The University of Western Australia
Faculty of Engineering, Computing and Mathematics
School of Electrical, Electronic and Computer Engineering
Centre for Intelligent Information Processing Systems

FPGA Based Embedded Stereo Vision Algorithms

Final Year Project

David English (10225810)

Supervisors: Associate Professor Thomas Bräunl
Associate Professor Anthony Zaknich

Submitted 27th October 2006

Letter of Transmittal

David English
20 Neville Rd
Dalkeith WA 6009

27th October 2006

The Dean
Faculty of Engineering
The University of Western Australia
Nedlands WA 6009

Dear Sir,

It is with great honour that I submit this thesis entitled "FPGA Based Embedded Stereo Vision Algorithms" as partial fulfilment of the requirements for a degree of Bachelor of Engineering with Honours.

Yours Faithfully,

David English

Abstract

With improvements in technology, especially microprocessors and Field Programmable Gate Arrays (FPGA), it is possible to bring rapidly increasing levels of intelligence to autonomous robots and other embedded systems. The University of Western Australia and other research groups have need for a robotics control platform which offers traditional features such as motor and servo control alongside modern features such as fast processors, large memory, FPGAs, Bluetooth networking for control and self-organisation, as well as high speed USB interfaces for expansion. Such a platform needs to be small form factor, low power consumption and cost effective in small to medium quantities.

This project, working in a team of three, focuses on the design and verification of a new hardware platform that aims to meet all of the goals outlined, as well as accompanying software and FPGA logic designs to make the features accessible from user space under a standard embedded Linux operating system. Individually, research was also undertaken into stereo vision, with the aim of proposing an algorithm which can be made to fit this specific hardware platform, while maximising quality and performance where possible.

Significant effort has been put not only into the design of the systems presented, but also verification. Extensive testing of the timing critical components on the PCB was undertaken, and results with accompanying analysis of maximum stable speed is presented where relevant. On the logic design side, a software tool to assist in bridging the gap between traditional test bench code and VHDL designs is presented. This framework allows the rapid application of many automated test cases to logic simulations at both the behavioural and timing levels. The stereo vision research code presented includes a powerful visualisation and analysis system to assist in testing algorithms for validity and comparing against expected results.

Acknowledgements

I would like to thank a number of people for their invaluable help during the year.

First I would like to thank my parents, for their infinite patience and support, especially after being woken up countless times by my stumbling home and into bed at early hours of the morning.

Next I would like to thank Bernard Blackham and Lixin Chin, the other members of the core design team for the M6 hardware platform and associated software and logic design. The sheer scale of the task made teamwork vital and the level of commitment and determination from all members was admirable.

I also owe a great deal to my original supervisor, Associate Professor Thomas Bräunl, who made such an exciting project available to us. His expertise, support and encouragement were vital in the difficult, early stages of the project.

Many thanks to Ivan Neubronner in the electronics workshop. I could not have possibly hoped for a more patient or knowledgeable support than he was able to offer with the manufacturing side of the project. Ivan's accumulated wisdom from previous Eyebot generations and his dedication to their support and success is astonishing.

I would also like to thank my secondary supervisor Associate Professor Anthony Zaknich for his advice in many areas, especially his encouragement to seek new and interesting results.

Thanks to Grace Choo, for not only being the second last undergraduate remaining in the Robotics Lab as the year reached its conclusion, but also for keeping me on my toes during tea breaks and other potentially catastrophic moments.

Finally a very special thanks to everyone who missed out on a mention due to lack of space, I feel priviledged to have known so many great people.

Table of Contents

1 **Introduction** 1

1.1 Robotics, Automation and the Rise of the FPGA 1

1.2 Embedded Vision 2

1.3 Project Motivations and Objectives 3

1.4 Outline of Thesis 3

1.5 The Eyebot M6 Feature Set 5

1.6 Alternative Platforms 5

1.7 Eyebot M6 FPGA Sub-System Details 6

2 **FPGA Image Processing Base** 9

2.1 M6 Platform FPGA Resources and Clocking 9

2.2 Component Design 11

2.3 FPGA Bus Arbitration 11

2.4 CPU Interface 13

2.4.1 CPU Addressing 14

2.4.2 CPU Reads 14

2.4.3 CPU Writes 16

2.5 FPGA External Memory Bus (SRAM) 17

2.5.1 Controller Design 17

2.5.2 Determining Maximum Memory Clock 21

2.6 FPGA Camera Interface 23

3 **FPGA Convolution Implementation** 25

3.1 2D Convolution 25

3.1.1 Windowed Convolution 25

3.1.2 Windowed Convolution Examples 27

3.2 FPGA Implementation 29

3.2.1 Performance Results 32

4 **Geometry of Depth Perception** 35

4.1 Projective Geometry 36
4.2 Full Pinhole Camera Geometry 37
4.3 Epipolar Geometry 38
4.4 Co-planar Images 38
4.5 Practical Rectification 39
4.5.1 Affine Transform Rectification 41
4.5.2 General Stereo Rectification 42
5 Real Time Disparity Mapping 43
5.1 Disparity Mapping Methods 43
5.2 Pre-Processing Filters 44
5.3 Choice of Match Comparison Metric 46
5.4 Selecting Window Size 47
5.5 Performance Optimisation of SAD Algorithm 51
5.5.1 Minimisation of Search Area 51
5.5.2 Multiple Resolution Search 52
5.5.3 Reusing Earlier Results as a Lookup Table 52
5.6 FPGA Performance Estimates 53
6 Disparity Errors 55
6.1 Sub-pixel Matching 58
6.2 Disparity Analysis Tool 59
6.3 Accuracy Improvement through Confidence Estimation 60
6.4 Applications of the Uniqueness Constraint 64
6.4.1 Variable-Window Matching 64
6.4.2 Second Map Correspondence Test 65
6.5 Statistically Derived Method 66
6.5.1 Second Best Match Distance 66
6.5.2 SAD Window Comparison Minimum Error Thresholding 66
6.6 Implementation Details 70
7 Conclusion 73

7.1 Outcomes 73
7.2 Future Work 74
References **77**
A Stereo Vision Analysis Source Code Overview **81**
A.1 Source Overview 81
A.1.1 Algorithms (Native C++) 82
A.1.2 User Interface (Managed C++ - .Net 2.0) 83
A.2 Overview of Important Classes 84
B Stereo Vision Analysis Logs **87**

List of Figures

Figure 2-1 FPGA image processing internal inter-connect bus sample timing diagram 12
Figure 2-2 Example schematic for internal image processing bus inter-connection. 13
Figure 2-3 Oscilloscope output for FPGA side (top) and CPU side (bottom) of buffer chip for 2mA (left) and 6mA(right) FPGA drive current strength 16
Figure 2-4 FPGA external memory system overview 19
Figure 2-5 Oscilloscope output for FPGA to SRAM data pin 0 21
Figure 2-6 timing diagram for FPGA to SRAM transfers, back to back read and write modes 22
Figure 3-1 Examples of edge detectors using 3x3 windowed convolutions. Source image reproduced courtesy of Lixin Chin. 27
Figure 3-2 2D Gaussian filter in frequency (left) and spatial (right) domains... 28
Figure 3-3 Noisy source image (above right) and resultant image with reduced noise and texture (right) after application of 4x4 windowed convolution Gaussian filter (above). An enlarged region (red box) helps identify the difference in noise. 29
Figure 3-4 FPGA resource utilisation for 4x4 convolution block. 31
Figure 3-5 C++ test harness with .NET UI. 32
Figure 3-6 Performance estimates for full real-time convolution system running on FPGA, with comparison to PC performance (Reference algorithm running on a 1.83GHz Intel Core Duo, binaries built with Microsoft Visual Studio 8. 33
Figure 4-1 Basic projective geometry[12] 37
Figure 4-2 Geometry for a pair of cameras 39
Figure 4-3 A comparison of the different components of relative rotation for two camera images, including the depth skew from the yaw and pitch components. 41
Figure 5-1 Example discrete 5x5 Laplacian of Gaussian (LoG) filter[18] 45

Figure 5-2 Graphical summary of data from Table 5-1. 46
Figure 5-3 Disparity Error for various window sizes using the plain SAD disparity method 49
Figure 5-4 Disparity maps and error relative to ground truth[19, 25] for basic disparity algorithm applied to Tsukuba image set at selected window sizes. 51
Figure 5-5 FPGA Implementation SAD performance estimates for 50MHz Xilinx Spartan S3E500 FPGA with 100MHz RAM compared to 1.83GHz Intel Core Duo based PC (single thread code) with 2MB cache and 667MHz DDR2 RAM (352x288 source image resolution, 7x7 window.) 54
Figure 6-1 The two types of visible area discrepancy between stereo image pairs 56
Figure 6-2 Discrepancy in quantised stereo image windows due to sub-pixel offsets. 56
Figure 6-3 Discrepancy in stereo image windows due to noise in detector ADC 57
Figure 6-4 Discrepancy in quantised stereo image windows due to sub-pixel offsets 57
Figure 6-5 7x7 window Tsukuba disparity map with example low confidence matches highlighted next to ground truth 61
Figure 6-6 An early build of the final disparity analysis application written in support of this project. A simultaneous cursor and detailed analysis logging can be seen in action. 61
Figure 6-7 Disparity maps with low confidence regions highlighted for various estimation techniques. 63
Figure 6-8 Mean disparity error value as a function of the minimum SAD window's error value 67
Figure 6-9 Mean error and area adjusted pixel counts as a function of the best matching SAD window's error value. Counts are adjusted so that all areas along the logarithmic graph correspond to the same amount of pixels. 68
Figure 6-10 Plot of the adaptive moving average of SAD disparity error compared to the true disparity for different minimum SAD window error values.

The adaptive moving average groups the values into equally sized bins, irrespective of the density of data points along the x-axis. 69
Figure 6-11 3D visualisation of minimum window error threshold corrected depth map(green) as compared to original (red) and truth (grey.) Green areas represent points where the depth map was not completely fixed, while red errors represent areas that are wrong in the plain disparity map but fixed after thresholding and nearest replacement..... 71
Figure 6-12 Results of applying threshold test followed by various corrective procedures to test images..... 72

List of Tables

Table 2-1 M6 Platform FPGA Resources [5]. 10
Table 2-2 Relative performance of bidirectional transfers 20
Table 2-3 Time period variables for FPGA - SRAM bus timing 22
Table 2-4 Data pin 0 rise times and estimated / experimental maximum clock settings for the two PCBs tested 23
Table 2-5 Summary of relevant camera output pins 24
Table 4-1 Stereo source images captured with a single camera moved by hand, along with SAD disparity map and minimum window error. It can be seen that the SAD algorithm, as discussed later, does not require perfect alignment. In particular many of the regions with poor accuracy are due to deficiencies in the SAD algorithm irrespective of rectification, which will be covered in more detail in the later chapters. 40
Table 5-1 Error compared to ground truth[19] for Tsukuba image set using various pre-processing methods followed by a 7x7 SAD disparity map 45
Table 5-2 Results of comparing the effectiveness of different window sizes on calculating the depth of the Tsukuba image pair as compared to the supplied ground truth[19]. 48
Table 5-3 Comparison of selected FPGA resource / timing requirements and errors incurred for different SAD window side lengths 50
Table 6-1 Error compared to ground truth for Tsukuba image set for a 7x7 SAD disparity map in the region expected to have particular difficulties[29]. 58
Table 6-2 Error totals and counts of good pixels mistakenly marked as errors for various confidence estimation techniques and parameters. 62
Table 6-3 Error totals for nearest replacement scheme revised depth maps for various confidence estimation techniques and parameters. 62

Terminology

M6 Platform	The new generation of Eyebot robot controller board developed during the course of this project.
CIIPS	Centre for Intelligent Information Processing Systems - A part of the Department of Electrical, Electronic and Computer Engineering at the University of Western Australia
FPGA	Field Programmable Gate Array – A chip containing large quantities of reprogrammable logic with IO and other capabilities
SAD	Sum of Absolute Differences – The combined difference of each pair of values sharing the same location in each of two comparison windows.
SASD	Sum of Absolute Squared Difference – Similar to SAD but the difference of each pair of values is squared before the summation
ADC	Analog to Digital Converter – Using inside a camera chip to convert the analog signal generated by light hitting each photo-detector into a binary digital signal.
SRAM	Static Random Access Memory – High performance, low density (high cost) usually clock synchronous memory which does not require refresh periods and is usually capable of multi-port access.
SDRAM	Synchronous Dynamic Random Access Memory – Lower performance, higher density, completely clock synchronous memory.

Variable Table

X	3D coordinate components (Horizontal)
Y	(Vertical)
Z	(Depth)
U	2D image coordinate components (Horizontal) / 2D frequency component
V	2D image coordinate components (Vertical) / 2D frequency component
K	Camera Projection Matrix (intrinsic parameters)
T	Camera Transformation Matrix (simple extrinsic parameters)
C	Camera Calibration Matrix
P_{bidir}	Performance of fully loaded interleaved bi-directional data transfers
P_{unidir}	Performance of fully loaded uni-directional data transfers
N_{batch}	Number of items transferred before a change of direction occurs under full bi-directional load.

Chapter 1

Introduction

1.1 Robotics, Automation and the Rise of the FPGA

As computing power has grown and cost has decreased, robots and other automation machinery have increasingly replaced human labour and supervision required for manufacturing and other repetitive tasks. Similar automated intelligence can play an active safety role, for instance in cars. Often automated systems are only viable if they are reasonably priced, fit modest dimensions and do not consume excessive amounts of power. Since the dawn of microprocessors, which pack many transistors onto a small area, the problem for most systems has not been size of the electronics, but power consumption and the size, cost and or weight of the supply required. In modern times, with chips consisting of tens of millions of transistors becoming cheap and requiring only a few square centimetres of area on a circuit board, this problem has only exacerbated.

Increasing on chip transistor counts have allowed general purpose processing architectures to grow in performance. Usually, with more complex designs this performance gain is much smaller than the increase in power consumption. For instance, a wider issue on a super-scalar processing grows the power consumption per instruction[1]. A slightly less power hungry approach is to go for multiple, seperate cores or larger cache sizes. Still dedicated logic, because it can be designed to perform many fixed steps with each clock cycle, can gain much greater performance to power and performance to transistor count ratios.

It accomplishes this by removing unnecessary operations, having no instruction decoding and no execution tracking and automated fetch and cache management. On suitable tasks dedicated logic can still achieve greater parallelism combined with lower clock speeds to realise the same or better throughput as a general purpose CPU. Despite this, dedicated logic is still used for very few tasks due to the difficulties in design and testing of such logic, and the very high costs of manufacture in small quantities. Recently the decrease in costs of large transistor counts has made Field Programmable Gate Arrays (FPGAs) become competitive, offering a good middle ground between software and true hard-wired designs. They have dedicated logic to perform certain tasks, and vast arrays of reconfigurable (through programming) logic to perform any task with power efficiency not far behind dedicated logic. This means FPGA designs can bring high end computing performance to cost and power conscious embedded systems.

1.2 Embedded Vision

Computer vision is the body of theory related to the processing of images by computational systems. Image data is two dimensional, and modern cameras will produce pixels with hundreds of thousands or even millions of pixels at video frame rates. Performing processing tasks on such data sets can involve high orders of complexity, especially for advanced procedures such as stereo vision depth recovery. Getting such tasks to run in real-time poses a challenge even to high end PCs, for embedded systems they have traditionally been out of the question.

As covered in the previous section, advances in technology to aid processing performance, but dedicated logic will always be superior for most tasks. Computer vision tasks in particular tend to have high degrees of exploitable parallelism and repetitiveness, because of their large input data sets. Using

FPGAs to implement algorithms with low power consumption and low cost cost in small to medium quantities is therefore a rapidly growing area of research.

1.3 Project Motivations and Objectives

The University of Western Australia and other research institutions are developing autonomous robots which can perceive and interact with the world. This project aims to provide a base computing platform which can serve as the nerve centre for these robots.

The specific objectives of this project were to:

- Design a major overhaul of the ageing Eyebot platform with modern computational power and expanded IO
- Maintain an affordable price in small quantity production
- Maintain a small form factor and low power consumption
- Utilise an FPGA to maximize IO capabilities and image processing performance
- Implement stereo vision to give robots utilising the new platform depth perception

1.4 Outline of Thesis

Since the overall project began at the same time as this thesis project, the work performed included selection, sourcing, design and testing of hardware in collaboration with a team of other CIIPS students. The development of the FPGA logic and software for the board were at many points stalled by the hardware development. Therefore this thesis focuses on the following areas of work performed:

- Design and implementation of the FPGA logic and software testing frameworks required for development of FPGA enhanced image processing algorithms on the M6 platform. This includes the development and verification of: an internal memory bus, communication with external cache memory, communication with the main CPU on the board, and communication with the cameras.

- Development and testing of FPGA logic for a two-dimensional windowed convolution algorithm. The windowed convolution is a common, multi-purpose tool in image processing. The implementation of this algorithm helped in determining the requirements of the FPGA internal processing layout, and also serves as a guide to the resource utilisation and performance of image processing constructs within the FPGA. The results of the latter will be important in evaluating the suitability of other, more complex algorithms before extensive time is spent on implementation.

- Development of code to link the Xilinx ISE and other HDL simulators to C code. This can be used to evaluate circuit results on the fly during logic level simulation, aiding debugging and verification of current and future image processing designs.

- Research, analysis, implementation and evaluation of software stereo vision algorithms with the aim of creating a correct reference algorithm which is suitable to FPGA implementation. An algorithm will be proposed for implementation based on its quality, performance and feasible resource requirements within the context of the M6 platform.

1.5 The Eyebot M6 Feature Set

The new Eyebot M6 is designed to be a vision processing platform. To this end significant computation processing power has been included. For general purpose calculations there is a 400MHz XScale processor, 64mb of SD-RAM, 16MB of flash. This processor runs Linux 2.6 and forms the basic user space programming area for the board. A Spartan 3E FPGA with attached 1MB SRAM provides space for the implementation of dedicated vision processing acceleration logic.

The Eyebot M6 IO is mostly a super set of the original Eyebot functionality. The new features that can be found include onboard USB slave and host ports, 10/100mbit Ethernet and Bluetooth for better expansion and communications capabilties. In addition four motors are now supported, and all motors, servos, PSDs, infrared ports, serial ports and other peripherals can be used simultaneously. The ability to connect two cameras to the FPGA is provided. These cameras are oriented to face the same direction and have modest spacing to aide stereo vision processing.

There have been few compromises with respect to reproducing the old system's functionality. The number of user analogue inputs has been reduced from 5 to 3. This is not anticipated to be a problem as analogue inputs can now be added in any desired quantity via USB connections. The new CPU is not binary compatible with the original Eyebot, but it will be possible to provide almost perfect backwards compatibility to C code through a port of the standard RoBIOS libraries, similar to how EyeSim current operates. This will allow existing applications to be modified only where they need to take advantage of the new functionality.

1.6 Alternative Platforms

The updated Eyebot M6 developed during the course of this project is broadly similar in concept to the Balloon Board[2] developed jointly by the University of Cambridge and MIT. Although this board has been released during the course of the project, it does not meet all the objectives of the M6 project. Whilst most immediate is the lack of onboard Ethernet and Bluetooth networking, a potentially more serious disadvantage is the lack of dedicated FPGA to camera connectors. Without this the vision processing potential of the board is hampered greatly.

1.7 Eyebot M6 FPGA Sub-System Details

This FPGA is a Xilinx Spartan3E 500 that saw commercial release during the year this project was undertaken. At the time of its release, the Spartan3E series was manufactured in the most up to date process of any FPGA (90nm) and offered the maximum logic to dollar ratio of any suitably sized FPGA available. The Spartan 3E differs from the Spartan3s of the same packaging in that it offers more logic slices, Block RAM, hardware multipliers and input pins for the same packaging. The disadvantage is mainly that it has less power pins, and is therefore not capable of as high a total simultaneous drive current on its output pins. Since all inputs and outputs from the FPGA in the M6 platform are logic level, this disadvantage is not considered important. The additional internal resources and input pins, on the other hand, are important as they allow more features than would be possible with the best Spartan3.

A comparison between the selected FPGA and non-Xilinx FPGAs is available in [3]. A very strong argument in favour of the Xilinx part for image processing was the largest amount of internal memory. For the other functions of the platform, the high IO count available in a non-BGA was an important consideration. For all applications the maturity of Xilinx's free HDL development environment was an important factor.

One of the key intentions of the new hardware platform is to utilise the FPGA to not only provide more IO than is possible from the CPU, but also to offload many processing tasks to improve system performance. These tasks can be divided into two categories. The interrupting tasks involve little computation, but would need to interrupt the CPU often. This would cause a significant performance penalty through inefficiency introduced by switching contexts. The offloading of these tasks to the FPGA on the Eyebot M6 platform is covered in [3]. The other category of processing task that can be offloaded is image processing. Computer vision systems typically require highly repetitive operations performed across large 2D data sets. These tasks are often ideally suited to offloading to the FPGA as the FPGA is at its most efficient when performing tasks repetitively, and has multiple cache RAM blocks which can all be accessed simultaneously to achieve much higher bandwidth per clock than a conventional processor. On the Eyebot M6 platform the FPGA also has its own RAM, so these tasks can be performed without placing a high load on the CPU's memory bus. This is particularly important because the CPU's memory bus has to sustain the LCD refreshes, in addition to serving the software memory accesses. The rest of this thesis will describe the methods and attempts to offload processing tasks from the CPU to the FPGA on the Eyebot M6 platform. A discussion covering the offloading of different tasks on this platform can be found in [4].

Chapter 2

FPGA Image Processing Base

As outlined earlier, the new Eyebot M6 platform features an FPGA to extend the IO and image processing capabilities of the board. The implementation of the ground work required in support of the image processing side is covered in this chapter.

2.1 M6 Platform FPGA Resources and Clocking

The M6 platform features a Xilinx Spartan3E S3E500 FPGA chip in a PQ208 package. The resources available on this FPGA are summarised in Table 2-1. Utilisation figures discussed in this thesis will be as a percentage of those available in this particular FPGA. It is important to note that FPGA utilisation does not sum strictly linearly. The placement and therefore routing methods available vary as different combinations of logic components are included. Still these figures offer a rough guide which can help in estimating the feasibility of a final design before it is fully implemented.

Feature	**Capacity**
Multipliers	20 (18bitx18bit => 36bit)
Block RAM	(1024x18bit) x 20 [360Kbit total]
Distributed RAM	(16x1bit) x 4672 [73Kbit total]

Feature	Capacity
Registers	(1bit) x 9312 [9.1Kbit total]
Logic Slices	4656
Digital Clock Managers	4

Table 2-1 M6 Platform FPGA Resources [5].

The FPGA receives one external clock signal from a 50MHz oscillator, referred to internally as M6CLK. This clock speed was selected as a lowest common denominator based on experimentation with the clock period of various algorithms. Reasonable degrees of pipelining were employed at all stages to ensure practical results. A second clock which is usually between one and a half and two times faster than the main system clock is generated using a DCM. This second clock is used by the memory controller, and is referred to as MEMCLK, but is also potentially available to other logic. Extreme care must be taken when performing IO between logic of different clock speeds.

It is recommended that algorithms aim to run purely off the main clock where possible. While it is possible to generate additional clock signals with the remaining three DCMs available on the chip, the Xilinx XST synthesiser has difficulty routing circuits with complicated clocking constraints. This occurs because the FPGA is divided into clock regions, and if there are more than four clocks, the FPGA must decide which areas will receive which clocks. The additional clocks required for the cameras and memory controllers already require special care to realise a circuit with valid routing. It is therefore recommended that processing algorithms utilise only the clocks provided. Using additional clocks is likely to result in unpredictable incompatibility with other features.

2.2 Component Design

Developing efficient implementations of algorithms to fit an FPGA's processing model is a laborious process. It is important that each routine implemented be able to co-exist in the greater system. To this end, several small interface standards were agreed during the implementation of the first image processing routines on the FPGA. These common standards deal mainly with the method for streaming data between different routines and external memory.

One of the primary goals of the design was to componentise the FPGA logic as much as possible. It is hoped that users in the future will be able to mix and match different FPGA components within the limits of the resources available. Typically designing for an FPGA involves HDL coding which is far from intuitive. Alternative design systems that feel more like conventional procedural programming languages do exist. These alternatives still require specialist skills[6], and are often tailored very specifically to fit within a particular eco-system such as the AMD Opteron CPU and Xilinx Virtex FPGA pairings popularised by Cray's XD-1 super-computer[7].

2.3 FPGA Bus Arbitration

Individual image processing components within the FPGA need to be able to stream data to each other with the maximum practical bandwidth. Since most data sets to be processed are larger than the FPGA's entire internal memory, it is also necessary that one or both ends of this bus for each device may be connected to a buffer in the external cache memory. Furthermore, the bus must arbitrate requests so that multiple devices making simultaneous access to the external memory do not interfere with each other.

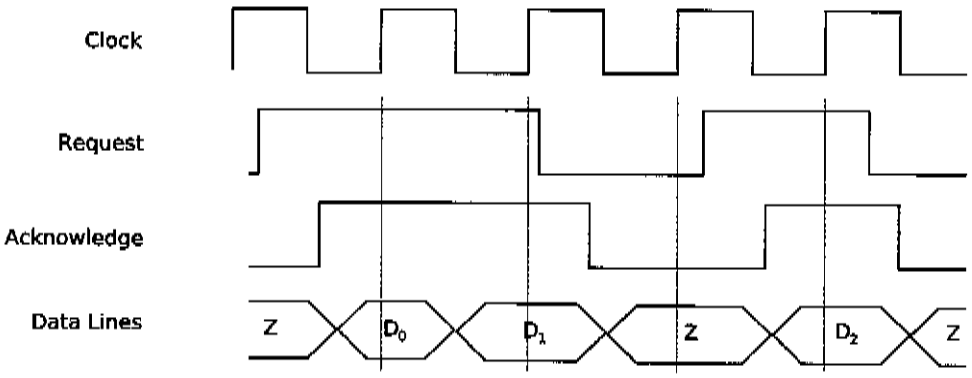


Figure 2-1 FPGA image processing internal inter-connect bus sample timing diagram

To solve all of these objectives in a single, uniform system, a request/acknowledge scheme was implemented with timing as shown in Figure 2-1. This underlying protocol forms the building block of all data streaming connections. The request/acknowledge scheme chosen ensures that source devices know when they need to throttle their output speed for the destination device to keep up, but does not limit transfer rate when the second device is able to process data on every clock cycle.

It is important to note that devices are meant to send their output data lines high impedance unless the host is asserting the acknowledge line. This is to allow for the use of an arbitration tree to divide up access to the memory bus. A binary tree arbitration scheme is used as propagation delay is sufficiently fast as to not slow down devices running off the main clock. The binary tree arbitration scheme is also the simplest to implement and reconfigure priority wise, as great flexibility can be achieved utilizing only the two types of arbitration blocks provided. The first, called the alternate arbiter, alternates between each of its inputs every time it gets access to the bus. The second type, called the priority arbiter, always gives priority to one of its request line, and only allows the other through when the first is idle.

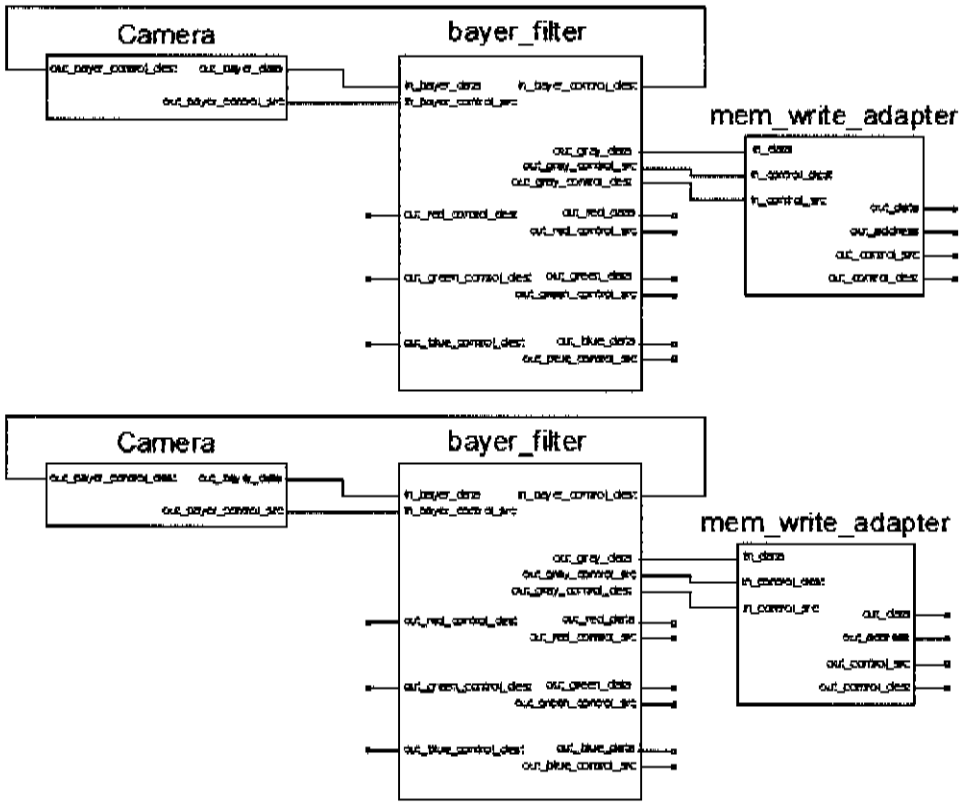


Figure 2-2 Example schematic for internal image processing bus inter-connection.

2.4 CPU Interface

The centre of the Eyebot M6 platform from a user's perspective is the ARM9 CPU. It is necessary that the CPU and FPGA can reliably communicate, so the CPU can give instructions and retrieve results in a timely fashion. The discussion of CPU to FPGA communication will be split into the two separate cases of read and write. In all cases, the CPU is the bus master, and the FPGA implementation must give consideration to this and the other slaves on the PCB, holding the data pins in a high impedance state unless it is meant to be writing to the bus.

The pin layout on the initial PCB configuration required that reads be performed in VLIO mode. In addition, the ready pin was tied to a constant value making it always enabled. This meant, despite using VLIO transfers, the FPGA always had to respond in the minimum time possible. The pin limitation forcing the board to run in VLIO mode has been corrected on the latest PCB generation, and this may make even faster transfer rates possible in the future. The implementation covered below is for the always ready VLIO mode necessary on the first version of the board. The techniques described here allow the FPGA to communicate accurately on all current PCB layouts with all PXA255 bus latencies set to their lowest (fastest) settings.

2.4.1 CPU Addressing

The FPGA has 20 address lines connected to the CPU bus, allowing for a 20 bit address space. The top level FPGA module carves this address space up into 32 device IDs, and each lower module is allowed to read and write through one or more of these IDs. This leaves each device with 15 bits of address space, which corresponds to a 64KB addressable region as the lowest address bit corresponds to 16 bit words.

2.4.2 CPU Reads

Allowing the CPU to read data from the FPGA is the most difficult task. The CPU makes a request on the address lines, and the FPGA needs to respond by placing the appropriate information on the data lines in a timely fashion. Under minimum VLIO timing conditions (highest performance), the total response time must be, at most, four of the PXA processor's memory clock cycles[8].

Working backwards from the point where the CPU latches the data the FPGA has placed on the bus, there is a propagation delay dependent on the selected drive current between the FPGA placing data on the bus and the CPU being able to latch it. This is caused by the delay of the buffer chip (approximately 0.5ns), the rise time of the buffer chip output and the rise time of the FPGA output (the last is the only one which is drive current dependent). Since only one of these times can be altered, there is a minimum time in which the FPGA must place data on the bus early of approximately 4ns. Considering the problem from the start, the FPGA must also latch the appropriate address from the CPU on an edge, and since the address data could settle immediately after the appropriate edge has passed, a full FPGA clock period (20ns) must be allowed for the FPGA to latch the data from the time the bus settles. In addition, there is a small propagation delay for the address lines (approx. 3 ns) and a small propagation delay internally to the FPGA to send the appropriate device's bus output high impedance (negligible). This makes for a maximum total delay of approximately 27ns. This is perfectly acceptable as the period of 4 of the CPUs memory clocks is 37ns (108MHz memory bus).

A problem arises when an attempt is made to directly map FPGA Block RAM into the CPU's address space. Since both addresses and data are synchronous for Block RAM, an additional 20ns delay is introduced between the FPGA latching the address and being able to present the correct data at the output. Although this happens in parallel with the other internal FPGA delays, this still brings the total propagation delay to approximately 46ns, much greater than the 37ns maximum response time. The solution to this, as utilized in the CPU to FPGA SRAM DMA controller logic, is to latch the Block RAM addresses on the falling edge of the clock, and the data output registers on the rising edge. This is perfectly acceptable as the Block RAM is effectively now clocked at 100MHz, far below it's 200MHz maximum. It also halves the added response delay from 20ns to 10ns, bringing the total response time to a total of 36ns, just slightly faster than the 37ns required. This technique has been extensively tested

with a variety of test patterns and multiple gigabytes of data, with no errors recorded.

Since the PXA255 CPU is the bus master, it writes all addresses to the bus. Therefore the FPGA drive current only affects the data portion of reading on the bus. The effect of varying FPGA drive current is demonstrated in Figure 2-3. A 6mA drive current was chosen because it provided a sufficiently small delay for all tested PCBs to work flawlessly over large volumes of test data. It is possible, but unlikely that, if future FPGA images have significantly slower timing characteristics between the output data being latched internally and the same data reaching the output pins (due to a particular routing scenario), a higher drive current of up to 16mA may resolve the issue. It is preferable to keep the drive current to the minimum which does not cause errors, however, as an unnecessarily high drive current causes more interference than need be with other traces on the PCB, as well as causing more difficulties for the power supply circuitry in the form of sharper transients on the source and ground.

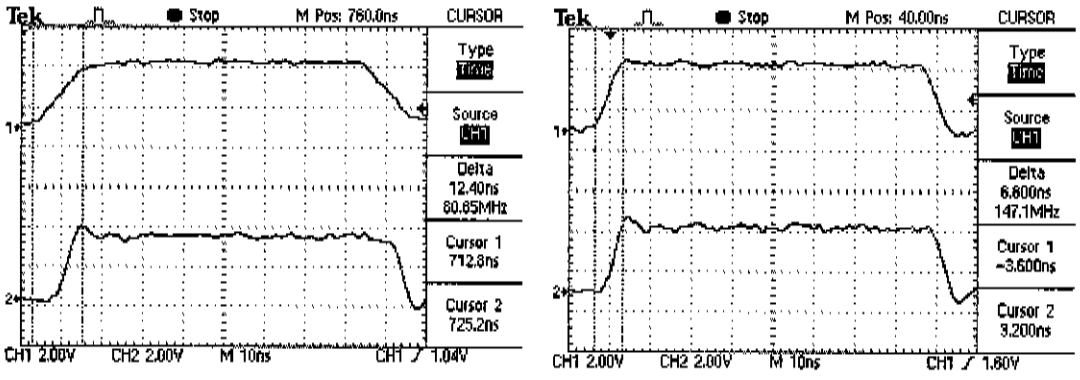


Figure 2-3 Oscilloscope output for FPGA side (top) and CPU side (bottom) of buffer chip for 2mA (left) and 6mA(right) FPGA drive current strength

2.4.3 CPU Writes

CPU Writes present a different set of challenges from the reads. Now the data pins are driven by first the PXA255 CPU, and then the buffer, and unlike the FPGA it is not possible to adjust the drive strength of the PXA255, so it is not possible to improve the rise or fall times without PCB modifications. This is not a problem however as timing is much simpler for bus master writes. The bus master elects the address to write to, and therefore sends both address and data at the same time. The FPGA merely needs to latch both signals at any point in time where they are simultaneously valid on the bus. This is taken care of in the top level modules, and individual devices simply need to be aware that they must accept data if it is available or it will be lost (maximum of once every two FPGA clock cycles.)

2.5 FPGA External Memory Bus (SRAM)

The Eyebot M6 platform's FPGA is directly connected to a RAM chip. This cache memory is dedicated image processing working memory. The chip selected for this task is a single-port Static RAM (SRAM). A dual port chip was not selected, not simply because this would have increased cost, but also because it would have taken additional FPGA IO pins, and this would have required the sacrifice of other features on the board. The SRAM chip was selected over conventional Synchronous Dynamic RAM (SDRAM) for its high performance relative to the complexity of the logic required to utilize it. Unlike SDRAM, the SRAM chip does not have refresh periods and has a simple, consistent low access latency.

2.5.1 Controller Design

The chip selected supports back to back reads and writes at 100Mhz clock speed. There are also burst modes for compatibility with the Pentium CPU's as an external L2 cache[9]. These do not offer any bandwidth advantage compared to

the back to back modes. The back to back modes have the advantage that a completely new address can be specified for every individual read or write operation. An overview of this mode is included in Figure 2-6.

Since back to back mode is utilized, there is no need for components to make their requests in bursts. Therefore, to maximize flexibility, the bus is designed to accept all requests in units of individual SRAM data words (18 bits.) Conveniently this corresponds with the word size of the FPGA's internal SRAMs (Referred to by Xilinx as Block RAMs.) When using the SRAM's back to back mode, there is one factor that must be taken into consideration: The effect of switching between write and read mode on the bandwidth of the chip. Although a read or write can be performed every single clock cycle for unidirectional transfers, the individual reads actually take two clock cycles, one for the address and a second for the data. These cannot be overlapped with the writes as the write requires both sets of lines during the same cycle. The result is that switching from read to write mode requires a clock cycle where no new request initiates, and the result is that, alternating between read and write mode each access limits the chips to performing two memory accesses only every three clock cycles.

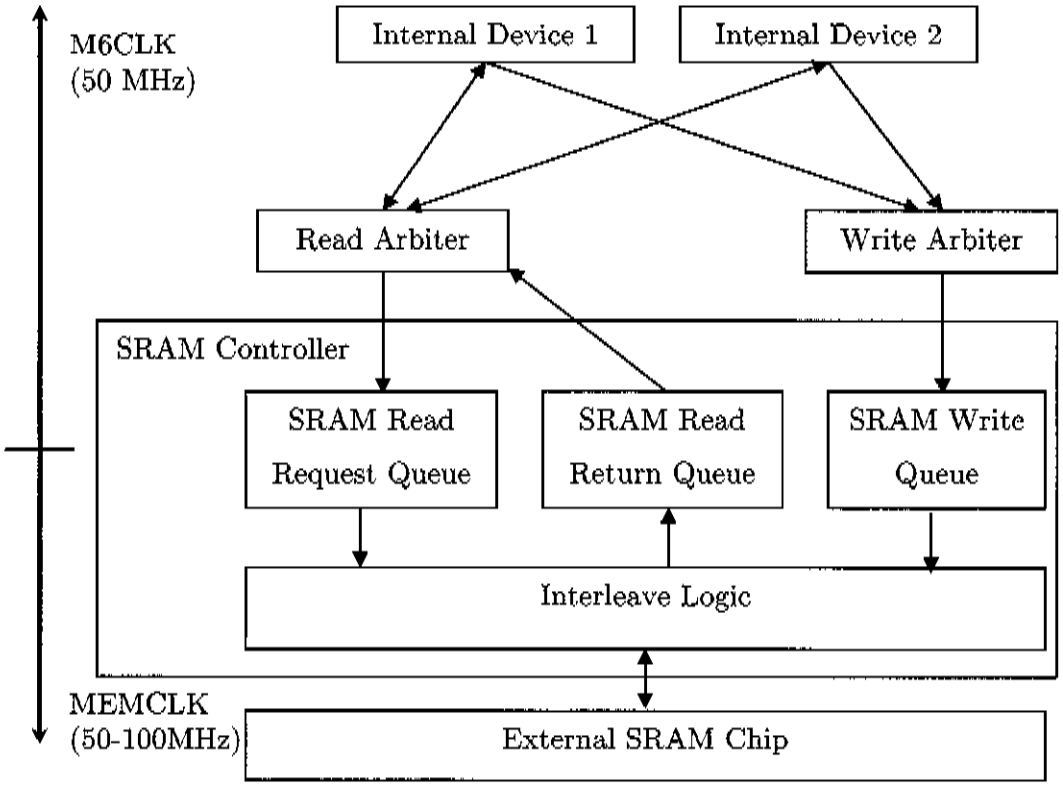


Figure 2-4 FPGA external memory system overview

To reduce the performance penalty, it is necessary to avoid mixing of read and write operations too often. This can be achieved by queueing both both read and write requests, and then performing each type of request in batches. The maximum bidirectional performance can be expressed thus:

$$P_{bidir} = \frac{2N_{batch}}{2N_{batch}+1} P_{unidir} \qquad (2.1)$$

The maximum bidirectional performance as a percentage of the common unidirectional performance for selected batch sizes is indicated in Table 2-2. There is a diminishing performance gain with increasing batch size and clearly queue sizes greater than 16 offer extremely small gains relative to additional queue size.

Request Batch Size (N_{batch})	Performance (% Unidir)
1	66.7%
2	80.0%

3	85.7%
4	88.9%
6	92.3%
8	94.1%
10	95.2%
12	96.0%
14	96.6%
16	97.0%
24	98.0%
32	98.5%
Infinite	100.0%

Table 2-2 Relative performance of bidirectional transfers compared to unidirectional for selected batch sizes

Xilinx Core Generator offers a pre-designed, heavily customisable FIFO with a minimum queue size of 16. Although this FIFO consumes more logic than a custom solution might require, it has several advantages compared to a simpler design. One advantage of using Xilinx's solution is that it contains already heavily tested and proven code, which is very important to a logic design as faults can be difficult to diagnose. Another advantage lies in its key feature, a grey code counter connecting the two ends of the FIFO. This allows each end to run off independent clock signals without glitches. Asynchronous clocking is very useful because the two ends of the FIFO have different clock speed limitations. Whereas the algorithms in the FPGA operate on a 50Mhz clock, the SRAM chip and associated FPGA logic can operate at 100Mhz. The actual maximum memory clock can be slightly lower than 100MHz which the chips can support, depending on the PCB layout and other external factors. Even so, by combining a 50Mhz read and a 50Mhz write bus and connecting them to a near 100Mhz, single port SRAM, a pseudo dual-port arrangement is achieved. This arrangement running at 80MHz is able to achieve unidirectional transfers at the maximum 50MHz supported by the FPGA devices, while batching requests up to twelve at a time can achieve 77% of the bandwidth of a true dual-port 50MHz SRAM under worst case (full load) conditions. This is a significant improvement over a synchronous (50MHz) single port configuration which would achieve only 48% of the bandwidth of the dual port configuration. This pseudo dual port RAM offers most of the benefit of true dual port RAM, whilst saving

the cost difference for a true dual port chip, and more importantly for the small form factor design desired, a large amount of FPGA IO pins and PCB real estate.

2.5.2 Determining Maximum Memory Clock

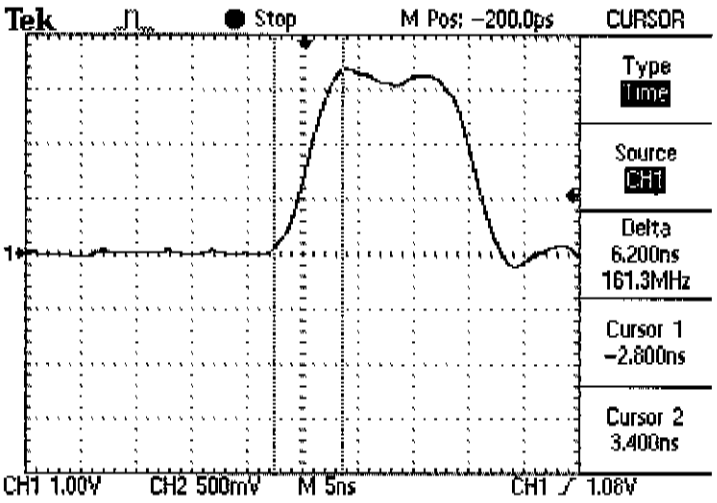


Figure 2-5 Oscilloscope output for FPGA to SRAM data pin 0

The maximum memory clock of an individual PCB can be estimated by examining the time required for the pins to change levels on an oscilloscope. The output of the oscilloscope for data pin 0 on the second PCB revision is shown in Figure 2-5. It can be observed that the measured rise time is 6.2ns. This corresponds to a frequency of 161MHz, twice the 80MHz max reliable transfer rate found for this board. Similarly, for the first PCB built the rise time was 6.0ns, which corresponds to a frequency of 167MHz, or twice the 83MHz clock speed which was the maximum stable transfer rate achieved with this board. These findings are summarized in Table 2-4.

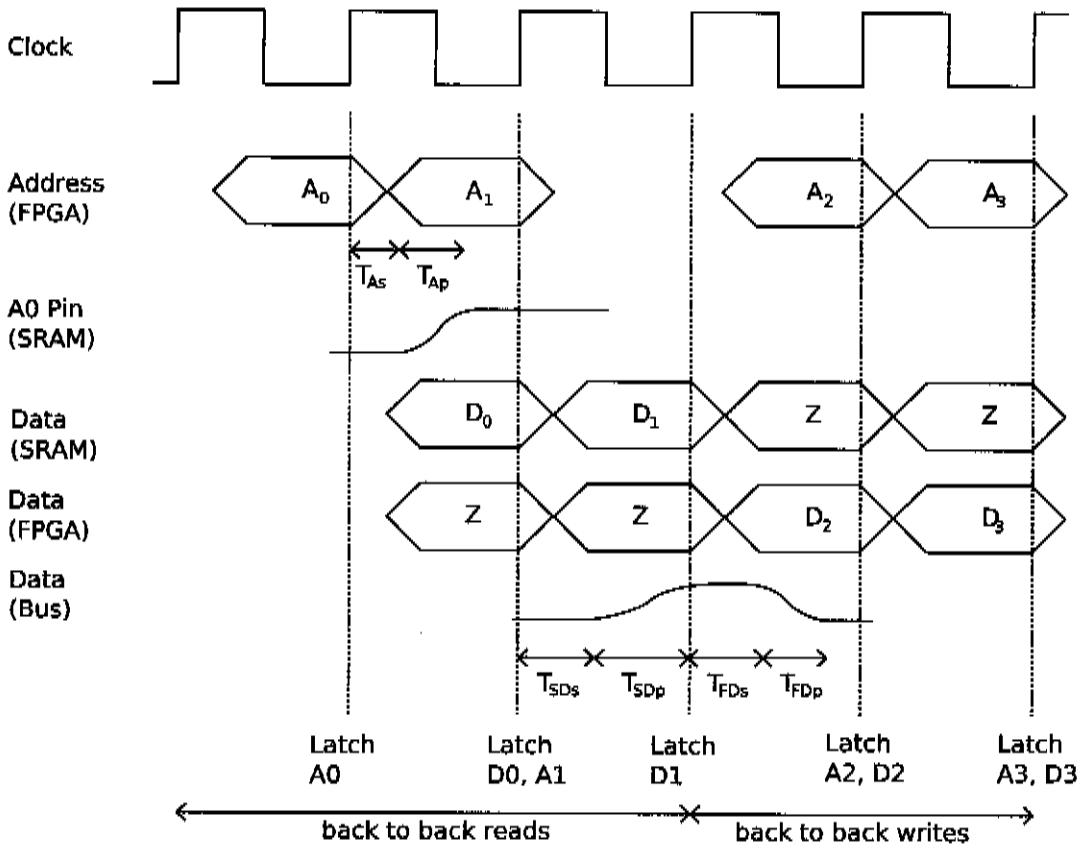


Figure 2-6 timing diagram for FPGA to SRAM transfers, back to back read and write modes

Period Variable	Description
T_{As}	Address set-up time
T_{Ap}	Address propagation time
T_{SDs}	SRAM output data set-up time
T_{SDp}	SRAM output data propagation time
T_{FDs}	FPGA output data set-up time
T_{FDp}	FPGA output data propagation time

Table 2-3 Time period variables for FPGA - SRAM bus timing

There are two reasons the clock speed achievable is highly PCB dependent. Since the drive current for the SRAM chip cannot be altered, the rise and fall time when the FPGA reads data from the SRAM (T_{SDp}) cannot be shortened by raising the current. By comparison, T_{FDp}, the equivalent transition time for the FPGA, can be shortened by raising the software configurable drive current as

necessary. The propagation of T_{SDp} is the major performance bottleneck therefore, and was the signal whose rise time showed a directly proportional relationship to minimum clock period. It is important to consider that the RAM is not Double Data Rate (DDR), and therefore the clock signal has to both rise and fall during every clock period, almost twice as fast as the address and data signals. Fortunately this signal is output by the FPGA and the drive current can be raised to achieve a sufficiently fast rise and fall time. It is necessary to raise the clock drive current from the default of 2mA to at least 8mA to achieve the speeds indicated on the current board.

PCB	Rise time (t_r)	Est. Frequency ($1/2t_r$)	Highest stable setting in testing (mult./div.)
First	6.0ns	81MHz	8/5 (80MHz)
Second	6.2ns	83MHz	5/3 (83MHz)

Table 2-4 Data pin 0 rise times and estimated / experimental maximum clock settings for the two PCBs tested

2.6 FPGA Camera Interface

The M6 platform has been designed so that either one or two COMedia C3038 cameras may be connected to the FPGA. If two cameras are connected, the FPGA can gather data from both simultaneously. In the standard setup each camera outputs a number of signals and the FPGA is merely a passive listener. It would be possible for the FPGA to provide a common pixel clock to both cameras if their on board clocks were disabled, but the implementation discussed here is for use with unmodified cameras.

For the FPGA to receive appropriate data from the cameras, it is necessary to first program the cameras settings via an I^2C bus. The details of this along with an outline of all the special considerations relating to multiple OV6630 chips on one I^2C bus are available[3].

Camera Output	Description
PCLK	Pixel Clock – Each rising edge of this clock during a line corresponds to one pixel being available on the bus.
HSYNC	This signal is held high for exactly the duration of each line. The falling and rising edges of this signal occur between each line, and pixels received while this signal is low are invalid. This signal remains low during vertical blanking periods.
VSYNC	This signal encodes the start of each frame as a pulse that is longer than the period of the pixel clock. A return to low occurs before the next line begins (HSYNC goes high.)
Y0:Y7	The low 8 bits of the data bus are fed into the FPGA. All pixel data is received via these lines.
UV0:UV7	The remainder of the 16 bit bus is not connected on the M6 platform. To receive complete data it is necessary to request an 8 bit mode from the camera (default is 16 bit.)

Table 2-5 Summary of relevant camera output pins

Since the maximum frequency of the cameras in 8 bit mode as utilised is approximately 18MHz[10], it is unreliable to attempt to observe the PCLK signal using sampling methods. It is therefore necessary to treat this as an FPGA clock signal, and latch the outer camera pixel data registers off this. This latched data is then fed to the M6CLK circuitry using a toggle input, for which sampling is reliable. A similar technique was necessary to observe the HCLK signal with reliable relative timing to the PCLK sampling. Although it is possible to apply this technique to VSYNC, doing so results in routing problems for Xilinx's XST synthesiser. During testing the result has been that if only PCLK and HSYNC are treated as clocks, as necessary for reliability, then all devices can coexist without causing XST's clock routing to fail.

Chapter 3

FPGA Convolution Implementation

3.1 2D Convolution

The multiplication of two functions in the frequency domain is equivalent to the convolution of the functions in the time domain, and vice versa:

$$C_{x,y} = F(u,v).G(u,v) \tag{3.1}$$

$$c_{x,y} = f * g(x,y) \tag{3.2}$$

$$= \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f(x+i, y+j).g(i,j) \tag{3.3}$$

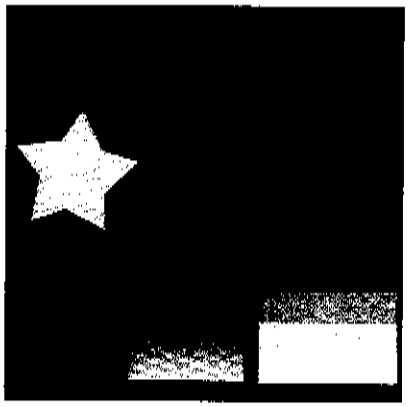
3.1.1 Windowed Convolution

Theoretically the calculation of the convolution value in time domain at a single (x,y) coordinate requires a sum over the entire overlapping range of the (post-translation for each step) functions. To compute the convolution for every value in an image or other data source with a mask of the same size as the image (as required for equivalence to an arbitrary frequency domain multiplication) requires an $O(N_1\hat{}2\ N_2\hat{}2)$ order calculation where N_1 and N_2 are the image dimensions. Defining M_1 and M_2 as the dimensions of the image N_1 and N_2 rounded up to the nearest powers of two respectively, the FFT and inverse FFT

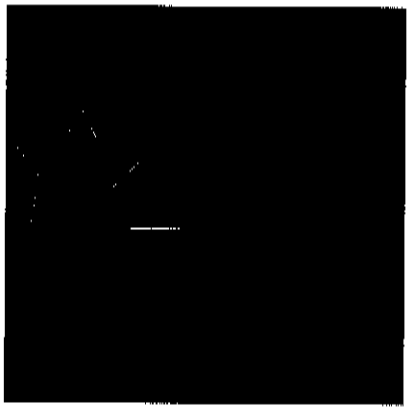
of the data can be calculated with a computational order of $O(M_1M_2.\log_2(M_1M_2))$. We can also ignore the bit reversals on the FFT and inverse FFT when performing convolution via this method[11]. Clearly the basic 2D convolution is not an efficient way of performing an arbitrary frequency domain multiplication.

The windowed convolution is an optimisation which does not maintain direct equivalence with a frequency domain multiplication. It works on the principle that a function which consists entirely of zeros outside a small range of values can be computed in a much shorter time than a full convolution. The zero values themselves and the source values which multiply with the zeros in the source image do not need to be considered at each step. Therefore a windowed convolution can be computed in a complexity that is only $O(N_1N_2.W_1W_2)$ where W_1 and W_2 are the dimensions of the window. Assuming the best case for the FFT where $N_1 = M_1$ and $N_2 = M_2$, if W_1W_2 is smaller than $\log(M_1M_2)$ then the windowed convolution will still have a lower complexity. In practice, the window size at which the windowed convolution gains the complexity advantage is much smaller, due to its more efficient handling of non-power of two sizes. Another big advantage for an FPGA implementation is the relative simplicity of the operations in the convolution, compared to the FFT, multiply and inverse FFT alternative. This advantage is important because the implementation must be hardwired, rather than implemented in software, and therefore the logic cannot easily be reused for another task when the convolution is not taking place.

3.1.2 Windowed Convolution Examples

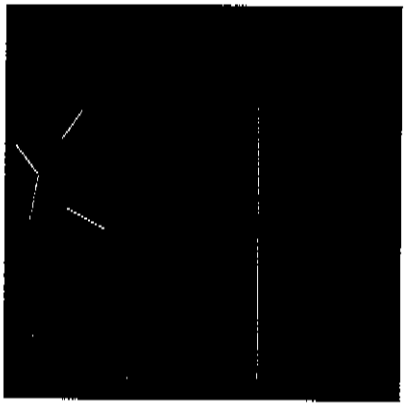


Original Image



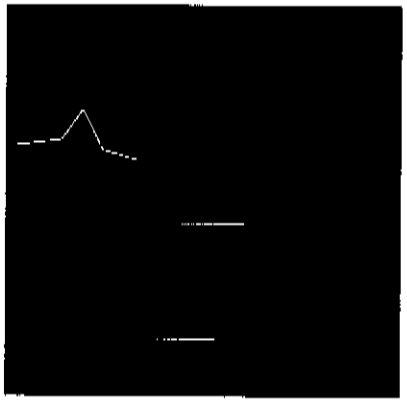
-1	-1	-1
-1	8	-1
-1	-1	-1

Laplace Edge Detector



-1	0	1
-2	0	2
-1	0	1

Sobel Horizontal Edge Detector



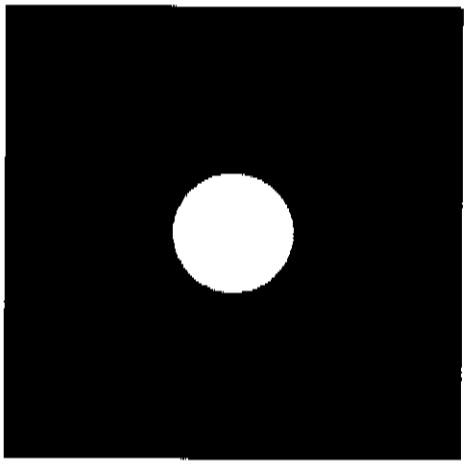
-1	-2	-1
0	0	0
1	2	1

Sobel Vertical Edge Detector

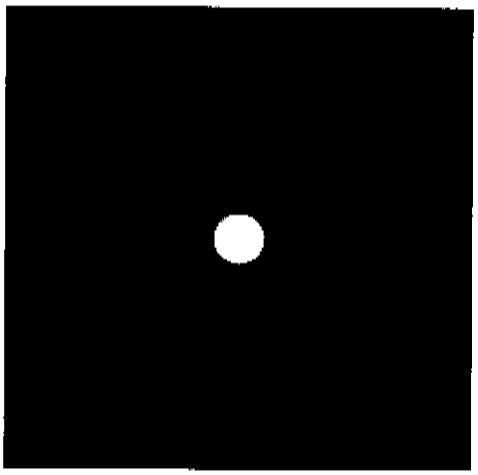
Figure 3-1 Examples of edge detectors using 3x3 windowed convolutions. Source image reproduced courtesy of Lixin Chin.

Several functions have been developed which exactly fit the windowed convolution model. Examples include the Sobel and Laplacian edge

detectors(Figure 3-1), which operate a little like directional and omni-directional high pass filters respectively, but locked closely to a specific frequency range such that they tend to retain features with a dimension of a single pixel in the appropriate directions. Although only a limited number of candidate functions can be expressed exactly like this, many can be approximated adequately. A particularly common and conceptually simple example is the Gaussian low pass filter. The inverse Fourier transform (time domain) version of a 2D Gaussian function is itself a 2D Gaussian function. Gaussian functions in general tend towards zero rapidly, to a few decimal digits of precision, beyond a similar number of standard deviations away from the centre point(Figure 3-2). Therefore a very small window is often appropriate for approximating this function. It is also extremely useful in image processing, as it can be used to reduce both texture and noise from a source (Figure 3-3). This noise reducing property will be applied in Section 5.2.



Gaussian low-pass filter (frequency domain)



Inverse 2D FFT of Gaussian low-pass filter (spatial domain.)

Figure 3-2 2D Gaussian filter in frequency (left) and spatial (right) domains

0.018	0.049	0.049	0.018
0.049	0.134	0.134	0.049
0.049	0.134	0.134	0.049
0.018	0.049	0.049	0.018

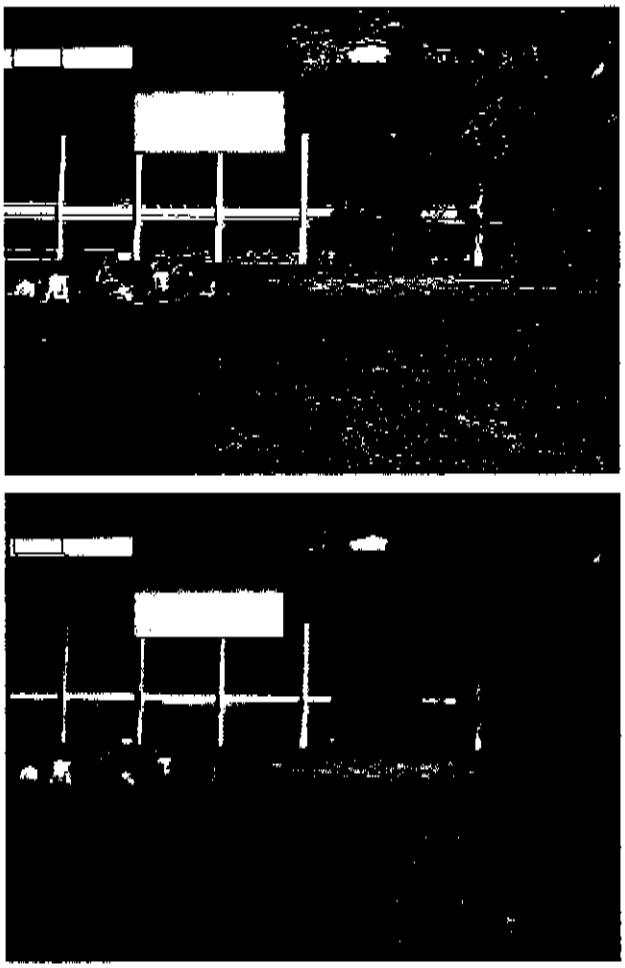


Figure 3-3 Noisy source image (above right) and resultant image with reduced noise and texture (right) after application of 4x4 windowed convolution Gaussian filter (above). An enlarged region (red box) helps identify the difference in noise.

3.2 FPGA Implementation

Implementing algorithms in an FPGA is very different to writing software code to perform the equivalent task. Although it is possible in theory to design an FPGA layout which executes software code similar to a CPU, this would be extremely inefficient compared to the real CPU. The strength of the FPGA lies in the ability of the designer to construct logic that is highly specialised to a particular task. An FPGA circuit necessarily operates at a slower clock rate compared to a hard-wired circuit implemented in the same technology, but the FPGA can be hardwired to perform many steps of a repetitive task

simultaneously, and gain a performance advantage through parallelism. The advantage of doing this with an FPGA over a regular ASIC or CMOS design is the ability to reprogram a new layout, allowing one chip to perform multiple tasks, minimizing low volume ($<$ 100,000 unit) design and production costs, and maximising flexibility.

From an FPGA's perspective, the most difficult aspect of the convolution algorithm is the multiplication. For each output pixel, for which the total number is similar to the number of input pixels, The number of multiplications is W_1W_2, where W_1 and W_2 are the window width and height respectively. Each single multiplier would take up an enormous portion of the FPGA logic, and operate at a slow speed. Fortunately to solve this, Xilinx provide 20 dedicated multipliers on the Spartan3E chip utilized. These multipliers are a precious resource, and since they can perform a multiplication every clock cycle for a 50MHz system, a circuit for calculating convolutions using only a single multiplier, and taking W_1W_2 clock cycles per multiplication was devised. The initial implementation uses a window size of 4x4, as memory performance suffers dramatically if the number of block RAMs in the hardwired pre-fetch cache is not at least equal to the number of lines of the window plus one divided by 5, rounding up. Since both block RAMs and multipliers exist in the same quantity on most Xilinx chip, this symmetry in utilisation for each convolution operator makes sense. Convolution can be speeded up further by adding multiple convolutions in parallel on different image locations. Convolutions of larger window sizes can be achieved by summing the results of smaller convolutions. Combining these techniques reduces the variability in convolution implementations necessary.

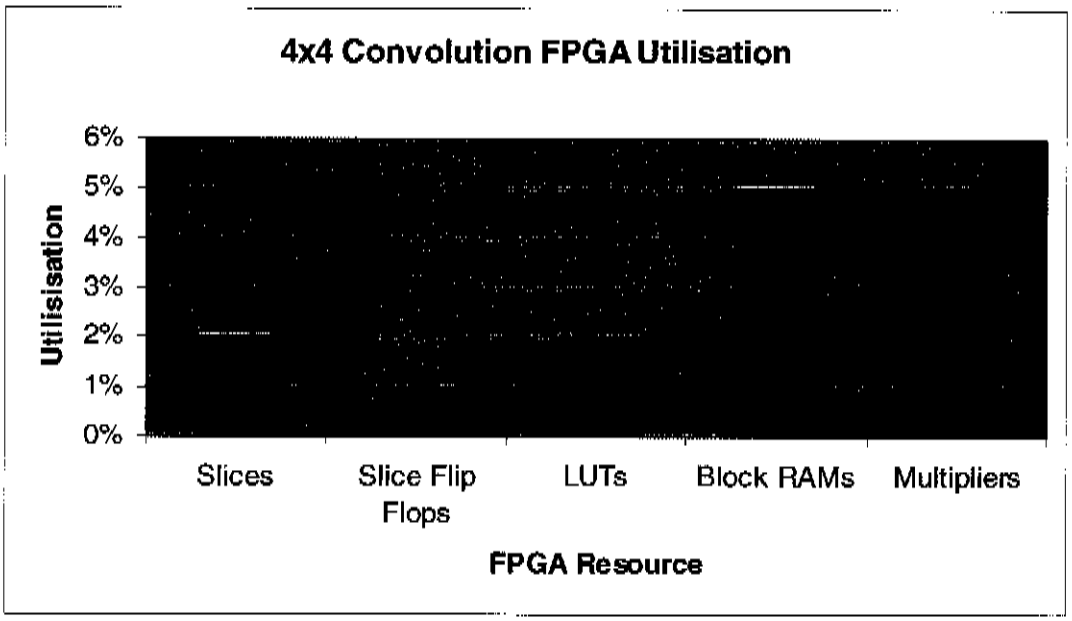


Figure 3-4 FPGA resource utilisation for 4x4 convolution block.

One difficulty with implementing an algorithm in an FPGA context is the inability to run the chip in any kind of debug mode. Combined with the complexity of adding simple debugging IO to the design, locating bugs becomes extremely time consuming. To address this, special code was written to the Xilinx VHDL logic and timing level simulators to external C++ applications. This code uses only standard VHDL and C++ libraries, and so should work with any development simulator/programming environment combination. In addition, a GUI for analysis the convolution algorithm under different conditions was implemented.

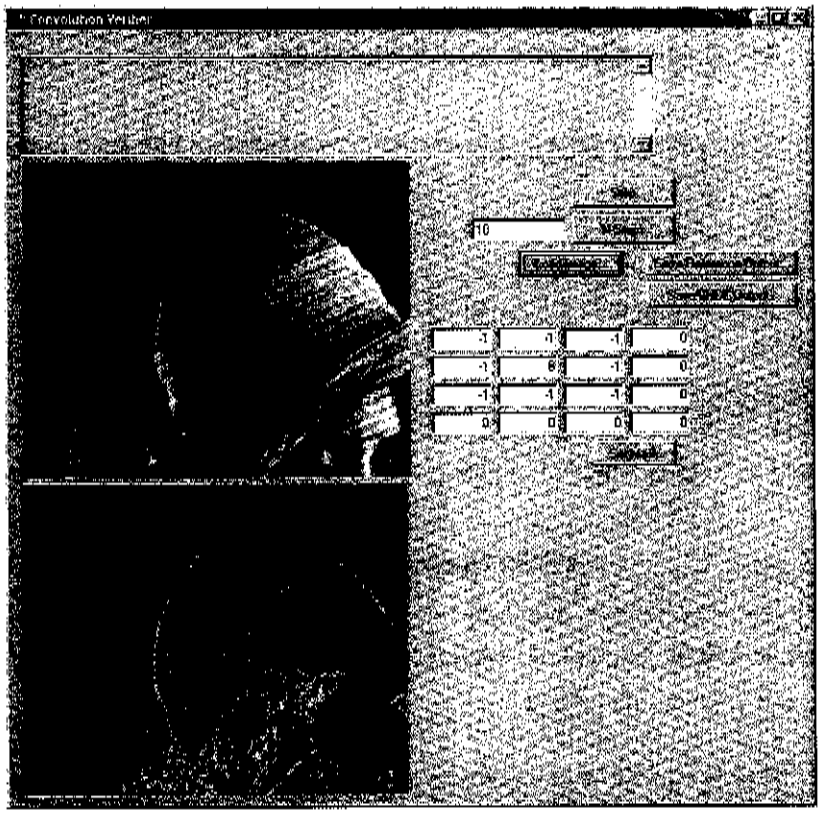


Figure 3-5 C++ test harness with .NET UI.

3.2.1 Performance Results

The performance results for the convolution algorithm are quite predictable. As work on the FPGA peripherals could not begin until the PCB was completed, time did not allow for the testing of convolution algorithms on the real FPGA. Still the hard-wired logic has predictable performance characteristics, from which estimates were derived(Figure 2-5).

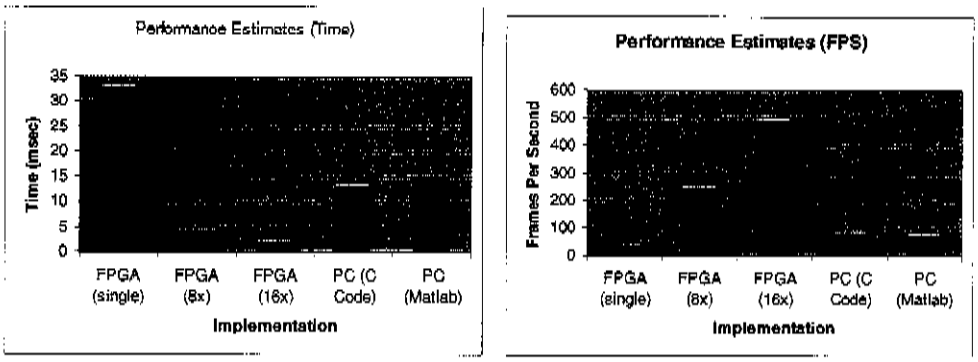


Figure 3-6 Performance estimates for full real-time convolution system running on FPGA, with comparison to PC performance (Reference algorithm running on a 1.83GHz Intel Core Duo, binaries built with Microsoft Visual Studio 8.

Chapter 4

Geometry of Depth Perception

The Eyebot M6 platform has a standard arrangement of two cameras facing the same direction, spaced a small distance apart. This arrangement, similar to the human visual system, gives the platform stereo-vision. The key benefit of this arrangement is the potential ability to perceive depth. The determination of depth from multiple, two-dimensional images can often be achieved through comparison of the varying relative locations of points visible in both images. The initial steps in the development of an implementation of this technology on the Eyebot platform will be discussed.

Stereo-vision algorithms require large amounts of computation to achieve good results. It is not expected that a satisfactory real-time, general purpose stereo vision system can be implemented using the general purpose CPU of the new eyebot platform. Instead the focus of this discussion will be the selection and optimisation of algorithms in such a way as to fit an implementation on the platform's Spartan 3E FPGA. Consideration will be given to the utilisation level of the specific resources available on this FPGA at every stage. The goal is to derive a stereo algorithm which can be implemented so as to provide several frames of depth information per second, whilst offering the maximum information possible.

4.1 Projective Geometry

The first step in deriving a stereo algorithm is to consider the geometry involved, and a method of relating the 2D image data to the true 3D scene. To do this, the geometry of the camera is considered to be like that of a pin-hole camera (cameras that introduce no significant lens distortion to the image.) Understanding this simplified relationship mathematically begins by defining the projection matrix K which converts between a coordinate (u,v) in 2D image space and a coordinate (x,y,z) in 3D space. The explanation which follows has in part been adapted from [12]. Equation (4.1) shows the construction of a matrix from the following parameters:

f	:	focal length
(uo,vo)	:	intersection of Z axis with image plane

The geometry for this situation is illustrated in Figure 4-1.

Note that square pixels, as found almost exclusively on modern cameras, have been assumed here. For rectangular pixels with different widths and heights two separate focal lengths fu and fv would be required.

$$K = \begin{bmatrix} f & 0 & u_0 & 0 \\ 0 & f & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{4.1}$$

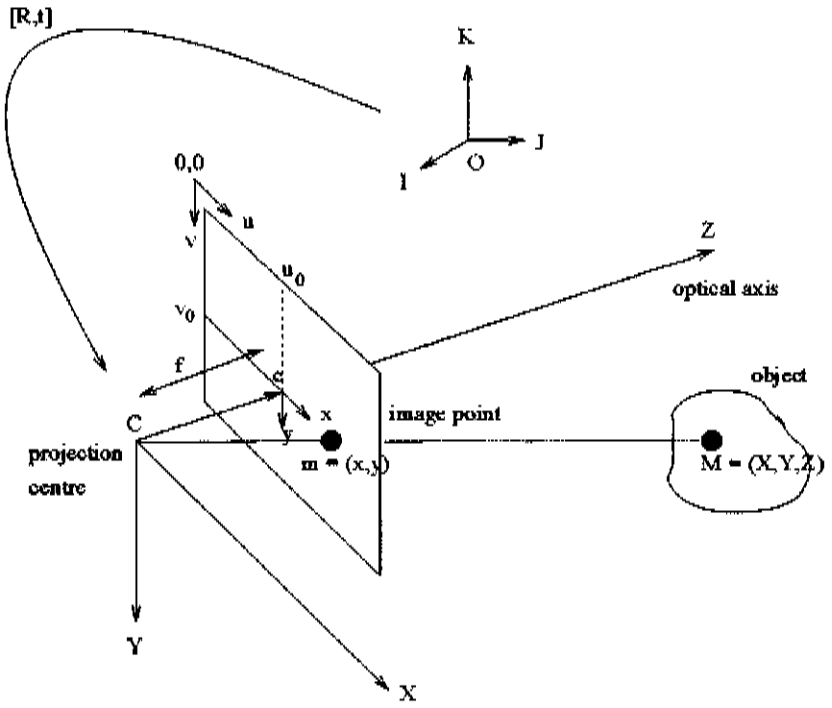


Figure 4-1 Basic projective geometry[12]

4.2 Full Pinhole Camera Geometry

The projection matrix K requires the camera to be at a fixed position in world space, which is obviously incompatible with the notion of two cameras viewing the same scene. To generalize to multiple cameras, a calibration matrix C can be constructed which includes an arbitrary transformation of the camera relative to world coordinates. The additional parameters for this matrix, shown in equation (4.2) are:

R	:	3x3 rotation matrix
t	:	3x1 translation vector

$$T = \begin{bmatrix} t & R \\ 0_3^t & 1 \end{bmatrix} \tag{4.2}$$

$$C = KT \tag{4.3}$$

This relationship for a pair of cameras viewing the same 3D scene is expressed in equation (4.4).

$$C_1\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} s_1 u_1 \\ s_1 v_1 \\ s_1 \end{bmatrix} \qquad C_2\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} s_2 u_2 \\ s_2 v_2 \\ s_2 \end{bmatrix} \tag{4.4}$$

4.3 Epipolar Geometry

The geometry for two pinhole cameras as derived in 4.2 is illustrated in Figure 4-1. There are two crucial points to be made at this stage. The first is that, for each point seen by one camera, there are infinitely many object positions which share the relationship so that the lines connecting them to the camera's projection centre overlap, and therefore share the same image coordinate. The second is that all of these object positions project to different points in the second image (ignoring the lack of non-discrete pixel locations), and all of these points form a single line on which lies the epipole (The point where the baseline intersects the image plane.) This is the well known epipolar constraint that reduces depth searching from a 2D to 1D problem.

4.4 Co-planar Images

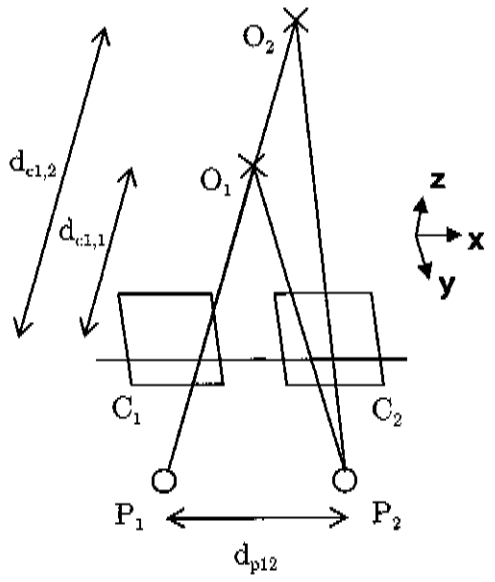
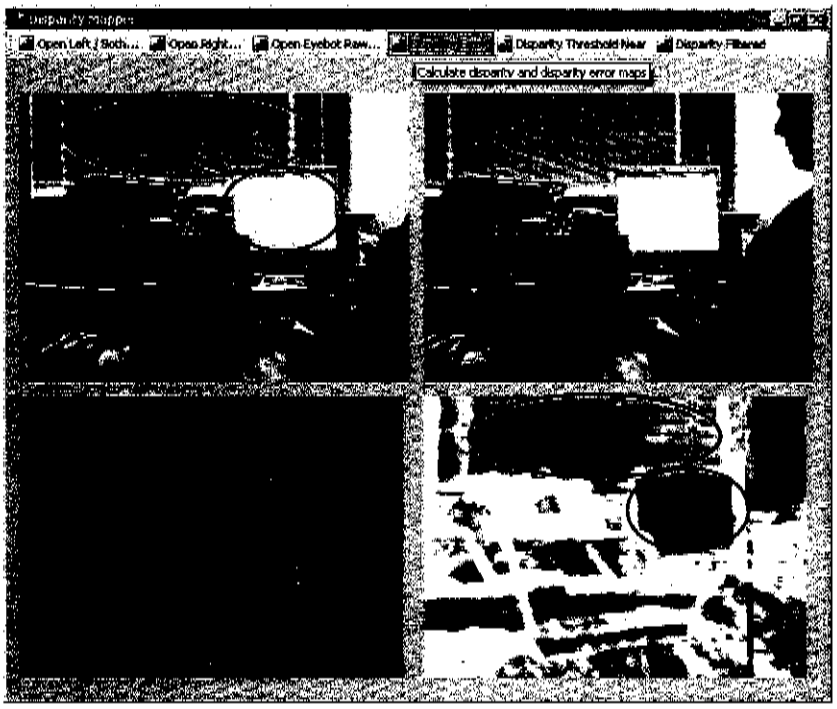


Figure 4-2 Geometry for a pair of cameras aligned to give parallel image planes

If two pinhole cameras are lined up such that they share a common image plane, then the epipoles will now be at infinity. Since they only meet at infinity, all epipolar lines will be parallel and horizontal[13]. The projective geometry for this situation is illustrated in Figure 4-2. If the x and y axis are parallel to the image planes of the cameras, objects O1 and O2 are spaced in between the two cameras on the x-axis or have the identical y coordinates, and both objects project to the same coordinate in the left hand camera's image, then the corresponding projected points in the right image will have the same y coordinate as in the left, and therefore be on the same scanline.

4.5 Practical Rectification

Disparity mapping algorithms, as discussed in Chapter 5, calculate disparity by searching image sets for matching regions and calculating distances between them. It will be shown that, if it is possible, using the techniques discussed in this chapter, to constrain all equivalent points to lie on epi-polar lines that are parallel in the source images, then it will be possible to compute high quality disparity maps in real-time on the M6 platform. This becomes possible because the search space for disparity maps is reduced substantially, and the use of rectangular windows which can have their SAD computation optimised, rather than circular windows, is possible.



Repetitive

Plain

Table 4-1 Stereo source images captured with a single camera moved by hand, along with SAD disparity map and minimum window error. It can be seen that the SAD algorithm, as discussed later, does not require perfect alignment. In particular many of the regions with poor accuracy are due to deficiencies in the SAD algorithm irrespective of rectification, which will be covered in more detail in the later chapters.

Two cameras that look similar physically (same make, model) may not, when their casings are lined up side by side facing the same direction, give a good approximation of co-planar images. Large stereo systems typically employ a bi-prism setup[14], where a single sensor is used to capture the image data, and a special lens system with a prism focuses views from two disparate positions onto this sensor. The bi-prism system ensures that the stereo image pair will be co-planar regardless of the small variation in the position and orientation of the capture surface in different cameras. The optical portions of the camera may introduce a rotation with unacceptable pitch and yaw components. In our testing of three Cameras of identical make and model (OV6630) we found that two had only a roll and translation as the significant misalignment components, while the third had pitch and yaw significant enough to prevent the affine

transform from recovering a suitably rectified pair. Therefore it is necessary to be cautious when assuming that a simpler transformation will be sufficient based off the external geometry of the cameras utilised.

4.5.1 Affine Transform Rectification

If we restrict ourselves to roll rotations, x translations and y translations, then the transform required on each coordinate in the second image relative to the first is independent of the depth. A single transformation can then be found and applied to the second image only. This transformation will remain constant between frames as long as the relative alignment of the cameras is unchanged.

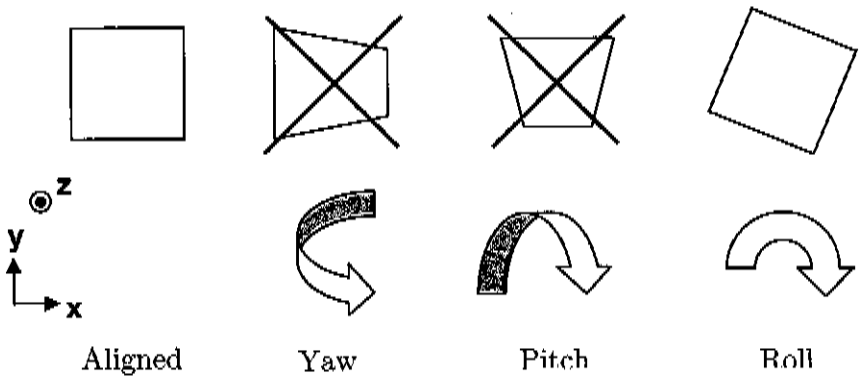


Figure 4-3 A comparison of the different components of relative rotation for two camera images, including the depth skew from the yaw and pitch components.

This transformation would be at most half as computationally complex as the full calibration method. If the roll rotation is within a few degrees then it is possible to allocate sufficient block RAM within the FPGA to only require reading each pixel from the SRAM once during the transform. In this case the impact of the transformation on the performance of other components is greatly reduced.

It is likely that, with further testing, it can be demonstrated that the Eyebot M6 can give useable quality depth information from its stereo cameras using

only this level of rectification. The yaw and pitch will not be significant with most camera pairs as the physical geometry always aligns them correctly externally. This would make a real-time (multiple frames per second) implementation feasible.

4.5.2 General Stereo Rectification

There are four steps required for rectification with complete generality with respect to camera position[15].

- Rotate the left camera so that the epipole goes to infinity along the horizontal axis (i.e. the left image plane becomes parallel to the baseline of the system).
- Apply the same rotation to the right camera to recover the original geometry.
- Rotate the right camera by R.
- Adjust the scale in both camera reference frames.

There are two problems with supporting this general case on an embedded platform. The first is the complexity of the full transformation, which is significant not only because it requires extremely high performance multiplication, but also because it has unpredictable memory access characteristics which do not fit well with the FPGA user designed cacheing model. If this level of rectification is required, it may be necessary to involve the CPU, which will almost certainly bring performance down to below one frame per second.

Chapter 5

Real Time Disparity Mapping

5.1 Disparity Mapping Methods

It has been established that the geometry of stereo vision lends itself to the estimation of depth through observing the varying relative location (disparity) of common points in a set of images simultaneously viewing the same scene. The most significant challenge in stereo vision is the analysis of the image set (pair in the case of the M6 platform which has two cameras) in order to ascertain the location of common points. Algorithms which attempt this task are called disparity mapping algorithms, since their result is a set of data which identifies the disparity as a function of position in one or more of the images.

There are two fundamental classes of disparity mapping algorithms[16], distinguished by their feature detection being either area based or feature based. Area based algorithms identify a local window surrounding each point for which they wish to ascertain a disparity, and compare this to similarly sized windows at all possible corresponding points in the other images. Generally a metric is used to evaluate the difference of the windows compared, and a local minimum of this difference is selected as the most likely match. In this way, area based algorithms produce a disparity map which provides a result for all, or at least most, of the points in the original images. This property means they are said to produce "dense" disparity maps. The alternative, feature based, detection methods look for a small set of particularly distinctive features, and then

attempt to pair as many of these as possible with the distinctive features present in the rest of the image set. This class of algorithms provide sparse disparity data since they typically attempt to determine a best match for only a small minority of the original image pixels. In a software implementation, feature based methods tend to be faster and more reliable, but offer much less information than area based methods. By contrast, area based methods offer a much greater potential pool of information. Another advantage of area based methods is their highly repetitive and parallel nature, which it will be shown makes them amenable to high performance FPGA implementation, important for achieving real time mapping rates from the M6 platform.

There are a number of properties which can be varied to arrive at a particular area based disparity technique. This makes for an enormous set of potential combinations from which an algorithm must be chosen. Since these potential combinations are almost endless, it is not possible to evaluate all of them. Further, the ideal set of parameters depends on the objects viewed, depths involved, and camera properties. Instead of attempting to optimise globally, a variety of conditions will be tried, and assumptions of generality made to arrive at a recommendation on how to narrow the search space of candidate algorithms for an FPGA implementation. Finally a recommendation for the M6 platform will be made, based not only on these quality tests, but an estimate of performance and resource usage derived from experience implementing other features.

5.2 Pre-Processing Filters

Typically when calculating disparity maps with the SAD metric, a "Laplacian of Gaussian" (LoG) filter is first applied to each source image by convolution. The Gaussian portion of the filter reduces noise, and the Laplacian acts as a feature analyser to reduce the complexity of the input data to its most significant signal. This can reduce the computation required to complete the SAD

comparisons, by allowing a logarithmic conversion of the output data down to a much smaller word size, without sacrificing a great deal of quality[17].

0	0	-1	0	0
0	-1	-2	-1	0
-1	-2	16	-2	-1
0	-1	-2	-1	0
0	0	-1	0	0

Figure 5-1 Example discrete 5x5 Laplacian of Gaussian (LoG) filter[18]

Although the LoG filter technique makes a great deal of sense when faster processing allows the addition of more cameras to the system for greater final quality, it makes little sense in our system which is limited to only two cameras. Further, the LoG algorithm would require additional hardware to compute, counter-acting its SAD calculation benefits on the FPGA resources. In summary this algorithm would would result in slightly faster computation through reduced memory bandwidth requirements, especially for the SAD previous line lookup table discussed in 5.5.3, but would not save resources and would drastically lower the final quality.

Pre-processing	Count	Sum	Sum Squared
None	28990	1176288	99348992
Discretised 5x5 LoG	33527	1926640	213116672
Gaussian 5x5 ($\sigma = 1$)	26611	1057024	81472512

Table 5-1 Error compared to ground truth[19] for Tsukuba image set using various pre-processing methods followed by a 7x7 SAD disparity map

Another option is to apply only a small Gaussian filter, exchanging feature detail for reduced noise in the source images. This pre-processing method can slightly increase the quality of SAD disparity maps, at the expense of an extra pre-processing step. Now both FPGA resource utilisation and computation time have increased, but quality is superior. This technique has merits in certain situations, but another method will be realised in section 6.5.2 which can remove a other, potentially bigger types of errors with much lower FPGA resource requirements, and no additional time. This technique relies on the presence of the very errors that the Gaussian filter helps remove in order to locate much

bigger sources of errors. Therefore it is recommended that the images used for disparity mapping do not undergo pre-processing, unless they suffer from greater than usual amounts of camera sensor noise.

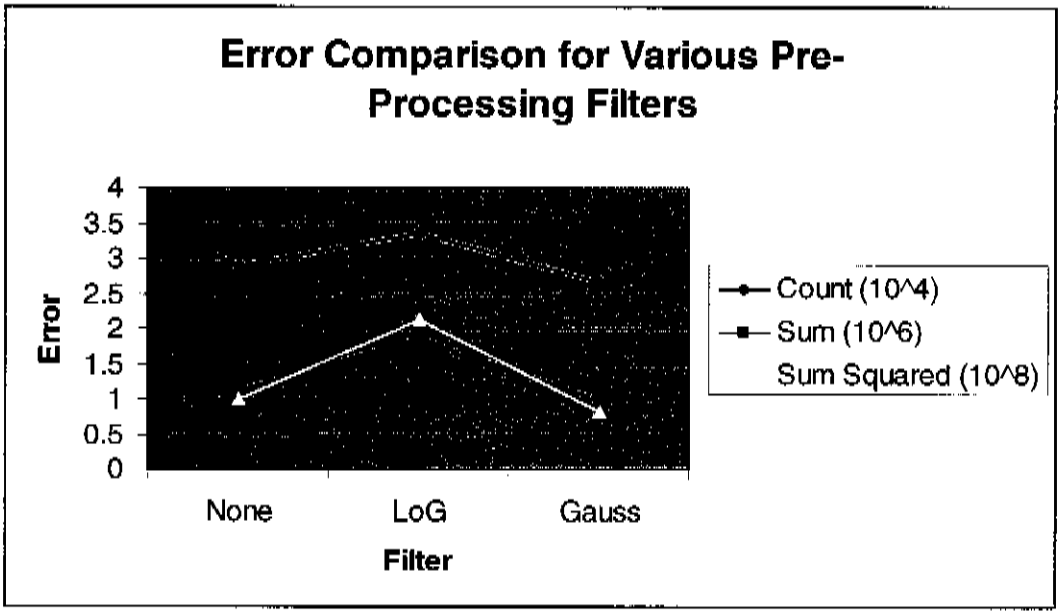


Figure 5-2 Graphical summary of data from Table 5-1.

5.3 Choice of Match Comparison Metric

Area based disparity map algorithms invariably require some type of similarity metric to determine which pairs of windows make the best matches. A number of these have been proposed in the past, including Sum of Absolute Differences (SAD), Sum of Squared Absolute Distances (SSAD), Cencus and a class of algorithms referred to as Rank (non-parametric algorithms that rely on ordering). Comparisons have been done in the past, with for instance a comparison of the first three finding that SAD produced equal best quality with SSAD, and Cencus producing the lowest quality[20]. Another more detailed study including Rank algorithms found that SAD was the best as long as a

zero-mean condition was applied[21]. This seems unnecessary in testing done for this thesis and it is suggested that this condition may be necessary for cameras with incorrect gamma balance or other serious relative colour problems. It has been suggested that Rank style ordinal methods can also be more effective in cancelling the effects of difficult geometry or gamma variation errors to the point where SAD is always inferior under non-ideal conditions[22, 23]. Whether this proves a genuine advantage of the SAD algorithm in the case of the Eyebots is an area that requires further investigation with real Eyebot setups.

An approximation of the resource requirements for different match metrics no FPGAs has been done previously[24]. According to this study, the FPGA resource requirements for the SAD algorithm are a close second lowest, behind the Rank method. SSD and Cencus are significantly more expensive.

Overall, the SAD algorithm, seemingly traditionally popular in real-time implementations[17], seems both the best and safest choice for implementation. A rank based algorithm may be effective as an alternative for scenarios where the SAD algorithm is not sufficiently accurate. Cencus and SSD are more expensive than SAD and offer little or no quality gain, and therefore should be avoided. Based on this, the SAD algorithm is used as the basis for the rest of this analysis.

$$SAD_{x,y}(v_x, v_y) = \sum_i \sum_j I_1(x+i, y+j) I_2(x+v_x+i, y+v_y+j) \qquad (5.1)$$

5.4 Selecting Window Size

One property that can be varied for all area based disparity algorithms is the size and shape of the window to use for each comparison. The two obvious

choices of window geometry are a square and a circle. The circular window has the advantage of being directionless, a useful property when the camera geometry is unknown. The square window is similarly effective where the geometry is known and the images can be rectified (see Section 4.5.2), and is much simpler to calculate, especially since it is more amenable to optimisation as covered in the next section.

Since the camera geometry is fixed on the M6 platform and efficiency is paramount to enable real-time analysis, a square window should be employed. The remaining question is what size window to use. The quality of the basic SAD algorithm performed on a number of window sizes is summarised below:

Window Size	Count of bad pixels	Disparity Error (Sum)	Error (Sum of Squares)
3x3	40777	2206912	235035648
5x5	32757	1452240	133423872
7x7	28990	1176288	99348992
9x9	26733	1024000	80260608
11x11	25479	956016	72347392
13x13	24600	926112	69288448
15x15	24084	911648	66949120
17x17	23811	904800	65426432
19x19	23606	909280	65165824
21x21	23367	919264	66016768

Table 5-2 Results of comparing the effectiveness of different window sizes on calculating the depth of the Tsukuba image pair as compared to the supplied ground truth[19].

There is a clear trend for quality (inverse of errors) as a function of window size. For both the sum and sum of squared error metrics the result drops with increasing window size down to a minimum point, and then increases again after that. The total count of pixels with errors continues to drop right up to the last window size tested, but is expected to reach a similar minimum soon after. Therefore to minimize initial error a large window size (17x17, 19x19 or 21x21)

is most appropriate. In percentage terms however, the increase from a 3x3 to 5x5 reduces both sum and sum squared errors by 35% and 45% respectively, whereas the next step to a 7x7 window only yields reductions of a 13% and 26%. For the count of errors the improvements are even smaller.

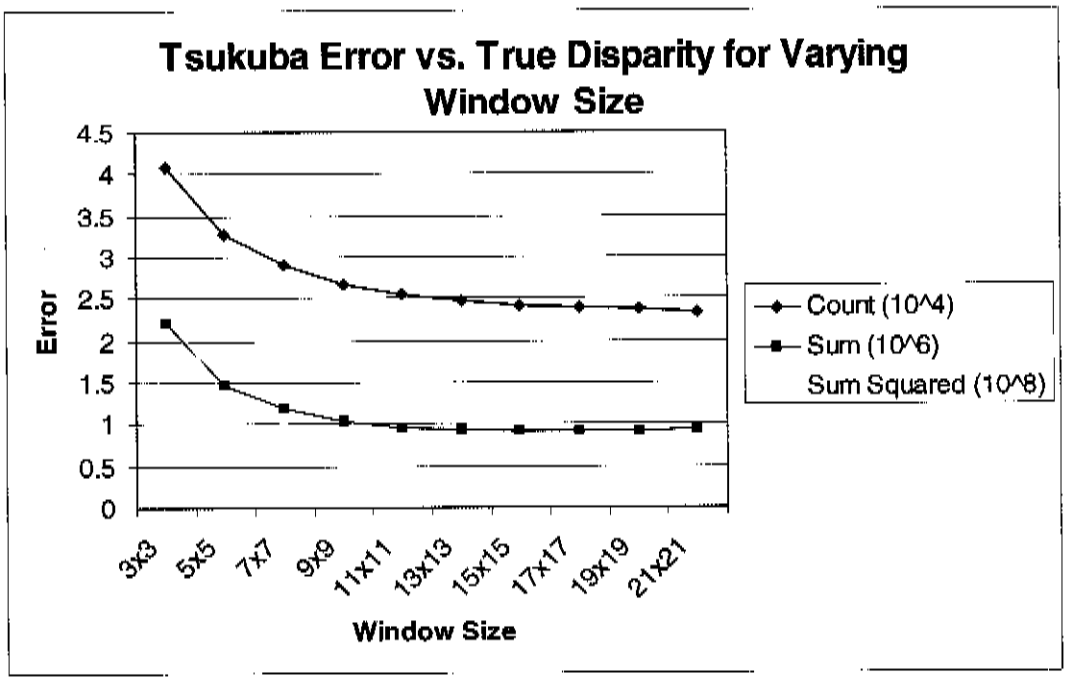


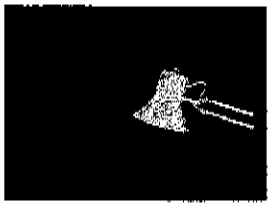
Figure 5-3 Disparity Error for various window sizes using the plain SAD disparity method

Computational complexity of the optimised SAD algorithm grows linearly with the size of a single side of the SAD window. When real-time performance is desired, window size has to be minimized. A window size of between 7x7 and 13x13 therefore seems a more logical choice. In the case of the FPGA, there is another factor to consider. Although the total amortized clock cycles per window required does not grow with the size of the window, the hardware resources required and minimum time period of each clock cycle do. There is a SAD calculator required in numbers which grow linearly with the side length of the window. It will be difficult to fit 26 of these into the M6 platform's FPGA alongside the other logic required to implement stereo vision. Assuming 352x288 source data (the resolution of the M6 platform's camera), the are n/4 Block RAM resource required, where n is the number of lines of the window plus one.

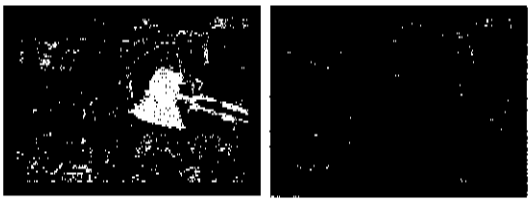
If this much Block RAM is not available, appropriate caching is not possible and performance will degrade dramatically due to memory bus bottlenecks. there is also a summation circuit, the length of which grows base two logarithmically with the window side length. Table 5-3 summarises these requirements. We can see that the 7x7 window requires dramatically less resources and cycle times compared to the 9x9, with only a small additional error introduced. Therefore, in the absence of evidence a larger size can co-exist alongside rectification, camera input, memory controller and other required circuitry, this is the recommended starting point for a real-time implementation. The 11x11 window also offers a good balance between resource usage and quality, but may prove too expensive for real-time implementation on the M6 platform in a complete system context.

			Error (increase cf. 15x15)		
Side Length	Block RAMs	Tree Depth	Count	Sum	Sum Squared
5	2	3	36%	59%	99%
7	2	3	20%	29%	48%
9	3	4	11%	12%	20%
11	3	4	6%	5%	8%
13	4	4	2%	2%	3%
15	4	4	0	0	0

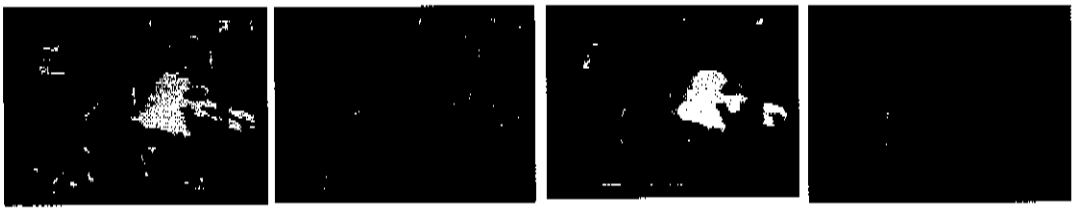
Table 5-3 Comparison of selected FPGA resource / timing requirements and errors incurred for different SAD window side lengths



Ground Truth



3x3 Window Disparity / Error



7x7 Window Disparity / Error 13x13 Window Disparity / Error

Figure 5-4 Disparity maps and error relative to ground truth[19, 25] for basic disparity algorithm applied to Tsukuba image set at selected window sizes.

5.5 Performance Optimisation of SAD Algorithm

5.5.1 Minimisation of Search Area

The most basic optimisation to make in calculating the SAD algorithm is to limit the disparity search to a range that covers all viewable source depths. If rectification ensures that epi-polar lines are near horizontal, and points of infinite distance are identically positioned in the two images, then the search width required is related to the camera parameters and the closeness of objects which must be viewed.

For the Tsukuba image set, the maximum disparity in the ground truth corresponds to a separation of 15 pixels. Although maximum performance could be obtained by limiting search width to this length, this is not realistic as a larger search width (as required for closer objects) will degrade quality (greater uncertainty about location of matching point.) To keep results more realistic to the Eyebot case where detection is desired even when objects are close, a search width of 20 was used when performing error evaluations. Although a larger still search width could be used, it is not necessary to include an extremely large value for disparity evaluation as localisation of similar objects means the likelihood of a mistaken match being caused by a wider search width diminishes

with distance. For performance evaluations presented at the end of this section, a search width of 32, which covers objects very close to the viewer is used instead, to demonstrate the real-time expectations for the system under a wide variety of conditions. In general this value should be selected based on the minimum distance of object desired to be searched form, with minimized width for quality reasons only a secondary concern.

5.5.2 Multiple Resolution Search

One method is to approximate the SAD search by performing initial searches at lower resolutions[26], and performing a detailed search of the remaining region. This method will not be considered for pixel level precision disparity map computation. The main advantage of this method occurs when the search must be performed on a 2D area rather than a 1D epi-polar constrained search. This will not normally be the case for the Eyebot M6 platform where the relative camera positions are fixed, and calibration is possible. This optimisation also has a number of deficiencies from an FPGA implementation perspective. Since the number of different operations in the algorithm has increased, the logic required is substantially greater, and therefore the resource usage and power requirements increase. In addition, the final result suffers from lower accuracy, owing to the stepwise minimum search which may miss the global minimum. Therefore this is an inappropriate choice in an FPGA implementation context unless the camera geometry is not known.

5.5.3 Reusing Earlier Results as a Lookup Table

One method of improving the accuracy of the SAD algorithm without a reduction in accuracy is to reuse earlier values. References to this technique have been found in past attempted real-time implementations[27] although the

algorithm is not explained in detail. The technique exploits the fact that in comparing two windows, the previous . This relationship is explained in equation (5.2), where Vx represents the disparity offset being tested, x and y represent the centre of the window on the first image, Total is the sum of absolute differences total for the window, and $P_1(x,y)$ and $P_2(x,y)$ are the pixel value at the specified locations for the first and second images respectively.

$$Total_{x,y}(V_x) = Total_{x,y-1}(V_x) - \sum_{i=-M}^{M} SAD_{x+i,y-4}(V_x) + \sum_{i=-M}^{M} SAD_{x+i,y+3}(V_x) \tag{5.2}$$

$$M = \frac{Size_{Window} - 1}{2} \tag{5.3}$$

$$SAD_{x,y}(V_x) = \left|P_1(x,y) - P_2(x+V_x,y)\right| \tag{5.4}$$

5.6 FPGA Performance Estimates

The challenge of implementing real-time stereo vision utilising dedicated logic is quite different to that faced in software. Whereas software writers tend to worry about the total number of calculations, an FPGA designer is more concerned with the diversity of logic and throughput required of different memory sub-systems systems. The FPGA is excellent for simple, repetitive, parallelisable tasks where the high bandwidth of the multiple parallel block RAM caches can be utilised. When it is hampered by external memory bandwidth, the FPGA may be no more efficient and significantly more complicated as a solution than a conventional processor. Where the task requires many complicated operations the FPGA may even be inferior.

Fortunately, after optimization, the SAD algorithm is an excellent candidate for FPGA implementation. The level of exploitable parallelism is above what a SIMD processor can supply, but achievable by the FPGA. The FPGA implementation would also have a near constant turn around time, the ideal

case for real-time systems. A reference PC implementation was obtained by performing the optimized SAD calculation a number of ways, compiling with the latest Microsoft C++ Compiler (Visual C 8.0). The best algorithm was selected, executed five times, and the best two results averaged to generate the results below. An FPGA implementation based derived from the discussion in this chapter should have a near constant performance which can be estimated simply as the number of comparisons to make, multiplied by the time per comparison at the FPGAs clock rate. Extra comparisons were incorporated to account for the starting condition when the lookup table requires filling. Although this still only yields an amortized time, it should be extremely close to the final total. These results are summarised in Figure 5-5.

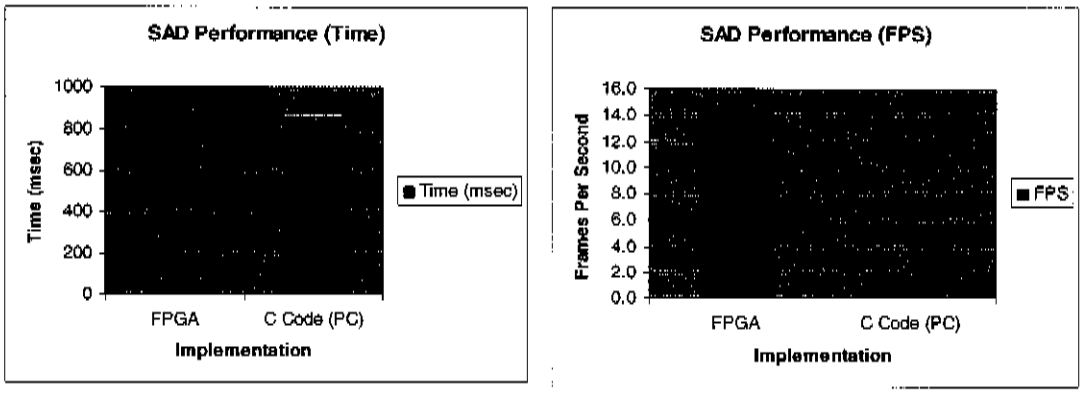


Figure 5-5 FPGA Implementation SAD performance estimates for 50MHz Xilinx Spartan S3E500 FPGA with 100MHz RAM compared to 1.83GHz Intel Core Duo based PC (single thread code) with 2MB cache and 667MHz DDR2 RAM (352x288 source image resolution, 7x7 window.)

Chapter 6

Disparity Errors

There are a number of difficulties in establishing depth through disparity of corresponding image regions. There can never be a perfect correspondence routine as pixels in each image do not always derive from a point in 3D space which corresponds to the visible region of the second image's view projection (Figure 6-1). In addition, because of sensor noise (Figure 6-2), detector sensitivity discrepancies(Figure 6-4), 2D spatial quantisation (Figure 6-2), and lens distortion, a portion of an object visible in both images of a stereo pair, being viewed from similar angles and distances in each image, will not necessarily produce identical pixels in the images. Finally different distances between cameras and points will lead to different sizes of areas in the images, which will cause problems for any algorithm which makes the assumption of objects appearing the same size and orientation in each image after rectification. A variety of solutions to these problems have been proposed in the past, and many of these will be explored and compared along with other suggested, high performance methods.

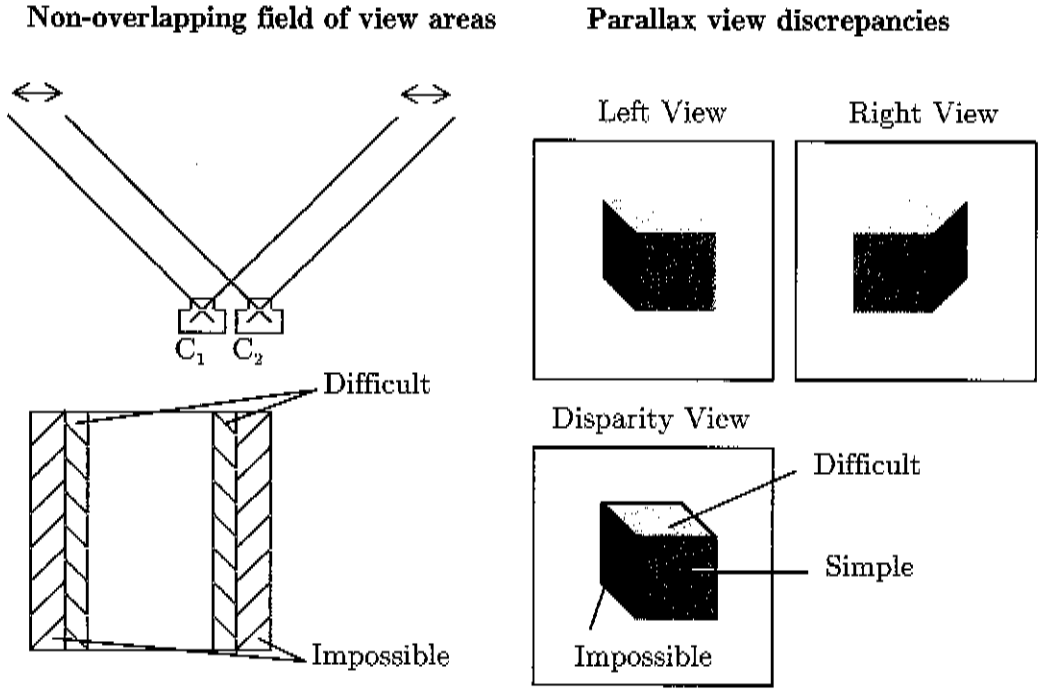


Figure 6-1 The two types of visible area discrepancy between stereo image pairs

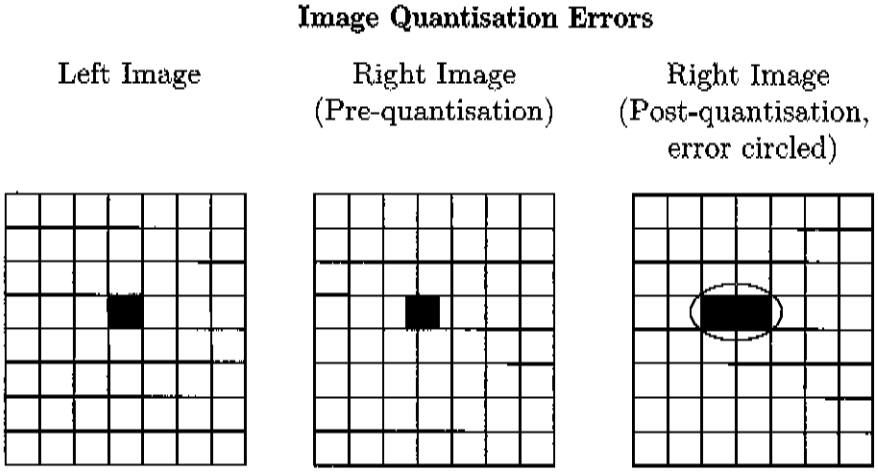


Figure 6-2 Discrepancy in quantised stereo image windows due to sub-pixel offsets.

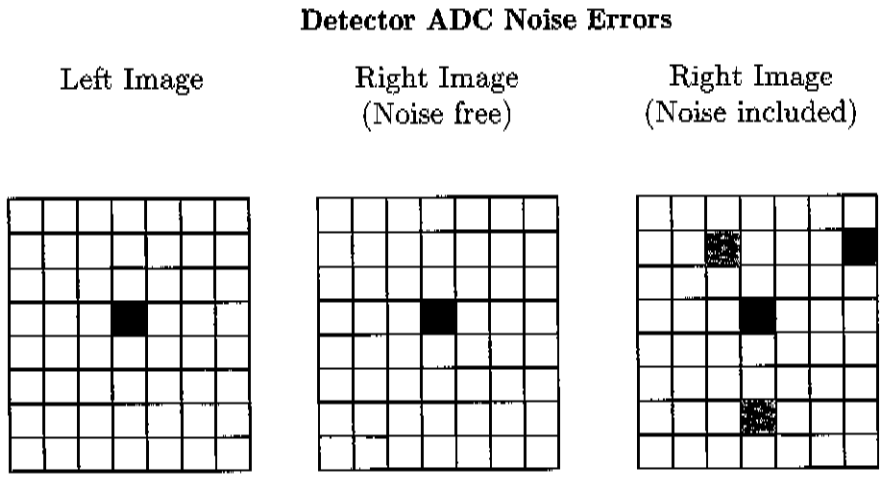


Figure 6-3 Discrepancy in stereo image windows due to noise in detector ADC

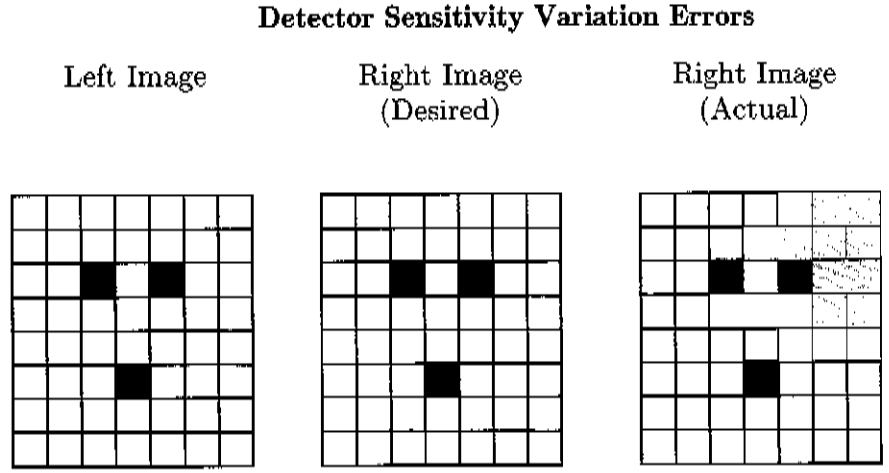


Figure 6-4 Discrepancy in quantised stereo image windows due to sub-pixel offsets

The errors outlined previously can be roughly sorted into two categories. The random errors which occur due to quantisation and sensor noise, and the expected errors caused by the geometry and texture of the scene. It is the latter, scene dependent errors which contribute most to the error sum. One of the obvious problems with area based metrics such as SAD is the fundamental inability to cope with regions that lack detail (texture and boundaries.) These areas are often not distinguishable by simply comparing pixel intensity or colour values. Similarly, windows containing multiple depths will fail to match because of the size and parallax position problems[28].

		Errrors		
Error Suggested	**Area Size**	**Count**	**Sum**	**Sum Squared**
Depth Discontinuities	17008 (19%)	7119 (25%)	457008 (39%)	45153024 (45%)
Textureless	47538 (54%)	19732 (68%)	674224 (57%)	49523968 (50%)
Occluded	2258 (3%)	2006 (7%)	160400 (14%)	15695616 (16%)
Combined	60744 (69%)	24737 (85%)	1031888 (88%)	86522624 (87%)
Entire Image	87696	28990	1176288	99348992

Table 6-1 Error compared to ground truth for Tsukuba image set for a 7x7 SAD disparity map in the region expected to have particular difficulties[29].

An evaluation of the accepted known error sources[29] with the Tsukuba image set reveals these account for 69% of pixels with error, and more importantly, 88% and 87% respectively of the sum and sum of squared values for these errors. Clearly these are far more taxing than camera ADC noise, quantization and other random sources.

6.1 Sub-pixel Matching

Sub-pixel matching is can improve both the success rate and precision of the correspondence search. Sub-pixel matching can significantly reduce errors caused by quantisation. Implementing true sub-pixel precision requires utilising some type of filter (typically linear) to upsize the secondary image much larger than the original source provides. Generating this image on the fly, even linearly, requires significant additional resources. Each new pixel value will have four inputs from the original image, and multiplication or division will be required for any size other than a multiple of two in each dimension. Even more significantly, there will be a linear increase in the search complexity with the increase in the total pixel count:

$$O(Search_{sub-pixel}) = O(Search_{original}) \times \frac{width_{sub-pixel} - width_{original}}{width_{original}} \times \frac{height_{sub-pixel} - height_{original}}{height_{original}}$$

True sub-pixel searching is therefore a poor choice for a low power, embedded platform.

An alternative to this is to consider it as a multiple resolution optimisation and only perform sub-pixel matching in the region surrounding the best match at the original level. This technique makes perfect sense in a software implementation, as the slower algorithm can be applied to a small target area, and only this small area need undergo up-sampling, hence a small increase in total run-time will yield a higher quality result. Unfortunately this does not translate well to an FPGA implementation where a longer, slower routine applied occasionally leads to a large decrease in implementation efficiency. The circuit required to implement this dynamic programming optimisation would be approximately twice that required by the original disparity technique. It is not possible to merely trade additional runtime for the application of this technique. Since it was established earlier that sub-pixel errors do not account for a large proportion of disparity errors, and this technique is difficult to implement, this optimisation is not a good candidate for FPGA implementation on the M6 platform.

6.2 Disparity Analysis Tool

To evaluate the effectiveness of different confidence algorithms, an application was written to compute, visualise and transform the various data sets. This program allows the user to test hypothesise, both subjectively and analytically, through a toolbox of image editing and comparison functions. The ability to quickly view numerical values at the same point in multiple 2D data sets also aids in verifying and debugging algorithms. In the future this application could

be combined with the tools presented earlier to verify the accuracy of simulated hardware implementations.

The user interface of the disparity analysis tool provides a powerful manipulation environment. A significant part of this is the use of binary masks, which can be created by performing binary operations on other masks or thresholded image data. The application implements all algorithms for which results are provided in this Chapter, with an emphasis on replicating as closely as possible the integer methods which would be employed in a future FPGA implementation. In addition, data analysis tools for evaluating results, detailed logs of all action taken during a session, and a three dimensional visualisation environment to aid subjective evaluation of results are all provided. A more detailed description of the usage and source code layout of this application is provided in Appendix A.

6.3 Accuracy Improvement through Confidence Estimation

Rather than improving the precision of the original test, an alternative option to improve accuracy is to find and reduce errors by assigning values a confidence, and then ignore, replace or blend values of low confidence. This method is particularly suitable to a hardware implementation as many confidence estimation techniques would require very little additional hardware resources or processing time.

Figure 6-5 7x7 window Tsukuba disparity map with example low confidence matches highlighted next to ground truth

Figure 6-6 An early build of the final disparity analysis application written in support of this project. A simultaneous cursor and detailed analysis logging can be seen in action.

Several algorithms for deriving properties with potentially useful statistical correlations to confidence were tested. Some of these were found through experimentation with the custom Disparity Analyser application, whilst others were located in the literature. A quick summary of these algorithms and their success in detecting or correcting errors is given in Table 6-2. An analysis of

their ability to improve the complete depth map through the use of the high quality nearest replacement technique is also provided (Table 6-3).

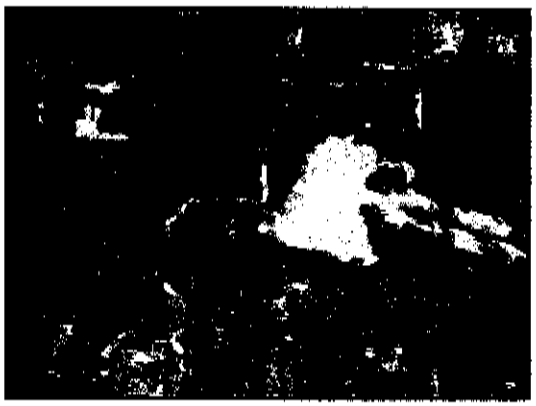
		False Errors	Errors Detected		
Algorithm	**Parameters**	**Count**	**Count**	**Sum**	**Sum Squared**
Multi-window distance.	threshold=1	6125	11534	520912	46363392
	threshold=2	5149	8409	400192	3673344
Minimum Window	low threshold=128 high threshold=750	42323	20820	798800	64646912
2nd Best Dist.	threshold=2	22190	15201	620864	51801600
Correspondence	threshold=1	18543	19475	905264	85280512

Table 6-2 Error totals and counts of good pixels mistakenly marked as errors for various confidence estimation techniques and parameters.

		Post-Fill Nearest Error		
Algorithm	**Parameters**	**Count**	**Sum**	**Sum Squared**
Multi-window distance.	threshold=1	28939	1141434	91072300
	threshold=2	28580	1134860	90467080
Minimum Window	low threshold=128 high threshold=750	21663	913184	73636864
2nd Best Dist.	threshold=2	27718	1243165	134018067
Correspondence	threshold=1	29144	1079248	76039424

Table 6-3 Error totals for nearest replacement scheme revised depth maps for various confidence estimation techniques and parameters.

Multi-window distance (threshold=1)



Multi-window distance (threshold=2)

Minimum window (Low threshold=128 high threshold=750)

2^{nd} best dist. (threshold − 2)

Correspondence (threshold=1)

Figure 6-7 Disparity maps with low confidence regions highlighted for various estimation techniques.

6.4 Applications of the Uniqueness Constraint

6.4.1 Variable-Window Matching

Another method of improving the quality of disparity maps is to apply multiple SAD tests for each pixel. Examples of such algorithms include testing multiple window sizes[30] and using an adaptive window[31]. There algorithms require significant additional implementation work on an FPGA platform. Since the computation of the smaller window is, in fact, a subset of the original problem, it should not require a large amount of additional FPGA resources to implement in an efficient design. Unfortunately the opportunity for parallelism, if re-utilising the same hardware, will drop in proportion with the window size, and so the additional execution time will be the same, not smaller, than the time required for the base disparity search. Also significant is that the largest consumer of memory bandwidth, the lookup table, will also double in size and so the total time required by an FPGA implementation of this algorithm is approximately twice that of the single window size disparity search. Correspondingly, the achievable frame rate would halve assuming the non-disparity portions of the operation are negligible.

The multi-window method for which results were presented is comparing the distance between the matched locations for the window sizes. The secondary window size was chosen to be 2 smaller in each dimension than the base window size. As expected, this method proved effective at eliminating many of the faults which occur normally due to multiple depths in the large window (parallax errors.)

Tests were also performed on a basic, non-adaptive multi-window approach. This method produced a lower quality than the most basic single window size

approach and is therefore not included in the testing. Although this result may not hold for images with significantly more occluded regions, the adverse consequences under typical conditions combined with the doubled run-time means it is recommended that a non-adaptive multi-window approach be avoided.

Adaptive windows may offer advantages, but introduce further complexity. An additional algorithm is required to determine the most appropriate window such that it improves the reliability of the final match. If quality was of utmost importance, then it would be worth investigating strategies employing this technique. It may be that a method which does not burden the FPGA significantly beyond the doubled SAD computation time may be possible. This would likely involve some other method for statistically predicting the preferred window.

6.4.2 Second Map Correspondence Test

The correspondence test involves calculating a second entire depth map, with the source images swapped. It is then verified that for each point in the left image, the point in the right image it was matched to was also matched back to the original point in the left image. A slightly more advanced version of this was implemented where the offset between the corresponding matches was calculated, rather than just a binary true/false correspondence.

Statistically this was the most successful technique, but suffers from the disadvantage that it requires slightly more than twice as much computation as the original disparity technique. This can be accomplished in half the time rather than double the resource usage however, so this technique is worth considering for high quality, slower speed matching.

6.5 Statistically Derived Method

6.5.1 Second Best Match Distance

This method works on the assumption that, if the two best matches are close to each other, they are probably identifying the same, correct match. If the two best matches are spaced far apart, they are signifying the choice of match was more likely to have been random and incorrect. This algorithm is cheap to computer, as the disparity circuit simply has to retain two, rather than one, of the best windows it has seen so far at each step. After this a single sum of absolute difference of the distance of the points is computed and the result is calculated. Unfortunately this method did not make a good improvement, and therefore this method is not recommended.

6.5.2 SAD Window Comparison Minimum Error Thresholding

SAD window error thresholding is a technique which was derived statistically using the analysis application written for this project. A reference to this technique was not found in any literature, although it is likely to be in use (at least commercially) due to its computational simplicity and strong correlation with image error.

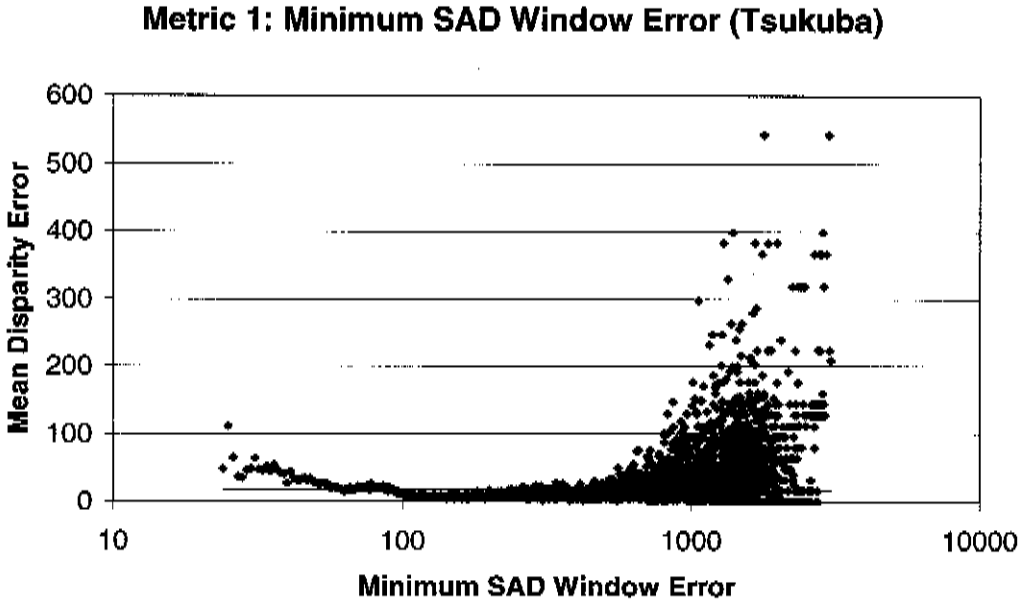


Figure 6-8 Mean disparity error value as a function of the minimum SAD window's error value

This chart shows that the values of low average error when compared to the true disparity actually congregate towards the middle of the chart. What makes this a particularly worthwhile test is that the majority of the values congregate in this good region. It is therefore a very simple method of locating a small subset of values with particularly low confidence.

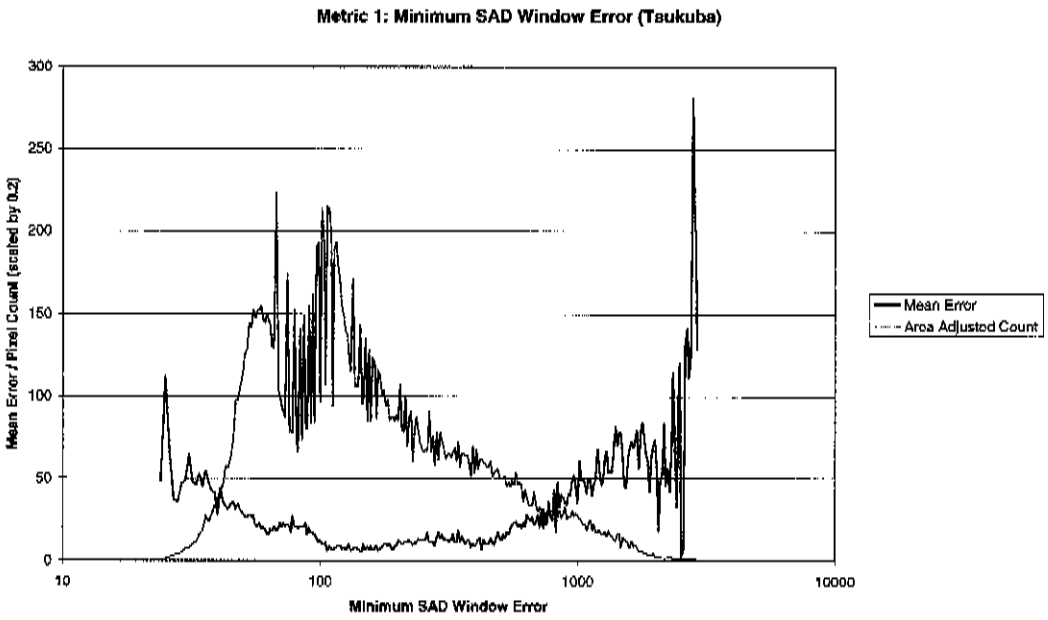


Figure 6-9 Mean error and area adjusted pixel counts as a function of the best matching SAD window's error value. Counts are adjusted so that all areas along the logarithmic graph correspond to the same amount of pixels.

It is intuitively obvious that values with a high minimum SAD error should have a low confidence. These values correspond to regions where no pair of regions formed a particularly close match. These regions are likely to correspond with regions where parallax errors mean the objects visible in each image differ substantially.

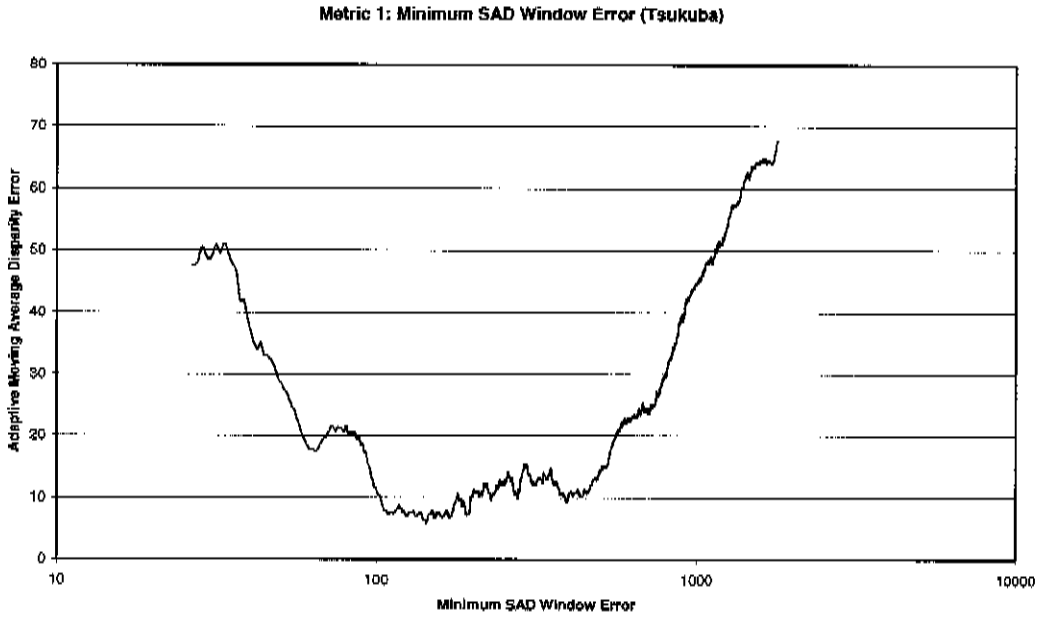


Figure 6-10 Plot of the adaptive moving average of SAD disparity error compared to the true disparity for different minimum SAD window error values. The adaptive moving average groups the values into equally sized bins, irrespective of the density of data points along the x-axis.

Whereas the relationship between high minimum SAD error values and poor confidence is obvious, the more difficult question must also be asked: why it is that values with lower errors can in fact be less reliable than those that fall in the midrange. An alternate angle on this is to consider why having a higher error appears to make a value more reliable, up to a point. One possible explanation is that the random sensor noise and quantisation errors, which should be ever constant, appear to have disappeared when the errors value is too small. This would happen in regions of flat colour, which normally cause disparity error. Flat regions are inherently less susceptible to quantisation errors, since the strength of such errors is relative to the difference in intensity between neighbouring pixels. These regions also suffer from a low signal to noise ratio: They can pick from several similar areas when matching, since the true grey scale signal is relatively uniform and thus weak, and therefore choose areas where the random noise contributions are most similar. This increased selection would enable a better noise match than if the area were limited by the presence of distinct features, as is usually the case in those areas which make a valid

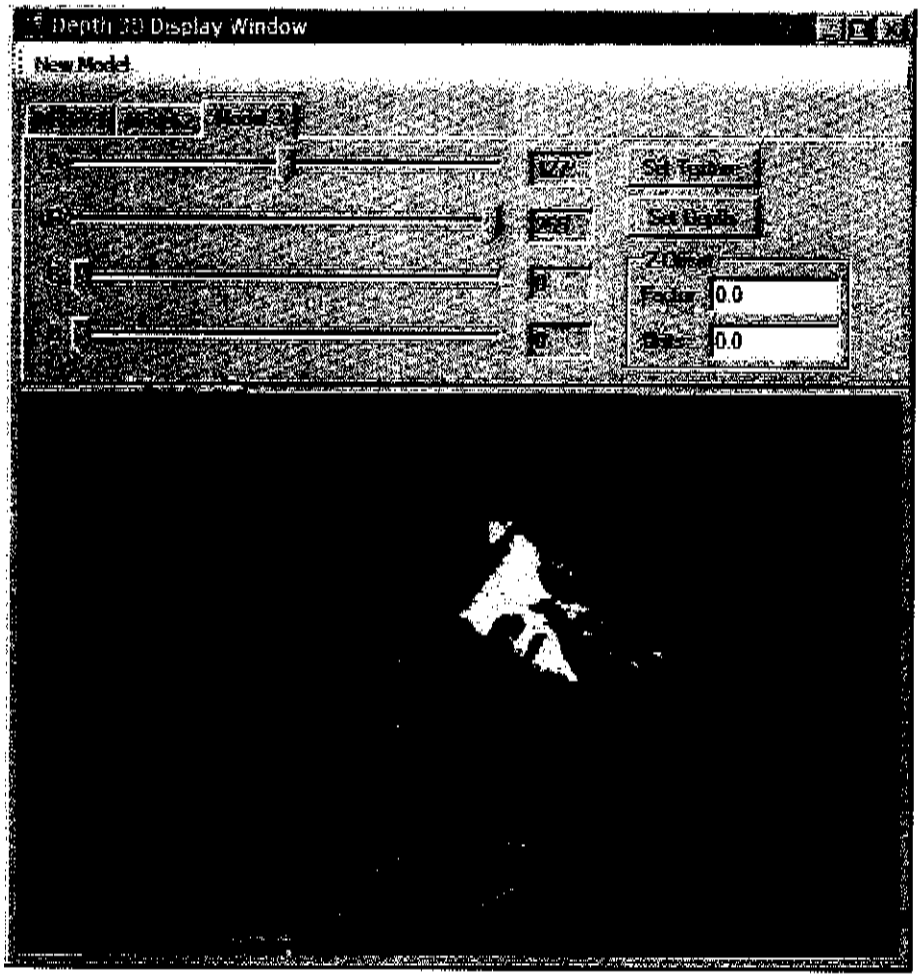


Figure 6-11 3D visualisation of minimum window error threshold corrected depth map(green) as compared to original (red) and truth (grey.) Green areas represent points where the depth map was not completely fixed, while red errors represent areas that are wrong in the plain disparity map but fixed after thresholding and nearest replacement.

Once regions of poor accuracy in depth maps have been identified, there is an application dependent decision of the fix to make. The standard method is to blank out the bad regions (and expect algorithms to be able to ignore them.) A number of other corrective methods (Figure 6-12) were implemented. These vary in how accurately they correct the image and how much FPGA processing time and resources would be required for implementation.

match. It is suggested therefore that there is a value below which having a lower error actually makes a match less likely to be correct, and that this low value will depend on the statistical noise properties of a particular camera and environment combination.

Statistically this technique was the second best at detecting errors, and in fact yielded the best corrected depth map. This technique would be extremely cheap to implement, as it is simply a matter of including a pair of comparators. On the other hand, it does require calibration which can be a disadvantage when conditions such as lighting are highly variable.

6.6 Implementation Details

The simple thresholding and second best distance test algorithms are special because they require a very tiny amount of additional effort (both processing time and FPGA logic) to implement. Both of these algorithms produce good results compared to the more complicated solutions, and are therefore deemed more appropriate to a high performance, low power embedded implementation. The advantage of the second algorithm is its invariance to image parameters, and therefore lack of need for calibration. The minimum and maximum threshold method produces a better result with the appropriate calibration, but how easy it will be to maintain such calibration when conditions such as lighting are variable it is difficult to quantify. It may be that the ideal algorithm involves some combination of these two methods. No combination was found during testing that outperformed the minimum and maximum algorithm alone, but if the calibration situation became more difficult this would likely change.

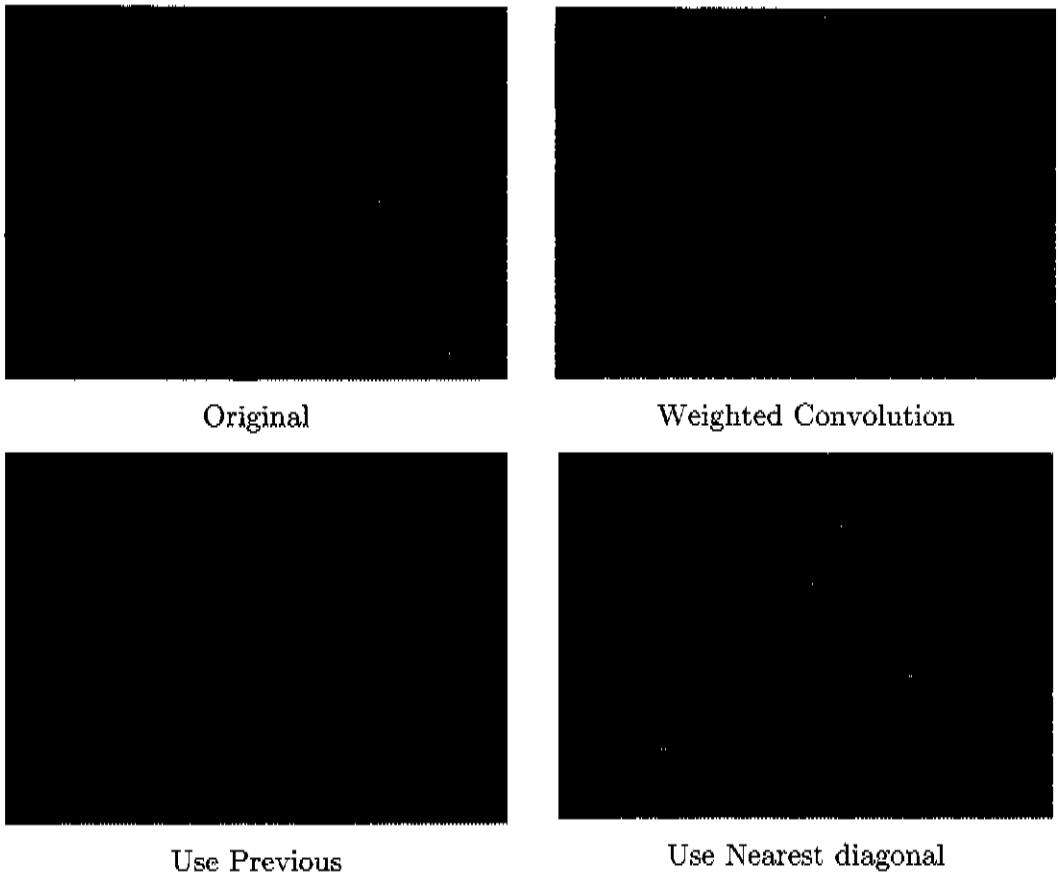


Figure 6-12 Results of applying threshold test followed by various corrective procedures to test images.

Chapter 7

Conclusion

7.1 Outcomes

This project has seen the part selection, schematic design and post-fabrication debugging of a new generation Eyebot platform conducted as part of a small team. This project has also covered the implementation of a variety of logic design elements for the new platform including

- Optimised, multi-clock domain memory controller
- Compatible binary tree arbitrating memory bus and high speed interconnect bus
- Camera I/O with constrained clock signal availability
- Optimised (within constraints) communication with an Intel PXA255 CPU
- Optimised windowed convolution element with small resource foot print

In addition, code was developed to integrate C++ verification suites with simulated VHDL code. A GUI application for convolution verification was also implemented.

A large GUI application was written to analyse, verify and visualise disparity mapping algorithms. Analysis was performed on a variety of techniques for

calculating and refining disparity maps. M6 platform specific recommendations have been made with regards choice of metrics, window sizes and confidence estimation.

7.2 Future Work

The base FPGA image processing system for the M6 platform lacks programmer friendliness at present. Each IO device group is separated into its own code base, and there exists no user space library which can adequately shield users from the lowest level Linux kernel IO. Time to obtain the initial test boards was a big limiting factor in this regard, and it is expected that future project which can start from the base provided in this project will be able to easily improve on this.

There exists the possibility for improvemed CPU to FPGA communication performance on the second revision M6 PCBs through the use of ported IO. This was not an option on the original board as the pins necessary for the CPU to read data at higher rates than those achieved were not connected. This would greatly reduce the CPU bus time absorbed communicating with the FPGA.

The most important stereo vision work which remains is the implementation of rectification. Rectification is currently the biggest unknown in all areas of concern (speed, FPGA resource usage and effect on final image quality.) Again it was difficult to enter into a more thorough analysis of this subject as many other areas had to be dealt with first, most of which now have. Further analysis of disparity algorithms is perhaps the most exciting work which has not yet been completed. It is likely that multi-variate statistics applied to existing data in this project could find an efficient algorithm with even greater accuracy than those discussed here. Even with univariate statistics, further scope exists with the current stereo application to statistically draw conclusions as to which types of errors are better corrected with each algorithm. This would not lead directly

to an improved algorithm, but instead act as a guide to likely areas of success in multi-variate analysis. An implementation and corresponding analysis of a good Rank method may also prove beneficial to users of the M6 platform, especially those who have to deal with challenging conditions (lighting, geometry.) Finally, a true FPGA implementation of one or more of the the real-time disparity algorithms derived needs to be implemented. This will require verification and optimisation of VHDL code similar to that documented for the convolution algorithm is needed.

References

[1] V. V. Zyuban and P. M. Kogge, "Inherently Lower-Power High-Performance Superscalar Architectures," *IEEE Transactions on Computers*, vol. 50, pp. 268-285, 2001.

[2] Cambridge-MIT Institute, "MDP Microprocessor Board." http://www-mdp.eng.cam.ac.uk/micro/intro.html. [On-line, accessed 18-8-06].

[3] B. Blackham, "The Development of a Hardware Platform for Real-time Image Processing," in *School of Electrical, Electronic and Computer Engineering*. Perth: The University of Western Australia, 2006.

[4] L.Chin, "FPGA-based embedded vision systems.," in *School of Electrical, Electronic and Computer Engineering*. Perth: The University of Western Australia, 2006.

[5] Xilinx, Inc., "Spartan-3E Complete Data Sheet." http://direct.xilinx.com/bvdocs/publications/ds312.pdf. [On-line, accessed 11-5-2006].

[6] "SA-C Compiler Overview." http://www.cs.colostate.edu/cameron/compiler.html. [On-line, accessed 12-9-2006].

[7] techfocus media, inc., "Cray Goes FPGA." FPGA and Programmable Logic Journal http://www.fpgajournal.com/articles_2005/pdf/20050405_cray.pdf. [On-line, accessed 23-9-06].

[8] Intel Corporation, "Intel PXA255 processor: Developer's manual." http://www.intel.com/design/pca/applicationsprocessors/manuals/278693.htm. [On-line, accessed 10-Oct-2006].

[9] Cypress Semiconductor Corporation, "CY7C1381D, CY7C1383D SRAM Product Data Sheet." http://download.cypress.com/publishedcontent/publish/design_resources/datasheets/contents/cy7c1383d_8.pdf. [On-line, accessed 14-7-06].

[10] Omnivision, "Advanced Information (Preliminary) OV6630/OV6130 Digital Camera." http://home.pacific.net.hk/~comedia/ov6630.pdf. [On-line, accessed 28-7-06].

[11] W. H. Press, *Numerical recipes in C : the art of scientific computing*, 2nd ed. Cambridge: Cambridge University Press, 1992.

[12] School of Computer Science & Software Engineering, The University of Western Australia, "Camera calibration." http://undergraduate.csse.uwa.edu.au/units/CITS4240/Lectures/calibration.pdf. [On-line, accessed 15-11-06].

[13] O. Ghita, J. Mallon, and P. F. Whelan, "Epipolar line extraction using feature matching," presented at Irish Machine Vision and Image Processing Conference 2001, NUI Maynooth, 2001.

[14] D. H. Lee, I. S. Kweon, and R. Cipolla, "A Biprism-Stereo Camera System " presented at IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'99), 1999.

[15] Computer Science Department, University of Auckland, "Stereo Vision Lecture: Rectification." http://www.cs.auckland.ac.nz/compsci775s2t/lectures/lectA-775S2T.htm. [On-line, accessed 15-10-06].

[16] M. Z. Brown, D. Burschka, and G. D. Hager, "Advances in Computational Stereo " *IEEE Transactions on pattern analysis and machine intelligence*, vol. 25, pp. 993-1008, 2003.

[17] T. Kanade, "Development of a Video-Rate Stereo Machine," presented at Proceedings of 94 ARPA Image Understanding Workshop, Monttey, California, 1994.

[18] L. d. F. Costa and R. M. Cesar, *Shape analysis and classification : theory and practice*. Boca Raton, FL: CRC Press, 2001.

[19] D. Scharstein and R. Szeliski, " High-accuracy stereo depth maps using structured light," presented at IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2003), Madison, WI, 2003.

[20] A. Kuhl, "Comparison of Stereo Matching Algorithms for Mobile Robots," in *School of Electrical, Electronic and Computer Engineering*. Perth: The University of Western Australia, 2005.

[21] J. Banks and P. Corke, "Quantitative Evaluation of Matching Methods and Validity Measures for Stereo Vision," *Int. Journal of Robotics Research*, vol. 20, pp. 512–532, 2001.

[22] D. N. Bhat and S. K. Nayar, "Ordinal Measures for Image Correspondence," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, pp. 415-423, 1998.

[23] N. Sebe, M. S. Lew, and D. P. Huijsmans, "Toward Improved Ranking Metrics " *IEEE Transactions on pattern analysis and machine intelligence*, vol. 22, pp. 1132-1143, 2000.

[24] R. B. Porter and N. W. Bergmann, "A generic implementation framework for FPGA based

stereo matching " presented at IEEE TENCON - Speech and Image Technologies for Computing and Telecommunications Brisbane, Australia, 1997.

[25] Middlebury College, "Stereo Vision Research Page - Data." http://cat.middlebury.edu/stereo/data.html. [On-line, accessed 12-10-06].

[26] W. Li and E. Salari, "Successive elimination algorithm for motion estimation," *IEEE Transactions Image Processing*, vol. 4, pp. 105–107, Successive elimination algorithm for motion estimation.

[27] T. Kanade, H. Kano, S. Kimura, E. Kawamura, A. Yoshida, and K. Oda, "Development of a video-rate stereo machine," presented at Intelligent Robots and Systems 95. 'Human Robot Interaction and Cooperative Robots', Proceedings. 1995 IEEE/RSJ International Conference on, Pittsburgh, PA, USA, 1995.

[28] K.-J. Yoon and I.-S. Kweon, "Adaptive Support-Weight Approach for Correspondence Search," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 28, pp. 650-656, 2006.

[29] D. Scharstein and R. Szeliski, "A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms," *International Journal of Computer Vision*, vol. Volume 47, pp. 7-42, 2002.

[30] H. Hirschmüller, P. R. Innocent, and J. Garibaldi, "Real-time correlationbased stereo vision with reduced border errors," *International Journal of Computer Vision*, vol. 47, pp. 229-246, 2002.

[31] A. Fusiello, V. Roberto, and E. Trucco, "Efficient Stereo with Multiple Windowing," presented at 1997 Conference on Computer Vision and Pattern Recognition (CVPR '97), Puerto Rico, 1997.

Appendix A

Stereo Vision Analysis Source Code Overview

Disparity Analyser was the third and final prototyping system for stereo vision algorithms developed during the course of this project. It is a GUI based system which attempts to ease the debugging, verification, analysis and interpretation of primarily disparity maps.

There are two main features which enable this. The first is the use global cursor positioning, as indicated in Figure 6-6. This feature is important both to quickly ascertain if algorithms are functioning as intended, and to utilise the viewer's ability to identify trends. The second important feature of the program is the ability to apply algorithms and other manipulations to the data set and immediately inspect the results after each step. It is this combination of features which allows the effective creation, verification and quality optimisation of algorithms.

A.1 Source Overview

The complete source code of Disparity Analyser is written in a hybrid language called Managed C++. Managed C++ is Microsoft's replacement for managed extensions for C++ as a method for mixing .Net and regular C++ code. The advantage of using a hybrid language is that it allows the mixing of garbage

collected, byte coded APIs such as the .Net Forms user interface with regular high speed completely system native code. The advantage of Managed C++ over its predecessor managed extensions is that it provides a relatively clean syntax for both languages simultaneously. Another advantage is the availability of a free compiler which supports virtually all features and libraries as the older standard Visual Studio editions with the notable exception of MFC.

Important sections of the code are documented with doxygen style comments. Unfortunately with the Managed C++ language only becoming available the year before this project was undertaken, the main doxygen distribution does not support the language. Patches are available from the doxygen forums and the feature is due to appear in the main releases within a month. A temporary alternative to doxygen generated comments is provided later in this document.

The remainder of this overview section is divided into two parts, the native C++ algorithms and the wrapper code which allows them to interoperate with the managed environment, and the managed user interface which provides a powerful mechanism for applying and evaluating the algorithms. The native algorithm code is written in standard C++ and can be easily separated from the user interface if an implementation on a different platform is desired.

A.1.1 Algorithms (Native C++)

Most of the features of Disparity Analyser are based around 2D Regions. The concept of a region is applied at two different abstraction levels within the code base. The first of these, discussed here, is essentially a 2D array of any type of object. Care should be taken not to confuse this with the UI's more specific concept of a region, covered in the next section, and built up from this low level implementation. The 2D region class has a template parameter which, depending on the function, is usually intended to be either a numerical data

type or a Boolean. Boolean types are usually specified explicitly whereas numerical types are left as a parameter to allow maximum reuse flexibility. This offers an advantage over the most common scripted image manipulation environment Matlab, which casts all data to doubles before performing most operations.

There are three small classes in the algorithms header, two being very specific to storing a particular type of result data set, and the third, Line, being the 1D equivalent of Region. The line data type is rarely useful as its functionality is very similar to the standard C++ library's vector data type.

A.1.2 User Interface (Managed C++ - .Net 2.0)

The user interface, like the algorithms, is built around the concept of a 2D region. The user interface's 2D region actually represents two low level regions. The first is referred to as the source region, and the main display bitmap for visualizing the region is usually derived directly from this, hence the name. This region contains 32bit integer values, which may, depending on the situation, be treated as 24bit RGB colour pixels with 8 bits per colour component, or more commonly as scalar values allowed to occupy the full range of the int data type.

In addition to the source region, there is the mask region, which contains Boolean values. The mask region, when enabled, affects almost every operation, controlling which portion of the image is modified or included in calculations. Usually for modification operations the mask is equivalent to applying the operation to the entire image, and then incorporating the masked portion of the new data set with the unmasked portion of the original data set. This is not always strictly true, however, as, for instance, with the fill operations. This lack of consistent definition of a mask is one of the reasons this code could not efficiently be implemented directly in a conventional manipulation environment

such as Matlab. The mask is extremely important as it allows regions to effectively be of any shape, including those made from multiple (disconnected) areas.

A.2 Overview of Important Classes

Unless noted, classes are written in Managed C++ containing a mixture of .Net and regular C++ code. All code was built with Visual Studio 2005, and managed classes may require .NET 2.0 and the Win32 Platform SDK. Certain parts of the code require OpenGL and or SG mathematics library. This is noted in their descriptions. The Visual C++ 2005 Express edition available freely from Microsoft is capable of building this source code. The Win32 Platform SDK and OpenGL libraries are also available freely from Microsoft. The SG mathematics library is available free as a component of the PLIB portable games library.

Only the algorithm and OpenGL related classes are documented below. Purely .Net User interface classes have been omitted.

Class Name	File Location	Description
Namespace: DAlgorithms		
Region	Algorithms.h	A templated class which stores a two dimensional set of its target parameter, created with the element's new allocator. The templated type needs to be bool or a numerical type for most operations to make sense. All disparity algorithms are implemented in this class. (Does not require .Net or Win32, although conversion from Win32 image types will be excluded if Win32 is not available.)
RegionManaged	AlgorithmsManaged.h	A .NET Wrapper for the Region Class.
Line	Algorithms.h	Similar to the region class but stores a one dimensional data set. (Does not

		require .Net or Win32.)
Histogram	Algorithms.h	Stores the result of a histogram operation - min and max range covered plus a Line containing the actual values. (Does not require .Net or Win32)
MetricDomain Stat	Algorithms.h Outer class: Region	Stores the result of an intensity domain conversion. (Does not require .Net or Win32)
Namespace: DepthGL		
GLCamera	GLCamera.h	Stores the position and orientation of the camera, and sends appropriate matrices to OpenGL. (Does not require .Net or Win32 but requires OpenGL and SG)
DepthModel	DepthModel.h	Stores the settings and data for a single disparity map model. Can send appropriate rendering commands to OpenGL. (Does not require .Net or Win32 but requires OpenGL)
DepthModelSettingsControl	DepthModelSettingsControl.h / DepthModelSettingsControl.cpp	A wrapper to support modification of a DepthModel through a .Net GUI. Serves events which notify of the need to repaint the model when modified.
DepthGLControl	DepthGLControl.h	Wraps the actual OpenGL context and provides most of the initialisation code. Also provides some of the basic geometry setup and the outer-most render routine.
Namespace: Stereo3		
RegionDisplayControl	RegionDisplayContro.cpp / RegionDisplayControl.h	Represents a complete region set (image, mask and the actual region.) Implements many of the algorithms, mostly by deferring calculations to the Region class. Implements the undo functionality.
StereoPair	StereoPair.h / StereoPair.cpp	Represents a pair of source images, and implements the disparity algorithms through use of the base algorithms and other features in the Region class.

Appendix B

Stereo Vision Analysis Logs

Transcriptions of test logs for data used in the dissertation. Logs have been omitted from the printed version for tests involving intensity domain transforms as these produce extremely large data sets. Note that transcripts are edited to remove irrelevant lines. This should only be apparent in discontinuous numbers, and should not affect reproducibility.

[001] Opened Left Source from C:\stereo\t\left.tif

[002] Opened Right Source from C:\stereo\t\right.tif

[003] Created New Disparity14Error Pair1 from sources Left Source and Right Source

Settings - Type: Std. Minimum Sum, Direction: Left, Window Size: 4, Left Search Width: 20, Right Search Width: 0

[004] Pair1 Disparity[ ] : Multiplied with 16

[005] Opened Source1 from C:\stereo\t\truedisp.png

[006] Pair1 Disparity Mask : and with Source1 mask

[007] Pair1 Disparity Mask : and (boolean) with Source1

[008] Pair1 Disparity Mask : inverted

[009] Pair1 Disparity[M] : Multiplied with 0

[010] Created New Disparity14Error Pair2 from sources Left Source and Right Source

Settings - Type: Std. Minimum Sum, Direction: Left, Window Size: 3, Left Search Width: 20, Right Search Width: 0

[011] Pair2 Disparity[ ] : Multiplied with 16

[012] Created New Disparity14Error Pair3 from sources Left Source and Right Source

Settings - Type: Std. Minimum Sum, Direction: Left, Window Size: 5, Left Search Width: 20, Right Search Width: 0

[013] Pair2 Disparity Mask : and (boolean) with Source1

[014] Pair2 Disparity Mask : inverted

[015] Pair2 Disparity[M] : Multiplied with 0

[016] Pair3 Disparity Mask : copied from Pair2 Disparity

[017] Pair3 Disparity[M] : Multiplied with 0

[018] Pair3 Disparity[ ] : Multiplied with 16

[019] Cmp1[ ] : Sum = 1176288

[020] Cmp1[ ] : Sum of Squares = 99348992

[021] Cmp3[ ] : Sum = 1024000

[022] Cmp3[ ] : Sum of Squares = 80260608

[023] Cmp4[ ] : Sum = 956016

[024] Cmp4[ ] : Sum of Squares = 72347392

[025] Created New Disparity14Error Pair4 from sources Left Source and Right Source

Settings - Type: Std. Minimum Sum, Direction: Left, Window Size: 6, Left Search Width: 20, Right Search Width: 0

[026] Pair4 Disparity[ ] : Multiplied with 16

[027] Pair4 Disparity Mask : and (boolean) with Source1

[028] Pair4 Disparity Mask : inverted

[029] Pair4 Disparity[M] : Multiplied with 0

[030] Cmp5[ ] : Sum = 926112

[031] Cmp5[ ] : Sum of Squares = 69288448

[032] Created New Disparity14Error Pair5 from sources Left Source and Right Source

Settings - Type: Std. Minimum Sum, Direction: Left, Window Size: 7, Left Search Width: 20, Right Search Width: 0

[033] Pair5 Disparity[ ] : Multiplied with 16

[034] Pair5 Disparity Mask : and (boolean) with Source1

[035] Pair5 Disparity Mask : inverted

[036] Pair5 Disparity[M] : Multiplied with 0

[037] Cmp6[ ] : Sum = 911648

[038] Cmp6[ ] : Sum of Squares = 66949120

[039] Created New Disparity14Error Pair6 from sources Left Source and Right Source

Settings - Type: Std. Minimum Sum, Direction: Left, Window Size: 8, Left Search Width: 20, Right Search Width: 0

[040] Pair6 Disparity[ ] : Multiplied with 16

[041] Pair6 Disparity Mask : and (boolean) with Source1

[042] Pair6 Disparity Mask : inverted

[043] Pair6 Disparity[M] : Multiplied with 0

[044] Cmp7[ ] : Sum = 904800

[045] Cmp7[ ] : Sum of Squares = 65426432

[046] Created New Disparity14Error Pair7 from sources Left Source and Right Source

Settings - Type: Std. Minimum Sum, Direction: Left, Window Size: 9, Left Search Width: 20, Right Search Width: 0

[047] Pair7 Disparity[ ] : Multiplied with 16

[048] Pair7 Disparity Mask : and (boolean) with Source1

[049] Pair7 Disparity Mask : inverted

[050] Pair7 Disparity[M] : Multiplied with 0

[051] Pair7 Disparity[ ] : Multiplied with 16

[052] Undo of [051] Pair7 Disparity[ ] : Multiplied with 16

[053] Cmp8[ ] : Sum = 909280

[054] Cmp8[ ] : Sum of Squares = 65165824

[055] Created New Disparity14Error Pair8 from sources Left Source and Right Source

Settings - Type: Std. Minimum Sum, Direction: Left, Window Size: 10, Left Search Width: 20, Right Search Width: 0

[056] Pair8 Disparity[ ] : Multiplied with 16

[057] Pair8 Disparity Mask : and (boolean) with Source1

[058] Pair8 Disparity Mask : inverted

[059] Pair8 Disparity[M] : Multiplied with 0

[060] Cmp9[ ] : Sum = 919264

[061] Cmp9[ ] : Sum of Squares = 66016768

[072] Created New Disparity14Error Pair10 from sources Left Source and Right Source

Settings - Type: Std. Minimum Sum, Direction: Left, Window Size: 2, Left Search Width: 20, Right Search Width: 0

[073] Pair10 Disparity[ ] : Multiplied with 16

[074] Pair10 Disparity Mask : and (boolean) with Source1

[075] Pair10 Disparity Mask : inverted

[076] Pair10 Disparity[M] : Multiplied with 0

[077] Cmp10[ ] : Sum = 1452240

[078] Cmp10[ ] : Sum of Squares = 133423872

[[001] Opened Left Source from C:\stereo\t\left.tif

[002] Opened Right Source from C:\stereo\t\right.tif

[003] Opened Source1 from C:\stereo\t\truedisp.png

[004] Created New Disparity14Error Pair1 from sources Left Source and Right Source

Settings - Type: Std. Minimum Sum, Direction: Left, Window Size: 1, Left Search Width: 20, Right Search Width: 0

[005] Pair1 Disparity[ ] : Multiplied with 16

```
[006] Pair1 Disparity Mask : and (boolean) with Source1
[007] Pair1 Disparity Mask : inverted
[008] Pair1 Disparity[M] : Multiplied with 0
[009] Cmp1[ ] : Count = 40777
[010] Cmp1[ ] : Sum = 2206912
[011] Cmp1[ ] : Sum of Squares = 235035648
[012] Opened Source2 from
C:\stereo\t\window_size_comparion\Cmp1_Window3_Plain.png
[013] Source2[ ] : Count = 28990
[014] Opened Left Source from
C:\stereo\t\window_size_comparion\Cmp3_Window4_Plain.png
[015] Left Source[ ] : Count = 26733
[016] Opened Source3 from
C:\stereo\t\window_size_comparion\Cmp4_Window5_Plain.png
[017] Source3[ ] : Count = 25479
[018] Opened Source4 from
C:\stereo\t\window_size_comparion\Cmp5_Window6_Plain.png
[019] Source4[ ] : Count = 24600
[020] Opened Source5 from
C:\stereo\t\window_size_comparion\Cmp5_Window6_Plain.png
[021] Source5[ ] : Count = 24600
[022] Opened Source6 from
C:\stereo\t\window_size_comparion\Cmp6_Window7_Plain.png
[023] Source6[ ] : Count = 24084
[024] Opened Source7 from
C:\stereo\t\window_size_comparion\Cmp7_Window8_Plain.png
[025] Source7[ ] : Count = 23811
[026] Opened Source8 from
C:\stereo\t\window_size_comparion\Cmp8_Window9_Plain.png
[027] Source8[ ] : Count = 23606
```

[028] Opened Source9 from C:\stereo\t\window_size_comparion\Cmp9_Window10_Plain.png

[029] Source9[ ] : Count = 23367

[001] Opened Left Source from C:\stereo\t\left.tif

[002] Opened Right Source from C:\stereo\t\right.tif

[003] Created New Disparity14Error Pair1 from sources Left Source and Right Source

Settings - Type: Std. Minimum Sum, Direction: Left, Window Size: 3, Left Search Width: 20, Right Search Width: 0

[004] Pair1 Disparity[ ] : Multiplied with 16

[005] Opened Source1 from C:\stereo\t\truedisp.png

[006] Pair1 Disparity Mask : and (boolean) with Source1

[007] Pair1 Disparity Mask : inverted

[008] Pair1 Disparity[M] : Multiplied with 0

[009] Pair1 Disparity : Comparison created with Source1 as Cmp1

[010] Created New Disparity14Error Pair2 from sources Left Source and Right Source

Settings - Type: Std. Minimum Sum, Direction: Left, Window Size: 2, Left Search Width: 20, Right Search Width: 0

[011] Pair2 Disparity Mask : inverted

[012] Pair2 Disparity Mask : inverted

[013] Pair2 Disparity Mask : and (boolean) with Source1

[014] Pair2 Disparity Mask : inverted

[015] Pair2 Disparity[M] : Multiplied with 0

[016] Pair2 Disparity[ ] : Multiplied with 16

[017] Pair2 Disparity : Comparison created with Source1 as Cmp2

[018] Cmp2[ ] : Count = 32757

[019] Cmp2[ ] : Sum = 1452240

[020] Cmp2[ ] : Sum of Squares = 133423872

Log of window size comparison data generation

[001] Opened Left Source from C:\stereo\t\left.tif

[002] Opened Right Source from C:\stereo\t\right.tif

[003] Opened Source1 from C:\stereo\t\truedisp.png

[004] Created New Disparity14Error Pair1 from sources Left Source and Right Source

Settings - Type: Std. Minimum Sum, Direction: Left, Window Size: 3, Left Search Width: 20, Right Search Width: 0

[005] Pair1 Disparity[ ] : Multiplied with 16

[006] Opened Source2 from C:\stereo\LoG.png

[007] Source2[ ] : Added -2

[008] Left Source : Convolved with Source2

[009] Right Source : Convolved with Source2

[010] Created New Disparity14Error Pair2 from sources Left Source and Right Source

Settings - Type: Std. Minimum Sum, Direction: Left, Window Size: 3, Left Search Width: 20, Right Search Width: 0

[011] Pair2 Disparity[ ] : Multiplied with 16

[012] Opened Left Source from C:\stereo\t\left.tif

[013] Opened Right Source from C:\stereo\t\right.tif

[014] Opened Source3 from C:\stereo\gaussian-5x5_std1.png

[015] Left Source : Convolved with Source3

[016] Right Source : Convolved with Source3

[017] Created New Disparity14Error Pair3 from sources Left Source and Right Source

Settings - Type: Std. Minimum Sum, Direction: Left, Window Size: 3, Left Search Width: 20, Right Search Width: 0

[018] Pair3 Disparity[ ] : Multiplied with 16

[019] Pair1 Disparity Mask : and (boolean) with Source1

[020] Pair1 Disparity Mask : inverted

[021] Pair1 Disparity[M] : Multiplied with 0

[022] Pair2 Disparity Mask : copied from Pair1 Disparity

[023] Pair2 Disparity[M] : Multiplied with 0

[024] Pair3 Disparity Mask : copied from Pair1 Disparity

[025] Pair3 Disparity[M] : Multiplied with 0

[026] Pair1 Disparity : Comparison created with Source1 as Cmp1

[027] Pair2 Disparity : Comparison created with Source1 as Cmp2

[028] Pair3 Disparity : Comparison created with Source1 as Cmp3

[029] Cmp1[ ] : Count = 28990

[030] Cmp1[ ] : Sum = 1176288

[031] Cmp1[ ] : Sum of Squares = 99348992

[034] Cmp2[ ] : Count = 33527

[035] Cmp2[ ] : Sum = 1926640

[036] Cmp2[ ] : Sum of Squares = 213116672

[037] Cmp3[ ] : Count = 26611

[038] Cmp3[ ] : Sum = 1057024

[039] Cmp3[ ] : Sum of Squares = 81472512

Log of pre-processing filter comparison data generation

[001] Opened Left Source from C:\stereo\t\left.tif

[002] Opened Right Source from C:\stereo\t\right.tif

[003] Opened Source1 from C:\stereo\t\depth_discont.png

[004] Opened Source2 from C:\stereo\t\textureless.png

[005] Opened Source3 from C:\stereo\t\occl.png

[006] Created New Disparity14Error Pair1 from sources Left Source and Right Source

Settings - Type: Std. Minimum Sum, Direction: Left, Window Size: 3, Left Search Width: 20, Right Search Width: 0

[007] Pair1 Disparity[ ] : Multiplied with 16

[008] Opened Source4 from C:\stereo\t\truedisp.png

[009] Pair1 Disparity : Comparison created with Source4 as Cmp1

[010] Cmp1 Mask : and (boolean) with Source4

[011] Cmp1 Mask : inverted

[012] Cmp1[M] : Multiplied with 0

[013] Cmp1[ ] : Min = 0

[014] Cmp1[ ] : Max = 240

[015] Cmp1 Mask : reset

[016] Cmp1 Mask : and (boolean) with Source1

[017] Cmp1[M] : Multiplied with 0

[018] Cmp1[ ] : Count = 7119

[019] Cmp1[ ] : Sum = 457008

[020] Cmp1[ ] : Sum of Squares = 45153024

[021] Opened Source5 from C:\stereo\t\error_causes_test\Cmp1_Plain_7x7_possibly_too_large_range_to_save.png

[022] Source5 Mask : and (boolean) with Source2

[023] Source5[M] : Multiplied with 0

[024] Source5 Mask : reset

[025] Source5 Mask : and (boolean) with Source4

[026] Source5 Mask : inverted

[027] Source5[M] : Multiplied with 0

[028] Source5[ ] : Count = 19732

[029] Source5[ ] : Sum = 674224

[030] Source5[ ] : Sum of Squares = 49523968

[031] Source5[ ] : Count = 19732

[032] Opened Source6 from C:\stereo\t\error_causes_test\Cmp1_Plain_7x7_possibly_too_large_range_to_save.png

[033] Source6 Mask : inverted

[034] Source6 Mask : inverted

[035] Source6 Mask : and (boolean) with Source3

[036] Source6[M] : Multiplied with 0

[037] Source6[ ] : Count = 2006

[038] Source6[ ] : Sum = 160400

[039] Source6[ ] : Sum of Squares = 15695616

[040] Opened Source7 from C:\stereo\t\error_causes_test\Cmp1_Plain_7x7_possibly_too_large_range_to_save.png

[041] Source7 Mask : and (boolean) with Source3

[042] Source7 Mask : reset

[043] Source7 Mask : or (boolean) with Source3

[044] Source3[ ] : Inverted (binary)

[045] Source3 Mask : and (boolean) with Source7

[046] Source7 Mask : and (boolean) with Source3

[047] Source1[ ] : Inverted (binary)

[048] Source7 Mask : or (boolean) with Source1

[049] Source2[ ] : Inverted (binary)

[050] Source7 Mask : or (boolean) with Source2

[051] Source7[M] : Count = 24737

[052] Source7[M] : Sum = 1031888

[053] Source7[M] : Sum of Squares = 86522624

[054] Opened Source8 from C:\stereo\t\error_causes_test\Cmp1_Plain_7x7_possibly_too_large_range_to_save.png

[055] Source8[ ] : Count = 28990

[056] Source8[ ] : Sum = 1176288

[057] Source8[ ] : Sum of Squares = 99348992

[058] Source1[ ] : Count = 17008

[059] Source2[ ] : Count = 47538

[060] Source3[ ] : Count = 2258

[061] Opened Source9 from C:\stereo\t\depth_discont.png

[062] Source9[ ] : Inverted (binary)

[063] Source9 Mask : and (boolean) with Source2

[064] Source9 Mask : reset

[065] Source9 Mask : or (boolean) with Source2

[066] Source9 Mask : and (boolean) with Source3

[067] Source9[M] : Added 1

[068] Source9 Mask : reset

[069] Source9 Mask : and (boolean) with Source2

[070] Source9[M] : Added 1

[071] Source9[ ] : Count = 60744

Log of disparity error type testing

[001] Opened Left Source from C:\stereo\t\left.tif

[002] Opened Right Source from C:\stereo\t\right.tif

[005] Created New Disparity14Error Pair2 from sources Left Source and Right Source

Settings - Type: Multi Window Dist., Direction: Left, Window Size: 3, Left Search Width: 20, Right Search Width: 0

[006] Pair2 Disparity[ ] : Multiplied with 16

[007] Pair2 Error 1[ ] : Thresholded with value 2

[008] Pair2 Disparity Mask : and (boolean) with Pair2 Error 1

[034] Opened Source2 from C:\stereo\t\error_finder_test\Cmp1_Window7_NoPre.png

[035] Source2 Mask : copied from Pair2 Disparity

[036] Source2[M] : Count = 8409

[037] Source2[M] : Sum = 400192

[038] Source2[M] : Sum of Squares = 36793344

[042] Source2 Mask : inverted

[043] Source2[M] : Count = 20581

[044] Source2[M] : Sum = 776096

[045] Source2[M] : Sum of Squares = 62555648

[046] Undo of [007] Pair2 Error 1[ ] : Thresholded with value 2

[047] Pair2 Error 1[ ] : Thresholded with value 1

[048] Pair2 Disparity Mask : reset

[049] Pair2 Disparity Mask : and (boolean) with Pair2 Error 1

[050] Opened Source3 from C:\stereo\t\error_finder_test\Cmp1_Window7_NoPre.png

[051] Source3 Mask : copied from Pair2 Disparity

[052] Source3[M] : Count = 11534

[053] Source3[M] : Sum = 520912

[054] Source3[M] : Sum of Squares = 46363392

[055] Opened Source4 from C:\stereo\t\error_finder_test\Cmp1_Window7_NoPre.png

[056] Source4 Mask : copied from Source3

[057] Source4 Mask : and (boolean) with Source1

[058] Source4[ ] : Inverted (binary)

[059] Source4[M] : Sum = 6125

[060] Source4[M] : Count = 6125

[077] Opened Source5 from C:\stereo\t\error_finder_test\Disparity_Map_Basic.png

[078] Source5 Mask : copied from Source4

[079] Source5[M] : Filled masked region from previous unmasked

[080] Source5[M] : Filled masked region from nearest unmasked

[081] Source5 : Comparison created with Source1 as Cmp2

[082] Cmp2 Mask : and (boolean) with Source1

[083] Cmp2 Mask : inverted

[084] Cmp2[M] : Multiplied with 0

[085] Cmp2[ ] : Count = 28580

[086] Cmp2[ ] : Sum = 1134860

[087] Cmp2[ ] : Sum of Squares = 90467080

[090] Opened Source6 from C:\stereo\t\error_finder_test\Disparity_Map_Basic.png

[091] Source6 Mask : copied from Source3

[092] Source6[M] : Filled masked region from nearest unmasked

[093] Source6 : Comparison created with Source1 as Cmp3

[094] Cmp3 Mask : and (boolean) with Source1

[095] Cmp3 Mask : inverted

[096] Cmp3[M] : Multiplied with 0

[097] Cmp3[ ] : Count = 28939

[098] Cmp3[ ] : Sum = 1141434

[099] Cmp3[ ] : Sum of Squares = 91072300

Multi-window Distance test

[001] Opened Left Source from C:\stereo\t\error_finder_test\right.tif

[002] Opened Right Source from C:\stereo\t\error_finder_test\left.tif

[003] Created New Disparity14Error Pair1 from sources Left Source and Right Source

Settings - Type: Std. Minimum Sum, Direction: Left, Window Size: 3, Left Search Width: 20, Right Search Width: 0

[004] Opened Left Source from C:\stereo\t\error_finder_test\left.tif

[005] Opened Right Source from C:\stereo\t\error_finder_test\right.tif

[006] Created New Disparity14Error Pair2 from sources Left Source and Right Source

Settings - Type: Std. Minimum Sum, Direction: Left, Window Size: 3, Left Search Width: 20, Right Search Width: 0

[007] Pair2 Disparity[ ] : Multiplied with 16

[008] Pair2 Error 1[ ] : Thresholded with value 128

[009] Pair2 Disparity Mask : reset

[010] Pair2 Disparity Mask : and (boolean) with Pair2 Error 1

[011] Pair2 Disparity Mask : inverted

[012] Undo of [008] Pair2 Error 1[ ] : Thresholded with value 128

[013] Pair2 Error 1[ ] : Thresholded with value 750

[014] Pair2 Disparity Mask : or (boolean) with Pair2 Error 1

[015] Undo of [013] Pair2 Error 1[ ] : Thresholded with value 750

[016] Opened Source1 from C:\stereo\t\error_finder_test\Cmp1_Window7_NoPre.png

[017] Source1 Mask : copied from Pair2 Disparity

[018] Source1[M] : Count = 20820

[019] Source1[M] : Sum = 798800

[020] Source1[M] : Sum of Squares = 64646912

[021] Opened Source2 from C:\stereo\t\error_finder_test\Cmp1_Window7_NoPre.png

[022] Source2[ ] : Inverted (binary)

[023] Source2 Mask : copied from Source1

[024] Source2[M] : Count = 42323

[025] Pair2 Disparity[M] : Filled masked region from nearest unmasked

[026] Opened Source3 from C:\stereo\t\error_finder_test\truedisp.png

[027] Pair2 Disparity : Comparison created with Source3 as Cmp1

[028] Cmp1 Mask : and (boolean) with Source3

[029] Cmp1 Mask : inverted

[030] Cmp1[M] : Multiplied with 0

[031] Cmp1[ ] : Count = 21663

[032] Cmp1[ ] : Sum = 913184

[033] Cmp1[ ] : Sum of Squares = 73636864

Minimum window threshold test

[001] Opened Left Source from C:\stereo\t\error_finder_test\left.tif

[002] Opened Right Source from C:\stereo\t\error_finder_test\right.tif

[003] Created New Disparity14Error Pair1 from sources Left Source and Right Source

Settings - Type: 2nd Best Dist., Direction: Left, Window Size: 3, Left Search Width: 20, Right Search Width: 20

[004] Pair1 Error 1[ ] : Thresholded with value 2

[005] Opened Source1 from C:\stereo\t\error_finder_test\Cmp1_Window7_NoPre.png

[006] Pair1 Disparity Mask : and (boolean) with Pair1 Error 1

[007] Source1 Mask : copied from Pair1 Disparity

[008] Pair1 Disparity[ ] : Multiplied with 16

[009] Source1[M] : Count = 15201

[010] Source1[M] : Sum = 620864

[011] Source1[M] : Sum of Squares = 51801600

[012] Opened Source2 from C:\stereo\t\error_finder_test\Cmp1_Window7_NoPre.png

[013] Source2[ ] : Inverted (binary)

[014] Source2 Mask : and (boolean) with Pair1 Error 1

[015] Opened Source3 from C:\stereo\t\error_finder_test\truedisp.png

[016] Opened Source4 from C:\stereo\t\error_finder_test\Cmp1_Window7_NoPre.png

[017] Source4[ ] : Inverted (binary)

[018] Source4 Mask : and (boolean) with Source3

[019] Source4[M] : Inverted (binary)

[020] Undo of [019] Source4[M] : Inverted (binary)

[021] Source4 Mask : inverted

[022] Source4[M] : Multiplied with 0

[023] Source4[ ] : Inverted (binary)

[024] Source4 Mask : copied from Source1

[025] Source4[M] : Count = 22190

[026] Source4[M] : Sum = 22190

[027] Pair1 Disparity[M] : Filled masked region from nearest unmasked

[028] Pair1 Disparity : Comparison created with Source3 as Cmp1

[029] Cmp1 Mask : and (boolean) with Source3

[030] Cmp1[M] : Multiplied with 0

[031] Undo of [030] Cmp1[M] : Multiplied with 0

[032] Cmp1 Mask : inverted

[033] Cmp1[M] : Multiplied with 0

[034] Cmp1[ ] : Count = 27718

[035] Cmp1[ ] : Sum = 1243165

[036] Cmp1[ ] : Sum of Squares = 134018067

2^{nd} best distance log

[001] Opened Left Source from C:\stereo\t\error_finder_test\left.tif

[002] Opened Right Source from C:\stereo\t\error_finder_test\right.tif

[003] Created New Disparity14Error Pair1 from sources Left Source and Right Source

Settings - Type: Equivalent Right Map Dist., Direction: Left, Window Size: 3, Left Search Width: 20, Right Search Width: 0

[004] Pair1 Disparity[ ] : Multiplied with 16

[005] Pair1 Error 1[ ] : Thresholded with value 1

[006] Pair1 Disparity Mask : and (boolean) with Pair1 Error 1

[007] Opened Left Source from C:\stereo\t\error_finder_test\Cmp1_Window7_NoPre.png

[008] Opened Source1 from C:\stereo\t\error_finder_test\Cmp1_Window7_NoPre.png

[009] Source1 Mask : copied from Pair1 Disparity

[010] Source1[M] : Count = 19475

[011] Source1[M] : Sum = 905264

[012] Source1[M] : Sum of Squares = 85280512

[013] Opened Source2 from C:\stereo\t\error_finder_test\correct_count_tester.png

[014] Source2 Mask : copied from Pair1 Disparity

[015] Opened Source3 from C:\stereo\t\error_finder_test\truedisp.png

[016] Source2 Mask : and (boolean) with Source3

[017] Source2[M] : Count = 19475

[018] Pair1 Disparity[M] : Filled masked region from nearest unmasked

[019] Pair1 Disparity : Comparison created with Source3 as Cmp1

[020] Cmp1 Mask : and (boolean) with Source3

[021] Cmp1[M] : Inverted (binary)

[022] Undo of [021] Cmp1[M] : Inverted (binary)

[023] Cmp1 Mask : inverted

[024] Cmp1[M] : Multiplied with 0

[025] Cmp1[ ] : Count = 29144

[026] Cmp1[ ] : Sum = 1079248

[027] Cmp1[ ] : Sum of Squares = 76039424

Correspondence test log