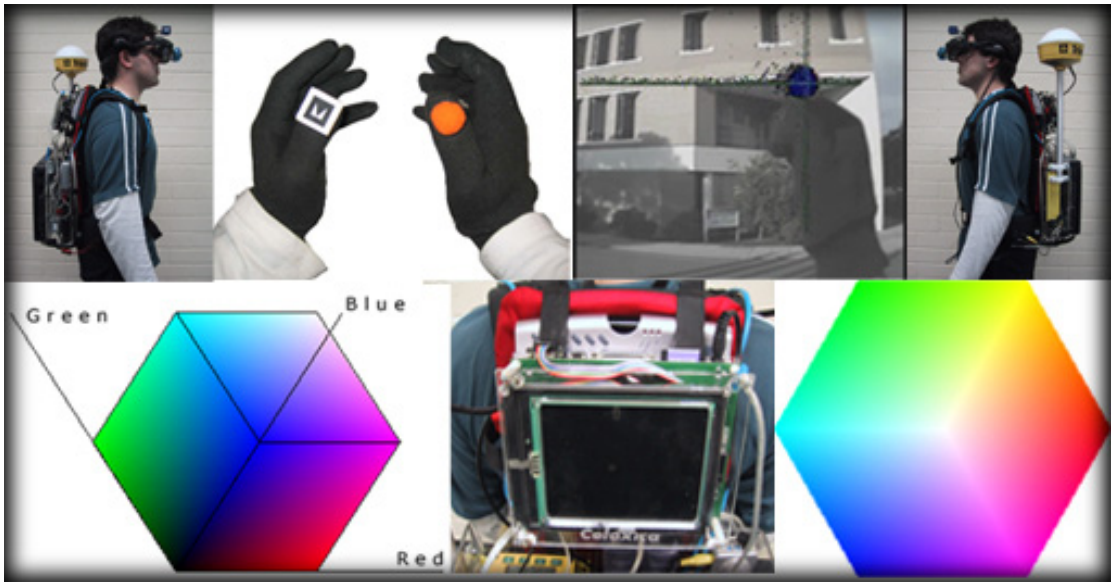
Mobile Hand Tracking Using FPGAs for Low Powered Augmented Reality



Honours Research Thesis

By Ross Smith

Bachelor of Information Technology (Software Engineering) (Hons)
University of South Australia
November 2004

Supervisors: Dr Wayne Piekarski and Mr Grant Wigley

Wearable Computer Lab
School of Computer and Information Science
Division of Information Technology, Engineering and the Environment
The University of South Australia

Abstract

Reconfigurable hardware has been used in various application domains for improved performance, ease of implementation, and application prototyping. One domain that would greatly benefit from these advantages is augmented reality, which is the combination of both computer generated graphics and a view of the physical world using a head mounted display. Outdoor augmented reality systems such as Tinmith, developed at the University of South Australia, use the operator's hands to interact with computer generated objects. Currently the ARToolkit software library is used to find the location of the user's thumbs relative to a head mounted camera. A computer generated cursor is then overlaid on the user's thumbs which is used for controlling menu options and object manipulations.

The existing tracking software uses complex image processing algorithms to calculate the location of fiducial markers with six degrees of freedom. The overhead of these algorithms is a burden to Tinmith's limited mobile computer resources. An alternative solution is to offload the tracking process to dedicated hardware that operates independently of the rest of the system. Application specific integrated circuits are a well known category of customised hardware and could be used to provide algorithmic speedup and lower power consumption. However they are not always desirable as they can be very costly for short production runs, require large amounts of engineering expertise to design, can take many months to design and verify, and can not be modified once fabricated. Another type of hardware available is a field programmable gate array, which still provides similar benefits but may be re-programmed through software. It was determined that a field programmable gate array is a more flexible and suitable solution for this application, where an iterative design process during implementation of a vision tracking algorithm is required. This has also allowed us to experiment with a number of different algorithms and assess their performance in different environmental conditions.

When Tinmith generates the cursors overlay it requires 2D coordinates from the vision tracker to render the cursor in the correct location. A problem with ARToolkit is the black and white markers used do not work well outdoors particularly when in direct sunlight. Instead of using complex template matching techniques, I propose to use simple blob tracking to help overcome the above limitations. A common approach to blob tracking is to use a uniquely coloured marker and extract its location from an image. Statistical filters such as mean, median or mode can be used to find the centre of mass providing the required X and Y coordinates. To allow the new tracker to work in a wide range of lighting

conditions, the segmentation process can be adapted to use a different colour model other than RGB. Models such as YCrCb, YIQ or YUV separate out one channel which is dedicated to processing luminance (brightness) information.

This dissertation describes in detail the technical processes and steps used for the implementation of such an algorithm for the implementation on an FPGA. It presents the research completed, provides design details and documentation of each of the components developed as well as a performance evaluation performed on the Tinmith wearable backpack computer.

Declaration

I declare that this thesis does not incorporate without acknowledgment any material previously submitted for a degree or diploma in any university and that to the best of knowledge it does not contain any materials previously published or written by another person except where due reference is made in the text.

_____

Ross Smith

November 2004

Acknowledgements

During my time at the University of South Australia I have been lucky enough to work with the extremely talented people from the Wearable Computer Laboratory. They have been kind enough to provide me with the equipment and technical knowledge to support my research. I especially wish to thank my supervisors Wayne Piekarski and Grant Wigley who have guided and supported me throughout my degree. I would also like to thank Bruce Thomas, Ben Close, Ben Avery, Aaron Toney and Aaron Stafford for their support and friendship.

Table of Contents

List of Figures

List of Tables

1 Introduction

Wearable computing is a relatively new research area and as computers have become more powerful and reduced in size, wearable computing has become technologically viable and more socially acceptable. A wearable computer is a self powered computing device that can be worn on the body without requiring the hands to carry it and can be used while performing other tasks [32].

Another closely associated research area is augmented reality (AR). AR is the process of overlaying computer generated images on to the physical world. This allows the user to view both the physical world and the computer generated world at the same time [1]. One method of presenting AR to the user is with an optical Head Mounted Display (HMD) (as depicted in Figure 1). Another technique that may be used is video overlay, where a video camera is used to capture the physical world and a computer is used to render the augmented overlay, with the two combined and then rendered on to a HMD (as shown in Figure 2).

The aim of the research presented in this thesis has been to explore the feasibility of developing a hardware based hand tracking system which will be used as an input device to the Tinmith AR modelling system which allows creation and editing of 3D geometry outdoors [31]. The Tinmith AR system was developed at UniSA by Piekarski et al. [33] consists of a mobile backpack computer and head mounted display, as depicted in Figure 3.

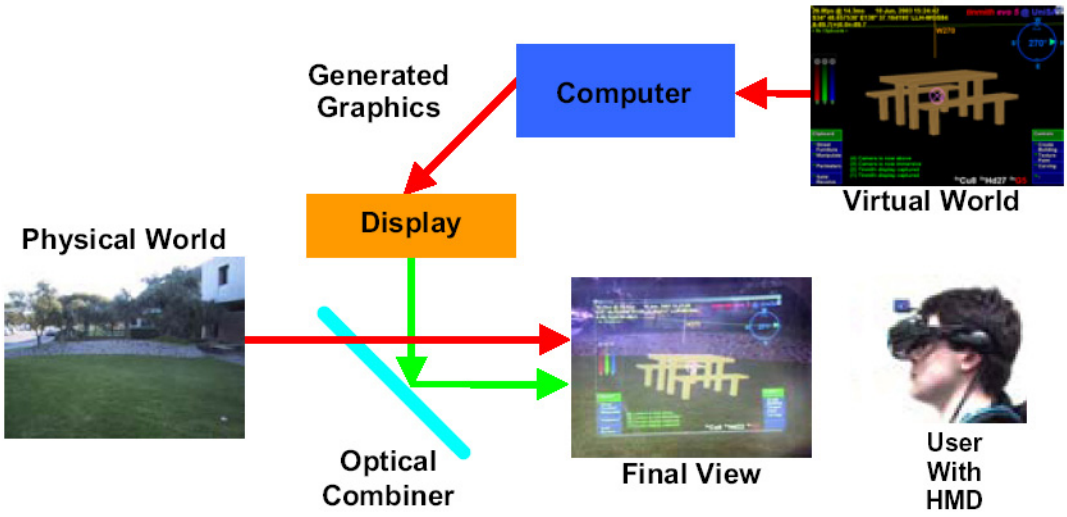


Figure 1 - Augmented reality using optical see through combination
(Image courtesy of Wayne Piekarski – University of South Australia)

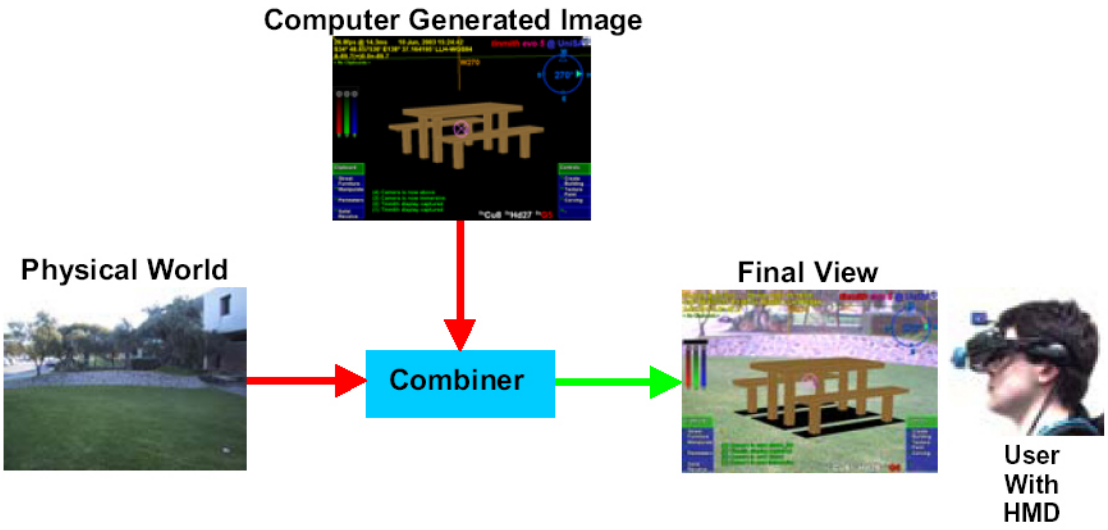


Figure 2 - Augmented reality using video see-through combination
(Image courtesy of Wayne Piekarski – University of South Australia)

To interact with the Tinmith modelling software, the user wears a set of gloves with metallic contacts that controls a custom user interface. Attached to each glove is a unique fiducial marker, and through a process known as pattern matching, the position of the hands relative to the head can be calculated. A user can control interactions by touching their fingers to their thumb, and this performs a discrete operation similar to a keyboard or mouse click. However, the pattern matching algorithms currently used by Tinmith,(as well as other

Figure 3 - The Tinmith 2004 wearable backpack AR system

vision tracking systems) are computationally complex and require a high end microprocessor to operate. With high end microprocessors consuming large amounts of power, the operation is limited to a heavy laptop computer with large batteries. Since Tinmith is designed to be carried by a user outdoors, reducing the size and weight of the hardware is very desirable.

The existing Tinmith backpack system (like many others which support similar tasks) uses a laptop computer to generate 3D computer graphics, track the location of fiducial markers, capture a video stream, and combine the resulting generated environment and video stream for an output to the HMD. In an effort to minimise the laptops usage I have developed a customised hardware device to perform the tracking. This allows us to use a smaller and more efficient laptop and may one day allow us to remove it altogether.

Developing a custom hardware solution for tracking will reduce power consumption by reducing the processor size required on Tinmith's laptop and allow the use of smaller wearable computers, while still retaining similar if not better tracking quality. However, the development of a custom hardware solution is very expensive and requires teams of experienced hardware designers to implement. An alternative to a custom hardware solution is the use of a field programmable gate array (FPGA). An FPGA is a hardware device that can have its architecture configured by the use of software to suit the application at hand. The advantage of an FPGA is they are small and can be low powered devices that can be easily re-programmed to iteratively refine the implementation.

The aim of this research has been to develop an FPGA based hardware hand tracking system that can be easily integrated into the Tinmith augmented reality system. This is useful as it takes one of the most computationally complex tasks away from the general purpose laptop processor, thus reducing the overall power consumption. This results in the ability to transfer Tinmith to a more portable computing platform that is smaller and lighter, such as a hand held computer.

The objective has been to supply a small physical hardware device that can be easily mounted on the backpack and interfaced to the Tinmith software. The hardware device consists of an FPGA mounted onto a RC200 supplied by Celoxica [6], a commercial supplier of reconfigurable computers. Connected to the FPGA is a video camera that provides images of the gloves to be tracked, and a serial cable that is used to transmit

tracking information to Tinmith. The FPGA is configured with a custom designed hardware circuit that performs the hand tracking algorithm and transmits the results via the serial port.

After having outlined the problem in this chapter, Chapter 2 introduces FPGAs and the current stare of the research domain. This includes a summary of the design and architecture of FPGAs followed by programming languages and previous applications developed. The next section of this chapter presents a definition of augmented reality and the associated tracking techniques used within the field. It examines the advantages and disadvantages of the current approaches to tracking and why these are not suited to hardware implementation. The next sub section presents the main techniques used within the vision tracking research domain as well as a summary of applications. Finally a discussion of alternate colour models and how they might be used to improve the previously described segmentation process.

Chapter 3 explains the reasoning behind each of the design decisions made. This includes a description of the algorithms considered for the tracking process, with a summary of the advantages and disadvantages of the mean, median and mode statistical functions. This is followed by a justification for the mean function in the final implementation. Next, the design of the tracking marker properties such as size, shape and surface are considered.

Chapter 4 explains the implementation phase, which includes the platform and language selection used followed by a detailed description of my software architecture. The architecture section explains my unique design used for an application designed on an FPGA. The four main process flows are explained describing the parallelism which is achieved in the final implementation.

Chapter 5 presents the results for a variety of different conditions. A range of different coloured markers were tested to evaluate their performance and measure robustness. This also includes the methodology I used for the tuning each of the different coloured markers.

Finally, Chapter 6 concludes this thesis with a summary of each section's findings, an overview of the resulting tracking solution, and a description of the contributions made. This chapter also summarises the significance of this work and a proposal for future work.

During the research and development stages I have published a number of fully refereed international conference papers contributing to the augmenter reality and user interface research areas[33, 34, 44].

The contributions I have made can be summarised as the following:

- Chosen suitable vision algorithm
- Used YCrCb colour space
- Implemented a working version of the tracker on the RC200 FPGA Platform.
- Integrated with the Tinmith system
- Improved tracking performance and robustness compared to ARToolkit
- Developed a low powered stand alone tracking solution
- Performed testing outdoors to find superior colours and measure accuracy of operation

2 Background

This chapter introduces four research areas which are relevant to this thesis: reconfigurable computing, augmented reality, vision tracking and colour models. First I begin by defining field programmable gate arrays and explain what a reconfigurable computer is, common architecture designs, specific programming languages used, and previous research in the field. The next section defines AR and provides a general summary of the research area. The following section introduces tracking techniques as well as some existing applications which have been developed. After these technologies have been explained I summarise the common vision tracking techniques currently used and how alternate colour models can be can be incorporated with existing vision tracking methods.

2.1 Field Programmable Gate Arrays

Hardware solutions have been a used widely for a variety of applications for many years. General purpose computers do not always provide the best software solution; some tasks can be performed in dedicated hardware with improved performance. One of the biggest advantages offered by dedicated hardware is its parallel processing capability. Algorithms can often be modified to operate in parallel modules which can greatly increase their execution speed. This is often the case with real time applications where algorithmic speed-ups can be achieved. Application Specific Integrated Circuits (ASIC) have been traditionally used for hardware solutions designed for a specific application. However ASICs are not desirable in all situations as they can be very costly for short production runs, require large amounts of engineering expertise to design, can take many months to design and verify, and can not be modified once fabricated [42]. An alternative to an ASIC that does not have these drawbacks but retains a similar algorithmic speedup is a field programmable gate array (FPGA). This makes FPGAs an appealing option for many applications, particularly where an iterative design process is desirable.

A reconfigurable computer (RC) combines the use of an FPGA and a standard microprocessor. RCs are available with many features such as onboard RAM, smart cards, network interfaces, video and audio, and external communications ports, and all connected to an FPGA device. Celoxica [6] produce a range of different prototyping platforms, and these boards are well suited to applications where fast prototyping is a required by programmers with minimal hardware design experience.

The biggest limitations of FPGAs are their inability to implement floating point numbers and division. In the past when the size of FPGAs was much smaller, both floating point and division were avoided altogether but as FPGAs have become denser it is now becoming possible. Reconfigurable hardware is frequently used together with general purpose processors and areas of an application which do not perform well on reconfigurable hardware are executed on a host processor.

2.1.1 Architecture

This section explores common architecture designs of FPGAs. They consist of three main components [14] combinational logic blocks (CLB) also called Logic Blocks (LB), programmable connections and input output (I/O) blocks, as shown in Figure 4. There has been a great deal of research on the design of the computational elements built into an FPGA, and it has been well established that the best functional block design for FPGAs used for random digital logic is an N-input lookup table (LUT) [10]. A typical CLB has one or more 4 input LUTs, D flip-flops, and fast carry logic, and any general logic function can be programmed into LUTs. FPGAs have many CLBs which are inter-connected through programmable connections, and involves complex routing algorithms. Finally, I/O blocks give us access to external pins and are used for bus communications and connecting to other processors, sensors, and other ASICs.

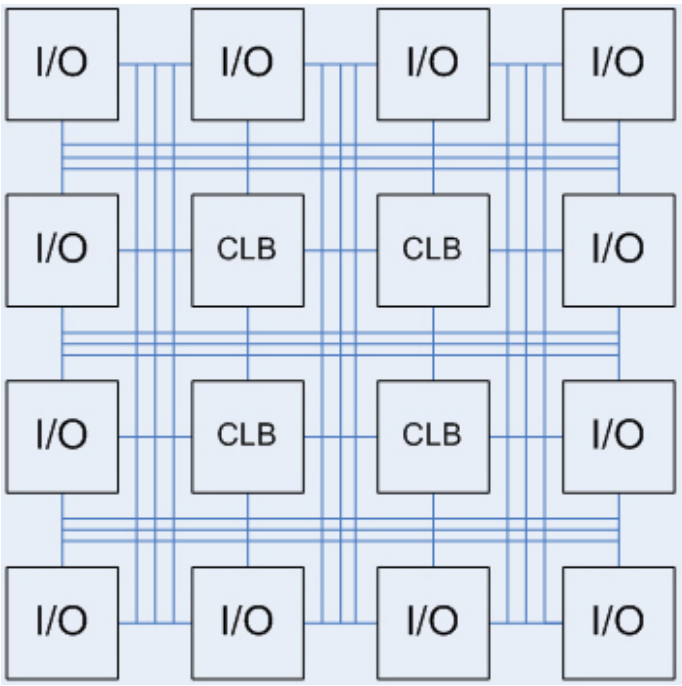


Figure 4 - High level architecture diagram of a general FPGA demonstrating I/O blocks, CLBs and programmable routing connections

The CLBs of an FPGA vary in complexity from simple three input blocks to 4 bit ALUs; this complexity is referred to as the CLB granularity. An example of a very-fine grained logic block implementation is the Xilinx [52] 6200 series FPGAs which are capable of performing any two-input function. This type of block is well suited to bit manipulation tasks but is very inefficient for multipliers. Medium-grained logical blocks are capable of performing multiplication more efficiently, have greater numbers of inputs, and can be used more efficiently for a wider variety of operations. Finally coarse-grained architectures are well suited to word-width datapaths and domain specific applications. Unfortunately, when performing one bit operations the course grained architecture will waste more area and suffer a slower speed.

Programmable connection blocks are used to connect the CLBs and I/O pins on an FPGA. There are many different correct ways in which connection blocks can be configured to provide a working solution. Usually this process is done by the automated routing process which needs to consider the most efficient way the connections can be made so the application designer does not consider how the connections are made.

The routing connections are usually laid out in a two dimensional fashion as depicted in Figure 5. Optimal two dimensional designs are particularly hard to calculate as there are a large number of possibilities for the placement and routing software to calculate. FPGA routing is commonly conceptualised as a graph with cost associated with the different routing paths which are chosen and used to indicate the usage of the device. One way of simplifying the problem is to consider the routing as a one dimensional array as demonstrated by [19] where 1D connections are made, also depicted in Figure 5. This unfortunately can lead to using all the available connections before the circuit is complete, where a two dimensional approach would have been able to connect the blocks more efficiently. Other systems use combinations of both axes (such as Ebeling et al. [15]) who use a technique where the majority of word length connections are made along the horizontal axis while other interconnections are made on the vertical axis to provide a more optimal solution. Another interesting approach was presented by Tessier [47] who proposed using an A* search instead of an exhaustive depth first search to find an optimal solution. They have shown their system can reduce routing runtime substantially, with minimum track counts in most cases.

2.1.2 Programming Languages

Hardware Description Languages (HDL) have been traditionally used to program FPGAs. The most commonly known languages are Very High Speed Integrated Circuits Description Language (VHDL) and Verilog. Recently there has been the development of new HDLs based on subsets of common programming languages such as C and Java. Some examples of these are Handel-C [6], System-C [4] and Join Java [20]. This section of the report looks at some of the advantages and disadvantages of each of these options.

VHDL is one of the most well known hardware description languages, and was originally developed by the US Department of Defence to documenting electronic systems. Soon VHDL became an IEEE specification standard used to describe and simulate chip designs before fabrication. VHDL has been a popular language used for describing FPGA designs since Diamond et al. [12] and Hossack et al. [22] proposed the use of VHDL for FPGA design.

Verilog is a traditional HDL first designed by Phil Moorby [49] in 1985 and extended substantially in 1987. One of the most appealing features was the XL-Algorithm an efficient method of doing fast and efficient gate level simulation. Synopsys developed the first logic synthesizer which used Verilog as an input language, and designs may be represented as a netlist (describing the routings) and modelled behaviourally and translated into gates through Synopsys. At this time one of the biggest uses for Verilog was for sign off certification by ASIC vendors, and it was not until later that authors Gannot and Ligthart [18] proposed the use of Verilog for FPGA design. Although this is an abstraction from the gate level, Verilog is considered a low level language by today's standards, especially when comparing it to languages such as Handel-C.

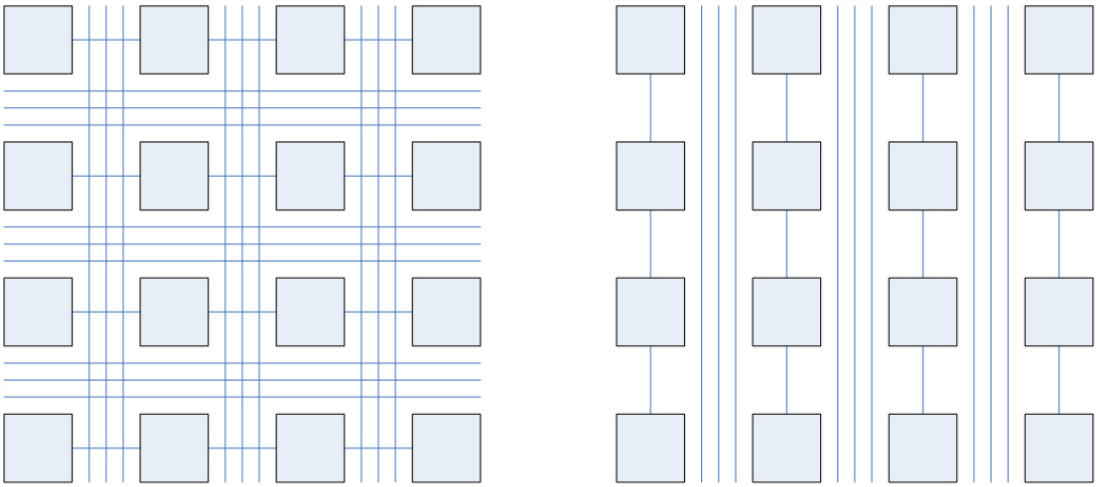


Figure 5 - (Left) A two dimensional routing grid, (Right) A one dimensional routing grid

Celoxica [6] have commercialised Handel-C a HDL customised form hardware development based on the ANSI-C specification. The advantage with Handel-C compared to traditional HDLs is the ability for software engineers to use it with minimal hardware knowledge. A suitable analogy would be comparing Handel-C to Java and VHDL to assembly programming, solutions can be developed rapidly but they may not be optimal in terms of area use in the FPGA. However as fabrication technologies have advanced and more dense FPGAs are available it can be argued that space is no longer a concern and the time saved makes Handel-C an excellent development option.

SA-C [14] is another C based programming language designed as a high level language which is used to generate hardware logic. It has been designed to develop loop level and instruction level parallelism. There are three main differences between SA-C and ANSI-C. SA-C includes data-bit precision exploiting the FPGAs ability to have arbitrary length precision. It includes extensions to C supporting parallel looping, and it removes pointers and recursion. Parallelism of loops is achieved by unrolling loops and allocating separate logic for each iteration of a loop. This technique is quite powerful but also uses large amounts of the FPGAs area and can not always be used depending on the algorithm design.

Another interesting contribution to FPGA programming languages is Join Java [20]. It provides an interface based on standard Java classes to the software side of a design. With the growing popularity of Java this allows a larger audience to be able to write hardware related code. The compiler is capable of parsing a restricted subset [21] of Java code which generates VHDL, which can then compiled and synthesised using other tools.

2.1.3 Previous FPGA Applications

Reconfigurable computers have been used for many different applications, in this section I will present a brief summary of the most prominent applications. This will include research prototype applications and those that are commercially available.

There are many implementations used for specific electronic tasks which I have classified under signal processing. These may not strictly be applications in a software sense but nonetheless deserve a mention. One example is where an FPGA was used to provide a system bus between an array of microcontrollers [28]. The authors improved performance compared to using microcontrollers for the bus architecture and also found the design time was reduced. Cheung et al. [7] used an FPGA for processing audio signals providing digital

to analogue conversion at 96 kHz in a 24 bit configuration, operating at a high quality level supporting CD and DVD audio.

Drapier et al. [14] demonstrated how the use of SA-C can be used to quickly develop image processing applications. They compared the performance of a Xilinx XV2000E and a Pentium III 800MHz whilst performing common image processing functions such as scalar addition, Prewitt, Canny, Wavelet, Dilates and Probing filtering algorithms. They measured performance increases between 10 and 800 times depending on the complexity of the task. Computers may have become faster since this was published in 2002 but they also stated FPGAs also follow Moore's law the same as general purpose microprocessors.

The use of FPGAs in image processing has even been used for processing in space [11]. Dawood et al. discuss natural disaster monitoring and detection using an FPGA from space where issues such as radiation, payload and performance are all considerations. They discuss the common image processing tasks with implementation details of a Gaussian and convolution filters for their system.

Another fruitful area of research is bio-technology and recently studies have been made on the feasibility and benefits FPGAs can provide. Gene sequence searching involves databases with millions of elements for matching and partially matching patterns. Puttegowda et al. [40] presented a system using the Smith-Waterman algorithm implemented using a number of different architectures. Their design included the use of runtime reconfiguration which increased their final performance of up to a trillion cell updates per second providing an order of magnitude improvement when compared to commercially available systems.

In the wearable computing and AR domains, the use of FPGAs is still quite rare. This is perhaps due to the extra complexity of implementing hardware in VHDL or Verilog rather than software. Plessel et al. [38] described a wearable system that performs simple tasks such as audio and video decoding through the use of reconfigurable modules located on an FPGA. As particular applications are required, the FPGA loads the appropriate hardware module and performs the task in hardware. Luk et al [26, 27] used a reconfigurable computer to support basic functions for AR applications: video mixing, image extraction and object tracking. The image extraction and object tracking stages in this system were performed using a fixed position camera, which significantly reduces the difficultly in performing these tasks. Matsushita et al [29] described ID Cam, which uses custom

hardware and high speed cameras to extract identification codes from flashing beacons in a scene. The camera contained custom silicon to perform most of the high speed extraction, and an FPGA was used to process the final result.

As the size available on FPGAs is increasing at such a fast rate, as per Moore's law, an increasing number of reconfigurable applications are being developed. Wigley et al. [51] proposed the first operating system designed for a reconfigurable computer used to manage multiple applications. They have considered the essential components of a traditional operating system: loader, scheduler, virtual memory, cache management, inter-process communications, and carefully considered how they can be adapted to a reconfigurable computer.

FPGAs have also been used to show performance increases in areas of cryptography. Data Encryption Standard (DES) has been used for over 25 years and is still considered a highly secure encryption standard. It effectively uses one 56bit key for both encryption and decryption. Cheung et al. [8] assessed the feasibility of using a CPLD device to increase the performance of the DES algorithm. They state there are 16 logical rounds in the DES algorithm that can be performed in one logical component to perform the entire DES algorithm, which is well suited to hardware implementation. Finally, Bednara et al. [3] demonstrated the benefits FPGAs provide regarding field multiplication specifically for performing elliptical curve cryptography.

2.1.4 FPGA Summary

FPGAs are a growing area of research and recently as it is becoming more well known, applications are being developed frequently providing accelerated application execution times and performance. FPGAs provides benefits compared to traditional ASICs which were limited to an elite group of hardware designers. With the development of C and Java based languages, reconfigurable computers can now be used by computer programmers with little or no hardware experience. I have presented here an introduction to reconfigurable computers and the current state of the research based on some of the more prominent contributions as an effort to better understand the field. It is clear that further research needs to be done in order the fully exploit the benefits reconfigurable computers have to offer.

2.2 Augmented Reality

AR is the process of combining computer generated graphics registered with a real world view [1]. The concept of computer generating artificial stimulus was first proposed by Ivan Sutherland [45] in 1965 with his seminal paper 'The Ultimate Display'. He proposed the concept could be used to present graphical information which does not necessarily follow the laws of physical reality and one could 'See through matter' using this technique. Soon after in 1968 Sutherland [46] developed the first optical Head Mounted Display (HMD) which was capable of projecting computer generated images over the real world as depicted in Figure 6.

Another similar technology is Virtual Reality (VR) shown in Figure 7, first coined by Jaron Lanier, whose research explored the use of entirely virtual worlds with more that one person interacting at one time in them. Both VR and AR use hardware trackers to record a user's motion. Tracking the location of the users head position is used to render computer graphics that are aligned correctly with the real world.

Other parts of the body can also be tracked and used for controlling the computer generated output. The Tinmith backpack computer uses the user's thumbs to control menus

Figure 6 - Augmented reality demonstrating a virtual table and chairs anchored relative to the physical word view
(Image courtesy of Wayne Piekarski – University of South Australia)

Figure 7 – Example of a virtual reality world
(Image courtesy of Ben Avery – University of South Australia)

and perform manipulations of 3D objects [37]. There are many different techniques used for tracking. A summary of the different types include mechanical, accelerometers, gyroscopes, ultrasonic, passive magnetic, active magnetic, Global Positioning System (GPS), optical and vision based tracking. Previously Piekarski [32] found that vision based systems are well suited to the hand tracking problem. The problem is particularly difficult when working outdoors with known problems such as power consumption, size, weight, and support infrastructure.

2.2.1 Tracking Techniques

As mentioned previously tracking allows us to register the computer generated environment with the real world view. There are a many different types of tracking systems available and in this section I summarise the available techniques.

Sutherland developed the first mechanical tracker as part of his initial work in [46]. The tracker consisted of a mechanical arm with one end fixed to the ceiling and the other to the user's head. Through the use of sensors at the joints in the mechanical arm it is possible to calculate the location and orientation of the users head. This provides very accurate tracking of location and orientation but is limited to a fixed area. It is also heavy and uncomfortable for the user.

Polhemus [39], a well known manufacturer of magnetic trackers, produce a range of tracking systems based on active magnetic fields. These systems transmit a magnetic field from a base station, which is constructed of three coils which sequentially pulse a magnetic field. The strength and the orientation of the field is measured by sensors placed on the objects to be tracked. These types of sensors provide both location and orientation with excelent accuracy. One of the disadvantages with active magnetic trackers is they only operate over short distances which limits their flexibility.

Accelerometers are designed to measure linear accelerations and are completely sourceless. They work by measuring small capacitive differences of a comb drive as movement occurs. A comb drive is a Microelectromechanical system (MEMS) [43] consisting of two plates with which look like a hair comb attached to springs. To achieve tilt sensing three of these are arranged along the X, Y and Z axis so they can measure Earth's gravity vector.

Gyroscopes measure rotational forces and provide orientation through integration. Traditional gyroscopes were constructed using a wheel which spins around an axis. As motion occurs the forces perpendicular to the axis can be measured. Recently the development of MEMS based gyroscopes has reduced their size significantly.

Figure 8 - Inertia Cube 2 hybrid tracker
(Image courtesy of Wayne Piekarski – University of South Australia)

Sutherland [46] also developed an ultrasonic based tracker that was not tethered, it worked by sending pulses of ultra sonic sounds and measured the time taken to reach sensors on the ceiling. Although this tracker provided a position the orientation is more difficult to obtain. Foxlin et al. [17] discussed that in the Constellation tracking system, the orientation values are combined with accelerometers and gyroscopes using a Kalman filter to achieve smooth the output.

There are a number of hybrid technologies which combine some of the above tracking techniques to provide more robust tracking solutions. Intersense [23] provide a tracking solution which uses accelerometers, magnetic sensors and gyroscopes all together. The Intersense Inertia Cube 2 uses a combination of nine sensors in total, and combines the tracking information of all of these sensors through a Kalman filtering algorithm which corrects accumulated errors and smooths outputs.

2.2.2 Vision Tracking

There has been much previous research in the vision tracking domain and in this section I will explain common vision tracking techniques and present examples of each of these. A common approach to vision tracking uses a camera to provide a stream of digital or analogue images which are analysed for particular characteristics.

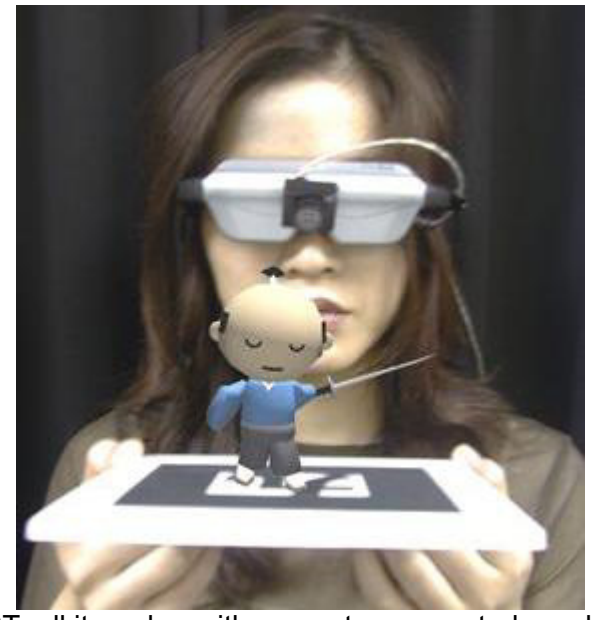


Figure 9 - ARToolkit marker with computer generated graphics overlayed
(Image courtesy of Mark Billinghurst – University of Washington)

Figure 10 - Fiducial marker as used by ARToolkit

Since this thesis is about tracking the location of a user's thumbs in real-time, this section focuses on techniques which can be performed without excessively slowing the speed of images captured. Many existing tracking techniques are based on one of the following methods: edge extraction, region based correlation or template matching, and segmentation techniques [5, 41]. Region-based correlation compares known templates to the video stream to locate pre defined markers. ARToolkit [25] is an example of template matching, and uses fiducial markers show in Figure 9 and Figure 10, to calculate a cameras position and orientation relative to the marker.

Edge extraction is another commonly used technique in video and image processing sharp changes in the brightness of an image are used to determine the location of edges. When edge extraction is performed on an image, a new image is generated with just the edges shown, as depicted in Figure 11. A problem with edge extraction is surface textures can be interpreted incorrectly as edges, particularly when the texture has many different colours and brightness levels. Edge extraction is often used as one of the techniques in tracking, however there are usually other steps involved which match particular patterns and shapes to find the object of interest.

The segmentation process separates a particular colour range of an image. An object of interest can be tracked through the use of colour alone, as demonstrated by [41]. For example, when working in the RGB colour model it is possible to separate a range of orange by accepting all the pixels which meet the criteria ($250 < Red < 255$) and ($120 < Green < 130$) and ($0 < Blue < 5$). The results of this segmentation process are shown in Figure 12.

Figure 11-(Top) input image before edge detection. (Bottom) Output image after edge detection performed

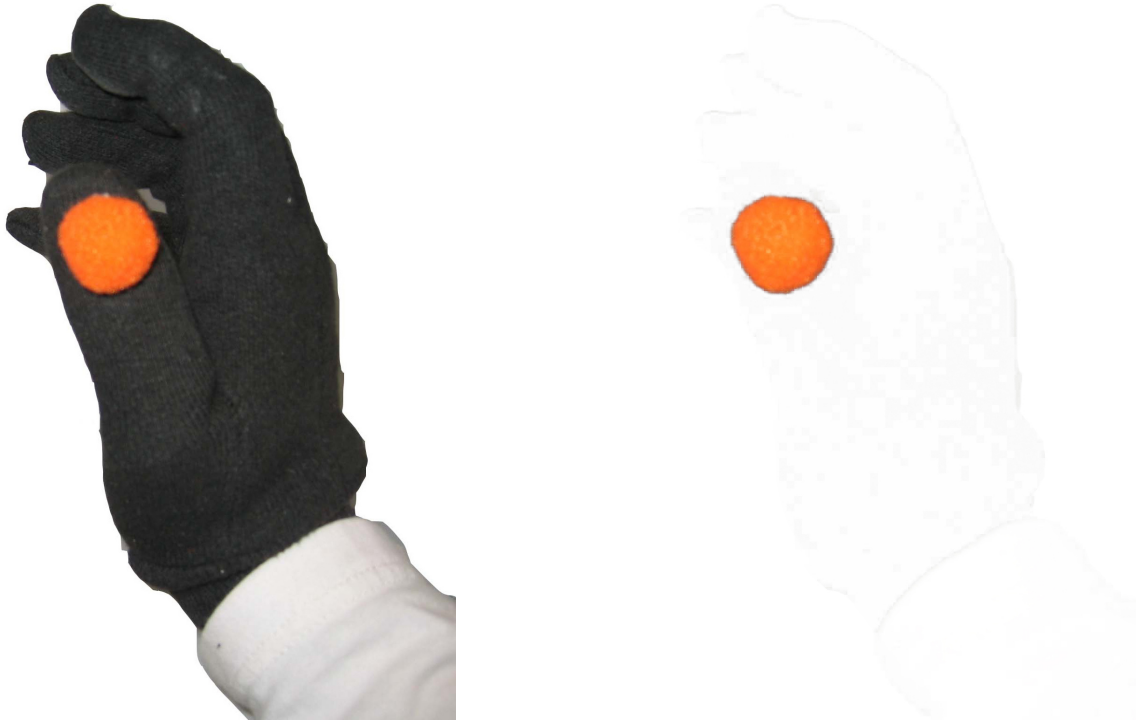


Figure 12 – (Left) Orange marker used to indicate the location of the thumb. (Right) Image after segmentation performed with threshold set to orange colour range

2.2.3 Previous Vision Tracking Work

This section provides a summary of vision tracking applications and ideas published previously which are relevant to this thesis. Rasmussen et al. [41] reviewed a number of different tracking techniques including edge detection, region based correlation, and blob tracking. They explain that the tracking of simple blobs is much simpler than other techniques, and I believe that this will assist with its implementation on an FPGA. The authors described how most existing algorithms for blob tracking rely on static or selective colour distribution to segment an image accurately. They defined a custom colour space to

assist with an accurate threshold but did not compare against other already available colour spaces which might be easier to implement.

Brusey et al [5] discussed the problems with recognizing images alone in many colour spaces, and that in all cases there are different objects which are not distinguishable. They present a decision tree approach with a custom colour space that separates brightness, using individual colour channels to make decisions.

There are a number of universities who participate in the RoboCup robot soccer competitions [2, 50]. These competitive events rely heavily on the tracking of coloured markers for estimating the positions of all the robot players. Since the lighting conditions are fixed, many of the competitors perform simple segmentation in RGB space, although other colour spaces are also used. For other applications of these trackers, Jebara et al. [24] implemented a system which allows a user wearing a HMD to visualise predicted ball motions in a game of billiards. A vision tracking system was used to automatically capture the locations of the balls in real time.

Cipolla [9] implemented an indoor vision tracker using motion parallax to estimate the 3D pose of gloves worn by a user. Dorfmuller-Ulhass et al. [13] described the use of blob tracking with retro-reflective markers and an infra-red light to detect the rotations of various joints in the hands. They discuss how they initially used rings around the fingers, but found that with blobs the centres were much easier to locate.

2.3 Colour Models

The properties or behaviours of colours are expressed through what is known as a colour model. There are many different colour models available because there is no one model which can be used to represent all aspects of colour. So in order to express particular characteristics of colours there is a range of different models available. Selecting an appropriate colour model requires close analysis of the system requirements and operating conditions.

Many image processing techniques can be adapted to use different colour models. Segmentation in particular can operate with much better accuracy as it relies on separating particular colours from an image finding a particular colour range. It would be beneficial to the segmentation process if the brightness of the image was considered on a separate channel to the colour information. In particular this improves performance in outdoor conditions where sunlight brightness is changing constantly.

In this section I explain some simple physics of light, colour and commonly used terms when describing colours. This is followed by a detailed description of the RGB, CMY, YIQ, YCrCb and HLS colour models.

2.3.1 General Characteristics

Light has many different characteristics which need to considered. In physics, colours are represented by varying frequencies of electromagnetic radiation. Figure 14 shows the visible frequencies and colours they map to while Figure 14 shows a broader range of electromagnetic frequency mappings. For the purposes of this research I am interested in the visible and infrared frequencies seen by video cameras. Besides frequency there are other important properties of colour, namely hue, saturation and brightness [16]. Brightness refers to the total light energy, and is also referred to as the luminance of a light. Saturation measures how close a colour is to a pure spectral colour, thus pale colours have a low saturation whereas pure red has 100% saturation. Hue measures the dominant wavelength, however there is not always a dominant wavelength as in the case of brown where a combination of colours contribute.

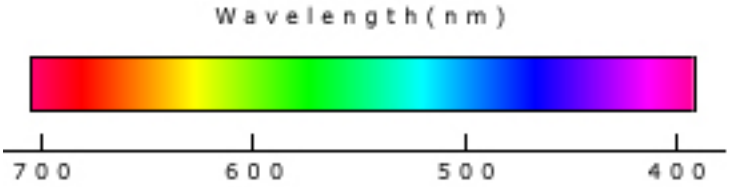


Figure 13- Visible colours and associated frequencies

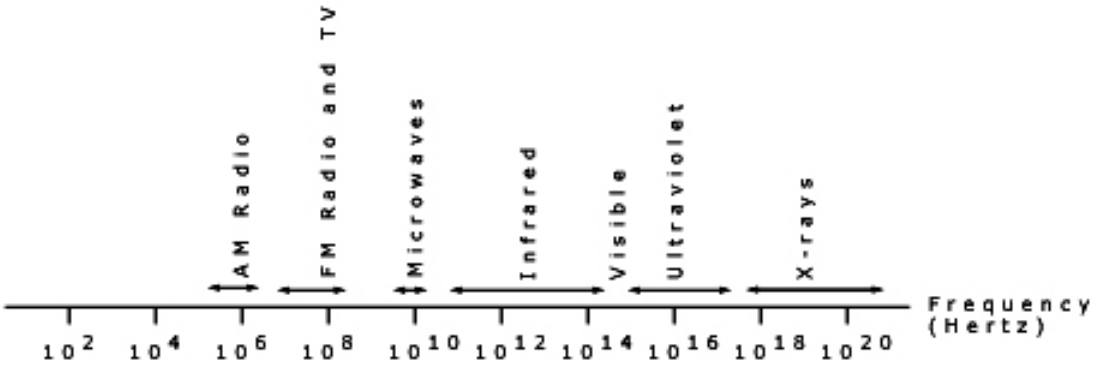


Figure 14 - Electromagnetic frequencies and corresponding names

2.3.2 RGB and CMY Colour Models

The red, green and blue (RGB) colour model is one of the most commonly known models. The RGB colour model is based on the tristimulus theory [30] which describes how humans interoperate colour. There are three visual pigments of colour in the cones of the retina: one of the pigments is most sensitive to a wavelength of around 630nm (red), the second is most sensitive to 530nm (green) and the last most sensitive to 450nm (blue). Our brain compares the intensities of each of cones and perceives a colour.

In computer graphics the three colour components are often represented by a value from 0 to 255, where 0 is no contribution and 255 is the greatest intensity. It is also common to represent the RGB model as a colour cube where each colour channel is mapped to axes, as shown in Figure 16 and Figure 15. The diagonal of the cube is where each colour channel adds equal contributions to produce greyscale colours, shown in Figure 15. In order to better understand the RGB colour model, looking at the cube from different angles shows which colours are generated along the different axes, depicted in Figure 18 and Figure 17. The varying contributions of each of the colour channels produce all the possible colours available in the RGB colour space.

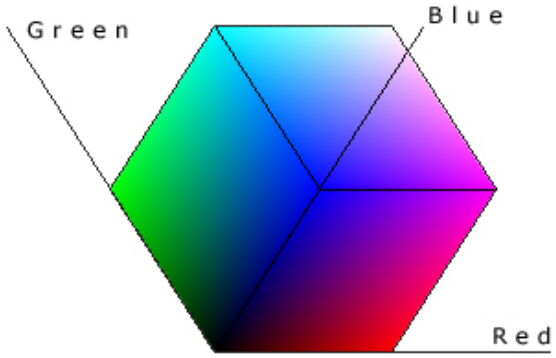


Figure 16 - The RGB colour model viewed with a bounding box and axis lines indicating corresponding colour channels

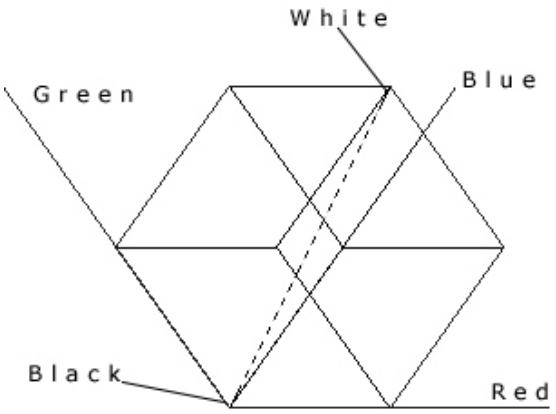


Figure 15 - RGB cube outline showing the grey scale running diagonally through the centre

Two techniques used to generate colours are the additive and subtractive processes. A monitor uses an additive process where light is emitted from the screen phosphorus where as a printer uses a subtractive process where people see the colours on paper through reflected light. A subtractive process can be modelled using the primary colours cyan, magenta and yellow: when yellow is added the light reflects the Red and green while blue is subtracted. So when all three primary colours are present the combined colour is black, whereas in the RGB model when red, green and blue is combined the resulting colour is white. The CMY colour model can be generated easily from the RGB model using transformation matrix in Equation 1 and Equation 2.

2.3.3 YIQ, YUV, and YCrCb Colour Models

The YIQ, YUV, and YCrCb colour models are all very similar: each has a Y component to represent luminance (brightness), and two channels for chromaticity (colour) information. The YIQ model is used in the Unites States as the commercial colour television broadcasting standard NTSC (National Television System Committee). However the lower bandwidth assigned to the chromaticity provides an impaired colour quality, consequently variations of the YIQ model have been developed. The YUV model is used in the PAL (Phase Alteration Line) standard and provides an improved quality and is used by most of

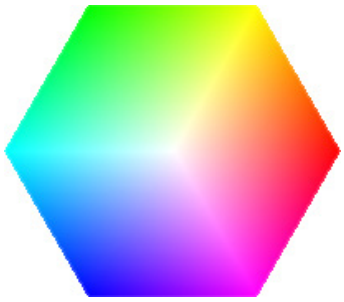


Figure 18 - RGB colour space looking along the diagonal from white to black

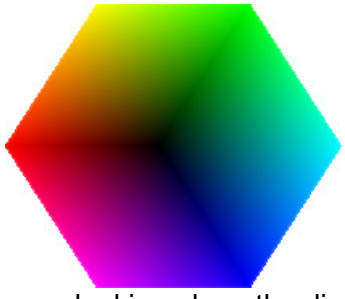


Figure 17 - RGB colour space looking along the diagonal from black to white

Europe, Africa, and Australia. Finally, the YCrCb model is another variation of the YIQ model used in file formats such as JPEG and digital video formats. I will discuss the YIQ model from here on, although the concepts apply directly all three of the above colour models.

The three colour channels are quite different from the RGB channels: Luminance (Y) and chromaticity (IQ). This model dedicates one entire channel for luminance (brightness) and uses the other two channels for chrominance (colours).

To move from the RGB colour space to YIQ, a matrix multiplication (in Equation 3) and the transformation will produce the new YIQ. Converting back from YIQ to RGB can also be done using the inverse transformation as shown in Equation 4.

Figure 19 shows the transformation from the RGB colour model to the YIQ model. This demonstrates how the colour space has been modified and can be use in a different manner. Notice the Y axis is now representing brightness information, and the white brightest area can be seen.

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Equation 1 - RGB to CMY transformation matrix

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} C \\ M \\ Y \end{bmatrix}$$

Equation 2 - CMY to RGB transformation matrix

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.144 \\ 0.701 & -0.587 & -0.114 \\ -0.299 & -0.587 & 0.886 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Equation 3 - RGB to YIQ transformation matrix

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.000 & 1.000 & 0.000 \\ 1.000 & -0.509 & -0.194 \\ 1.000 & 0.000 & 1.000 \end{bmatrix} \cdot \begin{bmatrix} Y \\ I \\ Q \end{bmatrix}$$

Equation 4 - YIQ to RGB transformation matrix

One of the advantages to this model is that it is well suited to compression - this is based on the fact that the human eye is more sensitive to particular wavelengths. For example, as our eyes are not very sensitive to blue so frequency of the blue component can be reduced somewhat before humans notice a quality loss. Also humans are much more sensitive to brightness. Much of an images definition and sharpness information is stored in the Y component as such leaving this channel relatively untouched makes the pictures quality loss unnoticed by humans.

When performing segmentation the YIQ colour space can be used to improve robustness. This is because when selecting the colour to be segmented from an image it can be accepted over a wide range of lighting conditions. When the threshold parameters are set the Y channel should accept a vide range for example 30 – 220 and the IQ channels should only accept a small range which varies depending on the colour being segmented from the image.

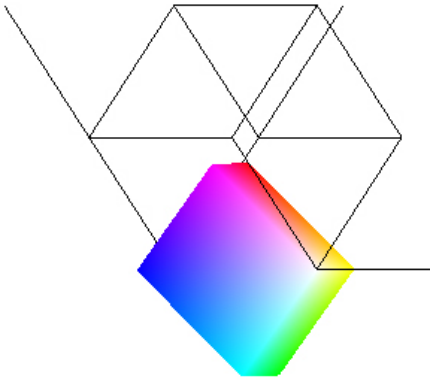


Figure 19- YIQ colour model with RGB cube outline to demonstrate the transformation

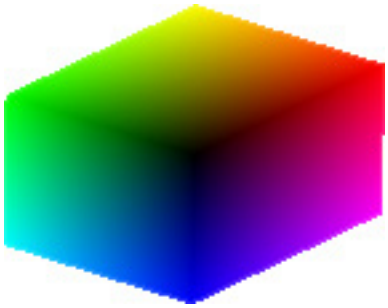


Figure 20 - YIQ colour model viewed along the Y axis

2.3.4 HSV Colour Model

The Hue Saturation Value (HSV) colour model is designed to be more intuitive for humans to use. It maps the Hue as an angle, the Saturation across the horizontal axis, and the Value along the vertical axis. Artists commonly use this colour model as they can easily modify the black and white component by changing the V (value) channel, change the tint by altering the S (saturation) value and the colour by the H (hue) represented by an angle. Visually this model is represented by an upside-down hexagonal pyramid as shown in Figure 21. There are a number of other similar colour models such as HLS (Hue Lightness and Saturation), and HSB (Hue Saturation and Brightness) which all use an angular Hue value instead of a rectangular chromaticity component. They are all very useful for human interpretation of colour but they are not suitable for digital signal processing. The hue value is an angle and requires many trigonometric calculations to convert colour models such as RGB.

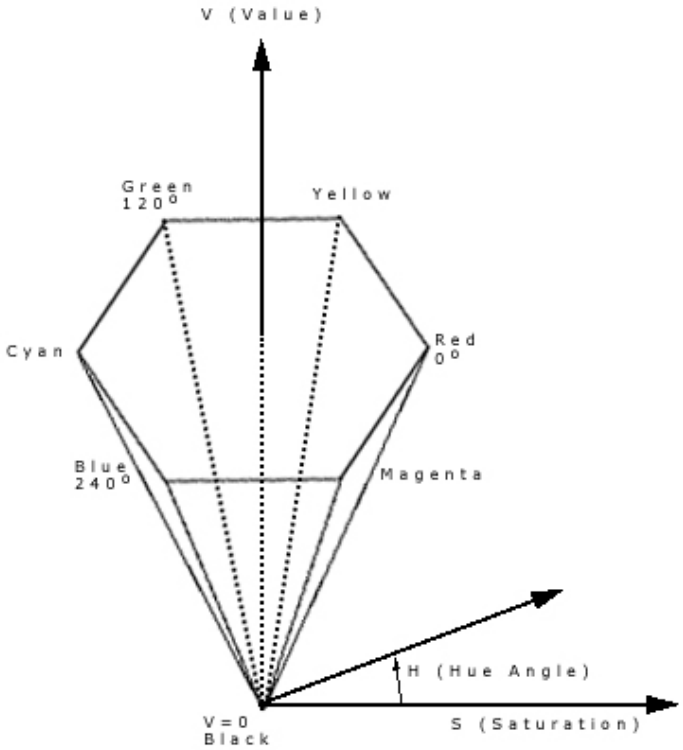


Figure 21 - HSV Colour model

3 Selection Criteria

There have been a number of important decisions I have made throughout the development of the system, and this chapter explains the reasoning behind each of the decisions. This chapter is based on both theoretical selection decisions as well as practical design decisions. Firstly I explain the reasoning behind the algorithm selected for the vision tracking to be implemented on a reconfigurable computer. Next I present the type of marker used to indicate the location of the user's thumbs. This involves exploring a number of different shapes, sizes and colours used to determine which provide the best results. Finally the techniques used for programming the FPGA and some of the unique considerations relating to programming parallel hardware are discussed.

3.1 Algorithm Selection

As mentioned earlier the Tinmith system uses ARToolkit for the tracking of the user's thumbs. ARToolkit uses a template matching technique to find pre-defined markers in the incoming video stream. The tracking data gives a full 6 degrees of freedom (DOF) tracking information for each of the defined markers. However Tinmith only requires 2 DOF for a fully functional user interface, as described in [35]. Working planes are used to model 3D objects at a distance using 2D coordinates similar to techniques used in CAD modelling [30]. In search of an appropriate algorithm I have considered any algorithm in which can provide 2 DOF or more, but generally less DOF simplifies the algorithm.

There are many different vision tracking algorithms but not all of them can be easily implemented on a FPGA. To make a selection I have followed some simple guidelines. Firstly the use of floating point numbers or division should be minimal or avoided all together. The use of floating point numbers or division on FPGAs is undesirable because large areas of the FPGA are required to implement the needed circuitry. Handel-C, as discussed earlier in section 2.1.2, does not directly support floating point numbers but this can be overcome by using lookup tables where all the answers are stored for a known problem set. Another approach is to use a prewritten module in another language such as VHDL or Verilog and import the functionality into Handel-C. At the beginning of this project, DK2 was the latest version of Celoxica's development suite. Since then, the release of DK3 and the latest version of the platform developer's kit (PDK) provide support for floating point numbers. However it still applies that the use of them requires large areas of the currently available space on the RC200's Xilinx [52] Vertex II 1000 FPGA. It is also

desirable to keep the algorithm simple so as to minimise the area used on the FPGA allowing room for other tasks like serial communications, video capture, and VGA output.

With these criteria I have found that the region based correlation techniques have many floating point calculations and the complexity is higher than other algorithms. Next I considered edge detection techniques, a simple Sobel edge detection is easily implemented on an FPGA, but this does not return information in the form of an X and Y coordinate of a marker. To do this, further processing is required to find an object's location which also introduces extra complexity. Finally, segmentation techniques were considered, which can easily be performed on an FPGA in runtime. The statistical filters mean, median and mode can then be used to find the centre of mass based on a colour. Each of the different algorithms provides similar tracking results and can be implemented on an FPGA.

3.1.1 Median

The median algorithm can be used to find an approximate centre of mass after segmentation. The statistical meaning of median is to calculate the middle value of several readings, where the readings are sorted in increasing order. In software it can be described as being implemented with an array of buckets for every row and column in the image as depicted in Figure 22. As pixels are found, the matching buckets are incremented. At the end of this process the buckets in the row and column arrays are individually traversed and added up until the total reaches half the total number of hits, when this occurs then the median pixel is found.

It was found previously by [48] that the median is more tolerant to noise pixels, and will not track the wrong object entirely when the area of the marker is greater than that of the combined noise pixels. Also when the markers centre coordinates are missing the accuracy of the median does not perform as well as the mode.

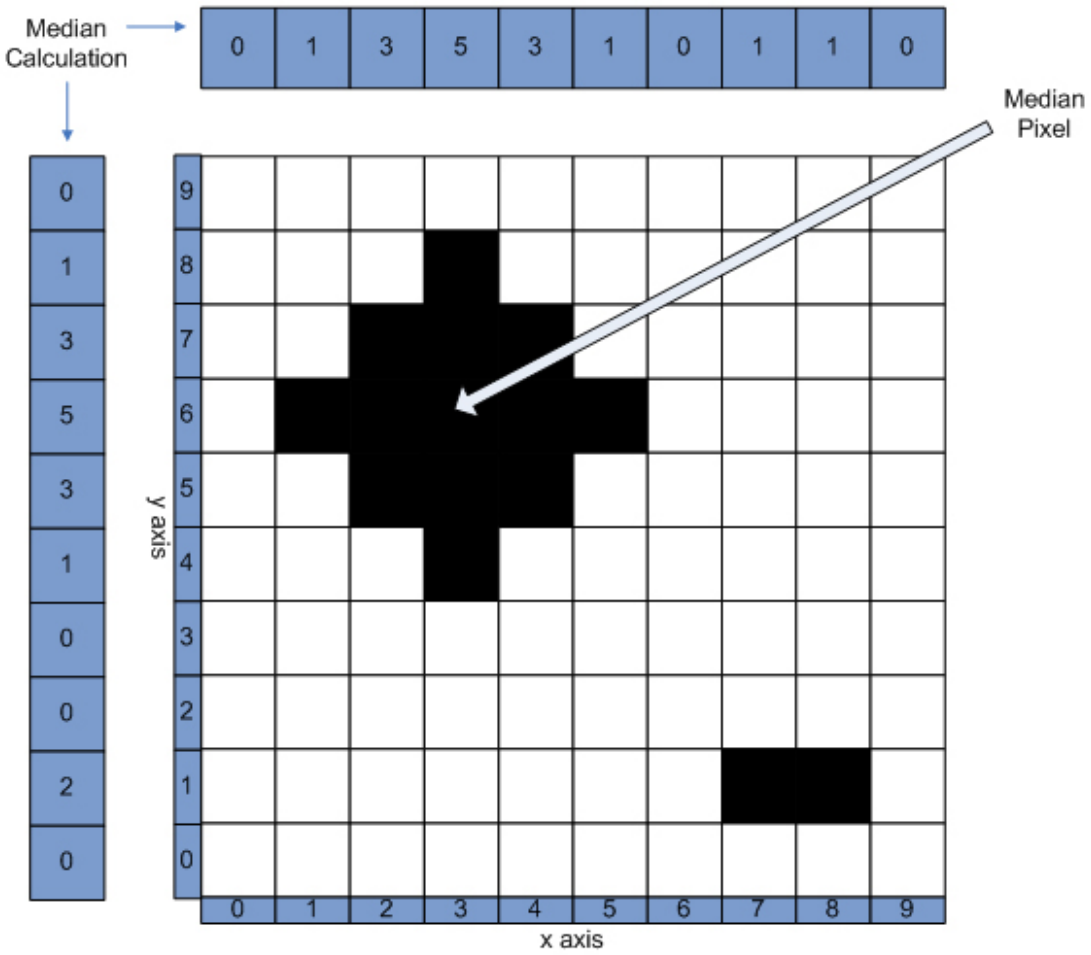


Figure 22 - Calculation of the median pixel in a 10 X 10 image

3.1.2 Mode

In statistics the mode is the most common value found in a group consisting of several readings. When calculating the mode it also required two arrays of buckets to be maintained, as shown in Figure 23. As each row and column is traversed, when a matching pixel is found the corresponding bucket is incremented. After all pixels have been processed the two arrays are traversed to find the most commonly occurring location for both X and Y.

The mode algorithm is very resistant to noise pixels unless they are concentrated along a single axis. It also finds an accurate centre of mass when there are obstructions to the marker. However the mode will fail with concentrated noise in the shape of a long narrow object which can have a smaller mass compared to the marker [48].

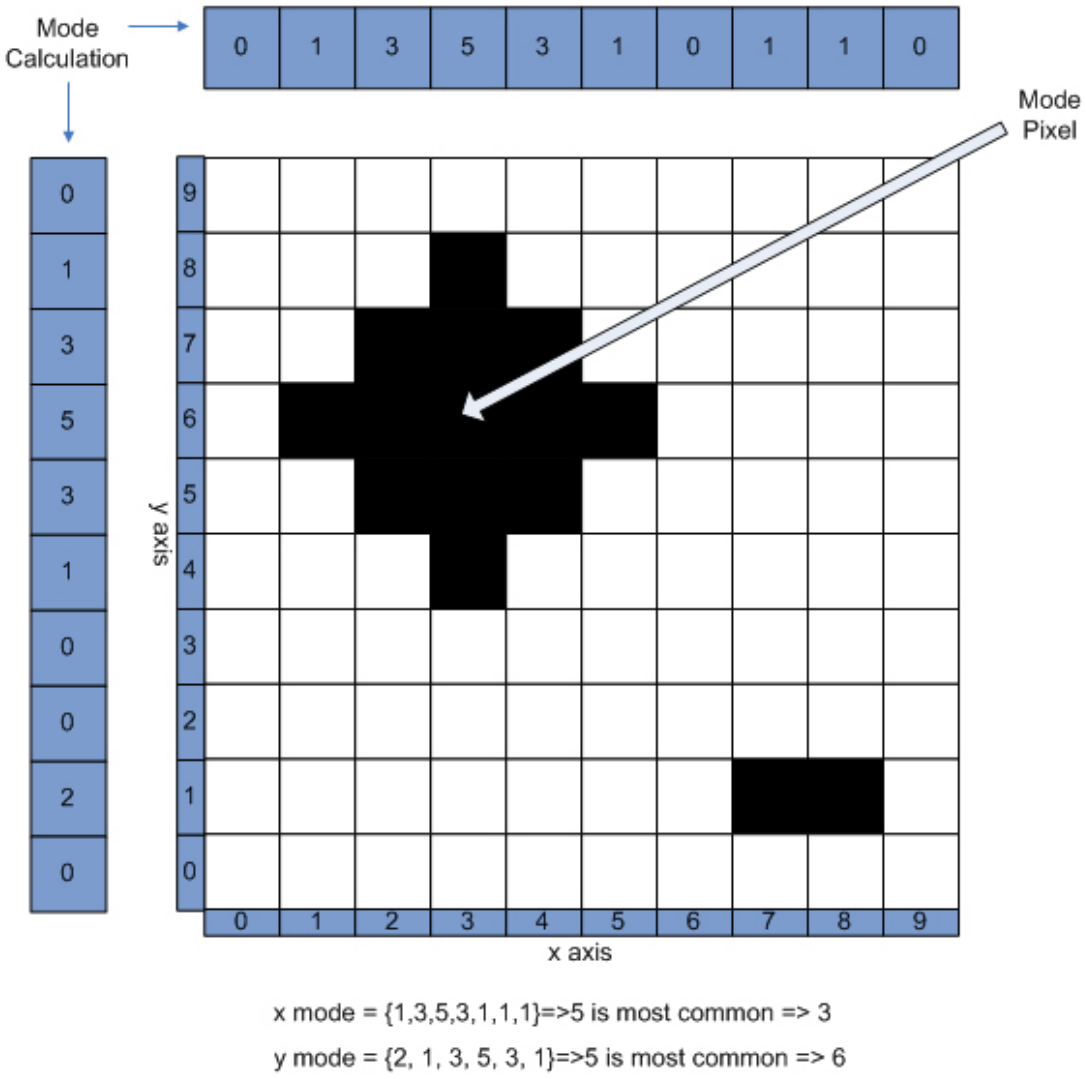


Figure 23 - Calculation of the mode pixel in a 10 X 10 image

3.1.3 Mean

The mean algorithm (also known as the arithmetic average) finds the average centre location of a group of values. The mean is calculated by adding up all the given numbers and dividing the sum by the total count, depicted in Figure 24. When calculating the mean of the segmented area the software adds up the X (xtotal) and Y (ytotal) locations of the accepted pixels and divides each by the total number of accepted pixels.

The mean will not track completely the wrong object until the area of the noise pixels becomes greater than the area of the marker pixels. It performs better compared to the median and mode when the centre of the marker pixels is occluded. Its weakness is any noise in the image cause the centre of mass to be pulled in the direction of the noise [48].

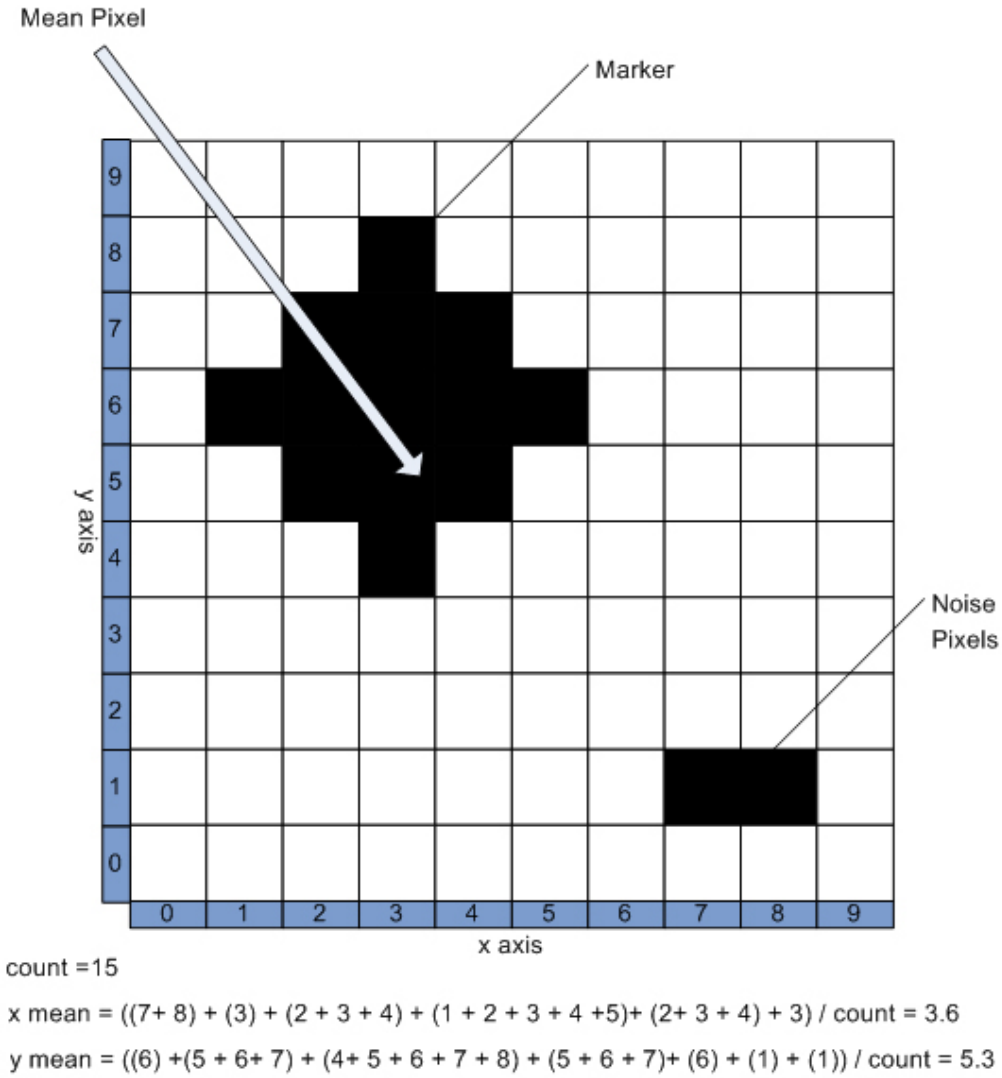


Figure 24 - Calculation of the mean pixel in a 10 X 10 image

3.1.4 Algorithm Summary

I studied the accuracy of each of the filters with a variety of software test cases on a PC and found they all performed reasonably well but with varying failure conditions. My final decision was based on which was best suited to implementation in hardware. The median and mode algorithms both require two arrays of buckets to be maintained in memory, and random memory access times on the RC200 are slow in comparison to the time it takes to traverse the pixel array linearly. The mean algorithm only requires three counters to be maintained with Xtotal, Ytotal and the number of hits; thus I decided mean is the best suited filter to implement.

3.2 Marker Shape, Size, and Surface

Based on the segmentation algorithm I found to be appropriate for FPGA implementation in the previous section, a unique marker is needed to identify the location of the user's thumbs. Tinmith previously used fiducial markers supported by ARToolkit, which uses region based correlation tracking techniques and as such are not suitable for segmentation based tracking.

3.2.1 Shape

As segmentation is not looking for a particular shape or pattern I am not restricted to any predefined shape. To select the best shape there are some properties which can be used to make the tracker more reliable. As the user moves their hands and head the angle at which the camera is viewing the marker varies. So selecting a shape that appears the same from all angles helps improve the accuracy of the centre of mass calculation. For example if we use a sphere it appears to be the same shape from all angles, shown in Figure 25, as long as no occlusion occurs. However if we use a disk its shape appears to change when we look at it from different angles, shown in Figure 26.

One of the conditions which can not be avoided is when the marker becomes occluded. One of the ways this can happen is when the user blocks the marker with their other hand; this of course can not be avoided. Another problem occurs when the user rotates their hand so the back of their thumb faces the camera. Both of these conditions can not be prevented by using a different shape marker and are generally avoided while using the Tinmith system.

Figure 25 - Markers viewed from different angles. (Left) Completely occluded by the users thumb. (Others) appear circular in shape

Figure 26 - Disc shaped marker viewed from different angles

3.2.2 Size

When selecting the size of the marker from the usability aspect it is desirable to use the smallest one possible. One reason for doing this is to avoid interfering with the user's dexterity. If the marker is too large or heavy this may affect their control while using the Tinmith system.

On the other hand it is important not to use a marker which is too small. When performing segmentation to find the markers it is common for there to be noise or unwanted pixels accepted. If the marker is too small the noise pixels will affect the accuracy of the centre of mass algorithm and provide inaccurate results.

I propose using a marker which is approximately 1cm in diameter so that it is not too small causing the tracker to fail and not too large where it would be a distraction to the user.

3.2.3 Surface

The surface used for the marker affects the quality of segmentation quite dramatically. When sunlight hits the marker, the surface reflects the light which causes bright spots on the surface (specular highlight) as shown in Figure 27. By reducing the bright spot we can reduce the threshold range. Shiny surfaces tent to be affected the most by sunlight and should be avoided. Matte surfaces tend not to have such a large specular highlight but it is still noticeable. Another option is to use a surface with a furry finish, when the light reflects off this kind of surface it is scattered in many different directions and the specular highlights no longer form a singular bright spot. Figure 28 illustrates a furry marker photographed in a range of different lighting conditions.

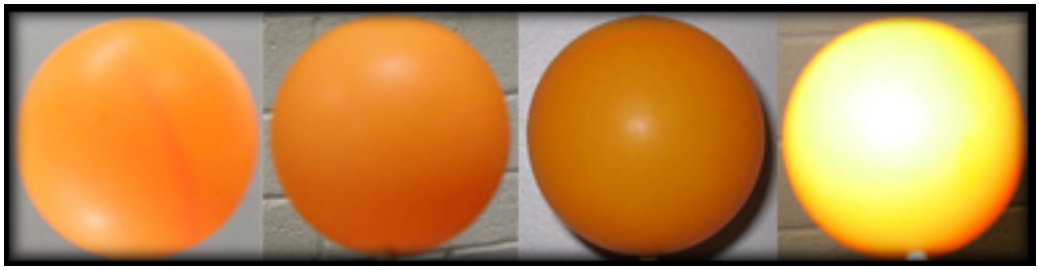


Figure 27 - Orange ping pong ball in a range of different lighting conditions

Figure 28 - Fury marker used to reduce the specular highlights

3.3 Colour Space Selection

One of the most difficult parts in developing a tracking system is designing it to work in a variety of different environments. Outdoors operation, as mentioned previously, requires the tracker to be resilient to varying lighting conditions. When capturing a video stream from a camera a common input format is RGB. When segmentation is performed using the red, green, and blue colour channels the brightness information is represented through all three channels. The earlier discussion of colour spaces explained how the greyscale colours are represented through the diagonal of the cube with black at one end and white at the other. So to accept the marker colour over a range of lighting conditions the range of all three (RGB) channel thresholds must be increased over the diagonal. Doing this also accepts more colours in the threshold and increases the noise and reduces the trackers accuracy.

An alternative is to use a colour model which dedicates one channel to brightness information. Colour models such as YCrCb, YIQ, and YUV all use the Y (luminance) channel to represent brightness. If we choose to use the YCrCb colour model we can select a particular colour using the Cr and Cb channels using a very small range while accepting a large range in the Y channel. This has the affect of selecting a colour over a range of different lighting conditions.

This thesis focuses on processing an image in an outdoor environment where there will be continually changing lighting conditions. I am also interested in performing segmentation on a particular colour regardless of the changing lighting condition. So from

the above research there are three models which should be considered YIQ, YUV and YCrCb. Each of these models allocate a separate channel the luminance and two channels to colour. So when performing segmentation a colour can be specified specifically and a range of luminance conditions can be accepted during the segmentation process.

4 Implementation

The design of FPGA circuits is somewhat different to that of conventional software design. The biggest design difference between the two is the parallel architecture of the proposed FPGA. Parallel processing allows us to develop designs which have more than one process executing simultaneously. The RC200 platform has a maximum clock frequency of 80MHz which is considerably slower compared to that of a modern desktop computer, however with careful design parallelism can be used to develop designs than run at equal or often improved speeds compared to software implementations. In this chapter I explain the fine grained details of implementing the previously explained tracking algorithm onto a reconfigurable computer.

4.1 RC200 Platform

The Celoxica RC200 reconfigurable computer was chosen as the target platform for the hand tracking system, this is shown in Figure 29. The hardware consists of a Xilinx Virtex II 1000 FPGA, 8 Megabytes of external memory, programmable clocks, TFT touch sensitive screen, Ethernet, audio, video out, VGA out, video in, parallel, and RS-232 serial ports. The RC200 is a single board with dimensions of 190mm x 150mm and can run from a 12V power source. The platform contains a Phillips video capture device and provides synchronous streaming of pixels to the FPGA. This particular platform was selected because it has a dense FPGA, supports streaming video, has a serial port for connection to the host, and has an extensive Handel-C application programming interface.

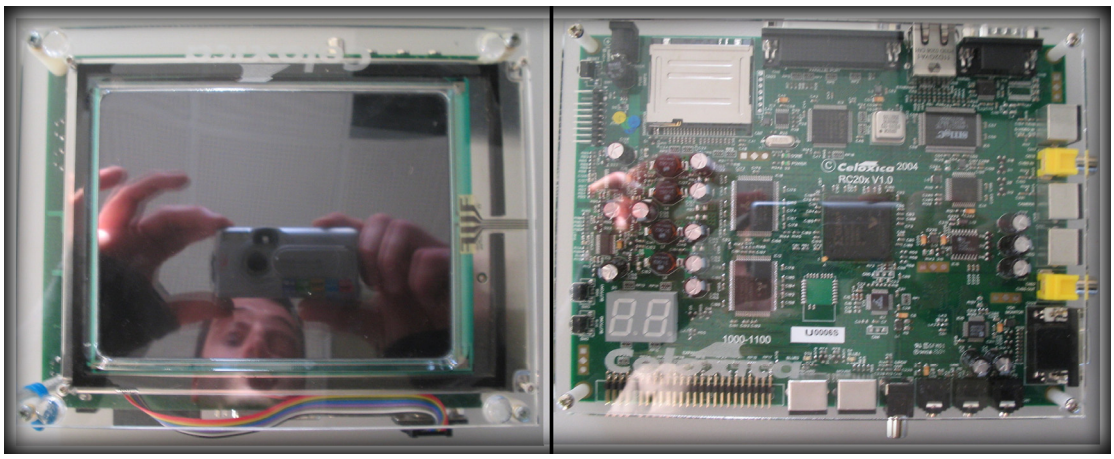


Figure 29 - RC200 Reconfigurable computer platform.
(Left) - Top view of TFT screen. (Right) - Bottom view of FPGA and other electrical components.

4.2 Language Selection

Of the available programming languages discussed earlier I decided to use Handel-C [6] for two main reasons, firstly Handel-C supports the features of the RC200 platform and provides an interface to all the additional features. Secondly Handel-C is a hardware description language based on an extended subset of the standard ANSI-C software programming language. The major advantage of it is there are no intermediate stages and it allows hardware to be directly targeted from software.

To produce the bitstream for the final program there are a number of steps involved. Firstly the code is developed using a subset of the ANSI-C specification, one of the most noticeable differences is its N length data types where the actual bit length is explicitly specified. The language also provides the ability to control parallel execution of processes through the use of the *par* operator. Figure 30 is a small snippet of Handel-C source code using both a variable width declaration and the par operator to initialise two variables simultaneously.

Another important feature of the language is the timing control the keyword *delay* can be used to wait for exactly one clock cycle. This can be used to synchronise processes where there are uneven execution times that need to be matched.

DK is then used to compile the source code, and a number of different output formats are available such as EDIF, Verilog, VHDL and simulation DLLs. I chose to generate EDIF which is compatible with Xilinx ISE [52]. I then used Xilinx's tools to generate the final bitstream which can be loaded onto the FPGA.

Overall Handel-C provides the necessary features that allow software engineers to develop hardware applications, although the development time on hardware still should be expected to take longer than an equivalent software program. After the initial learning of the new language, the increased development time is due mainly to the compilation time required to generate the bitstream used to configure the hardware platform, which can take more than one hour.

```
unsigned 4 i;
unsigned 4 j;

par{
  i=10;
  j=20;
}
```

Figure 30 - Handel-C source code snippet initialising two variables simultaneously

4.3 Software Architecture

This sub section describes the distinctive design architecture developed specifically for operating on a hardware platform. A summary of each of the intercommunicating parallel processes developed for the tracking system is also presented. Also it explains how Tinmith was successfully integrated with the hardware tracking solution.

4.3.1 Pipelined Design

The design of the circuit architecture requires a somewhat different approach compared to that of a software application. One of the biggest considerations is to exploit the parallel processing capability available on the FPGA. Using a high level language such as Handel-C, some parallelism is already implemented in the libraries. For example when you import the floating point libraries in Handel-C the programmer is provided with an interface to access optimised floating point operations. However further performance increases can be achieved by carefully designing the architecture and flow of an application to exploit parallelism.

A unique design idea that can use in parallel architectures is pipelining. This technique can be used to improve the performance of some algorithms. It breaks up the data path into several logical steps that are all dependant on the previous, as shown in Figure 31. This approach increases throughput but also increases latency. It is particularly well suited to FPGA design as each of the logical blocks can exploit the parallel architecture of FPGAs,

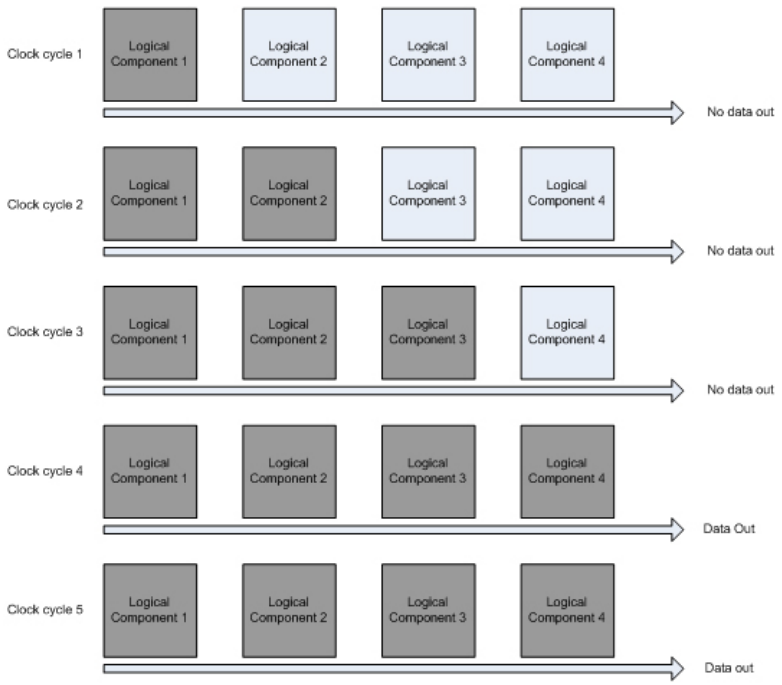


Figure 31 - Pipelined datapath over 5 clock cycles

and also pipelining maps particularly well to register rich architectures like those provided by Xilinx.

There are many different ways pipelining can be implemented, for the tracker design I assessed all the processes required, at a high level these can be summarised as follows:

- Capture a pixel from the video image
- Perform segmentation to separate the marker from the image
- If the pixel is the correct colour accumulate the X and Y location from within the frame. Also add one to a counter for each accepted pixel
- At the end of the frame divide the accumulated X and Y values by the counter
- Send the result to the RS232 serial port
- Render the video image to the video display

To maintain a system which runs in real time without a reduced frame rate it is required that each pixel is processed before the onboard Phillips SAA711H capture chip presents the next pixel for reading. To achieve this, the dataflow path can be broken up into 4 parallel processes: pixel capture and segmentation, division for mean calculation, RS232 serial communications and video display. These processes are well suited to a pipelined architecture and can be applied as shown in Figure 32.

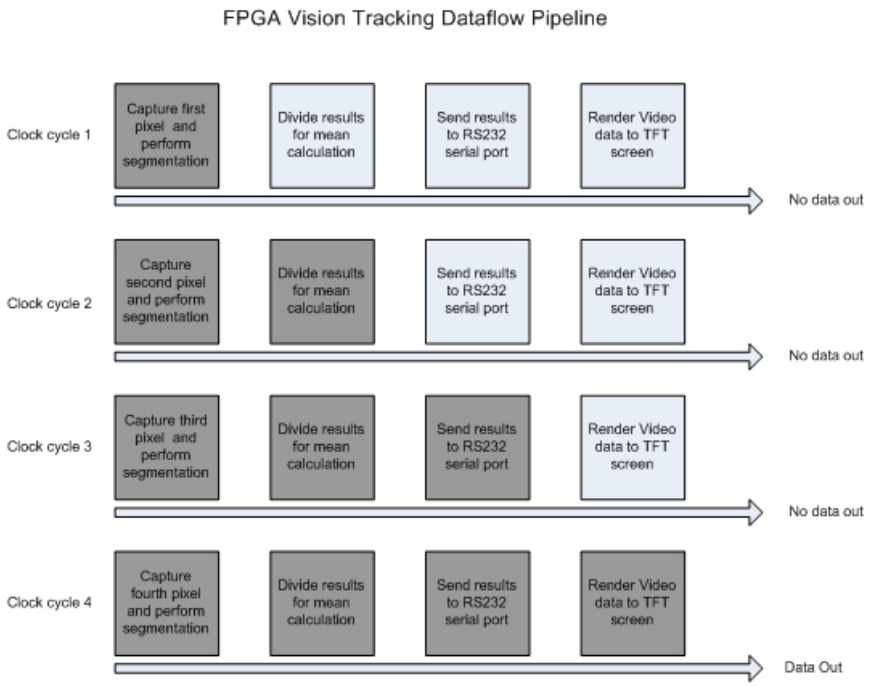


Figure 32 - Vision tracking data path pipeline. Note clock cycles used as pipeline reference time internal block processes may take more than one clock cycle

4.3.2 Circuit Design

The application written consists of four parallel processes: video input, VGA output, mean calculation (division), and RS-232 serial communications, each of which will be discussed in more detail in the following sub-sections. Although the four processes are run in parallel to each other, they are all closely interlinked using control flags for accessing common data values, as depicted in Figure 33. The flow of the system starts with the video stream; a pixel is read and evaluated according to the threshold values. For all the values that are accepted, a running total of the X and Y pixel locations is stored. When the end of the frame is reached the mean calculation process computes the centre of mass of the accepted pixels. The result from this is then sent via the RS-232 serial port and displayed to the TFT display for debugging using a cross to indicate the location of the marker being tracked. The TFT screen is typically used as a debugging device when tuning the tracker

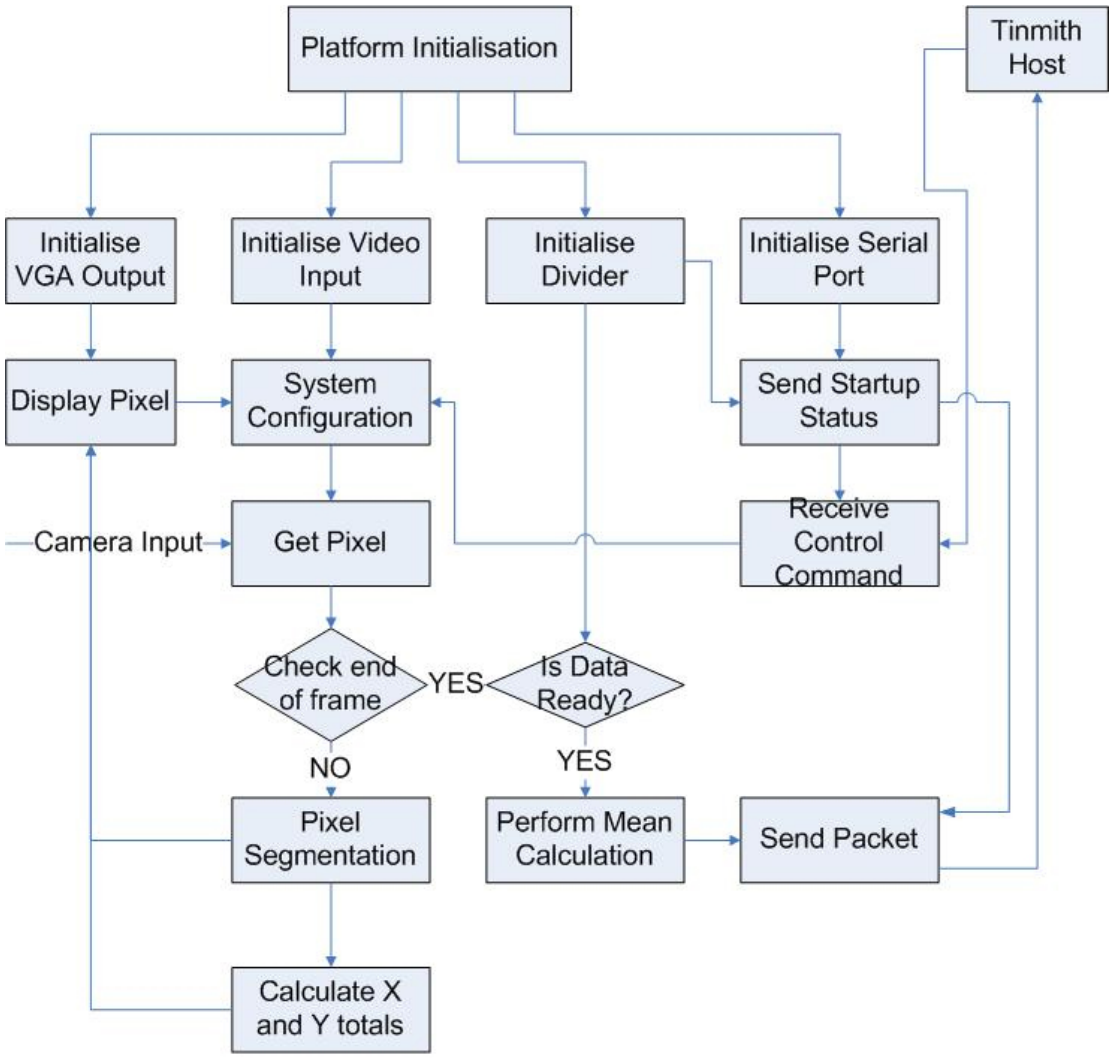


Figure 33 - Flowchart of the parallelised hand tracking algorithm implemented in hardware

outdoors. The pseudo code for the design has been presented in Figure 34, which outlines the flow of the system on a pixel by pixel basis.

Using the TFT display requires the main clock on the FPGA to be set to exactly 25.175MHz. During the development of the algorithm it was quite difficult to optimise each of the processes so that they completed their processing before the next pixel was presented. This involved using optimised Handle-C code, for example instead of using a *for* loop a *while* loop would be used. In most languages this would make minimal difference to the performance but when using Handel-C the operation of a while loop can be parallelised (in most cases) more than compared to a for loop [6].

Figure 35 depicts the final mapping of the circuit on the FPGA, which is a layout diagram generated with the Xilinx tools and is useful for quickly visualising the area used and can also be used to optimise the placement and routing of the components.

4.3.2.1 Video Input

The video input process captures a video stream from the Phillips SAA711H chip. This provides a synchronous stream of pixels which are evaluated in real time. The pixels can be captured in a range of formats, and I have chosen YCrCb as this separates brightness and colour information which makes it well suited to outdoor tracking. As each pixel is captured, the scan position is used to evaluate what path will be executed in the circuit. The first and most common path is executed when a pixel is read and at this point segmentation is performed to determine if it falls between the threshold values. When a pixel is within the threshold values, the X and Y locations are processed using the decided algorithm until the end of the frame. The second path is executed when the X and Y scan values indicate the

end of a frame (X = 720 and Y=576). Finally, a shared control flag is set to indicate the frame is complete and the overall calculation process can begin.

```
While (True)
  For each pixel in the image
    If pixel falls between segmentation ranges
      Add x coordinate to xtotal
      Add y coordinate to ytotal
      Increment hit counter by one
    End If
  End For

  Calculate xmean with xtotal divided by hits
  Calculate ymean with ytotal divided by hits
  Send results to PC via RS-232 port
End While
```

Figure 34 - Pseudo code for FPGA vision tracking algorithm

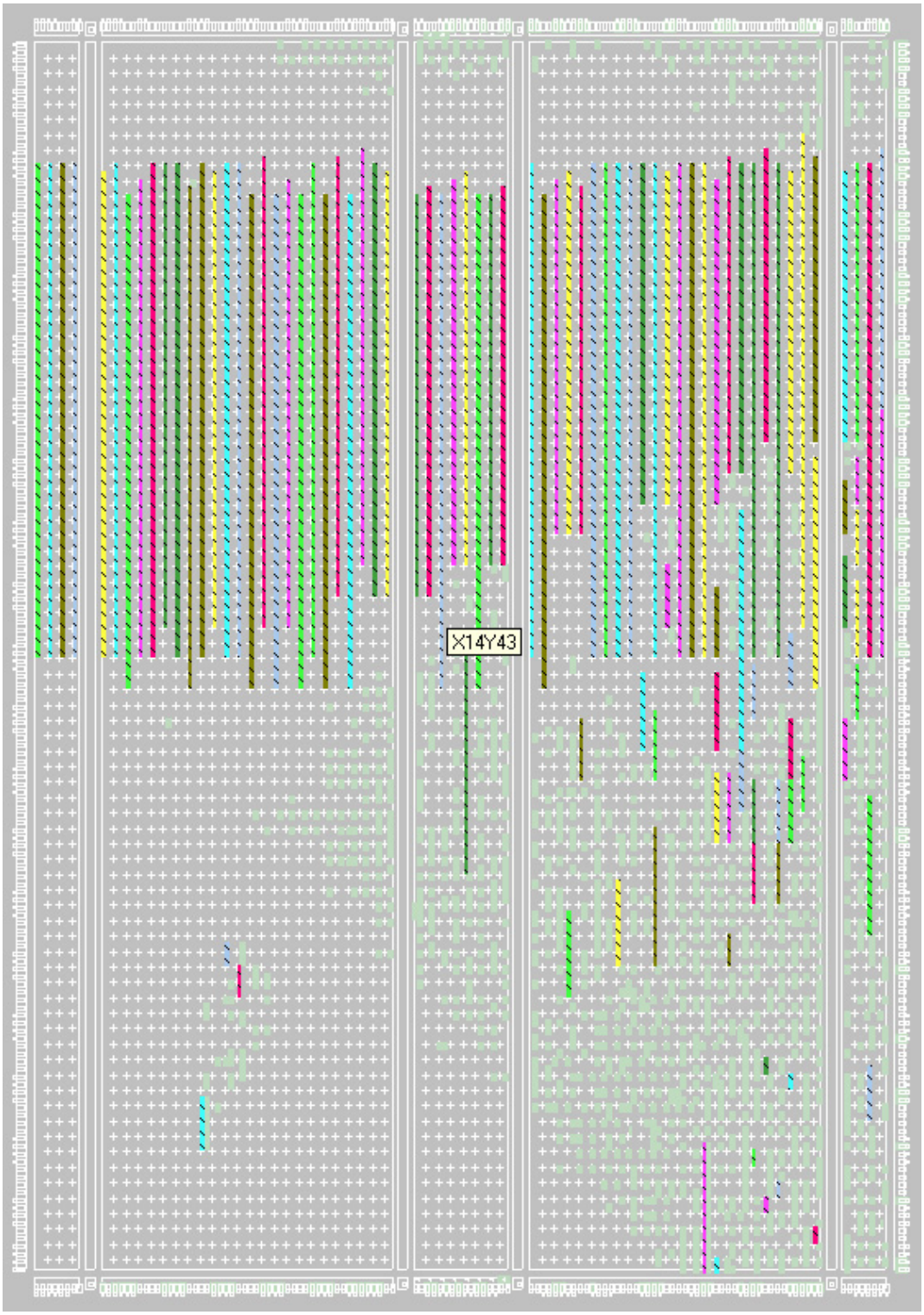


Figure 35 - FPGA circuit layout generated with Xilinx Floorplanner

4.3.2.2 Mean Calculation (Division)

The purpose of the mean calculation process is to perform division on the results calculated in the section above. The mean calculation process runs as an endless loop, and a control flag is used to signal when new values are ready and the division can be preformed. When this occurs the X and Y totals are divided by the frame hit counter to provide the final average X and Y results. When this step is completed another control flag is set indicating the results are ready for the video output process. Finally, the results are sent to the RS-232 circuit in a custom binary protocol to the Tinmith system.

4.3.2.3 Video Output

The video output process is used to display a picture on the TFT screen and the VGA out of the RC200. The results from the mean calculation are used to display the location of the blob being tracked, as shown in Figure 36. This video output is not essential for the operation of the algorithm but has proven to be a valuable tool when tuning the different coloured blobs. It also means the RC200 is a stand alone tracker not relying on the accompanying laptop computer to demonstrate its operation if needed.

Figure 36 - Output from the RC200 showing image threshold and the calculated marker centre point, combined with the camera view of the outdoor environment

4.3.2.4 RS-232 Serial Communications

The RS-232 serial port is used to send results to the host computer, as well as receive commands used to configure the RC200. When the RC200 is reset, a set of system initialisation packets are sent to the host indicating the status of the RC200. The RC200 then enters normal operation where it reads incoming commands used for system configuration and sends tracking information to the host. Incoming commands include setting the threshold values for tracking of different coloured blobs, setting the camera input port, and running in debugging mode when the image segmentation is displayed to the TFT screen. Outgoing tracking data is sent when the mean calculation process indicates a new result is ready. The system sends approximately 25 updates a second, which is the standard interlaced PAL refresh rate provided by the Phillips capture chip.

It was decided the configuration of the serial port would be set 8N1 (8 data bits, no parity and 1 stop bit) at a baud rate of 38400 bps. These were chosen to match some of the existing serial communications settings in software. The packet structure was designed to handle configurations of multiple markers allowing for future expansion and new features to be added to the system. The incoming control packet structure is described in Table 1 and the outgoing result packet structure in Table 2.

Field	Size (Bits)	Value	Description
Sentinel Head	8	0x55	Marks the beginning of the packet
Command	8	0-255	Range of different control commands used to configure the tracking system
Val1	16	0-65535	Used as the first variable value for the different commands
Val2	16	0-65535	Used as the second variable value for the different commands
Marker Number	8	0 – 255	Used to indicate the marker the settings should be applied to
Sentinel Tail	8	0xAA	Marks the end of the packet

Table 1 - Packet Structure for incoming commands used to configure the tracker

Field	Size (Bits)	Value	Description
Sentinel Head	8	0x55	Marks the beginning of the packet
Command	8	0-255	Used to indicate tracking result packet
XPos	16	0-65535	X coordinate of the marker being tracked
YPos	16	0-65535	Y coordinate of the marker being tracked
Hits	16	0-65535	Number of pixels accepted after segmentation
Sentinel Tail	8	0xAA	Marks the end of the packet

Table 2 - Results packet structure sent from tracker toTinmith

4.3.2.5 Tinmith Integration

The vision tracking system in the RC200 was integrated with the existing Tinmith system so that it could be combined with the existing research. Figure 37 is a screen capture showing how the various devices in the system work together to perform the vision tracking and video overlay task. The RC200 communicates with the PC by generating 10 byte data packets and transmitting these via an RS-232 serial cable. The laptop then reads these

Figure 37 - Overall system operation, showing the results of the RC200 integrated with the general purpose laptop, and video AR implemented using hardware overlay of the two video signals

packets in and draws the cursor position as part of the 3D overlay and is used to supply the user interface with 2D tracking information. A new interface was created relatively easily for Tinmith which completed the integration of the new tracking system.

With the image processing no longer being performed in the laptop, it is now possible to perform video overlay in a specialised video overlay device, shown in Figure 38. This device performs chroma keying on two video signals, a specified colour is set in stream one, anywhere this colour is used the second video stream is overlayed on top of this colour, the combination of the two streams is shown in Figure 39. The laptop still performs the remaining processing of sensor data for rendering, with the Tinmith-Metro application being implemented using the Tinmith-evo5 software architecture [36].

Figure 38 - MagicView chroma key box used for video overlay

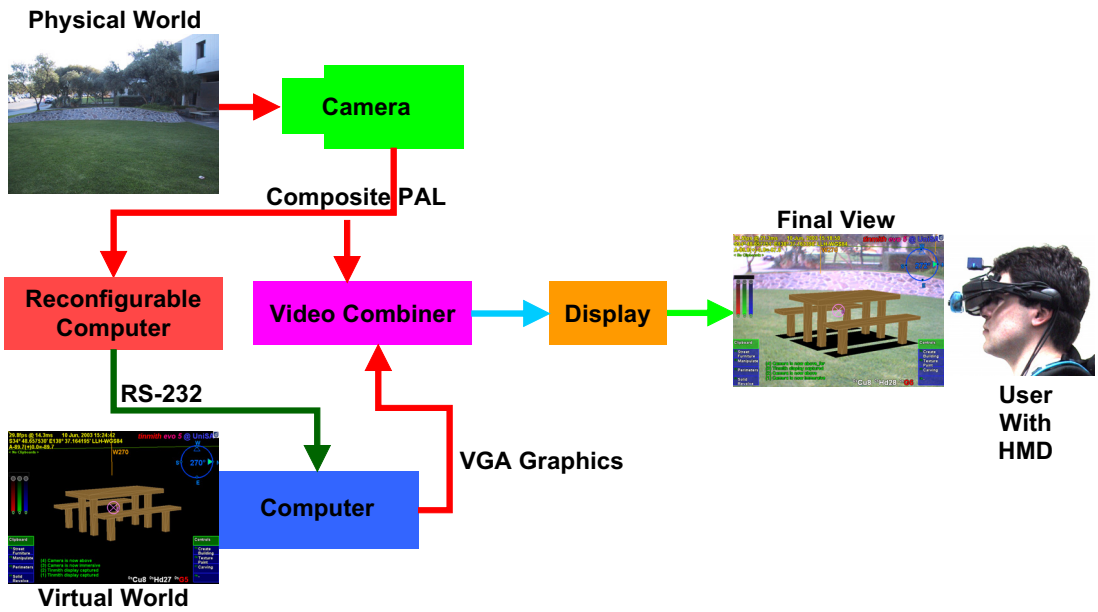


Figure 39 - Final design flow with video combiner unit

4.4 Parallel Code Examples

During the development of the software I developed a number of interesting programming methods to achieve a synchronised parallel design. This section outlines some of the techniques I was required to use during the development phase. It does not describe the operation of each of the processes but explains some of the unique syntax and timing issues which were exploited.

4.4.1 System Initialisation

Figure 40 is a small code snippet of the initialisation where the four main processes begin execution, and the ordering of each of the processes is important. The first process initialises the RS-232 port setting up the required parameters of 8 data bits per packet, no parity and 1 stop bit (8N1) at a baud rate of 38400 bps. Following this the division method is started (further explained in the next section) which begins by sending an initialisation packets indicating the tracker is starting up. The next two processes Videoin and VGA out require the enable calls to be performed in parallel, however the remaining calls that set the input source and begin capturing the video stream require a sequential execution. This is done by using the *seq* operator within the *par* statements allowing the correct execution order.

```
/*Start the four main processes (RS232,MeanCalculation,Video in and VGA out)in parallel*/
par{
    /*Initialise the RS232 serial port*/
    initRS232(currentData);

    /*Start the divider used for the mean calculation*/
    divider();

    /*Start the Video input in paralel with the Video in enable*/
    PalVideoInRun (VideoIn, ClockRate);

    /*Start frame buffer (VGAoutput) in parallel with the FB enable*/
    PalFrameBuffer16Run (&FBPtr, RAM, VideoOut, ClockRate);

    /*Force the enables before setting the input source and video stream*/
    seq{
        par{
            /*Matching enable statements*/
            PalVideoInEnable (VideoIn);
            PalFrameBuffer16Enable (FBPtr);
        }

        /*Continue initialising, set the video input source*/
        RC200VideoInSetInput(RC200VideoInInputComposite);

        /*Begin the Video input reading frames*/
        ReadFrames (VideoIn, VideoOut, FBPtr, currentData);
    }
}
```

Figure 40 - System initialisation

4.4.2 Mean Calculation (Division)

Another process that required careful consideration was the division process. As explained earlier I chose to use the pre compiled fixed point libraries supplied by Celoxica [6]. The definition of MyFixed type defines the accuracy in which the variables will operate. I set the definition to use 32 bit exponent and 0 bits for the mantissa as no decimal places are needed in the division calculation. The first step in this method required a delay of 500 clock cycles before the process could begin. This was required so the serial port could be completely initialised before it is used. Following this a series of initialisation packets are sent indicating the tracker has completed its initialisation phase and the results will be following these packets.

The rest of the mean calculation process operates from within a while(1) loop indication the process runs in parallel for the remaining execution time. At this stage there are three other parallel processes running so unlike a traditional software program, a while(1) loop can operate without all the programme's logic within it. The first check made inside the

```
void divider(){

    /*Create the floating point sections in logic*/
    MyFixed fixedNumber1, fixedNumber2, fixedNumber3;
    int 10 i;

    /*Wait for 500 clock cycles before beginning*/
    for(i=0;i<500;i++)
        delay;
    /*Send initialisation packets to RS232*/
    for(i=0;i<5;i++)
        sendAlive();

    /*Start divider loop*/
    while(1){

        /*When the segmentation process sets the ready flag*/
        if(newValue){

            /*Finish calculating the mean pixel*/
            fixedNumber1 = FixedLiteralFromInts(FIXED_ISUNSIGNED, 32, 0, finalX, 0);
            fixedNumber2 = FixedLiteralFromInts(FIXED_ISUNSIGNED, 32, 0, finalHits, 0);
            fixedNumber3 = FixedDivUnsigned(fixedNumber1, fixedNumber2);

            newXPos = (FixedToInt(fixedNumber3))[9:0];

            fixedNumber1 = FixedLiteralFromInts(FIXED_ISUNSIGNED, 32, 0, finalY, 0);
            fixedNumber2 = FixedLiteralFromInts(FIXED_ISUNSIGNED, 32, 0, finalHits, 0);
            fixedNumber3 = FixedDivUnsigned(fixedNumber1, fixedNumber2);

            newYPos = (FixedToInt(fixedNumber3))[9:0];

            /*wait for 1 clock cycle*/
            delay;

            /*Send the calculated results via RS232*/
            sendXY();

            /*Wait for 1 clock cycle*/
            delay;

            /*Set the flags notifying the VGA output a result is ready*/
            newLaptopCalcX = 1;
            newLaptopCalcY = 1;
        }
    }
}
```

Figure 41 - Snippet of the mean calculation process

loop is used to check if the parallel segmentation process has indicated a video frame has been completed. This allows two parallel processes to intercommunicate through the use of a shared data flag, and when this occurs the division of the X and Y totals with the hits is performed. This call to the FixedDivUnsigned method requires more than one clock cycle, the actual number varies depending on the input, and on completion it was required to include a delay of an additional clock cycle before the X and Y location of the marker is sent. Finally the two flags indicating a new result is ready for display are set which is read by the VGA output process.

4.4.3 Simulation and Compilation Issues

Celoxica's development suite DK2, as discussed earlier, can compile source code into a number of different formats. One of these can be used to create a Dynamic Link Library (DLL) for use with the simulation software also provided in DK2. The simulation software is designed to run the application being developed on the PC whilst simulating the execution of a parallel program. Whilst using the simulator the speed the simulation runs at is much slower when compared to running it on an FPGA., however when compiling the required DLL it is much faster compared to compiling the bitstream. Unfortunately when using the simulator it was necessary to make some small modifications before compilation. Also, instead of using a video stream you can select a static image for testing the video input. At the time of development there were some bugs relating to the RAM access and simulation could only be used for some parts of the tracking application.

Finally a note about compilation, when developing this application the biggest difference compared to developing a software application is the compilation time. During development the compilation times increased as the complexity increased with compilation times of up to 1 hour. Taking this into consideration much more time was spend carefully designing and checking code logic before compilation.

4.4.4 Parallel Programming Summary

Designing and programming on a parallel architecture device is currently more difficult when compared to doing the same task on a general purpose computer in a high level language such as C++ or java. However the improvements that can be achieved through using a good design with an appropriate language to express the design are quite impressive. This tracking system was developed with the clock rate fixed at 25.175MHz with room still available on the FPGA for other tasks. The software prototypes written in

C++ ran on a 400MHz machine and suffered from a reduced frame rate in the processed video stream. Although the programming was somewhat difficult during the development of this application some of the problems experienced are due to the immature age of the development environments. Handel-C is only 8 years old and currently has a limited audience interested in programming FPGAs, I feel some of the difficulty experienced during the development of this software was due to bugs in the compiler and associated synthesis techniques used to create the bitstream. Others were associated with the strict timing synchronisation required by parallel designed architectures.

5 Results

After the implementation of the algorithm on the FPGA, I performed a number of tests to evaluate its performance and robustness. Firstly a performance evaluation outlines the device usage and maximum running frequency. This is followed by tests to evaluate the operation of the tracker outdoors, in particular the range of lighting conditions in which the tracker can operate.

5.1 Hardware Performance

Once the hand tracking algorithm was implemented on the FPGA, both the clock speed and device utilisation were recorded. These values were captured from the output files generated from the Xilinx place and route tools. Including the algorithm and a 32 bit divider circuit, the total device utilisation was 83%, or 4263 out of 5120 slices of the FPGA. The theoretical maximum clock speed was 105.27MHz, however the actual clock speed of the system was set to 25.175MHz due to the requirements of the TFT display device.

To determine the actual device utilisation of the hand tracking algorithm itself without the divider circuit (something that could be easily performed on the host in a general purpose CPU), the application was recompiled with it removed. The percentage of FPGA consumed dropped to 23% or 1177 slices. Therefore, the 32-bit divider circuit consumed 3086 slices or 60% of the FPGA. Using 23% of the available room it is possible to configure the FPGA to run four trackers simultaneously. Also with the newest version of the RC200, the RC203 (which has a FPGA with 3 times the density), it is possible to have at least twelve of the trackers running.

5.2 Marker Shape Size and Surface

Selection process for the marker used to locate the user's thumbs was described in Section 3.2. After the tracker was developed I performed a number of tests to find which markers performed with the best accuracy.

5.2.1 Shape

The shape chosen for the markers is a sphere. During testing I experimented with other shapes such as a thin coloured decoration disc. As expected, as the user rotates their hand the area of the disc becomes too small for the tracker to locate. A number of other shapes were used during these experiments but it was found a spherical shape outperformed them all while the user rotates their hands.

5.2.2 Size

As discussed earlier the goal regarding the marker's size has been to use the smallest possible size marker whilst maintaining good tracking results. To find the best size I experimented with spherical markers from .25cm to 3cm in diameter. After using each of the different sized markers it was found that 0.5cm to 3cm markers would perform accurately at an arms length from the camera. Considering this 0.5cm diameter markers have been used in the final system, since they are no larger than the users thumbs.

In terms of accuracy, all of the marker sizes tested produced cursors which were within the bounds of the marker, assuming that the signal to noise ratio is relatively high. When smaller markers are used, the signal to noise ratio is reduced, and the accuracy will suffer. Appropriate marker sizes must be chosen to ensure that in noisy environments the tracker will perform properly, while still being small enough to point to objects accurately. Of the 1 cm, 2 cm, and 3.5 cm markers tested, I decided to use the 2 cm markers since they are the size of the finger tips.

5.2.3 Surface

The first marker I begin with was an orange ping pong ball. The problem with this ball is that the surface is very smooth and is affected by specular reflections. The reflections on the top of the ball would make it appear to be white rather than orange, shown in Figure 42. To overcome this problem I experimented with other surfaces and found the furry surface of a pompom reduced the reflections significantly. By using pompoms the segmented image would show a well defined circular result and thus improve the operation of the centre of mass algorithm.

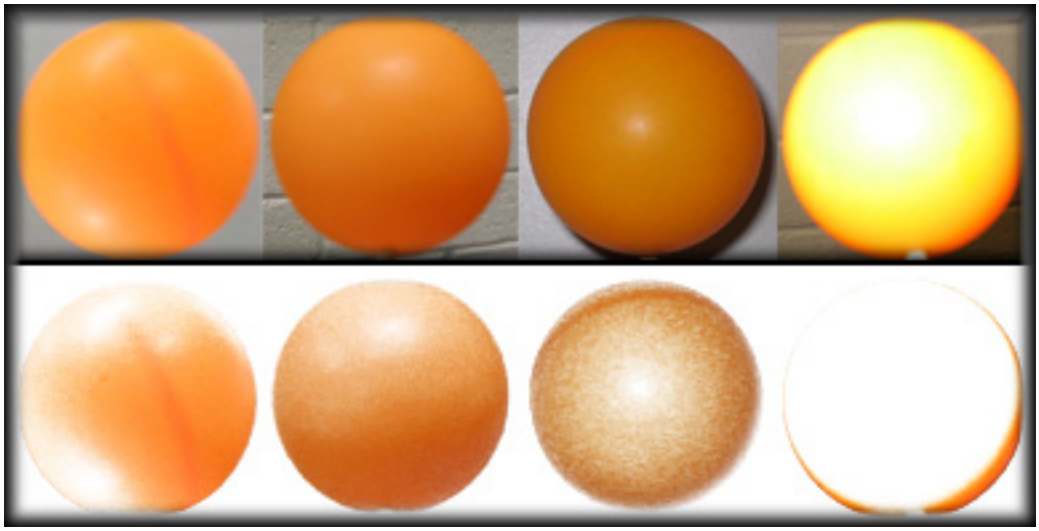


Figure 42 - White spot on shiny marker surface causes poor segmentation as shown in bottom row

5.3 Marker Tuning

To determine which markers operate the best a number of different experiments were performed. I begun by selecting a range of different coloured pompoms (orange, red, green, blue and yellow) then for each of these tuned the threshold values, using the interface shown in Figure 43, and tested their operation outdoors. To tune colours, the YCrCb filter is opened up to full range (0-255 with 8 bits per channel) and the segmented image is viewed on the RC200 TFT screen. The minimum (min) value for the Cr channel is then adjusted until just before it begins to filter out the blob. The maximum (max) value is adjusted in the opposite direction until the ball is just accepted. The min and max values are recorded and then opened back up to full range. The same process for min and max is then performed for the Cb channel so that the ball is segmented. The Y channel is slightly restricted to 10-240 so that it removes out black and white pixels (which contain no real colour information) and then the Cr and Cb ranges are both set to the measured values. The colours are then tested against the environment to see if there are matches against any other objects to interfere.

During testing it was noticed that the CCD camera sees colours in the environment different than our own eyes because the sensor operates over different light wavelengths. When looking at the output, some colours such as blue appeared with a noticeable green colour. When selecting coloured balls to use, the colour the camera sees (and not what the human sees) must be taken into consideration. The following sub sections summarise the colours that were tested outdoors.

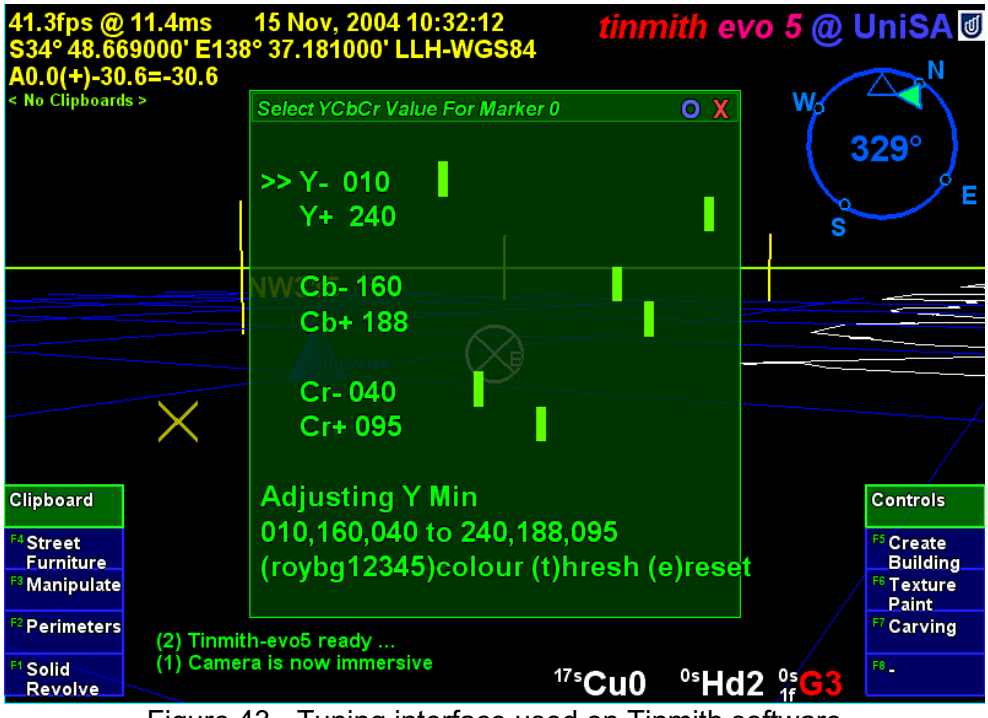


Figure 43 - Tuning interface used on Tinmith software

5.3.1 Green Markers

When testing the green balls, I noticed that they conflicted with many of the leaves in the trees nearby, but not the grass. Under close inspection the blades of grass actually contain quite a lot of yellow and so there was no conflict. Unfortunately, the amount of noise from the trees was enough to affect the tracker.

5.3.2 Blue Markers

The blue coloured balls occasionally conflicted with the sky under certain brightness conditions, such as when the camera's auto adjustment darkened the overall image. The blue ball appeared to be a turquoise colour in the camera, but when a more pure blue colour was used it still conflicted with the sky colour.

5.3.3 Orange markers

The orange colour generated the best results with my tracking system. After calibrating it was not common for there to be objects in the environment of a similar colour that would cause the tracker to operate incorrectly. The only time where a conflict occurred was when looking at a pedestrian crossing sign, which is understandable considering it was a similar shade of orange.

5.3.4 Yellow Markers

The yellow marker balls experienced slight conflicts with the grass, which as mentioned previously contained large amounts of yellow. This colour would be suitable for use in environments without grass however, such as on concrete or dirt perhaps. Another problem with yellow is that it is similar to orange and it is not possible to separate these two colours from each other with their YCrCb ranges.

5.3.5 Red Markers

The second best results were with the red coloured balls. There were no conflicts for this colour with the rest of the environment, except for a stop sign on campus. The results were not quite as good as orange, but this could perhaps be improved with further tuning of the YCrCb ranges. Once again, this colour is similar enough to orange that it too prevents them from being used together.

From the experiments, it was discovered that trying to provide highly saturated colours for the camera was more difficult than first thought. Even when using highly saturated coloured balls the camera tends to capture them with a slightly washed out colour which

somewhat reduces their distinctiveness. This may be reduced as higher quality cameras are used but may be difficult to remove altogether.

5.4 Hand Tracker Results

The purpose of this research was to develop a separate hand tracker using an FPGA to integrate with the Tinmith-Metro software. The RC200 contains an RS-232 serial port in which is used to transmit 10 byte packets to the PC for each update to indicate the X and Y coordinates of the cursor, as well as the number of pixels used in the calculation as a confidence factor. The RC200 captures frames at the PAL refresh rate of 50 Hz, but only provides frames to the FPGA at 25 Hz due to interlacing in the video signals. The RC200 processes frames in real-time and so the results are available within 1/25th of a second, although there is additional delay added by transmission across the RS-232 cable and in the host laptop operating system. Figure 44 shows a capture of the segmented overlay and extracted centre point from the RC200, combined with the view from the camera.

When the 2D cursor is plotted on the display in the Tinmith-Metro software, there is some slight lag with the cursor. This lag is noticeable because the RC200 and the video overlay hardware operate at PAL refresh rates, while the laptop is slightly behind with its processing and rendering of the 3D AR overlay. Previously this effect was not noticeable because the entire output was delayed, but now some parts of the display are faster than others. Possibly a delay could be added to the video stream so that the lag is synchronised, but then this would make the entire system lag from the physical world.

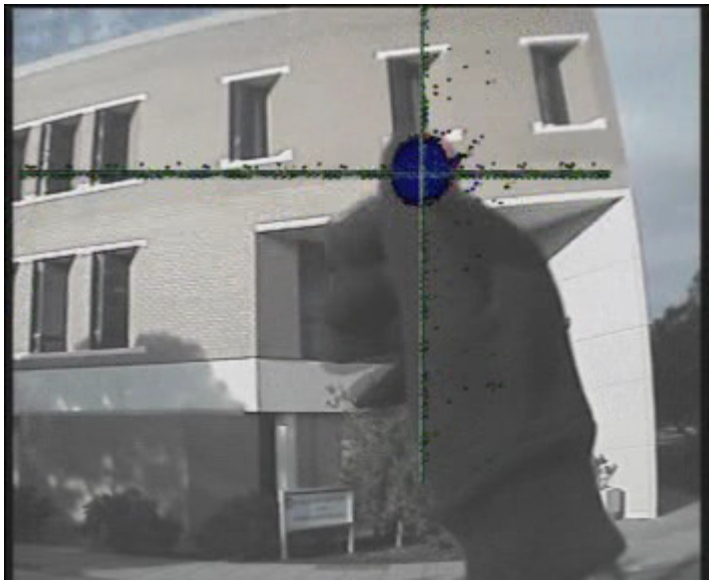


Figure 44 - RC200 performing tracking – overlaid cursor and segmented area are combined with the view from the camera

In comparison to the previous ARToolKit based tracker, the new tracker appears to be more robust in outdoor environments under most conditions. It is able to more easily survive extreme lighting conditions such as when the sun is almost in the field of view of the camera, which is very common. The tracker's main weakness is operating under twilight conditions when the camera is unable to distinguish colours in the environment as easily, while ARToolKit uses only black and white fiducials. However, the ARToolKit tracker fails in scenes that are half bright and half dark, and with specular highlights on the flat markers.

6 Conclusion

The research completed for this thesis has facilitated the development of a new tracking system developed on a FPGA. This has been used to replace ARToolkit used previously by the Tinmith AR backpack computer. A significant performance increase was achieved, particularly when used outdoors.

In this thesis I initially presented an overview of the research domains of field programmable gate arrays, augmented reality, vision tracking and colour models. The next section then explained the selection criteria I used to find a practical approach to implementing a new tracking system on an FPGA.

I have described how the previous software hand tracking system used by Tinmith was replaced with a reconfigurable hardware solution. The purpose of this was to relieve the current laptop microprocessor of the task so it would reduce the number of clock cycles used and result in lower power consumption by allowing it to be replaced with a less powerful and smaller unit.

I have demonstrated how a simple hand tracking algorithm for this application could be implemented using segmentation combined with a statistical filter such as mean, mode, or median. The tracker I developed has been integrated into the Tinmith system and is used to perform 3D modelling in outdoor environments. The tracker is able to operate well in a wide range of environments and with varying lighting conditions, making it ideal for outdoor use.

During the research and development stages I have published a number of fully refereed international conference papers contributing to the augmenter reality and user interface research areas[33, 34, 44].

The contributions I have made can be summarised as the following:

- Chosen suitable vision algorithm
- Used YCrCb colour space
- Implemented a working version of the tracker on the RC200 FPGA Platform.
- Integrated with the Tinmith system
- Improved tracking performance and robustness compared to ARToolkit

- Developed a low powered stand alone tracking solution
- Performed testing outdoors to find superior colours and measure accuracy of operation

Future work to follow on from this research has been considered there are a number of extensions that would be useful. Firstly the ability to track more than one object simultaneously would allow for interactions to be performed with both hands while using the Tinmith system. This could be done by using a different coloured marker on each of the user's thumbs. There are two considerations which also need further research: firstly the selection of an appropriate colour which would not interfere with the first marker, and secondly the time taken for the additional processing should be performed without slowing the current frame rate. Finally, the tracking system developed from this project was designed for the Tinmith backpack but is not limited to only being used here the tracker has no dependencies with Tinmith and could be used for any application where it is appropriate the track an object optically by attaching a coloured marker. Since development, there has been some interest in using the tracker in conjunction with another researcher who proposed using it to locate bushfires from an unmanned aerial vehicle.

7 References

1. Azuma, R. *A Survey of Augmented Reality*. in *Presence: Teleoperators and Virtual Environments,* pp 355-385. 1997.

2. Bandlow, T., M. Klupsch, R. Hanek, and T. Schmitt. *Fast Image Segmentation, Object Recognition and Localization in a RoboCup Scenario*. in *Robocup-99: Proceedings of the Third RoboCup Workshop*. July 1999.

3. Bednara, M., M. Daldrup, J. Teich, J. von zur Gathem, and J. Shokrollahi. *Tradeoff analysis of FPGA based elliptical curve cryptography*. in *IEEE International Symposium on Circuits and Systems*. May, 2002.

4. Bhadker, J. *A SystemC primer*. 2002: Star Galaxy Publishing.

5. Brusey, J. and L. Padgham. *Techniques for Obtaining Robust, Real-Time, Colour-Based Vision for Robotics*. in *RoboCup-99: Robot Soccer World Cup III,* pp 243-253. 2000. Springer-Verlag.

6. Celoxica, *Celoxica* http://www.celoxica.com. 2004.

7. Cheung, R.C.C., K.P. Pun, S.C.L. Yuen, K.H. Tsoi, and P.H.W. Leong. *An FPGA-based Re-configurable 24-bit 96kHz Sigma-Delta Audio DAC*. in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM),* pp 110-117. 2003. Tokyo.

8. Chung, T.P., Z.M. Yusoff, and A. Zuri Sha'ameri. *Implementation of a Pipelined Data Encryption Standard (DES) Using Altera CPLD*. in *TENCON 2000*. 2000. Kuala Lumpur, Malaysia.

9. Cipola, R., Y. Okamoto, and Y. Kuno. *Robust Structure from Motion using Motion Paralax*. in *4th Int'l Conference on Computer Vision,* pp 374-382. May 1993. Berlin, Germany.

10. Compton, K. and S. Hauck, *Reconfigurable Computing: A Survey of Systems and Software.* ACM Computing Surveys, June 2002: p. 171-210.

11. Dawood, A.S., S.J. Visser, and J.A. Williams. *Reconfigurable FPGAS for real time image processing in space*. in *Digital Signal Processing, 2002. DSP 2002. 2002 14th International Conference on,* pp 845-848 vol.2. 2002.

12. Dimond, K. and K. Pang. *Mapping VHDL descriptions of digital systems to FPGAs*. in *Software Support and CAD Techniques for FPGAs, IEE Colloquium on,* pp 9/1-9/3. 1994.

13. Dorfmuller-Ulhaas, K. and D. Schmalstieg. *Finger Tracking for Interaction in Augmented Environments*. in *2nd Int'l Symposium on Augmented Reality*. Oct 2001. New York.

14. Draper, B., R. Beveridge, W. Bohm, C. Ross, and M. Chawathe. *Accelerated Image Processing on FPGAs*. in *International Conference on Pattern Recognition*. April 2002. Quebec City.

15. Ebeling, C., D. Cronquist, and P. Franklin. *Lecture notes in Computer Science*. in *Proceedings of the 6th International Workshop on Field Programmable Logic, Smart Applications, New Paradigms and Compilers,* pp 126-135. 1996. London, UK: Springer-Verlang.

16. Foley, J., A. Van Dam, S. Feiner, J. Hughes, and R. Phillips, *Computer Graphics, Principles and Practice*. 1990: Addison-Wesley Publishing Company.

17. Foxlin, E., M. Harrington, and G. Pfeifer. *Constellation: A Wide-Range Wireless Motion Tracking System for Augmented Reality and Virtual Set Applications*. in *Int'l Conference on Computer Graphics and Interactive Techniques,* pp 371-378. Jul 1998. Orlando, Fl.

18. Gannot, G. and M. Lingthart. *Verilog HDL based FPGA Design*. in *Verilog HDL Conferance,* pp 86 - 92. 1994. Santa Clara, CA USA.

19. Hauser, J.R. and J. Wawrzynek. *A MIPS processor with a reconfigurable coprocessor*. in *IEEE Symposium on Field-Programmable Custom Computing Machines,* pp 12 - 21. 1997.

20. Hopf, J., G.S. Itzstein, and D. Kearney. *Hardware Join Java: a high level language for reconfigurable hardware development*. in *Field-Programmable Technology, 2002. (FPT). Proceedings. 2002 IEEE International Conference on,* pp 344-347. 2002.

21. Hopf, J. and D. Kearney. *Specification and integration of software and reconfigurable hardware using Hardware Join Java*. in *Field-Programmable Technology (FPT), 2003. Proceedings. 2003 IEEE International Conference on,* pp 379-382. 2003.

22. Hossack, C.J. and C.G. Guy. *The route from VHDL to FPGA using synthesis*. in *Software Support and CAD Techniques for FPGAs, IEE Colloquium on,* pp 8/1-8/4. 1994.

23. Intersense, *Intersense Intertia Cube 2*. 2004.

24. Jebara, T., T. Eyster, J. Weaver, T. Starner, and A. Pentland. *Stochasticks: augmenting the billiards experience with probablistic vision and wearable computers*. in *1st Int'l Symposium on Wearable Computers,* pp 138-145. Oct 1997. Cambridge, Ma.

25. Kato, H. and M. Billinghurst. *Marker Tracking and HMD Calibration for a Video-based Augmented Reality Conferencing System*. in *2nd Int'l Workshop on Augmented Reality,* pp 85-94. Oct 1999. San Francisco, Ca.

26. Luk, W., *A Reconfigurable Engine for real-time Video Processing.* Field Programmable Logic and Applications, 1998. **LNCS 1482**: p. 169-178.

27. Luk, W., T. Lee, J. Rice, and P. Cheung, *Reconfigurable Computing for Augmented Reality.* 7th IEEE Symposium on Field-Programmable Custom Computing Machines, 1999: p. 136-145.

28. Maslennikov, O., J. Shevtshenko, and A. Sergyienko. *Configurable microcontroller array*. in *Parallel Computing in Electrical Engineering, 2002. PARELEC '02. Proceedings. International Conference on,* pp 47-49. 2002.

29. Matsushita, N., D. Hihara, T. Ushiro, S. Yoshimura, J. Rekimoto, and Y. Yamamoto, *ID CAM: A Smart Camera for Scene Capturing and ID Recognition.* 2nd Int'l Symposium on Mixed and Augmented Reality, 2003: p. 227-236.

30. Mine, M.R., *Exploiting Perception in Virtual-Environment Interaction*, in *Department of Computer Science*. 1997, University of North Carolina: Chapel Hill, NC.

31. Piekarski, W. and B.H. Thomas. *Interactive Augmented Reality Techniques for Construction at a Distance of 3D Geometry*. in *Immersive Projection Technology / Eurographics Virtual Environments*. 2003. Zurich, Switzerland.

32. Piekarski, W., *Interactive 3D Modelling in Outdoor Augmented Reality Worlds*, in *Computer and Information Science*. February 2004, University of South Australia: South Australia. p. 264.

33. Piekarski, W., R. Smith, and B.H. Thomas. *Designing Backpacks for High Fidelity Mobile Outdoor Augmented Reality*. in *3rd IEEE and ACM Int'l symposium on mixed and augmented reality,* pp 280-281. Nov 2004. Arlington, VA, USA.

34. Piekarski, W., R. Smith, G. Wigley, B.H. Thomas, and D. Kearney. *Mobile Hand Tracking Using FPGAs for Low Powered Augmented Reality*. in *8th Int'l symposium on wearable computers,* pp 190-191. Nov 2004. Arlington, VA, USA.

35. Piekarski, W. and B.H. Thomas. *Augmented Reality Working Planes: A Foundation for Action and Construction at a Distance*. in *3rd IEEE and ACM Int'l Symposium on Mixed and Augmented Reality,* pp 162-171. Nov 2004. Arlington, VA, USA.

36. Piekarski, W. and B.H. Thomas. *An Object-Oriented Software Architecture for 3D Mixed Reality Applications*. in *2nd Int'l Symposium on Mixed and Augmented Reality*. Oct 2003. Tokyo, Japan.

37. Piekarski, W. and B.H. Thomas. *Using ARToolkit for 3D Hand Position Tracking in Mobile Outdoor Environments*. in *1st Int'l Augmented Reality Toolkit Workshop*. Sep 2002. Darmstadt, Germany.

38. Plessl, C. and a. et. *Reconfigurable Hardware in Wearable Computing Nodes*. in *6th Int'l Sympossium on Wearable Computers*. 2002. Seattle, Wa.

39. Polhemus, *Polhemus Homepage*. 2004.

40. Puttegowda, K., W. Worek, N. Pappas, A. Dandapani, P. Athanas, and A. Dickerman. *A run-time reconfigurable system for gene-sequence searching*. in *VLSI Design, 2003. Proceedings. 16th International Conference on,* pp 561-566. 2003.

41. Rasmussen, C., K. Toyama, and G. Hager D. *Tracking Objects By Color Alone*. in *Technocal Report, Yale University - Department of Computer Science*. Jun 1996. New Haven, CT.

42. Robles, R. *ASIC Versus Reconfigurable Compute Fabric (RFC) Solutions*. in *Technical Report, Motorola.* March 2003. Denver, Co.

43. Roylance, L.M. and J.B. Angell. *A batch fabricated silicon accelerometer*. in *IEEE Trans. on Electron Devices 26,* pp 1911-1917. 1979.

44. Smith, R., W. Piekarski, and G. Wigley. *Hand Tracking for Low Powered Mobile AR User Interfaces*. in *6th Int'l Australasian User Interface Conference*. 2005. Newcastle, Australia.

45. Sutherland, I.E. *The Ultimate Display*. in *IFIP Congress,* pp 506-508. 1965. New York.

46. Sutherland, I.E. *A Head-Mounted Three-Dimensional Display*. in *AFIPS Fall Joint Computer Conference,* pp 757-764. 1968. Washington, DC.

47. Tessier, R. *Negotiated A* Routing for FPGAs*. in *Fifth Canadian Workshop on FIeldProgrammable Devices*. June 1998. Montreal, Quebec.

48. Thiele, N., *Outdoor Hand Tracking Using Reconfigurable Hardware*, in *Computer and Information Science*. 2003, University of South Australia: Adelaide. p. 77.

49. Thomas, D.E. and P. Moorby. *The Verilog Hardware Description language*. 1991: Kluwer Academic Publishers.

50. Wang, C., H. Wang, W. Soh, and H. Wang. *A Real Time Vision System for Robotic Soccer*. in *4th Asian Conference on Robots and its Applications*. Jun 2001. Singapore.

51. Wigley, G. and D. Kearney. *The first real operating system for reconfigurable computers*. in *Computer Systems Architecture Conference, 2001. ACSAC 2001. Proceedings. 6th Australasian,* pp 130-137. 2001.

52. Xilinx, *Xilinx: Programmable Logic Devices, PPGA & CPLD*. 2004.

8 Appendix 1

Final Handel-C source code with provisional structure designed for multi object tracking.

```
/*Structures and constants used from the blob tracking apllication*/
struct RecieveResult{
    unsigned 8 head;
    unsigned 8 command;
    unsigned 8 xVal1;
    unsigned 8 xVal2;
    unsigned 8 yVal1;
    unsigned 8 yVal2;
    unsigned 8 markerNum;
    unsigned 8 tail;
};

/*Threashold value struct*/
typedef struct {
    unsigned 8 YMIN;
    unsigned 8 YMAX;
    unsigned 8 CRMIN;
    unsigned 8 CRMAX;
    unsigned 8 CBMIN;
    unsigned 8 CBMAX;
}THValues;


/*Constants*/
#define SENTINAL_HEAD       0x55
#define SENTINAL_TAIL       0xAA
#define SET_Y               0x01
#define SET_CR              0x02
#define SET_CB              0x03
#define SHOW_THREASHOLD     0x04
#define SET_VISIBLE_WIDTH   0x05
#define SET_VISIBLE_HEIGHT  0x06
#define SET_CAMERA_INPUT    0x07
#define SET_LCD_STATE       0x08
#define SET_MAX_MARKERS     0x09

#define FPGA_CAMERA         0x00
#define CVBS_IN             0x01
#define SVIDEO_IN           0x02

#define MARKER_1            0x01
#define MARKER_2            0x02
#define MARKER_3            0x03
#define MARKER_4            0x04

#define TRACKING_LOCATION   0x01
#define DEVICE_RESET        0x02

#define LCD_OFF             0x01
#define LCD_ON              0x02

#define TRUE                1
#define FALSE               0
```

```
/*************************************************************************************************
* This program is used to preform blob tracking of an object based on colour alone.  To implement   *
* this we have used the Platform abstraction Layer in conjunction with RC200 calls.  The PAL libs   *
* are used for the camera setup, while RC200 calls are then used to capture pixels in YCrCB         *
* colour space.                                                                                     *
*                                                                                                   *
* After each pixel is captured a threasholding routine is used to determine if the pixel falls      *
* into the acceptable colour space of the blob to be tracked.  During the XY scan four variables    *
* are used to remember the left, right, top and bottom loaction of the blob.  These variables are   *
* then used to display the four crosses displaying the four courners of the blob being tracked.     *
*                                                                                                   *
* The results of the tracking are transmitted over the RS232 pors at a baud rate of 115200in the    *
* following format:                                                                                 *
*                                                                                                   *
* [SENTINAL HEAD:8][COMMAND:8][MARKER_NUMBER:8][X_VALUE:16][Y_VALUE:16][HITS:16][SENTINAL_TAIL:8]   *
*                                                                                                   *
* The calibration of the threasholding values is controlled with six variables. Each of the three   *
* colour chanels (Y, Cr and Cb) have a min and max value determining the operation of the           *
* threasholding.  THe Y chanel is used for luminance and thus controlls the lighting conditions. Cr *
* and Cb are used to controll the colour of the blob to be tracked.  Each marker being tracked must *
* have these defined uniquely.                                                                      *
*                                                                                                   *
* Streaming from the FPGA                                                                           *
*                                                                                                   *
* [SENTINAL HEAD:8][COMMAND:8][MARKER_NUMBER:8][X_VALUE:16][Y_VALUE:16][HITS:16][SENTINAL_TAIL:8]   *
*                                                                                                   *
* Command Options are:                                                                              *
*                                                                                                   *
* TRACKING_LOCATION     0x01                                                                        *
* DEVICE_RESET          0x02                                                                        *
*                                                                                                   *
* Controll packets interperated by the FPGA are in the following format:                            *
*                                                                                                   *
* [SENTINAL_HEAD:8][COMMAND:8][XVAL:16][YVAL:16][MARKER_NUM:8][SENTINAL_TAIL:8]                     *
*                                                                                                   *
* SENTINAL_HEAD  = 0x55                                                                             *
* SENTINAL _TAIL = 0xAA                                                                             *
*                                                                                                   *
* Command Options are:                                                                              *
*                                                                                                   *
* SET_Y                 0x01    (XVAL = YMIN, YVAL = YMAX)                                          *
* SET_CR                0x02    (XVAL = CRMIN, YVAL = CRMAX)                                        *
* SET_CB                0x03    (XVAL = CBMIN, YVAL = CBMAX)                                        *
* SHOW_THREASHOLD       0x04    (XVAL = TRUE/FALSE)                                                 *
* SET_VISIBLE_WIDTH     0x05    (XVAL = VISIBLE_VIDEO_WIDTH)                                        *
* SET_VISIBLE_HEIGHT    0x06    (XVAL = VISIBLE_VIDEO_HEIGHT)                                       *
* SET_CAMERA_INPUT      0x07    (XVAL = FPGA_CAMERA/CVBS_IN/SVIDEO_IN)                              *
* SET_LCD_STATE         0x08    (XVAL = LCD_ON/LCD_OFF)                                             *
*                                                                                                   *
* Both incomming and outgoing packets have a marker field defining what marker the packet is        *
* representing, they are as follows:                                                                *
*                                                                                                   *
* MARKER_1          0x01                                                                            *
* MARKER_2          0x02                                                                            *
* MARKER_3          0x03                                                                            *
* MARKER_4          0x04                                                                            *
*                                                                                                   *
*                                                                                                   *
*                                                                                                   *
* Author: Ross Smith                                                                                *
* Date: 30th June 2004                                                                             *
*                                                                                                   *
*************************************************************************************************/

#define PAL_TARGET_CLOCK_RATE 25175000

#include <fixed.hch>
#include "pal_master.hch"
#include "pal_framebuffer16.hch"
#include "pal_console.hch"
#include "MultiTrack.hch"

/* The number of markers to be tracked*/
#define MAX_MARKERS 4
//The number of bits used to represent the number of markers -1.
#define WIDTH 1

/*Set up the default colour threshold values Orange Blue Red Yellow*/
THValues thValues[MAX_MARKERS] =    {{10,240,163,205,66,132},       {10,240,40,95,160,188},
                                     {49, 90, 162, 210, 116, 133}, {10,240,11,120,96,126}};

/*Global data values*/
unsigned 1 show_threashold = MAX_MARKERS;
unsigned 1 newValue;
unsigned 1 newPos;

unsigned 32 finalX[MAX_MARKERS];
unsigned 32 finalY[MAX_MARKERS];
unsigned 32 finalHits[MAX_MARKERS];
unsigned 32 XLocTotal[MAX_MARKERS];
unsigned 32 YLocTotal[MAX_MARKERS];

unsigned 10 VISIBLE_VIDEO_WIDTH;
unsigned 10 VISIBLE_VIDEO_HEIGHT;
unsigned 10 xPos, yPos;
unsigned 10 newXPos[MAX_MARKERS];
unsigned 10 newYPos[MAX_MARKERS];
unsigned 3  markersTracking;

macro expr RS232Port = PalRS232PortCT (0);
static macro expr ClockRate = PAL_ACTUAL_CLOCK_RATE;
macro expr VideoOut = PalVideoOutOptimalCT (ClockRate);


typedef FIXED_UNSIGNED(32, 0) MyFixed;

/*
 * Send 32 bits of data across the serial port from a 32 bit variable.
 */
void RS232send32Bits(unsigned 31 RS232Port, unsigned 32 data){

    PalDataPortWrite(RS232Port, data[7:0]);
    PalDataPortWrite(RS232Port, data[15:8]);
    PalDataPortWrite(RS232Port, data[23:16]);
    PalDataPortWrite(RS232Port, data[31:24]);
}

/*
 * Send a packet across the serial port in the following format:
 *
```

```
            //Make sure it is a valid marker.
            if(marker <= MAX_MARKERS-1 && marker >=0){
                if(recievedPacket.tail == SENTINAL_TAIL){
                    switch (recievedPacket.command){
                    case SET_Y:
                        thValues[marker].YMIN = recievedPacket.xVal1;
                        thValues[marker].YMAX = recievedPacket.yVal1;
                        break;
                    case SET_CR:
                        thValues[marker].CRMIN = recievedPacket.xVal1;
                        thValues[marker].CRMAX = recievedPacket.yVal1;
                        break;
                    case SET_CB:
                        thValues[marker].CBMIN = recievedPacket.xVal1;
                        thValues[marker].CBMAX = recievedPacket.yVal1;
                        break;
                    case SHOW_THREASHOLD:
                        if(recievedPacket.xVal1 == 1)
                            show_threashold = recievedPacket.markerNum;
                        else
                            show_threashold = MAX_MARKERS;
                        break;
                    case SET_VISIBLE_WIDTH:
                        VISIBLE_VIDEO_WIDTH = (recievedPacket.xVal2 @ recievedPacket.xVal1)[9:0];
                        break;
                    case SET_VISIBLE_HEIGHT:
                        VISIBLE_VIDEO_HEIGHT = (recievedPacket.xVal2 @ recievedPacket.xVal1)[9:0];
                        break;
                    case SET_CAMERA_INPUT:
                        if(recievedPacket.xVal1 == FPGA_CAMERA)
                            RC200VideoInSetInput(RC200VideoInInputCamera);
                        else if(recievedPacket.xVal1 == CVBS_IN)
                            RC200VideoInSetInput(RC200VideoInInputComposite);
                        else if(recievedPacket.xVal1 == SVIDEO_IN)
                            RC200VideoInSetInput(RC200VideoInInputSVideo);
                        break;
                    case SET_MAX_MARKERS:
                        if(recievedPacket.xVal1 <=4)
                            markersTracking = recievedPacket.xVal1[2:0];
                        break;
                        /* This was used to determine if the LCD could be disabled via code, the scree
                        turned off but the power consumption remained the same, thus is was of no use
                        for this project.
                    case SET_LCD_STATE:
                    if(recievedPacket.xVal1 == LCD_OFF){
                         PalVideoOutDisable(VideoOut);

                         }
                    if(recievedPacket.xVal1 == LCD_ON){
                         PalVideoOutEnable(VideoOut);

                         }
                         */
                    }
                }
                recievedPacket.head=0;
                recievedPacket.tail=0;
            }
        }
        * [SENTINAL HEAD:8][COMMAND:8][MARKER_NUMBER:8][X_VALUE:16][Y_VALUE:16][HITS:16][SENTINAL_TAIL:8]
        */
        void sendXY(unsigned 10 xPos, unsigned 10 yPos, unsigned 32 finalHits, unsigned 8 markerNum){
            PalDataPortWrite(RS232Port, SENTINAL_HEAD);
            PalDataPortWrite(RS232Port, TRACKING_LOCATION);
            PalDataPortWrite(RS232Port, markerNum);

            PalDataPortWrite(RS232Port, xPos[7:0]);
            PalDataPortWrite(RS232Port, 0@xPos[9:8]);

            PalDataPortWrite(RS232Port, yPos[7:0]);
            PalDataPortWrite(RS232Port, 0@yPos[9:8]);

            PalDataPortWrite(RS232Port, finalHits[7:0]);
            PalDataPortWrite(RS232Port, 0@finalHits[9:8]);

            PalDataPortWrite(RS232Port, SENTINAL_TAIL);

        }

        /*Sends a packet across the serial port indicating the RC200 has been reset, this may be used by
         *the host system to indicate configuration settings need to be sent.
         */
        void sendAlive(){
            PalDataPortWrite(RS232Port, SENTINAL_HEAD);
            PalDataPortWrite(RS232Port, DEVICE_RESET);
            PalDataPortWrite(RS232Port, 0x00);
            PalDataPortWrite(RS232Port, 0x00);
            PalDataPortWrite(RS232Port, 0x00);
            PalDataPortWrite(RS232Port, 0x00);
            PalDataPortWrite(RS232Port, 0x00);
            PalDataPortWrite(RS232Port, 0x00);
            PalDataPortWrite(RS232Port, 0x00);
            PalDataPortWrite(RS232Port, SENTINAL_TAIL);
        }

        /*
         *This methiod reads the serial port and decodes the incomming packets.  Each packet is in the
         *following format:
         *
         * [SENTINAL_HEAD:8][COMMAND:8][XVAL:16][YVAL:16][MARKER_NUM:8][SENTINAL_TAIL:8]
         */
        void getCommand(){
            struct RecieveResult recievedPacket;
            unsigned command;
            int 8 i;
            unsigned marker;

            while(1){
                    while(recievedPacket.head !=SENTINAL_HEAD)
                    PalDataPortRead(RS232Port, &recievedPacket.head);
                    PalDataPortRead(RS232Port, &recievedPacket.command);
                    PalDataPortRead(RS232Port, &recievedPacket.xVal1);
                    PalDataPortRead(RS232Port, &recievedPacket.xVal2);
                    PalDataPortRead(RS232Port, &recievedPacket.yVal1);
                    PalDataPortRead(RS232Port, &recievedPacket.yVal2);
                    PalDataPortRead(RS232Port, &recievedPacket.markerNum);
                    PalDataPortRead(RS232Port, &recievedPacket.tail);

                    marker = recievedPacket.markerNum[WIDTH:0];
```

```
}

/*
 *This method initialises the RS232 serial port then starts the reading command whih decodes incomming
 *control packets.
 */
static macro proc initRS232(){
    par{
        PalDataPortRun (RS232Port, ClockRate);
        getCommand();
    }
}

/*
 *The divider method starts by sending 5 'alive' packets through the serial port to the host computer
 *letting the system know either the tracking system has just been started or reset.  It then enteres
 *normal operating mode where division is performed on the X and Y total by the number of hits in the
 *frame to find the centre locatioin of the marlers being tracked.
 */
void divider(){
    unsigned 32 result;
    MyFixed fixedNumber1, fixedNumber2, fixedNumber3;
    int 10 i;
    unsigned markerNum;

    for(i=0;i<500;i++)
        delay;
    for(i=0;i<5;i++)
        sendAlive();

    while(1){
        if(newValue){

            for(markerNum=0;markerNum<=MAX_MARKERS-1;markerNum++){
                seq{
                    fixedNumber1 = FixedLiteralFromInts(FIXED_ISUNSIGNED, 32, 0, finalX[markerNum], 0)
                    fixedNumber2 = FixedLiteralFromInts(FIXED_ISUNSIGNED, 32, 0, finalHits[markerNum],
                    fixedNumber3 = FixedDivUnsigned(fixedNumber1, fixedNumber2);

                    newXPos[markerNum] = (FixedToInt(fixedNumber3))[9:0];

                    fixedNumber1 = FixedLiteralFromInts(FIXED_ISUNSIGNED, 32, 0, finalY[markerNum], 0)
                    fixedNumber2 = FixedLiteralFromInts(FIXED_ISUNSIGNED, 32, 0, finalHits[markerNum],
                    fixedNumber3 = FixedDivUnsigned(fixedNumber1, fixedNumber2);

                    newYPos[markerNum] = (FixedToInt(fixedNumber3))[9:0];
                    delay;
                    sendXY(newXPos[markerNum], newYPos[markerNum], finalHits[markerNum], 0@markerNum);
                    delay;
                }
            }

            newPos=TRUE;
            newValue=FALSE;
        }
    }
}

/*
 *Read frames captures a video stream and processes it pixel by pixel.  When each pixel arrives
      *segmentation is performed for each of the different marker colours.  If a pixel is within the
      *defined threshold values it's position is added to a total for that frame, a hit counter is
      *incramented.
      */
     static macro proc ReadFrames (VideoIn, VideoOut, FBPtr){

         unsigned 10 X, Y;
         unsigned 32 hits[MAX_MARKERS];
         unsigned 32 Pixel;
         unsigned 24 modPixel;
         unsigned markers;
         unsigned CROSS_SIZE;

         CROSS_SIZE              = 50;   //Used to change the size of the trackign cross
         VISIBLE_VIDEO_WIDTH     = 640;  //Width of the video stream to process. Max=720
         VISIBLE_VIDEO_HEIGHT    = 480;  //Height of the video strean to process. Max=576

         while(1){
             //Read in a pixel in YCrCb
             RC200VideoInReadPixelPairYCrCb(&X, &Y, &Pixel);

             //If we are at the end of the frame.
             if(X== VISIBLE_VIDEO_WIDTH && Y== VISIBLE_VIDEO_HEIGHT){
                 par(markers=0;markers<MAX_MARKERS;markers++){
                     finalX[markers] = XLocTotal[markers];
                     finalY[markers] = YLocTotal[markers];
                     finalHits[markers] = hits[markers];
                     newValue=1;
                     hits[markers]=0;
                     XLocTotal[markers]=0;
                     YLocTotal[markers]=0;
                 }
             }
             else{
                 modPixel = Pixel[31:24] @ Pixel[15:8] @ Pixel[7:0];
                 //If we are in the area of frame to be processed.
                 if(X< VISIBLE_VIDEO_WIDTH && Y< VISIBLE_VIDEO_HEIGHT){

                     //For all the markers
                     par(markers=0;markers<MAX_MARKERS;markers++){
                         if((   Pixel[7:0]   >= thValues[markers].YMIN  && Pixel[7:0]   <= thValues[markers
                             && Pixel[15:8]  >= thValues[markers].CRMIN && Pixel[15:8]  <= thValues[markers
                             && Pixel[31:24] > thValues[markers].CBMIN && Pixel[31:24] <thValues[markers].C

                             //Fix Me ...

                             // to allow all colours to operate individually.
                             //Show the accepted pixels
                             if(show_threashold == 0@markers)
                                 modPixel = 0x0000FF;

                             //Add the current location to a sum of the total location for this frame.
                             XLocTotal[markers] = ((0@X)+XLocTotal[markers]);
                             YLocTotal[markers] = ((0@Y)+YLocTotal[markers]);
                             //Incrament the number of hits.
                             hits[markers]++;

                         }
                         else{
```

```
                        //Show the accepted pixels
                        if(show_threashold == 0@markers)
                            modPixel = 0xFF0000;
                    }
                }
            }
        }

        //Draw the crosses onto the video stream.
        for(markers=0;markers<=(markersTracking-1)[1:0];markers++){
            if((newXPos[markers]==X || newXPos[markers]==X+1) && Y > newYPos[markers] - CROSS_SIZE
            && Y < newYPos[markers] + CROSS_SIZE){
                //modPixel=0x00FF00;
                modPixel = thValues[markers].YMIN @ thValues[markers].CRMIN @ thValues[markers].CBMIN;

            }
            if((newYPos[markers]==Y || newYPos[markers]==Y+1) && X > newXPos[markers] - CROSS_SIZE
            && X < newXPos[markers] + CROSS_SIZE){
                //modPixel = 0x00FF00;
                modPixel = thValues[markers].YMIN @ thValues[markers].CRMIN @ thValues[markers].CBMIN;
            }
        }

        //Write the pixel to the video buffer.
        if(X<=VISIBLE_VIDEO_WIDTH && Y <= VISIBLE_VIDEO_HEIGHT)
            PalFrameBuffer16WritePair (FBPtr, X, Y, modPixel, modPixel);
    }
}

/*Beginnig of program execution initialises four parallel processes*/
void main(){
 PalFrameBuffer16 *FBPtr;

    unsigned markers;
    macro expr VideoIn   = PalVideoInCT (1);
    macro expr RAM       = PalPL1RAMCT (0);

    PalVersionRequire  (1, 0);
    PalVideoOutRequire (1);
    PalVideoInRequire  (1);
    PalPL1RAMRequire   (1);

    markersTracking = MAX_MARKERS;

    //Start the serial port, divider and Video input parallel processes.
        /*Start the four main processes (RS232,MeanCalculation,Video in and VGA out)in parallel*/
    par{
        /*Initialise the RS232 serial port*/
        initRS232(currentData);

        /*Start the divider used for the mean calculation*/
        divider();

        /*Start the Video input in paralel with the Video in enable*/
        PalVideoInRun (VideoIn, ClockRate);

        /*Start frame buffer (VGAoutput) in parallel with the FB enable*/
        PalFrameBuffer16Run (&FBPtr, RAM, VideoOut, ClockRate);

        /*Force the enables before setting the input source and video stream*/
        seq{
            par{
                /*Matching enable statements*/
                PalVideoInEnable (VideoIn);
                PalFrameBuffer16Enable (FBPtr);
            }

            /*Continue initialising, set the video input source*/
            RC200VideoInSetInput(RC200VideoInInputComposite);

            /*Begin the Video input reading frames*/
            ReadFrames (VideoIn, VideoOut, FBPtr, currentData);
        }
    }
}

/*This method writes human readable characters to the serial port*/
void RS232HexToAscii(unsigned 31 RS232Port,unsigned 32 hex){

    unsigned 32 num, lsb;
    num=0@hex;


        if(num>=90000){
            seq{
                PalDataPortWrite(RS232Port, '9');
                num = num - 90000;
            }
        }
        if(num>=80000){
            seq{
                PalDataPortWrite(RS232Port, '8');
                num = num - 80000;
            }
        }
        if(num>=70000){
            seq{
                PalDataPortWrite(RS232Port, '7');
                num = num - 70000;
            }
        }
        if(num>=60000){
            seq{
                PalDataPortWrite(RS232Port, '6');
                num = num - 60000;
            }
        }
        else if(num>=50000){
            seq{
                PalDataPortWrite(RS232Port, '5');
                num = num - 50000;
            }
        }
        else if(num>=40000){
            seq{
                PalDataPortWrite(RS232Port, '4');
                num = num - 40000;
            }
        }
        else if(num>=30000){
```

```
    seq{
        PalDataPortWrite(RS232Port, '3');
        num = num - 30000;
    }
}
else if(num>=20000){
    seq{
        PalDataPortWrite(RS232Port, '2');
        num = num - 20000;
    }
}

else if(num>=10000 && num<20000){
    seq{
        PalDataPortWrite(RS232Port, '1');
        num = num - 10000;
    }
}


if(num>=9000){
    seq{
        PalDataPortWrite(RS232Port, '9');
        num = num - 9000;
    }
}
if(num>=8000){
    seq{
        PalDataPortWrite(RS232Port, '8');
        num = num - 8000;
    }
}
if(num>=7000){
    seq{
        PalDataPortWrite(RS232Port, '7');
        num = num - 7000;
    }
}
if(num>=6000){
    seq{
        PalDataPortWrite(RS232Port, '6');
        num = num - 6000;
    }
}
else if(num>=5000){
    seq{
        PalDataPortWrite(RS232Port, '5');
        num = num - 5000;
    }
}
else if(num>=4000){
    seq{
        PalDataPortWrite(RS232Port, '4');
        num = num - 4000;
    }
}
else if(num>=3000){
    seq{
        PalDataPortWrite(RS232Port, '3');
        num = num - 3000;
    }
}
else if(num>=2000){
    seq{
        PalDataPortWrite(RS232Port, '2');
        num = num - 2000;
    }
}

else if(num>=1000 && num<2000){
    seq{
        PalDataPortWrite(RS232Port, '1');
        num = num - 1000;
    }
}

if(num>=900 ){
    seq{
        PalDataPortWrite(RS232Port, '9');
        num = num - 900;
    }
}
if(num>=800){
    seq{
        PalDataPortWrite(RS232Port, '8');
        num = num - 800;
    }
}
if(num>=700){
    seq{
        PalDataPortWrite(RS232Port, '7');
        num = num - 700;
    }
}
if(num>=600){
    seq{
        PalDataPortWrite(RS232Port, '6');
        num = num - 600;
    }
}
else if(num>=500){
    seq{
        PalDataPortWrite(RS232Port, '5');
        num = num - 500;
    }
}
else if(num>=400){
    seq{
        PalDataPortWrite(RS232Port, '4');
        num = num - 400;
    }
}
else if(num>=300){
    seq{
        PalDataPortWrite(RS232Port, '3');
        num = num - 300;
    }
}
else if(num>=200){
    seq{
```

```
                PalDataPortWrite(RS232Port, '2');
                num = num - 200;
            }
        }

        else if(num>=100 && num<200){
            seq{
                PalDataPortWrite(RS232Port, '1');
                num = num - 100;
            }
        }

        if(num>=90){
            PalDataPortWrite(RS232Port, '9');
            num = num-90;}
        else if(num>=80){
            PalDataPortWrite(RS232Port, '8');
            num = num-80;}
        else if(num>=70){
            PalDataPortWrite(RS232Port, '7');
            num = num-70;}
        else if(num>=60){
            PalDataPortWrite(RS232Port, '6');
            num = num-60;}
        else if(num>=50){
            PalDataPortWrite(RS232Port, '5');
            num = num-50;}
        else if(num>=40){
            PalDataPortWrite(RS232Port, '4');
            num = num-40;}
        else if(num>=30){
            PalDataPortWrite(RS232Port, '3');
            num = num-30;}
        else if(num>=20){
            PalDataPortWrite(RS232Port, '2');
            num = num-20;}
        else if(num>=10){
            PalDataPortWrite(RS232Port, '1');
            num = num-10;}
        else
            PalDataPortWrite(RS232Port, '0');


        num = num + 48;
        PalDataPortWrite(RS232Port, num[7:0]);

}
```