Multi-Camera Computational Video Architecture

Proefschrift voorgelegd tot het behalen van de graad van doctor in de informatica

Andy Motten

Promotor: prof. dr. ir. Luc Claesen
Acknowledgements

It has been a long journey, with lot of excitement, successes and also frustrations. I was fortunate to be surrounded by fantastic people, not only in my professional life, but also in my private life. Without their advice and support, I would not be able to finish this dissertation.

First and foremost, I would like to express my very great appreciation to prof. dr. ir. Luc Claesen for his valuable and constructive suggestions during the planning and development of this research work. His willingness to give his time so generously has been very much appreciated. I am also very grateful to him for giving me the opportunity and support to work at the Zhejiang University (ZJU) located in Hangzhou.

In Hangzhou, I had the opportunity to meet many interesting people. In particular I wish to thank prof. dr. Yun Pan for accepting me into his group. His continuous support and advice were very helpful. I feel grateful to have experienced a different research culture and want to give thanks to the complete team at ZJU for providing me with this great experience. I am also very grateful to prof. dr. Xiaolang Yan for giving me the chance and support to work at ZJU.

Special thanks to my thesis committee, prof. dr. Frank Van Reeth, prof. dr. Philippe Bekært and prof. dr. Yun Pan for their support, guidance and helpful suggestions. I also want to address special thanks to prof. dr. Ruohong Huan for her detailed and valuable comments with respect to this dissertation. Thanks also to the other members of my jury: prof. dr. Terng-Yin Hsu, prof. dr. ir. Dirk Stroobandt and Prof. dr. Jean Manca.
I am also very grateful to prof. dr. Eddy Flerackers, our managing director, and prof. dr. Frank Van Reeth, our deputy managing director, for giving me the opportunity to work at the Expertise Centre for Digital Media (EDM). A special thanks goes to Peter Vandoren, Roger Claes and Ingrid Konings for helping me with all administrative tasks.

I would like to thank all my colleagues from EDM for being so welcoming and supportive. In particular dr. Anastasiia Beznosyk, Qian Li and dr. Nasim Mahmud for sharing the PhD experience with me. I would also like to thank my colleagues from ZJU, by providing me with a different view on scientific research which I will always treasure. In particular dr. Yimu Wang and Ning Zheng for the interesting conversations and discussions.

My gratitude also extends to all my friends who where always there for me when I needed some time off, especially Lily Tang, Tsang Fai Tang, Ronald Vossen, Ray Feng, dr. Osman Ali, dr. Abdellatif Bey-Temsamani, prof. dr. Tegoeh Tjahjowidodo and Christophe Raskin.

I want to thank my parents, André Motten and Mariette Schepers, and my complete family for giving me the opportunity to follow my dreams. Without their continuous support, I would not be able to be the man that I am now.

Most importantly, I want to thank my love Pui Si Lee for her continuous support in the past decade. Her love and patience kept me going when I was feeling overwhelmed and uncertain. She has been the silent force within me throughout this long journey.
Abstract

Image sensors have become ubiquitous. Adding extra image sensors to a vision system does not need to be costly. Processing all this information however, is non-trivial. The goal of this dissertation is to investigate a multi camera architecture especially designed for processing multiple image sensors for single chip implementation. This research aims to demonstrate and formalize better understanding on the design aspects and trade-offs. This will require the design, realization and implementation of dedicated hardware structures and architectures on a Field Programmable Gate Array (FPGA).

A real-time stereo vision system is selected as use case for this dissertation. Instead of using one camera pair, the goal is to use multiple camera pairs. By using the cooperative results to obtain a good and robust depth information instead of pursuing the best results for each camera pair separately.

By providing an architecture for multi camera systems and their collaboration, we have shown that it is possible to implement them into a single chip implementation. To illustrate this approach, a depth camera has been developed which is working according to this principle. It is a true single chip implementation.
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contents</td>
<td>V</td>
</tr>
<tr>
<td>List of Figures</td>
<td>IX</td>
</tr>
<tr>
<td>List of Tables</td>
<td>XV</td>
</tr>
</tbody>
</table>

### 1 Introduction
- 1.1 Problem Statement and Motivation | 1
- 1.2 Research Approach | 2
- 1.3 Summary of Contributions | 3
- 1.4 Thesis Structure | 4

### I Pixel Stream Processing

#### 2 Basic Concepts
- 2.1 Introduction | 9
- 2.2 Pipelined Parallel Processing | 9
- 2.3 Pixel Stream Processing | 10
  - 2.3.1 System Input and Output | 11
  - 2.3.2 Demosaicing | 13
  - 2.3.3 Median Filter | 15
- 2.4 Multi Stream Processing: Stereo Vision as Use Case | 16
3 Image Sensor Alignment

3.1 Introduction .............................................. 19
3.2 Warping Overview ........................................ 21
  3.2.1 Reverse Mapping Coordinates .................... 21
  3.2.2 Storage of the Source Image ..................... 22
  3.2.3 Sub-Pixel Resampling .............................. 22
  3.2.4 Examples ........................................... 24
3.3 Storage of Reverse Mapping Coordinates ................. 25
3.4 Memory Architecture .................................... 32
  3.4.1 Circular Buffer ..................................... 34
  3.4.2 Split Circular Buffer ............................... 35
  3.4.3 Adaptive Split Circular Buffer ................... 36
  3.4.4 Circular Buffer with Line Read Postponing .... 38
  3.4.5 Adaptive Split Circular Buffer with Line Read Postponing .... 40
3.5 Hardware Architecture .................................. 43
3.6 Implementation .......................................... 50
3.7 Conclusion .............................................. 51

4 Tiled Parallel Memory Architecture .......................... 53

4.1 Introduction .............................................. 54
4.2 On-Chip Memory Blocks .................................. 55
4.3 Line Buffers ............................................. 56
4.4 Parallel Memory Architecture ............................. 56
4.5 Window Translation ....................................... 60
4.6 Modified Parallel Memory Architecture .................. 61
4.7 Implementation and Results .............................. 63
4.8 Conclusion .............................................. 63

5 Pixel Stream Comparison .................................... 65

5.1 Introduction .............................................. 65
5.2 Similarity Computation ................................... 67
5.3 Adaptive Support Weights ................................ 71
  5.3.1 Original Algorithm ................................. 71
5.3.2 Binary Support Weights ........................................ 73
5.3.3 Results and Performance .................................... 76
5.3.4 Complexity .................................................. 78
5.4 Implementation ................................................. 81
5.5 Conclusion ..................................................... 86

II Multi-Camera Vision Systems .................................. 87

6 Pattern Recognition, Binary Confidence Evaluation .......... 89
  6.1 Introduction ................................................... 89
  6.2 Confidence Metrics .......................................... 90
    6.2.1 Confidence Metrics Based on the Cost Function .......... 91
    6.2.2 Consistency Between Left and Right Disparity Maps .... 92
    6.2.3 Confidence Metrics Based on Neighbourhood Information .. 92
    6.2.4 Confidence Metrics Based on Multiple Pixel Streams .... 94
  6.3 Investigated Classification Methods .......................... 94
    6.3.1 One Feature Threshold .................................. 95
    6.3.2 Decision Tree ........................................... 95
    6.3.3 Artificial Neural Network ................................ 96
  6.4 Hierarchical Classification ................................... 97
  6.5 Zoning ...................................................... 97
  6.6 Classification of Disparity Values ............................ 98
  6.7 Conclusion .................................................. 105

7 Architecture Generation Framework ............................... 107
  7.1 Introduction ................................................ 107
  7.2 Framework Overview ......................................... 108
  7.3 Example Design Flow ........................................ 111
  7.4 Synchronisation ............................................. 112
  7.5 Conclusion .................................................. 114

8 Real World Feasibility and Evaluation ............................ 115
  8.1 Introduction ................................................ 115
  8.2 Binocular Stereo Camera ..................................... 117
    8.2.1 System Design ......................................... 118
8.2.2 Implementation ........................................ 123
8.2.3 Conclusion ............................................ 123
8.3 Trinocular Stereo Camera .................................. 126
  8.3.1 General Architecture .................................. 127
  8.3.2 Hierarchical Classification .............................. 129
  8.3.3 System Design ........................................ 133
  8.3.4 Implementation ...................................... 138
  8.3.5 Conclusion ............................................ 139
8.4 Evaluation ................................................ 141
8.5 Conclusion ................................................ 143

9 Conclusion and Future Research Directions 145
  9.1 Conclusion .............................................. 145
  9.2 Future Research Directions ............................... 146

Scientific Contributions and Publications 147

Samenvatting (Dutch Summary) 151

Glossary 153

Bibliography 157
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Global Overview</td>
</tr>
<tr>
<td>2.1</td>
<td>Adder calculation (left: sequential, right: tree-based parallel)</td>
</tr>
<tr>
<td>2.2</td>
<td>Pipelined tree-based parallel adder</td>
</tr>
<tr>
<td>2.3</td>
<td>Example basic camera system</td>
</tr>
<tr>
<td>2.4</td>
<td>Camera system without frame buffer</td>
</tr>
<tr>
<td>2.5</td>
<td>Bayer pattern</td>
</tr>
<tr>
<td>2.6</td>
<td>Basic demosaicing algorithm</td>
</tr>
<tr>
<td>2.7</td>
<td>Linear interpolation based demosaicing algorithm (Malvar et al. [2004])</td>
</tr>
<tr>
<td>2.8</td>
<td>Odd/Even transposition sort network using a nine-stage pipeline (CS: Compare and Swap element, D: Delay element)</td>
</tr>
<tr>
<td>2.9</td>
<td>Median filter for usage on RAW pixel stream (red centre pixel)</td>
</tr>
<tr>
<td>2.10</td>
<td>Binocular camera system</td>
</tr>
<tr>
<td>2.11</td>
<td>Alligned binocular camera system</td>
</tr>
<tr>
<td>2.12</td>
<td>Alligned binocular camera system, both images are placed on top of each other</td>
</tr>
<tr>
<td>3.1</td>
<td>Image warping</td>
</tr>
<tr>
<td>3.2</td>
<td>Reverse mapping coordinates (left: source image, middle: vertical reverse mapping coordinates, right: warped image)</td>
</tr>
<tr>
<td>3.3</td>
<td>Warping of input to output pixel stream using a circular buffer</td>
</tr>
</tbody>
</table>
3.4 Reverse mapping coordinates using sub-pixel resampling (left: source image, middle: vertical reverse mapping coordinates, right: warped image). .................................................. 23
3.5 Sub-pixel resampling (top: without resampling, bottom: with resampling). ................................................................. 23
3.6 Warping examples, from top to bottom: wave, expand and compress and barrel distortion (left: source image, middle: vertical reverse mapping coordinates, right: warped image). ........................................ 24
3.7 Vertical reverse mapping coordinates regular grid. .............. 27
3.8 Regular grid based mapping. ............................................. 27
3.9 Irregular grid based mapping. .......................................... 28
3.10 Three main structures of an irregular grid. ......................... 31
3.11 Vertical reverse mapping coordinates irregular grid. .......... 33
3.12 Vertical reverse mapping coordinates without known pattern, boxes indicate the memory access envelope. ......................... 34
3.13 Circular buffer. ......................................................... 35
3.14 Vertical reverse mapping coordinates with wave pattern. .... 35
3.15 Split circular buffer. ................................................... 36
3.16 Vertical reverse mapping coordinates with changing wave pattern. .......................... 37
3.17 Vertical reverse mapping coordinates with changing wave pattern. ........... 37
3.18 Adaptive split circular buffer. ....................................... 38
3.19 Vertical reverse mapping coordinates with decreasing random pattern. .................. 39
3.20 Reordering of blanking lines. ...................................... 39
3.21 Vertical reverse mapping coordinates with decreasing random pattern, modified for line read postponing. ....................... 40
3.22 Vertical reverse mapping coordinates for barrel distortion. ...... 41
3.23 Vertical reverse mapping coordinates for barrel distortion. ...... 41
3.24 Vertical reverse mapping coordinates for barrel distortion, modified for line read postponing. .................................. 42
3.25 Vertical reverse mapping coordinates for barrel distortion, modified for line read postponing. .................................. 43
3.26 Vertical reverse mapping coordinates for barrel distortion, modified for line read postponing. .................................. 44
3.27 Vertical reverse mapping coordinates for barrel distortion, modified for line read postponing. .......................... 45
3.28 Vertical reverse mapping coordinates for stereo rectification. .......... 46
3.29 Vertical reverse mapping coordinates for stereo rectification. .......... 46
3.30 Image warping architecture. ........................................ 47
3.31 Input buffer architecture. ........................................... 47
3.32 Vertical reverse mapping coordinates with changing wave pattern. ... 48
3.33 Vertical reverse mapping coordinates with changing wave pattern. ... 48
3.34 Image rectification results before (top) and after (bottom) warping for a stereo camera. ........................................ 52

4.1 Line buffer implementation for a support region of $3 \times 3$. ............ 56
4.2 Complete image from camera. ........................................ 57
4.3 Parallel memory organisation (colour of blocks indicates contents of memory - pixel colour). ........................................ 57
4.4 Parallel memory implementation. ...................................... 59
4.5 Timed shared window read by increasing the frequency of the read clock. 60
4.6 Window orientation mismatch. ........................................ 61
4.7 Window orientation translation. ....................................... 62
4.8 Modified parallel memory implementation: global data write connec-
tions (left) and close-up of one memory element (right). .................. 62

5.1 TAD parameters for the Middlebury dataset (average equals the sum of the datasets divided by four). .................. 68
5.2 TSD parameters for the Middlebury dataset (average equals the sum of the datasets divided by four). .................. 69
5.3 Error percentage in function of window size for the different similarity calculations. ........................................ 70
5.4 Disparity map quality of the Tsukuba stereo pair in function of window size and dissimilarity computation type (from left to right: 5×5, 11×11, 23×23, 35×35 window size). ........................................ 72
5.5 Spatial distance (size: $35 \times 35$). ................................... 74
5.6 Adaptive binary weights, threshold parameter. .......................... 75
5.7 Adaptive binary weights for a window size of $35 \times 35$ (left: colour window, middle: ABW, right: ABW with connectivity preservation.) . 75
5.8 Extending ABW with connection preservation. .............................. 76
5.9 Adaptive support weights, squared area around $p$ (size: 35 × 35). .... 77
5.10 Error percentage in function of window size for the different adaptive support weights .................................................................................................................... 78
5.11 Disparity map quality in function of dataset and adaptive support type
(from left to right: Cones, Teddy, Tsukuba and Venus dataset). .... 79
5.12 Adaptive support weights with separate vertical and horizontal aggregation step, squared area around $p$ (size: 35 × 35). .................. 80
5.13 Absolute difference. ................................................................. 81
5.14 Truncated absolute difference. .................................................... 82
5.15 Chroma difference. ................................................................. 82
5.16 Connectivity preservation. ......................................................... 83
5.17 Cost aggregation, $q_t \in N(p_t), q_r \in N(p_r)$. ............................... 83
5.18 Minimum cost selection. ........................................................... 84

6.1 Window comparison cost function. ................................................. 91
6.2 Different support regions. ............................................................ 93
6.3 One feature threshold. ............................................................... 95
6.4 Decision tree example: two-dimensional feature space (left) and resulting DT (right). .............................................................. 96
6.5 Artificial neural network (Structure: 5 - 3 - 2 - 1). ........................ 97
6.6 Hierarchical classification. .......................................................... 98
6.7 Example of different timing zones for classification. ....................... 99
6.8 Disparity map quality of the investigated datasets (Sawtooth, Tsukuba, Teddy) using fixed weights size of 11 × 11 (black pixels indicates a confidence of zero). ...................................................... 103
6.9 Disparity map quality of the investigated datasets (Sawtooth, Tsukuba, Teddy) using adaptable binary weights size of 11 × 11 (black pixels indicates a confidence of zero). ............................ 104

7.1 Framework for the development of multi-camera vision systems. .... 109
7.2 Testbench for behavioural simulation (left: ModelSim HDL simulation, right: Matlab script execution). ................................................. 110
7.3 Testbench for hardware verification, gray area indicates the SUT. .... 110
7.4 Example instance pre-processing module. ..................................... 111
List of Figures

7.5 Example instance window comparison module. .............................. 112
7.6 Unsynchronised adders. ............................................................... 113
7.7 Unsynchronised window comparison. Only valid when all blocks are
processed in one clock cycle. .......................................................... 113
7.8 Synchronised window comparison. ............................................... 114
8.1 Binocular stereo setup. ................................................................. 117
8.2 Binocular stereo, global architecture. .......................................... 118
8.3 Binocular stereo, pre-processing module. .................................... 118
8.4 Binocular stereo, misalignment (grey lines are the mismatched epipolar
lines) .............................................................................................. 119
8.5 Binocular stereo, window comparison module. ............................... 120
8.6 Binocular stereo, window comparison of different data streams. .... 121
8.7 Binocular stereo, post-processing module. .................................... 122
8.8 Binocular stereo, confidence calculation. ....................................... 122
8.9 Binocular stereo, hardware block diagram. ................................... 124
8.10 Binocular stereo, disparity map quality of the Tsukuba Stereo Pair for
a 7 × 7 binaire adaptive window: without confidence metric (top-left),
with DT confidence metric (top-right), one iteration of disparity re-
finement (bottom-left), ten iterations of disparity refinement (bottom-
right) (black pixels indicate a confidence of zero). ........................... 125
8.11 Trinocular stereo setup. ................................................................. 127
8.12 Trinocular stereo, global architecture. .......................................... 128
8.13 Trinocular stereo, different window matching. ............................. 129
8.14 Trinocular stereo, hierarchical classification. ............................... 129
8.15 Trinocular stereo, disparity map quality of the Tsukuba dataset for a
fixed window size of 7 × 7 after the first level of classification (black
pixels indicate a confidence value of zero). ..................................... 132
8.16 Trinocular stereo, disparity map quality of the investigated datasets
(Tsukuba, Teddy, Art). Comparison of Disparity Space Image (DSI)
generated from \( CL_0 \) data stream (Top row) and DSI generated from
the combination of \( CL_0, CL_1, CR_0 \) and \( CR_1 \) data streams (Bottom
row). .......................................................................................... 134
8.17 Trinocular stereo, pre-processing module. ................................... 135
8.18 Trinocular stereo, window comparison module. . . . . . . . . . . . . . . 136
8.19 Trinocular stereo, window comparison of different data streams. . . . . 137
8.20 Trinocular stereo, hardware block diagram. . . . . . . . . . . . . . . . 140
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Demosaicing resource usage.</td>
<td>15</td>
</tr>
<tr>
<td>2.2</td>
<td>Median filter resource usage.</td>
<td>16</td>
</tr>
<tr>
<td>3.1</td>
<td>Example storage for the reverse mapping coordinates</td>
<td>29</td>
</tr>
<tr>
<td>3.2</td>
<td>Reverse mapping coordinates number of bits to store</td>
<td>32</td>
</tr>
<tr>
<td>3.3</td>
<td>Memory allocation for individual slices and configurations.</td>
<td>45</td>
</tr>
<tr>
<td>3.4</td>
<td>Memory allocation for individual blocks</td>
<td>47</td>
</tr>
<tr>
<td>3.5</td>
<td>Resource usage for the rectification of a stereo camera.</td>
<td>51</td>
</tr>
<tr>
<td>4.1</td>
<td>Resource usage using different memory architectures</td>
<td>63</td>
</tr>
<tr>
<td>5.1</td>
<td>Algebraic complexity for different similarity calculations.</td>
<td>70</td>
</tr>
<tr>
<td>5.2</td>
<td>Algebraic complexity for different adaptable weight calculations</td>
<td>78</td>
</tr>
<tr>
<td>5.3</td>
<td>Logic element usage, without spatial distance region support selection.</td>
<td>85</td>
</tr>
<tr>
<td>5.4</td>
<td>Logic element usage, with spatial distance region support selection.</td>
<td>85</td>
</tr>
<tr>
<td>6.1</td>
<td>One Feature Threshold</td>
<td>101</td>
</tr>
<tr>
<td>6.2</td>
<td>Decision Tree (Part 1)</td>
<td>102</td>
</tr>
<tr>
<td>6.3</td>
<td>Decision Tree (Part 2)</td>
<td>103</td>
</tr>
<tr>
<td>6.4</td>
<td>Artificial Neural Network</td>
<td>104</td>
</tr>
<tr>
<td>8.1</td>
<td>Trinocular stereo, synthesis results overview</td>
<td>126</td>
</tr>
<tr>
<td>8.2</td>
<td>Trinocular stereo, first level classification</td>
<td>131</td>
</tr>
<tr>
<td>Table</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>-------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>8.3</td>
<td>Trinocular stereo, first level classification</td>
<td>134</td>
</tr>
<tr>
<td>8.4</td>
<td>Trinocular stereo, synthesis results overview</td>
<td>139</td>
</tr>
<tr>
<td>8.5</td>
<td>Comparison of real-time performance and resource usage of published stereo vision systems implemented on FPGA.</td>
<td>142</td>
</tr>
<tr>
<td>8.6</td>
<td>Comparison of real-time performance and resource usage of published stereo vision systems implemented on FPGA.</td>
<td>143</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Problem Statement and Motivation

Dedicated hardware realizations of computational video- and vision systems have the potential to be advantageous for larger scale problems, higher resolutions, higher frame rates, lower power consumption, miniaturization and portability. In order not to jeopardise these advantages with respect to other more straightforward solutions, such design needs to be done very carefully, especially in order to efficiently use the limited hardware resources available.

In this aspect, this dissertation proposes novel contributions which will reduce the memory footprint of the on-chip hardware, and this without requiring external memories, as is the case in previous approaches.

Image sensors have become ubiquitous. Adding extra image sensors to a vision system does not need to be costly. Processing all this information however, is non-trivial. The goal of this dissertation is to investigate a multi camera architecture especially designed for processing multiple image sensors for single chip implementation. This research aims to demonstrate and formalize better understanding on the design aspects and trade-offs. This will require the design, realization and implementation of dedicated hardware structures and architectures on a Field Programmable Gate Array (FPGA).

A real-time stereo vision system is selected as the use case for this dissertation.
Instead of using one camera pair, the goal is to use multiple camera pairs. By using the cooperative results to obtain a good and robust disparity space image Disparity Space Image (DSI) instead of pursuing the best results for each camera pair separately.

### 1.2 Research Approach

Figure 1.1 illustrates the overall architecture of a "correlated imaging depth vision system" and its composition in terms of subsystems. Each camera in the imaging system requires a number of pre-processing steps, of which camera warping is the most memory consuming. The on-chip memory needed, in traditional approaches, for multiple image sensors can easily exhaust all available resources on an FPGA. In Chapter 3 we investigate several techniques to reduce the memory usage of image warping subsystem. In this architecture, the main processing part is shared between different camera pairs using a time multiplexing approach. The new tiled memory approach proposed in Chapter 4 has been especially developed for allowing the time sharing of resources. The comparison of pixel streams originating from different cameras is done by using a window comparison module, which is explained in Chapter 5. The usage of multiple camera pairs can give multiple depth results per corresponding pixel. Methods to combine these are presented in Chapter 6.

![Figure 1.1: Global Overview.](image)

Architectures are generated according to the general template of Figure 1.1. Con-
constructing specific architectures and adapting this to different requirements, parameters and image sensors is a difficult and error prone task. To solve this, in Chapter 8 we present a framework that has been developed as part of this thesis to generate and test several variances of a chosen architecture.

1.3 Summary of Contributions

As part of the underlying research in this thesis, new reconfigurable hardware architectures and building blocks, as well as their generic hardware design methodology, for generic video and vision systems are presented. Instances of the architectures proposed, designed with the new framework based methods, have been experimented in actual efficient hardware implementations on main-stream FPGA’s. The architectures and methods proposed enable a modular approach in the development of new computational vision systems. Depending on the computational video or vision application at hand, a trade-off has to be made between the implementations of subsystems in hardware and other subsystems in software.

A short explanation of the main contributions of this dissertation are as follows:

**Memory Efficient Image Warping.** Image warping is necessary to align the multiple image sensors. A new memory allocation technique is introduced which takes the underlying pattern of the non-alignments into account. Depending on the warping this allows for a reduction of the required memory resources of up to 63% in comparison to a straightforward implementation.

**Tiling Parallel Memory Architecture.** Most vision architectures use a combination of line-buffers with shift registers to store and retrieve specific regions of images to be processed. The traditional scan line based FIFO architecture does not fully exploit the parallelism that is available with multiple on-chip memory blocks. We have implemented a new tiled based parallel System-on-Chip (SoC) memory architecture. It allows for parallel access to all pixels located in a chosen window. Using this architecture, a complete window refresh in one clock cycle is possible.

**Adaptive Binary Support for Window Matching.** When matching pixels in an image, a region around the pixel is used to make the matching more robust. Se-
lection of the ideal support region is important: it should be large enough to contain distinguished features, but small enough to maintain depth discontinuities. We have implemented a novel binary adaptable support region for incorporation in a window stereo matching SoC architecture. For each pixel, an adaptable region is generated which selects the supporting pixels in the cost aggregation phase of the stereo algorithm. This selection is performed by using colour similarity and spatial distance metrics.

Confidence Metric and Hierarchical Classification for Pixel Streams. A methodology to construct a binary confidence value for every pixel of a disparity map is proposed. Such binary confidence classifiers are needed for a higher quality stereo depth image generation in hardware. This method has also been extended for multiple pixel stream processing in a cooperative multi camera system.

Real-Time Low-Latency Single-Chip Multi-Camera Depth Measurement. A depth camera is constructed which uses the methods described in this dissertation. Compared with the state-of-the-art, it is a true one-chip solution for obtaining depth information from several image sensors. An experimental trinocular correlation based depth camera system has been realized without requiring external memories. This demonstrates a very cost-efficient solution, and provides a very low latency calculation of a few scan line periods. This is in contrast to frame-buffer based methods that have latencies consisting of a few frame periods, which are usually a few orders of magnitude larger than line periods.

1.4 Thesis Structure

This dissertation is divided in two main parts. Part I introduces the concept of pixel stream processing and how this relates to the inputs and outputs of the system as a whole. First some common processing steps like demosaicing and kernel based filtering are discussed in relation with pixel stream processing (Chapter 2). Chapter 3 elaborates on the importance of image warping and its implementation in hardware. Next, a novel and alternative multiple memory architecture of the common line-buffer architecture is presented in Chapter 4. Chapter 5 concludes Part I. It compares several different window matching algorithms and presents a new adaptive binary
weight support algorithm tailored to SoC implementation.

In Part II, we incorporate the methods presented into a complete multi camera vision system. Chapter 6 presents the usage of pattern classification methods for giving a confidence value to each pixel in the pixel stream. First, the most suitable features are presented and compared. Second, several strategies to use these features for providing a confidence value are described. Next, in Chapter 7, we present a framework to generate and validate the hardware for a multi camera vision system. In Chapter 8 we use this framework for generating instances of correlation based depth camera systems.

A brief conclusion is presented in chapter 9.
Part I

Pixel Stream Processing
Basic Concepts

Contents

1.1 Problem Statement and Motivation .......................... 1
1.2 Research Approach ............................................ 2
1.3 Summary of Contributions ................................. 3
1.4 Thesis Structure .............................................. 4

2.1 Introduction

This chapter introduces the concept of pixel stream processing and how this relates to the inputs and outputs of the system as a whole. First the main consideration for hardware implementation is introduced. Second the pipelined processing data path for a single camera is presented. Finally, the stereo vision application is discussed which will be used as main driver example in this dissertation.

2.2 Pipelined Parallel Processing

The main advantage of dedicated Very Large Scale Integration (VLSI) or System-on-Chip (SoC) architectures in comparison to general purpose Central Processing Unit
(CPU) or Graphics Processing Unit (GPU) is its inherent parallelism and independence of architecture. Dedicated hardware can process data more efficiently by using customized data paths. The maximization of parallel calculations and a fine grained pipelined processing data paths are presented as the major focus of the architecture.

When placing several operations after each other, the maximal frequency is depending on the longest delay path (Benkoski et al. [1990]) between two clocked registers. The more operations are placed between two clocked registers, the lower the maximal operational frequency. The difference between a sequential and tree-based parallel adder calculation is indicated on Figure 2.1. The longest path for the tree-based adder calculation is clearly shorter, which will allow for a higher operational frequency. Also the way in which the adder blocks are constructed has a huge impact on the delay. The traditional carry-chain adders are slower than e.g. carry-bypass and/or carry look ahead configuration (Benkoski et al. [1990]).

![Figure 2.1: Adder calculation (left: sequential, right: tree-based parallel).](image)

When adding extra clocked registered in the data path (Figure 2.2), the operational frequency can be further increased.

### 2.3 Pixel Stream Processing

A data stream is a synchronized source of pixels originating from the image sensor where the first element of the stream is the top left pixel in the image and the following elements are corresponding pixels on the same line.

An example basic camera system consists out of four different entities (Figure 2.3): first the image sensor provides a stream of pixel data, second a median filter is used
to remove defected pixels, next a demosaicing algorithm is used to reconstruct the colour image and lastly the data is send to a screen.

Most operations do not process each pixel individually, instead they use a region around the pixel. Pixel \( p \) is defined as the pixel under consideration, while the support region \( N(p) \) is defined as the set of pixels centred in \( p \). Both the demosaicing module as the median filter use this support region \( N(p) \).

### 2.3.1 System Input and Output

Commonly, frame buffers are used to capture the pixels coming from the image sensor. Another option is to configure the image sensor to output a pixel stream that can be directly used to connect to a screen. An additional First In, First Out (FIFO) buffer is needed to temporally store a single line, but it eliminates the need to use a complete frame buffer. The basic idea is to regulate the line period of the image sensor in such way that it equals the line period of the screen. Where the line period
is the combination of the line field period and the line blanking period (2.1) (2.2).

\[
\frac{\text{field}_{h,sensor} + \text{blanking}_{h,sensor}}{\text{clk}_{sensor}} = \frac{\text{field}_{h,screen} + \text{blanking}_{h,screen}}{\text{clk}_{screen}} \tag{2.1}
\]

\[
\text{blanking}_{h,sensor} = \frac{\text{clk}_{sensor}}{\text{clk}_{screen}} \cdot (\text{field}_{h,screen} + \text{blanking}_{h,screen}) - \text{field}_{h,sensor} \tag{2.2}
\]

To clarify this method, the parameters for a Complementary Metal Oxide Semiconductor (CMOS) sensor with a pixel clock of 90 MHz are adapted for connecting to a Video Graphics Array (VGA) screen with a resolution of 800 × 600 pixels and a pixel clock of 36 MHz (2.3). Only the first 1,024 pixels of each line are written to the FIFO. This way the FIFO can be read out at 36 MHz without creating an underflow or overflow of the FIFO. The part of the blanking period that is written to the FIFO is the same size as the horizontal blanking period of the VGA specifications for a resolution of 800 × 600 pixels with a frame rate of 56 Hz. In this way, the output of the FIFO can directly be connected to a VGA screen (Figure 2.4).

\[
\text{blanking}_{h,sensor} = \frac{90 \text{ MHz}}{36 \text{ MHz}} \cdot (800 + 224) \text{ pixels} - 800 \text{ pixels} = 1760 \text{ pixels} \tag{2.3}
\]

![Diagram](image)

Figure 2.4: Camera system without frame buffer.
2.3.2 Demosaicing

Pixels generated by an image sensor are formatted in a Bayer pattern (Bayer [1976]) consisting of the four colours: Red (R), Green1 (G1), Blue (B) and Green2 (G2), representing the three colour filters (Figure 2.5). Since the human eye is more sensitive to the green colour, the number of green pixels is 50 % of the total amount of pixels. Two methods have been implemented in hardware to convert these raw pixel values to RGB pixel values.

![Figure 2.5: Bayer pattern.](image)

The first method is the bilinear interpolation (Jean [2010]). The missing colour channels are averaged from their direct neighbours. The different kernels are illustrated in Figure 2.6, a support region \( N(p) \) of 3 \( \times \) 3 is needed to calculate the kernels.

An improvement is the high-quality linear interpolation method proposed by Malvar et al. [2004]. It is a gradient-corrected bilinear interpolated approach with predefined gain parameter to control how much correction is needed. The different kernels are illustrated in Figure 2.7. A 9-point region is needed for the green channel, while the red and blue channels require an 11-point region. A support region \( N(p) \) of 5 \( \times \) 5 is used to calculate the kernels.

Both methods can be adapted for parallel computation without major issues. Table 2.1 indicates that the main resource difference between the two methods is not necessarily the number of calculations (indicated by the Logic Element (LE) usage), but the use of a larger support region (indicated by the memory usage).
Figure 2.6: Basic demosaicing algorithm.

Figure 2.7: Linear interpolation based demosaicing algorithm (Malvar et al. [2004]).
### 2.3.3 Median Filter

The median filter takes the median value of the pixels in the support region \( N(p) \). It is a non-linear filtering technique that preserves edges and removes defective pixels (Gonzalez and Woods [2007]). All pixels in the support region \( N(p) \) are sorted and stored in an array. Hence, the middle element of this array is the output of the filter. The implementation on Figure 2.8 uses the odd/even transposition sort network proposed by Oflazer [1983]. It is a structure that can be pipelined consisting of several compare and swap stages.

![Odd/Even transposition sort network using a nine-stage pipeline](image)

Figure 2.8: Odd/Even transposition sort network using a nine-stage pipeline (CS: Compare and Swap element, D: Delay element).

This filter is particularly suited for removing defective pixels. When using it directly on the RAW pixel stream, the Bayer pattern needs to be taken into account. Only pixels of the same colour filter (R, G1, G2 or B) are being considered. For
example, on Figure 2.9, only the shaded pixels are used during the median calculation of the red centre pixel $p$.

Table 2.2 illustrates the resource usage of this module. Reduction of logic element resources can be obtained by implementing more advanced sorting methods as proposed by Vega-Rodriguez et al. [2002]. This method could be implemented in the future, however from a system viewpoint the resource reduction is not significant.

### 2.4 Multi Stream Processing: Stereo Vision as Use Case

The goal of stereo vision is to obtain depth information by the comparison of two images. When an object is seen from two different camera positions, it is possible to calculate the object’s 3D position in the environment (see Figure 2.10). On each image, the object needs to be identified in order to obtain the $x$ and $y$ coordinates from the viewpoint of each image sensor. These two sets of coordinates can be used, together with the spatial relationship between the image sensors, to find the 3D position. For each pixel in one image sensor, the corresponding pixel needs to be
located on the other image sensor. If the spatial relationship between the cameras is known, this search can be limited to a single slanted line, known as the epipolar line.

In order to aid the pixel matching, the two image sensors can be oriented so that they have a vertical alignment and a horizontal offset (Figure 2.11) (Szeliski [2011]). This is in the assumption of calibrated camera images. In this case, the corresponding pixels will appear on the same horizontal line. The difference between the object location on the $x$ axis of both cameras is called the disparity. Objects that are close to the cameras will have a larger horizontal disparity than objects that are far away. An example is given on Figure 2.12.

2.5 Conclusion

In this chapter, the basic modules have been presented for the construction of a camera system. By directly processing the pixel data, a real-time system with very low latency can be achieved.
Figure 2.11: Alligned binocular camera system.

Figure 2.12: Alligned binocular camera system, both images are placed on top of each other.
3

Chapter

Image Sensor Alignment

Contents

2.1 Introduction ......................................................... 9
2.2 Pipelined Parallel Processing ................................. 9
2.3 Pixel Stream Processing ........................................ 10
   2.3.1 System Input and Output ................................. 11
   2.3.2 Demosaicing ................................................ 13
   2.3.3 Median Filter .............................................. 15
2.4 Multi Stream Processing: Stereo Vision as Use Case .... 16
2.5 Conclusion ......................................................... 17

3.1 Introduction

Image warping is an important research topic in computer vision and graphics. It is mainly used for correcting lens distortions and aligning multiple image sensors. It is a spatial transformation of an image based on a geometric relationship (King [1996]). The image warping module takes a stream of pixels from a camera, temporally stores them and outputs them in a different order (Figure 3.1). The specific order depends on the geometric relation which can either be calculated on-line or pre-calculated and stored in a look-up table (LUT). In a video pipeline, the data stream is a synchronized
Image Sensor Alignment

source of pixels coming from an image where the first element of the stream is the top left pixel in the image and the following elements are corresponding pixels on the same line. For each pixel in the output image a colour is selected from the input image using its inverse-warped location. Real-time image warping is necessary in several applications. In medical endoscopy it is used to remove the large radial distortion caused by the small size of the lens (Melo et al. [2012]). For the generation of panoramic videos, the images of the cameras need to be warped with respect to each other in order to stitch them together (F. Huang and Chang [2010]). In a stereo camera setup, two different kinds of distortions are present; the first one is the lens distortion, the second one is a misalignment of the two cameras. Since the search space for stereo matching is located on the epipolar line, both distortions should be resolved before the matching can be performed (Fusiello et al. [2000]). Other applications include image rotation, scaling, translation or a combination of them (King [1996]).

![Image Warping Diagram](image)

Figure 3.1: Image warping.

For all of these applications, not only real-time processing is important, but also low latency and cost. We believe that this can only be achieved by implementing warping into hardware and by avoiding the use of external memories. Luo et al. [2011] implemented a LUT based image warping module in hardware. By using a compressed LUT they do not need off-chip memory to store the full warping table. However, they still need off-chip memory to temporally store the input stream of the camera. Oh and Kim [2008] implemented an FPGA based fast image warping module which focuses on latency reduction. A single LUT multiple access method is proposed which stores the LUT in on-chip memories. Off-chip memories are used to store the output image. Rodrigues and Ferreira [2010] implemented a real-time rectification
module for stereo images. They use a Microblaze processor to calculate the reverse mapping coordinates and make use of off-chip memories to store the input stream.

The goal of the architecture presented in this thesis is to reduce memory usage in order to implement the complete warping module on-chip without the use of off-chip memories. The focus is oriented towards the reduction of the pixel input buffer. Several data structures are presented which reduce the memory usage by taking the underlying pattern of the reverse mapping coordinates into account.

3.2 Warping Overview

Image warping consists of three main parts. First, the reverse mapping coordinates need to be provided. They can be pre-calculated and stored in a LUT or calculated on-line. The LUT based method has the advantage that no expensive calculations are needed, but this is at the cost of additional memory usage. For this architecture, a memory efficient LUT based implementation is chosen. Second, the input pixel stream needs to be stored in order to select the output pixels from. Third, the output pixels are re-sampled in order to get sub-pixel accuracy.

3.2.1 Reverse Mapping Coordinates

In Figure 3.2, the original image is located on the left side. The warped image is located on the right side. The mapping information between both images is illustrated in the middle plot. For each pixel of the warped image, a pixel of the source image is selected. The index difference between the warped and the source image pixel are called the reverse mapping coordinates. Notice that only the vertical reverse mapping coordinates are provided in Figure 3.2, the horizontal reverse mapping coordinates are assumed zero.

Figure 3.2: Reverse mapping coordinates (left: source image, middle: vertical reverse mapping coordinates, right: warped image).
For each pixel, a reverse mapped coordinate needs to be provided. When the mapping coordinates do not change drastically from pixel to pixel, it suffices to only store the mapping coordinates of certain pixels. This will be further explained in Section 3.3.

### 3.2.2 Storage of the Source Image

The warped image is constructed by selecting pixels from the source image. The information about which pixels to select are provided by the reverse mapping coordinates. In order to allow the selection of pixels from previous lines, it is necessary to store a previous number of pixel lines in a memory (Figure 3.3). Most commonly a circular buffer is used to temporarily store the pixels (Jin et al. [2010], Tomasi et al. [2012]). However this is not efficient when the underlying structure of the reverse mapping coordinates is known. This will be discussed in more detail in Section 3.4.

![Figure 3.3: Warping of input to output pixel stream using a circular buffer.](image)

### 3.2.3 Sub-Pixel Resampling

When the mapping coordinates are integer, only one source pixel is needed for the warped image (Figure 3.2). However, a better result can be obtained when using mapping coordinates which contain fractional values (Figure 3.4). When using mapping coordinates with sub-pixel accuracy, a window of four pixels is used from the source image (Figure 3.5). Bilinear interpolation is hence used to calculate the resulting warped pixel (Gribbon and Bailey [2004]).
Figure 3.4: Reverse mapping coordinates using sub-pixel resampling (left: source image, middle: vertical reverse mapping coordinates, right: warped image).

Figure 3.5: Sub-pixel resampling (top: without resampling, bottom: with resampling).
3.2.4 Examples

Figure 3.6 illustrates some examples of common warping mappings. The original image is located on the left side. The warped image is located on the right side. The mapping information between both images is illustrated in the middle plots. The $X$ and $Y$ axis represent the coordinates of the image, the $Z$ axis represents the reverse mapping coordinates. The methods proposed in this chapter are suitable for different warping patterns. The influence on the memory requirement will be discussed in the next section.

![Warping examples](image)

Figure 3.6: Warping examples, from top to bottom: wave, expand and compress and barrel distortion (left: source image, middle: vertical reverse mapping coordinates, right: warped image).
3.3 Storage of Reverse Mapping Coordinates

Storing the mapping coordinates for each pixel uses a large amount of memory. When the mapping coordinates do not change drastically from pixel to pixel, it suffices to only store the mapping coordinates of certain pixels. Interpolating between those pixels will provide the missing information. Two methods are described in the following section. The first method uses a basic regular grid structure uniformly spaced on the image. The second method, newly proposed in this dissertation, improves this method by allowing more grid points to be located on the part of the image where more variation is present.

In the first method, the pixels are located on a regular grid. Such a regular structure makes it straightforward to select the corner points for interpolation. The desired grid rectangle size depends on the amount of distortion in the image. This grid rectangle size will be the same for all regions of the image. Figure 3.7 illustrates the reverse mapping coordinates for correcting the lens distortion of a typical camera system. A grid, where each rectangle measures 16 pixels by 16 pixels, is sufficient to reconstruct the mapping coordinates. In this example, where the image size is $800 \times 600$, the number of reverse mapping grid corner points becomes $51 \times 39$.

Bilinear interpolation is used to reconstruct the mapping coordinates for the complete image (Figure 3.8). For every pixel of the image, it is determined which corner points of the grid (‘a’, ’b’, ’c’ and ’d’) need to be loaded by calculating their memory address. Two registers ‘e’ and ’f’ are used to determine the position of the current pixel with respect to the selected grid rectangle and to switch to another grid rectangle (see Listing 3.1 and Listing 3.2). On each clock cycle, ’f’ increments. When it equals the width of the grid rectangle, the next grid rectangle is chosen on the same line. When a new line starts, ’e’ increments. As soon as ’e’ equals the height of the grid rectangle, a new grid rectangle is chosen on the next grid line, otherwise the first grid rectangle of the current grid line is selected.

Listing 3.1: Main steps for ’e’ and ’f’ calculation for a regular grid

```
if (i_new_frame), f = 1; e = 1; end
if (i_new_line)
    f = 1; if (e == GRID_RECTANGLE_HEIGHT), e = 1; else, e++; end,
end
if (f == GRID_RECTANGLE_WIDTH), f = 1; else, f++; end
```
Listing 3.2: Main steps for the calculation of the grid corner points for a regular grid

```matlab
if (i_new_frame) #Return to the start of the grid
    mem_address_a_o = 0; mem_address_o = 1;
    mem_address_c_o = GRID_NR_COLUMNS; mem_address_d_o = GRID_NR_COLUMNS+1;
end

if (i_new_line)
    if (e == GRID_RECTANGLE_HEIGHT) #Go to the next grid rectangle line
        mem_address_a_o = mem_address_a_o + GRID_NR_COLUMNS;
        mem_address_b_o = mem_address_b_o + GRID_NR_COLUMNS;
        mem_address_c_o = mem_address_c_o + GRID_NR_COLUMNS;
        mem_address_d_o = mem_address_d_o + GRID_NR_COLUMNS;
    end
    #Go to the first grid rectangle on the same row
    mem_address_a = mem_address_a_o;
    mem_address_b = mem_address_b_o;
    mem_address_c = mem_address_c_o;
    mem_address_d = mem_address_d_o;
end

if (f == GRID_WIDTH) #Go to the next grid rectangle on the current row
    mem_address_a = mem_address_a + 1;
    mem_address_b = mem_address_b + 1;
    mem_address_c = mem_address_c + 1;
    mem_address_d = mem_address_d + 1;
end
```

Formula (3.1) illustrates how the mapping coordinates for pixel 'p' (map_p) are calculated from the mapping coordinates of pixel 'a' (map_a), 'b' (map_b), 'c' (map_c) and 'd' (map_d), which are located on the rectangular grid.

\[
\begin{align*}
    map_r &= (|ar| \cdot map_b + |rb| \cdot map_a) / |ab| \\
    map_s &= (|cs| \cdot map_d + |sd| \cdot map_c) / |cd| \\
    map_p &= (|rp| \cdot map_s + |ps| \cdot map_r) / |rs| 
\end{align*}
\]  

(3.1)

The second method also uses a grid, it is however not a regular grid. When using this newly proposed irregular grid, more grid points can be located on the part of the image where more variation is present. The number of grid points will be lower compared to the regular grid method, but for each grid point extra information is needed. The additional data is used to determine which corner points are necessary.
Figure 3.7: Vertical reverse mapping coordinates regular grid.

Figure 3.8: Regular grid based mapping.
for interpolation. This is illustrated in Figure 3.9 where three memory address offset values are needed to determine the corners of the current grid rectangle. In addition, two values are needed to know the width and the height of the current grid rectangle, since they are not constant any more.

![Figure 3.9: Irregular grid based mapping.](image)

Table 3.1 contains the mapping information for the irregular grid of Figure 3.9. By selecting the memory address, this table provides the mapping data of the top corner of the grid rectangle, and the information to obtain the three other corner points. The ”Address Memory” is the index for the memory where all grid points are stored. Next, the ”Memory Address Offset” is the offset needed to calculate the memory addresses of the three other corner points from the current memory address. Last, the ”Grid Rectangle Size” contains the size of the selected grid rectangle.

The calculation of ‘f’ remains the same, the only difference is that the width of the grid rectangles is not a constant any more (see Listing 3.3), it depends on the current grid rectangle, e.g. grid rectangle (1 − 2 − 7 − 8) is smaller than grid rectangle (10 − 12 − 22 − 23).

**Listing 3.3: Main steps for ‘f’ calculation for an irregular grid**

```plaintext
if (i_new_frame), f = 1; end
if (i_new_line), f = 1; end
if (f == grid_rectangle_width), f = 1; else, f++; end
```

Value ‘e’ is in this case not the same for all grid rectangles on the current line. Instead it changes from grid rectangle to grid rectangle. Each time a grid rectangle
Storage of Reverse Mapping Coordinates

<table>
<thead>
<tr>
<th>Memory Address</th>
<th>Reverse Mapping Coordinates</th>
<th>Memory Address Offset</th>
<th>Grid Rectangle Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>mapping data</td>
<td>6 1 1 4</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>mapping data</td>
<td>8 2 2 8</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>mapping data</td>
<td>0 / / /</td>
<td>/</td>
</tr>
<tr>
<td>10</td>
<td>mapping data</td>
<td>12 2 1 8</td>
<td>12</td>
</tr>
<tr>
<td>11</td>
<td>mapping data</td>
<td>0 / / /</td>
<td>/</td>
</tr>
<tr>
<td>12</td>
<td>mapping data</td>
<td>11 / / /</td>
<td>/</td>
</tr>
<tr>
<td>13</td>
<td>mapping data</td>
<td>2 1 1 4</td>
<td>4</td>
</tr>
<tr>
<td>14</td>
<td>mapping data</td>
<td>2 / / /</td>
<td>/</td>
</tr>
<tr>
<td>15</td>
<td>mapping data</td>
<td>4 1 1 4</td>
<td>4</td>
</tr>
<tr>
<td>16</td>
<td>mapping data</td>
<td>4 1 1 4</td>
<td>4</td>
</tr>
<tr>
<td>17</td>
<td>mapping data</td>
<td>4 1 1 4</td>
<td>4</td>
</tr>
<tr>
<td>18</td>
<td>mapping data</td>
<td>4 / / /</td>
<td>/</td>
</tr>
</tbody>
</table>

Table 3.1: Example storage for the reverse mapping coordinates

switch happens, the previous e value is stored in a First In, First Out (FIFO) memory and a new ‘e’ value is read from this FIFO. The depth necessary for this FIFO is depending on the maximal number of grid rectangles in one image line. Initially the FIFO will contain an ‘e’ value of one for all grid rectangles in the first image line. This is 1−1−1−1−1 for the example in Figure 3.9. Another FIFO is used to store the top-left corners (‘a’) of each grid rectangle for a single line. Initially the FIFO will contain an ‘a’ memory address value for all grid rectangles in the first image line. This is 1−2−3−4−5 for the example in Figure 3.9.

Reconstruction of the grid structure is performed one image line at the time. When ‘f’ equals 0, it is checked whether this is the last line of the grid rectangle (e == grid_rectangle_height). If not, the memory address to the top-left corner of the grid rectangle ‘a’ is stored in the FIFO together with its e value. If this is the last line of the grid rectangle, three scenarios are possible. This is illustrated in Figure 3.10, where the grey dots indicate the current grid rectangle. On the left image, no new grid rectangle is needed, so nothing is stored in the FIFO. In the middle image, a new grid rectangle with the same width is needed, so the new grid corner point ‘c’, is stored in the FIFO together with an ‘e’ value of one. On the right image, two
new grid rectangles are needed. First the new grid corner point 'c' is stored in the FIFO together with an 'e' value of one. Second, extra grid corner points are added until the width of the current grid rectangle |ab| equals the accumulated widths of the new grid rectangles. Listing 3.4 illustrates these calculations. Multiple reads are performed from the memory to reconstruct the structure of the grid.

Listing 3.4: Main steps for the calculation of the grid structure for an irregular grid

```plaintext
if (f == 1) %Start of a grid rectangle row
    if (e == grid_rectangle_height) %Current grid rectangle is finished
        if (mem_address_offset_ac(mem_address_c) == 0)
            %No grid rectangle is added
            else
                %Add first a single grid rectangle
                mem_address_a_tmp = mem_address_c;
                FIFO_mem_address_a = mem_address_a_tmp;
                FIFO_e = 1;
                width_sum = grid_size_ab(mem_address_a_tmp);
        %Keep adding grid rectangles until their accumulated width equals the current grid_rectangle_width
        while (width_sum < grid_rectangle_width)
            mem_address_a_tmp = mem_address_a_tmp +
                mem_address_offset_ab(mem_address_a_tmp);
            FIFO_mem_address_a = mem_address_a_tmp;
            FIFO_e = 1;
            width_sum = width_sum + grid_size_ab(mem_address_a_tmp);
        end
    end
    else
        %Current grid should be placed back in the FIFO
        FIFO_mem_address_a = mem_address_a;
        FIFO_e = e;
    end
end
```

Listing 3.5 illustrated the calculation of the memory addresses. Every time 'f' equals the grid rectangle width, the top-left corner point 'a', together with its 'e' value, is loaded from the FIFO. Since this is a new line for this grid rectangle, the 'e' value is increased by one. The memory addresses for corner points 'b-c-d' are calculated and the grid rectangle width and heights are loaded.
Listing 3.5: Main steps for the calculation of the grid corner points for an irregular grid

```plaintext
if (f == grid_rectangle_width)  %Go to the next grid on the same row
    %Read FIFO
    mem_address_a = FIFO_mem_address_a;
    e = FIFO_e + 1;

    %Get corner points of current grid
    mem_address_b = mem_address_a + mem_address_offset_ab(mem_address_a);
    mem_address_c = mem_address_a + mem_address_offset_ac(mem_address_a);
    mem_address_d = mem_address_a + mem_address_offset_ac(mem_address_a) +
                    mem_address_offset_cd(mem_address_a);

    %Get size of current grid
    grid_rectangle_width = grid_size_ab(mem_address_a);
    grid_rectangle_height = grid_size_ac(mem_address_a);
end
```

Table 3.2 indicates the bit usage for the current implementation. When making use of the irregular grid, 36 bits are needed for each grid point, compared with 18 bits when using the regular grid. This could be further reduced by reordering the reverse mapping coordinate storage, in a way that the memory address offset of "a → e" uses less bits. The bit usage of the grid rectangle size values |ac| and |ab| could also be reduced when restricting their variation. This implementation restricts |ab| and |ac| to be a power of two (4, 8, 16, 32, 64, 128, 256). This will further lower the bits needed for the representation of |ac| and |ab| and allows the replacement of the division operator in Formula (3.1). A further reduction could be obtained by investigating if the reverse mapping coordinates storage (see Table 3.1 and Table 3.2)
could be split up into a part with only the mapping information and a part with the grid structure information.

An iterative program has been developed which accepts the complete reverse mapping coordinates as input. It adds more grid points depending on the variation of the mapping coordinates. Using this approach, for the example in Figure 3.7, gives the results illustrated in Figure 3.11. The number of grid points for regions of high variation remains the same, while the number of grid corner points for regions with a low variation decreases. In this example, the number of grid corner points decreases from $1,989 \times 18 = 35,802$ bits to $836 \times 36 = 30,096$ bits).

Depending on the variation in the reverse mapping coordinates, the reduction in bit usage will change. Most gain will be obtained with reverse mapping coordinates which have a large localised variation. This new irregular grid has already showed benefits compared to the regular grid, however more research is needed to improve these results.

### 3.4 Memory Architecture

In order to perform reverse mapping, the source pixels need to be stored in a temporal buffer. The size required by this buffer is primarily determined by the vertical warping coordinates. The larger its range across the image, the more lines need to be buffered. The horizontal warping coordinates will only determine some additional pixel buffers. In the remainder of this chapter, only vertical warping coordinates are taken into account. In this section several memory allocation options are discussed.

<table>
<thead>
<tr>
<th>Reverse Mapping Coordinates</th>
<th>Memory Address Offset</th>
<th>Grid Rectangle Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Horizontal</td>
<td>Vertical</td>
<td>$a \rightarrow c$</td>
</tr>
<tr>
<td>9 bits</td>
<td>9 bits</td>
<td>6 bits</td>
</tr>
</tbody>
</table>

Table 3.2: Reverse mapping coordinates number of bits to store.
Figure 3.11: Vertical reverse mapping coordinates irregular grid.
3.4.1 Circular Buffer

When the vertical offset is uniformly distributed on an image line or when the pattern of the reverse mapping coordinates are not known, a circular buffer implementation is the most suitable. Figure 3.12 shows an example where no pattern can be found in the reverse mapping coordinates. The $X$ and $Y$ axis represent the coordinates of the image, the $Z$ axis represents the reverse mapping coordinates. The boxes around the plot indicate the memory access envelope.

![Figure 3.12: Vertical reverse mapping coordinates without known pattern, boxes indicate the memory access envelope.](image)

The size of the buffer equals the width of the image multiplied by the number of lines in the buffer. The amount of lines that are temporarily stored equals the highest vertical offset in the reverse mapping coordinates. Equation (3.2) calculates this for the example presented in Figure 3.12. $I_{Width}$ stand for the screen width, which is 640 for the examples in this chapter. In this case the memory needs 12,800 elements to temporarily store the pixels. A circular buffer is used as data structure (see Figure 3.13).

$$buffer_{size} = I_{Width} \times \max(\text{reverse mapping coordinates})$$

$$= 640 \times 20$$

$$= 12,800$$
3.4.2 Split Circular Buffer

When the vertical offset is not uniformly distributed on an image line, parts of the stored pixels are never actually read out. On Figure 3.14, the mapping coordinates have a sinusoidal pattern. The resulting memory access is depicted on Figure 3.14. The left side of the image needs a large buffer, while the right side of the image only uses a much smaller buffer.

Instead of using a circular buffer with a fixed vertical offset across the image line, it is more memory efficient to split up the circular buffer into two (or more) slices: One large circular buffer for the left slice of the image and one smaller circular buffer for the right slice. Notice that these circular buffers can be located in the same physical memory. For each memory slice, a write pointer is stored (see Figure 3.15). For every pixel write, it is determined in which memory slice the pixel needs to be written.

The read pointer is calculated from the current write pointer: the additional
calculation step includes the determination of which circular buffer to read from.

For the example presented in Figure 3.14, a memory size of 7,360 elements is needed to temporarily store the pixels (3.3). Compared with 9,600 elements (640×15) for a circular buffer, a reduction in memory usage of 23.33% is obtained.

\[
buffer_{\text{size}} = \sum_{s=1}^{2} \left( \text{slice } s_{\text{width}} \times \max_{\text{slice } s}(\text{reverse mapping coordinates}) \right) = 320 \times 15 + 320 \times 8 = 7,360
\]  

(3.3)

### 3.4.3 Adaptive Split Circular Buffer

When the vertical offset is not uniformly distributed on an image line and when the horizontal offset changes during consecutive image lines, an adaptive split circular buffer is needed. This situation occurs when in the first slice, the image is expanded vertically while in the second slice, the image is compressed vertically (Figure 3.16 and Figure 3.17). The memory access will change gradually with consecutive lines. The split circular buffer needs to be adapted in between image lines to accommodate this change.

Equation (3.4) calculates the buffer size for the example presented in Figure 3.16. In this case the memory needs 5,760 elements to temporarily store the pixels. Compared with 7,680 elements (640×12) for a circular buffer, a reduction in memory
Figure 3.16: Vertical reverse mapping coordinates with changing wave pattern.

Figure 3.17: Vertical reverse mapping coordinates with changing wave pattern.
usage of 25% is obtained.

\[
buffer_{size} = \max \left( buffer_{size, config\ 1}, buffer_{size, config\ 2} \right) \\
= \max \left( (320 \times 12 + 320 \times 6), (320 \times 6 + 320 \times 12) \right) \\
= 5,760
\]  

In this scenario, the data structure consists of several circular buffers which are linked together. These linkages will change during consecutive image lines. In the top of the image, the grey memory block (Figure 3.18) will be assigned to the left slice of the image, while on the bottom of the image it will be assigned to the right slice of the image.

![Adaptive split circular buffer](image)

```
Write Pointer 1
```
```
Write Pointer 2
```

The information about memory allocation for different slices of the image can be stored in small additional memories.

### 3.4.4 Circular Buffer with Line Read Postponing

In the scenario on Figure 3.19, the vertical offset decreases during consecutive image lines. First the complete buffer needs to be filled with pixel data before reading can be started. This is a result from the fact that all lines are normally processed one after another. Second, on each line, only parts of the buffer are used. It is more memory efficient to take a smaller memory and to postpone line reads when needed.

Every time, when reading is postponed for a complete line, the next line that is being read will have an additional offset of plus one. The assumption is that the input
pixel stream has blanking lines and that the next image processing steps are prepared for handling them. The first assumption holds when using a simple Complementary Metal Oxide Semiconductor (CMOS) camera. After the last line, it will send a couple of pixel lines that are black, the number of these blanking lines can be programmed. The second assumption depends on the architecture of the next processing steps. Figure 3.20 shows an image where reading has been stopped for a couple of lines on two separate places. This leads to a reordering of the blanking lines.

The reverse mapping LUT has to be changed every time a read line is skipped.
Since the line skip is predetermined, the LUT can be changed off-line in order to reduce the size of it. For each line skip, all mapping coordinates from that line on will be decremented by one. Figure 3.21 shows the result of the modification of the reverse mapping coordinates of Figure 3.19 when line skipping is used. In this case a reduction in memory usage of more than 50% is obtained.

![Figure 3.21: Vertical reverse mapping coordinates with decreasing random pattern, modified for line read postponing.](image)

### 3.4.5 Adaptive Split Circular Buffer with Line Read Postponing

This is the most complex case investigated in this chapter and resembles the scenario when a camera is corrected from its barrel distortion. As seen on Figure 3.22 and Figure 3.23, the vertical offset decreases continuously during consecutive lines. The vertical offset is not uniform within a line and it changes during consecutive lines. When only using a split circular buffer approach, not much reduction in memory usage is obtained (6.25% in this example). The reason is the flatness of the reverse mapping curve in the first configuration (see Figure 3.23).

When only using the circular buffer with line read postponing, a reduction of 50% is obtained since the value range of each line in the reverse mapping LUT is half of the maximum value of this LUT. When adding the adaptive split circular buffer approach
Figure 3.22: Vertical reverse mapping coordinates for barrel distortion.

Figure 3.23: Vertical reverse mapping coordinates for barrel distortion.
with line read postponing, a larger reduction of memory usage is obtained. On the lower part of the image, read line skipping is used to reduce the need for extra line buffers. Figure 3.24 and Figure 3.25 show the result of the modification of the reverse mapping coordinates of Figure 3.22 when line skipping is used.

![Figure 3.24: Vertical reverse mapping coordinates for barrel distortion, modified for line read postponing.](image)

For the example presented in Figure 3.24, the combination of the adaptive split circular buffer with line read postponing leads to a reduction of 56.25%. Increasing the number of slices, will provide a high reduction. This can be seen on Figure 3.26 and Figure 3.27. In this case more slices are used, so that a total memory reduction of 63.67% is obtained, compared to a normal circular buffer.

A real-world example of reverse coordinates for stereo calibration is illustrated on Figure 3.28 and Figure 3.29.
3.5 Hardware Architecture

The goal of the presented architecture is to reduce the memory usage in order to implement the warping module on-chip with room available for additional applications like stereo vision. The image warping module receives pixels originating from a camera and places them in an input buffer (see Figure 3.30). The reverse mapping coordinates are calculated from the LUT using bilinear interpolation. The integer part of these coordinates is used to select the four part window from the input buffer. The fractional part is used to calculate the warped pixel values based on the values of the four nearest pixels at integer grid positions by using bilinear interpolation.

The main focus of this chapter is the reduction of the memory usage in the input buffer. Instead of using a single circular buffer, a method is chosen which enables adaptable circular buffers for different slices of the images (see previous section) and allows the postponing of a line read. The on-chip memory is a dual port RAM which has separate read and write address inputs and a four pixel window output (see Figure 3.31).

The memory allocation LUT provides the write and read address calculation blocks with information about the structure of the memory. It is a list containing the memory blocks and their linkage for each slice of the image (see Table 3.3 and Table 3.4). It also
Figure 3.26: Vertical reverse mapping coordinates for barrel distortion, modified for line read postponing.
Figure 3.27: Vertical reverse mapping coordinates for barrel distortion, modified for line read postponing.

<table>
<thead>
<tr>
<th>Slice Size</th>
<th>Slice Width</th>
<th>Start Block</th>
</tr>
</thead>
<tbody>
<tr>
<td>First Configuration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>3,840</td>
<td>320</td>
</tr>
<tr>
<td>1</td>
<td>1,920</td>
<td>320</td>
</tr>
<tr>
<td>Second Configuration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1,920</td>
<td>320</td>
</tr>
<tr>
<td>3</td>
<td>3,840</td>
<td>320</td>
</tr>
</tbody>
</table>

Table 3.3: Memory allocation for individual slices and configurations.

contains information about when to switch from one memory allocation to another and when to postpone reading of a pixel line. The information in the tables is linked to the example on Figure 3.33 and Figure 3.32. The number of the block is annotated in the left corner of each block. Notice that block one and four are mapped to the same memory region.

For each memory slice (e.g. left and right slice in Figure 3.32) a write address pointer is stored. In Listing 3.6 this is denoted by the array \( \text{WR} \_\text{addr} \), where the index \( nr \) indicates the current image slice. Register \( a \) keeps track of the current horizontal position with respect to the start of the image slice; when \( a \) equals the width of the image slice, it jumps to the next slice.
Figure 3.28: Vertical reverse mapping coordinates for stereo rectification.

Figure 3.29: Vertical reverse mapping coordinates for stereo rectification.
**Hardware Architecture**

---

Figure 3.30: Image warping architecture.

Figure 3.31: Input buffer architecture.

<table>
<thead>
<tr>
<th>Start Address</th>
<th>Block Size</th>
<th>Next Block</th>
<th>Block Nr.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>First Configuration</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1,920</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1,920</td>
<td>1,920</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>3,840</td>
<td>1,920</td>
<td>2</td>
</tr>
<tr>
<td><strong>Second Configuration</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1,920</td>
<td>1,920</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1,920</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>3,840</td>
<td>1,920</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 3.4: Memory allocation for individual blocks.
Figure 3.32: Vertical reverse mapping coordinates with changing wave pattern.

Figure 3.33: Vertical reverse mapping coordinates with changing wave pattern.
Each memory slice can consist of one or more memory blocks which are linked together. When \texttt{WR_addr} equals the end of the current block, the \texttt{WR_addr} jumps to the next memory block (or the start of its own block).

**Listing 3.6: Main steps for the write address calculation**

```plaintext
if (a > width(nr))
  a = 1; nr++;
else
  a++;
end

if WR_addr(nr) >
  size(current_block(nr)) + start(current_block(nr))
  current_block(nr) = next_block(current_block(nr));
  WR_addr(nr) = start(current_block(nr));
else
  WR_addr(nr) ++;
end

mem(WR_addr(nr)) = Io(x,y);
```

The read address is calculated from the memory allocation LUT and the calculated reverse mapping coordinates. The first step consists of determining in which image slice the warping pixel is located (Listing 3.7). This depends on the value of the horizontal mapping and the current horizontal position with regards to the start of the image slice \(a\). The \(RD_addr\) is initially calculated from the assumption that each image slice only contains one memory block, the \textit{relative_write_address} is the write address in this fictional memory block. Next the vertical offset is subtracted. If the result is lower than zero, it is subtracted from the total size of this fictional memory block. The last part calculates the position of the \texttt{RD_addr} pointer in the separate memory blocks using a mapping module.

Incorporation of the line read postponing function is straightforward. On the start of each image line, it is checked if it needs to be postponed or not. If yes, the read address calculation module does not do anything for this line. Notice that the number of read lines will not be changed; every line read that is postponed will add an extra line read at the end of the frame. The reverse mapping coordinates do not need to be modified online since they already have been modified off-line.
Listing 3.7: Main steps for the read address calculation

```
if (horizontal_offset > a)
    nr_read = nr-1;
    RD_addr = relative_write_address(nr_read) -
              horizontal_offset + a;
else
    nr_read = nr;
    RD_addr = relative_write_address(nr_read) -
              horizontal_offset;
end

RD_addr = RD_addr - width(nr_read)* vertical_offset

if RD_addr < 0
    RD_addr = RD_addr + slice_size(nr_read);
end

RD_addr = mapping(RD_addr);
Iw(x,y) = mem(RD_addr);
```

### 3.6 Implementation

A specific generator has been used to generate and optimize the various look-up tables. From the full warping coordinates, the best memory architecture is chosen, taking into consideration the additional calculation steps of more complex architectures. The architecture and methods presented in this chapter have been implemented on an Field Programmable Gate Array (FPGA) system, based on an Altera Cyclone IV with 114,480 logic elements and 432 memory blocks. The sources of the input stream are two cameras with a resolution of $640 \times 480$ and a pixel clock of 16 MHz resulting in a frame refresh rate of 52 Hz. The application chosen is the rectification of two stereo streams. In this case, the proposed memory buffer is an adaptive split circular buffer with line read postponing. The architecture has been constructed to reduce memory usage. Hence there is no need for external memories. The reduction of external memory usage has the additional advantage that the latency between input frame and output frame becomes minimal. Table 3.5 shows the resource consumption of the main blocks presented in this chapter. The synthesis results are obtained using Quartus II 11.0 for an Altera Cyclone IV. Notice that the proposed buffer
### 3.5 Resource Usage for the Rectification of a Stereo Camera

<table>
<thead>
<tr>
<th>Module Name</th>
<th>Logic Elements</th>
<th>Memory Blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Nr.</td>
<td>Single</td>
</tr>
<tr>
<td>Input Buffer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Circular Buffer</td>
<td>2</td>
<td>60</td>
</tr>
<tr>
<td>OR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Proposed Buffer</td>
<td>2</td>
<td>840</td>
</tr>
<tr>
<td>Reverse Mapping</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LUT</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Horizontal Interpolation</td>
<td>2</td>
<td>428</td>
</tr>
<tr>
<td>Vertical Interpolation</td>
<td>2</td>
<td>428</td>
</tr>
<tr>
<td>Resampling</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Resampling</td>
<td>2</td>
<td>960</td>
</tr>
<tr>
<td>Total with Circular Buffer</td>
<td>3,732</td>
<td></td>
</tr>
<tr>
<td>Total with Proposed Buffer</td>
<td>5,312</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.5: Resource usage for the rectification of a stereo camera.

The presented method reduces the memory usage significantly (reduction of 70 blocks equals 16% of available memory blocks on this type of FPGA) while just using a small number of extra logic elements (increase of 1,580 logic elements equals 1% of available logic elements). Figure 3.34 shows the results before and after the warping module. It is clear that matching along the epipolar line (white line) improves after rectification of both cameras.

### 3.7 Conclusion

A real-time image warping module has been presented. The main focus was the storage of the input stream before the pixels are warped. The presented architecture uses an adaptable memory allocation which can change the depth and the position of the circular buffers between lines. It is shown that the use of this method reduces the memory requirement in several use-cases. The reverse mapping coordinates are stored in a new LUT based method using a irregular grid structure and bilinear interpolation is used to calculate sub-pixel accuracy. In the case of stereo vision, the image warping module is used to perform real-time rectification. A memory reduction of approximately 50% is obtained compared to the common circular buffer implementation. The architecture has been implemented in an FPGA without making
use of external memories.

Figure 3.34: Image rectification results before (top) and after (bottom) warping for a stereo camera.
Chapter 4

Tiled Parallel Memory Architecture

Contents

3.1 Introduction ............................................. 19
3.2 Warping Overview ...................................... 21
  3.2.1 Reverse Mapping Coordinates .................... 21
  3.2.2 Storage of the Source Image ...................... 22
  3.2.3 Sub-Pixel Resampling ............................. 22
  3.2.4 Examples ........................................... 24
3.3 Storage of Reverse Mapping Coordinates .......... 25
3.4 Memory Architecture ................................. 32
  3.4.1 Circular Buffer ................................... 34
  3.4.2 Split Circular Buffer .............................. 35
  3.4.3 Adaptive Split Circular Buffer .................... 36
  3.4.4 Circular Buffer with Line Read Postponing .... 38
  3.4.5 Adaptive Split Circular Buffer with Line Read Postponing 40
3.5 Hardware Architecture ............................... 43
3.6 Implementation ....................................... 50
3.7 Conclusion ........................................... 51
4.1 Introduction

When processing a pixel $p$, many vision algorithms perform their calculations on a rectangular support region $N(p)$ which is centred in $p$. This can be seen as a small window of pixels inside the image.

To store and retrieve image windows, most architectures use a combination of line buffers and shift registers. However, the traditional scan line based First In, First Out (FIFO) architecture does not fully exploit the parallelism that is available with multiple on-chip memory blocks.

When using scan line buffers in combination with registers, a single clock is used for both reading as writing. For each pixel write, a new window is placed in registers. If the writing of a pixel and the reading of a window could be separated, it is possible to allow a sharing of resources across time.

Parallel memory architectures for block sized motion estimation are presented extensively in the literature (Vanne et al. [2008]). Virtual memory tiling for region based memory structures are proposed by McCormack and Hogg [1997]. A cache tiling structure for morphological image processing is proposed by Wittenbrink and Somani [1993]. They all are similar in the way that they organise the memory structure into tiles, which represents the region of interest.

This chapter introduces a tiling parallel memory architecture especially suited for image matching purposes and for implementation on an Field Programmable Gate Array (FPGA). The major focus of the architecture presented in this chapter is on the maximization of the parallel calculations needed for vision processing. Memory usage is of a particular importance since it is not feasible to calculate data in parallel if there is no timely access to parallel data to be processed. A good balance is needed. In this context, a parallel memory architecture is proposed that uses multiple on-chip memory blocks.

The proposed memory architecture in this chapter allows for a complete window refresh on each clock cycle. This makes it possible to load additional windows by increasing its operating frequency with respect to the pixel clock. It also allows for a convenient time sharing of resources across several camera streams.
4.2 On-Chip Memory Blocks

Timely access to parallel data is a bottleneck in many high speed Application Specific Integrated Circuit (ASIC) as well as in FPGA architecture designs. When comparing the characteristics of off-chip memory with on-chip memory, it should be noted that off-chip memories generally have a larger capacity but longer access times, due to limitations of inter-chip communication delays. On the other hand, on-chip memories have a smaller size than dedicated memory technologies (e.g. Dynamic Random Access Memory (DRAM)) that are not compatible with standard Complementary Metal Oxide Semiconductor (CMOS) technology. System-on-Chip (SoC) on-chip memories are however much faster and have the additional benefit that multiple memories can be used and accessed in parallel. To fulfil the processing requirements, the parallelism capabilities of on-chip memories are exploited in the architecture presented in order to feed parallel data to the next block.

In ASIC or FPGA designs, two kinds of primitives can be used to implement on-chip memory. The first kind is the internal register, which is constructed out of unstructured flip-flops. With high memory needs, the number of flip-flops used is large and can substantially drain the on-chip available resources, which can present significant routing issues. This is certainly true with low cost FPGA’s, where the number of logic elements and routing possibilities are limited.

The second type of internal memory is an on-chip memory block which consists of a compact and regular dedicated structured memory layout. These memory blocks can easily be added to a SoC or FPGA without using too much die space. The number of separate on-chip memory blocks available on an ASIC or FPGA determines the data parallelism that is possible on it. For example, the Altera Cyclone IV GX FPGA contains between 540K and 6,480Kbits of memory organized in 60 to 720 memory blocks. Each memory block has a data interface of 36 bits with 9,216 bits of total memory. This means that a Cyclone IV GX has the possibility to have an on-chip parallel memory interface of 2,160 bits for the smallest version to 25,920 bits for the largest version of that FPGA family.
4.3 Line Buffers

Due to the sequential way in which digital video data is presented, video signal processing architectures are traditionally built around line buffers (see Figure 4.1). In the line buffers, a number of the most recent scan lines are memorized on-chip. Line buffers could be implemented as shift registers, but are currently efficiently implemented as on-chip memory blocks with dedicated addressing logic. They are used as FIFO’s, typically one FIFO per scan line. At the outputs of the FIFO’s, corresponding column pixels for the recent scan lines are accessed. These can then be stored in a shift register array with the size of the region of interest for window based video operations such as filtering, edge detection, sharpening, resampling etc.

![Line Buffer Diagram](image)

Figure 4.1: Line buffer implementation for a support region of $3 \times 3$.

However, the traditional scan line based FIFO architecture does not fully exploit the parallelism that is available with multiple on-chip memory blocks.

4.4 Parallel Memory Architecture

We propose an on-chip memory architecture that allows parallel access to all pixels located in a chosen window. A window is defined here as a rectangular region of interest of the image. This region should be read out on each clock cycle for real-time operation and in order to avoid that memory access becomes a bottleneck in the data path.

The on-chip memory blocks are organized in a matrix structure, with the number of on-chip memory blocks equal to the size of the window. An example of this memory organization for a window of $3 \times 3$ can be seen in figure 4.3. The complete image is shown in Figure 4.2, where each square represents a single pixel.
Figure 4.2: Complete image from camera.

Figure 4.3: Parallel memory organisation (colour of blocks indicates contents of memory - pixel colour).
Figure 4.3 illustrates a memory structure of which the window size equals $3 \times 3$. Nine different on-chip memories are needed to parallel read out a complete window, they are labelled M1 to M9. The first pixel of the stream is written in memory block M1, the second pixel in memory block M2 and the third pixel in memory block M3. Since the window width is three, only three pixels need to be accessed on the same line at the same time. This means that the fourth pixel from the first line can also be written in memory M1, but at another address. When a new image row starts, the new pixel will be written in another memory block and will always start at address zero.

The new pixel data is stored in exactly one on-chip memory block. An Address Management Unit (AMU) is needed to keep track of which on-chip memory to write to (called the base memory). This functionality can be implemented using a simple counter that keeps track of the current memory row and width. The AMU also needs to keep track of which address to write to (called the base address).

On every clock cycle, reading the on-chip memory contents for a chosen window is performed in parallel. Depending on the window that is chosen, the address of the memory should be set accordingly. For reading the on-chip memories, the base memory and the base address are defined respectively as the memory block that contains the right bottom pixels of the window and its address. Figure 4.3 illustrates that, depending on which memory column contains the last column of the window, the memory addressing for the other memory columns is different from the base address. When the memory column number is greater than the base memory column number (the column that contains the utmost right column of the window), the address equals the base address minus one, otherwise the address equals the base address.

For example when selecting a $3 \times 3$ window with base memory M8, the base memory column equals 2 (M2, M5 and M8). This means that for memory column 1 (M1, M4 and M7) the address is equal to the base address and for memory column 3 (M3, M6 and M9) the address is equal to the base address minus one.

When a window is read, only the base address will be selected. Since the address decoder only keeps track of window width and window height, the address decoder scales well with an increase of window size. The resulting memory architecture for both reading as writing for a $3 \times 3$ window is shown on Figure 4.4.

This architecture allows for random window access. For a stereo matching application, this makes it possible to read out two different windows in order to effectively
Figure 4.4: Parallel memory implementation.
double the disparity range possibility without doubling the consumed resources (see Figure 4.5). However, for this to work, the clock frequency of the stereo matching algorithm needs to be double that of the pixel stream clock. Doubling or even quadrupling the frequency for a real-time vision system is realistic since the pixel stream clock is not that high: e.g. a camera with a resolution of $800 \times 600$ and a refresh rate of $56$ Hz, has a pixel stream clock of $36$ MHz.

![Figure 4.5: Timed shared window read by increasing the frequency of the read clock.](image)

4.5 Window Translation

The window retrieved from the memories is not immediately usable for further processing. Although all window pixels are available at the same clock cycle, they still need to be reorganized in a way that the window orientation in the memory represents the window orientation on the image. For example if a window with base memory M9 and base address “2” is selected, the results in Figure 4.6 are obtained. In this case the window orientations in the memory and on the image are the same. However, if a window with base memory M2 and address “4” is selected the results in Figure 4.6...
are obtained. In this case the window orientations in the memory and on the image are different.

Due to the addressing order of the AMU, the pixels contained in a memory window are scrambled in comparison with the image window. Two matrix circular shifts in both dimensions are needed to unscramble the window. The size of the circular shift depends on the memory column and row of the base memory block.

The descrambling can be implemented by either a switch fabric or by matrix circular shifts. In case the descrambling is implemented by matrix circular shifts the descrambling is replaced by two array circular shift operations (see Figure 4.7). First: the memory elements are row concatenated and bit shifted, so that the resulting window is rotated down. Second: this output is column concatenated and bit shifted so that the resulting window is rotated to the left.

### 4.6 Modified Parallel Memory Architecture

To avoid the need for a vertical translation, a modification of the memory structure is possible. In this architecture, the new pixels are always stored on the top layer of the memory elements (see Figure 4.8). While the lower layers of the memory elements store the data previously stored in the upper layers.

Instead of writing only to one memory element, all memory elements in the same column are written to. Each new pixel value is stored in the top layer of the memory...
elements. These memory elements contain the current image line. The pixel value currently stored in this memory will be read out and written to a memory element in the same column but at a lower layer. This will continue for all layers.

The reading part is almost the same as the one described in Section 4.4. However, to allow the rewrite of the old data to a lower layer, the read and write address should be the same for at least one clock cycle.

Figure 4.7: Window orientation translation.

Figure 4.8: Modified parallel memory implementation: global data write connections (left) and close-up of one memory element (right).
4.7 Implementation and Results

The architecture and methods presented in this chapter have been implemented on an FPGA system, based on an Altera Cyclone II with 68,416 logic elements and 250 memory blocks. The source of the input stream is a camera setup providing two 16 bit colour images and the disparity information is stored in an external frame buffer that is connected to an LCD screen.

The proposed memory architecture is compared with the standard line buffer based architecture in function of their logic element and memory element usage with different window sizes. The Quartus-II version 9.0 logic synthesis and fitter tool have been used for the hardware implementation.

Table 4.1 illustrates that the proposed memory architecture (on-chip memory blocks + window translation) uses approximately 30% less FPGA resources than the line buffer based memory architecture by maximizing the use of the parallel memory blocks available on the FPGA.

Window translation takes up most of the resources in the proposed memory architecture. This can be solved by tailoring the downstream algorithms to accommodate for a scrambled window. In the case of a stereo matching implementation, only the horizontal translation is needed. This partial window orientation will have no effect on the quality of the generated disparity maps.

4.8 Conclusion

An on-chip memory block architecture is presented that allows for parallel access to all pixels located in a chosen window of the image. In comparison to a standard line-buffer
implementation, the proposed architecture makes better use of available resources. However a badly chosen window translation implementation can substantially increase the required resources.

The proposed architecture allows random window access which can be used in stereo vision to increase the disparity range linearly with respect to the operating frequency.
Chapter 5

Pixel Stream Comparison

Contents

4.1 Introduction .................................................. 54
4.2 On-Chip Memory Blocks ................................. 55
4.3 Line Buffers .................................................. 56
4.4 Parallel Memory Architecture ....................... 56
4.5 Window Translation ....................................... 60
4.6 Modified Parallel Memory Architecture .......... 61
4.7 Implementation and Results ......................... 63
4.8 Conclusion .................................................. 63

5.1 Introduction

Matching pixels within images is an import subject in computer vision. Dense motion estimation and stereo correspondence are two main application fields for image matching. Matching a single pixel $p$ will lead to many mismatches. Taking a rectangular support region $N(p)$ centred around $p$ will provide better results. Selection of the support region is important: it should be large enough to contain distinguishing features, but small enough to keep depth discontinuities. More advanced local based
matching methods use colour information to select the optimal support region. A good overview of these methods can be found in Wang and Gong [2006].

The adaptive weight algorithm proposed by Yoon and Kweon [2005] gives a support weight to each pixel in a fixed squared region. The support weights are depending on the colour and spatial difference between each pixel in the region \( N(p) \) and the centre pixel \( p \). It drastically improves the disparity map, in particular with large window sizes. Tombari et al. [2007] extends the adaptive-weight algorithm of Yoon and Kweon [2005] by using information from segmentation. It allows inclusion of connectedness of pixels and segment shapes, instead of only relying on colour and spatial distance. Implementations on hardware of these methods are difficult to find. Most stereo vision implementations on a custom System-on-Chip (SoC) or an Field Programmable Gate Array (FPGA) use a fixed cost aggregation window (Lee et al. [2005], Yi et al. [2004], Murphy et al. [2007], Mingxiang and Yunde [2006]) or an adaptable rectangular cost aggregation window (Hariyama and Kameyama [2001]). Recently Ding et al. [2011] proposed a hardware implementation where the adaptive weight algorithm is decomposed in a horizontal and vertical part.

In this chapter, different similarity measurements are compared. Including a novel binary support area, which is tailored to save SoC resources.

The stereo vision application is used as a reference for comparing the dissimilarity costs. Four stereo data sets of the Middlebury benchmark (Scharstein and Szeliski [2002], Scharstein and Szeliski [2003]) are used. These are the Cones, Teddy, Tsukuba and the Venus dataset. The average error rate in the non occluded areas is used as benchmark metric. Only looking at this error value, will give a distorted view of the performance. It is certainly as important to look at the generated disparity maps.

The two input streams of the cameras are compared with each other using a Matching Cost (MC) computation. For each disparity value \( d \) in the set of disparity valued \( S_d \) (also called the disparity range), pixel \( p_r \) on the reference image is compared with pixel \( p_t \) on the target image (5.1). The disparity value which results in the lowest MC is chosen as the disparity value \( d_p \) at pixel \( p_r \).

\[
d_p = \arg\min_{d \in S_d} MC(p_t, p_r)
\]  

(5.1)

Most of the methods discussed in this dissertation make use of the \( YC_B C_R \) colour space. The reconstructed \( RGB \) colour space can be transformed into the \( YC_B C_R \)
colour space using the following formulas.

\[
\begin{align*}
Y &= 16 + (66 \cdot R + 129 \cdot G + 25 \cdot B) \\
C_B &= 128 + (-38 \cdot R - 74 \cdot G + 112 \cdot B) \\
C_R &= 128 + (112 \cdot R - 94 \cdot G - 18 \cdot B)
\end{align*}
\] (5.2)

### 5.2 Similarity Computation

Many methods exist to calculate the difference between two images. Overviews can be found in Hirschmuller and Scharstein [2007], Gong et al. [2007] and Cyganek [2004]. This section discusses two of the methods which are most commonly used in hardware implementation: the sum of intensity differences (Lee et al. [2005], Yi et al. [2004], Murphy et al. [2007], Mingxiang and Yunde [2006]) and the census transform (Jin et al. [2010]). The MC between a pixel \( p_r \) in the reference image and \( p_t \) in the target image can be expressed as

\[
MC(p_t, p_r) = \sum_{q_t \in N(p_t), q_r \in N(p_r)} e(q_t, q_r)
\] (5.3)

For each pixel \( p \) under consideration, a support region \( N(p) \) is defined. This support region is rectangular and centred in \( p \). A pixel \( q \) is defined as a pixel belonging to \( N(p) \). Function \( e(q_t, q_r) \) computes the similarity between a pixel on the reference image \( (q_r) \) with a pixel on the target image \( (q_t) \).

The most used dissimilarity computation is the Sum of Absolute Differences (SAD) (Hirschmuller and Scharstein [2007]). The matching cost is obtained by calculating the absolute value of the difference between the intensities of both pixels (5.4).

\[
e(q_t, q_r) = |I_t(q_t) - I_r(q_r)|
\] (5.4)

The Sum of Squared Differences (SSD) has a higher complexity compared to SAD. The matching cost is obtained by calculating the squared difference of both pixels (5.5).

\[
e(q_t, q_r) = (I_t(q_t) - I_r(q_r))^2
\] (5.5)

To limit the influence of outliers, both Absolute Differences (AD) and Squared Differences (SD) can be extended with a minimal operator (5.6) (5.7). Results that
are above a certain threshold $T$ are truncated. This adaptation reduces the effect of occluded pixels (Ding et al. [2011]). In Figure 5.1 and Figure 5.2, several tests have been performed using the Middlebury dataset. For each image pair, a Winner Takes All (WTA) approach with Truncated sum of Absolute Differences (TAD) or Truncated sum of Squared Differences (TSD) as matching cost is used. The threshold $T$ is in relationship with the pixel range, which equals 256 for the chosen datasets. Figure 5.1 and Figure 5.2 illustrates that the lowest error is obtained with a threshold around 16 both TAD as TSD. This threshold has also an impact in the complexity of the aggregation step, since this directly influences the bit-size of the adders for the matching cost aggregation step.

\[
e(q_t, q_r) = \min \{ |I_t(q_t) - I_r(q_r)|, T \}\]

(5.6)

\[
e(q_t, q_r) = \min \left\{ \left( I_t(q_t) - I_r(q_r) \right)^2, T^2 \right\}\]

(5.7)

![Figure 5.1: TAD parameters for the Middlebury dataset (average equals the sum of the datasets divided by four).](image)

The Census Transform (CT) is a non-parametric local area-based dissimilarity calculation (Cyganek [2004]). It has high robustness to illumination variations. It
first calculates a census bit-vector for both the reference as the target window (5.8). Each pixel $I(q)$ in the support area is compared with the centre pixel $I(p)$. Second, the Hamming distance ($hdist$) is calculated over the two census bit-vectors (5.9).

$$census(p, q) = I(q) < I(p)$$ (5.8)

$$e(p_t, p_r) = hdist(census(p_t, q_t), census(p_r, q_r))$$ (5.9)

When using a fixed squared aggregation window of $n \times n$, Table 5.1 illustrates the algebraic complexity for each dissimilarity calculation. It is clear that SSD and TSD has a much higher complexity compared to the other methods. The large number of multipliers needed for a realistic window size will easily use up all available hardware multipliers on a FPGA implementation. The CT has the least amount of complexity, this is because the input to the cost aggregation step of the CT only contains single bit values.

A comparison of different window sizes can be found in Figure 5.3. For each image pair of the Middlebury dataset, a WTA approach with the different matching costs
is used. For TAD and TSD a threshold $T$ of 16 is selected. Notice that the errors of SAD and SSD are almost similar. Truncation of the output values of SAD and SSD clearly has a positive influence on the error percentage, indicating that outliers have an influence for both SAD as SSD. The CT is clearly the worst matching cost computation.

![Figure 5.3: Error percentage in function of window size for the different similarity calculations.](image)

Figure 5.3 illustrates the different dissimilarity calculations for a fixed support region $N(p)$. For each image pair of the Middlebury dataset, a WTA approach with the different matching costs is used. For TAD and TSD a threshold $T$ of 16 is selected.
With all methods, the image becomes washed out with increased window size.

### 5.3 Adaptive Support Weights

#### 5.3.1 Original Algorithm

When using a fixed window shape like in the previous section, implicitly depth continuity across this window is assumed. This assumption is not correct at depth edges, where the centre pixel depth is different from some (or the majority) of the surrounding pixels depths. A more conservative assumption is to assume depth continuity across pixels with similar colour. The Adaptive Weights (AW) algorithm proposed by Yoon and Kweon [2005] assumes that neighbouring pixels with similar colour and a distance closer to the centre pixel are likely from the same object. After this modification, the matching costs between a pixel $p_t$ and $p_r$ can be expressed as:

$$MC(p_t, p_r) = \frac{\sum_{q_t \in N(p_t), q_r \in N(p_r)} w(p_t, q_t) w(p_r, q_r) e(q_t, q_r)}{\sum_{q_t \in N(p_t), q_r \in N(p_r)} w(p_t, q_t) w(p_r, q_r)}$$  \hspace{1cm} (5.10)

Ding et al. [2011] proposed to simplify equation (5.10) for hardware implementation, by removing the adaptive weight calculation for the target image (5.11).

$$MC(p_t, p_r) = \frac{\sum_{q_t \in N(p_t), q_r \in N(p_r)} w(p_r, q_r) e(q_t, q_r)}{\sum_{q_r \in N(p_r)} w(p_r, q_r)}$$  \hspace{1cm} (5.11)

$w(p_r, q_r)$ is independent of the target pixel $p_t$ and will be the same for all disparity values. Hence, the denominator of equation (5.11) can be removed. This leads to the following simplified equation:

$$MC(p_t, p_r) = \sum_{q_r \in N(p_r)} w(p_r, q_r) e(q_t, q_r)$$  \hspace{1cm} (5.12)

Where $w(p_r, q_r)$ are the adaptive support weights (AW) which changes the influence of certain pixel matches. The support weight of a pixel is in function of its colour difference and its spatial distance and can be written as:

$$w(p, q) = f_c(\triangle c_{pq}) \times f_p(\triangle g_{pq})$$  \hspace{1cm} (5.13)
Figure 5.4: Disparity map quality of the Tsukuba stereo pair in function of window size and dissimilarity computation type (from left to right: $5 \times 5$, $11 \times 11$, $23 \times 23$, $35 \times 35$ window size).
The weights are assigned by a Gaussian function:

\[ f(p, q) = \exp \left( -\frac{\triangle pq}{\gamma} \right) \]  \hspace{1cm} (5.14)

Substituting (5.14) in (5.13), gives equation (5.15) where the Gaussian variance \( \gamma \) tunes the strengths of the resulting weights.

\[ w(p, q) = \exp \left( -\left( \frac{\triangle c_{pq}}{\gamma_c} + \frac{\triangle g_{pq}}{\gamma_p} \right) \right) \]  \hspace{1cm} (5.15)

The colour difference \( \triangle c_{pq} \) is calculated by comparing the chroma components of each pixel (5.16). When the combined chroma differences are low, the support weights should be high.

\[ \triangle c_{pq} = |C_R(q) - C_R(p)| + |C_B(q) - C_B(p)| \]  \hspace{1cm} (5.16)

The spatial distance \( \triangle g_{pq} \) calculates the euclidean distance between the pixel in the support region \( q \) and the centre \( p \) (5.17).

\[ \triangle g_{pq} = \sqrt{(q_x - p_x)^2 + (q_y - p_y)^2} \]  \hspace{1cm} (5.17)

In Figure 5.5, the spatial distance is illustrated for a support window with a size of 35×35. The left image (5.5a) illustrates the real values, the middle figure (5.5b) shows the truncated conversion and the figure on the right (5.5c) the binary conversion. From left to right, the regulation possibility of the distance parameter becomes less nuanced but can be implemented with more efficiency. The spatial distance can be pre-calculated and stored in a look-up table.

In order to avoid using multiplications in formula (5.12), Ding et al. [2011] proposed to truncate the AW values to powers of two. In this case the multiplications can be replaced by shift registers.

### 5.3.2 Binary Support Weights

To save SoC resources, a derivative of the AW has been introduced in this dissertation which uses binary weights, this is called the (Adaptive Binary Weights (ABW)). A value of 0 means that this pixel does not belong to the support window of the centre.
pixel and a value of 1 means that this pixel belongs to the support window of the centre pixel. There is no gradient between these two extremes. In formula (5.18), $\triangle c_{pq}$ stands for the colour difference between the centre pixel $p$ and each point $q$ belonging to region $N(p)$ centred in $p$. This colour difference is compared with a threshold $th$ in Formula (5.19) to obtain the binary support region $w$. Note that this support window can be used for any shape possible within the original window. However, just like the original AW algorithm, connectivity between pixels and segment shape are not taken into account.

$$\begin{align*}
\triangle c_r, pq &= |C_R(q) - C_R(p)| \\
\triangle c_b, pq &= |C_B(q) - C_B(p)|
\end{align*}$$  \hspace{1cm} (5.18)

$$w(p, q) = \begin{cases} 
0, & \text{if } (\triangle c_r, pq > th) \land (\triangle c_b, pq > th) \\
1, & \text{otherwise}
\end{cases}$$  \hspace{1cm} (5.19)

The ABW has also been extended to incorporate the spatial distance presented before.

$$w(p, q) = \begin{cases} 
0, & \text{if } (\triangle c_r, pq > th) \land (\triangle c_b, pq > th) \land (\triangle g_{pq} = 1) \\
1, & \text{otherwise}
\end{cases}$$  \hspace{1cm} (5.20)

Figure 5.6 maps the error percentage against the threshold value, for each image pair of the Middlebury dataset. It illustrates that the lowest error is obtained with a threshold at 32, which is 1/8 of the pixel range.

Both types of support window do not provide a method to distinguish unconnected regions. In order to do so, the support window generated by the ABW algorithm is passed through a connectivity preserving step. The goal of this algorithm is to remove
regions which are not connected with the centre pixel $p$. For each layer around $p$, the connectivity with the previous layer is checked. If the pixel is not connected to an active pixel in the previous layer, the adaptive weights will become 0. This calculation is repeated for each layer around $p$. Figure 5.7 illustrates the difference between the ABW with and without connectivity preservation step.

Figure 5.7: Adaptive binary weights for a window size of $35 \times 35$ (left: colour window, middle: ABW, right: ABW with connectivity preservation.)

Figure 5.8 and equation (5.21) illustrate an example where the top row of layer two is being calculated (indicated by the arrows). For each pixel, two or three OR operators and one AND operator are needed.
\[
\begin{align*}
w_c(u-1, v-2) &= w(u-1, v-2) \land (w(u-1, v-1) \lor w(u, v-1)) \\
w_c(u, v-2) &= w(u, v-2) \land (w(u-1, v-1) \lor w(u, v-1) \lor w(u+1, v-1)) \\
w_c(u+1, v-2) &= w(u+1, v-2) \land (w(u+1, v-1) \lor w(u, v-1))
\end{align*}
\] (5.21)

5.3.3 Results and Performance

The resulting adaptive weights windows for the different presented methods illustrated in Figure 5.9. The proposed Adaptive Binary Weights (ABW) method performs well in finding objects in the support region. The extra connectivity preservation step removes regions which have no connectivity path to the centre pixel \( p \).

Figure 5.10 illustrates the error difference between the different support regions. The Adaptive Weights (AW) method provides a lower error percentage with larger window sizes compared to the full support region. The ABW method has a slightly higher error percentage compared to AW.

The advantage of using adaptive support weights becomes more pronounced when looking at Figure 5.11. It shows the disparity maps of the four datasets for a window size of 35 × 35. The results indicate that the quality of the resulting disparity map increases when using an adaptive support weight. With larger window sizes, the smoothing effect remains while preserving small details around the edges.
Figure 5.9: Adaptive support weights, squared area around $p$ (size: 35 × 35).
5.3.4 Complexity

When comparing the complexity between the different support windows on Table 5.2, it is clear that the ABW method used the lowest number of resources. For the AW method, the large number of multipliers needed for a realistic window size will, in case of an implementation on an FPGA, easily use up all available hardware multipliers.

To reduce the overall complexity, fixed region window aggregation can easily be split into a horizontal and vertical aggregation step, also called the sliding window approach (Lee et al. [2005]). The complexity of the aggregation step is hence reduced

<table>
<thead>
<tr>
<th></th>
<th>Multiplication</th>
<th>Adders</th>
<th>Shift</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AW</strong></td>
<td>$2n^2$</td>
<td>$2(n-1)^2$</td>
<td>0</td>
</tr>
<tr>
<td>+Truncation</td>
<td>0</td>
<td>$2(n-1)^2$</td>
<td>2n$^2$</td>
</tr>
<tr>
<td><strong>ABW</strong></td>
<td>0</td>
<td>$3(n-1)^2$</td>
<td>0</td>
</tr>
<tr>
<td>+Connectivity preservation</td>
<td>0</td>
<td>$4(n-1)^2$</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 5.2: Algebraic complexity for different adaptable weight calculations
Figure 5.11: Disparity map quality in function of dataset and adaptive support type (from left to right: Cones, Teddy, Tsukuba and Venus dataset).
from $n^2$ to $2n$. When using an adaptive support region, this is not feasible anymore. Recently several methods have been proposed to solve this issue by also splitting the AW method into a horizontal and vertical support window (Ding et al. [2011]). These however do not provide correct support in windows with slanted lines (see Figure 5.12). A better solution is to use a sliding window segmentation approach (Gerrits and Bekaert [2006], Tombari et al. [2007], Hosni et al. [2010]). Further research is needed to investigate whether this method can be adapted for local window comparison without the need for a pre-segmented image.

(a) Colour.

(b) Adaptive support weights.

(c) Binary adaptive support weights.

Figure 5.12: Adaptive support weights with separate vertical and horizontal aggregation step, squared area around $p$ (size: $35 \times 35$).
5.4 Implementation

The implementation is specially adapted for System-on-Chip (SoC) implementation. All operations for each pixel $q$ in the support region $N$ are applied in parallel. The binary spatial distance (see Figure 5.5c) directly influences the size of the support region $N$ centred in $p$. Pixels with no support will have no resources connected to them.

The truncated absolute difference has been selected for matching cost hardware development. First the absolute difference for two numbers is calculated; the difference is taken and if the result is negative, the two’s complement of this result is taken (see Figure 5.13).

Second the absolute value is truncated to a certain value (see Figure 5.14). This threshold is selected to be a power of two. In this case, if one of the more significant bits is high, the output will be truncated.

Binary Window with connectivity preservation is selected as adaptable support window. First the ABW is calculated by comparing the blue and red chrome components (in the YCrCb colour space) between pixels in the support window ($q$) and the centre pixel ($p$). For each chroma component, the absolute difference is compared with a threshold value (see Figure 5.15). Also in this case the threshold is selected to be a power of two.

Second this window is passed through a connectivity preserving step. For each layer around the centre pixel $p$, pixels are checked if they are connected to an active pixel and if they are active themselves. This is an iterative process for each layer around $p$. Figure 5.16 illustrates an example of this calculation for one pixel.
Figure 5.14: Truncated absolute difference.

Figure 5.15: Chroma difference.
The summation part or cost aggregation step of the Truncated sum of Absolute Differences (TAD) uses \( n \) number of adders using a tree structure. The matching costs are pair wise summed and the results are stored in registers. Afterwards, these registers are pair wise summed and stored in registers. These steps are repeated until one value remains (see Figure 5.17).

Since the adaptive support weights in this architecture are 0 or 1, the multiplication in Figure 5.17 can be replaced by an AND operator. This will accommodate for an efficient hardware implementation (see Figure 5.17).
The Winner Takes All (WTA) minima selection is based on an iterative minima tree calculation (see Figure 5.18). The TAD results are pair wise compared, while each time the lowest value is stored in a register. Afterwards, these registers are pair wise compared and stored in registers. These steps are repeated until one value remains.

All of these parts are suitable for pipelining by storing of the intermediate results in registers. IP block generators have been developed to generate this hardware architecture for a given window size.

Resource usage of the methods discussed in this section can be found in Table 5.3 and Table 5.4. For each pixel on the reference image, only one ABW calculation is needed, however multiple Sum of Absolute Differences (SAD) or TAD calculations are necessary. For this reason, the logic elements resource usage is more important with SAD and TAD compared to ABW.

The region size determines the resource usage of the window comparison steps. In Table 5.3, this region size has a quadratic relationship with the window size. In Table 5.4, the region size is reduced by approximately 20%, by using the spatial distance parameter. This leads to an overall reduction of 20% in logic element usage.
### Table 5.3: Logic element usage, without spatial distance region support selection.

<table>
<thead>
<tr>
<th>Window Size</th>
<th>Region Size</th>
<th>SAD</th>
<th>TAD</th>
<th>ABW</th>
<th>Connectivity Preservation</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 × 3</td>
<td>9</td>
<td>244</td>
<td>266</td>
<td>338</td>
<td>0</td>
</tr>
<tr>
<td>5 × 5</td>
<td>25</td>
<td>697</td>
<td>745</td>
<td>986</td>
<td>25</td>
</tr>
<tr>
<td>7 × 7</td>
<td>49</td>
<td>1,381</td>
<td>1,470</td>
<td>1,970</td>
<td>99</td>
</tr>
<tr>
<td>9 × 9</td>
<td>81</td>
<td>2,289</td>
<td>2,441</td>
<td>3,282</td>
<td>245</td>
</tr>
<tr>
<td>11 × 11</td>
<td>121</td>
<td>3,407</td>
<td>3,632</td>
<td>4,922</td>
<td>487</td>
</tr>
<tr>
<td>13 × 13</td>
<td>169</td>
<td>4,779</td>
<td>5,101</td>
<td>6,890</td>
<td>849</td>
</tr>
<tr>
<td>15 × 15</td>
<td>225</td>
<td>6,337</td>
<td>6,776</td>
<td>9,186</td>
<td>1,355</td>
</tr>
<tr>
<td>17 × 17</td>
<td>289</td>
<td>8,172</td>
<td>8,712</td>
<td>11,810</td>
<td>2,895</td>
</tr>
<tr>
<td>19 × 19</td>
<td>361</td>
<td>10,223</td>
<td>10,898</td>
<td>14,762</td>
<td>2,895</td>
</tr>
<tr>
<td>21 × 21</td>
<td>441</td>
<td>12,469</td>
<td>13,294</td>
<td>18,042</td>
<td>3,977</td>
</tr>
</tbody>
</table>

### Table 5.4: Logic element usage, with spatial distance region support selection.

<table>
<thead>
<tr>
<th>Window Size</th>
<th>Region Size</th>
<th>SAD</th>
<th>TAD</th>
<th>ABW</th>
<th>Connectivity Preservation</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 × 3</td>
<td>9</td>
<td>244</td>
<td>266</td>
<td>338</td>
<td>0</td>
</tr>
<tr>
<td>5 × 5</td>
<td>21</td>
<td>613</td>
<td>641</td>
<td>822</td>
<td>21</td>
</tr>
<tr>
<td>7 × 7</td>
<td>45</td>
<td>1,288</td>
<td>1,362</td>
<td>1,806</td>
<td>91</td>
</tr>
<tr>
<td>9 × 9</td>
<td>69</td>
<td>1,952</td>
<td>2,084</td>
<td>2,790</td>
<td>209</td>
</tr>
<tr>
<td>11 × 11</td>
<td>101</td>
<td>2,281</td>
<td>3,055</td>
<td>4,102</td>
<td>407</td>
</tr>
<tr>
<td>13 × 13</td>
<td>145</td>
<td>4,099</td>
<td>4,376</td>
<td>5,906</td>
<td>729</td>
</tr>
<tr>
<td>15 × 15</td>
<td>285</td>
<td>5,214</td>
<td>5,567</td>
<td>7,546</td>
<td>1,115</td>
</tr>
<tr>
<td>17 × 17</td>
<td>241</td>
<td>6,790</td>
<td>7,246</td>
<td>9,842</td>
<td>1,693</td>
</tr>
<tr>
<td>19 × 19</td>
<td>301</td>
<td>8,554</td>
<td>9,109</td>
<td>12,302</td>
<td>2,415</td>
</tr>
<tr>
<td>21 × 21</td>
<td>373</td>
<td>10,577</td>
<td>11,262</td>
<td>15,254</td>
<td>3,365</td>
</tr>
</tbody>
</table>
5.5 Conclusion

A novel binary adaptive weight aggregation architecture is presented that allows for depth discontinuity preservation within the reference support region $N(p)$. For each support region, binary weights are calculated which select the supporting pixels in the cost aggregation phase of the matching cost computation. This selection is performed using colour similarity and a spatial distance metric. The results of the implementation indicate that the binary weight implementation improves the disparity map quality with both small as well as large window sizes. It is shown that this architecture can be implemented efficiently into a SoC design, however since the sliding window approach is not used, the resource usage rises rapidly with increased window size.

This architecture is chosen with particular attention given to pipelining and parallelism possibilities and is fully scalable to allow for custom SoC implementations, as well as rapid prototyping on Field Programmable Gate Array (FPGA)’s.
Part II

Multi-Camera Vision Systems
6.1 Introduction

Ambiguous matches are common with the matching methods presented in Chapter 5. A major issue is to know which disparity matches are correct. This Chapter presents a methodology to construct a binary confidence value for every pixel of a disparity map. We start by constructing 72 different confidence metrics, including more traditional
ones and new metrics based on neighbourhood information. Construction of the binary confidence value from these metrics is hence viewed as a two-class classification problem where three classifiers are evaluated. Only metrics and classifiers that are suitable for VLSI hardware implementation will be evaluated. Evaluation of the constructed classifiers is performed on an indoor dataset of stereo images.

Different confidence metrics have been proposed in the literature, of which the left-right consistency check is most often used (Yoon and Kweon [2005], Jin et al. [2010], Ambrosch and Kubinger [2010]). A thorough overview of different confidence metrics based on the matching cost curve have been performed by Hu and Mordohai [2010], Hu and Mordohai [2012]. Haeusler and Klette [2012] recently, extended the former research with information from the region \( N(p) \) around the pixel \( p \) under consideration.

Independent of Haeusler and Klette [2012], we have developed and investigated several cost metrics based on neighbourhood information. We extended this approach by also taking into account an adaptable support region (see Chapter 5). Our results indicate that this approach gives cost metrics with a higher classification power.

Having a binary confidence map can be used to assist post-processing tasks like obstacle avoidance (Nalpantidis et al. [2009]; Nalpantidis and Gasteratos [2009]). It can help selecting which disparity values to choose while combining disparity maps generated from different window sizes or image pyramids. Or assist in the disparity refinement step by indicating the bad matches.

### 6.2 Confidence Metrics

The confidence measurement that is primarily used for detecting incorrect disparities, is the left right consistency test (Yoon and Kweon [2005], Jin et al. [2010], Ambrosch and Kubinger [2010]). A good comparison between different cost metrics can be found in the evaluation paper of Hu and Mordohai [2010]. However they do not incorporate cost metrics constructed from the neighbouring pixels. This section presents several cost metrics for usage with image matching algorithms.
6.2.1 Confidence Metrics Based on the Cost Function

These features are based on the cost function generated by the window comparison algorithm. For every window that needs to be matched, a Sum of Absolute Differences (SAD) calculation is performed. The larger the disparity search width, the more SAD calculations are needed. The result is an array that contains a SAD score for each disparity value (usually starting from 0). This array is also known as the cost function (see Figure 6.1). In this dissertation, $C_1$ stands for the lowest SAD score (the minima of the Cost Curve). $C_2$ stands for the second lowest SAD score, and so on. Their corresponding disparity values are indicated by $D_1$ and $D_2$. Most stereo vision algorithms calculate the disparity from the cost function using a ”Winner Takes All” (WTA) approach. Doing so, the minima of the cost function ($C_1$) will become the calculated disparity: $D_1$.

![Figure 6.1: Window comparison cost function.](image)

The **Matching Cost** (MC) is the minimum value of the cost function. A high score will be a good indication of a wrong disparity value.

$$MC = C_1 \quad (6.1)$$

The **Curvature** (CUR) calculates the shape of the cost function around the minimum value. It is an indication of the sharpness of the cost function around the best match.

$$CUR = -2 * C(D_1) + C(D_1 - 1) + C(D_1 + 1) \quad (6.2)$$
When \((D_1 - 1)\) or \((D_1 + 1)\) is outside the disparity range, the available neighbour is used twice.

The **Peak Ratio (PKR)** calculates the division of the best match divided by the second best score.

\[
PKR = \frac{C_1}{C_2}
\]  
(6.3)

The **Cost Curve Threshold (CCT)** calculates the number of cost function points that are located above a certain threshold. This threshold is determined by multiplying the score of the best match \((C_1)\) with a tuning parameter \((a)\).

\[
CCT = \sum_d C_d > (a \times C_1)
\]  
(6.4)

### 6.2.2 Consistency Between Left and Right Disparity Maps

These features check if the disparity map of the left image is consistent with the disparity map of the right image.

The **Left Right Consistency (LRC)** calculates the difference between the left and right disparity map. It is often used as a single threshold confidence check (Yoon and Kweon [2006], Jin et al. [2010], Ambrosch and Kubinger [2010]).

\[
LRC(x, y) = |D_1 - D_R(x - D_1, y)|
\]  
(6.5)

The **Left Right Difference (LRD)** is a combination of a Cost Curve calculation of the difference of the two best matches and the LRC

\[
LRD(x, y) = \frac{(C_2 - C_1)}{|D_1 - D_R(x - D_1, y)|}
\]  
(6.6)

### 6.2.3 Confidence Metrics Based on Neighbourhood Information

These features look at information that neighbouring pixels can provide (see Figure 6.2).

The **Texture (TEX)** uses a fixed window of luminance information around the investigated pixel and measures the amount of texture it contains. The intuition
behind it is that textureless regions will provide more incorrect disparity values.

\[ TEX(p_r) = \max_{q_r \in N(p_r)} (I(q_r)) - \min_{q_r \in N(p_r)} (I(q_r)) \] (6.7)

The **Segmentation Size (SEG)** calculates the sum of the Adaptive Binary Weights (ABW) proposed in Chapter 5. The calculation of the ABW can be the same as used in the cost aggregation phase.

\[ SEG(p_r) = \sum_{q_r \in N(p_r)} w(p_r, q_r) \] (6.8)

The **Sum of Neighbouring Disparity Differences (SNDD)** uses a fixed region \( N(p) \) of disparity values around the investigated pixel \( p \) and calculates the disparity differences in this region.

\[ SNDD(p_r) = \sum_{q_r \in N(p_r)} |D_1(p_r) - D_1(q_r)| \] (6.9)

The **Sum of Neighbouring Disparity Differences using adaptive Binary Weights (SNDDBW)** is similar to SNDD, but instead of using a fixed region it uses only the neighbouring pixels of which the ABW are one. This is a different usage of the binary weights presented in Chapter 5. The assumption is only to have disparity continuity across pixels of the same object.

\[ SNDDBW(p_r) = \frac{\sum_{q_r \in N(p_r)} w(p_r, q_r) \cdot |D_1(p_r) - D_1(q_r)|}{\sum_{q_r \in N(p_r)} w(p_r, q_r)} \] (6.10)
6.2.4 Confidence Metrics Based on Multiple Pixel Streams

The **Sum of Streaming Disparity Differences (SSDD)** calculates the disparity difference between the different disparity streams taking the confidence value into account. Since this feature uses the confidence value generated from the first level of classification, this feature is only used for the second level of classification.

\[
SSDD = \sum_{i=1}^{\text{streams}} \text{confidence}(i) \cdot |D_1 - D_1(i)| \tag{6.11}
\]

The **Sum of Streaming Confidence metrics (SSC)** calculates the number of streams which have a positive confidence. This feature is also only used for the second level of classification.

\[
SSC = \sum_{i=1}^{\text{streams}} \text{confidence}(i) \tag{6.12}
\]

6.3 Investigated Classification Methods

The goal of the classification step is to construct a binary confidence map, when the disparity is correct, the output of the binary classifier should be \(p\), otherwise it should be \(n\). A confidence value of \(p\) indicates a strong confidence in the correctness of the disparity value. A confidence value of \(n\) indicates a weak confidence in the correctness of the disparity value.

In order to train a classifier, it is needed to define a target output. In this case, the preferable output would be a Boolean value indicating the correctness of the disparity value (the confidence value). A pixel is defined to be correctly matched with its corresponding disparity when the calculated disparity \((D_c)\) and the real disparity \((D_r)\) do not differ more than one unit disparity value (6.13) (6.14).

\[
\text{Error Map}(p) = \begin{cases} 
1, & \text{if } D_c(p) \notin \{D_r(p), D_r(p) \pm 1\} \\
0, & \text{otherwise}
\end{cases} \tag{6.13}
\]

\[
\text{Misclassification Rate}(p) = \frac{\sum_{p \in I} (\text{Error Map}(p))}{\sum_{p \in I} (1)} \tag{6.14}
\]

When the output of the classifier is positive, and the real output value is also
positive, then this is called a True Positive (TP), on the other hand if the real output value is negative, this is called a False Positive (FP). Vice versa, a True Negative (TN) happens when both the value of the classifier and real output are negative. A False Negative (FN) occurs when the output of the classifier is negative, and the real output value is positive. In most cases, the emphasize is on minimization the false positives.

6.3.1 One Feature Threshold

The most basic classifier is to use a threshold on one feature (see Figure 6.3). During the training phase a threshold is chosen for every feature that minimizes a certain error function. The feature with the lowest score on this error function will then be chosen as a single feature in the validation step. In the end we will only have one feature and one threshold. This method can be implemented very efficiently in hardware. Implementation of threshold detection can be found in (Yoon and Kweon [2005], Jin et al. [2010], Ambrosch and Kubinger [2010]).

![Figure 6.3: One feature threshold.](image)

6.3.2 Decision Tree

The first level classifier consists of a Decision Tree (DT) for each disparity stream individually (Duda et al. [2000]). The decision tree is a top-down tree structure consisting of internal nodes, leaf nodes, and branches. Each internal node represents a decision on a feature, and each outgoing branch corresponds to a possible outcome.
Each leaf node represents a class (0 or 1 in this case). The main advantage of a decision tree is the ease of interpretation and implementation, while still being able to separate hard to separate classes. In the example of Figure 6.4, two classes are separated by the class boundary which is constructed using a small DT.

![Decision Tree Example](image)

Figure 6.4: Decision tree example: two-dimensional feature space (left) and resulting DT (right).

### 6.3.3 Artificial Neural Network

A simple Artificial Neural Network (ANN) is used as a more advanced classifier. Training of the ANN is performed in Matlab using the back-propagation algorithm (Rumelhart et al. [1986]) and a sigmoid shaped transfer function. Only small networks are checked due to the high resource usage of this method (Ferrer et al. [2004], Misra and Saha [2010]). Figure 6.5 shows a simplified example of a feedforward ANN with five input nodes, two hidden layers and one output node. Every arrow resembles a weight vector, that needs to be trained and will result in a multiplication in hardware. After the training is finished, the weights of the neural connections are known. These can be used for constructing the hardware of the ANN. Many implementations can be found in the literature, but all of them need considerable more resources compared to the previous two classification methods (Misra and Saha [2010], Ferrer et al. [2004]).

Since ANN is a black box method, it is difficult to know which feature contains the highest information for the classification problem. It is advisable to reduce the number of features using a feature selection or reduction algorithm before feeding them to an ANN in order to reduce the number of nodes in the input layer.
6.4 Hierarchical Classification

In the previous section it is explained how to construct a binary classifier for a disparity stream. This section discusses the situation when multiple disparity streams are available. In order to select one of them for generating the Disparity Space Image (DSI), a two level hierarchical classifier is constructed (see Figure 6.6). In the first level of the hierarchy, the disparity values are investigated independently of each other. For each disparity value, a binary confidence classifier is constructed using the methods presented in the previous section. These confidences are passed on to the second level classifier which selects the disparity to use, or indicates that no disparity has been found. The second level classifier chooses the final disparity from the different disparity streams by choosing the one with the lowest Sum of Streaming Disparity Differences (SSDD). A DT classifier is trained to construct a confidence value for the final disparity value. An example of this method can be found in Chapter 8.

6.5 Zoning

When having a closer look at the system architecture, we notice that most confidence metrics cannot be efficiently calculated at the same moment in the streaming pipeline.

For the first level of classification, three main timing zones have been specified.
Zone 1: Features based on $N(p)$ regarding to the input image e.g. Texture ($TEX$).

Zone 2: Features based on the cost function e.g. Matching Cost ($MC$).

Zone 3: Features based on $N(p)$ regarding the disparity image e.g. Sum of Neighbouring Disparity Differences ($SNDD$).

Features of different timing zones are expensive to synchronize. Each feature needs a buffer to temporally store its value. This is of particular importance when combining features from zone one and three. Zone one features are calculated even before the calculation of the disparity value, while zone three features are based on a region of disparity values ($N(p)$) around the disparity value of which the feature is calculated. This means that a delay of several image lines is to be expected. A solution is to split the classification method into separate classification methods for each zone. For a DT classifier, no major changes are needed, each branch of the DT can be pre-calculated in a different zone (see Figure 6.7). Since only the results need to be buffered, the width of the buffer is reduced to one. For the ANN algorithm where all inputs should be provided in the same instance, buffers need to be provided.

6.6 Classification of Disparity Values

In this section, the features presented in section 6.2 are used to train the three classification methods discussed in section 6.3. Matlab has been used to generate and classify the different features using five-fold cross validation (Duda et al. [2000]).
Figure 6.7: Example of different timing zones for classification.

dataset is partitioned into five subsets, where four subsets are used for training the classifier and one subset is used for validating the results. This is repeated five times, so that each of the five sets is used exactly once as validation set. The five validation results are averaged to generate the final result. In this chapter, three different datasets have been used to verify the results: Sawtooth, Tsukuba and Teddy. For these three datasets cost functions have been calculated using the Sum of Absolute Differences (SAD) aggregation method for both a fixed as well as a binary adaptable window using the following window sizes: $3 \times 3$, $7 \times 7$ and $11 \times 11$. Out of these cost functions, the following 72 features which are presented in section 6.2 are calculated:

- $MC, CUR, PKR, LRC, LRD$
- $CCT$, with the following parameters:
  - Multiplication: 1.2, 1.4, 1.6, 1.8, 2, 5, 10.
- $TEX$, with the following parameters:
  - Window Size: $3 \times 3, 7 \times 7, 11 \times 11, 21 \times 21$.
  - Colour Information: Luminance or Chroma.
- $SNDD$, with the following parameters:
  - Window Size: $3 \times 3, 7 \times 7, 11 \times 11, 21 \times 21$. 
• \textit{SNDDBW} and \textit{SEG}, with the following parameters:

  - Window Size: $3 \times 3, 7 \times 7, 11 \times 11, 21 \times 21$.
  - Chroma Threshold: 1, 5 or 10.
  - Distance metric: enabled or disabled.

The features are indicated in the following tables by their acronym and in subscript their parameters: e.g. \textit{SNDDBW}_{11,10,1} means \textit{Sum of Neighbouring Disparity Differences using adaptive Binary Weights (SNDDBW)} with a window size of $11 \times 11$, a Chroma threshold of 10 and the distance metric enabled. These generated features are used in the next step as input for the classification methods presented in section 6.3. In this chapter we will allow for a small deviation of one unit disparity value between the calculated and the real disparity values, so the error map will be used as classification output. The classification methods will be trained such that labelling a bad pixel good will (FP) give a twenty times higher error than labelling a good pixel bad (FN). As long as there are enough pixels left, most post processing steps prefer losing good pixels to be certain that most pixels are correct.

The results of the One Feature Threshold method can be seen in Table 6.1. It is clear that feature \textit{SNDDBW} is the best choice when using one threshold. In most cases we can remove almost 95\% of incorrect disparity values, while losing less than 50\% of correct disparity values.

The results of the Decision Tree method can be seen in Table 6.2 and table 6.3. For every test, two features are shown which are most important in the construction of the tree. In most cases we can remove almost 96\% of incorrect disparity values, while losing less than 40\% of correct disparity values. This increase is due to the extra classification ability gained by adding more features. The most important features for constructing the trees are \textit{SNDDBW}, \textit{SNDD} and \textit{TEX}. Notice that these are all neighbourhood confidence metrics.

Table 6.4 presents the results from the Artificial Neural Network (ANN) classification method. Not all features are used to construct the ANN. A subset is chosen that contains the best features from the "One Feature Threshold" classification approach and has the least amount of correlation between features. The following features are left after the feature selection step: \textit{MC}, \textit{CUR}, \textit{LRC}, \textit{CCT}_2, \textit{CCT}_{10}, \textit{TEX}_{Luminance}, \textit{SNDDBW}_{[21,11],[10,5],1}, \textit{SNDD}_{21}, \textit{SEG}_{21,5,1}. 
<table>
<thead>
<tr>
<th>Data Set</th>
<th>Feature Name</th>
<th>False Positive</th>
<th>False Negative</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 × 3 Without Support Weights</td>
<td>Sawtooth</td>
<td>3.99 ± 0.39</td>
<td>12.13 ± 0.17</td>
</tr>
<tr>
<td></td>
<td>Tsukuba</td>
<td>2.14 ± 0.51</td>
<td>48.38 ± 3.29</td>
</tr>
<tr>
<td></td>
<td>Teddy</td>
<td>1.99 ± 0.19</td>
<td>47.10 ± 0.56</td>
</tr>
<tr>
<td>3 × 3 With Binary Support Weights</td>
<td>Sawtooth</td>
<td>4.92 ± 0.36</td>
<td>8.94 ± 0.58</td>
</tr>
<tr>
<td></td>
<td>Tsukuba</td>
<td>1.98 ± 0.12</td>
<td>48.52 ± 0.79</td>
</tr>
<tr>
<td></td>
<td>Teddy</td>
<td>2.85 ± 0.15</td>
<td>43.47 ± 0.61</td>
</tr>
<tr>
<td>7 × 7 Without Support Weights</td>
<td>Sawtooth</td>
<td>7.53 ± 1.04</td>
<td>7.96 ± 0.15</td>
</tr>
<tr>
<td></td>
<td>Tsukuba</td>
<td>1.77 ± 0.20</td>
<td>40.60 ± 0.39</td>
</tr>
<tr>
<td></td>
<td>Teddy</td>
<td>1.79 ± 0.26</td>
<td>47.14 ± 0.29</td>
</tr>
<tr>
<td>7 × 7 With Binary Support Weights</td>
<td>Sawtooth</td>
<td>6.32 ± 0.89</td>
<td>6.49 ± 0.16</td>
</tr>
<tr>
<td></td>
<td>Tsukuba</td>
<td>2.17 ± 0.24</td>
<td>38.86 ± 0.32</td>
</tr>
<tr>
<td></td>
<td>Teddy</td>
<td>4.20 ± 0.38</td>
<td>25.01 ± 0.36</td>
</tr>
<tr>
<td>11 × 11 Without Support Weights</td>
<td>Sawtooth</td>
<td>6.32 ± 0.89</td>
<td>6.49 ± 0.16</td>
</tr>
<tr>
<td></td>
<td>Tsukuba</td>
<td>2.00 ± 0.13</td>
<td>40.09 ± 0.21</td>
</tr>
<tr>
<td></td>
<td>Teddy</td>
<td>5.05 ± 0.25</td>
<td>36.11 ± 0.49</td>
</tr>
<tr>
<td>11 × 11 With Binary Support Weights</td>
<td>Sawtooth</td>
<td>6.35 ± 0.53</td>
<td>8.17 ± 0.27</td>
</tr>
<tr>
<td></td>
<td>Tsukuba</td>
<td>1.96 ± 0.36</td>
<td>40.41 ± 1.49</td>
</tr>
<tr>
<td></td>
<td>Teddy</td>
<td>4.27 ± 0.42</td>
<td>26.35 ± 0.36</td>
</tr>
</tbody>
</table>

Table 6.1: One Feature Threshold
<table>
<thead>
<tr>
<th>Data Set</th>
<th>Feature Name</th>
<th>Misclassification Rate (%)</th>
<th>False Positive</th>
<th>False Negative</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 × 3 Without Support Weights</td>
<td>Sawtooth</td>
<td>$SNDDBW_{21,5,1}$</td>
<td>4.08 ± 1.64</td>
<td>9.30 ± 3.09</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$SNDDBW_{21,10,1}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Tsukuba</td>
<td>$SNDDBW_{21,10,1}$</td>
<td>1.98 ± 0.20</td>
<td>45.15 ± 1.12</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$SNDDBW_{25,5,1}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Teddy</td>
<td>$SNDD_{21}$</td>
<td>1.30 ± 0.23</td>
<td>50.26 ± 3.11</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$SNDDBW_{21,5,1}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 × 3 With Binary Support Weights</td>
<td>Sawtooth</td>
<td>$SNDDBW_{21,10,1}$</td>
<td>1.69 ± 0.24</td>
<td>12.67 ± 0.62</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$SNDDBW_{21,10,1}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Tsukuba</td>
<td>$SNDDBW_{21,10,1}$</td>
<td>1.42 ± 0.17</td>
<td>44.62 ± 0.51</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$SNDDBW_{21,5,1}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Teddy</td>
<td>$SNDDBW_{21,10,1}$</td>
<td>1.34 ± 0.70</td>
<td>53.61 ± 6.94</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$SNDD_{21}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7 × 7 Without Support Weights</td>
<td>Sawtooth</td>
<td>$SNDDBW_{21,10,1}$</td>
<td>7.45 ± 0.54</td>
<td>5.20 ± 0.37</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$SNDDBW_{21,5,1}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Tsukuba</td>
<td>$SNDDBW_{21,10,1}$</td>
<td>2.28 ± 0.22</td>
<td>28.92 ± 0.40</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$SNDDBW_{11,5,1}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Teddy</td>
<td>$SNDDBW_{21,10,1}$</td>
<td>2.39 ± 1.05</td>
<td>36.65 ± 8.08</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$SNDD_{21}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7 × 7 With Binary Support Weights</td>
<td>Sawtooth</td>
<td>$SNDDBW_{21,5,1}$</td>
<td>6.79 ± 0.49</td>
<td>3.82 ± 0.13</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$SNDDBW_{21,5,1}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Tsukuba</td>
<td>$SNDDBW_{21,10,1}$</td>
<td>2.61 ± 0.25</td>
<td>27.79 ± 0.25</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$SNDDBW_{11,5,1}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Teddy</td>
<td>$SNDDBW_{21,10,1}$</td>
<td>2.18 ± 0.47</td>
<td>32.66 ± 1.08</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$TEX_{7,Luminance}$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 6.2: Decision Tree (Part 1)
### Classification of Disparity Values

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Feature Name</th>
<th>Misclassification Rate (%)</th>
<th>False Positive</th>
<th>False Negative</th>
</tr>
</thead>
<tbody>
<tr>
<td>11 × 11 Without Support Weights</td>
<td>Sawtooth $SNDDBW_{21,10,1}$</td>
<td>7.13 ± 0.44</td>
<td>5.95 ± 0.33</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MC $SNDDBW_{21,10,1}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Tsukuba $SNDDBW_{21,10,1}$</td>
<td>2.92 ± 1.00</td>
<td>29.32 ± 6.28</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$SNDDBW_{21,5,1}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Teddy $SNDDBW_{21,10,1}$</td>
<td>3.73 ± 0.74</td>
<td>31.43 ± 2.30</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$TEX_{21,Luminance}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11 × 11 With Binary Support Weights</td>
<td>Sawtooth $SNDDBW_{21,10,1}$</td>
<td>5.57 ± 0.77</td>
<td>6.04 ± 0.52</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$SNDDBW_{21,5,1}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Tsukuba $SNDDBW_{21,10,1}$</td>
<td>3.01 ± 0.34</td>
<td>26.28 ± 0.38</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$SNDDBW_{11,10,1}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Teddy $SNDDBW_{21,10,1}$</td>
<td>2.70 ± 0.54</td>
<td>27.91 ± 2.57</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$TEX_{21,Luminance}$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 6.3: Decision Tree (Part 2)

In most cases we can remove almost 94% of incorrect disparity values, while losing less than 80% of correct disparity values. However, for most applications, this increase of classification ability for good pixels is not substantial enough to justify the extra hardware resources needed. With all classification methods, we see that increasing the windows size reduces the misclassification rate. The choice between window cost aggregation types is not that obvious from the table. Overall the scores are similar. However, when we take a look at the generated disparity maps (see Figure 6.8 and 6.9) we notice that, with a binary adaptive window small details are preserved.

![Disparity Maps](image-url)

Figure 6.8: Disparity map quality of the investigated datasets (Sawtooth, Tsukuba, Teddy) using fixed weights size of 11 × 11 (black pixels indicates a confidence of zero).
<table>
<thead>
<tr>
<th>Data Set</th>
<th>Neural Network Structure</th>
<th>False Positive</th>
<th>False Negative</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 × 3 Without Support Weights</td>
<td>Sawtooth 12 – 3 – 2 – 1</td>
<td>4.93 ± 0.29</td>
<td>4.93 ± 0.21</td>
</tr>
<tr>
<td></td>
<td>Tsukuba 12 – 7 – 1</td>
<td>3.00 ± 0.78</td>
<td>33.30 ± 5.99</td>
</tr>
<tr>
<td></td>
<td>Teddy 12 – 7 – 1</td>
<td>4.28 ± 0.63</td>
<td>27.37 ± 1.98</td>
</tr>
<tr>
<td>3 × 3 With Binary Support Weights</td>
<td>Sawtooth 12 – 7 – 1</td>
<td>5.28 ± 0.56</td>
<td>6.09 ± 0.69</td>
</tr>
<tr>
<td></td>
<td>Tsukuba 12 – 7 – 1</td>
<td>3.39 ± 0.37</td>
<td>29.13 ± 9.50</td>
</tr>
<tr>
<td></td>
<td>Teddy 12 – 3 – 2 – 1</td>
<td>4.55 ± 0.34</td>
<td>27.21 ± 1.36</td>
</tr>
<tr>
<td>7 × 7 Without Support Weights</td>
<td>Sawtooth 12 – 3 – 2 – 1</td>
<td>6.33 ± 0.83</td>
<td>2.48 ± 0.79</td>
</tr>
<tr>
<td></td>
<td>Tsukuba 12 – 3 – 1</td>
<td>5.27 ± 0.34</td>
<td>22.29 ± 0.24</td>
</tr>
<tr>
<td></td>
<td>Teddy 12 – 7 – 1</td>
<td>6.98 ± 0.60</td>
<td>15.68 ± 0.61</td>
</tr>
<tr>
<td>7 × 7 With Binary Support Weights</td>
<td>Sawtooth 12 – 3 – 2 – 1</td>
<td>9.05 ± 2.91</td>
<td>2.53 ± 0.10</td>
</tr>
<tr>
<td></td>
<td>Tsukuba 12 – 7 – 1</td>
<td>5.75 ± 0.78</td>
<td>19.02 ± 4.63</td>
</tr>
<tr>
<td></td>
<td>Teddy 12 – 7 – 1</td>
<td>6.64 ± 0.72</td>
<td>15.01 ± 0.41</td>
</tr>
<tr>
<td>11 × 11 Without Support Weights</td>
<td>Sawtooth 12 – 3 – 1</td>
<td>8.38 ± 0.40</td>
<td>2.70 ± 0.30</td>
</tr>
<tr>
<td></td>
<td>Tsukuba 12 – 3 – 1</td>
<td>6.91 ± 0.41</td>
<td>19.81 ± 0.09</td>
</tr>
<tr>
<td></td>
<td>Teddy 12 – 3 – 2 – 1</td>
<td>6.72 ± 0.12</td>
<td>15.42 ± 0.45</td>
</tr>
<tr>
<td>11 × 11 With Binary Support Weights</td>
<td>Sawtooth 12 – 3 – 2 – 1</td>
<td>7.55 ± 1.49</td>
<td>1.61 ± 0.01</td>
</tr>
<tr>
<td></td>
<td>Tsukuba 12 – 3 – 1</td>
<td>7.44 ± 0.13</td>
<td>16.26 ± 7.01</td>
</tr>
<tr>
<td></td>
<td>Teddy 12 – 3 – 2 – 1</td>
<td>7.77 ± 1.43</td>
<td>13.94 ± 0.59</td>
</tr>
</tbody>
</table>

Table 6.4: Artificial Neural Network

Figure 6.9: Disparity map quality of the investigated datasets (Sawtooth, Tsukuba, Teddy) using adaptable binary weights size of 11 × 11 (black pixels indicates a confidence of zero).
6.7 Conclusion

A methodology to generate and test binary confidence classifiers has been presented. Such binary confidence classifiers are needed for a higher quality stereo disparity image generation in hardware. We compare 72 different features and three different classifiers. After comparison of the misclassification rate on three datasets and minimization of the hardware resources. We come to the conclusion that neighbouring confidence metrics in combination with a small decision tree will lead to the best binary confidence map. The features and classifiers presented in this chapter are easily implemented in hardware and can be added to most stereo vision architectures without committing many resources. The generated binary confidence map can be used to aid post processing tasks like path planning or steps like disparity refinement.
# Architecture Generation Framework

## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1 Introduction</td>
<td>89</td>
</tr>
<tr>
<td>6.2 Confidence Metrics</td>
<td>90</td>
</tr>
<tr>
<td>6.2.1 Confidence Metrics Based on the Cost Function</td>
<td>91</td>
</tr>
<tr>
<td>6.2.2 Consistency Between Left and Right Disparity Maps</td>
<td>92</td>
</tr>
<tr>
<td>6.2.3 Confidence Metrics Based on Neighbourhood Information</td>
<td>92</td>
</tr>
<tr>
<td>6.2.4 Confidence Metrics Based on Multiple Pixel Streams</td>
<td>94</td>
</tr>
<tr>
<td>6.3 Investigated Classification Methods</td>
<td>94</td>
</tr>
<tr>
<td>6.3.1 One Feature Threshold</td>
<td>95</td>
</tr>
<tr>
<td>6.3.2 Decision Tree</td>
<td>95</td>
</tr>
<tr>
<td>6.3.3 Artificial Neural Network</td>
<td>96</td>
</tr>
<tr>
<td>6.4 Hierarchical Classification</td>
<td>97</td>
</tr>
<tr>
<td>6.5 Zoning</td>
<td>97</td>
</tr>
<tr>
<td>6.6 Classification of Disparity Values</td>
<td>98</td>
</tr>
<tr>
<td>6.7 Conclusion</td>
<td>105</td>
</tr>
</tbody>
</table>

## 7.1 Introduction

To enable the flexible design and validation of dedicated computational video applications, a multi-camera architecture generation framework has been realized as part
of this thesis.

In this chapter, the methods used to generate and validate a multi camera vision architecture are discussed.

The modules presented in the previous section allows the adaptation of several parameters to create custom System-on-Chip (SoC) implementations, as well as rapid prototyping on Field Programmable Gate Array (FPGA)’s. Designs with a large support region are difficult to construct manually. Rewriting the architecture for different window sizes is a long and error prone process. However, this process can be automated using a parameterized design.

The second issue is the time it takes before a specific hardware architecture can be completely tested. This long period does not allow easy comparison between different architectural decisions.

7.2 Framework Overview

The proposed solution to both issues is to auto generate the architecture for both simulation and hardware generation from a high level description (see Figure 7.1). This allows an initial check of the stereo matching architecture in Matlab before implementation on the actual hardware. Using this framework, comparisons between different stereo matching implementations can be rapidly performed.

A multi camera architecture generator has been realized in Matlab. Out of the chosen parameters and the high level architecture, a complete multi camera matching architecture is generated. It performs this task by generating a target specific implementation block for each block in the architecture. These blocks are globally similar, but are different in the primitives they are using. E.g. the Matlab primitive for absolute difference is \( \text{abs}(p1 - p2) \) while a specific Verilog module is constructed for this calculation. In this framework, the target specific implementation blocks are defined in templates, which define the location of the target specific implementation blocks in the data flow. There are templates available for Matlab and Verilog. Both templates contain information on how to get pixel information and how to show the disparity information to the user. The difference between both templates is how and when those target specific implementation blocks are called. In case of the Matlab template, the blocks are called sequentially, while in the case of the Verilog template, the blocks can be pipelined. It is important to note that the templates are independent of the
parameters.

All modules discussed in previous chapters have similar inputs and outputs. This makes it feasible to modularly construct a complete system out of these building blocks.

The functional specification consists out of two parts. First, each module in the data flow will be configured independently. Second, the data flow is defined by naming the input and output bus on each module.

![Diagram](image)

Figure 7.1: Framework for the development of multi-camera vision systems.

After behavioural simulation in Matlab, the next step is to generate and simulate the Hardware Description Language (HDL) implementation. At this time, it is possible to perform verification between the Matlab and the HDL implementation (Figure 7.2). Often problems with synchronization are discovered at this step.

Last the hardware implementation can be generated. The images resulting from the hardware setup can be stored in memory (Figure 7.3). The images in this memory can be read out and compared with the Matlab implementation. Timing problems are the main focus of this verification step.
Figure 7.2: Testbench for behavioural simulation (left: ModelSim HDL simulation, right: Matlab script execution).

Figure 7.3: Testbench for hardware verification, gray area indicates the SUT.
7.3 Example Design Flow

One design flow, containing two main modules, will be briefly explained in this section. First, a module is generated for each image sensor individually. Second, a module is generated which compares the different pixel streams.

For all camera systems some basic pre-processing steps are necessary, like e.g. demosaicing. In the case of a multi-camera vision system, the correct alignment between the image sensors is necessary. First multiple images of a chequerboard pattern are taken with the multi-camera system. These are then transferred to the framework in order to find the lens distortions and the camera alignments. The intrinsic and extrinsic parameters of the cameras individually, and the transformation matrix of the stereo setup are calculated as proposed by Zhang [1999]. This information is later used by the toolbox to generate a memory efficient image warping module (see Chapter 3).

For each image sensor a module is generated which contains all the pre-processing steps. Figure 7.4 illustrates the input and outputs of such a module. Most modules generated using the framework will adhere to this convention. This allows for easy insertion of modules in existing structures.

Figure 7.4: Example instance pre-processing module.

The next step is to select which method is going to be used for image matching. The first option is to select a memory architecture, like line buffer implementation or the variants of the tiled memory architecture (see Chapter 4). Several options also exist with the window comparison module (see Chapter 5), Truncated sum of Absolute Differences (TAD) or Sum of Absolute Differences (SAD) with or without Adaptive
Binary Weights (ABW). If ABW is selected, the threshold needs to be chosen and connectivity preservation can be selected. All these options are available using the framework, when selection is done, all modules are generated. Figure 7.5 illustrates the inputs and outputs of a comparison module.

![Diagram of a Window Comparison Module](image)

**Figure 7.5: Example instance window comparison module.**

When the modules are generated, four main files are constructed. First off, the Verilog instance file is generated. Then, its testbench is constructed. Afterwards, a Matlab implementation is generated suitable for the cross-verification of the correct implementation of the module. And last, a random set of input vectors is generated for module testing.

### 7.4 Synchronisation

Synchronising different data streams in a pipelined environment is a major issue. Figure 7.6 illustrates a pipelined tree adder where one branch has a missing delay element. This will give a completely different result than anticipated. Omitting a single register in a module is possible, but can easily be located during testing.

However, when just naively connecting modules like in Figure 7.7, the same prob-
lem occurs and that is not easily located. The two windows at the input of the window comparison are not in relation with each other. Additional delay buffers are needed in the luminance stream such that both inputs are aligned. The question arises where the delay should be implemented: before or after the window comparison module. This will depend on the bitsize of the data streams. This is not always clear from looking at the implementation, already a good understanding of the different modules and interconnection are needed to optimize the insertion of buffers.

The number of pipelining steps for each module are depending on its configuration parameters. For example when increasing the window size, the number of pipelining steps will also be increased. For basic systems, these delays can be easily managed without a specific tool. However when using more complex systems, it can be time consuming to define all buffers manually. A tool could be useful to provide guidelines on restructuring the delays, for example using techniques from just in time manage-
Figure 7.8: Synchronised window comparison.

7.5 Conclusion

The methods described in this chapter have been used to accelerate the development of several stereo vision applications (see Chapter 8). Both initial development and verifications proved to be indispensable. Further work is needed to automate the insertions of delay buffers for synchronising multiple pixel streams.
Chapter 8

Real World Feasibility and Evaluation

 Contents

 7.1 Introduction .................................................. 107
 7.2 Framework Overview ....................................... 108
 7.3 Example Design Flow ....................................... 111
 7.4 Synchronisation ............................................. 112
 7.5 Conclusion .................................................. 114

8.1 Introduction

In the past decade, several stereo vision hardware implementations have been proposed. Most of these stereo vision implementations on a custom System-on-Chip (SoC) or an Field Programmable Gate Array (FPGA) uses of a fixed cost aggregation window (Lee et al. [2005], Yi et al. [2004], Murphy et al. [2007], Mingxiang and Yunde [2006]) or an adaptable rectangular cost aggregation window (Hariyama and Kameyama [2001]).

In recent years more advanced stereo implementations have been developed for hardware implementations. A real-time FPGA-based stereo vision system is presented
by Jin et al. [2010] that uses the census transform. Their system includes all the pre-
and post-processing functions such as: rectification, LR-check and uniqueness test
in a single FPGA. Ding et al. [2011] proposed a hardware implementation using the
Adaptive Weights (AW) algorithm, however it is decomposed in a horizontal and
vertical part.

When adding an extra camera to a binocular stereo system, a trinocular stereo
system is obtained. An improvement of occlusion handling in trinocular vision com-
pared to stereo vision is achieved by Mozerov et al. [2009]. The main idea is based
on the assumption that any occluded region in a matched stereo pair (centre-left im-
ages) in general is not occluded in the opposite matched pair (centre-right images).
They use a global optimization technique to derive the composite Disparity Space
Image (DSI). Bidirectional matching using trinocular stereo is used by Ueshiba [2006]
and by Mulligan and Daniilidis [2001] to detect half-occlusions and to discard false
matches. It uses a cumulative cost function derived from a summation of both cost
curves. Hardware implementations of a trinocular disparity processor are limited.
An implementation using the summation of Sum of Absolute Differences (SAD) from
both image pairs can be found in Mingxiang and Yunde [2006].

In this chapter, two depth vision implementations are presented. In contrast with
other implementations, both stereo vision applications use full adaptive binary weights
for aggregating the matching costs (see Chapter 5).

- The first application divides the stereo vision problem into two parts: first a
  rough disparity map is constructed using a local window based stereo matching
  architecture, second a disparity refinement module identifies false matches and
  replaces them with new estimates.

- The second application extends the first application by using more image sen-
sors. It generates several disparity images from pairwise matching the pixel
streams from the different image sensors. Comparable with the first applica-
tion, the disparity images are generated using a local window based stereo
matching architecture. For each disparity stream a confidence value is gener-
ated to identify false matches. A multi-level classification structure is used to
select the most promising disparity result.

One of the goals of this architecture is to limit the number of external components.
The architecture has been constructed to reduce memory usage. Hence there is no
need for external memories. This allows for a true SoC where only one chip is needed to provide the proposed functionalities. The reduction of external memory usage has the additional advantage that the latency between input frame and output frame becomes minimal. This makes this system suitable to be incorporated in real-time control loops.

8.2 Binocular Stereo Camera

The stereo matching algorithm takes two undistorted and rectified images that have been taken by two cameras that have a vertical alignment and a horizontal offset (Figure 8.1). Objects will appear on both images on the same horizontal line (the epipolar line). The horizontal distance between the same objects on the left and right images is called the disparity. Objects that are close to the cameras will have a larger horizontal disparity than objects that are far away. The goal of the stereo matching algorithm is to measure the disparity between all pixels in the left and the right image.

![Figure 8.1: Binocular stereo setup.](image-url)
8.2.1 System Design

Figure 8.2 illustrates the global architecture. The pixel streams provided by the image sensors first pass a pre-processing module (Chapter 2 and 3). Next, the two input streams are compared using an Adaptive Binary Weights (ABW) based SAD Winner Takes All (WTA) algorithm (Chapter 5), which outputs the disparity value. Lastly, this disparity value is passed into the post-processing module where a binary confidence metric is calculated (Chapter 6), which indicates mismatched pixels and calculates new disparity values from the neighbouring disparity values.

![Figure 8.2: Binocular stereo, global architecture.](image)

8.2.1.1 Pre-Processing Module

The pre-processing module consists of three different entities: first a median filter is used to remove defected pixels of the camera. Next a demosaicing algorithm is used to reconstruct the colour image and lastly a rectification module is used to remove lens distortion and perform stereo calibration (Figure 8.3).

![Figure 8.3: Binocular stereo, pre-processing module.](image)

Two different kinds of distortions are present in a stereo camera setup. The first one are the lens distortions, the second one is a misalignment of the two cameras (Figure 8.4). Since the search space is only located on the epipolar line, both distortions...
should be resolved before the matching can be performed.

Figure 8.4: Binocular stereo, misalignment (grey lines are the mismatched epipolar lines)

In order to remove these distortions, multiple images of a checkerboard pattern are taken using the camera setup and imported into Matlab. The intrinsic and extrinsic parameters of the cameras individually, and the transformation matrix of the stereo setup are calculated like proposed by Zhang [1999]. These parameters are then used to construct the x and y mapping coordinates for each pixel in the image.

8.2.1.2 Window Comparison Module

The two input streams of the cameras are compared with each other using an adaptive weights based SAD calculation (Figure 8.5). During every clock cycle a window of the right camera is compared with four windows of the left camera. Since four successive pixels are stored in one memory location, one memory read accesses four pixels, hence four comparison modules are running in parallel.

The clock frequency of the comparison module directly controls the possible disparity range of the stereo matching architecture and can be adapted to the available resources. In the example on Figure 8.5 a frequency of 180 MHz is chosen, which touches the design limit for implementation in a Cyclone II FPGA. In this application, the disparity module runs twenty times faster than a memory write. This allows a disparity range of eighty when the comparison module has a disparity of four.

The disparity range of a single comparison module is limited to a small disparity range. It can be increased to accommodate a larger disparity range if the maximum operating frequency is reached. On each clock cycle (indicated by CC in Figure 8.6), the comparison module compares the reference window with four other windows.
Figure 8.5: Binocular stereo, window comparison module.
The lowest SAD score and its corresponding index are saved in a register, so that on the next clock cycle this lowest SAD score can be compared against the SAD scores of the next four windows.

![Diagram of binocular stereo camera]

Figure 8.6: Binocular stereo, window comparison of different data streams.

### 8.2.1.3 Post-Processing Module

The post-processing module consists of two separate entities: a decision tree based confidence metric module and a disparity refinement module (Figure 8.7). Note that the post-processing module can be run on a higher frequency compared to the input stream. This allows for an iterative refinement of the disparity map. A selection of different confidence metrics have been combined to train a Decision Tree (DT) confidence metric as described in Chapter 6.

The window size and the chroma threshold is chosen to be the same as in the cost aggregation phase such that the calculated mask window (5.20) (5.21), which is available from the cost aggregation module, can be reused. After training and pruning of the DT, the results can be seen in Figure 8.8.

For every disparity value that is labelled as unreliable, the disparity refinement step calculates a new value. This new disparity value is calculated from the mean of
Figure 8.7: Binocular stereo, post-processing module.

Figure 8.8: Binocular stereo, confidence calculation.
the disparity values of all neighbouring pixels, which have a similar colour (8.1).

\[
D(p) = \begin{cases} 
  D(p), & \text{if } \text{Confidence} == 1 \\
  \text{mean}_{q \in N_p}(D(q)), & \text{otherwise}
\end{cases}
\]  

(8.1)

The selected neighbouring pixels are chosen to be the same as in the cost aggregation phase such that the ABW needed to select similar colour pixels does not need to be recalculated.

8.2.2 Implementation

Out of the chosen parameters and the high level architecture, a complete stereo matching architecture is generated, using our framework, for both simulation and hardware generation. This allows an initial check of the stereo matching architecture in Matlab before implementation on the actual hardware. Using this framework, comparison between different stereo matching parameters and architectures can be rapidly performed. The architecture and methods presented in this section have been implemented on an FPGA system, based on an Altera Cyclone II with 68,416 logic elements and 250 memory blocks. The sources of the input streams are two cameras with a resolution of \(800 \times 600\) and a frame rate of 56 Hz. The current implementation consists of a \(7 \times 7\) ABW based SAD with a disparity range of 80, a DT confidence metric and a disparity refinement module. The hardware block diagram is depicted in Figure 8.9. The synthesis results can be found in Table 8.1.

Figure 8.10 shows the disparity maps of the Tsukuba stereo pair (Scharstein and Szeliski [2002]) without confidence measurement compared with the DT confidence metric and after disparity refinements. The results indicate that the quality of the resulting disparity map increases when using the disparity refinement steps. It can also be seen that disparity mismatches not detected by the binary confidence DT metric will enlarge during the disparity refinement steps. This is because the disparity refinement module is not robust against larger groups of mismatches or against mismatches isolated from correct matches.

8.2.3 Conclusion

In this section a real-time stereo vision SoC architecture for a depth-field generation processor is presented. It allows for a reduction of memory and logic resource require-
Figure 8.9: Binocular stereo, hardware block diagram.
Figure 8.10: Binocular stereo, disparity map quality of the Tsukuba Stereo Pair for a $7 \times 7$ binary adaptive window: without confidence metric (top-left), with DT confidence metric (top-right), one iteration of disparity refinement (bottom-left), ten iterations of disparity refinement (bottom-right) (black pixels indicate a confidence of zero).
Table 8.1: Trinocular stereo, synthesis results overview.

<table>
<thead>
<tr>
<th>Module Name</th>
<th>Logic Elements</th>
<th>Memory Blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Nr.</td>
<td>Single</td>
</tr>
<tr>
<td>Pre-Processing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Median Filter</td>
<td>2</td>
<td>613</td>
</tr>
<tr>
<td>Demosaicing</td>
<td>2</td>
<td>176</td>
</tr>
<tr>
<td>Rectification</td>
<td>2</td>
<td>840</td>
</tr>
<tr>
<td>Window Comparison</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory Left Y</td>
<td>1</td>
<td>3,344</td>
</tr>
<tr>
<td>Memory Right Y</td>
<td>1</td>
<td>825</td>
</tr>
<tr>
<td>Memory Right C</td>
<td>1</td>
<td>825</td>
</tr>
<tr>
<td>Adaptable Support Region</td>
<td>1</td>
<td>2,641</td>
</tr>
<tr>
<td>Comparison Module</td>
<td>1</td>
<td>27,312</td>
</tr>
<tr>
<td>Post-Processing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parallel Memory</td>
<td>1</td>
<td>825</td>
</tr>
<tr>
<td>DT Confidence Metric</td>
<td>1</td>
<td>1,788</td>
</tr>
<tr>
<td>Disparity Refinement</td>
<td>1</td>
<td>1,216</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>42,034</td>
</tr>
</tbody>
</table>

The dense stereo problem is divided into two parts: First a rough disparity map is constructed using a segmentation based SAD window comparison. The pixel streams originating from the images sensors pass a pre-processing module consisting of a median filter, demosaicing unit and a rectification unit. Next, the two input streams are compared using a ABW based SAD WTA algorithm which outputs the disparity value. Second a disparity refinement module identifies false matches using a binary confidence metric and replaces them with new estimates derived from the neighbouring disparity values. It is shown that addition of the post-processing module increases significantly the quality of the disparity map.

8.3 Trinocular Stereo Camera

Trinocular vision uses three cameras to calculate a combined DSI. Each DSI is generated by pairwise matching the images from the different cameras which is based on a local window based stereo matching architecture.

The method presented in this section uses a hierarchical classifier to select the most
likely disparity for each pixel in the final DSI. This section combines the strengths of an advanced stereo vision system with a two-scale adaptive window SAD incorporated in a trinocular setup.

8.3.1 General Architecture

The trinocular disparity processor takes three images that have been taken by three cameras that have a vertical alignment and a horizontal offset (see Figure 8.11). The objective is to calculate a DSI where dark pixels represent a distance further away from the cameras and a light pixel represents a distance closer by the camera. Objects appear on the same horizontal line (called the epipolar line) on all images. The horizontal distance between the same objects on the centre image and the left (or right) image is called the disparity. If calibrated correctly, the disparity of an object between the centre-left and the centre-right image pair is the same. This characteristic can be used to discard false matches using bidirectional matching (Ueshiba [2006]) or to improve the quality of the DSI especially in occluded regions (Mozerov et al. [2009]).

![Figure 8.11: Trinocular stereo setup.](image)

The architecture consists of three main blocks (see Figure 8.12), pre-processing for each image sensor, window comparison for each camera pair and last hierarchical classification. The first block captures the pixel streams, generates the scaled images
and places them in multiple on-chip parallel memories (see Chapter 4). The second block performs a pairwise window comparison of the different streams using a binary adaptable SAD cost aggregation (see Chapter 5). The third block calculates its confidence for each data stream and selects the final disparity value (see Chapter 6).

In this architecture nine different cost curves are calculated for each pixel in the DSI (see Figure 8.13). The first step is to calculate the cost curve the centre-right ($SAD_{CR1}$) and centre-left ($SAD_{CL1}$) image pairs as their one-quarter sized scaled down image pairs ($SAD_{CR0}$ and $SAD_{CL0}$). The second step is to calculate the summation of these cost curves (8.2). The summation of cost curves is inspired by Mingxiang and Yunde [2006], they used this approach to combine the SAD functions of the centre-left and centre-right camera pairs.

\[
\begin{align*}
SAD_{CL1}, SAD_{CR1}, SAD_{CL0}, SAD_{CR0} \\
SAD_{CLR1} & = SAD_{CL1} + SAD_{CR1} \\
SAD_{CLR0} & = SAD_{CL0} + SAD_{CR0} \\
SAD_{CL01} & = SAD_{CL1} + SAD_{CL0} \\
SAD_{CR01} & = SAD_{CR1} + SAD_{CR0} \\
SAD_{CLR01} & = SAD_{CLR1} + SAD_{CLR0}
\end{align*}
\]
8.3.2 Hierarchical Classification

In the previous section it is explained that nine disparity values are generated for each pixel (8.2). In order to select one of them for generating the DSI, a two level hierarchical classifier is constructed (see Figure 8.14). In the first level of the hierarchy, the disparity values are investigated independently of each other. For each disparity value a binary confidence classifier is constructed using the methods presented in Chapter 6. These confidences are passed on to the second level classifier which selects the disparity to use, or indicates that no disparity has been found.

For each level of the hierarchy, a different set of features is needed for classification.
The first level of classifiers uses information obtained from the pixel neighbourhood and from its corresponding cost curve. The second level classifier uses the generated binary confidence values together with the agreement between the different disparity values. A binary confidence value of 1 indicates a strong confidence in the correctness of the disparity value. A binary confidence value of 0 indicates a weak confidence in the correctness of the disparity value.

Three different datasets have been used to verify the results:

- Tsukuba (Scharstein and Szeliski [2002]): 384 x 288 (Maximal disparity of 30).
- Teddy (Scharstein and Szeliski [2003]): 450 x 375 (Maximal disparity of 30).
- Art (Hirschmuller and Scharstein [2007]): 695 x 555 (Maximal disparity of 30).

The first level classifier consists of a DT for each disparity stream individually. The second level classifier selects the final disparity from the different disparity streams by choosing the one with the lowest *Sum of Streaming Disparity Differences (SSDD)*. Both classification methods are easily implemented in hardware without using many resources.

### 8.3.2.1 First Level Classification Evaluation

Matlab has been used to generate and classify the different features as explained in Chapter 6. For each dataset, cost curves have been calculated using the SAD aggregation method with adaptable binary support weights for a window size of $7 \times 7$.

The results of the first level classification are listed in table 8.2. For every test, the feature is shown which is the most important in the construction of the tree. This table illustrates that the error rate between the different disparity streams is different. By summation of the SADs a more correct DSI is constructed.

Figure 8.15 illustrates that the different DSIs have different areas which are indicated as correct. The centre-left image comparison ($CL_1$) provides a good result across borders where the area on the left side of the border is further away than the area on the right side of the border. The centre-right image ($CR_1$) provides good results when the border is reversed. A summation of both SADs ($CLR_1$) gives a lower global error rate, but the borders are less clear.

As expected, the scaled down image pair comparisons ($CL_0, CR_0, CLR_0$) provides better results on large texture-less areas compared with the normal size image pair.
Table 8.2: Trinocular stereo, first level classification

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Feature Name</th>
<th>Error Rate</th>
<th>Misclassification Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>DSI</td>
<td>Binary Classifier</td>
</tr>
<tr>
<td>CL₀</td>
<td>Tsukuba</td>
<td>27.30%</td>
<td>17.13%</td>
</tr>
<tr>
<td></td>
<td>Teddy</td>
<td>13.35%</td>
<td>5.35%</td>
</tr>
<tr>
<td></td>
<td>Art</td>
<td>19.65%</td>
<td>10.60%</td>
</tr>
<tr>
<td>CR₀</td>
<td>Tsukuba</td>
<td>29.46%</td>
<td>21.15%</td>
</tr>
<tr>
<td></td>
<td>Teddy</td>
<td>18.26%</td>
<td>6.12%</td>
</tr>
<tr>
<td></td>
<td>Art</td>
<td>19.27%</td>
<td>10.64%</td>
</tr>
<tr>
<td>CL₁</td>
<td>Tsukuba</td>
<td>22.98%</td>
<td>11.51%</td>
</tr>
<tr>
<td></td>
<td>Teddy</td>
<td>14.53%</td>
<td>13.39%</td>
</tr>
<tr>
<td></td>
<td>Art</td>
<td>23.74%</td>
<td>18.12%</td>
</tr>
<tr>
<td>CR₁</td>
<td>Tsukuba</td>
<td>25.05%</td>
<td>11.88%</td>
</tr>
<tr>
<td></td>
<td>Teddy</td>
<td>23.55%</td>
<td>16.25%</td>
</tr>
<tr>
<td></td>
<td>Art</td>
<td>24.45%</td>
<td>19.53%</td>
</tr>
<tr>
<td>CL₀₁</td>
<td>Tsukuba</td>
<td>22.92%</td>
<td>14.77%</td>
</tr>
<tr>
<td></td>
<td>Teddy</td>
<td>11.60%</td>
<td>6.05%</td>
</tr>
<tr>
<td></td>
<td>Art</td>
<td>19.11%</td>
<td>10.21%</td>
</tr>
<tr>
<td>CR₀₁</td>
<td>Tsukuba</td>
<td>24.83%</td>
<td>18.53%</td>
</tr>
<tr>
<td></td>
<td>Teddy</td>
<td>17.06%</td>
<td>6.83%</td>
</tr>
<tr>
<td></td>
<td>Art</td>
<td>18.99%</td>
<td>11.50%</td>
</tr>
<tr>
<td>CLR₀</td>
<td>Tsukuba</td>
<td>25.19%</td>
<td>20.06%</td>
</tr>
<tr>
<td></td>
<td>Teddy</td>
<td>14.02%</td>
<td>7.47%</td>
</tr>
<tr>
<td></td>
<td>Art</td>
<td>16.30%</td>
<td>10.13%</td>
</tr>
<tr>
<td>CLR₁</td>
<td>Tsukuba</td>
<td>22.40%</td>
<td>11.85%</td>
</tr>
<tr>
<td></td>
<td>Teddy</td>
<td>18.15%</td>
<td>21.59%</td>
</tr>
<tr>
<td></td>
<td>Art</td>
<td>19.12%</td>
<td>14.06%</td>
</tr>
<tr>
<td>CLR₀₁</td>
<td>Tsukuba</td>
<td>22.52%</td>
<td>16.67%</td>
</tr>
<tr>
<td></td>
<td>Teddy</td>
<td>14.17%</td>
<td>8.10%</td>
</tr>
<tr>
<td></td>
<td>Art</td>
<td>16.73%</td>
<td>10.45%</td>
</tr>
</tbody>
</table>
Figure 8.15: Trinocular stereo, disparity map quality of the Tsukuba dataset for a fixed window size of $7 \times 7$ after the first level of classification (black pixels indicate a confidence value of zero).
comparisons \((CL_1, CR_1, CLR_1)\). This is particular true for the background. However they have problems finding the correct disparity value for small objects, like the bars on the lamp. The summation of the SADs generated by the normal size and the scaled downs image pairs \((CL_{01}, CR_{01})\) gives a lower global error rate, keeps small details and has a better result with texture-less areas.

This indicates that by combining the DSIs, a higher quality DSI can be obtained. However before combining them, it is useful to know which part of each individual DSI is correct. A binary classifier is constructed to provide a confidence value for each DSI. The more correct this classifier, the more correct the combined DSI will be. Depending on the dataset, a misclassification rate between 5\% and 20\% is obtained. This could be improved by using a more discriminative classifier like an artificial neural network (see Chapter 6).

### 8.3.2.2 Second Level Classification Evaluation

The goal of this classification level is to select the most promising disparity value (8.3). The input of this classification level is the SSDD for each disparity stream. The output from this classification level is a disparity selection. A confidence value is afterwards generated using the same method as for each individual stream although using Sum of Streaming Confidence metrics (SSC) as an extra input feature.

\[
\text{Disparity} = \min_{i \in S} SSDD(i) \quad (8.3)
\]

An exhaustive search is performed in order to know which combination of streams provides the highest disparity improvement. A selected set of results can be seen in Table 8.3. The results indicate that, by combining extra streams, the classification rate for all investigated datasets is improved. Figure 8.16 illustrates that the addition of extra streams improves the quality of the DSI. The trinocular setup improves the DSI most noticeably at occluded regions. The scaled image improves the disparity map at parts with little texture.

### 8.3.3 System Design

The hardware architecture consists of three main modules. First a filter and sub sampling module has been added to the pre-processing module of the previous section so that a scaled image is generated with one-fourth the size of the original image.
Table 8.3: Trinocular stereo, first level classification

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Error Rate</th>
<th>Misclassification Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>$CL_0, CR_1, CL_{01}, CR_{01}, CLR_1, CLR_0$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tsukuba</td>
<td>16.50%</td>
<td>12.90%</td>
</tr>
<tr>
<td>Teddy</td>
<td>7.82%</td>
<td>7.18%</td>
</tr>
<tr>
<td>Art</td>
<td>11.86%</td>
<td>10.09%</td>
</tr>
<tr>
<td>$CL_1, CR_1, CL_0, CR_0$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tsukuba</td>
<td>16.52%</td>
<td>12.47%</td>
</tr>
<tr>
<td>Teddy</td>
<td>7.74%</td>
<td>6.71%</td>
</tr>
<tr>
<td>Art</td>
<td>12.43%</td>
<td>11.50%</td>
</tr>
<tr>
<td>$CL_1, CR_1, CL_{01}, CR_{01}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tsukuba</td>
<td>16.85%</td>
<td>12.57%</td>
</tr>
<tr>
<td>Teddy</td>
<td>8.05%</td>
<td>7.23%</td>
</tr>
<tr>
<td>Art</td>
<td>16.09%</td>
<td>11.38%</td>
</tr>
<tr>
<td>$CL_{01}, CR_{01}, CLR_1, CLR_0$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tsukuba</td>
<td>17.25%</td>
<td>13.22%</td>
</tr>
<tr>
<td>Teddy</td>
<td>7.58%</td>
<td>6.47%</td>
</tr>
<tr>
<td>Art</td>
<td>11.82%</td>
<td>10.99%</td>
</tr>
</tbody>
</table>

Figure 8.16: Trinocular stereo, disparity map quality of the investigated datasets (Tsukuba, Teddy, Art). Comparison of DSI generated from $CL_0$ data stream (Top row) and DSI generated from the combination of $CL_0$, $CL_1$, $CR_0$ and $CR_1$ data streams (Bottom row).
Second the window matching module is modified from previous section to allow for multiple data stream matching. Third a hierarchic classification module is constructed to select the most promising disparity from the different disparity results.

### 8.3.3.1 Pre-Processing Module

The pre-processing module (see Figure 8.17) consists of four different entities for each pixel stream: first a Bayer demosaicing algorithm is used to reconstruct the colour image, next a rectification module is used to remove lens distortion and perform trinocular calibration and lastly the image is filtered and down sampled to generate a scaled image.

![Figure 8.17: Trinocular stereo, pre-processing module.](image)

### 8.3.3.2 Window Comparison Module

The pixel streams originating from the right and left cameras are compared with the centre camera using an adaptive weights based SAD calculation (see Figure 8.18). During every clock cycle a window of the centre camera is compared with four windows of the left or right camera. Since four successive pixels are stored in one memory location, one memory read accesses four pixels; hence four comparison modules are running in parallel.

On every clock cycle, the Stream Selection Unit (SSU) determines where each data stream is written to and which windows are compared. The frequency of the window matching module directly controls the possible disparity search width of the trinocular matching architecture and can be adapted to the available resources. The
Figure 8.18: Trinocular stereo, window comparison module.
higher the frequency difference between the pixel stream and the window matching module, the more comparisons can be executed. In the example on Figure 8.19 the window matching module is clocked twenty-four times faster than the pixel streams.

Figure 8.19: Trinocular stereo, window comparison of different data streams.

On each clock cycle, the comparison module compares the reference window with four consecutive windows (see Figure 8.19). The lowest SAD score and its corresponding index are saved in a register, so that on the next clock cycle this lowest SAD score can be compared against the SAD scores of the next four windows. When the end of a search window is reached, the index indicates the disparity result and a new search window is initiated. In our example, in the first eight clock cycles, the centre image is compared with the right image. In the next eight clock cycles the centre image is compared with the left image. In the following four clock cycles the scaled centre image is compared with the scaled right image and in the last four clock cycles the scaled centre image is compared with the scaled left image. This leads to a combined disparity search width of thirty-two. This architecture makes it possible to easily change the disparity search width and comparison data streams for each pixel in the DSI. By adapting the SSU it is possible to switch between a trinocular and a stereo disparity search. The trade-off is the disparity range; on each clock cycle, four comparisons can be performed. When using only two cameras, all clock cycles can be used for this camera pair. While with three cameras, only half of the clock cycles remain for each camera pair. This will lead to a reduction of the disparity search width.
8.3.3.3 Hierarchical Classification Module

The hierarchical classification module consists out of the generation of the features used during the classification phase and the two classification steps. The first level classifier calculates the confidence of each stream in the selection (8.4, 8.5). For each stream, different thresholds are selected. However, the main structure of the classifier remains the same. The second level classifier selects the most promising disparity stream for the final DSI (8.6, 8.7).

\[ S = \{ CL_0, CL_1, CR_0, CR_1 \} \]  \hspace{1cm} (8.4)

\[
\text{confidence}_x = \begin{cases} 
0, & \text{if } SNDDBW_{21,10} > a \\
1, & \text{if } TEX > b \\
0, & \text{otherwise} 
\end{cases} , x \in S \]  \hspace{1cm} (8.5)

\[
SSDD = \sum_{x \in S} \text{confidence}_x \cdot |D_1 - D_1(x)| \]  \hspace{1cm} (8.6)

\[
\begin{align*}
\text{Disparity Selection} &= \arg \min_{x \in S} SSDD_y \\
\text{Disparity} &= S(\text{Disparity Selection})
\end{align*} \]  \hspace{1cm} (8.7)

8.3.4 Implementation

The architecture and methods presented in this section have been implemented on an FPGA system (Terasic DE2-115 development board), based on an Altera Cyclone IV (EP4CE115F29C7N) with 114,480 logic elements and 432 memory blocks. The sources of the input streams are three cameras with a resolution of 640 × 480 and a pixel clock of 16 MHz resulting in a frame rate of 52 Hz. The current implementation consists of the proposed design using a 7×7 ABW based SAD with a window matching clock of 96 MHz. The hardware block diagram is depicted in Figure 8.20. The architecture has been conceived in order to reduce on-chip and off-chip memory usage. Hence there is no need for external memories. The reduction of external memory usage has the additional advantage that the latency between input frame and output frame becomes minimal. This makes this system suitable to be incorporated in real-time control loops. In addition to the evaluation presented in section 8.3.2, the system has
Table 8.4: Trinocular stereo, synthesis results overview.

also been tested in real life environments. The synthesis results can be found in Table 8.4.

### 8.3.5 Conclusion

A trinocular disparity processor has been proposed. Nine cost curves resulting from pairwise comparison of three cameras have been investigated. Each data stream has been investigated independently from one another and ultimately a hierarchic classification algorithm selects the most promising disparity value. For each of the nine cost curves, a classification algorithm is trained in order to provide a confidence indication for their disparity values. These confidences are passed on to the second level classifier which selects the disparity to use, or indicates that no disparity has been
Real World Feasibility and Evaluation

Figure 8.20: Trinocular stereo, hardware block diagram.
found. The selection of classification algorithms has been used as guideline for the implementation in an FPGA. These results illustrate that the quality of the disparity space image increases by using more cost curves from a trinocular camera. Due to the adaptability of the window matching module and the hierarchic classification structure, the system can easily be expanded with more data streams to further improve the disparity space image. Compared to the literature, disparity maps of different camera pairs are combined using binary classification.

8.4 Evaluation

Table 8.5 and Table 8.6 give an overview of the state of the art stereo cameras with respect to the required hardware resources. PDS stands for points times disparities per second. It is the number of disparities that is being calculated in each second. Comparing these results with our approach is difficult, the performance depends heavily on the kind of matching method and the type of FPGA. The implementation in this dissertation uses a complex adaptive weight cost aggregation, which has not been used by other implementations. Ding et al. [2011] have used adaptive weights in their implementation, but modified it for a sliding window approach, which leads to non-optimal weights. Most implementations in the table use more powerful FPGA’s (Virtex ad Stratix). Compared to our applications, they provide more hardware resources, but are more expensive.
<table>
<thead>
<tr>
<th>Method</th>
<th>Image Size</th>
<th>Disparity Range</th>
<th>FPS</th>
<th>PDS ($\times 10^6$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lee et al. [2005]</td>
<td>SAD</td>
<td>640 $\times$ 480</td>
<td>64</td>
<td>31</td>
</tr>
<tr>
<td>Mingxiang and Yunde [2006]</td>
<td>SAD</td>
<td>640 $\times$ 480</td>
<td>32</td>
<td>100</td>
</tr>
<tr>
<td>Murphy et al. [2007]</td>
<td>SAD</td>
<td>320 $\times$ 240</td>
<td>20</td>
<td>150</td>
</tr>
<tr>
<td>Jin et al. [2010]</td>
<td>Census</td>
<td>640 $\times$ 480</td>
<td>64</td>
<td>200</td>
</tr>
<tr>
<td>Ambrosch and Kubinger [2010]</td>
<td>SAD</td>
<td>750 $\times$ 400</td>
<td>60</td>
<td>60</td>
</tr>
<tr>
<td>Villalpando et al. [2011]</td>
<td>SAD5</td>
<td>1024 $\times$ 768</td>
<td>.</td>
<td>15</td>
</tr>
<tr>
<td>Ding et al. [2011]</td>
<td>TSAD</td>
<td>640 $\times$ 480</td>
<td>60</td>
<td>51</td>
</tr>
<tr>
<td>Tomasi et al. [2012]</td>
<td>Adaptive Weights</td>
<td>640 $\times$ 480</td>
<td>256</td>
<td>57</td>
</tr>
</tbody>
</table>

| Stereo                  |            |                |    |                    |    |
| Binocular Stereo        | SAD        | 800 $\times$ 600 | 80  | 56                 | 2,150 |
| Trinocular Stereo       | Adaptive Weights | 640 $\times$ 480 | 24  | 52                 | 412 |

Table 8.5: Comparison of real-time performance and resource usage of published stereo vision systems implemented on FPGA.
Table 8.6: Comparison of real-time performance and resource usage of published stereo vision systems implemented on FPGA.

The applications presented in this chapter also include all processing steps including rectification without using external memories. Only Jin et al. [2010] report a global hardware complexity, however on a more powerful FPGA.

8.5 Conclusion

In this chapter, a camera capable of capturing depth information has been presented. It consists of multiple CMOS sensors connected directly to an FPGA. The pixel streams originating from the sensors are directly processed on the FPGA without usage of a frame buffer. This allows for a very low latency system of only a couple of image lines. The architecture is constructed in such a way that adding additional image sensors is straightforward.

Usage of multiple image sensors provides more robustness and a more complete overview of the environment. However it will also provide redundant information: e.g. multiple possible disparity values for each pixel. A streaming classifier is developed to select the correct value. All functionalities like image demosaicing, box filter, rectification, window comparison, feature generation and classification have been implemented in hardware. All modules have been designed especially to reduce memory footprint such that the complete system can be implemented without using external memories.
Conclusion and Future Research Directions

9.1 Conclusion

In this dissertation, real time video architectures have been investigated for the simultaneous processing of multiple image sensors. In this respect, dedicated hardware structures and architectures have been designed. The feasibility of real world implementation is performed using an Field Programmable Gate Array (FPGA).

Reconfigurable hardware building blocks for generic video and vision systems are developed, with special focus on the memory footprint of the on-chip hardware. A memory efficient image warping technique is presented which allows alignment of multiple image sensors using a single FPGA. Next, a tiling parallel memory architecture is introduced. It allows multiple pixel streams to be processed using a time sharing approach. To improve image matching, a novel binary adaptable support region for incorporation in a window stereo matching System-on-Chip (SoC) architecture is proposed.

It is shown that multiple image sensors can work together to provide a good Disparity Space Image (DSI) using a hierarchic classification method. To illustrate this approach, a depth camera has been developed which is working according to this principle. It is a true single chip implementation.
9.2 Future Research Directions

Several parts of this dissertation can be extended with future work. For image matching, a closer look should be taken on adapting the proposed method for a sliding window approach. Adapting successfully such an approach could significantly reduce the resource usage. Recently deep learning artificial neural networks (Ciresan et al. [2012]) are highly successful in learning patterns for multilevel problems, adapting this technique for classification of the disparity information could prove interesting.

It is also possible to extend the proposed framework for different applications like image stitching. Initial Alignment of multiple image sensors is already available in the framework. The image matching module can help with solving the parallax problem with image stitching.

Overall, investigation of how to include the time dimension could prove interesting.

The next step is to investigate how multiple of these cameras can be used together to look at the environment from different perspectives and greater distances. The cameras should be able to cooperate with each other to resolve situations which are not clear from a single viewpoint.
Scientific Contributions and Publications

The results presented in this dissertation were collected during four years of research. During this period, several scientific articles and contributions in proceedings were published, reflecting different parts of the research presented in this dissertation.

Book Chapters:

- Motten et al. [2014]: A. Motten, L. Claesen, Y. Pan, ”Trinocular Stereo Vision using a Multi Level Hierarchical Classification Structure,” in VLSI-SoC: Forward-Looking Trends in IC and System Design (Best Papers of VLSI-SoC 2012), Springer, Invited Book Chapter to be Published.

Journal Publication:


Conference Publications:


Beeldsensoren zijn alomtegenwoordig geworden. Het toevoegen van extra beeldsensoren aan een visiesysteem hoeft niet duur te zijn. Deze beeldsensoren genereren echter wel een grote hoeveelheid gegevens, waarvan de tijdige verwerking niet triviaal is. Het doel van dit doctoraal onderzoek is een camera-architectuur te ontwikkelen, dewelke specifiek gericht is op het verwerken van beelden die komen van meerdere beeldsensoren. Om deze verwerking in werkelijke tijd met een zo klein mogelijke vertraging uit te voeren, werd er gekozen voor een enkele chipoplossing.

Een dieptecamera werd gekozen als specifieke toepassing voor deze architectuur. Wanneer een object gelocaliseerd wordt op twee of meerdere beeldsensoren, kan de afstand tussen de beeldsensoren en het object bepaald worden. In deze toepassing wordt een volledig dieptebeeld berekend door de vergelijking van iedere pixel op de verschillende beeldsensoren. Hierbij wordt gebruik gemaakt van de omgeving rond de pixel om deze vergelijking meer robuust te maken.

De beelden resulterend van de verschillende sensoren hebben eerst enkele bewerkingen nodig vooraleer ze met elkaar kunnen vergeleken worden. Voornamelijk de uitlijning van de verschillende beeldsensoren heeft een grote hoeveelheid geheugen nodig. In dit doctoraat wordt een nieuwe geheugentechniek voorgesteld dewelke rekening houdt met het patroon van de vervorming. Deze leidt tot een aanzienlijke vermindering (tot 63 %) van de benodigde geheugens.

De volgende stap is een vergelijking maken tussen de verschillende beeldsensoren. Bij het vergelijken van pixels in een afbeelding, wordt een gebied rond de pixel gebruikt om de vergelijking meer robuust te maken. De meeste visie-architecturen gebruiken een combinatie van lijnbuffers met schuifregisters om specifieke gebieden van afbeeldingen op te slaan en halen deze op voor verwerking. Deze traditionele methode
maakt geen optimaal gebruik van de beschikbare parallelle geheugenblokken. Dit doctoraat behandelt een nieuwe geheugenarchitectuur dewelke parallelle toegang toelaat tot alle pixels in een geselecteerd venster. Deze architectuur laat eveneens toe om het volledige gebied te vernieuwen in een enkele klokcyclus. Hierdoor kan de verwerking van de verschillende beeldsensoren opgedeeld worden in de tijdsdimensie.

Selectie van de ideale ondersteunende omgeving is belangrijk: deze moet groot genoeg zijn om te kunnen discrimineren, maar Klein genoeg zijn om scherpe diepteverschillen te behouden. In dit doctoraat wordt een aanpasbare omgeving voorgesteld, waarvan de selectie afhankelijk is van kleurgelijkens en ruimtelijke afstandsberekenningen. Pixels die niet behoren tot deze selectie zullen geen invloed hebben op het uiteindelijke resultaat. Na deze vergelijking bekomen we een dieptebeeld.

Verschillende beelden vergelijken is nooit volledig correct. Meerdere factoren kunnen de correctheid van het dieptebeeld negatief beïnvloeden. Dit onderzoek toont aan dat een binaire vertrouwenswaarde berekend kan worden voor elke pixel. Deze methode is uitgebreid voor meerdere beeldsensoren in een coöperatief multi-camera systeem.

Ondersteuning om specifieke architecturen te genereren uit deze bouwblokken is belangrijk. Onze ontwerpomgeving maakt het mogelijk om verschillende architecturen automatisch te genereren, afhankelijk van de vereisten, parameters en aantal beeldsensoren. Eveneens worden bestanden gecreëerd die het mogelijk maken om de gegenereerde architectuur te testen op verschillende abstractieniveaus, van een pure software simulatie tot en met een hardware implementatie op een FPGA.

In dit doctoraat werd aangetoond dat een meerdere-camera-architectuur efficiënt kan geïmplementeerd worden in een enkele chip zonder gebruik te maken van aparte geheugenchips.
Glossary

**ABW**  Adaptive Binary Weights. 73–76, 78, 81, 84, 93, 111, 112, 118, 123, 126, 138

**AD**  Absolute Differences. 67

**AMU**  Address Management Unit. 58

**ANN**  Artificial Neural Network. 96, 98, 100

**ASIC**  Application Specific Integrated Circuit. 55

**AW**  Adaptive Weights. 71, 73, 74, 76, 78, 80, 116

**CCT**  Cost Curve Threshold. 92

**CMOS**  Complementary Metal Oxide Semiconductor. 12, 39, 55

**CPU**  Central Processing Unit. 9

**CT**  Census Transform. 68–70

**CUR**  Curvature. 91

**DRAM**  Dynamic Random Access Memory. 55

**DSI**  Disparity Space Image. XIII, 2, 97, 116, 126–130, 133, 134, 137, 138, 145

**DT**  Decision Tree. 95–98, 121, 123, 130
**FIFO** First In, First Out. 11, 12, 29, 30, 54, 56

**FN** False Negative. 95, 100

**FP** False Positive. 95, 100


**GPU** Graphics Processing Unit. 10

**HDL** Hardware Description Language. 109

**LE** Logic Element. 13

**LRC** Left Right Consistency. 92

**LRD** Left Right Difference. 92

**LUT** look-up table. 19–21, 39, 40, 43, 51

**MC** Matching Cost. 66, 67, 91, 98

**PKR** Peak Ratio. 92

**SAD** Sum of Absolute Differences. 67, 70, 84, 91, 99, 111, 116, 118, 119, 121, 123, 126–128, 130, 133, 135, 137, 138

**SD** Squared Differences. 67

**SEG** Segmentation Size. 93

**SNDD** Sum of Neighbouring Disparity Differences. 93, 98, 100

**SNDDBW** Sum of Neighbouring Disparity Differences using adaptive Binary Weights. 93, 100

**SoC** System-on-Chip. 3–5, 9, 55, 66, 73, 81, 86, 108, 115, 117, 145

**SSC** Sum of Streaming Confidence metrics. 94, 133
SSD  Sum of Squared Differences. 67, 69, 70

SSDD  Sum of Streaming Disparity Differences. 94, 97, 130, 133

SSU  Stream Selection Unit. 135

TAD  Truncated sum of Absolute Differences. 68, 70, 83, 84, 111

TEX  Texture. 92, 98, 100

TN  True Negative. 95

TP  True Positive. 95

TSD  Truncated sum of Squared Differences. 68–70

VGA  Video Graphics Array. 12

VLSI  Very Large Scale Integration. 9

WTA  Winner Takes All. 68–70, 84, 118, 126
Bibliography


Miguel A. Vega-Rodriguez, Juan M Sanchez-Perez, and Juan A. Gomez-Pulido. An FPGA-based implementation for median filter meeting the real-time requirements of automated visual inspection systems. 2002.


