using a visibility graph bends at the obstacle vertices. Therefore a path can be planned only based on the vertices information of the polygonal obstacles. However construction

Figure 2.2: Path planning using Visibility Graph

of visibility graph can be tricky if the obstacles have non-polygonal shapes and no specific vertices. A straightforward algorithm for the construction of visibility graph has low efficiency and time complexity is of the order of $O(n^3)$. Ghosh and Mount [24] provided an algorithm which computes the visibility graph of a set of n disjoint line segments in time $O(K + n \log n)$, where K is the number of edges in the visibility graph. As the path traversed is always very close to the obstacles, the chances of collision are high due to the dimensions of the navigator.

Voronoi diagram is a kind of decomposition of the configuration space into Voronoi cells, such that the boundaries of the cells are determined by the distances to the discrete set of objects present. Each cell consists of all the points that are closer to the object enclosed than any other object. The Voronoi diagram of point objects is shown in Figure 2.3. The segments of the Voronoi diagram consists of set of points that are equidistant to two nearest objects. The nodes of a Voronoi diagram represent the set of points that are equidistant to two or more objects in the configuration space. Therefore traversing

Figure 2.3: Voronoi diagram of points

along the segments will let the navigator to be as far as possible from the objects to avoid collision.

In Cell decomposition method, the configuration space is divided into non-overlapping grid cells. The cells are divided into three categories depending upon the presence of obstacles: empty cell, mixed cell and full cell. An empty cell doesn't contain any obstacles. A full cell is fully occupied by an obstacle and a mixed cell contains free space as well as obstacles. A collision free path is found by decomposing the map into cells and searching for a sequence of mixed or free cells that connects the start point to the goal point. The mixed cells are further decomposed and the steps are repeated until a sequence of free cells is found. Sequence of free cells is obtained by constructing a Quadtree. This is illustrated in Figure 2.4.

Figure 2.4: (a) Decomposition of the configuration space (b) Construction of a Quadtree

Potential field approach uses a scalar function called the potential. The potential function has a minimum at the goal point and high values on obstacle region. On other places, the function slopes down towards the goal point, so that the autonomous navigator can reach the goal point by following the negative gradient of the potential.

In [25], a sensor-based three-dimensional path planning algorithm was proposed to help underwater robotic vehicles perform real-time path planning in a static and unknown environment. An artificial immune network based path planning algorithm was proposed in [26] which is capable of achieving near-optimal collision free path in unknown environments that is presented by grids.

In [27], an algorithmic solution method is presented for the problem of autonomous robot motion in completely unknown environments. Their approach is based on the alternate execution of two fundamental processes: map building and navigation. In the former, range measures are collected through the robot exteroceptive sensors and processed in order to build a local representation of the surrounding area. This representation is then integrated in the global map so far reconstructed by filtering out insufficient or conflicting information. In the navigation phase, an A*-based planner generates a local path from the current robot position to the goal.

A system that builds a three-dimensional map of an indoor environment for a mobile robot is presented in [28]. The approach uses visual features extracted from stereo images as landmarks. A learning rule associated with each landmark is used to compute its existence state. New landmarks are merged into the map and transient landmarks are removed from the map over time. The location of the landmarks in the map is continuously refined from observations. The position of the robot is estimated by combining sensor readings, motion commands, and the current map state by means of an Extended Kalman Filter.

Robot motion planning using genetic algorithms is presented in [29]. It presents fast motion planning for robots with six or more degrees of freedom. It expresses the path planning problem as an optimization problem and shows that it can be solved using genetic algorithm.

2.2 Hardware Architectures for Path Planning

Most of the available hardware architectures for path planning are based on the two geometric structures, namely, visibility graph and Voronoi diagram.

2.2.1 Visibility Graph-Based Architectures

The authors in [30] presented an interesting approach for determining supporting segments for visibility graph construction based on the assignment of binary codes to the vertices of the objects. The approach is digital hardware-friendly. In [31], hardware-efficient schemes have been developed for key elements such as logarithmic approximation and binary search in the construction of basic visibility graph.

In [32], a hardware-directed algorithm has been proposed for the construction of the complete visibility graph. However, the visibility graph-based approaches assume the approximation of obstacles and robots by circumscribing polygons. The approximation requires considerable preprocessing. It is also necessary to transform the polygonal robot to a point and then expand the polygonal obstacles with respect to the polygonal robot before constructing the visibility graph. Since the visibility graph methods for path planning are not image based, they are computationally less intensive. The edges and vertices of the expanded polygonal obstacles are given as input for visibility graph construction.

In [32], the design for an environment with obstacles containing roughly 60 vertices has been accommodated on a XCV3200E device of Xilinx. The maximum frequency of operation is close to 60 MHz. The work in [33] proposed a custom architecture for the computation of the shortest path based on linear programming on a graph representing the environment. The design for a graph with 58 nodes and 82 edges has been fitted on one XC2V6000 device of Xilinx. The visibility graph and related approaches are model-based and they require modeling the environment before computing the graph. Moreover, the obstacles are approximated by polygonal shapes.

2.2.2 Voronoi Diagram-Based Architectures

Some designs of array-type architectures for the construction of Voronoi diagram on an environment image are available in the literature [34], [35]. In [34], a cellular architecture has been proposed for constructing a d_4 metric-based Voronoi diagram. The authors have applied it to path planning for a diamond shaped robot on a synthesised image containing simple obstacles. The diagram is constructed considering the interactions between features belonging to the same obstacle as well apart from those of different obstacles

and therefore it has extra branches. The Voronoi diagram is quite complex when an image of real obstacles is considered and the path planning on such a diagram is difficult. The cellular architecture for a 32×32 image size has been mapped on to an ASIC chip of dimension $2.35mm \times 2.29mm = 5.38mm^2$ which can be operated with a maximum frequency of 60 MHz. However, the actual time taken for the construction of Voronoi diagram has not been reported.

An algorithm for the construction of the exact Voronoi diagram based on Euclidean distance metric and its ASIC implementation were presented in [35]. The algorithm takes a binary image consisting of object and background pixels as input. Each object in the image is assumed to be a connected component, and the objects are disjoint. In order to construct the Voronoi diagram, each object is dilated iteratively, and the connectivity between the neighboring pixels belonging to the same dilated object is established at each iteration. The iterative dilation process is stopped when the entire image is occupied by the dilated objects. The algorithm considers the entire obstacle and not its features for the construction of Voronoi diagram and hence no extra branches. The algorithm is mapped onto a two-dimensional cellular architecture due to local neighborhood calculations on reduced bit-width data. Only 16 cells of the architecture have been implemented on an ASIC chip of dimension $3.16mm \times 3.16mm$ with a 50 MHz maximum operating frequency.

Recently, an FPGA based hardware solution has been proposed [36] for the construction of generalized Voronoi diagram using ultrasonic sensors. However, path planning has not been attempted using the diagram in any of these works. Very recently, an algorithm has been presented in [37] for computing the actual path on a binary image of an environment. The method constructs the path from the start to the goal but on the Euclidean distance transform of the image. A straightforward realization of the method

in hardware has been presented.

Though some hardware architectures for path planning have been developed in the past, they don't provide a complete solution for the problem. Most of them provide only intermediate solutions such as construction of visibility graphs and Voronoi diagrams. It is therefore necessary to design an efficient architecture for path planning that will take in an environment image as input and will provide the sequence of moves to reach the goal.

2.3 Overview of Face Recognition Algorithms

An autonomous navigator should be able to plan its path and also recognize humans in order to be interactive. Face is an important biometric modality that can be used to recognize a person. Face recognition can be defined as the process of identification of a person based on his facial image from a video or still image using a stored database of face images. Automatic identification of a person based on his/her physiological characteristics is gaining importance in recent years due to growing security concerns in most part of the world [38]. Since the human faces contain many identifying features of an individual, it has become one of the most popular ways for high-confidence authentication[39],[40]. It is a non-intrusive method, and is also applicable for building automated video surveillance and authentication systems.

Face recognition can be done both on still images as well as video frames. The face region has to be detected and extracted from the image or frame and then subjected to recognition algorithms to find the identity of the person. A detailed survey of face recognition algorithms is given in [41] which classifies the different approaches to perform face recognition into three broad categories. They are holistic approach, feature-based

approach, and hybrid approach.

2.3.1 Holistic Approach

In holistic approach, face recognition is performed by taking into account the whole face region. Generally the intensity values of the pixels comprising the face region don't tend to be processed easily to recognize a face. Therefore they are usually represented in a lower-dimensional domain which can be processed easily and faster. Examples of holistic methods are eigenfaces (most widely used method for face recognition), probabilistic eigenfaces, fisherfaces, support vector machines, nearest feature lines (NFL) and independent-component analysis approaches. They are all based on principal component-analysis (PCA) techniques that can be used to simplify a dataset into lower dimension while retaining the characteristics of the dataset.

Eigenface-based Methods

Eigenfaces are a set of standardized face components based on statistical analysis of various face images. Mathematically speaking, eigenfaces are a set of eigenvectors derived from the covariance matrix of a higher dimensional vector that represents possible faces of humans. Any human face can be represented by a linear combination of eigenface images. In Turk and Pentland's work [42], motivated by principal component analysis (PCA), the authors proposed a method, where principal components of a set of faces are extracted and used to represent the faces. PCA techniques are also known as Karhunen-Loeve methods which choose a dimensionality reducing linear projection that maximizes the scatter of all projected samples. The paper although over two decades old has become the classical literature on face recognition that authors often refer to it and compare their

results with it. The authors reported 96% correct classification over lighting variations, 85% over orientation variations and 64% over size variations. Zhang et al. [43] showed that eigenface based techniques deteriorates in performance when the lighting variations are not minimal.

Penev and Sirovich [44] discussed the problem of the dimensionality of the “face space” when eigenfaces are used for representation. They employed a perceptual criterion and argue that the dimensionality of the PCA subspace necessary for adequate representation of the identity information in relatively tightly cropped faces is in the range of 400-700. Yang et al [45] presented a new technique called 2D PCA for image representation. 2D PCA is based on 2D image matrices rather than 1D vectors so that the image matrix does not need to be transformed into a vector prior to feature extraction. The authors showed that using 2D PCA for the extraction of image features is computationally more efficient than using PCA.

Using a probabilistic measure of similarity rather than standard L2 norms or subspace restricted norms, Moghaddam et al [46] extended the standard eigenface approach to a bayesian approach. A large performance improvement of this probabilistic matching technique over standard nearest-neighbor eigenspace matching was reported using large face datasets including the FERET database.

Fisherface Method

Belhumeur et al [47] propose fisherfaces method by using Fisher’s Linear Discriminant Analysis (LDA) to produce subspace projection matrix similar to eigenspace matrix, but insensitive to large variation in lighting direction and illumination. One of the main drawbacks of eigenface approach is that the scatter being maximized can also be due to within-

class scatter, which can increase detection error rate if there is a significant variation in pose or lighting condition within same face images. Considering these changes due to different illumination and head pose are almost always greater than the variation due to difference in face identity, a robust detection system should be able to handle the problem. Since the fisherfaces approach takes advantage of within-class information, minimizing variation within each class, yet maximizing class separation, the problem with the variations in the same images such as different lighting conditions can be overcome.

Both PCA and Fisher's LDA have been established to be useful under different circumstances. However, PCA has been shown to be superior to LDA particularly when the number of training samples per class is small. Furthermore, for surveillance type of applications, only one training sample may be available for each subject and here LDA is not suitable since the within-class scatter matrix turns out to be a zero matrix.

Support Vector Machines

Support Vector Machines (SVM) were suggested for face recognition in [48]. Face recognition is a K class problem where K is the number of individual faces present in the database. But, SVM is a binary classification method. Therefore, the authors applied SVM to face recognition by reformulating the recognition as a problem in *difference space*. In this space, the two classes are the dissimilarities between the faces of the same person and that of different persons. SVMs incorporated with a binary tree recognition strategy are proposed to tackle the multi-class face recognition problem in [49].

Nearest Feature Line Method

In [50], the authors present Nearest Feature Lines method (NFL) for face recognition, which is one of the holistic matching methods to deal with problems in eigenfaces approach. To create a feature point in feature space, it is assumed that there should be at least two prototype feature points available from the same class (image). A line passing the two prototype features forms a Feature Line (FL) that generalizes the two feature points. A feature line represents an approximate of two prototypes (images) that are captured under different conditions, such as different head gaze direction or different light illumination. An input image is then identified with a corresponding class, according to the distance between feature point of the given image and FL of the prototype images.

2.3.2 Feature-Based Approach

In feature-based approaches, local features on face such as nose and eyes are segmented and then used as input data for a structural classifier. Pure geometry, dynamic link architecture, and hidden Markov model methods belong to this category. Reference [51] presents a feature based approach to face detection, where features are derived from intensity data without assuming any knowledge of the face structure. The model they employ to extract those features is biologically motivated and the location of the features generally correspond to salient face features like eyes, nose, etc. The authors use topology graphs to locate these facial features. Reference [52] presents a feature based face recognition method for a setting in which only one face image is available for each individual for training. A novel distance function is constructed based on local second order statistics as estimated by modeling the training data as a mixture of normal densities.

Wiskott et al [53] presented a face recognition technique using Elastic Bunch Graph Matching method. Faces are represented by labeled graphs, based on a Gabor wavelet transform. Image graphs of new faces are extracted by an elastic graph matching process and can be compared by a simple similarity function. Image graph extraction is based on a novel data structure, the bunch graph, which is constructed from a small set of sample image graphs.

Hidden Markov Models Method

Hidden Markov models (HMM) is another promising method that works well for images with variation in different lighting, facial expression, and orientation. A hidden Markov model (HMM) is a statistical model in which the system being modeled is assumed to be a Markov process with unobserved state. A Markov process is a time varying random phenomenon for which the Markov property holds i.e., if the conditional probability distribution of future states of the process depend only upon the present state. Prior to the application of HMM to face recognition, they were successfully applied to speech recognition where the data is only 1-D. The system being modeled is assumed to be a Markov process with unknown parameters, and the goal is to find hidden parameters from the observable parameters. Each state in HMM has a probability distribution over the possible output whereas each state in a regular Markov model is observable.

In [54], the authors use HMM approach for face recognition based on the extraction of 2-dimensional discrete cosine transformation (DCT) feature vectors. The authors take advantage of DCT compression property for feature extraction.

Later a new system for face recognition was proposed based on Hidden Markov Models (HMMs) and wavelet coding in [55]. Here, a sequence of overlapping sub-images

is extracted from each face image, computing the wavelet coefficients for each of them. The authors have employed the non standard decomposition, which alternates between row and column processing, allowing a more efficient computation of coefficients. The proposed algorithm calculates the coefficients representing the image with a normalized two-dimensional Haar basis, sorting these coefficients in the order of decreasing magnitude.

Face recognition using Embedded HMM (EHMM) was proposed in [56]. The proposed method partitions the face image into a 2D lattice type, composed of many blocks. Each block is represented using the second-order block-specific observation consisting of a combination of first and second-order feature vectors. EHMM is trained by a sequence of features obtained from top-to-bottom and the left-to-right scanned blocks. Recognition is performed by identifying the person of the model that provides the highest value of observation probability.

2.3.3 Hybrid Approach

The idea of this method comes from how human vision system perceives both local feature and whole face. There are modular eigenfaces, hybrid local feature, shape-normalized, component-based methods in hybrid approach.

Pentland et al. [57] extended the capabilities of their earlier system [42] in several directions. They generated a modular representation of the face using a layered approach (eigenfaces and eigenfeatures). The additional layer of description in terms of facial features like eyes, noses, lips are called the eigen features. A coarse (low-resolution) description of the whole face is augmented with additional (high-resolution) eigenfeatures, thereby yielding better performance. The system showed 95% recognition rate on the

FERET database, which contains 7562 images of approximately 3000 individuals. But they only used a small subset of this database containing a representative sample of 45 individuals.

Local Feature Analysis (LFA) for deriving local topographic representations for any class of objects was presented in [58]. The authors had suggested it as an alternative to PCA which is a global and nontopographic representation not amenable to further processing. LFA provides the description of the object in terms of statistically derived local features and their positions. Lanitis et al. [59] presented a face identification system using *Flexible Appearance Models*. These models are controlled by a small number of parameters which can be used to code the overall appearance of a face for image compression and classification purposes. The model parameters control both inter-class and within-class variation. Discriminant analysis techniques are employed to enhance the effect of those parameters affecting inter-class variation, which are useful for classification.

Another hybrid approach for component-based face recognition using 3D morphable models was given in [60]. A 3D morphable model is used to compute 3D face models from three input images of each subject in the training database. The 3D models are rendered under varying pose and illumination conditions to build a large set of synthetic images. These images are then used for training a component-based face recognition system.

Comparative studies have been performed by many researchers [61, 47, 62, 63]. These studies show that each method performs better than others in some respects. PCA turns out to be a very suitable for security-related applications since only few samples per subject, as in the case of mugshot face database, may be available.

2.3.4 Face Detection Algorithms

Face detection is an expensive pre-processing step for face recognition. It is essential for localizing and extracting face region from the background. It also has other applications like face tracking, automatic focal length adjustment in a digital camera, video conferencing, surveillance etc. As human face is a dynamic object subject to a wide range of variations and movements, face detection is one of the difficult problems in computer vision. This section provides a brief overview of face detection algorithms available in the literature.

A detailed survey of face detection algorithms is presented in [64]. Turk and Pentland [42] introduced “distance from face space” (DFFS) measure for face detection. Pentland et al [57] later proposed a face detector using the DFFS measure obtained from a combination of eigenfeatures (eigeneyes, eigennoses, eigenmouths). DFFS measure has also been used along with Fisher’s linear discriminant analysis for face and facial feature detection in [65]. The same measure was used along with colour, 3D and motion information for face detection and tracking in [66]. A similar approach is presented in [67] where PCA has been applied for modeling both the class of faces and the class of pseudo-faces (nonfaces, but face-like patterns), together with matching criteria based on a generalized likelihood ratio.

Neural network has also been used for face detection. Some fundamental neural network based face detection algorithms have been proposed in [68, 69, 70]. The first advanced face detection method using retinally connected neural networks with results on a larger dataset was given in [71]. Hidden Markov Models have also been applied for face detection in [72], [73].

Viola and Jones [74] proposed a fast and robust face detection algorithm using sim-

ple haar like features. A strong classifier was built using Adaboost algorithm from a set of weak classifiers obtained from haar like features and detection was performed using cascaded stages. They also introduced the concept of integral image to increase the computation efficiency of the algorithm. Lienhart and Maydt [75] extended the boosted cascade by introducing novel set of rotated haar-like features. A thorough analysis of the detection performance and computational complexity of different boosting algorithms (namely discrete, real and gentle Adaboost) has been done in [76].

As PCA-based face recognition and detection are explored in this thesis, a short mathematical primer for PCA is presented below.

2.3.5 PCA for Face Recognition and Detection

PCA is a mathematical tool to extract eigenfaces used in face recognition and detection. PCA is defined as an orthogonal linear transformation which transforms the data set into a new co-ordinate system such that the data has maximum variance along the first co-ordinate and the variance is in decreasing order along the successive co-ordinates. Such a transformation helps to reduce the dimensionality of a given dataset. Principal Component Analysis can be performed by computing the eigenvectors of the co-variance matrix of the data set arranged as a matrix.

Let us consider a face image of size $N \times N$ with N^2 pixels and each image can be represented by a vector of size N^2 . In other words each image will be a point in N^2 dimensional face space. Let the database contain K images. As the images of faces have certain common features, these vectors will not be distributed randomly in the face space. Therefore they can be described using a smaller dimensional sub-space and the aim of the PCA is to find the lower dimensional representation of the huge face image data. Let the

training database have K face images $\Gamma_1, \Gamma_2, \Gamma_3, \dots, \Gamma_K$. Let the average face be defined by $\Psi = \frac{1}{K} \sum_{n=1}^K \Gamma_n$. For dimensionality reduction using PCA, the data set i.e., the image data in this case has to be centered. Therefore, the mean face image is subtracted from each of the image data to center the image set. Let the centered images be represented by $\Phi_i = \Gamma_i - \Psi$.

The aim of the Principal Component Analysis is to describe the above large data by a set of K orthonormal vectors u_k , which best describe the data distribution. The k^{th} vector u_k is chosen such that

$$\lambda_k = \frac{1}{K} \sum_{n=1}^K (u_k^T \Phi_n)^2 \quad (2.1)$$

is a maximum, subject to

$$\mathbf{u}_l^T \mathbf{u}_k = \delta_{lk} = \begin{cases} 1, & \text{if } l=k \\ 0, & \text{otherwise} \end{cases} \quad (2.2)$$

The vectors $\mathbf{u}_k$ and scalars λ_k are the eigenvectors and eigenvalues, respectively of the covariance matrix given by the following equation

$$C = \frac{1}{K} \sum_{n=1}^K \Phi_n \Phi_n^T = AA^T \quad (2.3)$$

where the matrix $A = [\Phi_1, \Phi_2, \dots, \Phi_K]$. The matrix C is of the size N^2 by N^2 and so determining N^2 eigenvectors become intractable as the size of the image increases. If the number of images in the database (K) is smaller than the number of pixels in the image i.e., $K \ll N^2$ which is usually the case, the number of useful eigenvectors will only be $K - 1$. The other eigenvectors will have an associated eigenvalue of 0.

The eigenvectors arranged in a 2-D array are called eigenfaces. The eigenfaces span a basis set with which the face images can be described. But for all practical purposes, the first few basis vectors provide a reasonably good reconstruction of the original image. Therefore only a reduced set of K' bases are sufficient to describe the face image. The K' bases are chosen such that they have the largest associated eigenvalues. A given face image Γ is transformed into its eigenface components by the following simple operation,

$$\omega_k = \mathbf{u}_k^T (\Gamma - \Psi) \quad (2.4)$$

for $k = 1, \dots, K'$. These projections or weights ω_k form a vector $\Omega^T = [\omega_1, \omega_2, \dots, \omega_{K'}]$ and each weight ω_k represents the contribution of k^{th} eigenface. Eigenfaces can be used for face recognition by measuring the distance between the projections of faces from query set and target set. The Eigenface-based face recognition is robust to slight translation, rotation, scale changes and high blurring effects, but the performance maybe affected by significant local variations such as expression changes and directional lighting. These limitations can be overcome by using a local Eigen analysis method insensitive to local variations.

Eigenfaces can be used for face detection by measuring the distance between the different sub-images of a given image and the face space and creating a distance map. The distances in the distance map can then be compared to a threshold to find the location of face. The sub-images containing faces will have lower values on the distance map. The distance between the image and a particular eigenvector u_k is given by $\Phi_{fk} = \omega_k u_k$. The distance between the image and face space is given by the sum of squares of difference between the mean adjusted input image $\Phi = \Gamma - \Psi$ and its projection onto face space $\Phi_f = \sum_{i=1}^{K'} \omega_i u_i$ given by the following equation

$$\epsilon^2 = \| \Phi - \Phi_f \|^2 \quad (2.5)$$

The distance from each of the eigenvectors is given by the following equation

$$\epsilon_k^2 = \| \Phi_{fk} - \Phi_f \|^2 \quad (2.6)$$

2.4 Hardware Architectures for Face Recognition

In real-time face recognition, it is necessary to detect and recognize faces in the video images arriving at a rate of nearly 30 frames per second. Moreover, the recognition process involves comparing the given face with the faces in a database. The recognition time depends on the size of the database which can be as large as thousands of faces (a watch list for example) in security applications at busy places like airports. In order to meet the real-time performance, specialized hardware for face detection and recognition is important. Since the face recognition algorithm is computation-intensive, specialized hardware is desirable for power-efficient realization of the face-based access control to low power personal devices. When face recognition is used as the method of human recognition by autonomous navigators, dedicated hardware architectures are essential due to the wide variety of algorithms needed to be executed by the navigator.

A few work has been done on hardware/software co-design of face recognition schemes. In early 90s, Gilbert and Yang [77] proposed a hardware/software co-design based on template correlation. For the hardware part of the system a VLSI image correlator has been designed in this paper. The preprocessing and post-processing steps in [77] are performed in software, where the preprocessing involves the detection and normal-

ization of the face in the image captured by the camera, and the post-processing involves comparing the face with the ones in the database using the correlation scores computed by a correlator chip. Since the post-processing time depends on the size of the database and it is performed in software, the system could achieve real-time performance only for small and moderately sized database of about 500 images but not for large databases.

Recently, two other hardware-software combined designs for eigenface-based recognition have been proposed in [78],[79] where eigenfaces are computed in software and stored in ROM. Besides, the projections of face images in database onto the face space spanned by eigenfaces are also computed in software. A parallel hardware architecture is designed only for the recognition of a new face. However, when the face database changes, the eigenfaces and projections are recomputed and reloaded manually. Some hardware architectures have been developed for face detection as well. A parallel architecture for face detection using haar-like features has been proposed in [80]. An embedded hardware face detection system using neural networks has been developed in [81]. An architectural level design methodology for embedded face detection has been presented in [82].

In the case of authentication systems, whenever a person authenticates, his faces in the database should be automatically replaced and the eigenfaces of the new database should be computed. Prior architectural efforts, to the best of our knowledge, have not examined the computation of eigenfaces in custom hardware. In this thesis, we propose an array type VLSI architecture that extracts eigenfaces as well as recognizes a new face. As an eigenface based approach has been explored for face recognition, a similar approach for face detection will reduce the computational complexity for the overall face detection and recognition system. It is therefore necessary to develop hardware-friendly algorithm for face detection that will have the simplicity of eigenface based approach and increased

detection performance offered by boosting based algorithms.

2.5 Array Architectures

Both navigation and recognition tasks of vision-based autonomous systems are known to be computationally intensive. Several research work has been done so far in designing algorithms and architectures for these tasks mainly due to a rich set of commercial applications. Inherent limitations posed by the uniprocessor architectures especially in catering to the real-time requirements have led to the development of multiprocessor architectures for executing vision algorithms. From the algorithmic point of view, they are developed to fully utilize all the architectural features while striving to give better results. From the architectural point of view, the architecture with multiple processing elements should have low hardware complexity, and preferably, be composed of components that can be easily replicated thus making it suitable for VLSI implementation [83].

In this section, we provide the background of two types of VLSI architectures, namely systolic and cellular architectures. A survey of these architectural designs for signal and image processing applications is also presented. In cellular and systolic architectures, simple custom processing elements are replicated systematically. Such architectures increase the throughput of the computing system through concurrency.

2.5.1 Systolic Architectures

Background

Any computational task can be formally classified into two families — compute-bound and I/O-bound [84]. Those tasks which perform more number of computations than the number of I/Os are compute-bound and those which perform less computations than the number of I/Os are I/O-bound. Compute-bound tasks can be solved by employing systolic architectures. Systolic architecture is a network of simple and identical processing elements arranged regularly in an array, connected by localized communication links. Each PE computes rhythmically and pumps data synchronously in and out such that a regular flow of data is maintained across the array for a fully pipelined computation.

As a single data passes through the networked processing elements, several computations can be performed for every I/O access from memory. Systolic architectures are easily scalable by simply increasing the number of processing elements as each processing element is identical and has same I/O behavior. A conventional architecture has to load and store data from memory for every operation it performs. Caches provide a way to overcome frequent I/O operations to some extent. But as the number of computations and data increase, the separation between CPU and memory leads to *von Neumann bottleneck*. Systolic architectures perform multiple computations per memory access and thus speed up compute-bound computations without increasing I/O requirements.

Systolic mapping provides a systematic method to transform an algorithmic description into an architecture. This involves several steps like expressing the sequential algorithm in a parallel form, appropriate graphical representation of the operations and mapping them onto the architecture. A Dependence Graph (DG) is drawn to represent the

algorithm which is a directed graph. Every node in this graph represents a computation and every edge represents the precedence constraints among the nodes. An algorithm represented using a DG is computable if and only if the DG doesn't contain any loops. Figure 2.5 shows a DG representation of a matrix vector multiplication $c = Ab$ (A is a 3×3 matrix, b and c are 3-point vectors). The figure shows the input/output behavior of the node and arrows in the graph denote the dependencies.

Figure 2.5: Dependence Graph for a matrix-vector multiplication and description of a node

From the DGs, Signal Flow Graphs (SFG) are derived. An SFG consists of processing nodes, communicating edges and delays along the communication edges. The nodes of an SFG denote the processors in the array. Nodes of a DG which are along a straight

line are assigned to a common processing element in the SFG. The projection of nodes in DG onto processors in SFG is guided by projection rules which make sure that timing precedence constraints are met. The projection of nodes of the DG shown in Figure 2.5 along the horizontal direction is shown in Figure 2.6. All three nodes along the horizontal direction are mapped onto one processor as shown without affecting the precedence constraints shown in the DG.

Figure 2.6: Projection of dependence graph onto signal flow graph

Systolic designs for applications

Reference [85] was one of the earliest works describing the systolic processing and implementation for signal and image processing. The systolic concept offers guidelines in building a cost-effective system which balances the I/O with computation. The authors described a linear systolic array capable of performing a large class of inner-product based functions used in signal and image processing applications. Applications like matrix multiplication, multi-dimensional convolution using fixed or time-varying kernels as well as various non-linear functions of vectors were included. The basic cell structure, algorithms

and applications based on that structure and a system organization of a working prototype were all described. The authors have emphasized that with appropriate architecture support and suitable software interface, a systolic processor for signal and image processing can provide both the performance of special-purpose hardware and versatility of more generic systems.

The architecture, implementation and performance of Warp computer—a systolic array of linearly connected cells each with a capability of performing more than 10 million floating point operations per second (MFLOPS) was presented in [86]. The Warp machine was attached to a host processor running Unix operating system and is accessed through a programmable command interpreter called the Warp shell. Each cell in the array communicates with its left and right neighbor cells through point-to-point links. In [87], automatic mapping of large signal processing systems to the Warp computer was presented.

A new two-dimensional systolic array for image processing and neural-network applications was proposed in [88]. The systolic array presented is capable of performing two-dimensional convolution with kernels sized larger than the physical array of processing elements. The array is particularly well suited for neural network image processing algorithms that use large connected neighborhoods to model the transformation between layers of neurons. A conventional linear memory architecture for storing and accessing data from the external memory is used.

A two-dimensional systolic array processor with its design and implementation for applications in image processing and machine vision has been described in [89]. The processor architecture is based on an SIMD array of 4-bit processing elements interconnected by a mesh network of 4 nearest neighbors. The PE array is programmable allowing

the user to develop application specific algorithms for performing analysis on image data. Each PE has a single 4-bit arithmetic/logic unit (ALU), 4×4 single cycle multiplier, 4-bit arithmetic/logic shift unit, and a 256-word by 4-bit static RAM per PE.

Reference [90] has presented an overview of systolic processors applied to machine vision systems. The design of reconfigurable systolic processor for iconic processing of images in real time (video rate) has been described. The authors have presented the components of the processor like basic cell structure, delay elements and interconnection network with a special attention to the cell architecture. The basic cell is made up of a RAM memory, an ALU, a latch of 13 bits and a bypass latch of 8 bits. The RAM is of 28×13 bits. It is directed by the pixel value coded into 8 bits and the resulting value is put into the ALU. The width of data path in ALU is 13 bits. The authors have also described the different configurations in which the processor would work like 3×3 convolution, 3×3 pseudo-median filtering etc.

A systolic algorithm and architecture for image thinning has been presented in [91]. The architecture described is based on an algorithm that manages to achieve maximum level of parallelism. The algorithm proposed in this paper computes the skeleton of multiple objects in an image in linear time by making two scans over the 4-distance transform of the image. The algorithm has been mapped onto a linear systolic array of simple processing elements (PE) and for an $N \times N$ image, the architecture requires N processing elements. The total time to fill the pipeline is $6N$ cycles for an array N PE's and $N \times N$ image.

Systolic arrays for 2-D decimation filters have been described in [92]. Decimation filtering is a subset of convolution in which the convolution of an image includes decimation filter products. The array proposed by the authors completes the operation in time

proportional to the image size and the array size is proportional to the window size. The authors have modified existing systolic architectures for convolution to produce dedicated decimation filters which have reduced number of processing elements. It should also be noted that the design exhibits minimum latency for the algorithm used. The array architecture has been derived using a multi-rate array synthesis technique which begins with an algorithm expressed as a system of affine recurrence equations and involves reducing the dimensions of the array size while ensuring that the array is not proportional to the problem size.

A high speed systolic architecture for labeling connected components in an image has been described in [93]. Connected components detection and labeling is a very important step in many of the image processing and machine vision algorithms. The efficiency of the processing of this step is essentially going to affect the overall performance of the algorithm. The authors have utilized the advances in the field of parallel processing and VLSI technology to design algorithms and hardware architectures for high speed and better throughput. The algorithm processes the image in a row-wise manner and requires N processing elements to process an image with row size of N pixels. The algorithm requires two rows of the image at a time and makes two passes over the image. The basic architecture consists of N processing elements connected linearly each with 3 modules corresponding to different steps in the algorithm. For processing a $M \times N$ image, the architecture takes $3N + MN$ cycles for the first pass and $2N + MN$ cycles for the second pass. Hence the total processing time required is $5N + 2MN$ clock cycles. The extended input image to the systolic array is provided by the host.

In [94], the authors have presented a custom computing approach to meet the computation and communication requirements of computer vision algorithms. They have described the mapping of vision algorithms like 2D convolution, image segmentation and

fingerprint matching onto Splash2 — a custom computing platform built with a series of Xilinx FPGAs. The authors have argued that the reconfigurability of FPGAs can be exploited to perform different levels of vision tasks.

A systolic array architecture for Gabor decomposition has been provided in [95]. It is a combined systolic-array and content addressable memory architecture for real-time Gabor decomposition. Gabor decomposition can be computed as a product of a Gabor transform matrix and a vector of image data. The architecture has N processing elements for an image size of $N \times N$ each built with a content addressable memory. Systolic arrays are attractive for matrix multiplication and content addressable memories provide fast memory access and hence a combined architecture offers a potential for real time execution of Gabor decomposition. Each systolic array module is capable of performing 1-D Gabor decomposition and hence two such modules are needed for 2-D Gabor decomposition. Each PE is a simple inner-product processor and has a local content addressable memory of size $2p$ and p being the bandwidth of the Gabor kernel.

A generic systolic processor for real-time grayscale morphology has been described in [96]. Mathematical morphology is a powerful and widespread tool for image analysis and coding, but for large analysis neighborhoods, it becomes increasingly time-consuming on general purpose machines. The systolic architecture described in this paper is able to perform erosion, dilation, openings and closings during a unique image pass whatever the neighborhood shape and size. A simple global control unit is sufficient for this system. It realizes the address management and write-enable signal generation at the beginning of each row.

An FPGA based customizable systolic architecture for image processing applications has been presented in [97]. The architecture described is customizable providing the

possibility to perform window operations for masks of 3×3 , 5×5 and 7×7 coefficients. A 2D systolic array of processing elements have been implemented, based on parallel modules with internal pipeline operation where every processing element can be configured according to a control word. In addition, the array is provided with a group of image buffers to reduce the number of access to data memory and to extend the array capabilities allowing the possibility of chaining interconnection of multiple processing blocks. Every buffer constitutes a repository of data that can be reused for different processing blocks. Each PE has been specially designed to support the operations involved in most window-based operators in image processing. Moreover this basic architecture can be modified to support different operations, varying the ALU complexity or adding a storage element to allow processing with local data.

Systolic designs for DCT using a low-complexity concurrent convolutional formulation have been explained in [98]. Two variants of systolic structures using ROM-based multipliers are presented for efficient implementation of the proposed algorithm. The proposed structures are found to offer significant saving of hardware, require less latency, and yield more throughput over the existing structures. Apart from simplicity and regularity, the proposed structures would also have flexibility of implementation by CORDIC circuits and canonical-signed-digit-based multipliers as well.

A systolic architecture based low-level image processing algorithms accelerator platform has been presented in [99]. The architecture is customizable providing the possibility of performing window operations with masks of variable size and every processing element in the array can be configured according to a control word. The architecture comprises a scheme to reduce the number of accesses to data memory and router elements to handle data movement among different structures inside the same architecture.

2.5.2 Cellular Architectures

The cellular architecture is a 2D form of linear systolic architecture. The architecture does not have any hardware overhead as it is not pipelined. It is simple in design, modular, cascadable and has high speed of operation due to local interconnection. Each cell is a processing element with combinational logic and storage elements designed according to the requirements of the algorithm. The architecture is based on the concept of cellular automata [100],[101],[102]. The values stored in a cell represent the local state of the cell and the global state of the architecture is represented by the vector of local states. All cells are operated synchronously, and the architecture executes the algorithm in discrete time steps. Figure 2.7 shows a cellular architecture in which each cell is connected to all its neighboring cells. Two dimensional cellular architectures are feasible for image processing [103].

Typical Applications of Cellular Architecture

Neighborhood operations are one of the most common operations in image preprocessing. They use a small neighborhood of a pixel in an input image to get a new gray or brightness value in the output image. The usual neighborhood window is of 3×3 size encompassing 8 pixels around a pixel under consideration. The window size can be larger depending upon the needs of the algorithm used. Such preprocessing operations are generally termed as filtering. Typically in a cellular architecture, each cell performs the computations required by a single pixel. Since each cell is connected to all the neighboring cells, it has easy and fast access to their values thereby making computations faster. Further, the local neighborhood operations of all pixels can be done in parallel. Some of the other preprocessing operations like inversion and binary operations are called point operations as the

Figure 2.7: A 2-D cellular architecture

output pixel value is dependent only upon the input pixel value. Cellular architectures are ideal for implementing point and neighborhood operations.

Architecture Design

The algorithm is first analyzed to identify different kinds of computations involved and the type of operations, whether point or neighborhood or both. A pixel or a group of pixels can be assigned a cell. The processing element or the cell is designed with combinational and sequential elements, so that the desired output can be achieved while the cells are operated in parallel. Also, each cell receives data from its neighboring cells and passes its contents to them. The order of execution is analyzed carefully so that the cells have desired output after the required cycles of operation. If the algorithm involves different phases, the same

cell can be used to perform all of them. Separate control logic is designed to instruct the cell about the phase of operation. Multiplexers and de-multiplexers are used within the processing element to select the different inputs based on the phase of operation or to channel the inputs to desired computation units in the processing element. The control unit can be designed either by using a finite state machine or by using micro-programmed lookup table depending upon the complexity of the processing element.

Generally, the complexity of the processing element is much less compared to that of an instruction-set processor. The processing element is not designed to support instruction-set based computations, but just provides fast computations for specific operations. Same processing element can be used to perform different computations with added control hardware. But, since the control logic is simple and the cellular architecture provides massive parallelism, algorithms can be executed several hundred times faster than a normal instruction-set processor.

2.6 Conclusions

This chapter has presented an overview of path planning algorithms and architectures first. The merits and demerits of the existing algorithms and the need for an algorithm that can be easily mapped onto an array-type architecture have also been discussed. The face recognition problem has been introduced and different types of approaches available for solving the problem have been discussed. Finally a comprehensive overview of systolic and cellular architectures for some related applications has been provided. With this background, the consecutive chapters explain our work in detail. The algorithms and architectures developed and implemented for path planning, face detection and recognition are discussed in-depth.

CHAPTER 3

Path Planning on Binary Image of an Environment

Path planning refers to the preparation of a path to be followed by an autonomous navigation system. The navigation system has to typically move through a terrain with objects interspersed. An algorithm to plan the path to be traced by the system to reach its goal will need an input about the environment. The input should clearly describe the position of the objects that might interfere with navigation.

An image of the environment captured by an overhead camera can describe the environment precisely. For the purpose of path planning, only the location and size of the objects are necessary to trace a path that would guide the navigation system to move around without colliding with the objects. As texture and colour details of the objects are not necessary, a binary image representation of the environment is sufficient. Therefore the grayscale or colour image captured by the overhead camera is converted to a binary image in the preprocessing stage. The proposed method then constructs a distance map for the binary image to determine the collision-free region. The shortest path is finally constructed in the collision-free region.

3.1 Construction of Distance Map

For a binary image of obstacles, the Euclidean Distance Transform (EDT) of the image is first computed. The EDT converts the binary image to a multivalued image in which each pixel p is assigned the Euclidean distance between p and the nearest obstacle pixel. A parallel algorithm which is amenable to VLSI implementation is presented in [104]. The salient feature of the algorithm is that the computation of EDT involves only integer arithmetic operations within a small neighborhood of each pixel and therefore it is suitable for mapping onto a high-speed array architecture. The algorithm is summarized here. The algorithm computes the distance vector $(\Delta x, \Delta y)$ for each pixel where Δx and Δy are the number of rows and columns by which a pixel is displaced from its nearest object pixel. The Euclidean distance d is given by $\sqrt{\Delta x^2 + \Delta y^2}$. $(\Delta x, \Delta y)$ of object pixels are initialized to (0,0) and those of free-space pixels are computed iteratively starting from the pixels near the object and moving towards the far away pixels.

At any iteration k , $(\Delta x(p), \Delta y(p))$ of those pixels p whose nearest integer approximation to their Euclidean distance $d(p)$ equals k are computed. That is, the values of $d(p)$ of these pixels lie within $(k - 0.5, k + 0.5]$. $d(p)$ is not an integer and hence $d^2(p)$ is considered. $d^2(p)$ lies within $(k^2 - k, k^2 + k]$ since $d^2(p)$ is an integer. However, $d^2(p)$ is quite large in magnitude and it requires a large storage space in hardware. A new integer quantity $\delta(p)$ which is much smaller than $d^2(p)$ is defined as $(k^2 + k) - d^2(p)$ and it lies in the range $[0, 2k)$. δ is used for the computation of $(\Delta x, \Delta y)$. $d^2(p)$ can be derived using the already computed $(\Delta x, \Delta y)$ of eight neighbors p_i , $i = 1$ to 8, surrounding p as

in Figure 3.1. It is given by $\min[\Delta x_i^2 + \Delta y_i^2]$ where

$$\Delta x_i = \begin{cases} \Delta x(p_i) & \text{for } i = 1\&3 \\ \Delta x(p_i) + 1 & \text{otherwise} \end{cases} \quad (3.1)$$

The increment by 1 is due to p being displaced from p_i by one row. Similarly, Δy_i is given in terms of $\Delta y(p_i)$. $d^2(p)$ is now expressed as $\min[d^2(p_i) + \Delta X_i + \Delta Y_i]$ where

$$\Delta X_i = \begin{cases} 0 & \text{for } i = 1\&3 \\ 2\Delta x(p_i) + 1 & \text{otherwise} \end{cases} \quad (3.2)$$

$\delta(p)$ is finally expressed as follows.

$$\begin{aligned} \delta(p) &= k^2 + k - d^2(p) \\ &= \max[k^2 + k - d^2(p_i) - \Delta X_i - \Delta Y_i] \\ &= \max[\delta(p_i) - \Delta X_i - \Delta Y_i] = \max[\delta_i] \end{aligned} \quad (3.3)$$

$\delta(p) \geq 0$ implies $d(p)$ is less than $k + 0.5$.

p_7	p_8	p_1
p_6	p	p_2
p_5	p_4	p_3

Figure 3.1: Neighborhood N of p

The iterative computation of $(\Delta x, \Delta y)$ proceeds as follows. δ values of the object pixels are initialized to 0. At each iteration k , δ values of free-space pixels whose $(\Delta x, \Delta y)$ are not yet known are computed using already computed δ , Δx and Δy of

neighbors. If $\delta(p) \geq 0$, then $(\Delta x(p), \Delta y(p))$ corresponds to $(\Delta x_i, \Delta y_i)$ where subscript i pertains to the pixel p_i that gives maximum δ_i in Equation (3.3). p can be assigned an integer distance d_I which is equal to k . This integer distance is sufficient to determine the collision-free region for the given autonomous navigator.

Once $(\Delta x, \Delta y)$ of a pixel is known, its δ should be updated for at least two successive iterations, as it depends on k . The updating allows the use of δ for the computation of $(\Delta x, \Delta y)$ of neighbors. $\delta_k(p)$ at iteration k is derived from $\delta_{k-1}(p)$ at iteration $k - 1$ as follows.

$$\begin{aligned}\delta_k(p) &= k^2 + k - d^2(p) \\ &= 2k + [(k-1)^2 + (k-1) - d^2(p)] \\ &= 2k + \delta_{k-1}(p)\end{aligned}\tag{3.4}$$

To keep track of pixels whose $(\Delta x, \Delta y)$ have been computed, a flag *done* is assigned to each pixel, whose value is set to 1 when the transform values of pixels are computed at any iteration. The $d_I(p)$ given by the iteration number k , when *done* is set, forms the distance map.

3.2 Path Planning on the Distance Map

The distance map is used to find first the collision-free region for the autonomous navigator. Let d_{far} be the distance between the center of rotation and the farthest point of the autonomous navigator from the center. The collision-free region consists of those pixels whose distance values (d_I) are greater than d_{far} . Consider the collision-free region as a graph. Each pixel is represented as a node and it is connected to the eight neighboring

pixels surrounding it by edges.

The construction of the shortest collision-free path involves the construction of the breadth-first search (*bfs*) tree on the collision-free graph with its root being the goal pixel. The construction of *bfs* tree is continued until the start pixel is encountered. The construction involves storing the parent of each pixel. The path is then traced by following the parent nodes from the start pixel until the root node is reached. The path constructed by this method on the image is the shortest path in terms of the number of pixels. The autonomous navigator is moved from one pixel to the next along the path by aligning its center of rotation to every pixel.

The method described above is illustrated in Figure 3.2. Consider a 10×10 image shown in Figure 3.2 (a). The integer approximation of EDT values are shown in Figure 3.2 (b). Let d_{far} be two pixel units. The corresponding collision region whose pixels having integer distance values less than or equal to 2 is as shown in Figure 3.2 (c) and the collision-free graph is displayed in Figure 3.2 (d). Consider the pixel at (2,6) as start pixel and the pixel at (9,1) as goal pixel. The *bfs* tree constructed with goal pixel's node as root is shown in Figure 3.2 (e). The collision-free shortest path on the image is shown in Figure 3.2 (f) by arrows. Figure 3.3 illustrates the computation of $(\Delta x, \Delta y)$ and δ while performing EDT.

As described earlier, each pixel is connected to eight of its neighboring pixels. The *bfs* tree is built step-by-step and in each step, the next level pixels (nodes) are assigned the parent node. Depending upon the location of object pixels, at most 8 pixels will be available to be assigned as the parent pixel. Most of the times, more than one pixel can be assigned as the parent to the pixel in question. Therefore, a priority scheme has to be devised to assign the parent pixel. Normally the pixels will be prioritized from the right

Figure 3.2: Illustration of the proposed method. (a) Example image. (b) Integer EDT values. (c) Collision region for $d_{far} = 2$ pixel units. (d) Collision-free graph. (e) Constructed *bfs* tree. (f) Collision-free path on the image.

[illegible]

Iteration 1

(a)

(0,0) 6	(0,0) 6	(0,1) 5	(0,2) 2		(0,2) 2	(0,1) 5	(0,0) 6	(0,0) 6
(0,0) 6	(0,0) 6	(0,1) 5	(0,2) 2		(0,2) 2	(0,1) 5	(0,0) 6	(0,0) 6
(1,0) 5	(1,0) 5	(1,1) 4	(1,2) 1		(1,2) 1	(1,1) 4	(1,0) 5	(1,0) 5
(2,0) 2	(2,0) 2	(2,1) 1				(2,1) 1	(2,0) 2	(2,0) 2
			(2,1) 1	(2,0) 2	(2,0) 2	(2,1) 1		
		(1,2) 1	(1,1) 4	(1,0) 5	(1,0) 4	(1,1) 4	(1,2) 1	
		(0,2) 2	(0,1) 5	(0,0) 6	(0,0) 5	(0,1) 5	(0,2) 2	
		(0,2) 2	(0,1) 5	(0,0) 6	(0,0) 5	(0,1) 5	(0,2) 2	

Iteration 2

(b)

(0,0) 12	(0,0) 12	(0,1) 11	(0,2) 8	(0,3) 3	(0,3) 3	(0,2) 8	(0,1) 11	(0,0) 12	(0,0) 12
(0,0) 12	(0,0) 12	(0,1) 11	(0,2) 8	(0,3) 3	(0,3) 3	(0,2) 8	(0,1) 11	(0,0) 12	(0,0) 12
(1,0) 11	(1,0) 11	(1,1) 7	(1,2) 7	(1,3) 2	(1,3) 2	(1,2) 7	(1,1) 10	(1,0) 11	(1,0) 11
(2,0) 8	(2,0) 8	(2,1) 7	(2,2) 4			(2,2) 4	(2,1) 7	(2,0) 8	(2,0) 8
(3,0) 3	(3,0) 3	(3,1) 2					(3,1) 2	(3,0) 3	(3,0) 3
			(3,1) 2	(3,0) 3	(3,0) 3	(3,1) 2			
		(2,2) 4	(2,1) 7	(2,0) 8	(2,0) 8	(2,1) 7	(2,2) 4		
(1,3) 2	(1,2) 7	(1,1) 10	(1,0) 11	(1,0) 11	(1,1) 10	(1,2) 7	(1,3) 2		
(0,3) 3	(0,2) 8	(0,1) 11	(0,0) 12	(0,0) 12	(0,1) 11	(0,2) 8	(0,3) 3		
(0,3) 3	(0,2) 8	(0,1) 11	(0,0) 12	(0,0) 12	(0,1) 11	(0,2) 8	(0,3) 3		

Iteration 3

(c)

(0, 0)	(0, 20)	(0, 1)	(0, 2)	(0, 3)	(0, 3)	(0, 1)	(0, 0)	(0, 20)
	20	20	11	11	11	16	16	
(0, 0)	(0, 20)	(0, 1)	(0, 2)	(0, 3)	(0, 3)	(0, 1)	(0, 0)	(0, 20)
	20	20	11	11	11	16	16	
(1, 0)	(1, 19)	(1, 1)	(1, 2)	(1, 3)	(1, 3)	(1, 1)	(1, 0)	(1, 19)
	19	18	15	10	10	15	11	
(2, 0)	(2, 16)	(2, 1)	(2, 2)	(2, 3)	(2, 3)	(2, 1)	(2, 0)	(2, 16)
	16	15	12	7	7	12	15	
(3, 0)	(3, 11)	(3, 1)	(3, 2)	(3, 4)	(3, 4)	(3, 1)	(3, 0)	(3, 11)
	11	10	7	4	7	10	11	
(4, 0)	(4, 7)	(4, 2)	(4, 1)	(4, 3)	(4, 3)	(4, 2)	(4, 0)	(4, 7)
	7	7	11	11	11	7	4	
(2, 4)	(2, 3)	(2, 2)	(2, 1)	(2, 0)	(2, 0)	(2, 2)	(2, 3)	(2, 4)
	7	12	15	16	16	15	12	
(1, 4)	(1, 3)	(1, 2)	(1, 1)	(1, 0)	(1, 0)	(1, 2)	(1, 3)	(1, 4)
	10	15	18	19	19	18	15	
(0, 4)	(0, 3)	(0, 2)	(0, 1)	(0, 0)	(0, 0)	(0, 2)	(0, 3)	(0, 4)
	11	16	19	20	20	16	11	
(0, 4)	(0, 3)	(0, 2)	(0, 1)	(0, 0)	(0, 0)	(0, 2)	(0, 3)	(0, 4)
	11	16	19	20	20	16	11	

Iteration 4

(d)

Figure 3.3: Computation of $(\Delta x, \Delta y)$ and δ for EDT

Figure 3.4: Path traced using two types of neighborhood

side in a clockwise fashion. Since the neighboring pixels are not equidistant from the middle pixel, this ordering will not provide an efficient path. Diagonal pixels are slightly farther than the pixels at right angles. Therefore a priority scheme is designed such that the horizontal and vertical neighbors of the given pixel will be given first priority followed by the diagonal pixels in assigning as the parent pixel. Figure 3.4 shows the path traced by both neighborhoods. It is clear from the paths traced that the simple neighborhood results in a path with longer absolute distance even though the number of steps will be the same.

The path planning can be extended to a scenario with multiple goals. In this case, *bfs* trees are constructed simultaneously with each goal pixel as a root. Each tree consists of those pixels that are closer to the goal as its root than other goal pixels. The path from a start pixel is traced to the nearest goal.

3.3 Proposed Algorithm

A new parallel algorithm is designed for the proposed method. Given the binary image of the environment with obstacles and free space pixels, and start pixel p_s , goal pixel p_g and d_{far} of an autonomous navigator, the algorithm iteratively constructs the distance map and then the *bfs* tree. A path for the autonomous navigator from the start pixel to the goal pixel is derived from the *bfs* tree. The parent of a pixel in the *bfs* tree is one of the eight neighbors surrounding the pixel in the 3×3 neighborhood N . The pointer value corresponding to each neighbor p_i is assigned as i . The pseudocode of the proposed algorithm *Parallel_Path_Plan* is as follows.

ALGORITHM: *Parallel_Path_Plan*

Inputs: An $n \times n$ binary image, d_{far} , p_s and p_g

Outputs: Sequence of moves for the autonomous navigator from p_s to p_g

Step 1: Initialization

$\Delta x = \Delta y = \delta = 0$ & $done = 1$ for object pixels

$$visited(p) = \begin{cases} 1 & p = p_g \\ 0 & otherwise \end{cases}$$

$parent(p) = 0$ for all p

Step 2: Find EDT

repeat for $k=1,2,\dots$

for all pixels p **do in parallel**

 Compute $\Delta x_i, \Delta y_i$ and $\delta_i, i = 1$ to 8

$$\delta_m = \max \{ \delta_i \text{ with } done(p_i) = 1 \}.$$

if $(done(p) = 0 \wedge \delta_m \geq 0)$

$$\Delta x(p) = \Delta x_m$$

$$\Delta y(p) = \Delta y_m$$

$$\delta(p) = \delta_m$$

$$done(p) = 1$$

else if $done(p) = 1$

$$\delta(p) = \delta(p) + 2k$$

end if

end for

until $done = 1$ for all pixels

Step 3: Find collision-free region

for every pixel p **do in parallel**

if $(d_I(p) > d_{far})$ $cfr(p) = 1$

else $cfr(p) = 0$

end for

Step 4: Construction of *bfs* tree

repeat

for every pixel p **do in parallel**

if $(visited(p)=0) \wedge (\exists p_i \in N[visited(p) = 1])$

$visited(p) = 1$

$parent(p)$ is pointed to p_i with minimum i

end if

end for

until ($visited = 1$ for all p where $cfr = 1$)

Step 5: Find sequence of moves

$p = p_s$

while ($parent(p) \neq 0$)

print $parent(p)$

$p = \text{parent pixel of } p$

end while

The algorithm is executed in five different steps. In Step 1, Δx , Δy and δ are initialized to '0' and $done$ is initialized '1'. $visited$ register is set to '1' for the goal pixel and it is set to '0' for all other pixels. As the path is yet to be traced, $parent$ register for all the pixels is set to '0'. In Step 2, EDT is performed on the binary image. Δx_i , Δy_i and δ_i are calculated for all the pixels in parallel. A pixel's δ value is updated with the maximum δ value of its neighbors whose EDT value have already been computed. Δx and Δy values are also updated accordingly and the pixel's $done$ value is set to '1' to indicate that the EDT value for that pixel has been calculated already. This process is performed until the EDT values are calculated for all the pixels i.e., until $done = 1$ for all the pixels.

In Step 3, Collision free region is calculated. cfr register is set to '1' if the Euclidean distance calculated is greater than a given d_{far} . Construction of bfs tree is performed in

Step 4. The tree construction starts from the goal pixel whose *visited* register is set to ‘1’ during initialization. *visited* register of a pixel is set to ‘1’ if any one of its neighboring pixel is already visited. Its parent pixel is also pointed accordingly in this step.

In Step 5, the sequence of moves from source to goal is traced using the *parent* registers. The sequence of moves is given by the pointers. The autonomous navigator can take horizontal, vertical or diagonal step based on the pointer value. It should be noted that once the *bfs* tree is constructed in Step 4, the shortest path from any pixel to the given goal pixel can be found out.

3.3.1 Complexity Analysis

The time and space complexities are given via Propositions 1 and 2.

Proposition 1. *The time complexity of Algorithm Parallel_Path_Plan is $O(n^2)$.*

Proof. The time complexity of the proposed algorithm depends mainly on the **repeat-until** and **while** loops. The statements within **for-end for** are executed in parallel for all pixels. The **repeat-until** loop in Step 2 runs until *done* is set to 1 for all pixels. In the worst case, the loop is executed for $\lceil d_{max} \rceil$ times where d_{max} is the maximum possible distance value which is $\sqrt{2}n$ in the case of an $n \times n$ image and $\sqrt{n^2 + m^2}$ in the case of an $m \times n$ image. *cfr* computation in Step 3 takes only constant time. The **repeat-until** loop in Step 4 is executed close to n^2 (mn for $m \times n$ image) times in the worst case when the path obtained is a zig-zag one. The **while** loop in Step 5 also runs close to n^2 (mn for $m \times n$ image) times in the worst case. Hence the overall time complexity of the algorithm is $O(n^2)$ ($O(mn)$ for $m \times n$ image). Q.E.D.

Proposition 2. *The space complexity of Algorithm Parallel_Path_Plan is also $O(n^2)$.*

Proof. In the algorithm, the values of Δx , Δy , δ , cfr , *visited* and *parent* of each pixel are stored. Therefore the space complexity of the algorithm is equal to the number of pixels. Hence the space complexity is $O(n^2)$ for $n \times n$ image and $O(mn)$ for $m \times n$ image. Q.E.D.

3.3.2 Simulation Studies

The algorithm was coded in C and simulated on a PC with Core 2 Duo E8500 dual processors and 8 GB RAM running Ubuntu Linux 9.10 operating system.

Static Environment

The first experiment considers the image of a static environment as shown in Figure 3.5 (a). The size of this image is 100×130 . Its binary version in Figure 3.5 (b) segmenting the objects from free-space was obtained by thresholding. The gray value of the threshold was set to 158. The collision region is shaded in yellow. For $d_{far} = 4$ and two goals A and B at (50,110) and (85,85) respectively, the paths traced to the nearest goal from three different starting locations ((10,80), (40,40) and (80,30)) are shown in Figure 3.5 (b). The paths are shown in three different colors and the overlapping part of two paths is shown in blue. The number of moves for the autonomous navigator on these paths are 68, 78 and 71 respectively. The time taken for computing the paths is 3 seconds. Instead of thresholding, edge detection was also applied. Both thresholding and edge detection are simple operations and they take a single scan over the image. The binary edge map was obtained by Sobel operator [105]. Similar results for path planning were obtained on

(a) Test Image

(b) Paths on thresholded binary image

(c) Paths on edgemap

Figure 3.5: Results of path planning on a binary image

Figure 3.6: Dynamic environment with a ball moving as shown. (a) Initial Path. (b)-(d) Paths on successive frames. (e) Final path.

the edge map. The collision region and the paths traced are shown in Figure 3.5 (c) for this case.

Dynamic Environment

The second experiment considers the dynamic environment as shown in Figure 3.6. The motion of the ball was simulated by sliding a (16,16) image containing the ball and its trajectory is shown in the figure. Initially, the bottom left end of this ‘ball image’ was at (99,42). The start and goal pixels of the autonomous navigator were (95,75) and (35,30) respectively. The path traced on the thresholded binary image for this initial setup is shown in Figure 3.6 (a). It was assumed that the new image frame was considered for updating the path after the autonomous navigator had moved five steps in its current path. Successive image frames were generated by sliding the ‘ball image’ at an angle of about 33 degrees upwards. That is, the image was moved three pixels up and two pixels to the right for every frame. A new *bfs* tree was constructed for each frame and the autonomous navigator was moved five steps along the new path in each frame. The time taken for computing the paths is 12.3 seconds. Figures 3.6 (b)-(d) show the updated paths from new start pixels to the goal in three successive image frames. The final path followed by the autonomous navigator is shown in Figure 3.6 (e).

3.4 Conclusions

An algorithm for tracing the shortest path of a translating and rotating autonomous navigator on the binary image of an environment is given in this chapter. The algorithm first constructs the distance map of the image to obtain a collision-free region and then constructs the *bfs* tree of pixels in the collision-free region which defines the shortest path

from any start pixel to a specific goal pixel. If multiple goals are provided, the path is traced to the nearest goal. The algorithm is also extended to be applicable in a dynamic environment where objects are moving. The proposed algorithm is based on local neighborhood operations and is therefore appropriate for mapping onto a cellular architecture. Such a cellular architecture is described in next chapter.

CHAPTER 4

Cellular Architecture for Path Planning

Previous chapter presented a parallel algorithm for path planning on the image of an environment. As the algorithm relies mainly on neighborhood operations, it is suitable for mapping onto a 2D cellular architecture. A cellular architecture consists of locally connected identical processing elements. This chapter presents the architecture, FPGA implementation and comparison with other related architectures.

4.1 Proposed Cellular Architecture

In Algorithm *Parallel_Path_Plan*, the construction of EDT (Step 2) and *bfs* tree (Step 4) are compute-intensive since they are iterative and at each iteration, every pixel has to be processed. In this section, a cellular architecture is designed for the operations from Step 1 to Step 4 of the algorithm. The resulting *bfs* tree with ‘goal’ pixel as the root facilitates the tracing of path from any ‘start’ pixel to a given ‘goal’ pixel using the computed *parent* values. For an $n \times n$ image, the architecture consists of an $n \times n$ array of cells in which each cell is connected to the eight neighboring cells surrounding it. A 4×4 cellular array is shown in Figure 4.1.

A cell is a sequential logic consisting of storage elements Δx , Δy , δ , d_I and *done* for the computation of EDT and *cfr*, *visited* and *parent* for constructing a collision-free

 Figure 4.1: A 4×4 cellular array.

path using EDT. The computation of these values is implemented by using adders, subtractors, comparators and a few logic gates. The *init* and *out* signals for initialization and output delivery, and the iteration number k are generated by an external counter and they are given to all the cells. The d_{far} value is also fed to all the cells. The interconnections between cells carry the values Δx , Δy , δ , *done* and *visited*. The cells are updated synchronously with an external clock clk . *done* and *visited* signals are fed to the leftmost cells for initialization. The *parent* values are the outputs of the architecture which are

also received from the leftmost cells. The following proposition gives the number of I/O lines of the architecture.

Proposition 3. *The architecture has $5n + \lceil \log_2 \frac{n}{2} \rceil + 2$ I/O lines.*

Proof. The *START*, *clk*, *done* and *visited* are one bit width lines. The maximum possible value of d_{far} is $\frac{n}{2}$ (half the number of rows/columns of image grid) and therefore its binary representation takes $\lceil \log_2 \frac{n}{2} \rceil$ bits. The pointer to neighbor p_i in Figure 3.1 is denoted by $i - 1$ in hardware. The *parent* is therefore represented in three bits since the path tracing is done in software for which the goal pixel (the root of the tree) is given. Since five I/O lines are connected to each of the n leftmost cells, the total number of I/O lines is $5n + \lceil \log_2 \frac{n}{2} \rceil + 2$. Q.E.D.

The speed of operation of the cellular architecture depends primarily on the delay due to a cell. The delay due to interconnection between cells is negligible because the cells are locally connected. The design of a cell facilitates quick estimation of the delay and therefore the maximum operating frequency of the architecture.

4.1.1 Design of a Cell

The different modules of a cell are shown in Figure 4.2. The values stored in the storage elements are integers and the sizes of the storage elements depend on the size of the image. *done*, *cfr* and *visited* are one bit size. For an image of size $n \times n$, the size of Δx and Δy is $\lceil \log_2 n \rceil$ bits. The size of δ can be derived as follows.

If a pixel p receives its EDT parameter values at some iteration k , then $\delta(p)$ is

Figure 4.2: Architecture of a cell (CL denotes the combinational logic).

bounded by $2k$ as per its definition. Also, once p 's transform values have been computed, $\delta(p)$ needs to be updated in the next two iterations, i.e., $k+1$ and $k+2$, for the computation of transform values of neighbors. This is done by adding $2(k+1)$ and $2(k+2)$ to $\delta(p)$. The value of $\delta(p)$ up to $2k + 2(k+1) + 2(k+2)$ is useful. Since the maximum number of iterations for an $n \times n$ image is d_{max} , where $d_{max} = \lceil \sqrt{2}n \rceil$, the size of storage element δ of any cell can be $\lceil \log_2 6d_{max} \rceil$ bits.

The ADD-SUB module computes δ_i , $i = 1$ to 8. The computation involves an addition and a subtraction for $i = 5, 6, 7$ & 8 and only a subtraction for $i = 1, 2, 3$ & 4. Therefore, eight subtractors and four adders are required to carry out this computation. The INC module computes Δx_i and Δy_i . The computation requires twelve incrementers, six for

implementing $\Delta x_i = |\Delta x(p_i)| + 1$, $i=2,4,5,6,7$ & 8 , and another six for implementing $\Delta y_i = |\Delta y(p_i)| + 1$, $i=1,3,5,6,7$ & 8 .

Once δ_i , Δx_i and Δy_i for $i=1$ to 8 have been computed, these values are given to a MAX module along with $done(p_i)$ and borrow bits b_i from the subtractors of the ADD-SUB module. The MAX module has seven CMP-MUX (comparator-multiplexer) modules arranged in three-level binary tree structure to compute $\max[\delta_i]$, $i = 1$ to 8 . In the Figure 4.2, $\max[\delta_i]$ is denoted by δ_m . Further, the MAX module allows the values of Δx_i and Δy_i , corresponding to the i that yields δ_m . These values are denoted by Δx_m and Δy_m .

The CMP-MUX module takes two sets of inputs, $set_j = \{\delta_j, \Delta x_j, \Delta y_j, done(p_j), b_j\}$ and $set_l = \{\delta_l, \Delta x_l, \Delta y_l, done(p_l), b_l\}$. It has a comparator to compare $\delta_j > \delta_l$ and a multiplexer that outputs either set_j or set_l depending on the value of sel input of multiplexer. set_j is output when $sel = 0$ while set_l is output when $sel = 1$. The sel is generated using the output cmp of comparator and the values of $done$ and borrow bits. The comparator is designed for the simple case of comparing two unsigned binary numbers. $done(p_j) = 1$ means the value of δ_j is valid. $b_j = 1$ means δ_j is negative and $cmp = 1$ means $\delta_j > \delta_l$. The value of sel for different cases is tabulated in Table 4.1.

The output δ_m of the MAX module is added to $2k$ where k is the iteration number generated by an external counter. The output $\delta(p)$ of the adder is given as input to the register δ . The outputs Δx_m and Δy_m of the MAX module are given as inputs to the registers Δx and Δy . The $done$ flip-flop is input with logic 1. In the design, these storage elements are loaded with the available inputs during the rising edge of the clock. The clock is activated only when the following conditions are satisfied.

1. $done$ of cell is not set.

Table 4.1: Value of sel for inputs to CMP-MUX module.

$done(p_j)$	$done(p_l)$	b_j	b_l	cmp	sel
0	0	-	-	-	-
1	0	-	-	-	0
0	1	-	-	-	1
1	1	0	0	1	0
1	1	0	0	0	1
1	1	1	1	1	0
1	1	1	1	0	1
1	1	0	1	-	0
1	1	1	0	-	1

2. At least a value of $done$ of neighbors is logic ‘1’.
3. $\delta(p)$ is positive. In two’s complement representation of $\delta(p)$, this means that the MSB is 0.

In the case of register δ , the clock is activated when the conditions 2 and 3 are satisfied. Let cdn be the signal whose value is ‘1’ if the above three conditions are satisfied. cdn can be generated using a simple AND-OR logic circuit.

During the EDT computation, the storage element d_I of each cell receives the integer Euclidean distance value. A comparator compares d_I with the input d_{far} and sets cfr accordingly to determine the collision-free region. The $visited$ and $parent$ values are then computed by the cells in different clock cycles using the computed cfr and the $visited$ values of the neighbors. The 3-bit $parent$ values pointing to the neighbors are generated inside the cell. A simple combinational logic sets the correct $parent$ value.

4.1.2 Operation of Hardware

The architecture is operated in three modes. The modes are determined by *init* and *out* signals.

Mode 1: Initialization

The signal *init* is set and the storage elements are initialized. The *done* of cells corresponding to obstacle pixels are initialized to '1' and the *visited* of the goal pixels are set to '1'. All the remaining storage elements are initialized to 0. The initialization is done by feeding the *done* and *visited* of each row to the corresponding leftmost cell in pipeline. The inputs to all the leftmost cells are given in parallel. When clocked, the inputs are passed through the array from left to right in each row. The INIT logic in the cell performs initialization when *init* is set high.

Mode 2: 'parent' computation

This mode has three phases of operation. EDT computation is done first. Using the initial *done* values, distance values d_I are computed first. The input k to all the cells is incremented in each cycle. Once EDT is computed, *cfr* is computed in the next phase using the input d_{far} and the computed d_I in each cell. In the third phase, the architecture is allowed to run to construct the *bfs* tree using *cfr* and *visited* values. The tree is represented by the *parent* values computed in the cells.

Mode 3: Output delivery

The signal *out* is set. The array outputs the computed *parent* values from the leftmost cells by passing them from right to left through the cells in each row. The OUT logic in the cell performs this operation when *out* is set high. The total number of clock cycles taken by the hardware is given by the following proposition.

Proposition 4. *The total number of clock cycles taken by the hardware is given by $N_{pp} = N_{init} + N_{EDT} + N_{bfs} + N_{cfr} + N_{out} = n^2 + 2n + \sqrt{2}n + 1$.*

Proof. The initialization takes n clock cycles since there are n cells in a row. The computation of EDT is carried out during the initial few clock cycles. The number of clock cycles (N_{EDT}) needed to compute EDT is set to d_{max} , the maximum possible distance value, which is $\sqrt{2}n$ in the case of an $n \times n$ image (as indicated in Proposition 1). The *cfr* computation takes a single clock cycle. The number of clock cycles required for the construction of *bfs* tree (N_{bfs}) for finding a path on it is n^2 . This is because in the worst case, the path obtained is in zig-zag form. The delivery of outputs takes n clock cycles. Hence the total number of clock cycles is given by $N_{pp} = N_{init} + N_{EDT} + N_{bfs} + N_{cfr} + N_{out} = n^2 + 2n + \sqrt{2}n + 1$. Q.E.D.

4.2 Implementation and Testing

The proposed cellular architecture was implemented on a commercially available Xilinx ML403 evaluation platform for embedded systems development. The platform is equipped with a Xilinx Virtex-4 XC4VFX12 FPGA, and industry-standard peripheral connectors and interfaces. The details of the development platform are given in Appendix A. The architecture design was first implemented on the FPGA device and then tested on the evaluation platform.

Figure 4.3: Outputs of simulator of Xilinx ISE 10.1 displaying the waveforms of different signal variables of selected cells in the design for the input image shown in Figure 3.2(a) during EDT computation. Arrows show the cause and effect relationships in the waveforms.

4.2.1 FPGA Implementation

The design of an $n \times n$ cellular architecture was coded in Verilog HDL [106] and its functional behavior was tested in the simulator of Xilinx ISE 10.1 with different images. The values stored in the storage elements after running the design were checked for correctness with the ones obtained in C implementation. The snapshots of the simulation waveforms generated by Xilinx simulator during the computation of EDT and the construction of *bfs* tree are shown in Figure 4.3 and Figure 4.4 respectively for the input image as in Figure 3.2(a). The waveforms in Figure 4.3 show the Δx , Δy and δ computed by a few cells at different iterations. $\Delta x(a, b)$, $\Delta y(a, b)$ and $\delta(a, b)$ represent the outputs of the storage elements Δx , Δy and δ of the cell at row a and column b of the cellular array. The waveforms in Figure 4.4 show the *visited* and *parent* values of the cells along the

Figure 4.4: Outputs of simulator of Xilinx ISE 10.1 displaying the waveforms of different signal variables of selected cells in the design for the input image shown in Figure 3.2(a) during *bfs* tree construction. Arrows show the cause and effect relationships in the waveforms.

path in Figure 3.2(f). These cells receive the values at successive clock cycles.

After the functional testing, implementation of the design was carried out in Xilinx ISE 10.1. As a single cell decides the frequency of operation of the entire architecture, the design for a single cell was implemented first. The design was mapped onto the target FPGA device. The specifications of the target device are as follows - Family: Virtex 4; Device: XC4VFX12; Package: SF363; Speed: -12. The maximum operating frequency obtained after placement and routing is 375 MHz. Once the clock frequency was obtained,

Table 4.2: FPGA implementation results for image size of 40×40 .

Resource	Available	Used	Usage
Slice	5472	4815	88%
Slice Flip-flop	10944	2407	22%
4 input LUT	10944	9630	88%
Bonded IOB	240	206	86%
GCLK	32	1	3%

the entire $n \times n$ cellular array was then implemented on the device. The maximum size of

the array that was accommodated on the device is 40×40 . The implementation results are given in Table 4.2. It shows the different hardware resources in the device and their usage. Slice flip-flops implement the storage elements and LUTs implement the combinational logic in the cells. Bonded IOBs are assigned to external inputs and outputs of the design. The *clk* signal is generated inside the device by a global clock. A larger FPGA with more IOBs in the same family (XC4VFX140 with 768 bonded IOBs for example) can accommodate a cellular array of size up to 100×100 and can be operated at nearly the same clock frequency.

Figure 4.5: Time taken for the construction of *bfs* tree in PC and FPGA for different image sizes.

Since a cellular architecture consists of locally connected identical cells, the delay due to interconnections is negligible. Hence, the time (T_{pp}) taken to compute the path on the image of size $n \times n$ can be estimated from the clock rate (f_c) obtained from the implementation of a cell. The period (T_c) of a clock pulse is $1/f_c$ seconds. Hence, $T_{pp} = N_{pp} * T_c$. N_{pp} is given by Proposition 4. The number of images (N_I) processed per second

on the cellular architecture is greater than or equal to $\lceil 1/T_{pp} \rceil$. Figure 4.5 shows a plot of time taken by software and hardware implementations for the construction of *bfs* tree for path planning on different image sizes. FPGA-based processing is around five orders of magnitude faster than PC-based processing.

Some results and comments for an image size 100×100 are as follows. d_{max} for this case is 141.4. The frequency of operation of cell is 375 MHz. T_{pp} is around $27.6 \mu s$ and hence N_I is approximately 36258. N_I is much greater than the video rate, which is about 30 frames per second and hence the path planning on a cellular architecture is quite suitable for dynamic environment.

4.2.2 Testing of FPGA Design

Figure 4.6: ML403 board components and connections relevant to our experiment.

For the purpose of testing the FPGA design, the netlist for a 10×10 cellular ar-

ray was downloaded to FPGA on the evaluation platform and tested on the evaluation platform with actual binary image data inputs from PC. The experimental setup is shown in Figure 4.6. Data was sent from PC to ML403 board using HyperTerminal running on Windows XP. Xilinx's IP core UARTLite was instantiated to receive data through the RS232 port. Both the HyperTerminal and IP core were configured to operate at a baud rate of 115200 with no parity bits. The data received at Universal Asynchronous Receiver/Transmitter (UART) buffer was then transferred to the custom user design in FPGA by an interfacing program running on PowerPC (hardcore processor in the FPGA device). The PowerPC-based embedded system was developed using Xilinx Platform Studio to interface the FPGA design with RS232 and LCD. Processor Local Buses (PLB) were used for interfacing and all the interfacing programs were implemented in C and run on PowerPC.

The *done* and *visited* values corresponding to the binary image in Figure 3.2(a) were sent from the HyperTerminal running on PC. The FPGA received the values through RS232 interface during initialization mode. The data inputs to the custom user design in a clock cycle were formatted into a data word by the interfacing program running on PowerPC. After initialization, the FPGA performed the Euclidean distance transformation and the construction of *bfs* tree. When the *out* signal changes to 1, a message 'bfs over' was displayed on LCD as shown in Figure 6.13. The output *parent* values were then sent to PC in data words. The data words displayed on the HyperTerminal were checked for correctness.

4.2.3 Scalability of Design

The regular structure of the proposed cellular architecture facilitates scalability of design for larger image sizes. An image larger than 40×40 was processed on the same XC4VFX12 device using architectural enhancements described below. The proposed approach is based on a new idea of processing of more pixels in a single cell in pipeline.

(a)

pixel 1	pixel 2
pixel 4	pixel 3

Clock	1	2	3	4
Pixel processed	pixel 1	pixel 2	pixel 3	pixel 4

(b)

Figure 4.7: Scalability via processing of multiple pixels in a cell in pipeline. (a) Architecture of a cell processing four pixels. (b) Block of pixels and their pipelined processing by a cell in an iteration during EDT computation and *bfs* tree construction.

An $N \times M$ block of pixels is assigned to a cell. An $n \times n$ cellular array can therefore process $nN \times nM$ image. A typical cell that processes four pixels is shown in Figure 4.7. The cell stores the various values ($\Delta x, \Delta y, \delta, done, d_I, cfr, visited$ and *parent*) of each pixel in the block. However, the combinational logic for the computation of these values

is the same and it processes each pixel of the block in a cycle. The pixels are therefore processed in pipeline. Multiplexers select appropriate inputs to the combinational logic. Temporary storage elements for Δx , Δy , δ and *visited* of $NM - 1$ pixels are necessary to store their updated values in an iteration so that the original values are available to neighbors till the end of each iteration. These values are moved to their original storage in the last cycle of each iteration. Hence each iteration of EDT computation and *bfs* tree construction comprises of NM clock cycles since all the pixels of the image are not processed in parallel at a time. The storage elements of pixels in a cell are linearly connected. Hence the initialization and output delivery modes of operation take nNM clock cycles each. The interconnections between cells pass Δx , Δy , δ and *visited* values of NM pixels of each cell to the neighboring cells. If $P = nN$ and $Q = nM$, then processing a $P \times Q$ image on an $n \times n$ architecture takes $(PQ + \sqrt{P^2 + Q^2} + 2n + 1)NM$ clock cycles (refer Proposition 4). The number of iterations for EDT computation and *bfs* tree construction depends on the size of image. Therefore, they are PQ and $\sqrt{P^2 + Q^2}$ respectively.

The design of an $n \times n$ cellular architecture with each cell processing four pixels as shown in Figure 4.7 was coded in Verilog HDL and its functional behavior was tested in the simulator of Xilinx ISE 10.1. The snapshots of the simulation waveforms are shown in Figure 4.8 and Figure 4.9 for the input image in Figure 3.2(a). It can be observed that the iteration value k changes after every four clock cycles. The waveforms of the signal variables of the cell at (2,2) that processes the pixels at (3,3), (3,4), (4,4) and (4,3) are shown in Figure 4.8. The EDT values are initially stored in temporary locations. In the fourth cycle of the iteration in which they are computed, these values are transferred to their final storage locations. In Figure 4.9, the waveforms of *visited* and *parent* of some pixels (at (5,5), (4,6), (3,6) and (2,6)) along the path in Figure 3.2(f) are shown.

Figure 4.8: Outputs of simulator of Xilinx ISE 10.1 displaying the waveforms of different signal variables of selected cells in the scalable design with each cell processing four pixels as in Figure 4.7 during EDT computation. Arrows show the cause and effect relationships in the waveforms. $T\Delta x(a, b)$, $T\Delta y(a, b)$ and $T\delta(a, b)$ denote the temporary storage and $\Delta x(a, b)$, $\Delta y(a, b)$ and $\delta(a, b)$ denote the actual storage for Δx , Δy and δ of a pixel at row a and column b .

The pixels receive their values in successive iterations starting from the fourth iteration. Each iteration consists of four clock cycles to process the four pixels of the cell. It can be seen that the *visited* flag is set first in its temporary storage in a certain clock cycle of an iteration. The exact cycle in which the flag is set depends on the order in which the specific pixel is processed in a cell (Figure 4.7(b)). The value is transferred to its final storage in the last cycle of the iteration.

In the case of one pixel per cell, Table 4.2 shows that the utilization of slice flip-flops is much less when compared to LUTs in the device. A two pixels per cell implementation leads to a three-fold increase in slice flip-flops for storage elements but only a slight increase in LUTs. A cell has two sets of storage elements for storing different values of both pixels. One set of temporary storage is also required for processing the pixels in

Figure 4.9: Outputs of simulator of Xilinx ISE 10.1 displaying the waveforms of different signal variables of selected cells in the scalable design with each cell processing four pixels as in Figure 4.7 during *bfs* tree construction. Arrows show the cause and effect relationships in the waveforms.

a pipelined fashion. The slight increase in LUTs is due to extra logic elements such as MUX for selecting the pixel to be processed. A 40×80 image could be processed on a XC4VFX12 device by utilizing the unused slice flip-flops for implementing two pixels in a cell of a 40×40 cellular array (Further increase in the number of pixels per cell resulted in insufficient LUTs in this device). The time for processing a 40×80 image was computed to be $18 \mu s$ and N_I is approximately 55630. While there is a slight degradation in the processing time, the performance is well within the real-time requirements. The device utilization is hence maximized without losing the required performance by assigning multiple pixels to a cell for processing higher resolution image.

On the other hand, a given image size can be processed on a smaller array with a slight decrease in speed (but still maintaining video rate) to obtain an area-efficient solution. A 40×40 image was processed on a 20×20 cellular array by assigning four pixels per cell. Although the slice flip-flops used were almost doubled due to temporary storage, the usage of LUTs came down to nearly one-fourth and IOBs to about half of the usage

for implementing 40×40 array. Considerable reduction in LUTs enables implementing additional logic in the design.

4.3 Comparison Studies

A comparison of the proposed path planning approach with prior hardware-based path planning schemes [107, 108, 34, 109, 110, 111, 112] is given in Table 4.3. Most of the previous works aim at the construction of structures such as visibility graph [107, 108] and Voronoi diagram [34, 109, 110] for path planning. A complete path planner has not been attempted using these structures. Further, the visibility graph-based approaches assume an environment model with obstacles approximated by polygonal shapes.

Table 4.3: Comparison with existing hardware solutions for path planning

Hardware solution	Algorithm		Architecture Type	Path computation time on FPGA
	Input	Output		
[34, 109]	Binary environment image	Voronoi diagram	Cellular architecture	Path not computed
[36]	Ultrasonic sensor inputs	Voronoi diagram	Direct implementation	Path not computed
[107, 108]	Vertices of polygonal objects	Visibility graph	Direct implementation	≈ 0.2 ms for 70 vertices
[111]	Vertices and weighted edges of the graph representing environment	Shortest path on the graph	Direct implementation	
[112]	EDT of environment image	Sequence of moves from start pixel to goal	Direct implementation	1.86 ms on an average for a 128×128 image
Proposed	Binary environment image	Sequence of moves from start pixel to goal	Cellular architecture	0.12 ms for a 128×128 image

The hardware solutions in [111, 112] provide a complete path from a start point to goal point for an autonomous navigator. However, the algorithm in [111] requires the environment to be represented as a weighted directed graph. The work in [112] finds the path on the EDT of the binary image of an environment. The hardware architecture is a straightforward realization of the algorithm, and therefore it is not time-efficient and also not easily scalable.

This chapter has given a complete path planning solution directly for the binary image of an environment (not for the EDT of the image) via an array architecture. Given the binary image, the algorithm finds the shortest path in terms of number of moves from any start pixel to the nearest goal. The architecture is a two-dimensional cellular array with locally interconnected simple processing elements operating at a very high frequency. It is regular and easily scalable. The image-based approach proposed in this chapter allows for capture of images periodically and can therefore be used as such to recompute the path even in a dynamic environment containing obstacles with arbitrary motions.

In order to compare the performance of the proposed architecture with the image-based path planning architecture in [112], we synthesized the design of cellular architecture (with cells without EDT computation) for the target device chosen in [112]. The specifications of the device are as follows - Family: Spartan-3; Device: XC3S1500L; Package: FG320; Speed grade: -4. The maximum frequency of operation of FPGA design is 181.29 MHz. The implementation in [104] was considered for the computation of EDT as in [112] for the purpose of comparison. While the design in [112] takes 1.86 ms on an average for the computation of path on a 128×128 image, the proposed cellular architecture design takes only 0.12 ms. The graph-based hardware solution (visibility graph construction and path finding on the graph) takes approximately 0.2 ms on FPGA for computing the path in an environment with a total of 70 vertices. This assumes that polygonal approximations of obstacles and the navigator are available. The time taken includes expanding the obstacles while shrinking the navigator to a point and construction of visibility graph of the environment on an XCV3200E-FG1156 FPGA [108], and finding the path using the graph on an XC2V6000-BF957 FPGA [111]. It is worth noting that considerable additional time is required to obtain a circumscribing polygonal description of vertices of obstacles as well as the navigator. Further, there are issues with regard

to the number of vertices that would constitute a good polygonal approximation of the obstacles.

4.4 Conclusions

The cellular architecture described in this chapter consists of an $n \times n$ array of identical cells. The local interconnection contributes to excellent performance (in terms of speed). The architecture is highly scalable and handles images larger than $n \times n$ to maximize utilization of hardware space. Further, an architecture with fewer processing elements is adequate for an image of size $n \times n$ to obtain video rate. This chapter demonstrates how scalability is achieved via assignment of multiple pixels to a cell. Details of pipelined processing of the pixels assigned to a cell are also presented.

The architecture has been implemented on a Xilinx Virtex-4 FPGA and it handles the processing of an image of size 100×100 at a rate of more than 30,000 frames per second. FPGA-based processing is around five orders of magnitude faster than PC-based processing. Such a high processing speed guarantees that the architecture would deliver the required performance while acting as a co-processor in a bigger system. The speed of processing on FPGA is adequate for path planning in a dynamic environment containing obstacles of arbitrary shape. Further, the dynamic obstacles in the environment can have arbitrary motions.

CHAPTER 5

Neural Network-Based Face Recognition Algorithm

Previous two chapters described our work in navigation while this chapter and the next focus on face recognition. Neural network is one among several tools that has been applied to face recognition. The neural network-based face recognition approaches include the use of convolutional neural networks [113], radial basis neural networks [114] and other types of neural networks [115, 116, 117]. All of these focus on recognition performance leading to complex learning algorithms and non-linear neurons. In several of these works, the neural networks act as classifiers. Separate feature extraction algorithms extract relevant features that are fed to the neural network classifiers. The complexity of the learning algorithms and the feature extraction algorithms make the existing neural network-based face recognition methods inefficient for hardware mapping.

Eigenface based methods (Section 2.3) on the other hand provide robust face recognition and can be mapped onto hardware efficiently as detailed in this chapter. The performance problems arising due to local variations and expression changes are overcome by using a modular PCA method. PCA (Details in Section 2.3.5) is used to extract Eigenfaces from a large set of face database. PCA can be performed by a neural network. The neural network-based technique is advantageous over the classical techniques for the following reasons:

- It is not necessary to compute the data covariance matrix thereby minimizing the

memory requirements compared to other statistical methods.

- The learning rule has only simple arithmetic operations and possesses inherent parallelism that facilitate hardware implementation.

Principal Component Neural Network (PCNN) [118] is a single-layer feed-forward network of linear neurons. It can extract the eigenface features iteratively with a simple Hebbian learning algorithm. A PCNN-based face recognition system is described in this chapter.

Before describing the system, in Table 5.1, we put forth the notations which we have used widely in the remainder of this chapter.

5.1 System Description

The proposed face recognition system (Figure 5.1) assumes the inputs to be normalized face images from a face detection system which segments and normalizes the facial regions from the images captured by a camera. A single-layer linear feed-forward neural network with generalized Hebbian learning [119, 120] as shown in the figure can extract the eigenfaces of a set S of training faces. In real-world applications, the faces stored in the enrolled face database constitute the training set S . S consists of normalized face images. The input $X = [x_1, x_2, \dots, x_N]$ to the network is a face image in the vectorized form. The number of neurons (or outputs) M determines the number of eigenfaces. The weight vector $W_j = [w_{j1}, w_{j2}, \dots, w_{jN}]$ associated with neuron j converges to the j th eigenface when trained with faces in S . w_{ji} is the weight associated with the connection from i^{th} input to j^{th} neuron. μ is the learning rate in the range $[0,1]$. The network is operated in two phases, i.e., (1) Training phase and (2) Recognition phase.

S	Set of face images
X_p	Face image in vector form
x_{pi}	i^{th} element of X_p
P	Number of face images
I	Number of iterations
μ	Learning rate
W_j	Weight vector corresponding to j^{th} neuron
w_{ji}	Weight of the connection between j^{th} neuron and i th input
N	Number of network inputs
M	Number of network outputs
y_{pj}	Output of j^{th} neuron for p^{th} face image input
$\widehat{W}_j$	j^{th} eigenface in vector form
$\widehat{Y}_p$	Projections (in vector form) of X_p onto M eigenfaces
$\hat{y}_{pj}$	j^{th} element of $\widehat{Y}_p$
X^*	New face in vector form
Y^*	Outputs of trained network in vector form when X^* is fed
d_p	l_1 norm of difference between Y^* and $\widehat{Y}_p$
d	minimum of d_p values
p^*	Identity of new face
PS_{ji}, PD_j	Partial sums
S_k	Select signals of MUXs and DMUXs
M_k	Mode-control signals
n	Number of sub-images
C_T	Number of clock cycles taken for training phase
C_R	Number of clock cycles taken for recognition phase

Table 5.1: Notations

Figure 5.1: PCNN-based face recognition system

5.1.1 Training Phase

The inputs are face images in the training set S . This phase involves the computation of eigenfaces of S and the projection of each face in S on to the space spanned by the eigenfaces. Let S has P face images. That is, $S = [X_1, X_2, \dots, X_P]$. The network is fed with these face images X_p , for $p = 1$ to P , one after another, while the weights are updated for each input according to the learning rule shown in Figure 5.1. The training set of faces is provided as input to the network for several iterations until the convergence of

weights, such that after convergence, the maximum change in the weight values remains within a predetermined tolerable limit. The number of iterations can normally be fixed to a larger number I within which the weights converge to eigenfaces $\widehat{W}_j$. Once the network is trained, the network output vector $Y = [y_1, y_2, \dots, y_M]$ for each face in S is computed and stored. Let $\widehat{Y}_p = [\hat{y}_{p1}, \hat{y}_{p2}, \dots, \hat{y}_{pM}]$ be the output of the trained network for face $X_p = [x_{p1}, x_{p2}, \dots, x_{pN}]$. The elements of vector $\widehat{Y}_p$ are the projections of X_p on to eigenfaces $\widehat{W}_j$. The recognition of a new face (a normalized face from the detection system) is done using the trained network and the projection vectors $\widehat{Y}_p$, for $p = 1$ to P .

5.1.2 Recognition Phase

The given new face X^* is fed as input to the trained network and the vector difference between the network output Y^* and each of $\widehat{Y}_p$, for $p = 1$ to P , is computed. The minimum of the l_1 norm values of these difference vectors is given by $d = \min_p(d_p)$ where $d_p = \|Y^* - \widehat{Y}_p\|_1$. If $d < \tau$, where τ being a threshold value, then the corresponding p which gives d is considered as the identity p^* of the new face. Otherwise, the face is classified as “unknown”.

For smaller τ , the rejection rate increases and for larger τ , the recognition accuracy decreases. The trade off between rejection rate and recognition accuracy will be different for each of the various face recognition applications. In the case of surveillance applications such as security checks at border crossing, the training set consists of mug shots of criminals. τ should be high enough to have the rejection rate close to zero. In the case of applications that involve authentication such as identity check at Automated Teller Machines (ATMs) and entries to restricted areas, τ should be low enough to accept only known faces in the training set.

5.1.3 Improving Recognition Performance

To obtain a solution that is robust to directional lighting and expression changes and to reduce the size of the network, a modular PCA approach that is based on [121] is performed on sub-images. For this purpose, the face image is divided into smaller equal-sized n sub-images (see Figure 5.2) and those sub-images are fed to the network instead of the entire face image. During recognition, the identities of parts of new face are first found out. The identity of the entire face is p^* if the maximum number of its parts is identified as p^* .

Further, in the case of very large training set, the training set is divided into different subsets and each subset of faces is applied to train one PCNN to improve the performance during recognition. The number of trained PCNNs will be equal to the number of subsets. The face images in a subset are projected onto eigenfaces extracted by the corresponding PCNN. These projections are the outputs of the trained network when the face images are given as input. During recognition, the given new face is presented to all PCNNs. The output vector of each PCNN is compared with the projection vectors of faces in its training subset. Each comparison results in a scalar difference value (d_p). The identity of new face is the one that corresponds to the minimum d_p . Besides improving recognition performance, dividing large training set has the advantage of reduced retraining time when there is amendment in the training set. Only those PCNNs corresponding to the amended subsets are retrained.

Figure 5.2: Sub-images of face images

Figure 5.3: Sample images of faces of a person in the Yale (top) and FRGC (bottom) databases. The first two FRGC samples are controlled images and the other two are uncontrolled images.

5.2 Performance Studies

The performance of PCNN-based modular eigenface analysis has been evaluated on Yale [122] and FRGC [123, 124] face databases. The neural network-based face recognition system has been implemented in Matlab running on a PC with Intel EM64T processor, 3 GHz clock, 4 GB physical memory and Windows XP operating system.

5.2.1 Database Description

The Yale database has normalized face images of 15 persons, each with 11 samples. The images vary widely in illumination and expression. Figure 5.3 shows samples of one person.

The FRGC database is a large-scale face database consisting of images of persons captured at different sessions using a Canon Power Shot G2 camera (Fall 2002, Spring 2003, Fall 2003, Spring 2004). There are 4003 subject sessions in total. A subject session is the set of all images taken from a particular subject in a particular session. The database has frontal faces with two expressions, smiling and neutral. Four controlled and two uncontrolled still images were collected from each person in a session. The controlled images were taken in a studio setting under two lighting conditions while the uncontrolled images were taken in varying illumination conditions, e.g., hallways, atriums or outside. The database consists of three data sets. A “training set” is meant for building the face model (computing eigenfaces for example). A “target set” and a “query set” consist of controlled and uncontrolled images respectively of same persons. The database consists of data sets for training and validation. The images for training were collected in the 2002-2003 academic year while the images for validation were collected during 2003-

2004 academic year. FRGC version 2.0 provides data sets for six experiments. We have considered experiment 4 whose data set has both controlled and uncontrolled images, which is designed for the most challenging 2D face recognition under uncontrolled illumination. The data set consists of 12,776 face images (both controlled and uncontrolled) from 222 people for the purpose of training and 24,042 face images (both controlled and uncontrolled) from 466 people for validation purpose, while the target set has 16,028 face images and the query set has 8014 face images from 466 people. The FRGC database also provides files containing the locations of eyes, nose and mouth. The face region in each image were segmented using this information. Figure 5.3 shows samples of segmented faces of a person.

5.2.2 Performance Evaluation

In our experiments, five samples of each person in the Yale database have been used for training, and the rest have been used for testing. Altogether, 75 images form the training set and 90 images form the test set. In the case of FRGC, four training images (two controlled and two uncontrolled) of each subject have been considered for training. For recognition studies, four images (two controlled and two uncontrolled) of each subject from the validation set of Experiment 4 have been considered. Hence, the training and test sets have 888 and 1864 images respectively. The images were converted to gray-scale images. All the images used for experiments have been resized to 32×32 . Smaller size images have been chosen because increasing the size didn't have a significant impact on the recognition performance as shown in Figure 5.4.

Two different approaches are used to train the PCNN using sub-images for modular PCA method. One approach uses a single PCNN to process all the sub-images of a given

image in a sequence, starting from top-left of the image to the bottom-right of the image. The second approach applies different PCNNs to different parts of the image. The results are plotted in Figure 5.6 and Figure 5.7 for both single-PCNN and multi-PCNN cases. The figures show the plots of recognition rate for varying number of neurons M and the normalized τ set to 0.35 (determined empirically based on inter-class and intra-class d_p values). The plots are for different number of sub-images n . The recognition rate is defined as the ratio of the number of images that were recognized correctly to the number of the images in the test set. It can be observed that the recognition rate increases as M increases and it reaches saturation for a particular value of M , say M_n for each n . There is no significant improvement in performance for $M > M_n$. M_n is close to 16 in both cases. It can be seen that best performance in the case of Yale and FRGC databases is achieved for $n = 4$ and $n = 8$ respectively. For larger values of n , the global facial features in the image parts are lost and hence the recognition rate deteriorates. The best values of n differ in both databases because Yale database faces have wide expression variations resulting in significant local distortion for large n . The sub-images for $n = 4$ and $n = 8$ are shown in Figure 5.2. Further division of these sub-images results in division of important facial features like eyes and hence it results in degrading performance. It can also be observed from Figures 5.6 and 5.7 that multi-PCNN shows an improvement in recognition performance over single-PCNN especially in the case of Yale database. However, multi-PCNN consumes more hardware resources compared to single-PCNN. Single-PCNN has been chosen for hardware implementation described later in Section 6.3.

The performance of division of training set into different subsets as proposed in Section 5.1.3 is also evaluated on FRGC data set. M and n are set to 16 and 8 respectively. Experiments are conducted for varying subset sizes. In the case of each subset size,

different PCNNs are trained with different subsets of training images. The test faces are then given to these trained PCNNs to compute the recognition rate. The plot of recognition rates for the cases of different subset sizes is shown in Figure 5.5. It can be observed that the recognition performance improves for smaller subset sizes.

The time taken for training and recognition has been computed in order to assess the real-time performance of the proposed system. Since FRGC data set is larger, it has been chosen for studies pertaining to execution time. For the purpose of comparison, the neural network-based face recognition system has been implemented in C++ running on a PC with Intel Core2 Duo E8500 processor, 3.16 GHz clock, 4 GB physical memory and Windows 7 operating system. In the case of single-PCNN with $M = 16$ and $n = 8$, the time taken for training for 100 iterations is 3.46 s while the average time taken to recognize a test image is 0.0465 s. This results in the recognition of only 21 faces per second.

Figure 5.4: Recognition performance for varying sizes of face images using multi-PCNN with 16 eigenfaces on FRGC database.

Figure 5.5: Recognition performance for the division of training set

5.2.3 Word Length Effects

In the performance studies, data is represented as a floating point number using 32 bits. The word length influences the recognition rate. Representing data with a higher precision requires increased memory resources and also increases the computational complexity. Fixed point representation is a good alternative for the more complicated and computationally expensive floating point representation. In fixed point representation, a real valued number is represented using a fixed number of digits before and after a radix point(or decimal point). Normalized image pixel values range from 0 to 1 and the weight values range from -1 to 1. In a fixed point representation each real valued number is represented using I integer bits, F fraction bits and 1 sign bit. Simulations were per-

(a)

(b)

Figure 5.6: Performance of single-PCNN and multi-PCNN on Yale face database. The numbers in the legend are the number of sub-images.

(a)

(b)

Figure 5.7: Performance of single-PCNN and multi-PCNN on FRGC face database. The numbers in the legend are the number of sub-images.

formed for varying number of fraction bits to represent the image pixel values and weight vectors keeping one integer bit. Simulation results show that maximum performance is achieved with a minimum word length of 8 bits with 1 integer bit and 7 fraction bits. The results also show that even with shorter word lengths like 4 bits, reasonable performance is achieved. The results in Figure 5.8 show the recognition rate for different word lengths and number of eigenfaces chosen.

Figure 5.8: Word length effects on recognition rate. The numbers in the legend are the word lengths chosen

5.3 Conclusions

A principal component neural network (PCNN) based face recognition system has been proposed. The proposed system computes eigenfaces and recognizes a new face using the same network. The performance evaluation of the system on Yale and FRGC face databases shows that a recognition rate of more than 85% has been achieved by suitable choice of parameters such as number of neurons and sub-images. The PCNN with linear neurons using the generalized Hebbian learning algorithm facilitates reduced-complexity hardware realization of the computationally-intensive face recognition task. The Matlab implementation of the method on a high-speed PC results only in a recognition of 5 faces per second. The recognition time is too large for real-time video surveillance applications in which images normally arrive at a rate of 30 frames per second. Moreover, more than one face may have to be recognized in a frame. In order to achieve substantial speed-up, hardware implementation of the system is necessary. The mapping of the face recognition system onto a reconfigurable systolic architecture is presented in the next chapter.

CHAPTER 6

Systolic Architecture of Neural Network-Based Face Recognition Algorithm

In real-world situations, new faces are added and/or some already enrolled faces may be removed from the database quite often. Further due to ageing, changes occur in the face and hence, faces stored have to be replaced by the most recent ones. In view of these, when the face database changes, the eigenfaces and projections have to be recomputed and reloaded manually and it is advantageous therefore to perform eigenface extraction using dedicated hardware. Prior architectural efforts, to the best of our knowledge, have not examined the computation of eigenfaces in custom hardware. This chapter presents a simple, linear and reconfigurable systolic array realization of the neural network-based face recognition system.

6.1 Proposed Systolic Mapping of the Neural Algorithm

As discussed in Section 5.1, the proposed PCNN-based face recognition system operates in two distinct phases, namely, the training phase and the recognition phase. Both these phases could be implemented through the following steps:

Steps in Training Phase:

T-1: Computation of neuron outputs for a training face X_p

T-2: Updating the neuron weights

T-3: Computation of projections of X_p onto the eigenfaces.

Steps T-1 and T-2 are repeated for P training faces during each iteration of training.

Steps in Recognition Phase:

R-1: Computation of neuron outputs for a new face X^*

R-2: Computation of d_p .

We construct here the fine-grained Signal Flow Graph (SFG) of the computations involved in each of these steps and map all the SFGs corresponding to different computational steps into a single systolic array architecture.

6.1.1 Computation of Neuron Outputs for Training Faces

The neuron outputs y_{pj} , $1 \leq j \leq M$, for an input face $X_p = [x_{p1}, x_{p2}, \dots, x_{pN}]$ in the vectorized form are computed according to GHA as a set of inner-products, given by

$$y_{pj} = \sum_{i=1}^N w_{ji} x_{pi} \quad (6.1)$$

where w_{ji} , for $i = 1, 2, \dots, N$ and $j = 1, 2, \dots, M$ are the weights connecting the i^{th} input to the j^{th} neuron. The index p refers to the training face, for $1 \leq p \leq P$. A fine-grained SFG depicting all the inputs, outputs, computations and dependencies is shown in Figure 6.1(a). It consists of N pairs of nodes, where each pair consists of a node- A_1 and a node- B . The function of nodes A_1 and B are shown in Figures 6.1(b) and

6.1(c), respectively. The inputs to the node- B include the gray values x_{pi} , $1 \leq i \leq N$, and the weight values as shown in the figure. It multiplies the input gray value with the corresponding weight, and passes the product value to node- A_1 . Node- A_1 accumulates the input available from node- B . A pair of nodes (B and A_1) of the SFG thus performs a multiply-accumulate operation to compute the value of y_{pj} (according to Equation (6.1)) in N consecutive time-steps, such that y_{pj} for $j = 1, 2, \dots, M$ are computed in M different SFG nodes.

6.1.2 Updating Weights

Once the neuron outputs are computed, the weights are updated. The new weight values of the neurons are computed according to the following equation

$$\begin{aligned} (w_{ji})_{new} &= (w_{ji})_{old} + \Delta w_{ji} \\ \Delta w_{ji} &= \mu y_{pj} [x_{pi} - \sum_{k=1}^j w_{ki} y_{pk}] \end{aligned} \quad (6.2)$$

for $1 \leq j \leq M$, $1 \leq i \leq N$, and $1 \leq p \leq P$.

The updating of weights is done in two stages. In the first stage, the summations in Equation (6.2) are performed as

$$PS_{ji} = \sum_{k=1}^j w_{ki} y_{pk} = PS_{(j-1)i} + w_{ji} y_{pj} \quad (6.3)$$

for $1 \leq j \leq M$, $1 \leq i \leq N$ and $1 \leq p \leq P$.

In the second stage, the new values of weights are computed using PS_{ji} as

$$(w_{ji})_{new} = (w_{ji})_{old} + \mu y_{pj} [x_{pi} - PS_{ji}] \quad (6.4)$$

Figure 6.1: (a) SFG for computation of neuron outputs for a training face. (b) Function of node A_1 . (c) Function of node B . 'D' represents a unit delay on an edge.

for $1 \leq j \leq M$, $1 \leq i \leq N$ and $1 \leq p \leq P$.

A fine-grained SFG for both of these stages of weight updating is shown in Figure 6.2. The function of node- B is the same as that described in Figure 6.1(c). The function of nodes A_2 and A_3 are shown in Figures 6.2(c) and 6.2(d), respectively. For the first stage of weight updating (according to Equation (6.3)), the j^{th} node- B is fed with the neuron output y_{pj} which is available after the computation of previous step. The node multiplies y_{pj} with the corresponding weight value and passes the product value to node- A_2 . Node- A_2 adds the product received from node- B with the partial sum received from its predecessor node as horizontal input X_{in} . The first node- A_2 of the SFG is fed with '0' as horizontal input X_{in} since it has no previous node. The new partial sum computed by node- A_2 is passed to the next node as well as to node- A_3 for the computation of the second stage of weight updating (according to Equation (6.4)). The pixel values x_{pi} , $1 \leq i \leq N$, are fed sequentially to node- A_3 as the other input. This node computes the difference $[x_{pi} - PS_{ji}]$ and passes it to node- B . The node- B multiplies the input received (from node- A_3) with the scaled input y_{pj} . The scaling is done by node- S_μ . The resulting product from the j^{th} node- B is Δw_{ji} which is passed to the j^{th} node- A_1 for updating w_{ji} .

6.1.3 Computation of Projections of Training Faces onto Eigenfaces

After repeated updating of weights for different input faces X_p in several iterations, the weights of the trained network converge to optimized eigenfaces $\widehat{W}_j$. The projection of each face X_p onto $\widehat{W}_j$ is computed as follows.

$$\hat{y}_{pj} = \sum_{i=1}^N \hat{w}_{ji} x_{pi} \quad (6.5)$$

Figure 6.2: SFGs for weights updating. (a) SFG for the first stage given by Equation (6.3). (b) SFG for the second stage given by Equation (6.4). (c) Function of node- A_2 . (d) Function of node- A_3 . (e) Function of node- S_μ .

The SFG depicting the computation is similar to Figure 6.1(a) and it is shown in Figure 6.3. Here $\hat{w}_{ji}$ s are the optimized weight values and the outputs are the projections of training faces onto eigenfaces given by the optimized weights.

Figure 6.3: SFG for the computation of projections of training faces onto eigenfaces.

6.1.4 Computation of Neuron Outputs for a New Face

Given a new face $X^* = [x_1^*, x_2^*, \dots, x_N^*]$, the projections of X^* onto eigenfaces $\hat{W}_j$ are computed as

$$y_j^* = \sum_{i=1}^N \hat{w}_{ji} x_i^* \quad (6.6)$$

The SFG depicting the computation is the same as Figure 6.3, except that the inputs

in this case are the pixel values of new face. The SFG for the computation of y_j^* is shown in Figure 6.4.

Figure 6.4: SFG for the computation of neuron outputs for a new face.

6.1.5 Computation of Difference Between Projections

d_p is computed as the sum of differences between the corresponding components of the projected outputs $Y^* = [y_1^*, y_2^*, \dots, y_M^*]$ of a new face X^* and $\hat{Y}_p = [\hat{y}_{p1}, \hat{y}_{p2}, \dots, \hat{y}_{pM}]$ of each of the P face images in set S . The computations of Y^* and $\hat{Y}_p$ are shown in Figures 6.4 and 6.3 respectively. d_p is computed for each face image p according to the equation

$$d_p = \sum_{j=1}^M |\hat{y}_{pj} - y_j^*| \text{ for } 1 \leq p \leq P \quad (6.7)$$

The fine-grained SFG depicting the computation of d_p is shown in Figure 6.5(a). The function of node- A_4 is shown in Figure 6.5(b), while that of node- A_2 is described in Figure 6.2(c). Node- A_4 is fed with $\hat{y}_{pj}$ and y_j^* which computes the absolute difference between them. The difference value is then passed to node- A_2 as vertical input which adds it to the accumulated differences received from the previous node- A_2 as horizontal input. The j^{th} node- A_2 thus computes the partial sum $PD_j = \sum_{k=1}^j |\hat{y}_{pk} - y_k^*|$. The horizontal input X_{in} to the first node- A_2 is '0'. d_1 is the first output of the M^{th} node A_2 (the right-most node A_2) after M time-steps. In the successive $M - 1$ time-steps, d_2 to d_M are available as output from the same node A_2 .

Figure 6.5: (a) SFG for the computation of d_p . (b) Function of node A_4 .

6.2 Architecture Design

In this section, we analyze the computations associated with the SFGs of different steps of processing in the proposed PCNN-based face recognition system, and formulate a mapping of all the SFGs to a single systolic array architecture. The computations shown in the SFGs for the steps T-1 (Figure 6.1(a)), T-3 (Figure 6.3) and R-1 (Figure 6.4) are identical. All the five steps of the training and recognition phases therefore could be classified into three modes of operations, viz,

Mode-1: Computation of neuron outputs,

Mode-2: Updating weights and

Mode-3: Computation of d_p .

In a straightforward implementation, the SFGs for the different modes could be mapped onto different systolic arrays each consisting of M processing elements (PEs) by feed-forward cut-set retiming. But that would involve high hardware cost and would lead to inefficient hardware utilization. Since the operations of different modes are performed sequentially, most of the time the processors would remain idle. For better hardware utilization without significantly degrading the timing performance, the computation of all the five steps could be multiplexed into a single systolic array of M PEs as shown in Figure 6.6. M determines the number of eigenfaces extracted by the array. The control functionalities and internal structures of the PEs are required to be designed appropriately to take care of the computation of all the modes of operation with reduced hardware complexity and least possible cycle time.

The overall face recognition system, however, could be obtained by interfacing appropriate inputs and outputs to the PEs at different modes of operation. The presentation

Figure 6.6: Systolic architecture. M_1, M_2 and M_3 are mode-control signals corresponding to Mode-1, Mode-2 and Mode-3. PS_j and PD_j are the partial sums computed by the j^{th} PE in Mode-2 and Mode-3 respectively.

of mode-control signals determines whether the system is in training or in recognition phase. When the system is in training phase, Mode-1 and Mode-2 are repeated for training with the input faces. In recognition phase, the system is put in Mode-1 followed by Mode-3. The face images are stored in a buffer from which they are sent (pixel by pixel) to the systolic array. The output d_p of the array during the recognition phase is given to a minimum finding logic to determine d , the minimum of d_p values, and its corresponding identity p^* .

6.2.1 Processing Element Design

For the implementation of operations of Mode-1 (given by SFGs in Figures 6.1, 6.3 and 6.4), each PE needs one multiplier, one adder and a register for output accumulation. For the implementation of Mode-2 operations (given by SFG in Figure 6.2), two multiplications, three additions and one scaling operation are required. This could be implemented by a PE consisting of one multiplier, two adders and one hardwired scaling unit, if we assume the cycle time to be of duration $T = T_M + 2T_A$, where T_M and T_A are the time required for one multiplication and one addition, provided that the SFG is implemented in two cycles and the scaling is performed with the multiplications in parallel during the first cycle. The implementation of Mode-3 operations (given by SFG in Figure 6.5) is

relatively simpler. It involves only two adders. Out of all the three modes of operations, Mode-2 involves maximum hardware resources consisting of two multipliers, two adders and a scaling unit for a PE. For this mode, two cycles are required by each PE in the systolic array, while for all other modes only one cycle is required by the PEs for implementing the SFGs.

The proposed structure of the PE with reusable arithmetic units is shown in Figure 6.7. Each PE has two memory units. The weight memory is of size N to store the eigenface extracted by that PE, while the output memory is of size P to store the P output values y_{pj} , for $1 \leq i \leq P$. The arithmetic units in a PE are: (i) one adder, (ii) one multiplier, (iii) one accumulator and (iv) a scaling unit. Besides, multiplexers and demultiplexers are employed to select appropriate inputs and outputs of these arithmetic units during different modes of operation of the PE. The select signals for the multiplexers and demultiplexers are derived from the mode-control signals M_1 , M_2 and M_3 . The data flow and computations during different phases and the generation of control signals are explained in detail next.

Data Flow and Computations During Training Phase

In Step T-1 (given by SFG in Figure 6.1), x_{pi} is fed to the PE. The PE is in Mode-1. The MUX0 selects x_{pi} (for $S_0 = 0$) and the MUX2 selects w_{ji} (for $S_2 = 0$), so that x_{pi} and w_{ji} are pair of inputs to the multiplier. The output is demultiplexed (for $S_3 = 0$) and added to the partial accumulated products $\sum_{k=1}^{i-1} x_{pk}w_{jk}$. The PE performs this operation for N inputs, x_{pi} , $i = 1$ to N , of a training image. The final output y_{pj} of the accumulator is stored in the memory block by choosing $S_4 = 0$.

Step T-2 is executed in two cycles. The PE is in Mode-2. x_{pi} is fed to the PE and

Figure 6.7: Processing element architecture. The structure and function refers to the j^{th} PE. SU indicates a scaling unit to perform multiplication with the learning rate μ .

stays at the input line for both cycles. In the first cycle, as given by the SFG in Figure 6.2(a), y_{pj} is fed to the multiplier. This is selected by the multiplexers MUX1 with $S_1 = 0$ and MUX0 with $S_0 = 1$. The other input w_{ji} to the multiplier is selected by MUX2 with $S_2 = 0$. The multiplied output is passed to the adder by selecting $S_3 = 1$ and $S_5 = 0$. It is added with the accumulated product value, $PS_{(j-1)i} = \sum_{k=1}^{j-1} w_{ki}y_{pk}$, received by the PE. Through the multiplexers MUX6 and MUX7 with $S_6 = 0$ and $S_7 = 0$, $PS_{(j-1)i}$ is sent to the adder. The output of the adder, PS_{ji} , is passed to the next PE through the demultiplexers DMUX8 and DMUX11 with select signals $S_8 = 0$ and $S_{11} = 0$. In the second cycle, the output PS_{ji} of the adder is negated by two's complement operation and fed back to the adder. The select signals S_9, S_{12} and S_5 of the multiplexers MUX9, MUX12 and MUX5 take the values 0, 0 and 1, respectively. The negated PS_{ji} is added with input x_{pi} that is fed to the adder when $S_{10} = 0$ and $S_7 = 1$. The output of the adder, $x_{pi} - PS_{ji}$, is passed to the multiplier by selecting $S_8 = 1$ and $S_2 = 1$. The other input of the multiplier is the scaled y_{pj} , i.e. μy_{pj} , which is obtained by selecting $S_1 = 0$ and $S_0 = 1$. The scaling by learning rate is done by a dedicated scaling unit SU. The product $\mu y_{pj}[x_{pi} - PS_{ji}]$ is added to the weight w_{ji} stored in the accumulator. The output of the accumulator is the new weight which is stored in the weight memory. The select signals of the demultiplexers DMUX3 and DMUX4 are $S_3 = 0$ and $S_4 = 1$. Note that the operation in the Step T-3 is same as that of Step T-1.

Data Flow and Computations During Recognition Phase

In Step R-1, the data flow in PE is the same as Step T-1. x_i^* is fed to the PE. y_j^* is computed using the trained weights, $w_{ji}, i = 1$ to N , available from the memory of the PE and the inputs $x_i^*, i = 1$ to N , of the new face.

Step R-2 as described by the SFG in Figure 6.5 is executed by utilizing the adder in both half cycles of a cycle. The PE is in Mode-3. In the first half cycle of Step R-2, y_j^* stored in the accumulator is negated and fed to the adder by choosing $S_9 = 1$, $S_{12} = 0$ and $S_5 = 1$ of the multiplexers MUX9, MUX12 and MUX5. The second input to the adder, $\hat{y}_{pj}$, is taken from the memory by choosing $S_{10} = 1$ and $S_7 = 1$ of the multiplexers. In the second half cycle, the output of adder is fed back to itself after taking the absolute value of it. This is done by testing the MSB of the output keeping $S_{12} = 1$. If the output is negative, then $MSB = 1$ and the select signal of MUX12 will be 1. The negated output is fed back to the adder. If the output is positive ($MSB = 0$), then it is directly fed back to the adder. S_5 and S_9 of multiplexers are set to 1 and 0, respectively. This absolute value is added to PD_{j-1} . PD_{j-1} is fed to the adder by choosing $S_6 = 1$ and $S_7 = 0$. The output PD_j is fed to the next PE by setting S_8 to 0 and S_{11} to 1 of the demultiplexers.

Generation of Control Signals

The select signals (S_0 to S_{12}) of the multiplexers and demultiplexers control the flow of data to the arithmetic units in different modes of operations. These control signals can be generated from the mode-control signals and the clock signal. Table 6.1 shows the values of control signals during different modes of operations. Cycle-1 and Cycle-2 are alternate cycles. HalfCycle-1 and HalfCycle-2 refer to the first and the second halves of a cycle. The circuit for the generation of control signals of the PE from the clock signal and the mode-control signals M_1 , M_2 and M_3 are shown in Figure 6.8.

The sequence of activation of mode-control signals and the presentation of inputs during the training and recognition phases in the j^{th} PE is given by the flowcharts in Figure 6.9. The values of p and iteration numbers are generated by external counters. During

Table 6.1: Control signal generation

Signal	Value	Condition
S_0	0	Mode-1
	1	Mode-2
S_1	0	Mode-2 and Cycle-1
	1	Mode-2 and Cycle-2
S_2	0	Mode-1 or (Mode-2 and Cycle-1)
	1	Mode-2 and Cycle-2
S_3	0	Mode-1 or (Mode-2 and Cycle-2)
	1	Mode-2 and Cycle-1
S_4	0	Mode-1
	1	Mode-2 and Cycle-2
S_5	0	Mode-2 and Cycle-1
	1	(Mode-2 and Cycle-2) or Mode-3
S_6	0	Mode-2 and Cycle-1
	1	Mode-3
S_7	0	(Mode-2 and Cycle-1) or (Mode-3 and HalfCycle-2)
	1	(Mode-2 and Cycle-2) or (Mode-3 and HalfCycle-1)
S_8	0	(Mode-2 and Cycle-1) or (Mode-3 and HalfCycle-2)
	1	Mode-2 and Cycle-2
S_9	0	Mode-2 and Cycle-2
	1	Mode-3
S_{10}	0	Mode-2 and Cycle-2
	1	Mode-3
S_{11}	0	Mode-2 and Cycle-1
	1	Mode-3 and HalfCycle-2
S_{12}	1	Mode-3 and HalfCycle-2

Figure 6.8: Circuit for control signal generation.

the training phase, the architecture is input with training faces X_p when it is allowed to run in Mode-1 and Mode-2 alternatively. The PE is in Mode-1 for N cycles and in Mode-2 for $2N$ cycles. The training phase results in the computation of projection vector elements $\hat{y}_{pj}$ s. In the recognition phase, the PE is input with new face X^* . It is in Mode-1 for N cycles to compute y_j^* . Then the PE is put in Mode-3 for P cycles. The stored $\hat{y}_{pj}$ s along with the computed y_j^* are used when the PE is in Mode-3 to compute the partial sum PD_j for the d_p values. It is worth noting that all PEs have the same sequence of operations. However, they need not be in the same mode of operation at any time.

The PE is *reconfigurable* in run time. It changes its functionalities depending on the mode of operation. To achieve such reconfigurability, the data-flow pattern in the PE are reconfigured according to the mode of operation to perform the desired function. The

Figure 6.9: Combined control flow diagram for the j^{th} PE in training and recognition phases.

reconfiguration is achieved by using multiplexers and demultiplexers which change the data-flow by appropriately generating the select signals (derived from the mode-control signals). The arithmetic units in the PE are reused in different modes of operation.

6.2.2 Operation of Overall Architecture

During each cycle, the systolic architecture (shown in Figure 6.6) is fed with the mode control signals and a pixel value. These inputs are passed on to the successive PEs at successive clock cycles along with the partial sum PS_{ji} or PD_j if computed. The scheduling of different steps in the training and the recognition phases for the overall systolic architecture is shown in Figure 6.10 and Figure 6.11. Note that different PEs may be functioning in different modes of operation at a given cycle in the pipelined structure. The overlapping of different computations at certain time intervals (for example, between N^{th} and $(N + M - 1)^{th}$ clock cycles in Figure 6.10) indicates the PEs in different modes. Mode-1 takes $N + M - 1$ cycles to process a face image vector with N elements. Mode-2 and Mode-3 take $2(N + M - 1)$ and $(P + M - 1)$ cycles, respectively. External counters are used to count the pixels input to the system and to set the mode-control signal inputs to the array at appropriate time specified in Table 6.2. The presentation of data for different mode-control inputs is also shown in the table.

During training phase, M_1 is set for the first N clock cycles followed by setting M_2 for the next $2N$ cycles. The input data will be the pixels x_{pi} , $1 \leq i \leq N$ of the first training image ($p = 1$). When M_2 is set, each pixel is input for two clock cycles. After a delay of $M - 1$ cycles, M_1 and M_2 are set again for the same number of cycles and the pixels of second training image ($p = 2$) are fed as before. The pattern of setting these two mode control signals and the presentation of image pixels is repeated for IP times.

This performs the computation of neuron outputs and weights updating. The projections are then computed by the architecture by setting M_1 for PN cycles. During recognition phase, M_1 is set for N cycles followed by M_3 for P cycles. When M_1 is set, pixels x_i^* , $1 \leq i \leq N$, of the new face image are fed to the array one after another. When M_3 is set, the computation of d_p starts. However the output d_1 is available from the M^{th} PE after $M - 1$ cycles. The remaining outputs d_2 to d_P are available in the successive clock cycles.

Table 6.2: Activation of mode-control signals and presentation of input data

Phase	Mode-control signal	Starting clock cycle	Ending clock cycle	Input data
Training	M_1	$l(3N + M - 1) + 1$ $0 \leq l < IP$	$l(3N + M - 1) + N$ $0 \leq l < IP$	$[x_{p1}, x_{p2}, \dots, x_{pN}]$ $1 \leq p \leq P$ for I times
		$IP(3N + M - 1) + 1$	$IP(3N + M - 1) + PN$	$[x_{p1}, x_{p2}, \dots, x_{pN}]$ $1 \leq p \leq P$
	M_2	$l(3N + M - 1) + N + 1$ $0 \leq l < IP$	$l(3N + M - 1) + 3N$ $0 \leq l < IP$	$[x_{p1}, x_{p1}, x_{p2}, x_{p2}, \dots, x_{pN}, x_{pN}]$ $1 \leq p \leq P$ for I times
Recognition	M_1	1	N	$[x_1^*, x_2^*, \dots, x_N^*]$
	M_3	N+1	N+P	–

Computational Analysis

We give here a detailed computational analysis for the proposed architecture assuming that a total of M PEs are employed. The overall results can be summarized as follows and the details are given via Propositions 1, 2, 3 and 4 (and Remarks 1 and 2).

- The entire training phase takes C_T clock cycles (Propositions 1 and 2), where

$$C_T = IP(3N + M - 1) + PN + (M - 1) \quad (6.8)$$

- The recognition phase takes C_R clock cycles (Propositions 3 and 4), where

$$C_R = N + P + M - 1 \quad (6.9)$$

Figure 6.10: Scheduling of computations during the training phase.

Figure 6.11: Scheduling of computations during the recognition phase.

Proposition 1. *The computation of neuron outputs and weight updating takes $(IP(3N + M - 1) + M - 1)$ cycles.*

Proof. Each of the I iterations during the training phase involves operations of Mode-1 followed by Mode-2 for each of the P faces to be trained as shown in Figure 6.10. In the first iteration, when the first face image is fed, the computation of Mode-1 is started in the first cycle and continues till the $(N + M - 1)^{th}$ cycle. The execution of Mode-2 then starts at N^{th} cycle and continues till the cycle $(3N + 2M - 2)$. For the second face, the computation starts at $(3N + M - 1)^{th}$ cycle and continues till cycle labeled $2(3N + M - 1) + M - 1$. The first iteration for the P^{th} face image, therefore starts at $((P - 1)(3N + M - 1))^{th}$ cycle and ends at $(P(3N + M - 1) + M - 1)^{th}$ cycle. The computations for other iterations are scheduled similarly one after the next so that all the I iterations of training for all the P face images are completed in the $(IP(3N + M - 1) + M - 1)^{th}$ cycle. Q.E.D.

Proposition 2. *The computation of projections takes $(PN + M - 1)$ cycles.*

Proof. Once the training for all the face images are over at the first PE at $(IP(3N + M - 1))^{th}$ cycle, the computation of step T-3 starts for the computation of the projections. The computation of the projection for the first face image is performed for $(N + M - 1)$ cycles, but that of the second face can start after N cycles, since the first PE completes its computation for the first image in N cycles. The computation of the projection for the P^{th} face starts after $(P - 1)N$ cycles and continues for the next $(N + M - 1)$ cycles. The computation of projections therefore starts at $(IP(3N + M - 1))^{th}$ cycle and gets completed at $(IP(3N + M - 1) + PN + M - 1)^{th}$ cycle. Q.E.D.

We now present the results pertaining to recognition phase. Please refer Figure 6.11 for the following propositions.

Proposition 3. *The computation of neuron outputs for a new face takes $(N + M - 1)$ cycles.*

Proof. Let us assume that the computation of neuron outputs for a new face during the recognition phase starts at the first PE in the first cycle. It continues in the first PE up to N^{th} cycle. In the second PE, it starts from the second cycle and continues for the next N cycles. In the M^{th} PE, it continues from M^{th} cycle to $(N + M - 1)^{th}$ cycle. Q.E.D

Proposition 4. *The computation of the projection differences takes $(P + M - 1)$ cycles.*

Proof. The computation of projection difference d_1 starts at the first PE at the N^{th} cycle and gets completed in $(N + M - 1)^{th}$ cycle (in the M^{th} PE). The computation of d_2 starts in the $(N + 1)^{th}$ cycle and gets completed in $(N + M)^{th}$ cycle, while the computation of d_p starts at $(N + P - 1)^{th}$ cycle and gets completed in $(N + M + P - 1)^{th}$ cycle. Q.E.D

Remark 1. *It may be noted that there is an overlap of $M - 1$ cycles between the computation of projections (Proposition 2), and the computation of neuron outputs and weight updating (Proposition 1) in the training phase. Similarly there is an overlap of $M - 1$ cycles between the computation of neuron outputs (Proposition 3) and the computation of projection differences (Proposition 4) in the recognition phase. Hence, the total number of clock cycles taken in each phase is not given by the summation of cycles for the two operations.*

Remark 2. *The benefits of a systolic realization are seen from the low hardware complexity and high operational frequency. Each PE is devoid of arithmetic units that are not amenable for a hardware-friendly solution. The cycle time is small due to simple computational units.*

In the case of a single-PCNN based modular PCA method, the architecture is trained with all the sub-images. Hence P is replaced by nP . The number of clock cycles taken by the training phase in this case, denoted by C_T^s , is given by

$$C_T^s = nIP(3N + M - 1) + nPN + (M - 1) \quad (6.10)$$

During recognition, each sub-image of the new face is compared with the corresponding sub-images of P training faces. Since there are n sub-images, the number of clock cycles for recognition phase, denoted by C_R^s , is given by

$$C_R^s = n(N + P + M - 1) \quad (6.11)$$

6.3 Implementation and Comparisons

6.3.1 Hardware Implementation

The proposed face recognition system has been implemented in a commercially available Xilinx ML403 evaluation platform for embedded systems development. The platform is equipped with a Xilinx Virtex-4 XC4VFX12 FPGA, 64MB DDR SDRAM and industry-standard peripheral connectors and interfaces. The proposed systolic design has been first implemented on the FPGA device and then tested on the evaluation platform.

Implementation on FPGA

The proposed systolic architecture for the single PCNN case has been coded in Verilog in behavioural style. From Figure 5.6 and 5.7, it is clear that there is no significant change in recognition rate when the number of neurons exceeds 16. Therefore, the number of PEs has been taken to be 16 although the proposed architecture is easily scalable for any number of PEs. The size of the registers has been taken to be 16 bits which is double the size of input pixel value (to avoid overflow in the result of multiplication). All the results of computations have been represented using fixed point representation based on the simulation studies. In the case of $n = 8$, an image of size 32×32 is divided into eight parts, each of size 8×16 . Hence, N has been taken to be 128 in the design. Memory elements have been used to store the weights corresponding to $N = 128$ and also the projected outputs of P training images.

The designs for different values of P in powers of two have been then implemented on the target device XC4VFX12 using Xilinx ISE 10.1. The target device has a package number of SF363 and a speed grade of -12. The various options set for synthesis in ISE 10.1 are as follows: Optimization Goal: Speed, Optimization Effort: High, Global Optimization Goal: AllClockNets and RAM Style: Block. Maximum usage of device occurred at $P = 1024$ and the implementation results in this case are given in Table 6.3. For all other lower values of P , the usage of components is nearly the same due to fixed number of PEs. The number of Block RAMs (BRAMs) consumed by the designs is 32 since the design synthesizes 16 PEs and each PE has two memory units. However, when P increases, there is a significant rise in the actual usage of block RAMs synthesizing the output memory unit. Maximum usage of these block RAMs occurred for $P = 1024$. When the training set is larger, there is a need to divide the training set into different

subsets so that the FPGA implementation is constrained by the subset's size instead of the size of the entire training set.

The maximum usable frequency of operation is 136.243 MHz. This remains constant for the designs for any value of P because the clock frequency is determined by the maximum signal propagation delay in the combinational logic path. The delays introduced by the two adders and a multiplier mainly contribute to the overall delay. In the case of training set of 888 images chosen for performance studies in Section 5.2.2 and for parameter settings $I = 100$, $N = 128$, $M = 16$, $P = 888$ and $n = 8$, the training and recognition times on FPGA are estimated to be 2.09s and $60\mu s$ respectively using the frequency of operation obtained and the number of clock cycles (C_T^s and C_R^s) required. This implies that the proposed FPGA-based face recognition system can recognize up to 16518 faces per second compared to only 5 faces per second by the PC-based system as discussed in Section 5.2.2. For a video stream of 30 frames per second, up to 550 faces can be recognized in a frame. The graph showing the recognition performance for different training set size (P) is given in Figure 6.12. It is observed that the FPGA-based recognition system can recognize more than 400 faces per frame for a reasonably large training set. The system is therefore suitable for real-time surveillance in busy public places.

Testing of the Design

The netlist for $P = 1024$ has been downloaded to FPGA on the evaluation platform and tested on the evaluation platform with actual face image inputs from PC. Data was sent from PC to the ML403 board using HyperTerminal running on Windows XP. Xilinx's IP core UARTLite was instantiated to receive data through the RS232 port. Both the

HyperTerminal and IP core were configured to operate at a baud rate of 115200 with no parity bits. The data received at Universal Asynchronous Receiver/Transmitter (UART) buffer was then transferred to DDR RAM by an interfacing program running on PowerPC (hardcore processor in the FPGA device). The PowerPC-based embedded system was developed using Xilinx Platform Studio to interface the FPGA design with the necessary peripherals. The FPGA design was interfaced with LCD and RAM in the board through PowerPC processor. Processor Local Buses (PLB) were used to connect the peripherals with the processor. All interfacing programs were implemented in C and run on PowerPC.

Figure 6.12: Performance of FPGA-based recognition system.

The overall experimental setup is shown in Figure 6.13. The training set of faces

containing normalized face images is stored in the PC. The SDRAM receives face images from the HyperTerminal through RS232 interface and sends the data (pixel by pixel) to the FPGA during training and recognition phases. During training, all the training faces are transferred to SDRAM and the computation of eigenfaces and projections of training faces onto eigenfaces are performed on the FPGA. These computed quantities are then used for recognition. It is worth noting that the entire training set (of faces) is not stored in the FPGA. After training, the face image to be recognized is sent from HyperTerminal. The identity of a new face computed by FPGA during recognition phase is displayed on the LCD by the appropriate interfacing program running on PowerPC. '0' is displayed if the identity is not known.

Figure 6.13: ML403 board components and connections relevant to our experiment.

The system has been tested with FRGC ver 2.0 Experiment 4 images. The set of training images chosen for experiments reported in Section 5.2.2 are fed from PC for

Table 6.3: Resource utilization for the FPGA design for the training set of 1024 faces

Component	Available	Usage
Slices	5472	1420 (25%)
Slice Flip Flops	10944	888 (8%)
4 input LUTs	10944	2384 (21%)
Bonded IOBs	240	25 (10%)
18Kb Block RAMs	36	32 (88%)
GCLKs	32	2 (6%)

training. After training, images from the validation set are presented one at a time. The identity displayed on the LCD of the board has been checked. A new face recognized as the 12th face in the training set is displayed as shown in Figure 6.13.

The FPGA in our experimental setup acts as a co-processor executing the computation-intensive face recognition task of a complete system with face detection and recognition. In the experimental setup, it is assumed that a camera is connected to a PC, and the PC runs the face detection program to segment facial regions in the captured video images and normalize them. These normalized face images are either added to the training set for training or directly sent to FPGA co-processor for recognition. However, if the face detection is performed by a dedicated processor, then the complete system could be integrated into a smart surveillance camera.

Handling Large Training Set Size by Run-Time Configuration of BRAMs on FPGA

The FPGA implementation has been extended to handle training sets that have more than 1024 images. The training set is divided into subsets of size 1024 or less. The number of PCNNs is equal to the number of subsets as described in Section 5.1.3 in the case of single-PCNN. However, the systolic architecture is the same for all PCNNs and only the

contents of weights and output memory units vary. Since these memories are mapped onto BRAMs in FPGA, these BRAMs are only reconfigured during training and recognition phases. The weights and projections of the subsets are stored in the external RAM on the board. When a face is added or removed, these values of the amended subset are recalculated. The systolic architecture implementation in FPGA is put in training phase and it is fed with the faces of that subset. After training, the BRAM contents giving the new weight and projection values are transferred to the external RAM. Hence, the total training time (T_T) involves the data transfer time (t_1) and the training time of the systolic array (t_T). That is, $T_T = t_T + t_1$. Since T_T depends on the subset's size and not on the entire training set's size (P), T_T is constant for any P when the subset size is fixed. During recognition, the BRAMs are loaded with the weights and projections of each subset at a time and the new face is fed to the systolic array implementation on FPGA. If there are s subsets, then the total recognition time $T_R = s(t_R + t_2)$ where t_2 is the loading time for BRAMs and t_R is the recognition time of systolic array.

A C program for the data transfer between BRAMs and SDRAM has been written and run on PowerPC in order to estimate t_1 and t_2 . For the subset size of 1024 and for the single-PCNN design implemented on FPGA (Section 6.3.1), t_1 and t_2 are estimated to be 0.025s. t_T in the case of 1024 faces is 2.4s (see Figure 6.12). Hence $T_T = 2.425$ s. The recognition performance of FPGA-based system for the training set size (P) larger than 1024 is shown in Table 6.4. For P as large as 10,000 images, the FPGA-based recognition system with an external RAM can recognize around 2 faces per second. Due to reduced throughput, it is required to consider the video frames sampled with an interval of one second. The recognition system realized on a Virtex-4 FPGA and an external RAM is still suitable for surveillance at narrow or restricted entrances through which only one person can pass. The performance could be enhanced for surveillance at busy places

by implementing the design on a more powerful FPGA device with more BRAMs (for e.g., Virtex-5 XC5VLX330 FPGA with 51,840 slices and 576 18Kb BRAMs, Virtex-6 XC6VSX475T FPGA with 74,400 slices and 1064 36Kb BRAMs).

Table 6.4: Recognition performance of FPGA-based face recognition system with external RAM for large training sets

Training set size (P)	2048	4096	8192	16384
Faces recognized per second	19	9	4	2

6.3.2 Comparison Studies

Comparison with Implementation Based on Run-Time Partial Reconfiguration on FPGA

For comparison purpose, we have implemented the PCNN-based face recognition system consisting of 16 PEs in three configurations (corresponding to three modes) using the run-time partial reconfiguration feature of FPGA device. The memory units, the minimum finding logic and the external control logic have been implemented as static modules while the systolic array designs without memory units have been implemented as reconfigurable modules. The reconfigurable and static modules have been separately synthesized for Virtex-4 XC4VFX12 FPGA in Xilinx ISE. The netlists were then imported into PlanAhead Lite 10.1.1 for implementing the partial reconfiguration. All the three reconfigurable modules were assigned the same reconfigurable region on the FPGA device. The initial configuration for the region was set to the reconfigurable module of Mode-1. The partial reconfiguration was triggered by the change in the mode-control signal during

run-time. The partial bitstreams of the reconfigurable modules were stored in the external DDR SDRAM of ML403 platform. The reconfiguration time depends on the size of the partial bitstream.

During training phase, the reconfiguration is performed $2IP$ times since Mode-1 and Mode-2 operations are repeated IP times. Since all the PEs need to be reconfigured at the same time, the overlapping between Mode-1 and Mode-2 as seen in Figure 6.10 should be avoided. This in turn increases the training time. The training time (T_T^r) in the case of single-PCNN is estimated to be

$$\begin{aligned} T_T^r = & nIP(3(N + M - 1)t_c + t_{r1} + t_{r2}) \\ & + (nPN + M - 1)t_c + t_{r1} \end{aligned} \quad (6.12)$$

where t_c is the clock period, and t_{r1} , t_{r2} and t_{r3} are the reconfiguration times for the bitstreams of Mode-1, Mode-2 and Mode-3, respectively. During recognition phase, the reconfiguration is performed twice, once for each mode of operation. Since there should not be any overlapping between the two modes (Mode-1 and Mode-3), the recognition time (T_R^r) is estimated to be

$$T_R^r = n((N + M - 1)t_c + t_{r1} + (P + M - 1)t_c + t_{r3}) \quad (6.13)$$

From the implementation results, t_{r1} , t_{r2} , t_{r3} and t_c were found to be 0.045s, 0.052s, 0.049s and 7.4ns respectively. For $I = 100$, $N = 128$, $M = 16$, $P = 888$ and $n = 8$ as taken in Section 6.3.1, T_T^r is more than 10 hours and T_R^r is estimated to be 0.75s. Note that the run-time reconfiguration scheme performs recognition of 4 faces in three seconds while the proposed architecture performs recognition of 16518 faces per second using the

same FPGA device.

Comparison with Existing Architectures

A comparison of proposed architecture with the two existing eigenface-based architectures [78, 79] is given in Table 6.5. All the three face recognition algorithms are based on modular PCA. However, the architectures in [78, 79] implement only the recognition phase, which involves the computation of projection onto eigenfaces and the classifier. The eigenfaces are precomputed in software and stored in the memory of architecture. The proposed systolic architecture, on the other hand, implements both training and recognition phases. It is neural network-based and therefore it is area-efficient. Results of implementation of the proposed architecture design (for training set sizes 128 and 1024) in Altera's EP20K600CB652C7 device demonstrate the area-efficiency of the solution. The logic elements consumed are much less when compared to the other two architecture implementations even though the proposed architecture implements both training and recognition phases. This is due to simple reusable logic components in the PEs. The architecture can also be operated at a high speed. The recognition time for a new face is estimated to be $85 \mu s$ for the similar settings as in [78] ($N = 16 \times 16$, $M = 20$, $P = 128$ and $n = 16$ in C_R^s given by Equation 6.11). The recognition time is much less when compared to others due to high throughput of the systolic architecture.

Table 6.5: Comparison with existing architectures

Description		Architecture in [79]	Architecture in [78]	Proposed	
Algorithm	PCA	Weighted modular	Modular	Modular	
	Neural network based	no	no	yes	
Architecture	Type	Direct implementation of computations	Direct implementation of computations in multi-lane	Area-time efficient reconfigurable systolic architecture	
	Eigenface computation	Software	Software	Hardware	
	Computation of projections	Hardware	Hardware	Hardware	
	Classifier	Hardware	Hardware	Hardware	
Synthesis for FPGA	Device	Altera's EP20K200EFC484	Altera's EP20K600CB652C7	Altera's EP20K600CB652C7	
	Training set size	15	120	128	1024
	Logic elements used	7760	19572	5009	5066
	Maximum clock	33 MHz	91 MHz	75 MHz	75 MHz
	Recognition time	38 ms	1.4 ms	85 μ s	0.2 ms
FPGA Implementation	Platform	Altera Stratix II	Nil	Xilinx ML 403	
	Provision for handling large training set	–	–	By run-time configuration of BRAMs	

6.4 Conclusions

An easily scalable and reconfigurable systolic architecture has been designed for efficient hardware realization of the proposed PCNN-based face recognition system through fine-grained SFGs of different phases of operation of the system. Unlike the existing architectures which involve both software and hardware components, the proposed architecture has been fully implemented only in hardware. Besides, the proposed architecture is different from existing architectures in terms of its ability to update eigenfaces whenever faces are added to or removed from the enrolled face database. A reconfigurable PE of the architecture has been designed with minimum resources and maximum hardware utilization by appropriate time-multiplexing of the computations pertaining to different modes of operation.

The proposed systolic design has been implemented and tested on a Xilinx FPGA-based evaluation platform. A systolic array of 16 PEs has been implemented in a Xilinx FPGA device. From the implementation results we find that the proposed system can recognize more than 400 faces in an image frame with the device storing the features of 1024 faces of enrolled face database. For larger databases, the FPGA-based recognition system along with an external storage device can still process at least one face in a second. The proposed face recognition system would therefore be quite suitable for human recognition by an autonomous navigator and also for real-time surveillance.

CHAPTER 7

FFT-Based Fast Face Detection Using Boosted Eigenfaces

Face detection is a necessary preprocessing step for face recognition and it is needed to locate the faces in an image or video frame. This chapter describes a new eigenface based face detection using boosted eigen features. Though many face detection techniques are available, an eigenface based method will reduce the cost of computing additional features in an eigenface based face detection and recognition system. Eigenfaces have long been used for face detection and recognition. The basic detection and recognition system works by projecting the face images onto a face space constructed from training set of faces. But the distance from the face space is not a reliable measure to classify faces from non-faces as some of the non-faces may also lie close to the face space. Our approach is based on AdaBoost in which a strong classifier is built by boosting a set of weak classifiers. The set of weak classifiers are constructed from the projections onto the eigenvectors of the face space. The proposed system provides significantly better performance compared to the distance measure. We also propose to improve the speed of detection in real images using Fast Fourier Transform (FFT) and provide relevant derivations. A systolic mapping for the integral image calculation is designed.

7.1 Constructing Strong Classifier from PCA Features

PCA (details in Section 2.3.5) has been demonstrated to be an excellent tool for dimensionality reduction, by retaining only those characteristics that contribute most to a given data set. But, PCA is not optimized for class separability. Therefore they cannot be effectively used to classify a given data to be belonging to a particular class or not. To improve classification, we propose to construct weak classifiers from PCA features and combine them to form a strong classifier using AdaBoost algorithm. AdaBoost is a boosting algorithm that has been used to combine weak classifiers built from haar-like features for face detection [74]. AdaBoost takes weak classifiers which perform at least marginally better than a random guess and guarantees to produce a strong classifier whose training error rate decreases exponentially with the number of weak classifiers. PCA features generally have much lower error-rate than the required 0.5 by AdaBoost. Therefore few weak classifiers will be required to build a strong classifier using AdaBoost.

Let $(I_1, \gamma_1), (I_2, \gamma_2), \dots, (I_n, \gamma_n)$ be a labelled set of face and non-face images with I being the image and γ being the corresponding label. γ belongs to the label space $\gamma = \{+1, -1\}$ with $\gamma = +1$ for face and $\gamma = -1$ for non-face. A set of K images $(I_1, I_2, \dots, I_K)$ are chosen from the labelled set of face images for PCA. The PCA results in a set of K' eigenvectors which will be used as a basis to represent the rest of dataset.

7.1.1 Feature Extraction

The eigenvectors obtained from PCA are used to extract the features from labelled set of faces and non-faces. The first set of features is the projections ω_k (Equation 2.4) of both faces and non-faces onto the K' eigenvectors resulting in K' features for each image in

the training set. The next set of features is the distance from the eigenspace ϵ_k^2 (Equation 2.6) for each of the eigenvectors, again resulting in K' features per image. After all such features are extracted, each image in the training database is represented only using those features.

7.1.2 Thresholding Based Weak Classifiers

A weak classifier $h_i(\delta_i, f_i, s_i, \theta_i)$ evaluates a particular feature f_i of an image window δ_i and outputs either 0 or 1. It is given by,

$$h_i(\delta_i, f_i, s_i, \theta_i) = \begin{cases} 1, & \text{if } s_i f_i(\delta_i) < s_i \theta_i \\ 0, & \text{otherwise} \end{cases} \quad (7.1)$$

where s_i is the parity and θ_i is the threshold.

Any feature f_i spans a distribution $p_f(f_i)$ for face images and $p_{nf}(f_i)$ for non-faces. A threshold θ_i is chosen in such a way that the sum of area under the probability distribution curve for false positives and false negatives is minimal, thereby making the classification error minimal. The distribution $p_{nf}(f_i)$ is subtracted from $p_f(f_i)$ and the zero crossing is chosen as the threshold point. If the distribution of the feature values of faces lie below the chosen threshold θ_i then parity s_i is 1, else 0. The threshold selection is shown in Figure 7.1. Therefore, every feature has a corresponding weak classifier constructed based on thresholding.

Figure 7.1: Threshold selection

7.1.3 Boosting

The next step after constructing the weak classifiers is to combine them to form a strong classifier using boosting. The core concept of AdaBoost algorithm is the assignment of a weight distribution to the sample set, and modification of the weights during each iteration depending upon whether that particular sample was classified correctly or not. Initially all the samples are given equal weights. After each iteration, each of the weak classifiers chosen above gives a classification output. Those misclassified are weighted higher than those correctly classified. This is to ensure that subsequent weak classifiers will be weighted and chosen based on how they perform on the samples mis-classified by the previous weak classifiers.

AdaBoost calls a weak classifier repeatedly in rounds $t = 1, 2, \dots, T$. At a given iteration, a classifier $h_t : I \rightarrow \{-1, +1\}$ that minimizes the error with respect to the distribution ω_t is found. The final classifier is constructed as a weighted linear combination of these T hypotheses. The weights α_t used for combining the weak classifier are calculated as $\alpha_t = \ln((1 - \epsilon_t)/\epsilon_t)$, where ϵ_t is the error rate.

Inputs: Example images $(I_1, I_2, \dots, I_n)$ and associated labels $(\gamma_1, \gamma_2, \dots, \gamma_n)$. m is the number of negative examples and $l = n - m$, the number of positive examples.

Initialize: Set the n weights to

$$w_{1,i} = \begin{cases} (2m)^{-1}, & \text{if } \gamma_i = -1 \\ (2l)^{-1}, & \text{if } \gamma_i = 1 \end{cases} \quad (7.2)$$

for $t = 1, \dots, T$ **do**

1. Normalize the weights,

$$w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^n w_{t,j}}$$

so that w_t is a probability distribution.

2. For each feature f_j train a classifier h_j which is restricted to using a single feature.

The error is evaluated with respect to the $w_{t,i}$ s as $\epsilon_j = \sum_i w_{t,i} |h_j(\delta_i) - \gamma_i|$.

3. Choose the classifier h_t as the h_j that gives the lowest error ϵ_j . Set ϵ_t to ϵ_j .

4. Update the weights:

$$w_{t+1,i} = w_{t,i} \beta_t^{1-e_i}$$

where $e_i = (1)0$ if example δ_i is classified (in)correctly, and $\beta_i = \frac{\epsilon_i}{1-\epsilon_i}$

end for

Output: A strong classifier defined by:

$$h(\delta) = \begin{cases} 1, & \text{if } \sum_{t=1}^T \alpha_t h_t(\delta) \geq \frac{1}{2} \sum_{t=1}^T \alpha_t \\ 0, & \text{otherwise} \end{cases} \quad (7.3)$$

where $\alpha_t = \ln(\frac{1}{\beta_t})$

7.1.4 Experimental Results

MIT-CBCL Face database [125] has been used for training. It has 2,429 faces and 4,548 non-faces of size 19×19 . Figure 7.2 shows sample face and non-face images in the database. A subset of training images containing 500 face images are first used for PCA to extract the eigenvectors with high variances. The eigenvectors thus extracted will form the basis vectors on which all other training images will be represented. Each image in the training set, both face and non-face is projected on to the extracted eigenvectors and the corresponding projection values form the feature values as described in the previous sections. The top 50 eigenvectors are chosen for projection. All the projection values and corresponding distance values are calculated and used as feature values. Boosting algorithm chooses one having the least weighted error for training samples amongst these weak classifiers in each iteration. Threshold selection in one of the iterations for boosting is shown in Figure 7.3. The chosen features are then weighted according to their performance and combined to form a strong classifier.

Figure 7.2: Sample images from the database

The constructed strong classifier is used to classify the set of test images containing face and non-faces. The classifier is found to perform very well giving an error rate of only 1.89% for faces and 2.48% for non-faces. The false positive rate is found to be only

Figure 7.3: Selection of threshold θ_i

slightly higher than the false negative rate.

7.2 Extending the Classifier to Detect Faces in Images Using FFT

The strong classifier proposed above can be extended to detect faces in images. Basically, each sub-image of a given image should be checked for the presence of face. The proposed classifier should be applied to each sub-image separately to detect faces. The calculation of the features for each of the sub-images has to be done to apply the classifier. As a given image contains several sub-images to be checked, calculation of features for

all the sub-images is computationally intensive.

7.2.1 FFT Based Computation of Projection Values

We propose an FFT based method for computing these features, so that the total number of computations required will be reduced. Let the given image $G(x, y)$ for detecting faces contain p sub-images $\Gamma_1, \Gamma_2, \dots, \Gamma_p$. There will be one sub-image corresponding to each pixel at location (x, y) in the given image $G(x, y)$. Each of the features required by the classifier has to be computed for all the p sub-images. For example, one of the features, the projection ω_i on to a particular eigenvector is given by $\omega_i = \Psi^T u_i$. Ψ is the centered sub-image represented as a column vector.

The dot product of Ψ and u_i can be considered as a correlation operation given by $\omega_i = \Psi \otimes u_i$. Therefore, the projection value ω_i for all the sub-images can be calculated by correlating the eigenvector with the given image $G(x, y)$ in two-dimension. The correlation operation can be easily performed in the fourier domain as multiplication, thereby reducing the number of computations.

$$\begin{aligned}\omega_i(x, y) &= G(x, y) \otimes u_i \\ &= \mathcal{F}^{-1}\{\mathcal{F}(G(x, y)) \times \mathcal{F}(u_i)\}\end{aligned}\tag{7.4}$$

7.2.2 FFT Based Computation of Distance Values

The distance from a specific eigenvector is given by

$$\epsilon_i^2(x, y) = \Phi^T(x, y)\Phi(x, y) - \omega_i^2(x, y) \quad (7.5)$$

The projection ω_i is given by $\omega_i = \Phi^T u_i$. Therefore,

$$\begin{aligned} \omega_i^2(x, y) &= \Phi(x, y)^T u_i^2 \\ &= [(\Gamma(x, y) - \Psi)^T u_i]^2 \\ &= [(\Gamma(x, y)^T u_i - \Psi^T u_i)]^2 \\ &= [(\Gamma(x, y) \otimes u_i - \Psi \otimes u_i)]^2 \end{aligned} \quad (7.6)$$

Replacing Γ by the given image G ,

$$\omega_i^2(x, y) = [(G(x, y) \otimes u_i - \Psi \otimes u_i)]^2 \quad (7.7)$$

The first term in Equation 7.5 is given by the following equation as $\Phi = \Gamma - \Psi$,

$$\Phi^T(x, y)\Phi(x, y) = \Gamma^T(x, y)\Gamma(x, y) - 2\Gamma(x, y) \otimes \Psi + \Psi^T \Psi \quad (7.8)$$

so that the distance value is given by,

$$\epsilon_i^2(x, y) = G^T(x, y)G(x, y) - 2G(x, y) \otimes \Psi + \Psi^T \Psi$$

$$-[(G(x, y) \otimes u_i - \Psi \otimes u_i)]^2 \quad (7.9)$$

The nature of operations in Equation 7.9 allows us to simplify the computation using FFT. The first term is nothing but the sum of squares of the image pixels, within the window of same size as eigenface for every pixel. This can be easily calculated by using the concept of integral image introduced in [74]. Each pixel in the image is squared first. The integral square image $II^2(x, y)$ in this case at any point (x, y) consists of the sum of squares of all the pixel values to the left and above of the given point. Therefore sum of squares of pixels in any sub-image can be calculated easily by subtracting the sum of bottom left and top right pixels from the sum of bottom right and top left pixels.

If the sub-image size is $n \times n$, then the sum of squares $SS(x, y)$ is given by

$$\begin{aligned} SS(x, y) = & II^2(x, y) + II^2(x - n, y - n) \\ & - II^2(x - n, y) - II^2(x, y - n) \end{aligned} \quad (7.10)$$

Once the integral square image is calculated, it takes the same number of computations to calculate the sum of squares for sub-images of any given size. This reduces the number of computations especially when sum of squares have to be computed at multiple scales. Therefore Equation 7.9 can be rewritten as,

$$\begin{aligned} \epsilon_i^2(x, y) = & SS(x, y) - 2G(x, y) \otimes \Psi + \Psi^T \Psi \\ & - [(G(x, y) \otimes u_i - \Psi \otimes u_i)]^2 \end{aligned} \quad (7.11)$$

From Fourier theory we know that correlation in spatial domain is equivalent to

Figure 7.4: Detected faces

multiplication in the fourier domain. The above equation can be rewritten as

$$\begin{aligned} \epsilon_i^2(x, y) = & SS(x, y) - \mathcal{F}^{-1}\{\mathcal{F}\{2G(x, y)\} \times \mathcal{F}\{\Psi\}\} \\ & + \Psi^T \Psi + [\mathcal{F}^{-1}\{\mathcal{F}\{G(x, y)\} \times \mathcal{F}\{u_i\}\} - \Psi \otimes u_i]^2 \end{aligned} \quad (7.12)$$

Thus FFT can be used to effectively reduce the number of computations. The speed-up achieved varies for different sizes of images and speed-up achieved for a given image of size 600×600 is 6.7 in a C++ implementation. Any given feature for all sub-images is represented by a matrix of the same size as the given image. Several such feature matrices are combined according to the weighted weak classifier rule to output the result. Generally multiple pixels around a face region would report a detected face. In that case, mean of these pixels is considered to be the location of the detected face. Figure 7.4 shows the detected faces.

7.3 Hardware Mapping

The two computationally expensive parts of the Equation 7.12 are the integral image calculation and FFT computation of the input image. Integral image (II) of a given image contains the sum of all the pixels that are above and to the left of a given pixel. For a given image of size $N \times N$, let $I_{1,1}$ be the top left pixel and $I_{N,N}$ be the bottom right pixel.

Let $R_{i,j}$ be the sum of all the pixels to the left of a given pixel in a row. Then the integral image at (i, j) is given by the sum of all the pixels in the i^{th} row up to j^{th} column and the integral image value $(i - 1)^{th}$ row and j^{th} column. The equations for the row sum and integral image are given below.

$$\begin{aligned} R_{i,j} &= R_{i,j-1} + I_{i,j} \\ II_{i,j} &= II_{i-1,j} + R_{i,j} \end{aligned} \tag{7.13}$$

The dependence graph of the computations is shown in Figure 7.5. The schedule vector based on equitemporal hyperplanes is shown in Figure 7.6. The nodes lying along the same hyperplane are executed at the same time. The schedule vector is drawn such that all the hyperplanes meet the causality condition. A linear projection is applied to obtain the signal flow graph shown in Figure 7.7.

Each PE of the systolic array as shown in Figure 7.8 consists of two adders for performing the two additions in Equation 7.13. It also contains two registers for storing $II_{i,j}$ and $R_{i,j}$. In every cycle, a new PE starts its computations and at the end of N cycles,

Figure 7.5: Dependency graph for integral image computation

Figure 7.6: Schedule vector and hyperplanes

Figure 7.7: Signal flow graph. ‘D’ represents unit delay

Figure 7.8: Processing element

all the PEs are in operation. One output is calculated in the 1st cycle, 2 outputs in the second cycle and N outputs from N^{th} cycle onwards. The entire process takes $2N$ cycles to complete. The output integral image values $II_{i,j}$ are stored in an array of memory to be used by the other modules of the face detection system.

The other computationally intensive operation is the calculation of the FFT of the given image. Several architectures have been proposed so far in the literature [126],[127] and one of them can be chosen for building the face detection system.

7.4 Conclusions

A fast boosted eigenface based face detection algorithm has been presented. The proposed classification algorithm has been tested on MIT-CBCL face database and found to provide good performance. A method to extend the classification algorithm for fast detection of faces in real images using correlation and FFT has also been described. A systolic architecture mapping of the integral image calculation has been proposed. The proposed architecture can be combined with an existing FFT architecture to design the face detection system.

CHAPTER 8

Conclusions and Future Directions

In this thesis, new algorithms and VLSI array architectures have been proposed for navigation and recognition sub-systems of vision-guided systems. The algorithms were first simulated in PC using C and MATLAB to study their performance. The sequential nature of algorithms running on PC place severe limitations on the processing performance. Therefore, the algorithms were studied to extract parallelism at the lowest level and then mapped onto a suitable array architecture. Even though powerful PCs can provide near real-time performances, embedded systems cannot afford to have such powerful processors due to size, weight and power considerations. The architectures proposed are intended for powerful co-processors in complex vision-guided embedded systems.

8.1 Conclusions

In Chapter 3, a parallel algorithm for path planning has been proposed. The algorithm performs path planning on an environment image by constructing an Euclidean distance map of the binary image captured using an overhead camera. The proposed algorithm is also capable of tracing multiple paths from multiple start pixels to goal pixels based on the same distance map. Simulations were also performed to study the performance of the algorithm in a dynamic environment. Another important contribution is the decomposi-

tion of different operations in path planning into simple neighborhood operations thereby making it amenable for mapping onto an array architecture.

In Chapter 4, mapping of the parallel algorithm developed for path planning onto a 2D cellular architecture has been presented. The architecture consists of identical processing elements connected to 8 neighboring processing elements. The proposed architecture has been coded in Verilog and implemented on a Xilinx Virtex 4 FPGA. The maximum frequency of operation of a cell is 375 MHz. As the architecture is constructed by simple local neighborhood connections of identical processing elements, the frequency of operation of the entire 2D cellular architecture will also be 375 MHz. Therefore, the important contribution in this architecture is the scalable design that can be extended to perform path planning on any number of pixels without compromising on the performance. The maximum size of the image that can be processed is limited by the number of slices present in the FPGA. A method to overcome this limitation by assigning more pixels to be processed by a single cell but compromising with the speed of operation is also proposed. But as the FPGA technology progresses, the number of slices available in a single FPGA chip keeps on increasing.

In Chapter 5, a Principal Component Neural Network (PCNN) based face recognition algorithm has been proposed. The performance of PCNN-based modular eigenface analysis has been evaluated on Yale and FRGC databases. The performance studies have revealed that there is no significant change in recognition performance with the increase in number of neurons above 16 for both FRGC and Yale databases. The number of sub-images giving the best recognition performance is 4 for Yale and 8 for FRGC databases. During simulation, the data is represented using 32-bit floating point number on the PC. In order to reduce the storage resources and computation complexity in hardware implementation, a performance study has been carried out to study the effects of word length

on the recognition performance. The study has revealed that a mere 8-bit fixed point representation could give good recognition performance.

In Chapter 6, a simple, linear and reconfigurable systolic array realization of the neural network based face recognition algorithm has been proposed. Signal Flow Graphs (SFGs) have been constructed for different computations involved in the various steps in training and recognition phases. A processing element with appropriate control unit, arithmetic logic unit and memory unit has been designed and implemented on a Xilinx Virtex 4 FPGA. The maximum frequency of operation achieved is 136.243 Mhz. The number of training images that can be accommodated on the FPGA is limited by the availability of BRAM blocks. The maximum training set size that could be accommodated on the target FPGA is 1024. In order to overcome this training set limit, a method to handle large training set using external RAM and run-time configuration of BRAMs on FPGA has also been proposed.

In Chapter 7, a new eigenface-based face detection method has been proposed using boosted eigenfeatures. Conventional eigenface-based face detection works by classification based on the distance between the query image and the face space. An alternative method has been proposed which uses the projections of the query images onto the eigenfaces as weak classifiers. The AdaBoost algorithm has been used to build a strong classifier from these weak classifiers. The mapping of this algorithm onto an array architecture has also been discussed. A systolic architecture for the computation of the integral image has been proposed.

The array architectures designed in this thesis are regular, easily scalable and exhibit real-time performance in processing video images for navigation and face recognition of vision-guided autonomous systems.

8.2 Future Directions

Real-time path planning requires images to be pre-processed to obtain a binary image input to with the proposed path planning architecture. Several FPGA development boards and development tools provide pre-built modules for image acquisition and pre-processing tasks. One future extension is the integration of image acquisition and pre-processing modules with the proposed path planning architecture for real applications. The added pre-processing modules will reduce the overall frame rate that can be processed. As the proposed architecture is capable of processing the frames at a very high rate, the added pre-processing overheads will not prevent the architecture from working at the normal video rate of 30 frames per second. Future work can also analyze the ideal number of frames to be processed based on the speed of the guiding system and speed of the objects moving. The protocols for data transfer between different modules also need to be developed in such a way that the function of individual modules are not affected.

Another future research is the development of path planning methodology for large indoor environment deployed with multiple overhead cameras. Efficient technique should be designed to selectively process the images from the cameras.

The proposed face recognition architecture assumes the availability of normalized images. Future work will combine the proposed face recognition architecture with the suggested face detection architecture to develop a complete face recognition system. Possibility of building a pipelined system can also be explored.

Independent Component Analysis (ICA) is an extension of PCA and it uses the set of principal components extracted using PCA. ICA is expected to outperform PCA in face recognition though it is computationally more intensive. As the architecture for extracting

eigenvectors using PCNN is already proposed, future work can extend this to develop an architecture for face recognition using ICA. Detailed analysis of the performance of modular ICA for face recognition can be done and the optimum values of various parameters like the number of independent components, ideal word length can be found out.

A suitable framework should be developed for combining and implementing both path planning and face recognition architectures in a vision-guided embedded system. As the proposed architectures are based on simple processing elements, they can be implemented on a single high-end FPGA.

List of Publications

Refereed Journals

1. N. Sudha and **A. R. Mohan**, “Design of a hardware accelerator for path planning on the Euclidean distance transform”, *Journal of Systems Architecture*, vol. 54, pp. 253–264, 2008.
2. N. Sudha and **A. R. Mohan**, “Hardware-efficient image-based robotic path planning in a dynamic environment and its FPGA implementation”, in *IEEE Transactions on Industrial Electronics*, DOI (10.1109/TIE.2010.2054051).
3. **A. R. Mohan** and N. Sudha, “FFT-based fast face detection using boosted Eigenfaces”, accepted for publication in *International Journal of Automated Identification Technology*.

Book Chapters

1. N. Sudha and **A. R. Mohan**, “Vision-based path planning with onboard VLSI array processors”, Chapter 8, *New Research on Mobile Robotics*, Editors: Ernest V. Gaines and Lawrence W. Peskov, ISBN 978-1-60456-651-2, Nova Science Publishers, USA, July 2008.
2. N. Sudha and **A. R. Mohan**, “Cellular Array VLSI Architecture for Vision-Based

Navigation of Mobile Robots”, Chapter 7, *VLSI and Computer Architecture*, Editor: Kenzo Watanabe, ISBN 978-1-60692-075-6, Nova Science Publishers, USA, 2008 (4th quarter).

Refereed Conferences

1. **A. R. Mohan** and N. Sudha, “Fast face detection using boosted eigenfaces”, in *Proceedings of IEEE International Symposium on Industrial Electronics and Applications*, Kuala Lumpur, Malaysia, Oct 2009.
2. **A. R. Mohan** and N. Sudha, “Hardware directed Fast Eigenface based face detection algorithm using FFT”, in *Proceedings of IEEE International Symposium on Industrial Electronics*, Seoul, Korea, July 2009.
3. N. Sudha and **A. R. Mohan**, “A Vision-Based Path Planning on a Distance Map and its Cellular Array VLSI Architecture”, in *Proceedings of IEEE International Symposium on Industrial Electronics*, Seoul, Korea, July 2009.
4. **A. R. Mohan**, N. Sudha and Pramod K. Meher, “An embedded face recognition system on a VLSI array architecture and its FPGA implementation”, in *Proceedings of 34th Annual Conference of the IEEE Industrial Electronics Society*, pp. 2432-2437, Orlando, USA, Nov 2008.
5. N. Sudha, **A. R. Mohan** and Pramod K. Meher, “Systolic array realization of a neural network-based face recognition system”, in *Proceedings of IEEE International Conference on Industrial Electronics and Applications*, Singapore, June 2008.

Bibliography

- [1] T. Bräunl, *Embedded Robotics: Mobile robot design and applications with embedded systems*. Springer, 2006.
- [2] S. Dubowsky, F. Genot *et al.*, “PAMM - a robotic aid to the elderly for mobility assistance and monitoring: A “helping-hand” for the elderly,” in *Proceedings of IEEE International Conference on Robotics and Automation*, 2000, pp. 570–576.
- [3] N. Roy, G. Baltus, D. Fox, and F. Gemperle, “Towards personal service robots for the elderly,” in *Proceedings of the Workshop on Interactive Robots and Entertainment*, 2000.
- [4] W. Zhao, R. Chellappa, P. J. Phillips, and A. Rosenfeld, “Face recognition: A literature survey,” *ACM Computing Surveys*, vol. 35, no. 4, pp. 399–458, 2003.
- [5] B. Adams, C. Breazeal, R. A. Brooks, and B. Scassellati, “Humanoid robots: A new kind of tool,” *IEEE Intelligent Systems*, vol. 15, no. 4, pp. 25–31, 2000.
- [6] T. Braun, K. Szentpetery, and K. Berns., “Detecting and following humans with a mobile robot,” in *Proceedings of EOS Conference on Industrial Imaging and Machine Vision*, June 2005.
- [7] G. Paya-Vay, J. Martin-Langerwerf, and P. Pirsch, “Rapanui: A case study in rapid prototyping for multiprocessor system-on-chip,” in *Proceedings of the 10th Euromicro Conference on Digital System Design Architectures, Methods and Tools*, 2007, pp. 215–221.

- [8] G. N. DeSouza and A. C. Kak, "Vision for mobile robot navigation: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, pp. 237–267, 2002.
- [9] P. Salo, R. Sim, A. Shokoufandeh, and S. Dickinson, "Landmark selection for vision-based navigation," *IEEE Transactions on Robotics*, vol. 22, no. 2, pp. 334–349, April 2006.
- [10] K. JongBae and J. HeeSung, "Vision-based location positioning using augmented reality for indoor navigation," *IEEE Transactions on Consumer Electronics*, vol. 54, no. 3, pp. 954–962, 2008.
- [11] R. Rodrigo, M. Zouqi, Z. Chen, and J. Samarabandu, "Robust and efficient feature tracking for indoor navigation," *IEEE Transactions on Systems, Man and Cybernetics, Part B*, vol. 39, no. 3, pp. 658–671, 2009.
- [12] D. Kim and R. Nevatia, "Representation and computation of the spatial environment for indoor navigation," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 1994, pp. 475–482.
- [13] D. Pomerleau and T. Jochem, "Rapidly adapting machine vision for automated vehicle steering," *IEEE Expert: Intelligent Systems and Their Applications*, vol. 11, no. 2, pp. 19–27, 1996.
- [14] K. Chen and W. Tsai, "Vision-based autonomous land vehicle guidance in outdoor road environments using combined line and road following techniques," *Journal of Robotic Systems*, vol. 14, no. 10, pp. 711–728, October 1997.

BIBLIOGRAPHY

- [15] E. D. Dickmanns, "Computer vision and highway automation," *Vehicle System Dynamics: International Journal of Vehicle Mechanics and Mobility*, vol. 31, pp. 325–343, June 1999.
- [16] D. Song, H. N. Lee, J. Yi, and A. Levandowski, "Vision-based motion planning for an autonomous motorcycle on ill-structured roads," *Autonomous Robots*, vol. 23, no. 3, pp. 197–212, 2007.
- [17] H. Mitchell, "An algorithmic approach to some problems in terrain navigation," *Artificial Intelligence*, vol. 37, pp. 171–201, December 1988.
- [18] D. T. Lee and R. L. Drysdale, "Generalization of Voronoi diagrams in the plane," *SIAM Journal of Computing*, vol. 10, pp. 73–87, 1981.
- [19] C. O'Dunlaing and C. Yap, "A retraction method for planning the motion of a disc," *Journal of Algorithms*, vol. 6, pp. 104–111, 1982.
- [20] F. Lingelbach, "Path planning for mobile manipulation using probabilistic cell decomposition," in *Proceedings of the IEEE/RSJ International Conference Intelligent Robots and Systems*, vol. 3, 2004, pp. 2807–2812.
- [21] J. Rosell and P. Iniguez, "Path planning using harmonic functions and probabilistic cell decomposition," in *Proceedings of the IEEE International Conference on Robotics and Automation*, April 2005, pp. 1803–1808.
- [22] Y. K. Hwang and N. Ahuja, "A potential field approach to path planning," *IEEE Transactions on Robotics and Automation*, vol. 8, no. 1, pp. 23–32, 1992.
- [23] J. Barraquand, B. Langlois, and J. C. Latombe, "Numerical potential field techniques for robot path planning," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 22, no. 2, March 1992.

BIBLIOGRAPHY

- [24] M. Ghosh, S. Kumar, and M. David, “An output-sensitive algorithm for computing visibility,” *SIAM Journal on Computing*, vol. 20, no. 5, pp. 888–910, 1991.
- [25] N. Ying and L. Eicher, “Real-time 3D path planning for sensor-based underwater robotics vehicles in unknown environment,” in *Proceedings of the OCEANS MTS/IEEE Conference and Exhibition*, vol. 3, September 2000, pp. 2051–2058.
- [26] T. Wan, H. Chen, and R. Earnshaw, “Real-time path planning for navigation in unknown environment,” in *Proceedings of the International Conference on Theory and Practice of Computer Graphics*, June 2003, pp. 138–145.
- [27] G. Oriolo, G. Ulivi, and M. Vendittelli, “Real-time map building and navigation for autonomous robots in unknown environments,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 28, pp. 316–333, 1999.
- [28] J. Andrade-cetto and A. Sanfeliu, “Learning of dynamic environments by a mobile robot from stereo cues,” in *Proceedings of the IEEE International Conference on Multisensor Fusion Integration Intelligent Systems, Baden-Baden*, 2001, pp. 305–310.
- [29] J. M. Ahuactzin, E.-G. Talbi, P. Bessière, and E. Mazer, “Using genetic algorithms for robot motion planning,” *LNCS 708*, pp. 84–93, 1993.
- [30] J. Janet, R. Luo, and M. Kay, “Autonomous mobile robot global motion planning and geometric beacon collection using traversability vectors,” *IEEE Transactions on Robotics and Automation*, vol. 13, no. 1, pp. 132–140, February 1997.
- [31] L. Kei, K. Sridharan, and T. Srikanthan, “Hardware-efficient schemes for logarithmic approximation and binary search with application to visibility graph construc-

- tion,” *IEEE Transactions on Industrial Electronics*, vol. 51, no. 6, pp. 1346 – 1348, December 2004.
- [32] K. Sridharan and T. Priya, “The design of a hardware accelerator for real-time complete visibility graph construction and efficient FPGA implementation,” *IEEE Transactions on Industrial Electronics*, vol. 52, no. 4, pp. 1185 – 1187, August 2005.
- [33] T. K. Priya, P. R. Kumar, and K. Sridharan, “A hardware-efficient scheme and FPGA realization for computation of single pair shortest path for a mobile automaton,” *Journal of Microprocessors and Microsystems*, vol. 30, no. 7, pp. 413–424, 2006.
- [34] P. Tzionas, A. Thanailakis, and P. Tsalides, “Collision-free path planning for a diamond-shaped robot using two-dimensional cellular automata,” *IEEE Transactions on Robotics and Automation*, vol. 13, no. 2, pp. 237 –250, April 1997.
- [35] N. Sudha and K. Sridharan, “A high-speed VLSI design and ASIC implementation for constructing Euclidean distance-based discrete Voronoi diagram,” *IEEE Transactions on Robotics and Automation*, vol. 20, no. 2, pp. 352 – 358, April 2004.
- [36] L. Vachhani and K. Sridharan, “Hardware-efficient prediction-correction-based generalized-Voronoi-diagram construction and FPGA implementation,” *IEEE Transactions on Industrial Electronics*, vol. 55, no. 4, pp. 1558 –1569, April 2008.
- [37] N. Sudha and A. R. Mohan, “Design of a hardware accelerator for path planning on the Euclidean distance transform,” *Journal of System Architectures*, vol. 54, no. 1-2, pp. 253–264, 2008.

BIBLIOGRAPHY

- [38] A. K. Jain, A. Ross, and S. Prabhakar, “An introduction to biometric recognition,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 14, pp. 4–20, 2004.
- [39] Y. Pang, Y. Yuan, and X. Li, “Gabor-based region covariance matrices for face recognition,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 18, no. 7, pp. 989–993, July 2008.
- [40] D. Ngo, A. Teoh, and A. Goh, “Biometric hash: High-confidence face recognition,” *IEEE Transactions on Circuits and Systems for Video*, vol. 16, no. 6, pp. 771–775, June 2006.
- [41] W. Zhao, R. Chellappa, P. J. Phillips, and A. Rosenfeld, “Face recognition: A literature survey,” *ACM Computing Surveys*, vol. 35, no. 4, pp. 399–458, 2003.
- [42] M. Turk and A. Pentland, “Eigenfaces for recognition,” *Journal of Cognitive Neuroscience*, vol. 3, no. 1, pp. 71–86, 1991.
- [43] J. Zhang, Y. Yan, and M. Lades, “Face recognition: Eigenface, elastic matching, and neural nets,” *Proceedings of the IEEE*, vol. 85, pp. 1422–1435, 1997.
- [44] P. S. Penev and L. Sirovich, “The global dimensionality of face space,” in *Proceedings of the Fourth IEEE International Conference on Automatic Face and Gesture Recognition*, 2000, pp. 264–270.
- [45] J. Yang, D. Zhang, A. Frangi, and J.-y. Yang, “Two-dimensional PCA: A new approach to appearance-based face representation and recognition,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 1, pp. 131–137, January 2004.

BIBLIOGRAPHY

- [46] B. Moghaddam, W. Wahid, and A. Pentland, “Beyond eigenfaces: Probabilistic matching for face recognition,” in *Proceedings of the 3rd. International Conference on Face & Gesture Recognition*. Washington, DC, USA: IEEE Computer Society, 1998, pp. 30–35.
- [47] P. N. Belhumeur, J. P. Hespanha, and D. J. Kriegman, “Eigenfaces vs. fisherfaces: Recognition using class specific linear projection,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 7, pp. 711–720, 1997.
- [48] P. J. Phillips, “Support vector machines applied to face recognition,” in *Proceedings of the conference on Advances in neural information processing systems II*. Cambridge, MA, USA: MIT Press, 1999, pp. 803–809.
- [49] G. Guo, S. Z. Li, and K. Chan, “Face recognition by support vector machines,” in *Proceedings of the Fourth IEEE International Conference on Automatic Face and Gesture Recognition*. Washington, DC, USA: IEEE Computer Society, 2000, p. 196.
- [50] S. Z. Li and J. Lu, “Face recognition using the nearest feature line method,” *IEEE Transactions on Neural Networks*, vol. 10, pp. 439–443, 1999.
- [51] B. Manjunath, R. Chellappa, and C. Von der Malsburg, “A feature based approach to face recognition,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, June 1992, pp. 373 –378.
- [52] I. J. Cox, J. Ghosn, and P. N. Yianilos, “Feature-based face recognition using mixture-distance,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1996, pp. 209–216.

BIBLIOGRAPHY

- [53] L. Wiskott, J.-M. Fellous, N. Krüger, and C. von der Malsburg, “Face recognition by elastic bunch graph matching,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 7, pp. 775–779, 1997.
- [54] A. Nefian and M. Hayes, “Hidden markov models for face recognition,” in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, vol. 5, May 1998, pp. 2721–2724.
- [55] M. Bicego, U. Castellani, and V. Murino, “Using hidden markov models and wavelets for face recognition,” in *Proceedings of the 12th International Conference on Image Analysis and Processing*. Washington, DC, USA: IEEE Computer Society, 2003, p. 52.
- [56] M.-S. Kim, D. Kim, and S.-Y. Lee, “Face recognition using the embedded hmm with second-order block-specific observations,” *Pattern Recognition*, vol. 36, pp. 2723–2735, 2003.
- [57] A. Pentland, B. Moghaddam, and T. Starner, “View-based and modular eigenspaces for face recognition,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, June 1994, pp. 84–91.
- [58] P. S. Penev and J. J. Atick, “Local feature analysis: a general statistical theory for object representation,” *Network: Computation in Neural Systems*, vol. 7, pp. 477–500, August 1996.
- [59] A. Lanitis, C. Taylor, and T. F. Cootes, “An automatic face identification system using flexible appearance models,” *Image and Vision Computing*, vol. 13, pp. 393–401, 1995.

- [60] B. Weyrauch, B. Heisele, J. Huang, and V. Blanz, "Component-based face recognition with 3D morphable models," in *Proceedings of the 2004 Conference on Computer Vision and Pattern Recognition Workshop*, vol. 5. Washington, DC, USA: IEEE Computer Society, 2004, p. 85.
- [61] A. Martinez and A. Kak, "PCA versus LDA," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 23, no. 2, pp. 228–233, February 2001.
- [62] M. Sharkas and M. Elenien, "Eigenfaces vs Fisherface vs ICA for face recognition: A comparative study," *Proceedings of IEEE International Conference on Signal Processing*, pp. 914–919, 2008.
- [63] J. Liu and S. Chen, "Discriminant common vectors versus neighbourhood components analysis and laplacianfaces: A comparative study in small sample size problem," *Image and Vision Computing*, vol. 24, pp. 249–262, 2006.
- [64] M.-H. Yang, D. J. Kriegman, and N. Ahuja, "Detecting faces in images: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 1, pp. 34–58, 2002.
- [65] A. W. Senior, "Face and feature finding for a face recognition system," in *Proceedings of the Second International Conference on Audio and Video-based Biometric Person Authentication*, 1999, pp. 154–159.
- [66] T. S. Jebara and A. Pentland, "Parametrized structure from motion for 3D adaptive feedback tracking of faces," in *Proceedings of the International Conference on Computer Vision and Pattern Recognition*, 1997, pp. 144–150.

BIBLIOGRAPHY

- [67] C. Kervrann, F. Davoine *et al.*, “Generalized likelihood ratio-based face detection and extraction of mouth features,” *Pattern Recognition Letters*, vol. 18, pp. 899–912, 1997.
- [68] G. Burel and D. Carel, “Detection and localization of faces on digital images,” *Pattern Recognition Letters*, vol. 15, pp. 963–967, October 1994.
- [69] P. Jueli and R. Marsh, “A hierarchical neural network for human face detection,” *Pattern Recognition*, vol. 29, pp. 781–787, May 1996.
- [70] M. Propp and A. Samal, “Artificial neural network architecture for human face detection,” *Intelligent Engineering Systems through Artificial Neural Networks*, vol. 2, pp. 535–540, November 1992.
- [71] H. A. Rowley, S. Baluja, and T. Kanade, “Neural network-based face detection,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 1, pp. 23–38, 1998.
- [72] S. Marchand-Millet and B. Merialdo, “Pseudo two-dimensional Hidden Markov Models for face detection in colour images,” in *Proceedings of the Second International Conference on Audio and Video-based Biometric Person Authentication*, 1999, pp. 13–18.
- [73] L. Meng, T. Nguyen, and D. Castanon, “An image-based bayesian framework for face detection,” in *Proceedings of the International Conference on Computer Vision and Pattern Recognition*, 2000, pp. 302–307.
- [74] P. Viola and M. Jones, “Rapid object detection using a boosted cascade of simple features,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2001, pp. 511–518.

BIBLIOGRAPHY

- [75] R. Lienhart and J. Maydt, “An extended set of haar-like features for rapid object detection,” in *Proceedings of the International Conference on Image Processing*, 2002, pp. 900–903.
- [76] R. Lienhart, A. Kuranov, and V. Pisarevsky, “Empirical analysis of detection cascades of boosted classifiers for rapid object detection,” *Pattern Recognition*, vol. 2781, pp. 297–304, 2003.
- [77] J. Gilbert and W. Yang, “A real-time face recognition system using custom VLSI hardware,” in *Proceedings of the IEEE Workshop on Computer Architectures for Machine Perception*, December 1993, pp. 58–66.
- [78] R. Gottumukkal, H. Ngo, and V. Asari, “Multi-lane architecture for eigenface based real-time face recognition,” *Journal of Microprocessors and Microsystems*, vol. 30, no. 4, pp. 216–224, June 2006.
- [79] A. Pavan Kumar, V. Kamakoti, and S. Das, “System-on-programmable-chip implementation for on-line face recognition,” *Pattern Recognition Letters*, vol. 28, no. 3, pp. 342–349, 2007.
- [80] T. Theocharides, N. Vijaykrishnan, and M. J. Irwin, “A parallel architecture for hardware face detection,” in *IEEE Annual Symposium on VLSI: Emerging VLSI Technologies and Architectures*, 2006, pp. 452–453.
- [81] T. Theocharides, G. Link, N. Vijaykrishnan, M. J. Irwin, and W. Wolf, “Embedded hardware face detection,” in *Proceedings of the 17th International Conference on VLSI Design*, 2004, pp. 133–138.
- [82] V. Kianzad, S. Saha *et al.*, “An architectural level design methodology for embedded face detection,” in *Proceedings of the 3rd IEEE/ACM/IFIP International*

BIBLIOGRAPHY

- Conference on Hardware/Software Codesign and System Synthesis*. New York, NY, USA: ACM, 2005, pp. 136–141.
- [83] S. M. Bhandarkar and H. R. Arabnia, “Parallel computer vision on a reconfigurable multiprocessor network,” *IEEE Transactions on Parallel Distributed Systems*, vol. 8, no. 3, pp. 292–309, 1997.
- [84] H. T. Kung, “Why systolic architectures?” *Computer*, vol. 15, no. 1, pp. 37–46, 1982.
- [85] A. V. Kulkarni and D. W. L. Yen, “Systolic processing and an implementation for signal and image processing,” *IEEE Transactions on Computers*, vol. 31, no. 10, pp. 1000–1009, 1982.
- [86] M. Annaratone, E. Arnould, T. Gross, H. T. Kung, and M. Lam, “The warp computer: Architecture, implementation, and performance,” *IEEE Transactions on Computers*, vol. 36, no. 12, pp. 1523–1538, 1987.
- [87] H. W. Printz, “Automatic mapping of large signal processing systems to a parallel machine,” Ph.D. dissertation, Pittsburgh, PA, USA, 1991.
- [88] R. Means, “A new two-dimensional systolic array for image processing and neural network applications,” in *Proceedings of the Seattle International Joint Conference on Neural Networks*, vol. 2, July 1991, p. 925.
- [89] M. Patel, P. McCabe, and N. Ranganathan, “SIBA: A VLSI systolic array chip for image processing,” in *Proceedings of the 11th IAPR International Conference on Architectures for Vision and Pattern Recognition*, August-September 1992, pp. 15–18.

BIBLIOGRAPHY

- [90] F. Serratos, P. Millan, and E. Montseny, "Systolic processors applied to computer vision systems," in *Proceedings of the International Conference on Computer Architectures for Machine Perception*, September 1995, pp. 178 –183.
- [91] N. Ranganathan and K. Doreswamy, "A systolic algorithm and architecture for image thinning," in *Proceedings of the Fifth Great Lakes Symposium on VLSI*, March 1995, pp. 138 –143.
- [92] N. Dunstan and P. M. Lenders, "Systolic arrays for 2 – D decimation filters," in *Proceedings of the Second International conference on Algorithms and Architectures for Parallel Processing*, June 1996, pp. 14–20.
- [93] N. Ranganathan, R. Mehrotra, and S. Subramanian, "A high speed systolic architecture for labeling connected components in an image," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 25, no. 3, pp. 415 –423, March 1995.
- [94] N. K. Ratha and A. Jain, "Computer vision algorithms on reconfigurable logic arrays," *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, no. 1, January 1999.
- [95] G. Iyengar and S. Panchanathan, "Systolic array architecture for gabor decomposition," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 5, no. 4, pp. 355 –359, August 1995.
- [96] O. Deforges and N. Normand, "A generic systolic processor for real time grayscale morphology," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 6, 2000, pp. 3331 –3334.

BIBLIOGRAPHY

- [97] G. Saldana and M. Arias-Estrada, “FPGA-based customizable systolic architecture for image processing applications,” in *Proceedings of the International Conference on Reconfigurable Computing and FPGAs*, September 2005, pp. 3 – 8.
- [98] P. Meher, “Systolic designs for DCT using a low-complexity concurrent convolutional formulation,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 16, no. 9, pp. 1041–1050, September 2006.
- [99] G. Saldana and M. Arias-Estrada, “A low-level image processing algorithms accelerator platform,” in *Proceedings of the 18th International Conference on Electronics, Communications and Computers*, March 2008, pp. 117–122.
- [100] P. P. Chaudhuri, D. Chowdhury, S. Nandi, and S. Chattopadhyay, *Additive Cellular Automata - Theory and Applications*. Wiley-IEEE Computer Society Press, 1997.
- [101] S. Wolfram, “Statistical mechanics of cellular automata,” *Reviews of Modern Physics*, vol. 55, pp. 601–644, 1983.
- [102] M. Sipper, “The emergence of cellular computing,” *Computer*, vol. 32, no. 7, pp. 18–26, 1999.
- [103] T. Ikenaga and T. Ogura, “*CAM*²: A highly-parallel two-dimensional cellular automaton architecture,” *IEEE Transactions on Computers*, vol. 47, pp. 788–801, 1998.
- [104] N. Sudha, S. Nandi, and K. Sridharan, “Cellular architecture for Euclidean distance transformation,” *IEE Proceedings - Computers and Digital Techniques*, vol. 147, pp. 335–342, September 2000.
- [105] S. Umbaugh, *Computer Vision and Image Processing: A Practical Approach using CVIPTools*. Prentice Hall, 1998.

BIBLIOGRAPHY

- [106] S.Palnitkar, *Verilog HDL: A Guide to Digital Design and Synthesis*. Prentice Hall, 2003.
- [107] L. Kei, K.Sridharan, and T.Srikanthan, “Hardware-efficient schemes for logarithmic approximation and binary search with application to visibility graph construction,” *IEEE Transactions on Industrial Electronics*, vol. 51, no. 6, pp. 1346–1348, December 2004.
- [108] K.Sridharan and T. K. Priya, “The design of a hardware accelerator for real-time complete visibility graph construction and efficient FPGA implementation,” *IEEE Transactions on Industrial Electronics*, vol. 52, pp. 1185–1187, August 2005.
- [109] N.Sudha and K.Sridharan, “A high-speed VLSI design and ASIC implementation for constructing Euclidean distance-based discrete Voronoi diagram,” *IEEE Transactions on Robotics and Automation*, vol. 20, no. 2, pp. 352–358, 2004.
- [110] L. Vachhani and K. Sridharan, “Hardware-efficient prediction-correction-based generalized-Voronoi-diagram construction and FPGA implementation,” *IEEE Transactions on Industrial Electronics*, vol. 55, no. 4, pp. 1558–1569, April 2008.
- [111] T.K.Priya, P.R.Kumar, and K.Sridharan, “A hardware-efficient scheme and FPGA realization for computation of single pair shortest path for a mobile automaton,” *Journal of Microprocessors and Microsystems*, vol. 30, pp. 413–424, 2006.
- [112] N.Sudha and A.R.Mohan, “Design of a hardware accelerator for path planning on the Euclidean distance transform,” *Journal of Systems Architecture*, vol. 54, pp. 253–264, 2008.

BIBLIOGRAPHY

- [113] S. Lawrence, C. Giles, A. Tsoi, and A. D. Back, "Face recognition: A convolutional neural-network approach," *IEEE Transactions on Neural Networks*, vol. 8, no. 1, pp. 98–111, January 1997.
- [114] M. Er, S. Wu, J. Lu, and H. Toh, "Face recognition with Radial Basis Function (RBF) neural networks," *IEEE Transactions on Neural Networks*, vol. 13, no. 3, pp. 697–710, May 2002.
- [115] J. Lu, X. Yuan, and T. Yahagi, "A method of face recognition based on fuzzy C-means clustering and associated sub-NNs," *IEEE Transactions on Neural Networks*, vol. 18, no. 1, pp. 150–160, January 2007.
- [116] P. Melin, C. Felix, and O. Castillo, "Face recognition using modular neural networks and the fuzzy sugeno integral for response integration," *International Journal of Intelligent Systems*, vol. 20, no. 2, pp. 275–291, February 2005.
- [117] Z. Zhao, D. Huang, and B. Sun, "Human face recognition based on multi-features using neural networks committee," *Pattern Recognition Letters*, vol. 25, no. 12, pp. 1351–1358, September 2004.
- [118] K. I. Diamantaras and S. Y. Kung, *Principal component neural networks: Theory and applications*. John Wiley and Sons, New York, 1996.
- [119] S. Haykins, *Neural Networks: A comprehensive foundation*. Prentice Hall, 1999.
- [120] T. D. Sanger, "Optimal unsupervised learning in a single layer feedforward neural network," *Neural Networks*, vol. 2, pp. 459–73, 1989.
- [121] R. Gottumukkal and V. K. Asari, "An improved face recognition technique based on modular PCA approach," *Pattern Recognition Letters*, vol. 25, no. 4, pp. 429–436, March 2004.

BIBLIOGRAPHY

- [122] *Yale database*. [Online]. Available: <http://cvc.yale.edu/projects/yalefaces/yalefaces.html>
- [123] P. J. Phillips, P. J. Flynn *et al.*, “Overview of the face recognition grand challenge,” in *Proceedings of the 2005 IEEE Conference on Computer Vision and Pattern Recognition*, vol. 1, 2005, pp. 947–954.
- [124] “FRGC website.” [Online]. Available: <http://face.nist.gov/frgc>
- [125] “MIT-CBCL Face Database,” <http://www.ai.mit.edu/projects/cbcl>.
- [126] P. K. Meher, J. C. Patra, and A. P. Vinod, “Efficient systolic designs for 1- and 2-dimensional DFT of general transform-lengths for high-speed wireless communication applications,” *Journal of Signal Procesing Systems*, vol. 60, no. 1, pp. 1–14, 2010.
- [127] C. Cheng and K. Parhi, “High-throughput VLSI architecture for FFT computation,” *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 54, no. 10, pp. 863–867, October 2007.

APPENDIX A

ML403 FPGA Evaluation Platform

This appendix describes the ML403 evaluation platform and Xilinx Embedded Development Kit used to implement and verify the hardware architectures developed in this thesis.

A.1 ML403 Board

ML403 is packed with powerful features like Virtex-4 FPGA, 64-MB DDR SDRAM with 32-bit interface running up to 266 MHz data rate, Stereo AC97 audio codec, RS-232 serial port, 16 character $\times$ 2-line LCD, VGA out, PS/2 mouse and keyboard connectors, system ACE compact flash, Digital Clock Manager (DCM), ethernet Media Access Controller (MAC) etc. A snapshot of the ML403 board is shown in Fig.A.1 and its block diagram is shown in Fig.A.2.

A.1.1 Virtex-4 FPGA

ML403 has a Virtex-4 FX series FPGA XC4VFX12, optimized for high performance embedded platform applications. The FPGA is packed with powerful 500 MHz XtremeDSP slices with 18×18 multiplier and MAC. It also contains Integrated Block Memory with

Figure A.1: ML403 snapshot

Figure A.2: ML403 block diagram

Component	Resources
Logic Cells	12,312
Slices	5,472
Max Distributed RAM(Kb)	86
XtremeDSP Slices	32
DCMs	4
PowerPC Blocks	1
Ethernet MACs	2
Total I/O Banks	9
Max User I/O	320

Table A.1: Virtex-4 XC4VFX12 resources

dual-port architecture, independent read and write port width selection and byte-write capability. The resources available in XC4VFX12 FPGA are listed in Table A.1.

A.1.2 DDR SDRAM

The board contains 64MB of DDR SDRAM using two Infineon chips. Each chip is 16-bit wide and together form a 32-bit data bus capable of operating up to 266 MHz. The board can support up to 256MB of total DDR SDRAM memory and an extra pin is present to extend the memory support up to 1-Gb DDR chips. DDR SDRAM interface gives higher data rates using *double pumping* technique in which data is transferred at both rising and falling edges.

A.1.3 RS-232 Serial Port and LCD

ML403 provides one male DB-9 RS-232 serial port for communication between the FPGA and another serial device. The serial port on the FPGA is wired as a host device and hence a null modem cable is required to connect with the serial port on a PC. The serial port can operate at a maximum baud rate of 115200 Bd. The voltage level

between the FPGA and RS-232 signals is shifted using an interface chip.

The ML403 evaluation platform also provides a 16-character $\times$ 2-line LCD on the board to display text information to the user. The data interface connected to the FPGA supports 4-bit mode. Voltage level between the LCD and the FPGA is shifted using a level translator chip.

A.2 Xilinx Embedded Development Kit

The Embedded Development Kit (EDK) is an integrated design suite of tools and Intellectual Property (IP) modules for designing a complete embedded system including an embedded processor (like PowerPC) for implementation on a Xilinx FPGA device. It consists of Xilinx Platform Studio (XPS), Xilinx Integrated Software Environment (ISE) and Software Development Kit (SDK). XPS is a development environment for designing the hardware portion of the embedded system. Xilinx ISE is used to develop and verify custom hardware modules and generate netlist for the same. Xilinx SDK is an integrated environment used to develop C/C++ embedded software application to be run on either PowerPC or MicroBlaze processors inside the FPGA.

A.2.1 Base System Builder

Base System Builder is used to build a system targeted to a specific platform like ML403. Processor type and frequency, peripherals like GPIO, SDRAM and UART, bus connections using Processor Local Bus (PLB) or On-chip Peripheral Bus (OPB) are specified in the wizard and it generates the basic system according to the parameters given. Xilinx EDK automatically creates a Microprocessor Hardware Specification (MHS) and a

Microprocessor Software Specification (MSS) files. Definitions for bus architecture, peripherals, processor, system connectivity and address space are present in the MHS file. MSS file contains the specifications of drivers for the hardware modules and peripherals instantiated in the MHS file. The wizard also generates the default *bootloop* application to be run on the embedded processor when it boots up first. Additional software projects can be created and initialized to be run when the system powers up.

A.2.2 Serial Data Communication from PC to FPGA

For the verification and demonstration of the architectures described in this thesis, data is usually sent serially from a PC to the ML403 evaluation platform for processing. A receiver to receive the serial data and reassemble it into bytes to be used by the co-processor is need to be present in the FPGA. Xilinx EDK provides a soft IP core of UARTLite that connects to the PLB and provides the necessary control interface for asynchronous serial data transfer. The maximum baud rate supported is 115200 bps. UARTLite core contains a memory module that consists of memory mapped registers interfacing through PLB interface module. The memory mapped registers include an 8-bit status register, an 8-bit control register and a pair of 8-bit Transmit/Receive FIFOs as shown in Fig.A.3. The drivers for communicating with this core are automatically generated by Xilinx Platform Studio and are found in the files *xuartlite.h*, *xuartlite_1.h* and *xuartlite_i.h*.

A.2.3 Configuring the LCD

ML403 evaluation board contains a 16 character $\times$ 2 line LCD. The LCD driver used on the ML403 board is a Samsung S6A0069 dot matrix LCD driver and controller LSI device. It can display 1 or 2 lines with a 5×8 or a 5×11 dots matrix. It can be set to

Figure A.3: Block diagram of XPS UARTLite

Figure A.4: LCD wiring on the ML403 board

use an 8 bit or 4 bit data bus. On the ML403 board the bus is 4 bits wide. The signal wirings of the LCD to the FPGA are shown in Fig.A.4. XPS provides General Purpose I/O (GPIO) interface for connecting through the external pins. As there are seven signals connected to the LCD, the width of GPIO is chosen to be 7. The signals, their description and the location on the FPGA are enumerated in Table A.2.

Signal Name	Description	GPIO pin	FPGA Pin Location
LCD_E	Read/Write Enable Pulse 0: Disabled 1: Read/Write operation enabled	0	AE13
LCD_RS	Register Select 0: Instruction register during write 1: Data for read/write operation	1	AC17
LCD_RW	Read/Write Control 0: Write, LCD reads data 1: Read, LCD writes data	2	AB17
LCD_DB7	Data Bus bit 7	3	AF12
LCD_DB6	Data Bus bit 6	4	AE12
LCD_DB5	Data Bus bit 5	5	AC10
LCD_DB4	Data Bus bit 4	6	AB10

Table A.2: LCD signal wiring on the ML403 board

A.2.4 Implementing the custom hardware as a peripheral

The custom hardware architecture designed is added to the embedded platform as a peripheral. For the custom hardware to work in compliance with the system, it has to adhere to a standard interface. Therefore the custom hardware module developed is wrapped using an interface module to connect with the PLB to interact with the rest of the system. The functionality of the core is designed and verified separately before importing it into the EDK. Xilinx Platform Studio generates the appropriate platform specification format and interface files while importing the custom hardware so that the rest of the tool-chain can recognize it as a peripheral. Imported peripheral is then connected to the appropriate PLB and the tool generates the address range for this peripheral. The Verilog source code along with the platform specifications is then synthesized to produce the netlist as shown in Fig.A.5.

Figure A.5: Hardware netlist generation flow

A.2.5 Development of Control Software

After the hardware design phase, appropriate software has to be developed for controlling all the peripherals. For the purpose of the designs described in this thesis, embedded software is used to transfer data from the PC and to control peripherals like LCD. The embedded application development flow is shown in Fig.A.6. Using the drivers generated before, software code is written in C to transfer data from the PC and display the results on the LCD.

A.2.6 Implementing the design and configuring the FPGA

The final implementation involves generating the bitstream (.bit) files for programming the FPGA from netlist (.NGC) and User Constraints File (.UCF). The design constraints like pin locations for all the external ports, clock pins and timing constraints are specified

Figure A.6: Embedded application development flow

in the UCF. The NGC files are processed, along with the system constraints, through Xilinx tools (NGDBuild, MAP, PAR, and TRACE) when XPS generates the bitstream.

Virtex-4 devices are configured by loading application-specific configuration data i.e., the bitstream into internal memory. Because Xilinx FPGA configuration memory is volatile, it must be configured each time it is powered-up. The bitstream is loaded into the device through special configuration pins. Software can be downloaded either separately through Xilinx SDK or along with the bitstream. *Platform Cable USB* is used to download the configuration data.