1999

In-camera defect detection with applications to Web inspection systems.

Sayed Hossain. Hajimowlana

University of Windsor

Follow this and additional works at: https://scholar.uwindsor.ca/etd

Recommended Citation


https://scholar.uwindsor.ca/etd/2417
INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

UMI®
In-Camera Defect Detection with Applications to Web Inspection Systems

by

Hossain Hajimowlna

A Dissertation
Submitted to the College of Graduate Studies and Research through the Electrical Engineering in partial fulfillment of the requirements for the Degree of Doctor of Philosophy at the University of Windsor

Windsor, Ontario, Canada
September 1999
The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author’s permission.

L’auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L’auteur conserve la propriété du droit d’auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-62320-3
© 1999 Hossain Hajimowlana

All Rights Reserved. No part of this document may be reproduced, stored or otherwise retained in a retrieval system or transmitted in any form, on any medium or by any means without the prior written permission of the author.
Abstract

One of the aims of industrial machine vision is to develop computer and electronic systems to replace human vision in quality control of industrial production. Traditionally these systems consist of a line scan camera, host computer, frame grabber and one or more dedicated processing boards. The work reported in this thesis develops defect detection algorithms for real-time processing of the camera video stream. The processing system is mounted inside the camera and provides sufficient defect detection capabilities to eliminate the need for an external frame grabber and other associated host computer peripheral systems. The system is targeted for web inspection but has the potential for broader application areas.

The output data from the camera is reduced by many orders of magnitude by only transmitting the “interesting” pixels of the image to be processed, and this can significantly reduce both the downstream processing hardware required and the bandwidth of the digital data received from the camera. The use of such special purpose cameras has the potential not only to improve the performance of machine vision systems for a wide variety of applications, but to improve the economic viability of these applications through reductions in hardware cost and complexity.

This real-time system must perform all of the required operations at the video bandwidth of the camera, and the work reported in this thesis uses hardware associated with the in-camera processing system, developed in the VLSI Laboratory at the University of Windsor, which includes programmable logic (Field Programmable Gate Array) directly connected to the video stream, and ancillary signal processing and control hardware (a DSP chip). These hardware limitations apply constraints to the algorithms, and we are almost always unable to use traditional image processing algorithms; rather we choose and develop algorithms based on their potential for identification based on minimal storage of
a pixel-serial raster data.

In this thesis we report the following novel developments:

1. For non-textured background materials, three algorithms have been developed for the in-camera system: two (or multi) level thresholding; zero order background tracking; and delta modulation background tracking.

2. Auto-regressive techniques have been developed and implemented as a statistical approach to analyze textured backgrounds and to identify possible defects. This method of analysis has been extensively used to study visual textures. In the simplest form, the image is scanned to provide a one-dimensional series of gray level fluctuations, which is treated as a one-dimensional stochastic process evolving in “time”. In a more comprehensive form, a pixel value is assumed to depend upon a certain part of its neighborhood. The coefficients of dependence are extracted using time series analysis techniques.

3. A novel algorithm for defect detect detection based on fuzzy fusion of texture features is developed, simulated and successfully implemented on the experimental test setup. Conventional approaches for web defect detection involve making “crisp” decisions for image analysis and recognition where imprecise or incomplete specifications are usually either ignored or discarded. The fuzzy logic algorithm uses imprecise or ambiguous image data caused by instrumental error or environmental noise such as dust or small variations in illumination to obtain a precise result. The developed algorithm can be applied to both textured and non textured materials and offers superior performance over traditional template matching methods.
Acknowledgments

There are several people who deserve my sincere thanks for their generous contributions to this project.

First I would like to express my sincere thanks and appreciation to my supervisor, Dr. G. A. Jullien for his invaluable advice, guidance and constant encouragement throughout the progress of this thesis. I would also like to thank my committee members Dr. M. Ahmadi, Dr. J.J Soltis and Dr. R. Lashkari for their time and advice.

I must recognize the invaluable contributions of Mr. Roberto Muscedere who developed the hardware system for this project and without him, the completion of this research would not have become a reality.

I would finally like to thank Dr. Saeid Sadeghi Emamchaie for the many fruitful discussions that we had on this project.
# Table of Contents

Chapter 1  Introduction ............................................................................................................. 1

1.1  Introduction ...................................................................................................................... 1
1.2  Machine Inspection Versus Human Inspection .............................................................. 2
1.3  One Step Inspection Versus Multi Inspection Process ...................................................... 3
1.4  Challenges in Vision Systems .......................................................................................... 4
1.5  Research Objectives and Review ..................................................................................... 6
1.6  Thesis Organization ............................................................................................................ 7

Chapter 2  Inspection Systems, Introduction and Definitions ................................................. 9

2.1  Introduction ...................................................................................................................... 9
2.2  Web Inspection ................................................................................................................ 9
2.3  Defect - Definition, Detection ....................................................................................... 10
2.4  Real Time Issue .............................................................................................................. 11
2.5  Vision Systems' Sensors .................................................................................................. 11
   Linear array sensors ........................................................................................................ 12
   Cameras ........................................................................................................................... 12
   Camera transfer function and noise .............................................................................. 13
   Resolution ....................................................................................................................... 13
2.6  Illumination ..................................................................................................................... 14
2.7  Texture Definition or Image Modeling ............................................................................ 14
   Statistical approach: ...................................................................................................... 15
   Structural approach ....................................................................................................... 16
   Defect detection on textured images: .......................................................................... 16
2.8  Algorithms Employed for Defect Detection .................................................................. 17
   Geometry-driven (template matching) inspection ....................................................... 17
   Morphological analysis: ............................................................................................... 19
   Fuzzy logic .................................................................................................................... 20
2.9  Commercial systems ....................................................................................................... 20
2.10 Hardware Issues ............................................................................................................. 21
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.11</td>
<td>Software Environment</td>
<td>22</td>
</tr>
<tr>
<td>2.12</td>
<td>Conclusion</td>
<td>22</td>
</tr>
<tr>
<td><strong>Chapter 3</strong></td>
<td><strong>System Overview</strong></td>
<td></td>
</tr>
<tr>
<td>3.1</td>
<td>Introduction</td>
<td>24</td>
</tr>
<tr>
<td>3.2</td>
<td>Test setup</td>
<td>24</td>
</tr>
<tr>
<td>3.2.1</td>
<td>DALSA TDI Line Scan Camera</td>
<td>26</td>
</tr>
<tr>
<td>3.2.2</td>
<td>CCD and TDI technology</td>
<td>26</td>
</tr>
<tr>
<td>3.2.3</td>
<td>TIME-DELAY and INTEGRATION (TDI)</td>
<td>27</td>
</tr>
<tr>
<td>3.2.4</td>
<td>Array Architecture</td>
<td>28</td>
</tr>
<tr>
<td>3.2.5</td>
<td>CMOS Sensors Versus CCDs</td>
<td>29</td>
</tr>
<tr>
<td>3.3</td>
<td>Processing Board - Hardware</td>
<td>32</td>
</tr>
<tr>
<td>3.3.1</td>
<td>New Processing Hardware</td>
<td>33</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Firewire Technology</td>
<td>34</td>
</tr>
<tr>
<td>3.4</td>
<td>Lighting and Image Acquisition</td>
<td>34</td>
</tr>
<tr>
<td>3.4.1</td>
<td>Illumination used in the Test Setup</td>
<td>38</td>
</tr>
<tr>
<td>3.5</td>
<td>Conclusion</td>
<td>39</td>
</tr>
<tr>
<td><strong>Chapter 4</strong></td>
<td><strong>Thresholding Techniques</strong></td>
<td></td>
</tr>
<tr>
<td>4.1</td>
<td>Introduction</td>
<td>40</td>
</tr>
<tr>
<td>4.2</td>
<td>Thresholding</td>
<td>41</td>
</tr>
<tr>
<td>4.3</td>
<td>Adaptive Thresholding</td>
<td>42</td>
</tr>
<tr>
<td>4.4</td>
<td>Two (and multi) level thresholding</td>
<td>43</td>
</tr>
<tr>
<td>4.5</td>
<td>Zero-order Background Tracking</td>
<td>45</td>
</tr>
<tr>
<td>4.6</td>
<td>False alarm rejection by trigger association</td>
<td>46</td>
</tr>
<tr>
<td>4.7</td>
<td>Performance of thresholding techniques in the presence of noise</td>
<td>50</td>
</tr>
<tr>
<td>4.8</td>
<td>Two-threshold detector</td>
<td>50</td>
</tr>
<tr>
<td>4.9</td>
<td>Excess over thresholds filter</td>
<td>51</td>
</tr>
<tr>
<td>4.10</td>
<td>Delta modulation background tracking</td>
<td>56</td>
</tr>
<tr>
<td>4.11</td>
<td>Conclusions</td>
<td>58</td>
</tr>
<tr>
<td><strong>Chapter 5</strong></td>
<td><strong>Time Series Analysis of Textures</strong></td>
<td></td>
</tr>
<tr>
<td>5.1</td>
<td>Introduction</td>
<td>59</td>
</tr>
<tr>
<td>5.2</td>
<td>Theory and definitions</td>
<td>60</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Rational transfer function modeling methods:</td>
<td>60</td>
</tr>
<tr>
<td>5.2.2</td>
<td>The AR modeling - linear prediction</td>
<td>62</td>
</tr>
<tr>
<td>5.2.3</td>
<td>Model Order Selection</td>
<td>64</td>
</tr>
<tr>
<td>5.3</td>
<td>Implementation of the Algorithm</td>
<td>65</td>
</tr>
<tr>
<td>5.3.1</td>
<td>The 1-D AR algorithm</td>
<td>68</td>
</tr>
<tr>
<td>5.3.2</td>
<td>Simulation experiments</td>
<td>68</td>
</tr>
<tr>
<td>5.3.3</td>
<td>Performance of the algorithm in a noisy environment</td>
<td>73</td>
</tr>
</tbody>
</table>
Chapter 6  Fuzzy Logic in Defect Detection Systems.................................79

6.1  Introduction..............................................................................79
6.2  An Introduction to Fuzzy Modeling .........................................80
Set Theoretic Operations: .................................................................84
6.3  Fuzzy Logic System: A High-Level Introduction .......................86
Mamdani and Sugeno's inference engines: ......................................89
Some Terminology on Rules: .........................................................90
6.4  Adaptive Neuro Fuzzy Inference System (ANFIS) ....................91
What Is ANFIS? ...........................................................................92
FIS Structure and Parameter Adjustment .......................................92
6.5  Fuzzy logic and image processing .............................................93
Application of fuzzy logic for defect detection ..............................96
Theory .......................................................................................99
Development of a fuzzy logic defect detection system in brief ........101
6.6  Simulation Results ..................................................................102
6.7  Hardware Implementation .......................................................115
6.8  Fuzzy Logic and Neural Networks ...........................................116
6.9  Conclusion .............................................................................117

Chapter 7  Conclusions.....................................................................119

7.1  Summary and Contributions .....................................................119
7.2  Suggestions for Future Work ...................................................121

REFERENCES.................................................................................125
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>(a) One step Inspection versus (b) continuous product inspection</td>
<td>4</td>
</tr>
<tr>
<td>2.1</td>
<td>Principle of the expansion - contraction method for detecting small defects</td>
<td>19</td>
</tr>
<tr>
<td>3.1</td>
<td>Test setup for production line simulation</td>
<td>25</td>
</tr>
<tr>
<td>3.2</td>
<td>DALSA TDI line scan camera</td>
<td>26</td>
</tr>
<tr>
<td>3.3</td>
<td>TDI concept. At time T1, the image is focused onto the first element. At</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td>T2 the image has moved to the next detector. Simultaneously, the charge</td>
<td></td>
</tr>
<tr>
<td></td>
<td>packet is clocked down to the next pixel site. After four stages, the signal</td>
<td></td>
</tr>
<tr>
<td></td>
<td>increases fourfold.</td>
<td></td>
</tr>
<tr>
<td>3.4</td>
<td>Block diagram of the overall system</td>
<td>32</td>
</tr>
<tr>
<td>3.5</td>
<td>Prototype (Larch) processing board</td>
<td>33</td>
</tr>
<tr>
<td>3.6</td>
<td>Block diagram of the new processing board</td>
<td>34</td>
</tr>
<tr>
<td>3.7</td>
<td>New Larch/Firewire system mounted on the DALSA camera</td>
<td>35</td>
</tr>
<tr>
<td>3.8</td>
<td>A scanned image from the test setup showing the effect of nonuniform</td>
<td>38</td>
</tr>
<tr>
<td></td>
<td>lighting</td>
<td></td>
</tr>
<tr>
<td>3.9</td>
<td>Foster optical light source and line light</td>
<td>38</td>
</tr>
<tr>
<td>4.1</td>
<td>Flowchart for symmetric 2-level thresholding</td>
<td>43</td>
</tr>
<tr>
<td>4.2</td>
<td>Effect of fixed level thresholding on defect detection by ignoring low</td>
<td>44</td>
</tr>
<tr>
<td></td>
<td>intensity parts of the defect</td>
<td></td>
</tr>
<tr>
<td>4.3</td>
<td>The result of applying the simple thresholding technique to the defect</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td>shown in Figure 4.2</td>
<td></td>
</tr>
<tr>
<td>4.4</td>
<td>Slowly changing background and the detector’s output signal by applying</td>
<td>46</td>
</tr>
<tr>
<td></td>
<td>zero background tracking algorithm</td>
<td></td>
</tr>
<tr>
<td>4.5</td>
<td>Extend simulation environment for zero order background tracking algorithm</td>
<td>47</td>
</tr>
<tr>
<td>4.6</td>
<td>A synthetic texture with slowly moving background. Maximum grey level is</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td>130 and Minimum grey level is 70</td>
<td></td>
</tr>
</tbody>
</table>
Figure 4.7 The synthesized texture shown in Figure 4.6 with three defects with grey levels of 60, 170, 200. ................................................................. 48
Figure 4.8 The defective texture shown in Figure 4.7 with added salt and pepper noise ................................................................. 49
Figure 4.9 The result after applying the zero order background tracking algorithm with false alarm rejection by trigger association. ........................................... 49
Figure 4.10 A modified thresholding technique to ignore isolated high contrast pixels as being considered as defect ........................................... 50
Figure 4.11 A defective sample scanned by the test setup. Nonuniform illumination makes a preprocessing step necessary ........................................... 52
Figure 4.12 The result after applying a simple thresholding technique on the defective sample shown in Figure 4.11 with a threshold of 70 ........................................... 53
Figure 4.13 The result after applying the simple thresholding technique on the defective sample shown in Figure 4.11 with a threshold of 100 ........................................... 54
Figure 4.14 The result after applying the thresholding technique described in “Excess over thresholds filter” on page 51 to the defective sample shown in Figure 4.11 with a threshold of 100. The excess over threshold level, , is 130. ........................................... 55
Figure 4.15 Simulation environment for the delta-sigma tracker ........................................... 56
Figure 4.16 Sample plots from a simulation run for the Delta-Sigma tracker ........................................... 57
Figure 5.1 A stream of data with added defect and accumulating prediction error ........................................... 67
Figure 5.2 A textured pattern with an added defect ........................................... 67
Figure 5.3 The output of the AR method applied to Figure 5.2 with n=7 and threshold=0.2 showing the effect of accumulative prediction error ........................................... 67
Figure 5.4 The output of the AR method applied to Figure 5.2 with n=5 and threshold=0.2 ........................................... 68
Figure 5.5 Horizontal bars ........................................... 69
Figure 5.6 Vertical bars ........................................... 69
Figure 5.7 Diagonal bars ........................................... 69
Figure 5.8 A Gradient pattern ........................................... 69
Figure 5.9 A synthesized pattern with an added defect ........................................... 70
Figure 5.10 Grey level of the defect portion from Figure 5.9 ........................................... 70
Figure 5.11 Magnitude of the relative error ........................................... 71
Figure 5.12 The final output of the AR method applied to Figure 5.9 with n=7 and threshold=0.3 ........................................... 71
Figure 5.13 A synthesized pattern with three defects in different sizes and grey levels ........................................... 71
Figure 5.14 The output of the AR method applied to Figure 5.13 with n=7 and threshold=0.3 ........................................... 72
Figure 5.15  The output of the AR method applied to Figure 5.13 with n=7 and threshold=1 ................................................................. 72

Figure 5.16  A defective sample and the result after applying the AR method ................................. 72
Figure 5.17  A defect sample with almost uniform background .............................................. 73
Figure 5.18  Defective sample shown in Figure 5.17 with added salt and pepper noise 74
Figure 5.19  The output of the AR method for the noisy defective sample of Figure 5.18 using a counter with the threshold of three ............................ 74
Figure 5.20  n – 1 pixel from the end of each line is used to predict the first pixel in the next coming line ............................................................................. 75
Figure 5.21  It is assumed that the pixels at the beginning of each line are defect free to start the process of the prediction ........................................ 75
Figure 5.22  Simplified signal flow diagram of the 1-D AR Algorithm .............................................. 78
Figure 6.1  Membership curve for brightness of a pixel for a grey level image ........................... 82
Figure 6.2  Trapezoidal and triangular membership functions .................................................... 83
Figure 6.3  Commonly used membership functions ................................................................. 83
Figure 6.4  Fuzzy logic system ..................................................................................................... 87
Figure 6.5  A texture with wrinkle defects ..................................................................................... 97
Figure 6.6  Grey level of 5 image rows for the texture shown in Figure 6.5 "A texture with wrinkle defects" on page 97 ......................................................... 98
Figure 6.7  Block diagram of a fuzzy defect detector engine with two inputs and one output .............................................................................................................. 101
Figure 6.8  A sample texture with stain defect ................................................................. 103
Figure 6.9  Histogram of the average grey level for each line for the non-defective portion of the texture shown in Figure 6.8 "A sample texture with stain defect" on page 103 ................................................. 104
Figure 6.10  Histogram of the average grey level for each line for the texture shown in Figure 6.8 "A sample texture with stain defect" on page 103 ................. 105
Figure 6.11  Histogram of the number of jumps (edges) in each line for the threshold of 25 for the defect free part of the texture shown in Figure 6.8 "A sample texture with stain defect" on page 103 ............................................. 106
Figure 6.12  Histogram of the number of jumps (edges) in each line for the threshold of 25 for the defective texture shown in Figure 6.8 "A sample texture with stain defect" on page 103 ................................................................................................................. 107
Figure 6.13  Three membership functions for the “AGL” input for the defect shown in Figure 6.8 "A sample texture with stain defect" on page 103 ................. 108
Figure 6.14  Two membership functions for the “# of jumps” input defined for the defect shown in Figure 6.8 "A sample texture with stain defect" on page 103 ................................................................................................................. 109
Figure 6.15  Three membership functions for the output “Defect” obtained for the defect shown in Figure 6.8 "A sample texture with stain defect" on page 103
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 3.1</td>
<td>The human eye versus Silicon</td>
<td>31</td>
</tr>
<tr>
<td>Table 6.1</td>
<td>Input and output membership functions and their associated parameters</td>
<td>110</td>
</tr>
<tr>
<td>Table 7.1</td>
<td>Comparison of the simulated and applied algorithms</td>
<td>123</td>
</tr>
<tr>
<td>Table C.1</td>
<td>XC4000E and XC4000X Series Field Programmable Gate</td>
<td>156</td>
</tr>
</tbody>
</table>
List of Abbreviations

ALU: Arithmetic Logic Unit

AR: AutoRegressive

ASIC: Application Specific Integrated Circuit

Byte: 8 bits

CCD: Charged Coupled Devices

CLB: Configurable Logic Block

CMOS: Complementary Metal Oxide Semiconductor

DRAM: Dynamic Random Access Memory

DSP: Digital Signal Processing (Processor)

EEPROM: Electrically Erasable Read Only Memory

EXSYNC: External Sync.

FIFO: First In First Out

FPGA: Field Programmable Gate Array
List of Abbreviations

LVAL: Line Valid

MAC: Multiplication followed by Accumulation

MCLK: Master Clock

MCU: Microcontroller Unit

PCB: Printed Circuit Board

PC: Personal Computer

PEL: Pixel per Line

PLD: Programmable Logic Device

PVAL: Pixel Valid

PE: Processing Element

RAM: Read Only Memory

SIMD: Single Instruction Multiple Data

SPMD: Single Program Multiple Data
List of Abbreviations

VLSI: Very Large Scale Integrated Circuit

TDI: Time Delay and Integration

VHDL: Very High Speed Integrated Circuits - VHSIC Hardware Description Language

USB: Universal Serial Bus
1.1 Introduction

Since the beginning of the Industrial Revolution, human vision has played an indispensable role in the process of manufacturing products. Human eyes did what no machines could do themselves: locating and positioning work, tracking the flow of parts, and inspecting output for quality and consistency. Today, however, the requirements of many manufacturing processes have surpassed the limits of human eyesight. Manufactured items often are produced too quickly or with tolerances too small to be analyzed by the human eye. In response to manufacturers' needs, a new technology known as "machine vision" emerged, providing manufacturing equipment with the gift of sight.

In a typical machine vision application, a video camera positioned on the production line captures an image of the part to be inspected and sends it to the machine vision computer. The computer then uses sophisticated image analysis software to extract information from images and generates decisions about those images. Once the vision system has processed the image and made any necessary analysis, the inspection result is then communicated to other equipment on the factory floor, such as an industrial controller, a
robotic arm, a deflector which removes the part from the line, or a positioning table that moves the part. This process is repeated for each part on the line, or continuously for process material, as it moves into position in front of the camera. In fact, vision systems can perform inspections quickly enough to keep pace with machines that process thousands of items or material feet per minute, thus increasing both quality and productivity.

In other words, industrial machine vision is the use of computer processing of images that arise from manufacturing processes. As such, industrial machine vision is a sub-field of the larger discipline of computer vision, which tackles the problem of computerized image interpretation. The business case for machine vision is well known: the subjectivity, fatigue, slowness and cost associated with human inspection can be overcome by the consistency, accuracy and round-the-clock repeatability offered by machine vision systems [22],[110]. As a consequence, the number of machine vision systems being deployed in manufacturing is steadily increasing. Machine vision must evolve to keep pace with rapid advancements in manufacturing technology as well as development of hardware and new sensors.

Most manufacturers are concerned with the cosmetic property of their products; customers quite often equate quality of appearance with functional quality. So, to ensure the successful long-term marketing of an item, it is highly desirable that its appearance is checked visually before packaging and shipping. Likewise, it is desirable that the inspection process be automated and effected without human intervention.

1.2 Machine Inspection Versus Human Inspection

Typically machine vision systems will perform the same task being done by humans, e.g., defect classification. However, machine vision systems will not necessarily displace humans, but rather, augment the capabilities of humans to perform inspection tasks. Thus machine vision systems as well as humans will have to work conjointly to solve manufacturing problems. It is essential to monitor the performance of the machine vision
system on a regular basis. One needs to check whether the system performs as expected. In all these cases, interactions between the vision system and humans are necessary.

The efficiency of a human inspector declines over time due to the dull and repetitive nature of the work. The replacement of the inspector with a cost-effective automated visual inspection systems is an attractive proposition. This, in fact, could offer other advantages such as the ability to perform inspection in hostile environments, the saving of labor costs, and the reduction in demand for highly skilled inspectors. Such an automatic visual inspection system would be able to keep records which may be used for statistical analysis and for optimizing management decisions. Moreover these systems allow high speed inspection to be matched with high speed production.

**Safety and reliability.** Considerations of safety and reliability usually arises in environments which are hazardous for humans (e.g in close proximity to a mining pit) or because manufactured parts are of critical importance and 100% inspection is required. Machine vision also facilitates consistency in inspection standards.

**Product quality.** High volume production using humans seldom facilitates inspection of all parts but automated visual inspection techniques may make it feasible; this depends on the complexity of the task and the effective throughput that is required by the manufacturing system. The latter consideration is particularly important if the vision system is to be incorporated in an on-line manner, i.e. inspecting each part as it is manufactured.

### 1.3 One Step Inspection Versus Multi Inspection Process

In the 1980s, the concept of continuous product or process improvement became popular in manufacturing industries as they strived to achieve incremental improvements in product design and manufacturing processes. The basic idea is to monitor critical process parameters throughout product development to determine the principal sources of product
and process variation. This knowledge is then applied to develop corrective /control processes, which are incorporated into the manufacturing cycle. The net effect is to continuously improve overall product/process quality over time. As illustrated in Figure 1.1, this translates into a shift away from a one or two step inspection process to a multi step (i.e. near continuous) inspection process.

Figure 1.1 (a) One step Inspection versus (b) continuous product inspection

1.4 Challenges in Vision Systems

On the one hand, the acceptance of visual inspection systems depends on the economical aspects. On the other hand, it depends on what the monitoring system can display in real-time conditions and how the system fits into a complex solution for process automation. The image processing system must provide details of the manufactured part, process, or machine that are relevant to the manufacturing technology. The solution to this problem is usually complex and context dependent. A domain-specific interpretation of the image is required. Not only is knowledge about the objects in an image necessary, but also
knowledge about the technical background is required [5]. The acquisition, representation, and use of domain-specific knowledge are the key points.

Despite significant advances, it is still not proven possible to design a flexible, cost effective, off the shelf inspection system suitable for all types of industrial applications due to the need to consider the individual requirements of each application.

If automated vision systems are to become more widely used for defect detection they must address the criteria of cost, speed and reliability. Cost limits the computing resource that is feasible to use, but more importantly, means that an inspection system should not have to be tailor made for each inspection task and so must be flexible and easily reconfigurable for new tasks. The speed factor arises because an inspection system is typically constrained to operate within the cycle time of the production process which can be as small as a few seconds. Since defects are themselves very rare but must be detected with high reliability when they occur, a useful system must achieve very low false positive and false negative rates and must be capable of running without supervision over many inspection cycles.

Considering the economic aspects, the system must perform the following tasks:

1. Continuously control the web or production with an optical electronic sensor.
2. Examine the defect type without operator intervention.
3. Support the operator in making his/her decision during disturbances.

For defect detection it is usually required to use some kind of training or learning mechanism. Typically, learning systems require instances of prototypes. However, if the product is in an early stage of manufacture, very few defect samples may be available for automatic training.
The vast majority of machine vision systems in use today are greyscale systems. Tasks such as gauging and measurement can be adequately performed with greyscale systems. However, colour is an indispensable feature to detect and classify defects in many applications, ranging from semiconductor inspection to food inspection.

There are still very few computer vision applications in manufacturing. The reason is that two of the major bottlenecks, speed of algorithm development and methods to validate algorithm performance with a limited data set, remain largely unsolved.

1.5 Research Objectives and Review

The main objective of this research is to develop suitable algorithms for real time defect detection in web inspection systems which can be implemented in the limited resources of our target in-camera processing system. Unlike the traditional techniques used to identify defects, which use algorithms based on the availability of full two dimensional image data, our proposed methods are based on one dimensional processing of data from raster scanned video output. The proposed algorithms should operate in real time. In this application, real-time means that the system must perform all of the required operations within the critical time frame allotted, usually at the video data rate.

In this thesis, for non-textured background materials, three algorithms have been developed for the in-camera system: two (or multi) level thresholding; zero order background tracking; and delta modulation background tracking [59], [60], [61]. Some modifications are applied in order to reduce the false alarm rate caused by the environment noise.

For textured materials, finding a suitable algorithm considering the hardware constraints is a challenging task. Auto-regressive techniques have been developed and implemented as a statistical approach to analyze textured backgrounds and to identify possible defects. This method of analysis has been extensively used to study visual textures. In the simplest form, the image is scanned to provide a one dimensional series of grey level fluctuations.
which is treated as a one-dimensional stochastic process evolving in "time". In a more comprehensive form, a pixel value is assumed to depend upon a certain part of its neighborhood. The coefficients of dependence are extracted using time series analysis techniques. This method can be applied to many two dimensional textured backgrounds and the experimental results are encouraging. Our literature survey shows no record of similar research.

We have successfully developed a novel algorithm for defect detect detection based on fuzzy fusion of texture features. We have been inspired by the fuzzy definition of defects in manual detection such as darker or brighter regions, and smaller or larger objects. Unlike the traditional methods, fuzzy computing is aimed at an accommodation with the pervasive imprecision of the real world. The guiding principle of fuzzy computing is to exploit the tolerance for imprecision, uncertainty, and partial truth to achieve tractability, robustness, and a low solution cost. The fuzzy logic algorithm uses imprecise or ambiguous image data caused by instrumental error or environmental noise such as dust or small variations in illumination to obtain a precise result. The proposed algorithm has been simulated on many defect samples from production web systems and successfully implemented on the experimental test setup. This algorithm can be applied to inspection of both textured and non textured materials and offers superior performance over traditional methods such as template matching. We believe that, in the coming years, fuzzy computing is likely to play an increasingly important role in the conception and design of systems whose complexity is much higher than that of systems designed by conventional methods.

1.6 Thesis Organization

This thesis is organized as follows. Chapter 2 provides a literature survey on inspection systems and the definition of the terminology used in this area. In Chapter 3 we explain the test setup used in this research. The DALSA TDI Line scan camera and its architecture is also explained in this chapter. First and second generation processing boards are explained at the block diagram level. Lighting and image acquisition, an important issue in any
image system, is also discussed. Chapter 4 introduces general thresholding techniques and their implementation in the limited resources of one FPGA. It also presents new thresholding algorithms as part of the early research work in this project. Chapter 5 discusses time series analysis of the texture and the application of 1D AR method for defect detection on textured and non-textured materials. The theory is supported by simulation results both on synthesized and some real world defective samples. In Chapter 6, we describe a novel approach which is used for defect line detection based on fuzzy fusion of texture features. The proposed method is simulated and successfully implemented in the limited resources of one XC4005E FPGA. Chapter 7 summarizes the thesis and its contributions and provides ideas and topics to carry the results obtained in this work to future milestones.

A survey of commercial machine vision systems hardware and software is provided in Appendix A and Appendix B, respectively. An introduction to Field Programmable logic is provided in Appendix C, and specifications for the DALSA TDI Line-Scan camera used in this work are provided in Appendix D. VHDL code for the fuzzy logic algorithm is given in Appendix E. In Appendix F Matlab and Simulink programs written for testing the proposed algorithms are presented. Appendix G discusses the selection of the most suitable FPGA for a target application and illustrates the issues with an example.
Chapter 2

*Inspection Systems, Introduction and Definitions*

### 2.1 Introduction

This chapter presents a literature survey in the area of defect detection systems and associated algorithms. We also explain some of the terminology used in this thesis. Vision sensors, which are used in inspection systems, are also explained. We also provide brief descriptions of algorithms used in general purpose machine vision systems. A survey of commercial machine vision systems (hardware and software) is also provided in Appendix A and Appendix B as a complement to this chapter.

### 2.2 Web Inspection

Web inspection applies machine vision techniques to materials that are made in a continuous web process such as paper, glass, aluminum, cold-rolled steel, film, and wood. In the general machine vision model a “frame-based” camera is used; i.e., a camera that takes a “snapshot” of the image N number of times per second and for each snapshot a two dimensional image is generated. In web inspection applications a line-scan camera is often used, with a composite picture being built up as the image moves across the sensor. The speed at which the line scan camera is clocked depends
on the speed of the web and the minimum size of defect to be detected. It is not unusual for webbed material to be travelling at a rate of 1000 feet per minute, and since the web is continuous, these applications tend to be more demanding than other machine vision applications [55].

The vast majority of machine vision systems in use today are greyscale systems. Tasks such as gauging and measurement can be adequately performed with greyscale systems. However, colour is also used to detect and classify defects in many applications, ranging from semiconductor inspection to food inspection.

2.3 Defect - Definition, Detection

Any deviation from the standard expected output is considered a defect. A standard output is a product or print that a process engineer defines as “correct”. Defect types are not only application dependent, but also vary from product to product, from level to level, and throughout the product cycle.

Uncertainty exists in human judgment itself concerning what are good products and what are not. Occasionally we are forced to define a defect in mathematical terms in order to make an automatic inspection system. Human sensory capability and even psychological effects on decision making must be investigated if we wish to design a machine which matches or even exceeds human inspection performance.

A defect detection system must imitate three essential abilities of a human operator:

1. The visual detection of a defect
2. The classification and interpretation of the kind of defect from visual information
3. The classification and interpretation of the process condition with a proposal for therapy
Points 2 and 3 cannot be considered separately; the complexity of the task requires them to be separated into two partial tasks. The following questions arise for the system designer:

1. What spatial resolution is necessary for the system to detect and analyze the defects?
2. Which visual features does the operator use in identifying the type of defect?
3. What is the structure of the classifier for identifying the type of defect?
4. How should the system be integrated on-line?

2.4 Real Time Issue

The phrase “real time” in vision applications originally meant any imaging system capable of operating at a rate of 30 frames per second. A more general definition is:

A real time system must perform all of the required operations within the critical time frame allotted [54].

This means that execution times must be guaranteed and, even under conditions of extreme system loading, the timing and logical sequence of the system’s response must be correct. Resources are as important as the system clock rate. For example, a 400 MHz system with a $4 \times 4$ hardware convolver would take twice as long to perform a convolution as a 200 Mhz system with an $8 \times 8$ hardware convolver, because the 400 MHz system would require four passes to complete the same algorithm that the 200 MHz system would perform in a single pass.

2.5 Vision Systems’ Sensors

At present there are a variety of sensor devices used in automatic visual inspection. These include vidicon cameras, linear array cameras, laser scanners, and infrared sensors. Vidicon cameras based on broadcast television standards are not expensive and are therefore quite popular. Some of their disadvantages are short tube life, and the need for
constant calibration for drift and aging. Laser scanners are employed mostly for depth measurements. Time Delay Integration (TDI) are much more sensitive to the light needed in high speed applications. This specialized video technology is designed for use in continuous scanning applications such as web or sheet metal inspection.

### 2.5.1 Linear array sensors

This is an important class of solid-state sensor and camera for machine vision. These sensors contain a one-dimensional array (row) of photosites, rather than a full two-dimensional array. A linear sensor can have a much larger numbers of photosites in a row compared to a conventional area sensors; but, since they are one dimensional devices they can only take pictures of slices of a two dimensional scene. Thus, these sensors are best suited to inspection applications in which the scene is to be scanned in a continuous linear motion.

A major advantage of the line scan camera is that it is much easier to provide uniform lighting along a row than for a 2D area [108].

Among the limitations of linear array camera based systems is very high light intensity requirements. The higher the speed, the higher is the requirement for the light intensity and integration time. One of the problems associated with the illumination for line scan cameras is that, as the speed of the web increases, the exposure or integration time must decrease in order to keep the moving web in synchronization with the sampling rate of the camera so that the blurring effect is minimized. To reduce this high integration time, the light intensity must be increased proportionally [41]. The introduction of the TDI system for moving images has greatly reduced the light intensity problem.

### 2.5.2 Cameras

Solid state CCD cameras are the most widely used in machine vision. The camera functions only as a transducer in the current generation of machine vision systems. All image processing happens after the camera's video signal is sampled and digitized to
produce a matrix of numbers in the frame buffer. This is significantly different from the human vision system in which substantial nonlinear preprocessing and image representation are performed in the eye and optic nerves. Transducer quality describes its ability to accurately record the signal it is intended to measure. The best camera is the one whose video output allows filling the frame buffer matrix with values which most faithfully represent the luminances at the corresponding points in the scene.

**Camera transfer function and noise**

A camera transfer function is conventionally defined as the function of the output voltage with respect to sensor irradiance. When using such a function, one assumes that all camera pixels have an identical response. In fact, variations in response between individual pixels is a major source of camera noise. The sensitivity can vary spatially across the surface of each individual pixel, the sensitivity of individual pixels are not perfectly matched, and the dark level outputs of pixels can differ. Also, there is cross-talk between adjacent pixels caused by charge or light diffusion or inefficient charge transfer during readout. This means that the voltage output from a particular pixel may depend not only upon the amount of light striking it, but also upon the amount of light striking adjacent pixels. In addition to the spatial variation in sensitivity and offset of the camera sensor, there are also temporal variations. This means that the effective gain and offset of an individual pixel may differ between successful image frames. These variations are caused by statistical fluctuations in photon arrival rate, electronic noise, and other effects and must be characterized by any camera measurement systems [107]

**Resolution**

Spatial resolution is an important camera characteristic. Machine vision systems often need to make measurements at or near the camera's resolution limit. Even when small feature detection is not important, improved camera resolution results in improved accuracy in determining object location, orientation, size, etc. The resolution is often specified by the size and number of pixels.
2.6 Illumination

Lighting is a major issue for many machine vision and image acquisition systems, and the best lighting approach is often found experimentally. Different applications have different lighting requirements. If the images fed to the vision system are not optimized in terms of illumination, a great deal of unnecessary computer processing may be required to achieve the desired defect detection performance, and this may cause significant reduction in the achievable throughput of an inspection system. For this reason, it seems wise to spend a considerable amount of time in optimizing the lighting for every inspection task [47].

In vision, image understanding is dependent upon information in the image, and this depends upon the illumination of the objects viewed. In designing an inspection system, the initial temptation is to aim a convenient light source at the unit being inspected, to obtain an image of the unit, and to leave the rest to be done by the software [105]. Unfortunately, the resulting image is often inadequate even with the best software algorithms. Many engineers working on machine vision have rediscovered the importance of lighting and will appreciate Thompson’s comment [104], “In fact, a successful industrial vision system may depend more on well-designed lighting than on sophisticated image analysis”. Batchelor et al discuss the issue of lighting equipments, optics and lighting techniques in their book on automatic visual inspection [106].

2.7 Texture Definition or Image Modeling

Texture perception is an important part of human vision. Despite its ubiquity in scene analysis a precise definition of texture does not exist. Clearly, any such definition must be relatively simple and incorporate all important features of the texture that determine its various perceptual attributes. Pickett [92] "R.M. Pickett, B.S. Lipkin and A. Rosenfeld Eds “Visual analysis of texture in the detection and recognition of objects- in Picture Processing and Psychopictorism,” New York: Academic 1970, pp. 289-308" on page 131 views texture as consisting of a large number of elements, each in some degree visible,
and, on the whole, densely and evenly (possibly randomly) arranged over the field of view such that there is a distinct characteristic spatial repetitiveness in the pattern.

Inspection of textured surfaces is a particularly challenging problem for the vision scientist. In general, whether the texture to be inspected is regular or random, in image terms it is characterized by local variations that render the problem of texture defect detection extremely difficult since defects are often manifested by grey-level changes. Even if their patterns differ from the normal pixel variation, defect detection involves more than a mere pixel comparison. In order to detect the defects in a textured material, the texture itself should be modeled faithfully. The subject of image modeling involves the construction of models or procedures for the specification of images. These models serve a dual role in that they can describe images that are observed and also can serve to generate synthetic images from the model parameters. There are four important areas of image processing in which texture plays an important role: classification, image segmentation, realism in computer graphics, and image encoding.

By a model of a texture, we mean a mathematical process which creates or describes the textured image. The models which have been used to generate and represent textures include: time series models; fractals; random mosaic models; mathematical morphology, synthetic methods; and linear models.

All treatments of texture by others so far have taken one of the following approaches [68], [70], [72]:

2.7.1 **Statistical approach:**

This approach attempts a global characterization of the texture. Statistical properties of the spatial distribution of the grey levels are used as texture descriptors. The key feature of this approach is the sole dependence of the description on point properties, with no explicit use of elements or subregions.
2.7.2 Structural approach

This approach conceives of texture as an arrangement of a set of spatial sub-patterns according to certain placement rules. The sub-patterns themselves are, in general, made up of smaller sub-patterns, positioned according to some placement rules. This recursive nature of the approach captures the hierarchical structure of natural scenes. Both the sub-patterns and their placement may be characterized statistically. Most existing texture models are based upon the first approach but it is obvious that images that are piecewise uniform with respect to some property, and are known to have been generated by a structural approach, are better modeled using a structural approach.

2.7.3 Defect detection on textured images:

Defect detection of textured materials is a challenging task that needs specialized hardware and dedicated software. Different approaches have been used to tackle this problem by many researchers. Some methods are based on statistical representation of the texture while the others are based on structural modeling to represent the texture. Generally the inspection systems used for defect detection on textured materials are more complex than those used for non textured materials. For many applications, the colour information of the texture is used for defect detection. This approach increases the complexity of the system. In this section some of the published work in this area is briefly described.

Tatari [121] presented an approach for the selection and provision of parameters for texture analysis methods. Some of these methods have been implemented in a system for inspecting carpets, softwood boards, and ceiling tiles. Chen and Jain [135] have reported a structural approach for detecting defects in textured images, based on location and length histograms of the skeleton of the textural images. They suggested that statistical measurements for fluctuation, mean jumps, can provide an effective way to detect defects. For the purpose of quality control Siew et al [136] presented a study to find measurements that reveal carpet wear. Two approaches have been used, one based on grey level co-occurrence matrices, and the other using statistics derived from the probability densities of
the values of various local properties of texture measurements. Takatoo et al [122] developed a system for inspecting defects such as stains on plain cloth and print errors on patterned cloth, using two dimensional greyscale image processing techniques. The system may also be applied to other materials like steel, paper, and leather sheets. Backstorm and Pulkinen [123] proposed a paper printability analysis workstation with a modular and interactive architecture. Here, a surface modeling module appears to be capable of texture identification using Markov random field analysis, 2D autocorrelation and Fourier analysis. After bright points are segmented, the application of statistical methods helps to detect paper surface impurities and holes. Cohen et al presented an inspection system for textile fabrics. The fabric textures are modeled by a Gaussian Markov random field, which proved able to capture the visual textural content of the different fabrics very well. The defect detection process is cast as a hypothesis-testing problem on statistics derived from the model.

2.8 Algorithms Employed for Defect Detection

The algorithms employed for defect detection are numerous. Some of the approaches used for defect analysis are cellular neural networks [50], artificial neural networks [15], [52] morphological image processing [56], thresholding [109], Fourier transform [39], Gabor filters [49], and template matching [102]. While an algorithm might be suitable for a specific target application, it might not perform well for other applications. Geometry-driven (template matching), morphological method, and fuzzy logic are explained in brief in the following sections while the others are outlined in the literature survey in Appendix B "Survey of Machine Vision Software" on page 141.

2.8.1 Geometry-driven (template matching) inspection

Geometry-driven inspection can be broadly divided into methods which use a reference part or 'golden template' as ground truth to compare and finally to detect possible defects. The part reference approach use image-difference techniques to detect differences between a reference part (the template) and a part being inspected [98]. The template method is easy to calculate and implement in a specialized hardware processor. However,
the method requires stability of location; otherwise, the current image aligns with the template image before the difference image is calculated. This approach has proven successful for VLSI photolithographic mask inspection, where components meet rigid geometric specifications and need to be placed to high precision [99],[100]. One problem here is how to interpret the differences which may arise from irrelevant variations in intensity due to illuminations, material nonuniformity and shadow. When using template matching for defect detection, large pixel size can act as a spatial filter. Thus no further alignment is needed. The template image is acquired on-line after a process engineer has concluded that the process is stable. The data acquisition unit takes a good image from the material and stores it in the memory of the processor. Much effort should be taken in making the data acquisition as perfect as possible to reduce the number of preprocessing and defect recognition steps. This prevents the algorithm implemented by the specialized processor from becoming too complicated. Further, this approach does not work well for manufacturing components such as castings, which have natural part-to-part dimensional variation. For illustrative purposes, the following describes two applications for template-based processing:

**Shape based diamond classification:** The shape of a diamond with optimal abrasive properties should lie midway between a cube and an octahedron. However, in practice, manufactured diamonds exhibit a wide range of shapes, ranging from cubic to octahedral. By defining a parameter $\tau$ which varies from 0 for an octagon to 1 for a cube, a system was developed for classifying diamond samples based on their calculated $\tau$ parameters. The algorithm runs on a SPARC2 workstation and processes diamonds at a rate of around one crystal a second [101].

**Aircraft engine parts:** Template-guided X-ray inspection is used for aircraft engine parts. In this application, a number of 2D X-ray images (typically three or four) are generated of an object at predetermined viewing angles. These images are used to check the integrity of the internal geometry of a part after machining operations. A number of template-based algorithms for automatically detecting drilling defects are developed. For example, the parameters derived from the deformed primitives can be compared to the
parameters of the best-fit template to detect geometric flaws such as inaccurately drilled hole diameters [102].

2.8.2 Morphological analysis:

Morphological inspection algorithms are based on the early work of Rosenfeld and Pfalz [117], and Montanari [118]. The basic operations of morphological processing are the expansion/dilation and the contraction/erosion of binarized images. In printed circuit board (PCB) inspection applications, the morphological transformation of the PCB image is carried out by repetitive erosion of the binary image, while preserving local connectivity of its digital components. The image now consists of some skeletonized entities; this procedure is usually called thinning or skeletonizing. Defect verification is executed by searching for false representations. Since the defects are transformed into simple geometrical forms, they can be easily detected by comparison with the shapes in an ideal pattern. The number of these abstract shapes are small, and they are, in fact, simple binary representations. This has allowed for a high-speed implementation using a pipeline architecture. Ejiri et al [119] developed a method based on expansion and contraction of inspecting PCBs. Their principle does not require any predefined model of perfect patterns.

![Diagram](image)

**Figure 2.1 Principle of the expansion - contraction method for detecting small defects**
2.8.3 Fuzzy logic

Zadeh conceived and introduced fuzzy set theory [73] which is aimed at handling imprecise data. It is an extension of classical set theory and provides a mechanism for representing linguistic attributes such as "little", "more or less", "often", "sometimes". This provides an ability to measure the degree to which a type of a pattern is present or an event has taken place. In contrast, classical set theory describes crisp events, i.e., events that either happen or do not happen. The combination of fuzzy sets and pattern recognition is a topic investigated by many authors. Zadeh also stated the relationship between pattern classes and fuzzy sets [73]. Bezdek [124], [125] studied fuzzy objective functions in great detail and developed a fuzzy k-nearest neighbour classifier. His research also established an extensive body of work dealing with the fuzzy c-mean algorithm. Pham and Correchano developed synergistic structures for pattern recognition combining fuzzy systems and neural networks and implemented a self organizing neural network based pattern clustering method with fuzzy outputs [126]. More discussion on applications of fuzzy logic in defect detection systems and also the development of a novel method is presented in Chapter 6.

2.9 Commercial systems

There are many commercial inspection systems manufactured by different companies. These systems employ a wide variety of different hardware and sophisticated software targeted for a wide variety of applications. Some systems have general purpose usage and can be used in different production lines. Generally, low cost inspection systems are those that have simple hardware and mainly use thresholding type algorithms. No current systems appear to apply the hardware structure and associated algorithms that are reported in this thesis.

For brevity in the main body of this thesis, the descriptions of commercial systems are available in Appendix A "Commercial Defect Detection Systems" on page 136.
2.10 Hardware Issues

General purpose processors are used to perform calculations in a wide variety of applications including image processing. Most processors have a single central processing unit (CPU), with one or more Arithmetic/Logic Units (ALU) connected to blocks of memory; most processors are based on synchronous design concepts using a single master clock. A simple $3 \times 3$ convolution operation, typical in image processing operations, could take the CPU 10 or more clock cycles to fetch the instructions, fetch the three new pieces of data to be operated on, perform the various mathematical operations and write the result back into the memory. Assuming a 200MHz clock, the ten clock cycle step now reduces the pixel processing rate to 20MHz per pixel for this step. Again, assuming a 10-step process of only monodic processes, the pixel processing rate is reduced down to 2MHz. With 200 Megabits per second of image data coming out of only one camera, it is easy to see how this architecture will quickly become overwhelmed.

One of the strategies used to increase the processing power of the Von Neumann architecture is to use more than one of these types of processors in tandem. In this way a particular algorithm or processing task can be divided up amongst the processors thereby decreasing the total time it will take to perform the operation. While this strategy can be effective, it also has its drawbacks. Parallel processing architectures tend to be difficult to program, and dividing the processing task equally and efficiently amongst the processors can be a challenge. It has been proven that a combination of pipelining and parallel processing gives the best result. Usually the desired output of an image processing system in machine vision is not an image but an answer: the part is either good or bad, the web has a defect on it or not. In order to arrive at these answers the data must be examined by a general purpose processor after it has been preprocessed by the pipeline architecture. For a complete system solution quite often the best combination of speed and flexibility is a hybrid architecture consisting of a pipeline processing architecture for the front-end preprocessing and a Von-Neumann processor for the back-end decision making. It should be noted that in this architecture, the more work the pipeline processor can do, the better.
Due to the increased computer power available on current workstations, and the increasing number of plug-in boards for image processing, the need to create special purpose hardware for inspection systems is diminishing. A new trend is the decreasing cost of workstations and personal computers. This means that inspection tasks which could not be automated in the past, due to cost/performance reasons, can now be reconsidered for implementation [111].

2.11 Software Environment

The environments under which most vision applications are being developed are becoming standardized. Most development is taking place on UNIX platforms with C and C++ code, and within the X-Windows display environment. This means that researchers can exchange code for different algorithms. Furthermore, image processing and vision software toolkits are available such as Khoros [115] and Simulink-Matlab [114], so that minimal or no programming is required to get simple applications running. This ease of use is partly being achieved through visual programming techniques, whereby the user has to simply connect different computational modules together on the display. The continued availability of such software tools and environments specialized for vision and image processing will improve the productivity of vision researchers.

Again, for brevity, the software survey has been placed in Appendix B "Survey of Machine Vision Software" on page 141.

2.12 Conclusion

The number of potential applications for inspection systems are extensive. However, until recently, these systems have had little impact. One reason for this is that inspection systems were often expensive. However things are changing. In the last few years, the price of computer equipment has decreased dramatically. Desktop workstation speed is now adequate for most vision algorithm computations. In addition powerful PCs are becoming available, which provide similar speed to workstations but at a reduced cost. A
challenge now is to demonstrate that algorithm development and validation time can be reduced to an acceptable level. One of the key problems in algorithm validation is that it can take a long time to gather statistically enough significant data sets on which to validate the algorithms. In this chapter we have explained the basic concepts of defect detection systems and followed this with a description of available systems in the market. Some of the research work published in this area was also examined.
3.1 Introduction

This chapter describes the test setup and the processing hardware embedded inside the camera which was used for the experimental tests in this research. A good understanding of the limitations in the processing hardware is vital in the selection of suitable algorithms for defect detection applications. We also describe the TDI technology used in the DALSA CCD cameras employed in this work. The importance of well controlled illumination in web inspection systems is also discussed.

3.1.1 Test setup

A test fixture that simulates a web manufacturing process and provides for variable speed operation has been setup as shown in Figure 3.1. The test setup comprises a DALSA TDI Line scan camera, a motorized drum with shaft encoder for TDI synchronization, and a D.C. light source with fibre optic light guides and output slits. Our FPGA processing board is mounted above the regular camera control boards. The camera uses a 55mm 1:2.8 Nikon zoom lens for flexibility. Defective samples from various web sources are used for testing and verification of
candidate algorithms. The defective samples are affixed to the revolving drum, and the scanning rate of the camera is synchronized with the angular velocity of the drum. The illumination is adjusted to obtain the best defect detection result from the algorithm. The size of a detectable defect depends on the canonical distance of the camera from web and the speed of the production line. The minimum size of detectable defect in our setup is 0.01mm.

Figure 3.1 Test setup for production line simulation
3.2 DALSA TDI Line Scan Camera

The DALSA CL-E1 camera is used in our system; this is a 1024 x 96 TDI line scan CCD camera (Figure 3.2). A TDI camera offers increased sensitivity of up to 80 times over comparable line scan cameras, making it ideally suited to applications with low ambient light levels. In TDI cameras the transfer rate should be synchronized to the image movement (web velocity in our target application) [128]. TDI sensors and/or cameras can cost a little more for the image capture hardware, but the performance benefits typically result in lower overall system costs due to reduced lighting requirements. Further, savings increase over the system lifetime due to reduced operating costs for low power lighting. For rapidly moving images the much increased light sensitivity and noise rejection associated with TDI cameras allows the use of quite simple defect detection algorithms that may not be effective in the presence of large amounts of image noise.

![Figure 3.2 DALSA TDI line scan camera](image)

3.2.1 CCD and TDI technology

Charged-coupled devices (CCDs) were invented by Boyle and Smith in 1970. Since then, considerable literature has been generated on the CCD physics, fabrication, and operation. The camera requires an optical system to image the scene onto the array’s photo sensitive
area. The array requires bias and clock signals. When properly configured, its output is a series of analog pulses that represent the scene intensity at a series of discrete locations.

CCD refers to a semiconductor architecture in which charge is read out of storage wells. The CCD architecture has three basic functions: (a) charge collection, (b) charge transfer, and (c) the conversion of charge into a measurable voltage. The basic building block of the CCD is the metal-oxide semiconductor (MOS) capacitor. By manipulating gate voltages, charge can be either stored or transferred. Charge generation in most devices occurs under a MOS capacitor (also called a photo gate). For some devices photodiodes create the charge. After charge generation, the transfer occurs in the MOS capacitors. The CCD register consists of a series of gates, and manipulation of the gate voltage in a systematic and sequential manner transfers the electrons from one gate to the next in a conveyor-belt-like fashion.

3.2.2 TIME-DELAY and INTEGRATION (TDI)

Time-delay and integration (TDI) is analogous to taking multiple exposures of the same object and summing them. The addition takes place automatically in the charge well and array timing produces the multiple images. As the image is swept across the array, the charge packets are clocked at the same rate. The relative motion between the image and the target can be achieved in many ways. With objects on a conveyor belt, the camera is stationary and the object moves.

Figure 3.3 illustrates four detectors operating in TDI mode. At time $T_1$ the image is on the first detector and creates a charge packet. At time $T_2$, the image has moved to the second detector. Simultaneously, the pixel clock moves the charge packet to the well under the second detector. Here, the image creates additional charge that is added to the charge created by the first detector. The charge (signal) increases linearly with the number of detectors in TDI. Noise also increases, but at the square root of the number of TDI elements, $N_{TDI}$. This results in an SNR improvement of $\sqrt{N_{TDI}}$. The well capacity is one
of the limiting factors determining the maximum number of TDI elements that can be used.

Figure 3.3 TDI concept. At time T1, the image is focused onto the first element. At T2 the image has moved to the next detector. Simultaneously, the charge packet is clocked down to the next pixel site. After four stages, the signal increases fourfold.

For this concept to work, the charge packet must always be in synchronization with the moving image. If there is a mismatch between the image scan velocity and pixel clock rate, the output is smeared and this adversely affects the in-scan MTF. The accuracy to which the image velocity is known also limits the number of useful TDI stages. If the pixels in a TDI array are 15 µm square and the object is moving at 2 m/sec and the object detail of interest is 150 µm, the lens magnification must be 150/15=10 and the TDI pixel clock rate is (2 m/sec)/150 µm =133 kHz.

3.2.3 Array Architecture

The array architecture is driven by the application. Full frame and frame transfer devices tend to be used for scientific applications. Interline transfer devices are used in consumer
camcorders and some professional television systems. Linear arrays, progressive scan and time-delay and integration (TDI) are used for industrial applications. Despite an ever increasing demand for colour cameras, grey scale cameras are still widely used for many scientific and industrial applications.

Progressive scan simply means the noninterlaced or sequential line-by-line scanning of the image. This is important to machine vision because it supplies accurate timing and has a simple format. Any application that requires digitization and a computer interface will probably perform better with progressive scanned imagery. However, few monitors can directly display progressive scan imagery so an interface is required. Frame capture boards normally provide the interface for computers.

Scientific grade arrays may be as large as 5120 x 5120 elements (26.20 x 10^6 elements). While large format arrays offer the highest resolution, their use is hampered by readout rate limitations. Large arrays can reduce sub-array readout rates by having multiple parallel ports servicing sub-arrays. Each sub-array requires separate vertical and horizontal clock signals. The trade-off is frame rate (speed) versus number of parallel ports (complexity of CCD design).

3.2.4 CMOS Sensors Versus CCDs

CMOS and CCD imagers are constructed from silicon and GaAs. This gives them fundamentally similar properties of sensitivity over the visible and near IR spectrum. Thus, both technologies convert incident light (photons) into electronic charge (electrons) by the same photo-conversion process. Both technologies can support two types of photo element - the photogate and the photodiode.

It is technically feasible, but not economical, to use the CCD process to integrate other camera functions, such as the clock drivers, timing logic, signal processing, etc. These are therefore normally implemented in secondary chips. Thus most CCD cameras comprise several chips, often as many as 8, and not fewer than 3.
Apart from the need to integrate the other camera electronics in a separate chip, the Achilles heel of all CCDs is the clock requirement. The clock amplitude and shape are critical to successful operation. Generating correctly sized and shaped clocks is normally the function of a specialized clock driver chip, and leads to two major disadvantages; multiple nonstandard supply voltages and high power consumption. It is not uncommon for CCDs to require 5 or 6 different supplies at critical and obscure values. If the user is offered a simple single voltage supply input, then several regulators will be employed internally to generate these supply requirements. On the plus side, CCDs have matured to provide excellent image quality with low noise. CCD processes are generally captive to the major manufacturers.

CMOS imagers sense light in the same way as CCD, but from the point of sensing onwards everything is different. The charge packets are not transferred, but they are instead detected as early as possible by charge sensing amplifiers, which are made from CMOS transistors.

In some CMOS sensors, amplifiers are implemented at the top of each column of pixels - the pixels themselves contain just one transistor which is used as a charge gate, switching the contents of the pixel to the charge amplifiers. These passive CMOS sensors operate like analog DRAMs.

In other CMOS sensors, amplifiers are implemented in each and every pixel - these are called “active pixel” CMOS sensors. Active pixel CMOS sensors usually contain at least 3 transistors per pixel. Generally, the active pixel form has lower noise but poorer packing density than passive pixel CMOS.

CMOS sensors of both types can be manufactured using standard CMOS processes from several foundry sources. Vendors of CMOS sensors also benefit from the large investments which are made continually to improve the quality and capacity of CMOS foundries.
CMOS sensors have the problem of matching the multiple different amplifiers within each sensor. Fortunately this problem has been overcome, reducing the residual level of fixed-pattern noise to insignificant proportions.

A major benefit of CMOS cameras over CCD lies in the high level of product integration that can be achieved through implementing virtually all of the electronic cameras functions onto the same chip. CMOS technology is ideal for this. The competitive prevalence of CMOS technology and the high level of camera integration also combine to bring about substantial cost advantages over CCD.

A CMOS image sensor uses just a single 5 Volt (3.3 Volt internal) power supply. Power consumption of a $512 \times 512$ pixel CMOS sensor is typically 50 mWatt for current fabrication processes. Very Large Scale Integration (VLSI) used in CMOS image devices makes it possible to integrate many other ordinary camera functions (A/D conversion, automatic gain control, etc.) in the sensor. A CMOS Active Pixel Sensor is, in fact, a camera on chip. This huge miniaturization in CMOS image sensors is a big advantage compared with CCDs. Table 3.1 on page 31 compares the human eye with CCD and CMOS sensors for several criteria. The specifications for the DALSA TDI Line scan camera used in our system are provided in Appendix D.

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Eye</th>
<th>CCD</th>
<th>CMOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>spectral response</td>
<td>400-700 nm</td>
<td>400-1000 nm</td>
<td>300-700 nm</td>
</tr>
<tr>
<td></td>
<td>peaked at 555</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dynamic range</td>
<td>$1 \times 10^6$ logarithmic</td>
<td>$1 \times 10^4$ linear</td>
<td>$1 \times 10^4$ nonlinear</td>
</tr>
<tr>
<td></td>
<td>$1 \times 10^2$ linear</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dark limit</td>
<td>0.001 lux</td>
<td>typ: 0.1 lux &lt;0.0001 possible</td>
<td>typ: 1 lux 0.001 possible</td>
</tr>
<tr>
<td></td>
<td>$1 \times 10^{-6}$ W/m2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>max. frame rate</td>
<td>15 Hz</td>
<td>10 kHz</td>
<td>&gt;&gt;10 kHz</td>
</tr>
<tr>
<td>pixels</td>
<td>120M</td>
<td>typical: 800K record: 60M</td>
<td>typical: 800K record: 4M</td>
</tr>
<tr>
<td>pixel pitch</td>
<td>2-3 μm</td>
<td>5-10 μm</td>
<td>5-10μm</td>
</tr>
</tbody>
</table>
Table 3.1 The human eye versus Silicon

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Eye</th>
<th>CCD</th>
<th>CMOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>power dissipation</td>
<td>&lt; 1 mW</td>
<td>500 mW typ</td>
<td>50 mW typ</td>
</tr>
<tr>
<td>cosmetic quality</td>
<td>perfect</td>
<td>very good</td>
<td>worse</td>
</tr>
<tr>
<td>colour</td>
<td>ideal</td>
<td>poor (only RGB)</td>
<td>poor</td>
</tr>
<tr>
<td>access method</td>
<td>data driven (focus of attention)</td>
<td>serial only</td>
<td>serial, random access,....</td>
</tr>
<tr>
<td>operating Temp.</td>
<td>36°C</td>
<td>-200°C...+200°C</td>
<td>0°C... 100°C</td>
</tr>
</tbody>
</table>

3.3 Processing Board - Hardware

A block diagram of the overall prototyping system used in this work is shown in Figure 3.4. In this figure we show the interconnection of camera, preprocessing section and the host computer that is used for monitoring and post-processing.

Our prototype system [59] consists of a Xilinx FPGA (4000E series), a Motorola microcontroller (MCU-68HC11 series), an EEPROM (for storing bitmap data for the FPGA), and an RS-232 interface (for connecting to the host computer). Figure 3.5 shows the prototype in-camera processor board.

![Block diagram of the overall system](image-url)
Unlike the standard EEPROM/FPGA interfaces, we use the MCU to control the FPGA, rather than directly connecting the EEPROM to the FPGA. This provides much more flexibility for controlling bitmap downloads to the FPGA, and eliminates the need to double port the EEPROM. In order to provide flexible access to the MCU pins, we have elected not to treat the EEPROM as a memory extension, but rather as a peripheral. In this way we do not have to commit limited port resources to dedicated external address/data lines. The trade-off is increased software overhead in accessing the EEPROM, but the extra time required to download data is not critical to the target applications.

3.3.1 New Processing Hardware

An expanded system that enhances the resources available from one FPGA is developed [127]. The system has been augmented by adding a DSP chip (TMS320BC52). Figure 3.6 shows the block diagram of the system. The preprocessed data from the FPGA are stored in a FIFO which buffers the data for further processing by the DSP chip. The new processing hardware is comprised of three PCBs: the FPGA board, the DSP chip board,
and Firewire board for linking to a PC. Other resources are shared between the boards, and they communicate through a custom bus. The complete hardware is shown in Figure 3.7.

![Figure 3.6 Block diagram of the new processing board](image)

**Firewire Technology**

FireWire is a high-speed serial input/output (I/O) technology for connecting peripherals to a computer. Originally developed by Apple, it is now an official industry standard (IEEE 1394) [139]. FireWire is targeted for use with multimedia peripherals such as video camcorders and other high-speed devices such as hard disk drives and printers. It can operate at up to 400 megabits per second. FireWire allows "hot plug" capability, eliminating any need to turn off or restart the computer when attaching a new peripheral.

### 3.4 Lighting and Image Acquisition

Lighting is almost always a critical factor and must be very carefully organized. Typically, the ambient room lighting will be totally inadequate, and even confusing, so that each inspection station will require its own set of dedicated lights, each designed for the task in hand.
Figure 3.7 New Larch/Firewire system mounted on the DALSA camera

Scene and object illumination play a key role in the machine vision process. The central purpose of imposing controlled constant illumination is to visually enhance the parts to be imaged so that their flaws, defects, and features are highlighted and their identification and classification by the visual system becomes somewhat easier. Although the choice of lighting will typically be application-dependent, some general points may be pertinent.
The common incandescent bulb is probably the simplest source of light. It is cost-effective and it is easily adjusted for light intensity; however, it generally provides directional illumination since it is, to an approximation, a point source of light. Hence, incandescent bulbs cast strong shadows which invariably cause problems for machine vision software. Special bulbs are normally required as degradation in emitted light intensity is common with age. Furthermore, incandescent bulbs emit considerable infrared radiation; this does not cause problems for humans as we are not sensitive to such lights but some camera sensors, particularly so-called CCD cameras, are sensitive and visual data can be washed out by the reflected infrared rays.

For most machine vision applications, a diffuse source of light is the most suitable. Diffuse lighting is non-directional and produces a minimum amount of shadow. Fluorescent lighting is the simplest and most common method of obtaining diffuse illumination and is especially good for providing illumination for large areas.

In situations where the only features that need to be inspected are evident from the silhouette of the object, back lighting is the most appropriate. Back lighting, e.g. in the form of a light table, provides high contrast between the object and the background upon which the objects rest. Its advantage is that it facilitates very simple object isolation or segmentation.

As many inspection systems base much of their analysis on the absolute intensity of the incident light, the control of the object illumination can be important. In particular, if image processing and analysis decisions are being made on the basis of a fixed intensity threshold, then some problems will occur if the illumination and hence the reflected light changes. If possible, the vision system should be able to adapt to such changes, although this does not necessarily mean that it should be capable of dealing with dynamic changes. Most illumination systems degrade quite slowly over time and it would be quite satisfactory if the system were capable of self calibration. However, other alternatives exist to this adaptive approach. One solution is to ensure that the illumination does in fact
remain constant by monitoring it using light meters and adjusting the illumination system appropriately.

It was mentioned above that incandescent lighting is not suitable for some cameras and in general, one should ensure that the lighting system is compatible with the image sensor. For example, mains-powered lighting is not suitable due to the a.c characteristics of the mains electricity supply. Humans do not notice this, in general, because they effectively 'integrate' (or average) the incident illumination over a short period of time. Machine vision sensors do not integrate in quite the same way and, when they acquire the image, the flicker can become apparent. The use of an appropriate (d.c.) power supply can alleviate this problem, when it does occur.

The type of defects being inspected conflict in terms of lighting requirements. Detection of highly reflective parts is best achieved when the light source and the camera are such that all the reflected light from the part is directed towards the camera. Dimensional measurements are accurate when the camera and the light source are close to perpendicular to the part so as not to cause occlusion and shadow. Detection of scratches of all orientations is achieved best by a diffused light source that creates a shadow in the scratch region. Therefore, it is clear that any single lighting set up that best detects a particular type of defect may not be optimal for detecting other types of defects. However, it is not practical to inspect the part for different defects separately because of cost considerations.

Figure 3.8 shows the effect of nonuniform lighting of a scanned moving image from the test setup on our prototype system.
3.4.1 Illumination used in the Test Setup

In our test setup, we used a 150W regulated fiber optic illuminator manufactured by Fostec [140] shown in Figure 3.9. Conventional Quartz Halogen fixtures provide high intensity of illumination, but output a significant amount of infrared light in the form of heat. Although illuminators provide an air or convention cooling mechanism for the bulb, direct lighting of an object or scene generally results in heat absorption. Fibre Optic light guides, used to deliver illumination from quartz halogen light sources, solve the heat problem. Optical fibers transmit visible illumination (some also transmit near infrared or near ultraviolet) while rejecting mid- and far-infrared light generally associated with heat. The resulting system yields intense visible illumination, without the heat output that can alter or damage subjects under investigation.

Figure 3.9 Foster optical light source and line light
3.5 Conclusion

Hardware limitations constrains the choice of suitable algorithms for defect detection. We have to choose and develop algorithms based on their potential for identification based on minimal storage of a pixel-serial raster data. This can only be achieved by a thorough understanding of the developed design environment and its limitations. In this chapter we have discussed the test setup and the hardware developed for in-camera processing applications. The DALSA Inc., TDI line scan camera and its signals were also described. Two different technologies of CCD and CMOS in camera sensors were also compared. The issue of lighting and the key role of uniform illumination in defect detection systems was also covered in this chapter.
Chapter 4

Thresholding Techniques

4.1 Introduction

As an introduction to basic types of preprocessing applications that have already been successfully applied to in-camera defect detection, this chapter discusses general thresholding techniques. The work described here presents both prior published and new thresholding algorithms developed as part of the early research work in this project. The preprocessing algorithms developed in this work can all be implemented in the limited resources of one FPGA without any need for 2D image data.

The preprocessing takes on the role of a conservative gross filter. Its objective is to detect all possible defects. The intent here is to provide a reliable means of rapidly identifying suspect regions that may or may not be finally classified as defective. Thresholding is the most widely used algorithm for image segmentation and defect detection. It can be easily implemented in a minimum hardware system but its performance is limited to non-textured backgrounds or in cases where the defects have visibly different grey levels. Zero order background tracking and delta modulation background tracking are the other two techniques that are discussed in this Chapter.
4.2 Thresholding

In many vision applications, it is useful to be able to separate out the regions of the image corresponding to objects in which we are interested, from the regions of the image that correspond to the background. Thresholding often provides an easy and convenient way to perform this segmentation on the basis of the different intensities or colours in the foreground and background regions of an image. In addition, it is often useful to be able to see what areas of an image consist of pixels whose values lie within a specified range, or band of intensities (or colours). Thresholding can also be used for this.

The input to a thresholding operation is typically a greyscale or colour image. In the simplest implementation, the output is a binary image representing the segmentation. Black pixels correspond to background and white pixels correspond to foreground (or vice versa). In simple implementations, the segmentation is determined by a single parameter known as the intensity threshold. In a single pass, each pixel in the image is compared with this threshold. If the pixel's intensity is higher than the threshold, the pixel is set to, say, white in the output. If it is less than the threshold, it is set to black.

In more sophisticated implementations, multiple thresholds can be specified, so that a band of intensity values can be set to white while everything else is set to black. For multi-spectral images, it may be possible to set different thresholds for each region. Another common variant is to set to black all those pixels corresponding to the background, but leave foreground (possible defect) pixels at their original colour/intensity (as opposed to forcing them to white), so that the information is not lost.

Not all images can be neatly segmented into foreground and background using simple thresholding. Whether or not an image can be correctly segmented this way can be determined by looking at an intensity histogram of the image. If it is possible to separate out the foreground of an image on the basis of pixel intensity, then the intensity of pixels within the foreground must be distinctly different from the intensity of pixels within the background. In this case, we expect to see a distinct peak in the histogram corresponding
to defects such that thresholds can be chosen to isolate this peak accordingly. If such a peak does not exist, then it is unlikely that simple thresholding will produce a good segmentation. In this case, adaptive thresholding may be a better answer.

4.3 Adaptive Thresholding

Whereas a conventional thresholding operator uses a global threshold for all pixels, adaptive thresholding changes the threshold dynamically over the image. This more sophisticated version of thresholding can accommodate changing lighting conditions in the image, e.g. those occurring as a result of a strong illumination gradient or shadows.

Adaptive thresholding takes a greyscale or colour image as the input and, in the simplest implementation, outputs a binary image representing the segmentation. For each pixel in the image, a threshold has to be calculated. If the pixel value is below the threshold, it is set to the background value, otherwise it assumes the foreground value. There are two main approaches to finding the threshold: (i) the Chow and Kanenko approach and (ii) local thresholding [40]. The assumption behind both methods is that smaller image regions are more likely to have approximately uniform illumination, thus being more suitable for thresholding. Chow and Kanenko divide an image into an array of overlapping subimages and then find the optimum threshold for each subimage by investigating its histogram. The threshold for each single pixel is found by interpolating the results of the subimages. The drawback of this method is that it is computationally expensive and, therefore, is not appropriate for real-time applications. An alternative approach to finding the local threshold is to statistically examine the intensity values of the local neighborhood of each pixel. The statistic which is most appropriate depends largely on the input image. Simple and fast functions include the mean of the local intensity distribution, the median value, or the mean of the minimum and maximum values. The size of the neighborhood has to be large enough to cover sufficient foreground and background pixels, otherwise a poor threshold is chosen. On the other hand, choosing regions which are too large can violate the assumption of approximately uniform illumination. This method is less computationally intensive than the Chow and Kanenko approach and produces good
results for some applications. In our defect detection application, the target is to apply uniform illumination in order to avoid applying adaptive thresholding as it will increase the complexity of the preprocessing and will require more hardware.

4.4 Two (and multi) level thresholding

In this algorithm any pixel whose grey level is outside of predefined thresholds is considered as belonging to a defect. Figure 4.1 shows the algorithm flowchart for a symmetric 2-value threshold ($\pm T$). A very simple run length encoding can be implemented using the transmission of numbers of pixels, $N_p(i,m)$, that were not previously transmitted. If $N_p(i,m) = 1$, then the previous transmitted pixel can be assumed to belong to a defect. By accumulating the transmitted $N_p(i,m)$, we can also keep track of the column address, $m$. The row address is $i$ and is incremented at the beginning of each line.

![Flowchart for symmetric 2-level thresholding](image)

**Figure 4.1 Flowchart for symmetric 2-level thresholding**
The thresholds can easily be adjusted for different backgrounds. This algorithm is very simple and efficient and needs a minimum number of CLBs but its performance drops off when the background has a solid texture. Very accurate control of the illumination intensity is required for this algorithm to be effective. In our implementation of this design we required FPGA resources of 68 CLBs.

Since the threshold levels are fixed, this technique will not be ideal for defects with slowly changing grey levels. In this situation, only a part of the defect with a greater than threshold grey level will be detected. This problem is illustrated in Figure 4.2. To solve the problem one can reduce the threshold level, but this solution will increase the sensitivity of the system. In this case, small drifts in the grey level, because of the variations in illuminations, will lead to false alarms. One of the possible solutions to this problem is explained in Section 4.5, “Zero-order Background Tracking,” on page 45.

Figure 4.2 Effect of fixed level thresholding on defect detection by ignoring low intensity parts of the defect
Figure 4.3 The result of applying the simple thresholding technique to the defect shown in Figure 4.2

The threshold levels are set using the histogram of the image in order to get the best performance in defect detection. Best performance is obtained when the negative and positive false alarms are minimum. Figure 4.3 shows the result after applying the thresholding technique on the defect shown in Figure 4.2.

4.5 Zero-order Background Tracking

This method is used when we have a slowly varying background. In this algorithm, each pixel is compared with the most recently sampled background pixel value, \( p_B(i) \), and if the difference is greater than the predefined threshold, it is considered to belong to a defect (see Figure 4.4). If this test fails, then this pixel value is assigned to \( p_B(i) \). Using the flow chart of Figure 4.1, we simply change the decision block to \( |p(i, m) - p_B(i)| \leq T \). The algorithm is simulated in Extend, an object oriented simulation environment, and shown in Figure 4.5. This algorithm works perfectly well for some textures and needs a moderate number of CLBs (80% of the available gates in a XC4005E). This algorithm fails to perform well, however, for backgrounds with a solid texture. As with the previous algorithm, very accurate control of the web illumination is required to prevent excessive noise from driving the zero-order tracker to incorrect background values (at which point...
the algorithm will fail completely.) The use of a TDI system is mandatory for this algorithm to be effective.

![Graph showing the relationship between grey level and detector output signal](image)

**Figure 4.4 Slowly changing background and the detector's output signal by applying zero background tracking algorithm**

### 4.6 False alarm rejection by trigger association:

Our investigations show that triggers due to noise are distributed uniformly and at random over the surface, whereas those arising from defects occur locally in clusters. That is, isolated triggers are probably false alarms due to random noise, whereas those forming compact clusters are assumed to be from a defect. This involves requiring that at least \( N \) triggers be generated or all triggers in the region will be discarded. The effectiveness of this scheme for selectively eliminating random noise triggers can be appreciated from the following analysis.
Figure 4.5 Extend simulation environment for zero order background tracking algorithm

The probability \( P(N, M) \) that \( N \) or more triggers are generated in the region due to noise alone is given by the cumulative binomial distribution:

\[
P(N, M) = \sum_{L = N}^{M} \frac{M!}{L!(M-L)!} p^L (1-p)^{(M-L)}
\]  

(4.1)

For typical values e.g. \( M=100, N=10, p=0.001 \), then \( P(10, 100) \) turns out to be about \( 10^{-15} \) which is vanishingly small. Thus if fewer than 10 triggers occur within the ensemble of 100 pixels they are rejected, but ten or more are regarded as being due to a defect. To test the algorithm, a synthesized image with slowly varying background is generated in the Matlab environment. Figure 4.6 shows the generated synthetic texture with the maximum grey level of 130 and minimum grey level of 70. In Figure 4.7 we have added 3 different defects with grey levels of 60, 170, and 200. In the next step we have added salt and pepper noise to the defective sample which is shown in Figure 4.8. Figure
4.9 shows the result after applying zero order background tracking technique using false alarm rejection by trigger association. It shows that salt and pepper noise is not detected as defects as they do not appear in clusters.

Figure 4.6 A synthetic texture with slowly moving background. Maximum grey level is 130 and Minimum grey level is 70

Figure 4.7 The synthesized texture shown in Figure 4.6 with three defects with grey levels of 60, 170, 200.
Figure 4.8 The defective texture shown in Figure 4.7 with added salt and pepper noise.

Figure 4.9 The result after applying the zero order background tracking algorithm with false alarm rejection by trigger association.
4.7 Performance of thresholding techniques in the presence of noise

It is obvious that the presence of noise will degrade the performance of any method that is applied to detect the defects. This effect will be more prominent in thresholding techniques as they are based on pixel grey levels. One solution to this problem is to low pass filter the image before applying any processing algorithm. This approach is suitable for many applications, however it might not be applicable for the cases when the exact pixel levels should be maintained. It might also remove thin vertical defects. In the following section some alternative approaches are proposed.

4.7.1 Two-threshold detector

A simple but very useful improvement to the basic thresholding is shown in Figure 4.10. This exploits the property that some high contrast defects are so small that they appear only in a single isolated pixel. The solution here is to have two thresholds, i.e., effectively two processing channels working in parallel. Triggers generated when the signal excursion is large enough to exceed the outermost threshold are not subjected to the binary filtering, whereas those resulting from crossing only the inner threshold are filtered. The output of the resulting two parallel channels are logically XORed together to provide the detector output. A complete theoretical analysis can be undertaken only with explicit knowledge of the statistical properties of the defect signals in addition to that of the noise.

![Diagram](image_url)  

Figure 4.10 A modified thresholding technique to ignore isolated high contrast pixels as being considered as defect.
4.7.2 Excess over thresholds filter

This approach can be used to eliminate the excessive number of false alarms generated when a sensitive threshold is used to detect low contrast defects. In this technique, pixels are examined in clusters and a threshold \( v_t \) is set for each individual pixel. All the pixels below this threshold generate a zero output following thresholding. For the pixels exceeding this threshold, however, the excess over the threshold is added to a sum for the cluster, denoted \( S \). Only if this sum exceeds a second threshold, \( S_t \), are the triggers retained; in this case, all triggers in the cluster are retained.

A test. Figure 4.11 shows a defective sample scanned by the test setup. Some defects are also manually added. Nonlinear illumination makes a simple thresholding technique inadequate for defect detection. Figure 4.12 and Figure 4.13 show the result of applying a simple thresholding algorithm to detect the defects. The lower threshold level (Threshold=70) in Figure 4.12 causes incomplete detection of defects. In this case some defects with low contrast are ignored. In Figure 4.13, a higher threshold level (Threshold = 100) leads to the false detection of defects in the left hand side of the image. Figure 4.14 shows the result after applying the technique discussed in Section 4.7.2. A cluster of 5 pixels were used in this experiment. The threshold level is 100 and the excess over the threshold i.e., \( S \), is set to 130. These values are obtained by trial and error.
Figure 4.11 A defective sample scanned by the test setup. Nonuniform illumination makes a preprocessing step necessary.
Figure 4.12 The result after applying a simple thresholding technique on the defective sample shown in Figure 4.11 with a threshold of 70.
Figure 4.13 The result after applying the simple thresholding technique on the defective sample shown in Figure 4.11 with a threshold of 100.
Figure 4.14 The result after applying the thresholding technique described in "Excess over thresholds filter" on page 51 to the defective sample shown in Figure 4.11 with a threshold of 100. The excess over threshold level, $S$, is 130.
4.7.3 Delta modulation background tracking

In an attempt to correct the problems with zero-order background tracking, a “delta modulation background tracking algorithm”, originally proposed by G.A. Jullien, is further developed in this work. The idea behind this algorithm is to increase the estimation order with non-linear estimators. Given our hardware limitations, the use of linear estimators is too expensive; therefore this algorithm uses simple nonlinear estimators as replacements. A simple delta-sigma block can be used to implement a nonlinear background filter with as large an ‘order’ as required. The simulation environment of the object oriented simulator (Extend™ using a set of custom scripted simulation objects) is shown in Figure 4.15 for the delta-sigma tracking algorithm.

![Diagram of Delta modulation background tracking](image)

**Figure 4.15 Simulation environment for the delta-sigma tracker**

A generator is used to provide simulated image pixels including a variety of background functions and defect statistics. The image pixel, \( p(n) \), is compared with the output of the
background tracker, \( \langle p(n) \rangle \), and if the pixel is within the defined threshold (the assumption in this case is that the pixel is part of the background) then \( \Delta p = p(n) - \langle p(n) \rangle \) is used to drive the Sigma accumulator; otherwise the previous value of \( \Delta p \) is used. By controlling the binary point in the up/down counter of the Delta Accumulator one can effectively change the ‘order’ of the tracker. As an example, if the binary point is set 3 digits from the LSB, then the background tracker will slew at a ratio of 1:8 given a steadily increasing background level.

Two samples of a plot from a simulation run with a triangular waveform for the background are shown in Figure 4.16. The plot on the left superimposes the raw pixel values, \( p(n) \), with the background tracker output, \( \langle p(n) \rangle \). Both slewing and constant input ‘jitter’ are in evidence on the \( \langle p(n) \rangle \) signal. The plot on the right shows the defect output based on the \( \text{In} \) signal generated by the algorithm. In the FPGA implementation, the \( \text{In} \) signal is used to gate defect pixels into the FIFO. The success of this algorithm, even in the presence of a rapidly varying background, can be clearly seen by the defect output plot on the right.

![Sample plots from a simulation run for the Delta-Sigma tracker](image)

**Figure 4.16 Sample plots from a simulation run for the Delta-Sigma tracker**
4.8 Conclusions

In this chapter we have discussed different thresholding techniques applied to defect detection. The main advantage of these techniques is that they can be implemented in our target setup with its limited processing resources. However, the techniques lack the generality of being applied to 2D textures or to the materials with textured backgrounds. Some modifications have been proposed (and simulated) that reduce the false alarm rate generated from environmental noise.
Chapter 5

Time Series Analysis of Textures

5.1 Introduction

This chapter deals with the application of one dimensional autoregressive (AR) models for defect detection in some textured web materials. The AR method has been used as a powerful technique in speech analysis because of its ability to provide extremely accurate estimates of the speech parameters, and its relative speed of computation [97]. 2-D AR models, on the other hand, have been found to be useful for describing textures and for subsequent recognition and classification [68],[69],[70]. The AR modeling approach expresses the spatial interaction among the neighbors of an observation. This model characterizes the grey level at a pixel as a linear combination of grey levels of finite neighboring pixels. This assumption is valid unless the image is simply random noise. Various forms of the dependence provide different models [71],[72]. We have experimented with the 1D-AR method as a statistical approach to analyze textured background and finally to detect potential defects. This method of analysis in the two dimensional case has been extensively used to study visual textures. In the simplest form, the image is scanned to provide a one dimensional series of grey level fluctuations, which is treated as a one-dimensional stochastic process evolving in “time”.
Alternatively, a point is assumed to depend upon a certain part of its neighborhood and on Gaussian noise. The coefficients of dependence are extracted using time series analysis techniques [97],[64].

5.2 Theory and definitions

In this section, we discuss the rational transfer function modeling of a deterministic and stochastic discrete-time processes. Three different modellings of ARMA, MA, and AR are discussed.

5.2.1 Rational transfer function modeling methods:

Many deterministic and stochastic discrete-time processes encountered in practice are well approximated by a rational transfer function model. In this model, an input deriving sequence \( \{n_n\} \) and the output sequence \( \{s_n\} \) that is to model the data are related by the linear difference equation,

\[
s_n = \sum_{t=0}^{q} b_t n_{n-t} - \sum_{k=1}^{p} a_k s_{n-k}
\]

(5.1)

This most general linear model is termed an ARMA model. The interest in these models stems from their relationship to linear filters with rational transfer functions.

The system function \( H(z) \) between the input \( n_n \) and output \( x_n \) for the ARMA process of (5.1) "" on page 60 is the rational expression in the form of:

\[
H(z) = \frac{B(z)}{A(z)}
\]

(5.2)

where
\[ A(z) = z \text{ transform of AR branch} = \sum_{m=0}^{p} a_m z^{-m} \]
\[ B(z) = z \text{ transform of MA branch} = \sum_{m=0}^{q} b_m z^{-m} \]

If all the \( \{a_n\} \) terms except \( a_0 = 1 \) vanish, then

\[ s_n = \sum_{l=0}^{q} b_l n_{n-l} \quad \text{(5.3)} \]

and the process is strictly a moving average of order \( q \). This model is sometimes termed an all-zero model.

If all the \( \{b_i\} \), except \( b_0 = 1 \) are zero, then

\[ s_n = -\sum_{k=1}^{p} a_k s_{n-k} + n_n \quad \text{(5.4)} \]

and the process is strictly an autoregressive of order \( p \). The process is termed AR in that the sequence \( s_n \) is a linear regression on itself with \( n_n \) representing the error. With this model, the present value of the process is expressed as a weighted sum of past values plus a noise term. This model is sometimes termed an all-pole model.

The Wold decomposition theorem \([94]\) relates the ARMA, MA, and AR models. Basically, the theorem asserts that any stationary ARMA or MA process of finite variance can be represented as a unique AR model of possibly infinite order; likewise, any ARMA or AR process can be represented as a MA process of possibly infinite order. This theorem is important because if we choose the wrong model among the three, we may still obtain a reasonable approximation by an AR model of higher order. Since the estimation of
parameters for an AR model results in linear equations, it has a computational advantage over ARMA and MA parameter estimation techniques.

5.2.2 The AR modeling - linear prediction

A linear predictor, with prediction coefficients, $\alpha_k$, predicts an output, $\tilde{s}(n)$, based on the previous $p$ samples:

$$\tilde{s}(n) = \sum_{k=1}^{p} \alpha_k s(n-k)$$  \hspace{1cm} (5.5)

The system function of a $p^{th}$ order linear predictor is the polynomial:

$$P(z) = \sum_{k=1}^{p} \alpha_k z^{-k}$$  \hspace{1cm} (5.6)

The basic problem is to determine the set of predictor coefficients, $\{\alpha_k\}$, directly from the signal in such a manner as to minimize some function of the error:

$$e(n) = s(n) - \tilde{s}(n) = s(n) - \sum_{k=1}^{p} \alpha_k s(n-k)$$  \hspace{1cm} (5.7)

$\{\alpha_1, \alpha_2, \ldots, \alpha_p\}$ are normally chosen to minimize the average squared prediction error:

$$E_n = \sum_m e_n^2(m) = \sum_m (s_n(m) - \tilde{s}_n(m))^2$$  \hspace{1cm} (5.8)

where $s_n(m) = s(m + n)$.

We can find the values of $\alpha_k$ that minimizes $E_n$ in Eqn. (5.8) by setting $\partial E_n/\partial \alpha_i = 0, i = 1, 2, \ldots, p$, thereby obtaining the equations.
\[
\sum_{k=1}^{p} \hat{\alpha}_k \sum_m s_n(m-i)s_n(m) = \sum_m s_n(m-i)s_n(m) \quad 1 \leq i \leq p \tag{5.9}
\]

where \(\hat{\alpha}_k\) are the values of \(\alpha_k\) that minimize \(E_n\). Since the \(\hat{\alpha}_k\) are unique, we will drop the caret and use the notation \(\alpha_k\) to denote the values that minimize \(E_n\).

For a short-time analysis procedure, the summation limits in Eqn. (5.8) must be over a finite interval. There are two basic approaches for limiting the summation, the autocorrelation method and the covariance method [64].

If we define

\[
R_n(i, k) = \sum_m s_n(m-i)s_n(m-k) \tag{5.10}
\]

then Eqn. (5.9) can be written as:

\[
\sum_{k=1}^{p} \alpha_k R_n(i, k) = R_n(i) \quad i = 1, 2, \ldots, p \tag{5.11}
\]

This set of \(p\) equations in \(p\) unknowns can be solved in an efficient manner for the unknown predictor coefficients \(\{\alpha_k\}\) that minimize the average squared prediction error for the segment \(s_n(m)\). Using Eqn. (5.8) and Eqn. (5.9), the minimum mean squared prediction error can be shown to be:

\[
E_n = \sum_m s_n^2(m) - \sum_{k=1}^{p} \alpha_k \sum_m s_n(m-i)s_n(m) \tag{5.12}
\]

and using Eqn. (5.11) we can express \(E_n\) as

\[
E_n = R_n(0) - \sum_{k=1}^{p} \alpha_k R_n(k) \tag{5.13}
\]
Thus the total minimum error consists of a fixed component, and a component which depends on the predictor coefficients.

To solve for the optimum predictor coefficients, we must first compute the quantities $R_n(i, k)$ for $1 \leq i \leq p$ and $0 \leq k \leq p$. Once this is done we only have to solve Eqn. (5.11) to obtain the $\alpha_k$.

The set of equations given by Eqn. (5.11) can be expressed in matrix form as:

$$
\begin{bmatrix}
R_n(0) & R_n(1) & R_n(2) & \ldots & R_n(p-1) \\
R_n(1) & R_n(0) & R_n(1) & \ldots & R_n(p-2) \\
R_n(2) & R_n(1) & R_n(0) & \ldots & R_n(p-3) \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
R_n(p-1) & R_n(p-2) & R_n(p-3) & \ldots & R_n(0)
\end{bmatrix}
\begin{bmatrix}
\alpha_1 \\
\alpha_2 \\
\alpha_3 \\
\vdots \\
\alpha_p
\end{bmatrix}
= 
\begin{bmatrix}
R_n(1) \\
R_n(2) \\
R_n(3) \\
\vdots \\
R_n(p)
\end{bmatrix}
\tag{5.14}
$$

The relationship between the AR parameters and the autocorrelation function shown in Eqn. (5.14) is known as the Yule-Walker equations. Note that the above autocorrelation matrix, $R_{xx}$, is Hermitian ($R_{xx}^H = R_{xx}$) and Toeplitz; i.e., it is symmetric and all the elements along a given diagonal are equal. This is an important property which makes the algorithm invariant to the starting point for prediction in a textured material. Any shift in the texture between the defect free sample that is used off line to obtain the AR coefficients and the texture of the production material will have no effect on this algorithm and there is no need to align the texture for comparison as is needed in the template matching method.

### 5.2.3 Model Order Selection

Since the best choice of filter order $p$ is not generally known apriori, it is usually necessary in practice to postulate several model orders. Based on these, one then computes some error criterion that indicates which model order to choose. Too low a guess for
model order results in a highly smoothed estimate. Too high an order introduces spurious
detail into the prediction. One intuitive approach is to construct AR models of increasing
order until the computed prediction error power reaches a minimum. As long as the filter
coefficient is nonzero, the prediction error power decreases. Thus the prediction error
power alone is not sufficient to indicate when to terminate the search.

Several criteria have been introduced as objective bases for the selection of the AR model
order. Akaike [96] has provided two criteria. His first criterion is the final prediction error
(FPE) where the order of the AR process is selected so that the average error for a one step
prediction is minimized. Akaike suggested a second-order selection criterion using a
maximum likelihood approach to derive a criterion termed the Akaike information
criterion (AIC). The AIC determines the model order by minimizing an information
theoretic function. Another method was proposed by Parzen [95] and is termed the
criterion autoregressive transfer (CAT) function. The order \( p \) is selected to be the one in
which the estimate of the difference of the mean-square errors between the true prediction
error filter and the estimated filter is minimum.

5.3 Implementation of the Algorithm

The AR coefficients are obtained off line and from a defect free sample (golden sample).
The defect free sample should be a good representative of the entire texture. A texture
might have more than one set of coefficients based on its complexity. Complicated
textures can be divided into smaller subregions where each region is modeled by its
relevant AR coefficients. It is obvious that this approach is computationally expensive, and
it is not implementable with the target architecture. In each line of the image, the grey
level of the next pixel is predicted by the \( n-1 \) previous pixel values; clearly these pixels are
assumed to be defect free. In order to remove this assumption the detection mechanism
can be modified based on the fusion of several different techniques where the first \( n - 1 \)
pixels are tested with one or more other algorithms. The predicted value is compared with
the next pixel scanned by the camera. This comparison is performed by the calculation of
a normalized error as follows:
\[ \hat{e}(n) = \frac{s(n) - \hat{s}(n)}{s(n)} \]  

(5.15)

where \( s(n) \) and \( \hat{s}(n) \) are the pixel value from the camera and predicted pixel value respectively.

If the absolute value of the normalized error \( \hat{e}(n) \), is above the predefined threshold level, that pixel is considered to belong to a defect and will be replaced with the predicted value; this predicted value will also be used in the prediction of the next pixel. If the error is negligible, the pixel value from the camera will be used for the next prediction. Experimental results show that the small error in the predicted value might accumulate and propagate and will lead to a false alarm for defects with larger horizontal size. This problem can be alleviated by finding the best order for the model. To overcome this problem, the histogram of the defect-free texture is obtained off line and the set of existing pixel levels are stored in a memory (look-up table). For defect pixels, the predicted values are compared with the ones stored in the memory and the closest one is used to replace the defect pixel. This solution does not seem to be practical (or, perhaps, impossible) for textures having continuous grey level pixels. The size of look-up table will increase exponentially, and finding the closest pixel value from the look-up table to the predicted value might even increase the error. Figure 5.1 illustrates this problem by showing a stream of data with an added defect of 30 pixels and its relevant accumulating prediction error (up to about 7% at the end of the defect data stream). In this example the data is AR modeled with an order of 7.
Figure 5.1 A stream of data with added defect and accumulating prediction error

Figure 5.2 shows a textured pattern with an added defect of size $50 \times 15$ pixels. The AR method is applied to Figure 5.2 with $n=7$ and threshold=0.2. The result that shows the accumulative prediction error is shown in Figure 5.3.

Figure 5.2 A textured pattern with an added defect

Figure 5.3 The output of the AR method applied to Figure 5.2 with $n=7$ and threshold=0.2 showing the effect of accumulative prediction error.
Figure 5.4 shows the output of the AR method applied to Figure 5.2 with $n=5$ and threshold=$0.2$. It shows the fact that lowering the order of the AR filter will reduce its sensitivity to detect the full defect; the portion of the defect that has a grey level close to the background will not be detected.

Figure 5.4 The output of the AR method applied to Figure 5.2 with $n=5$ and threshold=$0.2$

5.3.1 The 1-D AR algorithm

1. The AR coefficients are obtained off-line for non-defective sample patterns (our experiments used a Matlab program.) The existing grey levels are obtained from the histogram of the texture and are stored in memory.

2. The grey level of each pixel in a line of the texture is predicted by the 1-D AR coefficients.

3. A normalized error is calculated for each pixel using Eqn. (5.15).

4. If $|\hat{e}(n)|$ is greater than a predefined threshold, that pixel will be considered to belong to a defect. If this is the case, then the pixel value is replaced with the closest value stored in memory. Otherwise the pixel value from the camera is retained.

5.3.2 Simulation experiments

Matlab was used to generate several sample patterns to test the algorithm, and several synthesized images are shown in Figure 5.5-5.8. Although the generated patterns are 2-D patterns in nature, they can be represented by 1-D AR modeling. Simulations show that a 7th order AR filter is suitable for many textured backgrounds. 1-D AR method for defect detection might also be applied to a subset of 2-D patterns. In this method, the 2-D pattern is partitioned into several sub regions and 1-D AR coefficients are obtained for each
individual region. This approach will cause synchronization problems as the system should switch from one filter to the appropriate one when we move between regions.

Figure 5.5 Horizontal bars

Figure 5.6 Vertical bars

Figure 5.7 Diagonal bars

Figure 5.8 A Gradient pattern

The algorithm has been successfully applied to detect defects in synthesized images and some real defective samples. Different sizes of defects with different grey levels were added to the synthesized images. Figure 5.9 shows a synthesized pattern with an added defect. The grey level of the defect has been intentionally set close to the grey level of the texture to test the robustness of the proposed algorithm.
Figure 5.9 A synthesized pattern with an added defect

![Graph showing grey level vs. pixel numbers]

Figure 5.10 Grey level of the defect portion from Figure 5.9

Figure 5.10 shows the grey level of the defect portion in the sample image and the magnitude of the relative error is shown in Figure 5.11. It shows that the magnitude of the error jumps up for the defective portion of the texture. Figure 5.12 shows the final output of the AR method which only contains the defect pixel values.

**Error**
Figure 5.11 Magnitude of the relative error

Figure 5.12 The final output of the AR method applied to Figure 5.9 with $n=7$ and threshold=0.3

Figure 5.13 shows a synthesized pattern with three defects. The defects are in different sizes and grey levels (close to the grey levels of the background) to test the robustness of the proposed method. Figure 5.14 shows the output of the AR method with $n=7$ and a threshold of 0.3 without using any lookup table to remove the effect of accumulative error. Figure 5.15 illustrates the effect of increasing the threshold on the detection of the defects. It shows that increasing the threshold level will reduce the sensitivity of the algorithm to detect the defects whose grey levels are very close to the grey level of the background.

Figure 5.13 A synthesized pattern with three defects in different sizes and grey levels.
The AR method performs as a thresholder on patterns, similar to the one shown in Figure 5.16, because the background is quite constant and the defects have significantly different grey levels.

One of the advantages of the AR method is that it is not sensitive to a shift of the pattern. This means that after extracting the coefficients, the estimation can start from any pixel in the line.
5.3.3 Performance of the algorithm in a noisy environment

Noise can degrade the performance of the AR algorithm and consequently might lead to false alarms. Noise can be categorized into two kinds: 1- small variations in the grey level of the texture that usually span a wide scanning area, and usually caused by small variations in illumination; 2- salt and pepper noise that represent itself as spikes in the grey level and affects the area of relatively few pixels. The first kind of noise can be easily removed without increasing the complexity of the hardware. The problem can be solved by increasing the threshold level so as to reduce the sensitivity of the system. Since the normalized error is used as a decision criterion, the effect of this kind of noise is mitigated. Salt and pepper noise can be filtered out by using a counter as a post processor. The counter will count the number of consecutive defects and trigger when the defect size is greater than its threshold value. This will ignore small defect-like spots with the maximum size of, say, three pixels that are most likely salt and pepper noise or small defects that can be ignored and so should not be considered as a defect. This idea has been experimentally tested on the defect sample shown in Figure 5.18 (i.e., a 'noisy' version of Figure 5.17). A counter that is set to trigger for defects larger than three bits (horizontally) is used to remove salt and pepper noise. The output of this process is shown in Figure 5.19. Any other preprocessing approach, such as Kalman filtering, will increase the complexity of the hardware and will also reduce the resolution of the texture.

![Figure 5.17 A defect sample with almost uniform background](image)
Figure 5.18 Defective sample shown in Figure 5.17 with added salt and pepper noise

Figure 5.19 The output of the AR method for the noisy defective sample of Figure 5.18 using a counter with the threshold of three

5.3.4 Two methods of applying the algorithm

As we have already discussed, in predicting the next pixel in the data stream, it is assumed that the first \( n - 1 \) pixels are defect free, where \( n \) is the order of the AR method. This assumption may not always be true. For some textures, such as the samples shown in Figure 5.6, Figure 5.7, and Figure 5.16, we can use the \( n - 1 \) pixel data from the end of each line to predict the first pixel in the next coming line. In this case we have only to assume that the first \( n - 1 \) starting pixels are defect free. This method of prediction is shown in Figure 5.20. In some other textures, such as the sample shown in Figure 5.5, the grey levels in each line can not be considered as a continuation of the grey level of the previous line. In such textures we have to assume that the first \( n - 1 \) pixels at the beginning of each line are defect free. This assumption is shown in Figure 5.21. Obviously this is not a safe assumption and a fusion of different techniques should be used to detect the defects.
Figure 5.20 $n - 1$ pixel from the end of each line is used to predict the first pixel in the next coming line.

Figure 5.21 It is assumed that the $n - 1$ pixels at the beginning of each line are defect free to start the process of the prediction.

5.4 Two dimensional AR modeling of the texture

2-D AR models have been found to be useful for describing textures and for subsequent recognition and classification. This approach explicitly expresses the spatial interaction among the neighbors of an observation. In this section, an extension of the AR model...
concept from one-dimensional time series to two-dimensional spatial series is made. A pixel \( x(n_1 - n_2) \) is represented as a weighted summation of it is \( p_1 \times p_2 \) neighbouring elements [72]:

\[
x(n_1, n_2) = b_{00}e(n_1, n_2) - \sum_{k=0}^{p_1} \sum_{m=0}^{p_2} a_{km}x(n_1 - k, n_2 - m) \quad (5.16)
\]

where the input noise \( \{ e(n_1 - n_2) \} \) is composed of independent random variables with zero mean and variance \( \sigma^2 \).

\( x(n_1 - n_2) \) can be estimated as a linear weighted summation of the specific previous \( \sum_{k=0}^{p_1} \sum_{m=0}^{p_2} a_{km}x(n_1 - k, n_2 - m) \quad (5.17) \)

The prediction error, as it is usually referred to in time series analysis, is given by

\[
e(n_1, n_2) = x(n_1, n_2) - \hat{x}(n_1, n_2) = x(n_1, n_2) + \sum_{k=0}^{p_1} \sum_{m=0}^{p_2} a_{km}x(n_1 - k, n_2 - m) \quad (5.18)
\]

Our objective is to select the model coefficients \( a_{km} \) so as to minimize the following quadratic function of prediction error elements:

\[
f(a) = \sum_{n_1 = p_1 + 1}^{N_1} \sum_{n_2 = p_2 + 1}^{N_2} w(n_1, n_2)|e(n_1, n_2)|^2 \quad (5.19)
\]
where $w(n_1, n_2)$ are non-negative weights. As in the one-dimensional case, the minimization of (5.19) is carried out by using standard calculus and results in the following relationship.

$$
\sum_{k=0}^{p_1} \sum_{m=0}^{p_2} a_{km} Q(k, m, i, j) = -Q(0, 0, i, j)
$$

(5.20)

$k = m \neq 0$ simultaneously

\[i = 0, \ldots, p_1\]
\[j = 0, \ldots, p_2\]
\[i = j \neq 0 \text{ simultaneously}\]

where $Q(k, m, i, j)$ are defined by

$$
Q(k, m, i, j) = \sum_{n_1=0}^{N_1} \sum_{n_2=0}^{N_2} w(n_1, n_2) \cdot x(n_1-k, n_2-m) x^*(n_1-i, n_2-j)
$$

(5.21)

5.5 Hardware implementation of the 1-D AR Algorithm

The main advantage of the 1-D AR algorithm is that it can be implemented with an efficient processing structure which makes it possible to develop the entire system on our target in-camera architecture. Figure 5.22 shows the simplified signal flow of the 1-D AR algorithm. The predictor is an IIR filter, that is the inverse filter for the prediction error filter defined by Eqn. (5.7). The order of the IIR filter is equal to the order of the AR predictor. The output of the AR predictor is compared with the relevant pixel value from the camera and if the error is above the threshold level, that pixel is considered as belonging to a defect. Defect pixels coming out of the camera are replaced by a histogram look-up and are sent to the AR predictor through a multiplexer that is controlled by the error signal. This algorithm is more complex than can be implemented within one FPGA.
specially if we add the hardware of a look-up table to remove accumulative prediction error and a counter to ignore salt and pepper noise to the system. The algorithm can be implemented using a DSP chip in the new Hardware system.

![Diagram](image)

Figure 5.22 Simplified signal flow diagram of the 1-D AR Algorithm.

5.6 Conclusions

In this chapter we have introduced one dimensional AR modeling of background patterns for defect detection in some textured materials. The proposed algorithm is simulated on several synthesized textures with added simulated defects and some defect samples taken from production web systems. The effect of accumultating error on the prediction value is investigated and a look-up table, extracted from the histogram of the texture, is used to remove this error. We have also suggested a simple counter technique to ignore spot like defects, and the extension of the algorithm to two dimensions was also discussed.

---

1. We consider that the XC4010 is the densest FPGA that can be used.
Chapter 6

Fuzzy Logic in Defect Detection Systems

6.1 Introduction


The abstract idea presented was: if people can make decisions and act with high precision while the information about their environment is not precise and not numerical, then it is possible to control processes using the same approach, i.e., using imprecise data but accomplish precise results.

These ideas initially were not taken seriously, being met with reluctance and skepticism. It was only in recent years that this
general attitude has began to change, especially since a relatively large number of applications have been implemented successfully in Japan [89] "Sugeno, M., Industrial applications of fuzzy control. Elsevier Science Pub. Co., 1985." on page 131.

At the application level, fuzzy logic is just a method to use common knowledge. The idea of fuzzy sets aligns with the way humans think about things and the way they perceive them. For example, for the parameter “temperature”, it is difficult to define linguistic terms “warm” or “hot”. The cutoff points are not well specified and are very subjective. In fuzzy logic, the linguistic terms “warm” and “hot” will be associated with fuzzy sets.

Fuzzy logic has now found its way into everyday items such as household items and appliances. For instance, Hitachi and Matsushita [81] "D.G. Schwartz, and G.J. Klir, “Fuzzy logic flowers in Japan,” IEEE Spectrum, pp 32-35, July, 1992" on page 131 are making (so-called) “intelligent” washing machines. The first application of fuzzy logic in consumer electronics was marked by a project that started around 1989 by Toshinohu Haruki and Kenichi Kikuchi of Sanyo Electronics Co., Ltd. [82]. Fuzzy logic has also found its way into computer technology; for example, a number of companies are using fuzzy controllers for rigid disk drives where fuzzy logic technology has been used to reduce the disk drive seek time [78].

6.2 An Introduction to Fuzzy Modeling

Fuzzy sets are a generalization of conventional set theory, one of the basic structures underlying computational mathematics and models. Fuzzy models belong whenever they can provide either collateral or competitively better information about a physical process. No one will disagree that the binomial distribution is an optimal model, in some intuitive sense, for the flip of a fair coin. One could model this with a fuzzy technique, but the results would almost certainly be less satisfying in terms of a natural and verifiable representation of the process itself. On the other hand, while we could certainly represent the idea of “pretty close to 7” with a statistical model, it is easy to see that this is much less satisfactory than fuzzy models, simply from the intuitive view, because the notion of
chance is absent from our naive description of the process. From a different point of view, because every hard set is fuzzy, but not the converse, the mathematical embedding of conventional set theory into fuzzy sets is as natural as the idea of embedding the real numbers into the complex plane. In both cases we can expect the larger "space" to contain answers to real questions that can not be found in the smaller one. Thus the idea of fuzziness is one of enrichment, not a replacement.

There are many real-world problems in which one or more of the basic assumptions which are implicit in the classical probability theory are violated. "What is the probability that it will be a warm day tomorrow?" In this instance, the event may be characterized as a fuzzy subset, $A$, of the sample space $\Omega$, with $\mu_A$, the membership function of $A$, being a measurable function. Second even if $A$ is a well-defined nonfuzzy event, its probability, $P(A)$, may be ill-defined. For example, in response to the question, "What is the probability that, in word material, the next line after a defective line be defective?" it would patently be unreasonable to give an unequivocal number answer, e.g., 0.7. In this instance, a vague response like "quite probable," would be much more commensurate with our lack of understanding of the dynamics of defects.

In this section we provide a brief introduction to fuzzy systems modeling, more details can be found in [90]. Consider a system or relationship $U = f(V, W)$, where $U$ is the output (or consequent) variable and $V$ and $W$ are the input (or antecedent) variables. In fuzzy systems modeling we represent this relationship by a collection, $R$, of fuzzy if then rules of the form:

$$\text{If } V \text{ is } A_i \text{ and } W \text{ is } B_i \text{ then } U \text{ is } D_i$$

The $A_i$'s, $B_i$'s and $D_i$'s are normal fuzzy subsets over the spaces $X$, $Y$, and $Z$, which are usually subsets of the real line. In using fuzzy systems modeling we are essentially
partitioning the input space $X \times Y$ into fuzzy regions $A_i \times B_i$ in which we know the output value, $D_i$.

The support of a fuzzy set $A$ is the crisp set of all points $x$ in $U$ such that $\mu A(x) > 0$. For example, the support of the fuzzy set "bright" in an image can be defined as $x \in [150, 255]$. The element $x$ in $U$ at which $\mu A(x) = 0.5$ is called the crossover point. A fuzzy set whose support is a single point in $U$ with $\mu A(x) = 1$ is called a fuzzy singleton. Figure 6.1 "Membership curve for brightness of a pixel for a grey level image" on page 82 illustrates these points;

![Graph showing membership curve for brightness]

**Figure 6.1 Membership curve for brightness of a pixel for a grey level image**

A membership function (MF) is a curve that defines how each point in the input space is mapped to a membership value (or degree of membership) between 0 and 1. The input space is sometimes referred to as the universe of discourse. Trapezoidal and triangular are the simplest membership functions that are formed using straight lines (Figure 6.2 "Trapezoidal and triangular membership functions" on page 83).
Figure 6.2 Trapezoidal and triangular membership functions

Any curve can be used as a membership function. The most generally used membership functions are Gaussian curve, bell curve, and sigmoidal functions, as shown in Figure 6.3 "Commonly used membership functions" on page 83.

Figure 6.3 Commonly used membership functions

Membership values determine how much fuzziness a fuzzy set contains. For fuzzy sets, quantification of the amount of imprecision (fuzziness) captured depends on the extent to which the supporting objects (as individuals or in a group) do or do not possess the concept or property represented by the fuzzy set. The higher the extent, the lower is the fuzziness of the set.
This question of amount of fuzziness is related to similar questions in information theory such as, "how much information is contained in a particular message"? Many measures of fuzziness have been proposed. Some of these are based on notions such as the distance between a fuzzy set and its nearest crisp set. Usually, these measures have properties such as being minimum when \( m \) (the degree of membership) is crisp for \( x \) and maximum when \( m(x) = 0.5 \) for all \( x \). If we sharpen a fuzzy set, that is, increase its contrast around the value 0.5 (or make the membership function steeper in this neighborhood), then its fuzziness will decrease.

### 6.2.1 Set Theoretic Operations:

The most important thing to realize about fuzzy logical reasoning is the fact that it is a superset of standard Boolean logic. We expect that we can do analogous things with fuzzy sets. The elementary crisp-set operations of union, intersection, and complement are briefly reviewed here.

Let \( A \) and \( B \) be two subsets of \( U \). The union of \( A \) and \( B \), denoted \( A \cup B \), contains all of the elements in either \( A \) or \( B \), i.e., \( \mu A \cup B(x) = 1 \) if \( x \in A \) or \( x \in B \) and \( \mu A \cup B(x) = 0 \) if \( x \notin A \) and \( x \notin B \). The intersection of \( A \) and \( B \), denoted \( A \cap B \), contains all of the elements that are simultaneously in \( A \) and \( B \), i.e., \( \mu A \cap B(x) = 1 \) if \( x \in A \) and \( x \in B \), and \( \mu A \cap B(x) = 0 \) if \( x \notin A \) or \( x \notin B \). Let \( \overline{A} \) denote the complement of \( A \); it contains all the elements that are not in \( A \), i.e., \( \mu \overline{A}(x) = 1 \) if \( x \notin A \) and \( \mu \overline{A}(x) = 0 \) if \( x \in A \).

In Fuzzy logic, union, intersection and complement are defined in terms of their membership functions. One definition for them can be as follows:

\[
\mu A \cup B(x) = \max[\mu A(x), \mu B(x)]
\]
\[ \mu A \cap B(x) = \min[\mu A(x), \mu B(x)] \]

\[ \mu \bar{A}(x) = 1 - \mu A(x) \]

"Max" and "min" are not the only ones that could have been chosen to model fuzzy union and fuzzy intersection. G.J Klir and T.A. Folger in their book "Fuzzy Sets, Uncertainty, and Information" describe other ways to characterize these operations [84]. Zadeh in his pioneering first paper [73] "L.A. Zadeh, "Fuzzy sets," Information and Control, Vol. 8, pp. 338-352, 1965." on page 130 defined two operators each for fuzzy union and fuzzy intersection, namely: fuzzy union as maximum and algebraic sum:

\[ \mu A \cup B(x) = \mu A(x) + \mu B(x) - \mu A(x)\mu B(x) \]

fuzzy intersection as minimum and algebraic product:

\[ \mu A \cap B(x) = \mu A(x)\mu B(x) \]

Later other operators, which have an axiomatic basis, were introduced. \( t\)-conorm for fuzzy union also known as \( s\)-norm, and denoted \( \oplus \), and \( t\)-norm operators for fuzzy intersection (denoted \( \ast \)). Some examples of \( t\)-conorms are:

\textit{bounded sum} defined as:

\[ x \oplus y = \min(1, x + y) \]

and \textit{drastic sum} defined as:
$$x \oplus y = \begin{cases} 
    x & \text{if } y = 0 \\
    y & \text{if } x = 0 \\
    1 & \text{if } x, y > 0
\end{cases}$$

Some examples of t-norm are: bounded product defined as:

$$x \otimes y = \max[0, x + y - 1]$$

and drastic product defined as:

$$x \otimes y = \begin{cases} 
    x & \text{if } y = 1 \\
    y & \text{if } x = 1 \\
    0 & \text{if } x, y < 0
\end{cases}$$

The different t-norms, t-conorms and complements, that are available in fuzzy set theory, provide us with a plethora of richness and also with some (tough) choices that will have to be made in our fuzzy logic system. Most engineering applications of fuzzy sets use: 1) the min or algebraic product t-norm for fuzzy intersection; 2) the max t-conorm for fuzzy union; and 3) $1 - \mu_A(x)$ for the membership function of the fuzzy complement.

### 6.3 Fuzzy Logic System: A High-Level Introduction

Figure 6.4 "Fuzzy logic system" on page 87 depicts a FLS that is widely used in fuzzy logic controllers and signal processing applications.
Figure 6.4 Fuzzy logic system

A FLS maps crisp inputs into crisp outputs. It contain four components: rules, fuzzifier, inference engine, and defuzzifier. Once the rules have been established, a FLS can be viewed as a mapping from crisp inputs to crisp outputs and this mapping can be expressed quantitatively as \( y = f(x) \). Rules may be provided by experts or can be extracted from numerical data. In either case, engineering rules are expressed as a collection of IF-THEN statements, e.g., "IF \( u_1 \) is very warm and \( u_2 \) is quite low, THEN turn \( v \) somewhat to the right." To define a fuzzy logic system one needs an understanding of: 1) linguistic variables versus numerical values of a variable (e.g. very warm versus 36°C); 2) quantifying linguistic variables (e.g., \( u_1 \) may have a finite number of linguistic terms associated with it, ranging from extremely hot to extremely cold), which is done using fuzzy membership functions; 3) logical connections of linguistic variables (e.g., "and," "or," etc.); and 4) implications, i.e., "IF A THEN B."

The fuzzifier maps crisp numbers into fuzzy sets. It is needed in order to activate rules which are in terms of linguistic variables and have fuzzy sets associated with them. The most widely used fuzzifier is the singleton fuzzifier, i.e., \( A \) is a fuzzy singleton with support \( X' \) and \( \mu_A(X') = 1 \) for \( X = X' \) and \( \mu_A(X') = 0 \) for \( X \neq X' \). Singleton fuzzification may not always be adequate, especially when data is corrupted by measurement noise. Nonsingleton fuzzification provides a means for handling such uncertainties totally within the framework of FLS's. A nonsingleton fuzzifier is one for
which $\mu A(X') = 1$ and $\mu A(X')$ decreases from unity as $X$ moves away from $X'$. In nonsingleton fuzzification, $X'$ is mapped into a fuzzy number, i.e., a fuzzy membership function is associated with it. Examples of such membership functions are the Gaussian and triangular. The broader these functions are, the greater is the uncertainty about $X'$.

The inference engine of the FLS maps fuzzy sets into fuzzy sets. In the fuzzy inference engine, which is labeled as inference, fuzzy logic principles are used to combine fuzzy IF-THEN rules. Fuzzy rules map fuzzy input sets in $U = U_1 \times U_2 \times \ldots \times U_p$ to fuzzy output sets in $V$. Each rule is interpreted as a fuzzy implication. We treat the fuzzy inference engine as a system, one that maps fuzzy sets into fuzzy sets by means of $\mu A \rightarrow B(X, y)$. It handles the way in which the rules are combined. Just as we humans use many different types of inferential procedures to help us understand things or to make decisions, there are many different fuzzy logic inferential procedures. Only a very small number of them are actually being used in engineering applications of FL.

In many applications, crisp numbers must be obtained at the output of a FLS. The defuzzifier maps output sets into crisp numbers. In a signal processing applications, such a number may correspond to the location of a target or in our application possibility of a defect.

Defuzzification produces a crisp output for our FLS from the fuzzy set that is the output of the inference block in Figure 6.4. Many defuzzifiers have been proposed in the literature; however, there is no scientific bases for any of them (i.e., no defuzzifier has been derived from a first principle, such as maximization of fuzzy information or entropy); consequently, defuzzification is an art rather than a science. Because we are interested in engineering applications of FL, one criterion for the choice of a defuzzifier is computational simplicity. Maximum Defuzzifier, Mean of Maximum Defuzzifier, Centroid Defuzzifier, Height Defuzzifier, and Modified Height Defuzzifier are the candidate defuzzifiers that meet the above mentioned criterion. We will use the Modified Height Defuzzifier in our experiments; here we give a brief explanation.
**Modified Height Defuzzifier:** Let $y^{-1}$ denote the center of gravity of the fuzzy set $B^i$. The modified height defuzzifier first evaluates $\mu B^i(y)$ at $y^{-1}$ and then computes the output of the FLS as:

$$y_{mh} = \left[ \sum_{i=1}^{M} y^{-1} \mu B^i(y^{-1}) \delta^i \right] / \left[ \sum_{i=1}^{M} \mu B^i(y^{-1}) \delta^i \right]$$

where $\delta^i$ is a measure of the spread of the consequent for rule $R^{(i)}$. For triangular and trapezoidal membership functions, $\delta^i$ could be the support of the triangle or trapezoid, whereas, for Gaussian membership functions, $\delta^i$ could be its standard deviation. The modified height defuzzifier is also easy to use, although the $\delta^i$ parameters must be specified as well as $y^{-1}$ and $\mu B^i(y^{-1})$. The interested reader can refer to [85], [86], [87] for additional discussions on defuzzifiers.

There are several different possibilities to choose from in the four elements in a FLS. We must decide on the type of fuzzification (singleton or nonsingleton), functional forms for membership functions (triangular, trapezoidal, Gaussian, piecewise linear), parameters of membership functions (fixed ahead of time, tuned during a training procedure), composition (max-min, max-product), inference (minimum, product), and defuzzifier (centroid, height, modified height). Just choosing among the paranthetical possibilities leads to $2^{15}$ different FLS’s. This demonstrates the richness of FLS’s and that there is no such thing as “the FLS”.

### 6.3.1 Mamdani and Sugeno’s inference engines:

"Sugeno, M., Industrial applications of fuzzy control. Elsevier Science Pub. Co., 1985." on page 131 are used in fuzzy inference engines. These two methods are similar in many respects. In fact the first two parts of the fuzzy inference process, fuzzifying the inputs and applying the fuzzy operator, are exactly the same. The main difference between Mamdani-type of fuzzy inference and Sugeno-type is that the output membership functions are only linear or constant for Sugeno-type fuzzy inference.

A typical fuzzy rule in a zero-order Sugeno fuzzy model has the form:

\[ \text{If } x \text{ is } A \text{ and } y \text{ is } B \text{ then } z = k \]

where \( A \) and \( B \) are fuzzy sets in the antecedent, while \( k \) is a crisply defined constant in the consequent. When the output of each rule is a constant like this, the similarity with Mamdani’s method is striking. The only distinctions are the fact that all output membership functions are singleton spikes, and the implication and aggregation methods are fixed and can not be edited. The implication method is simply multiplication, and the aggregation operator just includes all of the singletons.

### 6.3.2 Some Terminology on Rules:

1) Incomplete IF Rules: Suppose that we have created a rule base where there are \( p \) inputs, e.g., IF \( u_1 \) is \( F_1 \) and \( u_2 \) is \( F_2 \) and... \( u_m \) is \( F_m \), THEN \( v \) is \( G \). Such rules are called incomplete IF rules, and apply regardless of \( u_{m+1}, \ldots, u_p \). They can be put into the format of the complete IF rule by treating the unnamed antecedents, e.g., \( u_{m+1}, \ldots, u_p \), as elements of fuzzy set INCOMPLETE (IN for short) where, by definition \( \mu_{\text{IN}}(u) = 1 \) for all \( u \in R \).

2) Mixed Rules: Not all rules use the “and” connective; some use the “or” connective, and some use a mixture of both connectives. Such rules are called mixed rules. These rules can
be decomposed into a collection of equivalent rules, using standard techniques from crisp logic. Suppose, for example, we have the rule: IF $u_1$ is $F_1$ and $u_2$ is $F_2$ and... $u_m$ is $F_m$, or $u_{m+1}$ is $F_{m+1}$ and...and $u_p$ is $F_p$ THEN $v$ is $G$. This rule can be expressed as the following two rules: $R^{(1)}$: IF $u_1$ is $F_1$ and $u_2$ is $F_2$ and... $u_m$ is $F_m$ THEN $v$ is $G$; and, $R^{(2)}$: IF $u_{m+1}$ is $F_{m+1}$ and...and $u_p$ is $F_p$ THEN $v$ is $G$.

3) Comparative Rules: Some rules are comparative, e.g., “The smaller the $u$, the bigger the $v$.” Such rules must first be reformulated as IF-THEN rules. The preceding rule can be expressed as “IF $u$ is $S$, then $v$ is $B$”, where $S$ is a fuzzy set representing smaller and $B$ is a fuzzy set representing bigger.

4) Unless Rules: Rules are sometimes stated using the connective “unless”; such rules are called unless rules and can be expressed as a collection of IF-THEN rules. For example, the rule $v$ is $G$ unless $u_1$ is $F_1$ and $u_2$ is $F_2$ and...and $u_p$ is $F_p$ can first be expressed as IF $u_1$ is not $F_1$ and $u_2$ is not $F_2$ and...and $u_p$ is not $F_p$, THEN $v$ is $G$.

6.4 Adaptive Neuro Fuzzy Inference System (ANFIS)

In modeling systems, usually rule structure is essentially predetermined by the user’s interpretation of the characteristics of the variables in the model but that is not always the case. In some cases, to apply fuzzy inference to a system we have a collection of input/output data that we would like to use for modeling, model-following, or some similar scenario. There is not necessarily a predetermined model structure based on characteristics of variables in the system.

There will be some modeling situations in which one can not just look at the data and discern what the membership functions should look like. Rather than arbitrarily choosing
the parameters associated with a given membership function, these parameters could be chosen so as to tailor the membership functions to the input/output data in order to account for these types of variations in the data values. This is where the so-called neuro-adaptive learning techniques can help.

The basic idea behind these neuro-adaptive learning techniques is very simple. These techniques provide a method for the fuzzy modeling procedure to learn information about a data set, in order to compute the membership function parameters that best allow the associated fuzzy inference system to track the given input/output data. This learning method works similarly to that of neural networks.

6.4.1 What Is ANFIS?

The acronym ANFIS derives its name from adaptive neuro-fuzzy inference system. Using a given input/output data set, ANFIS constructs a fuzzy inference system (FIS) whose membership function parameters are tuned (adjusted) using either a backpropagation algorithm alone, or in combination with a least squares type of method. This allows fuzzy systems to learn from the data they are modeling.

6.4.2 FIS Structure and Parameter Adjustment

A network-type structure similar to that of a neural network, which maps inputs through input membership functions and associated parameters, and then through output membership functions and associated parameters to outputs, can be used to interpret the input/output map. The parameters associated with the membership functions will change through the learning process. The computation of these parameters (or their adjustment) is facilitated by a gradient vector, which provides a measure of how well the fuzzy inference system is modeling the input/output data for a given set of parameters. Once the gradient vector is obtained, any of several optimization routines could be applied in order to adjust the parameters so as to reduce some error measure (usually defined by the sum of the squared difference between actual and desired outputs). ANFIS uses either back propagation or a combination of least squares estimation and backpropagation for
membership function parameter estimation. In general, this type of modeling works well if the training data presented to ANFIS for training (estimating) membership function parameters is fully representative of the features of the data that the trained FIS is intended to model. This is not always the case, however. In some cases, data is collected using noisy measurements, and the training data cannot be representative of all the features of the data that will be presented to the model. This is where model validation comes into play.

An example that shows how the two technologies can be used together in control is illustrated in a paper by J. R. Jang Figure 83. The paper deals with the control of a system through a self learning fuzzy controller without resorting to human experts. The aim is to come up with a method to build a fuzzy controller that can perform a predefined control task. The learning method is based on back propagation, which is used for training artificial neural networks.

### 6.5 Fuzzy logic and image processing

Fuzzy logic has found many applications in image processing and pattern recognition. Many of the basic concepts in image analysis (e.g. the concept of an edge or a corner or a relation between regions) do not lend themselves well to precise definition. Conventional approaches to image analysis and recognition consist of segmenting the image into meaningful regions, extracting their edges and skeletons, computing various features and properties (e.g. area perimeter, centroid, etc.) and primitives (e.g. line, corner, curve, etc.) and relationship among the regions, and finally, developing decision rules/grammars for describing, interpreting and/or classifying the image and its subregions. In a conventional system each of these operations involves crisp decisions i.e. yes or no, black or white, ‘0’ or ‘1’.

Because the regions in an image are not always crisply defined, uncertainty can arise within every phase of the above tasks. Any decision made at a particular level will have an impact on all higher level activities. A recognition or vision system should have sufficient provision for representing and manipulating the uncertainties involved at every processing
stage (i.e. in defining image regions, features, matching, and relations among them) so that the system retains as much of the information content of the data as possible. If this done, the ultimate output (result) of the system will possess minimal uncertainty (and, unlike conventional systems, it may not be biased or affected as much by lower level decision components). The image model should also be able to recognize, represent, manipulate, interpret, and use both fuzzy and statistical uncertainties. Statistical models deal with random events and outcomes while fuzzy models attempt to capture and quantify nonrandom imprecision.

A grey tone image possesses ambiguity within pixels because of the possible multivalued levels of brightness in the image. This indeterminacy is due to inherent vagueness rather than to randomness. Incertitude in an image pattern can be explained in terms of greyness ambiguity, spatial (i.e., geometrical) ambiguity or both. Greyness ambiguity means indefiniteness in deciding whether a pixel is white or black. Spatial ambiguity refers to indefiniteness in the shape and geometry of a region within the image.

In pattern recognition we search for structure in data. Pattern recognition is, by its very nature, an inexact science, and thus admits many approaches, sometimes complementary, sometimes competing, to the approximate solution of a given problem. Pattern recognition has found some applications in product testing and assembly, inspection and quality control.

Consider the case of a decision-theoretic approach to pattern classification. With conventional probabilistic and deterministic classifiers, the features characterizing the input vectors are quantitative (i.e. numerical) in nature. Vectors having imprecise or incomplete specification are usually either ignored or discarded from the design and test sets. Impreciseness (or ambiguity) in such data may arise from various sources. For example, instrumental error or noise corruption in the experiment may lead to partially reliable information available on a feature measurement. Again, in some cases the expense incurred in extracting a very precise exact value of a feature may be high or it may be difficult to decide on the most relevant features to be extracted. For those reasons, it may
become convenient to use linguistic variables and hedges (e.g. small, medium, high, very, more or less, etc.) in order to describe the feature information. In such cases, it is not appropriate to give an exact numerical representation to uncertain feature data. Rather, it is reasonable to represent uncertain feature information by fuzzy subsets.

**Image definition:** An image $X$ of size $M \times N$ with $L$ intensity levels available for each of its $MN$ pixels $\{x_{ij}\}$ can be considered as an array of fuzzy singletons, each having a value of membership denoting its degree of brightness relative to some brightness level $\lambda, \lambda = 0, 1, 2, \ldots L - 1$. We let $\{m_x(x_{ij})| (x_{ij} \in X)\}$, where $m_x(x_{ij}), 0 \leq m_x(x_{ij}) \leq 1$, denotes the extend to which each pixel possesses some property $m_x$ (brightness, edging, smoothness) or the degree of belongingness to some image subset (e.g., object, skeleton, or contour) by each pixel $x_{ij}$. In other words, a fuzzy subset $m$ of an image $X$ is, as it must be, a mapping $m$ from $X$ into $[0,1]$. For any point $x_{ij} \in X; m_x(x_{ij})$ is called the degree of membership of $x_{ij}$ in $m$. Note that an ordinary subset of $X$ can be regarded as a fuzzy subset for which $m$ takes on only the values of 0 and 1.

One can use either local or global information in an image to define membership functions characterizing various image properties. For example, brightness or darkness can be defined only in terms of the grey value of a pixel whereas edgings or textural properties need neighborhood information about a pixel to define their membership functions.

From these examples, we see that concept of fuzzy sets can be used at the feature level for representing input data as an array of membership values denoting the degree of possession of certain properties, at the classification level, for representing class membership of objects, and for providing an estimate (or a representation) of missing information or presence of certain defects in terms of membership values. Therefore, fuzzy set theory can be incorporated in the handling of uncertainties (arising from deficiencies in the available information caused by, among others, incomplete, imprecise,
ill-defined, not fully reliable, vague, and contradictory data and information) in various stages of pattern recognition, texture analysis, defect detection and classification.

6.5.1 Application of fuzzy logic for defect detection

A new and exciting application of fuzzy logic, introduced in this thesis, is for defect detection in web inspection systems. Environmental noises such as small changes in illumination and/or possible dust on web materials added to ill-defined boundaries of defects makes fuzzy logic an ideal choice for this application area. Environmental noises captured by web sensors (cameras) are not usually supposed to be considered as defects and are removed by a preprocessing stage. This approach can be quite complex and in some applications inefficient. This burden is removed with modifying membership functions and rules applied to fuzzy inference engine in our new method without any increase in the complexity of the system.

The main concern with conventional defect detection approaches is in the exact definition of the defect in the web material when its boundary is ill-defined. Any hard thresholding made for the extraction of the defect will propagate the associated uncertainty to subsequent stages and this, in turn, might affect feature analysis and recognition. It is convenient, natural, and appropriate to avoid making a specific hard decision (e.g. segmentation/thresholding, edge detection, and skeletonization), by allowing the segments, skeletons, or contours to be fuzzy subsets of the image, the subsets being characterized by the possibility or degree to which each pixel belongs to them. Similarly, for describing and interpreting ill-defined structural information in a pattern, it is natural to define primitives (line, corner, curve, etc.) and relations among them using labels of fuzzy sets. For example, primitives that do not lend themselves to precise definition can be defined in terms of arcs with varying grades of membership from 0 to 1 representing their degree of belonging to more than one class.

Figure 6.6 "Grey level of 5 image rows for the texture shown in Figure 6.5 "A texture with wrinkle defects" on page 97" on page 98 shows the grey level of 5 image rows for the
marble-like texture with two vertical wrinkle defects in Figure 6.5. Although the grey levels of each line are not random, and are highly correlated, they do not exactly overlap each other. The wrinkle defects are marked by a triangle sign in Figure 6.6. Some spikes can also be seen in the grey level in some lines that should not be considered as defect lines. They might arise from instrumental error or environmental noise that is usually filtered out in preprocessing. It is obvious that any traditional approach such as simple thresholding will fail to exactly locate the defects with ill-defined boundaries.

Figure 6.5 A texture with wrinkle defects
Figure 6.6 Grey level of 5 image rows for the texture shown in Figure 6.5 "A texture with wrinkle defects" on page 97

Fuzzy logic can also be used in defect classification. Consider the problem of determining the boundary or shape of a defect or texture from its sampled points or prototypes. Conventional approaches attempt to estimate an exact shape for the area in question by determining a boundary that contains (i.e., passes through) some or all of the sample points. However, this property is not necessarily desirable for boundaries in real images. For example, it may be necessary to extend the boundaries to represent obscured portions not represented in the sample points. Extended portions should have a lower membership in the boundary for such a class than the portions explicitly highlighted by the sample points. The size of extended regions should decrease with an increase in the number of sample points. This leads one to define multivalued or fuzzy (with continuum grade of belonging) shapes and boundaries of certain defects or textures.

Uncertainty in classification may arise from the overlapping nature of various classes: for example, edges in a defect do not have clear grey level boundaries— they really do overlap.
In conventional defect detection and classification techniques, it is usually assumed that pattern can belong to one and only one class, which is not necessarily realistic physically, and certainly not mathematically. Thus, feature vectors and the objects they represent can and should be allowed to have degrees of membership in more than one class.

6.5.2 Theory

A nonrandom texture $X$ of size $M \times N$ can be described by two sets of horizontal and vertical features, $S_h$ and $S_v$, where $k$ and $l$ are the number of texture features in each set.

$$S_h = \{F_1, F_2, \ldots, F_k\}$$

$$S_v = \{F_1, F_2, \ldots, F_l\}$$

Average grey level, number of jumps (edges) in the grey level (from a predefined threshold) or number of pixels in a specified grey level range can be considered as texture features where they are derived off line from the golden (defect free) template. These features are extracted for each texture by an image expert or a software program.

Although the algorithm is described in detail for one dimensional case where it is used to detect the defective lines, it can easily be expanded to two dimensions to locate the exact position of the defect.

A subset of texture features $s_h$, $s_h \subseteq S_h$ is selected that is more sensitive to possible occurring defects.

$$s_h = \{F_1, F_2, \ldots, F_p\}, \ p \leq K$$
For the texture features in $s_h$, appropriate membership functions are defined. The parameters of the membership functions can be obtained by trial and error or through an optimization program. To activate the fuzzy inference engine, appropriate rules are written. Employing only a subset of rules from the set of possible ones leads to satisfactory results and adding any unnecessary rules will increase the complexity of the fuzzy inference engine causing a reduction in the clarity of the system. In a fuzzy inference engine the rules are combined to output a variable that shows the degree of defectiveness of each line. Each rule can be assigned a weight as providing a degree of belief to the rule.

Successful implementation of the algorithm greatly depends on suitable choices of the appropriate texture features. Our experimental results show that employing only two texture features leads to satisfactory results in many practical applications. The features are obtained for a minimum of $M$ pixels in an image row where $M$ is a good representation of the texture. $D$ is the output of the inference engine that is calculated by fuzzy fusion of a subset of the texture features, $F_{i_h} \in s_h$, $i=1, \ldots, p \leq K$. It shows the degree of defectiveness of each scanned line, $D$. It is close to zero for non-defective scanned lines or for ones with ignorable defects, and it moves towards ‘1’ as the scanned line becomes more defective. By thresholding using the following pseudocode the defective lines are selected:

```
If D ≥ T then
    The line is defective
else
    The line is not defective
end If
```

The threshold level "T" is obtained in the learning phase and lower threshold levels will lead to false alarm errors where the non-defective lines are considered as containing
defects. Higher threshold levels will lead to the ignoring of defective lines errors. A reasonable method to calculate these errors can be through the following formulae, obtained for a frame of 1024 image rows. $E_F$ and $E_I$ are false alarm and ignore defective lines errors respectively.

\[
E_F = \frac{N_e}{1024 - N_D} \tag{6.1}
\]

\[
E_I = \frac{N_i}{N_D} \tag{6.2}
\]

$N_e$ and $N_i$ are number of lines that are erroneously selected or ignored by using this method. $N_D$ is the number of real defective lines in a 1024 row frame. It is interesting to note that it is possible to have both errors for a specified threshold level but one of the errors will be the dominant one while the second one can be ignored. Figure 6.7 shows the block diagram of a fuzzy defect detector engine with two inputs and one output.

Figure 6.7 Block diagram of a fuzzy defect detector engine with two inputs and one output

6.5.3 Development of a fuzzy logic defect detection system in brief

1. Data are collected from sensors (e.g. camera)

2. We search for underlying structure in the data that provides a basis for describing the texture features.
3. The best features that describe the texture and are more sensitive to possible occurring defects are selected.

4. The fuzzy inference engine is formalized by characterizing the process with inputs and output(s) (name, range, membership functions), rules (antecedent, consequent, weight, connections). In short, a model of the system is generated.

5. The fuzzy inference engine is often trained with training data. We find the parameters of the inference engine by providing it with correct (defect free) and defective samples.

6. The fuzzy system is tested for its sensitivity to response to possible defects, error rate performance and complexity.

7. The candidate parameters are selected and used in the real time defect detection engine.

6.6 Simulation Results

The proposed algorithm has been simulated on several defect samples taken from production web systems. We have used Matlab as our modeling and simulation tool. In this section we describe the procedure of applying the proposed method on some real defective samples. Figure 6.8 shows a random texture sample with several stain defects; the image is digitized at a resolution of 256 rows x 256 columns with 8 bits of grey level information. The marble-like nature of the pattern plus vagueness in the definition of the defects edges and environmental noise added in capturing the image makes the defect detection task quite complicated. To apply the algorithm, we search for suitable texture features that are sensitive to possible occurring defects where they change measurably for defective lines. Although, for the following example, features are obtained for the whole image row (256 pixels), it will be more efficient if the features are extracted and the algorithm applied for small fractions of each line that still provides a good representation of the texture. Figure 6.9 shows the histogram of the average grey level for the non-defective lines of the texture shown in Figure 6.8. The average grey level has a bell shape distribution with the center of 242 and the range of 239 to 244. Figure 6.10 plots the histogram of the average grey level for each line for the texture shown in Figure 6.8 "A sample texture with stain defect" on page 103 with defects. The average grey level spans
from 227 to 246 while mainly concentrating around 240 that represents the non-defective lines.

Figure 6.8 A sample texture with stain defect
Figure 6.9 Histogram of the average grey level for each line for the non-defective portion of the texture shown in Figure 6.8 "A sample texture with stain defect" on page 103
Figure 6.10 Histogram of the average grey level for each line for the texture shown in Figure 6.8 "A sample texture with stain defect" on page 103
The second feature selected for this texture is number of jumps (number of edges). Figure 6.11 shows the histogram of the number of jumps (edges) in each line for the threshold of 25 for the defect free portion of the texture shown in Figure 6.8. It shows that for a threshold of 25 the number of jumps is close to zero for most of the lines while for some small number of lines, it can go up to 2. Figure 6.12 plots the histogram of the number of jumps (edges) in each line for the threshold of 25 for the defective portion of texture. It shows that the maximum number of jumps can reach up to 15 for some defective lines but it is still mainly centered at zero because of the dominance of the non-defective lines.

![Histogram of number of jumps (edges) in each line for the threshold of 25](image)

Figure 6.11 Histogram of the number of jumps (edges) in each line for the threshold of 25 for the defect free part of the texture shown in Figure 6.8 "A sample texture with stain defect" on page 103
Figure 6.12 Histogram of the number of jumps (edges) in each line for the threshold of 25 for the defective texture shown in Figure 6.8 "A sample texture with stain defect" on page 103

Average grey level and Number of jumps are used as two inputs to the fuzzy inference engine. Defect is the only output of the engine. The following membership functions are defined for the inputs and the output:

Figure 6.13 "Three membership functions for the “AGL” input for the defect shown in Figure 6.8 "A sample texture with stain defect" on page 103" on page 108 shows the three membership functions defined for the “AGL” input. They are called Lower, Normal, and Higher and their curve parameters are adjusted by trial and error to get the best performance at the output of the fuzzy inference engine.
Figure 6.13 Three membership functions for the "AGL" input for the defect shown in Figure 6.8 "A sample texture with stain defect" on page 103

*Low* and *High* are the two membership functions defined for the # of Jumps input. Figure 6.14 "Two membership functions for the "# of jumps" input defined for the defect shown in Figure 6.8 "A sample texture with stain defect" on page 103" on page 109 illustrates the membership functions' curves for this input. Z-shaped and S-shaped curves are selected from the library of Matlab membership functions curves for the *Low* and *High* respectively.
Figure 6.14 Two membership functions for the "# of jumps" input defined for the defect shown in Figure 6.8 "A sample texture with stain defect" on page 103

The output fuzzy set for the fuzzy inference engine is called Defect and is represented by three membership functions. They are called less likely to contain a defect (LLD), more likely to contain a defect (MLD) and uncertain to contain a defective (UD). The membership functions are depicted in Figure 6.15. Table 6.1 "Input and output membership functions and their associated parameters" on page 110 summarizes the membership functions and their parameters used in this example.
Figure 6.15 Three membership functions for the output “Defect” obtained for the defect shown in Figure 6.8 "A sample texture with stain defect" on page 103

Table 6.1 Input and output membership functions and their associated parameters

<table>
<thead>
<tr>
<th>Inputs and Output</th>
<th>Membership functions</th>
<th>Membership function parameters</th>
<th>membership functions curve</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input: AGL</td>
<td>Lower</td>
<td>gbellmf(238, 50, 0)</td>
<td>Generalized bell</td>
</tr>
<tr>
<td></td>
<td>Normal</td>
<td>trapmf(239, 240, 245, 246)</td>
<td>Trapezoidal</td>
</tr>
<tr>
<td></td>
<td>Higher</td>
<td>gbellmf(10, 20, 256)</td>
<td>Generalized bell</td>
</tr>
<tr>
<td>Input: # of Jumps</td>
<td>Low</td>
<td>zmf(1, 2)</td>
<td>Z-shaped</td>
</tr>
<tr>
<td></td>
<td>High</td>
<td>smf(1, 4)</td>
<td>S-shaped</td>
</tr>
<tr>
<td>Output: Defect</td>
<td>LLTBD</td>
<td>zmf(0.1, 0.7)</td>
<td>Z-shaped</td>
</tr>
<tr>
<td></td>
<td>NSTBD</td>
<td>trimf(0.2, 0.368, 0.6)</td>
<td>Triangular</td>
</tr>
<tr>
<td></td>
<td>MLTBD</td>
<td>smf(0.0501, 0.6)</td>
<td>S-shaped</td>
</tr>
</tbody>
</table>
The following five rules are applied for defect detection:
1. If (AGL is Normal) and (# Jumps is Low) then (Defect is LLD)
2. If (AGL is Higher) and (# Jumps is High) then (Defect is MLD)
3. If (AGL is Lower) and (# Jumps is High) then (Defect is MLD)
4. If (AGL is Normal) and (# Jumps is High) then (Defect is UD)
5. If (AGL is Higher) and (# Jumps is Low) then (Defect is UD)

Experimental results show that increasing the number of rules will not increase the accuracy and performance of the system and will make further modifications more difficult.

A 3D rule-view graph is shown in Figure 6.16. It allows us to interpret the entire fuzzy inference process at once. It also helps to visualize how the shape of certain membership functions influences the overall result. Since it plots every part of every rule, it can become unwieldy for particularly large systems, but, for a relatively small number of inputs and outputs, it provides useful visual information.
The algorithm is simulated in the Matlab environment using the Simulink toolbox, as shown in Figure 6.17. Figure 6.18 "Average grey level for each line for the sample shown in Figure 6.8 "A sample texture with stain defect" on page 103" on page 113 and Figure 6.19 "Number of jumps for each line (threshold = 20) for the sample shown in Figure 6.8 "A sample texture with stain defect" on page 103" on page 113 plot the two inputs of \textit{AGL} and \textit{# of Jumps} where Figure 6.20 "Fuzzy output “Defect” for each line for the sample shown in Figure 6.8 "A sample texture with stain defect" on page 103" on page 114 shows the related output. By applying the threshold of 0.57 to the defect, all the defective lines will be selected with less that 1\% false alarm or ignoring defective lines error. The result is shown in Figure 6.21 "Selected defective lines from the sample shown in Figure 6.8 "A sample texture with stain defect" on page 103 for the threshold of 0.57" on page 114.

\begin{figure}[h]
\centering
\includegraphics[width=0.7\textwidth]{simulink_diagram.png}
\caption{Simulation environment in Simulink-Matlab}
\end{figure}
Figure 6.18 Average grey level for each line for the sample shown in Figure 6.8 "A sample texture with stain defect" on page 103

Figure 6.19 Number of jumps for each line (threshold =20) for the sample shown in Figure 6.8 "A sample texture with stain defect" on page 103
The threshold level should be chosen carefully in order to detect the exact defective lines. In some applications it might not be possible to select the best threshold level to obtain zero error rate. In such cases the threshold is shifted to a region with more manageable or acceptable error. For example in some applications it is vital to detect every possible defect to maintain the high quality of the production. In these scenarios it is preferred to have false alarm errors than to have ignore defect errors. In these situations the threshold should be reduced from its optimal level. There are several other applications where ignoring some of the defects will not have major impact on the quality of the material produced. In such applications the threshold can be set higher than the optimal level; this
will also result in more data compression. $E_F$ and $E_I$ are calculated using Eqn. (6.1) and Eqn. (6.2) for the texture in Figure 6.8; the curve is shown in Figure 6.22.

![Graph showing error generated versus different threshold levels at the output of the fuzzy inference engine for the defective texture shown in Figure 6.8.](image)

**Figure 6.22** Error generated versus different threshold levels at the output of the fuzzy inference engine for the defective texture shown in Figure 6.8 "A sample texture with stain defect" on page 103

### 6.7 Hardware Implementation

The algorithm has been implemented in a Xilinx 4005E FPGA [66] "The programmable Logic Data Book, Xilinx Inc., San Jose, California, 1995." on page 130 which is used in our experimental test setup [59]. 72 percent of the resources of the FPGA are used in
implementing this algorithm. Figure 6.23 shows the signal flow and subblocks inside the FPGA. The scanned data from the camera is directly fed to the feature extractor blocks (FE) and their outputs are used as address lines for the (LUT). Since the FEs process in parallel, the system is quite fast. The LUT output is thresholded and used as a flag signal to indicate whether or not the scanned line contains a defect. The LUT and thresher can be combined to make a smaller \((1 \times n)\) LUT. The size of the LUT in this sample was \(1 \times 256\). Locating the exact position of the defects is performed by either applying the proposed method in rows and columns of the scanned data (2D version) or employing other techniques such as a 1D autoregressive algorithm \([61]\). A VHDL program for the algorithm is included in Appendix E "VHDL Program for Fuzzy Fusion" on page 162. More examples and the related Matlab programs are listed in Appendix F "MATLAB Files" on page 168.

![Diagram of signal flow and FPGA subblocks](image)

Figure 6.23 The signal flow and the subblocks in the FPGA

### 6.8 Fuzzy Logic and Neural Networks

Fuzzy logic is often associated with research involving neural networks. Although the two technologies have some fundamental characteristics in common, they probably should be viewed as complementary. Neural nets are arrays of interconnected processing nodes called neurons; somewhat modeled on biological nerve cells. In artificial neural networks, each neuron receives one or more inputs, which it multiplies by weighting factors and sums together to produce an output. Neurons can be arranged in layers, with the first layer receiving initial inputs then passing its outputs to a second layer. More than two layers can
be implemented. Each set of neurons in a layer performs its own weighting and summation and the process continues until a final output produced.

The most important thing about neural networks is the fact that they can be designed to be self modifying. If the final output is not the one expected, the system can adjust the weights applied to each input to produce a new result. This is done by following certain training algorithms. The training process is done by iteration, that is input data are run through the network repeatedly, and each time the weights are tuned to improve the result. Eventually, the desired output is achieved. The fact that neural nets can learn from previous experience makes them suitable for use in fuzzy logic systems. In fact they have been used to make fuzzy controllers adaptive. In adaptive system the rules of the fuzzy controller are run through the neural net, or are monitored by it for optimization purposes. In this case, the neural net can learn new rules or modify existing ones.

FLS's and feedforward neural networks (FFNN's), can both be used to solve similar problems. Both are model free, i.e., all the information they are given is contained in some examples from which they are required to learn so to give correct or successful outputs when new inputs are presented. They are given the same information and asked to perform the same task; but, linguistic knowledge can only be used by a FLS whereas it cannot yet be used by a FFNNS. Such knowledge can be invaluable, especially if there is little numerical training data. Tuning a FLS can be done much faster than tuning a FFNNS because the parameters of the FFNS usually must be initialized randomly. Finally, the fuzzification subsystem within the FLS lets us handle the uncertainty in a natural way, totally within the framework of FLS's. To date there does not seem to be a comparable way to handle uncertainty in a FFNN.

6.9 Conclusion

Inspired by natural fuzzy descriptions of defects in web inspection and vagueness in defining their boundaries, we have produced a novel algorithm that is able to detect defects based on fuzzy fusion of texture features. To exactly define defects in web material while
boundaries are ill-defined, is the main concern with conventional defect detection approaches. Any hard thresholding made for the extraction of defects will propagate the associated uncertainty to subsequent stages and this, in turn, might affect feature analysis and recognition. In our new algorithm, we extend the boundaries to represent obscured portions and represent them with lower membership values. This leads one to define multivalued or fuzzy (with continuum grade of belonging) shapes and boundaries of certain defects or textures. This algorithm does not need any alignment of the texture before applying, unlike the requirements for exact alignment in template matching solutions. It can be applied to textured and non-textured materials and our simulation results show that the algorithm is capable of detecting the defects with a high probability. The algorithm has been implemented in the limited resources of one Xilinx XC4005 FPGA in our test setup. The simplicity and preciseness are the two main advantages of this algorithm. Since the FEs are extracted in parallel and the decision engine is just a lookup table, the processing time is minimized and is certainly applicable to high speed production applications.
Chapter 7

Conclusions

7.1 Summary and Contributions

This thesis presents the development of efficient algorithms for real time defect detection in web inspection systems. The data-stream algorithms are targeted to the limitations of in-camera hardware which is constrained to operate on raster scanned image data without recourse to traditional 2-D frame storage mechanisms. Unlike many traditional techniques to identify defects which use algorithms based on two dimensional image data, our new methods are based on one dimensional processing of data from raster scanned web materials.

Modifications to existing thresholding algorithms (multi-level thresholding, zero order background tracking, and Delta modulation background tracking) are introduced as algorithms which can be applied to defect detection on non-textured web materials and in certain cases to slowly changing backgrounds. Some modifications are applied in order to reduce the false alarm rate caused by environment noise. These algorithms can be implemented using less than 100 CLBs in an FPGA. The main limitation for these algorithms is that they can not be applied to textured materials. Considering the hardware constraints, finding a
suitable algorithm for defect detection on textured materials is quite challenging; we introduce the following novel algorithms:

**AR technique.** We have applied auto regressive methods as a statistical approach to analyze textured backgrounds and to identify possible defects. In the simplest form, the image is scanned to provide a one dimensional series of grey level fluctuations and a pixel is assumed to depend upon a certain number of pixels in its neighborhood. The coefficients of dependence are extracted using time series analysis techniques. This method can be applied to many two dimensional textured backgrounds; a literature survey shows no record of similar research. This algorithm is targeted to the resources of a combined DSP chip/FPGA architecture. One of the main advantages of this algorithm is that it is not sensitive to a shift in the texture pattern and so scanning can start anywhere on a line.

**Fuzzy logic technique.** We have successfully developed, simulated, and implemented a novel algorithm for detecting defective (sub)lines based on a fuzzy fusion of texture features. We have used the idea of fuzzy definition of defects in manual detection such as: darker or brighter regions; smaller or larger objects. Unlike traditional methods of computation, fuzzy computing accommodates the imprecisions of the real world. In this algorithm we use imprecise or ambiguous image data caused by instrumental error or environmental noise such as dust or small variations in illumination to achieve a tractable, robust, and low cost solution for our defect detection system. The proposed algorithm has been simulated on many defect samples from production web systems and successfully implemented on the experimental test setup. It has been experimentally shown that selecting as little as only two features of the texture leads to satisfactory result. Since the features are extracted and processed in parallel, the algorithm is quite fast. The lookup table for our experimental set-up is only a $1 \times 256$ ROM implemented by 32 CLBs in our target FPGA array. This algorithm can be applied to both textured and non textured materials and offers superior performance over traditional methods such as template matching. Although it is mainly applied to the detection of defective lines, it can also be applied to smaller regions of each line to increase the compression and locate the exact position of the defect. We are not able to apply the proposed algorithm in two dimensions
to identify the position of defect since the hardware is constrained to processing on a single line.

Table 7.1 on page 123 compares the five different algorithms investigated in this research. They are compared based on different criteria such as complexity of the algorithm, the resources used to implement in one XC4005 FPGA, their applications, sensitivity to illumination, etc. By studying the table, it is clear that choosing algorithms is an application dependent task, and that we may never be able to find the “optimal algorithm”, but merely one that performs the required task in an efficient manner. While the thresholding technique is simple to implement and needs very little off line processing, it can not be applied to textured backgrounds. On the other hand, the fuzzy logic algorithm requires more off line processing, such as extraction of texture features, generation of membership functions and their parameters; however, it can be successfully applied to a wider range of applications.

### 7.2 Suggestions for Future Work

No matter how extensive a research project is, it is never complete. Certainly there are many more avenues to pursue than those explored in this research work. Here we provide a list of ideas and topics to carry the results obtained in this work to the next milestone.

**Hardware.** In our design, the maximum density FPGA available to us was the XC4010E FPGA that has 400 CLBs. This number of CLBs was not adequate for implementing more complex algorithms. Virtex™ is the latest generation of FPGA from Xilinx that offers the densest FPGA at the clock speeds up to 160MHz with operating voltage of 2.5 volt with 5 volt tolerant I/Os, and the application of this new family of devices will both enhance the performance of the system and allow us to experiment with more complex algorithms. Increasing mutual communication between FPGA and DSP chip and using denser dual port RAM as a link between FPGA and DSP chip are some of the other issues that need to be considered in the next generation of the hardware.
Applying fuzzy logic algorithm in two dimension. We were not able to experiment with our novel fuzzy logic algorithm in two dimensions because of the hardware limitations. Theory and simulation results show that the algorithm can be generalized to two dimensions to locate the exact position of the defects. We believe that this will be an interesting subject to explore and define a hardware architecture to implement it.

Automatic extraction of texture features. As we mentioned in Chapter 6, successful implementation of the fuzzy logic algorithm highly depends on choosing the most suitable texture features. In our method these features are extracted by an image expert. The idea of automatic extraction of the most appropriate features of the texture seems an interesting research area.

User friendly software for reprogramming the FPGA. Probably reprogrammability is the main advantage of using FPGAs. In our system, the programming is done by an expert and requires the knowledge of VHDL/Verilog and/or related software. In an industrial environment, it will be more appealing to have a user friendly GUI for reprogramming the FPGA. In this environment the operator needs only to connect suitable blocks and modify their parameters for his target applications. This environment will greatly enhance the versatility of the system.
Table 7.1 Comparison of the simulated and applied algorithms

<table>
<thead>
<tr>
<th>Sensitivity to illumination</th>
<th>Fuzzy logic</th>
<th>Target applications</th>
<th>FPGA implementation</th>
<th>Resources used in XC4005 FPGA</th>
<th>Off line calculations</th>
<th>Reprogramming for other applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Two (multil)</td>
<td>Moderate</td>
<td>High</td>
<td>Slowly changing</td>
<td>Yes</td>
<td>Low</td>
<td>Simple</td>
</tr>
<tr>
<td>Two (multil)</td>
<td>Moderate</td>
<td>Slowly changing</td>
<td>Non textured</td>
<td>Yes</td>
<td>Low</td>
<td>Simple</td>
</tr>
<tr>
<td>Two (multil)</td>
<td>Moderate</td>
<td>Non textured</td>
<td>Non textured</td>
<td>Yes</td>
<td>Low</td>
<td>Simple</td>
</tr>
<tr>
<td>Zero Order modulation</td>
<td>Moderate</td>
<td>Slowly changing</td>
<td>Some 2D textures</td>
<td>Yes</td>
<td>Moderate</td>
<td>Simple</td>
</tr>
<tr>
<td>Delta modulation</td>
<td>Moderate</td>
<td>Slowly changing</td>
<td>Some 2D textures</td>
<td>Yes</td>
<td>High</td>
<td>Simple</td>
</tr>
<tr>
<td>1D-AR algorithm</td>
<td>Moderate</td>
<td>Non textured</td>
<td>Non textured</td>
<td>No</td>
<td>Moderate</td>
<td>Simple</td>
</tr>
</tbody>
</table>

Notes:
a. Moderate
b. Slowly changing
c. Depends on texture features
d. Almost all textures
### Table 7.1: Comparison of the simulated and applied algorithms

<table>
<thead>
<tr>
<th></th>
<th>1D-AR algorithm</th>
<th>Fuzzy logic</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Algorithmic complexity</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Low</td>
<td>Moderate</td>
<td></td>
</tr>
<tr>
<td>Low</td>
<td>Moderate</td>
<td></td>
</tr>
<tr>
<td>Moderate</td>
<td>High</td>
<td></td>
</tr>
<tr>
<td>Low</td>
<td>Moderate</td>
<td></td>
</tr>
<tr>
<td>Moderate</td>
<td>No</td>
<td>Moderate</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Presumption that certain pixels of the texture should be defect free</th>
<th>No</th>
<th>No</th>
<th>No</th>
<th>Yes(^m)</th>
<th>No</th>
</tr>
</thead>
</table>

---

a. Change in illumination will greatly affect the grey level of the pixels that are thresholded by a fixed level.

b. Since each pixel is compared to the previous pixel, this method is less sensitive to slowly varying illumination variations. However, rapid changes in illumination will affect the algorithm.

c. This algorithm can be applied to all textures except ones with random patterns.

d. Based on the hardware constraints, we are unable to implement this algorithm using the limited resources of one XC4005 FPGA.

e. Resources is a combination of I/O pins, flip flops, and CLBs.

f. Only threshold level should be calculated off line using histograms of the grey levels.

g. Threshold, order of the nonlinear filter should be calculated off line.

h. Texture features, membership functions, threshold level for defuzzification of the output, data for the look-up table should be calculated off line.

i. In order to apply this algorithm to different textures we only have to calculate the new threshold level.

j. Order and coefficients of the AR filter should be calculated for a new texture.

k. Complexity is a combination of off line calculations and the hardware resources needed to implement the algorithm.

l. Although calculating the AR filter coefficients is not complex, implementing the algorithm, however, needs more resources than are available in one XC4005 FPGA.

m. To apply the 1D AR algorithm, we assume that the first n-1 pixels of each line, or in some cases the n-1 pixels of the first line, are defect free.
REFERENCES


[132] http://www.vvl.co.uk/imputer/


Appendix A

Commercial Defect Detection Systems

A.1 Systems from Datacube Inc.

*Arachne* is one of the general purpose commercial defect detection systems manufactured by Datacube Inc. [137]. *Arachne* mounts on a customer's machine, a paper machine for example. It evaluates defects under real conditions on a running 1000-feet-per-second web. “Dynamic”, on-machine analysis also takes into account variations in lighting, machine speed, lens effects, and defect contrast, size, density, and orientation. It enables integrators to see what influence ambient lighting, air (temperature, dust, and moisture), and vibration may have on a system’s performance. *Arachne* does not sample the web. It records every all video information from the camera. Operators see continuous, real-time video images of the web showing all its characteristics, including defects, that are present at that point in the operation [103]. Often this ability helps customers understand how to characterize and classify their defects. The *Arachne* can record and playback the video images it captures. This tool enables integrators to use image processing software to determine the best parameters for a given inspection applications. *Arachne* performs inspection of a wide variety of web materials, including paper, film, and textiles. Depending upon the configuration, the system can view a web 10
meters wide and operate on production lines traveling at 1500 meters per minute. With appropriate camera setting for scale and resolution, *Arachne* can detect defects as small as 25 microns. Three key components help accomplish these capabilities:

1. Web Real-Time Disk for capture and playback of defect images to build a defect image database and evaluate image processing techniques.

2. Arachne Analysis Software for quick and easy evaluation of computer algorithms and image processing solutions without programming.

3. Black Widow Web Inspection Engine (Figure A.1) for implementation of the image processing solutions on the captured defect images on the production line.

![Figure A.1 Black Widow Web inspection system from Datacube](image)

**A.2 Systems from Cognex Inc.**

Cognex [131] offers a family of vision systems designed to meet the needs of a wide range of customers. Because of diverse needs of applications, the products come in different hardware platforms, development modes and price/performance modes.
Cognex application specific products combine Cognex hardware and software to create a solution that is tailored to the particular requirements of certain vision applications. These products are: acuReader/OCR II, acuReader/OCV, acuReader/2D, acuFinder/acuFinder 2, BGA Inspection Package, and SMD Placement Guidance Package. These packages include a Windows-based application GUI that allows users to quickly and easily set up the system. As an example, Cognex acuReader/OCR II is a PC plug-in industrial character recognition system designed to provide reliable, high speed reading performance despite quality degradations to the characters due to process variations. This system now offers a simplified operator interface called ID Expert to guide users through the process of setting up and reading an alphanumeric character string on silicon wafers. Typical applications for acuReader include identifying serial numbers on semiconductor wafers, disk heads, and other small parts.

is1500 and is2000 are two surface inspection systems from Cognex that use the highest performance camera technology to detect and classify defects in many applications. They can automatically detect, visualize, and classify surface flaws on a wide variety of high, value-added materials. Using high-speed digital CCD cameras, and sophisticated, yet user-friendly, software, these systems inspect web materials at full production speeds. Sensitive electronics locate defects and classify them according to the specific application requirements. Examples of material inspected by these systems include galvanized steel used in automotive manufacturing, copper foil used in the manufacture of printed circuit boards, and high-tech synthetics used in clothing and building materials.
A.3 Systems from Imputer Inc.

The high resolution Imputer 3 smart camera, from VLSI Vision Ltd. Edinburgh, Scotland, UK [132], combines a CMOS image sensor, framegrabber and image processor in a single package. It is targeted for a wide variety of applications including machine vision, industrial inspection, smart security, remote vision, quality control, traffic monitoring, character recognition, object tracking and object counting. It uses a multi-standard 768 x 574 resolution CMOS image sensor and an i960 RISC processor. The library function benchmark timings given by the company show that simple arithmetic functions can be done in real time but more complex ones such as morphological functions can not be performed in real time except in small regions of interest of the image.

A.4 System from VE Technologies

The VE 379 image processing system, from VE Technologies St. John's, NF, Canada [133], is based on an Intel computer with an attached multiprocessor DSP (TMS320C80) image processing board (see Figure A.3).
A callable library of basic image processing functions plus a collection of high level functions (e.g. pattern recognition, object classification, etc.) is also available. The VE-262 incorporates a monochrome camera with a 486PC (including hard disk, RAM, video card and Windows™ operating system), and a DSP frame grabber for rapid image processing. The VE-262 Smart Camera can automate a wide range of vision tasks such as surveillance, detection, counting, measuring and inspection. The Smart Camera has been applied in applications ranging from underwater fish counting to robot vision.
Appendix B
Survey of Machine Vision Software

Research has been done to inspect chip packages by Hyundai, South Korea [28]. The idealized machine has nine inspection points with one camera each. Each inspection point will look at specific characteristics of the finished package: leads, face, solder coating, and marking on the chip top. The inspection line at each station should take at most 0.7 seconds, for a total time of 5.6 seconds per chip. MaxVideo 20 from Datacube Inc. [137] which uses a pipelined architecture to achieve high speed and high degree of parallelism is used in this research. Its internal speed is 20Mhz (50 ns per pixel), which yields about 80 frames per second on a $512 \times 512$ frame. Different algorithms such as histogramming and inspection by comparison (template matching) are used for inspection in this system.

A part inspection system which learns good and bad parts in clutch drives by detecting missing rivets and springs is designed by A.D.Mcaulay et. at [52]. The system extracts features in a circular region of interest from a video image, processes these using a FFT for rotation invariance, and uses this as input to a neural network trained with back-propagation. The hardware consists of a Sun workstations, Datacube and a camera. The neural network in this system consists of three layers with 100 inputs, 10 hidden, and 2
output nodes. Back-propagation uses an iterative gradient algorithm to adjust the network’s weight so as to minimize the mean square error between the actual and the desired output of all the training patterns. The author states that 100 percent classification of good and bad parts was achieved but mentions that this method was used to detect the defects on round shaped objects and for complicated objects more sophisticated techniques should be used.

An automated fuel pellet inspection system is designed for the Westinghouse Commercial Nuclear Fuel Division which manufactures enriched uranium pellets for use in nuclear fuel assemblies [53]. The size and shape of the pellets make them difficult to physically manipulate at high speeds. Since the entire outer surface of the pellet must be inspected, the pellets are simultaneously rotated and translated axially past the camera. The production rate is greater than 5 pellets per second. A kind of thresholding technique is used to detect possible defects on the pellets.

A working system to detect defects on Printed circuit board was developed by Hara, Doi, Karasaki and Tida [116]. Printed circuit board (PCB) defects such as shorts, cuts, mousebites are detected by using violet or ultraviolet illumination and by detecting the patterns via a high sensitivity TV camera. This sensing captures the emitted fluorescent light by the base material after filtering. Pure reflection based systems are not completely reliable because they usually exhibit a much lower signal to noise ratio and a greater false alarm rate than the emitted light approach. The system is able to detect a defect of 0.01 mm. The time consumed in the inspection of a 500mm × 600mm PCB is approximately 18 minutes. 100% of the defects were detected.

Research has been carried out in New Zealand to develop a prototype system for wood panels to detect and classify the various defect types at production rates [29]. The range of surface defects occurring during the manufacturing of this product includes those having both colour and textural variations. The prototype uses a combination of general purpose processor and pipelined processing modules to process images obtained from the moving product. Algorithm development is performed on the Sun workstations using Khoros
software package. Then they are ported to the machine vision computer achieving a performance improvement of several orders of magnitude. The final implementation is written entirely in C and uses Datacube’s ImageFlow Image processing Library [103]. ImageFlow provides an object oriented approach for controlling a diverse range of pipeline processing modules. It has also been investigated that different lighting is required to detect different defects. Analyzing the images obtained with a single lighting technique and with only one algorithm detects only some categories of defects and provides little discrimination between the various defects. Two basic techniques are adopted for defect detection. An analysis of image irregularities or “blobs”, and a statistical analysis using the variance of distribution of brightness. In order to speed up the processing, subsampling is done on the pixels. The processed image is expanded back to the full size by replicated pixels. Laser scanning can also be used for surface inspection of wood panels. It offers potential advantages in terms of both uniformity and monochromatic nature of the light source. Possible disadvantages are the greater cost and complexity of the scanning system relative to a camera. The paper does not mention anything about the speed of the scanning.

A detection system has been developed for detection of humps on high tension cable insulators using reconfigurable technology circuits of Xilinx [51]. A software program has been developed so as to detect automatically the humps. The cable’s borders which are placed orthogonal to the image’s lines are analyzed to verify if the variation of columns belonging to a given line is greater than a given value. The design has been implemented in an XC3164 FPGA. The design has utilized 91 CLBs and 13 input/Output pads.

An optical configuration for the detection of weaving defects is developed and tested. The Optical Fourier Transformation is the basic working principle of the system. When good fabric passes in front of the optical system, the Fourier image captured by the camera, shows well defined spots corresponding to the spatial frequencies of the tissue. If a defect occurs during production on the loom, the pattern changes significantly and a defect is easily detected in real time. The paper does not give any information about the results and the speed of the system.
A method of detecting the surface defects of cast metals (fins and notches) is described by Y. Okawa [25]. The proposed automatic inspection consists of an industrial television camera (ITVC) and a microcomputer. A modified thresholding technique and simple pixel counting is used to detect the defects. It is stated that a large percentage of defective casts can be extracted by the proposed system. In this system the time to move the camera from one position to another was much longer than picture processing time. So the system performance is limited by the camera movement.

D.Chetverikov has employed a modified local type approach to detect defects in textures. In local approach, texture feature values are computed and compared in a sliding window to the reference values representing a perfect pattern. Such an operator gives a maximum response when the window covers an imperfection [23].

Choon-Woo Kim and A.J. Koivo in their paper [38] discuss that in designing a vision system with an automatic classifier, the effect of environment should carefully be considered. Indeed, in a dusty environment such as a wood plant, the board surfaces may be partly or entirely covered by a thin layer of dust which will affect the board grading and will decrease the accuracy of classification. There is extraneous noise in factories (such as electric sparks, smoke, dust, humidity, etc.) which can disturb the lighting conditions under which the inspection is being made. They may degrade the quality of an image in a way which can not be remedied by further computations.

A research aimed at developing an inspection system to locate and identify surface defects on wood is carried out by R.W. Conners et al [33]. The basic strategy was to divide the digital image of a board into a number of disjoint rectangular regions and classify each independently. This simple procedure has the advantage of allowing an obvious parallel processing implementation. The study shows that measures of tonal and pattern related qualities are needed. The tonal measures are the mean, variance, skewness, and kurtosis of the grey levels. The pattern related measures are those based on co-occurrence matrixes. An overall 88.3 percent correct classification is achieved on eight defects most commonly found in lumber.
When defects cause a global distortion of the basic structure of the material, Fourier analysis proves to be a suitable tool for defect detection and characterization. However, when defects are local phenomena, the Fourier analysis does not provide enough information. Methods that can localize in the spatial as well as the frequency domain appear to be necessary for the inspection of local defects. The wavelet transforms used as multi-resolution spectral filters provide both frequency and spatial information about an image. Gabor filters has also been used for the inspection of local defects in textile webs [49]. A Gabor filter consists of a sinusoid of some frequency and orientation modulated by a Gaussian function.

Morphological image processing is also used for detection of defects [42],[56]. The other approaches that are used for defect detections are Neural networks [15], cellular neural networks [50], and wavelet [9], [62].

The semiconductor industry is the biggest user of machine vision equipment, accounting for about 30% of sales, followed by the electronic industry, the container industry, wood and food. In semiconductor industry, the applications of computer vision date back to the early 1970s. These systems inspected objects with larger feature size, such as printing wiring boards and electronic assemblies. Subsequent systems were designed to inspect small objects such as chips. An important objective of much of today’s integrated circuit inspection is the early detection and identification of manufacturing process problems. Defect information obtained through inspection tools is used by process control engineers to make decisions resulting in process yield improvement.

On-line automated X-ray inspection has become the leading real-time inspection and process control tool for electronic components and assemblies. The reason for this is that visible light technology is not sufficient to inspect critical features such as volume and connectivity of solder joints. A similar trend is occurring in the semiconductor wafer inspection area where light is not sufficient to perform key measurements. Thus, with each new imaging mode (e.g. X-ray [111]), ultrasound [112] and Atomic Force Microscope [113] one has to develop algorithms that exploit the knowledge of the image formation
process to successfully recover relevant information. In most of these imaging modes, processing algorithms are not as sophisticated and well understood as those for visible light images.

Ando et al [134], from Fujitsu Laboratories, developed a new technique to inspect printed Wiring Boards (PWBs) which employs black-line sensing and a radial matching algorithms. This system can examine high density PWB patterns with high resolution. The radial matching algorithm express the pattern as a code. This is a run length code but in this case the pattern length is measured in four directions, each 45 degrees apart. Thus many kinds of patterns can be inspected simply by changing the dictionary of stored codes. The detected patterns are processed in radial matching logic units. This system can process 40 Mpixels per second with 5 μm/resolution. The system can detect defects as small as 100 μm.

J.W Roberts et al [10], [11] have designed a web inspection system that can operate at processing speeds of greater than 400 MBytes/sec. The system makes use of a novel post processing algorithm within a TDI camera itself. The scanning rate is adjustable to a maximum of 20,000 line per second. The system can use up to 12 cameras in parallel and all of the raw pixel data from the cameras is processed by a single image processing board (frame grabber). The paper does not discuss about the employed algorithm(s) and the minimum size of defect that can be detected by the system.

Corke et al [57] have developed a machine vision processing board which contains several large FPGAs for parallel computation. The board provides clock, several digital video interfaces, four switchable frame stores and a VME bus interface. The FPGA board called Configurable Logic Processor or CLP is intended to provide a highly flexible environment for simply and quickly constructing a customized processing circuit for machine vision applications. While the arithmetic capability is limited (multiplication and floating point representation consume too much logic), most vision applications can be carried out using 8-bit quantities and simple arithmetic or logic operations.
Surface quality is very important, since it can affect both the appearance and the function of a product. For example, due to dirt, scratches, dents or roughness, finished products may have a mat instead of a glossy finish, or a different colour from what is desired. Surface defect inspection has been studied extensively. Some inspection techniques model the surface on the basis of the theory of light scattering, while others are based on imaging techniques to characterize surface blemishes. Simple thresholding can be used when the light intensity of the defect is different from intensities of the light of the other points on the good surface. However, if the contrast of the defect is less than other variations in image intensity before thresholding, then the image could be filtered in order to enhance its contrast. High pass filtering is used to discard non-defect variations, while keeping the defect data. Conners et al [33] studied the inspection of rough wood surfaces. In this approach, the basic strategy was to divide the digital image of a board into a number of disjointed rectangular regions and classify each independently. This simple procedure is attractive for parallel processing implementation. Jain et al [120] addressed the problem of automatic inspection of the texture appearance of metallic finishes. They applied a multi-channel filtering technique to obtain a number of texture features, each of which captures the characteristics of the texture within a narrow range of frequencies.
Appendix C
Field Programmable Logic

C.1 Introduction
This Appendix provides a brief introduction to FPGA architecture and its applications in prototype systems. Since the fundamentals of the FPGAs for different vendors are almost the same, we will concentrate on Xilinx 4000E series architecture which is used in this project. A brief comparison between FPGA and CPLD and their applications are also given. The criteria of choosing the most suitable FPGA for a specific application is discussed by an example. FPGA implementation of the fuzzy logic algorithm which is discussed in Chapter 6 is also explained.

C.2 Architectural Support for Data Processing
The nature of image processing operations in inspection systems involves performing complex tasks repeatedly on a large set of image data. For applications where real-time performance is required, this data set may be processed at video rates of up to 25 times a second or even higher. Therefore, a fast realization of such a computationally intensive operation is essential.
Continuing advances with semiconductor technology have allowed the integration of increased functionality into general-purpose microprocessors. Today's high-performance microprocessors sport millions of transistors and include multiple functional units and large on-chip memories. Despite the fact that processor performance has steadily increased over the past decade, general-purpose processors (such as those used in high-end workstations and PCs) have failed to deliver the performance demanded by image processing applications. It has long been an observation in computing that generality and efficiency are inversely related to one another. The major inefficiency with general-purpose processors is that most of their silicon area goes into storing data and instructions and in communication and control circuitry. This is needed to support the processor's large operational diversity. Consequently, very little of the capacity inherent in a processor gets applied to the problem in hand. All of this silicon area is dedicated to allowing the computational tasks to heavily reuse a small active portion on the silicon, i.e. the Arithmetic and Logic Units (ALUs). For example, processors optimized for floating point calculations seldom use more than a few percent of the circuit when performing logical or arithmetic operations. Furthermore, this set of general-purpose instructions coupled with a fixed word width, makes general-purpose processors inefficient at processing bit or byte-level data. An alternative to the general-purpose processor for image processing is required.

To achieve this high performance, machine vision systems have moved away from the traditional approach of general-purpose computing towards systems containing special architectural support. A lot of research has been carried out into several such areas of architectural support, including parallel processing, Digital Signal Processing (DSP) processors and special purpose hardware. A brief overview of each of these approaches is given below:

C.3 Parallel processing

Like any other computationally intensive problem, parallel processing has been suggested as a possible solution. A lot of research has been carried out into parallel architectures and
programming languages. The approach behind parallel processing is to remove the physical limits of serial processors by employing several (or even thousands of) processors in parallel. In doing this, the problem can be solved in less time than it would take using only a single processor.

For image processing applications, most solutions have tended to be based upon pure SIMD (Single-Instruction-Multiple-Data) or SPMD (Single-Program-Multiple-Data) type architectures. These approaches, distributes the image over numerous Processing Elements (PEs), with all of these PEs processing its own section of the image in parallel.

However, despite the huge amount of research and the advances in parallel processing over the past decade, the field of parallel processing is still immature. Programming a parallel machine is still considered difficult and the actual performance achieved on parallel computers is often only a fraction of their theoretical peak performance.

C.4 Digital Signal Processing (DSP) processors

One method of increasing the performance of general purpose processors, is to attach a specialized processing unit in the form of DSP processor. A DSP processor is tailored to perform a specific operation (such as MAC -multiplication followed by accumulation) very fast. As a result DSP processors have been successfully used in a wide range of image processing applications. High-end workstations are dedicating more and more area to specialized DSP processors than to general purpose computing. Consequently, today’s high performance workstations exhibit an increasing disparity between the general-purpose core and its specialized DSP coprocessors. However, such processors still contain a fixed set of instructions that can only be processed sequentially one after the other. This approach merely provides an additional set of highly optimized instructions for the general-purpose processor. Simply adding fixed functional capacity will not produce the required high performance over a broad range of algorithms.
C.5 Special purpose hardware

Architectural support in the form of special purpose hardware can provide solutions that are specially tailored to the image processing algorithm in hand. There are many image processing algorithms that do not map well into the instruction-fetch-decode approach of general-purpose and DSP processors. Such operations could be implemented much more efficiently by a custom designed special purpose hardware solution using Application Specific Integrated Circuit (ASIC) technology. Such hardware solutions can perform a particular image processing operation an order of magnitude faster than its general-purpose processors counterpart. However, as with most custom hardware solutions, it is not without its disadvantages. Such solutions tend to have a long development time from design through to simulation and fabrication. They can also be expensive if it is a one-off solution or the volumes required cannot justify its fabrication. Another problem with such a solution is the fact that they are special purpose. As algorithms change or new ideas and techniques are developed, their lack of flexibility makes them problematic. With such a solution, a new piece of hardware is usually required for each new algorithm.

C.6 Reconfigurable hardware - a new computing approach

Arising in the mid 1980's, reconfigurable hardware devices in the form of Field Programmable Gate Arrays (FPGAs) have enjoyed the same advances as microprocessors in IC fabrication technology to emerge as viable system building blocks. These devices aim to combine the flexibility of a programmable device (such as a general-purpose processor) with the performance of application specific special purpose hardware (such as ASICs). Such devices have opened up an interesting space between the extremes of general-purpose computing and dedicated hardware. This space is in the domain of reconfigurable hardware.

This space is most easily understood by investigating the binding time for device functions. Special purpose hardware (such as ASICs) bind functions to active silicon at
fabrication time, making the silicon useful only for the designated function. General-purpose processors bind functions to active silicon only for the duration of a single processor cycle, limiting the amount the processor can accomplish in a single cycle. Reconfigurable hardware allow functions to be bound at a range of intervals within the final system, depending on the needs of the application. This flexibility in binding time allows reconfigurable hardware to make better use of the limited resources available. Despite their widespread use, general-purpose processors are not ideally matched to most of the applications they run. For almost any application, one can conjecture additions or modifications to the prevalent microprocessor architectures which would significantly enhance the application's performance. However, the additions differ from application to application, and there is insufficient commonality among applications to merit inclusion of such additions in a processor with a broad application base. The performance improvements gained by employing specialized coprocessors (such as DSP) demonstrate the advantages of handling a more limited application domain. Reconfigurable hardware presents an alternative technology that can adapt to the application in hand ideally with the ease of a general purpose processor, while delivering the performance advantages of a special purpose hardware solution. With this inherent speed and system adaptability, image processing seems an ideal candidate to benefit from the advantages that this type of reconfigurable hardware can offer.

C.7  Field Programmable Gate Array (FPGA)

A Field Programmable Gate Array (FPGA) consists of a large array of configurable cells (or logic blocks) contained on a single chip. Each of these cells contains a computation unit capable of implementing one of a set of logic level functions and/or perform routing to allow inter-cell communication to take place. All of these operations can take place simultaneously across the whole array of cells. The first commercially available FPGA was developed by a company called Xilinx [138] and appeared on the market in 1985. The basic architecture of an FPGA consists of a 2-D array of cells. The inter-cell communications take place through interconnection resources. The outer edge of the array
consists of special blocks capable of performing certain I/O operations to and from the chip. The architecture of a typical FPGA is illustrated in Figure 1.

The computation unit and routing configuration for each cell can be programmed via electronic programmable switches. Several technologies such as SRAM programming, Antifuse programming, Floating gate programming, are used to implement these programmable switches.

![FPGA Architecture Diagram]

**Figure C.1 FPGA architecture**

FPGA cells differ greatly in their size and implementation capacity. For example FPGA cells can range from implementing a single gate to cells containing lookup tables capable of implementing logic functions containing up to 5 inputs.

Configurable Logic Blocks implement most of the logic in an FPGA. Figure 2 illustrates the principal CLB elements in the Xilinx XC4000Es FPGAs. Two 4-input function generators (F and G) offer unrestricted versatility. Most combinatorial logic functions need four or fewer inputs. However, a third function generator (H) is provided. The H function generator has three inputs. Either zero, one, or two of these inputs can be the
outputs of F and G; the other input(s) are from outside the CLB. The CLB can, therefore, implement certain functions of up to nine variables, like parity check or expandable identity comparison of two sets of four inputs. Each CLB contains two storage elements that can be used to store the function generator outputs. However, the storage elements and function generators can also be used independently. These storage elements can be configured as flip-flops in both XC4000E and XC4000X devices; in the XC4000X they can optionally be configured as latches. DIN can be used as a direct input to either of the two storage elements. H1 can drive the other through the H function generator. Function generator outputs can also drive two outputs independent of the storage element outputs. This versatility increases logic capacity and simplifies routing. Thirteen CLB inputs and four CLB outputs provide access to the function generators and storage elements. These inputs and outputs connect to the programmable interconnect resources outside the block.

Figure C.2 Xilinx XC4000 Configurable Logic Block (CLB)
A new version of this family, the 4000E, has the additional feature that the RAM can be configured as a dual port RAM with a single write and two read ports. In the 4000E, RAM blocks can be synchronous RAM. Also, each XC4000 chip includes very wide AND-planes around the periphery of the logic block array to facilitate implementing circuit blocks such as wide decoders.

Besides logic, the other key feature that characterizes an FPGA is its interconnect structure. The XC4000 interconnect is arranged in horizontal and vertical channels. Each channel contains some number of short wire segments that span a single CLB (the number of segments in each channel depends on the specific part number), longer segments that span two CLBs, and very long segments that span the entire length or width of the chip. Programmable switches are available to connect the inputs and outputs of the CLBs to the wire segments, or to connect one wire segment to another. An important point worth noting about the Xilinx interconnect is that signals must pass through switches to reach one CLB from another, and the total number of switches traversed depends on the particular set of wire segments used. Thus, speed-performance of an implemented circuit depends in part on how the wire segments are allocated to individual signals by CAD tools.

The XC4000E and XC4000XL Series currently have 20 members, as shown in Table 1 on page 156. XC4000XL Series are the low voltage (3.3 V) version of XC4000E series.
### Table C.1. XC4000E and XC4000X Series Field Programmable Gate Array

<table>
<thead>
<tr>
<th>Device</th>
<th>Logic Cells</th>
<th>Max. Logic Gates (No RAM)</th>
<th>Max. RAM Bits (No Logic)</th>
<th>Typical Gate Range (Logic and RAM)</th>
<th>CLB Matrix</th>
<th>Total CLBs</th>
<th>Number of Flip-Flops</th>
<th>Max. User I/O</th>
</tr>
</thead>
<tbody>
<tr>
<td>XC4002XL</td>
<td>152</td>
<td>1.600</td>
<td>2.048</td>
<td>1,000 - 3,000</td>
<td>8 x 8</td>
<td>64</td>
<td>256</td>
<td>64</td>
</tr>
<tr>
<td>XC4003E</td>
<td>238</td>
<td>3.000</td>
<td>3.200</td>
<td>2,000 - 5,000</td>
<td>10 x 10</td>
<td>100</td>
<td>360</td>
<td>80</td>
</tr>
<tr>
<td>XC4003E/XL</td>
<td>466</td>
<td>5.000</td>
<td>6.272</td>
<td>3,000 - 9,000</td>
<td>14 x 14</td>
<td>196</td>
<td>616</td>
<td>112</td>
</tr>
<tr>
<td>XC4006E</td>
<td>608</td>
<td>6.000</td>
<td>8.192</td>
<td>4,000 - 12,000</td>
<td>16 x 16</td>
<td>256</td>
<td>768</td>
<td>128</td>
</tr>
<tr>
<td>XC4008E</td>
<td>770</td>
<td>8.000</td>
<td>10.368</td>
<td>6,000 - 15,000</td>
<td>18 x 18</td>
<td>324</td>
<td>936</td>
<td>144</td>
</tr>
<tr>
<td>XC4010E/XL</td>
<td>950</td>
<td>10.000</td>
<td>12.800</td>
<td>7,000 - 20,000</td>
<td>20 x 20</td>
<td>400</td>
<td>1,120</td>
<td>160</td>
</tr>
<tr>
<td>XC4013E/XL</td>
<td>1368</td>
<td>13.000</td>
<td>18.432</td>
<td>10,000 - 30,000</td>
<td>24 x 24</td>
<td>576</td>
<td>1,536</td>
<td>192</td>
</tr>
<tr>
<td>XC4025E</td>
<td>1962</td>
<td>20.000</td>
<td>25.088</td>
<td>13,000 - 40,000</td>
<td>28 x 28</td>
<td>784</td>
<td>2,016</td>
<td>224</td>
</tr>
<tr>
<td>XC4025E/XL</td>
<td>2432</td>
<td>25.000</td>
<td>32.768</td>
<td>15,000 - 45,000</td>
<td>32 x 32</td>
<td>1,024</td>
<td>2,560</td>
<td>256</td>
</tr>
<tr>
<td>XC4027E/XL</td>
<td>2432</td>
<td>28.000</td>
<td>32.768</td>
<td>18,000 - 50,000</td>
<td>32 x 32</td>
<td>1,024</td>
<td>2,560</td>
<td>256</td>
</tr>
<tr>
<td>XC4036E/XL</td>
<td>3078</td>
<td>36.000</td>
<td>41.472</td>
<td>22,000 - 65,000</td>
<td>36 x 36</td>
<td>1,296</td>
<td>3,168</td>
<td>288</td>
</tr>
<tr>
<td>XC4044XL</td>
<td>3800</td>
<td>44.000</td>
<td>51.200</td>
<td>27,000 - 80,000</td>
<td>40 x 40</td>
<td>1,500</td>
<td>3,840</td>
<td>320</td>
</tr>
<tr>
<td>XC4052XL</td>
<td>4598</td>
<td>52.000</td>
<td>61.952</td>
<td>33,000 - 100,000</td>
<td>44 x 44</td>
<td>1,936</td>
<td>4,576</td>
<td>352</td>
</tr>
<tr>
<td>XC4062XL</td>
<td>5472</td>
<td>62.000</td>
<td>73.728</td>
<td>40,000 - 130,000</td>
<td>48 x 48</td>
<td>2,304</td>
<td>5,376</td>
<td>384</td>
</tr>
<tr>
<td>XC4085XL</td>
<td>7448</td>
<td>55.000</td>
<td>100.352</td>
<td>55,000 - 180,000</td>
<td>56 x 56</td>
<td>3,136</td>
<td>7,168</td>
<td>448</td>
</tr>
</tbody>
</table>

### C.8 Arrays Characteristics of FPGAs over conventional processors

FPGAs allow computational tasks to be implemented spatially with intermediate results flowing directly from the producing cells to the receiving cells. With the large number of cells contained in a single device, FPGAs can take advantage of the inherent parallelism that is present in many algorithms. The key characteristics of an FPGA based computing engine against a conventional processor (such as general-purpose or DSP processor) are:

**Low bandwidth instruction distribution.** Instead of broadcasting a new instruction to the FPGA cells on every cycle, instructions are configured locally, allowing the FPGA to dedicate a minimal amount of cell resources to instruction distribution. The minimal amount of resources used for instruction distribution makes FPGA efficient at performing regular tasks (i.e. tasks which repeatedly perform the same collection of operations from cycle to cycle) on bit-level data.
Spatial routing of intermediate results. Space permitting, intermediate values are routed locally in parallel rather than the communication taking place in time through a central resource bottleneck.

Distributed resources. Resources such as memory, interconnections and function units are distributed as required rather than being centralized in large pools. Independent local access allows designs to take advantage of the fast and parallel on-chip bandwidth.

Narrow data path width. Almost always one-bit wide data paths. This allows the device to adapt to the problem in hand as opposed to a fixed large data path width (such as 32 or 64 bits), where on small data values, the computing resources are underutilized.

The emergence of FPGAs has lead to their use in the construction of several high performance systems. These reconfigurable systems accelerate the computationally intensive sections of many applications at a fraction of the cost of super-computers or special purpose hardware solutions. The architecture of some of these systems has been designed specifically with an application in mind (e.g. encryption/compression, physical simulations and pattern matching etc.) and some have been designed with a more general purpose architecture, allowing it to be tailored at a later date towards a particular application.

C.9 Applications of FPGAs

FPGAs have gained rapid acceptance and growth over the past decade because they can be applied to a very wide range of applications. A list of typical applications includes: random logic, integrating multiple SPLDs, device controllers, communication encoding and filtering, small to medium sized systems with SRAM blocks, and many more.

Other interesting applications of FPGAs are prototyping of designs later to be implemented in gate arrays, and also emulation of entire large hardware systems. The former of these applications might be possible using only a single large FPGA (which
corresponds to a small Gate Array in terms of capacity), and the latter would entail many
FPGAs connected by some sort of interconnect.

Another promising area for FPGA application, which is only beginning to be developed, is
the usage of FPGAs as custom computing machines. This involves using the
programmable parts to “execute” software, rather than compiling the software for
execution on a regular CPU.

C.10 CPLD versus FPGA and its applications

We will now briefly examine the types of applications which best suit Complex
Programmable Logic Devices (CPLD) architectures and their differences from FPGAs.
Because CPLDs offer high speeds and a range of capacities, they are useful for a very
wide assortment of applications, from implementing random glue logic to prototyping
small gate arrays. One of the most common uses in industry at this time, and a strong
reason for the large growth of the CPLD market, is the conversion of designs that consist
of multiple SPLDs into a smaller number of CPLDs.

CPLDs are considered logic rich because there are typically more logic gates than
registers available. The ratio of gates to registers can be as high as 5 to 1. FPGAs, on the
other hand, are register rich, with a logic to register ratio closer to 2 to 1. FPGAs logic
structure are optimized for functions narrower than those of PLDs. FPGAs have a smaller
speed quanta than do PLDs, so logic functions can be incremented in complexity while
incrementing the delay only a little each time.

CPLDs can realize reasonably complex designs, such as graphics controller, LAN
controllers, UARTs, cache control, and many others. As a general rule-of-thumb, circuits
that can exploit wide AND/OR gates, and do not need a very large number of flip-flops are
good candidates for implementation in CPLDs. A significant advantage of CPLDs is that
they provide simple design changes through reprogramming (all commercial CPLD
products are re-programmable). With in-system programmable CPLDs it is even possible
to reconfigure hardware (an example might be to change a protocol for a communications circuit) without power-down.

Designs often partition naturally into the SPLD like blocks in a CPLD. The result is more predictable speed-performance than would be the case if a design were split into many small pieces and then those pieces were mapped into different areas of the chip. Predictability of circuit implementation is one of the strongest advantages of CPLD architectures.

When designs are mapped into CPLDs, pieces of the design often map naturally to the SPLD like blocks. However, designs mapped into an FPGA are broken up into logic block-sized pieces and distributed through an area of the FPGA. Depending on the FPGA’s interconnect structure, there may be various delays associated with the interconnections between these logic blocks. Thus, FPGA performance often depends more upon how CAD tools map circuits into the chip than is the case for CPLDs.
Appendix D
DALSA Line-Scan TDI Specifications

DALSA Inc. offers several TDI line scan cameras that vary in the number of PELs (Pixels per line) and data rate and line rate. PELs can be 512, 1024, 2048 and data rate is in the range of 15 MHz to 120 MHz. The maximum line rate can be between 6.9 KHz up to 90.0 KHz. The number of TDI stages is 96 for most of the sensors. The most important input/output signals of the camera are described below:

- MCLK (Master Clock) MCKL is supplied to generate all internal camera timing. Data rate is derived from this signal. DALSA TDI line scan cameras have an internally generated MCLK, therefore an externally provided MCLK is not normally required.

- EXSYNC (External Sync) The EXSYNC is an edge triggered input which initiates readout from the camera. Line scan and TDI cameras will not output a line until a low (0) to high (1) clock is provided. EXSYNC should be triggered from an encoder or similar device to synchronize the line scan and TDI line scan cameras to the moving object of interest.

- LVAL (Line Valid) Line Valid is an output clock on all DALSA cameras and is sometimes referred to as horizontal synchronization signal. With line scan and TDI line scan cameras, LVAL high (1) indicates a valid line of pixels is being transmitted from the camera.
- PVAL (Pixel Valid) Pixel Valid is used to clock the analog output from DALSA's analog output model cameras. The frequency is equal to the data rate. Pixels are valid on the rising edge of the PVAL.

![Figure D.1 Camera Timing](image)

Figure D.1 Camera Timing
-- VHDL Program for defective line detection
using fuzzy fusion of different techniques.

Description
library IEEE;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

package ROMS is
  -- declare a 16x16 ROM called ROM
  constant ROM_WIDTH: INTEGER := 16;
  subtype ROM_WORD is STD_LOGIC_VECTOR (1 to ROM_WIDTH);
  subtype ROM_RANGE is INTEGER range 0 to 15;
  type ROM_TABLE is array (0 to 15) of ROM_WORD;
  constant ROM: ROM_TABLE := ROM_TABLE'(
    ROM_WORD'("1010110101101011"),
    ROM_WORD'("1010111000010101"),
    ROM_WORD'("111110101101011"),
    ROM_WORD'("1010111111110101"),
    ROM_WORD'("1000010101101011"),
    ROM_WORD'("1010110101101011"),
    ROM_WORD'("111110101101011"),
    ROM_WORD'("111110101101011"),
    ROM_WORD'("1010110101101011"),
    ROM_WORD'("1000010101101011"),
    ROM_WORD'("111110101101011"),
    ROM_WORD'("111110101101011"),
    ROM_WORD'("111110101101011"),


ROM_WORD'("1000010101101011"),
ROM_WORD'("1010110101101011"),
ROM_WORD'("111110101101011"),
ROM_WORD'("1111110101011011"));
end ROMS;

use work.ROMS.all; -- Entity that uses ROM

library IEEE;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity processor is
port(CLK: in STD_LOGIC;
    -- CLK: in STD_LOGIC;
    RESET: in STD_LOGIC;
    AD: in UNSIGNED(7 downto 0);
    BD: in UNSIGNED(7 downto 0);
    LVAL: in STD_LOGIC;
    PVAL: in STD_LOGIC;
    --BITRD: in STD_LOGIC_VECTOR(15 downto 0);
    --BITRA: out STD_LOGIC_VECTOR(3 downto 0);
    VAR0: in UNSIGNED(7 downto 0);
    VAR1: in UNSIGNED(7 downto 0);
    VAR2: in UNSIGNED(7 downto 0);
    VAR3: in UNSIGNED(7 downto 0);
    VAR4: in UNSIGNED(7 downto 0);
    VAR5: in UNSIGNED(7 downto 0);
    VAR6: in UNSIGNED(7 downto 0);
    VAR7: in UNSIGNED(7 downto 0);
    OUTDATA: out STD_LOGIC_VECTOR(26 downto 0);
    PUSH: out STD_LOGIC;
    CAPTUREON: in STD_LOGIC;
    OKTOPUSH: in STD_LOGIC);
end processor;

architecture behaviour of processor is
signal PROCLINE: STD_LOGIC;
signal LVALSIG: STD_LOGIC;
signal NEWLVALSIG: STD_LOGIC;
signal PVALSIG: STD_LOGIC;
signal TOBESYNCED: STD_LOGIC;
signal CURRENT: UNSIGNED(7 downto 0);
signal FIRSTCURRENT: STD_LOGIC;
signal COUNTER: UNSIGNED(10 downto 0);
signal TEMP0: UNSIGNED(9 downto 0);
signal TEMP1: UNSIGNED(10 downto 0);
signal INDEX: STD_LOGIC_VECTOR(1 downto 0);

-- My added variables
signal PRECURRENT: SIGNED(8 downto 0) := "010101010";

signal SUMLINECURRENT: UNSIGNED(17 downto 0) := "00000000000000000";
signal THRESHOLD: UNSIGNED(4 downto 0) := "11001";
signal NUMJUMPS: UNSIGNED(2 downto 0) := "000";
signal OUTROM: STD_LOGIC_VECTOR(15 downto 0);
signal FLAG: STD_LOGIC;
signal NET: STD_LOGIC;

signal BITRD: STD_LOGIC_VECTOR(15 downto 0);
signal BITRA: STD_LOGIC_VECTOR(3 downto 0);
--signal CLK: STD_LOGIC := '0';

-- AVELINECURRENT is the average of gray levels in a line scanned by the camera
-- SUMLINEPIEXL is the summation of 1024 pixels in a line.

begin

-- CLK <= NOT CLK after 50 ns;
BITRD <= ROM( CONV_INTEGER(BITRA));

TEMP0 <= "0" & VAR2(0) & VAR0;
TEMP1 <= "00" & VAR2(1) & VAR1;
THRESHOLD <= VAR3(4 downto 0);

camera_procl: process begin
  wait until CLK'event and CLK='0';

  CURRENT <= AD;
  LVALSIG <= LVAL;
  PVALSIG <= PVAL;

end process camera_procl;

camera_procd: process
variable AVELINECURRENT: UNSIGNED(7 downto 0);
variable AC_AVELINECURRENT: UNSIGNED (4 downto 0);
begin

wait until CLK'event and CLK='0';

if (PVALSIG='0' and CAPTUREON='1') then

    OUTDATA(25 downto 24) <= (others => '0');
    OUTDATA(26) <= TOBESYNCEDE;

    if (LVALSIG='1') then
        NET <= '0';

        if ((FIRSTCURRENT='1' and OKTOPUSH='0') or
            PROCLINE='0') then

            PROCLINE <= '0';
            INDEX <= "00";
            PUSH <= '0';
            TOBESYNCEDE <= '0';
            FIRSTCURRENT <= '0';

        else

            COUNTER <= COUNTER + 1;

            if ( (COUNTER >= (1023-TEMP0)) and
                 (COUNTER <= (1024 + TEMP1)) ) then

                FIRSTCURRENT <= '0';
                PROCLINE <= '1';

        -- Calculation of the average gray level of each line
        begins:
            SUMLINECURRENT <= SUMLINECURRENT+CURRENT;
            if ( (ABS (PRECURRENT-CURRENT) >=
                THRESHOLD) and (NUMJUMPS < 7 ) ) then
                NUMJUMPS<=NUMJUMPS+1;
            end if;
            PRECURREN<=CONV_SIGNED(CURRENT,9);

        case INDEX is
            when "00" =>
                OUTDATA(7 downto 0) <=
                CONV_STD_LOGIC_VECTOR(CURRENT,8);
                INDEX <= "01";

            when others =>

        end case;

        NUMJUMPS<=NUMJUMPS+1;

    end if;

end if;

end if;

end if;

end if;

end if;
PUSH <= '0';
when "01" =>
  OUTDATA(15 downto 8) <=
CONV_STD_LOGIC_VECTOR(CURRENT, 8);
  INDEX <= "10";
  PUSH <= '0';
when "10" =>
  OUTDATA(23 downto 16) <=
CONV_STD_LOGIC_VECTOR(CURRENT, 8);
  INDEX <= "00";
  PUSH <= '1';
  TOBESYNCED <= '0';
when others =>
  NULL;
end case;
else
FIRSTCURRENT <= '0';
PROCLINE <= '1';
PUSH <= '0';
end if;

end if;

else
  if (TOBESYNCED='0') then
    AVELINECURRENT := SUMLINECURRENT(17
downto 10);
    AC_AVELINECURRENT := AVELINECURRENT
(4 downto 0);
    BITRA <= (STD_LOGIC_VECTOR
(AC_AVELINECURRENT(3 downto 0)));
    NET <= '1';
  end if;
  if (NET='1') then
    FLAG <=
BITRD(CONV_INTEGER(AC_AVELINECURRENT(4) &
CONV_STD_LOGIC_VECTOR(NUMJUMPS, 3)));
    NET <= '0';
  end if;
end if;
if ( (not (INDEX = "00")) and TOBESYNCED='0')

) then

    PUSH <= '1';
else

    PUSH <= '0';
end if;

INDEX <= "00";
TOBESYNCED <= '1';
FIRSTCURRENT <= '1';
PROCLINE <= '1';
COUNTER <= (others => '0');

end if;
else

    PUSH <= '0';
    SUMLINECURRENT <="000000000000000000" ; --End of processing averaging pixel values of each line
    NUMJUMPS<="000";
end if;

end process camera_proc3;

end behaviour;
function dalsademo(action,s);
%*****************************************************************************
*****************************************************************************
% DALSA Project.
% This program is written by S.Hossain Haji-mowlan in Feb. 1996.
% University of Windsor
*****************************************************************************
*****************************************************************************
% Possible actions:
% initialize
% image
% operation
% button callbacks:
% Gray
% Brighten
% Equalize
% Filtering
% Contrast
% Histogram
% Profile
% Roicolor
% Colorbar
% Salt Noise
% Blur
% info
%************

global def_image
pl=Path;
path(pl,'La Cie 1060-Q:Hossain:DALSA:Matlab')

if nargin<1,
action='initialize';
end;

if strcmp(action, 'initialize'),
  figNumber=figure( ... 
    'Name', 'DALSA Project Demo', ..., 
    'NumberTitle', 'off');

%============================================= 
% Set up the image axes
axes( ...
  'Units', 'normalized', ..., 
  'Position', [0.10 0.35 0.6 0.6], ..., 
  'XTick', [], 'YTick', [], ..., 
  'Box', 'on');
set(figNumber, 'defaultaxesposition',[0.10 0.35 0.60 0.6])

%============================================= 
% Set up the Comment Window
top=0.25;
left=0.05;
right=0.725;
bottom=0.05;
labelHt=0.035;
spacing=0.005;
% First, the MiniCommand Window frame
frmBorder=0.02;
frmPos=[left-frmBorder bottom-frmBorder ... 
  (right-left)+2*frmBorder (top-bottom)+2*frmBorder];
uicontrol( ... 
  'Style', 'frame', ..., 
  'Units', 'normalized', ..., 
  'Position', frmPos, ..., 
  'BackgroundColor', [0.8 0.6 0.7]);
% Then the text label
labelPos=[left top-labelHt (right-left) labelHt];
uicontrol( ... 
  'Style', 'text', ..., 
  'Units', 'normalized', ..., 
  'Position', labelPos, ..., 
  'BackgroundColor', [0.7 0.7 0.45], ..., 
  'ForegroundColor', [1 1 1], ..., 
  'String', 'Comment Window');
% Then the editable text field
txtPos=[left bottom (right-left) top-bottom-labelHt- 
spacing];
txtHndl=uicontrol( ... 'Style','edit', ... 'Units','normalized', ... 'Max',10, ... 'BackgroundColor',[1 1 1], ... 'Position',txtPos, ... 'String',str2mat(' Choose a defected image to process', ... ' from the pop-up menu (e.g., Defect No.?).' ));
% Save this handle for future use
set(gcf,'UserData',txtHndl);

%============================================================================
% Information for all buttons (and menus)
labelColor=[0.8 0.8 0.8];
yInitPos=0.90;
menutope=0.95;
top=0.75;
left=0.785;
btnWid=0.175;
btnHt=0.06;
% Spacing between the button and the next command's label
spacing=0.025;

%============================================================================
% The CONSOLE frame
frmBorder=0.02;
yPos=0.05-frmBorder;
frmPos=[left-frmBorder yPos btnWid+2*frmBorder 0.9+2*frmBorder];
h=uicontrol(...
  'Style','frame', ...
  'Units','normalized', ...
  'Position',frmPos, ...
  'BackgroundColor',[0.45 0.45 0.45]);

%============================================================================
% The IMAGE Menu
menuNumber=1;
yPos=menutope-(menuNumber-1)*(btnHt+spacing);
label-
Str='320_0|453E1|453E2|453E3|453E_0|453FB10|453F_B10|453F_B
UB';
callbackStr='dalsademo(''image'');';

% Generic button information
btnPos=[left ypos-btnHt btnWid btnHt];
imageHndl=uicontrol(
    'Style','popupmenu', ...
    'Units','normalized', ...
    'Position',btnPos, ...
    'String',labelStr, ...
    'Interruptible','yes', ...
    'Callback',callbackStr);

%======================================================================
% The OPERATION Menu
menuNumber=2;
yPos=menutop-(menuNumber-1)*(btnHt+spacing);
labelStr='Modify Image|Process Image|Analyze Image';
callbackStr='dalsademo(''operation''');'

% Generic button information
btnPos=[left ypos-btnHt btnWid btnHt];
opHndl=uicontrol( ...
    'Style','popupmenu', ...
    'Units','normalized', ...
    'Position',btnPos, ...
    'String',labelStr, ...
    'Interruptible','yes', ...
    'Callback',callbackStr);

%======================================================================
% button 1
btnNumber=1;
yPos=top-(btnNumber-1)*(btnHt+spacing);
labelStr='Brighten';
callbackStr='dalsademo(''Brighten''');'

% Generic button information
btnPos=[left ypos-btnHt btnWid btnHt];
btn1Hndl=uicontrol( ...
    'Style','pushbutton', ...
    'Units','normalized', ...
    'Position',btnPos, ...
    'String',labelStr, ...
    'Callback',callbackStr);

%======================================================================
% button 2
btnNumber=2;
yPos=top-(btnNumber-1)*(btnHt+spacing);
labelStr='Darken';
% Setting userdata to 1 (=play) will continue the demo.
callbackStr='dalsademo(''Darken'');'

% Generic button information
btnPos=[left yPos-btnHt btnWid btnHt];
btn2Hnd1=uicontrol( ...
  'Style','pushbutton', ...
  'Units','normalized', ...
  'Position',btnPos, ...
  'String',labelStr, ...
  'Callback',callbackStr);

%==================================%button 3
btnNumber=3;
yPos=top-(btnNumber-1)*(btnHt+spacing);
labelStr='Blur';
% Setting userdata to -1 (=stop) will stop the demo.
callbackStr='dalsademo(''Blur'');'

% Generic button information
btnPos=[left yPos-btnHt btnWid btnHt];
btn3Hnd1=uicontrol( ...
  'Style','pushbutton', ...
  'Units','normalized', ...
  'Position',btnPos, ...
  'String',labelStr, ...
  'Callback',callbackStr);

%==================================%button 4
btnNumber=4;
yPos=top-(btnNumber-1)*(btnHt+spacing);
labelStr='Filtering';
% Setting userdata to -1 (=stop) will stop the demo.
callbackStr='dalsademo(''Filtering'');'

% Generic button information
btnPos=[left yPos-btnHt btnWid btnHt];
btn4Hnd1=uicontrol( ...
  'Style','pushbutton', ...
  'Units','normalized', ...
  'Position',btnPos, ...
  'String',labelStr, ...
  'Callback',callbackStr);

%==================================
% button 5
btnNumber=5;
yPos=top-(btnNumber-1)*(btnHt+spacing);
labelStr='Contrast';
% Setting userdata to -1 (=stop) will stop the demo.
callbackStr='dalsademo(''Contrast'');'';

% Generic button information
btnPos=[left ypos-btnHt btnWid btnHt];
btn5Hndl=uicontrol(...
    'Style','pushbutton', ...
    'Units','normalized', ...
    'Position',btnPos, ...
    'String',labelStr, ...
    'Callback',callbackStr);

%==========================================
% The INFO button
labelStr='Info';
callbackStr='dalsademo(''info'');'
helpHndl=uicontrol(...
    'Style','pushbutton', ...
    'Units','normalized', ...
    'Position',[left bottom+btnHt+spacing btnWid btnHt], ...
    'String',labelStr, ...
    'Callback',callbackStr);

%==========================================
% The CLOSE button
labelStr='Close';
callbackStr='close(gcf)';
closeHndl=uicontrol(...
    'Style','pushbutton', ...
    'Units','normalized', ...
    'Position',[left bottom btnWid btnHt], ...
    'String',labelStr, ...
    'Callback',callbackStr);

hndlList=[txtHndl imageHndl opHndl btn1Hndl btn2Hndl
        btn3Hndl btn4Hndl ...
        btn5Hndl helpHndl closeHndl];
set(figNumber, ...
    'Visible','on', ...
    'UserData',hndlList);

set(gcf,'Pointer','watch');
drawnow
load Windsor
imshow(X,map)

% dalsademo('operation')
set(gca,'Pointer','arrow');
return
end

% Protect against possible deletion all the whole figure
danger = strcmp(get(gca,'NextPlot'),'replace');

if strcmp(action,'image'),
    if danger,
        set(gca,'nextplot','add');
        h = get(gca,'children');
        for i=1:length(h),
            if strcmp(get(h(i),'type','axes'), delete(h(i)), end
        end
    end

    axHnd1(gca;
    hndlList(get(gca,'UserData');
    txtHnd1=hndlList(1);
    imageHnd1=hndlList(2);
    opHnd1=hndlList(3);
    btnHandls = hndlList(4:8);
    helpHnd1=hndlList(9);
    closeHnd1=hndlList(10);
    v = get(imageHnd1,'value');
    name = get(imageHnd1,'String');
    name =
    deblank(name(v,:));
    commentStr=str2mat([' load ',name,' imshow(X,map)']);
    set(txtHnd1,'String',commentStr);

    set(gca,'Pointer','watch');
    load(name)
    imshow(X,map)
    set(gca,'Pointer','arrow');
    return

elseif strcmp(action,'operation'),
    axHndl(gca;
    hndlList(get(gca,'UserData');
    txtHnd1=hndlList(1);
    imageHnd1=hndlList(2);
    opHnd1=hndlList(3);
    btnHndls = hndlList(4:8);
    helpHnd1=hndlList(9);
closeHndl=hn1List(10);
v = get(opHndl,'value');
op = get(opHndl,'String'); op = deblank(op(v,:));
set(btnHndls(2),'Interruptible','no');
if strncmp(op,'Modify Image'),
    labels = str2mat('Brighten','Darken','Blur','Filtering','Contrast');
elseif strncmp(op,'Analyze Image'),
    labels = str2mat('Histogram','Profile','Roicolor','Colorbar','EX**');
    set(btnHndls(2),'Interruptible','yes');
elseif strncmp(op,'Process Image'),
    labels = str2mat('Zoom','Whole Image','Add Salt Noise','Detect Defects','Save as TIFF');
else
    error('Unknown operation');
end

for i=1:length(btnHndls),
callbackStr = ['dalsademo(','',deblank(labels(i,:)),'');'];
    set(btnHndls(i),'String',deblank(labels(i,:)),'Callback',callbackStr)
end

commentStr=str2mat(' % Press a button to start an operation. ...',
    ' % The commands executed by each operation will', ...
    ' % display in this window.');
set(txtHndl,'String',commentStr);
return

elseif strncmp(action,'closehelp'),
    % Restore close button help behind helpfun's back
    ch = get(gcf,'ch');
    for i=1:length(ch),
        if strncmp(get(ch(i),'type'),'uicontrol'),
            if strncmp(lower(get(ch(i),'String')),'close'),
                callbackStr = get(ch(i),'callback');
                k = findstr(';',dalsademo(','callbackStr);
                callbackStr = callbackStr(1:k-1);
                set(ch(i),'callback',callbackStr)
                break;
            end
        end
    end
    end
    ch = get(0,'ch');
if ~isempty(find(ch==s)), figure(s), end % Make sure figure exists

else % Get common information for buttons below
    if danger,
        haxes = [];
        set(gcf,'nextplot','add');
        h = get(gcf,'children');
        for i=1:length(h),
            if strcmp(get(h(i),'type'),'axes'), haxes = [haxes,h(i)]; end
        end
    end
    axHndl=gca;
    hndlList=get(gcf,'UserData');
    txtHndl=hndlList(1);
    imageHndl=hndlList(2);
    opHndl=hndlList(3);
    btnHndls = hndlList(4:8);
    helpHndl=hndlList(9);
    closeHndl=hndlList(10);
    h = get(axHndl,'children');
    set(gcf,'pointer','watch')
    if strcmp(get(h(1),'type'),'image'),
        X = get(h(1),'cdata');
        map = colormap;
        if all(get(h(1),'UserData')==[0 1])
            I = (X-1)/(size(map,1)-1); grayimage = 1;
        else
            grayimage = 0;
        end
    else
        v = get(imageHndl,'value');
        name = get(imageHndl,'String'); name = deblank(name(v,:));
        load(name)
        grayimage = 0;
        if danger,
            delete(haxes), imshow(X,map), danger = 0;
        else
            imshow(X,map)
        end
    end
end

%---------------------------- BUTTON CALLBACKS -----------------------------
if strcmp(action,'Gray'),
    if ~grayimage,
        commentStr = str2mat(' I = ind2gray(X,map);','
imshow(I,64);'
    set(txtHndl,'String',commentStr), drawnow
    if danger, delete(haxes), end
    I = ind2gray(X,map);
imshow(I,64)
end

elseif strcmp(action,'Brighten')
    commentStr = ' brighten(.2)';
    set(txtHndl,'String',commentStr), drawnow
    brighten(.2)
    set(gcf,'pointer','arrow')

elseif strcmp(action,'Darken')
    commentStr = ' brighten(-.2)';
    set(txtHndl,'String',commentStr), drawnow
    brighten(-.2)
    set(gcf,'pointer','arrow')

elseif strcmp(action,'Zoom')
    commentStr = ' Zoom';
    set(txtHndl,'String',commentStr), drawnow
    zoom
    set(gcf,'pointer','arrow')

elseif strcmp(action,'Whole Image')
    commentStr = ' Whole Image';
    set(txtHndl,'String',commentStr), drawnow
    zoom out
    set(gcf,'pointer','arrow')

elseif strcmp(action,'Detect Defects')
    x=size(X);
    str1=' Detecting the defects is underway. Please be patient. :=)';
    str2=['Image size is ' sprintf('%d',x(1,1)) ' by ' sprintf('%d',x(1,2))];
    commentStr=str2mat(str1,str2);
    set(txtHndl,'String',commentStr), drawnow
    [num_row,num_col,def_image,map,def_size]=detect(X);
imshow(def_image,256)
imshow(def_image)
    comp_ratio=(num_col)*(num_row)/(def_size);
st1 = ['Compression Ratio = ' sprintf('%8.3f',comp_ratio)];
st2= ['Internal FIFO size = ' sprintf('%d',def_size) ' by 8 Wow!...'];
commentStr=str2mat(st1,st2);
set(txtHnd1,'String',commentStr), drawnow
set(gca,'pointer','arrow')

elseif strcmp(action,'Equalize')
  if grayimage,
    commentStr = ' histeq(I,64)';
    set(txtHndl,'String',commentStr), drawnow
    if danger, delete(haxes), end
    histeq(I,64)
  else
    commentStr = ' histeq(X,map)';
    set(txtHndl,'String',commentStr), drawnow
    if danger, delete(haxes), end
    histeq(X,map)
  end

elseif strcmp(action,'Filtering')
  if grayimage,
    commentStr = str2mat(' h = fspecial(''sobel'');', ...
    ' J = mat2gray(filter2(h,I));',' imshow(J,64)');
    set(txtHndl,'String',commentStr), drawnow
    h = fspecial('sobel');
    J = mat2gray(filter2(h,I));
    if danger, delete(haxes), end
    imshow(J,64)
  else
    commentStr = str2mat(' I = ind2gray(X,map); % Convert image to gray', ' h = fspecial(''sobel'');',' J = mat2gray(filter2(h,I));',' imshow(J,64)');
    set(txtHndl,'String',commentStr), drawnow
    I = ind2gray(X,map);
    h = fspecial('sobel');
    J = mat2gray(filter2(h,I));
    if danger, delete(haxes), end
    imshow(J,64)
  end

elseif strcmp(action,'Contrast')
  if grayimage,
    commentStr = str2mat(' h = fspecial(''unsharp'');', ...
' J = mat2gray(filter2(h,I),[0 1]);', 'imshow(J,64)');
    set(txtHndl,'String',commentStr), drawnow
    h = fspecial('unsharp');
    J = mat2gray(filter2(h,I),[0 1]);
    if danger, delete(haxes), end
    imshow(J,64)
else
    commentStr = str2mat(' I = ind2gray(X,map); % Convert image to gray', ...
      ' h = fspecial(''unsharp'');', ' J = mat2gray(filter2(h,I),[0 1]);', '...
      ' imshow(J,64)');
    set(txtHndl,'String',commentStr), drawnow
    I = ind2gray(X,map);
    h = fspecial('unsharp');
    J = mat2gray(filter2(h,I),[0 1]);
    if danger, delete(haxes), end
    imshow(J,64)
end

elseif strcmp(action,'Histogram')
    if grayimage,
      commentStr = ' imhist(I,128)';
      set(txtHndl,'String',commentStr), drawnow
      if danger, delete(haxes), end
      imhist(I,128)
    else
      commentStr = ' imhist(X,map)';
      set(txtHndl,'String',commentStr), drawnow
      if danger, delete(haxes), end
      imhist(X,map)
    end

elseif strcmp(action,'Profile')
    commentStr = str2mat(' % Profile displays the intensity profile', '...
            % along a line in the image. Select the ', '...
            % endpoints of line using mouse. End with <CR>.', 'improfile');
    set(txtHndl,'String',commentStr), drawnow
    name = get(gcf,'Name');
    set(gcf,'Name','Profile')
    improfile
    set(gcf,'Name',name)

elseif strcmp(action,'Roicolor')
if grayimage,
    commentStr = str2mat(' X = gray2ind(I,64); cm =
    [gray(64);[1 0 0]];','...
    ' d = find(roicolor(I,.25,.5)); X(d) =
    65*ones(size(d));',' ... 
    ' imshow(X,cm)');
    set(txtHndl,'String',commentStr), drawnow 
    X = gray2ind(I,64); cm = [gray(64);[1 0 0]]; 
    d = find(roicolor(I,.25,.5));
    X(d) = 65*ones(size(d));
    if danger, delete(haxes), end 
    imshow(X,cm)
else
    commentStr = str2mat( ...
    ' v = min(size(map,1),255); cm = [map(1:v,:);[1 1
    0]];',' ..., 
    ' d = find(roicolor(X,130,240)); X(d) =
    (v+1)*ones(size(d));',' ... 
    ' imshow(X,map)');
    set(txtHndl,'String',commentStr), drawnow 
    v = min(size(map,1),255); 
    cm = [map(1:v,:);[1 1 0]]; 
    d = find(roicolor(X,130,240));
    X(d) = (v+1)*ones(size(d));
    if danger, delete(haxes), end 
    imshow(X,map)
end

elseif strcmp(action,'Save as TIFF')
    v = get(imageHndl,'value');
    name = get(imageHndl,'String'); name =
    deblank(name(v,:));
    filename=[name 'out'];
    tiffwrite(def_image,map,filename) 
    commentStr=['Saves the result in filename '.tif file'];
    set(txtHndl,'String',commentStr), drawnow

elseif strcmp(action,'Colorbar')
    commentStr = str2mat(' % Add colorbar to existing
image.',' ... 
    ' colorbar');
    set(txtHndl,'String',commentStr), drawnow 
    if danger, set(gcf,'nextplot','replace'), end 
    colorbar

elseif strcmp(action,'Add Salt Noise')

if grayimage,
    commentStr = str2mat(' % Add salt and pepper noise to image.', ... 
    ' J = imnoise(I,''salt & pepper'');', 'imshow(J,64)');
    set(txtHnd1,'String',commentStr), drawnow
    J = imnoise(I,'salt & pepper');
    if danger, delete(haxes), end
    imshow(J,64)
else
    commentStr = str2mat(' I = ind2gray(X,map); % Convert image to gray', ... 
    ' % Add salt and pepper noise to image.', ...
    ' J = imnoise(I,''salt & pepper'');', 'imshow(J,64)');
    set(txtHnd1,'String',commentStr), drawnow
    I = ind2gray(X,map);
    J = imnoise(I,'salt & pepper');
    if danger, delete(haxes), end
    imshow(J,64)
end

elseif strcmp(action,'Blur')
    if grayimage,
        commentStr = str2mat(' h = fspecial(''gaussian'', [5 5]);', ' ... 
        ' J = filter2(h,I);', 'imshow(J,64)');
        set(txtHnd1,'String',commentStr), drawnow
        h = fspecial('gaussian',[5 5]);
        J = filter2(h,I);
        if danger, delete(haxes), end
        imshow(J,64)
    else
        commentStr = str2mat(' I = ind2gray(X,map); % Convert image to gray', ' ...
        ' h = fspecial(''gaussian'', [5 5]);', ' J = filter2(h,I);', ' ... 
        ' imshow(J,64)');
        set(txtHnd1,'String',commentStr), drawnow
        I = ind2gray(X,map);
        h = fspecial('gaussian',[5 5]);
        J = filter2(h,I);
        if danger, delete(haxes), end
        imshow(J,64)
    end

elseif strcmp(action,'info'),
    set(gcf,'pointer','arrow')
ttlStr = get(gcf, 'Name');
hlpStr1 = [ ... ' '];
hlpStr2 = [ ... ' '];

myFig = gcf;
helpfun(ttlStr, hlpStr1, hlpStr2);

% Protect against gcf changing -- Change close button behind
% helpfun's back
ch = get(gcf, 'ch');
for i=1:length(ch),
    if strcmp(get(ch(i), 'type'), 'uicontrol'),
        if strcmp(lower(get(ch(i), 'String')), 'close'),
            callbackStr = [get(ch(i), 'callback') ... ' ; dalsademo(''closehelp'', num2str(myFig) '')];
            set(ch(i), 'callback', callbackStr)
            return
        end
    end
end
return
end % if strcmp(action, ...

set(gcf, 'pointer', 'arrow')
Choose a defected image to process from the pop-up menu (e.g., Defect No.?).

Figure F.1 GUI simulation software written in Matlab (main menu)
Figure F.2 GUI simulation software written in Matlab

```matlab
% % Author: S. Hossain Hajimowlana % e-mail: hajimow@uwindsor.ca % 8th Aug. 1997 %------------------------------------------- clear clg format short %bardata1 was created by lin_bar.m program. load bardata1 % created by lin_bar.m num_row=size(pixl,2)/num_col; % zmpixl means zero mean pixl. zmpixl=pixl-mean(pixl); plot(pixl(550:600),'m') grid pause [x,z]=hist(pixl(1:num_col),20);```
nz_ele=find(x);
for l=1:size(nz_ele,2)
    x(nz_ele(l))=x(nz_ele(l))./x(nz_ele(l));
end
z=x.*z;
t=0;
n=7;
thd=1.1;
row=1;
no_err=n;
Al=lpc(zmpixl(1:2000),n-1);
Mn=mean(Dpixl);
zmDpixl=Dpixl;
hold on
plot(Dpixl(550:600),'y')
pause
hold off
for j=0:num_row-1
    for i=1:num_col+1-n
        t=j*num_col+i;
        s=0;
        s=sum(Al(2:n).*fliplr(zmDpixl(t:t+n-2)));
        err(t)=((zmDpixl(n+t-1)+(s))/(zmDpixl(n+t-1)));
        if abs(err(t))>thd
            Min=-s-(z(nz_ele(l)));
            for k=2:size(nz_ele,2)
                if abs(Min)>abs(-s-(z(nz_ele(k))))
                    Min=-s-(z(nz_ele(k)));
                end
            end
            %gg=round(-(-Min+s));
            zmDpixl(n+t-1)=round(-(-Min+s));
        end
    end
end

% The following three lines are used if we want to show the
% effect of not adjusting the pixel level for defects
% if abs(err(t))>thd
%    zmDpixl(n+m)=-s;
%end

if abs(err(t))>=thd & rem(t+n-1,num_col)==0
    com_data(row,1)=num_col;
    com_data(row,2)=Dpixl(t-1+n);
    row=row+1;
    com_data(row,1)=0;
com_data(row,2)=0;
row=row+1;
no_err=n;
elseif abs(err(t))>=thd & rem(t+n-1,num_col)>0
    com_data(row,1)=rem(t+n-1,num_col);
    com_data(row,2)=Dpix1(t-1+n);
    row=row+1;
elseif abs(err(t))<thd & rem(t+n-1,num_col)==0 &
    rem(no_err,num_col)==0
    com_data(row,1)=num_col;
    com_data(row,2)=0;
    row=row+1;
    no_err=n;
elseif abs(err(t))<thd & rem(t+n-1,num_col)>0
    no_err=no_err+1;
elseif abs(err(t))<thd & rem(t+n-1,num_col)==0 &
    rem(no_err,num_col)>0
    com_data(row,1)=0;
    com_data(row,2)=0;
    row=row+1;
    no_err=n;
else
end
end
%err=zeros(1,n-1) err;
plot(abs(err(550+n-1:600+n-1)),'r')
grid
save comp_back1 com_data num_col num_row

**********
%---------------------------------------------------------
% Author: S. Hossain Hajimowlana
% e-mail: hajimow@uwindsor.ca
% 6th Aug. 1997
%
% Zero order background tracking
% This program gets the data from lpc_bar program
% and reconstructs the image. The result will just
% show the defects.
%---------------------------------------------------------
load comp_back1
% In the next line first the background is generated.
R_img=22*ones(num_row,num_col);
n_col=1;
for i=1:size(com_data,1)
    if com_data(i,:)==[128,0]
n_col=n_col+1;
elseif com_data(i,:)==[0 0]
    n_col=n_col+1;
else
    R_img(n_col,com_data(i,1))=com_data(i,2);
end

end
max(max(R_img));
colormap(gray)
M=max(max(R_img));
m=min(min(R_img));
y=((R_img-m)*64/(M-m)*4);
image(R_img);
grid on
%colormap(gray)
tiffwrite(R_img,colormap,'result.tif')

*************
function y=genimg(typ,sizex,sizey,frqx,frqy)

% GENIMG Generates one of 11 different kinds of images
% useful as test inputs for wavelet transformation
% and multiresolution analysis.
%
% Y = GENIMG (TYP) generates a 128x128 image of
% type TYP.
%
% Y = GENIMG (TYP,SIZX,SIZY) forces the image to be
% SIZX x SIZY sized.
%
% Y = GENIMG (TYP,SIZX,SIZY,FRQX,FRQY) changes the
% frequencies used to generate the images. The
default
% value is 1 for both axis.
%
% See also: WTDEMO, SHOW, WT2D.

--------------------------------------------------------------------------------
% (c) Copyright 1994, by Universidad de Vigo
% under GNU conditions.
% Author: Sergio J. Garcia Galan
% e-mail: Uvi_Wave@tsc.uvigo.es
--------------------------------------------------------------------------------
if nargin<4
    frqx=1;
    frqy=1;
end;

if nargin<2
    sizex=128;
    sizey=128;
end;

frqx=128/sizex*frqx;
frqy=128/sizey*frqy;

if typ==0,
    disp('Wavelet test image');
    y=zeros(sizey,sizex);
    a=floor(sizex/(8*frqx));
    b=floor(sizey/(8*frqy));
    y(b+1,a+1:sizex-a)=ones(1,sizex-2*a);
    y(sizey-b,a+1:sizex-a)=ones(1,sizex-2*a);
    y(b+1:sizey-b,a+1)=ones(sizey-2*b,1);
    y(b+1:sizey-b,sizex-a)=ones(sizey-2*b,1);
    db=(sizey-2*b)/(sizex-2*a);
    b=b+1;
    for i=a+1:sizex-a-1,
        y(b,i)=1;
        y(sizey-b,i)=1;
        b=b+db;
    end;
end;

if typ==1,
    disp('Sharp wavy image');
    frqx=frqx*0.1;
    frqy=frqy*0.1;
    for j=1:sizey
        a(j,:)=linspace(0,sizex*frqx,sizex);
    end
    for x=1:sizex
        a(:,x)=a(:,x).*[linspace(0,sizey*frqy,sizey)]';
    end
    y=sin(a);
end;
if typ==2,
    disp('Diagonal image');
    frqx=frqx*1;
    frqy=frqy*1;
    for y=1:sizey
        a(y,:)=linspace(0,sizex*frqx,sizex);
    end
    for x=1:sizex
        a(:,x)=a(:,x)+[linspace(0,sizey*frqy,sizex)]';
    end
    y=sin(a);
    size(y)
end;

if typ==3,
    disp('Horizon image');
    frqx=frqx*0.5;
    frqy=frqy*0.04;
    for y=1:sizey
        a(y,:)=linspace(0,sizex*frqx,sizex);
    end
    for x=1:sizex
        a(:,x)=a(:,x)/([linspace(0,sizey*frqy,sizex])'+1);
    end
    y=sin(a);
end;

if typ==4,
    disp('Square filled image');
    frqx=frqx*0.5;
    frqy=frqy*0.5;
    for y=1:sizey
        a(y,:)=linspace(0,sizex*frqx,sizex);
    end
    a=sin(a);
    for x=1:sizex
        b(:,x)=[linspace(0,sizey*frqy,sizex)]';
    end
    b=sin(b);
    y=a+b;
end;

if typ==5,
disp('Growing squares image');
frqx=frqx*0.05;
frqy=frqy*0.05;
for y=1:sizey
    a(y,:)=linspace(0,sizex*frqx,sizex);
    a(y,:)=a(y,:).*a(y,:);
end
a=sin(a);
for x=1:sizex
    b(:,x)=[linspace(0,sizex*frqy,sizex)]';
    b(:,x)=b(:,x).*b(:,x);
end
b=sin(b);
y=a+b;
end;

if typ==6,
disp('Dust among bars image');
frqx=frqx*0.25;
frqy=frqy*0.25;
for y=1:sizey
    a(y,:)=linspace(0,sizex*frqx,sizex);
end
for x=1:sizex
    a(:,x)=a(:,x)+([linspace(0,sizex*frqy,sizey)]'+1);
end
y=sin(a.^2);
end;

if typ==7,
disp('Wavy diagonals image');
frqx=frqx*0.05;
frqy=frqy*0.05;
for y=1:sizey
    a(y,:)=linspace(0,sizex*frqx,sizex);
end
for x=1:sizex
    a(:,x)=a(:,x)+([linspace(0,sizey*frqy,sizey)]'+1);
end
y=sin(a.^2)+cos(a);
end;
if typ==8,
    disp('Round wavy 1 image');
    frqx=frqx*0.05;
    frqy=frqy*0.05;
    for y=1:sizey
        a(y,:)=linspace(0,sizex*frqx,sizex);
    end
    for x=1:sizex
        a(:,x)=a(:,x).*([linspace(0,sizey*frqy,sizey)]');
    end
    y=sin(a.^2)+cos(a);
end;

if typ==9,
    disp('Round wavy 2 image');
    frqx=frqx*0.05;
    frqy=frqy*0.05;
    for y=1:sizey
        a(y,:)=linspace(0,sizex*frqx,sizex);
    end
    for x=1:sizex
        a(:,x)=a(:,x).*([linspace(0,sizey*frqy,sizey)]');
    end
    y=sin(a)+cos(a.^2);
end;

if typ==10,
    disp('Multiple square image');
    frqx=frqx*0.1;
    frqy=frqy*0.1;
    for y=1:sizey
        a(y,:)=linspace(0,sizex*frqx,sizex);
    end
    for x=1:sizex
        b(:,x)=([linspace(0,sizey*frqy,sizey)]');
        a(:,x)=a(:,x)+b(:,x);
    end
    y=cos(sin(b.^2));
    y=y+y';
end;

**********
function [num_row,num_col]=detect(X)
    pic_size=size(X);
    num_row=pic_size(1);
    num_col=pic_size(2);
% The pixel value and its position is saved in an n by 2 savemat
% matrix
    thsd=130;
    saverow=0;
% defect is equal to 1 every time a defect is found and is
% each time it is on the background
    defect=0;
    for nr=0:num_row-1
        for nc=1:num_col
            pixl(nr*num_col+nc)=X(nr+1,nc);
        end
    end
    stackpix=pixl(1);
    pixlprev=pixl(1);
    for n=1:num_row*num_col
        diff=pixl(n)-pixlprev;
        aa=abs(pixl(n)-stackpix);
        if (defect==1 & abs(pixl(n)-stackpix) > thsd )
            saverow=saverow+1;
            pixnum=n;
            savemat(saverow,1)=pixnum;
            savemat(saverow,2)=pixl(n);
            pixlprev=pixl(n);
        elseif ( defect==0 & (abs(diff)>thsd ))
            defect=1;
            stackpix=pixlprev;
            saverow=saverow+1;
            pixnum=n;
            savemat(saverow,1)=pixnum;
            savemat(saverow,2)=pixl(n);
            pixlprev=pixl(n);
        else
            defect==0;
            pixlprev=pixl(n);
        end
    end

************
% This program will read an image and calculate statistical
% properties of it such as average gray level and number of
% jumps
% in a line.
% The program was written by Hossain Hajimowlana in June 11
1998.
% The program was updated for testing DALSA real defects
March 31 1999

clear
[x,map]=tiffread('La Cie 1060-Q:Users:Hossain:DALSA:Real
defective samples:DALSADETECTSS.tif');
[numrow,numcol]=size(x);
mmn=125;
for i=1:numrow
    sumgray=0;
    numjump=0;
    first_pixel=x(i,1);
    for j=7:numcol
        xx=(x(i,j)+x(i,j-1)+x(i,j-2)+x(i,j-3)+x(i,j-4)+x(i,j-5)+x(i,j-6))/7;
        sumgray=x(i,j)+sumgray;
        if abs(xx-mmn)>100
            numjump=numjump+1;
            first_pixel=x(i,j);
        else
            first_pixel=x(i,j);
        end
    end
    sumgrayline(i,1)=sumgray;
avegrayline(i,1)=sumgray/(numcol-6);
numjumpline(i,1)=numjump;
save datal
end

******
Figure F.3 Fuzzy logic editor environment in Matlab

Figure F.4 A defective sample
Figure F.5 The result after applying fuzzy logic algorithm to defective sample shown in Figure F.4

Figure F.6 A defective sample
Figure F.7 The result after applying fuzzy logic algorithm to defective sample shown in Figure F.6
Appendix G

FPGA Selection for Target Application

G.1 FPGA implementation of the algorithms

Two different methods have been developed to design and implement algorithms for this research. They are:

1. HDL programming.
2. Schematic design and CAD tools.

G.1.1 HDL programming:

HDL is a mnemonic for Hardware Description Language. As are all languages, there are predefined rules and symbols which allow a design to accurately describe a desired system. The language can then be translated into the often complex workings at the physical level.

HDL programming allows the designer to ignore the internal coding details and focus on the desired logic. In conjunction with FPGA, this can become a very powerful tool. Synthesis and mapping tools can be easily used by an experienced HDL designer to turn code into a custom circuit on the FPGA.
The use of an HDL allows the designer to model any concept in a manner which is easier to understand by others than a design using schematic logic. HDL makes it easier for logic libraries to be linked. VHDL was chosen as an HDL for creating the necessary algorithm.

G.2 Selecting the Best FPGA for a Target Application

The key to compare FPGA solutions is to highlight all issues that can significantly impact the end product and compare how employing the design with each member of a specified FPGA family address these concerns. Timing is one of the major issues in FPGA implementation. A design might be implemented by any members of a target FPGA family. The timing characteristics and the maximum operating frequency can be different for each implementation. Circuit boards must be built prior to FPGA completion to minimize schedule time. This requires manual definition of I/O location for the FPGA. In many FPGA architecture, manual I/O assignment can significantly impact the performance and capacity of a device. After selecting our target series of FPGA (e.g. XC4000 series) we have to choose an FPGA from the family that best suits our design. As an example, if we design the board for the 84 pin series of Xilinx XC4000 series FPGAs, we will have the following options to choose our FPGA from:

XC4002, XC4003, XC4004, XC4005, XC4006, XC4008, XC4010. The number of CLBs range from 64 to 400.

Our experiments show that the most suitable FPGA from a family of FPGAs for an application is the one that 45 to 60 percent of its resources is utilized. For example if a design needs 60 CLBs, the best performance will be obtained by using an FPGA that has 100 to 125 CLB. This theory is verified by implementing several different logic circuits.

Figure G.1 shows the propagation delay for a circuit of seven 16-bit U/D counters that is implemented with 56 CLBs. The I/O pins are defined randomly before implementation. It shows that for XC4002 the delay is the highest and for XC4003, XC4004, and XC4005
gives the best result. Using denser FPGAs will increase the propagation delay as the signal should pass more switches to reach the designated I/O.

![Graph showing propagation delay for Xilinx XC4000 family members]

**Figure G.1 Propagation delay for a circuit of seven 16-bit U/D counter**

### G.2.1 Virtex Series FPGAs

The VirtexTM family from Xilinx redefines the future of programmable logic by giving the next generation FPGAs that break density and performance barriers while offering unprecedented system level integration. Virtex series devices range from 50,000 to 1,000,000 system gates at clock speeds up to 160 MHz, and include many new features that address system level design challenges.

Virtex has over 500 user I/O with many package options. It operates with 2.5-volt with 5-volt tolerant I/Os. The Virtex series has transformed the FPGA from its former role as a
"glue logic" device into the industry's first programmable solution that can serve as the board-level center for system design. With the Virtex series, digital designers for the first time can use an FPGA to perform not only familiar logic functions, but also tasks that were formerly handled at the board level by separate, dedicated parts. The Virtex series eliminates the need for components such as phase lock loops, voltage translation buffers, and memory when on-chip RAM is sufficient. This high level of integration allows designers to reduce overall system power requirements, cut costs, and save board space.

The Virtex CLB implements logic using four independent four-input lookup tables, four independent set/reset registers, multiplexer logic, and specialized arithmetic logic. The CLB was developed in parallel with the synthesis tools for the Virtex Series to guarantee high performance results when used with VHDL and Verilog design methodologies. Complex logic such as 32-bit arithmetic functions, pipelined multiplication, and 64-to-1 multiplexing can be easily described in a high level language and will operate above 100 MHz in any Virtex series device.
Vita Auctoris

Sayed Hossain Hajimowlana (‘61, Tehran, Iran) received his B.Sc. degree from the Amir Kabir University of Technology (Tehran Polytechnic) Tehran-Iran, and his Master’s degree from KNT University of Technology, Tehran-Iran in Electrical Engineering in 1986 and 1991. During his master’s thesis research work at Aviation Research Center, he analyzed and designed a curved structure rotary joint for a radar system in X-band.

In Nov. 1991 he joined Sima Electronic company as a hardware design engineer where he was involved in designing different analog/digital electronic modules in low frequency and RF.

In September 1993 he started his Ph.D. program at VLSI Research Group, University of Windsor. He has held research and teaching assistantship positions in Electrical Engineering since then. His current area of research includes, Machine vision, Industrial applications of FPGA, Sonet, ATM switches, and RF circuits. He has been a member of the Professional Engineers of Ontario, P.Eng. since May 1998.