

Copyright is owned by the Author of the thesis. Permission is given for a copy to be downloaded by an individual for the purpose of research and private study only. The thesis may not be reproduced elsewhere without the permission of the Author.

Low latency vision-based control for robotics

A thesis presented in partial fulfilment of the requirements for the degree
of

Master of Engineering

In

Mechatronics

At Massey University, Manawatu, New Zealand

Joshua Lues

2018

Abstract

In this work, the problem of controlling a high-speed dynamic tracking and interception system using computer vision as the measurement unit was explored.

High-speed control systems alone present many challenges, and these challenges are compounded when combined with the high volume of data processing required by computer vision systems. A semi-automated foosball table was chosen as the test-bed system because it combines all the challenges associated with a vision-based control system into a single platform. While computer vision is extremely useful and can solve many problems, it can also introduce many problems such as latency, the need for lens and spatial calibration, potentially high power consumption, and high cost.

The objective of this work is to explore how to implement computer vision as the measurement unit in a high-speed controller, while minimising latencies caused by the vision itself, communication interfaces, data processing/strategy, instruction execution, and actuator control. Another objective was to implement the solution in one low-latency, low power, low cost embedded system. A field programmable gate array (FPGA) system on chip (SoC), which combines programmable digital logic with a dual core ARM processor (HPS) on the same chip, was hypothesised to be capable of running the described vision-based control system.

The FPGA was used to perform streamed image pre-processing, concurrent stepper motor control and provide communication channels for user input, while the HPS performed the lens distortion mapping, intercept calculation and “strategy” control tasks, as well as controlling overall function of the system. Individual vision systems were compared for latency performance. Interception performance of the semi-automated foosball table was then tested for straight, moderate-speed shots with limited view time, and latency was artificially added to the system and the interception results for the same, centre-field shot tested with a variety of different added latencies.

The FPGA based system performed the best in both steady-state latency, and novel event detection latency tests. The developed stepper motor control modules performed well in terms of speed, smoothness, resource consumption, and versatility. They are capable of constant velocity, constant acceleration and variable acceleration profiles, as well as being completely parameterisable. The interception modules on the foosball table achieved a 100% interception rate, with a confidence interval of 95%, and reliability of 98.4%. As artificial latency was added to the system, the performance dropped in terms of overall number of successful intercepts. The decrease in performance was roughly linear with a 60% in reduction in performance caused by 100 ms of added latency. Performance dropped to 0% successful intercepts when 166 ms of latency was added.

The implications of this work are that FPGA SoC technology may, in future, enable computer vision to be used as a general purpose, high-speed measurement system for a wide variety of control problems.

Acknowledgements

I would first like to thank my supervisors Donald Bailey and Gourab Sen Gupta for their support throughout this process. Between their busy schedules they gave me so much of their time and shared their knowledge with me whenever I needed it. Donald guided me with his extensive expertise in image processing, and Sen steered the overall direction of the project.

My gratitude to my wife, Nadia, cannot be expressed enough. I am grateful for all the meals, late nights she spent awake in solidarity, trips to the supermarket, and encouragement. She is truly a blessing and I am so lucky to have the most incredible wife a man could ask for.

I am thankful to my brother-in-law, Ruven, and my brother, Giulio. Ruven was my personal trainer, and a heathy body leads to a healthy mind. Giulio was an inspiration to me as he is the hardest working person I know. Every Skype conversation increased my motivation level.

I can't thank my parents (biological and in law) enough for all of their financial and emotional support. It really has been wonderful having such a loving family and I couldn't have done it without each and every one of them.

Table of contents

Acronyms and conventions used	ix
1. Introduction.....	1
1.1. Aims and objectives	3
1.2. Thesis statement	3
2. Subsystems required for low latency control of a vision-based system	4
2.1. Foosball table automation	5
2.2. Object tracking techniques	6
2.3. Computer vision-based control systems	6
2.3.1. Self-adjusting vision systems	6
2.3.2. Other methods to improve vision system robustness	7
2.3.3. Spatial calibration	8
2.3.4. Trajectory prediction and state estimation techniques.....	9
2.4. Image and signal processing on FPGA	11
2.4.1. Software and hardware-based image processing.....	11
2.4.2. Performance improvements	13
2.4.3. Power consumption and embedded capabilities.....	14
2.4.4. Programming and reconfigurability	14
2.5. Actuator control and sensing on FPGA	15
2.6. Conclusions.....	16
3. System overview, requirements and high-level architecture	17
3.1. Temporal and spatial resolution requirements – Camera parameters	17
3.2. System response.....	19
3.3. Interception performance	20
3.4. Power consumption.....	21
3.5. Modularity and versatility.....	21
3.6. Minimal latency from distribution of processes	22
3.7. Reconfigurability.....	24
3.8. Debuggability/traceability	24
3.9. Potential methods to achieve outcomes	25

3.9.1.	CPU or GPU based systems	26
3.9.2.	ASIC or custom hardware	27
3.9.3.	FPGA.....	27
3.9.4.	Summary of potential methods	29
3.9.5.	Proposed method	30
3.10.	High-level complete system architecture	30
3.10.1.	Semi-automated foosball table control system.....	30
3.10.2.	Compute subsystem	32
4.	Mechanical subsystem	34
4.1.	Introduction.....	34
4.1.1.	Custom aspects of foosball table design.....	34
4.2.	Materials.....	39
4.3.	Design aspects meeting official specifications.....	40
4.4.	Accurate vision system placement	42
5.	Vision subsystem	44
5.1.	Background.....	44
5.1.1.	Latency in control systems.....	44
5.1.2.	Related work	45
5.1.3.	Hardware tested	46
5.2.	Novel event detection latency.....	48
5.2.1.	Aim.....	48
5.2.2.	Methodology.....	48
5.2.3.	Results.....	49
5.3.	Steady state latency.....	50
5.3.1.	Aim	50
5.3.2.	Methodology.....	50
5.3.3.	Mathematics	51
5.3.4.	Results.....	52
5.4.	Discussion	53
5.4.1.	Immediate usefulness of results	53

5.4.2.	Comparison of performance	53
5.4.3.	Potential improvements	54
5.4.4.	Automated foosball	54
5.5.	Conclusions	55
5.6.	Implementation of final vision system	55
5.6.1.	Terasic D5M camera module	55
5.6.2.	Prototypes.....	56
5.6.3.	Foosball table interception simulation	58
5.6.4.	Lens changes	58
5.6.5.	Distortion correction.....	60
5.6.6.	Final implementation	61
6.	Compute subsystem	65
6.1.	Terasic DE1-SoC FPGA development board	65
6.1.1.	FPGA component	66
6.1.2.	HPS component	67
6.1.3.	SoC design	67
6.2.	Motor Control	73
6.2.1.	Background	73
6.2.2.	Mathematical approximation	74
6.2.3.	Stepper motor basic control algorithm.....	75
6.2.4.	Stepper motor control on FPGA.....	78
6.2.5.	Determining acceleration parameters	80
6.2.6.	Kicking algorithm	80
6.2.7.	Homing and sliding algorithm	81
6.2.8.	Interception algorithm and image to table spatial mapping	83
6.3.	Resource requirements	87
6.4.	Summary.....	88
7.	System integration	89
7.1.	Electrical system	89
7.1.1.	Overall electrical system architecture	89

7.1.2.	36V stage.....	91
7.1.3.	12V stage.....	91
7.1.4.	Limit and homing switches	92
7.1.5.	Wiring management and cabling	93
7.1.6.	Field illumination	93
8.	Testing and system performance	95
8.1.	Aims	95
8.2.	Spatial invariance for interception	95
8.2.1.	Testing method	95
8.2.2.	Overview of apparatus.....	96
8.2.3.	Data analysis	97
8.2.4.	Results.....	98
8.2.5.	Discussion	98
8.3.	Close-up interception performance	98
8.3.1.	Testing method	99
8.3.2.	Results.....	99
8.3.3.	Discussion	99
8.4.	Close-up interception performance with added latency.....	99
8.4.1.	Method to artificially add latency	99
8.4.2.	Testing method	100
8.4.3.	Results.....	100
8.4.4.	Discussion	101
8.5.	Automated foosball	102
8.6.	Potential improvements or additions.....	102
8.7.	Conclusions and recommendations.....	102
9.	Final conclusions and recommendations	103
	References.....	104
	Published work.....	110
	Appendices.....	111
	Appendix A.....	111

Appendix B.....	113
Appendix C.....	114
Appendix D.....	116
Appendix E.....	117
Appendix F.....	118
Appendix G.....	119
Appendix H.....	120
Appendix I.....	120
Appendix J.....	121

Acronyms and conventions used

A/D or D/A converter – Analogue to digital or digital to analogue

ASIC – Application specific integrated circuit

CCD – Charge coupled device

CMOS – Complementary metal oxide semiconductor

CPU – Central processing unit

DCS – Distributed control system

DLL – Delay locked loop

DSP – Digital signal processor

DVS – Dynamic vision sensor

FPGA – Field programmable gate array

FPS – Frames per second

GPIO – General purpose input/output

GPU – Graphics processing unit

HDL – Hardware description language

HPS – Hard processor system

IDE – Integrated development environment

MP – Megapixel

MPPA – Massively parallel processor array

MSB – Most significant bit

OS – Operating system

OTS – Off the shelf

PLL – Phase locked loop

USB – Universal serial bus

VHDL – Very high-speed integrated circuit HDL

List of figures

Figure 1-1 - Standard foosball table	3
Figure 2-1 - Cycle diagram representing algorithmic image processing	12
Figure 2-2 -Streamed image processing in FPGA hardware	13
Figure 3-1 - Representation of ball motion blur on foosball table – birds-eye view with foosmen hidden from view	18
Figure 3-2 - Foosball playing field captured from below, using the DE1-SoC and D5M camera.....	19
Figure 3-3 - One cycle of image capture through to system actuation response	20
Figure 3-4 - Latencies present in systems tested by Čížek et al. 2016	23
Figure 3-5 - Flowchart representing the data flow, communication, and data transfers in distributed PC based system.....	23
Figure 3-6 - Possible configurations of PC based image capture and actuator control systems	26
Figure 3-7 - Semi-automated foosball table.....	31
Figure 3-8 - CAD model of the automated actuation modules	31
Figure 3-9 - Compute system input and output signals	32
Figure 4-1 - Render of automated modules of CAD model.....	34
Figure 4-2 - Pull out torque curve of Nema 23 bi-polar stepper motor - (Pololu, 2018)	35
Figure 4-3 - Render of actuation modules	35
Figure 4-4 - Foosball table actuation module - rotational drive assembly.....	36
Figure 4-5 - Foosball table actuation module - linear drive assembly	37
Figure 4-6 – Close-up of belt connector on actuation module	37
Figure 4-7 - From left to right a bottom-up view of the glass base, the interlock switch and the safety lid	38
Figure 4-8 - From left to right - bottom-up view of foosball playing field with field illumination on and off respectively	39
Figure 4-9 - Image of completed, vinyl wrapped semi-automated foosball table – from front right.....	39
Figure 4-10 - Render of foosball table CAD model	40
Figure 4-11 - Render of foosball goal on CAD model	41
Figure 4-12 - Semi automated foosball table right hand ball return chute.....	41
Figure 4-13 - CAD model of the placement jig for the foosball table vision system	42
Figure 4-14 - Manufactured calibration homing jig with rotating locking tabs.....	43
Figure 5-1 - Image capture test systems	47
Figure 5-2 – Novel event detection latency results for all 5 experiments including PS3 eye at both resolutions.....	49
Figure 5-3 - CAD model of object marker apparatus used in steady state latency experiment	51
Figure 5-4 - Steady state latency results for all 4 experiments	52
Figure 5-5 - Vision system development process.....	55

Figure 5-6 - Terasic TRDB-D5M camera development board.....	56
Figure 5-7 - Bayer RGB pattern representation	56
Figure 5-8 - Output window of simple trajectory calculation and interception coordinate simulation	58
Figure 5-9 - Long focal-length (narrow angle) lens provided with the D5M camera module	59
Figure 5-10 - Sunex DSL215 fisheye lens	59
Figure 5-11 - MATLAB output image of all calibration grid intersection points found by distortion correction algorithm	61
Figure 5-12 - Uncalibrated (left) and calibrated (right) distortion point map	61
Figure 5-13 - Representation of overall image processing system	62
Figure 5-14 - Block diagram of 3x3 windowed Bayer interpolation (demosaicing) hardware implemented in FPGA. (Bailey, 2018). Reprinted with permission.....	62
Figure 5-15 - Block diagram of simplified RGB to YCbCr conversion implemented in FPGA hardware (Bailey, 2018). Reprinted with permission.	62
Figure 5-16 - Block diagram of a 5x5 morphological filter (erosion) implemented in FPGA hardware – for dilation, AND gates were replaced with OR gates. (Bailey, 2018). Reprinted with permission.	63
Figure 5-17 - Block diagram of the high-level architecture for connected components analysis (CCA) implemented in FPGA (Bailey, 2018). Reprinted with permission.	63
Figure 6-1 - Representation of vision-actuation control system data cycle from input to response.....	65
Figure 6-2 - Block diagram showing the master/slave relationship for each of the bridges between the FPGA and HPS (Altera, 2016).....	66
Figure 6-3 - Example Qsys HPS component	68
Figure 6-4 - Setup options for PIO in Qsys	69
Figure 6-5 - PIO connections and addresses in Qsys.....	69
Figure 6-6 - C code used to send an "active" signal to the FPGA, and disable all the stepper motors temporarily.....	70
Figure 6-7 - VHDL code to receive data/signals from the HPS	70
Figure 6-8 – Representation of data transfer method between FPGA and HPS	70
Figure 6-10 - Address map for Qsys SoC design	71
Figure 6-11 - Development process for the motor control design.....	73
Figure 6-12 - Nema 23 bipolar stepper motor - 200 steps per revolution	74
Figure 6-13 - Pull out torque curve of Nema 23 bi-polar stepper motor	74
Figure 6-14 - Acceleration and deceleration equations for stepper motor ramping	76
Figure 6-15 - Stepper motor simulation output curve for input distance = 600	76
Figure 6-16 - Stepper motor simulation output curve for input distance = 1200	77
Figure 6-17 - Output curves for constant velocity (top 3) and trapezoidal velocity (bottom 3) profiles ...	77
Figure 6-18 - Stepper module entity diagram showing inputs and outputs	78
Figure 6-19 - Block diagram of simplified variable acceleration pulse generation method, implemented in FPGA.....	79

Figure 6-20 – Birds-eye view of foosball table CAD model with automated rods annotated	82
Figure 6-21 – Birds-eye view of foosball rods - Module 4 to 1 from left to right.....	84
Figure 6-22 – Birds-eye view of foosball table with positions of each foosman on each module labelled	85
Figure 6-23 - C code used to calculate differences between ball and foosmen positions	86
Figure 6-24 - Block diagram representing entire SoC system design	87
Figure 7-1 - Development process for the mechanical/electrical system	89
Figure 7-2 - Electrical system architecture for the semi-automated foosball table.....	90
Figure 7-3 - Panel with 2 USB-B sockets, 1 D-Sub socket, 1 IEC socket, and a high-current switch	90
Figure 7-4 - PCB layout of 40-pin expansion board created using Altium Designer.....	92
Figure 7-5 - PCB with pluggable screw terminals and a 40-pin header.....	92
Figure 7-6 - One actuation module with each of the homing/limit switches labelled	93
Figure 7-7 - Wiring conduits on left of image and electrical screw glands on right	93
Figure 7-8 - Semi-automated foosball table with field illumination LEDs switched on	94
Figure 7-9 - XL4015 DC-DC voltage regulator module with adjustable output voltage	94
Figure 8-1 - Ball release apparatus positioned at the back of foosball table (closest to human goal)	96
Figure 8-2 - Experimental setup using module 2 for interception	97
Figure 8-3 - Code snippet of algorithm used to artificially delay system by 20 frames	100
Figure 8-4 - Output graph of interception performance versus artificially added latency for straight goal shots.....	101

List of tables

Table 5.1 - Important novel event detection latency data.....	50
Table 5.2 –Important steady state latency data	52
Table 5.3 - Resource consumption on DE1-SoC of basic image processing design	57
Table 6.1 - System PIOs and their respective functions	72
Table 6.2 - Time taken and maximum velocity for 3 simulated stepper motor velocity profiles	78
Table 6.3 - Increment or decrement based on current stepper motor cycle position.....	79
Table 6.4 - Comparison of resource utilisation for minimal design through to full SoC design	88

1. Introduction

Computer vision is an extremely powerful tool when used correctly. It has a plethora of applications including object tracking, object recognition, augmented reality, metrology, and many others. One application of computer vision in which further exploration would be beneficial is high-speed control systems or high-speed visual servoing. That is the focus of this thesis.

Control systems require accurate, low latency measurement systems, low latency control strategies and efficient processing of data to ensure all tasks are performed within the time specifications for each task (Dougherty & Laplante, 1995). In a critical response control system, with highly time-sensitive task execution, if any tasks are performed later than the specified execution time, then the control system will either be unstable or fail altogether. Individually, these requirements are all necessary within the context of a control system, however most of these requirements present a few limitations or problems of their own. Some systems require powerful or expensive processing hardware, some require sensitive equipment, and some are required to be calibrated or serviced very often. These limitations are significantly compounded when all the requirements are present in one system, for example a computer vision-based control system.

Low latency measurement is essential for any control system where the measurements are used as a feedback in the control loop, or when the measurement system provides input to the control system (Franklin, Powell, & Emami-Naeini, 2015). Measurement system latency must be sufficiently shorter than the required response time. This provides adequate phase margin and improves system stability (Engelberg, 2015). Achieving low latency image capture and processing from a computer vision-based measurement system is a challenge simply because of the volume of raw data and computation necessary to extract the required measurement data (Johnston, Gribbon, & Bailey, 2004).

Low latency control strategies are required when strategic planning or decision making is to be performed by the control system (Mueller, Censi, & Frazzoli, 2015). Large delays between the control system receiving the input signal and providing the output will result in poor system performance. This poor performance can be caused by delays in instruction execution due to inefficient control strategies and internal latency. Therefore, efficient processing of instruction data, calculations, and actuation tasks is essential and latency in control strategies should be minimised. Various methods of achieving this will be discussed further.

Additionally, during operation it can be difficult to debug these systems due to the unpredictable execution of tasks. In some cases, poor optimisation of internal processes can cause debugging mechanisms (such as printing to a serial console) to fail, due to the processor being saturated by other inefficient operations. In some cases, the interrupt caused by the debug mechanism can cause the internal process to fail depending on which has a higher interrupt priority. Another possibility is that interrupt service routines with higher priorities threaten to continuously supersede those with lower

priority, therefore blocking their execution altogether. Finally, control of more complex actuators like encoded DC motors or stepper motors requires some processing overhead and precise timing. This is particularly true with stepper motors as they are sensitive to timing jitter (Proctor & P. Shackelford, 2001). In an event driven control system with many actuators it can be a challenge to control many of these actuators with conventional methods due to processor power and other hardware resource restrictions, such as a limited number of hardware timers or limited CPU power, affecting the timing and execution of these calculations and tasks. These factors can make the control of complex systems with many asynchronous tasks and actuator control a significant challenge.

To overcome this problem, distributed control systems (DCS) use local processing of sensor and actuator data on dedicated hardware for a given process (Zhao, Paine, Kim, & Sentis, 2015). However, this generally comes at the cost of added latency caused by data transmission and communication, which needs to be accounted for in the design as communication latency can dramatically decrease the overall responsiveness of the control system (Tianjian & Fujimoto, 2006). Minimising communication latency is, therefore, another focus of this thesis.

Evidently, each of the above requirements for control systems have their own set of challenges. To explore and address these challenges, a system which utilises computer vision for low latency control of a complex mechatronic system is required as a test-bed. A semi-automated foosball table was chosen as the test-bed system because it combines all these challenges into a single system, while requiring vision-based sensing, distributed and embedded processing, low latency communication, and both high-level strategic control and low-level motor control for actuation.

Foosball, also known as table soccer, is a miniature soccer game played on a small table (shown in Figure 1-1) with a field area of approximately $1.2\text{ m} \times 0.7\text{ m}$. The game uses 1 ball and involves two to four players. It is played one-on-one or two-on-two, with players from the two teams standing and operating on opposite sides of the table. The players manually operate small rigid puppets (foosmen), attached to spinning and sliding metal rods to kick the ball into the opponent's goal, and to defend their own goal from opponent's kicks. There are four rods per team (eight rods in total) and between two and five foosmen per rod. It is a fast paced, dynamic game with ball speeds in excess of 10 m/s , and therefore requires very fast reflexes from the human players.



Figure 1-1 - Standard foosball table

1.1. Aims and objectives

The objective is to explore how to combine computer vision, communication interfaces, data processing/strategy, instruction execution, and scalable actuator control all in one low-latency, low power, embedded system.

This system will be integrated into a semi-automated foosball table and tested for ball interception performance as a method to establish the performance of the system. The semi-automated foosball table represents a reasonably high-speed control system problem, requiring low system response time in order to satisfy the real-time requirements involved in a vigorous game of foosball.

1.2. Thesis statement

A field programmable gate array (FPGA) system on chip (SoC), which combines programmable digital logic with a dual core ARM processor (HPS) on the same chip, should be capable of running a low latency control system. Latency can be reduced by distributing the tasks over the most appropriate processors, with programmable logic performing high-framerate streamed image pre-processing and actuator control, with the higher-level strategy and floating-point operations allocated to the HPS. Finally, communication latency may be minimised by utilising the high-performance AXI bridges between the FPGA and HPS on the SoC.

2. Subsystems required for low latency control of a vision-based system

The field of computer vision, object tracking, trajectory prediction, and control systems is massive, with research branching extensively into these fields. The use of computer vision as a feedback element for a high speed, low latency, mechatronics control systems has been explored to some extent with research in visual servoing. However, there are potential improvements that can be made to the speed of computer vision-based control systems.

This chapter will discuss the subsystems required for high-speed control of a system, using computer vision in the feedback loop. The subsystems required will be presented and discussed, in conjunction with some examples of computer vision-based control systems. Additionally, other semi-automated foosball tables will be compared.

There are several important topics to be addressed to make any improvements to state-of-the-art vision-based control systems. These topics include image processing hardware, the algorithms selected for predictive interception, the algorithmic processing hardware, the actuator control methodology, and the overall communication and latency within the control system. Additionally, for a smart image processing system, some level of intelligence is required. This may include smart colour classifiers, auto-adaptive colour thresholding, variable lighting intensity mapping, and object recognition, among other possible techniques. The integration and combination of techniques is of similar importance to the techniques used.

2.1. Foosball table automation

Some work has already been done regarding semi-automated foosball tables. Because semi-automated foosball is the development platform used in this project, the existing systems are covered in this and their contributions and limitations discussed.

Weigel (2005) developed a semi-automated foosball table with a vision-based ball tracking system. The camera ran at a speed of 50 Hz, at 384×288 pixels resolution. Despite the low resolution and framerate, their system performed well, winning 69.4% of games played – 354 out of 510 games played. The participants were mostly unskilled players without any special training. Because of the performance of their system, they later commercialised their design with the “Star-Kick” semi-automated foosball robot. Based on the ratio of goals received to goals scored – 2753 received to 4522 scored – it appears that the main area of improvement to be made to Weigel’s work was an increase in the goal defence ability of the robot. Given that the robot was able to win most games, the strategy was not a major area of weakness.

Janssen, de Best, and van de Molengraft (2010) discuss a semi-automated foosball table with a camera transmitting grayscale images to a computer which then performed the object recognition and tracking algorithms. The camera resolution was 657×446 pixels, and framerate was 100 Hz. In this paper the authors focussed on detection of the ball from above, where occlusion of the ball was a problem. They used a linear Kalman filter to predict the path of the ball for interception. The authors used a region of interest in their image processing algorithm to reduce the computational load. They also performed the distortion correction on the image stream, rather than post-processing the data. The defending rod was able to intercept a shot at 10 m/s (the maximum theoretical velocity based on tested values). The authors were not clear on the interception performance in terms of proportion of goals successfully defended, nor were they clear about their testing methodology. Interception performance is clearly a key area of potential improvement in semi-automated foosball tables.

Later, Janssen, Verrijt, de Best, and van de Molengraft (2012) built upon their initial survey paper with improved state estimation techniques, a higher camera framerate (200 Hz), and discussed why the initially proposed work was unsuitable for the application. The authors combined the use of colour filtering, and edge and shape detection to improve their detection and localisation of the ball. They achieved a peak error of 12mm in the predicted location of the ball for interception, however their system often missed the ball. This was assumed to be because of delays in their processing pipeline. Their image processing pipeline (calculated for greyscale images) was calculated as 17.5 ms in the worst-case scenario. Including image capture, communication, actuation and strategy calculations this number would have been significantly higher. Overall, the total system latency was the main cause of unsuccessful intercepts. The authors suggest simply predicting the location of the ball using the latency value assuming a constant velocity model, however this may not fully solve the problem as a constant velocity cannot always be assumed.

Given the advantages of the vision-based systems, computer vision appears to be the most useful method for object localisation in the context of a semi-automated foosball table. Some improvements to the systems could be made in terms of overall system response time by lowering individual components of latency in order to reduce the overall processing-actuation pipeline delay.

2.2. Object tracking techniques

Object tracking research includes the localisation of the object, the hardware and software required to perform the localisation, and the techniques used to detect or segment the object from its surroundings. The method used to detect the object is dependent on the required temporal and spatial precision, and various other factors such as cost, space and power consumption.

Many methods for tracking objects have been proposed, some with greater success than others. Generally, the use of computer vision tends to be the most successfully implemented method, as vision provides a large amount of data at high speeds and low relative cost. Given this fact, computer vision systems will be the focus of this review, though other methods do exist, such as the multi-touch screen implemented by (Korkalo & Honkamaa, 2010) using cameras with their optical plane parallel with the screen surface. In their work, they used several cameras at different offsets and angles to detect user touch inputs on the screen. While this method can have some advantages such as comparatively lower processing power requirements, and potentially higher tracking effectiveness under the right conditions, the foosball table was not a suitable testbed for this method of object localisation and tracking.

2.3. Computer vision-based control systems

2.3.1. Self-adjusting vision systems

Lighting is one of the most difficult issues to solve in vision-based systems (Hornberg & Jahr, 2017). Dynamic lighting conditions pose a problem for vision-based robotics applications. Slight changes in lighting over time, or spatial variation in lighting intensity, can cause a well calibrated system to malfunction as objects are not as effectively detected by colour segmentation techniques. Even greyscale images can be adversely affected by such varying lighting conditions. Therefore, it is very important that vision systems are either tuned for the desired image processing application every time the lighting changes, or better, the system self-tunes or auto-adjusts to accommodate different lighting conditions. Structured lighting conditions, where there is no variation in intensity or coloration and no ambient light ingress, are an effective method to eliminate this problem because the need to self-tune or self-adjust is generally not present in highly controlled lighting systems as the rest of the system is designed around the optimal lighting solution (Hornberg & Jahr, 2017). However, in some systems, particularly those in which human interaction is required, structured lighting is not always possible. This is the case with the semi-automated foosball table, as will be discussed later in the thesis.

A method of eliminating or mitigating the adverse effects of variation in image data is required in order to use vision in dynamic lighting conditions. Common methods such as automatic exposure control and

white balance exist (Kondo, Kikuchi, Kato, & Hirota, 1991), however they often do not fully solve the problem. Automatic exposure and white balance adjustments improve picture quality when the whole image is brighter or darker, however in applications where a very uniform lighting intensity distribution is required, increasing or decreasing the white balance or exposure of the entire image does not generally decrease the variation in illumination. A method to adjust only the affected parts of the image is required.

There are three common approaches used to automatically adjust for these dynamic lighting conditions.

The first method is “to output the images to describe the real scene as consistently as possible in different light conditions by auto-adjusting the camera parameters” (Lu, Zhang, Yang, & Zheng, 2010). This was mentioned previously and includes features such as auto-exposure and auto-white balance.

The second method involves transforms for colour constancy, as explored by Mayer, Utz, and Kraetzschmar (2002) using the Retinex algorithm, which attempts to mimic the human eye that can identify colours in differing lighting conditions. The Retinex algorithm uses dynamic range compression and tonal rendition with colour restoration to improve the observed disparity between human observed images and digitally captured images (Jobson, Rahman, & Woodell, 1997).

The third method is to use “adaptive or robust object recognition algorithms” in which “colour segmentation is combined with a shape detection to provide additional robustness and allow colour recalibration” as explored by several authors (Gönner, Rous, & Kraiss, 2005; Huimin, Zhiqiang, Fei, & Xiangke, 2008). In a tracking application such as a semi-automated foosball table where one object with distinct shape, size and colour (the ball) is being tracked, this could be a promising method. Improved segmentation techniques have been achieved in the work of Saber, Murat Tekalp, and Bozdagi (1997) by combining colour and edge information for better edge linking, and by Jianping, Yau, Elmagarmid, and Aref (2001) by using colour-edge extraction and seeded region growing to better identify object boundaries and distinct regions within images. Kryjak, Komorkiewicz, and Gorgon (2014) used a combination of colour segmentation and edge detection for applications such as background and foreground segmentation. These authors demonstrated the effectiveness of combining colour and shape features into their image processing algorithms.

Given the amount of effort that has been spent by other authors in this area it is obvious how important effective segmentation and identification of the object is. The complexity of the application, time constraints of processing, type of object being tracked, and non-uniformity of images determine which method is chosen.

2.3.2. Other methods to improve vision system robustness

Depending on the application, feature recognition can be the most effective method of improving the robustness of a vision-based system. With varying lighting conditions, the segmentation of the image will experience some variation and unpredictability. As demonstrated by Gönner et al. (2005), a

combination of colour segmentation and shape/feature detection resulted in highly effective object recognition. Their research was limited by the hardware (933 MHz Pentium III processor), as they were only able to process on average one image of 379×262 pixels in 33.84 milliseconds. Additionally, the recalibration, to remap the colour segmentation parameters and more effectively identify the object, took from 51 to 79 milliseconds, which, when low latency is important, is a very long time. With more powerful hardware, however, these numbers could be reduced significantly.

The work of Tsutsui, Nakamura, Hashimoto, Okuhata, and Onoye (2010) showed some potential; the authors implemented a real time FPGA based system which performed image enhancement, using the Retinex algorithm, on video stream of 1920×1080 resolution at 60 frames per second. This could potentially be implemented in the semi-automated foosball vision system to provide a more accurate output image to ensure correct segmentation.

2.3.3. Spatial calibration

In a robotic control system where a camera is used as the measurement system, the camera space needs to be calibrated such that the robotic actuators match the spatial coordinates and units of measurement of the camera or vice versa. In camera-based systems, an additional problem arises in which not only are the camera space and robot workspace initially uncalibrated, but the image is also distorted in some way. A common distortion is radial distortion.

In the case of the foosball table used in this research, the camera views the playing field from below. The wide view angle required by the camera necessitates the use of a wide-angle lens. This results in a radially distorted image. This distortion needs to be corrected to ensure accurate object coordinates are obtained by the vision system.

Weigel (2005) discuss their method of calibrating the camera space to correct for radial (barrel) distortion in their semi-automated foosball table, where the authors used the following truncated Taylor series representation:

$$r = r' + \alpha r'^3 \quad (1)$$

Where the r and r' terms are the corrected and uncorrected radii, respectively, about the centre of distortion. Some models also include tangential distortion terms, which shift the centre of distortion away from the image centre which would, depending on the system, normally be located at $(x, y) = (\frac{IM_{width}}{2}, \frac{IM_{height}}{2})$. The image centre is not always the centre of distortion however as has been shown by (Willson & Shafer, 1994).

The α term in equation (1) is an empirically derived scalar constant. In equation (1) including more terms results in lower residuals, however for most lenses which only introduce mild distortion, 1 to 2 terms is usually enough.

This approximation shown in equation (1) was used by Weigel (2005) to correct for the camera lens distortion. This is a common method and has also been implemented in the past by (Bailey, 2002; Gribbon, Johnston, & Bailey, 2003; Ngo & Asari, 2005) as well as many others.

In a mapping function with a single scalar term, such as equation (1), determining the value of the scalar constant is trivial. However, in polynomials with more terms such as the following equation, an optimisation loop is the most effective method to determine the constants.

$$r = r' \left(1 + (K_1(r'^2)) + (K_2(r'^4)) + (K_3(r'^6)) \right) \quad (2)$$

The process of finding terms K_1 , K_2 and K_3 would be very time consuming if calculated manually (brute force), and therefore an iterative non-linear optimisation algorithm such as the Levenberg-Marquardt algorithm would be the most effective way to determine the coefficients (Zelnik-Manor & Irani, 1996).

A calibration grid can be used to acquire an adequate number of measured points in the distorted image, and using those points, the mapping function in equation (2) can be used to correct the distortion with estimated initial values of the coefficients. The algorithm then compares the calculated points on the calibration grid with expected positions. The coefficients are then iteratively modified to minimise the observed error until convergence (minimised error) is achieved. Given that subpixel accuracy is not required for the semi-automated foosball table, this method should yield sufficiently accurate distortion correction with a small number of iterations.

Another method is a non-parametric model based method where the ball position is calculated using an empirically pre-determined map, and interpolating between points on the map with known distortion to determine the true position of the ball. Barreto, Swaminathan, and Roquette (2007) implemented a two stage non-parametric barrel distortion correction method for correcting distortion in endoscopic imaging systems with results surpassing the conventional methods in terms of “goodness of fit”, however their method required a calibration grid to define the distortion map, which in some systems is not practical or necessary.

(Nister, Stewenius, & Grossmann, 2005) devised a method of camera self-calibration based on what they called “motion” in the image, however this seemed to be aimed at images with irregular distortions in the image and is, therefore, more complex than necessary for the application of barrel distortion correction, where a simple 3rd or 5th order polynomial is usually sufficient.

Several methods of lens calibration are possible, and once again the selected method depends on system performance requirements such as precision, accuracy, and vision system timing constraints.

2.3.4. Trajectory prediction and state estimation techniques

With high-speed interception required by the semi-automated foosball table, one method to compensate for unavoidable system latency that cannot be minimised by hardware and software

efficiency optimisations is to implement some predictive calculation which uses previous states of the object to predict future states, and thus aid the system in interception.

This section will discuss several works on predicting the trajectories of flying, spinning, bouncing, and rolling objects. In the context of automated foosball, some of them are more important than others. Flying objects are not as important as rolling, spinning or bouncing, however they have still been considered as they present a complex control problem in terms of interception.

One of the most common methods of trajectory prediction or estimation is to use Kalman filters (Brookner, 1998). There are many different types of Kalman filter and many different applications. For example Prevost, Desbiens, and Gagnon (2007) used Kalman filters for state estimation and trajectory prediction of moving objects using UAVs. Rosales and Sclaroff (1998) used an Extended Kalman Filter for predicting the trajectory and future position of moving humans using their current positions and velocities, and Z. Jing and Sclaroff (2003) later improved this work to more effectively segment moving human targets (foreground) from varying, textured backgrounds. Pistohl, Ball, Schulze-Bonhage, Aertsen, and Mehring (2008) used a Kalman filter to predict the trajectory of human hands for use in a brain-machine interface for control of a cursor in paralysed patients, and finally Singer (1970) used Kalman filters for tracking and motion prediction of many different types of military vehicles.

In the context of robotic control, Kalman filters have been used by several authors including (Frese et al., 2001) for a robotic ball catching robot using stereo-vision and a PC running some pre-calibrated algorithms, and (Weigel, 2005) who used Kalman filtering in their semi-automated foosball table for trajectory prediction of the ball.

Some variations or extensions of Kalman filters include particle swarm filters, Markov Chain Monte Carlo (L. Jing & Vadakkepat, 2010), Interacting Multiple Model (Janssen et al., 2012), and several others. Many of these have limitations such as high complexity or long convergence time (around 150 ms on a 3 GHz Pentium IV processor) as demonstrated by L. Jing and Vadakkepat (2010) with their interacting MCMC particle filter for tracking of manoeuvring objects. Ramakoti, Vinay, and Jatoth (2009) and Xia and Ludwig (2016) use particle swarm optimisation (PSO) techniques to aid Kalman filters in tracking objects in a video stream. The complexity and computational requirements of these methods do, however, appear to limit their functionality in real-time applications.

Cigliano, Lippiello, Ruggiero, and Siciliano (2015) implemented an eye-in-hand ball catching robot which caught 98% of flying and rolling balls, and 89% of bouncing balls. The authors believe their rebound model was the reason for the lower proportion of caught balls in the bouncing case. In this research the authors used a camera operating at 140 FPS, with a resolution of 512×368 pixels. The camera was mounted in the hand of their articulated robot arm which performed the movement and interception of the ball. The camera observed the ball for its entire flight time.

de la Malla and Lopez-Moliner (2015) discuss the role of prediction versus online vision for interception of a ball by humans. Prediction, as the name implies, involves estimating where the ball will go based on previous observation of the ball's path. Online vision is the continuous observation of the ball's position and matching the position of the interception apparatus (the hand in the case of humans catching a ball) to the ball's position. Trials were performed by a human interacting with a data capture device called CyberGlove which captured finger positions at 100 Hz. A visual stimulus of a ball being launched in a fixed parabolic motion was created, and then projected stereoscopically for the participants to observe. Their findings indicated that both predictive and online vision were utilised in human vision for effective interception of the flying ball.

In sophisticated systems, such as the human limbic system, predictive mechanisms and online vision work in tandem (Wolpert & Flanagan, 2001). Prediction may guide us as to the rough interception point of the ball, while online vision may help us narrow the interception window sufficiently that we can catch the ball. This seems like a reasonable method to adopt for high-speed ball interception in a semi-automated foosball table. As mentioned by Wolpert and Flanagan (2001), the human predictive models need to be continuously updated, however so this should be considered in the design.

2.4. Image and signal processing on FPGA

Arguably the most important element in a computer vision-based control system is the vision system. In order to achieve high speed control, a high-performance image processing platform must be used. A popular method for high performance image processing is to use a Field Programmable Gate Array (FPGA) (Bailey, 2011; Johnston et al., 2004).

The use of FPGAs for image and signal processing is quite well developed, and this is one of the recommended applications for FPGAs given by FPGA chip manufacturer Intel (Intel, 2018). However, some background will be given regarding the benefits of using FPGA rather than conventional image processing hardware such as desktop CPUs, GPUs or ASIC hardware. Additionally, a brief introduction to the fundamental theory of software and hardware-based image processing will be presented.

2.4.1. Software and hardware-based image processing

Most modern image sensors are either CCD (charge coupled device) or CMOS (complementary metal oxide semiconductor), however in machine vision applications, CMOS has some advantages over CCD such as the ability to be integrated into the same die for low cost of manufacture (Rowe, Rosenberg, & Nourbakhsh, 2002), and higher framerates (El-Desouki et al., 2009). These image sensors are used in image capture devices such as cameras, which are then connected to a processing unit, which performs the image processing operations.

A digital colour image such as that captured using a CCD or CMOS sensor is an array of light intensity values with a fixed data width (for example 8 – 12 bits per channel) which have passed through a colour

filter. Usually a red, green, blue (RGB) colour filter is used. 8-bit RGB in Bayer pattern format is a common configuration for CMOS and CCD sensors in robot vision (Gamal & Eltoukhy, 2005).

One 8-bit image of 640×480 pixels has 307,200 total pixels, each with either red, green or blue 8-bit intensity value. Each pixel provides an intensity value for one of the three channels – red, green or blue. An 8-bit image is converted from Bayer pattern format to a true 24-bit image using demosaicing via Bayer interpolation, however this is only done after the image is captured and transferred to the processing unit. After image transfer and demosaicing, the image is usually transformed into a different colour space for more convenient colour segmentation. The image is then filtered (segmented) according to the application requirements, and the required information (centroids, histogram, image entropy, etc) is output.

In most software-based image processing systems, the data is received from the camera via USB or Ethernet, stored in RAM and then accessed by a software application sequentially as the pixel values are needed for processing. The processing operations in software-based image processing applications is performed according to the following cycle - Figure 2-1:

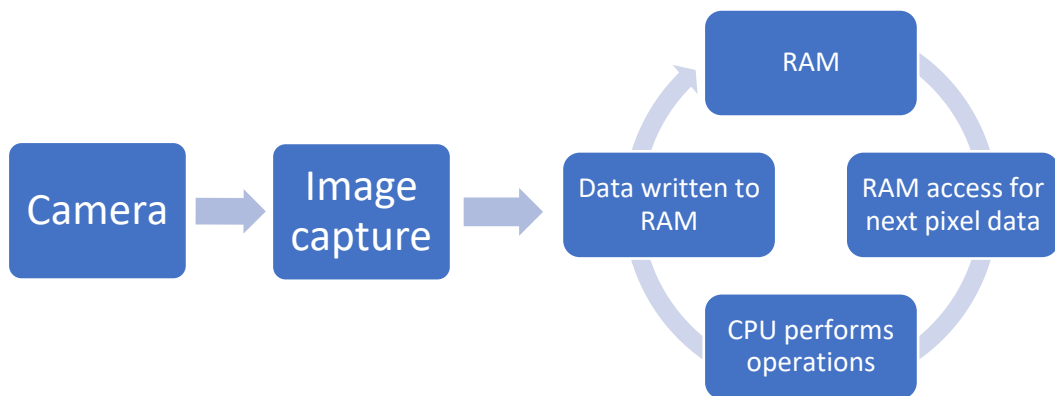


Figure 2-1 - Cycle diagram representing algorithmic image processing

In FPGA image processing, the most common arrangement is to stream the image data from the CMOS or CCD sensor via a parallel interface (typically at a rate of one pixel per clock cycle). As this data is received, the image data is streamed directly to hardware blocks implementing the processing operations. Operations are performed on the pixel data on-the-fly, without having to store the entire image in RAM before accessing it. Pipelining the process hugely reduces latency as the read/write operations are a large contributor to the total time taken by software-based image processing operations. The following block diagram represents the image processing data flow in an FPGA based system:

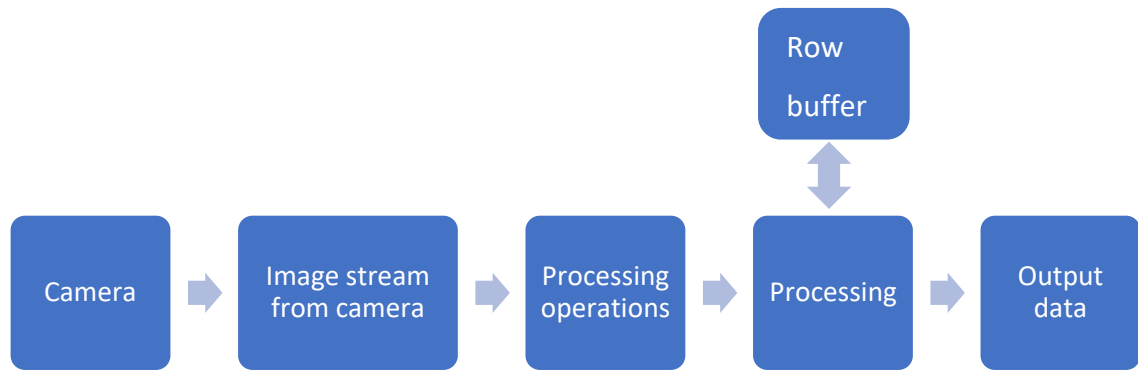


Figure 2-2 -Streamed image processing in FPGA hardware

With pipelined stream processing, data is continuously fed through the image processing operations. If local data is required, the required pixels are cached locally (in row buffers) using distributed RAM. Each operation runs in parallel with data streamed between operations at a constant rate. Using this method, operations can start their processing as soon as pixels start arriving; it is not necessary to wait until the previous operation has completed processing the whole image. This, combined with the reduced memory overhead significantly reduces latency, compared to software-based implementations.

2.4.2. Performance improvements

High performance in terms of framerate and latency is required if the vision system is to be used in a control system. Image processing on an FPGA can yield performance improvements of one to two orders of magnitude compared to CPUs and approximately double that of GPUs, for applications like 2D filters, stereo vision and K-means clustering (Asano, Maruyama, & Yamaguchi, 2009). Due to improved instruction efficiency, which the authors define as the level of pipelining and parallelism capabilities, FPGAs can achieve much higher performance than CPUs in benchmarks such as Prewitt edge detection, wavelet transform, and maximum filter (Guo, Najjar, Vahid, & Vissers, 2004). In a recent paper, (Safaei, Wu, & Yang, 2018) achieved an order of magnitude improvement using an SoC FPGA, versus a 3.4 GHz CPU running 16 threads for a background/foreground subtraction application. (Leeser, Miller, & Haiqian, 2004) discuss an implementation of stream-processing image data using an FPGA, with very low latencies – minimum of 250 microseconds for the first result of the filtering operations. The latency will vary depending on where the object appears in the image space, however. (Leeser et al., 2004) obtained a performance improvement of around 50 times for retinal vascular tracing and 20 times for particle image velocimetry, when compared with software implementations on a 2.6 GHz Xeon processor and a 1.5 GHz Xeon processor respectively.

In an industrial control problem involving an electrical drive system, the FPGA realisation was able to perform the execution of the control algorithm in roughly one quarter the time of a dedicated DSP controller (Monmasson & Cirstea, 2007).

All of these performance improvements indicate that FPGAs present a huge opportunity to replace CPUs or DSPs in computationally expensive tasks where parallelism can be exploited. Image processing in the context of robotic control is a perfect example of this.

2.4.3. Power consumption and embedded capabilities

Usually, high computational power is achieved at the cost of power efficiency. High performance CPU based computers generally consume large amounts of power, meaning that they are less suitable for embedded systems with either low overall power delivery capability, or limited power supply where the system is battery powered for instance. Therefore, a lower power computational unit with high performance capability is required. A key benefit of using FPGAs is their lower power consumption in signal and image processing, compared to competing platforms such as CPUs and GPUs. Low power consumption of the processing unit may enable high performance vision and signal processing to be implemented in embedded systems with limited power supply capacity.

Using an FPGA, high levels of performance can be achieved with as little as a 100 mW of power (García et al., 2014). As demonstrated by (Thomas, Howes, & Luk, 2009), in terms of random number generation, “the FPGA provides an order of magnitude more performance per joule than any other platform, and over 250 times that of the CPU”.

(Putnam et al., 2014) show that “a medium-scale deployment of FPGAs can increase ranking throughput in a production search infrastructure by 95% at comparable latency to a software-only solution. The added FPGA compute boards only increased power consumption by 10% . . . yielding a significant overall improvement in system efficiency”. System efficiency was defined as performance per joule of energy and was relative to other tested systems in the paper.

Given the low power consumption, and high performance of FPGAs they represent an ideal embedded processor. Honegger, Oleynikova, and Pollefeys (2014) implemented a stereo vision system performing disparity estimation on images of 752×480 pixels at 60 frames per second, all at a total power draw of 5 W. This is remarkably low, considering the high volume of data being captured and processed by the system.

Safaei et al. (2018) present a system based on a Zync-7000 SoC which separates the background from foreground of images with processing time of 0.96 s, a total cycle time of 4.18 ms and a data processing rate of 9,460 MB/s (26.2 GOPS). The total system power consumption was 1.747 W.

As new FPGAs come to the market, their power and performance characteristics continue to improve (Putnam et al., 2014). Overall, FPGAs provide extreme levels of performance without the power premium generally faced by serial processor architectures. Therefore, FPGAs represent an ideal low power embedded processor where high performance is required.

2.4.4. Programming and reconfigurability

A key advantage of FPGAs is that the designs contained within the chips are completely reconfigurable. This means that one FPGA chip can be used for a multitude of different applications. During the design process, the overall function of the chip can be completely changed which offers a lot of versatility. This high level of reconfigurability does come at some cost of programming complexity, which requires

modern synthesis tools to compensate for the high level of reconfigurability (Meeus, Van Beeck, Goedemé, Meel, & Stroobandt, 2012). Programming FPGAs requires a different way of thinking compared to algorithmic languages such as C (Mealy & Tappero, 2016). A hardware description language (HDL), such as VHDL, is used to describe hardware; this means that the resulting design is circuitry, rather than an algorithm which executes in sequential order. The timing in software is implicit in the sense that one function executes after the other. In HDLs, everything is concurrent, and timing and execution must be explicitly specified (Edwards, 2006), otherwise timing issues arise. Finally, debugging hardware designs is more difficult than software, despite the simulation tools that exist (Graham, 2001). Overall, hardware design requires a fundamentally different approach to algorithmic style of programming, and thus, is off-putting to some engineers who are unfamiliar with HDLs (Monmasson & Cirstea, 2007).

Difficulty in programming should not be of concern as there are many examples of difficulty in programming image processing algorithms on CPU or GPU (Asano et al., 2009). Additionally, “the reduction of the execution time of an algorithm in the case of a DSP implementation is only obtained by a long work of optimization of the corresponding assembler code. Such an optimization is no less consuming in terms of development time than the time needed for the design of an efficient architecture” (Monmasson & Cirstea, 2007).

Much improvement has occurred in the field of microcontrollers in terms of integrated development environments (IDEs), debugging, compiler efficiency and more. If similar effort is put into the development of programming and using FPGA technology, many of the current disadvantages of FPGA usage (long development cycle, expensive, conceptually difficult) could be significantly reduced (Monmasson & Cirstea, 2007; Putnam et al., 2014).

2.5. Actuator control and sensing on FPGA

Actuation and sensing are both very important in any control system. High speed and low latency are required in both actuation tasks and sensing in a high-speed control system. Therefore, a compute platform capable of performing the sensing and actuation tasks satisfying the timing and precision requirements of the control system is necessary.

While not as common as image and signal processing, actuator control in FPGA is an interesting application. Pratt, Willisson, Bolton, and Hofman (2004) discuss an FPGA based controller which inspects Hall effect sensor data and various other data, such as from a large number of analogue to digital (A/D) and digital to analogue (D/A) converters within a robot joint control system and performs actuator position control with a 1 KHz update rate. The controller was interfaced to a computer with a 400 Mbit/s bandwidth. The concurrent processing of the FPGA enabled all the data to be processed with minimal latency.

Tanaka et al. (2009) demonstrated the performance of a Xilinx Vertex-II Pro FPGA based PCI card in the control of a master-slave surgical robot with 12 degrees of freedom (DOF), where the control latency

was around 30 microseconds from master input to slave output. This meant that the time elapsed from the human operator performing an action to the robot initiating the mimicked movement was 30 microseconds.

Christopherson, Pickell, Koller, Kannan, and Johnson (2004) implemented a system where an FPGA is interfaced to a DSP via a 32-bit, 250 Mbit/s interface to control an unmanned helicopter (GTmax Research UAV) and a small, ducted fan UAV. In the GTmax, the FPGA was unable to outperform the previous software implementation due to the authors' use of an inferior GPS system which provided less accurate results than the unit provided with the GTmax from the factory, and therefore yielded higher control error. Despite the faster control loop achieved by the FPGA based system, the poorer quality GPS data resulted in greater error. The control system was, however, deemed adequate for the ducted fan UAV.

FPGA technology has improved dramatically since much of this research was published with modern SoCs achieving higher performance, greater FPGA-HPS communication bandwidth, lower power consumption, and greater FPGA resource availability. Additionally, tools for high-level synthesis have also been significantly improved (Meeus et al., 2012), and will likely continue to improve as more developers adopt the technology. These factors make FPGAs an ideal candidate for applications in high-speed control where numerous actuators and sensors are involved.

2.6. Conclusions

For the control of a high-speed vision-based system, in the context of the semi-automated foosball table, sufficiently powerful hardware is required to receive image data from a low-latency image capture device. The image must be calibrated such that the distortion introduced by the lens is removed (either before or after image processing operations). The image must be processed in such a way that the correct information is obtained – the ball x and y coordinates. Finally, the ball location information must be used in such a way that maximises the possible interception performance of the available mechanical system. The more latency that is removed from these processes, the higher the theoretical performance limit for the vision-based control system.

3. System overview, requirements and high-level architecture

This chapter will present the system performance and development requirements, as well as all the potential methods to meet the requirements based on the findings of the previous chapter. Finally, the high-level architecture of the complete system used for the development of high-speed vision-based control will be presented.

3.1. Temporal and spatial resolution requirements – Camera parameters

The temporal resolution required for the vision system must be sufficiently high that when the ball has been kicked by a human player, it can obtain 2 or more measurements (frames) of the ball's position before the ball reaches the module performing the intercept. The table playing field is 1200 mm in length and 693 mm in width.

Figure 3-1 represents a typical frame capture scenario with the ball being kicked straight forwards by one of the human opposition's foosmen. In Figure 3-1, x_1 , x_2 and x_3 denote individual frames captured, overlaid on top of each other, with x_0 being the rod which performed the kick. In x_1 , the ball's velocity is unknown. It may have just been kicked or it may still be stationary. In frame x_2 , we have frame x_1 as a reference of its previous position. So frame x_2 gives us an initial estimate of the ball's velocity, however in frame x_1 the ball may have still been in contact with the foosman, or the kick may have occurred between frames x_1 and x_2 , therefore frames x_2 and x_3 are the frames required to perform the velocity and trajectory calculation, while frames x_1 , x_2 and x_3 can all be used for the heading calculation. This is an important consideration for the required temporal precision.

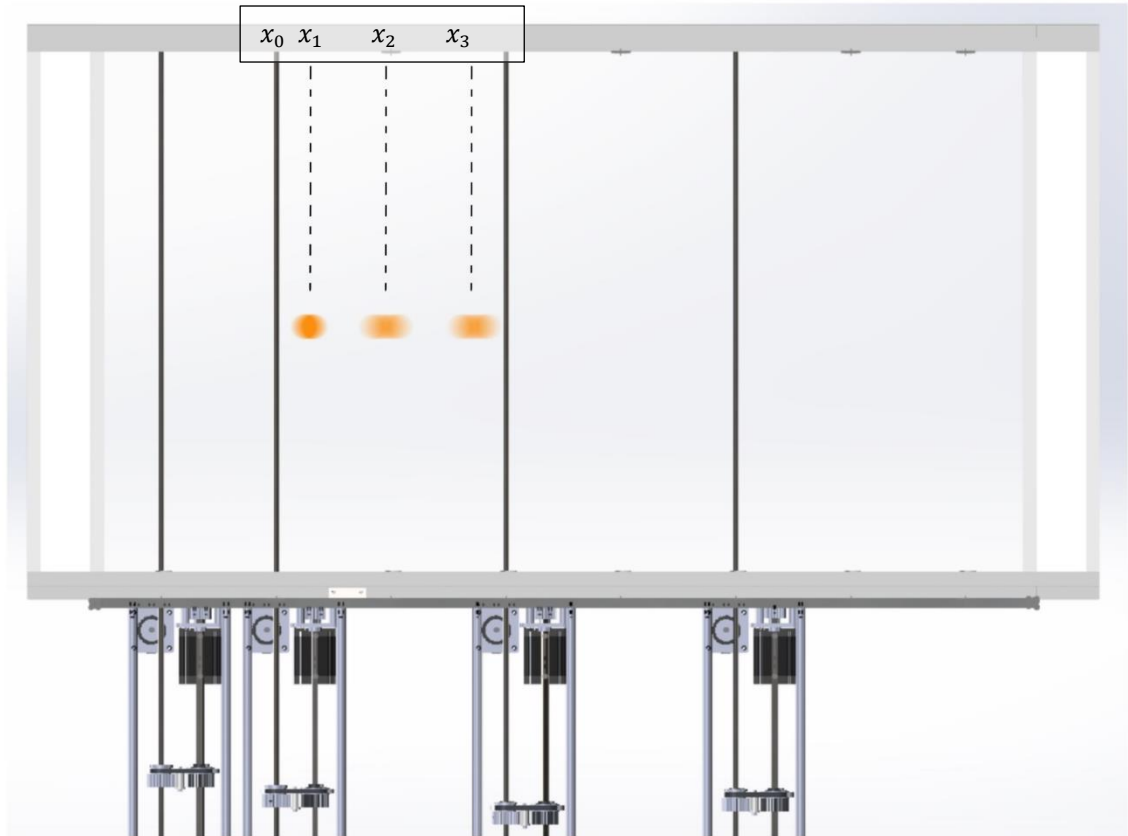


Figure 3-1 - Representation of ball motion blur on foosball table – birds-eye view with foosmen hidden from view

At a maximum speed of approximately 10m/s (10,000mm/s) it takes just over one tenth of a second for the ball to travel from one end of the table to the other, if the ball is travelling at maximum velocity. A camera capturing 60 FPS, for example, would capture 7 frames in that time. However, the ball only has a maximum of around 900mm of travel from the first rod to the opposing team’s goal. This means that a 60 FPS camera can capture around 4 or 5 useful frames in that distance. Therefore, the minimum camera framerate is 60 FPS, however a higher framerate is preferred if there is no detriment to other system performance attributes.

In terms of spatial precision, the foosball ball is approximately 35mm in diameter and the foosball table playing area is 1200 by 693 mm. If the image, at a resolution of 1024 × 768, is perfectly occupied by the table playing area, this would mean that each pixel corresponds to approximately 1mm. However, the image window does not perfectly align with both edges of the playing field as is shown in Figure 3-2. The aspect ratio of the playing field is approximately 7:4. The camera is constrained to a 4:3 aspect ratio due to the display input resolution requirements. The horizontal axis of the foosball table playing field has been fitted as closely as possible to the camera’s image width to maximise the number of pixels available for detection of the ball.

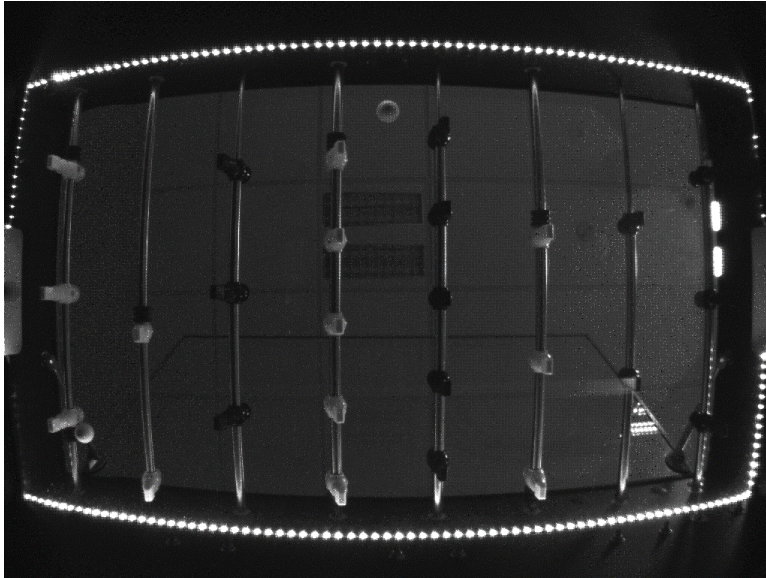


Figure 3-2 - Foosball playing field captured from below, using the DE1-SoC and D5M camera

From captured images of the playing field, the ball is, on average, 30 pixels in diameter, however some edge pixels will not be perfectly detected by the image processing algorithm, due to colour segmentation limitations.

By increasing the pixel density (the overall resolution at which the field space is captured) the number of pixels can be increased, which increases the accuracy with which the ball location is measured. Other authors who implemented object tracking systems have used camera resolutions such as 657×446 (Janssen et al., 2012), 384×288 (Weigel, 2005), or DVS resolutions of around 128×128 with highly structured or plain background conditions (Delbruck & Lang, 2013). Given that an increased number of total pixels increases latency for a given image transfer bandwidth, a resolution of 1024×768 pixels was selected. This resolution offered high precision and successfully fit the playing field into the image window (discussed further in later sections), while maintaining low latency and achieving the desired framerate.

At a resolution of 1024×768 , the precision achieved is more than enough to calculate the position of the ball to an accuracy of 1 mm. The accuracy obtained using 640 by 480 resolution would also be sufficient, provided the image window can capture the entire playing field. Therefore, the minimum required resolution is 640 by 480, and ball position detection to the nearest mm.

3.2. System response

System response is defined as the time from measurement system input to robot actuation execution. In the case of the foosball table, this would be the time between the ball having moved to a new location, that location to be captured by the computer vision measurement system, an output to be transacted by the FPGA, and then the foosman moved to the required interception position. Figure 3-3 represents the typical dataflow involved in one cycle.

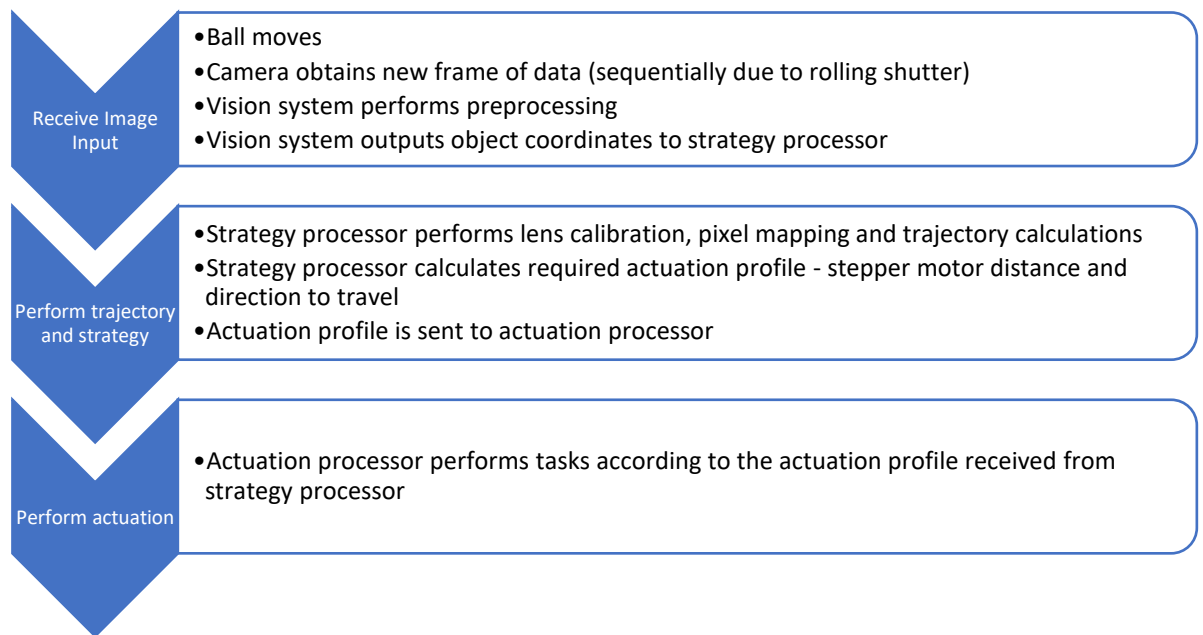


Figure 3-3 - One cycle of image capture through to system actuation response

From the above processes the following latencies are present:

1. Image capture latency
2. Image transfer latency
3. Data transfer latency – between processors
4. Processing latency
5. Actuation latency
6. Any latencies caused by an operating system running in the background

In order to achieve effective control, the overall system response time must be minimised. Each of the above latencies must therefore be minimised as much as is practically possible. The choice of hardware was therefore made on this basis.

3.3. Interception performance

For effective defence in a competitive game of foosball, a high proportion of successful interception is required for sub-maximal speed kick speeds. The goal for this work is therefore to achieve a 100% interception ability for straight shots (no angle) at sub-maximal kick speeds. This interception criteria is for each individual rod.

For shots with non-zero heading angle, relative to the length axis of the foosball table, a lower interception ability is required because using a highly effective interception strategy for each rod, their interception performance should compound. If one rod misses an interception, the subsequent rod(s) in the ball's path can attempt to intercept the ball. Therefore, only the straight-shot (zero heading angle relative to length axis) will be measured.

3.4. Power consumption

This research was partially focussed on enabling high performance computer vision in embedded applications. Power consumption was therefore a consideration for the processing component of the system.

In embedded systems, mains electricity is generally not the source of power for the system. Therefore, the compute unit must consume sufficiently low power to be powered by a battery. In embedded robotics, lithium-ion or lithium-polymer batteries are generally the preferred method of powering the robot or system. In the literature, a typical embedded image capture and processing power consumption of between 3 W (Fowers et al., 2007) and 5 W, was achieved (Honegger et al., 2014) and (Barry, Oleynikova, Honegger, Pollefeys, & Tedrake, 2015) all using FPGA based systems. Power consumption of 20 W was achieved using an ARM based “pushbroom” system (Barry et al., 2015). These power consumption values are for the image capture and image processing systems in the embedded applications only. Several other authors such as (Maxim & Zidek, 2012) and (Christopherson et al., 2004) report low power consumption for their image capture and processing systems, however they do not provide exact power draw figures. Although this is less of an issue for the mains powered semi-automated foosball table, it was nevertheless considered for this system.

Based on this, a maximum power draw of 20 W or less was specified for the entire compute system including image capture hardware, processor cores and peripherals.

3.5. Modularity and versatility

Reusability of code and of designs within FPGA is a very important method to improve efficiency of developing FPGA based solutions. Writing VHDL code in a modular, reusable way is important to enable efficient scaling or adaptation of one design to suit another application.

The importance of modular design cannot be stressed enough for VHDL as it is a key factor in what makes FPGA a powerful tool. For example, one can write a piece of VHDL code to perform some function (for example actuator control) within the architecture of the top-level entity; this is somewhat equivalent to the main loop in a C program. This is an acceptable thing to do from a language perspective, however if that same piece of code was created as a standalone module, it could then be instantiated multiple times, to perform multiple instances of the same function; much like declaring a function in a C program. This takes advantage of the inherent parallelism of FPGA, where the number of instantiations of these reusable modules that can be used in a design is limited only by FPGA resources and I/O. In many designs, the resources available will far exceed the resources required. This indicates that FPGA enables fairly easy scaling of designs.

In the context of robotics, automation, embedded systems and other applications, this is an extremely useful tool. In many embedded systems, the design requirements change throughout the project. The

ability to reuse modules or simply add more of them to the design is extremely useful, especially for things like actuator control, I/O, and communication interfaces.

A useful tool developed by Intel for FPGA development is the Qsys tool, which is part of the Quartus software package. Qsys enables creation of various IP based code using a GUI to represent all the connections and IP components within that design. Reusability of designs is greatly increased when the designs are created using Qsys. The use of this tool will be discussed in later sections.

Parameterisation of actuator control modules is required so that this actuator framework can be adapted to work with a variety of actuators. Given that the module is designed to control stepper motors, the acceleration, maximum speed, and deceleration characteristics of the motors should be easily adjustable such that any stepper motor can be substituted, and the motor control modules easily tuned to work effectively with the new motor.

Finally, the motor control code should be written such that transferring from one FPGA to another is a simple process of assigning new pins for clock, reset, IO and communication bus. The HPS is not required for control of stepper motors as it is a fully hardware-based operation, therefore it is not included in this requirement. Additionally, a Nios II soft core can replace the functionality of the HPS, provided a slightly lower level of performance is tolerable for the design.

3.6. Minimal latency from distribution of processes

In some early work, (Andersson, 1990) used custom hardware and communication interfaces in a stereo vision-based system to minimise system latency in a robot control application, mitigating the effect of their distributed system. They reported an overall control system response time of 32.2 ms.

As was seen in some cases in the literature, distribution of the design resulted in communication latencies compounding in the systems, decreasing overall responsiveness (Čížek, Faigl, & Masri, 2016). The authors discuss vision-based navigation systems utilising different image processing hardware. One key observation is the variation in the T_{sys} latency value reported for different systems. T_{sys} is defined as the latency caused by the operating system being used, including things like scheduling, communication and data transfers. Figure 3-4 shows the different proportions of the total latency caused by T_{sys} – the red parts of the bar chart.

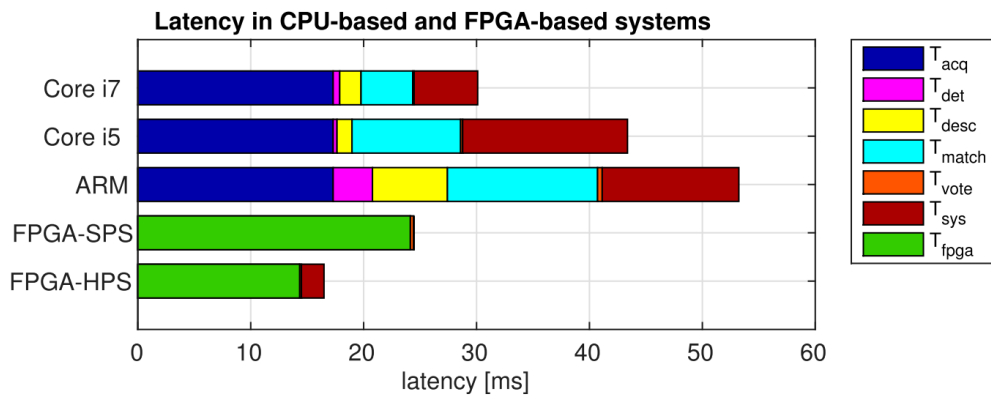


Figure 3-4 - Latencies present in systems tested by Čížek et al. 2016

As can be seen in Figure 3-4, the core i7, core i5 and ARM based systems all show a very high proportion of system latency, T_{sys} , relative to total latency. One reason for this is that these systems are all subject to communication delays, because the systems are distributed. T_{sys} in the FPGA based systems is much lower, both proportionally and overall. An interesting observation is that the FPGA-SPS system with the soft processor core implementation has lower system latency than the FPGA-HPS using the hard processor core. This is likely because the soft cores are implemented in FPGA logic, enabling much more efficient communication between the FPGA and the soft core. The FPGA-HPS system must transfer data between the FPGA and HPS over some communication interface (over several clock cycles), thus increasing system latency. However, given the much higher performance of the HPS, the FPGA-HPS system still provided the lowest overall latency.

Figure 3-5 represents the data path within a distributed, CPU based system, demonstrating the sources of latency generally found in distributed systems.

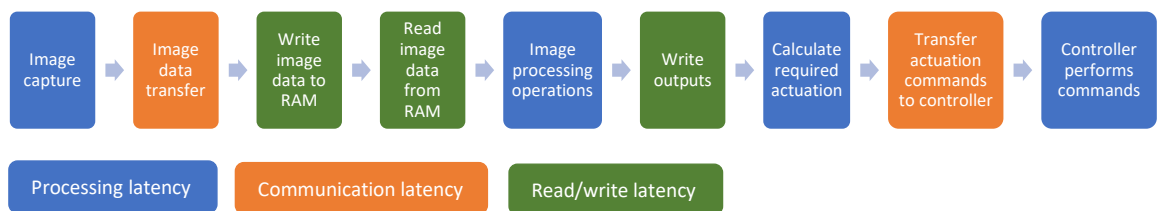


Figure 3-5 - Flowchart representing the data flow, communication, and data transfers in distributed PC based system

As can be seen in Figure 3-5, there are several instances where, given a sub-optimal communication or data transfer protocols, significant latency may compound in the system. These instances are the image data transfer, RAM read/write operations, and the transfer of actuation commands to the controllers. Each of these transfers take time and occur sequentially, therefore adding latency to the overall system pipeline, and reducing system responsiveness.

Based on these observations, a strict requirement for this system was to either minimise distribution of processing operations, process data at the source to reduce the volume of data that needs to be transferred, or to use a communication interface with low enough latency to not adversely impact system performance. A combination of these is also possible.

3.7. Reconfigurability

As was stated previously, reusability of code is important to enable simple scaling of the design. However, reconfigurability is also very important in the design of vision-based control systems. Reconfigurability is the ability to change the function of the hardware at a lower level than changing the code running on the hardware as is done with CPU based systems. A desktop computer or microcontroller represent systems with low reconfigurability. The internal logic and memory elements of the processors in these systems cannot be changed easily.

Reconfigurability is the enabling mechanism to achieve scalability or adaptability of a design. Therefore, the hardware used must be reconfigurable. An FPGA meets this requirement perfectly. The ability to completely change the function of the design without physically changing any components of the FPGA based system is the essence of its versatility.

Microcontrollers and CPUs are reconfigurable in the sense that the code they run can be changed. However, fixed attributes such as the number of hardware timers, or the processor data width cannot be changed without physical hardware changes. Additionally, resources like hardware timers that are not used in a design cannot be reconfigured for other purposes, or if the processor data width is surplus to requirements, it cannot be decreased in software to improve efficiency or minimise resource consumption, to free up resources for scaling of other aspects of the design.

3.8. Debuggability/traceability

In order to make the system easy to debug and improve the development experience, some traceability features are imperative to the design. These are as follows:

1. Real-time output of the image on a screen

This is the most effective method of debugging a vision system in real time. Without a real-time display of what the vision system is 'seeing', it is very difficult to determine the cause of problems.

2. Real-time output of the segmented object (tracked object) on the screen overlaid

In colour segmentation, it is very useful to see the pixels that have been "accepted" by the segmentation algorithm, also in real time, overlaid over the main image stream. This is so that if lighting parameters change or some other variation causes the colour segmentation algorithm to fail it is easy to detect the symptom and potentially the cause as well.

3. Simulation debugging of FPGA design

Debugging FPGA is typically quite difficult as it is hardware all running concurrently so detecting the source of problems can be a challenge as the algorithmic style of debugging (for example the use of breakpoints) is not applicable in FPGA. Simulation of hardware generally takes care of this; however it can still be more difficult or time consuming than with microcontrollers.

Hardware level simulation is also very slow, because the computer must simulate all the concurrent hardware interactions. This is particularly noticeable for image processing where many hundreds of thousands of clock cycles are required to process an image.

4. Source level debugging of the HPS C code, memory and registers

As with most embedded systems running microcontrollers, source level debugging is necessary. The fact that source level debugging is ubiquitous means that its necessity need not be justified.

5. Master access to the FPGA-HPS and HPS-FPGA bridges

A key technology utilised by some authors is a high-bandwidth communication interface between the FPGA and the ARM core. This interface is very powerful, however master access is required if it is to be utilised properly. This means access to registers and mastering transactions from a host PC is required for proper development.

6. A communication interface into the FPGA to provide debugging commands and debugging outputs to the user

Like any microcontroller application, user input into the system and simple diagnostic user output is required. This is obviously extremely important.

7. Soft and hard resets into the FPGA, HPS and image acquisition hardware

Once again, this is an important feature in any system. The soft resets are useful for simple debugging and the hard resets are necessary for global reset of all variables, logic elements and processors, returning the system to a rebooted state without needing to turn off mains power.

3.9. Potential methods to achieve outcomes

Based on the literature, there are many different technologies, algorithms, methods and configurations possible for the problem of low latency control of a robotic system utilising computer vision as the main sensor. Each technology and configuration of the surrounding algorithms and methods have their individual advantages and disadvantages. The following section will discuss some relevant configurations of the available technologies to address the problem of high-speed, low-latency robotic control with computer vision as the measurement system, as well as some of their limitations. Finally, the proposed method will be outlined and discussed with regard to how it can meet the requirements laid out in the previous section.

3.9.1. CPU or GPU based systems

One popular method to achieve high speed image processing-based control systems is with a distributed control platform where a standard computer running Windows, Linux or Mac is the compute unit performing the image processing algorithms on its CPU and/or GPU. Several authors have used this general approach method (Asfour et al., 2006; Behnke, Egorova, Gloye, Rojas, & Simon, 2004; Cigliano et al., 2015; Delbruck & Lang, 2013; Janssen et al., 2010; Padon, 2003; Weigel, 2005).

Another option is to use a microcontroller or microprocessor as was done by (Barry et al., 2015) who tested and compared two realisations of autonomous obstacle avoidance systems using stereo vision. The first system was an ARM only system termed “pushbroom”. The ARM processor was responsible for the image capture, image processing and the flight dynamics and control.

Several variations of the general CPU based framework are possible including (but not limited to) the variations shown in Figure 3-6.

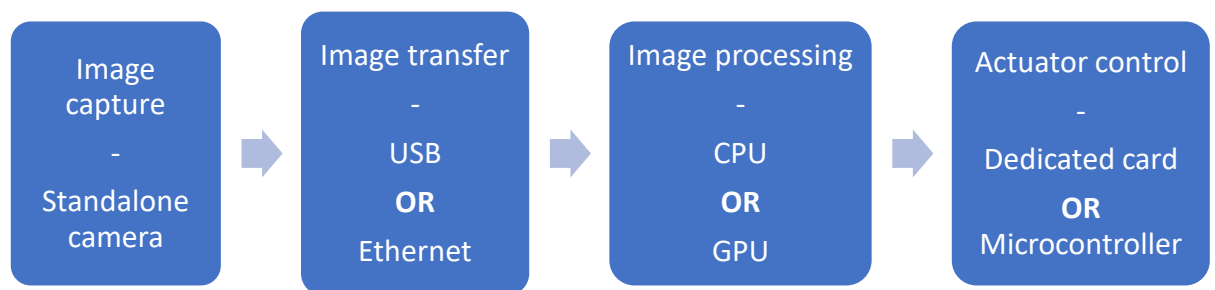


Figure 3-6 - Possible configurations of PC based image capture and actuator control systems

Figure 3-6 represents eight possible combinations of PC based systems. Variations of image transfer protocol, image processing hardware, and actuator control methodology are possible.

While some authors who used these systems have achieved low image processing latency, or even low overall measurement system latency, they all still involve the following problems:

1. The actuator control on all these systems is still performed by a separate control unit which must communicate with the compute unit.
2. The systems all require sequential transfer of image data and writing of image data to system memory, before any image processing operations can be performed.
3. PC's generally have very high-power consumption per unit performance, compared to other available technologies
4. PCs which are powerful enough to perform high-speed image processing are not very portable or capable of being an embedded processing unit. This is generally because PCs capable of such tasks usually contain powerful CPUs and GPUs. These consume large amounts of electricity and generate excessive heat.

3.9.2. ASIC or custom hardware

Many different works include custom hardware to achieve low latency control of robotics. These works include the use of dynamic vision sensors (DVS), custom chip arrays, in-house manufactured circuit boards containing microcontroller chips or other processing hardware, and other non-off-the-shelf solutions.

DVS sensors have been used for low latency control or feedback by several different authors. These include robotic pole balancing (Conradt, Berner, Cook, & Delbruck, 2009), a low latency, low CPU usage robotic goalie (Delbruck & Lang, 2013), low latency object localisation (Censi, Strubel, Brandli, Delbruck, & Scaramuzza, 2013), and micro-robotic haptic feedback (Ni, Bolopion, Agnus, Benosman, & Regnier, 2012). DVS sensors will likely improve in the future because of the unique benefits they offer, such as extremely low latencies. However, they are still generally low-resolution and are better suited to highly structured environments where the only moving target is the object being tracked and the lighting is carefully controlled.

Another option for parallelised image processing in a robotic control application is to use a programmable array of processors such as a massively parallel processor array or a custom designed board with numerous DSP chips or high-performance microprocessors such as the ARM range. Microprocessors offer high performance per unit power, W , or chip area, mm^2 . This can yield quite good performance as was shown by Andersson (1990) where the author used a custom compute platform based on what they called the TRIAX (an image processing platform) for stereo vision capture and pre-processing, and a 20MFLOPS JIFFE (processor for robotic control) for processing 3D data and performing the required calculations to control an articulated robot with up to 6 degrees of freedom. There were numerous limitations with this approach including the low spatial resolution where only 2000 pixels worth of 3D data were processed per frame; 100 lines of 20 pixels per line. However, even with this crude (by modern standards) approach, they could effectively control an articulated robot.

3.9.3. FPGA

FPGA for image processing

Several authors have succeeded at controlling various dynamic systems with FPGA based image processing as their measurement systems (Christopherson et al., 2004; Honegger et al., 2014; Linares-Barranco, Gomez-Rodriguez, Jimenez-Fernandez, Delbruck, & Lichtensteiner, 2007; Wei, Byung Hwa, Larson, & Voyles, 2005).

Some limitations with these implementations were that the FPGA was often responsible only for the image processing, while other aspects of control like actuation, strategy or sensing were left up to a conventional microcontroller which then had to pass the data to the FPGA. Despite this, some authors were still able to achieve effective control.

FPGA with DVS sensor

(Linares-Barranco et al., 2007) use an Address-Event Representation (AER) sensor connected to an FPGA, which also communicates with a Cygnal 80C51F320 microprocessor. The FPGA receives data from the AER sensor and analogue measurements from the microprocessor and performs control of the robot. The AER sensor has an array size (resolution) of 128x128.

FPGA SoC

The second system tested by (Barry et al., 2015) was an identical UAV as the ARM system tested by Barry et al., however the vision and control was performed by an FPGA SoC. FPGA SoC technology was also used by (Maxim & Zidek, 2012), who discussed an FPGA SoC system where the audio or video stream could be used for control of UAVs or UGVs. The ARM cores on the SoC used by the authors was programmed in “bare-metal” mode which means that the code is running on the lowest level possible, without an OS running in the background. One major limitation with their work is that they used USB webcams as their image input stream to the FPGAs. This would likely have added a large amount of latency to the systems. This is not an issue of SoC design in general, rather just a limitation with this particular design decision.

(Safaei et al., 2018) use an SoC FPGA for hardware acceleration of foreground and background segmentation in live video stream. They achieved outstanding performance at very low power consumption. Their SoC-based system was capable of processing an image of 1920 by 1080 pixels in 4.18 ms, at a total computation power consumption of 1.747 W.

Another method of using SoC FPGAs is to run an OS on the HPS portion of the SoC and to use the FPGA as a simple hardware accelerator as done by (Maxim & Zidek, 2012). However, running an OS generally adds latency to a system as the OS adds some protection layer so that the developer cannot cause sections of memory to become corrupted. The OS must read and write to memory through this protection layer which adds some latency to the system. Additionally, scheduling requirements of an OS can add some latency to the system.

Using an FPGA SoC, full advantage can be taken of the system when the FPGA performs image pre-processing and the HPS portion of the SoC is responsible for efficiently written bare-metal code such as floating-point operations and other calculations of that nature.

FPGA with parallel soft cores

Another possible use of FPGA technology is to implement numerous soft cores in FPGA logic. Intel/Altera provide a Nios II RISC core with varying performance levels and resource consumption. The higher-performance cores consume more resources than the less powerful cores. The Nios II cores would, in this case, take the place of the HPS for floating point calculations, communication, and other such software-oriented tasks. Depending on the complexity of the application, a single Nios II core or many cores could be implemented.

(Baklouti & Abid, 2014) tested how well scaling of the soft-cores on FPGAs performs, finding that as the core count increased, the amount of processor time consumed by communication increased such that after 32 cores, the improvement of increasing the core count to 64 cores was only very slight. The authors used 3 different operations as performance benchmarks, reporting computational efficiency relative to the number of soft-cores implemented. The 3 operations were Finite Impulse Response filtering, Spatial Laplacian filtering, and Matrix-Matrix Multiplication. For all 3 applications, increasing the core count did yield a performance improvement, however as more cores were added, processing efficiency declined due to increased communication overhead. The lowest marginal degradation in efficiency was seen in the Matrix-Matrix Multiplication. There are still some applications where multiple soft-cores can outperform GPU's, due to the soft-cores' ability to communicate with one another.

In the semi-automated foosball table, it could be useful to implement one Nios II core per actuation module to perform all the necessary calculations. This way the modules could all operate completely asynchronously.

3.9.4. Summary of potential methods

From chapters 4 and 5, several possible systems are presented which could be used to solve the problem of high-speed vision for low latency control of robotics. The potential system configurations are as follows:

1. PC based system where image capture is performed by camera connected via gigabit ethernet, image is processed on a GPU, and actuator control is performed by a dedicated actuator control card
2. Dynamic Vision Sensor based system incorporating either an FPGA or a PC to analyse the DVS output data, and the actuation is still performed by a dedicated actuator control card
3. FPGA – PC based system where image capture and pre-processing are performed by FPGA, processed data is passed to PC, and PC gives actuator commands to dedicated actuator control card which then performs actuation
4. FPGA – PC based system where image capture and pre-processing are performed by FPGA, processed data is passed to PC via USB or ethernet, PC performs any calculations necessary and actuation data is returned to FPGA which performs the actuator control
5. FPGA – SoC system where FPGA does image capture and pre-processing, processed image data is given to HPS via high-bandwidth interface, HPS returns actuation commands to FPGA via high-bandwidth interface, FPGA performs actuator control
6. FPGA – Nios II only system where image capture and pre-processing are performed by FPGA, processed image data is sent to Nios II via shared memory, Nios II returns actuation commands to FPGA via shared memory, and FPGA performs actuator control

3.9.5. Proposed method

The method proposed to satisfy all requirements detailed above and meet the outcomes for the project is as follows:

An FPGA-SoC system where:

1. Image data is streamed directly off the image sensor into the FPGA logic
2. Image pre-processing is performed on the streamed data by the FPGA
3. Processed image data (image information such as centre of gravity and object size) is passed directly to the HPS component of the FPGA via high-bandwidth FPGA to HPS bridge
4. HPS performs lens distortion mapping, trajectory and actuator profile calculations
5. HPS passes actuation commands back to the FPGA via the high-bandwidth HPS to FPGA bridge
6. The FPGA performs all actuation concurrently with independent actuator control modules

The proposed hardware to be used is the DE1-SoC development board by Terasic.

By implementing the above system configuration, low latency control of a mechatronic system should be possible using computer vision as the input and/or reference signal. In other words, the above system should enable high-speed visual servoing, without requiring high power consumption processors or custom ASIC boards.

The FPGA meets the requirements for low latency image processing capability, scalability, and reconfigurability. Using the ARM core running bare-metal C code with the high-bandwidth communication interfaces should minimise algorithmic processing latency and communication latency. Finally, using the FPGA for the control of the actuators should minimise latency and timing jitter that would arise if all the actuators were controlled by a microcontroller.

3.10. High-level complete system architecture

3.10.1. Semi-automated foosball table control system

In this research, a semi-automated foosball table shown in Figure 3-7, in which a human opponent plays against the table, is the test platform for the vision and control schemes. Therefore, the hardware and software discussed will be in the context of their usefulness in this application.



Figure 3-7 - Semi-automated foosball table

The table used in the experimentation is semi-automated. On one side, the human players have been replaced with a mechatronic control system comprising actuators to slide and rotate the metal rods. The rods are controlled by 8 stepper motors: 4 for the linear (sliding) movement of the rods, and 4 for the rotation (spinning). This is shown in Figure 3-8.

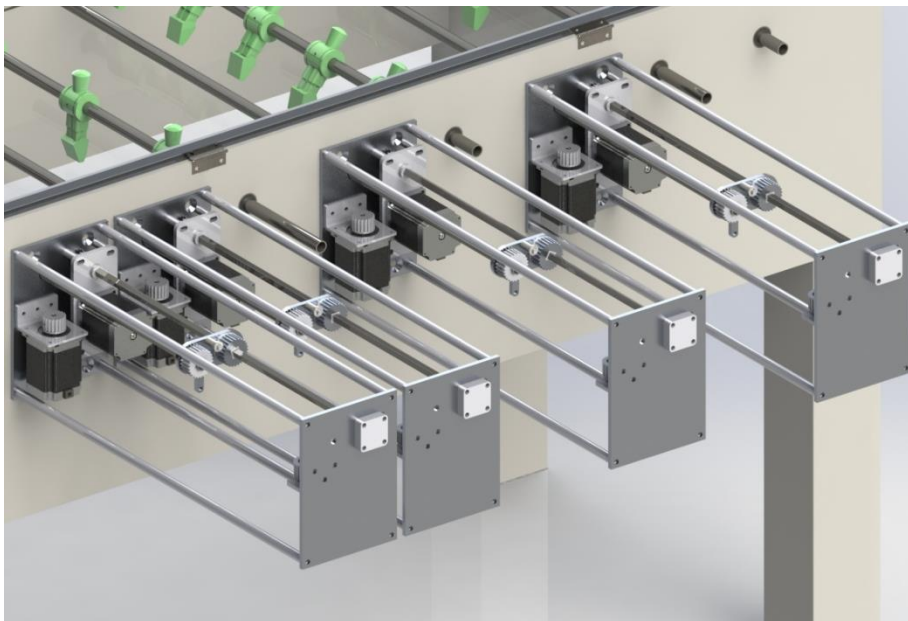


Figure 3-8 - CAD model of the automated actuation modules

The camera faces upwards from beneath the table and tracks the ball through the base of the table which is made of glass. The human player plays against the automated control system; both try to defend their own goal and shoot into the opponent's goal. The control system, therefore, is required to track the position of the ball with sufficient speed, low latency, and sufficient resolution to accurately

calculate where the ball is on the playing field. The temporal and spatial resolution requirements were specified in the first section of this chapter.

3.10.2. Compute subsystem

The following block diagram, Figure 3-9, represents the overall compute/control system hardware and sensors:

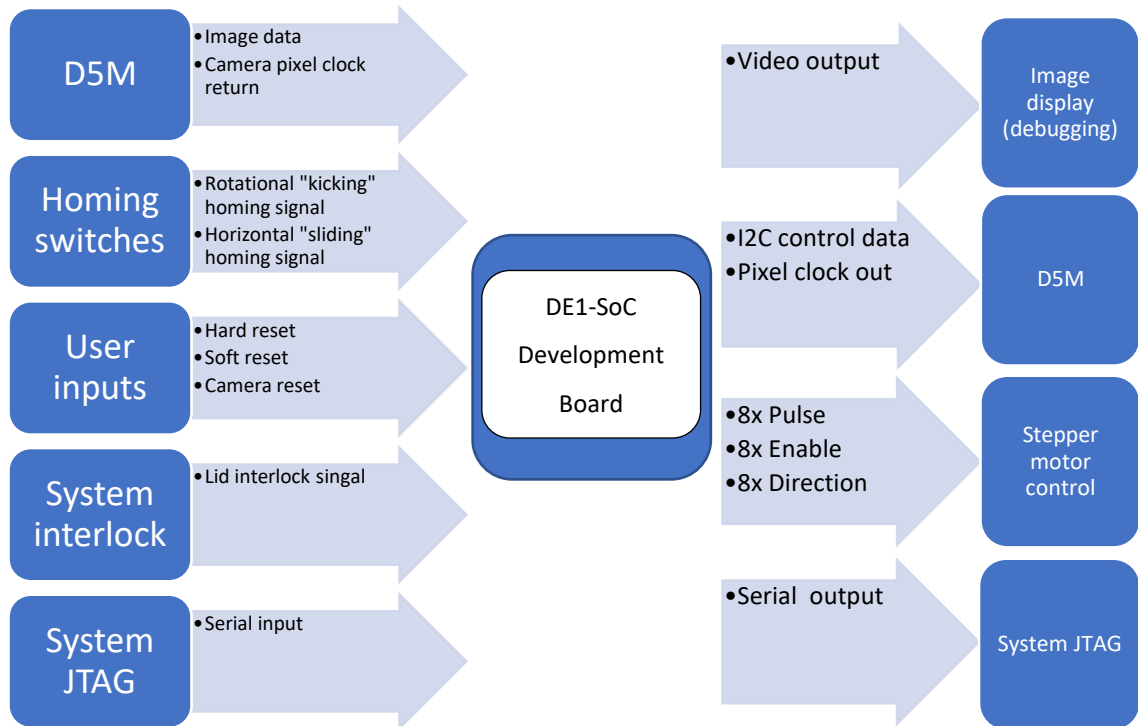


Figure 3-9 - Compute system input and output signals

The DE1-SoC is a development board which combines an 85,000 logic element FPGA chip with a dual core 800 MHz ARM A9 microprocessor on the same die, interconnected via 3 bridges known as the HPS to FPGA bridge (H2F), the FPGA to HPS bridge (F2H), and the lightweight HPS to FPGA bridge (LWH2F).

The FPGA component is responsible for:

1. Configuring the camera module via I2C
2. Providing clocks to all system components
3. Receiving the pixels coming from the camera
4. Performing Bayer interpolation
5. Colour space conversion
6. Performing colour filtering operations
7. Performing morphological erosion and dilation
8. Performing connected components analysis
9. Displaying image stream on screen via VGA

10. Passing detected object coordinates (x_{min} , x_{max} , y_{min} , y_{max}) and stepper motor positions to HPS via AXI bridge
11. Receiving stepper motor control data from HPS via AXI bridge
12. Performing stepper motor control
13. Overall accepting any user input signals including JTAG, resets, and interlock

The HPS (ARM core) is responsible for:

1. Receiving object coordinates and stepper motor data from FPGA
2. Use object coordinates (past and current) and stepper motor data to calculate:
 - a. Trajectory for the ball
 - b. Interception points for the foosmen rods
 - c. Distances required for each foosman to reach the interception point for its respective rod
 - d. Which foosman should take the intercept on each rod
 - e. Number of steps required for each rod and which direction to travel
3. Passing the above calculated values to the FPGA
4. Overall accepting any user inputs including JTAG, resets, and other debug related signals

4. Mechanical subsystem

4.1. Introduction

This chapter will discuss in detail the overall mechanical system. This includes the dimensions, materials, similarities to official USTSA foosball tables, and the differences. Where relevant to the thesis topic, the engineering design choices for the mechanical subsystem will be justified.

4.1.1. Custom aspects of foosball table design

There are a few major design differences between this table and regular foosball tables. These things include:

1. The automated modules which take the place of one human player/team
2. The glass base for the vision system to track the ball from beneath
3. The glass lid, to prevent injury to the human players during operation
4. The LED lighting around the outside of the playing field, to illuminate the ball

Automated modules

The automated modules shown in Figure 4-1, perform the kicking and sliding movement that a human player would normally perform.

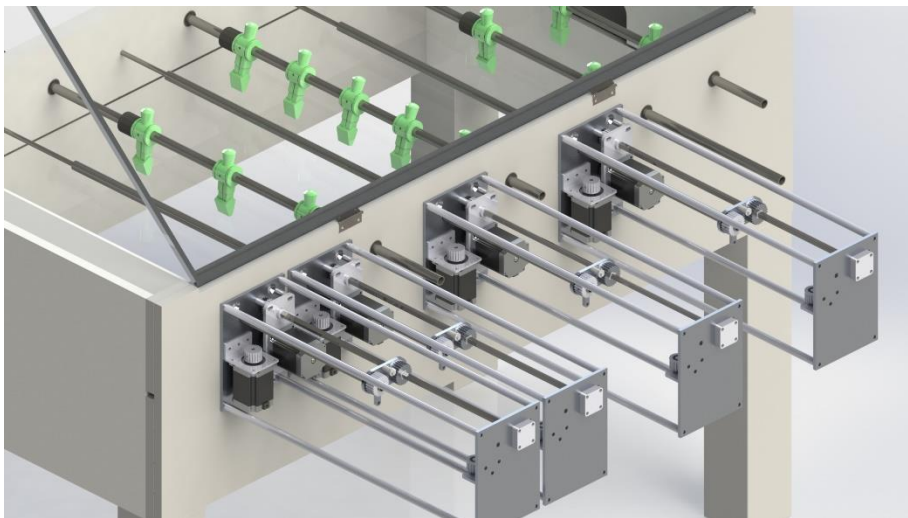


Figure 4-1 - Render of automated modules of CAD model

The modules are actuated by Nema 23 stepper motors with peak torque of around 130Ncm as seen in Figure 4-2.

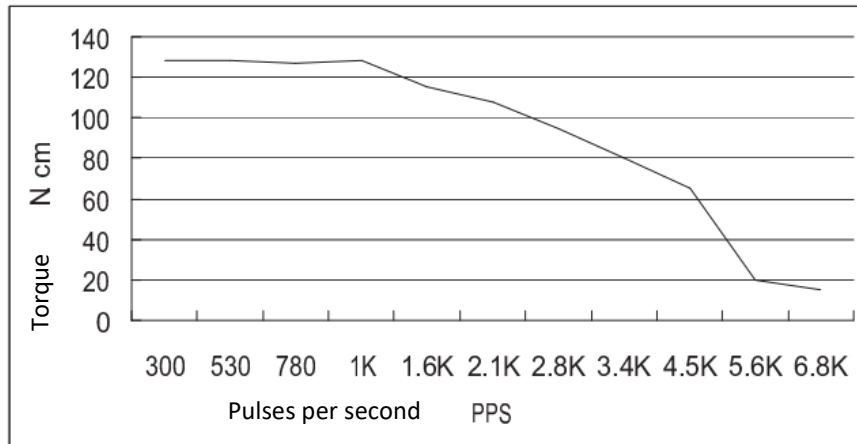


Figure 4-2 - Pull out torque curve of Nema 23 bi-polar stepper motor - (Pololu, 2018)

The purpose of these actuation modules, shown in more detail in Figure 4-3, Figure 4-4, and Figure 4-5, is to intercept the ball when the opposing (human) team attempts to play offensive shots such as forward passes and scoring shots, and to play offensive shots for the autonomous team.

There are two main movements required by the actuation modules. These are a rotation of the rod on which the foosmen are mounted, and a linear sliding motion of the same rod. These movements are achieved using the stepper motors connected with belt drives to the main rod. One stepper motor performs the rotational kicking movement, while the other performs the linear sliding movement.

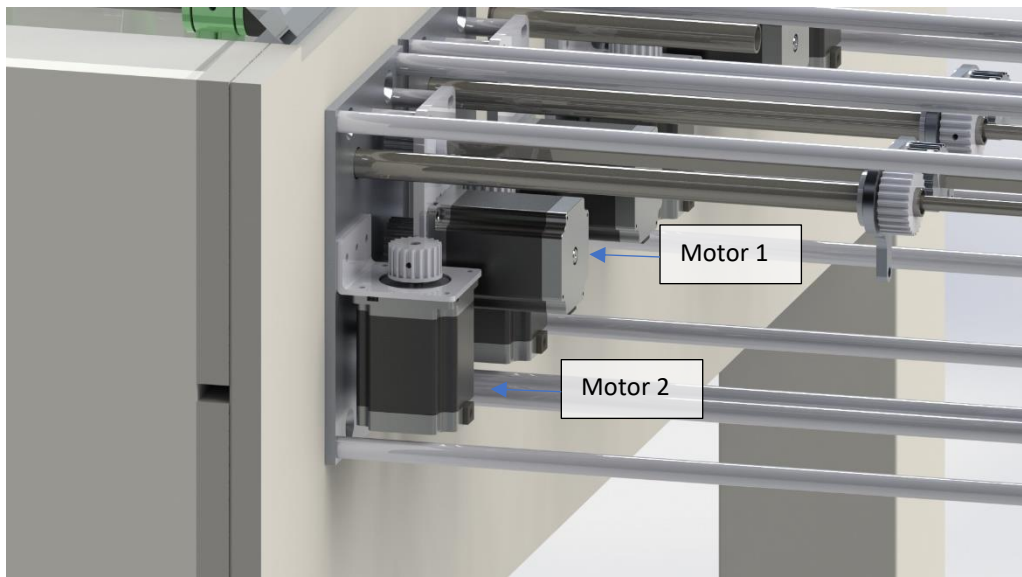


Figure 4-3 - Render of actuation modules

The motor mounted horizontally, labelled “motor 1” above as a CAD model in Figure 4-3, is connected via a two-stage belt arrangement to the main rod. This motor is responsible for the kicking motion. The belt arrangement for motor 1 is shown in Figure 4-4. The right hand image in Figure 4-4 shows belt assembly 1 behind the mounting plate shown on the left-hand side of Figure 4-4. Belt assembly 1

transfers the drive from the horizontally mounted stepper motor to the square shaft. Then belt assembly 2 transfers the drive from the square shaft to the main shaft on which the foosmen are fixed. There is a 1:1 gear ratio from the stepper motor to the main shaft.

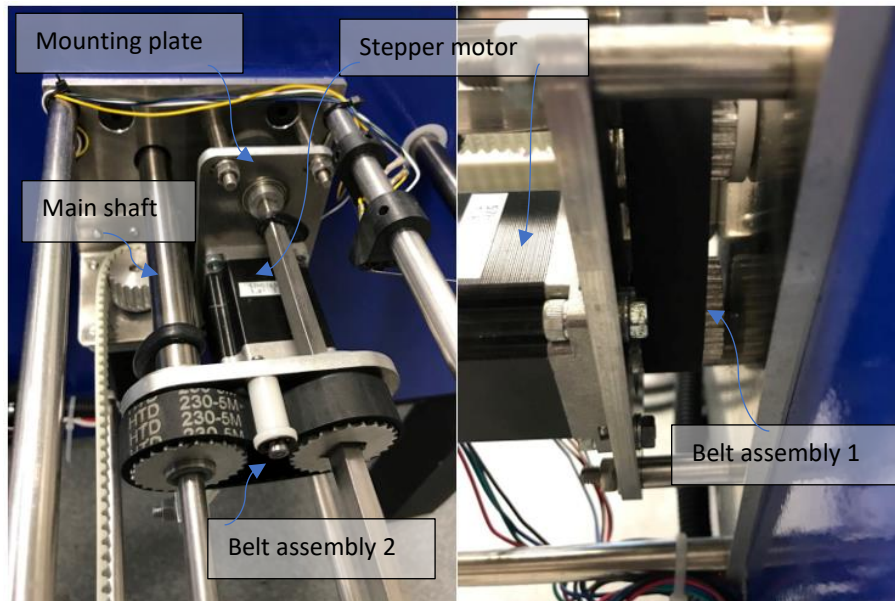


Figure 4-4 - Foosball table actuation module - rotational drive assembly

The vertically mounted motor, labelled motor 2 in the CAD model Figure 4-3, is connected to the shaft via a belt and connector shown in more detail in Figure 4-6 below. The system has been designed such that the rotational and linear motion is combined into the same shaft, much the way a human would actuate the shaft to move the foosmen. The ratio of steps to distance for the linear motion is 25mm per 100 steps. There is some backlash (around 2-3 mm) in the linear drive system, however this is easily corrected for with a small offset after the homing sequence. The stepper motor shown below in Figure 4-5, drives the belt which is connected to the main shaft via the connector shown in Figure 4-6.

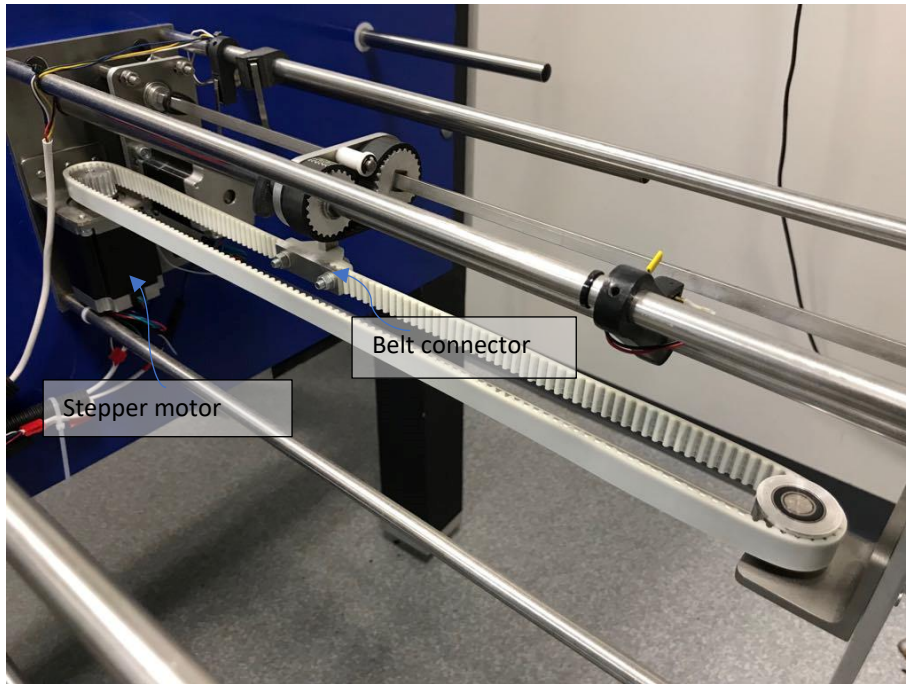


Figure 4-5 - Foosball table actuation module - linear drive assembly

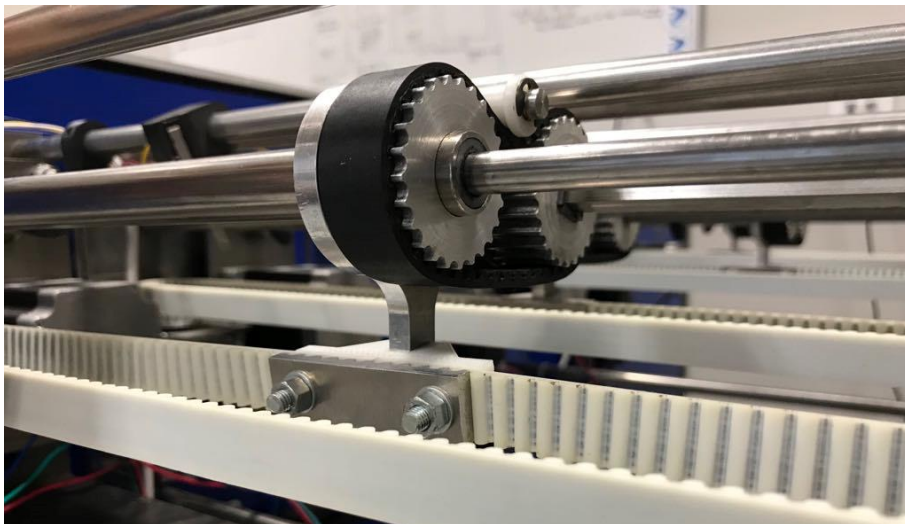


Figure 4-6 – Close-up of belt connector on actuation module

Stepper motors were used so that after initial setting of a datum (homing) was performed, high levels of accuracy and repeatability would be possible without the use of encoders. Additionally, they offered high levels of torque without the need for reduction gearboxes.

Toothed timing belts were used to ensure no slippage, and minimal backlash were achieved in the drive assembly for both linear and rotational motion.

Glass protector and playing field

To protect the human players from the automated modules, the table was built with a glass lid. The system is interlocked such that the motors will only be actuated when the lid is completely shut. This

system disables the motors when the lid is open as the motors are powerful and could cause harm. Figure 4-7 shows the base, interlock switch, and lid on the system.

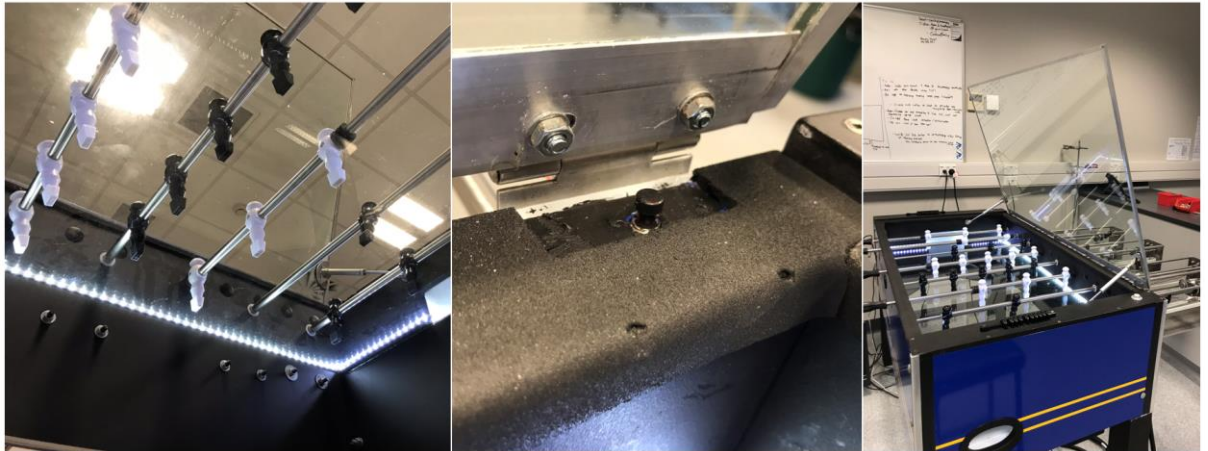


Figure 4-7 - From left to right a bottom-up view of the glass base, the interlock switch and the safety lid

The glass base was included so that the camera could be mounted beneath the table and look vertically upwards at the playing field. This solved a few problems that were present in the work of other authors:

1. Partial or full occlusion of the ball by the rods or players, respectively, as discussed by (Janssen et al., 2012)
2. Ergonomic issues with the camera mounted overhead, as discussed by (Weigel, 2005)
3. Occlusion issues where a human player would accidentally put their hands, heads or other objects between the camera and the playing field, as discussed by (Weigel, 2005)
4. Extra mounting costs and cable/electronics management issues – due to power supplies, compute system, stepper motor drivers being housed separately. This is not discussed explicitly by other authors but is an obvious problem

However, it also introduced new problems:

1. Reflectance from any illumination beneath the table such as LEDs on the FPGA development board, or from ambient lighting reflecting off the ground
2. Silhouetting or bright spots from any lights directly above the playing field within the camera field of view
3. Potential misidentification of the ball if human players, visible to the camera, are wearing clothing of a sufficiently similar colour to the ball

LED playing field lighting

This feature was included in the design as it was necessary to illuminate the ball to assist the image processing software to effectively segment the ball from its surroundings. In the absence of the field illumination, the contrast between the ball and the other items in the camera field of view was insufficient for the colour filtering operations to segment the ball from its surroundings. This can be

seen in Figure 4-8. In Figure 4-8, on the left-hand side the field illumination is on, which increases the illumination of the ball, while simultaneously reducing the exposure on the vision system's auto exposure, due to the brightness of the LEDs. This decreases the amount of background that the camera detects. The contrast can be seen in the right-hand image of Figure 4-8, in which the background is very visible.

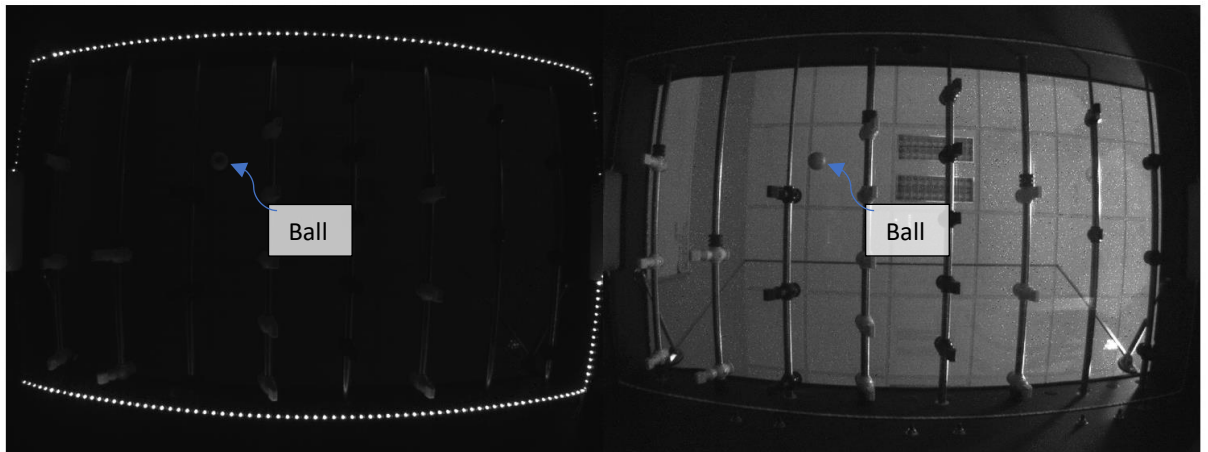


Figure 4-8 - From left to right - bottom-up view of foosball playing field with field illumination on and off respectively

4.2. Materials

The table body was manufactured out of MDF and finished with high quality Vinyl as shown in Figure 4-9.



Figure 4-9 - Image of completed, vinyl wrapped semi-automated foosball table – from front right

The actuation module frames were built from aluminium end plates and steel joining rods. All bearing housings, pulleys and mounts were manufactured from aluminium. The drive belts are polyurethane

timing belts with internal steel tension bands. The foosmen rods are steel hollow tube running on nylon/acetal bushes on the internal solid steel rods. The playing field and protective cover over the playing field are both made of glass.

4.3. Design aspects meeting official specifications

The foosball table is built to the same dimensions as a standard USTSA foosball table. The specifications are as follows: 56.5 inches in length, 34.5 inches in width and 29 inches in height. These dimensions are shown in Figure 4-10.

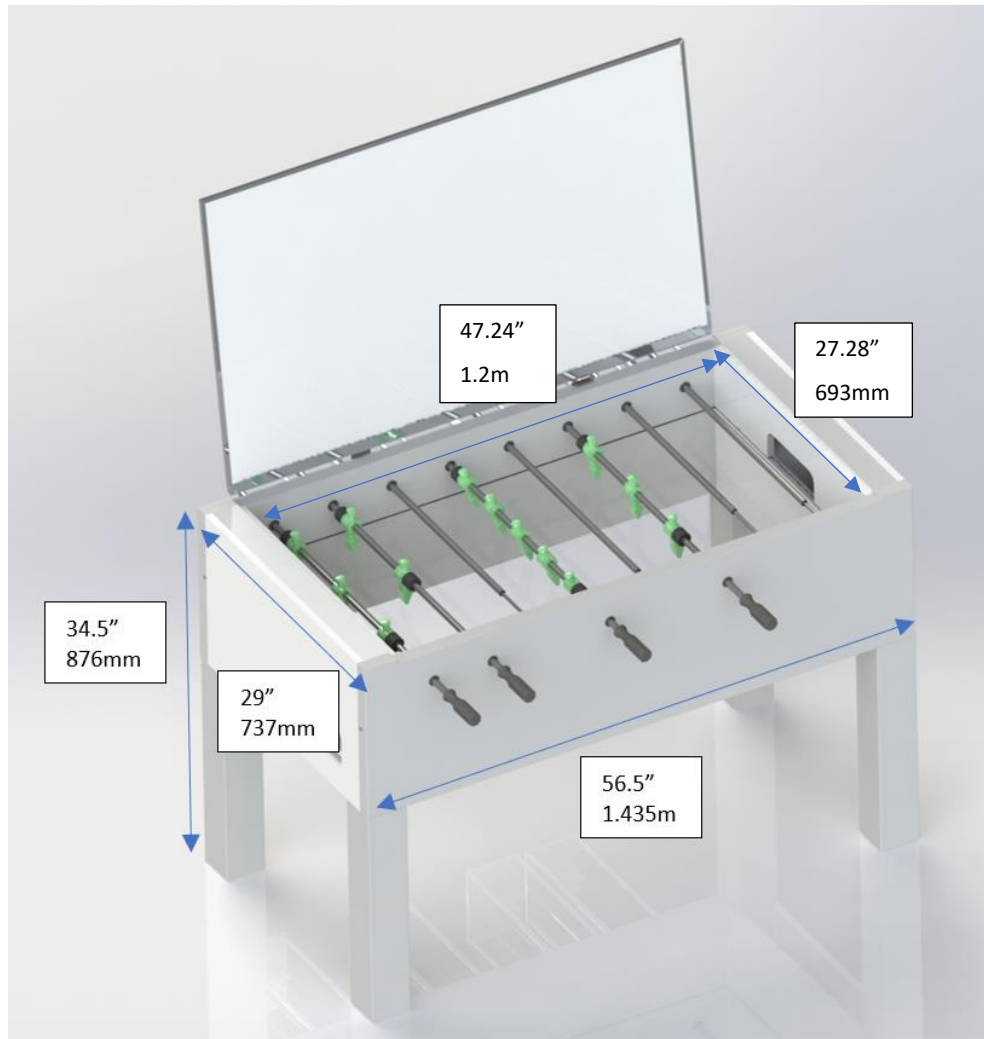


Figure 4-10 - Render of foosball table CAD model

The table consists of 8 rods in total, with four rods per team. The foosmen on the automated side have been suppressed (made invisible) to highlight which rods are actuated by the human players. Therefore, up to two players can play on the human side, as can be seen in Figure 4-10. The rods are spaced evenly apart at 6-inch (152.4 mm) intervals. The foosmen figurines used are of the USTSA dimensions. As with USTSA tables, the foot height is adjustable so that the table can be made perfectly level.

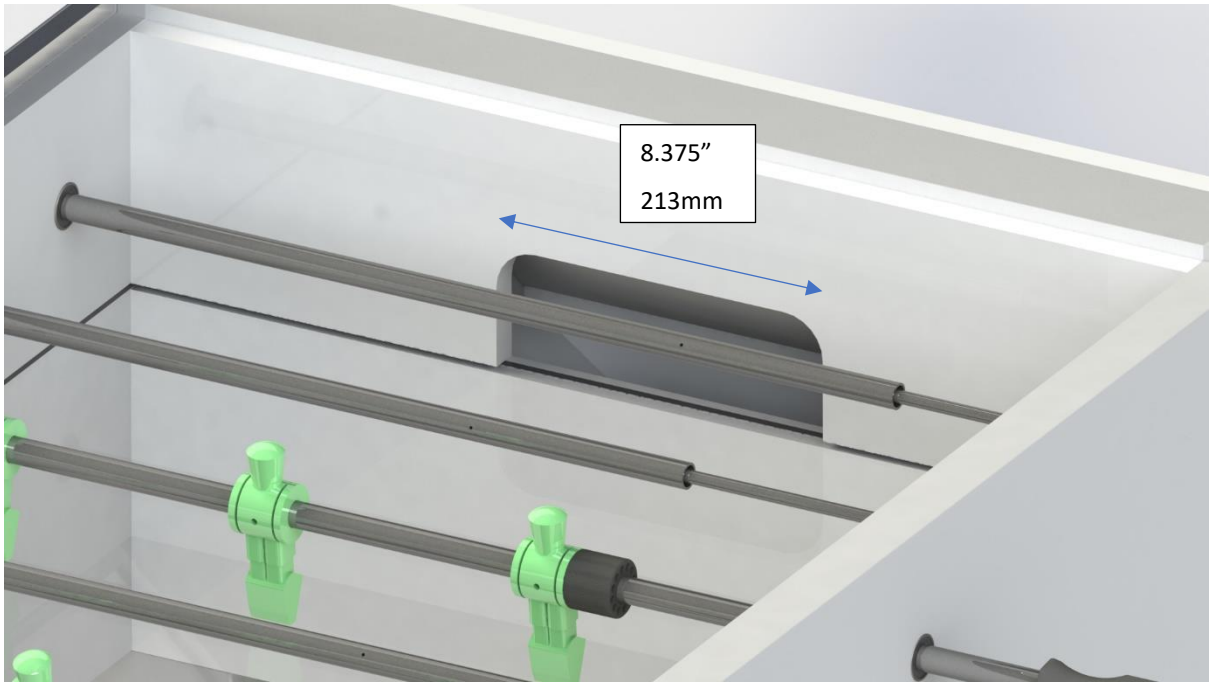


Figure 4-11 - Render of foosball goal on CAD model

The goals dimensions match those of USTSA foosball tables at 8.375 inches (213 mm) in width. This is shown in Figure 4-11. Both ball return chutes (one per goal) were designed to deliver the ball back to the human side for easy access and playability. Figure 4-12 shows one of the ball return chutes on the manufactured system.



Figure 4-12 - Semi automated foosball table right hand ball return chute

4.4. Accurate vision system placement

Given that the vision system is used to accurately estimate the position of the ball being tracked, a placement jig was required to ensure accurate repeatable placement of the vision system beneath the foosball table. This was done so that after initial calibration of the vision system, the electronics enclosure could be moved and replaced sufficiently accurately that no noticeable error would be introduced into the interception system. Figure 4-13 shows a CAD model of the placement jig, and Figure 4-14 shows the physical manufactured system. The top left and right images in Figure 4-13 show close up views from either side of the homing mechanism. The oval shaped tabs slot into the acrylic (transparent) backplate on the electronics enclosure.

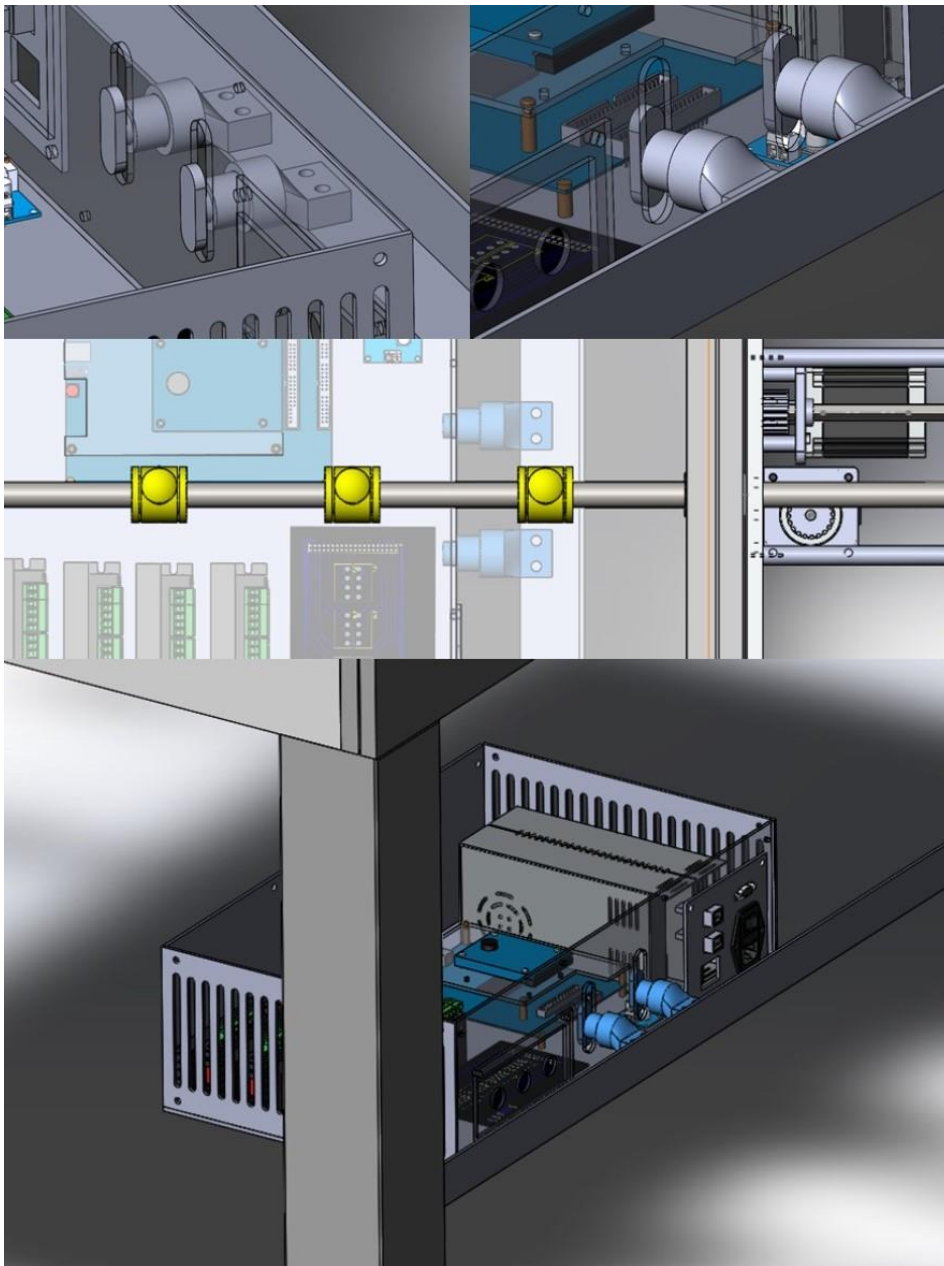


Figure 4-13 - CAD model of the placement jig for the foosball table vision system

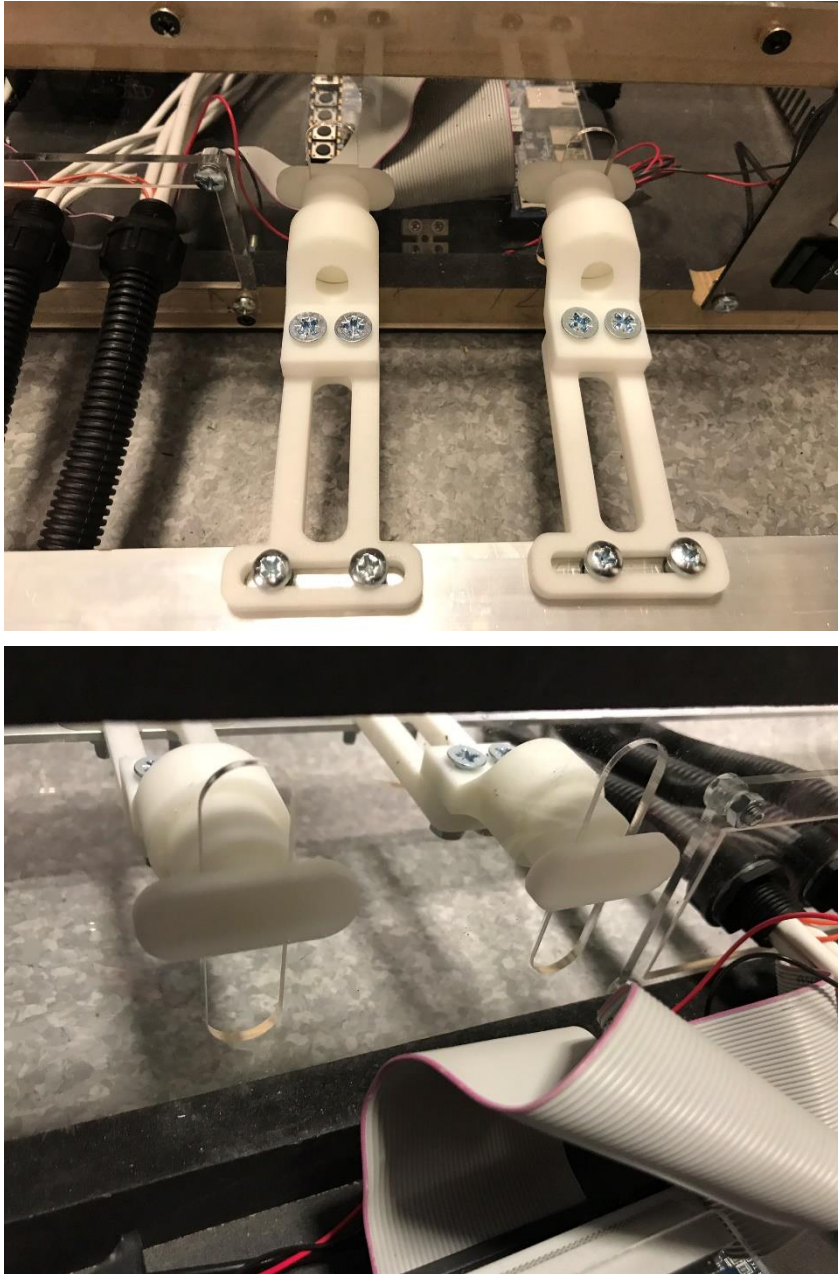


Figure 4-14 - Manufactured calibration homing jig with rotating locking tabs

As can be seen in Figure 4-14, the oval tabs rotate so that the enclosure, once locked in place, cannot be moved without unlocking the tabs.

5. Vision subsystem

Accurate, low latency, low noise data acquisition is very important in controlling a fast, dynamic system (Franklin et al., 2015). This is the case regardless of what system is being controlled – if it operates at a high speed, obtaining reliable measurements with minimal latency is imperative. On these grounds, a method to obtain, process and utilize visual data (image data) with minimal overall system latency is required when visual feedback is used in the context of a robotic control system.

This chapter will discuss control system latency and its importance both generally, and for the specific example of high-speed image processing, for control of an automated foosball table. The proposed vision system can be used as either the reference signal (the position of the ball which the foosball table rods are trying to match for interception), or both the reference and the feedback sensor element in the control loop.

Two types of latency are important to consider. Novel event recognition latency is the time it takes for a system to identify that a novel event (for example the appearance of an object) has occurred. In tracking applications, steady state latency is the time it takes for a vision system to report the updated position of a target object. Unfortunately, most papers only discuss image processing latency, rather than complete measurement system latency. These can be misleading in terms of estimating performance of feedback control.

For this reason, the performance of several different implementations of computer vision systems will be evaluated and compared in this chapter. These systems have been designed for high speed control. As such, latencies have been minimized where possible. The aim of the experiments was to determine both novel event recognition and steady state latencies associated with each of the candidate test systems.

5.1. Background

5.1.1. Latency in control systems

Control systems are designed to accept some bounded input and provide some bounded output, both within a specified range (Engelberg, 2015). Many dynamic control systems are run in closed-loop mode. This means that the system output (controlled variable) feeds back, via some sensor, into the controller of the system, and the system uses that feedback to adjust its control input (Franklin et al., 2015).

Within a control system, sensor latency can be viewed as the time between an event occurring, and the data associated with the event being captured by the measurement system and passed to the controller (Engelberg, 2015). In the experiments documented here, these events are the appearance or movement of an object. The image is captured with a camera and image processing is used to provide the position of the object.

In the control of an automated foosball table, the position of the ball is measured, and fed into the control algorithms to control the positions of the foosmen to intercept the ball. This is the feedback element of the positional control. In practice, the control is a little more complex than simple positional control. The ball's current and past positions are used to predict the motion of the ball which determines the position at which the rods controlling the foosmen will need to arrive when, or just before, the ball reaches the interception point. The system therefore represents a type of tracking control system with the predicted ball intercept position as the reference signal and the foosmen's position as the controlled variable (Franklin et al., 2015).

In control systems, two important parameters which determine stability and robustness of a system are phase and gain margins (Engelberg, 2015). The phase and gain margins indicate how much external disturbance (subtracted phase and added gain) the system can withstand without becoming unstable (oscillating) or failing entirely (Engelberg, 2015). Latency plays a strong part in this problem. If the sensor feedback is delayed by some amount, then this delay corresponds to a linear phase delay with frequency, with the delay proportional to the sensor and measurement system latency. As this appears within the loop gain, the latency reduces the phase margin, and reduces the actual performance of the system making it more difficult to control. If the latency exceeds a critical value, the system will become unstable.

Vision based control systems typically consist of some image capture hardware (camera), image transmission interface, image processing system (hardware or software compute engine), communication interface and actuator or output control hardware. The transmission of the data, processing, and communication delays all add up to cause substantial system latency. Some of the latency is caused by the sheer volume of data being transmitted and processed. Various methods including pipelining, stream-processing, and simply increasing the processing power of the compute engine, have been used to improve the performance of frame-based vision systems over time. However, the latency is ultimately limited by the system frame rate since the data is transferred from the sensor to the compute engine serially.

5.1.2. Related work

The literature describes several arrangements which use vision systems to capture and utilize motion data of various objects. The limiting factors of latency are mostly system-dependent; however, the general principle is that excessive latency can lead to instability. This is the primary motivation for efforts to minimize latency.

The most similar system (Janssen et al., 2012), was also an automated foosball table with a camera that operated at 200 FPS with a measurement system latency of 17.5 ms. The camera transmitted 8-bit monochrome images with a resolution of 657×446 to a PC. This was done via Gigabit Ethernet. The system's inability to defend against certain shots was determined to be due to latency in controlling the rods.

An FPGA SoC was used by (Čížek et al., 2016) for hardware acceleration of complex image processing tasks for vision-based navigation systems. The authors used a frame rate of 60 FPS at 640×480 resolution. For their application, they claimed an image processing latency of approximately 17ms.

Two different realizations of high-speed control of UAVs using stereo vision were discussed by (Barry et al., 2015). They achieved 2 ms image processing latency with the FPGA implementation and a worst case 16.6 ms image processing latency with the ARM processor implementation. Given the lower image size of 320×240, and the relatively low computational power required for the matching algorithm, the latency achieved in this work appears reasonable. Hardware based processing gave approximately an eightfold improvement over the equivalent ARM based processing. These reported latencies are only for the image processing, and not the complete system latency for object detection (including image capture). It does give a representation of the effectiveness of hardware image processing platforms such as FPGAs compared to traditional systems, such as the ARM processor.

(Andersson, 1990) demonstrated a 60 FPS, real time stereo vision system achieved through custom hardware with around 32 ms latency. The system's performance was achieved through custom hardware and software. The reported 32 ms latency was reasonably effective for the real time autonomous control of their robotic manipulator. However, it is likely that the latency and throughput requirements for their application are less demanding than an automated foosball table due to the relatively slow speeds at which their robot manipulator was actuated.

(Cigliano et al., 2015) discusses a distributed computing and control platform for a robotic arm and gripper assembly to catch a ball thrown at it. This represents a tracking control system. Latency was simply estimated and used to synchronize events between the various subsystems, rather than latency being minimized.

Many others (Berner, Brandli, Yang, Liu, & Delbruck, 2013; Censi et al., 2013; Conradt et al., 2009; Mueller et al., 2015) also discuss the use of dynamic vision sensors (DVS) for event based image processing systems with extremely low latencies, ranging from 12 μs to 15 μs. DVS sensors work by detecting changes in intensity in each pixel which can result in very low latencies, however DVS sensors appear to be more suited to highly structured environments and would not be suitable for the dynamic environment associated with a semi-autonomous foosball table. Additionally, due to the lower resolution of these sensors, between 128×128 and 240×180 pixels, and the lower level of configurability, they do not meet the requirements for automating a foosball table. The foosball table requires higher resolution due to the size of the ball relative to the size of the table, and the precision required for interception by the foosmen. They do, however, reflect the importance of high-speed, low latency vision-type sensors in the control of robotics.

5.1.3. Hardware tested

Based on the findings of this review, it was determined that for the application of semi-automated foosball, some experimentation was required to determine the true system latencies of several off-the-

shelf image capture systems, as opposed to optimistic estimates based purely on image processing speeds. Figure 5-1 shows the key hardware for each of the test systems.



Figure 5-1 - Image capture test systems

1. Altera DE1-SoC FPGA development board with a Terasic D5M camera module.

The camera is capable of capturing image at resolutions of up to 5 MP. At 640×480 resolution, the system can capture images at a rate of up to 127 FPS. The image data was streamed from the sensor and the FPGA performed the image processing operations directly on the image data stream. The FPGA and D5M camera were selected for their efficiency in the required image processing applications, as well as being relatively low cost.

2. CMUcam5 (PixyCam) embedded vision platform.

The system is capable of colour object tracking at 50 FPS with a resolution of 320×240. The image processing is done onboard the camera module with the object data sent to a microcontroller via either SPI, I2C or UART. I2C was used in this experiment. The microcontroller used was an Arduino Uno, clocked at 16 MHz. The CMUcam5 was selected as it represents a small, low power, embedded object tracking vision system that can be purchased inexpensively.

3. PS3 EYE.

This camera can capture images at around 75 FPS at 640×480 resolution and greater than 100 FPS at 320×240 resolution. It transmits the image data via USB 2.0. Note, the PS3 EYE was used for 2 experiments. The PS3 eye was selected as it was capable of high frame rates and could be purchased at very low cost.

4. Logitech C920 Pro Webcam.

This webcam is internally limited to 30 FPS. However, it is capable of many different resolutions, including 640×480. It also transmits the image data over USB 2.0. This camera was selected as it had higher image quality and was also relatively low cost. It was chosen as a reference point for performance from a camera with a lower framerate.

Additionally, a quad-core i7 Windows laptop (Asus N550-JV) was used for the MATLAB based image processing for the PS3 EYE and the Logitech C920 webcam.

For the timing of latencies, an Arduino Uno was used to implement a high precision timer, and an STM-F429ZI discovery board was used as the display unit. The display unit displayed the reference image for the vision systems to recognize.

5.2. Novel event detection latency

5.2.1. Aim

The aim was to measure and compare the novel event detection latency of several different realizations of computer vision systems.

5.2.2. Methodology

The experiment was carried out for each of the systems with the following steps:

- 1) An LCD display unit on an STM F429ZI is set to display a target object after a random time interval of between 5 and 10 seconds. The display latency is fixed and was the same for all experiments.
- 2) When the STM unit displays the object, it starts a high precision timer on a separate microcontroller (an Arduino Uno)
- 3) The vision system being tested continuously captures and processes images checking for the target object to appear on the display unit. This involves the following steps:
 - a) Capture images at given resolution and best frame rate
 - b) Colour space conversion (YUV for coloured object segmentation)
 - c) Filtering – colour detection, morphological filtering (erosion and dilation)
 - d) Blob analysis and centroid calculation
- 4) Once the object is correctly recognized, an “object recognized” signal is output. Where possible this output signal is in the form a logic high written to a GPIO pin. On systems where this was not practical, a serial command was output via a USB port to the precision timer.
- 5) Once the timer receives the object recognized signal, it immediately stops and outputs the elapsed time. This is the total latency from the time the object was displayed until the time the object was recognized by the vision system and the stop signal sent, closing the loop.

Each system was tailored to reduce latency as much as possible, and to make the comparisons as fair as possible. These methods included:

1. Where possible, using the same resolution for all the systems – 640×480;
2. Using the same image format and colour depth for all the image streams;

3. Not displaying the images on the PC based systems when performing the actual tests, as this can add processing overhead in these systems;
4. Minimizing the amount of data sent in the communication systems – as would be done on similar implementations;
5. Using the highest baud rate available if serial communication was required;
6. Using the maximum available framerate for all systems apart from the FPGA. The FPGA system was fixed at 60 FPS due to display timing constraints.

The overall latency is the combination of several components:

$$L_{total} = L_{target\ display} + L_{image\ capture} + L_{transfer} + L_{processing} + L_{comms} \quad (3)$$

5.2.3. Results

The experiments consisted of 20 samples for each test platform, resulting in a total of 100 samples. Minitab was used to analyze the data and produce a visual representation of the results. The boxplot in Figure 5-2 shows the latency for the 5 experiments. The latency data is presented in Table 5.1.

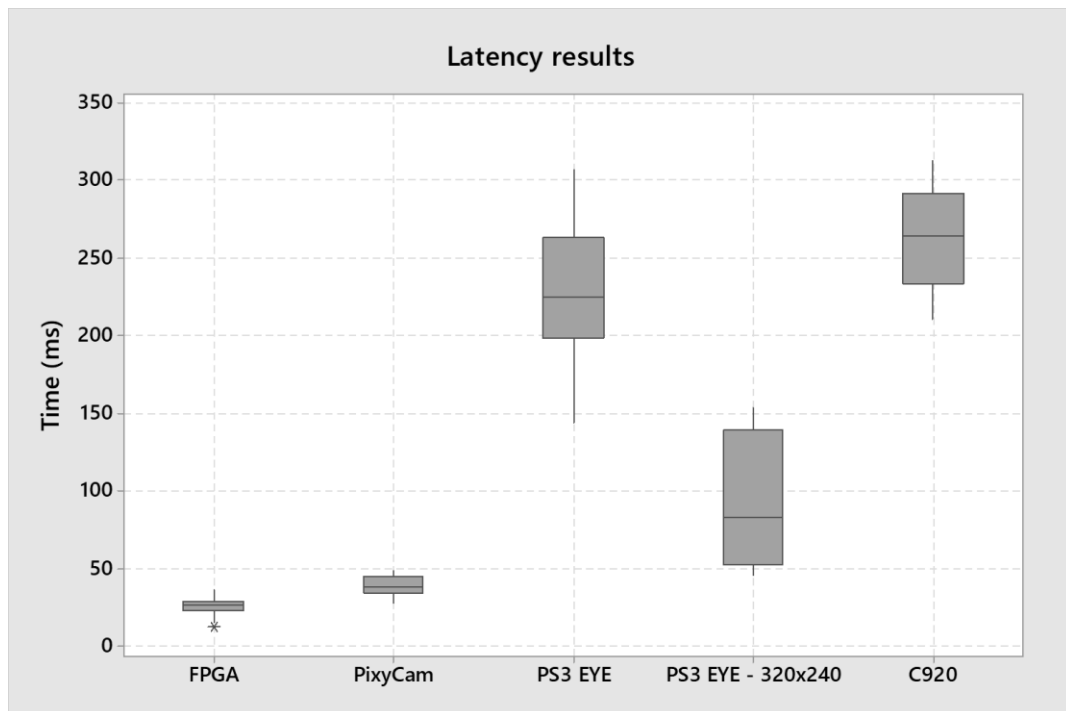


Figure 5-2 – Novel event detection latency results for all 5 experiments including PS3 eye at both resolutions

Table 5.1 - Important novel event detection latency data

Hardware	Mean latency (ms)	Std dev	Min	Max	Range	Resolution	Nominal framerate
FPGA	26.4	5.4	12.9	36.8	23.9	640×480	60
PixyCam	39.7	6.5	28.3	49.6	21.3	320×240	50
PS3 EYE 640×480	229.2	48.4	144.4	307.4	163	640×480	75
PS3 EYE 320×240	94.6	43.5	46.6	154.2	107.6	320×240	>100
Logitech C920	260.7	31.3	211.5	313.2	101.7	640×480	30

The reason for the PS3 EYE test being performed at both 640×480 and 320×240 was to test it at the specified resolution, as well as to compare it with the PixyCam, which operates at a resolution of 320×240. It can be seen in Figure 5-2 that as the average latency of novel event recognition increased, the variability increased too. This would be very unhelpful if implemented in a robot control system that requires prediction because the latency, upon which predictions are partially based, would vary significantly, making it difficult to accurately estimate the ball motion parameters such as velocity, heading angle, and curvature. This is also true with changing ball motion parameters, such as in acceleration or deceleration situations, which a constant velocity prediction model may not be able to account for.

5.3. Steady state latency

In addition to novel event detection, another type of system latency is also important in characterising overall system response. The second series of experiments conducted were to measure the steady state latency. This latency is the continuous time difference between an event occurring and the image processing system recognising/measuring the event or change. Steady state latency is not dependent on the relative timing between an event and the frame capture phase of the image processing algorithm as it is measured over a period of time.

5.3.1. Aim

The aim was to determine and compare the steady state latency of the various systems for continuous object tracking.

5.3.2. Methodology

To simulate a moving target, with precisely known position, a coloured marker was moved on a circular path. The movement was controlled by a stepper motor so that the position of the target was known. By the time the imaging system has estimated the location of the target, it will have moved, with the angle

difference between the current and reported position being proportional to the steady state latency. The assembly used is shown in Figure 5-3.

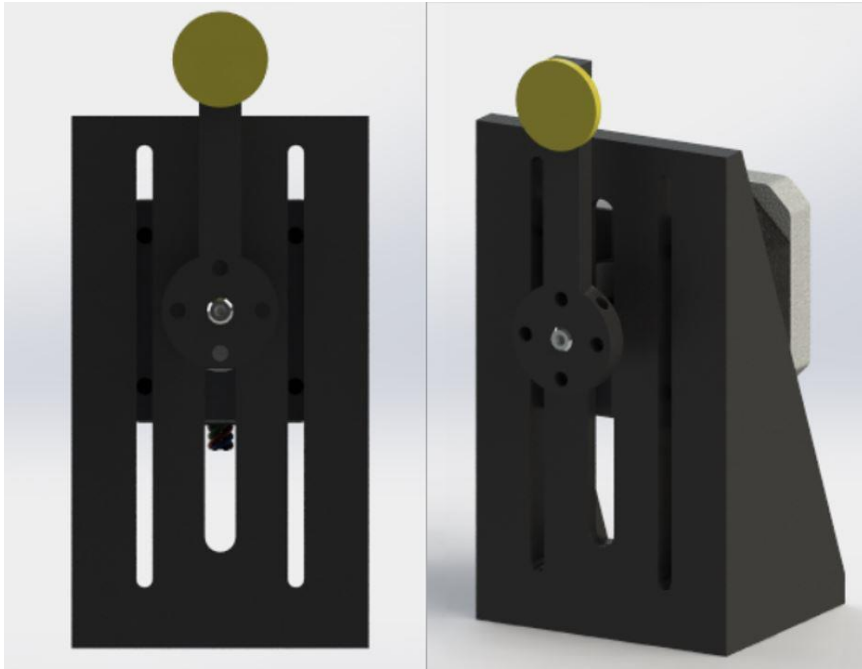


Figure 5-3 - CAD model of object marker apparatus used in steady state latency experiment

Note – the motor speed used for the FPGA and PixyCam was 600 degrees per second while the motor speed used in the PC system was 225 degrees per second. This was due to the relatively low framerate (around 10 FPS after processing operations were performed) achieved by the PC based systems. In order to achieve adequate timing resolution with the PC based systems, a lower motor speed was required.

With the motor spinning the coloured marker at a constant rate, the experiment was carried out for each of the systems with the following steps:

- 1) The vision system being tested continuously captures and processes images. This involves the same steps as the novel event detection latency experiment, however running continuously
- 2) In the FPGA and the PC based system, the angle about the centre is calculated by the vision system. The PixyCam, on the other hand, passes the x and y coordinates to the microcontroller-timer system which then calculates the angle
- 3) For each new angle value received or calculated, the timer system records this alongside the current known angle of the motor
- 4) These two angle values are output to the console in real time, and the differences in angle, $\delta\theta$, can then be calculated. The calculations are done using the following equations:

5.3.3. Mathematics

The angle of the spinning marker about the centre of the image is

$$\theta_1 = \arctan\left(\frac{X - 320}{Y - 240}\right). \quad (4)$$

The difference in angle between the spinning marker and the known motor position is

$$\delta\theta = \frac{\text{current motor angle} * 360}{400} - \theta_1. \quad (5)$$

Therefore, the latency is

$$t = \frac{\delta\theta}{V_{rot}} \quad (6)$$

where V_{rot} is the rotational velocity of the motor in degrees per second.

5.3.4. Results

The boxplot in Figure 5-4 shows the steady state latency for the 4 experiments. The latency data is presented in Table 5.2.

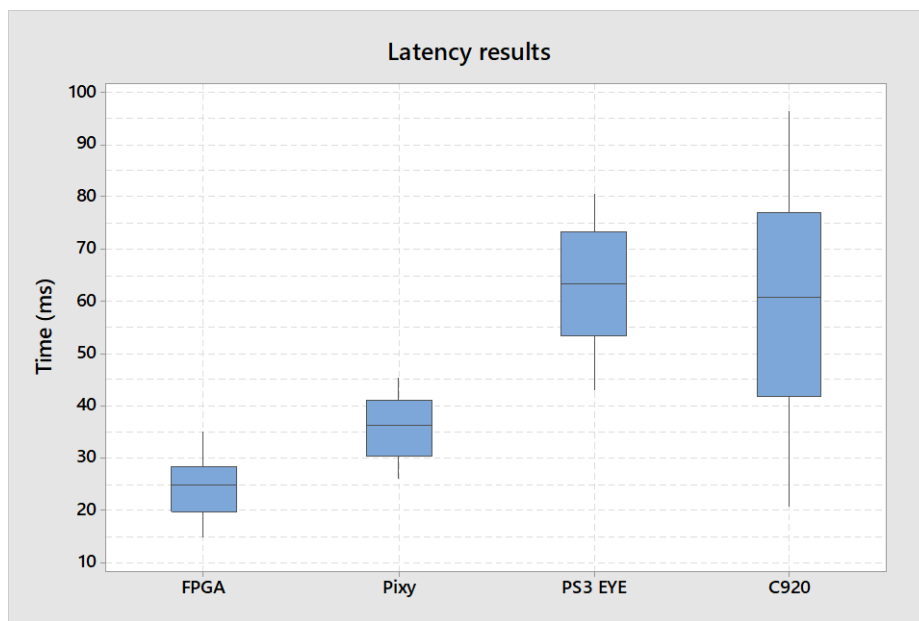


Figure 5-4 - Steady state latency results for all 4 experiments

Table 5.2 –Important steady state latency data

Hardware	Mean latency (ms)	Std dev	Min	Max	Range	Resolution	Nominal framerate
FPGA	24.9	5.4	15	35	20	640×480	60
PixyCam	35.8	5.5	26.2	45.3	19.1	320×240	50
PS3 EYE 640×480	63	11.5	43.3	80.61	37.4	640×480	75
Logitech C920	58.9	21.19	20.9	96.2	75.3	640×480	30

As shown in Figure 5-4, the best performance was achieved using the FPGA based system. The compounded latencies from image transfer, serial processing, and serial communication resulted in much larger latencies in the USB based camera systems.

5.4. Discussion

5.4.1. Immediate usefulness of results

It is impossible to have zero system latency, therefore experimentally determining the latency is important for fully characterizing the system for accurate modelling and future development purposes. It also serves as a benchmark to measure future improvements. Additionally, as mentioned before, it is useful to have accurately measured latencies as these can assist greatly with predictive compensations in systems with unavoidable latency.

5.4.2. Comparison of performance

Firstly, it should be noted that these experiments were performed with the FPGA system running at less than half of the maximum image capture and processing speed of which it is capable. This was done for display synchronization and development reasons, however future work will see the frame rate increased to above 120 FPS and the latency (and variability of latency) should, thus, decrease proportionately, due to the streamed nature of the processing.

Compared with the results for novel event recognition, the latencies seen in steady state are lower, with lower variability. This, despite more processing being required, is likely due to relative timing differences being minimized.

In systems with lower framerates, the relative timing made a larger difference than in those with higher framerates. Overall, systems with low latency and high framerate performed the best across both tests, in terms of both latency and variability.

It can be seen from the results that, even with the currently capped frame rate, the FPGA system still yielded the best performance in both novel event detection and steady state latency. 25 ms latency is similar to the latency experienced by the human eye when detecting new stimulus.

The PixyCam, which yielded the second lowest latency in both experiments was only running at a resolution of 320×240, so cannot truly be considered a close contender, given that the specifications for this application are a minimum resolution of 640×480. However, it was included for reference as it is representative of a simple system for embedded image processing where high resolution is not required. It appears that the lower resolution and the application specific hardware are the primary reasons for the PixyCam's low latency.

The PS3 EYE, running at 640×480 at 75 FPS, yielded a higher latency than the PixyCam, despite the higher frame rate. The experimental data demonstrates the importance of efficient image processing

algorithms and communication strategy. Because the image processing algorithms were implemented in MATLAB, there were processing overheads (within the operating system and within MATLAB) which likely slowed down the processing. Additionally, the communication overheads caused by using a non-streamed image acquisition method caused additional latency.

Finally, the test system using the Logitech C920 webcam, which was internally limited to 30 FPS, yielded the poorest performance of the tested systems. The latency was the highest (an order of magnitude higher than the FPGA system in novel event detection), and the most variable in both tests. It is likely that the combination of inefficient processing, combined with relative timing of the event and the image being captured was the cause of the latency. This, in a highly dynamic control system, would likely result in poor system stability.

Overall, the FPGA system yielded the best performance in steady state latency and in novel event detection. In addition to yielding the best latency performance, the FPGA based vision system also met all the requirements for the application of a semi-automated foosball table. These are minimum 640×480 resolution, 60 FPS, less than 30 ms latency, continuous display of the image, and non-saturation of the processing unit to enable other tasks, such as actuator control, to be carried out.

5.4.3. Potential improvements

In the FPGA system, the angle was calculated according to equation (4). This was done using a Nios II core instantiated in FPGA logic. The Nios II core is significantly less powerful than the HPS (ARM A9) core which is part of the DE1-SOC FPGA development board; 50 MHz versus 800 MHz. This will have contributed to some of the latency seen in the steady state latency results for the FPGA, however this can be improved by using the HPS rather than the Nios II core.

Some improvements could be made to the serial processing systems by using more powerful desktop processors, and more efficient image transfer protocols such as ethernet. However, without custom communication solutions, and streamed image processing, the serial processing systems would still likely be unable to achieve the same latency performance as the FPGA based systems. Additionally, increasing the power of the components without using a more efficient method of processing the images, such as streamed image processing, would cause the system to no longer meet the requirements of low power consumption (less than 20 W) stated in section 3.4.

5.4.4. Automated foosball

In the context of automated foosball, and of robotic control in general, it is obvious from the data that the best choice from the systems tested is the FPGA based image processing platform. The low latency is desirable from a control perspective (Honegger et al., 2014). Additionally, based on human systems, “predictive control is essential for the rapid movements commonly observed in dexterous behavior” (Wolpert & Flanagan, 2001). In foosball, there are very high speeds involved, relative to the size of the playing field. This means that high-speed image processing is imperative a reasonable level of response

is to be possible. Not only does the sensing need to be excellent, so do the actuation techniques. Using reconfigurable FPGA hardware to create stepper motor controllers, or general-purpose actuator control, could be an extremely effective method to solve the problems involved with the automated foosball table.

By combining low latency image processing (streamed image processing on Altera DE1 SoC FPGA), and low latency communication (high speed AXI bridge from FPGA to the embedded ARM cores), computer vision can be used as both a reference signal and a feedback element in a control loop. The system becomes even more powerful when actuator control and other general system functions (for example safety interlocks and user inputs) are also implemented within the FPGA. This results in comparatively low overall system latency for all elements of the control loop. Additionally, the use of an FPGA SoC enables scalability at a cost of minimal additional latency or scheduling issues present in conventional microcontrollers.

5.5. Conclusions

The novel event detection latencies ranged from 26.4 to 260.7 ms, and steady state latencies ranging from 24.95 ms to 62.99 ms, with the FPGA system yielding the lowest latency.

The experimentally determined latency values will be useful in future work for the prediction capabilities of the system, as well as for modelling purposes.

For the FPGA system to be a completely vision-controlled closed loop system, the camera could be used for tracking both the position of the foosmen, and for tracking the ball.

5.6. Implementation of final vision system

The implementation of the system involved many stages of concept, design, prototype, troubleshooting, debugging, simulation and testing. This is represented in Figure 5-5.



Figure 5-5 - Vision system development process

The first system developed and tested was the computer vision system. This included prototypes of software and hardware-based colour object tracking systems.

5.6.1. Terasic D5M camera module

The image capture device used for the FPGA-based image processing was the TRDB-D5M. The TRDB-D5M camera module, made by Terasic, shown in Figure 5-6, is a 5MP (2,592 H x 1,944 V) CMOS sensor on a development board which connects to the FPGA via a 40 pin 2 row header. The camera is controlled by an I2C communication interface. The gain, exposure, white balance, resolution, triggering, and windowing can be controlled via the I2C interface.



Figure 5-6 - Terasic TRDB-D5M camera development board

The camera is capable of operating in windowed mode with numerous different resolutions and framerates – governed by the selected pixel clock. Additionally, the window can be shifted to be placed anywhere on the active region of the CMOS chip, in 2-pixel increments due to the Bayer pattern readout.

The module is capable of a maximum data rate of 96 Mp/s at 96 Mhz clock speed.

The output image from the camera is in Bayer pattern format, as illustrated in Figure 5-7.

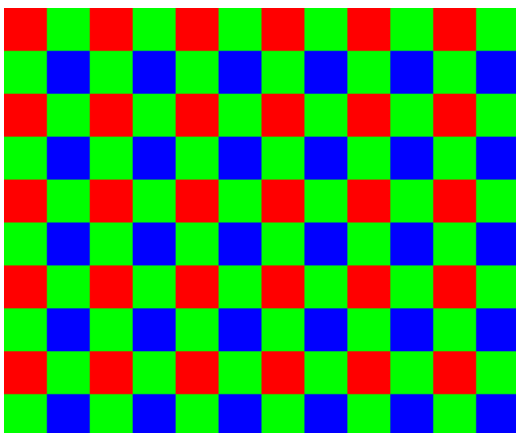


Figure 5-7 - Bayer RGB pattern representation

5.6.2. Prototypes

Prototypes were used for testing the image processing algorithms prior to implementing them in FPGA. To simply implement an image processing algorithm in FPGA without testing the effectiveness of the algorithm previously is generally a waste of time, given that implementing and testing the algorithm in software is typically a much shorter process.

Given this information, a test system was implemented using a Windows PC, MATLAB, and a Logitech C170 USB webcam. The MATLAB image acquisition and image processing toolboxes were used for this testing. The prototype vision system was used to determine the following:

- Which colour space to use
- Morphological filter structuring element size
- Whether other filtering methods like median filtering were required
- Blob detection parameters – minimum and maximum blob size of detected object
- Initial estimates for radial distortion correction requirements

These values assisted with the final implementation in FPGA as reconfiguring multiple times to determine initial ballpark values would have been a tedious, inefficient process. The MATLAB code used to perform this image processing is shown in Appendix A.

The vision system was then implemented in a Terasic DE1-SoC development board in order to test the image processing algorithms and determine what the resource consumption would be. This included hardware resources such as logic elements, GPIO pins, and PLL's.

This process involved several iterations of editing and compiling the VHDL design each time adding new components according to the methodology described in (Bailey, 2011). The order in which components were added was as follows:

1. Display simple pattern on VGA display
2. Capture and stream image data directly from camera with PLL in place to sync camera pixel stream with pixel stream of display
3. Implement Bayer interpolation, known as demosaicing, and display demosaiced camera image on screen
4. Implement colour-space conversion and colour filtering on row buffered image data
5. Implement bounding box and filtered pixel overlay to display
6. Implement noise reduction filtering – morphological erosion and then dilation – and display image on screen
7. Implement connected components analysis (CCA)

Table 5.3 - Resource consumption on DE1-SoC of basic image processing design

Resource type	Resource consumption
Logic utilisation in ALMs	1,259/32,070 (4%)
Total registers	1,460
Total pins	151/457 (33%)
Total block memory bits	110,694/4,065,280 (3%)
Total DSP blocks	0/87 (0%)

Table 5.3 gives the resources required by this minimum design used as a baseline for comparison in subsequent chapters.

5.6.3. Foosball table interception simulation

A vision and motor control simulation was done using MATLAB. The simulation received object coordinates from the FPGA, plotted the coordinates in image space, calculated the trajectory of the ball, and the interception points on each of the automated module rods. Finally, the difference between each of the foosmen and the predicted ball interception point (including the effects of bounces) was calculated. Using this value, the correct foosman for interception on each module was reported.

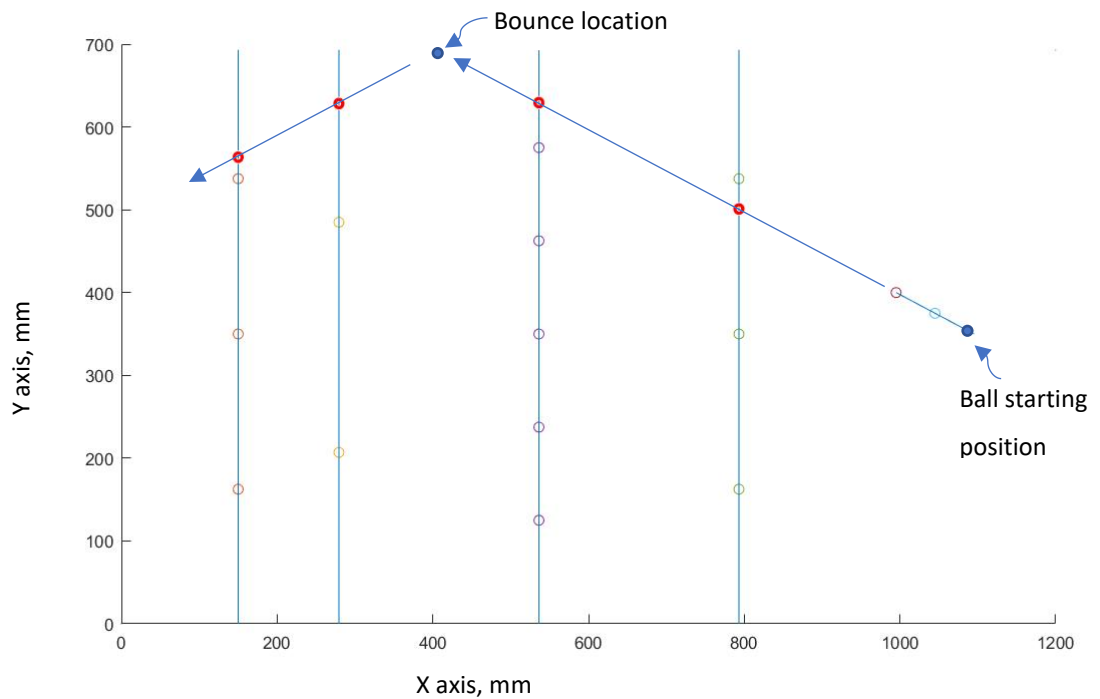
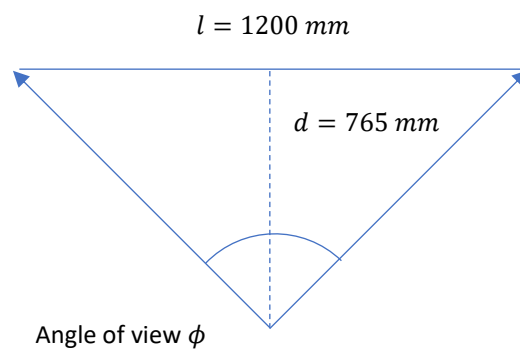


Figure 5-8 - Output window of simple trajectory calculation and interception coordinate simulation

Figure 5-8 shows the interception simulation output performed in MATLAB. The vertical lines represent the 4 actuation module rods with the circles representing the foosmen in their central locations. The red circles plotted on each of the lines indicate the predicted interception locations on each of the rods.

5.6.4. Lens changes

The angle of view captured by the camera is calculated using the following method:



For a given angle of view ϕ and field of view length l , the required distance for the camera to capture the entire field of view is

$$d = \left(\frac{l}{2 * \tan\left(\frac{\phi}{2}\right)} \right). \quad (7)$$

The required final angle of view is therefore

$$\phi = 2 * \arctan\left(\frac{l}{2*d}\right). \quad (8)$$

For the semi-automated foosball table, the required angle of view was 76° at 1024×768 resolution. The original lens provided with the D5M camera module, shown in Figure 5-9, had a 15° angle of view at 1024×768 . This was insufficient, so a new lens was required. The selected lens, shown in Figure 5-10, was a Sunex DSL215 miniature fisheye lens with 185° field of view. As can be seen in Figure 5-10, the Sunex DSL215 has a large, convex front element which provides the large angle of view.



Figure 5-9 - Long focal-length (narrow angle) lens provided with the D5M camera module



Figure 5-10 - Sunex DSL215 fisheye lens

While this lens enabled the vision system to capture the entire playing field, it did introduce severe barrel distortion. A method was required to correct the image for this distortion. Rather than correct the entire image before performing the image processing operations, the image was processed first according to the method presented next, in section 5.6.5, and the calculated object coordinates were corrected for the distortion.

5.6.5. Distortion correction

To correct the barrel distortion caused by the wide-angle lens, the method discussed in section 2.3.3 was used. This involved the following steps:

1. Capture image of a uniform, square grid using the D5M camera, with the image sent to the PC using RS232
2. Load the image into MATLAB
3. Convert the image into a binary image using Otsu's automatic threshold selection algorithm (Otsu, 1979)
4. Morphologically erode the vertical lines using a rectangular structuring element of 1×8 pixels and save the result as a new image
5. Morphologically erode the horizontal lines using a rectangular structuring element of 8×1 pixels and save the result as a new image
6. Find the horizontal and vertical lines in their respective images
7. Track along the vertical lines to find all the interception points
8. Save all the interception points into a large array
9. Using the array as a representative map of the original image's distortion, apply the mapping function, given by equation (2)
10. Use Levenberg-Marquardt optimisation to determine the values of K_1 , K_2 and K_3 that minimise the squared error with the undistorted coordinates
11. Apply the mapping function to the received object coordinates in the HPS as shown in Appendix B.

An image of the two morphologically eroded images with all the interception points found and displayed is shown in Figure 5-11. The black lines are the detected grid lines and the circles plotted on the vertices are the detected intersection points.

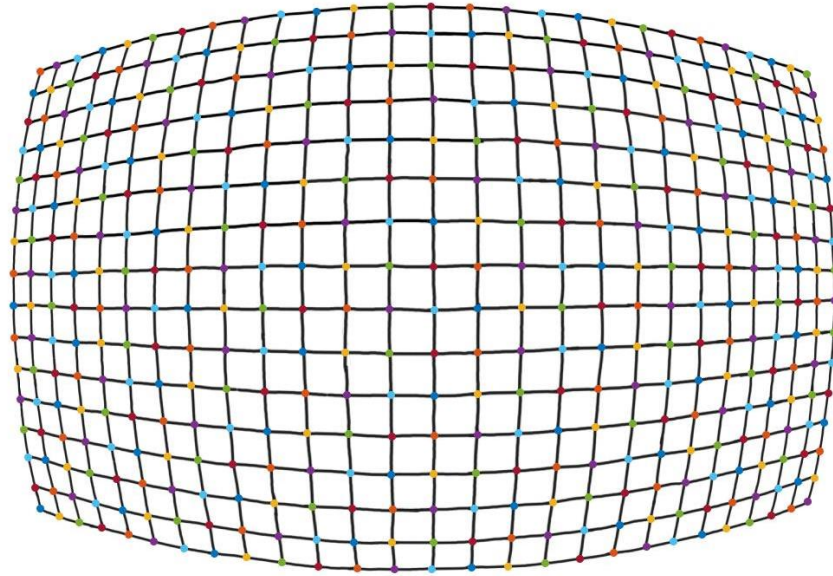


Figure 5-11 - MATLAB output image of all calibration grid intersection points found by distortion correction algorithm

The calibrated and uncalibrated representations of the distortion map are shown in Figure 5-12. As can be seen, the calibrated map is corrected such that the curvature has been straightened.

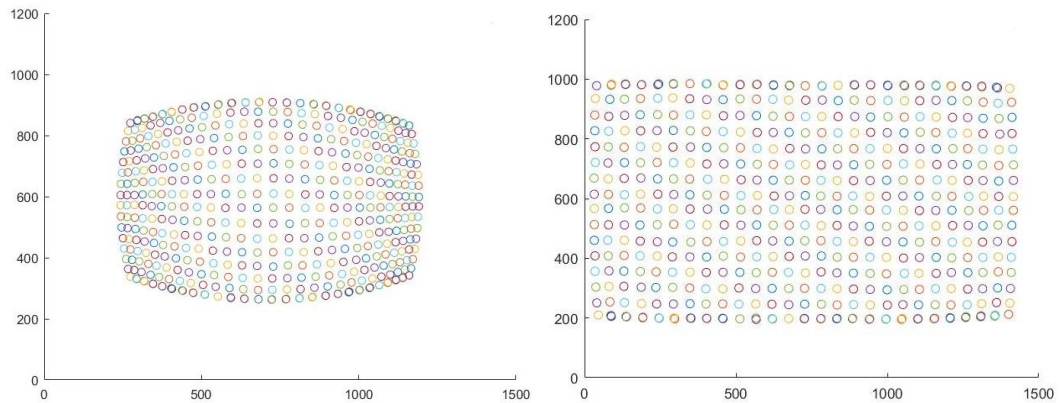


Figure 5-12 - Uncalibrated (left) and calibrated (right) distortion point map

The final steps to utilising this calibration method is to determine the correct X and Y offsets in the final system. This is done by first applying the mapping function in the final system, and then inserting the correct offsets such that when the ball is in the corner corresponding to $(x,y) = (0,0)$, the vision system outputs 0,0 as the true object coordinates. This is done retrospectively by testing the outputs.

5.6.6. Final implementation

The final image processing algorithm was implemented in the Terasic DE1-SoC development board with the D5M camera module. This section will discuss the image processing hardware and FPGA design, while section 0 will discuss the FPGA and HPS contained in the DE1-SoC development board.

The final image processing algorithm is represented by the block diagram shown in Figure 5-13.

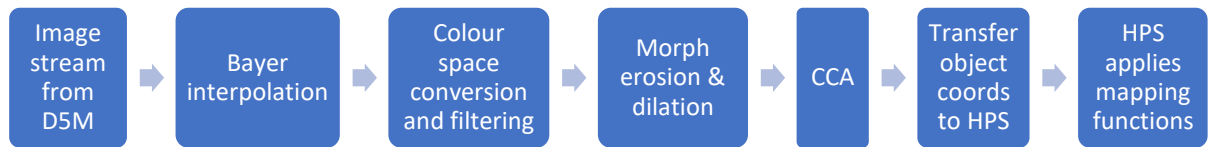


Figure 5-13 - Representation of overall image processing system

All the image processing operations mentioned in section 5.6.2 were included, however in the CCA instantiation, blob size checking was implemented to only output the coordinates of the detected blob if they met size constraints matching the ball size.

The above processes are implemented in FPGA hardware such that where possible, pipelined stream processing is utilised. Successive operations were pipelined, enabling a throughput of 1 pixel per clock cycle to be maintained. First a Bayer interpolation filter is used (see Figure 5-14) to demosaic the input image and give a full-colour image for subsequent processing.

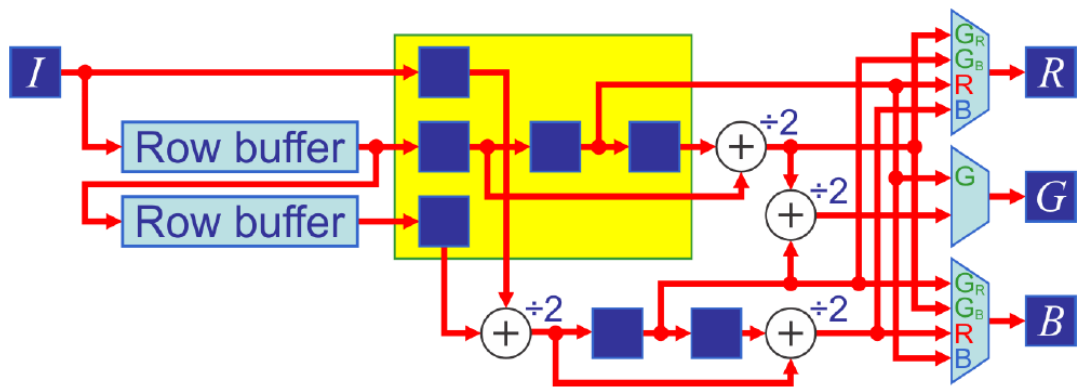


Figure 5-14 - Block diagram of 3x3 windowed Bayer interpolation (demosaicing) hardware implemented in FPGA. (Bailey, 2018). Reprinted with permission.

Following this, the image is converted from RGB colour space to YCbCr because the luminance and chrominance components are separated, which typically offers superior segmentation performance to the RGB or HSV/HIS colour spaces (Janssen et al., 2012; Jianping et al., 2001). This is done using the combinatorial logic shown in Figure 5-15.

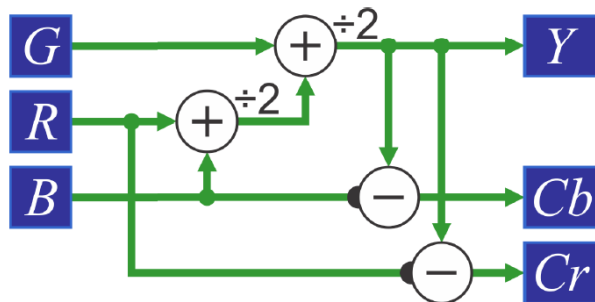


Figure 5-15 - Block diagram of simplified RGB to YCbCr conversion implemented in FPGA hardware (Bailey, 2018). Reprinted with permission.

After colour space conversion, the coloured ball is detected using predetermined colour thresholds and a binary image pixel stream is output. This pixel stream is then filtered using a morphological opening (using a 5×5 window) to remove small noise pixels.

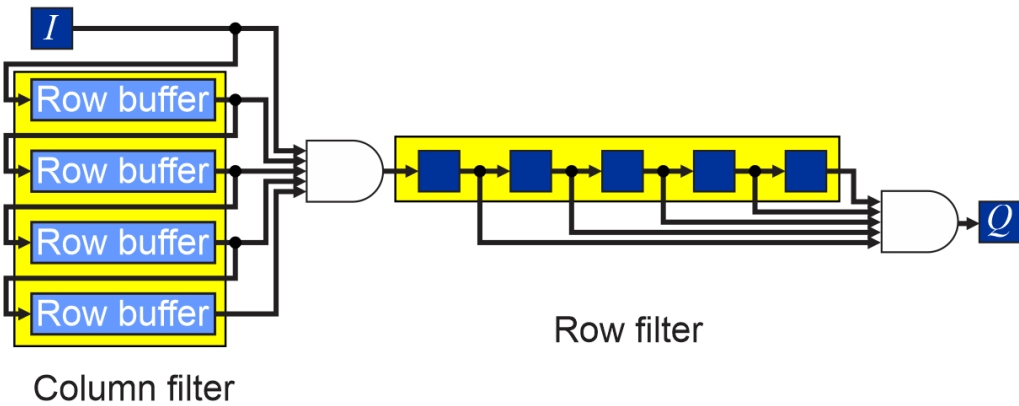


Figure 5-16 - Block diagram of a 5×5 morphological filter (erosion) implemented in FPGA hardware – for dilation, AND gates were replaced with OR gates. (Bailey, 2018). Reprinted with permission.

The final hardware-based image processing operation is connected components analysis. Connected components analysis extracts feature data from each set of connected pixels in the image (Klaiber, Bailey, Baroud, & Simon, 2016). Here the blob size was used to filter non-ball objects, and the bounding box was used to give the ball position. In this implementation, only blobs smaller than 1000 pixels were considered to be the ball. No minimum blob size was implemented as, due to the lighting on the table, there were times when the ball size would appear greatly reduced due to bright spot illumination when the ball was too close to the field illumination LEDs. The field illumination will be discussed in a subsequent chapter.

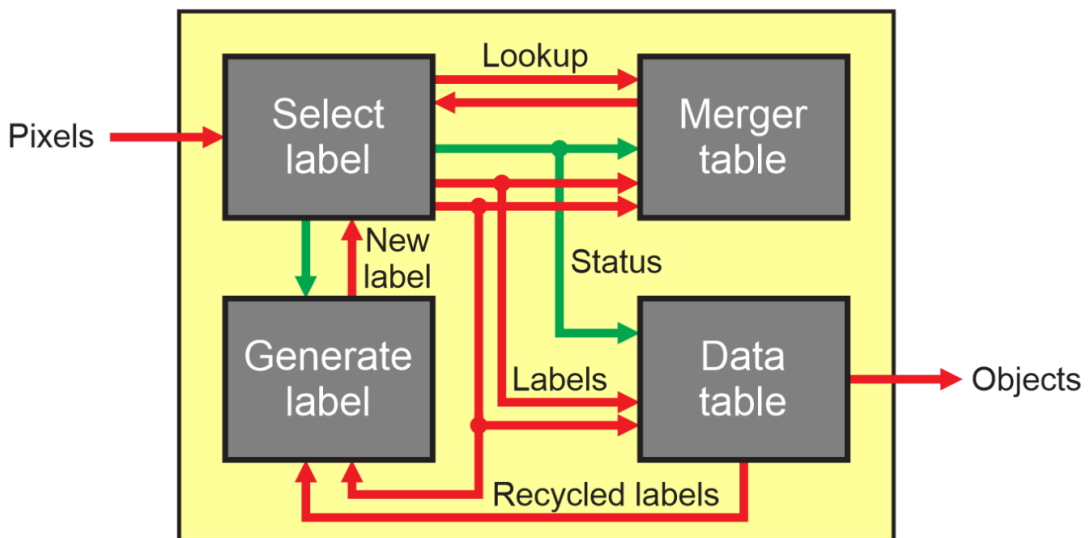


Figure 5-17 - Block diagram of the high-level architecture for connected components analysis (CCA) implemented in FPGA (Bailey, 2018). Reprinted with permission.

These pipelined processes were all used in the vision system tested in sections 5.2 and 5.3. As was seen in the experimental results, they provide low latency in both novel event detection and steady state latency.

The combination of the morphological erosion and dilation effectively removes the noise pixels and replaces the eroded pixels in the detected ball. The connected components analysis detects individual blobs and creates the bounding box for the detected ball. Without connected components analysis, the bounding box was created around the entire detected region from the colour thresholding. After the addition of connected components analysis, even with multiple separate objects in the detected region, the ball could still be identified.

6. Compute subsystem

This chapter will discuss the computational system including the hardware, software and control strategy. A detailed description will be given regarding the methodology for high-speed communication between the FPGA and ARM cores. The hardware-based motor control modules will also be discussed in detail.

The overall flow of data for one image capture → system response cycle is shown in Figure 6-1.

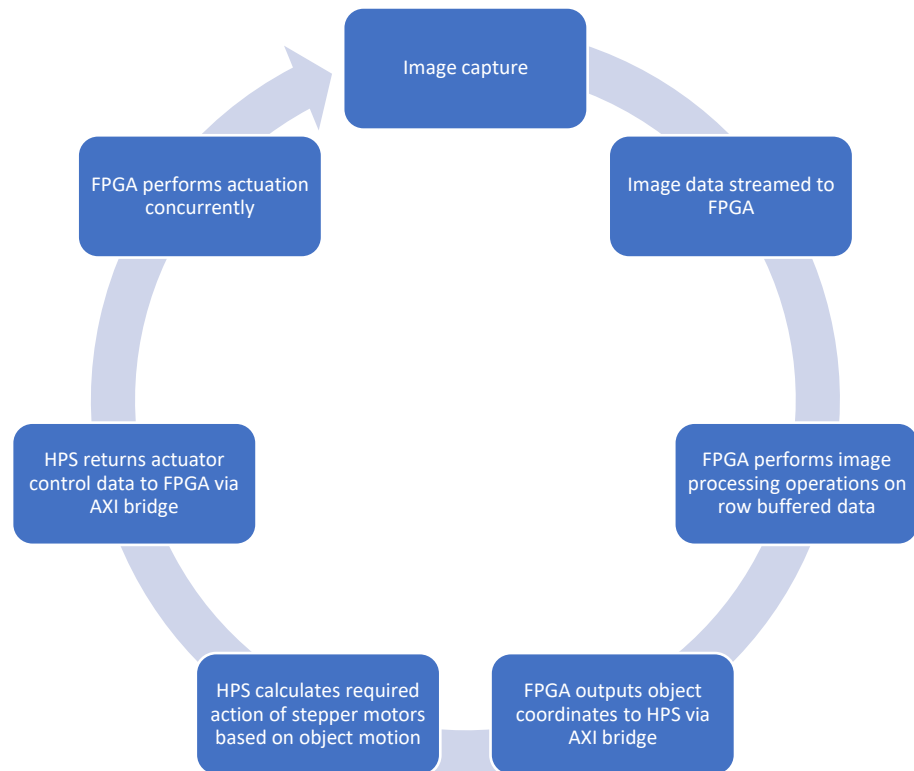


Figure 6-1 - Representation of vision-actuation control system data cycle from input to response

The data cycle for the entire compute system begins with image capture and processing, discussed in section 5.6.6. Once the ball has been successfully detected, its coordinates (x_{min} , x_{max} , y_{min} , y_{max}) are passed to the ARM core in the HPS via the high bandwidth AXI bridge. The HPS then performs the lens distortion calibration mapping, calculates the interception points for each automated module, and then returns the required actuation data to the FPGA. The motor control modules within the FPGA then output the required waveforms to control the 8 stepper motors concurrently. The data transfer methods, interception calculations, and motor control methodology will be described in detail in the following sections.

6.1. Terasic DE1-SoC FPGA development board

The DE1-SoC is a development board by Altera with a system on chip (SoC) which contains an FPGA and an 800 MHz Dual-core ARM Cortex-A9 MPCore processor on the same die, connected by a high-bandwidth communication interface. The interface consists of two high performance bridges which can

be configured in 32, 64 and 128-bit data width configurations, and one lower performance bridge which can only operate in 32-bit data width configuration. The two high performance bridges are called the FPGA to HPS bridge (F2H) and HPS to FPGA bridge (H2F). The lower performance bridge, mostly used for control and status registers and passing small data packets and commands, is called the lightweight HPS to FPGA bridge (LWH2F). The H2F, F2H and LWH2F bridges are all shown, including their master/slave capabilities, in Figure 6-2. The “L3 interconnect” in Figure 6-2 refers to the L3 cache within the HPS component.

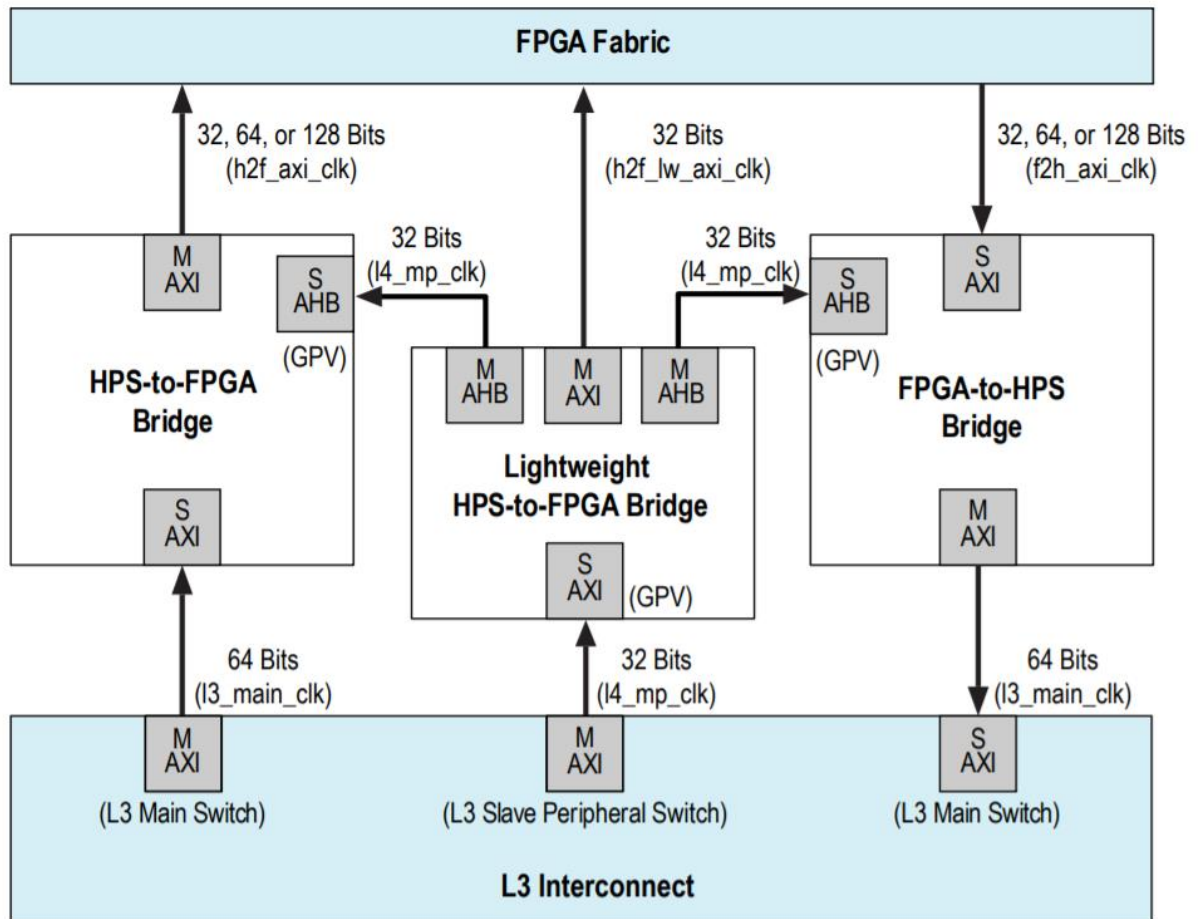


Figure 6-2 - Block diagram showing the master/slave relationship for each of the bridges between the FPGA and HPS (Altera, 2016).

The Global Programmers View (GPV shown in Figure 6-2), gives the programmer control over the behaviour of the three bridges through the lightweight HPS to FPGA bridge. In this design, the H2F bridge has been used for bidirectional data transfer, while the LWH2F bridge is used for commands and control/status registers. This will be detailed in later sections.

6.1.1. FPGA component

The FPGA chip consists of 85,000 logic elements, 4,450 Kbits of embedded memory, 457 pins, 87 DSP blocks, 6 fractional PLLs, and 4 DLLs.

The FPGA component of the system is programmed either with a .SOF file from the Quartus software, a .RBF file loaded by the Linux OS, or a .JIC file using the EPCQ programmer device which automatically loads a specified file into the FPGA portion of the chip when the board is powered up. The EPCQ chip is originally programmed from within the Quartus software.

For this design, the Quartus programmer is used to load the .SOF file onto the FPGA. This is done via the Intel FPGA Monitor Program, which accesses the Quartus programmer to load the .SOF file when the HPS C project is first opened.

6.1.2. HPS component

The HPS portion of the DE1-SoC consists of:

- 800 MHz Dual-core ARM Cortex-A9 MPCore processor
- 1GB DDR3 SDRAM (32-bit data bus)
- 1 Gigabit Ethernet PHY with RJ45 connector
- 2-port USB Host, normal Type-A USB connector
- Micro SD card socket

The HPS is either programmed with using the Intel FPGA Monitor Program or using the bootloader built into the Linux distributions provided by Altera and Terasic. There are several different distributions of Linux which run on the ARM core and they vary in size, speed and user interface – from simple command line to full desktop GUI.

In this design, the Intel FPGA Monitor Program is used to load bare-metal C code onto the ARM core. This is done after the .SOF file has been loaded onto the FPGA portion of the SoC. Using the Intel FPGA Monitor Program enables cross-compilation of the C code for the HPS. Native compilation requires an OS to be running on the system. This introduces two main problems:

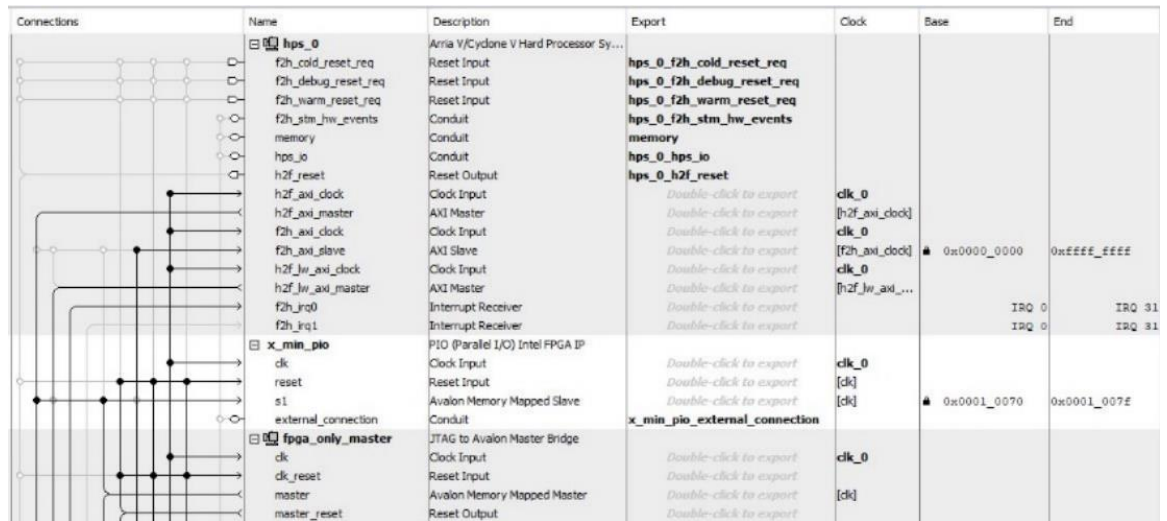
1. In the development process, if the C code causes a major error that freezes the ARM core, the entire device needs to be restarted to clear the error
2. Cross compilation is faster on a desktop PC due to the much more powerful processor present in the PC, than the 800 MHz dual core ARM processor

Bare-metal C code was selected instead of running one of the Linux operating systems available. This was done to avoid any additional latency caused by the OS, and to avoid previously mentioned development issues introduced by native compilation.

6.1.3. SoC design

The HPS is instantiated into the FPGA design using the Qsys GUI, by following the design guidelines provided by Intel (2017a). Other methods are possible such as reverse engineering the pre-built design from the Altera University Program, called “DE1-SoC Computer”. This is a less effective approach than the bottom up method of starting with a blank project and only instantiating the necessary components.

That was the method used for this project. The complete Qsys design is shown in Appendix C.



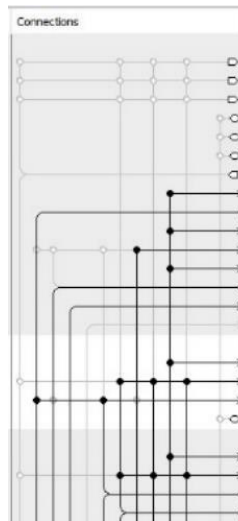
Connections	Name	Description	Export	Clock	Base	End
	hps_0	Arria V/Cyclone V Hard Processor Sy...				
	f2h_cold_reset_req	Reset Input	hps_0_f2h_cold_reset_req			
	f2h_debug_reset_req	Reset Input	hps_0_f2h_debug_reset_req			
	f2h_warm_reset_req	Reset Input	hps_0_f2h_warm_reset_req			
	f2h_stm_hw_events	Conduit	hps_0_f2h_stm_hw_events			
	memory	Conduit	memory			
	hps_io	Conduit	hps_0_hps_io			
	h2f_reset	Reset Output	hps_0_h2f_reset			
	h2f_axi_clock	Clock Input	clk_0	clk_0		
	h2f_axi_master	AXI Master	[h2f_axi_clock]			
	f2h_axi_clock	Clock Input	clk_0	clk_0		
	f2h_axi_slave	AXI Slave	[f2h_axi_clock]		0x0000_0000	0xFFFF_FFFF
	h2f_lw_axi_clock	Clock Input	clk_0	clk_0		
	h2f_lw_axi_master	AXI Master	[h2f_lw_axi_...			
	f2h_irq0	Interrupt Receiver				IRQ 0
f2h_irq1	Interrupt Receiver				IRQ 0	
	x_min_pio	PIO (Parallel I/O) Intel FPGA IP				
clk	Clock Input	clk_0	clk_0			
reset	Reset Input	[clk]				
s1	Avalon Memory Mapped Slave	[clk]		0x0001_0070	0x0001_007F	
external_connection	Conduit	x_min_pio_external_connection				
	fpga_only_master	JTAG to Avalon Master Bridge				
clk	Clock Input	clk_0	clk_0			
clk_reset	Reset Input	[clk]				
master	Avalon Memory Mapped Master	[clk]				
master_reset	Reset Output					

Figure 6-3 - Example Qsys HPS component

Figure 6-3 shows the top component in the Qsys project, the HPS component. For explanation on the Qsys tool, Altera University Program documentation can be consulted (Intel, 2017b). The important aspects of the HPS instantiation will be discussed, however.

On the far left-hand side, under the connections tab, the black lines show which components are connected to various parts of the HPS components. The important parts of the HPS component are the communication bridges: h2f_master, f2h_slave, and the lw2f.

To transfer data between the FPGA and HPS, a communication interface is required in conjunction with the bridges. These communication interfaces are called PIOs (parallel input/output). As is shown in Figure 6-4, when creating a PIO various settings data width, direction (input, output or bidirectional), reset vector and interrupt options need to be set. In Figure 6-4 the settings for a 16-bit input into the HPS are shown. This input (*x_min_pio*) is used to transfer one of the 4 object coordinate data values from the FPGA to the HPS. The other 3 are *x_max_pio*, *y_min_pio*, and *y_max_pio*.

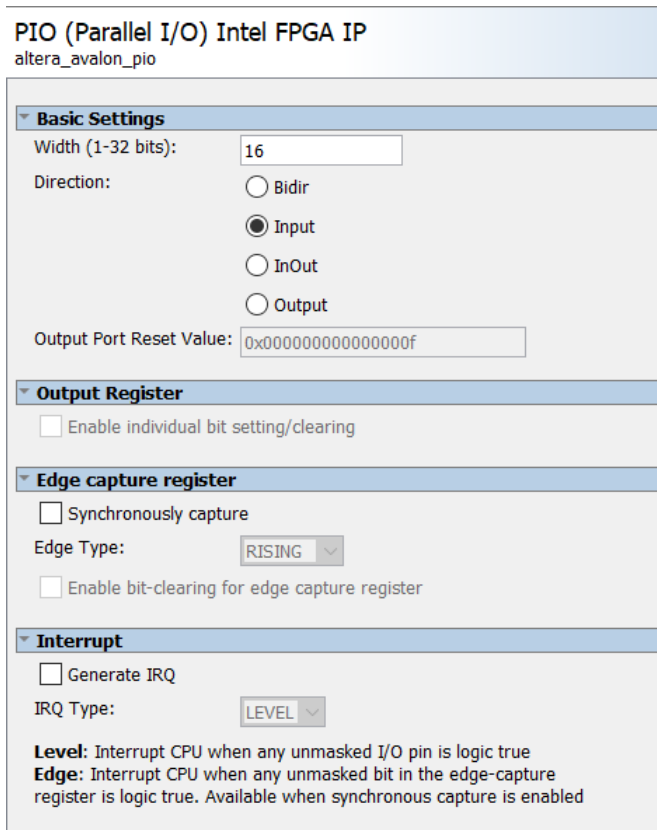


Figure 6-4 - Setup options for PIO in Qsys

Once these settings have been set, the internal connections are then made. This step determines which bridge the FPGA and HPS will communicate over.

clk	PIO (Parallel I/O) Intel FPGA IP	Double-click to export	clk_0		
reset	Clock Input	Double-click to export	[clk]		
s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0001_0070	0x0001_007f
external_connection	Conduit	x_min_pio_external_conec...			

Figure 6-5 - PIO connections and addresses in Qsys

As can be seen in Figure 6-5, the row highlighted in blue shows the connections on the left-hand side, and the memory start and end addresses. The two connections in Figure 6-5 are to the h2f_master, and to the f2h_slave. The values transmitted via the PIOs are accessed in the HPS by reading the values stored at the corresponding memory address. This is shown in Appendix D, which shows the interrupt service routine (ISR) which runs once for each new frame of image data received.

The return of data from the HPS to the FPGA is also performed with a PIO, however in the settings section, the direction is set to “output”. The HPS then writes to the address, an example of this is shown below in Figure 6-6, and the FPGA reads this data by assigning the value in the component instantiation of the HPS to an internal signal in the FPGA design. This is shown in Figure 6-7.

```

printf("dis step strobe\n en HPS_ACTIVE \n");
*(stepCmd_0_ptr) = 0b00000000; // write all stepper strobe signals low
*(HPS_ACTIVE_ptr) = 0xF; // tell the FPGA that the HPS is active

```

Figure 6-6 - C code used to send an "active" signal to the FPGA, and disable all the stepper motors temporarily

```

signal stepCmd : std_logic_vector(7 downto 0) := "00000000";
signal HPS_ACTIVE : std_logic;

--inside HPS component instantiation
step_cmd_0_external_connection_export      => stepCmd,
hps_active_external_connection_export      => HPS_ACTIVE,

```

Figure 6-7 - VHDL code to receive data/signals from the HPS

The signals "stepCmd" and "HPS_ACTIVE" are then used to drive the stepper motors and enable the interception modules, respectively. A simplified block diagram of the data transfer method is shown below in Figure 6-8.

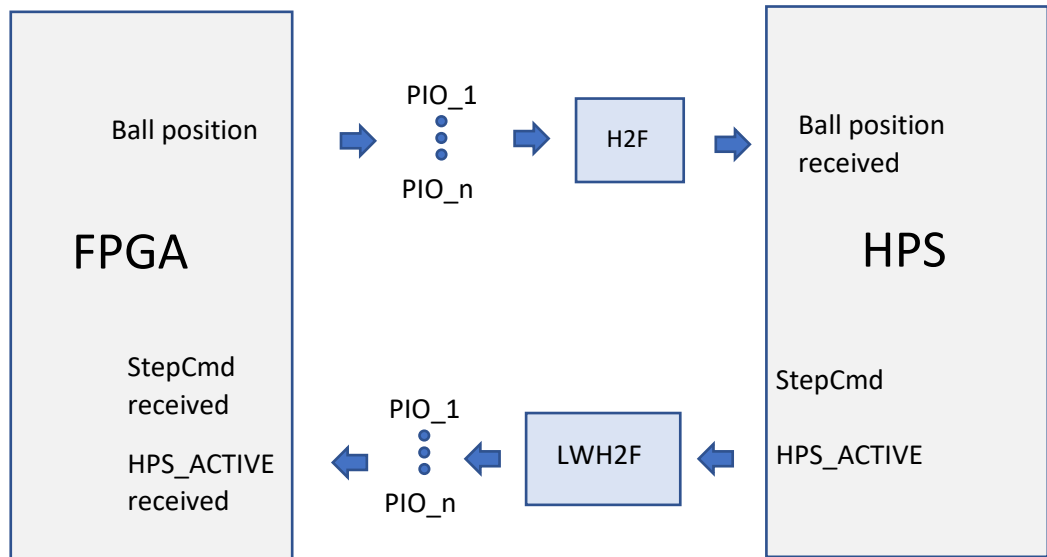


Figure 6-8 – Representation of data transfer method between FPGA and HPS

Any of the three bridges can be used for data transfer in either direction between the FPGA and HPS, however Intel recommends using the LWH2F bridge for control and status registers, and the "heavy weight" bridges (H2F and F2H) for data transfer (Intel, 2017a). This methodology has been used for the design of this control system. The following figure shows the address map and masters/slaves within the Qsys design of the SoC. The F2H bridge allows the FPGA master access to the HPS peripherals, while the H2F and LWH2F bridges provide the HPS master access to the FPGA peripherals (Altera, 2016).

System Contents		Address Map		Parameters	
System: soc_system Path: fpga_only_master.master					
	fpga_only_master.master	hps_0.h2f_axi_master	hps_0.h2f_lw_axi_master	hps_only_master.master	
HPS_ACTIVE.s1	0x0001_0110 - 0x0001_011f		0x0001_0110 - 0x0001_011f		
RS232_HPS_CMD.s1	0x0001_00d0 - 0x0001_00df	0x0001_00d0 - 0x0001_00df			
RS232_HPS_CMD_RETURN.s1	0x0001_0100 - 0x0001_010f		0x0001_0100 - 0x0001_010f		
RS232_HPS_CMD_ready.s1	0x0001_00e0 - 0x0001_00ef	0x0001_00e0 - 0x0001_00ef			
button_pio_0.s1	0x0001_0060 - 0x0001_006f		0x0001_0060 - 0x0001_006f		
frequency_from_HPS.s1	0x0001_0130 - 0x0001_013f		0x0001_0130 - 0x0001_013f		
hps_0.f2h_axi_slave				0x0000_0000 - 0xffff_ffff	
hps_cold_reset_pio.s1	0x0001_0010 - 0x0001_001f				
hps_debug_reset_pio.s1	0x0001_0030 - 0x0001_003f				
hps_warm_reset_pio.s1	0x0001_0020 - 0x0001_002f				
img_data_ready.s1	0x0001_0040 - 0x0001_004f		0x0001_0040 - 0x0001_004f		
intr_capturer_0.avalon_slave_0	0x0003_0000 - 0x0003_0007				
jtag_uart.avalon_jtag_slave	0x0002_0000 - 0x0002_0007		0x0002_0000 - 0x0002_0007		
onchip_memory2_0.s1	0x0000_0000 - 0x0000_ffff	0x0000_0000 - 0x0000_ffff			
step_cmd_0.s1	0x0001_00b0 - 0x0001_00bf		0x0001_00b0 - 0x0001_00bf		
step_data_0.s1	0x0001_00c0 - 0x0001_00cf	0x0001_00c0 - 0x0001_00cf			
step_dir_all.s1	0x0001_00f0 - 0x0001_00ff		0x0001_00f0 - 0x0001_00ff		
step_pos_0.s1	0x0001_0050 - 0x0001_005f	0x0001_0050 - 0x0001_005f			
stepper_0_busy.s1	0x0001_0120 - 0x0001_012f		0x0001_0120 - 0x0001_012f		
sysid_qsys.control_slave	0x0001_0000 - 0x0001_0007		0x0001_0000 - 0x0001_0007		
x_max_pio.s1	0x0001_0080 - 0x0001_008f	0x0001_0080 - 0x0001_008f			
x_min_pio.s1	0x0001_0070 - 0x0001_007f	0x0001_0070 - 0x0001_007f			
y_max_pio.s1	0x0001_00a0 - 0x0001_00af	0x0001_00a0 - 0x0001_00af			
y_min_pio.s1	0x0001_0090 - 0x0001_009f	0x0001_0090 - 0x0001_009f			

Figure 6-9 - Address map for Qsys SoC design

As is shown in Figure 6-9, numerous PIOs have been instantiated in the Qsys design. The PIOs used in this design perform communication and command tasks between the FPGA and the HPS according to Table 6.1.

Table 6.1 - System PIOs and their respective functions

PIO name	Direction	Function description
HPS_ACTIVE	HPS->FPGA	Signal for HPS to inform FPGA that HPS is running
RS232_HPS_CMD	FPGA->HPS	Command pass though for user input RS232
RS232_HPS_CMD_ready	FPGA->HPS	Interrupt generation signal for RS232 data
RS232_HPS_CMD_RETURN	HPS->FPGA	Generic channel to send any command to FPGA from HPS
button_pio	FPGA->HPS	PIO to interrupt HPS with pushbuttons
frequency_from_HPS	HPS->FPGA	PIO to send stepper motor frequency to FPGA
hps_0.f2h_axi_slave	HPS->FPGA	Channel for HPS to master F2H
hps_cold_reset	FPGA->HPS	Signal for FPGA to reset HPS
hps_debug_reset	FPGA->HPS	Signal for FPGA to reset HPS
hps_warm_reset	FPGA->HPS	Signal for FPGA to reset HPS
img_data_ready	FPGA->HPS	Signal for FPGA to interrupt HPS when each new frame of image data is ready
intr_capturer_0.avalon_slave_0	FPGA->HPS	Interrupt capturer module for FPGA to HPS interrupts
jtag_uart.jtag_avalon_slave	User Input	User input for master access to bridges
onchip_memory2_0	Both	On chip memory for memory mapped devices – bridges, peripherals etc
step_cmd_0	HPS->FPGA	PIO for HPS to send strobe signal to initiate stepper module operation
step_data_0	HPS->FPGA	PIO for HPS to send distance for stepper modules
step_pos_0	FPGA->HPS	PIO for stepper module to send distance values to HPS – used primarily for homing sequence
stepper_0_busy	FPGA->HPS	PIO for stepper motor busy flags from FPGA to HPS
sysid_qsys.control_slave	Internal	System ID PIO for checking correct function of bridge
x_max_pio	FPGA->HPS	PIO for maximum x coordinate of ball - send to HPS
x_min_pio	FPGA->HPS	PIO for minimum x coordinate of ball - send to HPS
y_max_pio	FPGA->HPS	PIO for maximum y coordinate of ball - send to HPS
y_min_pio	FPGA->HPS	PIO for minimum y coordinate of ball - send to HPS

6.2. Motor Control

This section will discuss the algorithms, mathematics, methodologies, hardware and software used to implement FPGA-based stepper motor control. Figure 6-10 shows the order of the development process of the FPGA-based motor control modules.

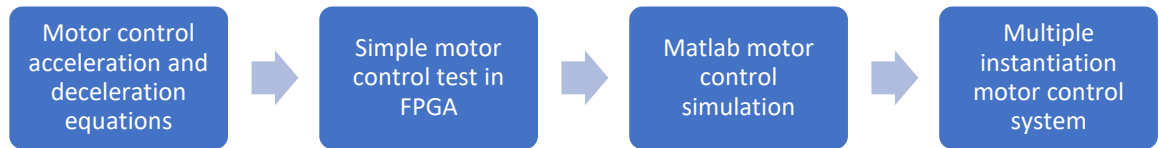


Figure 6-10 - Development process for the motor control design

As can be seen in Figure 6-10, the motor control algorithm mathematics were first simulated and developed within MATLAB. The resultant variable-acceleration ramping algorithm was then implemented in an Arduino microcontroller for qualitative performance testing relative to a trapezoidal (constant acceleration) ramping curve. The next phase involved testing a simple motor control module in FPGA hardware which accepted inputs such as distance, direction and speed and ran the motor at the specified velocity, for the specified distance, in the specified direction. The variable-acceleration ramping algorithm was then implemented in FPGA. Finally, multiple modules were implemented and tested in FPGA to establish resource requirements, compilation times and correct functionality. The motor control modules were developed in parallel with the SoC system design.

6.2.1. Background

For the semi-automated foosball table, stepper motors were selected as the actuators due to the combination of the ability to run in a semi-open loop, and their power density. A stepper motor is a DC driven electric motor which operates in discrete “steps”. The following data were considered important for the design of the semi-automated foosball table:

1. Torque vs speed curve
2. Voltage range
3. Current at a particular voltage – rated current

The selected stepper motors are Nema 23, bipolar stepper motors with 200 steps per rotation, maximum voltage of 40 V DC, and 2.8 A per phase.

The selected stepper drivers have 3 inputs, 5 A current delivery and are based on the TB6600 driver chip. The drivers are capable of up to 1/16th steps. For the semi-automated foosball table, full stepping is used for the linear sliding motion, and half stepping is used for the kicking motion. These settings correspond to 200 and 400 steps per revolution of the stepper motor, respectively. Figure 6-11 shows the stepper motor and driver used.

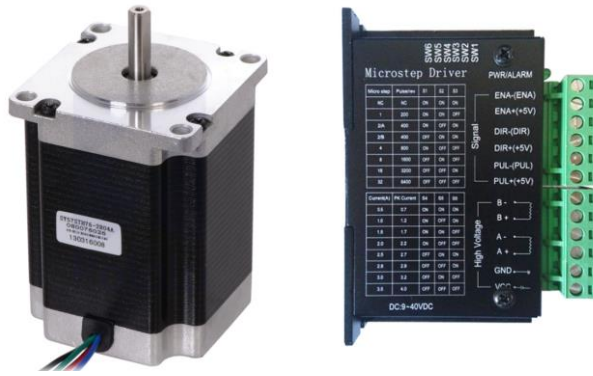


Figure 6-11 - Nema 23 bipolar stepper motor - 200 steps per revolution

6.2.2. Mathematical approximation

To control a stepper motor, a pulse generation system must be used. This can be a simple constant rate square wave, or it can be more complex, incorporating velocity changes, or even varying acceleration to smoothly change the velocity. To achieve the fastest acceleration from a stepper motor, the acceleration of the pulse rate should match the torque curve of the motor being used. Figure 6-12 shows the torque curve of the motors being used in this application. The following section will discuss the approximation used to match the acceleration parameters of that curve, and the other considerations necessary to achieve maximum performance from the actuation modules.

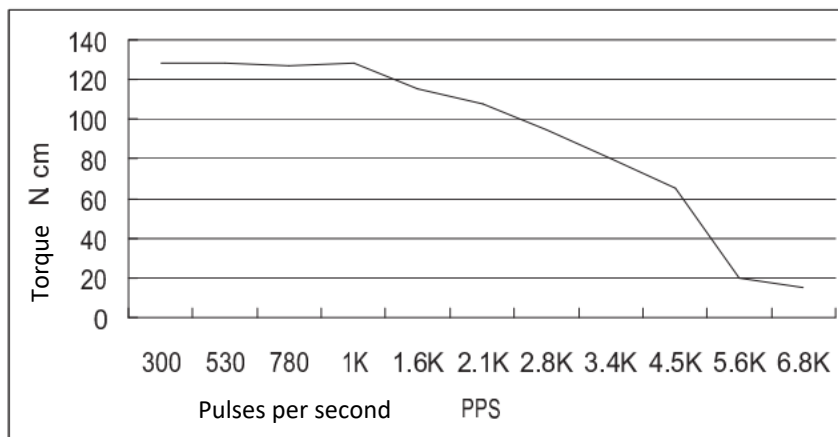


Figure 6-12 - Pull out torque curve of Nema 23 bi-polar stepper motor

In the semi-automated foosball table, stepper motors drive belt assemblies which either perform the kicking motion (rotation to rotation) or the sliding motion (rotation to linear). For the sliding motion, used for intercepting the ball, the motors must overcome the static friction to initiate the sliding and kicking movement. For interception, however, there is constant sliding friction of the outer shaft over the inner shaft, separated by acetal bushings, which the motor must overcome. For many materials, as the sliding speed increases, given a constant normal force, the coefficient of friction decreases (Chen, Kato, & Adachi, 2002; Chowdhury, Khalil, Nuruzzaman, & Rahaman, 2011). This implies that as the sliding speed of the interception rod increases, the amount of torque required to accelerate the rod may

decrease. Given that the motor torque decreases with speed, there may be some cancellation of effects caused by these two occurrences. The acceleration curve used by the stepper motors on the semi-automated foosball table, therefore, had to be empirically derived.

6.2.3. Stepper motor basic control algorithm

The basic stepper motor control algorithm takes the target position (number of steps) and automatically determines the ramping algorithm required. If the input distance is below 800 steps, the motor does not reach full speed in the time available therefore for the entire cycle of the stepper motor travel, it is either accelerating or decelerating. If input distance is greater than 800 steps, the motor can reach a maximum “cruise” speed. During this cruise phase the motor is neither accelerating nor decelerating. This is a potential time where the interception destination of the module could be changed on-the-fly, however this has not been implemented to improve reliability in the testing phase. Future work could include this feature. The following figure shows the MATLAB code used to test the functionality of the acceleration and deceleration algorithm used later in the FPGA implementation.

```

distance = 1600;    %input distance
stepsTaken = 0;    %keep track of where we are in step cycle
clcArray = zeros(distance); %array to record clock cycles at each step increment
stepFreq = 50000;  %initial stepping frequency
accelerator = 20;  %accelerator initial value
accelIncrement = 1; %integer to increment or decrement accelerator value
c = 0;            %counter for when to take a step (when reaches stepFreq value)
c2 = 0;          %counter for when to incr/decr accelIncrement
i=0;             %counter to keep track of clock cycles
distThresh = 800; %distance threshold for cruise or not

while(stepsTaken < distance)
    i = i+1; %% equal to clock cycles
    c = c+1; %% counter used to take a step on overflow
    if c == stepFreq
        fprintf('toggled\n');
        c=0;
        stepsTaken = stepsTaken + 1; %% increment step counter
        clcArray(stepsTaken) = i; %% clock cycle val at current step converted to sec
        if distance < distThresh
            if stepsTaken <= distance/2
                stepFreq = stepFreq - accelerator;
                c2 = c2 + 1;
                → if c2 == 5 %% every 5 steps increase the rate of acceleration
                    accelerator = accelerator + accelIncrement;
                    c2 = 0;
                end
            end
            if stepsTaken > distance/2
                stepFreq = stepFreq + accelerator;
                c2 = c2 + 1;
                → if c2 == 5 %% every 5 steps decrease the rate of acceleration
                    accelerator = accelerator - accelIncrement;
                    c2 = 0;
                end
            end
        end
    end
end
end

```



```

if distance >= distThresh
    if (stepsTaken <= (distThresh/2)) && (stepsTaken < (distance-400))
        → stepFreq = stepFreq - accelerator;
        c2 = c2 + 1;
        if c2 == 5 %% every 5 steps increase the rate of acceleration
            accelerator = accelerator + accelIncrement;
            c2 = 0;
        end
    end
    if stepsTaken > (distance-400)
        → stepFreq = stepFreq + accelerator;
        c2 = c2 + 1;
        if c2 == 5 %% every 5 steps decrease the rate of acceleration
            accelerator = accelerator - accelIncrement;
            c2 = 0;
        end
    end
end
end
end
end
end
end

```

Figure 6-13 - Acceleration and deceleration equations for stepper motor ramping

In Figure 6-13, the code to increment and decrement the pulse timer are shown. The lines of code that increment and decrement the overflow counter are marked with green arrows for the acceleration phase, and red arrows for the deceleration phase. The first two arrows are for the condition where the input distance is less than 800, and the second two arrows are for when the distance is greater than or equal to 800.

The parameters used in this code are not the exact parameters used in the FPGA stepper modules. As will be shown in the next section, those parameters and the starting speed were empirically derived. Figure 6-14 and Figure 6-15 show example outputs of the time period between steps “period”, and the frequency of stepping “velocity”. They are inverse of each other, in accordance with

$$T = \frac{1}{f} \quad (9)$$

where T is period, and f is frequency.

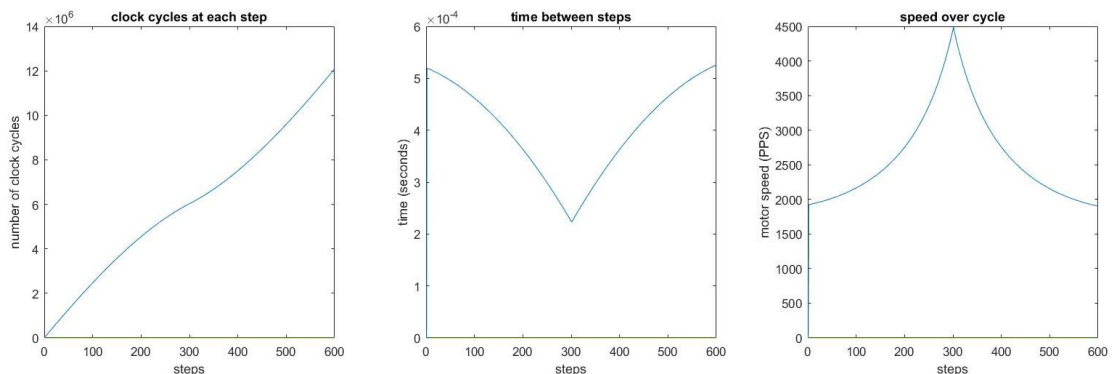


Figure 6-14 - Stepper motor simulation output curve for input distance = 600

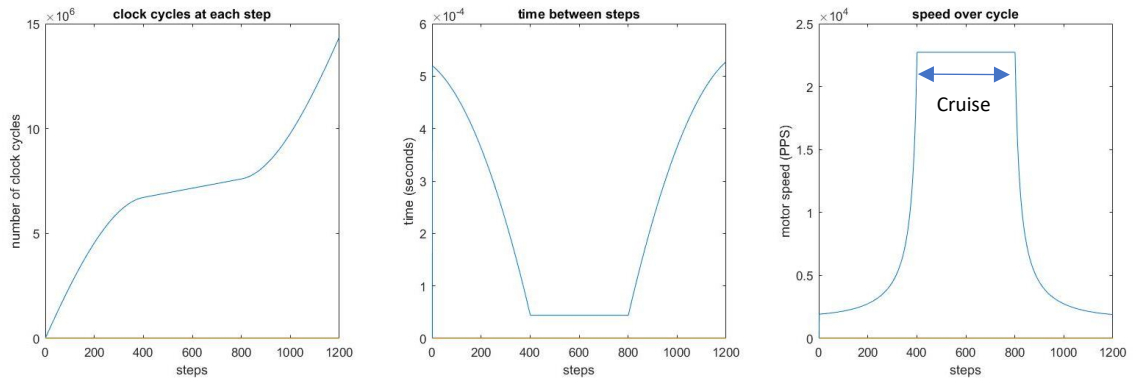


Figure 6-15 - Stepper motor simulation output curve for input distance = 1200

The “cruise” period, shown in Figure 6-15, was achieved because the input distance was greater than the 800-step threshold, therefore the motor could reach maximum speed.

For comparison, the 1200 step output has been replicated with a constant velocity model (no acceleration), and a trapezoidal ramping curve as done by (Wang et al., 2011). These output figures with execution times are shown in Figure 6-16 and Table 6.2, respectively.

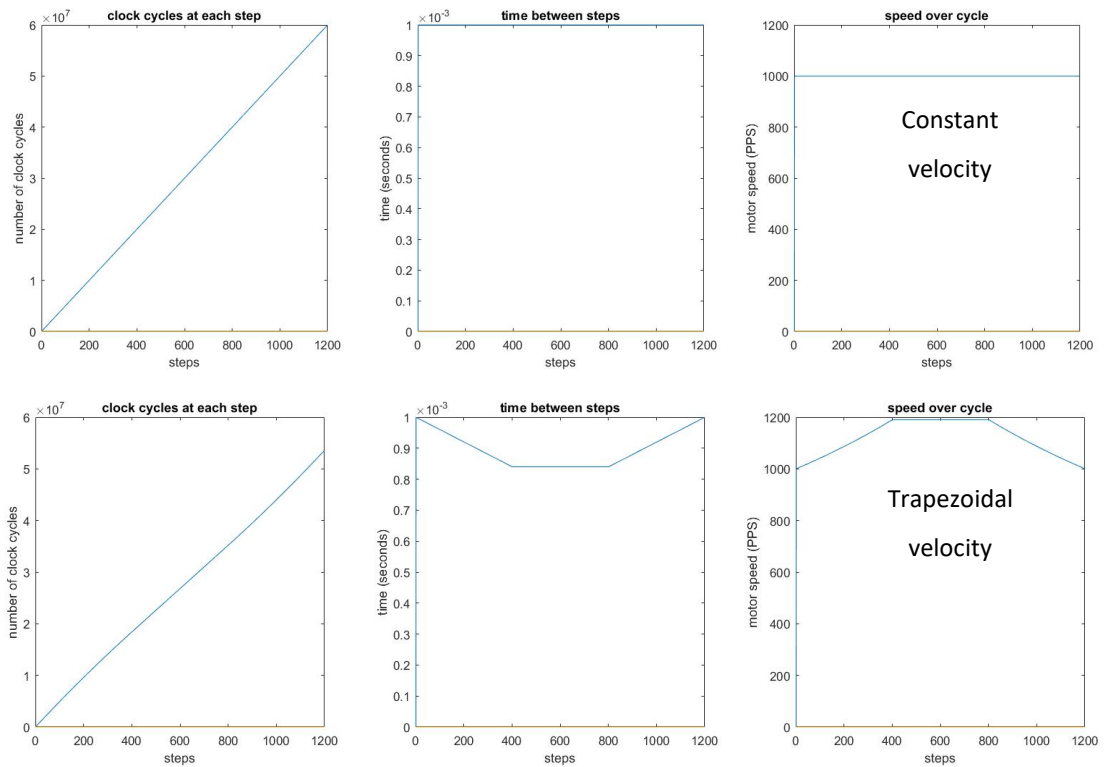


Figure 6-16 - Output curves for constant velocity (top 3) and trapezoidal velocity (bottom 3) profiles

Table 6.2 - Time taken and maximum velocity for 3 simulated stepper motor velocity profiles

Ramping type	Time taken for 1200 steps (s)	Maximum velocity (PPS)
Constant velocity (no ramp)	1.2	1000
Constant acceleration (trapezoidal ramp profile)	1.072	1190.5
Variable acceleration	0.8635	1908.4

The stepping speed was initialised at 1000 pulses per second for all 3 models. In the trapezoidal and variable acceleration profile, the acceleration increment was set to 20. In the variable acceleration model, the acceleration increment was set to 1, incrementing or decrementing every 5 steps taken, as shown in the simulation code.

For the constant acceleration model, the accelerator and incremter were both set to 0. While for the trapezoidal model, only the incremter was set to 0. As shown in Table 6.2, the variable acceleration achieves the fastest time for 1200 steps, and the highest top speed. Additionally, this type of curve offers the smoothest acceleration and deceleration, minimising jerks (Wang et al., 2011).

6.2.4. Stepper motor control on FPGA

In FPGA hardware, the variable acceleration ramping algorithm shown in Figure 6-13 was implemented. The main limiting factor is the difficulty to perform floating point calculations on FPGA, therefore the original simulation code was written only using integer counters to ensure the code could be correctly ported. A block diagram representing the required inputs and outputs for each stepper module is shown in Figure 6-17. The internal pulse generation method within the motor control module is then shown in Figure 6-18, however for simplicity some of the inputs and outputs such as the distance input, reset, stop command, homing switches, flags, and direction inputs and outputs have been omitted. The block diagram focusses on the pulse generation using the variable acceleration model outlined in Figure 6-13.

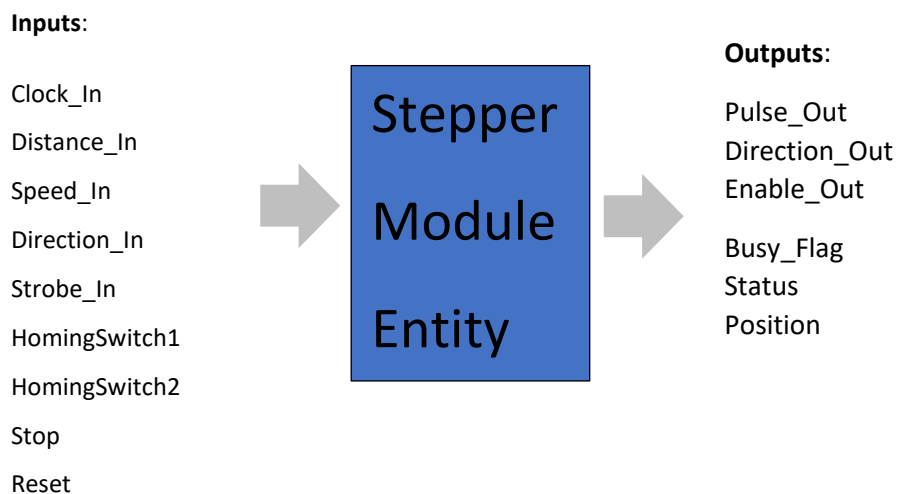


Figure 6-17 - Stepper module entity diagram showing inputs and outputs

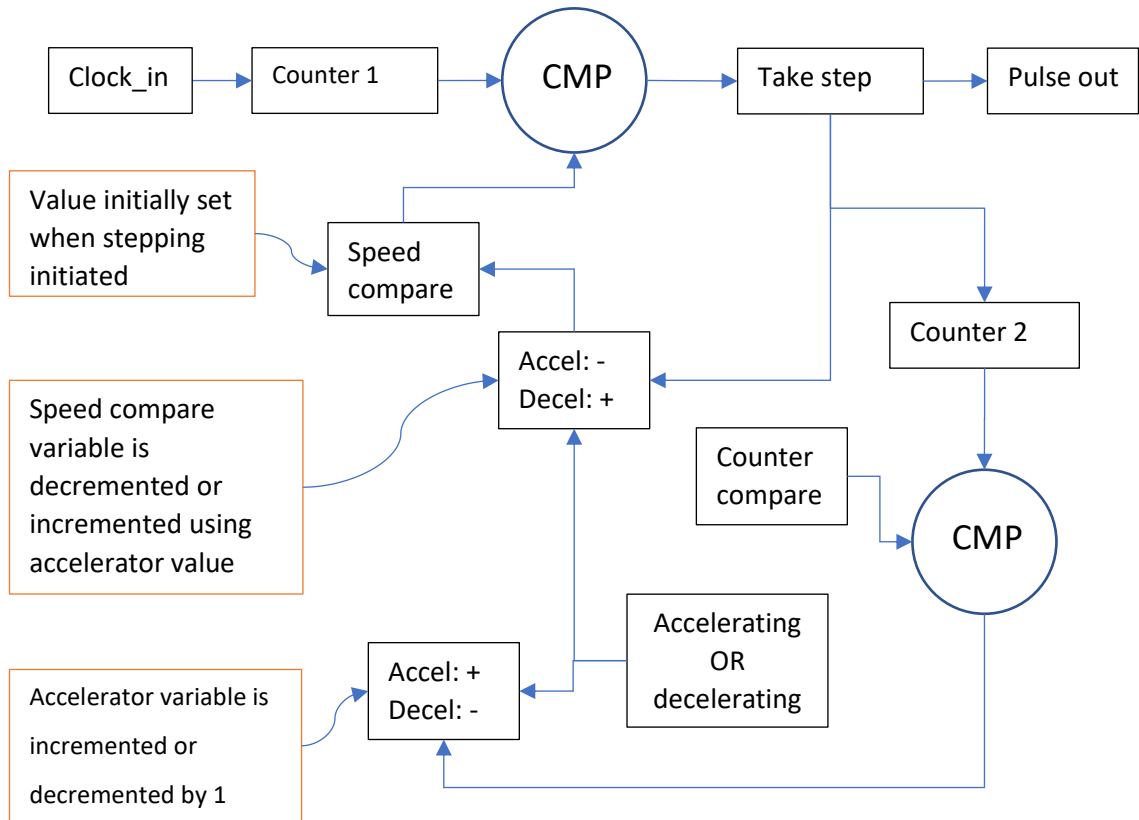


Figure 6-18 - Block diagram of simplified variable acceleration pulse generation method, implemented in FPGA

The key difference between the simulation and the FPGA realisation is that the number of steps taken by the motor is half the number of steps output by the simulation. This is because the stepper motor driver moves the motor by one step only on the rising edge of the pulse signal. Therefore 2 toggles of the pulse pin are required to perform a single step on the physical system. So, a pulse rate of 700 pulses per second in the simulation equates to 350 pulses per second on the physical system.

In Figure 6-18, the 50 MHz input clock drives the counter which is compared every clock cycle with the speed compare variable. If they are the same, the pulse output pin is toggled, and the second counter is incremented. When the second counter and the counter compare variable are equal, the accelerator (used to decrement or increment the speed compare variable each step) is incremented or decremented. The following table shows whether the accelerator and speed compare variables are incremented or decremented relative to the stepper motor cycle – acceleration, cruise, or deceleration.

Table 6.3 - Increment or decrement based on current stepper motor cycle position

Acceleration or deceleration	Speed compare	Accelerator
Acceleration	- accelerator	+1
Deceleration	+ accelerator	- 1

The VHDL code for the FPGA based stepper motor control algorithm with variable acceleration is shown in Appendix E and Appendix F. Appendix E shows the entity declaration, while Appendix F shows the architecture.

Wang et al. (2011) implemented a trapezoidal stepper motor velocity profile on an FPGA. Their control algorithm consumed 1276 logic elements. In our FPGA based variable acceleration ramping profile, each control module consumed an average of 1085 ALUTs, while also being completely parameterizable (variable speed, distance, direction, and the ability to create any of the 3 types of velocity profile), and including the homing functionality discussed in section 6.2.7.

While these resource utilisation values cannot be compared directly (because of differences in technology), they do give a reasonable comparison of the low resource requirements of our stepper motor control algorithm.

6.2.5. Determining acceleration parameters

Each of the 4 actuation modules on the semi-automated foosball table is slightly different, due to material imperfections, manufacture tolerances, bolt tightness, bearing friction, dirt build-up and many more factors. To compensate for this, the acceleration curve parameters such as initial frequency, secondary counter overflow value, accelerator initial value and increment value had to be determined by testing various parameters, finding the most aggressive ramping curve the module could tolerate, and then reducing this by a small margin as a factor of safety.

The starting frequency of around 350 motor pulses per second (speed counter initial value of 70000 and simulated initial pulse rate of 700 pulses per second) was chosen as this was the speed at which the motors produced the maximum torque. An initial value of 2 was selected for the accelerator, and 1 for the incrementer. The measured completion time for this cycle was around 2.9 seconds.

These values were progressively increased until the stepper motor began to skip steps. The combination of values at which this occurred was around 25000, 20 and 1 for speed counter, accelerator, and incrementer, respectively. The initial motor speed was 1000 pulses per second. The simulated completion time for this cycle was 0.8635 seconds. The measured completion time corresponded well to this value when steps were not skipped, however this was often not the case.

The starting frequency value was therefore dropped to improve reliability, and a starting frequency value of 27000 (corresponding to 926 motor pulses per second) was selected as this starting frequency reliably did not cause the system to miss steps. Further optimisation work could be done to improve the maximum speed of the modules, or to dynamically change the aggressiveness of the ramping profile depending on the input distance, however this was not included in the scope of this research.

6.2.6. Kicking algorithm

The gearing ratio from the stepper motor to the output shaft is a 1:1 ratio. So, one full rotation of the motor output shaft causes one full rotation of the kicking rod. For obvious reasons, the kicking algorithm

should accrue maximum momentum of the kicking rod before it collides with the foosball. Given that the rotational inertia of the rod doesn't change when the rod spins, in order to maximise angular momentum of the rod, maximum possible rotational velocity must be reached when the ball is kicked.

The rules of foosball state that a full 360° spin prior to or after kicking the ball is illegal. Therefore, this is the limiting factor in terms of maximum speed that can be achieved and therefore momentum that can be imparted by the kicking rod. The rod therefore is accelerated from 0 rad/s to maximum rotational speed possible in less than 360° of rotation, the foosman's "foot" collides with the ball, and then the rod is decelerated back to 0 rad/s in less than 360°.

The method to achieve this is as follows:

1. Intercept the ball and ensure it is stationary – done by storing the ball position values in a buffer and calculating the positional difference. When the difference is small enough, the ball is deemed to be stationary
2. Move the foosman's "foot" around the ball involving
 - a. Sliding to one side a sufficient distance to miss the ball (approximately 40mm)
 - b. Positive rotation sufficient angle to place foosman "foot" in front of the ball
 - c. Equal but opposite direction slide as was performed in step 2a
3. Accelerate and kick ball in under 360° of rotation – around 40 degrees of rotation required to miss the foosball – this leaves 320 degrees or less in which to accelerate and kick the ball
4. Foosman "foot" collides with ball, transferring as much kinetic energy as possible
5. Decelerate and stop the rod in ~359° (398 steps input to stepper module) of rotation

Considerations required for the algorithm

1. If the module is all the way to one side of the table, the rod may have to slide in the opposite direction to move around the ball
2. If the ball moves, the new position of the ball will need to be used as the location from which to kick the ball

By using the homing switches attached to the rotational assembly, combined with the fixed relationship between pulses and angle, the angle of the stepper motor and of the kicking rod is always known.

6.2.7. Homing and sliding algorithm

The gearing ratio of the sliding mechanism is 1:1, therefore only input steps to output linear distance had to be measured. The key information required by the module is the end limits for each of the modules, as they each have different work envelopes. Additionally, on initialisation of the semi-automated foosball table, the position of the rods is unknown to the system. Therefore, a method of calibrating each foosmen rod was required. This section will address that problem.

The objective of the homing sequence is for the modules to find the end limits for each rod. This is achieved by performing the following functions for each rod:

1. Step at low speed for a distance greater than the workspace (input distance greater than 2000 steps) in a predefined direction (let direction = 1)
2. When the switch on the corresponding end of the table is pressed, set the module output distance to 0, reverse the direction of travel (let direction = 0) and initiate a new stepping sequence with the same high distance of greater than 2000 steps
3. When the switch on the opposite side is eventually pressed, stop the module and output the distance reached *output_dist* to the HPS,
4. Finally, reverse the module to the centre of the table – this is the output distance $output_dist/2$ with the direction of travel set to the same as in step 1 (let direction = 1)

The code for this homing sequence is shown in Appendix G.

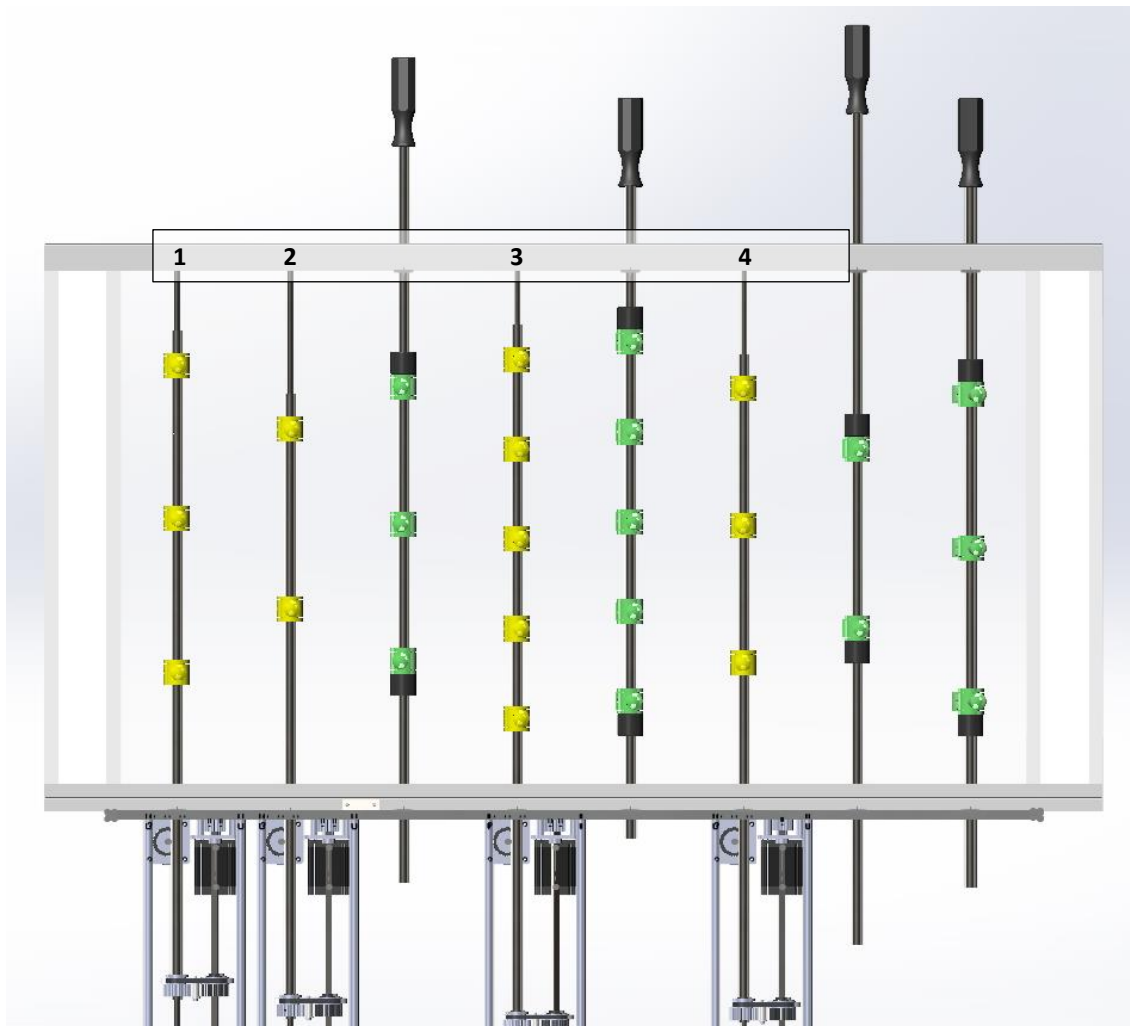


Figure 6-19 – Birds-eye view of foosball table CAD model with automated rods annotated

The total number of steps required for the total travel of each rod, which are numbered in Figure 6-19, are as follows:

Rod 1 – 241 mm = 964 steps

Rod 2 – 415 mm = 1660 steps

Rod 3 – 170 mm = 680 steps

Rod 4 – 287 mm = 1148 steps

The ratio of steps to mm is therefore 4:1. The total workspace for each of the rods (from one limit switch to another) is 690 mm, or 2760 stepper motor steps.

6.2.8. Interception algorithm and image to table spatial mapping

Once the image had been corrected for the radial distortion introduced by the wide angle lens, using the method discussed in section 5.6.5, the measured position of the ball needed to be converted to physical space for the interception calculations to be performed. Since both spaces are linear, this requires an offset and a scale factor.

These were obtained using the following steps:

1. Place ball at minimum Y position on the playing field – this corresponds to the human side of the table as close to the edge of the playing field as possible
2. Obtain lens distortion corrected ball coordinates – let this be Y_1 – this is equal to the offset
3. Place ball at maximum Y position on the playing field – this corresponds to the automated module side of the table as close to the edge of the playing field as possible
4. Record value – let this be Y_2
5. Calculate the difference between the two obtained Y values
6. Scale from image space (pixels) to the corresponding workspace (in stepper motor steps) – note that this is not the same as the total travel mentioned in section 6.2.7
7. Apply an offset in the Y direction such that the lowest position recorded for the ball is equal to 0 (this offset is equal in value and opposite in sign to the first recorded Y value)

The values obtained in this process were $Y_1 = 240$, $Y_2 = 912$.

As mentioned in the above steps, the total workspace for the module is equal to the width of the table in mm minus around 3mm (to compensate for the limit switches), converted to steps. This value is $D_T = 690 \text{ mm} \times 4 \frac{\text{steps}}{\text{mm}} = 2760 \text{ steps}$.

After offsetting the image Y values by -240 pixels the output map becomes

$$Y_{out} = (Y_{in} - Y_1) * \left(\frac{D_T}{Y_2 - Y_1}\right) \quad (10)$$

After correctly calculating the Y position of the ball, and applying the offset and scalar required to correct image coordinates into actuation module workspace, the interception points can be calculated.

For all 4 modules the following parameters are constant and pertain to Figure 6-20:

	$D_T = 690 \text{ mm} = 2760 \text{ steps}$	(11)
	$\text{foosman width } F_w = 32.5 \text{ mm}$	(12)
	$d_1 + d_2 + d_3 = d_4 + d_5 + d_6 = d_7 + d_8 + d_9 = d_{10} + d_{11} + d_{12} = 690 \text{ mm}$	(13)

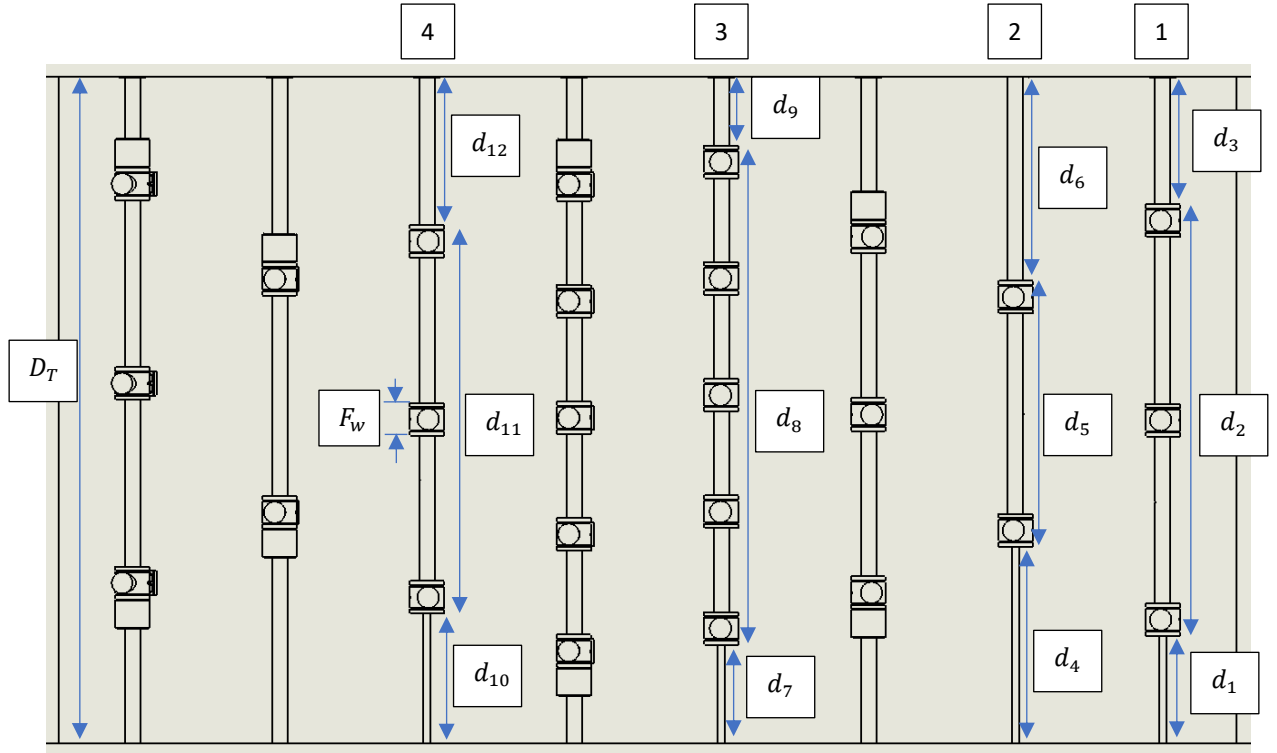


Figure 6-20 – Birds-eye view of foosball rods - Module 4 to 1 from left to right

Module 4	Module 3	Module 2	Module 1
$d_{10} + d_{12} = 287 \text{ mm}$	$d_7 + d_9 = 170 \text{ mm}$	$d_4 + d_6 = 415 \text{ mm}$	$d_1 + d_3 = 241 \text{ mm}$
$d_{11} = 690 - 287$	$d_8 = 690 - 170$	$d_5 = 690 - 415$	$d_2 = 690 - 241$
$d_{11} = 403 \text{ mm}$ $= 1612 \text{ steps}$	$d_8 = 520 \text{ mm}$ $= 2080 \text{ steps}$	$d_5 = 275 \text{ mm}$ $= 1100 \text{ steps}$	$d_2 = 449 \text{ mm}$ $= 1796 \text{ steps}$

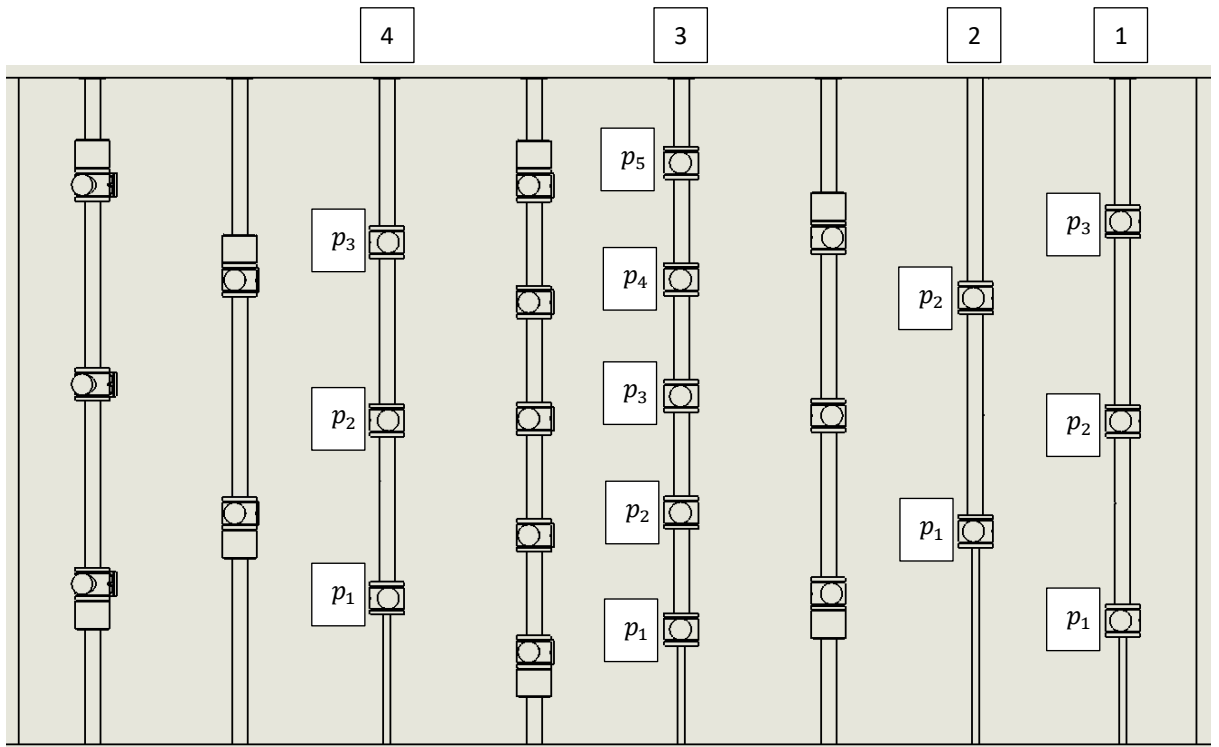


Figure 6-21 – Birds-eye view of foosball table with positions of each foosman on each module labelled

Module 4	Module 3	Module 2	Module 1
$p_1 = d_{10} + \frac{F_w}{2}$	$p_1 = d_7 + \frac{F_w}{2}$	$p_1 = d_4 + \frac{F_w}{2}$	$p_1 = d_1 + \frac{F_w}{2}$
$p_2 = d_{10} + \frac{d_{11}}{2}$	$p_2 = d_7 + \frac{d_8}{4} + \frac{F_w}{4}$	$p_2 = d_4 + d_5 - \frac{F_w}{2}$	$p_2 = d_1 + \frac{d_2}{2}$
$p_3 = d_{10} + d_{11} - \frac{F_w}{2}$	$p_3 = d_7 + \frac{d_8}{2}$		$p_3 = d_1 + d_2 - \frac{F_w}{2}$
	$p_4 = d_7 + d_8 * \frac{3}{4} - \frac{F_w}{4}$		
	$p_5 = d_7 + d_8 - \frac{F_w}{2}$		

With the above calculations for each module's position, and each foosman's position on each module, the intercept can now be calculated.

For brevity, example interception calculations will be shown for module 2. Figure 6-22 shows the C code section which calculates the difference between each foosman's current position and the ball's calibrated Y position.

```

int decide_possession = (int)(Ycor);
if (decide_possession > 1725)
    POSSESSION = 2;
else if (decide_possession < 1035)
    POSSESSION = 1;
else if ((decide_possession < 1725) && (decide_possession > 1035))
    POSSESSION = 3;
else
    POSSESSION = 0;

DELTA1 = (LIN_POS_GLOBAL+65) - Y2;
DELTA2 = (LIN_POS_GLOBAL+1035) - Y2;

if (POSSESSION == 1)
{
    DELTA_USE = (int)DELTA1;
}

else if (POSSESSION == 2)
{
    DELTA_USE = (int)DELTA2;
}

else if (POSSESSION == 3)
{
    if ((fabs(DELTA1)) < (fabs(DELTA2)))
    {
        DELTA_USE = (int)DELTA1;
    }
    else if ((fabs(DELTA2)) < (fabs(DELTA1)))
    {
        DELTA_USE = (int)DELTA2;
    }
}
}

```

Figure 6-22 - C code used to calculate differences between ball and foosmen positions

In Figure 6-22, the variable “decide_possession” is used to determine which foosman should take possession of the ball, depending on the ball’s coordinates. The differences between the ball’s Y coordinate (Ycor – the corrected Y position of the ball), and the two foosmen’s linear position is then calculated, and the possession decider determines which “delta” to use.

The values 1035 and 1725 represent the positions on the foosball table which correspond to the maximum reach of each foosman on rod 2. If the ball is at or below Y = 1035, only the first foosman can reach the ball. If the ball is at or above Y = 1725, only the second foosman can reach the ball. In between these values and either foosman can reach, therefore the closest foosman performs the intercept.

The relevant interception data is then sent from the HPS to the FPGA motor control modules with the direction in which to travel to intercept the ball. This is the non-predictive model. The predictive model uses multiple ball locations (X and Y) to calculate the velocity and heading angle of the ball. The intercepts are then calculated based on the heading angle and the appropriate foosman is used for interception.

Future work could factor the current motion of the foosmen rods into the equation and use the minimum time to intercept as the deciding variable for which foosman is used for interception. This would include the speed and direction of the module’s current motion and determine which foosman would take the least time to intercept the ball. The time-taken model was out of scope for this research, however.

6.3. Resource requirements

The final consideration with the SoC and motor control module design was resource utilisation and compilation time. The required elements in the complete SoC design are:

1. Vision – capture, process and display image
2. Code to send object coordinates to HPS
3. RS232 communication – user inputs
4. 8x motor control modules – 4x linear, 4x rotational
5. Homing code and limit switch reading – 12x limit switches total
6. HPS component and all required PIOs

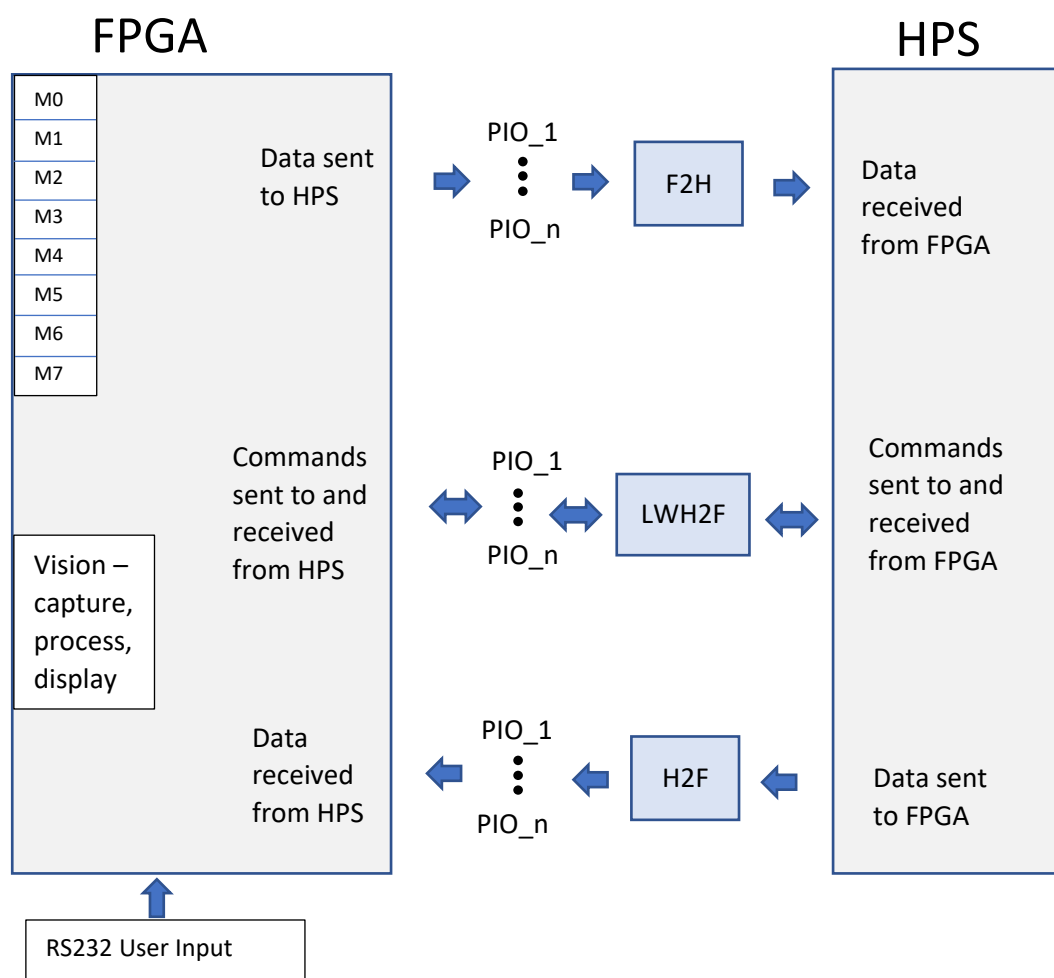


Figure 6-23 - Block diagram representing entire SoC system design

Figure 6-23 shows the high-level block diagram for the entire SoC system. M0 to M7 represent the 8 motor modules, with all the limit switch inputs.

In order to reduce compilation time, most of the development was done with a single motor module implemented. This resulted in a compilation time of around 11 minutes, and resource utilisation shown in Table 6.4.

Table 6.4 - Comparison of resource utilisation for minimal design through to full SoC design

Resource name	Vision only	Vision + HPS + 1 motor	Vision + HPS + 8 motors
Logic utilisation	1,259 (4 %)	7784 (24 %)	12, 437 (39 %)
Registers used	1,460	9636	10936
Pins used	151 (33 %)	311 (68 %)	347 (76 %)
Memory used	110,694 (3 %)	644,966 (16 %)	644,966 (16 %)
DSP blocks used	0 (0 %)	2 (2 %)	16 (18 %)

Table 6.4 shows the resource utilisation for the minimal vision system, the vision system with the HPS and one motor module instantiated, and the full SoC design with all 8 motor modules instantiated. Compilation time for the full system is around 15 minutes.

6.4. Summary

Overall, the FPGA and HPS pass data to one another with the FPGA effectively functioning as a hardware accelerator for the HPS, and the HPS providing a platform for implementing algorithms which would be difficult and inefficient in terms of resource requirements if implemented in hardware.

The FPGA is responsible for tasks which can utilise parallelism, such as vision, communication and motor control, while the HPS is responsible for tasks requiring algorithmic processing such as overall system operation, ball trajectory prediction and strategic gameplay control tasks.

7. System integration

This chapter will discuss all the steps required to integrate all the required subsystems including the vision, compute system, electronics, power delivery, mechanical components and other miscellaneous tasks. The process for developing the mechanical/electrical system is shown in Figure 7-1.

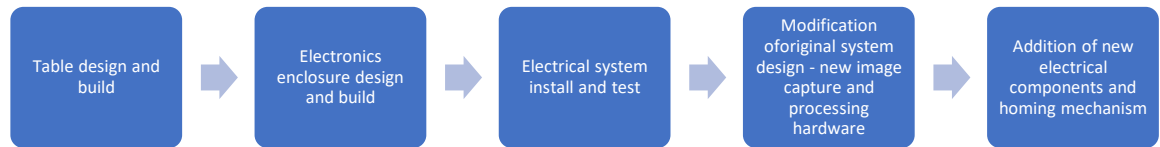


Figure 7-1 - Development process for the mechanical/electrical system

In a previous project we designed and built the foosball table and actuation modules. The electronics and power delivery were also implemented in that project along with a rudimentary object tracking and interception system. This system was based on the CMUcam5 tested in sections 5.2 and 5.3, which provided sub-optimal interception performance due to excessive system latency, and lack of lens distortion correction.

In this project the electronics enclosure was fitted with upgraded interface and service panels, improved development features, a calibration/homing jig and a completely redesigned image capture-processing-compute-control platform, as was discussed in previous sections.

7.1. Electrical system

7.1.1. Overall electrical system architecture

The architecture of the electrical system is represented by the block diagram shown in Figure 7-2.

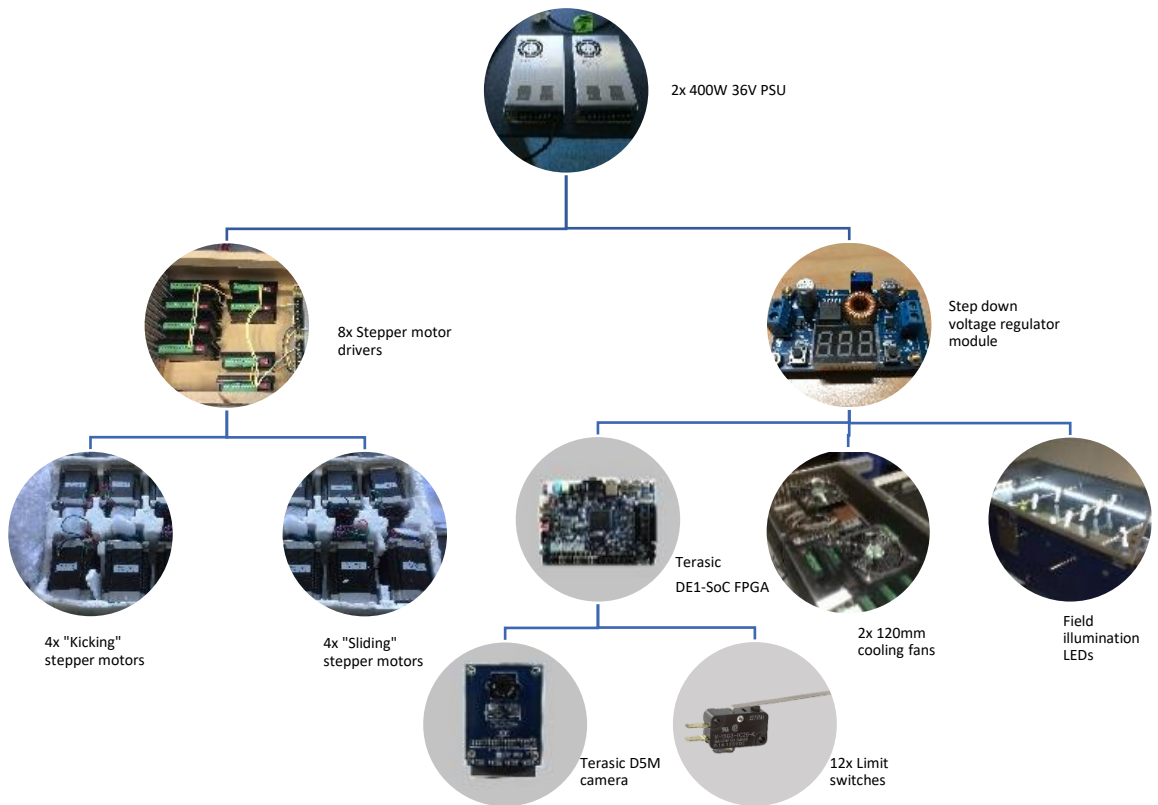


Figure 7-2 - Electrical system architecture for the semi-automated foosball table

The whole system is powered by two 36V DC, 400W power supplies in parallel providing a maximum total system power of 800W. The power supplies take one AC input and an external 10A fuse is in place at the system power switch shown in Figure 7-3.

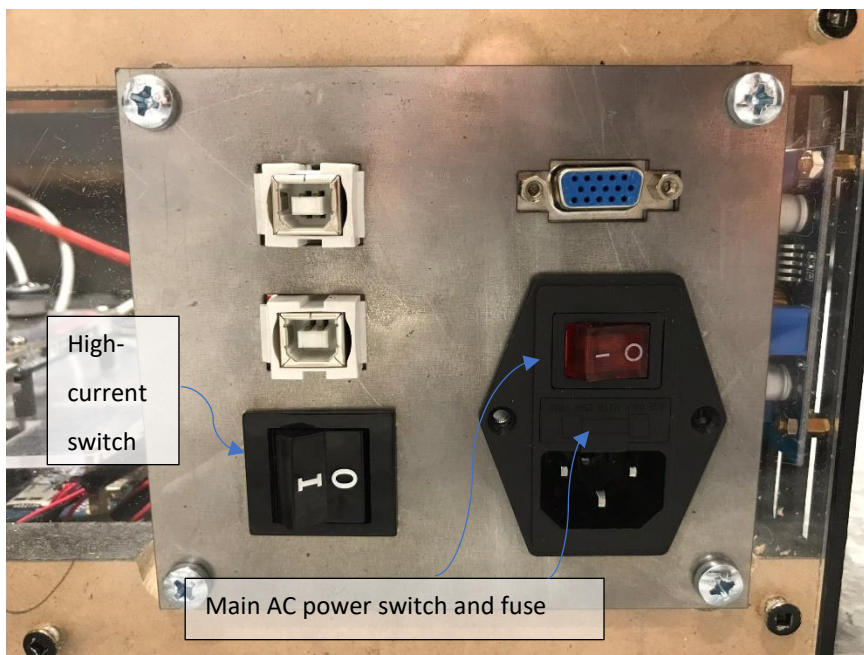


Figure 7-3 - Panel with 2 USB-B sockets, 1 D-Sub socket, 1 IEC socket, and a high-current switch

7.1.2. 36V stage

The 36V outputs are fed directly from the power supplies to the stepper motor drivers, thus the stepper motors are driven by 36V DC.

The stepper motors used are Nema 23 bi-polar stepper motors, as mentioned previously. These were selected for their high-power and high-torque from low speeds. This torque is required to accelerate the linear and kicking motion necessary for foosball. Additionally, stepper motors, when used correctly, offer high-precision open loop control.

The stepper motor drivers used are 5A peak, up to 40V DC, stepper motor drivers based on the TB6600 chip. They are capable of microstepping resolution up to 1/16 steps (3200 steps per revolution). They are used in full step mode (200 steps per revolution), however, as this setting offers the best acceleration characteristics. The drivers require 3 inputs; pulse, enable and direction.

After initial testing, it was determined that the ability to deactivate power to the stepper motors while retaining power to the compute unit was necessary, therefore the high-current switch, also shown in Figure 7-3, was added. This catered for situations when the stepper motors needed to be deactivated and the rods reset, due to debugging issues.

7.1.3. 12V stage

The following components are driven by the 12 V supply:

1. DE1-SoC FPGA
2. D5M camera
3. Field illumination LEDs

The voltage regulator module used is an XL4015 based module. With input voltage from 4.0V to 38V, and output from 1.25V to 36V, and maximum current of 5A. The module has a maximum power output of 75W with appropriate cooling. Due to the switching regulator chip used, this voltage regulator module produces far less waste heat than a linear regulator. Additionally, the module provides input and output voltage readouts which are useful for development.

The DE1-Soc requires a 12V DC input and consumes a maximum of 24W including the D5M camera and HPS.

The system runs two additional 120mm cooling fans, each of which can consume around 4W of power.

7.1.3.1. DE1-SoC 40 pin header expansion board

The 40-pin expansion board is a custom PCB with 8x pluggable 4 pin screw terminal blocks for the stepper motor drivers and 2x 5 pin pluggable screw terminal blocks for the linear and rotational homing switch inputs to the FPGA. This PCB design is shown in Figure 7-4 and the physical PCB is shown in Figure 7-5.

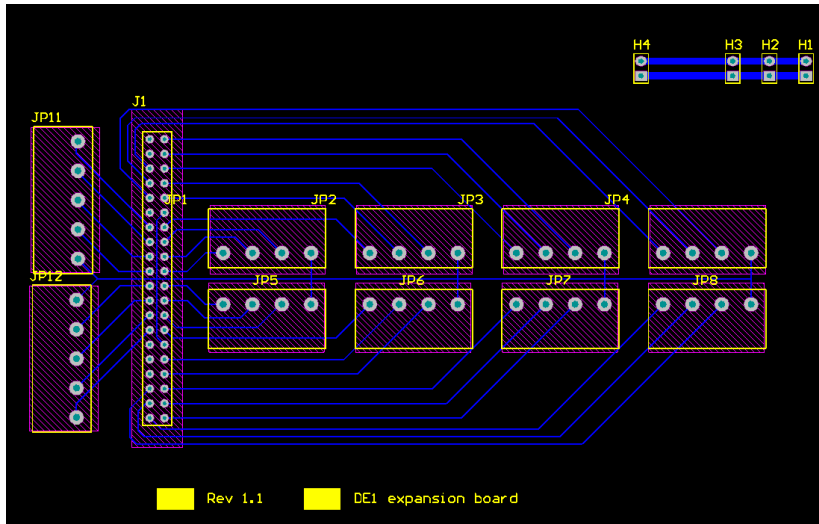


Figure 7-4 - PCB layout of 40-pin expansion board created using Altium Designer

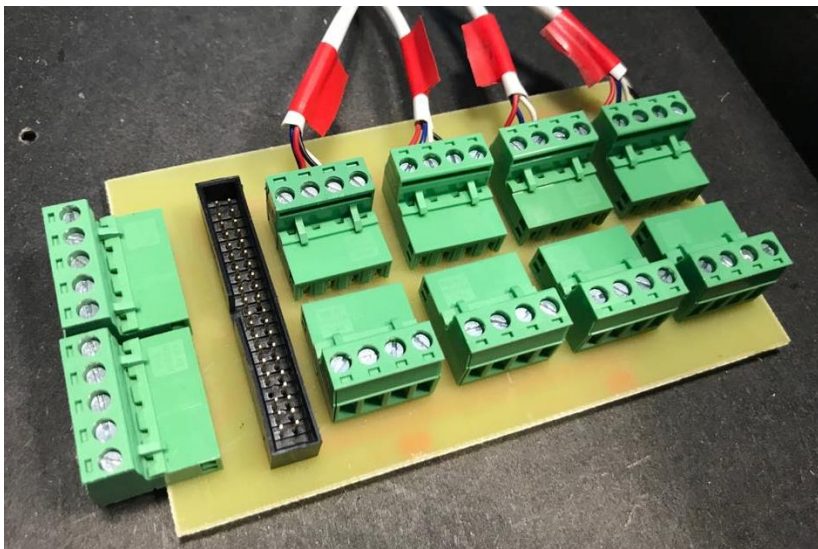


Figure 7-5 - PCB with pluggable screw terminals and a 40-pin header

This board was created for tidy, reliable, robust wiring between the FPGA and the stepper motor drivers. It also enables easy removal of the FPGA development board, the PCB, or any one of the stepper motor drivers without using any tools, and without damaging any of the components.

7.1.4. Limit and homing switches

In order to run the stepper motors in a semi-open loop, homing switches were required for the rotational drive assembly, and limit switches were required for the linear drive assembly. Using one switch for the rotational homing, and two switches for linear homing (one at each end of the module), the system could be precisely homed to avoid collisions and improve accuracy in calibration. A datum could be set, as well as providing the ability to measure the total workspace of each linear drive assembly, given that each assembly had a different work envelope due to the differing number of players per rod. The switches are shown in Figure 7-6.

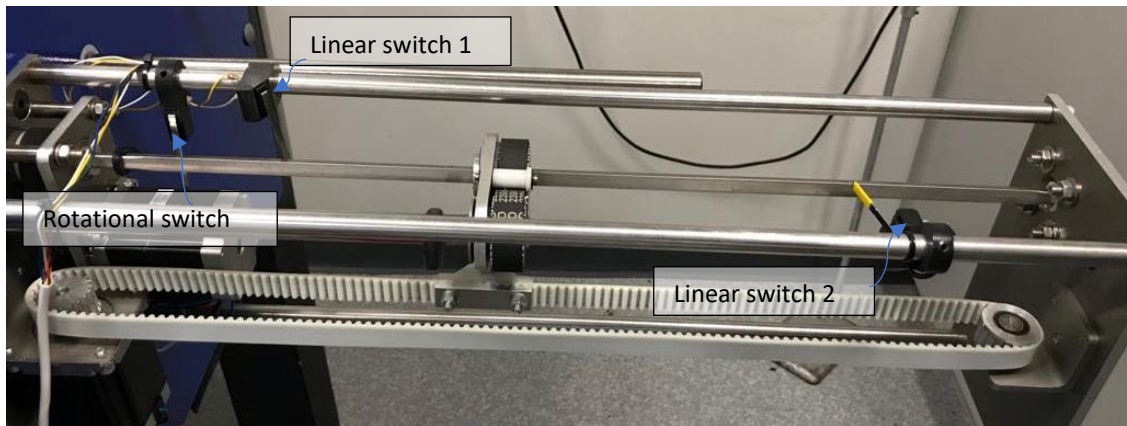


Figure 7-6 - One actuation module with each of the homing/limit switches labelled

7.1.5. Wiring management and cabling

All the wiring for the stepper motors and limit switches is run through the conduit shown on the left in Figure 7-7. Two of the conduits are for motor cabling and one is for limit switches. The conduits connect to the electronics enclosure with standard electrical screw glands, also shown in Figure 7-7 on the right.

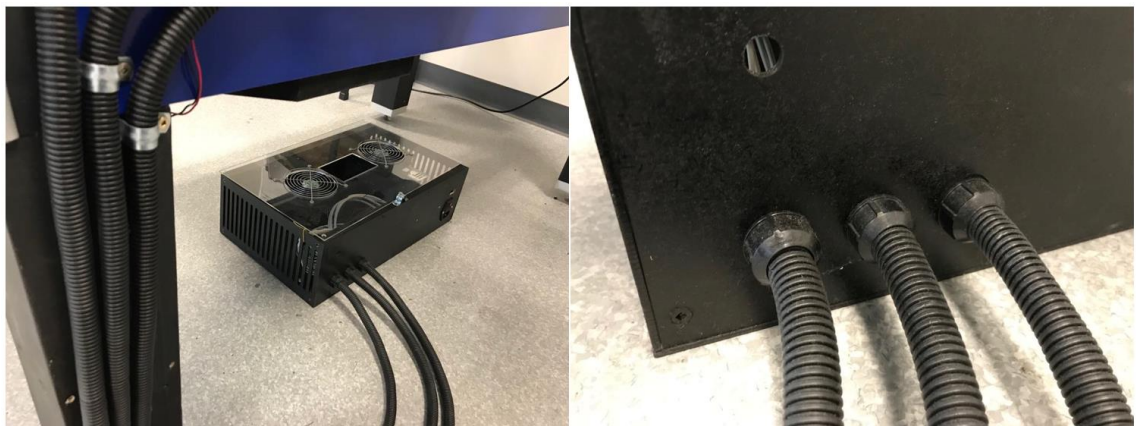


Figure 7-7 - Wiring conduits on left of image and electrical screw glands on right

7.1.6. Field illumination

The field illumination is provided by a series of LEDs of approximately 4,000 to 5,000 K colour temperature. The LEDs, shown in Figure 7-8, are run on 12V DC, however this can be increased or decreased depending on the illumination intensity required, given the background illumination conditions.

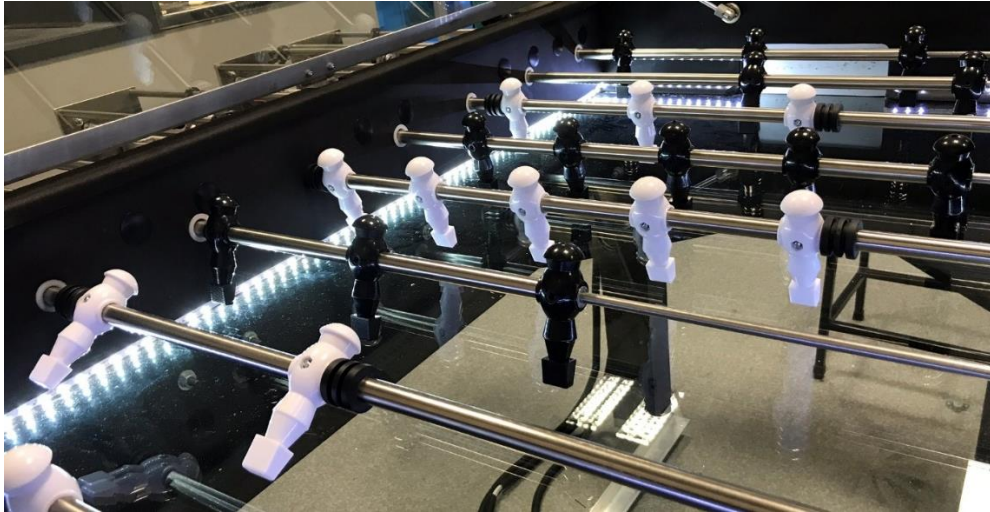


Figure 7-8 - Semi-automated foosball table with field illumination LEDs switched on

The trimpot screw of the variable resistor on the XL4015 board, shown in Figure 7-9, can be used to adjust the voltage and, therefore, the brightness of the LEDs. This also increases the power consumed so care should be taken not to exceed the power output capability of the XL4015 module.

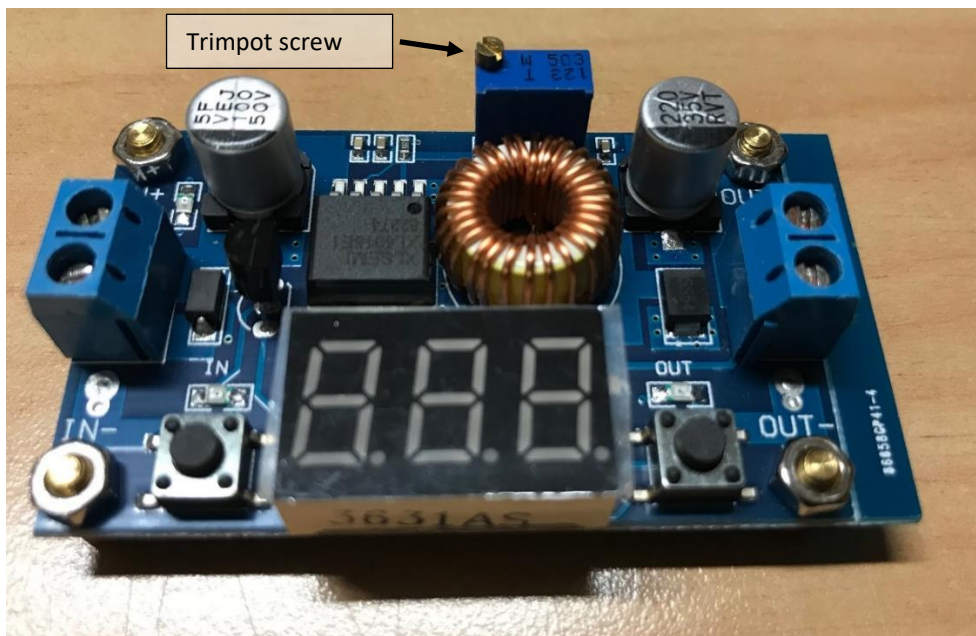


Figure 7-9 - XL4015 DC-DC voltage regulator module with adjustable output voltage

8. Testing and system performance

To quantitatively determine the performance of the high-speed vision-based control system, an experiment was required. The interception performance was identified as a key performance indicator that would be sensitive to detection and system latency. The thesis statement for this work was that by developing a system with sufficiently low latency, sufficiently high-performance stepper motor control, and enough spatial and temporal resolution, the capability for high-performance interception would be greatly improved.

In this chapter the system performance is tested relative to the interception requirement set out in chapter 3. This requirement is that for straight shots of sub-maximal velocity (significantly below 10 metres per second), each rod must successfully intercept 100% of shots. In these experiments a single rod is tested, and justification given for the selection of that rod.

One important aspect of the experiment described in this chapter is that “full coverage” was tested to ensure that the interception performance was spatially consistent. This means that regardless of the ball’s Y position at the time of interception, the module would be able to intercept the ball with the same efficacy.

8.1. Aims

- **Spatial invariance for interception** - the first aim was to establish the baseline interception performance of the semi-automated foosball table at all possible interception positions on the tested rod.
- **Close-up interception** - The second test was to test the close-up interception performance of the interception module relative to the performance requirements stated in chapter 3. Close-up means the distance between the release apparatus and the interception module is equal to the distance between the human team’s front most rod and the automated goal keeper rod. The requirements are for 100% interception performance for straight shots.
- **Close-up interception with added latency** - The third was to test the close-up performance of interception with varying quantities of artificial latency added to the system. This was to provide quantitative evidence that as the latency of the system increases, system performance (interception) decreases.

8.2. Spatial invariance for interception

8.2.1. Testing method

The method for carrying out each trial involves the following steps:

1. Disable the interception module
2. Set up the ball release apparatus by

- a) Aligning it with the corresponding position for the current trial number
 - b) Placing the ball in the relevant speed slot – the highest obviously resulting in the highest release velocity
3. Enable the interception module
 4. Send the calibration command to the interception module, causing it to perform the homing sequence detailed in section 6.2.7, to ensure that no positional errors had accumulated prior to each test
 5. Release the ball from the release apparatus at the top speed setting – 1 meter per second
 6. If the ball is successfully intercepted by the module record result as a pass, otherwise record result as a fail

Interception performance was tested 10 times for each of the 19 positions at the back of the table as shown in Figure 8-1 to ensure 100% interception capability in the best-case defence scenario – the scenario where each module would theoretically have as much time as possible to intercept. Additionally, this was done to qualitatively ensure correct functionality of the lens distortion and mapping functions shown in sections 5.6.5 and 6.2.8 respectively.

8.2.2. Overview of apparatus

Using the apparatus shown in Figure 8-1, the ball was released with the jig lined up with each of the black lines. The ball release jig is lined up on position 8 out of a total of 19 release positions.

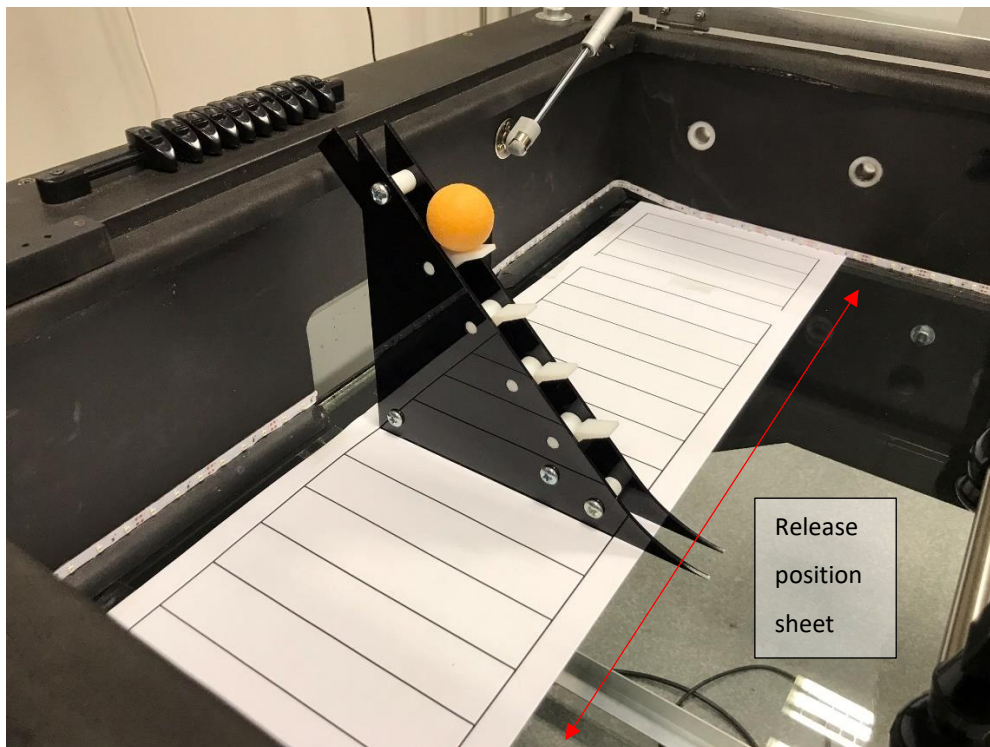


Figure 8-1 - Ball release apparatus positioned at the back of foosball table (closest to human goal)

The purpose of the release position sheet shown in Figure 8-1 is twofold; the primary purpose of the sheet is to consistently align the ball release apparatus for the experiments. This is to ensure repeatability of the test such that each trial is identical in terms of how far the module had to travel to intercept the ball, and that the ball is released at the same (zero) heading angle for each test. The second purpose is to block the ball from view of the camera until the ball is released. This will be explained further in subsequent sections.

Rod 2 was used for all interception experiments, as it represents the worst case in terms of how far the foosmen may have to travel to intercept the ball. Rod 2 has the largest workspace of 415.5mm because it has only 2 foosmen attached. Figure 8-2 shows the module used. The red lines indicate the approximate range of travel in either direction from the centre of the table.

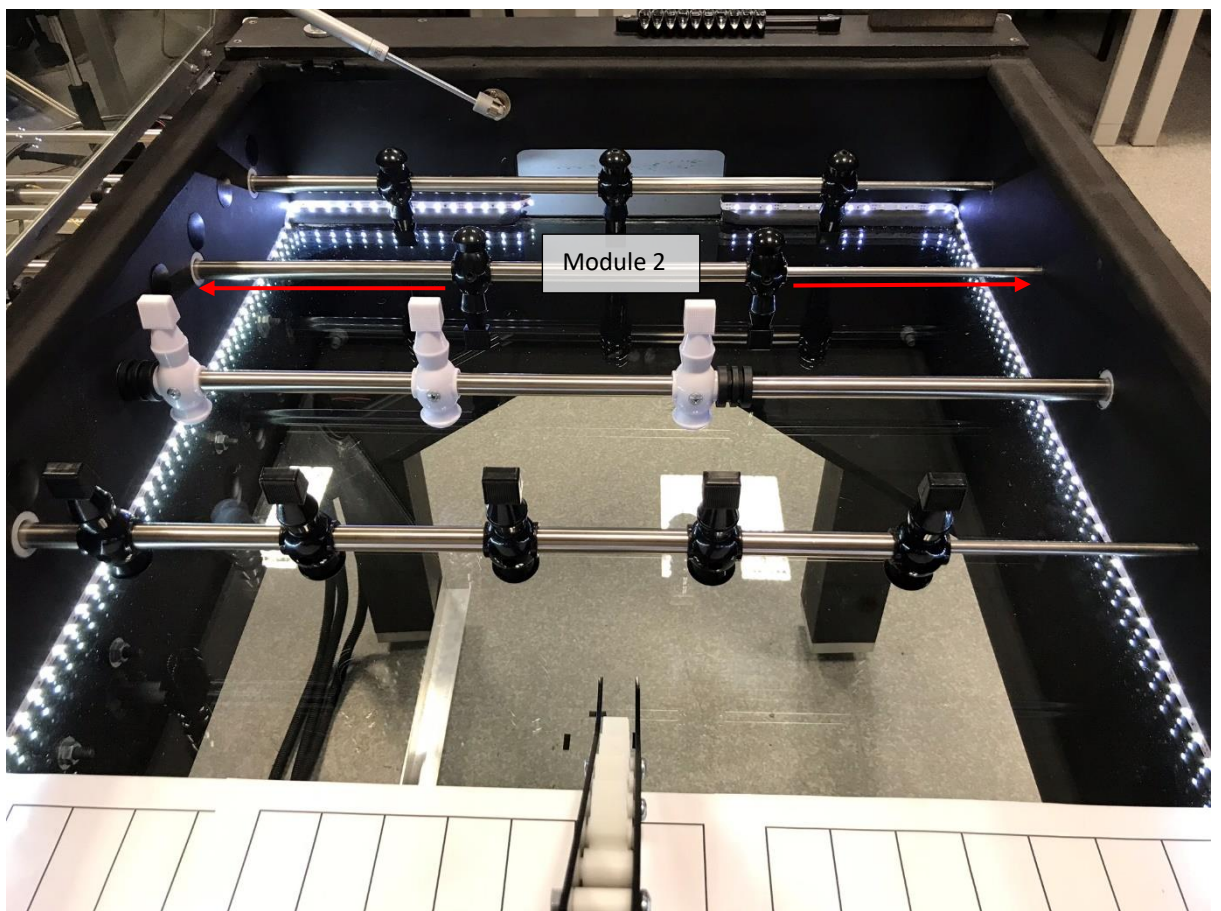


Figure 8-2 - Experimental setup using module 2 for interception

8.2.3. Data analysis

For statistical validity, the results were analysed with a 95% confidence interval. For attribute data (pass/fail) the number of samples, n , required to provide a given confidence interval (CI) is (Minitab, 2017)

$$n = \frac{\ln(1 - CI)}{\ln(\text{reliability})} \quad (14)$$

where *reliability* is the non-defect rate. 59 samples are required to be 95% confident that our results are 95% reliable:

$$n = \frac{\ln(0.05)}{\ln(0.95)} = 58.40397.$$

8.2.4. Results

The experimental results show that using a single module, 100% interception performance was achieved for straight sub-maximal shots. The raw experimental attribute data (pass or fail) is shown in Appendix H.

8.2.5. Discussion

Given the 190 samples obtained we can calculate the achieved reliability of results, by rearranging equation (14):

$$reliability = e^{\frac{\ln(CI)}{n}} = e^{\frac{\ln(0.05)}{190}} = 0.98436 \quad (15)$$

The reliability of these results is 98.4%.

Therefore, the results show, with 95% confidence and 98.4% reliability, that rod 2 can intercept sub-maximal straight shots at any interception point. This also shows that the lens distortion calibration and spatial mapping functions discussed in sections 5.6.5 and 6.2.8 were effective.

8.3. Close-up interception performance

Close-up interception performance was tested to verify that rod 2, with the furthest distance to travel, could intercept close-up shots at sub-maximal speeds. "Close-up" was defined as the distance between the front human rod, and the automated goalie rod. This position is shown in Figure 8-2.

The sub-maximal speed of 1 meter per second was selected for the following reasons:

1. Average shots during foosball gameplay are around 1 meter per second. This was measured using a 240 FPS slow motion camera.
2. Using the release position sheet to block the ball from view of the camera means that the system must respond to novel event data. In maximal speed (goal scoring) shots in actual gameplay, the ball will always be in sight prior to the shot being taken. This means that unless the ball is hit at a non-zero heading angle, the interception rod will already have placed the appropriate foosman in the required interception position. In order to test the responsiveness of the system, the ball was hidden from view of the camera and a slower release speed selected. Maximal speed shots at the close-up distance, combined with limited ball view time would likely result in zero successful intercepts.
3. Finally, to test the effect of additional latency, shots where the system is responding to novel event data are required. Note that the use of angle shots would not have provided an alternative as the latency can be compensated for with prediction (Behnke et al., 2004)

8.3.1. Testing method

The testing method was the same as the method described in section 8.2.1, however 59 samples were taken at a single position, to provide 95% confidence and 95% reliability.

The total ball view distance that the vision system had was 300 mm of ball travel. At 1 meter per second, this corresponds to a maximum of 18 visible frames before interception.

The position on the release sheet selected for these 59 samples was position 10, as this was half way between the two foosmen on rod 2. Additionally, the central position is the most important defence position because if the intercept is missed, the opposing team will most likely score a goal.

8.3.2. Results

The results for close-up interception performance show that 100% of the sub-maximal speed shots were intercepted by rod 2. The raw data is shown in Appendix I.

8.3.3. Discussion

Given the 59 samples taken, this shows with 95% confidence and 95% reliability, that rod 2 can intercept sub-maximal speed shots with limited view time of the ball.

At 1 metre per second the ball is visible for 300 ms. Therefore, maximum system response time is 300 ms, including actuation. Given the higher distance travelled on rod 2, it is likely that the response time of the remaining 3 modules is significantly lower, due to the shorter interception distances, and that actuation is a significant portion of the response time.

The interception performance shown above indicates that the vision system has sufficiently low latency to be used as the measurement system for interception in the semi-automated foosball table.

Additionally, the results show that the stepper motor control algorithm is effective.

Future work on the close-up interception performance could include increased ball release speeds up to and including maximal speeds, however this should only be done in the case that the ball is in view of the camera at all times, including before the ball is released. This would mimic real-life maximal or near-maximal speed shots the system would have to intercept. Performing maximal ball release, in conjunction with the ball being blocked from view would not accurately represent the interception performance required by the foosball table.

8.4. Close-up interception performance with added latency

The final part of the experiment involved artificially delaying the HPS access to the ball coordinates from the vision system.

8.4.1. Method to artificially add latency

In a vision system operating at 60 FPS, with a given amount of latency L , provided no frames are lost in processing (frame processing period does not exceed frame capture interval), the latency can be

expressed as a function of the incoming frames, where the system is responding to object coordinates that are delayed by a number of frames N :

$$L = N * \left(\frac{1}{60}\right) \quad (16)$$

The implementation code for artificially delaying the result is shown in Figure 8-3.

Procedure

```

volatile float coordsX [20]; // set up arrays to store X and Y values
volatile float coordsY [20];

if (coordsX[0] != 0) //as soon as values start arriving at start of array, start
using them
    X = coordsX[0]; //set X & Y vals to the 1st element of each array,
respectively
if (coordsX[0] != 0)
    Y = coordsY[0];

int i;
for (i = 0;i<19;) //shift end element back by 1 index value
{
    coordsX[i] = coordsX[i+1];
    coordsY[i] = coordsY[i+1];
    i++;
}

coordsX[19] = Xin; //set end element of arrays to input values of image data
coordsY[19] = Yin;

```

Figure 8-3 - Code snippet of algorithm used to artificially delay system by 20 frames

As is shown in Figure 8-3, two arrays are created as buffers for the ball X and Y coordinates. The length of these arrays is the number of frames, and therefore the time period, by which to delay the system. First, the X and Y values used by for interception calculations are set to the values of the first element of each array, provided the values are non-zero (they will only be non-zero once the image data arrives at the first array element). Then the values in the array are shifted left by one index value. Finally, the most recent image data is stored at the end of the buffers (index 19). This delays the system by 20 frames. The section labelled “Procedure” is called every time a new frame is received.

8.4.2. Testing method

The testing method was the same as the method described in section 8.2.1, however only 10 samples were taken for each added latency value. The ball release speed for these samples was kept at 1 meter per second, and the visible ball distance kept at 300 mm.

8.4.3. Results

As shown in Figure 8-4, as the added latency increased, the interception performance decreased. An increase in latency resulted in a lower proportion of successful intercepts.

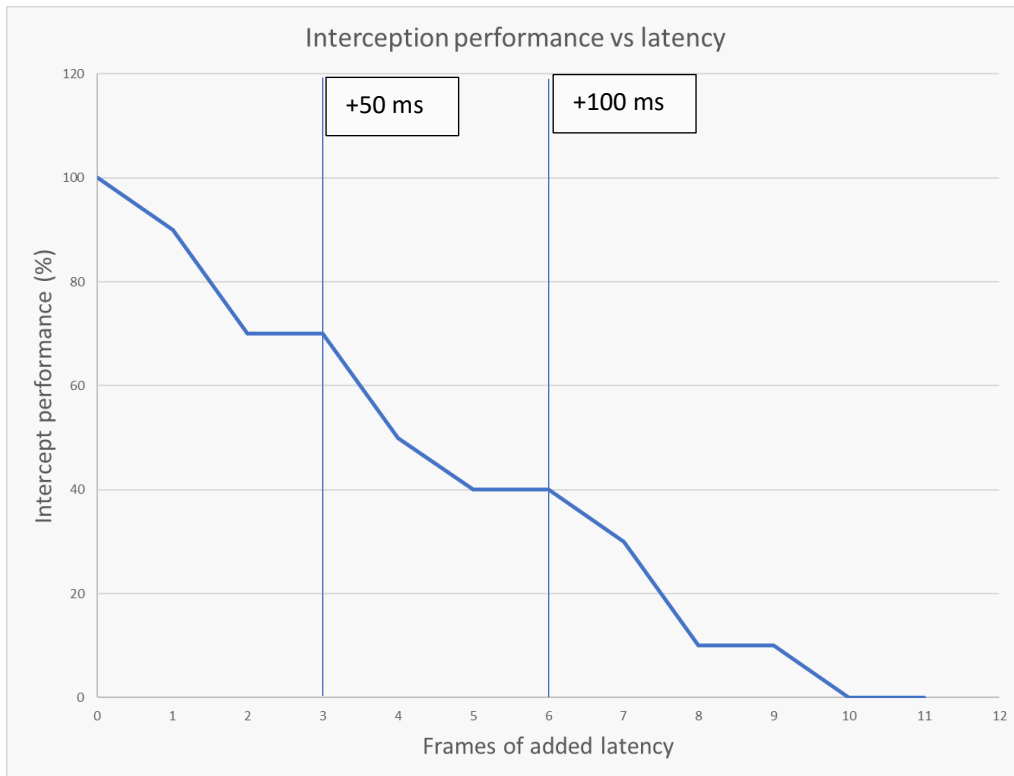


Figure 8-4 - Output graph of interception performance versus artificially added latency for straight goal shots

The raw data for the added latency trials is provided in Appendix J.

8.4.4. Discussion

For the results of the added latency, the reliability of the results is calculated below:

$$reliability = e^{\frac{\ln(0.05)}{10}} = 0.74113 \quad (17)$$

The lower sample size was used for the added latency tests to give a reasonably accurate, overall picture of the effects of added latency. For the purpose of providing a clear picture of the effects of adding latency to the system, approximately 75% reliability is sufficient.

The interception performance shown in Figure 8-4 indicates that at around 100 ms of additional latency, the interception performance drops to approximately 40% for straight goal shots. As little as 50 ms of additional latency causes the performance to drop appreciably.

The results clearly show that increased latency decreases interception performance. This decreased performance was due to the decreased responsiveness of the system, as it was operating on delayed object coordinates. Because the ball was hidden from view of the vision system, the modules did not have time to intercept the ball once the delayed ball coordinates had been processed, and interception commands given. This demonstrates the importance of low system latency.

8.5. Automated foosball

In a game of foosball, the majority of shots performed by amateur players are either passes from one of their foosmen to another on one rod, or straight kicks in the direction of the opposition goal. By maximising the ability of each module to intercept high speed straight shots, the defensive ability of the foosball table can be maximised.

For this experiment, the non-predictive model was used, as was described in section 6.2.8. By incorporating prediction, additional latency of at least 1 frame is added, which adds unnecessary latency when only straight shots are being tested with limited view time of the ball. Prediction was therefore excluded for the straight shot interception performance testing.

Once the foosball table has claimed possession of the ball, the strategic programming of the automated modules will determine the win/lose rate of the system.

8.6. Potential improvements or additions

In these experiments, angle shots were not tested. They are a part of the game of foosball, however for amateur players they happen far less frequently, and are usually unintentional. In future work it may be useful to test the system with angle shots directed toward the goal. This could provide some more useful information regarding the system performance and could give some further insight into the performance degradation experienced when latency is added to the system.

Only a single module was used for interception of straight shots with zero heading angle. It would be useful to establish real world performance tests with 4 modules performing interception and trials carried out in which a representative population of human players attempt to perform goal scoring shots, and the 4 modules attempt to block these. Work could be done to create intelligent defence strategies in which the maximum percentage of the goal is defended, depending on the ball's position on the table.

Finally, the interception performance of the whole system could be put to the test against semi-skilled or highly skilled players performing technical or high-speed shots.

8.7. Conclusions and recommendations

The results presented in this chapter show that the use of an FPGA SoC enabled interception of a moving target, using high-speed low-latency vision as the measurement.

A single module can defend straight shots 100% of the time, at all interception positions on the rod.

As the latency increases, the interception performance decreases.

Further work is required to complete the performance testing of the foosball table with all 4 modules defending against human players.

Further work involving defence against skilled players would be valuable as a final step in validation.

9. Final conclusions and recommendations

In this project, the objective was to research and develop a method to use computer vision for the high-speed control of robotics, where the image sensor functioned as the input to the tracking/interception control system. The testbed was a semi-automated foosball table in which one human team had been replaced by electromechanical modules which performed the sliding and rotational movements a human would normally perform to intercept and kick the ball.

The key metric for successfully controlling a robotic system at high-speed was that the robotic control system – the semi-automated foosball table – would be able to intercept shots equivalent to those that would normally be encountered in a vigorous game of amateur foosball. The quantitative requirement was that for straight shots, the automated system would be able to intercept 100% of straight-shots in which there was no heading angle. The qualitative requirement was that there would be no interception coordinate for which the module was unable to intercept the path of the ball.

The system developed was an FPGA SoC where image data was streamed from the sensor and image processing was performed on the streamed data. The foosball ball coordinates were then sent via high-bandwidth communication bridges to the embedded ARM cores, where the lens distortion and spatial calibration operations, and interception calculations were performed. The HPS then returns stepper motor commands back the FPGA which performs concurrent stepper motor control. Overall, the combination of all of these operations which were designed for low latency results in low system latency, and a high level of system responsiveness.

Scalable, parameterizable, variable acceleration stepper motor control modules were also implemented in FPGA hardware. The modules provided high levels of versatility, improved smoothness and improved performance compared to related work, with comparable resource requirements.

The effectiveness of the vision and actuator control was experimentally tested in section 8. The results of the experiments showed that at a 95% confidence interval, the foosball could intercept 100% of straight shots with a statistical reliability of 98.4%. Additionally, the experimental results showed that as artificial latency was added to the system, the performance dropped in an approximately linear fashion.

Future work could include testing real world performance of the foosball table which includes play versus amateur and trained players. Improved defence strategies could then be tested and compared to the work of other authors.

References

- Altera. (2016). Cyclone V Hard Processor System Technical Reference Manual. Retrieved from <https://www.intel.com/content/www/us/en/programmable/documentation/sfo1410143707420.html>
- Andersson, R. L. (1990, 5-7 Sep 1990). *A low-latency 60 Hz stereo vision system for real-time visual control*. Paper presented at the Proceedings. 5th IEEE International Symposium on Intelligent Control 1990, Philadelphia, PA, USA.
- Asano, S., Maruyama, T., & Yamaguchi, Y. (2009). *Performance Comparison of FPGA, GPU and CPU in Image Processing*. Paper presented at the Fpl: 2009 International Conference on Field Programmable Logic and Applications, Prague, Czech Republic.
- Asfour, T., Regenstein, K., Azad, P., Schroder, J., Bierbaum, A., Vahrenkamp, N., & Dillmann, R. (2006, 4-6 December 2006). *ARMAR-III: An Integrated Humanoid Platform for Sensory-Motor Control*. Paper presented at the 2006 6th IEEE-RAS International Conference on Humanoid Robots, Genova, Italy.
- Bailey, D. G. (2002). *A new approach to lens distortion correction*. Paper presented at the Proceedings Image and Vision Computing New Zealand, Auckland, New Zealand.
- Bailey, D. G. (2011). *Design for Embedded Image Processing on FPGAs*. Singapore: John Wiley and Sons (Asia) Pte Ltd.
- Bailey, D. G. (2018). *Image Processing in VHDL on FPGAs, in Massey University / IEEE NZ Central Section Three Day Workshop: Palmerston North, NZ*.
- Baklouti, M., & Abid, M. (2014). Multi-Softcore Architecture on FPGA. *International Journal of Reconfigurable Computing, 2014*, 1-13. doi:10.1155/2014/979327
- Barreto, J. P., Swaminathan, R., & Roquette, J. (2007). *Non Parametric Distortion Correction in Endoscopic Medical Images*. Paper presented at the 2007 3DTV Conference, Kos Island, Greece.
- Barry, A. J., Oleynikova, H., Honegger, D., Pollefeys, M., & Tedrake, R. (2015). *Fast onboard stereo vision for UAVs*. Paper presented at the Vision-based Control and Navigation of Small Lightweight UAV Workshop, International Conference On Intelligent Robots and Systems (IROS).
- Behnke, S., Egorova, A., Gloye, A., Rojas, R., & Simon, M. (2004). *Predicting Away Robot Control Latency*. Paper presented at the RoboCup 2003: Robot Soccer World Cup VII, Berlin, Heidelberg.
- Berner, R., Brandli, C., Yang, M., Liu, S. C., & Delbruck, T. (2013). *A 240 x 180 10mW 12us latency sparse-output vision sensor for mobile applications*. Paper presented at the 2013 Symposium on VLSI Circuits.
- Brookner, E. (1998). *Tracking and Kalman Filtering Made Easy*. New York: John Wiley & Sons, Inc.
- Censi, A., Strubel, J., Brandli, C., Delbruck, T., & Scaramuzza, D. (2013). *Low-latency localization by active LED markers tracking using a dynamic vision sensor*. Paper presented at the 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, Tokyo, Japan.
- Chen, M., Kato, K., & Adachi, K. (2002). The comparisons of sliding speed and normal load effect on friction coefficients of self-mated Si3N4 and SiC under water lubrication. *Tribology International, 35*(3), 129-135. doi:10.1016/S0301-679X(01)00105-0
- Chowdhury, M., Khalil, M. K., Nuruzzaman, D. M., & Rahaman, M. (2011). The effect of sliding speed and normal load on friction and wear property of aluminum. *International Journal of Mechanical and Mechanics Engineering, 11*, 53-57.
- Christopherson, H. B., Pickell, W. J., Koller, A. A., Kannan, S. K., & Johnson, E. N. (2004). *Small Adaptive Flight Control Systems for UAVs Using FPGA/DSP Technology*. Paper

- presented at the AIAA 3rd "Unmanned Unlimited" Technical Conference, Workshop and Exhibit.
- Cigliano, P., Lippiello, V., Ruggiero, F., & Siciliano, B. (2015). Robotic Ball Catching with an Eye-in-Hand Single-Camera System. *IEEE Transactions on Control Systems Technology*, 23(5), 1657-1671. doi:10.1109/TCST.2014.2380175
- Čížek, P., Faigl, J., & Masri, D. (2016). *Low-latency image processing for vision-based navigation systems*. Paper presented at the 2016 IEEE International Conference on Robotics and Automation (ICRA).
- Conradt, J., Berner, R., Cook, M., & Delbruck, T. (2009). *An embedded AER dynamic vision sensor for low-latency pole balancing*. Paper presented at the 2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops.
- de la Malla, C., & Lopez-Moliner, J. (2015). Predictive plus online visual information optimizes temporal precision in interception. *J Exp Psychol Hum Percept Perform*, 41(5), 1271-1280. doi:10.1037/xhp0000075
- Delbruck, T., & Lang, M. (2013). Robotic goalie with 3 ms reaction time at 4% CPU load using event-based dynamic vision sensor. *Frontiers in Neuroscience*, 7(223). doi:10.3389/fnins.2013.00223
- Dougherty, E. R., & Laplante, P. A. (1995). *Introduction to Real-Time Imaging*: Wiley-IEEE Press.
- Edwards, S. A. (2006). The Challenges of Synthesizing Hardware from C-Like Languages. *IEEE Design & Test of Computers*, 23(5), 375-386. doi:10.1109/MDT.2006.134
- El-Desouki, M., Jamal Deen, M., Fang, Q., Liu, L., Tse, F., & Armstrong, D. (2009). CMOS Image Sensors for High Speed Applications. *Sensors*, 9(1). doi:10.3390/s90100430
- Engelberg, S. (Ed.) (2015). *A Mathematical Introduction to Control Theory* (2nd ed. Vol. 4). London: Imperial College Press.
- Fowers, S. G., Lee, D. J., Tippetts, B. J., Lillywhite, K. D., Dennis, A. W., & Archibald, J. K. (2007). *Vision Aided Stabilization and the Development of a Quad-Rotor Micro UAV*. Paper presented at the 2007 International Symposium on Computational Intelligence in Robotics and Automation, Jacksonville, FL, USA.
- Franklin, G. F., Powell, J. D., & Emami-Naeini, A. (Eds.). (2015). *Feedback Control of Dynamic Systems* (7th ed.). New Jersey: Pearson.
- Frese, U., Bauml, B., Haidacher, S., Schreiber, G., Schaefer, I., Hahnle, M., & Hirzinger, G. (2001). *Off-the-shelf vision for a robotic ball catcher*. Paper presented at the Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems., Maui, HI, USA.
- Gamal, A. E., & Eltoukhy, H. (2005). CMOS image sensors. *IEEE Circuits and Devices Magazine*, 21(3), 6-20. doi:10.1109/MCD.2005.1438751
- García, J. G., Jara, A. C., Pomares, J., Alabdo, A., Poggi, M. L., & Torres, F. (2014). A Survey on FPGA-Based Sensor Systems: Towards Intelligent and Reconfigurable Low-Power Sensors for Computer Vision, Control and Signal Processing. *Sensors*, 14(4). doi:10.3390/s140406247
- Gönner, C., Rous, M., & Kraiss, K.-F. (2005). *Real-Time Adaptive Colour Segmentation for the RoboCup Middle Size League*. Paper presented at the RoboCup 2004: Robot Soccer World Cup VIII, Berlin, Heidelberg.
- Graham, P. S. (2001). *Logical Hardware Debuggers for FPGA-based Systems*. (Doctor of Philosophy PhD), Brigham Young University,
- Gribbon, K. T., Johnston, C. T., & Bailey, D. G. (2003). A real-time FPGA implementation of a barrel distortion correction algorithm with bilinear interpolation. *Image and Vision Computing NZ*, 408-413.
- Guo, Z., Najjar, W., Vahid, F., & Vissers, K. (2004). *A quantitative analysis of the speedup factors of FPGAs over processors*. Paper presented at the ACM/SIGDA 12th international symposium on Field programmable gate arrays, Monterey, CA, USA.

- Honegger, D., Oleynikova, H., & Pollefeys, M. (2014). *Real-time and low latency embedded computer vision hardware based on a combination of FPGA and mobile CPU*. Paper presented at the 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems.
- Hornberg, A., & Jahr, I. (2017). Lighting in Machine Vision. In A. Hornberg (Ed.), *Handbook of Machine and Computer Vision* (Vol. 2, pp. 63-178).
- Huimin, L., Zhiqiang, Z., Fei, L., & Xiangke, W. (2008). *A robust object recognition method for soccer robots*. Paper presented at the 2008 7th World Congress on Intelligent Control and Automation, Changsha, Hunan Province, China.
- Intel. (2017a). *Intel FPGA Technical Training - Designing with an ARM*-Based System on Chip*.
- Intel. (2017b). Introduction to the Qsys System Integration Tool. Retrieved from ftp://ftp.intel.com/Pub/fpgaup/pub/Intel_Material/17.0/Tutorials/Introduction_to_the_Qsys_Tool.pdf
- Intel. (2018). Video and Image Processing Suite Intel® FPGA IP. Retrieved from <https://www.intel.com/content/www/us/en/programmable/products/intellectual-property/ip/dsp/m-alt-vipsuite.html?wapkw=image+processing>
- Janssen, R., de Best, J., & van de Molengraft, R. (2010). *Real-Time Ball Tracking in a Semi-automated Foosball Table*. Paper presented at the RoboCup 2009: Robot Soccer World Cup XIII, Berlin, Heidelberg.
- Janssen, R., Verrijt, M., de Best, J., & van de Molengraft, R. (2012). Ball localization and tracking in a highly dynamic table soccer environment. *Mechatronics*, 22(4), 503-514. doi:10.1016/j.mechatronics.2012.02.009
- Jianping, F., Yau, D. K. Y., Elmagarmid, A. K., & Aref, W. G. (2001). Automatic image segmentation by integrating color-edge extraction and seeded region growing. *IEEE Transactions on Image Processing*, 10(10), 1454-1466. doi:10.1109/83.951532
- Jing, L., & Vadakkepat, P. (2010). Interacting MCMC particle filter for tracking maneuvering target. *Digital Signal Processing*, 20(2), 561-574. doi:10.1016/j.dsp.2009.08.011
- Jing, Z., & Sclaroff, S. (2003). *Segmenting foreground objects from a dynamic textured background via a robust Kalman filter*. Paper presented at the Proceedings Ninth IEEE International Conference on Computer Vision, Nice, France.
- Jobson, D. J., Rahman, Z., & Woodell, G. A. (1997). A multiscale retinex for bridging the gap between color images and the human observation of scenes. *IEEE Transactions on Image Processing*, 6(7), 965-976. doi:10.1109/83.597272
- Johnston, C. T., Gribbon, K. T., & Bailey, D. G. (2004). *Implementing image processing algorithms on FPGAs*. Paper presented at the Eleventh Electronics New Zealand Conference, Palmerston North, NZ.
- Klaiber, M. J., Bailey, D. G., Baroud, Y. O., & Simon, S. (2016). A Resource-Efficient Hardware Architecture for Connected Component Analysis. *IEEE Transactions on Circuits and Systems for Video Technology*, 26(7), 1334-1349. doi:10.1109/TCSVT.2015.2450371
- Kondo, T., Kikuchi, A., Kato, F., & Hirota, K. (1991). 5,093,716. United States Patent.
- Korkalo, O., & Honkamaa, P. (2010). *Construction and Evaluation of Multi-touch Screens Using Construction and Evaluation of Multi-touch Screens Using*. Paper presented at the International Conference on Interactive Tabletops and Surfaces, Saarbrücken, Germany.
- Kryjak, T., Komorkiewicz, M., & Gorgon, M. (2014). Real-time background generation and foreground object segmentation for high-definition colour video stream in FPGA device. *Journal of Real-Time Image Processing*, 9(1), 61-77. doi:10.1007/s11554-012-0290-5
- Leeser, M., Miller, S., & Haiqian, Y. (2004). *Smart camera based on reconfigurable hardware enables diverse real-time applications*. Paper presented at the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, Napa, CA, USA.

- Linares-Barranco, A., Gomez-Rodriguez, F., Jimenez-Fernandez, A., Delbruck, T., & Lichtensteiner, P. (2007). *Using FPGA for visuo-motor control with a silicon retina and a humanoid robot*. Paper presented at the 2007 IEEE International Symposium on Circuits and Systems, New Orleans, LA, USA.
- Lu, H., Zhang, H., Yang, S., & Zheng, Z. (Eds.). (2010). *A Novel Camera Parameters Auto-adjusting Method Based on Image Entropy*. Berlin, Heidelberg: Springer Berlin Heidelberg.
- Maxim, V., & Zidek, K. (2012). Design of High Performance Multimedia Control System for UAV/UGV Based on SoC/FPGA Core. *Procedia Engineering*, 48, 402-408. doi:10.1016/j.proeng.2012.09.532
- Mayer, G., Utz, H., & Kraetzschmar, G. (2002). *Towards autonomous vision self-calibration for soccer robots*. Paper presented at the IEEE/RSJ International Conference on Intelligent Robots and Systems, Lausanne, Switzerland.
- Mealy, B., & Tappero, F. (2016). *Free Range VHDL*. In *digital book released under the Creative Commons Attribution-ShareAlike Unported License*. Retrieved from <http://www.freerangefactory.org/>
- Meeus, W., Van Beeck, K., Goedemé, T., Meel, J., & Stroobandt, D. (2012). An overview of today's high-level synthesis tools. *Design Automation for Embedded Systems*, 16(3), 31-51. doi:10.1007/s10617-012-9096-8
- Minitab. (2017). How Many Samples Do You Need to Be Confident Your Product Is Good? Retrieved from <http://blog.minitab.com/blog/the-statistical-mentor/how-many-samples-do-you-need-to-be-confident-your-product-is-good>
- Monmasson, E., & Cirstea, M. N. (2007). FPGA Design Methodology for Industrial Control Systems - A Review. *IEEE Transactions on Industrial Electronics*, 54(4), 1824-1842. doi:10.1109/TIE.2007.898281
- Mueller, E., Censi, A., & Frazzoli, E. (2015). *Low-latency heading feedback control with neuromorphic vision sensors using efficient approximated incremental inference*. Paper presented at the 2015 54th IEEE Conference on Decision and Control (CDC), Osaka, Japan.
- Ngo, H. T., & Asari, V. K. (2005). A pipelined architecture for real-time correction of barrel distortion in wide-angle camera images. *IEEE Transactions on Circuits and Systems for Video Technology*, 15(3), 436-444. doi:10.1109/TCSVT.2004.842609
- Ni, Z., Bolopion, A., Agnus, J., Benosman, R., & Regnier, S. (2012). Asynchronous Event-Based Visual Shape Tracking for Stable Haptic Feedback in Microrobotics. *IEEE Transactions on Robotics*, 28(5), 1081-1089. doi:10.1109/TRO.2012.2198930
- Nister, D., Stewenius, H., & Grossmann, E. (2005, 17-21 Oct. 2005). *Non-parametric self-calibration*. Paper presented at the Tenth IEEE International Conference on Computer Vision Beijing, China.
- Otsu, N. (1979). A Threshold Selection Method from Gray-Level Histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1), 62-66. doi:10.1109/TSMC.1979.4310076
- Padon, O. (2003). *Development Of Robotic Foosball As A Versatile Platform For Robotics Research and Contests*.
- Pistohl, T., Ball, T., Schulze-Bonhage, A., Aertsen, A., & Mehring, C. (2008). Prediction of arm movement trajectories from ECoG-recordings in humans. *Journal of Neuroscience Methods*, 167(1), 105-114. doi:10.1016/j.jneumeth.2007.10.001
- Pololu. (2018). Stepper Motor: Bipolar, 200 Steps/Rev, 57x76mm, 3.2V, 2.8 A/Phase. Retrieved from <https://www.pololu.com/product/1478/resources>
- Pratt, G. A., Willisson, P., Bolton, C., & Hofman, A. (2004, June 30 2004-July 2 2004). *Late motor processing in low-impedance robots: impedance control of series-elastic actuators*. Paper presented at the Proceedings of the 2004 American Control Conference, Boston, MA, USA.

- Prevost, C. G., Desbiens, A., & Gagnon, E. (2007). *Extended Kalman Filter for State Estimation and Trajectory Prediction of a Moving Object Detected by an Unmanned Aerial Vehicle*. Paper presented at the 2007 American Control Conference, New York, NY, USA.
- Proctor, F., & P. Shackelford, W. (2001). *Real-time Operating System Timing Jitter and its Impact on Motor Control*. Paper presented at the Intelligent Systems and Advanced Manufacturing, Boston, MA, USA.
- Putnam, A., Caulfield, A. M., Chung, E. S., Chiou, D., Constantinides, K., Demme, J., . . . Burger, D. (2014). A reconfigurable fabric for accelerating large-scale datacenter services. *SIGARCH Comput. Archit. News*, 42(3), 13-24. doi:10.1145/2678373.2665678
- Ramakoti, N., Vinay, A., & Jatoth, R. K. (2009). *Particle Swarm Optimization Aided Kalman Filter for Object Tracking*. Paper presented at the 2009 International Conference on Advances in Computing, Control, and Telecommunication Technologies, Trivandrum, Kerala, India.
- Rosales, R., & Sclaroff, S. (1998). *Improved Tracking of Multiple Humans with Trajectory Prediction and Occlusion Modeling*. Paper presented at the IEEE Workshop on Interpretation of Visual Motion, Santa Barbara, CA, USA.
- Rowe, A., Rosenberg, C., & Nourbakhsh, I. (2002). *A low cost embedded color vision system*. Paper presented at the IEEE/RSJ International Conference on Intelligent Robots and Systems, Lausanne, Switzerland.
- Saber, E., Murat Tekalp, A., & Bozdagi, G. (1997). Fusion of color and edge information for improved segmentation and edge linking. *Image and Vision Computing*, 15(10), 769-780. doi:10.1016/S0262-8856(97)00019-X
- Safaei, A., Wu, Q. M. J., & Yang, Y. (2018). System-on-a-chip (SoC)-based hardware acceleration for foreground and background identification. *Journal of the Franklin Institute*, 355(4), 1888-1912. doi:10.1016/j.jfranklin.2017.07.037
- Singer, R. A. (1970). Estimating Optimal Tracking Filter Performance for Manned Maneuvering Targets. *IEEE Transactions on Aerospace and Electronic Systems*, AES-6(4), 473-483. doi:10.1109/TAES.1970.310128
- Tanaka, H., Ohnishi, K., Nishi, H., Kawai, T., Morikawa, Y., Ozawa, S., & Furukawa, T. (2009). Implementation of Bilateral Control System Based on Acceleration Control Using FPGA for Multi-DOF Haptic Endoscopic Surgery Robot. *IEEE Transactions on Industrial Electronics*, 56(3), 618-627. doi:10.1109/TIE.2008.2005710
- Thomas, D. B., Howes, L., & Luk, W. (2009). *A comparison of CPUs, GPUs, FPGAs, and massively parallel processor arrays for random number generation*. Paper presented at the Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays, Monterey, California, USA.
- Tianjian, L., & Fujimoto, Y. (2006). *A control system with high speed and real time communication links*. Paper presented at the 9th IEEE International Workshop on Advanced Motion Control, Istanbul, Turkey.
- Tsutsui, H., Nakamura, H., Hashimoto, R., Okuhata, H., & Onoye, T. (2010). *An FPGA implementation of real-time retinex video image enhancement*. Paper presented at the 2010 World Automation Congress, Kobe, Japan.
- Wang, B., Liu, Q., Zhou, L., Zhang, Y., Li, X., & Zhang, J. (2011). *Velocity profile algorithm realization on FPGA for stepper motor controller*. Paper presented at the 2nd International Conference on Artificial Intelligence, Management Science and Electronic Commerce (AIMSEC), Dengleng, China.
- Wei, Z., Byung Hwa, K., Larson, A. C., & Voyles, R. M. (2005). *FPGA implementation of closed-loop control system for small-scale robot*. Paper presented at the ICAR '05. Proceedings., 12th International Conference on Advanced Robotics, 2005., Seattle, WA, USA.

- Weigel, T. (2005). *KiRo - A Table Soccer Robot Ready for the Market*. Paper presented at the Proceedings of the 2005 IEEE International Conference on Robotics and Automation, Barcelona, Spain,.
- Willson, R. G., & Shafer, S. A. (1994). What is the center of the image? *Journal of the Optical Society of America A*, 11(11), 2946-2955. doi:10.1364/JOSAA.11.002946
- Wolpert, D. M., & Flanagan, J. R. (2001). Motor prediction. *Current Biology*, 11, R729-R732.
- Xia, G., & Ludwig, S. A. (2016). *Object-tracking based on particle filter using particle swarm optimization with density estimation*. Paper presented at the 2016 IEEE Congress on Evolutionary Computation (CEC), Vancouver, BC, Canada.
- Zelnik-Manor, L., & Irani, M. (1996). Some Aspects of Zoom Lens Camera Calibration. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 18(10), 1105-1116. doi:10.1109/34.544080
- Zhao, Y., Paine, N., Kim, K. S., & Sentis, L. (2015). Stability and Performance Limits of Latency-Prone Distributed Feedback Controllers. *IEEE Transactions on Industrial Electronics*, 62(11), 7151-7162. doi:10.1109/TIE.2015.2448513

Published work

Lues, J., Gupta, G. S., & Bailey, D. (2017). *Evaluation of High-Speed Image Processing for Low Latency Control of a Mechatronic System*. Paper presented at the International Conference on Robot Intelligence Technology and Applications, Daejeon, South Korea.

Appendices

Appendix A : Matlab computer vision prototype code

This section of code has been taken from a MATLAB Graphical User Interface (GUI) where the vision code ran continuously, capturing images and displaying the results of the filtered images. It has been shortened, with many of the unnecessary lines of code removed for brevity.

```
%initialise video object vid
vid = imaq.VideoDevice('winvideo', 2, 'MJPG_640x480');
% initialise winvideo 2 - number 1 is built in webcam
vid.ReturnedColorSpace = 'rgb'; % set colourspace of output vid to RGB

%create global array to store ball coordinates
global centroidArray
centroidArray = zeros(5,2);
global angleArray
angleArray = zeros(3,1);

%serial device setup
global Duino
Duino = serial('COM6','BaudRate',250000);
fopen(Duino);

%snap one frame of vid object
pic = step(vid);
%filter image using built-in matlab function from their colour thresholding app
filteredImage = createMaskBall(pic);
%create structuring elements for morphological filters - erosion and dilation
SE = strel('square',10);
SE2 = strel('square',20);
%morphologically erode the image with structuring element SE
eroded = imerode(filteredImage, SE);
%morphologically dilate the image with structuring element SE2
dilated = imdilate(eroded, SE2);
%acquire the ball coordinates
ballProps = regionprops(dilated, 'Centroid');
%catch all statement as error thrown if ballCentroids accessed when no object found
try
    ballCentroids = ballProps.Centroid;
    %save x and y coordinates of ball to elements x1 and y1 of array
    centroidArray(1,1) = ballCentroids(1);
    centroidArray(1,2) = ballCentroids(2);
    %circularly shift the elements backwards so latest centroid is at 5th
    %element of array - largest element = latest, smallest element = oldest
    centroidArray = circshift(centroidArray,-1);
    display(centroidArray)
catch
    fprintf('error - no ball found yet')
end
```

The next section of code, also taken from the GUI and edited for brevity, plots a line between the 5 most recent detected object coordinates. This was simply to show a “trail” of where the ball had been in recent frames, and to estimate whether the ball was travelling straight or not.

```
%draw last 5 centroids on the axes
scatter([0 640 centroidArray(1:5)], [0 480 centroidArray(6:10)])

%plot lines between each element of centroid array in order of their capture
line(centroidArray(1:2), centroidArray(6:7))
line(centroidArray(2:3), centroidArray(7:8))
line(centroidArray(3:4), centroidArray(8:9))
line(centroidArray(4:5), centroidArray(9:10))
line(centroidArray(5:1), centroidArray(10:6))

%calculate current trajectory in degrees
currentX = centroidArray(5,1);
currentY = centroidArray(5,2);
oldX = centroidArray(4,1);
oldY = centroidArray(4,2);
angle = atan2(currentY - oldY, currentX - oldX)*(180/pi);
display(angle)
angleArray(1,1) = angle;
angleArray = circshift(angleArray, -1);
display(angleArray);
%tell us whether the ball is going straight
deltaAngle = angleArray(3)-angleArray(2);
if deltaAngle < 5
    fprintf('ball is going straight')
else
    fprintf('ball has curved')
end
```

Appendix B : C code for calibrating image space for radial distortion in the HPS

```
X = Xin + pad;
Y = Yin + pad;
r = sqrt((centreX - X)*(centreX - X) + (centreY - Y)*(centreY - Y));
theta = atan((centreY - Y)/(centreX - X));
r = r/R;
s = r*(1 + (K1*r) + (K2*(r*r)) + (K3*(r*r*r*r)));
s2 = s*R;
X1 = s2*cos(theta);
Y1 = s2*sin(theta);
if (X < centreX)
{
    X2 = centreX - X1;
    Y2 = centreY - Y1;
}
if (X >= centreX)
{
    X2 = centreX + X1;
    Y2 = centreY + Y1;
}
X2 = X2 + 50;
Y2 = Y2 - 240;
Y2 = Y2*4.107;
```

Appendix C : Qsys connections view of entire system

Connections	Name	Description	Export	Clock	Base	End	IRQ		
	hps_0	Arria V/Cyclone V Hard Proces...	hps_0_f2h_cold_reset_req hps_0_f2h_debug_reset_req hps_0_f2h_warm_reset_req hps_0_f2h_stm_hw_events memory hps_io h2f_reset h2f_axi_clock h2f_axi_master f2h_axi_clock f2h_axi_slave h2f_lw_axi_clock h2f_lw_axi_master f2h_irq0 f2h_irq1	hps_0_f2h_cold_reset_req hps_0_f2h_debug_reset_req hps_0_f2h_warm_reset_req hps_0_f2h_stm_hw_events memory hps_0_hps_io hps_0_h2f_reset <i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i>	clk_0 [h2f_axi_cl...] clk_0 [f2h_axi_cl...] clk_0 [h2f_lw_axi...	0x0000_0000	0xFFFF_FFFF	IRQ 0 IRQ 0 IRQ 31 IRQ 31	
	hps_only_master	JTAG to Avalon Master Bridge	clk clk_reset master master_reset	<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i>	clk_0 [clk]				
	sysid_qsys	System ID Peripheral Intel FPG...	clk reset control_slave	<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i>	clk_0 [clk] [clk]	0x0001_0000	0x0001_0007		
	jtag_uart	JTAG UART Intel FPGA IP	clk reset avalon_jtag_slave irq	<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i>	clk_0 [clk] [clk] [clk]	0x0002_0000	0x0002_0007		
	fpga_only_master	JTAG to Avalon Master Bridge	clk clk_reset master master_reset	<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i>	clk_0 [clk]				
	clk_0	Clock Source	clk_in clk_in_reset clk clk_reset	clk reset <i>Double-click to export</i> <i>Double-click to export</i>	exported clk_0				
	hps_warm_reset_pio	PIO (Parallel I/O) Intel FPGA IP	clk reset s1 external_connection	<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i>	clk_0 [clk] [clk]	0x0001_0020	0x0001_002f		
		hps_cold_reset_pio	PIO (Parallel I/O) Intel FPGA IP	clk reset s1 external_connection	<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i>	clk_0 [clk] [clk]	0x0001_0010	0x0001_001f	
		hps_debug_reset_pio	PIO (Parallel I/O) Intel FPGA IP	clk reset s1 external_connection	<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i>	clk_0 [clk] [clk]	0x0001_0030	0x0001_003f	
		onchip_memory2_0	On-Chip Memory (RAM or RO...	clk1 s1 reset1	<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i>	clk_0 [clk1] [clk1]	0x0000_0000	0x0000_ffff	
		intr_capturer_0	Interrupt Capture Module	clock reset_sink avalon_slave_0 interrupt_receiver	<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i>	clk_0 [clock] [clock] [clock]	0x0003_0000	0x0003_0007	IRQ 0 IRQ 31
		x_min_pio	PIO (Parallel I/O) Intel FPGA IP	clk reset s1 external_connection	<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i>	clk_0 [clk] [clk]	0x0001_0070	0x0001_007f	
		x_max_pio	PIO (Parallel I/O) Intel FPGA IP	clk reset s1 external_connection	<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i>	clk_0 [clk] [clk]	0x0001_0080	0x0001_008f	
		y_min_pio	PIO (Parallel I/O) Intel FPGA IP	clk reset s1 external_connection	<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i>	clk_0 [clk] [clk]	0x0001_0090	0x0001_009f	
		y_max_pio	PIO (Parallel I/O) Intel FPGA IP	clk reset s1 external_connection	<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i>	clk_0 [clk] [clk]	0x0001_00a0	0x0001_00af	
		button_pio_0	PIO (Parallel I/O) Intel FPGA IP	clk reset s1 external_connection irq	<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i>	clk_0 [clk] [clk] [clk]	0x0001_0060	0x0001_006f	
		step_cmd_0	PIO (Parallel I/O) Intel FPGA IP	clk reset s1 external_connection	<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i>	clk_0 [clk] [clk]	0x0001_00b0	0x0001_00bf	

	step_data_0 clk reset s1 external_connection	PIO (Parallel I/O) Intel FPGA IP Clock Input Reset Input Avalon Memory Mapped Slave Conduit	<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i> step_data_0_external_conn...	clk_0 [clk] [clk]	0x0001_00c0 0x0001_00cf	
	img_data_ready clk reset s1 external_connection irq	PIO (Parallel I/O) Intel FPGA IP Clock Input Reset Input Avalon Memory Mapped Slave Conduit Interrupt Sender	<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i> img_data_ready_external_c...	clk_0 [clk] [clk]	0x0001_0040 0x0001_004f	
	step_pos_0 clk reset s1 external_connection	PIO (Parallel I/O) Intel FPGA IP Clock Input Reset Input Avalon Memory Mapped Slave Conduit	<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i> step_pos_0_external_conne...	clk_0 [clk] [clk]	0x0001_0050 0x0001_005f	
	RS232_HPS_CMD clk reset s1 external_connection	PIO (Parallel I/O) Intel FPGA IP Clock Input Reset Input Avalon Memory Mapped Slave Conduit	<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i> rs232_hps_cmd_external_c...	clk_0 [clk] [clk]	0x0001_00d0 0x0001_00df	
	RS232_HPS_CMD_rea... clk reset s1 external_connection irq	PIO (Parallel I/O) Intel FPGA IP Clock Input Reset Input Avalon Memory Mapped Slave Conduit Interrupt Sender	<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i> rs232_hps_cmd_ready_exte...	clk_0 [clk] [clk]	0x0001_00e0 0x0001_00ef	
	step_dir_all clk reset s1 external_connection	PIO (Parallel I/O) Intel FPGA IP Clock Input Reset Input Avalon Memory Mapped Slave Conduit	<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i> step_dir_all_external_conne...	clk_0 [clk] [clk]	0x0001_00f0 0x0001_00ff	
	RS232_HPS_CMD_RET... clk reset s1 external_connection	PIO (Parallel I/O) Intel FPGA IP Clock Input Reset Input Avalon Memory Mapped Slave Conduit	<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i> rs232_hps_cmd_return_ext...	clk_0 [clk] [clk]	0x0001_0100 0x0001_010f	
	HPS_ACTIVE clk reset s1 external_connection	PIO (Parallel I/O) Intel FPGA IP Clock Input Reset Input Avalon Memory Mapped Slave Conduit	<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i> hps_active_external_conne...	clk_0 [clk] [clk]	0x0001_0110 0x0001_011f	
	stepper_0_busy clk reset s1 external_connection	PIO (Parallel I/O) Intel FPGA IP Clock Input Reset Input Avalon Memory Mapped Slave Conduit	<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i> stepper_0_busy_external_c...	clk_0 [clk] [clk]	0x0001_0120 0x0001_012f	
	frequency_from_HPS clk reset s1 external_connection	PIO (Parallel I/O) Intel FPGA IP Clock Input Reset Input Avalon Memory Mapped Slave Conduit	<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i> frequency_from_hps_exter...	clk_0 [clk] [clk]	0x0001_0130 0x0001_013f	

Appendix D : Interrupt service routine for HPS receiving image data from FPGA

```
volatile int * xMin_ptr = (int * ) (ALT_FPGA_F2H_BASE + F2H_X_MIN_PIO_OFFSET);
volatile int * xMax_ptr = (int * ) (ALT_FPGA_F2H_BASE + F2H_X_MAX_PIO_OFFSET);
volatile int * yMin_ptr = (int * ) (ALT_FPGA_F2H_BASE + F2H_Y_MIN_PIO_OFFSET);
volatile int * yMax_ptr = (int * ) (ALT_FPGA_F2H_BASE + F2H_Y_MAX_PIO_OFFSET);

void img_data_ready_ISR( void )
{
    volatile int * IMG_READY_ptr =(int *) (ALT_LWFPGA_BASE+ALT_LWFPGA_IMG_READY_OFFSET);

    int xMinVal, xMaxVal, yMinVal, yMaxVal;
    xMinVal = *(xMin_ptr); // read from image data PIO signals
    xMaxVal = *(xMax_ptr);
    yMinVal = *(yMin_ptr);
    yMaxVal = *(yMax_ptr);

    Xin = (float)(xMinVal + xMaxVal)/2;
    // calculate centre of gravity in x and y directions
    Yin = (float)(yMinVal + yMaxVal)/2;

    press = *(IMG_READY_ptr + 3);
    // read the interrupt register
    *(IMG_READY_ptr + 3) = press;
    // Clear the interrupt
}
}
```

Appendix E : VHDL entity declaration for stepper motor module

```
entity stepperModules is
  generic( MODULE_NUM : integer;
           STEP_DIR : std_logic );
  port( --INPUTS
        clock_in : in std_logic;
        distanceIn : in integer;
        freqIn : in integer;
        strobe : in std_logic;
        homeSwitch : in std_logic;
        homeSwitch2 : in std_logic;
        resetKey_in : in std_logic;
        direction_in : in std_logic;
        STOP : in std_logic;

        --OUTPUTS
        stBusy : out std_logic;
        -- flag used to let HPS know the module is busy
        STATUS : out integer := 0;
        direction_out : out std_logic;
        pulse_out : out std_logic;
        enable_out : out std_logic;
        position_out : out unsigned
        );

end entity stepperModules;
```

Appendix F : VHDL code for stepper motor module (architecture)

```

architecture implementation of stepperModules is

begin

process(clock_in)
variable clkDivide : integer := 0;--Variable used to divide the clock speed
variable stepFreq : integer := 60000;--clkDivide increments to the value of stepFreq
variable toggleFlag : std_logic; --this flag is used to assign the value of the output pin
variable accelerator : integer := 20;-- used to change the frequency - change motor speed
variable distance : integer := 50;--2x the number of steps the motor is to take (steps = distance/2)
variable distCntr : unsigned(15 downto 0) := (others => '0');--used to check position vs total dist
variable startFlag : std_logic := '1';
variable distBy2 : integer := 0;
variable distSub400 : integer := 0;
variable modCntr : integer := 0;
variable modCmpr : integer := 10;
variable hitLimSwitch : integer := 0;

begin
if rising_edge(clock_in) then
if ((distCntr <= distance) and (startFlag = '1')) then
clkDivide := clkDivide + 1;--increment clock divider counter
if (clkDivide > stepFreq) then--if clock divider counter reached, take a step. Then
--reset counter to 0
pulse_out <= toggleFlag; --output value of toggleflag to stepper pulse pin
distCntr := distCntr + 1; --one step taken so increment distance counter
toggleFlag := not toggleFlag;--toggles the flag for stepper output pin
clkDivide := 0; --resets clock divider counter to 0
position_out <= distCntr;

if distance < 800 then
if (distCntr < (distBy2)) and ((distCntr < (distance-(distBy2)))) then
stepFreq := stepFreq - accelerator;
modCntr := modCntr + 1;
if modCntr = modCmpr then --Every nth (n = modCmpr) toggle of
--the pulse pin, the accelerator
--increases by 1- exponential ramp
accelerator := accelerator + 1;
modCntr := 0;
end if;
end if;
if (distCntr > (distance - (distBy2))) then
stepFreq := stepFreq + accelerator;
modCntr := modCntr + 1;
if modCntr = modCmpr then
accelerator := accelerator - 1;
modCntr := 0;
end if;
end if;
end if;

if distance >= 800 then
if (distCntr < 400) and ((distCntr < (distSub400))) then
stepFreq := stepFreq - accelerator;
modCntr := modCntr + 1;
if modCntr = modCmpr then
accelerator := accelerator + 1;
modCntr := 0;
end if;
end if;
if (distCntr > (distSub400)) then
stepFreq := stepFreq + accelerator;
modCntr := modCntr + 1;
if modCntr = modCmpr then
accelerator := accelerator - 1;
modCntr := 0;
end if;
end if;
end if;

end if;
end if;
if (distCntr >= distance) then --this stops the stepping after enough steps are taken
startFlag := '0';
enable_out <= '1';
pulse_out <= '0';
accelerator := 20;
stBusy <= '0';
position_out <= distCntr;
STATUS <= 0;
end if;
end process;
end architecture;

```

Appendix G : VHDL code for homing switch functionality

```
if homeSwitch2 = '1' and STATUS = 1 then -- home switch pressed first time
    STATUS <= 2;
    stBusy <= '0';
end if;
if homeSwitch2 = '0' and STATUS = 2 then -- home switch released after being
    -- pressed first time
    STATUS <= 3;
end if;

if homeSwitch = '1' and STATUS = 3 then -- second home switch pressed (will be
    -- opposite side of foosball table)
    STATUS <= 4;
    stBusy <= '0';
end if;

if homeSwitch = '0' and STATUS = 4 then -- second home switch un-pressed
    STATUS <= 5;
end if;

if STATUS = 5 and (distCntr = distance-1) then -- module backed off after
    -- pressing second home switch
    STATUS <= 6;
end if;

if STOP = '1' then -- stops all motion and outputs current distance to HPS signal
    startFlag := '0';
    enable_out <= '1';
    pulse_out <= '0';
    accelerator := 20;
    stBusy <= '0';
    position_out <= distCntr;
    STATUS <= 0;
end if;
```

Appendix H : Raw data (pass/fail) for preliminary interception test

	top speed setting - no added latency - number is positions tested																			
trial no.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	
1	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P
2	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P
3	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P
4	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P
5	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P
6	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P
7	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P
8	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P
9	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P
10	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P

pass fail

Appendix I : Raw data (pass/fail) for goal interception test – 59 samples total

Close-up performance baseline						
trial no.	1	2	3	4	5	6
1	P	P	P	P	P	P
2	P	P	P	P	P	P
3	P	P	P	P	P	P
4	P	P	P	P	P	P
5	P	P	P	P	P	P
6	P	P	P	P	P	P
7	P	P	P	P	P	P
8	P	P	P	P	P	P
9	P	P	P	P	P	P
10	P	P	P	P	P	

pass fail

Appendix J : Raw data for added latency interception experiment

trial no.	middle straight goal shot - added latency in frames												
	1	2	3	4	5	6	7	8	9	10	11	12	13
1	P	P	P	F	F	F	P	F	F	F	F	F	F
2	P	F	P	P	F	P	F	F	F	F	F	F	F
3	F	F	F	F	F	F	F	F	F	F	F	F	F
4	P	P	F	P	P	P	F	F	F	F	F	F	F
5	P	P	F	P	F	P	F	F	F	F	F	F	F
6	P	P	P	F	F	F	F	F	F	F	F	F	F
7	P	F	P	F	P	F	P	P	F	F	F	F	F
8	P	P	P	F	P	P	F	F	F	F	F	F	F
9	P	P	P	P	P	F	F	F	P	F	F	F	F
10	P	P	P	P	F	P	P	F	F	F	F	F	F

pass fail