Implementation of a Real-time Hand Tracking System in a Heterogeneous Architecture

Francis Snelgar

A report submitted in part for the degree of
Bachelor of Engineering R&D (Honours)
The Australian National University

October 2015

Except where otherwise indicated, this report is my own original work.

Francis Snelgar
30 October 2015

Contributions

This work is a joint effort between:

- the author, who conducted the majority of the analysis and implementation;
- the author's supervisors, Dr Jochen Trumpf and Dr Viorela Ila, who were of great assistance in discussing the issues encountered;
- Dr David Austin, who designed the hardware and assisted in developing some of the code and debugging;
- and Dr Desmond Chik, who developed the tracking algorithm.

Abstract

This report discusses the hardware based, real time implementation of a 3D articulated hand tracking system in a stochastic approximation framework. A control architecture is developed that is suitable for real time implementation. The architecture uses image processing pipelines implemented on an FPGA for computational efficiency, achieving maximum data rates of close to 10Gb/s. This results in a 150 fold speed increase over the previous system. Also presented is an analysis of tracker accuracy and convergence requirements at different video frame rates.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Marker-less hand tracking has experienced a renewal of interest in recent years due to its application to human-computer interactions (HCI). Historically, systems have struggled to track poses reliably due to computational expense, and in an attempt circumnavigate this have resorted to focusing on a small subset of possible poses. However as touchscreen technologies have become more widespread, gestures such as pinch and zoom have become commonplace. Focus has hence moved to fully articulated tracking which brings with it an increase in complexity and computational requirements. As such there are very few systems capable of performing fully articulated real time tracking, with the only two implementation known to date demonstrated by Microsoft [64] and Tompson [72], both of which use depth information.

Real-time and marker-less hand tracking is a complex task, and there are several well known problems in the field:

- **Complex Poses:** As the fingers have multiple degrees of freedom that are essentially independent from one another, self-occlusion occurs frequently. Self-occlusion happens when a portion of the hand obscures another from view, effectively hiding the pose. This can occur as a finger crossing in front of another, the palm presented side on to the camera, or examples such as making a fist, effectively hiding all fingers from view. Additionally finger flexion can reach high rotational [52] velocities, resulting in large changes in pose between frame n and frame $n+1$, increasing the difficulty of tracking.

- **Marker-less Tracking:** Marker-less systems which use no physical identifiers are particularly difficult to track as all information regarding the pose must be extracted from the camera images. Feature extraction of this nature currently cannot provide the level of detail or accuracy that markers can, such as the exact location of joints in 3D space.

- **Real time:** Systems which track in real time must either perform all required computations between frames, or operate at a delay of a frame or more. Either way the time window available for performing tracking computations is small. This leads to real time systems using a small subset of available hand poses, or the use of dedicated hardware to perform the bulk of the computations.

1.1 Report Outline

The aim of this report is to implement an existing real time hand tracking system on a heterogeneous architecture using stereo cameras. The tracking system was originally developed by Chik [18] in his PhD thesis, and a feasibility study of such an implementation was conducted by the author in [68]. The contributions made by the author in the current work are:

1. Design of a control architecture for heterogeneous hardware suitable for hand tracking applications with real time performance.
2. Implementation of the architecture on heterogeneous hardware.
3. Proof of concept unit test that allows the loading of an image stream into the beginning of the image processing pipeline implemented on the FPGA, and subsequent data transfers requested by the tracking algorithm implemented on the host computer. The unit test shows successful articulated hand tracking within the limitations of the design and as yet unsolved hardware bugs.
4. Analysis of the differences of the current implementation when compared to Chik's original tracker.
5. Analysis and discussion of the effect of frame rate on tracking accuracy and computational load.

The report is structured as follows:

Chapter 2 contains a survey of prior works in the field, focusing on the different approaches taken as well as a survey of the types of computer vision algorithms successfully implemented on FPGAs.
Chapter 3 introduces the original tracking system and its components.
Chapter 4 and **Chapter 5** introduce the hardware to be used for this implementation and the constraints that it imposes.
Chapter 6 validates the results of the current implementation against those of the original tracker as implemented in [18].
Chapter 7 explores the benefits of using a high frame rate camera, focusing on tracking accuracy and computational efficiency.

Chapter 2

Prior Works

2.1 Approaches to Hand Tracking

Hand tracking can be divided roughly into three types of systems, data glove systems which use physical sensors and markers, templated systems which match the input to a pose from a discrete set, and model based systems which estimates the pose using a kinematic model.

2.1.1 Glove Based Systems

A large portion of glove based systems use an electromechanical glove to measure the pose of the palm and knuckle angle directly such as in [61, 31, 9, 30]. There are several different methods for collecting data in this way, though gyroscopes and accelerometers are commonly used [31, 30] to measure pitch, roll and yaw at the joints. Fels *et al* [31] supplement these with fibre optic transducers to measure knuckle angles, then comparing the measurements to a database of 230 poses. The system requires extensive training with thousands of sample poses and the offline training process takes several hours.

A different approach was taken by Lamberti [48] where instead of inertial sensors, the glove consisted of different coloured sections. By segmenting the image, these regions can be identified, and the pose described by a vector of the angles and distances between these regions. By comparing against a database using a classifier, the pose can be recognized. While they produce an accurate estimation of the pose, data gloves require exact calibration to function correctly, are expensive, and greatly inhibit natural human motion making them undesirable for tracking applications such as human computer interaction. These drawbacks resulted in the development of contactless systems which use computer vision techniques to estimate the pose remotely. Contactless tracking systems can be divided into two discrete types; templated systems and model based systems.

2.1.2 Templated Systems

Appearance based systems map the visual input to a set of images using a variety of classifying techniques. Systems such as [24] use Self Organising Maps (SOMs) to map

the visual input to a network of nodes (the map). Each node has associated with it a weight vector, and a position described in the map space. At every iteration the visual input is randomly sampled, and the node closest in the map space is chosen. The weights and positions of this node and its neighbors are then updated to move them towards the sampled data. This is repeated for thousands of iterations until the map closely resembles the input data. Chang [14] maps the input to the database using a nearest neighbor approach. From the extracted silhouette the curvature scale space is computed using the closed curve of the silhouette (I.e. the outline of the silhouette, which forms a closed loop). By repeatedly convolving the closed curve with a kernel, combined with the gradually smoothed hand contour, peaks and concave features are extracted.The 5 maximum peaks and 4 maximum concavities are then identified and matched to the training set to identify the hand pose. Only 6 discrete poses were used with a training set of 600 images from different perspectives and subjects, in order to increase the system's robustness.

Other approaches use statistical models such as hidden Markov or point distribution models [15, 2], with one model per pose in the discreet set. Each model is evaluated against the current data, and the pose with the best fit is chosen. These appearance based systems provide robust real time implementation for a small, discrete set of poses, but become vastly more computationally expensive for larger sets. For example the computational expense of systems using a model per pose, would scale at best one to one with the number of poses modeled.

2.1.3 Model Based Systems

Model based systems use visual cues to fit a 3D model to the observed pose. Because of the high degree of freedom in a human hand, reduction of dimension methods are often employed, as well as dynamic modeling to increase the system's robustness. Model based algorithms can be largely divided into two distinct techniques: gradient based or particle filter based. Gradient based algorithms such as [18] use a cost function to quantify goodness of fit, and an optimisation algorithm to converge on the global maximum of the cost function. Particle filter based systems such as [25, 34, 50] use a hidden Markov model approach, where observable states (visual cues) are used to probabilistically determine hidden states (poses). Both of these techniques require the use of sample sets, as evaluation of every point is infeasible due to computational expense. There are a variety of different sampling schemes used such as [34] where the sample size is determined by the curvature of the model, or [18] where random samples are taken from each portion of the hand.

While some systems such as [18, 25] use a model with the full 26 degrees of freedom, reduced order models are common in an attempt to limit the complexity of the model. Tompson [72] uses a linear blend skin model with 42 degrees of freedom with a neural network for feature extraction based off depth images. The network is trained offline by projecting the model onto a 2D depth image, and the pose iteratively adjusted until the error is below a threshold. However this is a computationally expensive exercise and training the network took several days using a GPU and 24

core processor. For online tracking a reduced model of 23 degrees of freedom was used for computationally efficiency. The author notes that the reduced model fails to accurately model effects such as skin folding and muscle deformation. Systems such as [34] use a 26 DOF model, but reduce the parameter space through statistical techniques using Gaussian distributions, with restrictions placed on variables using frame rates. The parameter space used is then reduced to 8 DOF, 3 for translation, 3 for rotation, and a further 2 for statistical variables. The models used in [63] have 6 DOF. This dimension reduction is achieved by considering only the DOF of the palm, while the phalanges are considered fixed. Obviously using this model alone would constrict the range of poses to a very small subset, hence three different models are used with different arrangements of the phalanges. These models, combined with appearance based tracking methods, allow the hand to be tracked. A variety of techniques are also used to try and increase the robustness of the system. [34] uses behavioral modeling to estimate what action the user should be performing. For example if the user is reaching for an object the next motion will likely be 'grasping'. [63] takes a different approach by combining both appearance and model based techniques. Template matching is used to determine which of several restricted models to use, then visual cues and accelerometer data is used to refine the pose. Model based systems offer the most exact pose recognition, and have the advantage of being able to model any pose without having prior knowledge of it such as an appearance based system. However this comes at a cost, with model based systems generally requiring significantly more computational power than other approaches.

2.1.4 Visual Cues

Both appearance and model based systems rely heavily on visual cues to refine the pose estimate. This section will discuss the various cues used, and the techniques used to quantify them.

2.1.4.1 Silhouette

The silhouette of the hand is commonly used in both appearance based and model based approaches [14, 25, 34, 50, 63, 33], both for use in optimisation algorithms and template matching by constraining the area the model or pose can occupy. There are several different techniques used to calculate the silhouette. A common method is to use a learned histogram model which is trained offline [43]. This approach uses the property that skin colour is relatively uniform, hence occupying a compact region of the colour space. Using training data, the colourspace is classified as skin or nonskin, creating two probability distributions. Comparison of the distributions is used to classify pixels online. While this approach is accurate in identifying skin with the accuracy exceeding 90%, it has its drawbacks, mainly related to the requirements for offline training, as manually labeling pixels is time intensive and individual histograms need to be constructed for each subject due to differences in skin tone. Another technique used is adaptive background mixture models [69]. Each pixel is

modeled as a mixture of Gaussians. These Gaussian distributions are then evaluated to determine which Gaussians are most likely the result of a background process. Pixels that do not match background Gaussians are grouped as connected components and tracked over time using a multiple hypothesis tracker. However variations in lighting and introduction of new objects can cause objects to be falsely identified as foreground, with the time taken to correctly classify them being significant.

Almaadeed *et al* [3] sought to overcome some of these issues by using the Gaussian mixture model as an initial segmentation step only. All but one foreground segments are then discarded based on size and shape. The RANSAC algorithm [21] is then used to reject outliers and obtain the silhouettes. The background is then updated. A different approach taken by Felzenswalb *et al* [32] is to consider only the intensity of the image. The algorithm creates regions in such a way that the variation in intensity inside a region is less than the variation between regions. If the variation between neighboring border pixels of two different regions is less than the internal variation, the regions are merged in a bottom up approach. Drawbacks to this approach are that whilst the image is divided up in to discrete regions, there is no procedure for identifying the correct region.

2.1.4.2 Edge Detection

Edge detection is a very common cue both in appearance and model based systems [24, 15, 25, 33, 27, 62] as it is a robust operation that helps with database matching and cost function optimisation. The most famous edge detection algorithm, the Canny operator [13] locates where the gradient magnitude is a local maximum along the direction of the gradient, similar to an approach first suggested by Marr [55]. Thresholding with hysteresis is then applied to identify the edges. Initially a large threshold is used to identify points which are clearly edges. The threshold is then switched to a lower value which is traced along the already identified edges. This enables the faint edges to be picked up, while minimising false edges. Wenshuo *et al* [35] suggest an improvement on the Canny operator by additionally using a depth map. The depth map would show a change in intensity between overlapping objects as they are at different distances along the Z-axis (camera axis). However the Canny operator is not particularly robust in terms of noise. A suggested improvement was to use wavelet decomposition [17]. By decomposing the image into wavelets, an appropriate threshold can be chosen to remove the noisy component from the spectrum. The inverse can then be computed and the Canny operator used on an image with significantly reduced noise. A different approach taken by Hsaio *et al* [40] was to use an iterative approach using empirical mode decomposition. By repeated subtraction from the signal until an appropriate threshold is reached, the intrinsic mode functions can be found. These along with the residue function are used to identify edges in a colour image.

2.1.4.3 Optical Flow

Optical flow is another commonly used cue [15, 28] and has also been used in silhouette algorithms as part of background subtraction [75], though as a scalar rather than vector quantity. A popular approach is one suggested by Horn and Schunk in the 1980s [39]. It uses the approximation that a pixel's intensity does not change over small temporal displacements (adjacent frames), and hence any change in intensity is related to the relative velocity of the image. Taking partial derivatives results in the optical flow constraint equation. As this is a underdetermined linear system, additional constraints must be imposed to insure uniqueness. These are added in the form of smoothness and intensity constraints. This reduces the problem to minimising the error in intensity over a neighborhood and extrapolating to find the optical flow vectors. Chen *et al* [16] use these principles for motion compensation. However techniques such as this find the projection of the optical flow along the gradient. Chung *et al* [22] propose a back propagation approach which propagates the projection of the optical flow back along the gradient, and hence restoring the true value.

2.1.4.4 Stereo Vision

Stereo vision is used by many [27, 8, 12, 37, 51, 58, 59, 66, 77]to gain depth information of the scene by identifying common regions in both camera views and calculating depth using basic trigonometric relationships. Typically the depth map is computed only for visual features that are identified in both views - i.e. corners or edges that can be easily detected using Canny or Sobel filters. This depth information can then be used to build a map of the background [58], from which regions of interest (ROI) can be separated using background subtraction. Another technique which is more commonly used in object tracking [12, 37], is to build a plan or plane view map, where the physical plane is divided into bins. A physical stereo camera is used to generate the depth map as normal. This map is then orthographically projected to the viewpoint of an overhead camera, where each bin is assigned a single pixel value. This helps with object separation as often when two object partially occlude one another from the physical camera viewpoint, they will be clearly separated in the virtual viewpoint. While this would result in a loss of data, (as this is effectively a reduction in dimension) enough data is still present for tracking. Muñoz-Salinas *et al* [59] extend the idea of depth maps further by using multiple stereo cameras at different view points to build a depth and volume map. Using multiple cameras allows the depth of different faces of objects to be calculated and hence build a volume map of the object in question. Using multiple cameras also helps to deal with some inherent drawbacks of stereo cameras, namely noise and false identification. By using multiple cameras the author constructed a confidence map, which describes the level of certainty at that point in the depth map. Geometric shapes such as checkerboards stand out with high values in the confidence map, while regions of uniform colour have low values. While most approaches use stereo vision to generate depth maps for use in cost functions or template matching, other approaches integrate stereo vision more fully into the tracking systems. For example Ziodi *et al* [77] implement an

appearance based tracker where the left right camera images are used independently. Disparity maps for candidate ROI are generated separately against the current reference at the top of the reference stack. The similarity between the left right pairs is then computed and if its over the threshold, this pair is pushed on to the stack as the next reference, and the oldest pair is popped. Dankers *et al* [27] take a different approach and use disparity information to track motion, and hence the direction for the tracker. One camera is used as a reference, and one column of the image in the other camera is removed, and a column of padding introduced on the opposite side. This process is repeated for pixels on the left and right of the image, and the direction which produces the largest disparity is the right direction for the tracker. This is working on the assumption that object being tracked is a rigid body and that it is moving along the horizontal axis of the camera.

2.2 FPGA Implementations

2.2.1 Tracking Systems

While most of the tracking systems mentioned are implemented on standard CPU based architectures, others have taken advantage of the parallel computing power of FPGAs. However because of the architecture, tracking algorithms tend to be simpler then the CPU based systems. For example probabilistic histogram models are popular choices [4, 20], where regions of interest are tracked using colour values. [20] uses exhaustive search methods where the whole of the frame is searched for the best match, achieving frame rates of up to 81 fps. [4] uses a slightly more advanced search algorithm, which uses a gradient based iterative approach, increasing frame rates up to 290 fps for multiple targets. Hu [41] combines histogram thresholding with adjacency testing for a blob tracking algorithm using a softcore processor. Other blob tracking algorithms include [60, 6] where edge detection is used for locating geometric regions of interest. However algorithms using purely image processing techniques such as histograms and linear filters are sufficient only for blob tracking, lacking the complexity required for more advanced scenarios. Slighty more advanced blob tracking algorithms such as Nilsson *et al* [47] use the segmentation approach with frame rates of 30 fps, while [19] uses image subtraction and particle filters achieving frame rates of 79 fps. Several systems use multiple viewpoints [44, 46] for 3D positioning. Kikuchi [44] uses distributed nodes with each containing an FPGA for the computation of optical flow, while Kreizer [46] uses an FPGA for for ray tracing in the modeling of turbulent flow in fluids. FPGAs are also commonly used as embedded standalone systems in the automotive industry. Marzotto [57] implements a lane departure system using an FPGA for feature extraction and curve fitting using RANSAC, tracking the results over time with a Kalman filter. Ando [5] tracks drivers eye position using foreground extraction and template matching to extract the face region, then fitting circles to regions of interest to segment the eyes.

2.2.2 Optical Flow

Optical flow algorithms are another popular choice for implementation on FPGAs [36, 29]. Saranli *et al* implemented the Horn and Schunck algorithm on an FPGA, achieving frame rates of 257 fps. Martin [56] also successfully implemented the Horn and Schunk algorithm using external memory banks. Other approaches include [71] where the evaluation of cost in a window and pattern matching was used to determine optical flow. Diaz also successfully implemented optical flow algorithms using an FPGA along with external memory banks, with frame rates of 30 fps.

2.2.3 Distance Transforms

Distance transforms are a commonly used metric in computer vision algorithms and are applied spatially or to properties such as in clustering algorithms. As most such algorithms scale as $O(n^2)$, they are computationally expensive and for real time implementations FPGAs are commonly used. The most commonly used distance transform [38, 7] is the Borgefors transform [10] where a non linear filter is used to compute integer approximations to the Euclidean distance in a neighbourhood. These local distances are propagated to neighbouring pixels. Distances are propagated globally using a sliding window approach and consecutive passes in the forward and backward directions. Atay [7] implements a parallel architecture where each image is broken into sub images and dealt with separately. Border artefacts are then removed later. Sudha [70] also implements a parallel architecture where individual processing units compute the Euclidean distance in local neighborhoods iteratively, scaling at worse as $O(n)$. Gonzalez [1] calculates the distance to an edge as a disparity measure, focusing on vertical edges to remove the complexity of square root or divide operations. Distance transforms an also be applied to colourspaces such as in [76] where it is used to segment an image using the K-means clustering algorithm. Boudabous also applies it to a colourspace [11] where it is implemented as part of a directional filter for noise removal.

2.2.4 Random Number Generation

Random number generators (RNG) play a vital role in cryptographic applications as well as random sampling schemes. RNGs can be split into two broad categories, pseudo (PRNG) [54, 53, 67] where the output is produced through computation, and true RNG [65, 74] cannot be computed as they are nondeterministic. In recent years there has been a move to hardware based generators, as these offer an efficient and low power method of producing large streams of random numbers, and can be constructed in such a way which cannot be replicated. FPGAs prove a popular platform [65, 54, 53, 74, 67] for implementation. For true RNGs, a source of randomness is required. Ransinghe [65] forces timing violations in an FPGA to cause gate metastability, which produces random events due to race conditions. In [74], the Look Up Tables (LUTs) in the FPGA fabric are randomly connected at configuration, and on

power up, hot carrier injection is used to randomly seed these LUTs, where, by providing transistors with enough current over an extended period of time the oxide barriers between gates can be broken.

The most basic PRNG is the linear feedback shift register [45], where the output (least significant bit) is a combination of other bits using boolean operations. While this is not suitable for cryptographic applications as it is easily broken, it is sufficient for some modeling applications. More advanced methods such as in [54] use Euler approximation to solve a system of third order ODEs exploiting instability around stationary points. Other implementations focus on shaping the distribution of input random numbers such as Sileshi [67] where CORDIC functions are used on an FPGA to obtain a Gaussian distribution.

Chapter 3

Tracking System

The articulated tracker used in this work was originally designed by Chik [18]. A full account of the system can be found in Chapter 3 of his PhD thesis. The following is a brief overview of the system and its components.

The tracker uses an online optimisation algorithm to fit the estimated pose to the observed pose. There are four major components: stereo images produced from a camera pair, an articulated hand model, a cost function and an optimisation algorithm. Using cues extracted from the pair of images, the observed pose is compared to the projection of the hand model. The cost function is used to quantify this error which is back-propagated along with the image gradients to the optimisation algorithm, which refines the pose iteratively.

3.1 Stereo Cameras

The images are obtained from a pair of calibrated stereo cameras, configured as in Figure 3.1. Colour calibration is achieved through the use of a colour palette so that intensity values are consistent between both cameras. Checkerboards are used for the extraction of the intrinsic and extrinsic matrices which are required for the projection of the hand model.

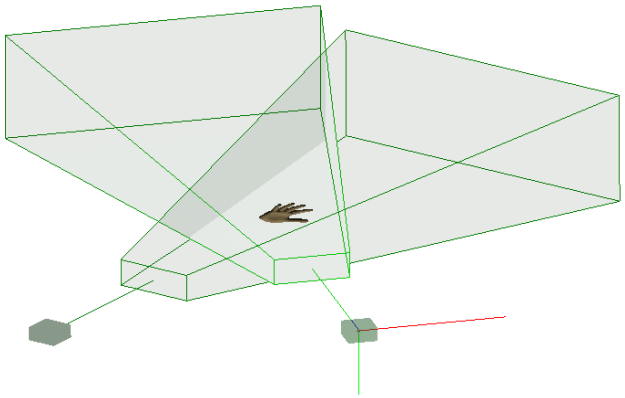

Figure 3.1: Physical arrangement of the tracking space, showing the stereo cameras and their viewing frustums, as well as the users hand.

3.2 Hand Model

The tracker uses a fully articulated hand model with 26 degrees of freedom. The underlying skeleton is expressed as a kinematic chain originating at the base of the palm. A deformable mesh is attached to the skeleton to form the skin, using linear blending to model skin deformation. The model is constrained through the use of static constraints to set the allowable rotation range at the joints, and kinematic constraints such as rotation at the MCP joint during finger flexion to avoid collisions. As projecting the entire hand model is computationally expensive, a random sample is projected instead, taken from the vertices of the skin mesh. Figure 3.2 shows the skeleton and labeled joints as well as sample points taken from the mesh.

Figure 3.2: Left: articulated skeleton showing the carpometacarpal (CMC), metacarpophalangeal (MCP), proximal-interphalangeal (PIP) and distal-interphalangeal (DIP) joints. Right: sample points taken from the mesh, showing the palm (green), thumb (red), index(blue), middle (violet), ring (yellow), and little (violet) fingers.

3.3 Cost Function

The cost function is used to quantify the differences between the observed pose and estimated pose by comparing the projection of model points and the corresponding pixels in the camera images. The cost function consists of three components: the photoconsistency function (C_p) which makes use of colour intensity, and the filling (C_f) and silhouette (C_s) functions, which use the silhouette extracted from the camera images. The total cost at estimated pose X, is given as the weighted sum of all three, shown in Equation 3.1, where α and β are the weights.

$$C_{X^*}(X) = \alpha C_s(X) + \beta C_f(X) + C_p(X) \tag{3.1}$$

3.3.1 Photoconsistency Function

If the estimated pose is correct, then the projection from a point on the surface of the hand to pixel coordinates in both camera images will have the same intensity. If $I(s_i, j)$ is a function mapping the i^{th} sample point from the surface of the hand model

to pixel coordinates in the j^{th} camera image, then the photoconsistency function is given by Equation 3.2, where N is the size of the sample set.

$$C_p(X) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{2} \left\| I(s_i, 1) - I(s_i, 2) \right\|^2 \tag{3.2}$$

3.3.2 Silhouette Function

The silhouette function increases the cost when projected points from the surface of the hand model lie outside the real silhouette. If $D(s_{i,j})$ is the distance transform [10] at the projection of the i^{th} sample point in the j^{th} camera view, then the silhouette function is given by Equation 3.3, where N is the size of the sample set.

$$C_s(X) = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{2} D(s_{i,j})^2 \tag{3.3}$$

3.3.3 Filling Function

The filling function increases the cost when the projection of the hand model does not fill the real silhouette completely. If $\hat{s}_{i,j}$ is the i^{th} sample in the j^{th} camera view from a sample set of size M taken from inside the real silhouette, and $h_{i,j}(X)$ is the point on the model surface whose projection is nearest, then the filling function is given by Equation 3.4.

$$C_f(X) = \frac{1}{M} \sum_{i=1}^{M} \sum_{j=1}^{2} \frac{1}{2} \left\| \hat{s}_{i,j} - h_{i,j}(X) \right\|^2 \tag{3.4}$$

3.4 Optimisation Algorithm

The tracker estimates the pose using the cost functions and their gradients. The optimal pose X^* is given by

$$X^* = \arg\min C_{X^*}(X) \tag{3.5}$$

such that $X \in \phi$ where ϕ is the allowable set of hand poses. As an exhaustive computation is not possible due to expense, the pose is estimated by

$$X^*_\Phi = \arg\min \mathbb{E}_\Phi[C_{X^*}(X, \Phi)] \tag{3.6}$$

where $C_{X^*}(X, \Phi)$ is the observed cost under the effects of noise due to point sampling. The optimal pose then becomes X^*_Φ. Chik [18] proved that under certain conditions stochastic approximation theory states that $X^*_\Phi \to X^*$, and that it is possible for the estimate of the cost function to converge to the optimal pose. Stochastic meta descent (SMD) is used, which offers faster convergence than traditional gradient descent. While there is no proof for convergence [68], it has been shown to work empirically.

Chapter 4

Hardware

The physical system consists of two self contained camera units, and a host computer. Each unit contains an FPGA, microcontroller, three SDRAM blocks, and a CMOS image sensor. The physical configuration of a unit is shown below in Figure 4.1. Note that only one of the three SDRAM blocks is shown for clarity.

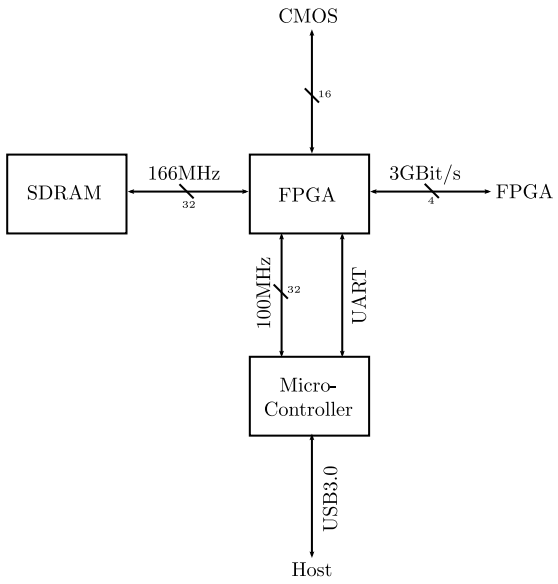

Figure 4.1: Block diagram of physical system design. Note that only one of the camera units is shown

The FPGA and microcontroller are connected by a UART interface, as well as a customizable 32bit interface. There is also a high speed inter-camera bus with 2bits in each direction at speeds up to 3Gbits/s.

4.1 FPGA

The FPGA used is the Lattice ECP3-150 [49], a high-speed, low power FPGA, the specifications of which can be found in Table 4.1. An external chip provides both a master clock for the CMOS sensor, and a 128MHz reference clock to the FPGA.

Resource	Quantity
LUTs	149,000
sysMem	6696 Kbits
Embedded Memory	6850 Kbits
Distributed RAM	303 Kbits
18×18 Mulitpliers	320
SERDES	4
PLLs/DLLs	10/2

Table 4.1: Lattice ECP3-150 Resource Specifications

Derived clocks can then be generated from the reference clock using phase locked loops (PLLs) [42]. Additional capabilities include serialiser/deserialiser (SERDES), which are used to provide an interface between the serial output from the CMOS sensor and the FPGA, allowing far higher data rates than are typically possible on a standard FPGA architecture. The ECP3 also includes a digital signal processing (DSP) architecture, with 320 slices configurable for 18×18 operations (multiplication, addition, multiply and carry etc).

4.2 Microcontroller

The microcontroller used is the Cypress FX3 USB3.0 controller [26], using an ARM926EJ core, a 32bit CPU running at 200MHz, and supports a fully IEEE 754 compliant soft floating point unit. There is support for connection to peripherals via a number of different buses, including UART and a general purpose interface (GPIF), a customisable 32bit bus running at speeds up to 100MHz.

4.3 SDRAM

There are three SDRAM blocks connected to each FPGA. The SDRAM used belongs to the Winbond 989 family [73], a low power SDRAM with 512Mbits of memory configured in 32bit words. Read and write speeds are up to 166MHz.

4.4 CMOS Sensor

The CMOS sensor used is the CMV2000 by CMOSIS [23]. It is a high speed image sensor, capable of 340fps with a frame resolution of 2048×1088 pixels, and significantly higher frame rates at lower resolutions. These high speeds are achieved by using a 16 channel parallel read-out, with the image divided into vertical strips.

Chapter 5

System Architecture

Chik's original implementation was not capable of running the tracking algorithm in real time (30fps) due to the computational load required, mainly in the computation of the cost function. Because of this a heterogeneous architecture is used, with different parts of the tracking algorithm implemented on the hardware best suited to those specific computations. FPGAs are well suited to performing large numbers of basic computations (fixed point), as parallel processing is readily implementable. However floating point operations, and more complex algorithms are non trivial to implement, and require large amounts of resources. To this end the optimisation algorithm and hand model are implemented on the host computer in C. The image processing and computation of the cost function are implemented on the FPGA. The random sample sets from the surface of the hand model will be constructed on the host, and sent to the FPGA via the GPIF. The requested values required for the cost function will then be returned. A more comprehensive discussion of the flow of data can be found in the authors previous work [68].

5.1 Computation Timing

However there are constraints imposed by real time implementation. As the optimisation algorithm iteratively computes the cost of a set of sample points, and this process can not start until the entire frame has been read out, real time processing would require that the tracking algorithm computations be constrained to the time between frames (23μs at 340fps) or to store the image in memory and operate at a delay of one frame. As the optimisation algorithm requires iterative computation of sample sets, this also requires the image to be buffered long enough for this process to be completed. Further discussion can be found in [68].

There are also limitations imposed by the physical hardware layout. The tracking algorithm requires numerous data transfers between the optimisation algorithm and cost function per frame. As mentioned these components are implemented on separate hardware architectures (CPU and FPGA respectively) and hence this requires numerous data transfers between these. The GPIF is the only high bandwidth data bus between the two and hence must be used for these transfers. Similarly the inter-camera bus is the only bus available for accessing data from the second camera unit.

5.2 Data Rate Limitations

Due to the extremely high data rates involved (up to 20Gbit/s), care needs to be taken when designing the system to avoid data overflow or potential computational bottlenecks. The following section identifies several areas where this could occur and discusses potential solutions.

5.2.1 Image Processing Pipeline

The accepted standard for computations on large data streams on FPGAs is the stream processing paradigm. Data is pushed through a pipeline where kernel functions operate on it at each step. By removing the read-write aspect of data manipulation much higher throughputs can be achieved. Each pipeline processes the raw data from the CMOS sensor and writes the results into SDRAM. The raw data is read out in 16 channels, at clock rates up to 48MHz [23]. At high clock rates, this results in the underlying pixel rates exceeding the maximum clock speeds possible on the FPGA (nominally 375MHz for -6 grade FPGA [49]). This leads to two possible solutions, downsampling or parallel channels, where vertical strips of the image are processed independently. By downsampling before the image pipeline by a factor of 2 in both directions, the underlying pixel rate is reduced by a factor of 4. This would decrease the maximum clock rate to 188MHz. Equivalently, the pipeline could be split into multiple channels, each of which processes a section of the image in a vertical strip. This has the advantage of keeping the high resolution of the image.

5.2.2 Data Storage

The data from the image pipeline is to be stored in SDRAM, accessed later for the computation of the cost function. However the maximum frequency of the SDRAM blocks is 166MHz, which is significantly less than the underlying pixel rates at higher frame rates. Additionally the volume of data and the format it is stored in also have to be considered. There are four different data streams that are required for the cost function: the intensity of the image, the distance map for the hand silhouette, and gradients of both of these functions. To store data at higher frame rates it is necessary to either reduce the data rate through downsampling, pack multiple pixels into a single SDRAM word, or use multiple SDRAM blocks for writing a single stream.

5.2.2.1 Downsampling

At full resolution and frame rate, the underlying pixel rate is 753MHz. In order to reduce the data rate past 166MHz so that it can be written as a single stream, the amount of data has to be reduced by at least a factor of six, reducing the frame resolution to VGA (640×480). Practically the data rate would have to be reduced further to allow for read operations and overheads such as refresh cycles, changes to the address pointer and the opening and closing of SDRAM banks and rows.

5.2.2.2 Multiple pixels in single word

By treating the left and right sides of the image separately, as well as downsampling by a factor of two in both dimensions, there are two possible implementations. Firstly when writing to SDRAM the two halves of the image could be overlaid, so that there are two pixels in every word. Equivalently the two halves could be written to different SDRAM banks, with each word containing values from multiple data streams. The difference between this and the former method is that in the former, two values from the same stream are packed into the same word, while in the latter, values from two different streams are packed into the same word. The advantage of the latter method is that the gradient can be stored in the same location as the intensity or distance map. As the gradient and value are required for each sample pixel, storing these values in the same word would reduce SDRAM read operations.

5.2.2.3 Data Format

There are three SDRAM blocks accessible from each FPGA, each with a 32bit width. If each half of the frame is processed independently, this means that only 48bits are available for all independent data streams in each image pipeline. However if using a standard 3 channel colourspace, and 8bits for each of the other streams, the pipeline produces 96bits of processed data. However all processed data is derived from other streams which are typically much more compact (i.e. Bayer or silhouette). Hence data can be stored in this format and processed when needed. The downside of this is that to compute each value, several neighbouring values are needed resulting in an increase in latency. The distance map is a non linear function, and as such it is not possible to predetermine which of the surrounding pixels are required for the computation, and hence a large bounded neighborhood would have to be read. The Sobel filter used for gradients and the colour space conversion both use smaller masks, making them preferable as fewer values need to be read. By writing the distance map and gradient in the processed format, and the colour space as Bayer, it is possible to store all relevant information while minimising the number of read operations needed. Note that this utilises all SDRAM blocks so a parallel read/write architecture is not viable and operations must be interleaved.

5.2.2.4 Implementation Selection

Table 5.1 provides a summary of possible implementations, however not all implementations are viable due to the physical constraints imposed by the hardware. Implementations 1-8, 12, and 16 all result in SDRAM data rates exceeding the maximum specified of 166MHz. Implementations 1, 2, 4, 5, 9, 10, 13, 14, 18, 19, 21 and 22 require more SDRAM bandwidth then is physically available. Of the remaining implementations 11 and 15 are not capable of the full frame rate. This leaves implementations 17, 20, 23 and 24 as being physically viable. Of these, implementations 17 and 20 reduce the data rate by storing the raw Bayer pixel values, while 23 and 24 use more aggressive downsampling. While all implementations are viable, the later method is used

as allows full frame rate while minimising the number of SDRAM read operations.

The selected implementation (implementation 24 in red) processes the left and right sides of the image separately at a resolution of 640x480 pixels. All processed data streams are then written to SDRAM by packing multiple streams into a single word, using the full 96bit bandwidth. The resulting SDRAM write rate is 104MHz at 340fps, well under the maximum of 166MHz. As all values are stored in their processed form and in the same word as the gradients, this implementation represents the minimum number of SDRAM read operations possible for a given sample size.

	Image Height	Image Width	Frame Rate	Underlying Pixel Rate	Pixels/ word	SDRAM Data Rate	Intensity Format	Gradient Format	Distance Format	Bits/ pixel	Total Bits
1.	2048	1088	169	377	2	188	10bit Bayer	16bit	8bit	82	164
2.	2048	1088	338	753	2	377	10bit Bayer	16bit	8bit	82	164
3.	2048	1088	169	377	1	377	10bit Bayer	16bit	8bit	82	82
4.	2048	1088	169	377	2	188	24bit YUV	16bit	8bit	96	192
5.	2048	1088	338	753	2	377	24bit YUV	16bit	8bit	96	192
6.	2048	1088	169	377	1	377	24bit YUV	16bit	8bit	96	96
7.	2048	1088	338	753	1	753	24bit YUV	16bit	8bit	96	96
8.	2048	1088	338	753	2	377	10bit Bayer	-	8bit	34	68
9.	1024	544	169	94	2	47	10bit Bayer	16bit	8bit	82	164
10.	1024	544	338	188	2	94	10bit Bayer	16bit	8bit	82	164
11.	1024	544	169	94	1	94	10bit Bayer	16bit	8bit	82	82
12.	1024	544	338	188	1	188	10bit Bayer	16bit	8bit	82	82
13.	1024	544	169	94	2	47	24bit YUV	16bit	8bit	96	192
14.	1024	544	338	188	2	94	24bit YUV	16bit	8bit	96	192
15.	1024	544	169	94	1	94	24bit YUV	16bit	8bit	96	96
16.	1024	544	338	188	1	188	24bit YUV	16bit	8bit	96	96
17.	1024	544	338	188	2	94	10bit Bayer	-	8bit	34	68
18.	640	480	169	52	2	26	10bit Bayer	16bit	8bit	82	164
19.	640	480	338	104	2	52	10bit Bayer	16bit	8bit	82	164
20.	640	480	338	104	1	104	10bit Bayer	16bit	8bit	82	82
21.	640	480	169	52	2	26	24bit YUV	16bit	8bit	96	192
22.	640	480	338	104	2	52	24bit YUV	16bit	8bit	96	192
23.	640	480	169	52	1	52	24bit YUV	16bit	8bit	96	96
24.	640	480	338	104	1	104	24bit YUV	16bit	8bit	96	96

Table 5.1: Summary of the different possible configurations of the image pipeline. Plausible configurations are highlighted in green, with the chosen configuration in red.

5.2.3 Stereo Vision

The architecture must allow for access of the processed image streams from each camera. There are three possible approaches to this, 1) sending the raw image data from the CMOS sensor from the slave FPGA to the master and processing and storing on the master FPGA, 2) processing on separate FPGAs and sending processed data which is stored on the master FPGA, or 3) processing and storing data independently on each FPGA, and only accessing the data needed for the cost function. For all of these options the inter-camera data bus is used. In raw Bayer form (10bpp), the image data from each frame amounts to 7.56Gbit/s. To transmit the frame in this form, the image would first need to be downsampled by a factor of 2 in both dimensions, resulting in a underlying data rate of 1.89Gbit/s. As the volume of data is significantly greater after processing, clearly it is not possible to transmit it in a processed form. The chosen option is to read only the data required by the cost function from the slave FPGA across the inter camera bus. At 340 fps, a maximum throughput of approximately 275,000 32bit words is achievable, several orders of magnitude greater than the required rate of 1024 samples per iteration.

5.2.4 Sample Set transfers

As mentioned, the GPIF is the only high bandwidth bus connecting the microcontroller and FPGA, and hence must be used for the transfer of sample sets to the FPGA, as well as the back-propagation of the cost and gradients. Each element of the cost function requires a set of sample coordinates, and both the image gradient, and gradient of the distance map must be returned for these sets. Bus utilisation for different parameters are shown in Table 5.2, with the maximum achievable frame rate shown. Clearly the transfer of the sample sets doesn't represent a bottleneck in the system, as the possible frame rates far exceed 340fps.

Set Size	Iterations	Maximum Frame Rate
256	20	9766
512	20	4883
1024	20	2441
256	30	6510
512	30	3255
1024	30	1628
256	40	4883
512	40	2441
1024	40	1221

Table 5.2: Summary of data transfer feasibility over GPIF interface

5.3 FPGA Modules

The final architecture configures the FPGAs as master and slave. The master FPGA controls all data transfers between itself, the slave FPGA, and the host. Each FPGA contains an image pipeline for data processing. The master also includes a softcore processing unit for reading the values from SDRAM, as requested by the host via the GPIF. The master also contains a test harness for experimentation with offline data. It allows an image frame to be fed into the image pipeline using the host computer as a source rather than the CMOS sensor. The slave FPGA is used for video streaming to the host computer via the GPIF. The UART interface is used on both FPGAs to implement a debug interface using the Wishbone bus protocol. Block diagrams for both master and slave FPGAs can be found in Figure 5.1 and Figure 5.2, respectively.

5.3.1 Clock Domains

Because there are several different peripheral devices connected to the FPGA, each with different clock requirements, it is necessary to have different areas of the FPGA synchronised to different clocks. These clocks are generated using phase locked loops, which use phase matching in a closed loop feedback control system [42]. The major clock domains are the SDRAM (128MHz), CMOS sensor (48MHz), and GPIF (100MHz). The clock speed for the softcore processor and image pipeline aren't fixed, however 128MHz is typically used as this corresponds to SDRAM speeds.

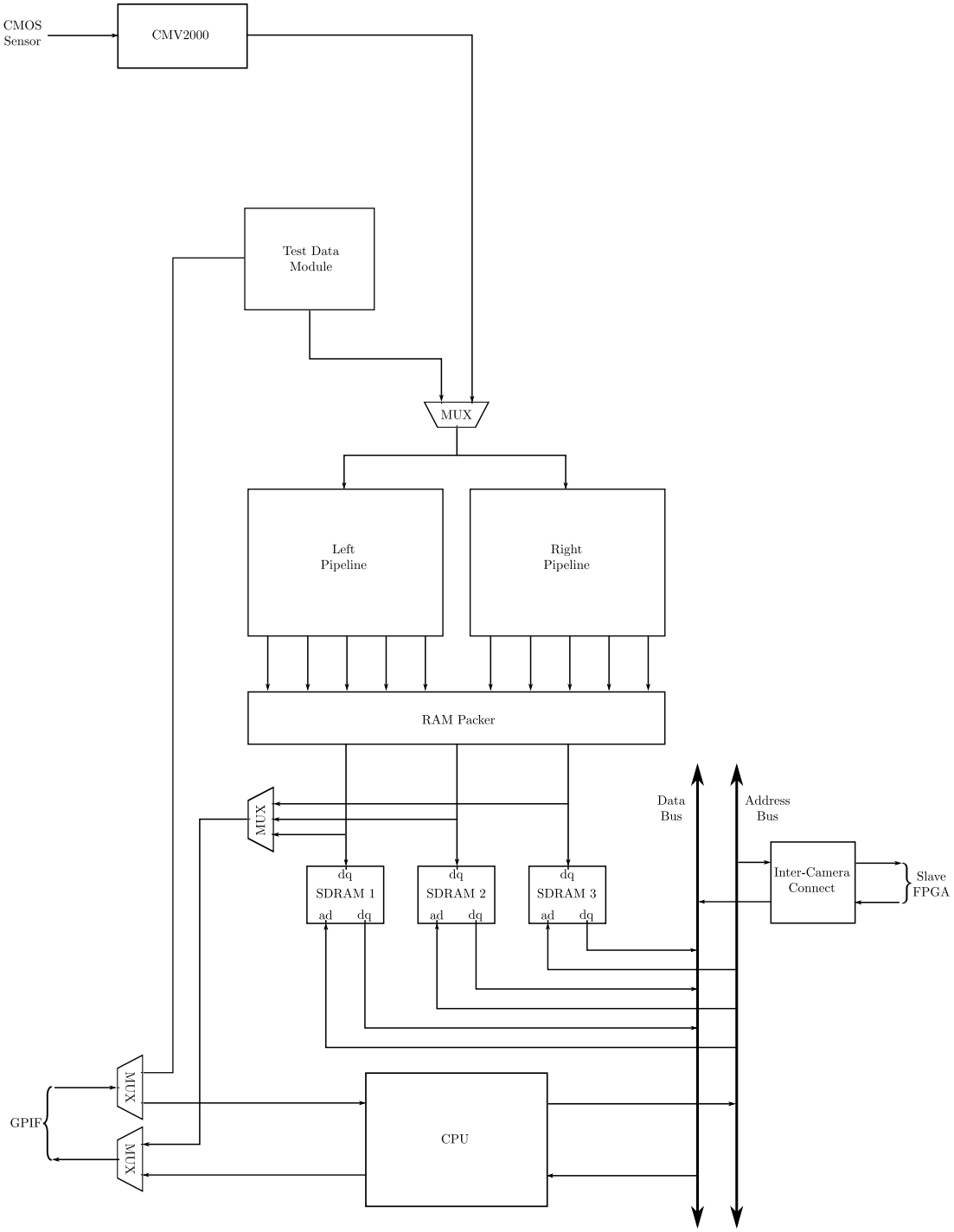

Figure 5.1: Block diagram of the master FPGA

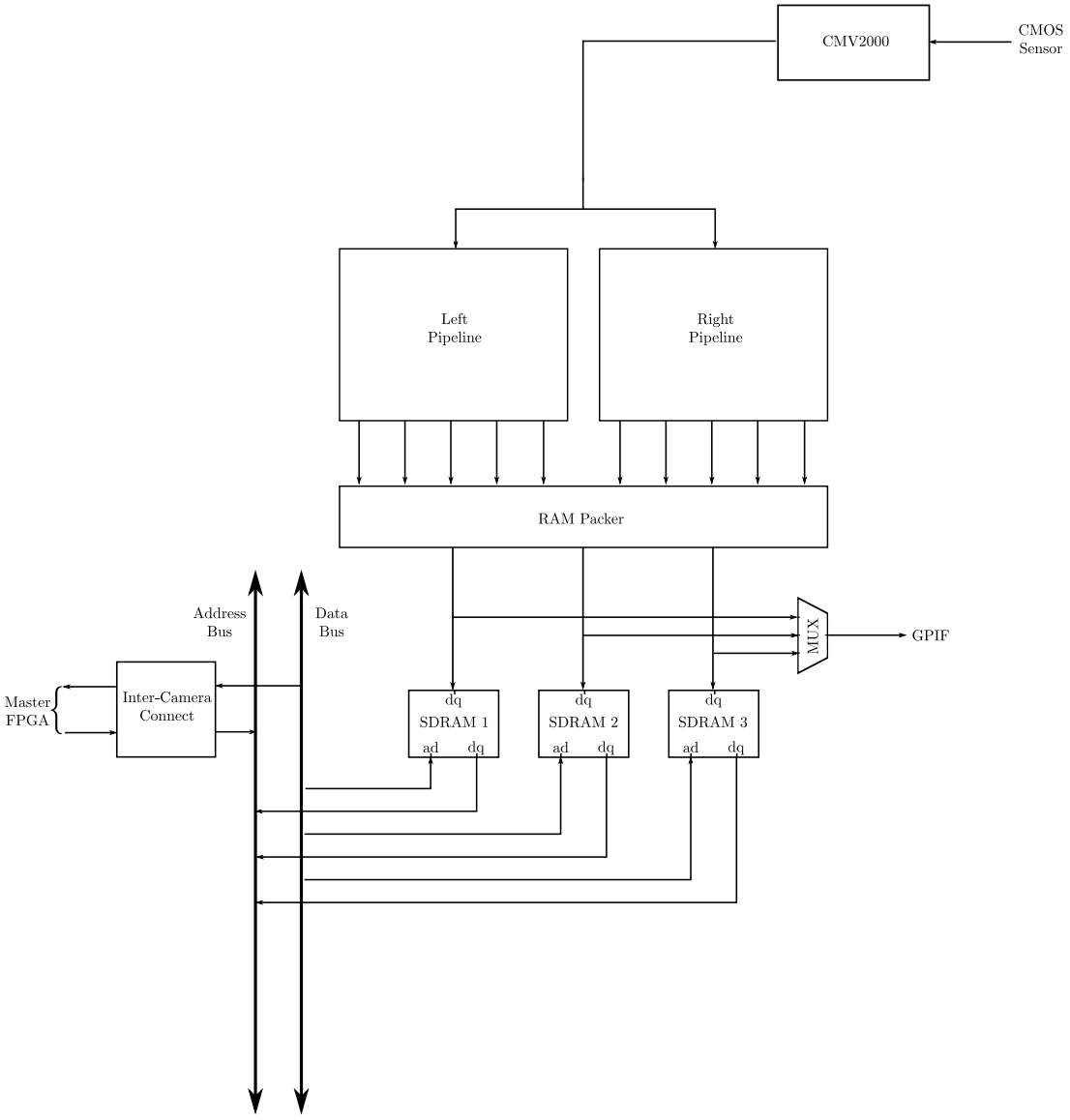

Figure 5.2: Block diagram of the slave FPGA

5.3.2 SDRAM Controller

The SDRAM needs to implement sequential write capabilities, much like a FIFO buffer, while also allowing random access read events. As each SDRAM block is written to for every frame, read and write operations must also be interleaved. The high level state machine controller is shown in Figure 5.3.

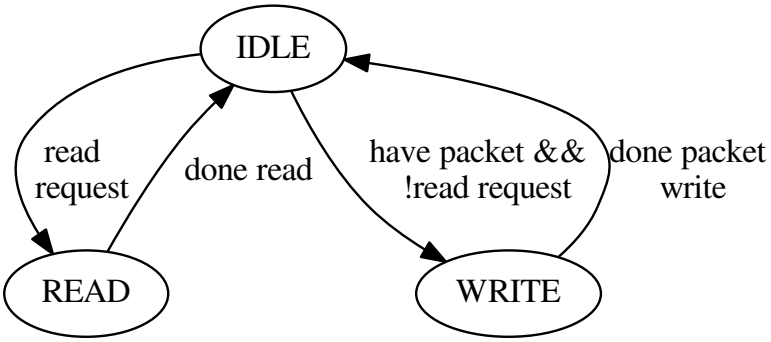

Figure 5.3: High level state machine for controlling read and write operations to the SDRAM

Incoming data is fed into a FIFO where it is buffered until a complete packet is received. The data is then read from the FIFO and written to SDRAM at sequential addresses. By burst writing to sequential addresses, SDRAM latencies are decreased as the delays created by opening and closing different banks are minimised. As the incoming data is already buffered and hence protected from data overflow, read operations are prioritised. Note that read operations are not performed in bursts. As sample sets have a normal distribution, members of the set are not sequential in SDRAM, and burst reading offers no performance increase and incurs latency penalties due to buffering.

Figure 5.4 shows burst writing interrupted by a read operation. Data is written into the input FIFO as a continuous stream. At 1022.9μs a read request is registered and latched into memory along with the address. Once the current packet is written, (1023.1μs), the controller processes the request and presents the data at 1023.4μs.

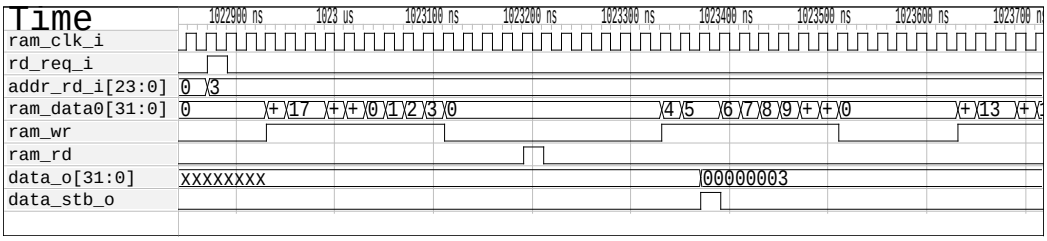

Figure 5.4: SDRAM timing showing interleaved read and write operations

Table 5.3 shows a summary of resources used for a single SDRAM controller. It can be seen resource usage is low, with approximately 3% of the FPGA being used for all three controllers. Note that as all three SDRAM blocks share a common clock only one PLL is needed. The block RAM is used for buffering and for the input FIFO.

Resource	Usage	Percentage
Registers	516	0.45%
Slices	614	0.82%
LUT	631	0.42%
PLL	1	10%
Block RAM	4	1.08%

Table 5.3: Summary of resource usage for a single SDRAM controller as a percentage of available system resources

Unit Test

Test images were written into SDRAM and the stored data compared to the original images. This revealed data was not always being correctly transferred as shown below in Figure 5.5. The majority of values are correct, with approximately 200 incorrect pixels per 640x480 test image. The incorrect values are not the result of missed SDRAM operations, as the correct number of words are read into SDRAM. There appears to be no pattern to the incorrect data, with no relationship between it and the preceding and following data. The error appears to be random bit inversion on the data bus, though not always the same bit.

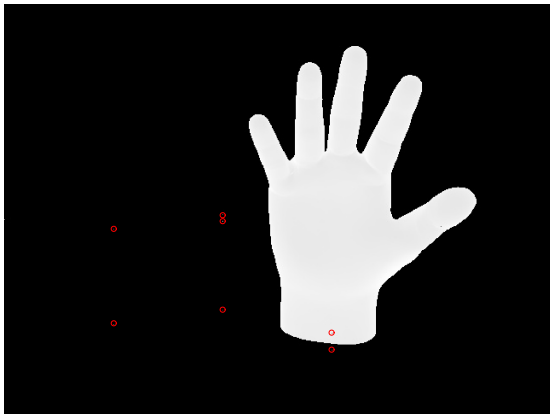

Figure 5.5: Test image with the incorrect regions of approximately 30 pixels highlighted in red

Note that the incorrect regions are vertically aligned, indicating that the issue is neither in the SDRAM or GPIF controllers as these operations take place in bursts of 256 pixels, and instead pointing to a line by line operation. However this is slightly

inexplicable as data transfers during this unit test occurred directly from the GPIF driver to the SDRAM, involving no line operations.

Attempted Solutions

- Several latency cycles were added before and after each write operation in an attempt to increase stability. This reduced errors from several thousand to approximately 200 per frame.
- SDRAM logic was implemented on a separate clock domain so that other logic constraints did not force SDRAM timing violations.

The fact that increasing latencies improved results suggests that problem is a timing issue with the physical SDRAM block, possibly due to set up and hold violations on the data and command buses. This is supported by the fact that read operations take approximately 5 times as many cycles as expected, suggesting that command flags are not being read correctly. Set up and hold times are largely down to the synthesiser, and while required timing constraints can be suggested, it is not always guaranteed that they will be met. At this time it is thought that the error is caused by the synthesiser relaxing these constraints, perhaps to avoid violating requirements in other parts of the design.

5.3.3 FPGA-microcontroller Bus

The FX3 microcontroller was designed with connection of non-standard peripherals in mind, and the architecture includes a 100MHz external bus (GPIF), with a 32bit data bus and additional pins for control. The GPIF is directly connected to the DMA fabric, and controllable through a configurable state machine. This is used to provide a high throughput data bus between the FPGA and the microcontroller, for the request and transfer of data stored in SDRAM. Therefore the bus must be configured to maximise data transfer in both directions. As the bus will also be used for transferring training data and varying sample set sizes and optimisation algorithm iterations, the bus should be flexible enough to allow general purpose packet transfers. A summary of data and control signals are shown in Table 5.4.

Signal	Bits	Description
usb_rd	1	Read strobe exerted by the FPGA master. Must be high for data to be read from the DMA fabric and driven onto the GPIF
usb_wr	1	Write strobe exerted by the FPGA master. Must be high for data to be written to DMA fabric.
usb_a	2	Address bus driven by FPGA master. Address 0 points to DMA Thread 0, address 3 points to DMA Thread 3.
usb_dma_flag	1	Flag driven by microcontroller, indicating availability of DMA thread currently pointed to by address bus.
usb_dat_io	32	Bi-directional data bus driven by either FPGA or micro controller depending on read and write strobe values.

Table 5.4: Control and Data signals of the GPIF data bus

The bus is configured as a bi-directional FIFO, controlled via a master state machine on the FPGA, shown in Figure 5.6. Data on the microcontroller is directly

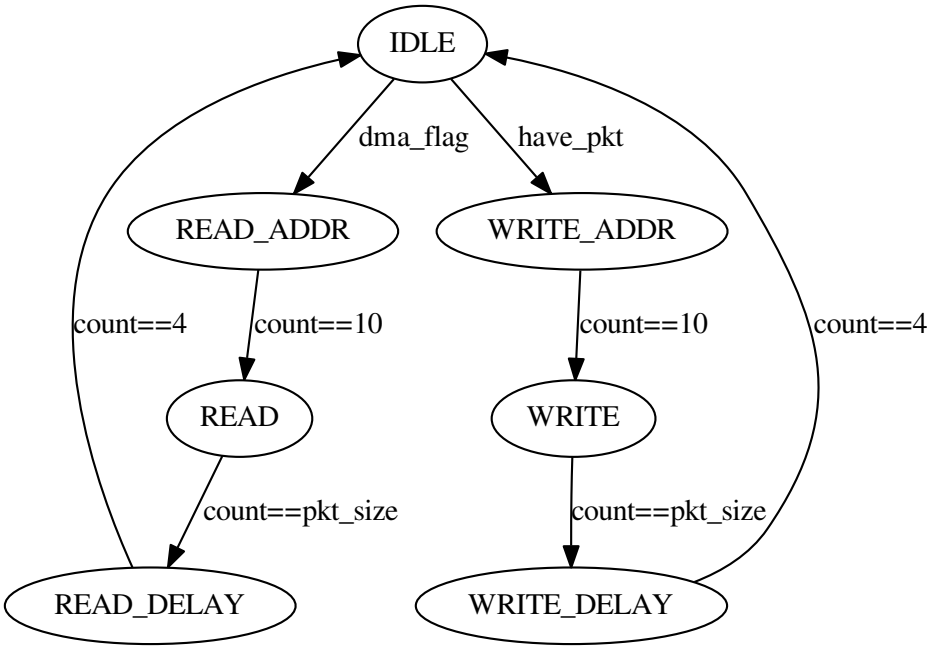

Figure 5.6: Master state machine for controlling packet transfers between the FPGA and microcontroller on the GPIF data bus.

transferred to or from DMA threads, with Thread 3 used for microcontroller to FPGA transfers, and Thread 0 used for FPGA to microcontroller transfers. The thread is selected via a 2bit address bus driven by the master state machine. Control signals include read and write enables are driven by the driven by the master state machine, while the microcontroller drives a flag indicating the availability of a DMA buffer. The master state machine constantly polls Thread 3, and initiates a packet transfer as soon as the microcontroller indicates a buffer is present, which is subsequently written to a FIFO on the FPGA. For FPGA to microcontroller transfers, data is written from the FPGA into an input FIFO to await transfer. As soon as a complete packets worth is present, the state machine switches the address bus to point to Thread 0, and if the DMA flag indicates a buffer is free, initiates the transfer. Read and write operations occur when the state machine is in READ or WRITE state respectively. The states immediately preceding and following these are used for delay to insure that the address bus is stable immediately before and after a transfer. Additionally these states insure that the DMA flag has been updated to the current status of the current thread, a process which is nondeterministic but cannot be neglected. All sig-

nals are updated on the negative edge of the clock, and are subsequently sampled at the positive edge. This insures that set up and hold times are not violated.

Resource utilisation can be found in Table 5.5. Registers and LUT are used for logic implementation such as next state and output logic of the state machine, while block RAM is used for packet FIFOs.

Resource	Usage	Percentage
Registers	422	0.37%
Slices	429	0.58%
LUT	324	0.22%
PLL	1	10%
Block RAM	2	0.54%

Table 5.5: Summary of resource usage for the GPIF controller as a percentage of available system resources

5.3.4 Softcore Processing Unit

A softcore processing unit is used for cost function computations as it provides a flexible platform for testing and parameter tuning. A reduced instruction set is used, which can be found in Table 5.6.

The softcore is modeled on conventional designs, consisting of a finite state machine

Instruction	Description
ADD	Adds the value stored in the memory location pointed to by opcode to the current value in the accumulator.
MULT	Multiples the value stored in the memory location pointed to by opcode to the current value in the accumulator.
DIFF	Absolute difference of the accumulator and value in memory location pointed to by opcode.
RESET	Resets the program counter to zero.
STOP	Suspends the softcore by not entering next fetch state. Can only be exited through external reset.
JUMP	Changes the program counter to the value stored in opcode.
JUMPIF	Changes the program counter to the value stored in opcode if the accumulator is less than or equal to zero.
LOAD	Loads value into the accumulator from memory location pointed to by opcode.
SAVE	Saves current value of accumulator into memory location pointed to by opcode.
READ	Loads value into accumulator from SDRAM addressed by opcode.
GPIFREAD	Loads value into accumulator from SDRAM addressed by GPIF.

Table 5.6: Reduced instruction set for softcore processing unit

where state transitions are determined by instructions stored in program memory. Control logic and the arithmetic logic unit (ALU) are implemented in combinatorial logic, the information register (IR), memory address register (MAR), and program counter (PC) are implemented in sequential logic, while program code and data are stored in embedded block RAM. Program code and data are stored in separate memory blocks, as per a Harvard architecture. A block diagram of the architecture can be found in Figure 5.7.

A three stage architecture is used, with each operation going through fetch, decode, and execute cycles, which typically are executed as follows.

Fetch

The program counter is loaded into the memory address register which is driven onto the address bus. The next instruction to be executed is read from the program memory and loaded into the information register. The program counter is then

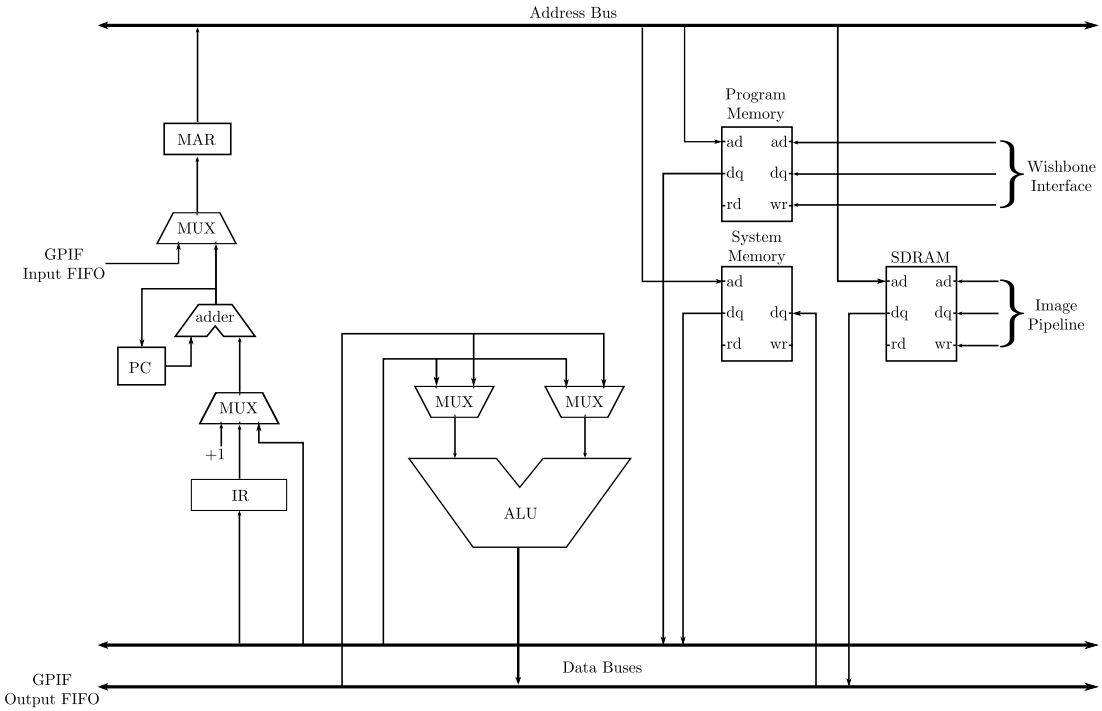

Figure 5.7: Block diagram of the softcore processing unit

incremented by one. Note the memory address register is necessary as it latches the current address which allows the program counter to be updated in the same cycle.

Decode

The instruction present in the information register is decoded and the appropriate control signals set. For read operations the address in the instruction set is driven onto the address bus as for the program counter previously. The data terms are then presented to the inputs of the arithmetic logic unit.

Execute

As the arithmetic logic unit is combinatorial, the result is already present at the output of the ALU and is simply latched into sequential logic.

Figure 5.8 shows an example of soft core operations. Values are read from RAM (PC: 0x06 and 0x09), summed and multiplied (PC: 0x07 and 0x0A), and the result saved back into RAM (PC: 0x08 and 0x0B). Note that each operation has a duration of at least 3 clock cycles to account for the fetch, decode and execute cycles.

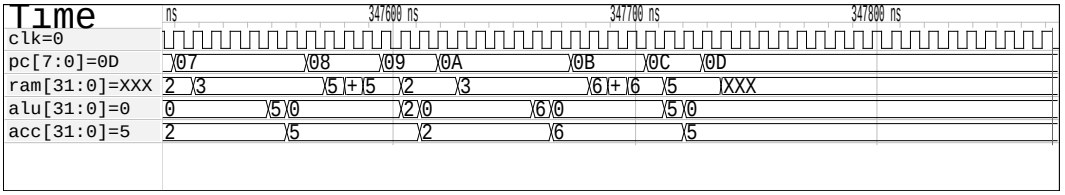

Figure 5.8: Simulation showing softcore functions. Values are read from RAM, and saved back into RAM after multiplication and addition operations.

Resource utilisation can be found in Table 5.7. Registers are used for temporarily latching data such as the IR, ACC, and PC, while LUTs implement the control signals. Distributed (embedded) RAM is used for the program and system memories. The ALU is formed using DSP slices to create separate pipes for each operation. Selection of these is controlled by the current opcode in the IR.

Resource	Usage	Percentage
Registers	354	0.31%
Slices	452	0.61%
LUT	619	0.42%
Dist RAM	24	0.02%
DSP Multipliers	6	0.94%

Table 5.7: Summary of resource usage for softcore processor as a percentage of available resources

5.3.5 Psuedo-Random Number Generator

The filling cost function requires a random sample of pixel coordinates taken from inside the real silhouette. Pseudo number generators can be readily implemented in the form of a Linear Feedback Shift Register (LFSR). While LFSRs are deterministic, they can give the appearance of randomness with cycle lengths of $2^n - 1$ for maximal length LFSRs of length n [45]. A LFSR is maximal length if:

- If $f(x)$ is a polynomial with coefficients in the Galois field of order 2 and $f(0) = 1$, i.e.

$$f(x) = \sum_{i=o}^{n} c_i x_i, f(0) = 1, c_i \in GF(2) \tag{5.1}$$

  then the period of $f(x)$ is the smallest T for which $x^T + 1$ is divisible by $f(x)$.

- An irreducible polynomial of degree n has a period which divides $2^n + 1$
- $f(x)$ is a primitive polynomial if it is irreducible of degree n and has a period of $2^n - 1$.

By using a 10bit LFSR with the characteristic polynomial

$$f(x) = x^10 + x^7 + 1 \tag{5.2}$$

it can be seen that it will be maximal length, as $x^{1023} + 1$ is the smallest such number with $f(x)$ as a factor. Similarly $f(x)$ is irreducible with period $2^{10} - 1$.

However to avoid cyclic sampling, the LFSR must be provided with a random seed at an interval less than the maximal length period. This is accomplished by using spare bandwidth on the GPIF interface for host to FPGA transfers, and occurs for every transfer. Figure 5.9 below shows a frequency histogram for a 10bit LFSR with an even distribution, and the resulting pixel coordinate sampling.

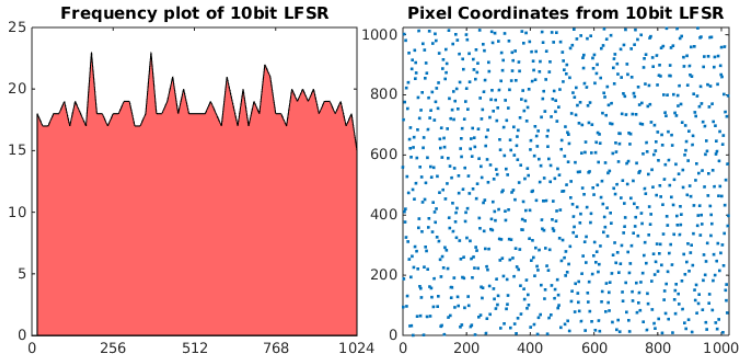

Figure 5.9: Plot showing frequency histogram of an LFSR (left) and pixel coordinates generated with a 10bit LFSR (right)

5.3.6 Image Processing Pipeline

The image processing pipelines receive the output from the CMOS sensors, perform the operations required for the cost function, and store the output data into SDRAM. Each pipeline has three major components: a silhouette extraction module, a distance transform module, both required for the filling and silhouette cost functions, and a Sobel filter module for the back propagation of gradients to the optimisation algorithm. The three components can be seen below in Figure 5.10.

Figure 5.10: Results for the silhouette, distance transform, and image gradient modules from the image processing pipeline.

5.3.6.1 Silhouette Extraction

For real images, the silhouette is extracted via thresholding after a colour space transformation as developed in [68]. By using principal component analysis a rotation matrix can be constructed which will orientate the region of the colourspace containing skin tone with the axes. The required online computations then become

$$I_i = R_{i,1}H + R_{i,2}S + R_{i,3}V \tag{5.3}$$

where R is the rotation matrix. For the synthetic images used in the subsequent experiments, simply thresholding the YUV colourspace is sufficient, shown below in Equation 5.4, with I_i denoting the intensity of the i^{th} channel.

$$S = \begin{cases} 0 & I_i \leq L_i \\ 1 & L_i \geq I_i \leq H_i \\ 0 & I_i \geq H_i \end{cases} \tag{5.4}$$

As the extraction is simply a threshold, the operation requires no buffering and very few resources with the threshold expressible as a LUT. In reality the synthesizer incorporates the logic into other modules in the pipeline.

5.3.6.2 Distance Map

The distance map computes the minimum distance from the pixel in question to the silhouette. The transform originally used by Chik [18] was an Euclidean approximation algorithm developed by Borgefors [10], intended for high speed CPU implementations. However the parallel nature of the FPGA lends itself to computing the true Euclidean distance within a mask as below in Equation 5.5.

$$D = \sum_{v=-5}^{5} \sum_{u=-5}^{5} \min_{I(u,v)=1} (u^2 + v^2) \tag{5.5}$$

As the distance at each location inside the mask can be precomputed and stored in LUTs, the problem reduces to a series of comparators. However there is a resource penalty in that each line inside the mask must be buffered in memory. Resource uses for the distance transform can be found below in Table 5.8.

Resource	Usage	Percentage
Registers	248	0.21%
Slices	195	0.26%
LUT	230	0.15%
Block RAM	1	0.26%

Table 5.8: Summary of resource usage for distance map using 640x480 resolution images

5.3.6.3 Image Gradients

As the intensity function and distance transform function are not differentiable, the gradients for these must be determined empirically. The gradients are obtained by convolving the images with a Sobel filter kernel, as below in Equation 5.6. Note that there is a Gaussian component for noise removal.

$$k = \begin{pmatrix} 137 & 618 & 1708 & 618 & 137 \\ 308 & 1384 & 2280 & 1384 & 308 \\ 0 & 0 & 0 & 0 & 0 \\ -308 & -1384 & -2280 & -1384 & -308 \\ -137 & -618 & -1708 & -618 & -137 \end{pmatrix} \tag{5.6}$$

As the gradients are to be stored as 8bit signed integers with range $[-127, 127]$, the result of the convolution is divided by 2^{14} to normalise to this range, accomplished via bitshifting for a resource efficient implementation. A complete resource list can be found in Table 5.9.

Resource	Usage	Percentage
Registers	2614	2.23%
Slices	2056	2.75%
LUT	3517	2.35%
Block RAM	16	4.3%
DSP Multipliers	28	8.8%

Table 5.9: Summary of resource usage for Sobel filter using 640x480 resolution images

5.3.6.4 Unbiased and a Uniformly Bounded Variance

In order for the tracker to converge to the unique global minimum under a stochastic sampling scheme, the gradient estimate must be unbiased and the variance uniformly bounded, such that:

$$E_{\phi}[\nabla C_{x^*}(x_t, \Phi)] = \nabla C_{x^*}(x_t) \tag{5.7}$$

$$\sup\{Var(\nabla C_{x^*}(x_t, \Phi)) : x_t \in \Re^k\} < \infty \tag{5.8}$$

where $C_{x^*}(x_t, \Phi)$ is the approximate cost under random sampling conditions Φ. As the Sobel kernel is symmetric, Equation 5.7 holds for all parts of the cost function. The YUV intensity channels required by the photoconsistency cost function are uniformly bounded in the range $[0, 255]$ and the distance transform is uniformly bounded by the mask size. Therefore Equation 5.8 also holds.

5.3.7 Design Summary

The final design is capable of processing the images at full frame rate after downsampling to 640x480 pixels. This results in a data throughput of nearly 10Gbit/s through the image processing pipeline. Further specifications of the architecture can be found in Table 5.10. A summary of total resource usage can be found in Table 5.11.

Specification	Value
Image Size	640x480 pixels
Frame Rate	338 fps
Pixel Rate	103.83 Mp/s
Pipeline Data Rate	9.97 Gbit/s
SDRAM bandwidth	96bits
Pipeline clock	128MHz
SDRAM clock	128 MHz
CPU clock	128 MHz
GPIF clock	80 MHz

Table 5.10: Summary of system specifications

Resource	SDRAM	GPIF	CPU	Distance Transform	Sobel Filter	Total Usage
Registers	1548	422	354	248	10456	11.42 %
Slices	1842	429	452	195	8224	14.95 %
LUT	1893	324	619	230	14068	11.50 %
Dist RAM	12	2	24	1	64	27.69 %
DSP Mult	0		6	0	112	36.88 %
PLL	1	1	0	0	0	20.00 %

Table 5.11: Summary of resource usage for the control architecture as a percentage of available resources

5.4 CPU-based Modules

It was originally intended for the optimisation algorithm and hand model to be implemented on the microcontroller, taking advantage of the ARM floating point library. However as this is not the focus of the report, the decision was made to implement these modules on the host computer. All modules were implemented in C. Modules widely used double precision floating point values, including for example the linear blending coefficients in the hand model and the parameters used in the optimisation algorithm.

Chapter 6

Initial Results

The following sections present the validation of the FPGA tracker against Chik's implementation. The experimental setup is discussed in Section 6.1. Section 6.2 compares the original tracker with the FPGA simulation, and discusses the error of each and its causes. The accuracy of these trackers are judged relatively, as well as absolutely using the magnitude of the cost at each frame and visual inspection of key frames. Section 6.3 discusses tracking results using the physical FPGA pipeline.

6.1 Experimental Setup

The initial experiments were conducted using three different tracker implementations on the same image sequence. The images used are synthetic, created in OpenGl.

6.1.1 Image Sequences

The synthetic sequences were generated using the hand model in OpenGl. The same skeleton is used as in the tracker, however the mesh used is a simplified one with 8,464 vertices as opposed to the trackers 45,181. The projections between the image plane, camera coordinates, and world coordinates remain the same. The two cameras are orientated approximately 30° apart, with the left-hand camera situated at the origin of the world coordinates. A single light source is situated above and to the left of the first camera to provide gradient across the hand. The background is nonuniform to provide texture. Examples of the synthetic images can be found in Figure 6.1.

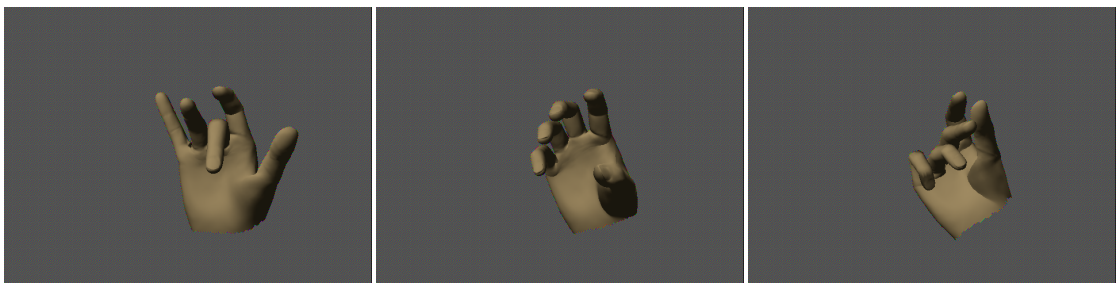

Figure 6.1: Sample synthetic images depicting various hand poses

6.1.2 Tracker Implementations

Three different implementations were used in initial experiments: the original tracker as in [18], a tracker with a simulation of the FPGA image processing pipeline implemented on the host computer, and a tracker using the physical FPGA based image processing pipeline.

6.1.3 Pose Sequences

Three image sequences are used in the following experiments, each of which represent a different pose. All start and finish at a neutral position, with an open palm facing the first camera. The poses are chosen as they are ones common in HCI, or are poses which are predicted to be difficult for the tracker to resolve.

Finger Flexion

This sequence simulates the sequential flexion of each finger (and thumb) through its natural range of motion. Finger flexion is an important motion to track as the majority of gestures used to control touch devices use variants of these motions, such as tap, swipe and pinch.

Grip and Twist

The grip and twist sequence simulates gripping a virtual dial and turning it, for example adjusting values in photo editing software such as Photoshop. Alternatively the gesture can be viewed as picking up an object and moving it, as would be the case in a virtual reality setting.

Palm Rotation

The third sequence consists of an open palm pose, which starts facing the first camera, then rotating through 180°, so that the back of the hand is presented, and back. This is a difficult pose for the tracker as at 90° only the edge of the hand is visible, with the majority occluded behind it.

6.2 Experiments

All experiments are conducted on the same image set with the same starting pose. The weighting of the cost components differ slightly between implementations with the values used in each shown in Table 6.1 and Table 6.2. All experiments used the SMD optimisation algorithm at 20 iterations per frame and 1024 samples distributed randomly across the model surface. A basic deceleration motion predictor is used throughout. As a measure of accuracy the absolute value of the cost is used, as introduced in Section 3.3. Note that the value is dependent on the weighting of the individual components, and therefore the trend is of more interest.

Cost Component	Weight
Photoconsistency	25
Filling	0.9
Silhouette	1.49

Table 6.1: Weights for the cost components used in the original tracker

Cost Component	Weight
Photoconsistency	20
Filling	0.93
Silhouette	0.49

Table 6.2: Weights for the cost components used in the simulation tracker

6.2.1 Finger Flexion

Both implementations produced similar results, as seen in Figure 6.2 with the magnitude and behavior of the cost being similar for both implementations across all frames. The FPGA simulation exhibits behavior with slightly greater instability, seen by the greater fluctuations in cost. The reasons for this are discussed below.

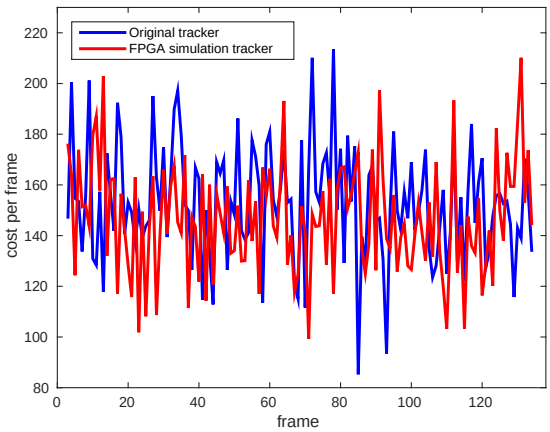

Figure 6.2: Cost per frame for finger flexion sequence. Original tracker results are in blue, FPGA simulation results in red. (best seen in colour)

The original tracker performs well in the sequence as exhibited in the top row of Figure 6.3. The tracker does struggle however with fast finger movement, as the motion predictor is not aggressive enough to force the model into the region of the real finger movement, as exhibited in frame 120 (top).

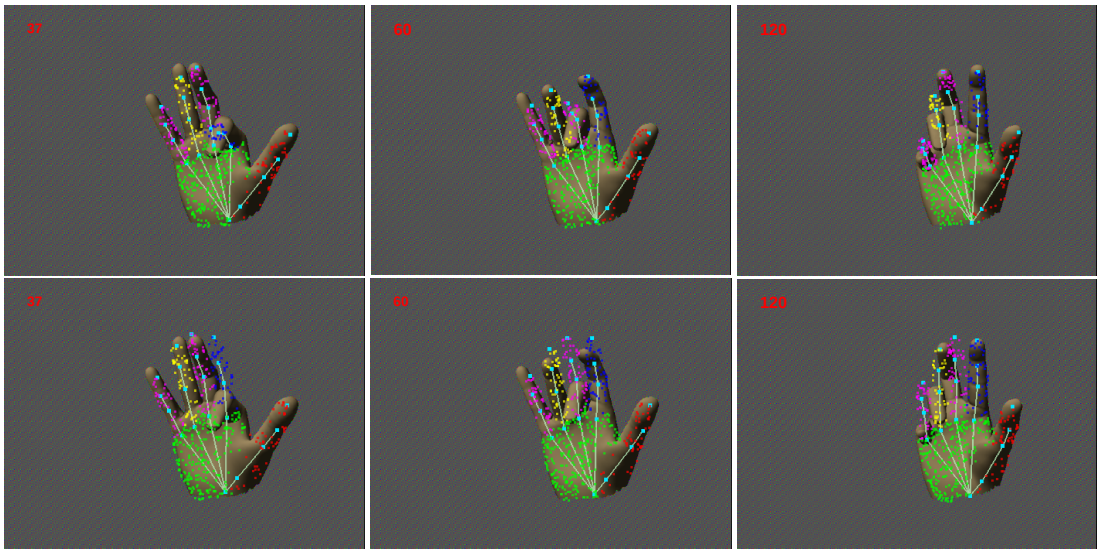

Figure 6.3: A selection of frames from the finger flexion sequence using SMD optimisation. The top row contains frames from the original tracker, while the bottom row contains the same frames for the FPGA simulation

Additionally, movement beyond the PIP and DIP joints prove difficult to track as seen in frames 37 and 60 (top). The FPGA simulation has similar problems, however to a much greater extent. As the silhouette function saturates at 5 pixels, the range of movement between frames that will be registered is small, and most finger movements will be too great. This leads to the silhouette function not forcing the model to bend in the correct direction. Because of this the projected points no longer lie on the model surface, effectively meaning that the photoconsistency function will not track in the correct direction. This is compounded by the motion predictor hindering tracking, as the prediction for the current frame will be based on the movement of the last frame, which is incorrect. This results in the type of behavior seen in frames 37 and 60 (bottom), where there is very little finger flexion. Once this has occurred it is very hard for the tracker to recover (in a local sense) as the rest of the hand remains stationary, and is effectively independent of finger flexion. A side effect of this behavior is that a slight change of orientation can bring neighboring fingers closer, resulting in the incorrect tracking onto this neighbor. This can be seen in frame 120 (bottom), where the ring finger has tracked to the middle finger.

6.2.2 Grip and Twist

Again both implementations produce similar results, with the original tracker showing spikes in cost as seen in Figure 6.4. A selection of frames for the original tracker and the FPGA simulation can be found in Figures 6.5 and 6.6 respectively.

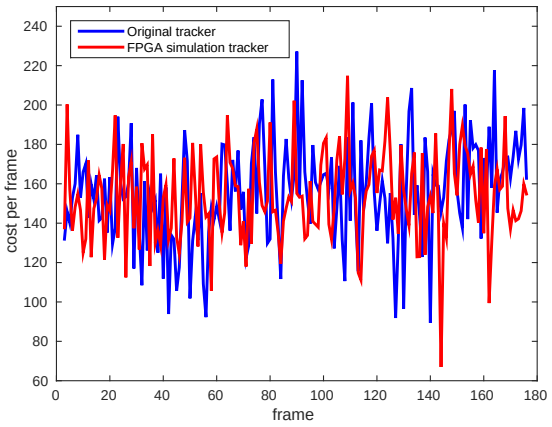

Figure 6.4: Cost per frame for grip and twist sequence. Original tracker results are in blue, FPGA simulation results in red. (best seen in colour)

The original tracker tracks the grip pose reasonably well, but again struggles with finger flexion at the DIP and PIP joints, resulting in the estimated pose to be positioned lower than the true pose as seen in frames 66, 70, 72 and 73 of Figure 6.5. The first portion of the twist motion is tracked well, with the estimated pose matching the true pose closely up until frame 66. However at this point the motion changes direction, causing large accelerations.

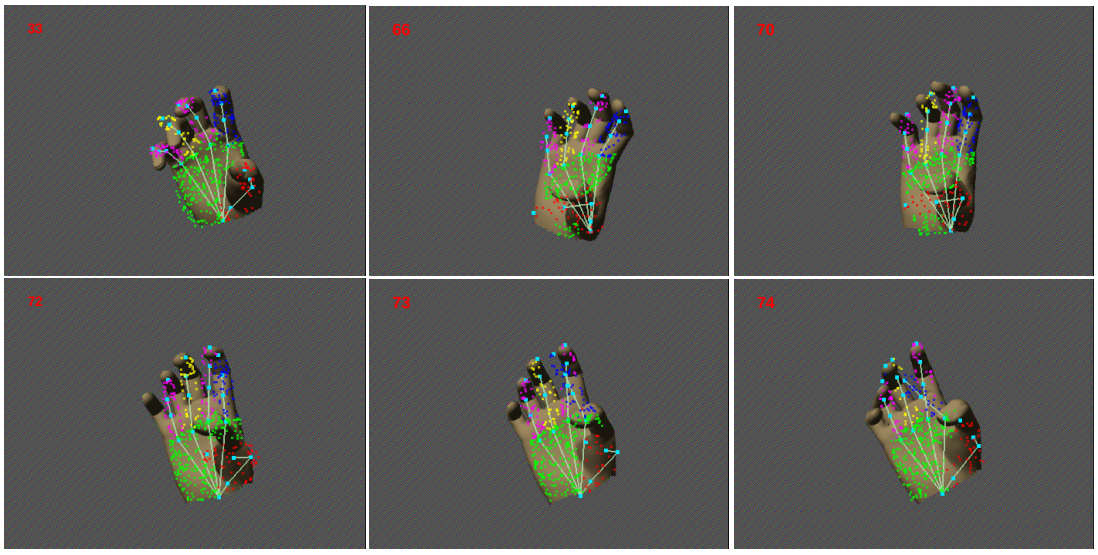

Figure 6.5: A selection of frames from the grip and twist sequence using SMD optimisation. All frames are from the original tracker

As the motion predictor is that of constant deceleration, the predictor lags behind the true pose, resulting in fingers tracking to the adjacent finger in the opposite direction of travel, as seen in frames 72, 73 and 74 of Figure 6.5.

The FPGA simulation tracks the grip pose poorly in a similar fashion to the finger flexion sequence, and again is caused by fast finger movement and the saturation of the silhouette function. While the first portion of the twist sequence is again tracked

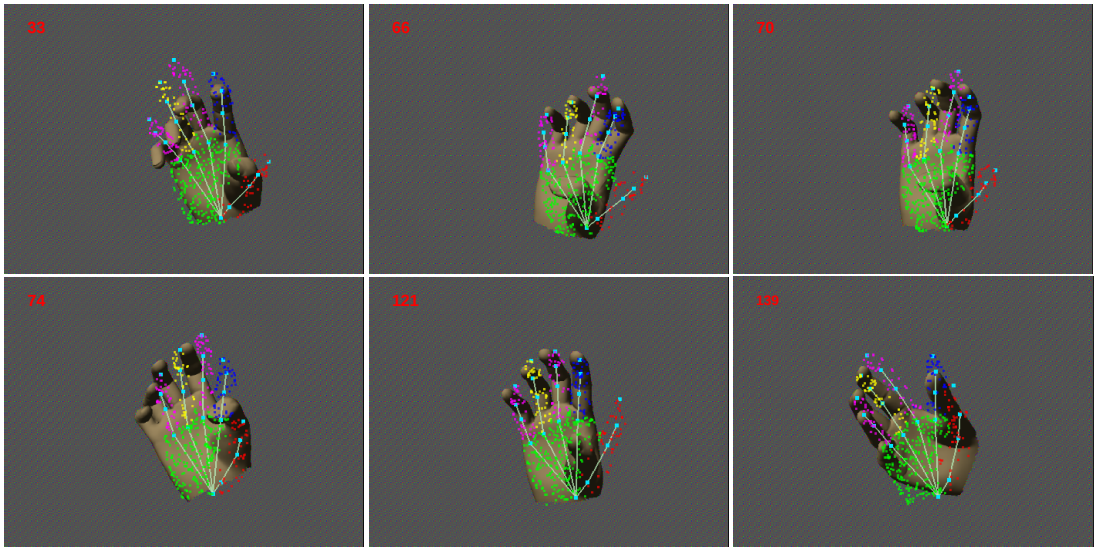

Figure 6.6: A selection of frames from the grip and twist sequence using SMD optimisation. All frames are from the FPGA simulation tracker

approximately, the return motion causes the same consequences as before. However they are exacerbated by the saturation of the silhouette function, as motion which causes the true pose to lie interlaced with the estimated pose, as in frame 74 of Figure 6.6, receive no penalty, and as the sample points lie off the surface of the hand, there are no gradients to force the estimated pose towards the true pose. This results in poor tracking, as in frames 74, 121 and 139 of Figure 6.6, where the estimated pose remains in approximately the same pose, while the true pose moves underneath it. Note also frames 66 and 121 of Figure 6.6, where the saturation of the silhouette function allows the thumb to remain off the surface of the hand.

6.2.3 Palm Rotation

As predicted, the palm rotation sequence proved difficult for both implementations to track. The cost per frame for each is shown in Figure 6.7. Up to approximately frame 20 both implementations produce similar levels of cost, however beyond this they diverge, with the original tracker first exhibiting a large peak in cost at frame 30, and then a constantly higher cost from frame 60. Select frames for the sequence are in Figure 6.8 for the original tracker, and Figure 6.9 for the FPGA simulation.

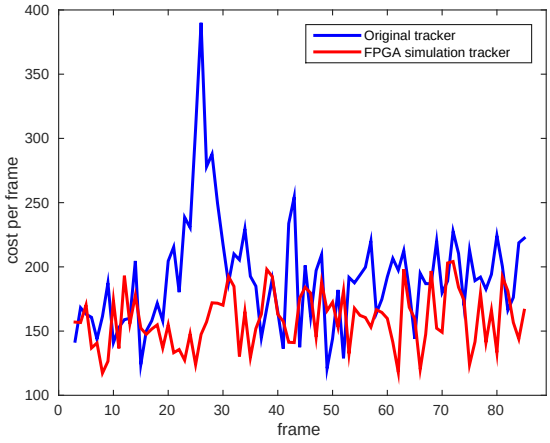

Figure 6.7: Cost per frame for palm rotation sequence. Original tracker results are in blue, FPGA simulation results in red. (best seen in colour)

From Figure 6.8 it can be seen that up until frame 22, the estimated pose matches the true pose approximately. However as the hand continues to rotate, the side of the palm is presented, and the thumb becomes occluded as in frame 24 of Figure 6.8, creating the large spike in error. Once this has occurred, the tracker is no longer able to follow the rotation, resulting in the mirror image seen in frame 49 of Figure 6.8, with the thumb tracking to the little finger and vice versa. This is due to the hand being approximately symmetrical around the axis of the middle finger. Once the hand starts to rotate back, this cause the model thumb to continue to be assigned to the little finger, as seen in frame 59 of Figure 6.8. A consequence of this is that once the palm is presented again, the model thumb is not in a position to track the real thumb as seen in frame 65 of Figure 6.8. This is due to it lying inside the model silhouette, so does not occur a penalty for the silhouette function, and as it is now positioned in the palm, the image gradients are not sufficient. However given enough iterations, it does recover back to the neutral pose.

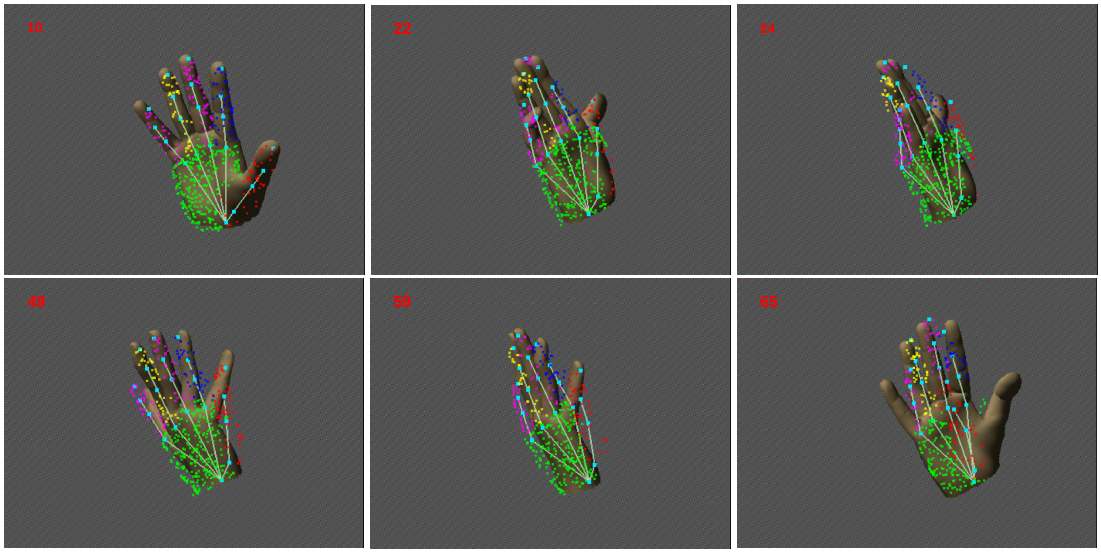

Figure 6.8: A selection of frames from the palm rotation sequence using SMD optimisation. All frames are from the original tracker

The FPGA simulation exhibits similar behavior up until frame 22, as seen in Figure 6.9. However once the frame is occluded in frame 24 of Figure 6.9, the saturation of the silhouette function means that the estimated pose is not forced into a mirror image of the true pose as effectively, seen in frames 24 and 49 of Figure 6.9. Because of this once the return rotation occurs, the model thumb is not assigned to the little finger as in frame 59 of Figure 6.9.

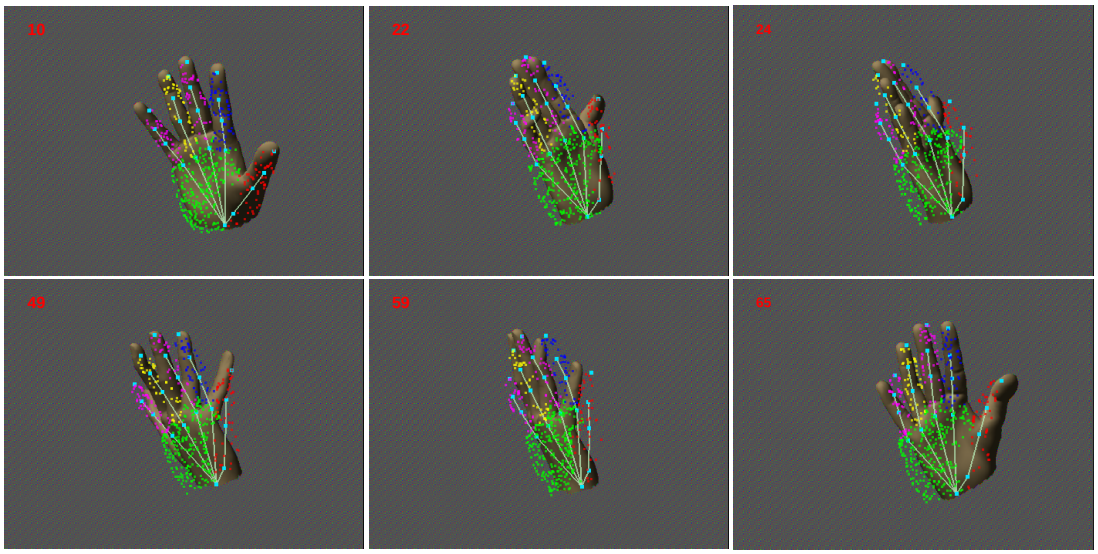

Figure 6.9: A selection of frames from the palm rotation sequence using SMD optimisation. All frames are from the FPGA simulation tracker

This has the consequence of the model being in a better position to recover once the palm is presented again, resulting in a faster convergence to the true pose as seen in frame 65 of Figure 6.9. Therefore while the FPGA implementation appears to have greater difficulty tracking, it also offers faster recovery, however only in scenarios where the true pose returns to the point at which tracking was lost.

6.3 Tracking with an FPGA-implemented Image Pipeline

This section demonstrates the control architecture implemented on the FPGA successfully supplying the optimisation algorithm with the requested values. However due to the SDRAM issues mentioned previously the tracker was not evaluated against the pose sequences. Instead it was evaluated visually on a frame by frame basis. A typical pose optimisation can be seen in Figure 6.10.

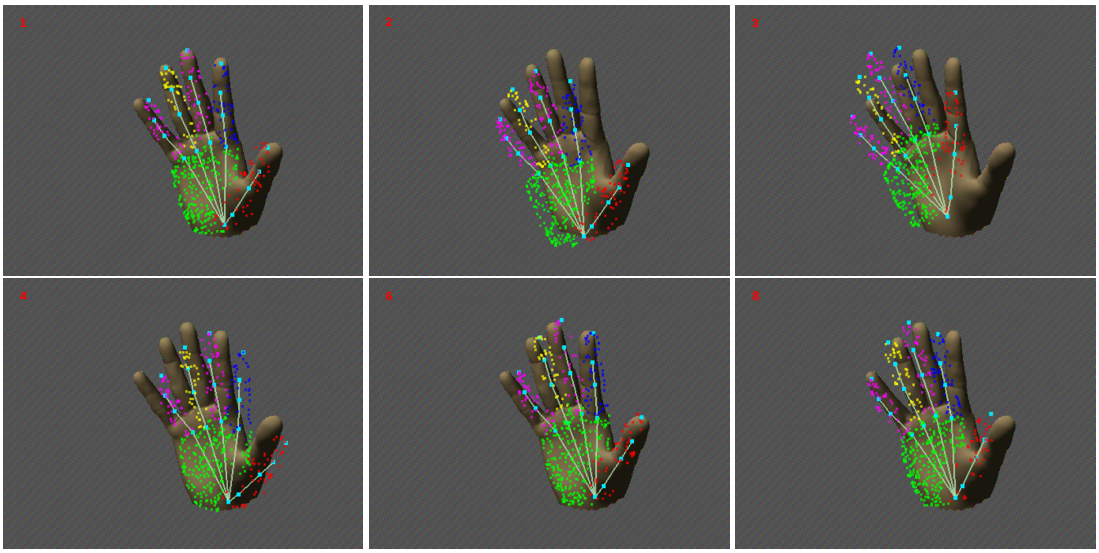

Figure 6.10: Frames from FPGA tracker showing predicted pose at iterations 1, 2, 3, 4, 6 and 8 of the SMD optimisation algorithm

It can be seen that while the estimated pose appears to be centered around the true pose, it typically overshoots as in frames 1 to 2 and 6 to 8 of Figure 6.10. It also appears to lack the fine scale adjustment required to achieve a stable convergence. This can be explained by the resolution of the gradient estimate from the Sobel filter. Typically gradients used by the photoconsistency cost function are in the range [-5 5], with fine scale pose adjustments using even smaller gradients, in the range [-1 1]. However the Sobel filter is limited to signed 8bits, giving the range of [-127 127] with a resolution of 1. Therefore the small gradients required for accurate tracking are not processed by the Sobel. As the gradients are typically well inside the range used by the Sobel, the assumption could be made that the more significant bits are not used, and the lower bits used instead. While this potentially could result in data overflow resulting in incorrect gradients, it is probably a safe assumption providing

edge sampling is not used. This would effectively increase the resolution, allowing for fine scale pose adjustment.

Additionally this limitation when combined with the silhouette limitations can cause unstable effects as in Figure 6.11. Occasionally when the tracker overshoots the true pose, the gradients at the next iteration no longer point in the correct direction. This can be seen clearly in frames 26 and 38 of Figure 6.11. The photoconsistency function is forcing the estimated pose to contort itself onto regions of similar intensity on the palm, aided by the silhouette function on the right of the palm, forcing the ring finger to the left. The silhouette function on the index finger, thumb, and base of palm is saturated, and does not provide the necessary gradients required to recover. A larger silhouette mask should improve the stability by limiting the saturation.

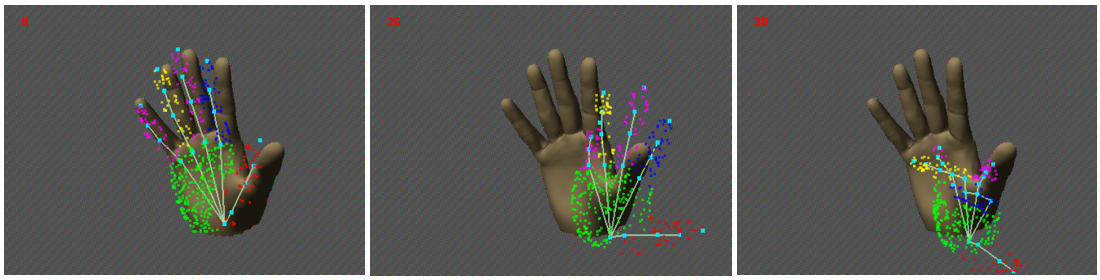

Figure 6.11: Frames from tracker showing predicted pose at iterations 8, 26 and 38 of the SMD optimisation algorithm

6.4 System Validation

The experiments presented in Section 6.2 show that the simulated tracker is capable of tracking hand poses albeit lacking fine scale accuracy due to some of the limitations imposed by the saturation of the distance transform. The following chapter will attempt to address this by assessing the tracker response at higher frame rates, which is hypothesized to decrease the number of sample points experiencing saturation. Section 6.3 shows that poor resolution of the image gradients (limited by SDRAM bandwidth) leads to a lack of fine scale pose adjustment and instability. Using the assumption that gradients are small will result in an increase in resolution.

Chapter 7

Frame Rate Experiments

The primary reason for using a high speed camera unit was the hypothesis that increasing frame rate could lead to greater accuracy and the ability to use simpler optimisation algorithms, leading to faster implementations. The remainder of the chapter discusses the effect frame rate has on tracker accuracy, as well as the implications it has on choice of optimisation algorithm and the resulting speed of such a tracker. The poses used are the same as those used in the previous chapter, with the frame rate increased to 240fps by interpolating between the original poses.

7.1 Tracking Accuracy

Initial results showed improvement over those in Chapter 6, however finger flexion tracking was still unsatisfactory. This is due to the saturation of the silhouette function combined with the relative weightings of the cost function components. Because the photoconsistency function has a much greater weight than the silhouette function, it dominates the optimisation algorithm. The exception to this is when the estimated pose is sufficiently different from the true pose that the distance transform is large enough to begin to influence the cost, resulting in the accuracy seen in the original tracker. However as the FPGA distance transform saturates, the distance is firstly not great enough to influence cost, and secondly if the values were great enough, the saturation would result in zero gradient. To remedy this the relative weights are adjusted so that the silhouette function plays a greater role in the overall cost. The combination of the frame rate and adjusted weighting means results in significantly greater tracking accuracy as seen in Figure 7.1, with frames 264, 296, 480 and 960 showing visual accuracy on par or greater than the original tracker. The side effect of the adjusted weighting can be seen in frame 560, where instead of flexion at the DIP joint the estimated pose has translated downwards. Additionally partial occlusion is still problematic as seen in frame 1112.

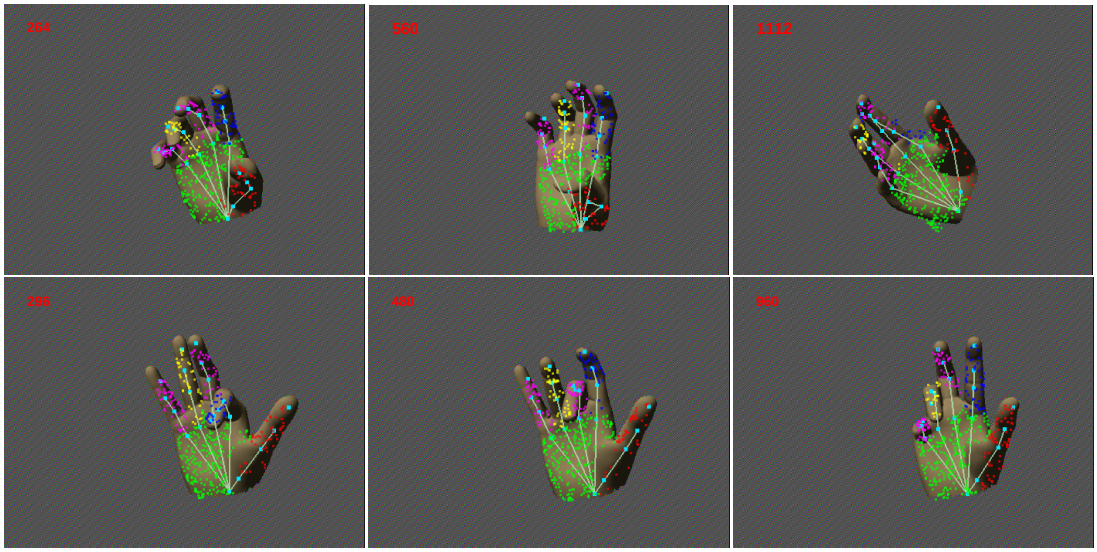

Figure 7.1: A selection of frames from the twist (top) and finger flexion (bottom) sequences using SMD optimisation.

However increasing frame rate appears to have little effect for palm rotations as seen in Figure 7.2. The tracker still struggles to recognise that the pose has reversed, tracking successfully until the palm is parallel to the camera axis, but unable to track past parallel. This is to be expected as when the palm is parallel, there is a large amount of occlusion, an artifact which is not effected by frame rate. To successfully track past parallel a more aggressive motion prediction scheme is needed.

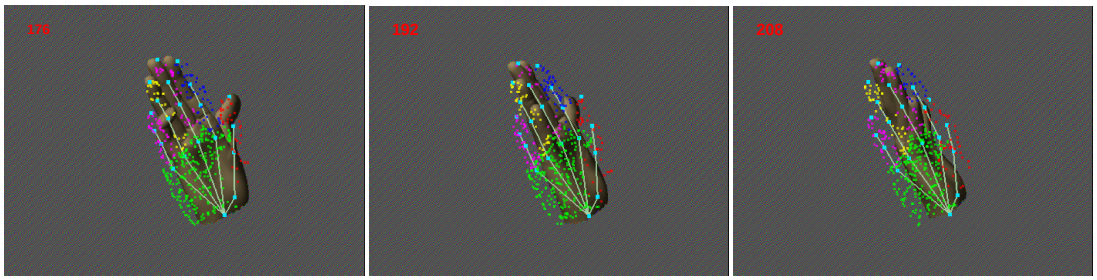

Figure 7.2: A selection of frames from the palm rotation sequence

7.2 Experiments

When conducting experiments at 30fps, Chik [18] found that increasing both the size of the sample set and the number of allowable iterations resulted in a decrease in cost. There are two possible explanations for this, the first being that the difference between the true pose and estimated pose is too great, forcing the number of steps needed (as the step size is bounded) to exceed the number of iterations. The second is that the sample set does not contain sufficient information for convergence to the true pose (i.e. the sample set is not sufficiently dense). By increasing frame rate it is hoped that these effects are mitigated. Supplementary frames for the results presented can be found in Appendix A.

7.2.1 Optimisation Iterations

Figure 7.3 shows the cost for the grip and twist sequence for different number of iterations of the SMD algorithm. At 30fps there are two regions (frames 20-40 and frames 80-120) where the cost for the different number of iterations diverges, with fewer iterations resulting in higher cost. However at 240fps this effect is far less dramatic, with only a slight increase in cost from frames 500-1000 and all lines having very similar profiles.

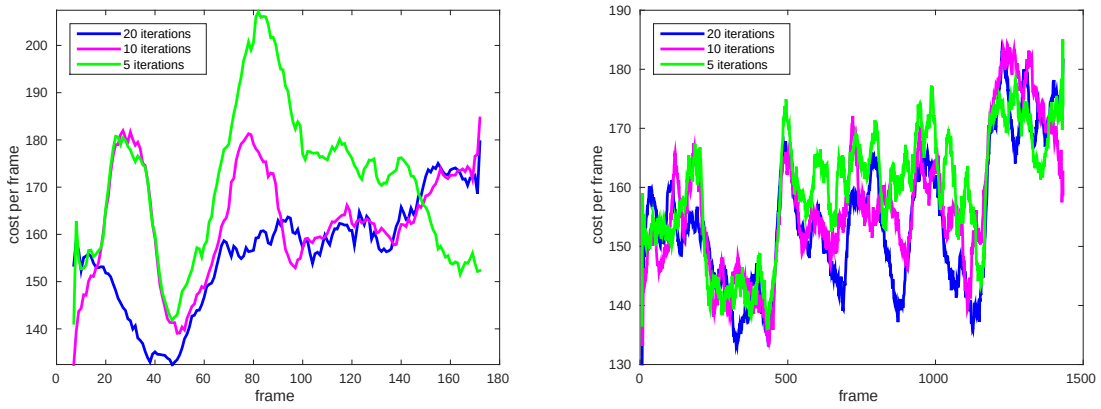

Figure 7.3: Cost as a function of optimisation algorithm iterations for the grip and twist sequence at left: 30fps, and right: 240fps (best seen in colour)

As the change of pose between frames is so small at the higher frame rate, fewer iterations are needed for convergence presuming that convergence had been reached at the previous frame. This results in the similar behavior across all iteration counts. This also explains the slight increase in cost observed at 240fps. The portion with increased cost involves global translation and rotation of the hand, with violent accelerations and decelerations. This results in greater differences between frames, and the need for a higher iterations count.

However there is a limit to how far the number of iterations can be decreased. Figure 7.4 shows tracking results for the finger flexion sequence using only a single iteration of SMD. Frame 4 shows the successful initial convergence to the neutral pose. Once the CMC joint of the thumb starts to rotate, the movement is too great for a single iteration, and the estimated pose progressively lags further behind the true pose as seen in frames 45 and 68.

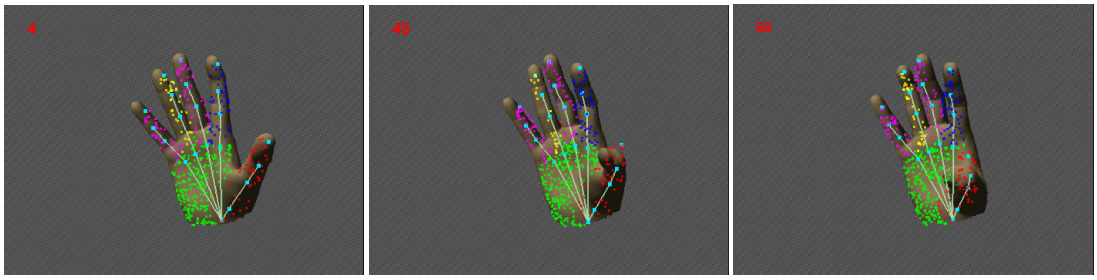

Figure 7.4: Selection of frames from the finger flexion sequence using 1 iteration of SMD and 1024 samples.

7.2.2 Sample Set Size

Figure 7.5 shows the cost for the grip and twist sequence at 30fps and 240fps. Sample sets sizes range from 1024 to 100. At 30fps there appears to be a trend of increasing cost with decreasing sample set size, particularly from frames 100-150. This trend is not present at 240fps, with all lines showing similar levels of cost. As the frame rate strongly implies that the estimated pose is correct at the previous frame, it appears that a sparse sample set is sufficient for accurate tracking, resulting in similar behavior for all sample sets at 240fps.

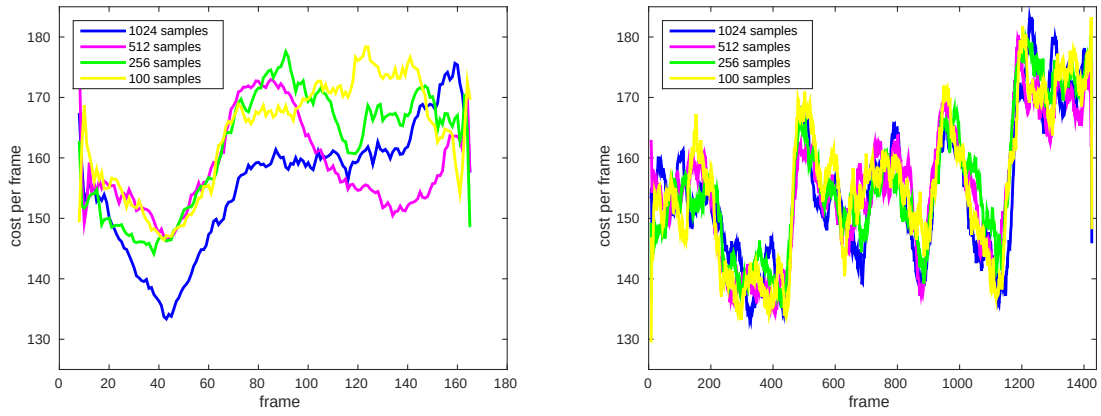

Figure 7.5: Cost as a function of sample set size for the grip and twist sequence at left: 30fps, and right: 240fps (best seen in colour)

Figure 7.6 shows a selection of frames using 100 samples. It can be observed in frames 170, 245 and 269 that the estimated pose is correct with the joint angles closely matching those of the true pose. However further decreasing the size of the sample set to 10 samples can result in some digits not being represented at all. This results in these regions remaining untracked due to the absence of cues, and can result unstable behavior as samples taken from these regions are entirely outside the silhouette of the observed pose.

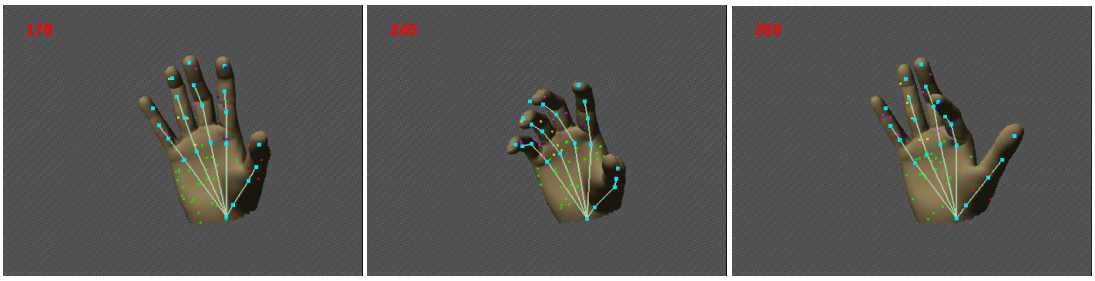

Figure 7.6: Selection of frames from the finger flexion and twist sequences using 20 iterations of SMD and 100 samples

7.2.3 Initial Convergence

The previous results assumed that the estimated pose at the previous frame is correct. While this assumption holds for these sequences, it does not take into account the initial convergence when the tracker is initialised. Figure 7.7 shows the initial convergence to a neutral pose for 5 and 20 iterations of the optimisation algorithm.

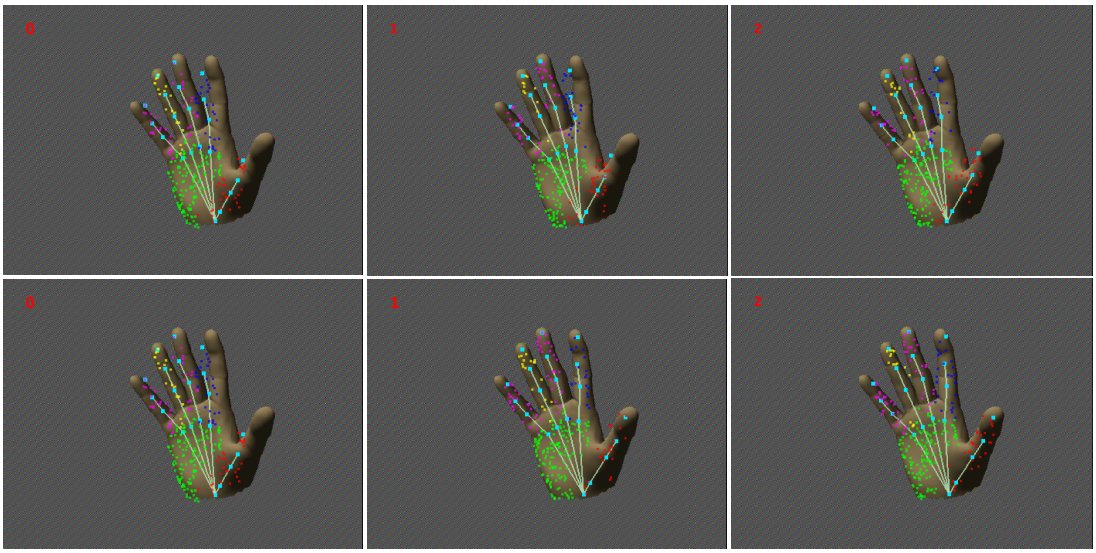

Figure 7.7: Selection of frames showing initial convergence for 5 iterations (top) and 20 iterations (bottom)

Note that while error is visually small, the initial estimate must be on the surface of the hand due to the saturation of the silhouette function. It can be seen that when using 20 iterations, the estimated pose converges to the true pose at the next frame, while when using 5 iterations, it is still converging several frames later. In practice this would require a predetermined pose to initialise the tracker (to nullify the saturation) as well as an increased iteration count for fast initial convergence.

7.3 Towards Real Time Implementation

Due to the limits mentioned previously, 5 iterations and 100 samples was found to be a good compromise between the number of computations and tracker accuracy. A selection of frames for these parameters are shown in Figure 7.8. It is apparent that there are enough iterations for convergence and sufficient information of the pose for accurate representation, with finger flexion accurately tracked.

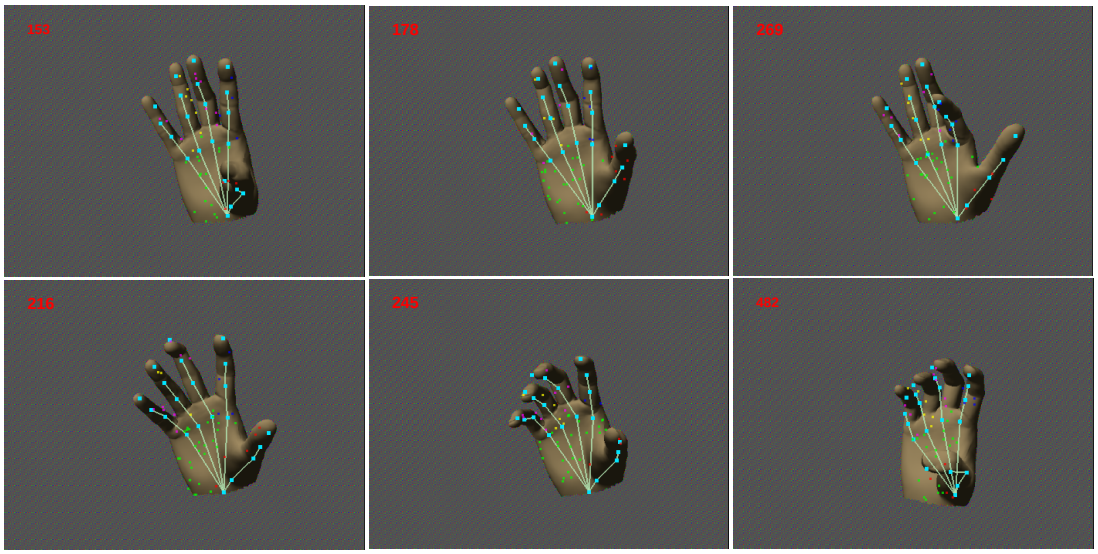

Figure 7.8: A selection of frames from the finger flexion (top) and twist (bottom) sequences using 5 SMD iterations and 100 samples.

The computation time required by the optimisation algorithm per frame is used as a measure of tracker performance as it is constant between implementations. Reducing the optimisation iterations and set size both decrease the number of computations required per frame, hence increasing the speed of the tracker.

Figure 7.9 shows the relationship between these and computation time to be linear, as expected as each scale the number of computations per frame linearly. Reducing to 5 iterations and 100 samples from 20 and 1024 decreases computation

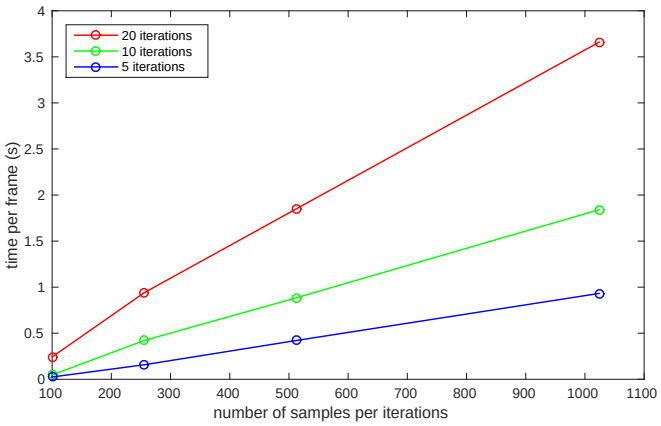

Figure 7.9: Execution time per frame for the SMD optimisation algorithm for different numbers of iterations and sample set sizes.

time per frame from 3.65s to 25ms, a 150 fold increase in speed. While this is still not a real time implementation it represents a considerable increase in speed over the original implementation, and there are several more areas for improvement. The current implementation of the optimisation algorithm uses several memory references per pixel (individual references for each portion of the cost function as well as the optimisation algorithm). By rearranging the way computations are conducted, the sample values could be passed to the optimisation algorithm directly from the FPGA, removing the need for several memory references per sample. However it should be noted that unlike the original tracker, this process can not be threaded due to the fact there is only a single data bus (GPIF) between the FPGA and CPU.

Chapter 8

Conclusion

This work progresses a real time hand tracking system further towards it goal of real time operation, through the use of a heterogeneous architecture. The tracking system has shown to be able to track a range of hand gestures, as well as cope with issues such as self-occlusion and fast finger movements.

A significant contribution made is the design and implementation of a control architecture suitable for real time hand tracking. Image processing pipelines are implemented on the FPGAs for processing raw data from CMOS sensors and storing in external SDRAM banks. Synchronous bus controllers are used for transferring data between the FPGA and microcontroller. A unit test demonstrating proof of concept is also presented, showing successful tracking within the limitations of the system. The major contribution of this work is to show the effect that high frame rate has on tracking accuracy and speed. By increasing the frame rate to 240fps it was shown that tracking accuracy was greater, with none of the non-convergence issues due to high velocities seen at 30fps. Additionally the higher frame rate allowed far fewer iterations and very sparse samples sets to be used whilst maintaining tracker accuracy. This resulted in a 150 fold increase in speed of the optimisation algorithm when compared to the original tracker.

Future Works

Currently only the image processing is done on the FPGA with the total cost being summed on the host computer. Because of this the majority of the optimisation time per frames is dealing with memory handling and the computation of the total cost. By moving the entire cost function routine to the FPGA, the amount of memory handling on the host computer will be dramatically decreased, theoretically allowing much greater computational efficiency.

While using a heterogeneous architecture has been shown to result in greater speed, it is not clear that the architecture used is the correct one. The main drawbacks of the current architecture are the bottleneck in bandwidth between the micro controller and FPGA, and the lack of floating point computation on the FPGA. A system on a chip architecture such as the Nvidia Jetson TK1 which includes a GPU core, image processing (IP) pipelines and a quad core CPU would allow grater flexibility

and might be more suited to the task, particularly as the GPU handles floating point operations and there is extensive support for development of algorithms provided by Nvidia. Nvidia also has available a release of OpenCV optimized for implementation on their CUDA GPU cores. It is proposed that the tracker be implemented on this hardware using the IP pipelines for image processing, the GPU for simple operations such as the computation of the cost function and kinematic chains for the hand model. The core tracking and optimisation algorithms will be implemented on the CPU using a threaded architecture.

One of the key drawbacks of the system for large scale adoption is the use of the hand model. As an individuals hand is unique, a model must be created separately for each one, a time intensive process. A highly desirable alternative would be something equivalent to the 'eigenface', or if this is not possible, at least an efficient and automated method for constructing the model initially. Using a 3D scanner a static model of the individuals hand could be produced, which could be used as the ground truth for adjusting the linear blending parameters of the skin mesh to best model the skin deformation characteristics of the individual. While this is a large problem it can be computed offline, making computational speed less of an issue.

Appendix A

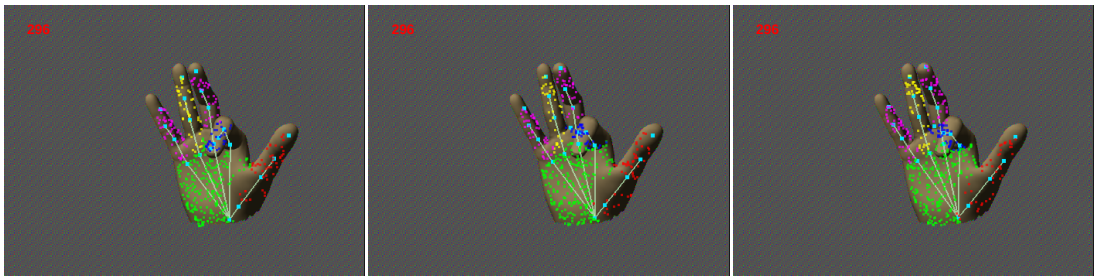

Figure 1: Frame 296 of the finger flexion sequence, showing 20, 10, and 5 iterations of SMD from left to right.

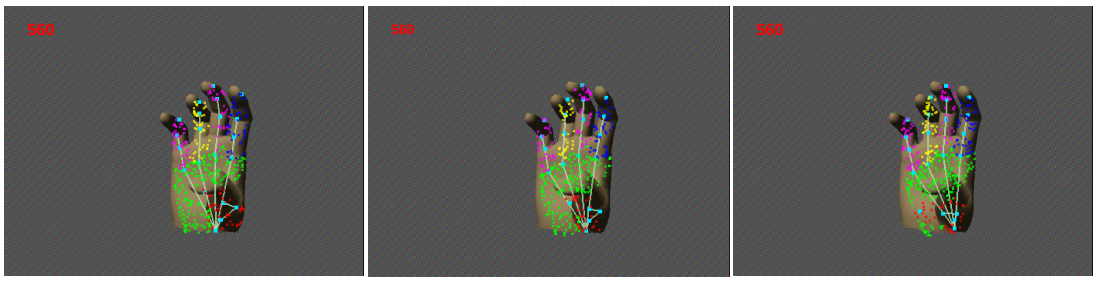

Figure 2: Frame 560 of the grip and twist sequence, showing 20, 10, and 5 iterations of SMD from left to right.

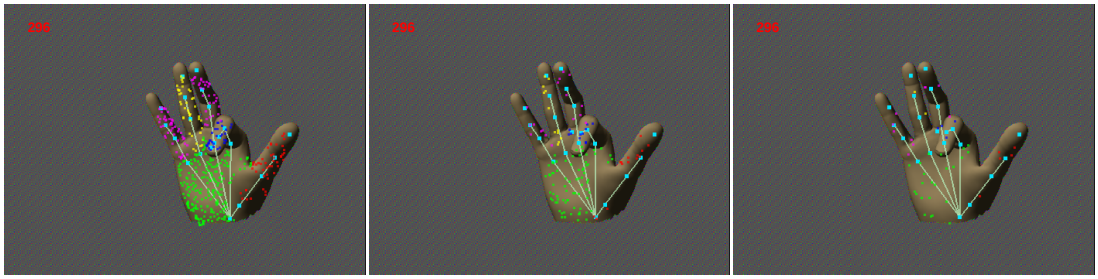

Figure 3: Frame 296 of the finger flexion sequence, showing sample sets of 1024, 256, and 100 from left to right.

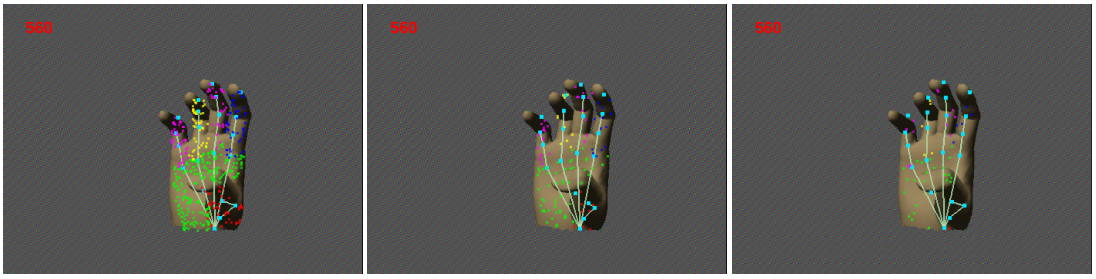

Figure 4: Frame 560 of the grip and twist sequence, showing sample sets of 1024, 256, and 100 from left to right.

Bibliography

1. Aguilar-Gonzalez, A.; Perez-Patricio, M.; Arias-Estrada, M.; Camas-Anzueto, J.-L.; Hernandez-de Leon, H.-R.; and Sanchez-Alegria, A., 2015. An FPGA correlation-edge distance approach for disparity map. In *2015 International Conference on Electronics, Communications and Computers*, 21–28. IEEE. (cited on page 9)

2. Ahmad, T.; Taylor, C. J.; Lanitis, A.; and Cootes, T. F., 1997. Tracking and recognising hand gestures, using statistical shape models. *Image and Vision Computing*, 15, 5 (1997), 345–352. (cited on page 4)

3. Al-Maadeed, S.; Almotaeryi, R.; Jiang, R.; and Bouridane, A., 2014. Robust human silhouette extraction with Laplacian fitting. *Pattern Recognition Letters*, 49 (2014), 69–76. (cited on page 6)

4. Ali, U. and Malik, M. B., 2010. Hardware/software co-design of a real-time kernel based tracking system. *Journal of Systems Architecture*, 56, 8 (2010), 317–326. (cited on page 8)

5. Ando, T.; Moshnyaga, V. G.; and Hashimoto, K., 2012. A low-power FPGA implementation of eye tracking. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 1573–1576. IEEE. (cited on page 8)

6. Arias-Estrada, M. and Rodríguez-Palacios, E., 2002. An FPGA co-processor for real-time visual tracking. In *Proceedings of the Reconfigurable Computing Is Going Mainstream, 12th International Conference on Field-Programmable Logic and Applications*, FPL '02, 710–719. Springer-Verlag, London, UK, UK. (cited on page 8)

7. Atay, M. and Yalcin, M. E., 2013. A parallelized distance transformation architecture for FPGAs. In *2013 European Conference on Circuit Theory and Design*, 1–4. TU Dresden. (cited on page 9)

8. Bae, K.-H.; Koo, J.-S.; and Kim, E.-S., 2003. A new stereo object tracking system using disparity motion vector. *Optics Communications*, 221, 1 (2003), 23–35. (cited on page 7)

9. Baudel, T. and Beaudouin-Lafon, M., 1993. Charade: remote control of objects using free-hand gestures. *Communications of the ACM*, 36, 7 (1993), 28–35. (cited on page 3)

10. Borgefors, G., 1986. Distance transformations in digital images. *Computer Vision Graphical Image Processing*, (3 1986), 344–371. (cited on pages 9, 13, and 36)

11. Boudabous, A.; Atitallah, A. B.; Khriji, L.; Kadionik, P.; and Masmoudi, N., 2011. FPGA implementation of vector directional distance filter based on HW/SW environment validation. *AEU - International Journal of Electronics and Communications*, 65, 3 (2011), 250 – 257. (cited on page 9)

12. Cai, L.; He, L.; Xu, Y.; Zhao, Y.; and Yang, X., 2010. Multi-object detection and tracking by stereo vision. *Pattern Recognition*, 43, 12 (2010), 4028–4041. (cited on page 7)

13. Canny, J., 1986. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8, 6 (1986), 679–698. (cited on page 6)

14. Chang, C.-C., 2006. Adaptive multiple sets of CSS features for hand posture recognition. *Neurocomputing*, 69, 16 (2006), 2017–2025. (cited on pages 4 and 5)

15. Chen, F.-S.; Fu, C.-M.; and Huang, C.-L., 2003. Hand gesture recognition using a real-time tracking method and hidden Markov models. *Image and Vision Computing*, 21, 8 (2003), 745–758. (cited on pages 4, 6, and 7)

16. Chen, W. and Mied, R. P., 2013. Optical flow estimation for motion-compensated compression. *Image and Vision Computing*, 31, 3 (2013), 275 – 289. (cited on page 7)

17. Chen, W.; Yue, H.; Wang, J.; and Wu, X., 2014. An improved edge detection algorithm for depth map inpainting. *Optics and Lasers in Engineering*, 55 (2014), 69–77. (cited on page 6)

18. Chik, D., 2009. *3D Hand Tracking in a Stochastic Approximation Framework*. Phd, Australian National University. (cited on pages 2, 4, 11, 13, 36, 40, and 51)

19. Cho, J. U.; Jin, S. H.; Pham, X. D.; Jeon, J. W.; Byun, J. E.; and Kang, H., 2006. A real-time object tracking system using a particle filter. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, 2822–2827. (cited on page 8)

20. Cho, J. U.; Jin, S. H.; Pham, X. D.; Kim, D.; and Jeon, J. W., 2007. FPGA-based real-time visual tracking system using adaptive color histograms. In *Robotics and Biomimetics, 2007. ROBIO 2007. IEEE International Conference on*, 172–177. (cited on page 8)

21. Choi, S.; Kim, T.; and Yu, W., 2009. Performance evaluation of RANSAC family. In *2009 British Machine Vision Conference (BMVC)*. Cited By 1. (cited on page 6)

22. Chung, P. .; Huang, C. .; and Chen, E. ., 2007. A region-based selective optical flow back-projection for genuine motion vector estimation. *Pattern Recognition*, 40, 3 (2007), 1066–1077. (cited on page 7)

23. CMOSIS, 2013. *Megapixel machine vision CMOS image sensor*. COMSIS Image Sensors. (cited on pages 15 and 17)

24. Coleca, F.; State, A.; Klement, S.; Barth, E.; and Martinetz, T., 2014. Self-organizing maps for hand and full body tracking. *Neurocomputing*, 147 (2014), 174. (cited on pages 3 and 6)

25. Cui, J. and Sun, Z., 2004. Model-based visual hand posture tracking for guiding a dexterous robotic hand. *Optics Communications*, 235, 4 (2004), 311–318. (cited on pages 4, 5, and 6)

26. Cypress, 2013. *EZ-USB FX3: SuperSpeed USB Controller*. Cypress. (cited on page 15)

27. Dankers, A.; Barnes, N.; and Zelinsky, A., 2007. MAP ZDF segmentation and tracking using active stereo vision: Hand tracking case study. *Computer Vision and Image Understanding*, 108, 1 (2007), 74–86. (cited on pages 6, 7, and 8)

28. Dente, E.; Bharath, A. A.; Ng, J.; Vrij, A.; Mann, S.; and Bull, A., 2006. Tracking hand and finger movements for behaviour analysis. *Pattern Recognition Letters*, 27, 15 (2006), 1797–1808. (cited on page 7)

29. Diaz, J.; Ros, E.; Pelayo, F.; Ortigosa, E. M.; and Mota, S., 2006. FPGA-based real-time optical-flow system. *IEEE Transactions on Circuits and Systems for Video Technology*, 16, 2 (2006), 274–279. (cited on page 9)

30. Dinu, D.; Bidiugan, R.; Natta, F.; and Houel, N., 2012. Preliminary study of accuracy and reliability of high-speed human-motion tracking using miniature inertial sensors. In *2012 Engineering of Sport Conference*, vol. 34, 790–794. Cited By 0. (cited on page 3)

31. Fels, S. S. and Hinton, G. E., 1993. Glove-talk: a neural network interface between a data-glove and a speech synthesizer. *IEEE Transactions on Neural Networks*, 4, 1 (1993), 2–8. (cited on page 3)

32. Felzenszwalb, P. F. and Huttenlocher, D. P., 1998. Image segmentation using local variation. In *1998 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 98–104. (cited on page 6)

33. Feng, Z.; Yang, B.; Chen, Y.; Zheng, Y.; Xu, T.; Xu, T.; Li, Y.; and Zhu, D., 2011. Features extraction from hand images based on new detection operators. *Pattern Recognition*, 44, 5 (2011), 1089–1105. (cited on pages 5 and 6)

34. Feng, Z.; Yang, B.; Tang, H.; Lv, N.; Meng, Q.; Yin, J.; and Feng, S., 2014. Behavioral-model-based freehand tracking in a Selection-Move-Release system. *Computers & Electrical Engineering*, 40, 6 (2014), 1827. (cited on pages 4 and 5)

35. Gao, W.; Zhang, X.; Yang, L.; and Liu, H., 2010. An improved Sobel edge detection. In *2010 3rd IEEE International Conference on Computer Science and Information Technology (ICCSIT)*, vol. 5, 67–71. (cited on page 6)

36. Gultekin, G. K. and Saranli, A., 2013. An FPGA based high performance optical flow hardware design for computer vision applications. *Microprocessors and Microsystems*, 37, 3 (2013), 270–286. (cited on page 9)

37. Harville, M., 2004. Stereo person tracking with adaptive plan-view templates of height and occupancy statistics. *Image and Vision Computing*, 22, 2 (2004), 127–142. (cited on page 7)

38. Hezel, S.; Kugel, A.; Manner, R.; and Gavrila, D. M., 2002. FPGA-based template matching using distance transforms. In *10th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, 89–97. IEEE. (cited on page 9)

39. Horn, B. K. P. and Schunck, B. G., 1981. Determining optical flow. *Artificial Intelligence*, 17, 1 (1981), 185–203. (cited on page 7)

40. Hsiao, Y.-Z. and Pei, S.-C., 2014. Edge detection, color quantization, segmentation, texture removal, and noise reduction of color image using quaternion iterative filtering. *Journal of Electronic Imaging*, 23, 4 (2014). (cited on page 6)

41. Hu, X.; Li, Q.; and Li, X., 2010. A real-time multipoint tracking system based on FPGA for multi-touch and motion tracking. In *Optical Metrology and Inspection for Industrial Applications*, vol. 7855. (cited on page 8)

42. Instruments, T., 1995. *AN-1006 Phase-Locked Loop Based Clock Generators*. Texas Instruments. (cited on pages 15 and 22)

43. Jones, M. J. and Rehg, J. M., 2002. Statistical color models with application to skin detection. *International Journal of Computer Vision*, 46, 1 (2002), 81–96. (cited on page 5)

44. Kikuchi, H. and Morioka, K., 2012. Development of wireless image sensor nodes based on FPGA for human tracking in intelligent space. In *38th Annual Conference on IEEE Industrial Electronics Society*, 5529–5534. IEEE. (cited on page 8)

45. Klein, A., 2013. Linear feedback shift registers. In *Stream Ciphers*, 17–58. Springer London. ISBN 978-1-4471-5078-7. (cited on pages 10 and 34)

46. Kreizer, M. and Liberzon, A., 2011. Three-dimensional particle tracking method using FPGA-based real-time image processing and four-view image splitter. *Experiments in Fluids*, 50, 3 (2011), 613–620. (cited on page 8)

47. Kristensen, F.; Hedberg, H.; Jiang, H.; Nilsson, P.; and Öwall, V., 2008. An embedded real-time surveillance system: Implementation and evaluation. *Journal of Signal Processing Systems*, 52, 1 (2008), 75–94. (cited on page 8)

48. Lamberti, L. and Camastra, F., 2012. Handy: A real-time three color glove-based gesture recognizer with learning vector quantization. *Expert Systems With Applications*, 39, 12 (2012), 10489. (cited on page 3)

49. Lattice, 2013. *Lattice ECP3 Family Datasheet*. Lattice Semiconductors. (cited on pages 14 and 17)

50. Lee, C.-S.; Chun, S.; and Park, S. W., 2013. Tracking hand rotation and various grasping gestures from an IR camera using extended cylindrical manifold embedding. *Computer Vision and Image Understanding*, 117, 12 (2013), 1711. (cited on pages 4 and 5)

51. Lee, J.-S.; Ko, J.-H.; and Kim, E.-S., 2001. Real-time stereo object tracking system by using block matching algorithm and optical binary phase extraction joint transform correlator. *Optics Communications*, 191, 3 (2001), 191–202. (cited on page 7)

52. Lee, S.-W. and Zhang, X., 2007. Biodynamic modeling, system identification, and variability of multi-finger movements. *Journal of Biomechanics*, 40, 14 (2007), 3215–3222. (cited on page 1)

53. Li, Y.; Chow, P.; Jiang, J.; Zhang, M.; and Wei, S., 2014. Software/hardware parallel long-period random number generation framework based on the WELL method. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 22, 5 (May 2014), 1054–1059. (cited on page 9)

54. Mansingka, A.; Radwan, A. G.; and Salama, K., 2012. Fully digital 1-D, 2-D and 3-D multiscroll chaos as hardware pseudo random number generators. In *Circuits and Systems (MWSCAS), 2012 IEEE 55th International Midwest Symposium on*, 1180–1183. (cited on pages 9 and 10)

55. Marr, D. and Hildreth, E., 1980. Theory of edge detection. *Proceedings of the Royal Society of London. Series B. Biological Sciences*, 207, 1167 (1980), 187–217. (cited on page 6)

56. Martin, J. L.; Zuloaga, A.; Cuadrado, C.; Lãzaro, J.; and Bidarte, U., 2005. Hardware implementation of optical flow constraint equation using FPGAs. *Computer Vision and Image Understanding*, 98, 3 (2005), 462–490. (cited on page 9)

57. Marzotto, R.; Zoratti, P.; Bagni, D.; Colombari, A.; and Murino, V., 2010. A real-time versatile roadway path extraction and tracking on an FPGA platform. *Computer Vision and Image Understanding*, 114, 11 (2010), 1164–1179. (cited on page 8)

58. Muñoz Salinas, R.; Aguirre, E.; and Garcia-Silvente, M., 2007. People detection and tracking using stereo vision and color. *Image and Vision Computing*, 25, 6 (2007), 995–1007. (cited on page 7)

59. Muñoz Salinas, R.; Medina-Carnicer, R.; Madrid-Cuevas, F. J.; and Carmona-Poyato, A., 2009. People detection and tracking with multiple stereo cameras using particle filters. *Journal of Visual Communication and Image Representation*, 20, 5 (2009), 339–350. (cited on page 7)

60. Nayak, S. and Pujari, S. S., 2015. Moving object tracking application: FPGA and model based implementation using image processing algorithms. In *2015 International Conference on Computing Communication Control and Automation*, 932–936. IEEE. (cited on page 8)

61. Placidi, G.; Avola, D.; Iacoviello, D.; and Cinque, L., 2013. Overall design and implementation of the virtual glove. *Computers in biology and medicine*, 43, 11 (2013), 1927–1940. (cited on page 3)

62. Premaratne, P.; Ajaz, S.; and Premaratne, M., 2013. Hand gesture tracking and recognition system using Lucas-Kanade algorithms for control of consumer electronics. *Neurocomputing*, 116 (2013), 242. (cited on page 6)

63. Prisacariu, V. A. and Reid, I., 2012. 3d hand tracking for human computer interaction. *Image and Vision Computing*, 30, 3 (2012), 236 – 250. Best of Automatic Face and Gesture Recognition 2011. (cited on page 5)

64. Qian, C., 2014. Realtime and robust hand tracking from depth. In *IEEE Conference on Computer Vision and Pattern Recognition*. (cited on page 1)

65. Ranasinghe, D. C.; Lim, D.; Devadas, S.; Jamali, B.; Zhu, Z.; and Cole, P. H., 2005. An integrable low-cost hardware random number generator. In *Proc. SPIE*, vol. 5649, 627–639. (cited on page 9)

66. Ruiz-Alzola, J.; Alberola-Lúpez, C.; and Corredera, J.-R. C., 2000. Model-based stereo-visual tracking: Covariance analysis and tracking schemes. *Signal Processing*, 80, 1 (2000), 23–43. (cited on page 7)

67. Sileshi, B.; Ferrer, C.; and Oliver, J., 2014. Accelerating hardware Gaussian random number generation using ziggurat and cordic algorithms. In *SENSORS, 2014 IEEE*, 2122–2125. (cited on pages 9 and 10)

68. Snelgar, F., 2014. *Aspects of a Real Time Hand Tracking System in a Stochastic Approximation Framework*. Undergraduate report, Australian National University. (cited on pages 2, 13, 16, and 35)

69. Stauffer, C. and Grimson, W. E. L., 1999. Adaptive background mixture models for real-time tracking. In *IEEE Computer Society on Computer Vision and Pattern Recognition*, vol. 2, 246–252 Vol. 2. (cited on page 5)

70. Sudha, N., 2005. A pipelined array architecture for Euclidean distance transformation and its FPGA implementation. *Microprocessors and Microsystems*, 29, 8 (2005), 405–410. (cited on page 9)

71. Tanabe, Y. and Maruyama, T., 2014. Fast and accurate optical flow estimation using FPGA. *ACM SIGARCH Computer Architecture News*, 42, 4 (2014), 27–32. (cited on page 9)

72. Tompson, J.; Stein, M.; LeCun, Y.; and Perlin, K., 2014. Real-time continuous pose recovery of human hands using convolutional networks. In *ACM Trans. Graph*. (cited on pages 1 and 4)

73. Winbond, 2014. *Winbond 512Mb Mobile LPSDR*. Winbond. (cited on page 15)

74. Xu, T. and Potkonjak, M., 2013. Lightweight digital hardware random number generators. In *SENSORS, 2013 IEEE*, 1–4. (cited on page 9)

75. Zafar, I.; Zakir, U.; Romanenko, I.; Jiang, R. M.; and Edirisinghe, E., 2010. Human silhouette extraction on FPGAs for infrared night vision military surveillance. In *2010 Second Pacific-Asia Conference on Circuits*, vol. 1, 63–66. (cited on page 7)

76. Zhao, J.; Thornberg, B.; Shi, Y.; Hashemi, A.; för informationsteknologi och medier, I.; Fakulteten för naturvetenskap, t. o. m.; and Mittuniversitetet, 2012. Color segmentation on FPGA using minimum distance classifier for automatic road sign detection. In *2012 IEEE International Conference on Imaging Systems and Techniques (IST)*, 516–521. IEEE. (cited on page 9)

77. Zoidi, O.; Nikolaidis, N.; Tefas, A.; and Pitas, I., 2014. Stereo object tracking with fusion of texture, color and disparity information. *Signal Processing: Image Communication*, 29, 5 (2014), 573–589. (cited on page 7)