Lakehead University
Department of Electrical Engineering

Object Detection and Tracking

ENGI-4969-YB: Degree Project (Electrical)

Date of Submission: April 6, 2018

Project Advisor: Dr. Yushi Zhou

Submitted by:

Ronald Chan, 0644916
Responsible for sections: 4.1-4.3, 5.1-5.2, 6.1, 6.6, 6.8

Luke Ferguson, 0586501
Responsible for sections: 5.3, 6.2-6.5, 6.8

Neal Palmer, 0646984
Responsible for sections: 4.4, 5.4, 6.7

Abstract

In this project, computer vision is used to control servo motors. The embedded system is based on a System-on-Chip (SoC). The chip contains both Field Programmable Gate Array (FPGA) fabric and a Hard Processor System (HPS). A camera will capture a real-time image, the image is processed by the HPS. The processed data is transferred to the FPGA fabric that controls the servo motors. The servo motors will move the camera to keep a user defined object within the center of the camera's view. Object detection requires the use of OpenCV libraries that run on the HPS. The servo motors will employ Pulse Width Modulation (PWM) to achieve control. The design and implementation of this system was a success. However, due to the low frame rate of the camera and HPS limitations, the system performance is lacking. Thus, future work would include the use of a camera with a better frame rate. Additionally, the use of OpenCL and/or designing the entire system within the FPGA fabric could result in increased system performance. Finally, further image processing development is required if this design is to be implemented in a real-world application.

Table of Contents

Abstract ii
List of Figures v
List of Tables vii
1. Introduction 8
2. Background 8
3. Design Specifications 9
3.1 Objectives 9
3.2 Components 10
4. Theory 12
4.1 Thresholding 12
4.2 Kalman Filter 14
4.3 Distance Measurement 18
4.4 Servo Control 19
5. Design 21
5.1 Object Detection 21
5.2 Kalman Filter 22
5.3 HPS Memory and Data Flow Design 24
5.4 Servo Control 25
6. Implementation 27
6.1 Tools used 27
6.2 AXI Implementation 27
6.3 Memory Implementation 30
6.4 HDMI Implementation 32
6.5 Servo PIO Implementation 36
6.6 Object Tracking 38
6.6.1 Overview of C++ Implementation 38
6.6.2 Thresholding 39
6.6.3 Object Detection 40
6.6.4 Kalman Filter 43
6.6.5 Distance Measurement 45
6.7 Servo Control 47
6.8 Putting It All Together 55
6.8.1 Create the Testbench 55
6.8.2 Servo Motor Setup 56
6.8.3 Running the Tracking Script 60
7. Project Management 61
8. Summary 62
8.1 Costs and marketability 62
8.2 Social Impact 62
8.3 Project Limitations 63
8.4 Lessons Learned 63
Appendix 64

Appendix A: Mechanical Design 64
Appendix B: The Top-Level Verilog Code 70
Appendix C: PWM 78
Appendix D: PWM 82
Appendix E: Counter used by the PWM controllers 85
Appendix F: The HPS physical addresses header 86
Appendix G: The Object Tracking C++ Code 92
Appendix H: Code from Terasic's Control Panel Reference Design 106

References 115

Resources Used 116

Index 117

List of Figures

Figure 1: System overview 9
Figure 2: Block diagram of the system. (courtesy of Terasic Inc.) 11
Figure 3:Major components of the DE-10 Nano. (courtesy of Terasic Inc.) 11
Figure 4: (a) Original image. (b) Resultant binary image after thresholding. 12
Figure 5: (a) Two-dimensional Gaussian function; (b) Gaussian Kernel 13
Figure 6: Gaussian blurring 13
Figure 7: Effects of (b) erosion and (c) dilation on an image. 14
Figure 8: Tracking an object with Kalman filter. (The Mathworks Inc., 2018) 15
Figure 9: Overview of distance measurement. 19
Figure 10: Servo Motor Pulse Width Modulation 19
Figure 11: Servo Internal Control Circuit 20
Figure 12: PLL 21
Figure 13: The contours found from the binary image shown in Figure 3b. 22
Figure 14: Block diagram of memory mapping. 24
Figure 15: Altera PLL IP block 26
Figure 16: The configuration of the AXI bridges. 28
Figure 17: Example connection between Pipeline Bridge and dip switches. 29
Figure 18: Connection from Pipeline Bridge to HPS AXI master port. 29
Figure 19: Physical memory access code (a). 30
Figure 20: Physical memory access code (b). 30
Figure 21: Physical memory access code (c). 31
Figure 22: Physical access code (d). 31
Figure 23: AXI bridges with regards to FPGA fabric, courtesy of Terasic. 32
Figure 24: Video data flow. 33
Figure 25: Altera PLL MegaCore. 33
Figure 26: Frame reader configuration. 34

Figure 27: Clocked Video Output configuration (a). 35
Figure 28: Clocked Video Output configuration (b). 35
Figure 29: Code to test the connection between Processor and FPGA peripherals. 36
Figure 30: Details of GPIO pin layout from Terasic's User Manual. 37
Figure 31: Servo PIO configuration. 38
Figure 32: Overview of object detection and tracking C++ script. 39
Figure 33: OpenCV headers used in project. 39
Figure 34: Thresholding implementation in C++ 40
Figure 35: Overview of object detection code 41
Figure 36: Finding the object based on its captured shape. 41
Figure 37: Finding the object's position. 42
Figure 38: Overview of Kalman filter implementation. 43
Figure 39: Kalman filter setup (1 of 2). 44
Figure 40: Kalman filter setup (2 of 2). 45
Figure 41: Overview of the initialize function implemented in the C++ script. 46
Figure 42: Video displayed with distance measurement enabled. 46
Figure 43: X and Y coordinates 48
Figure 44: Range of motion in X direction 50
Figure 45: Range of motion in Y direction 50
Figure 46: Logic level shifter circuit 51
Figure 47: Model-Sim Capture of a 0.5ms Pulse 52
Figure 48: 0.5ms Pulse Measured at GPIO 52
Figure 49: 0.5ms Pulse Measured after logic shifter 52
Figure 50: Model-Sim Capture of 1ms Pulse 53
Figure 51: 1ms Pulse Measured at GPIO 53
Figure 52: 1ms Pulse Measured After Logic Level Shifter 53
Figure 53: Model-Sim Capture of a 2ms Pulse 54
Figure 54: 2ms Pulse Measured at GPIO 54

Figure 55: 2ms Pulse Measured After Logic Level Shifter 54
Figure 56: Testbench setup 56
Figure 57: Servo test code........ 57
Figure 58: Servo calibration C++ code. 59
Figure 59: Project Gantt chart........ 61

List of Tables

Table 1: Physical memory addresses. 30
Table 2: Average distance measured 47
Table 3: Average distance measurement error........ 47
Table 4: Object location and corresponding motor direction 48
Table 5: PWM codeword refresh rate 49
Table 6: Range of motion 50
Table 7: Test results 55
Table 8: Servo motor limits 58
Table 9: Project costs 62

1.Introduction

The rapid advancement of technology has brought about many advantages. In particular, the ability to include an entire system on a chip (SoC). An example of a SoC includes both a field programmable gate array (FPGA) and a hard processor system (HPS). Our team proposes to use such a system for the purpose of image processing. Previously, image processing included the use of Digital Signal Processors (DSPs). However, DSPs execute algorithms in sequential order. Thus, a large quantity of sequential algorithms such as those used in image processing, will increase the time required for computation. Alternatively, an FPGA can decrease processing time because it uses clock cycles to perform computations concurrently. Multiple computations can execute simultaneously during overlapping time periods. Therefore, a processor working in conjunction with an FPGA would greatly improve real time image processing. In order to achieve the desired image processing, Open Source Computer Vision libraries (OpenCV) are used. Furthermore, a platform to demonstrate the image processing capabilities is required. This platform will be the object detection and tracking system. Essentially, a camera will capture a real-time image that will be processed by the ARM processors. The processed data will be sent to the FPGA fabric where a set of servo motors are controlled using Pulse Width Modulation (PWM). The servo motors will rotate to keep the object within the center view of the camera.

2. Background

Image processing is used in various industries. In the case of object detection and tracking, one major industrial sector comes to mind – the automotive industry. A big motivation to use object detection and tracking within the automotive industry is in autonomous vehicles. These vehicles utilize an array of cameras and a myriad of other sensors to safely control the vehicle in a variety of typical driving conditions. At its core, the technology involves computer vision techniques and machine learning to identify everything around the vehicle and observe traffic patterns to safely navigate public roads. Road safety is a very important public issue. Driver error, pedestrian error, and impairment were identified as the main causes of 95% of all traffic accidents [1]. Autonomous vehicles can greatly improve road safety and eliminate driver error and impaired driving. In terms of social perspective, autonomous vehicles will cause a decrease of public costs associated with accident prevention and management, as well as healthcare and other social costs linked to vehicular accidents. In addition, optimized route selection and communication links between vehicles may reduce greenhouse gas emissions and mitigate the burden for infrastructure investment such as additional lanes [2]. Although the technology is still in its infancy, every major car and tech company has begun to develop their own autonomous vehicle technology. The combined research and development spending in autonomous and connectivity technologies is expected to reach $230 billion between 2015 and 2025 [3]. Thus, computer vision will become a lucrative market and there will be a demand for this type of product. This project will examine techniques used in object detection and tracking to gain a further understanding of how such a system operates. On the completion of this project, team members are better equipped to take advantage of this lucrative industrial sector.

3.Design Specifications

3.1 Objectives

The focus of this project is to detect and track a user defined object. The system will capture a real-time video with a USB webcam. The image is processed in an environment containing OpenCV libraries to identify the object. Servo motors will rotate the camera mount to keep the camera centered on the object. The motors will be connected to general purpose input/output pins of the FPGA. Finally, it is important to note that the communication between the HPS and FPGA occurs through several Advanced Extensible Interface (AXI) bridges. For a graphical representation see the figure below.

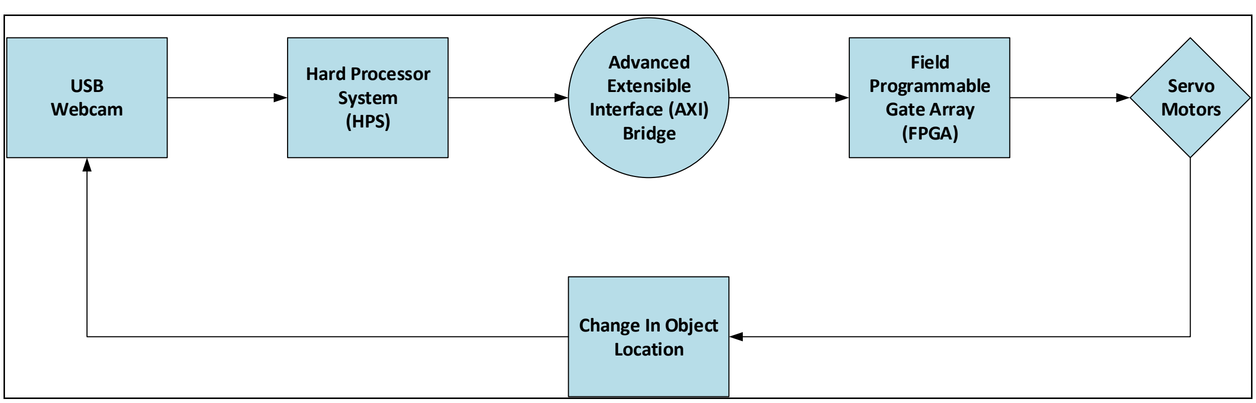


Figure 1: System overview

3.2 Components

The following is a list of the design components used in this design:

- DE10-Nano
  - The DE10-Nano is an ARM-based SoC and contains an FPGA in the same package. On the advice of the project advisor to incorporate more hardware design, an FPGA was chosen over microcontrollers like the Arduino. The dual nature of the Cyclone V provides the greatest amount of flexibility and performance at a reasonable cost.
- Camera
  - The camera used in this project is the Logitech C310 USB webcam. Initially, a much cheaper generic USB webcam was used. However, the generic webcam had a flimsy adjustable zoom lens providing too much zoom to be effective in tracking objects in close proximity. Another advantage of the Logitech C310 is that its housing design allowed for easy and stable mounting to the servo motors. The camera is capable of recording at 1280x720 pixels at 15 frames-per-second.
- Two servo motors
- 3D printed camera mount components

The DE10-Nano board contains a Cyclone V SE SoC. This chip contains a Hard Processor System (HPS) of which the processor is the dual-core ARM Cortex-A9. Figure 2 contains a block diagram describing the connections of the various peripherals. Figure 3 is a photograph of the board.

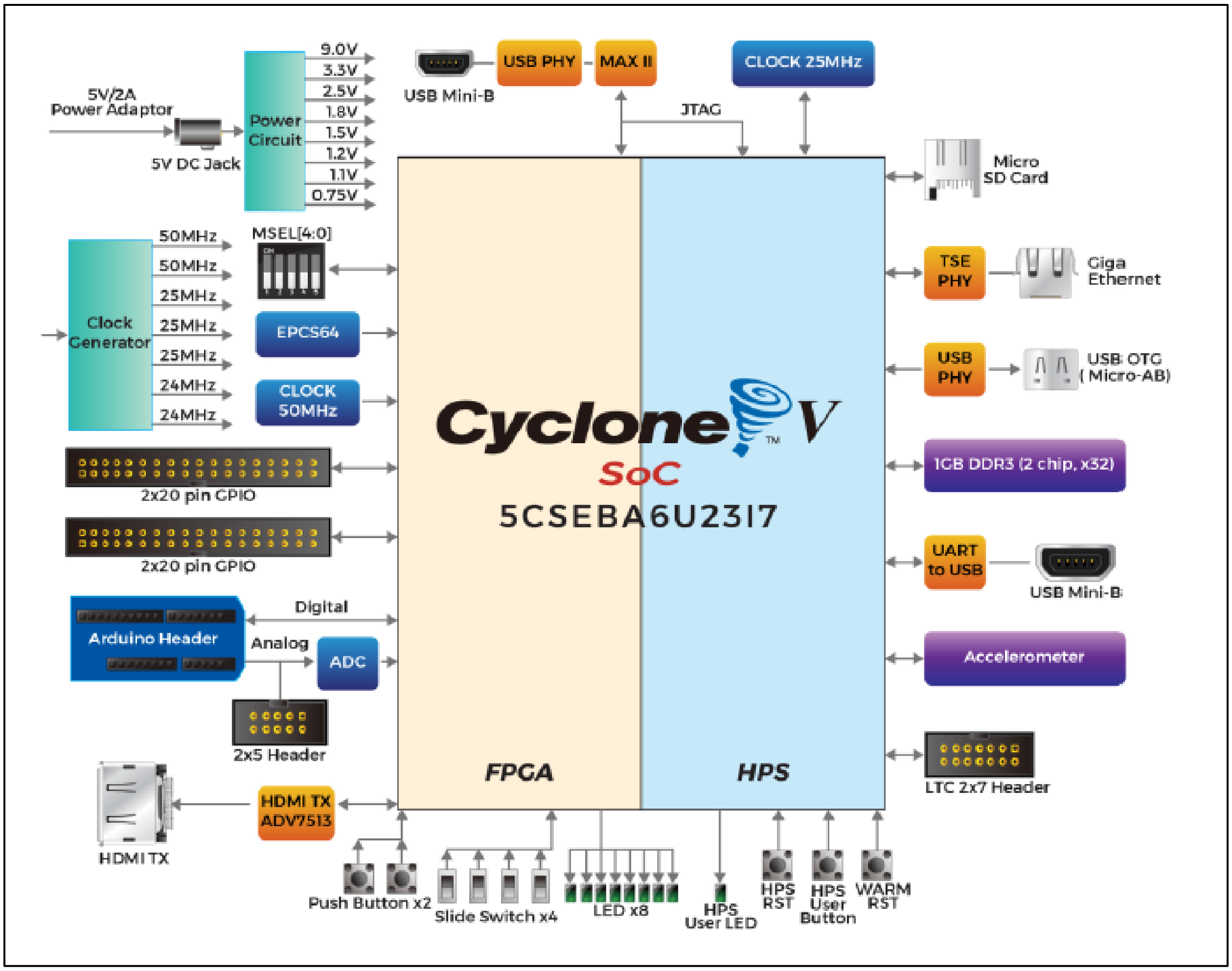


Figure 2: Block diagram of the system. (courtesy of Terasic Inc.)

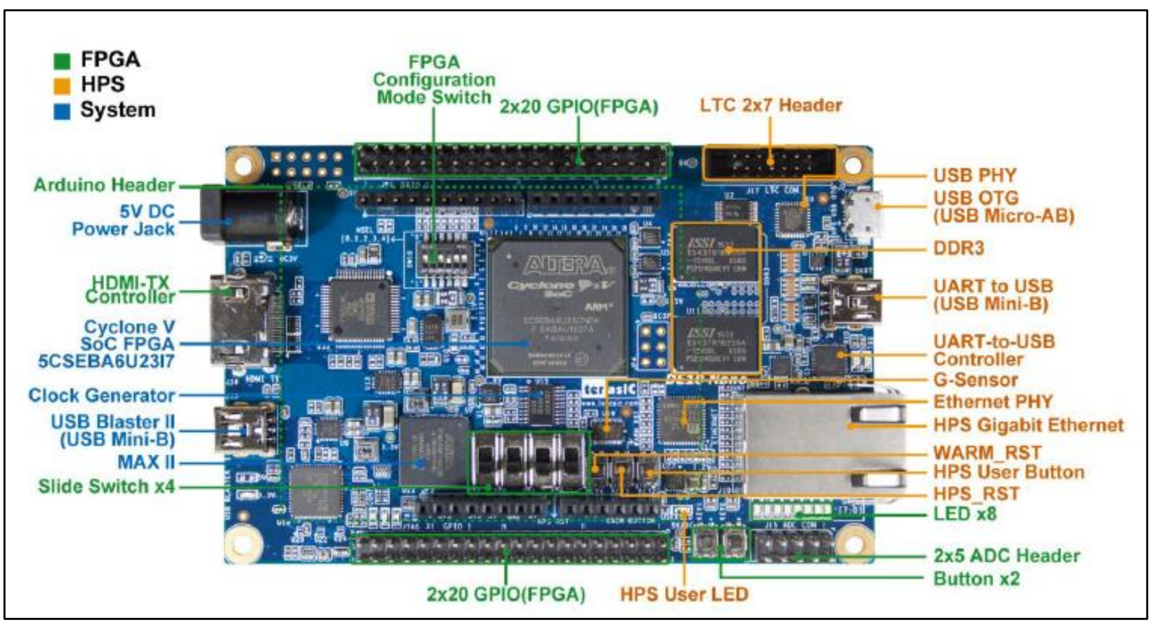


Figure 3:Major components of the DE-10 Nano. (courtesy of Terasic Inc.)

4.Theory

4.1 Thresholding

Thresholding is a filtering process done to an image to isolate desired features within the image. A thresholded image, shown in Figure 5, is a black and white image where only the desired features are prominent, with everything else in the frame masked out. Thresholding is typically accompanied by several image operations, with the goal of obtaining a mask that can best extract the desired features while suppressing everything else in the frame as cleanly as possible. The resulting binary image increases the accuracy of the object detection. Figure 5a highlights the importance of good thresholding in object detection applications, as poor thresholding can lead to the misidentification of objects. The process typically consists of the following: remove noise from the image, mask off unwanted features, and improve the quality of the edges in the binary image.

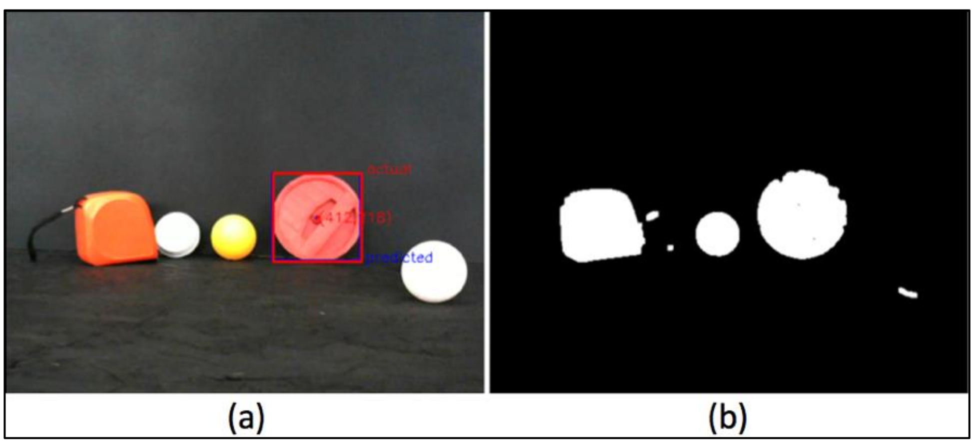
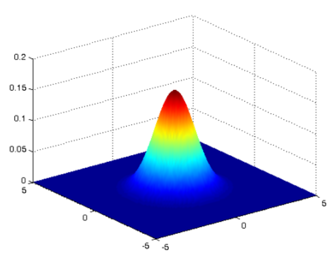


Figure 4: (a) Original image. (b) Resultant binary image after thresholding.
Note how the red object is incorrectly identified as the object of interest, which should have been the orange ball in the center of the image. This is the shortcoming of using such a simple object detection scheme.

First, a two-dimensional Gaussian function, represented as a matrix called a kernel, is convolved with each pixel to obtain a weighted average of that pixel in relation to its neighbouring pixels, which smoothens the image by reducing noise. The two-dimensional Gaussian function is the product of two one-dimensional Gaussian functions:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{1}$$

Graphically, the function is cone shaped, as shown in the three-dimensional plot in Figure 6a. The kernel is a square matrix of size *2n+1* elements and is typically kept small to retain speed. A 5x5 kernel with a standard deviation σ = 1 is shown in Figure 6b. Note that the origin is in the central element in the matrix. The smoothness of the resulting image is determined by the value of σ, as shown in Figure 7a. This two-dimensional convolution step is known as Gaussian blurring, and it improves the quality and accuracy of the extracted features in later steps, as demonstrated in Figure 7b.



(a)

$$\frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

(b)

Figure 5: (a) Two-dimensional Gaussian function; (b) Gaussian Kernel

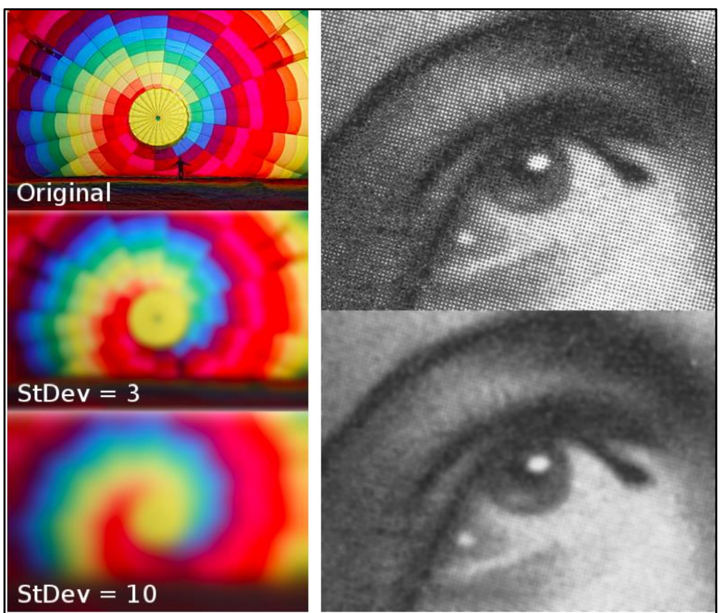


Figure 6: Gaussian blurring
(a) The effect of Gaussian blurring on an image (IkamusumeFan, "Cappacdocia Gaussian Blur", 2015). (b) Gaussian blurring improving the smoothness of a halftone image (Alex:D, "Halftone, Gaussian Blur", 2008). A halftone image uses dots of varying sizes and spacing to create a gradient effect.

The next step is the actual task of thresholding an image. There are many thresholding methods of varying complexities, but the result is similar – a binary image similar to the one produced in Figure 5b. The final thresholding step implemented on the system are two morphological operations. Morphological transformations are simple operations based on shapes in the image [4]. The two most common operations used are erosion and dilation. The erosion process erodes away the edges of features in an image, shrinking the foreground region, as shown in Figure 8b. A kernel is swept through the binary image and the resulting image consists of only the regions where all pixels under the kernel have a value of one; all other pixels are made to zero. The pixels along boundaries are discarded, removing white noise and detaching connected objects [5]. Dilation is the opposite of erosion, where a pixel is given a value of one if at least one pixel under the kernel has a value of one. This results in increasing the foreground region, as shown in Figure 8c. Erosion and dilation is typically done in that order, since dilation restores the shrunken regions cause by erosion, without the noise present in the original binary image. Broken parts of an object are also joined as a result of the process.

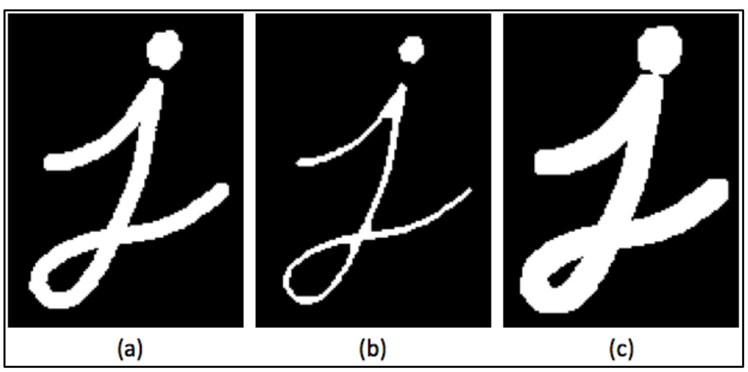


Figure 7: Effects of (b) erosion and (c) dilation on an image.

4.2 Kalman Filter

The Kalman filter is a linear, stochastic, recursive estimator, and is one of the most popular data fusion algorithms in the field of information processing [6]. Famously used on the Apollo navigation computer, Kalman filters are used today in every satellite navigation device, smart phone and can be applied to fields such as economics and robotics. The filter is based on a linear system model with a state equation and an output (measurement) equation. It uses a series of measurements observed over time, containing statistical noise and other inaccuracies, and produces estimates of unknown variables that tend to be more accurate than those based on a single measurement alone, by using a joint probability distribution over the variables for each timeframe. The filter consists of two steps: a prediction step which estimates the current state variables and their uncertainties, and the weighted average step, which gives a greater bias to estimates with higher certainties. The weights are calculated from covariance, a measure of the estimated uncertainty of the system's state prediction. The result of the weighted average is the new state estimate that lies between predicted and measured state and has a better estimated uncertainty than the individual parts alone. The gain of the filter is the relative weight given to the measurement and current state estimates. A high gain will place more weight on the most recent measurement, resulting in a more responsive, but less smooth operation. A low gain, however, better follows a model's predictions, and reduces the effects of noise at the cost of lower responsiveness.

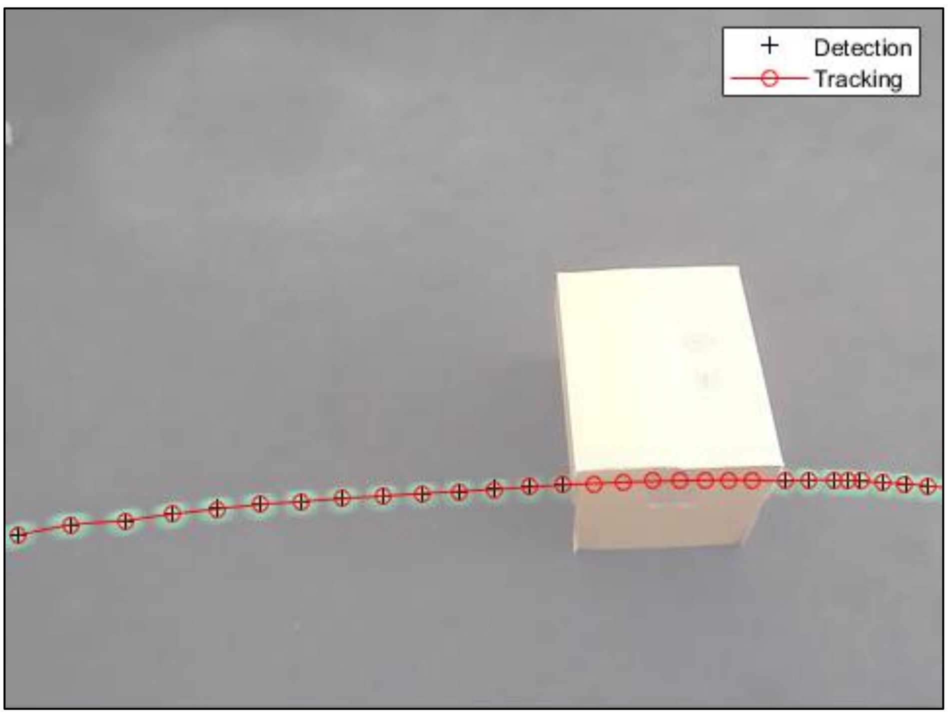


Figure 8: Tracking an object with Kalman filter. (The Mathworks Inc., 2018)

Before the Kalman filter can be implemented, three assumptions have to be made: (1) the system being modeled is a discrete-time linear dynamic system, (2) the measurement noise is "white", and (3) this noise is Gaussian [7]. The filter can be derived from first principles using one key property of the Gaussian distribution—that the product of two Gaussian distributions is another Gaussian distribution [8].

First, the matter of notation must be discussed. $\hat{x}_{n|m}$ represents the estimate of x at time n given observations up to and including the time at m ≤ n. The Kalman filter's system model assumes that the state of a system at time *t* is evolved from the previous state at time *t*-1, as shown:

$$\boldsymbol{x}_t = \boldsymbol{F}_t \boldsymbol{x}_{t-1} + \boldsymbol{B}_t \boldsymbol{u}_t + \boldsymbol{w}_t \tag{2}$$

Where $\boldsymbol{x}_t$ is the state vector at time *t*, $\boldsymbol{u}_t$ is the control input vector, $\boldsymbol{F}_t$ is the state transition matrix, $\boldsymbol{B}_t$ is the control input matrix, and $\boldsymbol{w}_t$ is the vector of the process noise of each state variable in the state vector. The state transition matrix applies the effect of each system state parameter at time *t*-1 to the system state at time *t.* The process noise is assumed to be zero-mean Gaussian white noise with the covariance given by the process noise covariance matrix $\boldsymbol{Q}_t$. Measurements of the system can also be taken according to the following model:

$$\boldsymbol{z}_t = \boldsymbol{H}_t \boldsymbol{x}_t + \boldsymbol{v}_t \tag{3}$$

Where $\boldsymbol{z}_t$ is the measurement vector, $\boldsymbol{H}_t$ is transformation matrix that maps state vector parameters into the measurement domain, and $\boldsymbol{v}_t$ is the vector of the measurement noise of each parameter in the measurement vector. Like the process noise, measurement noise is assumed to be zero-mean Gaussian white noise and is represented in the measurement noise covariance matrix $\boldsymbol{R}_t$.

The true state of the system $\boldsymbol{x}_t$ cannot be directly observed and so, the Kalman filter provides a method to produce an estimate of the state $\hat{\boldsymbol{x}}_t$ by combining models of the system and the noisy measurements of selected parameters. This estimate is provided by Gaussian probability density functions (pdfs). To fully describe the Gaussian functions, their variances and covariances are placed in the estimate error covariance matrix $\boldsymbol{P}_t$. The variances of the respective terms in the state vector are along the main diagonal of $\boldsymbol{P}_t$, and the remaining terms are the covariances between the elements of the state vector. It is the state estimate $\hat{\boldsymbol{x}}_t$ that is recursively calculated by combining prior knowledge, predictions from the system models, and noisy measurements.

The Kalman filter can be segmented into two stages: the prediction and measurement update. The equations for the prediction stage are:

$$\hat{\boldsymbol{x}}_{t|t-1} = \boldsymbol{F}_t \hat{\boldsymbol{x}}_{t-1|t-1} + \boldsymbol{B}_t \boldsymbol{u}_t \tag{4}$$

$$\boldsymbol{P}_{t|t-1} = \boldsymbol{F}_t \boldsymbol{P}_{t-1|t-1} {\boldsymbol{F}_t}^T + \boldsymbol{Q}_t \tag{5}$$

Equation (5) calculates the variance associated with the prediction found in Equation (4). The measurement update is given by the equations:

$$\hat{\boldsymbol{x}}_{t|t} = \hat{\boldsymbol{x}}_{t|t-1} + \boldsymbol{K}_t(\boldsymbol{z}_t - \boldsymbol{H}_t \hat{\boldsymbol{x}}_{t|t-1}) \tag{6}$$

$$\boldsymbol{P}_{t|t} = \boldsymbol{P}_{t|t-1} - \boldsymbol{K}_t \boldsymbol{H}_t \boldsymbol{P}_{t|t-1} \tag{7}$$

The expression $\left(\boldsymbol{z}_t - \boldsymbol{H}_t \hat{\boldsymbol{x}}_{t|t-1}\right)$ in (6) is the measurement innovation, or residual. The residual represents the difference between the observed measurement at time t and the predicted measurement at time t-1. The term $\boldsymbol{K}_t$ is the optimal gain which minimizes the error covariance found in Equation (7). This gain is often referred to as the Kalman gain and is given by,

$$\begin{aligned} \boldsymbol{K}_t &= \boldsymbol{P}_{t|t-1} {\boldsymbol{H}_t}^T (\boldsymbol{H}_t \boldsymbol{P}_{t|t-1} {\boldsymbol{H}_t}^T + \boldsymbol{R}_t)^{-1} \\ &= \frac{\boldsymbol{P}_{t|t-1} {\boldsymbol{H}_t}^T}{\boldsymbol{H}_t \boldsymbol{P}_{t|t-1} {\boldsymbol{H}_t}^T + \boldsymbol{R}_t} \end{aligned} \tag{8}$$

From Equation (8), as the measurement error covariance $\boldsymbol{R}_t$ approaches zero, the Kalman gain places more weight on the residual:

$$\lim_{\boldsymbol{R}_t \to 0} \boldsymbol{K}_t = {\boldsymbol{H}_t}^{-1} \tag{9}$$

Alternatively, as the estimate error covariance $\boldsymbol{P}_{t|t-1}$ approaches zero, the Kalman gain is less dependent on the residual [9]:

$$\lim_{\boldsymbol{P}_{t|t-1} \to 0} \boldsymbol{K}_t = 0 \tag{10}$$

At the time of every measurement, the prediction and measurement pdfs are multiplied together, resulting in another pdf that represents the best possible estimate of the system. As mentioned previously, it is this key property of the Gaussian function which makes the Kalman filter so widely used. It allows an endless number of Gaussian pdfs to be multiplied over time without any increase in the complexity of the result—the new pdf is completely represented by a Gaussian function.

4.3 Distance Measurement

Distance measurement is based on the observation that an object's size in an image will change relative to the distance between the object and the camera. This is shown in Figure 8, where F is the focal length. The focal length is defined as the distance from the center of a lens to its focus. By similar triangles, the focal length can be determined with the following equation, where P is the object's width on the image plane, D is the known distance between the object and camera, and W is the known width of the object [10]:

$$F = \frac{P \times D}{W} \tag{5.1}$$

Therefore, if both the distance to the object and the width of the object is known, the focal length can be inferred from the observed width of the object in the image. With the focal length found, all subsequent changes to the distance between the camera and the object can be determined by rearranging equation 1:

$$D = \frac{F \times W}{P} \tag{5.2}$$

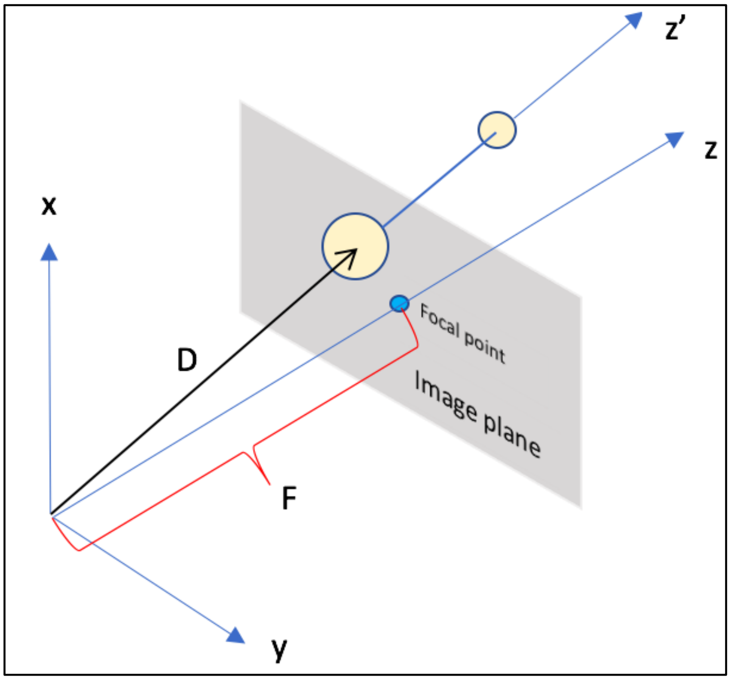


Figure 9: Overview of distance measurement.
Note the appearance of the object on the image plane, demonstrating the change in object size in relation to its distance from the viewpoint (the origin).

4.4 Servo Control

The radio controlled (RC) DC servo motor is controlled using pulse width modulation (PWM). The duty cycle varies from 0.5ms to 2ms, with a period of 16ms to 20ms. When the duty cycle is at its lowest value (0.5ms), the motor shaft moves all the way left (-90°). When the duty cycle is at the mid-way point (1.25ms), the motor shaft moves to the center (0°). When the duty cycle is at the highest value (2ms), the motor shaft moves all the way right (90°).

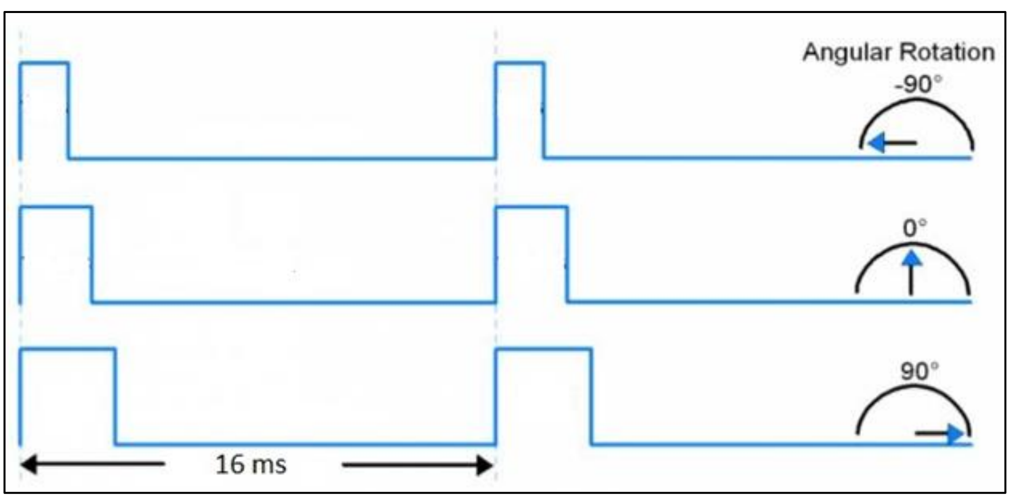


Figure 10: Servo Motor Pulse Width Modulation

The internal control circuit of the RC DC servo motor contains: a DC motor, a potentiometer and a voltage comparator. The DC motor's shaft is attached to the potentiometer. When the motor shaft moves, the potentiometer moves as well, causing a change in feedback voltage. The comparator has two inputs, the input control signal and the feedback voltage from the potentiometer. The comparator is used to detect the difference between the motors current position and the control signal. The feedback signal is subtracted from the control signal, resulting in an error signal. The motor will continue to rotate until the error signal reaches 0, indicating that the motor has reached its destination.

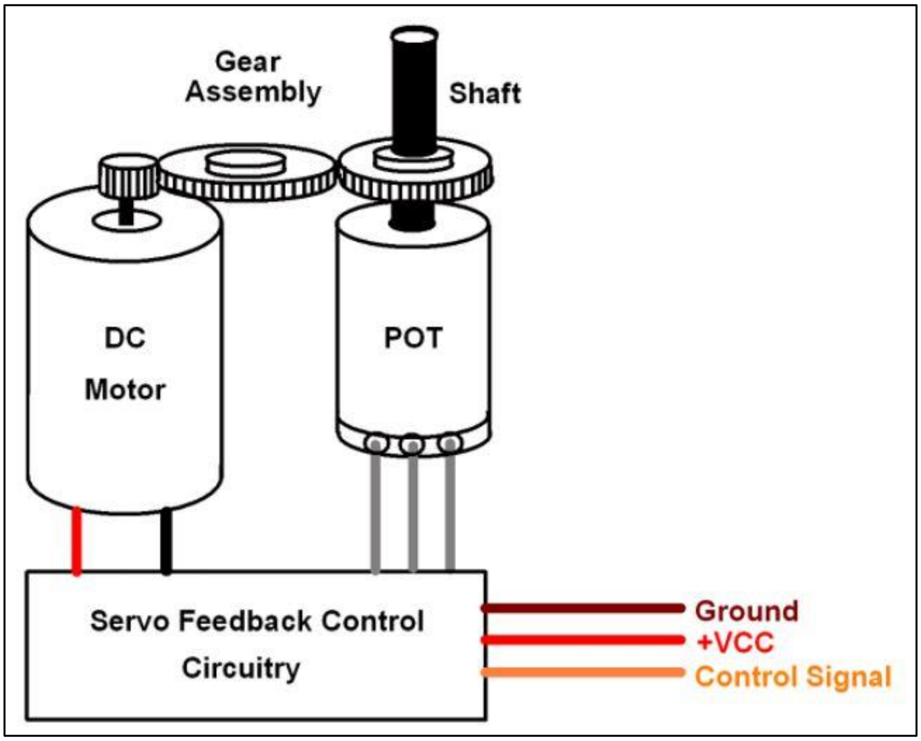


Figure 11: Servo Internal Control Circuit

A phase lock loop (PLL) is required in our design is to reduce the FPGA's 50 MHz clock to a desired frequency of 250 KHz. The PLL consists of; source frequency (F_{in}), counter block (N), reference frequency (F_{REF}), phase and frequency detection block (PFD), charge pump, loop filter, voltage controlled oscillator (VCO), VCO frequency (F_{VCO}), counter block (M), post divider (K) and output frequency (F_{OUT}). The N-block is used to decrease the input frequency by a factor of N. The M-block is used to decrease the VCO frequency by a factor of M. The PFD-block is used to detect the difference in frequency between the reference frequency and the VCO frequency that has been decreased by the M-block. The loop filter converts the difference to a voltage that is used to oscillate the VCO at a frequency that will make the feedback frequency equal to the reference frequency. The post divider K is used to further decrease the value of VCO frequency.

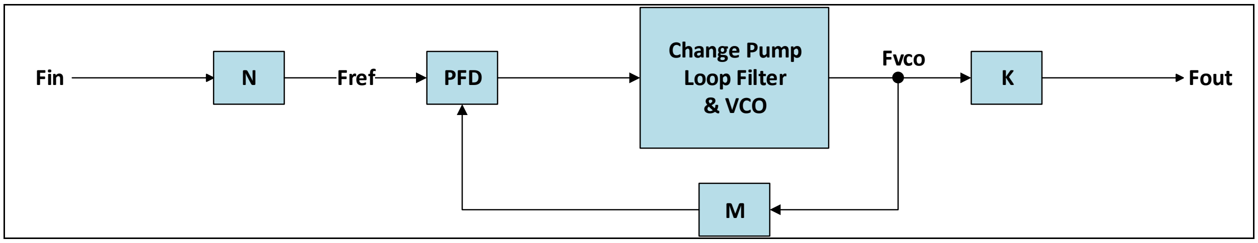


Figure 12: PLL

5.Design

5.1 Object Detection

The object will be detected by its shape and color. This object detection method was selected for its simplicity and performance, while satisfying the project objectives. A simple OpenCV application to threshold and display the image was created to obtain the range of the object's colors under different lighting conditions. Thresholding will be done in the HSV color space as opposed to the RGB representation to improve performance in more diverse lightning conditions. The HSV model characterizes a color in terms of its hue, saturation, and value. Unlike the RGB representation which uses three color channels, an HSV image separates luma, or the image intensity, from chroma, the color information [11]. This enables the system to become more robust to lighting changes and reduces errors due to shadows. The object's HSV color range was determined through trial and error. A black background was used in this process to create maximum contrast between the orange ball and background. The following values were obtained.

$$HSV_{lower} = (5, 0, 210)$$

$$HSV_{upper} = (40, 255, 255)$$

The shape of the object is determined by examining the contours of the binary image. Contours are the curves joining all the continuous points along boundaries contained in an image, as shown in Figure 10. These boundaries occur when there are distinct changes in color or intensity. Only the largest contour in the image should be considered to avoid detection errors. Because the object is a ball, the criteria we wish to check is the contour's roundness. If it is sufficiently round, then the ball is most likely detected. Imperfect thresholding was taken into consideration, as the ball will not always appear as a perfect circle in the binary image. Therefore, if the maximum width and height of the contour are within 25% of each other, we will consider the contour to be that of the ball's.

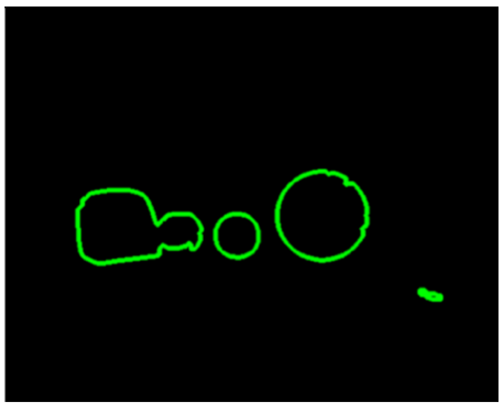


Figure 13: The contours found from the binary image shown in Figure 3b.

5.2 Kalman Filter

The Kalman filter requires the linear model of the system. To track the ball and measure distance, the state of system includes the position, velocity, and the size of ball in terms of pixels. Each frame captured by the camera contains only the position and the size of the ball. The state vector $\boldsymbol{x}_t$ and measurement vector $\boldsymbol{z}_t$ are as follows:

$$\boldsymbol{x}_t = \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \\ w \\ h \end{bmatrix}, \quad \boldsymbol{z}_t = \begin{bmatrix} x \\ y \\ w \\ h \end{bmatrix}$$

The corresponding state transition matrix $\boldsymbol{F}_t$ and measurement matrix $\boldsymbol{H}_t$ are:

$$\boldsymbol{F}_t = \begin{bmatrix} 1 & 0 & dT & 0 & 0 & 0 \\ 0 & 1 & 0 & dT & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\boldsymbol{H}_t = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

The control inputs of the system are the two servo motors that pan the camera in the vertical and horizontal axes. The control vector $\boldsymbol{u}_t$ and control matrix $\boldsymbol{B}_t$ are:

$$\boldsymbol{B}_t = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}, \quad \boldsymbol{u}_t = \begin{bmatrix} x \\ y \end{bmatrix}$$

The process noise covariance matrix $\boldsymbol{Q}_t$ is:

$$\boldsymbol{Q}_t = \begin{bmatrix} 1e-3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1e-3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1e-1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1e-1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1e-2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1e-2 \end{bmatrix}$$

The measurement noise covariance matrix $\boldsymbol{R}_t$ is:

$$\boldsymbol{R}_t = \begin{bmatrix} 1e-1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1e-1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1e-1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1e-1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1e-1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1e-1 \end{bmatrix}$$

The values of $\boldsymbol{Q}_t$ and $\boldsymbol{R}_t$ were determined through trial and error, while keeping Equations (8) to (10) in consideration. As we wish to obtain the best tracking performance behind obstacles, a low Kalman gain is desired. By reducing the process noise covariance $\boldsymbol{Q}_t$, the estimate error $\boldsymbol{P}_{t|t-1}$ is reduced, resulting in a small Kalman gain. The object's position was assumed to be much more certain than its velocity and dimensions, hence the smaller error given in the matrix $\boldsymbol{Q}_t$.

5.3 HPS Memory and Data Flow Design

There are limits on using only physical memory. For example, if the ARM is connected directly to the physical memory, all programs that are run on the processor will share the same physical memory address space. Thus, it is impossible to control what machine resources are accessed by a program. A solution to this problem, is to use a virtual memory address space between the ARM and the physical memory. Therefore, programs will each have a private address location that contains data and code. This will protect the program from other processes that are using the same physical memory address space. The operating system (OS) will create and manage the virtual memory address space. The address translation hardware, (controlled by the OS) will map the virtual addresses to physical addresses. The hardware that maps the virtual addresses to the physical addresses is the ARM's Memory Management Unit (MMU). The MMU will be managed by the Linux LXDE OS.

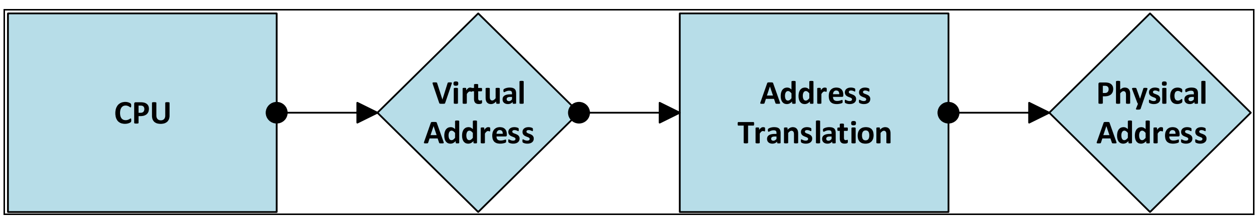


Figure 14: Block diagram of memory mapping.

To be able to output data from the processor a method of transferring data is required. The method of choice is to create Parallel Input/Ouput (PIO) on the HPS side. The PIO is created with a width of 32-bits matching the size of the Lightweight HPS-to-FPGA AXI bridge. Since the main image processing program, (run on the HPS) will produce four outputs there will be four PIO's created. The first two outputs contain the actual X and Y position, and the other two contain the predicted X and Y position. Therefore, the HPS will map the virtual address to the physical address. Then, the data will be sent over the Lightweight HPS-to-FPGA AXI bridge to the FPGA fabric. The FPGA fabric will then use the data to control the servo motors.

5.4 Servo Control

A servo motor with high resolution can make very precise movements. The PWM period was designed with this in mind. For each millisecond there is 256-bits, each bit indicating a new position for the servo motor to travel to.

$$size\ of\ counter = \frac{256bits}{ms} * 16ms = 4096bits$$

$$12bit\ counter = 2^{12} = 4096$$

This resulted in a counter with 4096-bits. Therefore, a 12-bit counter will give the appropriate period the servo motors need to operate. When the counter reaches its maximum value of $2^{12} - 1$, the counter will reset and start the period over again.

$$Frequency\ of\ PWM\ period = \frac{1}{T} = \frac{1}{16ms} = 62.5Hz$$

$$Frequency\ of\ Clock = Frequency\ of\ PWM\ period * size\ of\ counter$$

$$= 62.5Hz * 4096 = 256KHz$$

Using a 12-bit counter to create a 16ms period requires a frequency of 256 KHz. To get a reduced frequency, an Altera PLL IP block was used. The following parameters were chosen to get as close as possible to 256 KHz.

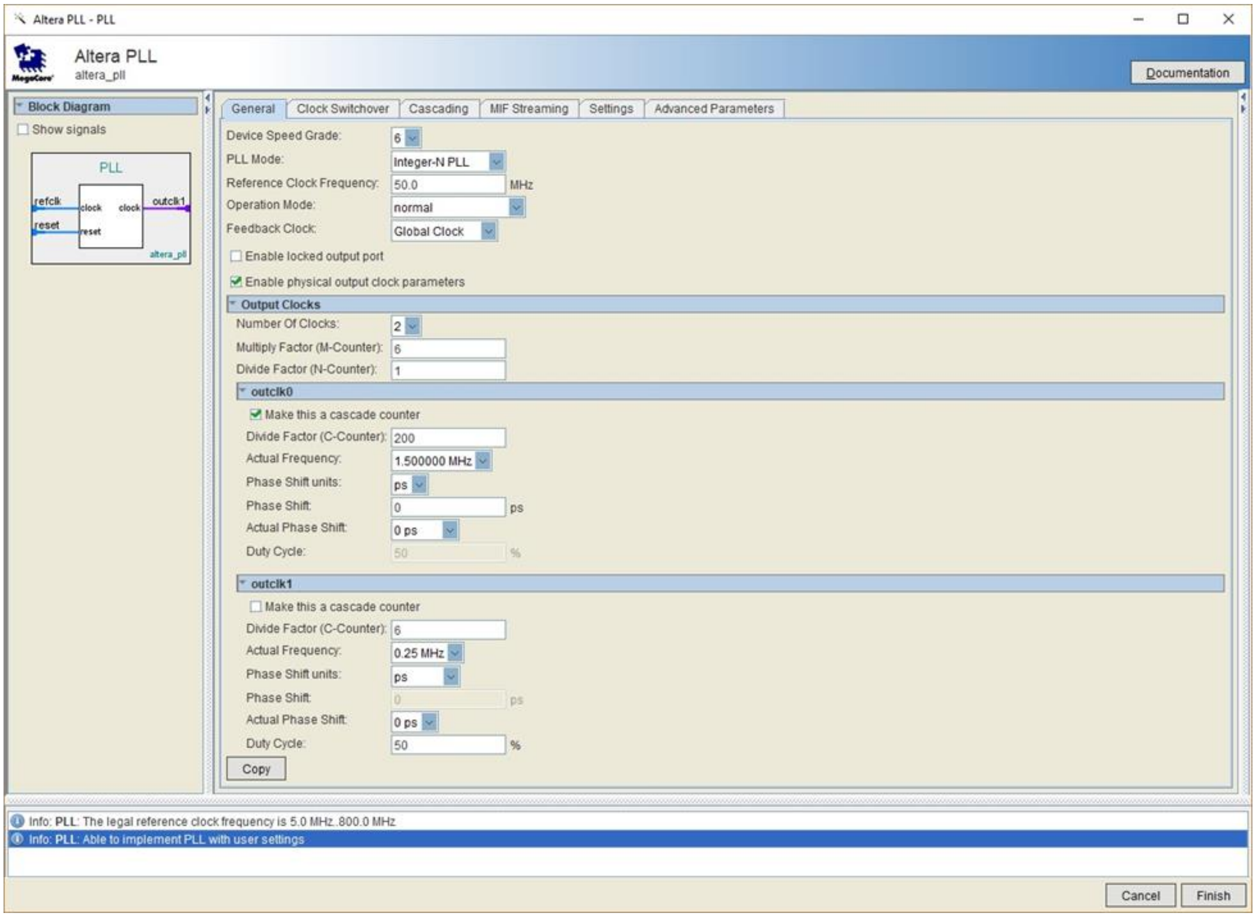


Figure 15: Altera PLL IP block

$$N = 1$$

$$M = 6$$

$$C - counter1 = 200$$

$$C - counter2 = 6$$

Using the above parameters, the following frequencies are found:

$$F_{IN} = 50MHz$$

$$F_{REF} = \frac{F_{IN}}{N} = 50MHz$$

$$F_{VCO} = \frac{F_{IN} * M}{N} = 300MHz$$

$$F_{OUT} = \frac{F_{IN} * M}{N * C - counter1 * C - counter2} = \frac{F_{VCO}}{C - counter1 * C - counter2} = 250KHz$$

$$T_{PWM} = \frac{size\ of counter}{F_{OUT}} = \frac{4096}{250KHz} = 16.384ms$$

This period is very close to the desired and is within the required 16-20ms and, therefore, is satisfactory.

6.Implementation

6.1 Tools used

The object tracking script was built using the Open Source Computer Vision (OpenCV) library of functions. Originally developed by Intel, OpenCV is the most popular computer vision library today. It is cross-platform and was designed for computational efficiency, with a strong focus on real-time applications. Originally, the object tracking program was developed in a Windows environment using Python. Python was the first choice due to its simplicity and ease of use in the Windows environment. However, it became apparent that Python was too inefficient when running on the DE10-Nano. A decision was made to rewrite the object tracking code in C++. The Verilog code was written and compiled in Quartus Prime Lite, provided by Intel. The HPS setup was done within Quartus Prime Lite using Intel's system integration tool, Qsys.

The early design flow consisted of writing the C++ code in Notepad++ on a Windows machine, sending it over to the DE10 through an ethernet connection using the PSFTP application included with PuTTY, and then compiling the code natively on the board through a serial connection in PuTTY. To speed up the compilation process, a Makefile was created. Later in the project, the C++ code was developed in a Linux virtual machine running Ubuntu. There are two advantages to this – compiling and executing the program on the development machine further expedites the build process, and the program execution on a separate machine provides a benchmark to measure the build's performance on the DE10-Nano.

6.2 AXI Implementation

Terasic provides multiple reference designs that contain all the necessary information required to setup the various components within Qsys. The reference designs can be found on Terasic's website under the CD-ROM section (see Resources Used). The first component discussed is the processor. The reference design used to setup the processor is the Golden Hardware Reference Design (GHRD). The main item discussed here is the FPGA interfaces section. This section, will setup the Advanced eXtensible Interface (AXI) bus that connects the processor to the FPGA resources, see Figure 16. There are three AXI bridges used in this SoC. The first, is the FPGA-to-HPS which can have a configurable data width up to 128 bit. The second, is the HPS-to-FPGA also configurable up to 128 bit. Finally, the Lightweight HPS-to-FPGA bridge which has a data width of only 32-bits. This design uses only two bridges the FPGA-to-HPS and Lightweight HPS-to-FPGA bridge. The first bridge is set to 128 bits. This is important because it will allow a large enough data width for the High Definition Multimedia Interface (HDMI) connection. The Lightweight bridge is used for sending data between the programs run on the HPS side and the FPGA fabric. The FPGA-to-HPS SDRAM Interface must be configured. The bridge is bidirectional and must be able to contain 160 bits. SDRAM comes in sizes that are of a power of 2, thus the width is set to 256 bits.

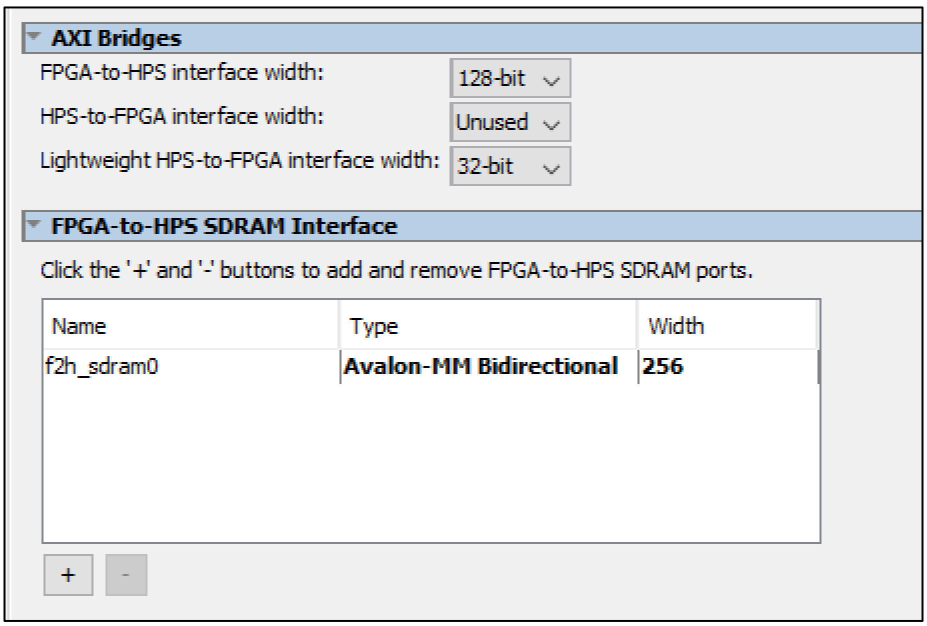


Figure 16: The configuration of the AXI bridges.

The connection of the components to the HPS is done in Qsys by connecting the AXI master port on the HPS to the AXI memory mapped slave port of the IP Core. There is an Avalon-MM Pipelined bridge between the FPGA components and the HPS AXI master port. This bridge facilitates connection between Lightweight HPS-to-FPGA bridge and system components. The Pipelined bridge is entered manually, however if not inserted manually Quartus will automatically insert this bridge during compilation. For example, the dip switches' memory mapped slave is connected to the pipelined bridge's memory mapped master. See Figure 17 below.

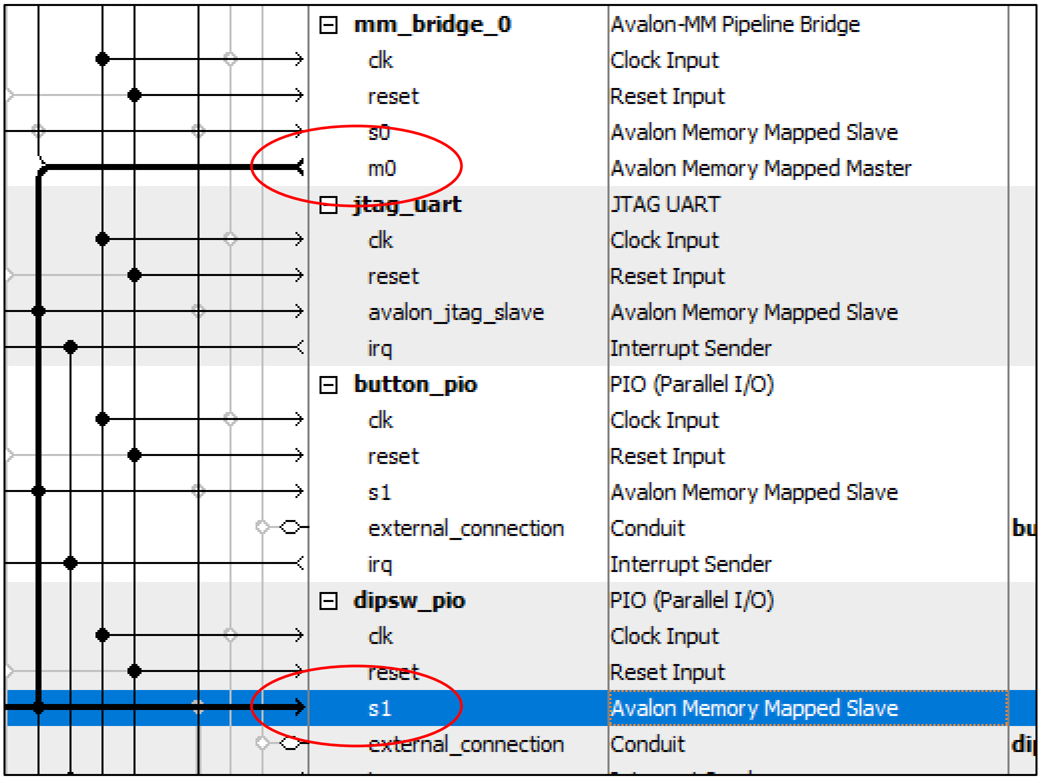


Figure 17: Example connection between Pipeline Bridge and dip switches.

The slave of the bridge is then connected to the AXI master port on the HPS, see Figure 18. This same procedure outlined in Figure 17 and Figure 18 is used to connect all the PIO in this project.

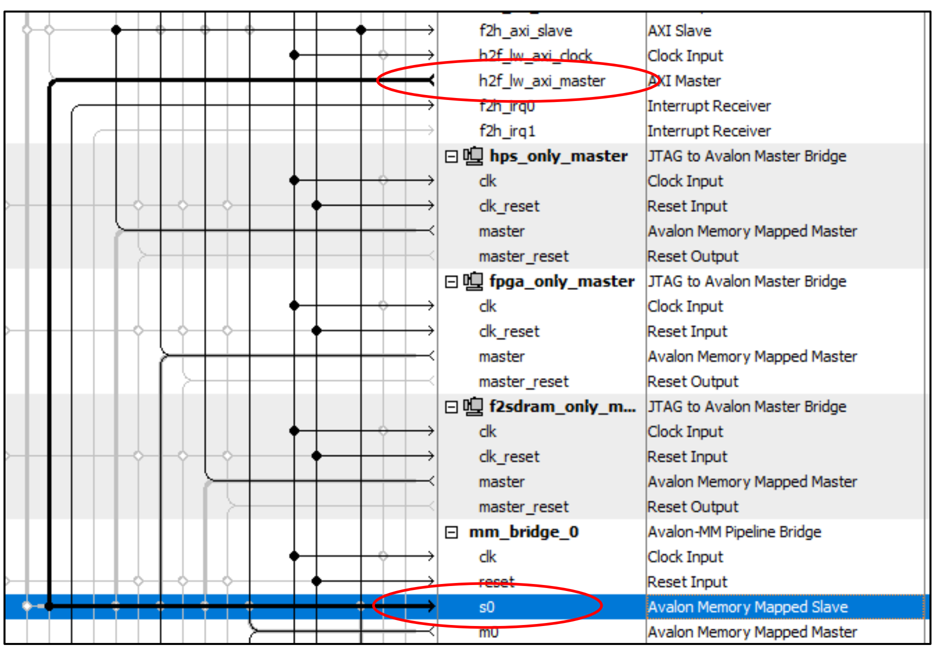


Figure 18: Connection from Pipeline Bridge to HPS AXI master port.

6.3 Memory Implementation

In this design the concept of virtual memory is important as the HPS must be able to send data to the various hardware components. The base physical addresses of the various components are assigned automatically through Qsys. However, if there is a conflict one can manually adjust the addresses. Table 1 below shows the physical addresses for the hardware used in this design.

Table 1: Physical memory addresses.

Hardware	Base address (dec)	End Address (dec)	Base Address (hex)	End Address (hex)
Push Buttons	20480	20495	0x00005000	0x0000500F
Dip Switches	16384	16399	0x00004000	0x0000400F
LEDs	12288	12303	0x00003000	0x0000300F
Predicted Servo X	32	63	0x00000020	0x0000003F
Actual Servo X	160	191	0x000000A0	0x000000BF
Predicted Servo Y	128	159	0x00000080	0x0000009F
Actual Servo Y	96	127	0x00000060	0x0000007F

Terasic kindly provides a shell script that will create a header file with the addresses discussed. The details of the header file are located in the Appendix F. The main C++ program used in this project utilizes all the hardware listed in Table 1. Therefore, it is important to include the header file in the source code. Figure 19, Figure 20, Figure 21, and Figure 22 below outline how the HPS accesses the physical memory using this header file. First, the header file must be included, then the base address and the address span of the Lightweight HPS-to-FPGA bridge are included.

```
#include <sys/mman.h>
#include "hps_0.h"
...
#define REG_BASE 0xff200000 //LW H2F Bridge address
#define REG_SPAN 0x00200000 //LW H2F Bridge Span
```

Figure 19: Physical memory access code (a).

The memory must then be opened to allow read and write access.

```
    fd = open("/dev/mem", O_RDWR|O_SYNC);
```

Figure 20: Physical memory access code (b).

The mapping is then done with the command mmap:

```
    base = mmap(NULL, REG_SPAN, PROT_READ|PROT_WRITE, MAP_SHARED, fd, REG_BASE);
```

Figure 21: Physical memory access code (c).

The first argument is the address, since it is NULL the kernel chooses the address where the mapping is created. The next argument is the length, which is the size of the memory mapping. The third argument, contains the desired memory protection. In this case, memory pages may be read and written. The fourth argument, contains flags that indicate weather memory mapping updates are visible to other processes that are mapping in the same region. In this project memory mapping updates are being shared. The fifth argument, contains the memory location. The last argument, contains the offset this indicates where to start writing the data to memory. Finally, the base addresses in the header file are added to the mmap base:

```
    servo_x_pred = (uint32_t*)(base+SERVO_PRED_X_BASE);
    servo_x_act = (uint32_t*)(base+SERVO_ACT_X_BASE);
    servo_y_pred = (uint32_t*)(base+SERVO_PRED_Y_BASE);
    servo_y_act = (uint32_t*)(base+SERVO_ACT_Y_BASE);
```

Figure 22: Physical access code (d).

The variables servo_x_pred and servo_x_act are 32 bit unsigned integers to match the width of the Lightweight HPS-to-FPGA bridge. The X direction predicted data will be written to the servo_x_pred variable and likewise the actual data written to servo_x_act. The exact same procedure occurs for the Y direction variables.

6.4 HDMI Implementation

The video data is sent through the FPGA-to-HPS bridge. Figure 23 below is a block diagram that shows the 3 different bridges and their respective connections to the FPGA fabric.

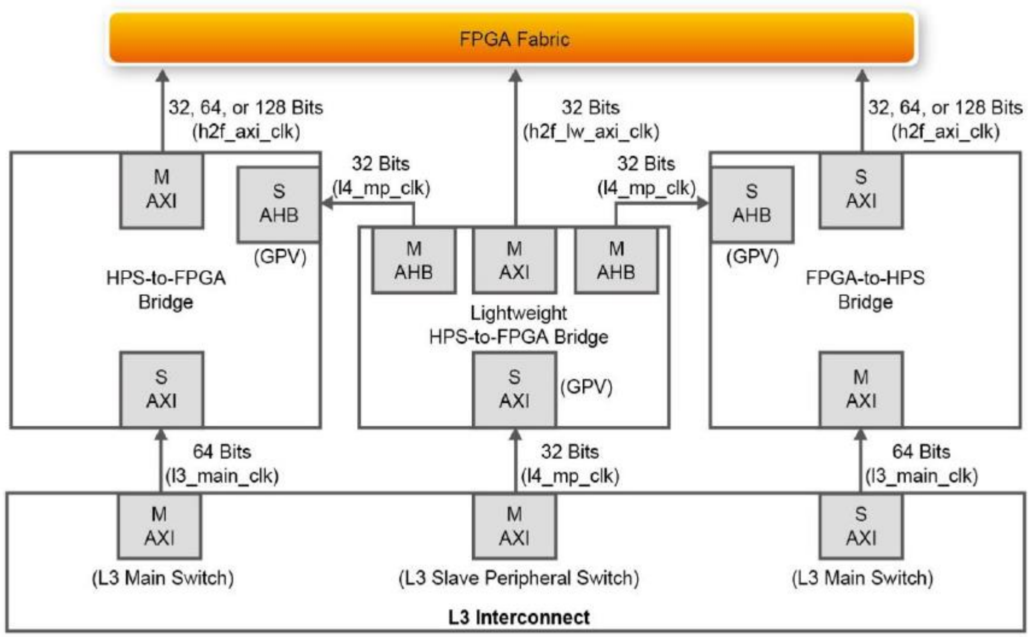


Figure 23: AXI bridges with regards to FPGA fabric, courtesy of Terasic.

The HDMI Transmitter chip is the ADV7513 this chip is controlled by an Inter-Integrated (I^2C) controller. Terasic provides the code for this controller, along with other related modules, by way of the Control Panel Reference design. The code can be found in Appendix H. The HDMI transmitter requires a 65MHz clock, see Figure 25. The clocks are generated from a phase locked loop (PLL) which creates two output clocks. The first output is the 65MHz clock, labelled as outclk0, which is connected to the HDMI transmitter chip. The second clock output is the 130MHz clock labelled as outclk1, this output is connected to the Clock Bridge in the HPS design. Verilog code that connects the 65MHz clock to the HDMI transmitter is found in Appendix H.1 line 16. To create the PLL Quartus Prime Lite contains a MegaCore wizard that is a graphical user interface. The MegaCore is the Altera PLL found in the Quartus IP Catalog. The MegaCore will generate Verilog files that are added to the project. The data used in the MegaCore was obtained from Terasic's Control Panel reference design. For the Verilog code generated by the MegaCore wizard see Appendix H.1 lines 9 to 15. Figure 24 is a block diagram that shows the video data flow.

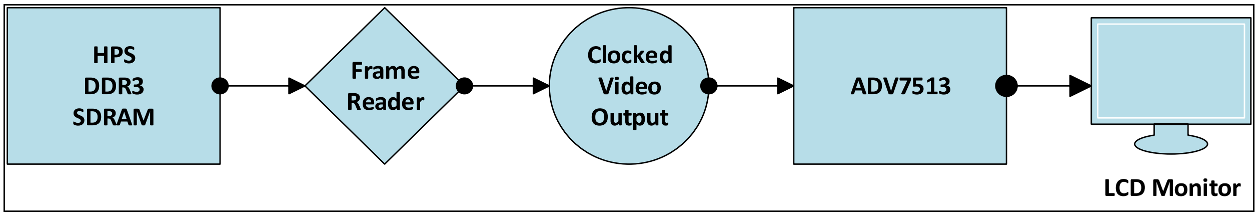


Figure 24: Video data flow.

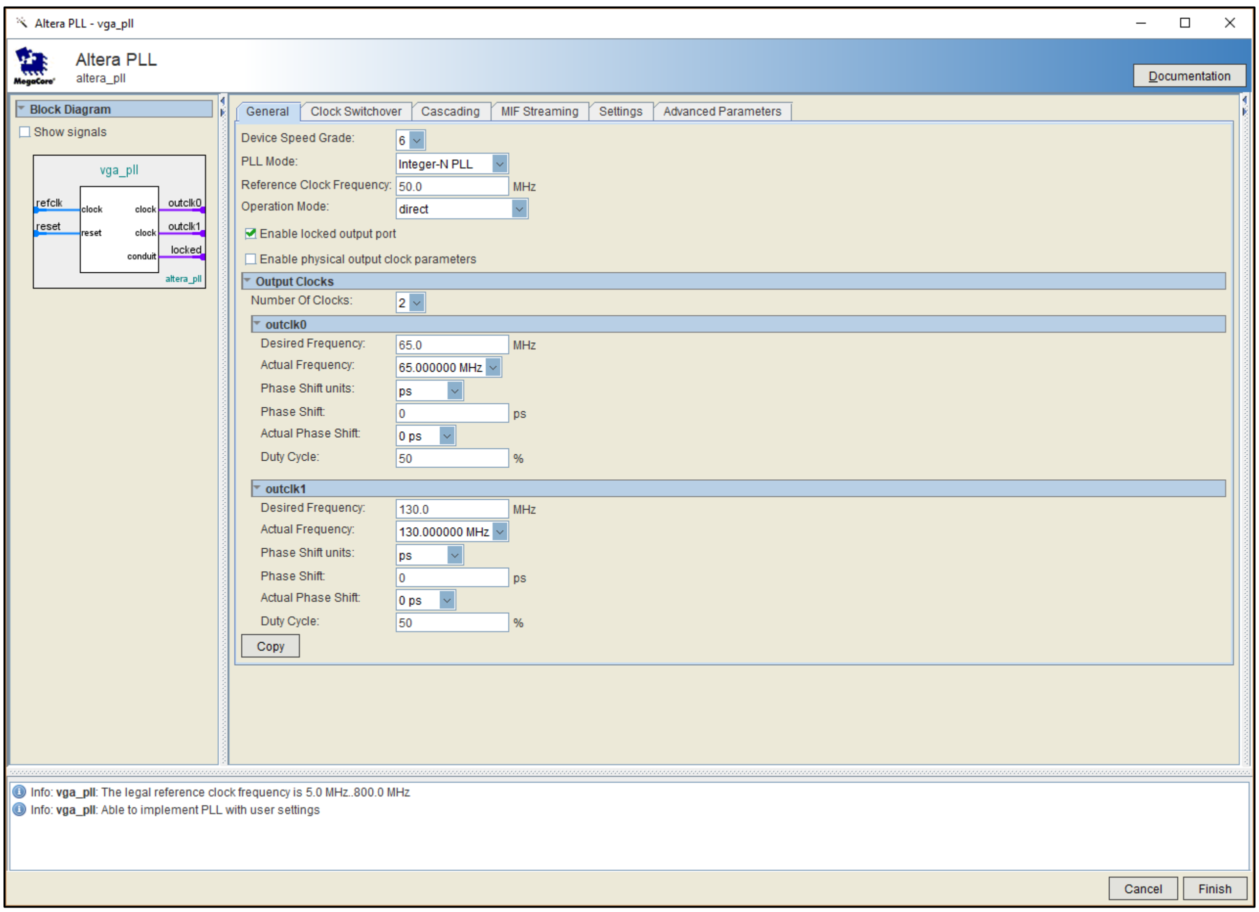


Figure 25: Altera PLL MegaCore.

Additionally, there is a frame reader that will read a video stream from video frames stored in a memory buffer. The configuration data for this core is found in Terasic's control panel reference design, also located on the demonstration CD-ROM. The HPS DDR3 SDRAM will be the memory buffer that the frame reader will access. The frames are sent over the FPGA-to-HPS AXI bridge. With regards to the configuration, the Master port width must match the size of the FPGA-to-HPS interface width as seen in Figure 26. The Bits per pixel per color plane is the number of bits used in each pixel per color plane. Number of color planes in parallel is the number of planes transmitted in parallel. The Number of color planes in sequence is the number of planes transmitted in sequence. Maximum Image width and height define the video resolution. The last two are self-explanatory.

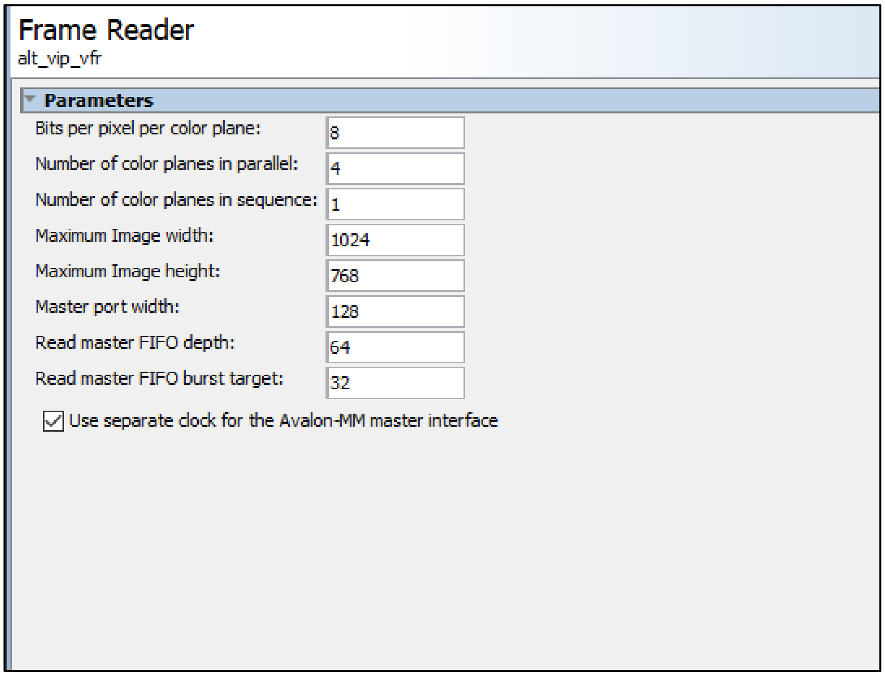


Figure 26: Frame reader configuration.

Furthermore, the Clocked Video Output is responsible for converting the Avalon-ST video stream from the Frame Reader into the 1024p format used by the HDMI transmitter chip. Avalon-ST video is Intel's standard video transmitting and receiving protocol. All video IP cores will transmit and receive video in the Avalon-ST video format. Therefore, the conversion is necessary when sending video to the external transmitter. The transmitter chip included on the DE10-Nano board is the ADV7513 by Analog Devices. The chip uses HDMI v1.4 features and with a frequency support of up to 165MHz it can support all video formats up to 1080p and UXGA. The Image Data format must match that which was configured in the frame reader. The rest of the data is obtained from the Control Panel reference design. See Figure 27 and Figure 28 for more detail on the configuration.

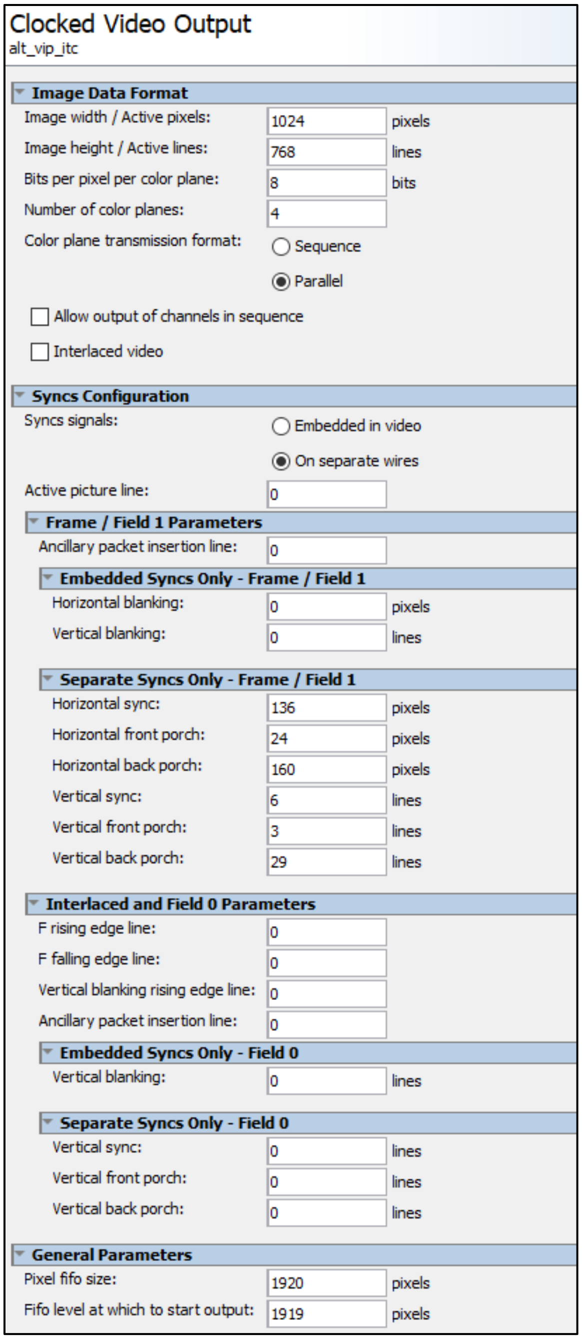


Figure 27: Clocked Video Output configuration (a).

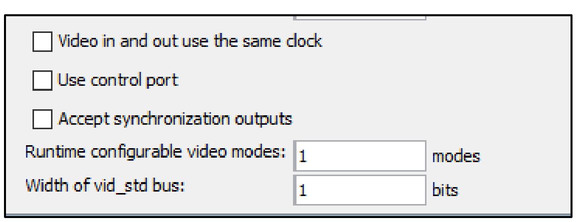


Figure 28: Clocked Video Output configuration (b).

6.5 Servo PIO Implementation

To fully understand how the HPS interacted with the FPGA peripherals, testing was required. The test involved attaching the GPIO pins to the HPS in the software. Furthermore, attaching off-board LEDs to the GPIO pins, this would allow a visual demonstration of the operation. A C program was written based on the memory mapping concept discussed in section 5.3. The program will map the virtual address to the physical address. Thus, allowing data to be sent to the FPGA fabric. See the code below for more details.

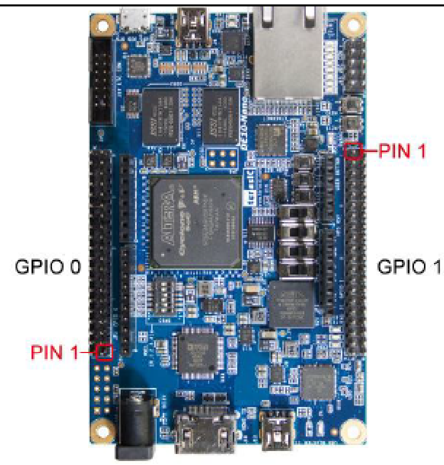
```c
void handler(int signo)
{
 *GPIO=0;
 munmap(base, REG_SPAN);
 close(fd);
 exit(0);
}

int main()
{
 fd=open("/dev/mem", O_RDWR|O_SYNC);
 if(fd<0)
 {
     printf("Can't open memory .\n");
     return -1;
 }
 base=mmap(NULL, REG_SPAN, PROT_READ|PROT_WRITE, MAP_SHARED, fd, REG_BASE);
 if(base==MAP_FAILED)
 {
     printf("Can't map memory. \n");
     close(fd);
     return -1;
 }
 GPIO=(uint32_t*)(base+GPIO_0_BASE);
 signal(SIGINT, handler);
 *GPIO=0x1;
 while(1)
 {
     *GPIO = 0x000000ff;
 }
 //Unnmap
 munmap(base, REG_SPAN);

 close(fd);
 return(0);
}
```

Figure 29: Code to test the connection between Processor and FPGA peripherals.

Line 29 in Figure 29 contains the data that is sent to the GPIO pins. The data is an unsigned 32 bit integer. In this case, 0x000000FF will activate the first 8 GPIO pins. That is, GPIO_0[0] to GPIO_0[7] see Figure 30.



GPIO 0 (JP1)

PIN_V12	GPIO_0[0]	1	2	GPIO_0[1]	PIN_E8
PIN_W12	GPIO_0[2]	3	4	GPIO_0[3]	PIN_D11
PIN_D8	GPIO_0[4]	5	6	GPIO_0[5]	PIN_AH13
PIN_AF7	GPIO_0[6]	7	8	GPIO_0[7]	PIN_AH14
PIN_AF4	GPIO_0[8]	9	10	GPIO_0[9]	PIN_AH3
	5V	11	12	GND	
PIN_AD5	GPIO_0[10]	13	14	GPIO_0[11]	PIN_AG14
PIN_AE23	GPIO_0[12]	15	16	GPIO_0[13]	PIN_AE6
PIN_AD23	GPIO_0[14]	17	18	GPIO_0[15]	PIN_AE24
PIN_D12	GPIO_0[16]	19	20	GPIO_0[17]	PIN_AD20
PIN_C12	GPIO_0[18]	21	22	GPIO_0[19]	PIN_AD17
PIN_AC23	GPIO_0[20]	23	24	GPIO_0[21]	PIN_AC22
PIN_Y19	GPIO_0[22]	25	26	GPIO_0[23]	PIN_AB23
PIN_AA19	GPIO_0[24]	27	28	GPIO_0[25]	PIN_W11
	3.3V	29	30	GND	
PIN_AA18	GPIO_0[26]	31	32	GPIO_0[27]	PIN_W14
PIN_Y18	GPIO_0[28]	33	34	GPIO_0[29]	PIN_Y17
PIN_AB25	GPIO_0[30]	35	36	GPIO_0[31]	PIN_AB26
PIN_Y11	GPIO_0[32]	37	38	GPIO_0[33]	PIN_AA26
PIN_AA13	GPIO_0[34]	39	40	GPIO_0[35]	PIN_AA11

GPIO 1 (JP7)

PIN_Y15	GPIO_1[0]	1	2	GPIO_1[1]	PIN_AC24
PIN_AA15	GPIO_1[2]	3	4	GPIO_1[3]	PIN_AD26
PIN_AG28	GPIO_1[4]	5	6	GPIO_1[5]	PIN_AF28
PIN_AE25	GPIO_1[6]	7	8	GPIO_1[7]	PIN_AF27
PIN_AG26	GPIO_1[8]	9	10	GPIO_1[9]	PIN_AH27
	5V	11	12	GND	
PIN_AG25	GPIO_1[10]	13	14	GPIO_1[11]	PIN_AH26
PIN_AH24	GPIO_1[12]	15	16	GPIO_1[13]	PIN_AF25
PIN_AG23	GPIO_1[14]	17	18	GPIO_1[15]	PIN_AF23
PIN_AG24	GPIO_1[16]	19	20	GPIO_1[17]	PIN_AH22
PIN_AH21	GPIO_1[18]	21	22	GPIO_1[19]	PIN_AG21
PIN_AH23	GPIO_1[20]	23	24	GPIO_1[21]	PIN_AA20
PIN_AF22	GPIO_1[22]	25	26	GPIO_1[23]	PIN_AE22
PIN_AG20	GPIO_1[24]	27	28	GPIO_1[25]	PIN_AF21
	3.3V	29	30	GND	
PIN_AG19	GPIO_1[26]	31	32	GPIO_1[27]	PIN_AH19
PIN_AG18	GPIO_1[28]	33	34	GPIO_1[29]	PIN_AH18
PIN_AF18	GPIO_1[30]	35	36	GPIO_1[31]	PIN_AF20
PIN_AG15	GPIO_1[32]	37	38	GPIO_1[33]	PIN_AE20
PIN_AE19	GPIO_1[34]	39	40	GPIO_1[35]	PIN_AE17

Figure 30: Details of GPIO pin layout from Terasic's User Manual.

The behavior observed in this test proved crucial as it lead to a better understanding of how data sent would affect the output. Once this test was complete the Servo PIO was implemented as it was discussed in the design section. These PIO ports are the necessary inputs to the code that controls the servo motors. The predicted (servo_pred_x and servo_pred_y), and the actual (servo_act_x and servo_act_y). The ports contain the position of the ball in the video frame and are used to relay this information to the hardware on the FPGA side. As the names suggest, the predicted contains the predicted values and the actual contains the actual values. Since the servo motors are off board peripherals this "bit banging" method of sending data will suffice. Figure 31 contains the details of how the PIO are setup. All four Servo Parallel Input/Outputs are configured the same way.

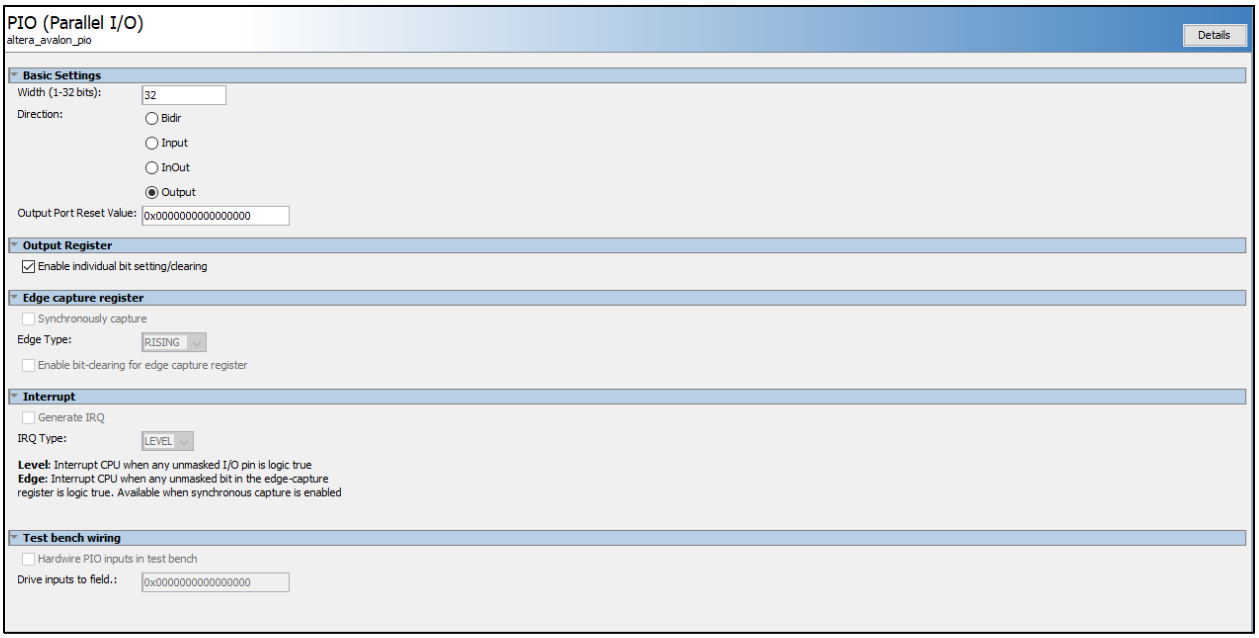


Figure 31: Servo PIO configuration.

6.6 Object Tracking

6.6.1 Overview of C++ Implementation

The object tracking was written in C++ using the OpenCV libraries with the intention to achieve all image processing operations on the ARM processor. The decision to implement object tracking strictly on the ARM processor was the result of the time limitation placed on this project. FPGA's can accomplish the same image processing tasks at a much faster rate and at a lower energy consumption. A brief description of the code can be found in Figure 30 below, and the entire code is listed in Appendix G. The OpenCV headers used in this project is listed in Figure 31. Note that mages in OpenCV are stored in matrices of type cv::Mat(). This greatly simplifies the…

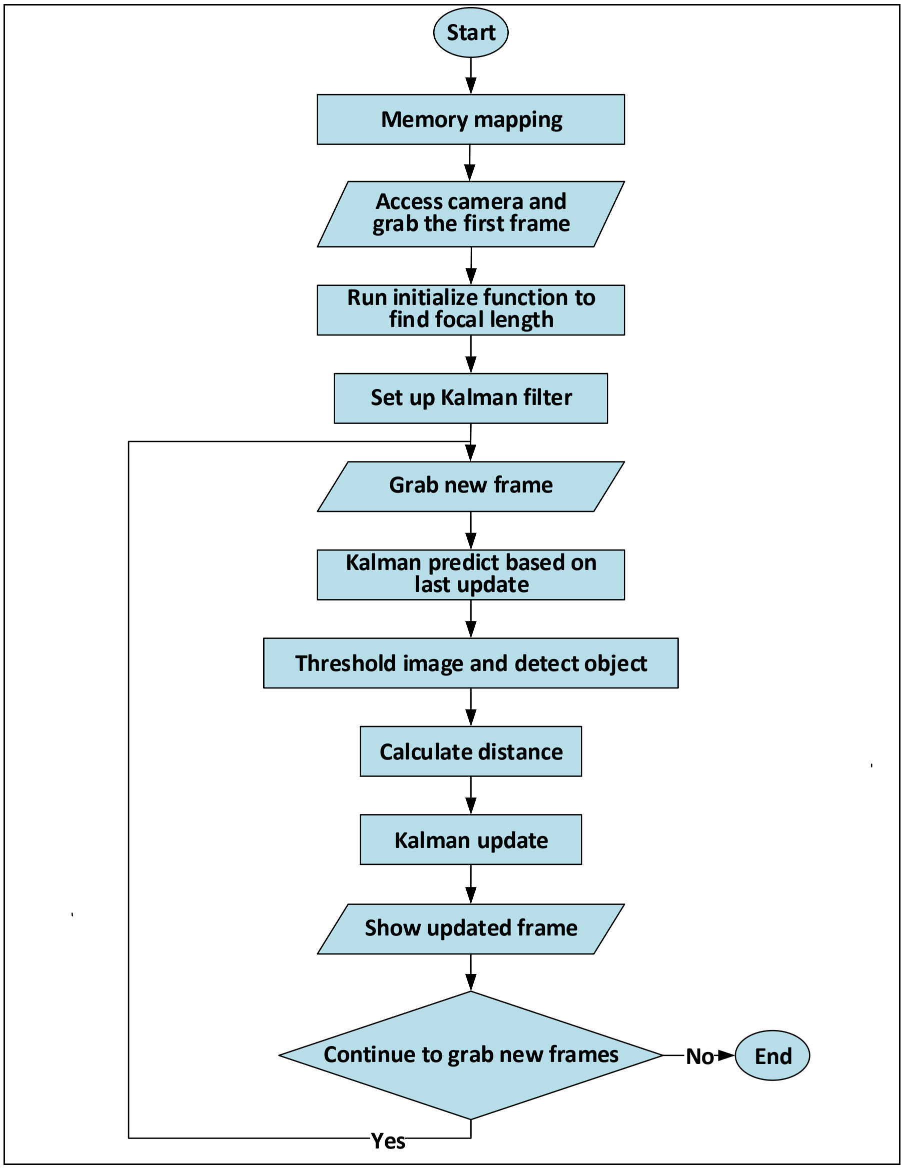


Figure 32: Overview of object detection and tracking C++ script.

```
#include "opencv2/videoio.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/imgproc.hpp"
#include <opencv2/video/video.hpp>
#include <opencv2/core/core.hpp>
```

Figure 33: OpenCV headers used in project.

6.6.2 Thresholding

The thresholding process is achieved in five steps. The image is blurred, converted from RGB to HSV, thresholded for the desired color range, and then enhanced via morphological operations. The OpenCV functions used here are cv::GaussianBlur(), cv::cvtColor(), cv::inRange(), cv::erode(), and cv::dilate(). Note that temporary matrices blur, frame_hsv, and mask were used to store the results of each operation.

```
        //smooth out noise with 5x5 Gaussian kernel, sigma X & Y are 3.0
        Mat blur;
        GaussianBlur(frame, blur, Size(5,5), 3.0, 3.0);

        //change to HSV color space
        Mat frame_hsv;
        cvtColor(frame, frame_hsv, COLOR_BGR2HSV);

        //threshold for desired color range
        Mat mask;
        inRange(frame_hsv, HSV_lower, HSV_upper, mask);

        //smooth edges with morphological operations
        //to remove noise and isolate contours
        erode(mask, mask, Mat(), Point(-1,-1), 2);
        dilate(mask, mask, Mat(), Point(-1,-1), 2);
```

Figure 34: Thresholding implementation in C++

6.6.3 Object Detection

The next step is to identify the object in the binary image. The object detection stage of the code is outlined in Figure 33 and the actual code used is shown in Figure 34. Since the color of the object is filtered through in the previous step, what remains is the detection of the object's shape. For this, we use the function cv::findContours() to find all contours in the binary image. The contours are stored in a vector of type cv::vector. The largest contour is found by comparing the area contained in a contour using cv::contourArea(). A bounding box is then placed around the largest contour, utilizing the function cv::boundingRect(). A bounding box is a rectangle whose dimensions are determined by the extreme points of the contour. The ratio of width to height of the bounding box is then compared. A ratio of greater than 0.75 implies that the selected contour is most likely round, and therefore, most likely the ball we wish to track.

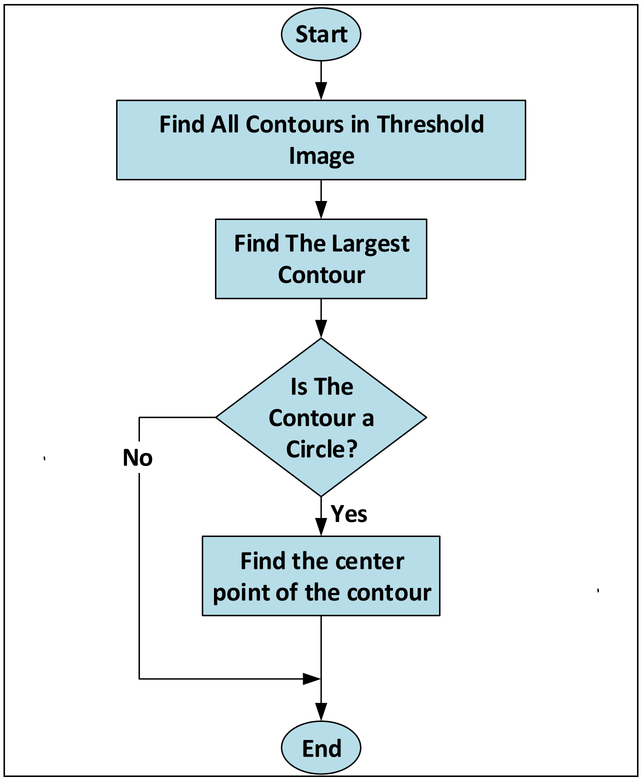


Figure 35: Overview of object detection code

```
        Rect bBox;

        if(contours.size()>0)
        {
            //iterate through each contour
            for(size_t i=0; i<abs(contours.size()); i++)
            {
                is_square = false; //reset "squareness" check
                double area = contourArea(contours[i]);//find area of contour
                //determine the largest contour in terms of area
                if(area>largest_area)
                {
                    largest_area = area;
                }

                //check for "roundness"
                bBox = boundingRect(contours[i]); //returns top left vertex

                float ratio = (float) bBox.width / (float) bBox.height;
                if (ratio > 1.0f)
                    ratio = 1.0f / ratio;

                // Searching for a bBox almost square
                if (ratio > 0.75)
                {
                    is_square = true;
                }
            }
```

Figure 36: Finding the object based on its captured shape.

The final core step of object detection is to find the object's coordinates with respect to the captured frame. This step can only execute if the detected object is sufficiently round and of certain size. The *x* and *y* position is stored in a cv::Point class. The implemented code is shown Figure 35 below. The remaining code in the object detection stage is to draw the coordinate point and the bounding box of the detected object onto the frame.

```
//only proceed if radius meets minimum value and bounding rectangle is square
            if(bBox.width > 1 && is_square)
            {
                Point actual_center; //pixel (0,0) is top-left vertex
                actual_center.x = bBox.x + bBox.width / 2;
                actual_center.y = bBox.y + bBox.height / 2;
```

Figure 37: Finding the object's position.

6.6.4 Kalman Filter

The Kalman filter was implemented in C++ script in the structure shown below in Figure 36. The built-in function cv::KalmanFilter() is used to simplify the implementation of the filter. The Kalman filter was set up according to Figures 37 and 38. The variables stateSize, measSize, and contrSize contain the number of elements in the state vector, measurement vector, and control vector, respectively. The parameters of the Kalman filter are implemented according to the design values found in Section 4.2.

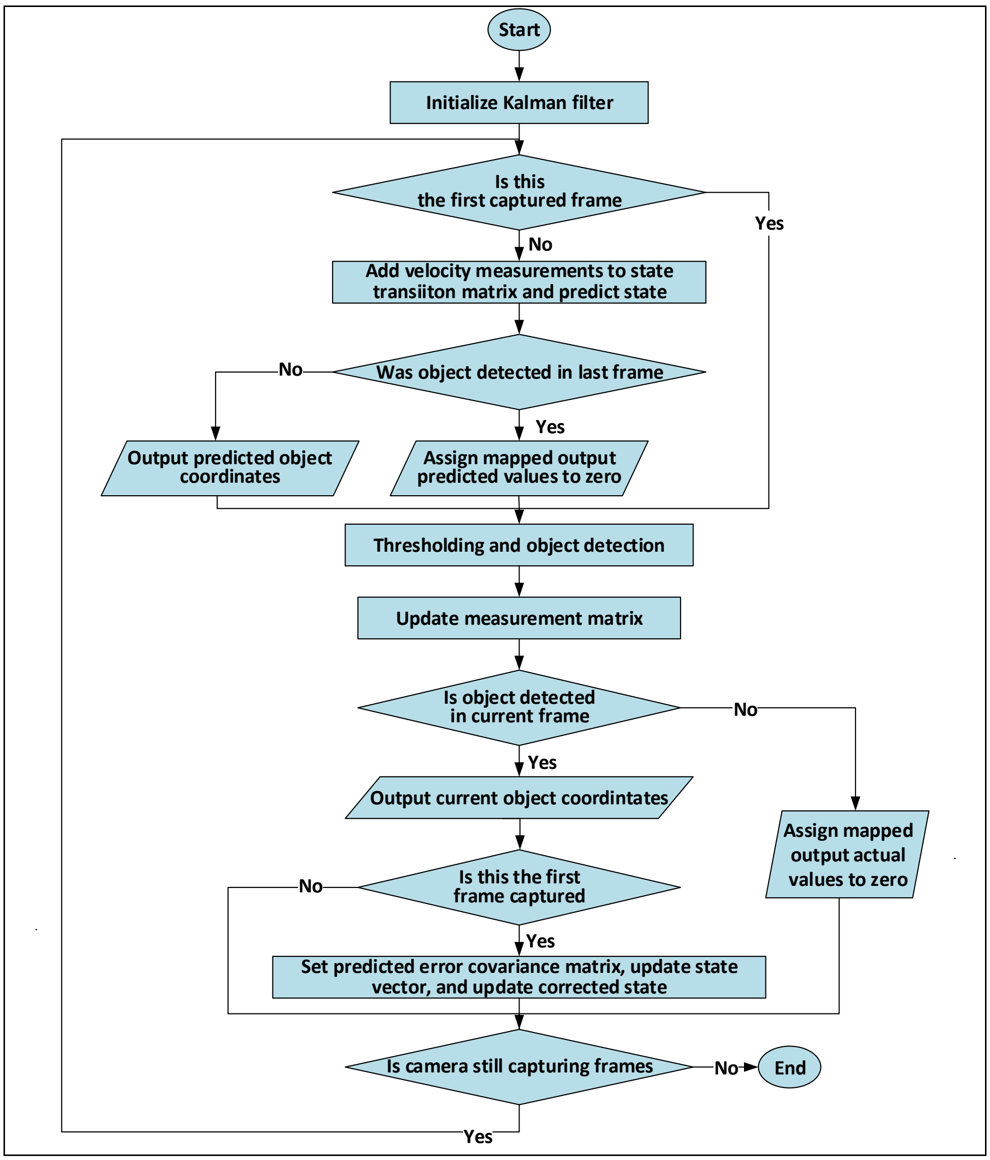


Figure 38: Overview of Kalman filter implementation.

```
    int stateSize = 6; //number of state variables
    int measSize = 4; //number of measurement variables
    int contrSize = 2; //size of control vector
    unsigned int type = CV_32F;

    //instantiate a Kalman filter with cv::KalmanFilter
    KalmanFilter kf(stateSize, measSize, contrSize, type);

    Mat state(stateSize, 1, type); //state vector [x ,y, v_x, v_y, w, h]'
    Mat meas(measSize, 1, type);  //measurement vector [z_x, z_y, z_w, z_h]'

    //transition State Matrix A
    //note: set dT at each processing step!
    // [ 1 0 dT 0  0 0 ]
    // [ 0 1 0  dT 0 0 ]
    // [ 0 0 1  0  0 0 ]
    // [ 0 0 0  1  0 0 ]
    // [ 0 0 0  0  1 0 ]
    // [ 0 0 0  0  0 1 ]
    //first create a 6x6 identity matrix --> dTs added on second measurement
    setIdentity(kf.transitionMatrix);

    //measure Matrix H
    // [ 1 0 0 0 0 0 ]
    // [ 0 1 0 0 0 0 ]
    // [ 0 0 0 0 1 0 ]
    // [ 0 0 0 0 0 1 ]
    kf.measurementMatrix = Mat::zeros(measSize, stateSize, type);
    kf.measurementMatrix.at<float>(0) = 1.0f;
    kf.measurementMatrix.at<float>(7) = 1.0f;
    kf.measurementMatrix.at<float>(16) = 1.0f;
    kf.measurementMatrix.at<float>(23) = 1.0f;

    //control matrix B
    // [ 1 0 ]
    // [ 0 1 ]
    // [ 0 0 ]
    // [ 0 0 ]
    // [ 0 0 ]
    // [ 0 0 ]
    setIdentity(kf.controlMatrix);
    kf.controlMatrix.at<float>(0) = 1.0f;
    kf.controlMatrix.at<float>(3) = 1.0f;
    //control inputs are the motions in the x and y directions
```

Figure 39: Kalman filter setup (1 of 2).

```
//Process Noise Covariance Matrix Q
//these values are found by trial and error
// [ Ex   0   0     0     0    0  ]
// [ 0    Ey  0     0     0    0  ]
// [ 0    0   Ev_x  0     0    0  ]
// [ 0    0   0     Ev_y  0    0  ]
// [ 0    0   0     0     Ew   0  ]
// [ 0    0   0     0     0    Eh ]
kf.processNoiseCov.at<float>(0) = 1e-3;
kf.processNoiseCov.at<float>(7) = 1e-3;
kf.processNoiseCov.at<float>(14) = 1e-1;
kf.processNoiseCov.at<float>(21) = 1e-1;
kf.processNoiseCov.at<float>(28) = 1e-2;
kf.processNoiseCov.at<float>(35) = 1e-2;

//measures noise covariance matrix R
setIdentity(kf.measurementNoiseCov, Scalar(1e-1));
```

Figure 40: Kalman filter setup (2 of 2).

The implemented Kalman predict and update operations are highlighted in Figure 39. Note that the servo controller is integral to predict and update portions of the code. We wish to output the actual position if available, else the predicted coordinates are sent to the servo controller in the FPGA fabric. The controller will assign the memory mapped variables accordingly.

6.6.5 Distance Measurement

The first step in distance measurement is to set a distance between the camera and the object where the first frame is captured. This is the reference distance, and on each execution of the C++ script, the object must be placed at the same distance away from the camera. The object's width must also be known beforehand. Both measurements must be in the same unit of distance; the unit of meters was chosen. As discussed in Section 5.6.1, immediately after the first frame is captured read, the initialize function is called. Called only once, this function takes the first frame as the input and returns the focal length. Figure 14 illustrates the key operations in the initialize function.

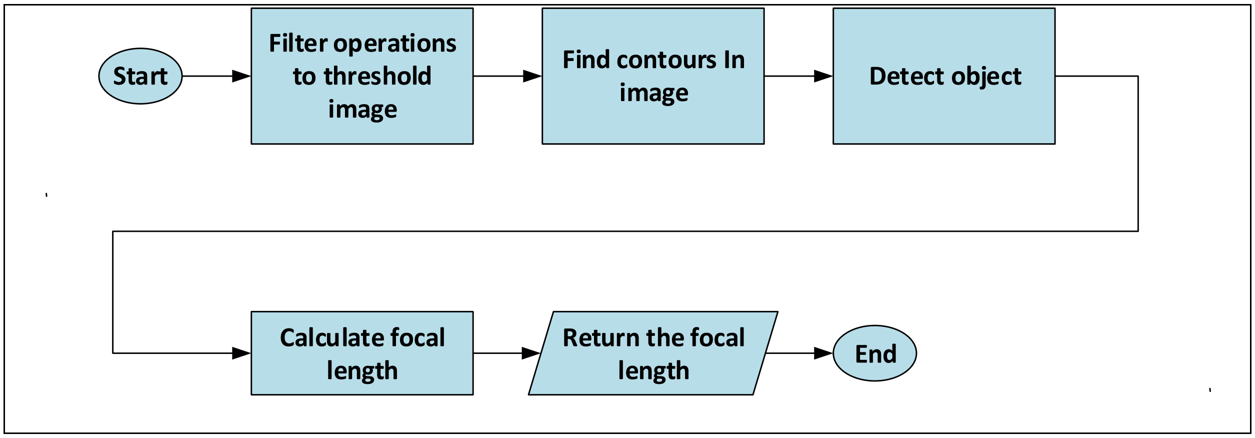


Figure 41: Overview of the initialize function implemented in the C++ script. This function is called after first frame is captured and read.

Once the focal length is found, the distance is calculated on every subsequent frame immediately after the object is detected in the image, as shown in Figure 16. The pixel width of the object is found by placing a box around the object's contour, using the cv::boundingRect function. The bounding rectangle enclosing the ball is also displayed in Figure 16.

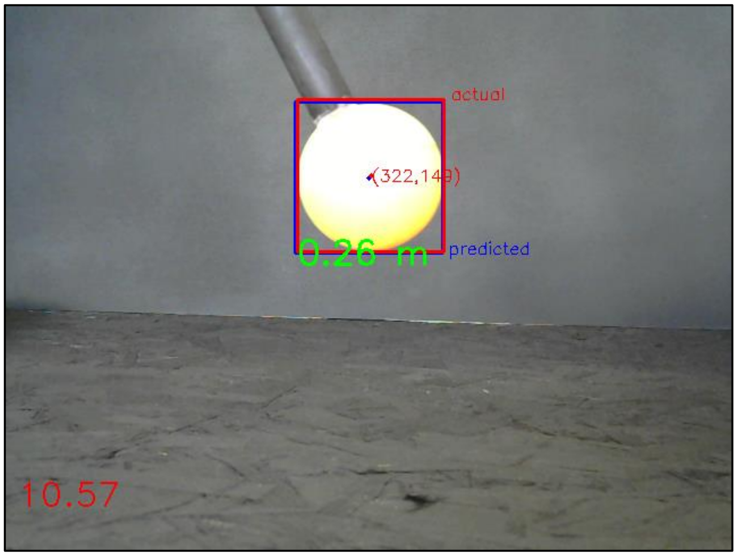


Figure 42: Video displayed with distance measurement enabled. The number displayed on the bottom left corner is the frames-per-second count.

The distance measurement accuracy was tested by placing a measuring tape below the camera and comparing the actual distance of the object to the distance calculated by the program. Three tests were done, and the measurements were recorded in Table 3. An average measurement error of about 3.6% was determined from the results of Table 4. There are several factors that affect the accuracy of distance measurement. First, the initialization of the system to calculate the focal length is extremely important. An attempt must be made to place the ball as close as possible to the set initial distance. Second, an increase in the capture resolution will increase the measurement accuracy. At a resolution of 320x240 pixels, the measurement precision is roughly 1cm.

Table 2: Average distance measured

distance (cm)	20	30	40	50	60
test #1	18.8	29.5	39.1	48.9	57.9
test #2	19.6	29	38.1	48.9	60.2
test #3	18.4	28.8	37.3	47.8	58.8
average distance(cm)	18.9	29.1	38.2	48.5	59.0

Table 3: Average distance measurement error

distance (cm)	20	30	40	50	60
test #1	1.2	0.5	0.9	1.1	2.1
test #2	0.4	1	1.9	1.1	0.2
test #3	1.6	1.2	2.7	2.2	1.2
Average error (cm)	1.1	0.9	1.8	1.5	1.2
% error	5.3	3.0	4.6	2.9	1.9

6.7 Servo Control

Quartus Prime was used to develop Verilog code for the control of the servo motors. There are 4-modules that are used to send PWM signals to the servo motors.

The top-level module is used to connect the FPGA’s inputs, outputs, and x and y-coordinates from the HPS to lower-level Verilog modules. The FPGA’s 50MHz clock is connected to the input of PLL to generate a reduced clock of 250 KHz to be used by the lower-level modules. The x and y-coordinates are connected to the PWM controller module to change the duty cycle of the outgoing PWM signal. The outgoing PWM signal is connected to the FPGA’s GPIO. Refer to Appendix B to view the top-level module.

The counter module is used to create the 16ms period. A 12-bit counter is incremented on every rising clock edge. The value of the counter is used as a reference to the PWM period in subsequent modules. Refer to Appendix E to view the counter module.

The PWM controller modules are used to determine the duty cycle each motor should receive based on the current position of the object being detected. The camera used has a resolution of 320x240 with a center located at (160,120). Therefore, the coordinates of the object (xdir and ydir) will take on values from 0 to 320 (xdir) and 0 to 240 (ydir). The servo motor will stop rotating when either of the limits are reached or when the object is in the center of the camera's view. To reduce the motors from oscillating (left/right or up/down) the center range has been increased by 10 pixels in every direction. The following table and figure illustrates how the x and y-coordinates change the duty cycle which changes the motor's direction. Refer to Appendix C and Appendix D to view the PWM Control modules.

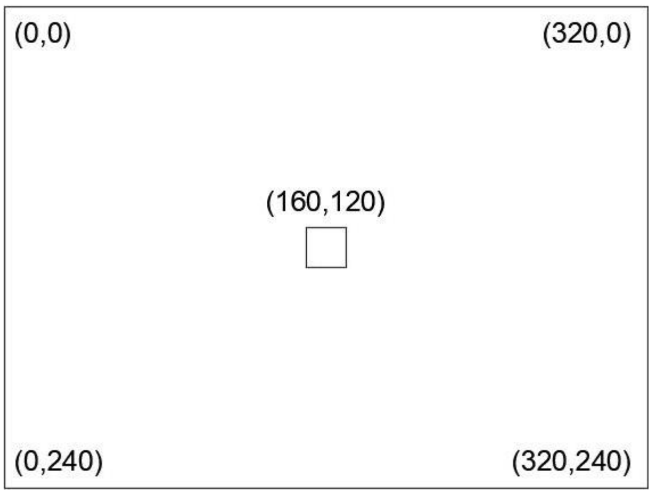


Figure 43: X and Y coordinates

Table 4: Object location and corresponding motor direction

Position of Object in X-direction (pixels)	Duty Cycle	Motor Direction
170 < xdir ≤ 320	Decreases	Rotates Rightward
150 ≤ xdir ≤ 170	Does Not Change	Keeps Same Position
0 ≤ xdir < 150	Increases	Rotates Leftward
Position of Object in Y-direction (pixels)	**Duty Cycle**	**Motor Direction**
130 < ydir ≤ 240	Increases	Rotates Upward
110 < ydir ≤ 130	Does not change	Keeps Same Position
0 < ydir ≤ 110	Decreases	Rotates Downward

The rate at which the duty cycle can change is dependent on the distance the object is away from the center. When the object is on the outer edge of view, the update rate is 16ms, the same as the PWM period. This ensures faster movement to get the object closer to the center position. When the object is very close to the center, the update rate is 262ms, one quarter of the frame rate. This ensures smooth operation when the object is close to center, eliminating oscillations that would occur if the motor was moving too fast. The following table illustrates how this is accomplished.

Table 5: PWM codeword refresh rate

Position of Object in X-direction (pixels)	Update Delay (ms)
250 ≤ xdir ≤ 320	16
200 ≤ xdir < 250	32
170 < xdir < 200	262
150 ≤ xdir ≤ 170	0
120 ≤ xdir < 150	262
70 ≤ xdir < 120	32
0 ≤ xdir < 70	16
Position of Object in Y-direction (pixels)	**Update Delay (ms)**
180 < ydir ≤ 240	16
130 < ydir ≤ 180	262
110 ≤ ydir ≤ 130	0
50 ≤ ydir < 110	262
0 ≤ ydir < 50	16

The PWM_CW refers to PWM codeword, which is the duty cycle. Limits were made to keep the camera view inside the enclosure. The range of motion in the x-direction is roughly 50-degrees. The range of motion in the y-direction is roughly 40-degrees. The following table illustrates the relationship between the codeword and the angle.

Table 6: Range of motion

PWM Codeword for X-direction		
Limit	Codeword	Degrees
Right	265	25.82
Centered	320	0
Left	370	-23.46
PWM Codeword for Y-direction		
Limit	Codeword	Degrees
Lower	235	-10.09
Centered	256	0
Upper	320	30.05

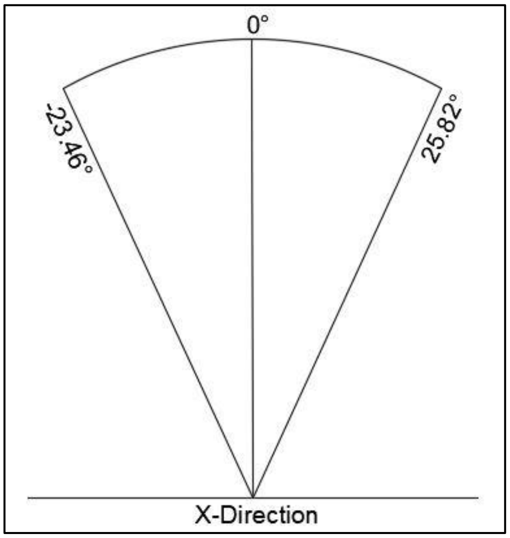


Figure 44: Range of motion in X direction

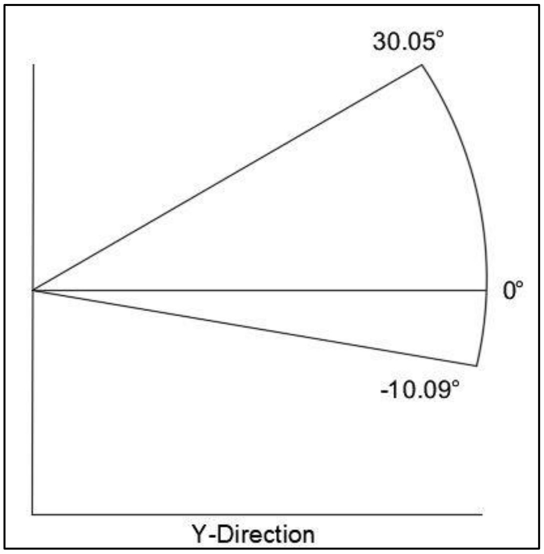


Figure 45: Range of motion in Y direction

The PWM_CW is then compared with the counter value that produces the period. When the duty cycle is greater than the counter value, the output is high. When the counter value has surpassed the value of the duty cycle, the output is low.

The servo motors require a 5V PWM signal and the FPGA GPIO outputs 3.3V, therefore, a logic shift is needed. The logic shifter uses two NPN transistors as switches to shift a voltage to either a higher or lower voltage, depending on what is designed. In our case, the logic shifter shifts a lower voltage to a higher voltage. The base of the Q1 is feed the PWM signal from the GPIO and the servo signal comes from the collector on Q2. If logic high is applied to the base of Q1, Q1 is a closed switch, tying the base of Q2 to ground, causing the servo motor to get a 5V signal. If logic low applied to the base of Q2, the opposite will occur.

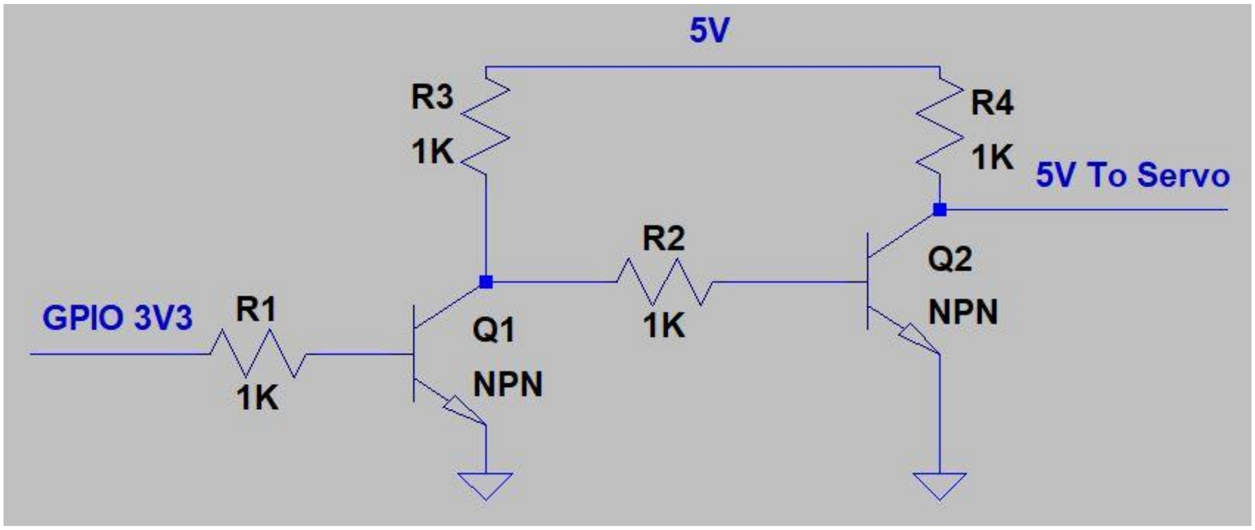


Figure 46: Logic level shifter circuit

Simulations and practical testing were completed to prove functionality of the PWM signals. The tests were done by setting the PWM_CW to 128 (0.5ms), 256 (1ms) and 511 (2ms). The following tables and figures illustrate the results.

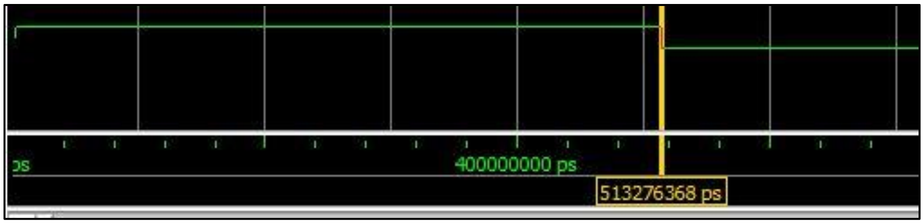


Figure 47: Model-Sim Capture of a 0.5ms Pulse

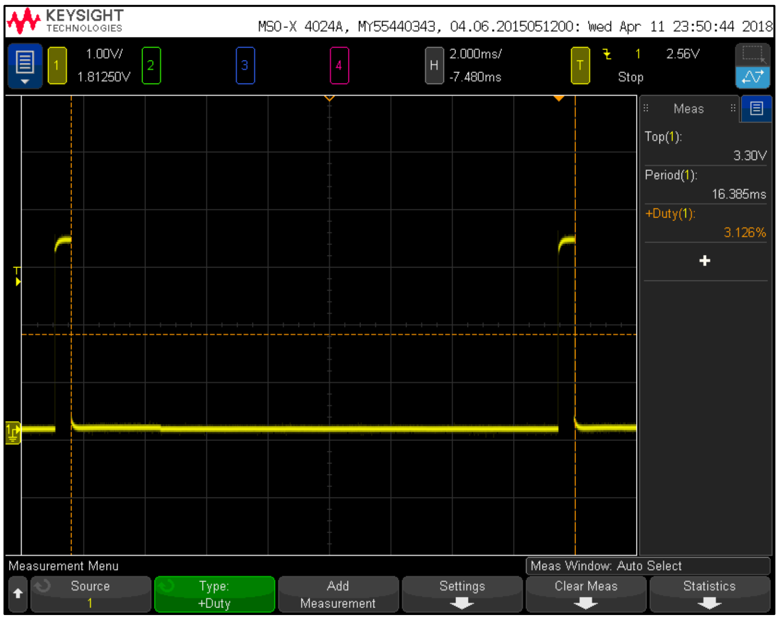


Figure 48: 0.5ms Pulse Measured at GPIO

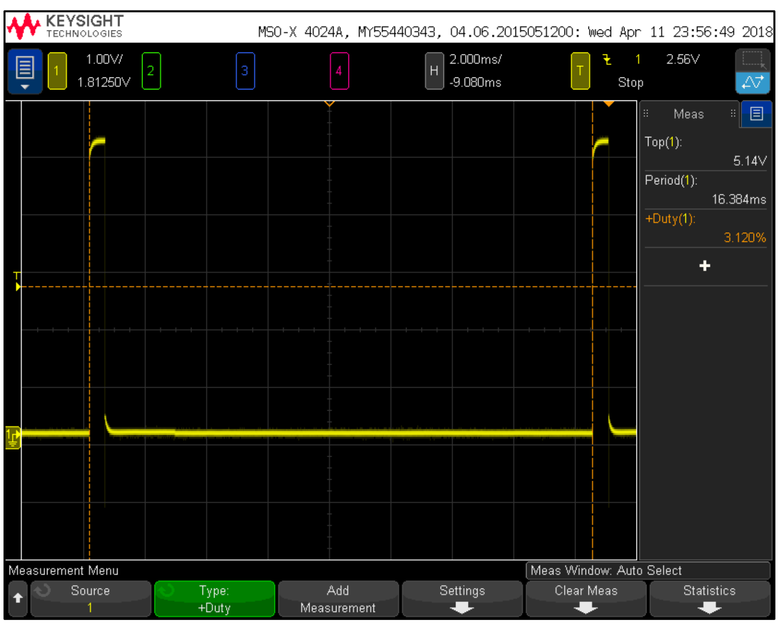


Figure 49: 0.5ms Pulse Measured after logic shifter

Figure 50: Model-Sim Capture of 1ms Pulse

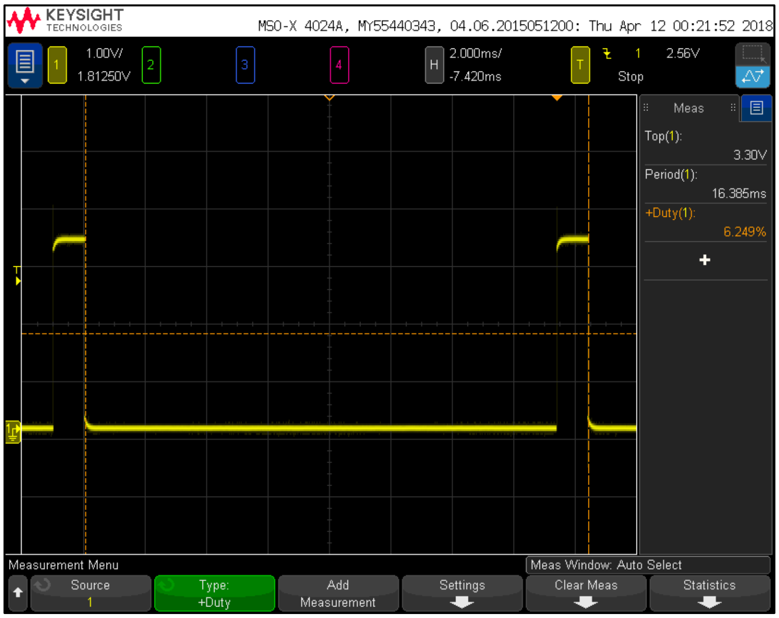


Figure 51: 1ms Pulse Measured at GPIO

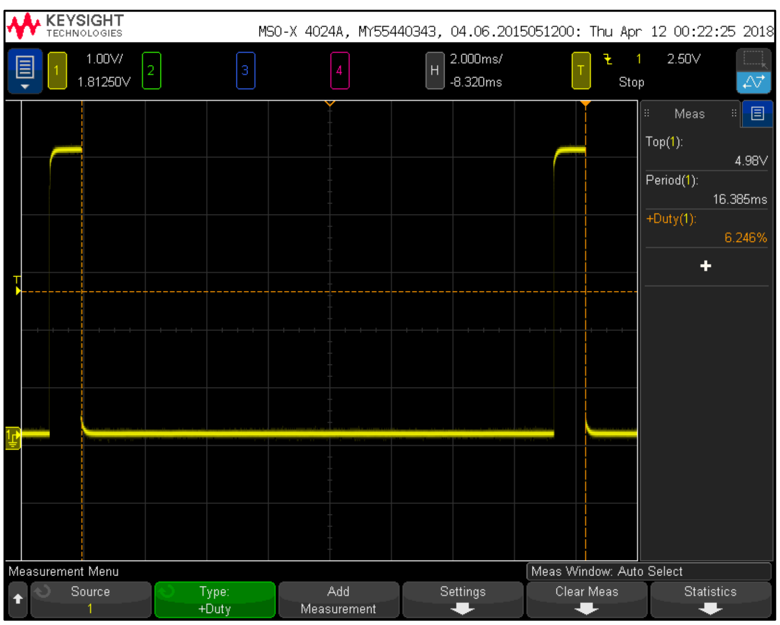


Figure 52: 1ms Pulse Measured After Logic Level Shifter

Figure 53: Model-Sim Capture of a 2ms Pulse

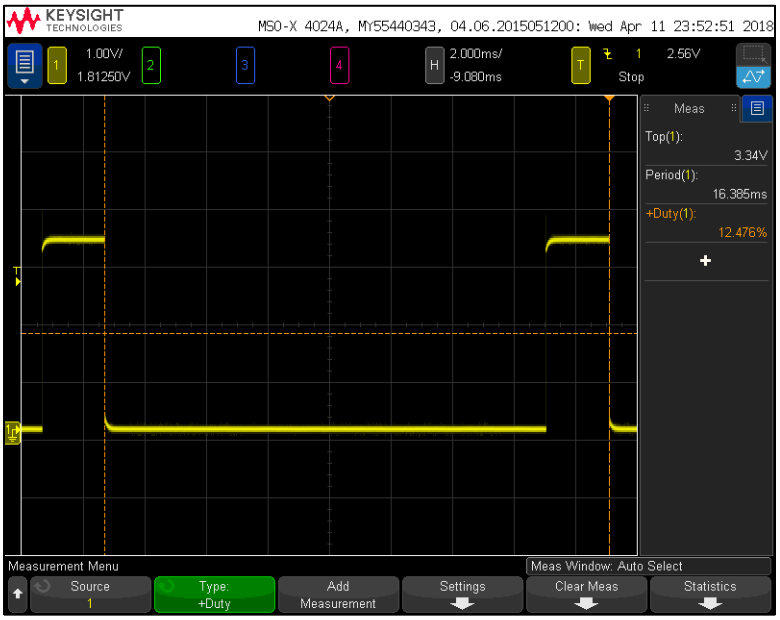


Figure 54: 2ms Pulse Measured at GPIO

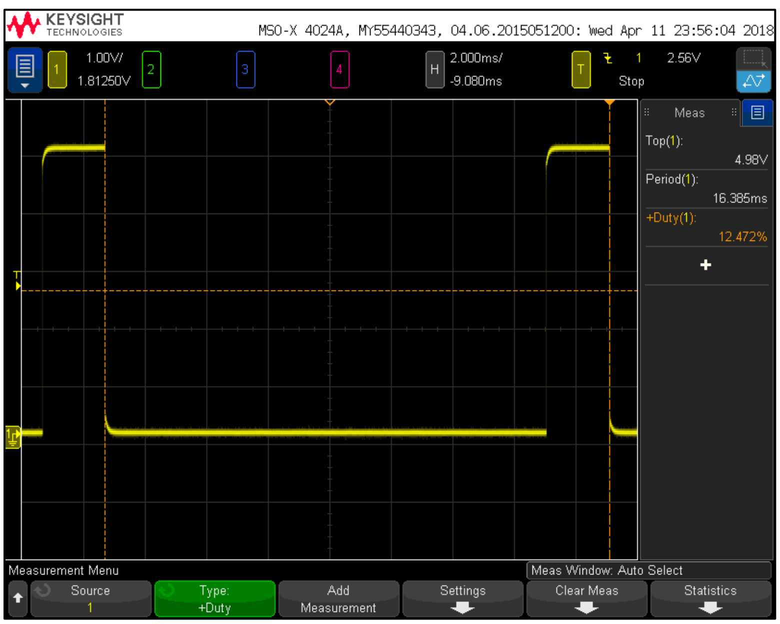


Figure 55: 2ms Pulse Measured After Logic Level Shifter

Table 7: Test results

0.5ms Pulse with PWM_CW =128			
	Duty Cylce (ms)	Period (ms)	Voltage Level (V)
Model-Sim	0.513	16.385	N/A
GPIO Output	0.512	16.385	3.30
Logic Shifter Output	0.511	16.384	5.14
1ms Pulse with PWM_CW =256			
	Duty Cylce (ms)	Period (ms)	Voltage Level (V)
Model-Sim	1.020	16.385	N/A
GPIO Output	1.023	16.385	3.30
Logic Shifter Output	1.023	16.385	4.98
2ms Pulse with PWM_CW =511			
	Duty Cylce (ms)	Period (ms)	Voltage Level (V)
Model-Sim	2.041	16.385	N/A
GPIO Output	2.044	16.385	3.34
Logic Shifter Output	2.043	16.385	4.98

Comparing the simulated results with the practical results, the duty cycle deviates at most by 0.35%, and the period deviates at most by 0.006%. The logic level shifter deviates from the desired 5V by 2.72% at most. These deviations are acceptable, therefore, the PWM signals have passed inspection.

6.8 Putting It All Together

6.8.1 Create the Testbench

It became apparent in the early stages of testing the object tracking code, that a proper testing setup was required to achieve the best tracking performance. A good setup will have a background of even color and will create the largest contrast possible between the object and the background. In addition, a good setup should eliminate shadows and unwanted reflections which may cause tracking errors. Therefore, a testbench with a black background was constructed, as shown in Figure 45. Finally, the camera is mounted to the testbench using custom 3D printed components. The detail of the design of the components can be seen in Appendix A.

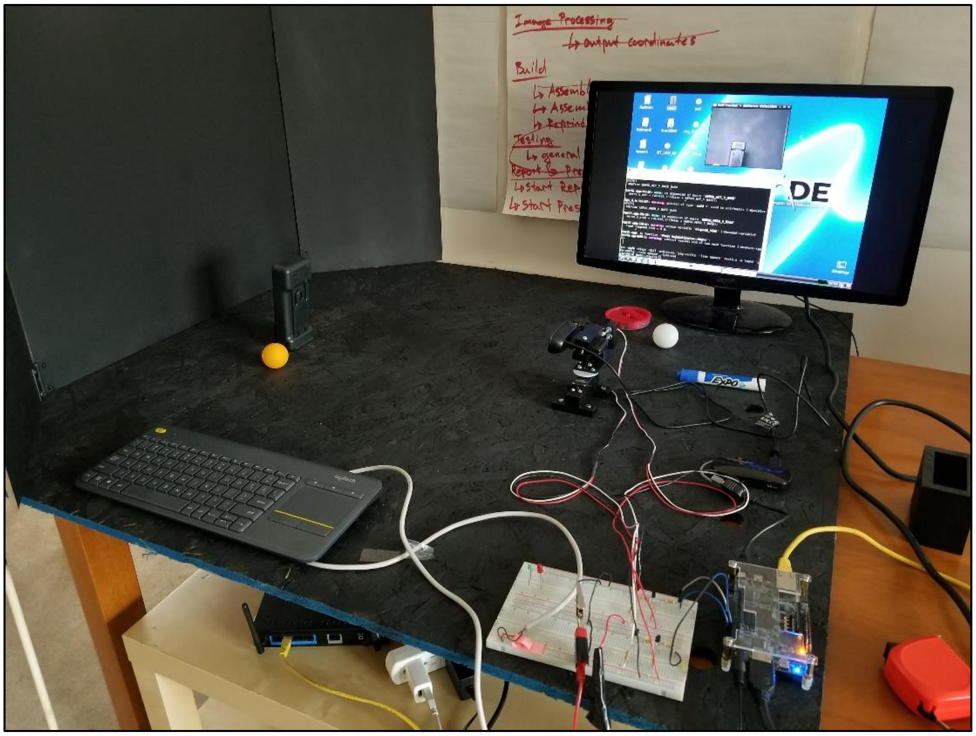


Figure 56: Testbench setup

6.8.2 Servo Motor Setup

Once the servo control system was created, the servo motors were tested to gain a better understanding of the motor movement and operation. Thus, the code discussed in section 3.6.3 was modified to contain the servo PIO. This allowed data to be sent to the Verilog module that controls the motors. Figure 57 below contains the changes.

```
void handler(int signo)
{
 *servo_x_pred=0, *servo_x_act=0;
 *servo_y_act=0, *servo_y_pred=0;
 munmap(base, REG_SPAN);
 close(fd);
 exit(0);
}

int main()
{
 fd = open("/dev/mem", O_RDWR|O_SYNC);
 if(fd<0)
 {
     printf("Can't open memory .\n");
     return -1;
 }
 base = mmap(NULL, REG_SPAN, PROT_READ|PROT_WRITE, MAP_SHARED, fd, REG_BASE);
 if(base == MAP_FAILED)
 {
     printf("Can't map memory. \n");
     close(fd);
     return -1;
 }
 servo_x_pred=(uint32_t*)(base+SERVO_PRED_X_BASE);
 servo_x_act=(uint32_t*)(base+SERVO_ACT_X_BASE);
 servo_y_pred=(uint32_t*)(base+SERVO_PRED_Y_BASE);
 servo_y_act=(uint32_t*)(base+SERVO_ACT_Y_BASE);

 signal(SIGINT, handler);
 *servo_x_pred=0x0, *servo_x_act=0x1;
 *servo_y_pred=0x0, *servo_y_act=0x1;

 while(1)
 {
     *servo_x_act = 258;
     *servo_y_act = 239;
 }
 //Unnmap
 munmap(base, REG_SPAN);
 close(fd);
 return(0);
}
```

Figure 57: Servo test code.

Lines 36 and 37 in the code above contain the data that is sent which sets the angle of each motor. The results of this test were pivotal in allowing a better understanding of the motors, also this code would further develop into the code used to calibrate the motors with the camera. Calibration of the motors with the camera occurred once the entire system was built. To help facilitate the calibration process, the code above was converted into C++. Furthermore, the argc and argv parameters are added see Figure 58. This addition allowed for the adjustment of the motors without having to recompile the code. Also, the conversion to C++ would make it easier to insert the code, used for reading and writing to memory, into the image processing C++ code. Furthermore, this calibration process produced the limits for the servo motors. The table below outlines the limits.

Table 8: Servo motor limits

PWM Codeword for X-direction		
Limit	Codeword	Degrees
Right	265	25.82
Centered	320	0
Left	370	-23.46
PWM Codeword for Y-direction		
Limit	Codeword	Degrees
Lower	235	-10.09
Centered	256	0
Upper	320	30.05

```
void handler(int signo)
{
 *servo_y_pred=0, *servo_y_act=0;
 *servo_x_pred=0, *servo_x_act=0;
 munmap(base, REG_SPAN);
 close(fd);
 exit(0);
}

int main(int argc, char *argv[])
{
 fd = open("/dev/mem", O_RDWR|O_SYNC);
 if(fd<0)
 {
     printf("Can't open memory .\n");
     return -1;
 }
 base=mmap(NULL, REG_SPAN, PROT_READ|PROT_WRITE, MAP_SHARED, fd, REG_BASE);
 if(base==MAP_FAILED)
 {
     printf("Can't map memory. \n");
     close(fd);
     return -1;
 }
 servo_x_pred=(uint32_t*)(base+SERVO_PRED_X_BASE);
 servo_x_act=(uint32_t*)(base+SERVO_ACT_X_BASE);
 servo_y_pred=(uint32_t*)(base+SERVO_PRED_Y_BASE);
 servo_y_act=(uint32_t*)(base+SERVO_ACT_Y_BASE);
 signal(SIGINT, handler);
 *servo_y_pred=0x0, *servo_y_act=0x1;
 *servo_x_pred=0x0, *servo_x_act=0x1;

 while(1)
 {
     *servo_x_act = atoi(argv[1]);
     *servo_y_act = atoi(argv[2]);
     printf("X = %d", *servo_x_act);
     printf("Y = %d", *servo_y_act);
 }
 //Unnmap
 munmap(base, REG_SPAN);
 close(fd);
 return(0);
}
```

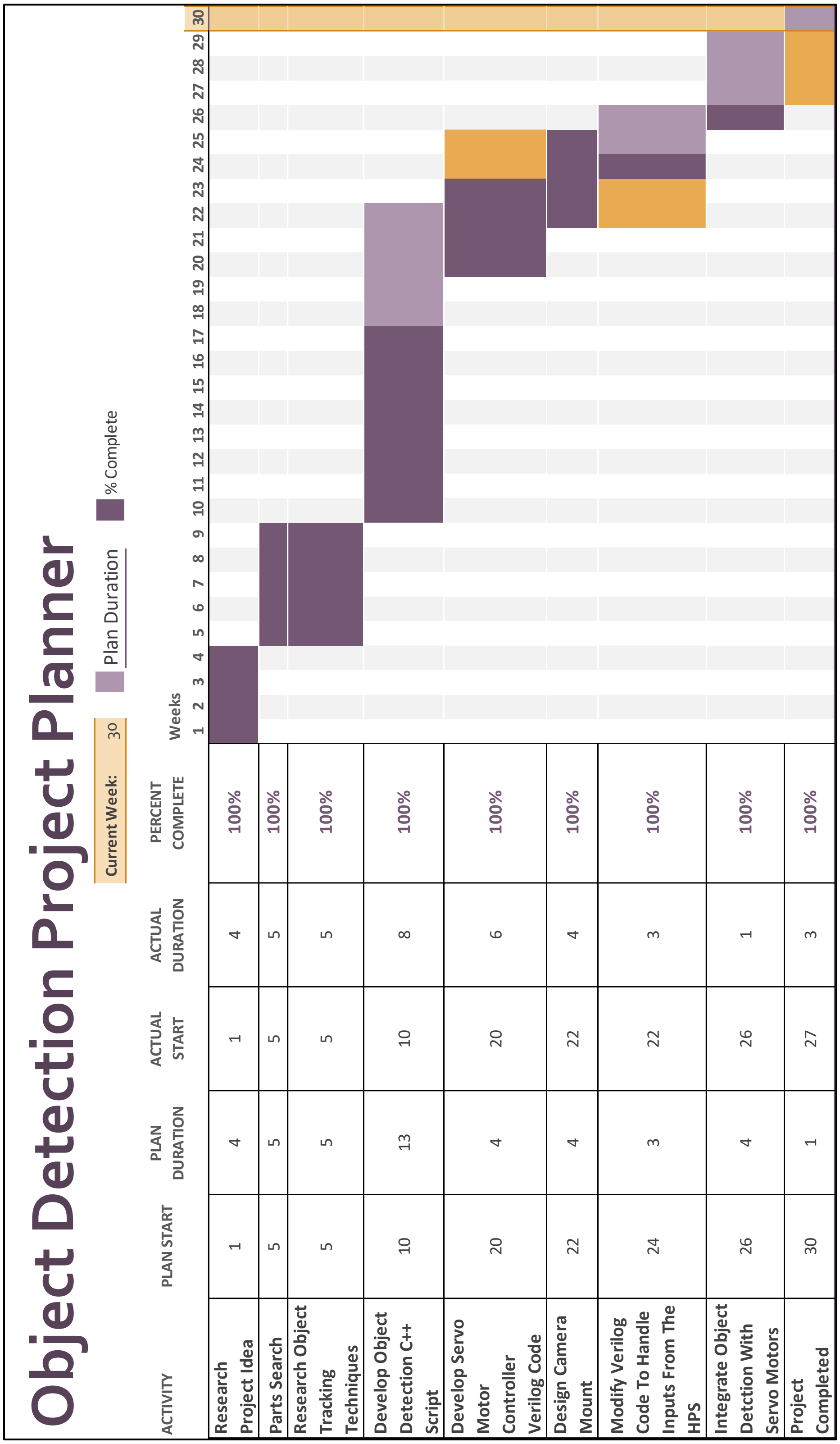
Figure 58: Servo calibration C++ code.

6.8.3 Running the Tracking Script

Upon the integration of the object tracking code to the FPGA fabric, the system behaved erratically, as the servo motors moved too quickly for the tracking program to keep up. Therefore, a decision was made to alter both the servo controller on the FPGA and the tracking program to improve performance. The object tracking code originally captured images at a resolution of 600x450 pixels, and an average rate of four frames-per-second (fps) was observed on the DE10-nano. In contrast, the same code executed on an ubuntu testbench running on a 2010 Lenovo Thinkpad T510 produced an average of 11fps. As mentioned earlier, the Logitech C310 webcam used in this project can capture at 15 fps at the full resolution of 1280x720 pixels. Running a barebones OpenCV script on the DE10 confirmed the manufacturer’s specification. The cause of this significantly lower fps count must then be the result of the object tracking code. After testing each section of the C++ script, the thresholding stage of the program was determined to be the limiting factor in the performance of the system. Due to the hardware limitations on the DE10-nano, the matrix operations in the thresholding stage take significantly longer to complete, resulting in a bottleneck. To reduce the number of matrix operations, the captured size of the image was shrunken down from 600x450 to 320x240 pixels. The result was an average of 12 fps on the DE10-nano – a speedup of close to three times the original. This significant increase of the tracking script performance led to noticeable changes in the controllability of the servo motors.

In addition to shrinking down the captured image, the range of colors to track was modified. The original range allowed too many unwanted colors through, specifically in the lower values towards pure white. A new, smaller range of colors was selected to better suit the testing conditions.

7.Project Management



ACTIVITY	PLAN START	PLAN DURATION	ACTUAL START	ACTUAL DURATION	PERCENT COMPLETE
Research Project Idea	1	4	1	4	100%
Parts Search	5	5	5	5	100%
Research Object Tracking Techniques	5	5	5	5	100%
Develop Object Detection C++ Script	10	13	10	8	100%
Develop Servo Motor Controller Verilog Code	20	4	20	6	100%
Design Camera Mount	22	4	22	4	100%
Modify Verilog Code To Handle Inputs From The HPS	24	3	22	3	100%
Integrate Object Detction With Servo Motors	26	4	26	1	100%
Project Completed	30	1	27	3	100%

Figure 59: Project Gantt chart

8.Summary

8.1 Costs and marketability

The cost of this project is outlined in Table 9. The components were selected based on its price as to limit the overall cost. A USB web cam was selected because it was significantly cheaper than a camera that connects through the FPGA's GPIO interface. A sizeable portion of the cost is on the testbench alone. The DE10-Nano was chosen because it provides the most performance for its price. If a commercially viable product is desired, additional research and development must be made to address the limitations mentioned in Section 8.3 below. This includes the costs of time needed to improve the design and the cost of new components to compliment the design changes. In addition, manufacturing, marketing and sales costs must be factored into the final cost of bringing this project into market.

Table 9: Project costs

Part	Cost (CAD)
Terasic DE-10 Nano Kit	180
16GB microSD card	22
Logitech Webcam	40
Camera Mount	10
2 x DC Servo Motors	20
Wood panels	8
2 x Black spray paint	16.50
Hardware (screws, nuts, hinges)	22
Total Cost	$318.50 CAD

8.2 Social Impact

Machine vision systems benefit all facets of modern society. Improving worker safety, increasing productivity, and improving quality of life are some of the applications of such a system. Furthermore, vision systems can vastly improve road safety. Motor vehicle collisions account for the greatest number of deaths among 15 to 29 year olds around the world [12]. Systems such as automatic lane change technology in modern vehicles removes human error from the equation, increasing safety for everyone.

8.3 Project Limitations

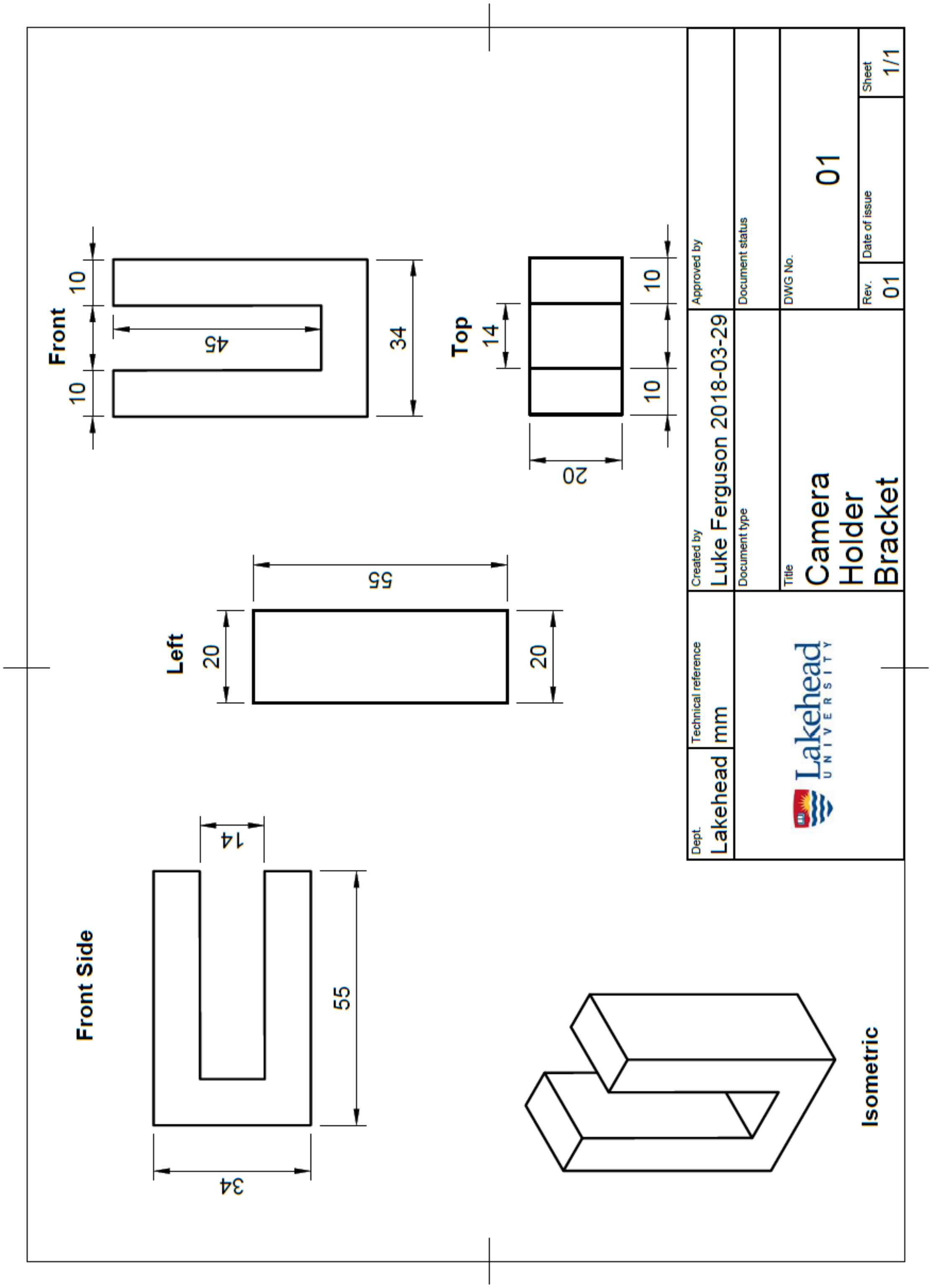
The tracking has been designed to work only within a certain space. This space has been designed to provide a strong contrasting background. The reason for this is to provide an environment with little to no interference, therefore, allowing for clear object detection and tracking. Furthermore, the camera's maximum frame rate is 15 frames per second. Since this is relatively slow compared to the system, a camera with a higher frame rate could potentially improve performance. However, the system design would have to be adjusted to allow for a faster frame rate. Additionally, for the system to operate in an environment containing more interference more advanced image detection techniques would be required.

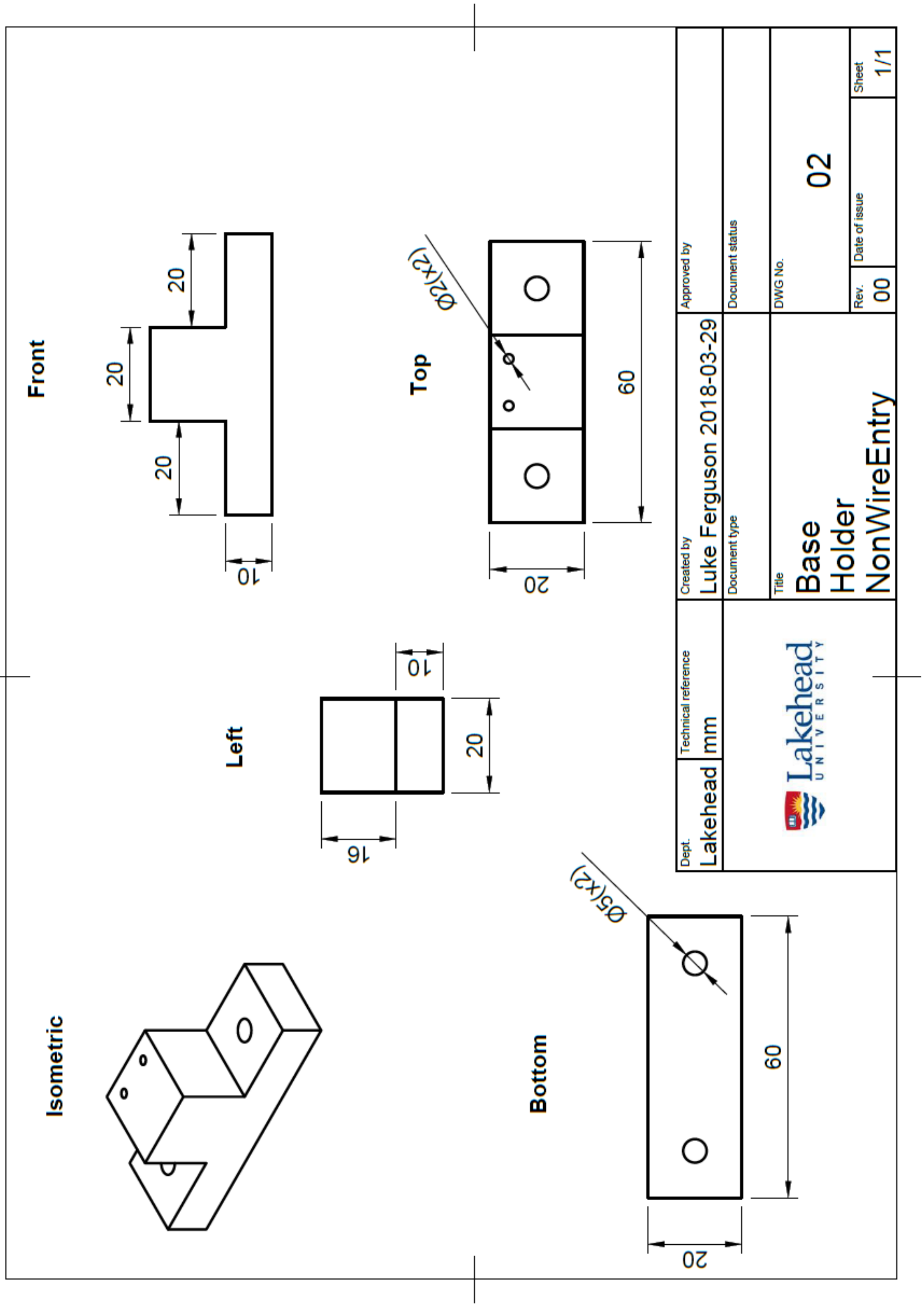
8.4 Lessons Learned

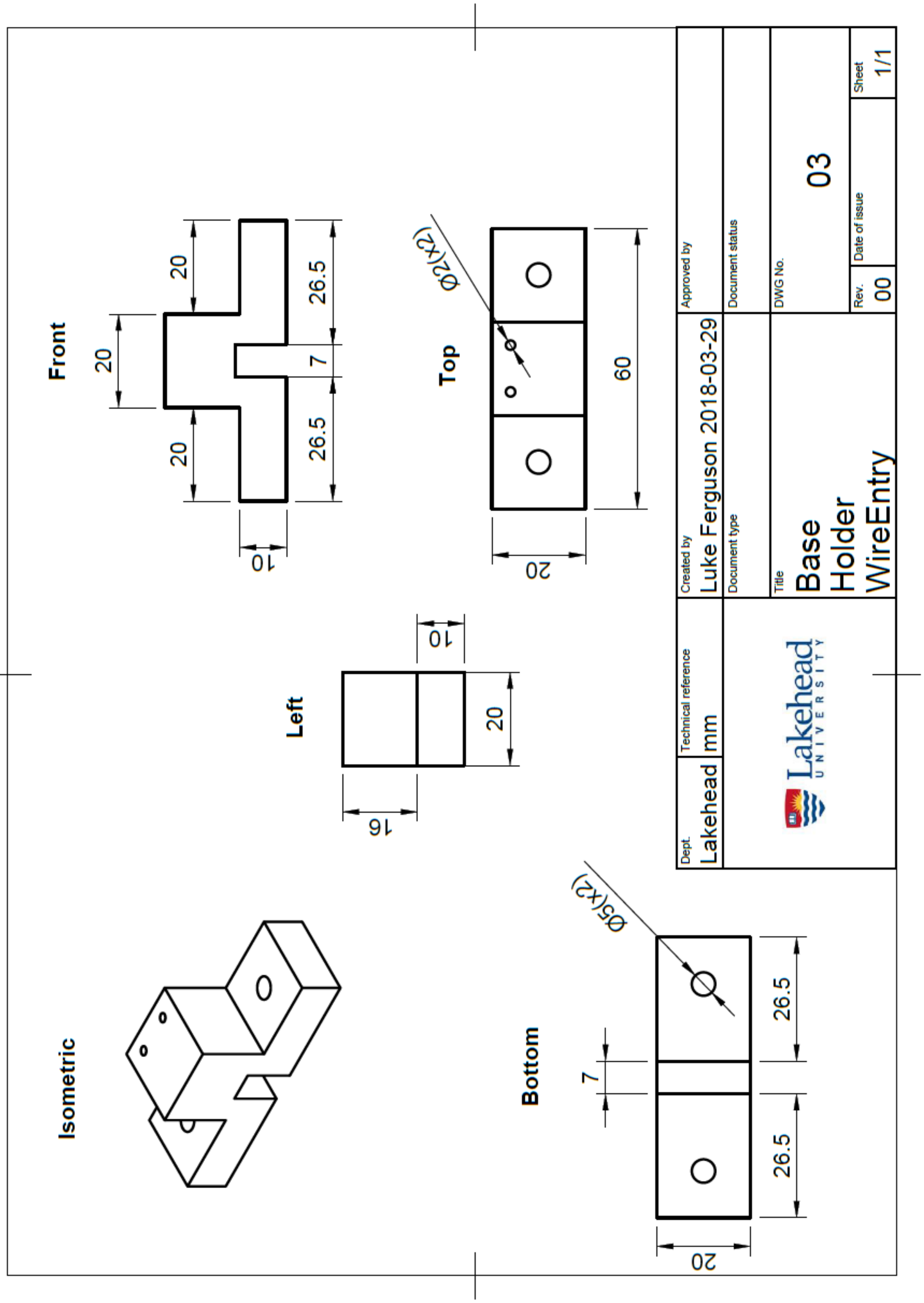
Overall the project was a success. Control of the servo motors was achieved using a computer vision system. Moreover, both the FPGA and the HPS were able to operate simultaneously to achieve control. However, system performance could be improved. The most notable improvement would be the use of OpenCL to attain parallel computing between the HPS and FPGA. OpenCL can result in a system speed up when used to solve matrix calculations. Since image processing contains large matrix operations a system speed up would be anticipated. This speed up would be accomplished by using the FPGA fabric to perform the image processing matrix operations in parallel, thus reducing the amount of resource intensive calculations done by the ARM processor. Alternatively, implementing the image processing entirely in the FPGA fabric would also achieve speed up. In summary, the method of object detection and tracking discussed achieved the intended goal. However, further improvement by increasing the system performance and reducing the project limitations would produce a more realistic method of object detection and tracking.

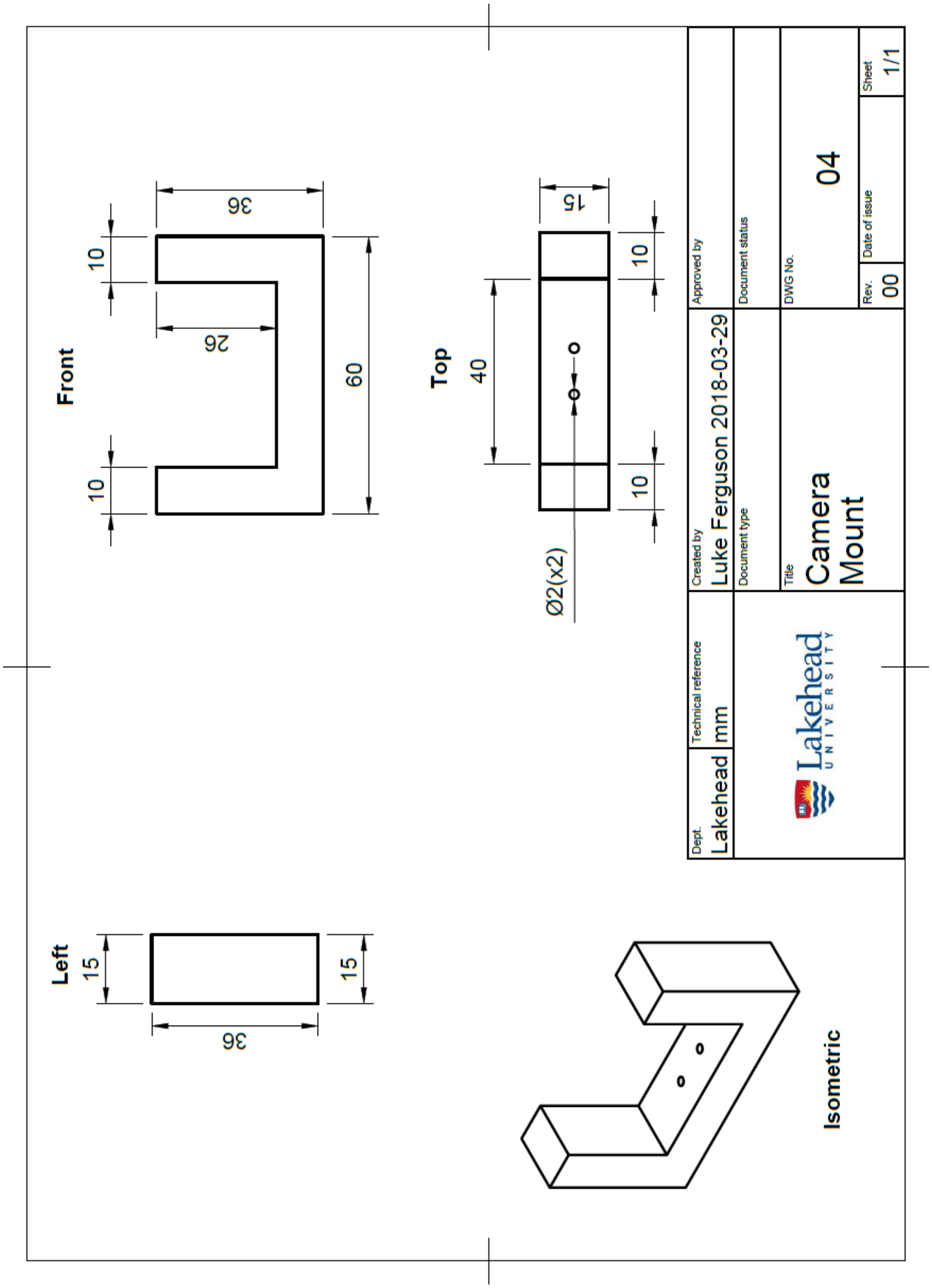
Appendix

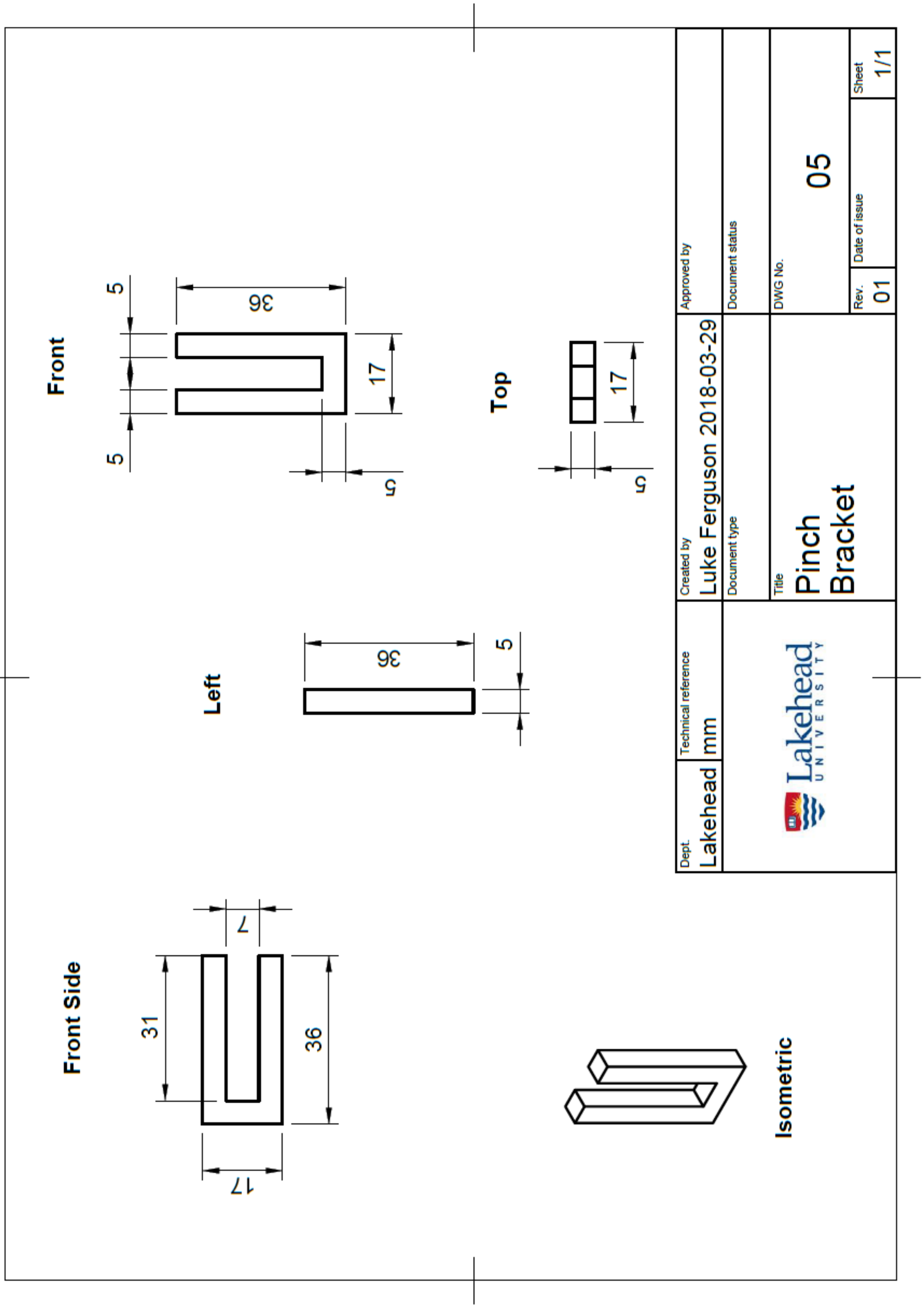
Appendix A: Mechanical Design

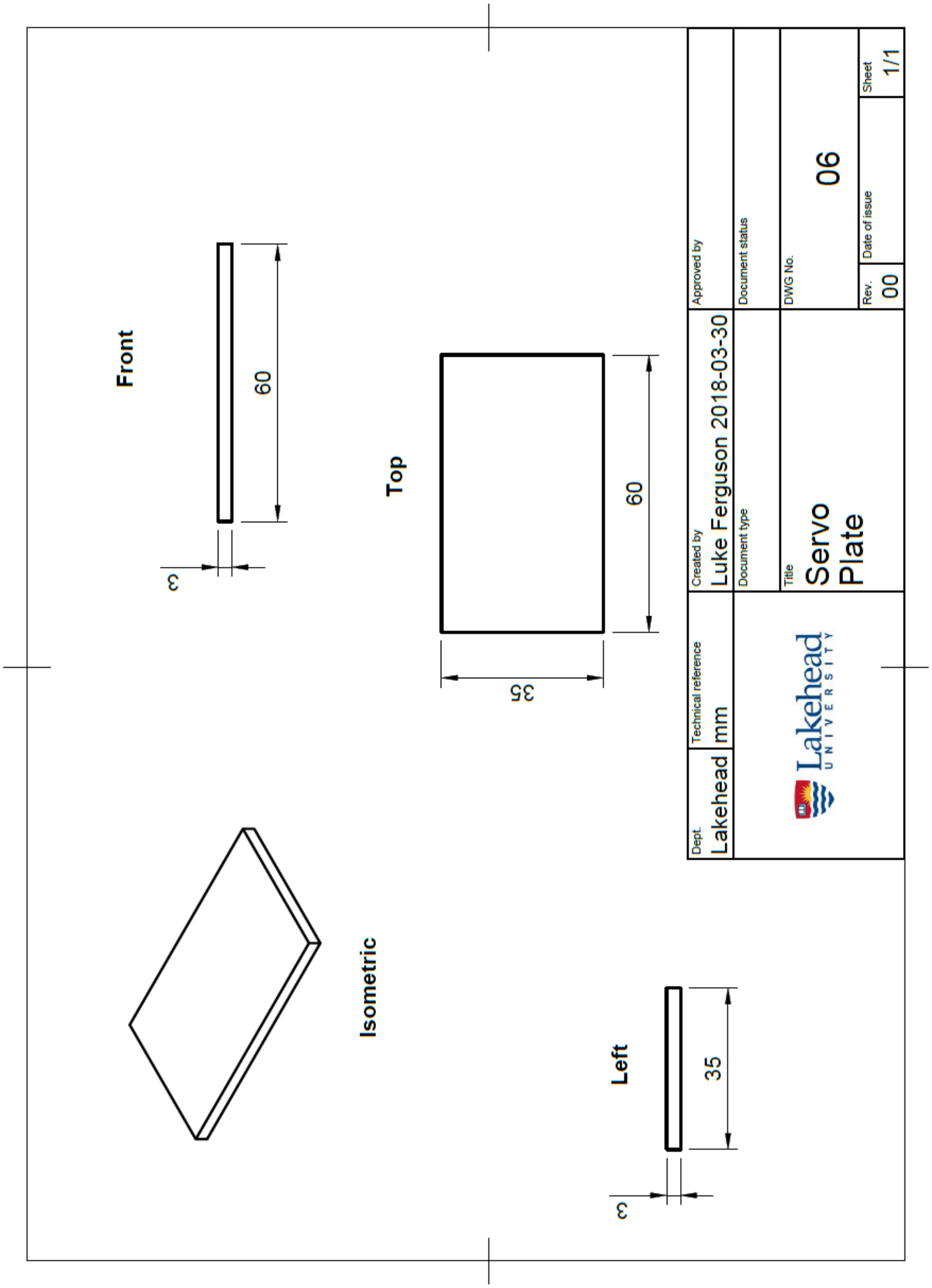



Front
Top
Left
Isometric
Bottom
20
20
20
10
60
20
Ø2(x2)
10
20
16
Ø5(x2)
60
20
Dept.
Lakehead
Technical reference
mm
Created by
Luke Ferguson 2018-03-29
Approved by
Document type
Document status
Title
Base Holder NonWireEntry
DWG No.
02
Rev.
00
Date of issue
Sheet
1/1


Front
20
20
20
26.5
7
26.5
10
Top
Φ2(x2)
60
20
Left
10
20
16
Isometric
Bottom
Φ5(x2)
7
26.5
26.5
20
Dept.
Lakehead
Technical reference
mm
Created by
Luke Ferguson 2018-03-29
Approved by
Document type
Document status
Title
Base Holder WireEntry
DWG No.
03
Rev.
00
Date of issue
Sheet
1/1


Front
10
10
36
26
60
Top
15
10
10
40
Ø2(x2)
Left
15
15
36
Isometric
Dept.
Lakehead
Technical reference
mm
Created by
Luke Ferguson 2018-03-29
Approved by
Document type
Document status
Title
Camera Mount
DWG No.
04
Rev.
00
Date of issue
Sheet
1/1


Front
Top
Left
Front Side
Isometric
36
17
5
5
5
5
36
5
17
7
31
36
17
Dept.
Lakehead
Technical reference
mm
Created by
Luke Ferguson 2018-03-29
Approved by
Document type
Document status
Title
Pinch Bracket
DWG No.
05
Rev.
01
Date of issue
Sheet
1/1


Front
60
3
Top
60
35
Isometric
Left
35
3
Dept.
Lakehead
Technical reference
mm
Created by
Luke Ferguson 2018-03-30
Approved by
Document type
Document status
Title
Servo
Plate
DWG No.
06
Rev.
00
Date of issue
Sheet
1/1
Lakehead UNIVERSITY

Appendix B: The Top-Level Verilog Code

```
/***********************************************************************/
/*      Project: Object Detection for degree project                  */
/*      File: PWM controller for the x direction servo motor          */
/*      Created by: Luke Ferguson                                     */
/*      Group Members: Neal Palmer & Ronald Chan                      */
/*      Semester: Fall & Winter 2017/2018                             */
/***********************************************************************/

//=======================================================
//  Input/Output declarations
//=======================================================

module OBJ_DET(

//This code is generated by Terasic System Builder
    //////////// CLOCK //////////
    input                           FPGA_CLK1_50,

    //////////// HDMI //////////
    inout                           HDMI_I2C_SCL,
    inout                           HDMI_I2C_SDA,
    inout                           HDMI_I2S,
    inout                           HDMI_LRCLK,
    inout                           HDMI_MCLK,
    inout                           HDMI_SCLK,
    output                          HDMI_TX_CLK,
    output                          HDMI_TX_DE,
    output              [23:0]      HDMI_TX_D,
    output                          HDMI_TX_HS,
    input                           HDMI_TX_INT,
    output                          HDMI_TX_VS,

    //////////// HPS //////////
    inout                           HPS_CONV_USB_N,
    output              [14:0]      HPS_DDR3_ADDR,
    output               [2:0]      HPS_DDR3_BA,
    output                          HPS_DDR3_CAS_N,
    output                          HPS_DDR3_CKE,
    output                          HPS_DDR3_CK_N,
    output                          HPS_DDR3_CK_P,
    output                          HPS_DDR3_CS_N,
```

```
	output		[3:0]	HPS_DDR3_DM,
	inout		[31:0]	HPS_DDR3_DQ,
	inout		[3:0]	HPS_DDR3_DQS_N,
	inout		[3:0]	HPS_DDR3_DQS_P,
	output			HPS_DDR3_ODT,
	output			HPS_DDR3_RAS_N,
	output			HPS_DDR3_RESET_N,
	input			HPS_DDR3_RZQ,
	output			HPS_DDR3_WE_N,
	output			HPS_ENET_GTX_CLK,
	inout			HPS_ENET_INT_N,
	output			HPS_ENET_MDC,
	inout			HPS_ENET_MDIO,
	input			HPS_ENET_RX_CLK,
	input		[3:0]	HPS_ENET_RX_DATA,
	input			HPS_ENET_RX_DV,
	output		[3:0]	HPS_ENET_TX_DATA,
	output			HPS_ENET_TX_EN,
	inout			HPS_GSENSOR_INT,
	inout			HPS_I2C0_SCLK,
	inout			HPS_I2C0_SDAT,
	inout			HPS_I2C1_SCLK,
	inout			HPS_I2C1_SDAT,
	inout			HPS_KEY,
	inout			HPS_LED,
	inout			HPS_LTC_GPIO,
	output			HPS_SD_CLK,
	inout			HPS_SD_CMD,
	inout		[3:0]	HPS_SD_DATA,
	output			HPS_SPIM_CLK,
	input			HPS_SPIM_MISO,
	output			HPS_SPIM_MOSI,
	inout			HPS_SPIM_SS,
	input			HPS_UART_RX,
	output			HPS_UART_TX,
	input			HPS_USB_CLKOUT,
	inout		[7:0]	HPS_USB_DATA,
	input			HPS_USB_DIR,
	input			HPS_USB_NXT,
	output			HPS_USB_STP,

	//////////// KEY //////////
	input		[1:0]	KEY,
```

```
    //////////// LED //////////
    output          [7:0]      LED,

    //////////// SW //////////
    input           [3:0]      SW,

    //////////// GPIO_0 //////////
    inout           [35:0]     GPIO_0

);



//=======================================================
//  REG/WIRE declarations
//=======================================================
wire                hps_fpga_reset_n;
wire     [1: 0]     fpga_debounced_buttons;
wire     [6: 0]     fpga_led_internal;
wire     [2: 0]     hps_reset_req;
wire                hps_cold_reset;
wire                hps_warm_reset;
wire                hps_debug_reset;
wire     [27: 0]    stm_hw_events;
wire                fpga_clk_50;
wire                clk_65;
wire                clk_130;
wire     [31: 0]    xdir;
wire     [31: 0]    ydir;
wire     [31: 0]    nothing_x;
wire     [31: 0]    nothing_y;
wire     [31: 0]    PWM_out_x;
wire     [31: 0]    PWM_out_y;
reg      [31: 0]    control_x_tmp;
reg      [31: 0]    control_y_tmp;
reg      [31: 0]    do_nothing_x;
reg      [31: 0]    do_nothing_y;
reg      [31: 0]    servo_pred_x_connection;
reg      [31: 0]    servo_act_x_connection;
reg      [31: 0]    servo_pred_y_connection;
reg      [31: 0]    servo_act_y_connection;
reg      [31: 0]    speed_fast_inc_reg;
reg      [31: 0]    speed_med_inc_reg;
reg      [31: 0]    speed_slow_inc_reg;
wire                clk;
```

```

//=======================================================
//  Connection of internal logics
//=======================================================

assign LED[7: 1] = fpga_led_internal;
assign fpga_clk_50 = FPGA_CLK1_50;
assign GPIO_0[0] = PWM_out_x;
assign GPIO_0[1] = PWM_out_y;

// Loop to determine which control signal to choose
always @ (servo_pred_x_connection
            or servo_act_x_connection
            or servo_pred_y_connection
            or servo_act_y_connection)
begin
    if((servo_act_x_connection==32'b0)
        && (servo_pred_x_connection != 32'b0)
        && (servo_pred_y_connection != 32'b0)
        && (servo_act_y_connection==32'b0))
        begin
        control_x_tmp <= servo_pred_x_connection;
        control_y_tmp <= servo_pred_y_connection;
        do_nothing_x <= 32'b0;
        do_nothing_y <= 32'b0;
        end
    else if((servo_pred_x_connection == 32'b0)
        && (servo_act_x_connection != 32'b0)
        && (servo_act_y_connection != 32'b0)
        && (servo_pred_y_connection == 32'b0))
        begin
        control_x_tmp <= servo_act_x_connection;
        control_y_tmp <= servo_act_y_connection;
        do_nothing_x <= 32'b0;
        do_nothing_y <= 32'b0;
        end
    else if((servo_pred_x_connection != 32'b0)
        && (servo_act_x_connection != 32'b0)
        && (servo_act_y_connection != 32'b0)
        && (servo_pred_y_connection != 32'b0))
        begin
        do_nothing_x <= 32'b1;
        do_nothing_y <= 32'b1;
        end
end
```

```
assign xdir = control_x_tmp;
assign ydir = control_y_tmp;
assign nothing_x = do_nothing_x;
assign nothing_y = do_nothing_y;

//=======================================================
//  Structural coding of HPS
//=======================================================

    soc_system u0 (
          //Clock & Reset
        .clk_clk(FPGA_CLK1_50),
        .reset_reset_n(hps_fpga_reset_n),
          //HPS ddr3
        .memory_mem_a(HPS_DDR3_ADDR),
        .memory_mem_ba(HPS_DDR3_BA),
        .memory_mem_ck(HPS_DDR3_CK_P),
        .memory_mem_ck_n(HPS_DDR3_CK_N),
        .memory_mem_cke(HPS_DDR3_CKE),
        .memory_mem_cs_n(HPS_DDR3_CS_N),
        .memory_mem_ras_n(HPS_DDR3_RAS_N),
        .memory_mem_cas_n(HPS_DDR3_CAS_N),
        .memory_mem_we_n(HPS_DDR3_WE_N),
        .memory_mem_reset_n(HPS_DDR3_RESET_N),
        .memory_mem_dq(HPS_DDR3_DQ),
        .memory_mem_dqs(HPS_DDR3_DQS_P),
        .memory_mem_dqs_n(HPS_DDR3_DQS_N),
        .memory_mem_odt(HPS_DDR3_ODT),
        .memory_mem_dm(HPS_DDR3_DM),
        .memory_oct_rzqin(HPS_DDR3_RZQ),
          //HPS Ethernet
        .hps_0_hps_io_hps_io_emac1_inst_TX_CLK(HPS_ENET_GTX_CLK),
        .hps_0_hps_io_hps_io_emac1_inst_TXD0(HPS_ENET_TX_DATA[0]),
        .hps_0_hps_io_hps_io_emac1_inst_TXD1(HPS_ENET_TX_DATA[1]),
        .hps_0_hps_io_hps_io_emac1_inst_TXD2(HPS_ENET_TX_DATA[2]),
        .hps_0_hps_io_hps_io_emac1_inst_TXD3(HPS_ENET_TX_DATA[3]),
        .hps_0_hps_io_hps_io_emac1_inst_RXD0(HPS_ENET_RX_DATA[0]),
        .hps_0_hps_io_hps_io_emac1_inst_MDIO(HPS_ENET_MDIO),
        .hps_0_hps_io_hps_io_emac1_inst_MDC(HPS_ENET_MDC),
        .hps_0_hps_io_hps_io_emac1_inst_RX_CTL(HPS_ENET_RX_DV),
        .hps_0_hps_io_hps_io_emac1_inst_TX_CTL(HPS_ENET_TX_EN),
        .hps_0_hps_io_hps_io_emac1_inst_RX_CLK(HPS_ENET_RX_CLK),
        .hps_0_hps_io_hps_io_emac1_inst_RXD1(HPS_ENET_RX_DATA[1]),
```

```
        .hps_0_hps_io_hps_io_emac1_inst_RXD2(HPS_ENET_RX_DATA[2]),
        .hps_0_hps_io_hps_io_emac1_inst_RXD3(HPS_ENET_RX_DATA[3]),
          //HPS SD card
        .hps_0_hps_io_hps_io_sdio_inst_CMD(HPS_SD_CMD),
        .hps_0_hps_io_hps_io_sdio_inst_D0(HPS_SD_DATA[0]),
        .hps_0_hps_io_hps_io_sdio_inst_D1(HPS_SD_DATA[1]),
        .hps_0_hps_io_hps_io_sdio_inst_CLK(HPS_SD_CLK),
        .hps_0_hps_io_hps_io_sdio_inst_D2(HPS_SD_DATA[2]),
        .hps_0_hps_io_hps_io_sdio_inst_D3(HPS_SD_DATA[3]),
          //HPS USB
        .hps_0_hps_io_hps_io_usb1_inst_D0(HPS_USB_DATA[0]),
        .hps_0_hps_io_hps_io_usb1_inst_D1(HPS_USB_DATA[1]),
        .hps_0_hps_io_hps_io_usb1_inst_D2(HPS_USB_DATA[2]),
        .hps_0_hps_io_hps_io_usb1_inst_D3(HPS_USB_DATA[3]),
        .hps_0_hps_io_hps_io_usb1_inst_D4(HPS_USB_DATA[4]),
        .hps_0_hps_io_hps_io_usb1_inst_D5(HPS_USB_DATA[5]),
        .hps_0_hps_io_hps_io_usb1_inst_D6(HPS_USB_DATA[6]),
        .hps_0_hps_io_hps_io_usb1_inst_D7(HPS_USB_DATA[7]),
        .hps_0_hps_io_hps_io_usb1_inst_CLK(HPS_USB_CLKOUT),
        .hps_0_hps_io_hps_io_usb1_inst_STP(HPS_USB_STP),
        .hps_0_hps_io_hps_io_usb1_inst_DIR(HPS_USB_DIR),
        .hps_0_hps_io_hps_io_usb1_inst_NXT(HPS_USB_NXT),
          //HPS SPI
        .hps_0_hps_io_hps_io_spim1_inst_CLK(HPS_SPIM_CLK),
        .hps_0_hps_io_hps_io_spim1_inst_MOSI(HPS_SPIM_MOSI),
        .hps_0_hps_io_hps_io_spim1_inst_MISO(HPS_SPIM_MISO),
        .hps_0_hps_io_hps_io_spim1_inst_SS0(HPS_SPIM_SS),
          //HPS UART
        .hps_0_hps_io_hps_io_uart0_inst_RX(HPS_UART_RX),
        .hps_0_hps_io_hps_io_uart0_inst_TX(HPS_UART_TX),
          //HPS I2C 0
        .hps_0_hps_io_hps_io_i2c0_inst_SDA(HPS_I2C0_SDAT),
        .hps_0_hps_io_hps_io_i2c0_inst_SCL(HPS_I2C0_SCLK),
          //HPS I2C 1
        .hps_0_hps_io_hps_io_i2c1_inst_SDA(HPS_I2C1_SDAT),
        .hps_0_hps_io_hps_io_i2c1_inst_SCL(HPS_I2C1_SCLK),
          //HPS GPIO
        .hps_0_hps_io_hps_io_gpio_inst_GPIO09(HPS_CONV_USB_N),
        .hps_0_hps_io_hps_io_gpio_inst_GPIO35(HPS_ENET_INT_N),
        .hps_0_hps_io_hps_io_gpio_inst_GPIO40(HPS_LTC_GPIO),
        .hps_0_hps_io_hps_io_gpio_inst_GPIO53(HPS_LED),
        .hps_0_hps_io_hps_io_gpio_inst_GPIO54(HPS_KEY),
        .hps_0_hps_io_hps_io_gpio_inst_GPIO61(HPS_GSENSOR_INT),
```

```
          //FPGA partition
        .led_pio_external_connection_export(fpga_led_internal),
        .button_pio_external_connection_export(fpga_debounced_buttons),
        .dipsw_pio_external_connection_export(SW),
        .hps_0_f2h_cold_reset_req_reset_n(~hps_cold_reset),
        .hps_0_f2h_debug_reset_req_reset_n(~hps_debug_reset),
        .hps_0_f2h_stm_hw_events_stm_hwevents(stm_hw_events),
        .hps_0_f2h_warm_reset_req_reset_n(~hps_warm_reset),
        .hps_0_h2f_reset_reset_n(hps_fpga_reset_n),
        //HPS VIP connections
        .alt_vip_itc_0_clocked_video_vid_clk(HDMI_TX_CLK),
        .alt_vip_itc_0_clocked_video_vid_data(HDMI_TX_D),
        .alt_vip_itc_0_clocked_video_underflow( ),
        .alt_vip_itc_0_clocked_video_vid_datavalid(HDMI_TX_DE),
        .alt_vip_itc_0_clocked_video_vid_v_sync(HDMI_TX_VS),
        .alt_vip_itc_0_clocked_video_vid_h_sync(HDMI_TX_HS),
        .alt_vip_itc_0_clocked_video_vid_f( ),
        .alt_vip_itc_0_clocked_video_vid_h( ),
        .alt_vip_itc_0_clocked_video_vid_v( ),
        .clk_130_clk(clk_130),
        .servo_act_x_external_connection_export(servo_act_x_connection),
        .servo_act_y_external_connection_export(servo_act_y_connection),
        .servo_pred_x_external_connection_export(servo_pred_x_connection),
        .servo_pred_y_external_connection_export(servo_pred_y_connection)

    );

//=======================================================
//  Servo control
//=======================================================

/*Horizontal servo Control*/
PWM_Controller PWM_x (
        .clk (clk),
        .PWM_out_x (PWM_out_x),
        .xdir (xdir [11:0]),
        .enable(SW[0]),
        .init(SW[1]),
        .does_nothing_x(nothing_x [31: 0]),
        .calib(SW[2])
    );

```

```
/*Vertical servo Control*/
PWM_Controller2 PWM_y (
        .clk (clk),
        .PWM_out_y (PWM_out_y),
        .ydir (ydir [11:0]),
        .enable(SW[0]),
        .init(SW[1]),
        .does_nothing_y(nothing_y [31: 0]),
        .calib(SW[2])
    );

/*The phase lock loop for the PWM controllers*/
PLL PLL_inst(
        .refclk (FPGA_CLK1_50),  //Input Clock
        .rst (1'b0),           // Reset
        .outclk_1 (clk)       // 1.5 KHz Output Clock
    );

endmodule
```

Appendix C: PWM Controller for the X Direction

```
/**************************************************************************/
/*      Project: Object Detection for degree project                     */
/*      File: PWM controller for the x direction servo motor             */
/*      Created by: Neal Palmer                                          */
/*      Group Members: Luke Ferguson & Ronald Chan                       */
/*      Semester: Fall & Winter 2017/2018                                */
/**************************************************************************/

//////////////// controller x direction ///////////////////////


`timescale 1ps / 1ps
module PWM_Controller (

    input clk,
    input init,
    input [11:0] xdir,
    output reg PWM_out_x,
    input enable,
    input [31:0] does_nothing_x,
    input calib

    );


    reg [8:0] PWM_CW_x;                    // 9 bit PWM input


    wire [11:0] counter_out;            // 12 bit counter output
    wire [12:0] delay1;
    wire [14:0] delay2;
    wire [15:0] delay3;



    always @ (posedge clk && enable)  //init = initalize sevo motor direction
    begin

```

```
/////////////// X-direction servo conditions //////////////////

    if ((320 >= xdir) && (xdir >= 250)
       && (counter_out==4095)
       && (does_nothing_x == 32'b0)
       && (init == 0) && (calib == 0))//12-bit delay
            begin
                    if (PWM_CW_x > 265)
                    PWM_CW_x <= PWM_CW_x - 1'b1;
                    else
                    PWM_CW_x <= PWM_CW_x;
            end

            else if ((250 > xdir)
                    && (xdir >= 200)
                    && (delay1==8191)
                    && (does_nothing_x == 32'b0)
                    && (init == 0)
                    && (calib == 0))    // 13-bit delay
            begin
                    if (PWM_CW_x > 265)
                    PWM_CW_x <= PWM_CW_x - 1'b1;
                    else
                    PWM_CW_x <= PWM_CW_x;
            end

            else if ((200 > xdir)
                    && (xdir > 170)
                    && (delay3==65535)
                    && (does_nothing_x == 32'b0)
                    && (init == 0)
                    && (calib == 0))        // 16-bit delay
            begin
                    if (PWM_CW_x > 265)
                    PWM_CW_x <= PWM_CW_x - 1'b1;
                    else
                    PWM_CW_x <= PWM_CW_x;
            end

            else if ((170 >= xdir)
                an && (xdir >= 150)
                && (does_nothing_x == 32'b0)
                && (init == 0)
```

```
                && (calib == 0))     // holds position
            begin
                    PWM_CW_x <= PWM_CW_x;
            end

            else if ((150 > xdir)
                && (xdir >= 120)
                && (delay3==65535)
                && (does_nothing_x == 32'b0)
                && (init == 0)
                && (calib == 0)) // 16-bit delay
            begin
                    if (PWM_CW_x < 370)
                    PWM_CW_x <= PWM_CW_x + 1'b1;
                    else
                    PWM_CW_x <= PWM_CW_x;
            end

            else if ((120 > xdir)
                && (xdir >= 70)
                && (delay1==8191)
                && (does_nothing_x == 32'b0)
                && (init == 0)
                && (calib == 0))   // 13-bit delay
            begin
                    if (PWM_CW_x < 370)
                    PWM_CW_x <= PWM_CW_x + 1'b1;
                    else
                    PWM_CW_x <= PWM_CW_x;
            end

            else if ((70 > xdir)
                && (xdir >= 0)
                && (counter_out==4095)
                && (does_nothing_x == 32'b0)
                && (init == 0)
                && (calib == 0))  //12-bit delay
            begin
                    if (PWM_CW_x < 370)
                    PWM_CW_x <= PWM_CW_x + 1'b1;
                    else
                    PWM_CW_x <= PWM_CW_x;
            end
```

```
//////////////// Initialization //////////////////////
else if (init)
            begin
                    PWM_CW_x <= 310;

            end

/////////////// X-direction Servo Output//////////////

if ({4'b000,PWM_CW_x} > counter_out)//compare 8-bit codeword & 12-bit counter

 PWM_out_x <= 1;


 else

 PWM_out_x <= 0;

/////////////// Calibration //////////////////////////
 if (calib)
  begin
        PWM_CW_x <= xdir;
  end

 end


   counter counter_inst(

        .clk (clk),
        .counter_out (counter_out),
        .delay1 (delay1),
        .delay2 (delay2),
        .delay3 (delay3)
        );



endmodule

```

Appendix D: PWM Controller for the Y Direction Servo Motor

```
/*************************************************************************/
/*      Project: Object Detection for degree project                     */
/*      File: PWM controller for the y direction servo motor             */
/*      Created by: Neal Palmer                                          */
/*      Group Members: Luke Ferguson & Ronald Chan                       */
/*      Semester: Fall & Winter 2017/2018                                */
/*************************************************************************/

/////////////////// controller y direction //////////////////////


`timescale 1ps / 1ps
module PWM_Controller2 (

    input clk,
    input init,
    input [11:0] ydir,
    output reg PWM_out_y,
    input enable,
    input [31: 0] does_nothing_y,
    input calib

    );

    reg [8:0] PWM_CW_y;

    wire [11:0] counter_out;  // 12 bit counter output
    wire [13:0] delay1;
    wire [14:0] delay2;
    wire [15:0] delay3;

    always @ (posedge clk && enable)
    begin

///////////////////Y-direction servo conditions//////////////////

    if ((240 >= ydir)
        && (ydir > 180)
        && (counter_out==4095)
        && (does_nothing_y == 32'b0)
        && (init==0)
```

```
        && (calib == 0))        // 12-bit delay
        begin
            if (PWM_CW_y < 320)
                PWM_CW_y <= PWM_CW_y + 1'b1;
            else
                PWM_CW_y <= PWM_CW_y;
        end

    else if ((180 >= ydir)
        && (ydir > 130)
        && (delay3==65535)
        && (does_nothing_y == 32'b0)
        && (init==0)
        && (calib == 0))        // 16-bit delay
        begin
            if (PWM_CW_y < 320)
                PWM_CW_y <= PWM_CW_y + 1'b1;
            else
            PWM_CW_y <= PWM_CW_y;
        end

    else if ((110 >= ydir)
        && (ydir <= 130)
        && (does_nothing_y == 32'b0)
        && (init==0)
        && (calib == 0))        // hold position
        begin
        PWM_CW_y <= PWM_CW_y;
        end

    else if ((110 > ydir)
        && (ydir >= 50)
        && (delay3==65535)
        && (does_nothing_y == 32'b0)
        && (init==0)
        && (calib == 0))        // 16-bit delay
        begin
            if (PWM_CW_y > 235)
                PWM_CW_y <= PWM_CW_y - 1'b1;
            else
                PWM_CW_y <= PWM_CW_y;
            end

```

```
    else if ((50 > ydir)
        && (ydir >= 0)
        && (counter_out==4095)
        && (does_nothing_y == 32'b0)
        && (init==0)
        && (calib == 0))        // 12-bit delay
        begin
            if (PWM_CW_y > 235)
                PWM_CW_y <= PWM_CW_y - 1'b1;
            else
                PWM_CW_y <= PWM_CW_y;
        end

//////////////////////// initialization //////////////////////////
    else if (init)
        begin
            PWM_CW_y <= 256;
        end

/////////////////// Y-direction Servo Output/////////////////////

if ({4'b000,PWM_CW_y} > counter_out)//compares 8-bit codeword/12-bit counter
        PWM_out_y <= 1;
    else
        PWM_out_y <= 0;

///////////////////// Calibration //////////////////////
    if (calib)
        begin
            PWM_CW_y <= ydir;
        end

    end

    counter counter_inst(

        .clk (clk),
        .counter_out (counter_out),
        .delay1 (delay1),
        .delay2 (delay2),
        .delay3 (delay3)
        );
endmodule
```

Appendix E: Counter used by the PWM controllers

```
/***************************************************************************/
/*        Project: Object Detection for degree project                    */
/*        File: Counter used by the PWM Controllers                       */
/*        Created by: Neal Palmer                                         */
/*        Group Members: Luke Ferguson & Ronald Chan                      */
/*        Semester: Fall & Winter 2017/2018                               */
/***************************************************************************/

//////////////////////////     counter    ///////////////////////////////
`timescale 1ps / 1ps
module counter (
 input clk,
 output reg [11:0] counter_out,   //period roughly = 16ms
 output reg [12:0] delay1,
 output reg [14:0] delay2,
 output reg [15:0] delay3
 );

always @(posedge clk)
 begin
     counter_out <= #1 counter_out + 1'b1;
     delay1 <= #1 delay1 + 1'b1;
     delay2 <= #1 delay2 + 1'b1;
     delay3 <= #1 delay3 + 1'b1;
 end

endmodule
```

Appendix F: The HPS physical addresses header

```
#ifndef _ALTERA_HPS_0_H_
#define _ALTERA_HPS_0_H_

 //
 // This file was automatically generated by the swinfo2header utility.
 //
 // Created from SOPC Builder system 'soc_system' in
 // file './soc_system.sopcinfo'.


 //
 // This file contains macros for module 'hps_0' and devices
 // connected to the following masters:
 //   h2f_axi_master
 //   h2f_lw_axi_master
 //
 // Do not include this header file and another header file created for a
 // different module or master group at the same time.
 // Doing so may result in duplicate macro names.
 // Instead, use the system header file which has macros with unique names.
 //



 //
 // Macros for device 'servo_pred_x', class 'altera_avalon_pio'
 // The macros are prefixed with 'SERVO_PRED_X_'.
 // The prefix is the slave descriptor.
 //
#define SERVO_PRED_X_COMPONENT_TYPE altera_avalon_pio
#define SERVO_PRED_X_COMPONENT_NAME servo_pred_x
#define SERVO_PRED_X_BASE 0x20
#define SERVO_PRED_X_SPAN 32
#define SERVO_PRED_X_END 0x3f
#define SERVO_PRED_X_BIT_CLEARING_EDGE_REGISTER 0
#define SERVO_PRED_X_BIT_MODIFYING_OUTPUT_REGISTER 1
#define SERVO_PRED_X_CAPTURE 0
#define SERVO_PRED_X_DATA_WIDTH 32
#define SERVO_PRED_X_DO_TEST_BENCH_WIRING 0
#define SERVO_PRED_X_DRIVEN_SIM_VALUE 0
#define SERVO_PRED_X_EDGE_TYPE NONE
#define SERVO_PRED_X_FREQ 50000000
```

```
#define SERVO_PRED_X_HAS_IN 0
#define SERVO_PRED_X_HAS_OUT 1
#define SERVO_PRED_X_HAS_TRI 0
#define SERVO_PRED_X_IRQ_TYPE NONE
#define SERVO_PRED_X_RESET_VALUE 0

 //
 // Macros for device 'servo_act_y', class 'altera_avalon_pio'
 // The macros are prefixed with 'SERVO_ACT_Y_'.
 // The prefix is the slave descriptor.
 //
#define SERVO_ACT_Y_COMPONENT_TYPE altera_avalon_pio
#define SERVO_ACT_Y_COMPONENT_NAME servo_act_y
#define SERVO_ACT_Y_BASE 0x60
#define SERVO_ACT_Y_SPAN 32
#define SERVO_ACT_Y_END 0x7f
#define SERVO_ACT_Y_BIT_CLEARING_EDGE_REGISTER 0
#define SERVO_ACT_Y_BIT_MODIFYING_OUTPUT_REGISTER 1
#define SERVO_ACT_Y_CAPTURE 0
#define SERVO_ACT_Y_DATA_WIDTH 32
#define SERVO_ACT_Y_DO_TEST_BENCH_WIRING 0
#define SERVO_ACT_Y_DRIVEN_SIM_VALUE 0
#define SERVO_ACT_Y_EDGE_TYPE NONE
#define SERVO_ACT_Y_FREQ 50000000
#define SERVO_ACT_Y_HAS_IN 0
#define SERVO_ACT_Y_HAS_OUT 1
#define SERVO_ACT_Y_HAS_TRI 0
#define SERVO_ACT_Y_IRQ_TYPE NONE
#define SERVO_ACT_Y_RESET_VALUE 0

 //
 // Macros for device 'servo_pred_y', class 'altera_avalon_pio'
 // The macros are prefixed with 'SERVO_PRED_Y_'.
 // The prefix is the slave descriptor.
 //
#define SERVO_PRED_Y_COMPONENT_TYPE altera_avalon_pio
#define SERVO_PRED_Y_COMPONENT_NAME servo_pred_y
#define SERVO_PRED_Y_BASE 0x80
#define SERVO_PRED_Y_SPAN 32
#define SERVO_PRED_Y_END 0x9f
#define SERVO_PRED_Y_BIT_CLEARING_EDGE_REGISTER 0
#define SERVO_PRED_Y_BIT_MODIFYING_OUTPUT_REGISTER 1
#define SERVO_PRED_Y_CAPTURE 0
#define SERVO_PRED_Y_DATA_WIDTH 32
```

```
#define SERVO_PRED_Y_DO_TEST_BENCH_WIRING 0
#define SERVO_PRED_Y_DRIVEN_SIM_VALUE 0
#define SERVO_PRED_Y_EDGE_TYPE NONE
#define SERVO_PRED_Y_FREQ 50000000
#define SERVO_PRED_Y_HAS_IN 0
#define SERVO_PRED_Y_HAS_OUT 1
#define SERVO_PRED_Y_HAS_TRI 0
#define SERVO_PRED_Y_IRQ_TYPE NONE
#define SERVO_PRED_Y_RESET_VALUE 0

 //
 // Macros for device 'servo_act_x', class 'altera_avalon_pio'
 // The macros are prefixed with 'SERVO_ACT_X_'.
 // The prefix is the slave descriptor.
 //
#define SERVO_ACT_X_COMPONENT_TYPE altera_avalon_pio
#define SERVO_ACT_X_COMPONENT_NAME servo_act_x
#define SERVO_ACT_X_BASE 0xa0
#define SERVO_ACT_X_SPAN 32
#define SERVO_ACT_X_END 0xbf
#define SERVO_ACT_X_BIT_CLEARING_EDGE_REGISTER 0
#define SERVO_ACT_X_BIT_MODIFYING_OUTPUT_REGISTER 1
#define SERVO_ACT_X_CAPTURE 0
#define SERVO_ACT_X_DATA_WIDTH 32
#define SERVO_ACT_X_DO_TEST_BENCH_WIRING 0
#define SERVO_ACT_X_DRIVEN_SIM_VALUE 0
#define SERVO_ACT_X_EDGE_TYPE NONE
#define SERVO_ACT_X_FREQ 50000000
#define SERVO_ACT_X_HAS_IN 0
#define SERVO_ACT_X_HAS_OUT 1
#define SERVO_ACT_X_HAS_TRI 0
#define SERVO_ACT_X_IRQ_TYPE NONE
#define SERVO_ACT_X_RESET_VALUE 0

 //
 // Macros for device 'sysid_qsys', class 'altera_avalon_sysid_qsys'
 // The macros are prefixed with 'SYSID_QSYS_'.
 // The prefix is the slave descriptor.
 //
#define SYSID_QSYS_COMPONENT_TYPE altera_avalon_sysid_qsys
#define SYSID_QSYS_COMPONENT_NAME sysid_qsys
#define SYSID_QSYS_BASE 0x1000
#define SYSID_QSYS_SPAN 8
#define SYSID_QSYS_END 0x1007
```

```
#define SYSID_QSYS_ID 2899645186
#define SYSID_QSYS_TIMESTAMP 1521068338

 //
 // Macros for device 'jtag_uart', class 'altera_avalon_jtag_uart'
 // The macros are prefixed with 'JTAG_UART_'.
 // The prefix is the slave descriptor.
 //
#define JTAG_UART_COMPONENT_TYPE altera_avalon_jtag_uart
#define JTAG_UART_COMPONENT_NAME jtag_uart
#define JTAG_UART_BASE 0x2000
#define JTAG_UART_SPAN 8
#define JTAG_UART_END 0x2007
#define JTAG_UART_IRQ 1
#define JTAG_UART_READ_DEPTH 64
#define JTAG_UART_READ_THRESHOLD 8
#define JTAG_UART_WRITE_DEPTH 64
#define JTAG_UART_WRITE_THRESHOLD 8

 //
 // Macros for device 'led_pio', class 'altera_avalon_pio'
 // The macros are prefixed with 'LED_PIO_'.
 // The prefix is the slave descriptor.
 //
#define LED_PIO_COMPONENT_TYPE altera_avalon_pio
#define LED_PIO_COMPONENT_NAME led_pio
#define LED_PIO_BASE 0x3000
#define LED_PIO_SPAN 16
#define LED_PIO_END 0x300f
#define LED_PIO_BIT_CLEARING_EDGE_REGISTER 0
#define LED_PIO_BIT_MODIFYING_OUTPUT_REGISTER 0
#define LED_PIO_CAPTURE 0
#define LED_PIO_DATA_WIDTH 7
#define LED_PIO_DO_TEST_BENCH_WIRING 0
#define LED_PIO_DRIVEN_SIM_VALUE 0
#define LED_PIO_EDGE_TYPE NONE
#define LED_PIO_FREQ 50000000
#define LED_PIO_HAS_IN 0
#define LED_PIO_HAS_OUT 1
#define LED_PIO_HAS_TRI 0
#define LED_PIO_IRQ_TYPE NONE
#define LED_PIO_RESET_VALUE 127

```

```
 //
 // Macros for device 'dipsw_pio', class 'altera_avalon_pio'
 // The macros are prefixed with 'DIPSW_PIO_'.
 // The prefix is the slave descriptor.
 //
#define DIPSW_PIO_COMPONENT_TYPE altera_avalon_pio
#define DIPSW_PIO_COMPONENT_NAME dipsw_pio
#define DIPSW_PIO_BASE 0x4000
#define DIPSW_PIO_SPAN 16
#define DIPSW_PIO_END 0x400f
#define DIPSW_PIO_IRQ 0
#define DIPSW_PIO_BIT_CLEARING_EDGE_REGISTER 1
#define DIPSW_PIO_BIT_MODIFYING_OUTPUT_REGISTER 0
#define DIPSW_PIO_CAPTURE 1
#define DIPSW_PIO_DATA_WIDTH 4
#define DIPSW_PIO_DO_TEST_BENCH_WIRING 0
#define DIPSW_PIO_DRIVEN_SIM_VALUE 0
#define DIPSW_PIO_EDGE_TYPE ANY
#define DIPSW_PIO_FREQ 50000000
#define DIPSW_PIO_HAS_IN 1
#define DIPSW_PIO_HAS_OUT 0
#define DIPSW_PIO_HAS_TRI 0
#define DIPSW_PIO_IRQ_TYPE EDGE
#define DIPSW_PIO_RESET_VALUE 0

 //
 // Macros for device 'button_pio', class 'altera_avalon_pio'
 // The macros are prefixed with 'BUTTON_PIO_'.
 // The prefix is the slave descriptor.
 //
#define BUTTON_PIO_COMPONENT_TYPE altera_avalon_pio
#define BUTTON_PIO_COMPONENT_NAME button_pio
#define BUTTON_PIO_BASE 0x5000
#define BUTTON_PIO_SPAN 16
#define BUTTON_PIO_END 0x500f
#define BUTTON_PIO_IRQ 2
#define BUTTON_PIO_BIT_CLEARING_EDGE_REGISTER 1
#define BUTTON_PIO_BIT_MODIFYING_OUTPUT_REGISTER 0
#define BUTTON_PIO_CAPTURE 1
#define BUTTON_PIO_DATA_WIDTH 2
#define BUTTON_PIO_DO_TEST_BENCH_WIRING 0
#define BUTTON_PIO_DRIVEN_SIM_VALUE 0
#define BUTTON_PIO_EDGE_TYPE FALLING
#define BUTTON_PIO_FREQ 50000000
```

```
#define BUTTON_PIO_HAS_IN 1
#define BUTTON_PIO_HAS_OUT 0
#define BUTTON_PIO_HAS_TRI 0
#define BUTTON_PIO_IRQ_TYPE EDGE
#define BUTTON_PIO_RESET_VALUE 0

 //
 // Macros for device 'ILC', class 'interrupt_latency_counter'
 // The macros are prefixed with 'ILC_'.
 // The prefix is the slave descriptor.
 //
#define ILC_COMPONENT_TYPE interrupt_latency_counter
#define ILC_COMPONENT_NAME ILC
#define ILC_BASE 0x30000
#define ILC_SPAN 256
#define ILC_END 0x300ff

 //
 // Macros for device 'alt_vip_vfr_hdmi', class 'alt_vip_vfr'
 // The macros are prefixed with 'ALT_VIP_VFR_HDMI_'.
 // The prefix is the slave descriptor.
 //
#define ALT_VIP_VFR_HDMI_COMPONENT_TYPE alt_vip_vfr
#define ALT_VIP_VFR_HDMI_COMPONENT_NAME alt_vip_vfr_hdmi
#define ALT_VIP_VFR_HDMI_BASE 0x31000
#define ALT_VIP_VFR_HDMI_SPAN 128
#define ALT_VIP_VFR_HDMI_END 0x3107f


#endif /// _ALTERA_HPS_0_H_ ///
```

Appendix G: The Object Tracking C++ Code

```
/************************************************/
/*Filename: ball_tracking.cpp                   */
/*Author: Ronald Chan (adapted from Python      */
/*        script also by Ronald Chan            */
/*        with parts inspired by                */
/*        A. Rosebrock [10]                     */
/*        and  W. Lucetti [9])                  */
/*Project: Degree Project                       */
/*Description: Script using OpenCV API to detect*/
/*             and track ball based on the      */
/*             ball's color.                    */
/*                                              */
/*rev.5: working distance measurement           */
/*       and object detection                   */
/*rev.6: added fps count                        */
/*rev.7: added Kalman filtering to improve      */
/*       tracking, major rework done to         */
/*       accommodate new features               */
/*rev.8: fixed bugs, improved speed             */
/*rev.9: added headers and code to              */
/*       write to FPGA fabric                   */
/************************************************/
#include "opencv2/videoio.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/imgproc.hpp"
#include <opencv2/video/video.hpp>
#include <opencv2/core/core.hpp>
#include <iostream>
#include <vector>
#include <thread>
#include <mutex>
#include <sys/time.h>

#include <fstream>
#include <unistd.h>
#include <fcntl.h>
#include <signal.h>
#include <sys/mman.h>
#include "hps_0.h"

using namespace std;
```

```
using namespace cv;

//function declarations
void handler(int signo);
float initialize(UMat frame);

//global variables
UMat frame, dummy_frame;
float focal_length;

//range of colors to track
#define HSV_lower Scalar(5,0,210)
#define HSV_upper Scalar(40,255,255)

//known diameter of ball
#define known_diameter 0.0381

//approximate distance from camera to object
#define known_distance 0.6858

//set size of captured frame
#define new_width 320
#define new_height 240

//(LW H2F Bridge address
#define REG_BASE 0xff200000
//(LW H2F Bridge Span
#define REG_SPAN 0x00200000

//addresses & variables used for memory mapping
void *base;
uint32_t *servo_x_act, *servo_x_pred;
uint32_t *servo_y_act, *servo_y_pred;
int fd;

/**********************************************/
/*function: mai                              */
/**********************************************/
int main(void) {

    // >>>>>> Memory mapping
    fd = open("/dev/mem", O_RDWR|O_SYNC);
```

```
    if(fd<0)
    {
        printf("Can't open memory .\n");
        return -1;
    }
    //calculate the base address
    base = mmap(NULL,
                REG_SPAN,
                PROT_READ|PROT_WRITE,
                MAP_SHARED,
                fd,
                REG_BASE);
    if(base == MAP_FAILED)
    {
        printf("Can't map memory. \n");
        close(fd);
        return -1;
    }
    //calculate the memory address of the four variables
    servo_x_act = (uint32_t*)(base + SERVO_ACT_X_BASE);
    servo_x_pred = (uint32_t*)(base + SERVO_PRED_X_BASE);
    servo_y_act = (uint32_t*)(base + SERVO_ACT_Y_BASE);
    servo_y_pred = (uint32_t*)(base + SERVO_PRED_Y_BASE);
    signal(SIGINT, handler);
    // <<<<<< Memory mapping


    // >>>>>> Kalman filter
    int stateSize = 6; //number of state variables
    int measSize = 4; //number of measurement variables
    int contrSize = 2; //size of control vector
    unsigned int type = CV_32F;

    //instantiate a Kalman filter with cv::KalmanFilter
    KalmanFilter kf(stateSize, measSize, contrSize, type);

    Mat state(stateSize, 1, type); //state vector [x ,y, v_x, v_y, w, h]'
    Mat meas(measSize, 1, type);  //measurement vector [z_x, z_y, z_w, z_h]'

```

```
    //transition State Matrix A
    //note: set dT at each processing step!
    // [ 1 0 dT 0  0 0 ]
    // [ 0 1 0  dT 0 0 ]
    // [ 0 0 1  0  0 0 ]
    // [ 0 0 0  1  0 0 ]
    // [ 0 0 0  0  1 0 ]
    // [ 0 0 0  0  0 1 ]
    //first create a 6x6 identity matrix --> dTs added on second measurement
    setIdentity(kf.transitionMatrix);

    //measure Matrix H
    // [ 1 0 0 0 0 0 ]
    // [ 0 1 0 0 0 0 ]
    // [ 0 0 0 0 1 0 ]
    // [ 0 0 0 0 0 1 ]
    kf.measurementMatrix = Mat::zeros(measSize, stateSize, type);
    kf.measurementMatrix.at<float>(0) = 1.0f;
    kf.measurementMatrix.at<float>(7) = 1.0f;
    kf.measurementMatrix.at<float>(16) = 1.0f;
    kf.measurementMatrix.at<float>(23) = 1.0f;

    //control matrix B
    // [ 1 0 ]
    // [ 0 1 ]
    // [ 0 0 ]
    // [ 0 0 ]
    // [ 0 0 ]
    // [ 0 0 ]
    setIdentity(kf.controlMatrix);
    kf.controlMatrix.at<float>(0) = 1.0f;
    kf.controlMatrix.at<float>(3) = 1.0f;
    //control inputs are the motions in the x and y directions

    //Process Noise Covariance Matrix Q
    //these values are found by trial and error
    // [ Ex   0    0     0     0    0  ]
    // [ 0    Ey   0     0     0    0  ]
    // [ 0    0    Ev_x  0     0    0  ]
    // [ 0    0    0     Ev_y  0    0  ]
    // [ 0    0    0     0     Ew   0  ]
    // [ 0    0    0     0     0    Eh ]
    kf.processNoiseCov.at<float>(0) = 1e-3;
    kf.processNoiseCov.at<float>(7) = 1e-3;
```

```
    kf.processNoiseCov.at<float>(14) = 1e-1;
    kf.processNoiseCov.at<float>(21) = 1e-1;
    kf.processNoiseCov.at<float>(28) = 1e-2;
    kf.processNoiseCov.at<float>(35) = 1e-2;

    //measures noise covariance matrix R
    setIdentity(kf.measurementNoiseCov, Scalar(1e-1));
    // <<<<<< Kalman Filter


    // >>>>>> Setup
    //create videoCapture object
    VideoCapture capture;

    //initialize window to display captured video
    namedWindow("ball tracker + distance detection", CV_WINDOW_AUTOSIZE);

    //start the video stream from web cam
    capture.open(-1); //open the first available camera

    //error handling for capture failure
    if(!capture.isOpened())
    {
        printf("Failed to access webcam\n");
        return -1;
    }

    //read the first captured frame and initialize the system
    capture.read(frame);

    //set captured frame size
    capture.set(CV_CAP_PROP_FRAME_WIDTH, new_width);
    capture.set(CV_CAP_PROP_FRAME_HEIGHT, new_height);

    //call initialize function to grab focal length
    focal_length = initialize(frame);

    //ball tracking setup before main loop
    double ticks = 0;
    bool found = false;
    bool measure_distance = false; //do not start with distance measurement
    bool center_select = false; //switch between predicted and actual centers
    int largest_area;
    bool is_square;
```

```
    bool show_thresholding = false;
    bool use_kalman = false;

    //fps count and elapsed time setup
    char str_time[50];
    static struct timeval last_time;
    struct timeval current_time;
    static float last_fps;
    float t;
    float fps;
    float elapsed_time = 0.0;
    // <<<<<< Setup


    // >>>>>> main loop
    while(capture.read(frame))
    {
        //error handling check for no frame captured
        if(frame.empty())
        {
            printf("No captured frame -- Break!");
            break;
        }

        //find t-t0
        double precTick = ticks;
        ticks = (double) getTickCount();
        double dT = (ticks - precTick) / getTickFrequency(); //seconds

        Mat new_frame;
        frame.copyTo(new_frame);


        // >>>>>> State prediction
        if (found) //should be true on second captured frame
        {
            //update matrix A
            kf.transitionMatrix.at<float>(2) = dT;
            kf.transitionMatrix.at<float>(9) = dT;

            //cout << "dT:" << endl << dT << endl;

            state = kf.predict(); //predict location
            //cout << "State post:" << endl << state << endl;
```

```
            //make a rectangle object with width, height and (x,y)
            Rect predicted_rect;
            predicted_rect.width = state.at<float>(4);
            predicted_rect.height = state.at<float>(5);
            predicted_rect.x = state.at<float>(0) - predicted_rect.width /2;
            predicted_rect.y = state.at<float>(1) - predicted_rect.height /2;

            //predicted center coordinates
            Point predicted_center;
            predicted_center.x = state.at<float>(0);
            predicted_center.y = state.at<float>(1);


            //select between actual and predicted coordinates output
            //if there is no previous actual coordinate, use predicted
            if (center_select && use_kalman)
            {
                //place a blue dot at predicted center of ball
                circle(new_frame, predicted_center, 2, CV_RGB(0,0,255), -1);

                //put a blue rectangle around predicted ball location
                rectangle(new_frame, predicted_rect, CV_RGB(0,0,255), 2);

                //label predicted location at bottom right of outline
                putText(new_frame, "predicted",
                Point(predicted_center.x + (predicted_rect.width / 2) + 5,
                    predicted_center.y + (predicted_rect.height / 2)),
                FONT_HERSHEY_SIMPLEX, 0.5, CV_RGB(0,0,255), 1);

                //assign predicted coordinates to memory addresses
                *servo_x_pred = predicted_center.x;
                *servo_y_pred = predicted_center.y;
                cout << "x pred: " << *servo_x_pred << endl;
                cout << "y pred: " << *servo_y_pred << endl;
                //display predicted center coordinates onto frame
                stringstream sstr_pred;
sstr_pred << "(" << predicted_center.x << "," << predicted_center.y << ")";
                putText(new_frame, sstr_pred.str(),
                        Point(predicted_center.x, predicted_center.y + 5),
                        FONT_HERSHEY_SIMPLEX, 0.5, CV_RGB(0,0,255), 1);
            }
```

```
            else
            {
                *servo_x_pred = 0x0;
                *servo_y_pred = 0x0;
                cout << "x pred: " << *servo_x_pred << endl;
                cout << "y pred: " << *servo_y_pred << endl;
            }
        }
        // <<<<<< State prediction


        // >>>>>> Thresholding
        //smooth out noise with 5x5 Gaussian kernel, sigma X & Y are 3.0
        UMat blur;
        GaussianBlur(frame, blur, Size(5,5), 3.0, 3.0);

        //change to HSV color space
        UMat frame_hsv;
        cvtColor(frame, frame_hsv, COLOR_BGR2HSV);

        //threshold for desired color range
        UMat mask;
        inRange(frame_hsv, HSV_lower, HSV_upper, mask);

        //smooth edges with morphological operations
        //to remove noise and isolate contours
        erode(mask, mask, Mat(), Point(-1,-1), 2);
        dilate(mask, mask, Mat(), Point(-1,-1), 2);

        //view thresholding
        if (show_thresholding == true)
                imshow("Threshold", mask);
        // <<<<<< Thresholding


        // >>>>>> Object detection
        //find all contours in thresholded image
        vector<vector<Point>> contours;

        findContours(mask, contours, RETR_EXTERNAL, CHAIN_APPROX_SIMPLE);

        largest_area = 0;

    //if there are any contours found, search for the largest circular contour
```

```

        Rect bBox;

        if(contours.size()>0)
        {
            //iterate through each contour
            for(size_t i=0; i<abs(contours.size()); i++)
            {
                is_square = false; //reset "squareness" check
                double area = contourArea(contours[i]);//find area of contour
                //determine the largest contour in terms of area
                if(area>largest_area)
                {
                    largest_area = area;
                }

                //check for "roundness"
                bBox = boundingRect(contours[i]); //returns top left vertex

                float ratio = (float) bBox.width / (float) bBox.height;
                if (ratio > 1.0f)
                    ratio = 1.0f / ratio;

                // Searching for a bBox almost square
                if (ratio > 0.75)
                {
                    is_square = true;
                }
            }

//only proceed if radius meets minimum value and bounding rectangle is square
            if(bBox.width > 1 && is_square)
            {
                Point actual_center; //pixel (0,0) is top-left vertex
                actual_center.x = bBox.x + bBox.width / 2;
                actual_center.y = bBox.y + bBox.height / 2;

                //draw red dot in center of actual ball location
                circle(new_frame, actual_center, 2, CV_RGB(255,0,0), -1);

                //draw box around actual ball location
                rectangle(new_frame, bBox, CV_RGB(255,0,0), 2);

```

```
        //label the measured object
        putText( new_frame, "actual",
            Point(actual_center.x + (bBox.width / 2) + 5,
                actual_center.y - bBox.height / 2),
            FONT_HERSHEY_SIMPLEX, 0.5, CV_RGB(255,0,0), 1);

        //show actual center coordinates
        stringstream sstr_act;
sstr_act << "(" << actual_center.x << "," << actual_center.y << ")";
        putText(new_frame, sstr_act.str(),
             Point(actual_center.x, actual_center.y + 5),
             FONT_HERSHEY_SIMPLEX, 0.5, CV_RGB(255,0,0), 1);
// <<<<<< Object detection


        // >>>>>> Distance measurement
 //calculate distance and display it in bottom right corner of frame
      float inches = ( known_diameter * focal_length ) / bBox.width;

        if(measure_distance == true)
        {
            char str_dist[50];
            sprintf(str_dist, "%.2f m", inches);

            //display calculated distance to object

            putText(new_frame, str_dist,
            Point(bBox.x, bBox.y + bBox.height + 20),
            FONT_HERSHEY_SIMPLEX, 1, CV_RGB(0,255,0), 2);
        }
        // <<<<<< Distance measurement


        // >>>>>> Kalman update
        //update Z matrix with new measurements
        meas.at<float>(0) = actual_center.x;
        meas.at<float>(1) = actual_center.y;
        meas.at<float>(2) = (float)bBox.width;
        meas.at<float>(3) = (float)bBox.height;

```

```
        //if there is actual object location found, don't use predicted values
                    center_select = false;
                    *servo_x_act = actual_center.x;
                    *servo_y_act = actual_center.y;
                    cout << "x act: " << *servo_x_act << endl;
                    cout << "y act: " << *servo_y_act << endl;

                    if (!found) // First detection!
                    {
                        // >>>> Initialization
                     kf.errorCovPre.at<float>(0) = 1; // px
                     kf.errorCovPre.at<float>(7) = 1; // px
                     kf.errorCovPre.at<float>(14) = 1;
                     kf.errorCovPre.at<float>(21) = 1;
                     kf.errorCovPre.at<float>(28) = 1; // px
                     kf.errorCovPre.at<float>(35) = 1; // px

                     state.at<float>(0) = meas.at<float>(0);
                     state.at<float>(1) = meas.at<float>(1);
                     state.at<float>(2) = 0; //velocity not inferred by one frame
                     state.at<float>(3) = 0;
                     state.at<float>(4) = meas.at<float>(2);
                     state.at<float>(5) = meas.at<float>(3);
                        // <<<< Initialization

                      kf.statePost = state;

                      found = true;

                    }
                    else
                        kf.correct(meas); // Kalman Correction

                    cout << "Measure matrix:" << endl << meas << endl;
                    // <<<<<< Kalman update


                }
                else
                {
```

```
                //if no actual object location found, use predicted location
                center_select = true;
                *servo_x_act = 0x0;
                *servo_y_act = 0x0;
                cout << "x act: " << *servo_x_act << endl;
                cout << "y act: " << *servo_y_act << endl;
            }
        }

        else
            found = false; //object not found, Kalman not updated

        //display fps count in bottom left corner of frame
        gettimeofday(&current_time, NULL);
        t = (current_time.tv_sec - last_time.tv_sec) +
            (current_time.tv_usec - last_time.tv_usec) / 1000000.;
        fps = 1. / t;
        fps = last_fps * 0.8 + fps * 0.2;
        last_fps = fps;
        last_time = current_time;
        sprintf(str_time, "%2.2f", fps);
        putText(new_frame, str_time, Point(5, new_height-5),
            FONT_HERSHEY_DUPLEX, 1, CV_RGB(255,0,0));

        //display frame
        imshow("ball tracker + distance detection", new_frame);

        //Check for key presses
        int c = waitKey(60); //check for key press every 60ms
    if (c == 27 || c == 'q' || c == 'Q') break; //quit when q or esc pressed
    if (c == 100) measure_distance = !measure_distance; //d to toggle dist.
    if (c == 116) show_thresholding = !show_thresholding; //t to show thres.
        if (c == 107) use_kalman = !use_kalman; //k to toggle Kalman
    }
    // <<<<<< main loop

    //clean up
    capture.release();
    destroyAllWindows();
    return EXIT_SUCCESS;
}
```

```
/***************************************************************/
/*function:  initialize                                        */
/*description: From the first frame, determine the focal length. */
/***************************************************************/
float initialize(UMat frame) {

    //smooth out noise (Gaussian) with 5x5 Gaussian kernel, sigma X & Y are 3.0
    UMat blur;
    GaussianBlur(frame, blur, Size(5,5), 3.0, 3.0);

    //change to HSV color space
    UMat frame_hsv;
    cvtColor(blur, frame_hsv, COLOR_BGR2HSV);

    //threshold for desired color range
    UMat mask;
    inRange(frame_hsv, HSV_lower, HSV_upper, mask);

    //smooth edges with morphological operations
    //to remove noise and isolate contours
    erode(mask, mask, Mat(), Point(-1,-1), 2);
    dilate(mask, mask, Mat(), Point(-1,-1), 2);

    vector<vector<Point>> contours;
    bool is_square;
    int largest_area = 0;

    //Find all contours in thresholded image and determine the largest one
    findContours(mask, contours, RETR_EXTERNAL, CHAIN_APPROX_SIMPLE);

    //if there are any contours found
    if(contours.size()>0)
    {
        vector<cv::Rect> ballsBox;

        //iterate through each contour
        for(size_t i=0; i<abs(contours.size()); i++)
        {
            is_square = false; //reset "squareness" check
            //find area of each contour, store in vector area
            double area=contourArea(contours[i]);
```

```
            if(area>largest_area)
            {
                largest_area = area;
            }
            Rect bBox;
            //returns top left vertex, width and height
            bBox = boundingRect(contours[i]);
            float ratio = (float) bBox.width / (float) bBox.height;
            if (ratio > 1.0f)
                ratio = 1.0f / ratio;
            //searching for a bBox almost square
            if (ratio > 0.75)
            {
                is_square = true;
                ballsBox.push_back(bBox);
            }
        }
        //calculate the focal length and return the value
        //only proceed if radius meets minimum value
        //and bounding rectangle is square
        if(ballsBox[0].width > 1 && is_square)
        {
        focal_length = ((ballsBox[0].width * known_distance)
                            / known_diameter);
            cout << "focal length: " << focal_length << endl;
        if(focal_length>0) return focal_length; //return non-negative value
            }
    }
}
/*******************************************************/
/*function:  handler                                   */
/*description: initialize addresses for memory mapping */
/*******************************************************/
void handler(int signo)
{
    *servo_x_act = 0;
    *servo_x_pred = 0;
    *servo_y_act = 0;
    *servo_y_pred = 0;
    munmap(base, REG_SPAN);
    close(fd);
    exit(0);
}
```

Appendix H: Code from Terasic's Control Panel Reference Design

H.1: The Top-Level HDMI Code

```
I2C_HDMI_Config u_I2C_HDMI_Config (
 .iCLK(FPGA_CLK1_50),
 .iRST_N( 1'b1),
 .I2C_SCLK(HDMI_I2C_SCL),
 .I2C_SDAT(HDMI_I2C_SDA),
 .HDMI_TX_INT(HDMI_TX_INT)
 );

vga_pll  vga_pll_inst(
         .refclk(fpga_clk_50),   //  refclk.clk
        .rst(1'b0),      //   reset.reset
        .outclk_0(clk_65), // outclk0.clk
        .outclk_1(clk_130), // outclk1.clk
        .locked()    //  locked.export
);
assign HDMI_TX_CLK = clk_65;
```

H.2: The I2C_HDMI_Config provided by Terasic

```
module I2C_HDMI_Config (    //  Host Side
                    iCLK,
                    iRST_N,
                    //  I2C Side
                    I2C_SCLK,
                    I2C_SDAT,
                    HDMI_TX_INT,
                    READY
                     );
//  Host Side
input               iCLK;
input               iRST_N;
//  I2C Side
output          I2C_SCLK;
inout               I2C_SDAT;
input               HDMI_TX_INT;
output READY;
```

```

//  Internal Registers/Wires
reg [15:0]  mI2C_CLK_DIV;
reg [23:0]  mI2C_DATA;
reg             mI2C_CTRL_CLK;
reg             mI2C_GO;
wire                mI2C_END;
wire                mI2C_ACK;
reg [15:0]  LUT_DATA;
reg [5:0]       LUT_INDEX;
reg [3:0]       mSetup_ST;
reg READY ;

//  Clock Setting
parameter   CLK_Freq    =   50000000;   //  50  MHz
parameter   I2C_Freq    =   20000;      //  20  KHz
//  LUT Data Number
parameter   LUT_SIZE    =   31;

/////////////////////   I2C Control Clock   ////////////////////////
always@(posedge iCLK or negedge iRST_N)
begin
    if(!iRST_N)
    begin
        mI2C_CTRL_CLK   <=  0;
        mI2C_CLK_DIV    <=  0;
    end
    else
    begin
        if( mI2C_CLK_DIV    < (CLK_Freq/I2C_Freq) )
            mI2C_CLK_DIV    <=  mI2C_CLK_DIV+1;
        else
        begin
            mI2C_CLK_DIV    <=  0;
            mI2C_CTRL_CLK   <=  ~mI2C_CTRL_CLK;
        end
    end
end
```

```
////////////////////////////////////////////////////////////////////
I2C_Controller  u0  (   .CLOCK(mI2C_CTRL_CLK),  //  Controller Work Clock
          .I2C_SCLK(I2C_SCLK),                //  I2C CLOCK
          .I2C_SDAT(I2C_SDAT),                //  I2C DATA
          .I2C_DATA(mI2C_DATA),           //  DATA:[SLAVE_ADDR,SUB_ADDR,DATA]
                        .GO(mI2C_GO),                     //  GO transfor
                        .END(mI2C_END),               //  END transfor
                        .ACK(mI2C_ACK),               //  ACK
                        .RESET(iRST_N)  );
////////////////////////////////////////////////////////////////////
//////////////////////  Config Control  ////////////////////////////
always@(posedge mI2C_CTRL_CLK or negedge iRST_N)
begin
    if(!iRST_N)
    begin
    READY<=0;
        LUT_INDEX   <=  0;
        mSetup_ST   <=  0;
        mI2C_GO     <=  0;
    end
    else
    begin
        if(LUT_INDEX<LUT_SIZE)
        begin
        READY<=0;
            case(mSetup_ST)
            0:  begin
                    mI2C_DATA   <=  {8'h72,LUT_DATA};
                    mI2C_GO     <=  1;
                    mSetup_ST   <=  1;
                end
            1:  begin
                    if(mI2C_END)
                    begin
                        if(!mI2C_ACK)
                        mSetup_ST   <=  2;
                        else
                        mSetup_ST   <=  0;
                        mI2C_GO     <=  0;
                    end
                end
```

```
            2:  begin
                    LUT_INDEX   <=  LUT_INDEX+1;
                    mSetup_ST   <=  0;
                end
            endcase
        end
        else
        begin
          READY<=1;
          if(!HDMI_TX_INT)
          begin
            LUT_INDEX <= 0;
          end
          else
            LUT_INDEX <= LUT_INDEX;
        end
    end
end
////////////////////////////////////////////////////////////////////
/////////////////////   Config Data LUT   //////////////////////////
always
begin
    case(LUT_INDEX)

    //  Video Config Data
    0   :   LUT_DATA    <=  16'h9803;  //Must be set to 0x03
    1   :   LUT_DATA    <=  16'h0100;  //Set 'N' value at 6144
    2   :   LUT_DATA    <=  16'h0218;  //Set 'N' value at 6144
    3   :   LUT_DATA    <=  16'h0300;  //Set 'N' value at 6144
    4   :   LUT_DATA    <=  16'h1470;  // Set Ch count in the channel status
    5   :   LUT_DATA    <=  16'h1520;  //Input 444 (RGB or YCrCb) with
    6   :   LUT_DATA    <=  16'h1630;  //Output format 444, 24-bit input
    7   :   LUT_DATA    <=  16'h1846;  //Disable CSC
    8   :   LUT_DATA    <=  16'h4080;  //General control packet enable
    9   :   LUT_DATA    <=  16'h4110;  //Power down control
    10  :   LUT_DATA    <=  16'h49A8;  //Set dither mode - 12-to-10 bit
    11  :   LUT_DATA    <=  16'h5510;  //Set RGB in AVI infoframe
    12  :   LUT_DATA    <=  16'h5608;  //Set active format aspect
    13  :   LUT_DATA    <=  16'h96F6;  //Set interrup
    14  :   LUT_DATA    <=  16'h7307;  //Info frame Ch count to 8
    15  :   LUT_DATA    <=  16'h761f;  //Set speaker allocation for 8
    16  :   LUT_DATA    <=  16'h9803;  //Must be set to 0x03 for proper
    17  :   LUT_DATA    <=  16'h9902;  //Must be set to Default Value
    18  :   LUT_DATA    <=  16'h9ae0;  //Must be set to 0b1110000
    19  :   LUT_DATA    <=  16'h9c30;  //PLL filter R1 value
```

```
    23  :   LUT_DATA    <=  16'ha504;  //Must be set to Default Value
    24  :   LUT_DATA    <=  16'hab40;  //Must be set to Default Value
    25  :   LUT_DATA    <=  16'haf16;  //Select HDMI mode
    26  :   LUT_DATA    <=  16'hba60;  //No clock delay
    27  :   LUT_DATA    <=  16'hd1ff;  //Must be set to Default Value
    28  :   LUT_DATA    <=  16'hde10;  //Must be set to Default for proper operation
    29  :   LUT_DATA    <=  16'he460;  //Must be set to Default Value
    30  :   LUT_DATA    <=  16'hfa7d;  //Nbr of times to look for good phase

    default:        LUT_DATA    <=  16'h9803;
    endcase
end
////////////////////////////////////////////////////////////////////
endmodule
```

H.3: The I2C_Controller provided by Terasic Inc.

```
module I2C_Controller (
 input  CLOCK,
 input  [23:0]I2C_DATA,
 input  GO,
 input  RESET,
 input  W_R,
     inout  I2C_SDAT,
 output I2C_SCLK,
 output END,
 output ACK
);

wire SDAO ;

assign I2C_SDAT = SDAO?1'bz :0  ;

I2C_WRITE_WDATA  wrd(
   .RESET_N  ( RESET),
 .PT_CK     ( CLOCK),
 .GO        ( GO   ),
 .END_OK    ( END  ),
 .ACK_OK    ( ACK  ),
 .BYTE_NUM ( 2     ),  //2byte
 .SDAI      ( I2C_SDAT ),//IN
 .SDAO      ( SDAO     ),//OUT
 .SCLO      ( I2C_SCLK ),
 .SLAVE_ADDRESS( I2C_DATA[23:16] ),
 .REG_DATA     ( I2C_DATA[15:0]  )
);



endmodule
```

H.4: The I2C_WRITE_WDATA provided by Terasic Inc.

```
module I2C_WRITE_WDATA  (
   input                 RESETN ,
    input                PT_CK,
    input                GO,
    input       [15:0] REG_DATA,
    input       [7:0]    SLAVE_ADDRESS,
    input                SDAI,
    output reg           SDAO,
    output reg           SCLO,
    output reg           END_OK,

    //--for test
    output reg [7:0]     ST ,
    output reg [7:0]     CNT,
    output reg [7:0]     BYTE,
    output reg           ACK_OK,
   input       [7:0]  BYTE_NUM  // 4 : 4 byte
);

//===reg/wire
reg   [8:0]A ;
reg   [7:0]DELY ;

always @( negedge RESET_N or posedge  PT_CK )begin
if (!RESET_N  ) ST <=0;
else
      case (ST)
        0: begin  //start
              SDAO   <=1;
             SCLO   <=1;
             ACK_OK <=0;
             CNT    <=0;
             END_OK <=1;
             BYTE   <=0;
             if (GO) ST  <=30 ; // inital
            end
        1: begin  //start
              ST <=2 ;
               { SDAO,  SCLO } <= 2'b01;
                A <= {SLAVE_ADDRESS ,1'b1 };//WRITE COMMAND
            end
```

```
   2: begin  //start
         ST <=3 ;
          { SDAO,  SCLO } <= 2'b00;
       end

   3: begin
         ST <=4 ;
          { SDAO, A } <= { A ,1'b0 };
       end
   4: begin
         ST <=5 ;
          SCLO <= 1'b1 ;
           CNT <= CNT +1 ;
       end

   5: begin
          SCLO <= 1'b0 ;
          if (CNT==9) begin
                if ( BYTE == BYTE_NUM )  ST <= 6 ;
                 else  begin
                          CNT <=0 ;
                          ST <= 2 ;
    if ( BYTE ==0 ) begin BYTE <=1  ; A <= {REG_DATA[15:8] ,1'b1 }; end
else if ( BYTE ==1 ) begin BYTE <=2  ; A <= {REG_DATA[7:0] ,1'b1 }; end
                         end
                 if (SDAI ) ACK_OK <=1 ;
            end
            else ST <= 2;
       end

   6: begin          //stop
         ST <=7 ;
          { SDAO,  SCLO } <= 2'b00;
    end

   7: begin          //stop
         ST <=8 ;
          { SDAO,  SCLO } <= 2'b01;
    end
   8: begin          //stop
         ST <=9 ;
          { SDAO,  SCLO } <= 2'b11;

    end
```

```
            9:  begin
                  ST      <= 30;
                    SDAO   <=1;
                 SCLO   <=1;
                 CNT    <=0;
                 END_OK <=1;
                 BYTE   <=0;
                 end
            //--- END ---
               30: begin
                if (!GO) ST  <=31;
              end
               31: begin  //
                  END_OK<=0;
                    ACK_OK<=0;
                    ST    <=1;
                end
          endcase
     end

endmodule
```

References

[1] P. Salmon, M. Regan and I. Johnston, "Human error and road transport," Monash University Accident Research Center, Melbourne, Australia, 2005.

[2] Bonneau, Vincent; Yi, Hao;, "Autonomous cars: a big opportunity for European industry," the Digital Transformation Monitor, on behalf of the European Commision, 2017.

[3] J. Chandran and V. Kaul, "$345 billion Autonomous, Connectivity and Electrification (ACE) R&D Spend by key Automakers by 2025," Frost & Sullivan, 14 Jul 2017. [Online]. Available: https://ww2.frost.com/frost-perspectives/345-billion-autonomous-connectivity-and-electrification-ace-rd-spend-key-automakers-2025/.

[4] MathWorks, "Morphological Dilation and Erosion," MathWorks, 1 January 2018. [Online]. Available: https://www.mathworks.com/help/images/morphological-dilation-and-erosion.html. [Accessed 9 April 2018].

[5] OpenCV.org, "Morphological Transformations," 18 December 2015. [Online]. Available: https://docs.opencv.org/3.1.0/d9/d61/tutorial_py_morphological_ops.html.

[6] J. De Schutter, J. De Geeter, T. Lefebvre and H. Bruyninckx, "Kalman Filters: A Tutorial," 29 October 1999. [Online]. Available: http://www.cs.ucf.edu/~mikel/Research/tutorials/kalman-filters-a-tutorial.pdf.

[7] L. Kleeman, "Monash University," 16 July 2007. [Online]. Available: https://ecse.monash.edu/centres/irrc/LKPubs/Kalman.pdf. [Accessed 20 Feburary 2018].

[8] R. Faragher, "Understanding the Basis of the Kalman Filter Via a simple and Intuitive Derivation [Lecture Notes]," *IEEE Signal Processing Magazine,* vol. 29, no. 5, pp. 128-132, 2012.

[9] G. Welch and G. Bishop, "An introduction to the Kalman Filter (Technical Report TR 95-041)," University of North Carolina, Chapel Hill, NC, 2006.

[10] G. G. &. A. Kaehler, Learning OpenCV: Computer Vision with the OpenCV Library, Sebastopol, CA, USA: O'Reilly, 2008.

[11] J. W. Shipman, "Introduction to color theory," 16 October 2012. [Online]. Available: http://infohost.nmt.edu/tcc/help/pubs/colortheory/web/hsv.html.

[12] W. H. O. (WHO), "Road Traffic Injuries," WHO Media Center, January 2018. [Online]. Available: http://www.who.int/mediacentre/factsheets/fs358/en/.

[13] W. Lucetti, "Simple OpenCV Kalman Tracker," 16 November 2016. [Online]. Available: https://github.com/Myzhar/simple-opencv-kalman-tracker.

[14] A. Rosebrock, "Ball Tracking with OpenCV," 14 September 2015. [Online]. Available: https://www.pyimagesearch.com/2015/09/14/ball-tracking-with-opencv/.

Resources Used

- Collection of tutorials and resources on Cyclone V SoC FPGA:
  - Cornell University ECE 5760 website, Dr. Bruce Land, http://people.ece.cornell.edu/land/courses/ece5760/
- OpenCV tutorials in Python:
  - Pyimagesearch website, Adrian Rosebrock, https://www.pyimagesearch.com/category/tutorials/
- OpenCV 3.1.0 documentation and tutorials:
  - Official OpenCV documentation, opencv.org, https://docs.opencv.org/3.1.0/index.html
- Install OpenCV in Linux tutorial:
  - Learnopencv website, Vaibhaw Singh Chandel, https://www.learnopencv.com/install-opencv3-on-ubuntu/
- DE10-Nano CD:
  - Terasic website, Terasic Inc., http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=205&No=1046&PartNo=4

Index

D

Distance Measurement
  Focal Length, 18, 45, 46, 47

H

HPS
  ARM, 8, 10, 24, 38, 63
  AXI, 9, 24, 27, 28, 29, 32, 33
  HDMI, 28, 32, 34, 106
  Memory, 24, 28, 30, 31, 33, 36, 45, 58
  PIO, 24, 29, 36, 37, 38, 56

K

Kalman Filter
  Covariance, 15, 16, 17, 23
  Gaussian Distribution, 15
  Gaussian White Noise, 16
  Prediction, 15, 16, 17
  Process Noise, 16, 23
  Weighted Average, 12, 15

O

Object Detection
  Contours, 22, 40
  Contrast, 21, 55, 60
  HSV, 21, 39
  RGB Representation, 21

S

Servo Control
  Duty Cycle, 19, 47, 48, 49, 51
  PWM, 8, 19, 25, 47, 48, 49, 50, 51, 55, 58, 78, 82, 85
Software
  OpenCV, 8, 9, 21, 27, 38, 39, 40, 60, 117
  Python, 27, 117
  Qsys, 27, 28, 30
  Quartus Prime, 27, 32, 47

T

Thresholding
  Dilation, 14
  Erosion, 14
  Gaussian Function, 12, 13, 17, 18