Ball Plate Balancing and System

Submitted by: Sara Kinghan
Student ID: 17964398

Supervisor: Dr. Mark Beckerleg

School of Engineering, Computer and Mathematical Science

A final year project report presented to the Auckland University of Technology in partial fulfilment of the requirements of the degree of Bachelor of Engineering (Honors)

2021

STATEMENT OF ORIGINALITY

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person (except where explicitly defined in the acknowledgements), nor material which to a substantial extent has been submitted for the award of any other degree or diploma of a university or other institution of higher learning.

Date: October 2021

Sign: Sara Kinghan

ACKNOWLEDGEMENTS

I would like to thank my supervisor Dr Mark Beckerleg for proposing this topic and for their support and guidance over this project. I am grateful to have had the opportunity to explore the many aspects within this project.

I would also like to express gratitude to my colleague, Daniel Dymond, who completed this project with me. This project was only realised due to the combined team effort. Not only was their work instrumental in the developed and implementation of the project but their assistance and support was an integral team aspect.

ABSTRACT

This report presents a ball-balancing system based on the two degree of freedom ball-plate model. The objectives of this project are to develop a system capable of controlling the ball's position both for a static set point and along paths, and to modify an existing ball-plate hardware system.

In this project, a camera and computer vision techniques are used to determine the position of the ball. The position is the input into the PID controller. Its output is sent to the servo motors. As the servo's move, the angle of the plate changes and this causes the ball to roll.

Existing hardware was used for the base of the system, consisting of the plate structure and servo's. The ball detection and PID controller are processed using a Raspberry Pi. A Cyclone IV FPGA controls the servos. UART communication is used to send the angle from the Raspberry Pi to the FPGA. The Raspberry Pi with a seven inch touch screen is mounted onto the existing hardware unit.

The touch screen is for the graphical user interface that was developed to enable users to interact with the system. It displays the live feed from the camera, indicates the ball when it is found in the frame and allows for the selection of different modes of control of the ball. Modes include to balance the ball in the center, to enter target position co-ordinates, have the ball follow a rectangle or circle pattern, and manual control over the servos via a joystick.

The coding language used on the Raspberry Pi is Python 3. OpenCV commands were utilized for colour detection method to detect the ball. PyQt5 was used to create the user interface.

ACRONYMS

DOF	Degrees of freedom
PID	Proportional Integral Derivative
Pi	Raspberry Pi
FPGA	Field Programmable Gate Array
GUI	Graphic User Interface
USB	Universal Serial Bus
UART	Universal Asynchronous Receiver-transmitter
ADC	Analogue-Digital Converter
SPI	Serial Peripheral Interface
HMI	Human Machine Interface
SPAS	Servo Plate Augmentation System
PCB	Printed Circuit Board
CSI	Camera Serial Interface
LED	Light-emitting diode

CONTENTS

Statement of Originality....2
Acknowledgements....3
Abstract....4
Acronyms....5
List of Figures....9
List of Tables....11
1 Introduction....12
1.1 Ball-Plate Systems....12
1.1.1 Ball-Plate Model....12
1.2 Project Aims....13
1.2.1 Existing Hardware....14
1.2.2 Project Scope....14
1.2.3 Overview of tasks....15
2 Literature Review....16
2.1 Ball Balancing Systems....16
3 System Modelling....18
3.1 Overview of the Dynamics....18
3.2 Mathematical Modelling....19
3.2.1 Free body Diagram....20
3.2.2 Deriving the equations....21
3.3 MATLAB PID Design....23
4 Design Process....25
4.1 Selection of variable factors....25
4.1.1 Control Method....25
4.1.2 Processors....25
4.1.3 Sensor....27
4.1.4 Manual servo control....28
4.2 Software Elements....28
4.2.1 Coding language....28
4.2.2 Computer vision library....29
4.2.3 GUI framework....29
Development Process....30

5 Hardware 30
5.1 Mechanical Modifications 30
5.1.1 Human Machine Interface panel 30
5.1.2 Camera mount 31
5.2 Electrical Modifications 32
5.2.1 Joystick PCB 32
5.2.2 FPGA configuration 32
6 Software 33
6.1 Image Processor 34
6.1.1 Determine the Ball's Position 34
6.1.2 Creating a Frame Object 39
6.2 PID controller 40
6.3 SPAS 43
6.4 UART 46
6.5 Director 47
6.6 Frame Collector 47
6.7 Graphical User Interface 48
6.8 Control 50
7 Results 52
7.1 Hardware – Mechanical 53
7.1.1 HMI panel 53
7.1.2 Camera mount 53
7.2 Hardware - Electrical 53
7.2.1 FPGA 53
7.2.2 Joystick 54
7.3 Software 54
7.3.1 Ball detection 54
7.3.2 Control over the ball 57
7.3.3 Graphical user interface 58
8 Conclusions and Future Work 62
Bibliography 63
9 Appendix 66
Appendix A 66

Appendix B 67
Appendix C 69

LIST OF FIGURES

Figure 1: Diagram of the two DOF ball-plate model obtained from (Morales et al., 2017) 12
Figure 2: The existing hardware unit that was provided for use with this project 14
Figure 3: Two equation expression for the position of the ball. X and y respectively. (Morales et al.) 18
Figure 4: Final equation expressions for the ball's position, x and y respectively. (Morales et al.) 19
Figure 5: Free body diagram of the ball beam model 20
Figure 6: Transfer function generated in MATLAB from the input variables 24
Figure 7: Exploded render view of HMI panel, showing how the touch screen and Pi connect to the housing. 31
Figure 8: Render of HMI panel. 31
Figure 9: Block diagram of the Python classes that make up the program. 33
Figure 10: Trackbars for HSV upper and lower values. 35
Figure 11: The binary image produced from applying the mask. The black pixels are the pixels who's value was outside of the set range, and white pixels are the pixels who's value fell within the set range. 36
Figure 12: Bounding box draw around the ball 37
Figure 13: Figure to show the cartesian set up on the plate 38
Figure 14: Figure to show the servo angle and the plate angle. 45
Figure 15: GUI Main Menu screen 49
Figure 16: HMI panel mounted onto the existing hardware 53
Figure 17: Screen shot of GUI with ball placed near to the centre, showing the ball's position, error value and PID output 54
Figure 19: Screen shot of GUI with ball placed in lower right quadrant, showing the ball's position, error value and PID output 55
Figure 18: Screen shot of GUI with ball placed in upper right quadrant, showing the ball's position, error value and PID output 55
Figure 21: Screen shot of GUI with ball placed in the upper left quadrant, showing the ball's position, error value and PID output 56
Figure 20: Screen shot of GUI with ball placed in lower left quadrant, showing the ball's position, error value and PID output 56
Figure 22: Several frames showing the path of the ball being pushed off from the set point and being rolled back to the set point from the plate's angle 57
Figure 23: Main Menu stacked widget 58
Figure 24: Balance at position mode with yellow dot representing a position entered for the upper right quadrant. 59
Figure 25: Balance at position mode with yellow dot representing a position entered for the lower right quadrant. 59
Figure 26: Balance at position mode with yellow dot representing a position entered for the lower left quadrant. 60
Figure 27: Balance at position mode with yellow dot representing a position entered for the upper left quadrant. 60

Figure 28: MATLAB's PIDTuner shows the tuned response and initial PID constants. 66
Figure 29: Dialog box showing the Controller Parameters and performance of the PD controllers current settings. 66
Figure 30: Qt Designer 67
Figure 31: Compilation of the four stacked widget menu screens.....68
Figure 32: Side on view of HMI panel mounted onto hardware 69
Figure 33: Front view of HMI panel with touch screen displaying the GUI.....69
Figure 34: Angled view of HMI panel.....70
Figure 35: Proposed design for camera mount. 71

LIST OF TABLES

Table 1: Table containing the parameters shown on the free body diagram 20

1 INTRODUCTION

This section provides a brief overview of ball-plate models, covering their dynamics, aims, main components and variable factors with reference to the specific parameters used for this project. Following this, is an outline of the project objectives including the scope and tasks.

1.1 BALL-PLATE SYSTEMS

The two degree of freedom (DOF) ball-plate model consists of a plate that can be tilted up and down along two perpendicular directions by two servo motors, one servo on each axis (x, y). The aim of ball balancing systems is to control the movement of the ball, getting it to a set point or to follow a path. Control over the ball is achieved only indirectly, by moving the servo's, which causes the plate tilt which in turn, rolls the ball. A more in-depth overview of ball balancing systems from their literature is covered in the following chapter, Literature Review.

Ball-plate models have two core components and several variable factors, which are discussed in general, in the following subsection, Ball-Plate Model, along with reference to the specific parameters used for this project.

1.1.1 Ball-Plate Model

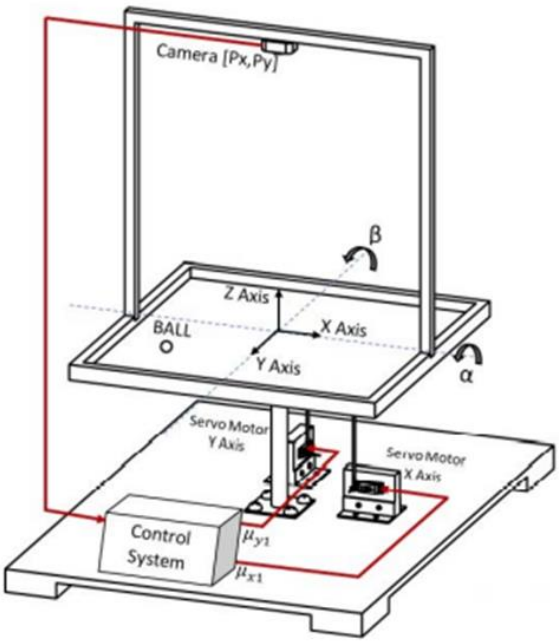

Figure 1: Diagram of the two DOF ball-plate model obtained from (Morales et al., 2017)

The core components of the two degree of freedom ball-plate model are; a plate and two servo motors. The plate's pivot point is at its centre. The arms of each servo are connected to the plate. As can be seen from Figure 1 , the servo's are mounted perpendicular to each other, providing each servo with control over a different axis of the plate. Servo X is

connected to the plate along the plate's x axis, giving control in that directional movement of the plate. Servo Y provides the same but for the y axis. When the servo motors move, the plate tilts up or down, causing the ball to roll.

The variable elements to the system are the following;

- The type of sensor,
- The control method applied,
- The processor (or processors) used to run the system.

The sensor measures the position of the ball. Sensor types used are either resistive touch or a camera. For this project, a camera is used as the sensor.

The measured position of the ball is the input into the control system and the output is the angle required to get the ball to the target position. Note that the output could be the angle for the plate or for the servo. This will depend on how the control system is designed.
This project uses a Proportional-Integral-Derivative (PID) controller.

The processor is the device that runs the sensor, control method and drives the servo motors. A single processor, one for each variable or a combination thereof, could be used. In this project, two processors are used, a Raspberry Pi (Pi) and a Cyclone IV Field-programmable gate-array (FPGA). The Pi is connected to the camera and runs the main code. The FPGA receives the angle from the Pi and drives the servos to that angle.

1.2 Project Aims

The overall aim of this project is to modify an existing ball-plate hardware unit to produce a system capable of controlling the ball's position while fulfilling the specified project constraints.

To produce the system, computer vision techniques are used to detect the ball, a PID controller was developed and applied, a graphical user interface (GUI) was designed and created to offer the user modes of interaction and the existing hardware was modified to meet the requirements. In order to achieve the project aims, the constraints and tasks are identified and outlined below.

Project constraints:

- To use the existing hardware unit,
- To use a camera as the sensor,
- The processor had to be mounted onto the existing hardware,
- Have a method of manual control over the servo's

1.2.1 Existing Hardware

Figure 2: The existing hardware unit that was provided for use with this project

This group was provided with an existing hardware unit shown in Figure 2, a physical ball-plate unit consisting of the base components; a plate and two servo's. The plate has resistive touch capability. Two RJX FS025HV hobby servos are attached to the hardware with their arms connected to the plate. The servos are connected perpendicular to each other, so that each has control of a different axis of the plate's inclination. Housed under the base of the unit is a circuit board which establishes the power supply for all connected components and contains a mounting point for a daughterboard. The female pin headers of the mounting point contain the footprint of a DE0-Nano Cyclone IV FPGA. Connected from the motherboard through the pins of the daughter board mount point, is power supply, an FTDI module, the resistive touch sensors and the data lines of the servo motors. The FTDI module has an external USB-C cable connection. The existing hardware had the FPGA mounted onto the mother board and it was available for use in this project.

1.2.2 Project Scope

The initial project brief was to use the existing hardware unit as is, to not remove any of its existing capabilities or components but to add to it the Raspberry Pi, with touch screen, joystick and camera.

However, early into the project, the scope was expanded, to require the replacement of the circuit boards of the hardware unit. This would essentially be a re-make of the existing circuit boards, as the unit was to retain all the functionality it currently offers –the resistive touch capability of the plate.

The first few months into the project were spent reviewing the circuit boards schematic diagrams, examining the system and planning for the design and manufacture of a new circuit board system. To re-make the circuit boards would mean testing and ensuring the functionality of all the components which included ones that were not used for this project, such as the resistive touch sensors. Over the process of examining the circuit boards and after a few discussions, it was decided between the group members and supervisor that this project would not require the re-production of the circuit boards. Since the existing boards currently offered all of the required functionality and have been used in projects prior, therefore proven to work, it was decided to leave the circuit board as is.

This expansion to the scope was abandoned and the group returned to the original scope of the project. That was to use the existing hardware unit as is, without removal of any current components, but to add to it the Raspberry Pi, touch screen, joystick and camera.

1.2.3 Overview of tasks

The tasks for this project can be separated into two main elements; hardware and software. Each element has its own chapter under the Development Process. An outline of the tasks to be achieved under each section is shown below.

Hardware - Mechanical

- Use the existing ball-plate hardware as the base system,
- Mount the Pi with touchscreen onto the existing hardware,
- Mount the camera,
- Attach the joystick.

Hardware - Electrical

- Set up the FPGA to control the servo motors,
- Establish UART communication between the Pi and FPGA,
- Establish a joystick to have manual control over the servo motors.

Software

- Use image processing to detect the ball within the frame,
- Develop and apply a PID controller,
- Design and create a GUI to be used on the touch screen to offer modes of interaction to the user,
- Design and write the software.

2 LITERATURE REVIEW

This section comprises of a brief overview of the findings from the literature review on ball balancing systems. Several interesting features were discovered from the literature and are covered in this section. Ball-plate models can be expressed as two separate beam models. Balancing systems are commonly used by students to apply and test control methods on, thus various control methods have been applied.

2.1 BALL BALANCING SYSTEMS

Balancing problems come in a variety of systems, "*such as inverted pendulum, double pendulum systems, cart-pole system, ball and beam, ball-plate*" (Morales et al., 2017). Ball balancing systems have two main model variations, the ball-beam and the ball-plate. Ball-plate systems are considered to be an enhanced version of the beam models (Dušek et al., 2017) as beam systems offer only one axis of control whereas plate systems offer two.

However, from the literature reviewed, both ball-beam and ball-plate systems were referred to as having two DOF. Morales et al states that the ball-plate system has two servo motors "*in order to provide two degrees of freedom that allow it to vary its inclination*" (Morales et al., 2017) along two axes. However, sources using beam systems refer to them as having two DOF (Ahmad & Hussain, 2017) (Soni & Sathans, 2018). "*One is the ball rolling up and down the beam, and the other is beam rotating through its central axis*" (Valluru et al., 2016).
A possible reason for this variation was found after reviewing the modelling of ball-plate systems. One of the assumptions applied is that plate systems can be treated as two decoupled ball-beams, hence a ball-plate system can be represented as two independent two DOF ball-beam models. Further exploration of this covered in chapter three, System Modelling.

Ball balancing systems have no direct control over the ball itself. The ball moves due to the angle of the plate, and the plate angle is altered by the servos connected to it. Therefore, influence over the ball is achieved only indirectly through moving the servo's. They are non-linear and open-loop unstable systems (Dušek et al., 2017). Due to the aim of such systems, to control the position of the ball, and also due to the complex dynamics of the system, ball balancing systems are common control problems. Not only used as "*a popular demonstration and benchmark process for different control algorithms*" (Kostamo et al., 2005) but also presented to engineering student to provide an opportunity to apply and test control methods on.

Their non-linear dynamics appeal not only as a platform to apply mathematical modelling and control concepts studied but also for testing more intelligent control methods that aren't based of mathematical models. *"The non-model based approach does not require mathematical operation to acquire the dynamic equations and to employ linearization*" (Maalini et al., 2016).

Some such examples "*of intelligent control are Fuzzy logic, neural network, genetic algorithm and particle swarm optimization*" (Isa et al., 2018).

Variables from other projects:

The sensor and control method variables used in this project are a camera and PID controller. The literature review revealed a number of variables have been applied to the ball balancing problem.

Firstly, only two types of sensors for ball-plate systems are used, camera or resistive touch. For ball beams, typically a distance or proximity sensor is used. From the papers sources for this project, Morales et al., Shi & Liu, Kostamo et al. and Fan at al. use computer vision techniques to determine the balls position. However, that does not mean that other ball-plate projects used resistive touch. Most ball-plate project articles found were simulated in MATLAB, therefore neither sensor type was used.

There are, however, many methods to control the ball plate problem in the literature (Bigharaz et al., 2013) . The purpose of some projects was to compare control methods. Valluru et al.'s paper presents the "*design of different controllers such as PID, state space, lag-lead, robust LQR and observer based LQG controllers for positioning control of ball on the beam*" (Valluru et al., 2016). Shi and Liu in a paper called 'An accuracy indicator of control system and its application in ball&plate system' compares the application of a "*classical LQR algorithm and an improved LQR algorithm*" (Shi & Liu, 2015). Morales et al presents a paper called 'A comparative analysis among different controllers applied to the experimental ball and plate system'. It implements three control methods, "*Proportional Integral Derivative Control (PID), Sliding-Mode Control (SMC) and Fuzzy Control*" (Morales et al., 2017).

Control methods found included neural networks (Bigharaz et al., 2013), Linear quadratic regulator (LQR) (Soni & Sathans, 2018) and (Kostamo et al., 2005), fuzzy logic (Isa et al., 2018), (Fan et al., 2004) and (Han & Liu, 2016) and PID (Ahmad & Hussain, 2017) and (Chen et al., 2012)

3 SYSTEM MODELLING

In designing a control system, the first step was to see if any mathematical models are present in literature that could provide a mathematical expression of the system (Shi & Liu, 2015). As covered in the Literature Review, ball-plate systems are non-linear and unstable. However, the literature revealed that a simplified linear expression can be obtained.

Full modelling of the ball-plate system was not the purpose of this paper and was only explored to gain a sense of the system. Therefore, this section highlights the modelling processing of the ball-plate system, in order to show the assumptions that are applied and provide the final equations found from literature for comparison. Review of the modelling helped identify specific assumptions for this project, which enabled Lagrange kinetic equations to be applied to a free body diagram to derive the final equations. The Mathematical Modelling subsection covers these steps.

As a PID controller is used, the final subsection, MATLAB PID Design, outlines the steps taken to generate the transfer function in MATLAB and determine initial values for the PID constants.

3.1 OVERVIEW OF THE DYNAMICS

To obtain the dynamic equation of the ball-plate system, a Lagrange method is often used. The derivation starts with the general form of the Euler-Lagrange equation (Spacek et al., 2017) which is shown below.

$$\frac{d}{dt}\frac{\partial T}{\partial qi} - \frac{\partial T}{\partial qi} + \frac{\partial V}{\partial qi} = Qi$$

(Equation sourced from (Spacek et al., 2017))

"*Where T is kinetic energy of the system, V is potential energy, Qi is the generalized force and qi is the general coordinate*" (Spacek et al., 2017).

$$\left(m_b + \frac{J_b}{{r_b}^2}\right)\ddot{x} - m_b x\dot{\alpha}^2 - m_b y\dot{\alpha}\dot{\beta} + m_b g\, sin(\alpha) = 0$$

$$\left(m_b + \frac{J_b}{{r_b}^2}\right)\ddot{y} - m_b y\dot{\beta}^2 - m_b x\dot{\alpha}\dot{\beta} + m_b g\, sin(\beta) = 0$$

Figure 3: Two equation expression for the position of the ball. X and y respectively. (Morales et al.)

From modelling, four equations are produced. The process from the general Euler-Lagrange equation to the four resulting equations is not be shown here, however the ongoing steps to linearize that system and produce the derived equation are shown.

From the four equations, two are for the ball position coordinates (x and y), shown in Figure 3, and two are for the plate axes, not shown. These equations reveal the system to be a

"*multivariate, nonlinear complex system*" (Han & Liu, 2016). A set of assumptions are applied in order to obtain a simplified linear expression.

First, the four initial equations are reduced down to two. This is achieved by applying two assumptions, that there is no slip in the ball and no friction (He & Chen, 2020). This eliminates the last two equations, leaving the two shown above in Figure 3.

$$\ddot{x} = -\frac{m_b g \sin(\alpha)}{\left(m_b + \frac{J_b}{r_b{}^2}\right)}$$

$$\ddot{y} = -\frac{m_b g \sin(\beta)}{\left(m_b + \frac{J_b}{r_b{}^2}\right)}$$

Figure 4: Final equation expressions for the ball's position, x and y respectively. (Morales et al.)

The next stage in linearization is decoupling the system. This treats a two servo ball-plate system as two independent ball-beam models, where the servo in the X axis controls the x-axis of the plate and, it is assumed, only the x-coordinate of the ball's position. The same for the y-axis. When the system is decoupled, that eliminates the two middle components from the equations. The final result is shown in Figure 4. This can be further simplified by applying the small angle approximation assumption. This will be shown in the next section.

This is a brief overview of the modelling found from literature. It identified the assumptions that are applied to linearize the system and enabled a free boy diagram of this project's system to be developed and equations derived.

3.2 Mathematical Modelling

A review of the mathematical modelling of the system from literature was undertaken. This revealed a number of assumptions.

The following assumptions were applied to this model:

- System can be treated as decoupled,
- No loss of contact between ball and plate,
- No slip in the ball,
- No friction,
- The servo motor dynamics are ignored,
- The angle of the plate will be very small,
- The ball is hollow.

With these assumption applied, a simplified free-body diagram can be produced. This section shows how the free-body diagram can be used to derive the equations shown in Figure 4 and generate the transfer function.

3.2.1 Free body Diagram

The first assumption is that the system is decoupled. The ball-plate hardware used for this project has two identical servo motors. The servo arms are the same length, as are the lever arms. The connection point of each lever arm to the plate is the same distance from the pivot point. The system is symmetrical and the dynamic characteristics are the same. Therefore, not only can the system be decoupled, but the x axis and y axis modelling is the same. Only one axis needs to be modelled as the x-axis model can be used to represent the y axis model. A free body diagram representation of a single axis (x axis) was produced with the parameters and forces shown. This is displayed in Figure 5. The parameters are shown in Table 1 below. Table containing the parameters shown on the free body diagram

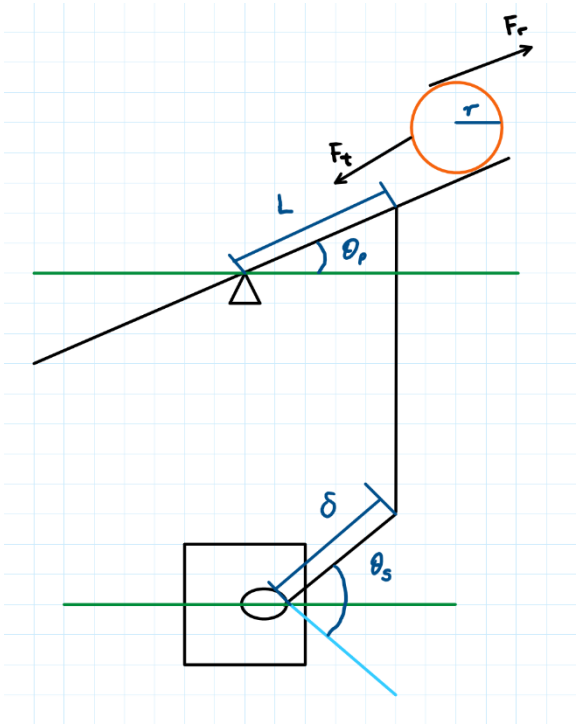

Figure 5: Free body diagram of the ball beam model

Table 1: Table containing the parameters shown on the free body diagram

m	Mass of ball
r	Radius of ball
x	Position of ball
L	Length from pivot point to lever arm point
δ	Off set of servo arm
θ_p	Angle of plate
θ_s	Angle of servo

3.2.2 Deriving the equations

From the free body diagram, a Lagrangian kinetic equation was used to generate equations of motion with four assumptions applied; that there is no friction, the ball and plate do not lose contact, the ball has no slip and the servo motor dynamics are ignored.

Equations of motion:

The mass of the ball multiplied by acceleration can be equated to the sum of all of the forces acting on the ball.

$$\sum F = m\ddot{x}$$

Equation 1

The forces acting on the ball are the force due to gravity and the force due to rotation.
F_t is the translational force due to gravity from plate inclination.
F_r is the rotational force due to inertia.

Therefore, the equation for the sum of the forces in the free body diagram is expressed in Equation 2 below.

$$m\ddot{x} = F_t - F_r$$

Equation 2

From Equation 2, m and $\ddot{x}$ represent the mass of the ball and the ball's acceleration, respectively.

Translational force:

$$F_t = mg\sin(\theta_p)$$

F_t is the translational force due to gravity from the inclination of the plate. g is the gravity constant and θ_p is the angle of the plate. The assumption made, is that the angle will be small, generally less than 15° and small angle approximation is applied. The sine component is dis-regarded (Chen et al., 2012). Therefore, $\sin(\theta_p) \cong \theta_p$ resulting in the expression of F_t shown in Equation 3.

$$F_t = mg\theta_p$$

Equation 3

Rotational force:

$$F_r = \frac{T}{r}$$

The rotational force is due to the torque produced from the rotational acceleration of the ball. The torque component is produced by the moment of inertia of the ball and its angular velocity.

T is the torque of the ball, and is represented as, $T = J\,\dot{\omega}$

J represents the moment of inertia of the ball, and $\dot{\omega}$ represents the angular velocity of the ball. The angular velocity is expressed as, $\dot{\omega} = \frac{\ddot{x}}{r}$.

Therefore, $T = \frac{J}{r}\ddot{x}$. This produces the expression for the rotational force shown in Equation 4.

$$F_r = \frac{T}{r} = \frac{J}{r^2}\,\ddot{x}$$

Equation 4

With expressions for F_t and F_r shown in Equation 3 and Equation 4 respectively, they can be substituted into Equation 2. This produces the following result.

$$m\ddot{x} = mg\theta_p - \frac{J}{r^2}\,\ddot{x}$$

This is then rearranged to provide an expression for acceleration of the ball ($\ddot{x}$) :

$$\ddot{x}\left(\frac{J}{r^2} + m\right) = mg\theta_p$$

$$\ddot{x} = \frac{mg\theta_p}{\frac{J}{r^2} + m}$$

Equation 5

Figure 4 shows the final equations produced from across literature, one for each axis. The x axis equation is shown again as Equation 6 below, to compare with Equation 5, the resulting equation obtaining from the derivation performed for this project's system. As can been seen, with the small angle approximation assumption applied, these equations are the same.

$$\left(m_b + \frac{J_b}{{r_b}^2}\right)\ddot{x} + m_b g \sin(\alpha)$$

Equation 6

<u>Ball's moment of inertia:</u>

Where the resulting equation can appear different, is due the expression used to represent the ball's moment of inertia as it varies depending on whether a solid ball or hollow ball are used. The moment of inertia of the ball, J , is expressed as,

$J = \frac{2}{5}\,m\,r^2$ for a solid ball

$J = \frac{2}{3}\,m\,r^2$ for a hollow ball.

This project uses a hollow ball, (ping pong) therefore the second expression will be used and is substituted into Equation 5Equation 6. It is then simplified down to produce Equation 7.

$$\ddot{x} = \frac{mg\theta_p}{\frac{\frac{2}{3}\,m\,r^2}{r^2} + m}$$

$$\ddot{x} = \frac{mg\theta_p}{\frac{2}{3}m + m}$$

$$\ddot{x} = \frac{3}{5}\,g\theta_p$$

Equation 7

Then the transfer function can be generated with the Laplace transform. At this point, literature takes different paths depending on the what the focus is for their project and the controller being applied.

$$G(s) = \frac{P(s)}{\theta_p(s)} = \frac{mg}{\frac{J}{r^2} + m}\,\frac{1}{s^2}\,\left[\frac{m}{rad}\right]$$

Equation 8

$P(s)$ is the ball position and $\theta_p(s)$ is the angle of the plate. Equation 8 is the transfer function generated for this project and is the one that is entered into MATLAB to obtain PID constant factors.

Note, that $\theta_p(s)$ is the plate angle, not the angle for servo. The mathematical expression for the relationship between the angel of the plate and the angel of the servo for the ball-plate hardware used in this project is shown in Equation 9 and can be used to translate the plate angle to the angle for the servo. This was not included into the transfer function but could be applied to the PID output with code if required. However, the control system was manually tunned, therefore it was not necessary.

$$\theta_s = \frac{\delta}{L}\,\theta_p$$

Equation 9

3.3 MATLAB PID DESIGN

In the section above, the transfer function for the ball-plate system was derived and is shown below.

$$G(s) = \frac{P(s)}{\theta_p(s)} = \frac{mg}{\frac{J}{r^2} + m}\,\frac{1}{s^2}\,\left[\frac{m}{rad}\right]$$

$P(s)$ is the position of the ball and $\theta_p(s)$ is the angle of the plate.

MATLAB has a PIDTuner, which takes in the transfer function and the type of PID (PI, PD or PID) and opens up a user interface to tune the system, showing a plot of the stepped behaviour. From this, the initial values for the PID constants are obtained.

First, the values for the variables are entered and then the transfer function generated. The code to do this in MATLAB is shown below.

```
%%  Setup system constant values
m = 0.002;  % mass of ping pong ball [kg]
r = 0.037/2;  % radius of ping pong ball [m]
g = -9.8; % gravitational acceleration [m/sec^2]
J = (2/3) * m * (r^2); % ball moment of inertia

%%  Tranfer function of Plant
s = tf('s');
Gs =(-m*g/(J/r^2+m))* (1/s^2)
```

```
Gs =

  5.88
  ----
  s^2

Continuous-time transfer function.
```

Figure 6: Transfer function generated in MATLAB from the input variables

Once these are entered into MATLAB, the transfer function is then generated using the input values. The output is shown in Figure 6. Now that that transfer function has been generated in MATLAB, PIDTuner can be run. The following command calls the PIDTuner in MATLAB. It takes in the transfer function and the type of PID.

```
pidTuner(Gs,'PD')
```

Figure 28 in Appendix A is a screen shot of MATLAB's PIDTuner, showing the step response of the transfer function with a PD controller. The two sliders at the top can be adjusted to produce the desired system response. When "Show Parameters" is clicked, a dialog box opens up, to display the controllers parameters and details of its performance. This is shown in Appendix A as Figure 29. This is how the initial values for kP, and kD were determined, to use as a start point of tunning the system.

4 DESIGN PROCESS

Ball-plate systems have three variable factors to be selected. In addition, there are specific constraints for this project. This section will cover the options considered for this project and the resulting selections made. This is separated into two parts; selection of the variable factors, and software elements.

Specific constraints for this project:

- To use a camera as the sensor,
- The processor had to be mounted onto existing hardware,
- Have a method of manual control over the servo's .

4.1 SELECTION OF VARIABLE FACTORS

The variable factors of ball-plate systems are:

- Control method
- Sensor type
- Processor

4.1.1 Control Method

The first factor considered was the control method to be used. A review of literature revealed a vast number of methods that had been applied to ball balancing projects. An overview of these can be found in the Literature Review section.

While there are a number of control methods that could have been used, the aim of this project is not to use the ball-plate model as a control problem. The primary aim of this project was to produce a complete ball-plate system that demonstrated some control over the ball.

A Proportional Integral Derivative (PID) controller was selected for this project. This control method is simple in concept and therefore user-friendly for future project groups to pick up from and adapt further. PID's a straight forward in design and implementation and literature suggested it would be able to achieve some control over the ball.

4.1.2 Processors

From the literature review the primary processor used was found to be a computer. As the ball-plate system is typically used as a platform for designing and applying control methods, often a physical system is not used but simulated instead. Chen et al, Dusek et al., Martinez and Ruiz, Soni and Sathans, and Isa et al. all used a computer or laptop and MATLAB's Simulink to simulate the system and apply their control method. Morales et al. also used a laptop and MATLAB's Simulink but connected to a physical ball-plate system. However, one constraint in this project was that the processor had to be mounted onto the existing hardware. Therefore, a computer or laptop were not possible options.

Another point of consideration was the motherboard on the existing hardware has a daughter board mounting point supporting the footprint of a Cyclone IV FPGA. This mounting point connects the FPGA to the data lines of each servo and to an external USB-C port through an FTDI module. The Cyclone IV FPGA was available for use for this project if desired.

4.1.2.1 FPGA

First considered was whether the FPGA would be used. The daughter board point offers connection to the data lines of the servos. Therefore it made sense to utilise this functionality to connect to the servo motors. UART communication is available from an external USB-C port through the mother board to the FPGA. This enables another device to send data to the FPGA.

Since the footprint is available, another daughter board could have been designed instead of using the FPGA. This was considered by the project group, as FPGA's have a higher barrier in ease of programming than microcontrollers have. However, the FPGA was already available.

It was decide to use the FPGA to drive the servo's. This minimised the learning curve in configuring an FPGA since it would be receiving data and generating a PWM and not running more complex systems.

4.1.2.2 Raspberry Pi

Another processor was needed to run the image processing and the controller. It was also to be mounted onto the hardware. This meant that a desktop PC could not be used as it is too large.

"*The Raspberry Pi is a low cost, credit-card sized computer*" ('What Is a Raspberry Pi?', n.d.). They have powerful processors, a number of USB ports and GPIO pins. As a pocket sized computer, they offer an ideal alternative to a desktop PC.

A Pi was selected as the main processor. A Raspberry Pi 4 Model B with 4GB ram was loaned from the University to use until one was purchased. A Pi of the same model and specifications was to be purchased for this project as it would be mounted on and become a part of the complete hardware. However, there has been a shortage of many electronic items, limited stock in store and supply chain issues worldwide due to the impact of Covid-19. A Raspberry Pi 3 Model B+ with 1GB ram was purchased. It arrived during lockdown and wasn't fitted to the hardware until after the completion of this paper.

4.1.2.3 Touch screen

A screen is required as a way of interacting with the software that was developed. It needed to be mountable onto the hardware, and therefore had to be small. Since a Pi was selected as the processor, Pi compatible products were looked at. Raspberry Pi brand has a 7-inch LCD touch screen designed for use with the Raspberry Pi.

This screen size was compact enough to fit un-obtrusively onto the hardware but also large enough to display frames from the camera and an interactive panel for users.

4.1.3 Sensor

The sensor is a camera. Once the processor was chosen, then a suitable camera for the processor could be selected. While looking at Raspberry Pi compatible devices, the Pi camera was found.

4.1.3.1 Pi camera

The Raspberry Pi camera module is a small, low-cost but capable camera, supporting 1080p at 30 fps (*Raspberry Pi Camera Board V2*, n.d.). It is compatible with the Pi and connects to it through the Pi's CSI interface, which is "*designed especially for interfacing to cameras*" (*Raspberry Pi Camera Board V2*, n.d.). To connect into the CSI interface, a ribbon cable is needed. A short one, around 15cm and a longer one, 1m are available. The distance from the camera to the Pi was initially calculated to be approximately 1m, since would need the actual camera to know its field of view and determine its optimal height. It would be possible that 1m might not be enough. Even if 1m was long enough, there was also the concern over transmitting the frames via a ribbon cable of 1m length.

Since the Pi does have 4 USB ports, USB webcams were looked into.

4.1.3.2 USB webcam

One main concern for a USB webcam, was that many webcams have been reported as to not work on Linux devices. Therefore, possible webcam options were limited to those that stated Linux supported. Webcams are typically used for video meetings which are more in demand now with people working from home. Due to this most webcams have strong LEDs to provide good lighting of the subjects being filmed. These lights, if adjustable, can usually only be adjusted through the company's software, which would likely not be supported on Linux. Light impacts the effectiveness of image processing especially when the detection method is colour based. A webcam without these lights would be highly preferred.
The A4TECH FullHD 1080P webcam was selected as the camera.

4.1.3.3 Camera mount

There are a number of tripods systems available to hold a camera steady in a set position. However an integrated camera mount was preferred for three reasons. The first two reasons relate to determining the ball's position.

The calculation of the pixel metric is dependent on the height of the camera. The pixel metric is how the coordinates of the ball are converted from pixels into meters. Further details of the pixel metric is covered in the Software chapter. While the height doesn't change the total number of pixels of a frame, it does change the size (in pixels) of items within the frame, thereby altering the pixel metric. If the camera is at a different height each time it is used, the pixel metric needs to be re-calculated and entered into the program.

Secondly, OpenCV identifies pixels starting with 0,0 at the top left corner. However, in order to capture the entire plate, the camera viewing area includes space around the plate as well. For ease of calculation, a centre based approach was chosen. The camera was mounted so that centre of its frame falls directly over the centre of the plate. The plate is represented in

the cartesian system with the centre as 0,0 and the ball's position is expressed relative to the centre of the plate.

Finally, this project had the intention to produce a complete contained system. A separate mount system could easily be lost, leaving those that wish to use it searching for device to hold the camera.

4.1.4 Manual servo control

There is a requirement to have some method to give manual control over the servo motors. Initially, using the touch screen was considered, by having a virtual pad or joystick on the GUI. However, it was discussed that using a touch screen method of control over motors wasn't ideal for users. Therefore, an Adafruit 2-axis analogue joystick was selected. It is small in size so it can be mounted onto the hardware, but robust enough for use. Images of the joystick can be seen in Figure 8 and Figure 7 in the Hardware chapter.

4.2 Software Elements

The software elements to be selected are:

- Coding language
- Computer vision library
- Graphical user interface framework

4.2.1 Coding language

The Raspbian OS comes with several languages pre-installed; Python 2, Python 3, C/C++ and Scratch.

Python 3

"*A large community supports and uses the Python language as it is easy to learn and code. And so, it is one of the most popular and hosted programming languages*" (Ward, 2017).

In research into coding on the Raspberry Pi and using OpenCV for colour detection, much of the code, forums, and documentation used Python 3. While Python 3 was a new coding language to both group members, the resources available and inspiration seen from what others were achieving using Python 3 with a Pi and OpenCV, encouraged the decision to use Python 3. While is it not the most efficient language option, the influencing factors were the shorter lines of code likely required to produce the program, likelihood the language more approachable for a wider number of potential future users of the program and resources found that aided in the development of the programs functions, such as the colour detection and GUI design.

4.2.2 Computer vision library

OpenCV

"*OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library*" ('About', n.d.). It has interfaces for a range of languages including Python and is supported on Linux. "OpenCV was built to provide a common infrastructure for computer vision applications" ('About', n.d.). It includes a vast library of pre-made algorithms. Simply, OpenCV was for image processing, and to make it accessible to people. It's popularity speaks to it ease of use, quality of performance and results in a vast number of resources to turn to. OpenCV algorithms were utilised for the detection of the ball.

4.2.3 GUI framework

Python has a number of GUI frameworks and toolkits available. The Python language was new to both group members, therefore when researching possible frameworks to use, a factor considered was the popularity of the framework for the additional support and documentation that is available.

Tkinter was considered as it offered several advantages. It was already installed, as Tkinter is bundled with Python. It is open-source, there are many resources available and it has an "established and active community"(*Qt Designer and Python: Build Your GUI Applications Faster – Real Python*, n.d.)

Qt is a very popular and powerful "*C++ cross-platform application framework … offering a large collection of GUI widgets*" (*Riverbank Computing | Introduction*, n.d.). Qt also has a graphical user interface designer, Qt Designer. This is a drag and drop design tool for making GUI's quickly without having to write the code for the design. Qt Designer produces is the .ui file of the description of the GUI.

PyQt was selected. PyQt is a GUI widgets toolkit which combines Qt and Python to provide a Python UI framework (*PyQt - Python Wiki*, n.d.). On installation of PyQt, comes Qt Designer. PyQt is able to generate Python code from Qt Designer, it can convert the Qt Designer generates .ui script into the Python script. This meant that GUI would not have to be hand coded. This was the main reason that PyQt was selected. It enabled a GUI to be quickly created and any future adjustments can easily be done.

DEVELOPMENT PROCESS

This section contains two main parts; hardware and software. Under each part, is listed the tasks to be achieved and then a walk-through of the process undertaken to complete them.

5 HARDWARE

The existing hardware was to be used as the base of the system and modified to fulfil the requirements of this project. This section is split into the mechanical and electrical hardware elements. Below is a review of the tasks to be achieved, followed with the mechanical and electrical modifications made.

Overview of Hardware tasks:

Mechanical

- Use the existing ball-plate hardware as the base system.
- Mount the Pi with touchscreen onto the existing hardware.
- Mount the camera.
- Mount the joystick.

Electrical

- Establish a joystick to have manual control over the servo motors.
- Set up the FPGA to control the servo motors.
- Establish UART communication between the Pi and FPGA.

5.1 MECHANICAL MODIFICATIONS

The design, development and implementation of the mechanical modifications was the work of the other member of the group. Full details can be found in the report of Daniel Dymond. Provided in this report is a brief overview of steps taken.

5.1.1 Human Machine Interface panel

The existing hardware unit had a tablet mounted onto it. That tablet did not work and was removed, but its mounts were left. These mounts were kept in mind for use during the design process for the housing and mounting of the Pi with touch screen.

Several design iterations took place, with various CAD renderings produced for consideration by the project members. The final design is a Human Machine Interface (HMI) panel to house the touch screen with Pi attached and the joystick. For production, the HMI panel was 3D printed.

Figure 8: Render of HMI panel.

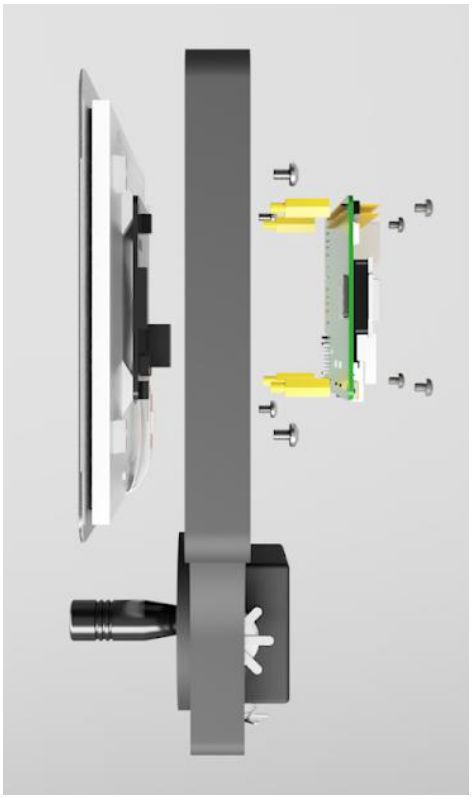

Figure 7: Exploded render view of HMI panel, showing how the touch screen and Pi connect to the housing.

Figure 7 is an exploded view of the CAD render, showing how the touch screen and Pi can be fitted into the panel. Figure 8 is provided to show the final CAD design produced for the HMI panel.

5.1.2 Camera mount

Design and construction of the camera mount did not start until the webcam selected for this project arrived. The webcam that would be used was needed in order to determine the optimal height to mount it at. The field of view and minimum focal distance had to be considered in determining the height and those factors depend on the camera itself.

To determine the height to ensure that the entire plate apparatus would be visible in the frame and the camera could still focus, the webcam was activated and held above the ball-plate hardware unit.

For the structure of the mount, PVC piping was considered. It is low-cost and readily available at hardware stores. Piping and a 90° elbow was purchased. The piping was cut into two pieces, one for the height of the mount and another for the width over the plate.

This prototype was to be tested to see if it was solid enough to stably support the camera while connected to the ball-plate hardware unit with the servo's and plate moving. Testing

and further development of the camera mount was halted due to the Covid-19 August 2021 lockdown.

5.2 Electrical Modifications

The design, development and implementation of the electrical modifications was the work of the other member of the group. Full details can be found in the report of Daniel Dymond. Provided in this report is a brief overview of steps taken.

5.2.1 Joystick PCB

The joystick selected is analogue and is housed in the HMI panel. The Pi is to interpret the joystick input and then transmit the angle to the FPGA. A circuit board was designed to enable the joystick to interface with the Raspberry Pi, to power the joystick from the Pi and to send the signal of the joystick's position. Serial Peripheral Interface (SPI) communication will be used to transmit the angle, as the GPIO pins on the Pi support SPI. However, the Pi doesn't have an Analogue-Digital Converter (ADC), so an ADC module is needed on the PCB.

The components were selected and then the schematic and PCB were designed. The PCB was sent to be printed and components ordered. The PCB board arrived during the August 2021 lockdown, but several of the components have not yet arrived.

5.2.2 FPGA configuration

Verilog hardware was implemented on the DE0-Nano Cyclone IV FPGA to convert the received serial data and generate the PWM for each servo.

The Pi transmits an 8-bit value for each servo at 9600 baud rate. One of those bits is an identifier to indicate which servo. The FPGA hardware generates the clock for the UART, receives the data from the Pi, interprets it and then generates the PWM for each servo.

To achieve this, four modules were designed and implemented onto the FPGA:

- UART clock generation
- servo clock generation
- data interpretation
- PWM generation

Comprehensive details of this process is covered in Daniel Dymond's report.

6 SOFTWARE

Covered in this section is walk-through of the Python program created. Below is a list of the software tasks to be achieved. Following that, a brief description of the completed software is covered in order to provide an overview of the program prior to going in depth over the steps taken to complete the tasks and produce the final program.

Overview of tasks

- Use image processing to detect the ball within the frame.
- Develop and apply a PID controller.
- Design and create a GUI to be used on the touch screen to offer modes of interaction to the user.
- Design and write the software.

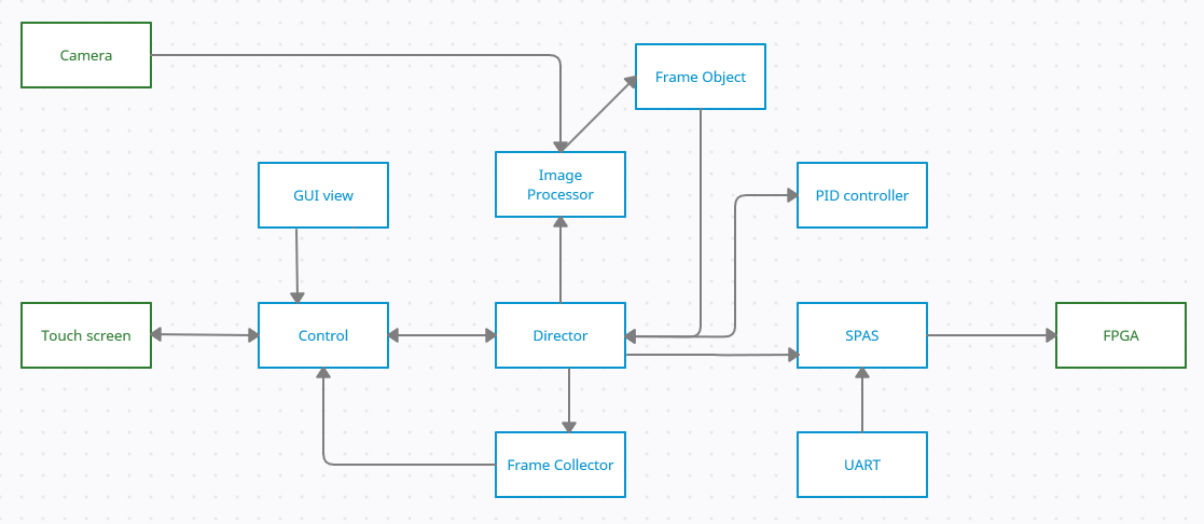

Figure 9: Block diagram of the Python classes that make up the program.

Figure 9 is a block diagram showing the nine Python classes that make up the program.
The Control script initiates the Director and runs the graphical user interface. When the Director object is initiated, it starts the Image Processor, PID controller and SPAS and controls the flow of information.
The Image Processor class activates the camera and fetches each frame and applies a ball detection method to find the ball within the frame and determine its position. If a ball was found, the Director passes the ball's position to the PID controller.
The PID controller class takes the ball's position and desired set point to calculate the error which is used in the calculations of the P, and D components. Each servo is treated as independent, so the x co-ordinate and y co-ordinate are each sent through different instances of the PID controller, to give an output value for each servo.
When the output angle is returned, the Director sends it to SPAS. This is the servo plate augmentation system. It was made to apply limits and checks on the output from the PID, before it is sent to the servo motors. SPAS then calls the UART class. This class converts the angle into binary and writes it to the UART TX to send the data to the FPGA.

6.1 IMAGE PROCESSOR

The Image Processor script has two main functions:

- Determine the ball's position
- Create a Frame Object

To determine the ball's position, the camera is activated and each frame is fetched from the video feed. A colour detection method to isolate the ball from within the frame is applied. The ball's co-ordinates are then determined in pixels and in meters.

The Frame Object class is then called to create an object called 'Frame Object'. This object contains the frame itself and information about the ball; whether a ball was found, its co-ordinates in pixels and in meters. It is passed into a queue to be accessed by the Director class.

The Image Processor is a thread object. The Director creates an instance of the Image Processor and starts the thread.

```
# Initialise the Image Processor Thread
self.imgQueue = Queue()
# self.imgProc = ImageProcessor.ImageProcessor(cameraID, self.imgQueue, False)
self.imgProc = ImgProcess(cameraID, self.imgQueue, verbose)
self.imgProc.start()
```

While the Director is running, it will check a queue for the Frame Object. When the queue has an item, the Director will fetch the Frame Object. If a ball was found, it passes the balls position to the PID controller.

Below walks-through the steps taken to get the Image Processor to perform its functions.

6.1.1 Determine the Ball's Position

Steps to determine the ball's position:

- Get each frame from the camera
- Convert the frame from BGR to HSV
- Apply the colour mask
- Search for contours
- Check the area of the contours
- Draw a box around the contour
- Get the centre point of the box
- Adjust this to the centre of the grid
- Apply pixel metric

Getting the frame:

OpenCV has a class called 'videocapture' which takes in the ID of the camera and creates a videocapture object for the camera connected to that ID. This is done on the initialisation of Image Processor.

```
self.cap = cv.VideoCapture(cameraID, cv.CAP_DSHOW)
```

The 'read()' command, reads the image from the videocapture object. If the 'read' function is placed in a loop, it will read each frame from the camera.

```
ret, frame = self.cap.read()
```

Convert to HSV:

OpenCV reads an image in the BGR colour space, which is a format of the RGB colour space, just with the three parameters in the reverse order. The HSV colour space is used for colour detection processes, instead of the BGR colour space (Dayala, 2020). The code below is used to convert the image from the BGR to the HSV colour space.

```
hsv = cv.cvtColor(img, cv.COLOR_BGR2HSV)
```

Mask:

The next step is to apply a mask over the image. For this, the OpenCV 'inRange()' function is used. It accepts an input image and the lower and upper limits of the colour to be searched for – the colour of the ping pong ball (orange). When applied, the function will compare every pixel to the upper and lower limits and return a binary image. The black pixels represents pixels that are outside of the applied range, and the white pixels represent pixels that are within the applied range.

Before the mask can be applied, the colour range values have to determined. In OpenCV the number ranges for each of the three HSV parameters are 0-179, 0-255 and 0-255 respectively (Dayala, 2020). Saturation and Value will vary depending on the lighting condition. However, Hue has a set of ranges for six basic colours. The hue value of orange falls between 5 and 25 on the HSV colour map (Dayala, 2020).

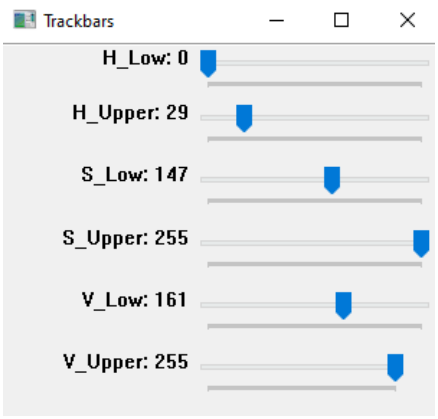

Figure 10: Trackbars for HSV upper and lower values.

In testing and development, a script was created to determine the optimal upper and lower HSV values. This script displays the frame, the mask image and six trackbars. There are two

track bars for each HSV component, one for the lower values of H, S and V, and one for the upper values. The trackbars are shown in Figure 10. The trackbars are adjusted until the optimal range is found. These numbers are then recorded and then entered into the Image Processor script.

```
lowOrange = np.array([ 5, 147, 161])
uppOrange = np.array([ 25, 255, 255])
```

Once the upper and lower boundary arrays are determined, then the mask can be applied.

```
mask = cv.inRange(hsv, self.lowOrange, self.uppOrange)
```

The mask goes pixel by pixel and compares the HSV value of each pixel to the upper and lower boundary values. For pixels that fall within the range of the boundary values, they will be converted to white, (255). Any pixel outside of the range will be converted to black, (0). This creates a binary image.

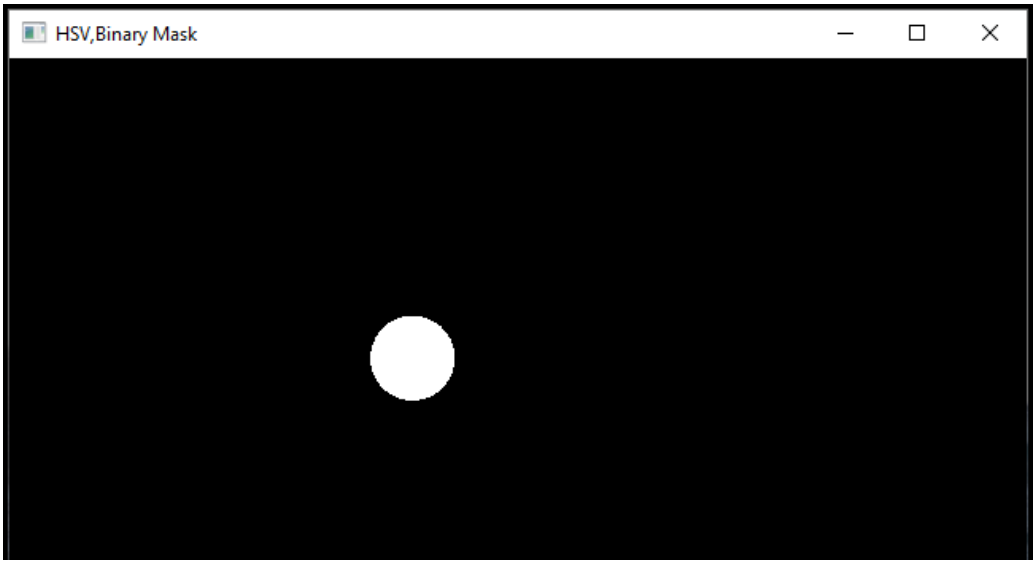

Figure 11: The binary image produced from applying the mask. The black pixels are the pixels who's value was outside of the set range, and white pixels are the pixels who's value fell within the set range.

Figure 11 shows the binary image produced from applying the mask. The shape of the ball can clearly be seen.

Contours:

'findContours()' is an OpenCV function that searches an image for a closed border of pixels that have the same intensity. It finds shapes. It accepts a binary image, and assumes that the white pixels represent the object to be detected. As it searches, each contour found is stored into an array. Also, each contour found is an array, containing the co-ordinates for every pixel that makes up the outline of the shape.

```
circFind, _ = cv.findContours(mask, cv.RETR_TREE, cv.CHAIN_APPROX_NONE)
```

If no contours are found, then a variable ‘nContours’ is set to zero. This indicates to Image Processor that no ball has been found, and sets up the return values to indicate that. ballFound will be False and the ball co-ordinates are set to zero.

```
if nContours == 0:
            BP_x = 0
            BP_y = 0
            ball_x = 0
            ball_y = 0
ballFound = (nContours == 1)
```

Check the area of the contours:

While the mask does identify the ball, there can still be some additional shapes found in the contour search. Such as the caps on the edges of the plate as they are orange. To eliminate these additional contours, the area of each contour is calculated and then compared to a range. The range consists of the upper and lower pixel area size of the ping pong ball.
To find the area of a contour, the OpenCV function ‘contourArea()’ is used.

```
circArea = cv.contourArea(contour)
```

It returns the area of the contour. Using the “Image Calibration” test script, the mask was applied and the area of all contours found was printed into the terminal. With optimal lighting, so that the ball was the only contour found, the upper and lower area ranges of the ping pong as it moves over the tilting plate were obtained.

```
lowArea = 1000
uppArea = 3000
if circArea >= self.lowArea and circArea <= self.uppArea:
```

This range was used to compare to each contour found so that only contours within that range where used. Given the performance of the mask as shown in Figure 11, there was confidence that the contour being used, is the contour representing the ball.

Draw a box around the contour:

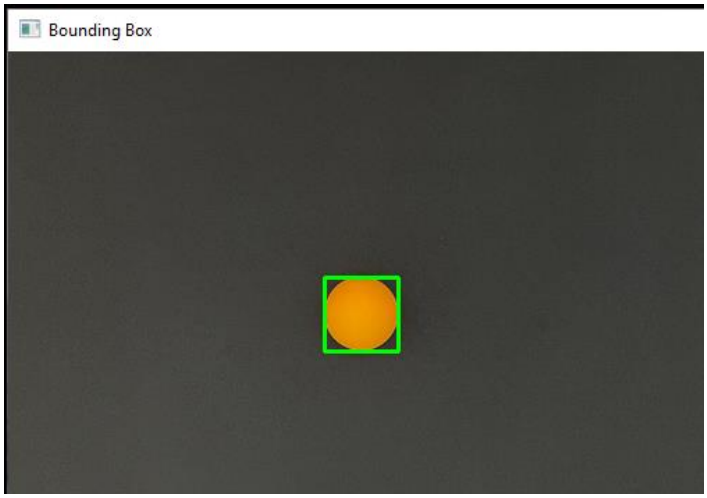

Figure 12: Bounding box draw around the ball

With that contour, a bounding box is placed around it. To create a bounding box, OpenCV has a function 'boundingRect' where the contour can be put straight into it. The bounding box returns the x and y of the upper left corner, and the number of pixels in the width and height.

```
x, y, w, h = cv.boundingRect(contour) #Draw bounding rectangle
```

The command to create the bounding box, doesn't draw the box onto the frame. Drawing the box onto the frame, as shown in Figure 12 was done just to show the box formed around the ball. It is not the drawing of box that is needed, it is the returned values from the 'boundingRect' function. Using the returned values, the center of the bounding box can be found, which is the center of the ball. This is done by taking the x pixel coordinate (x) and adding half the width (w), the y pixel coordinate (y) and adding half the height (h), to get the x,y pixel coordinate for the center.

```
ball_x = (w/2 + x)          # Get X co-ord for center of rectangle
ball_y = (h/2 + y)          # Get Y co-ord for conter of rectangle
```

<u>Adjust this to the center of the grid:</u>

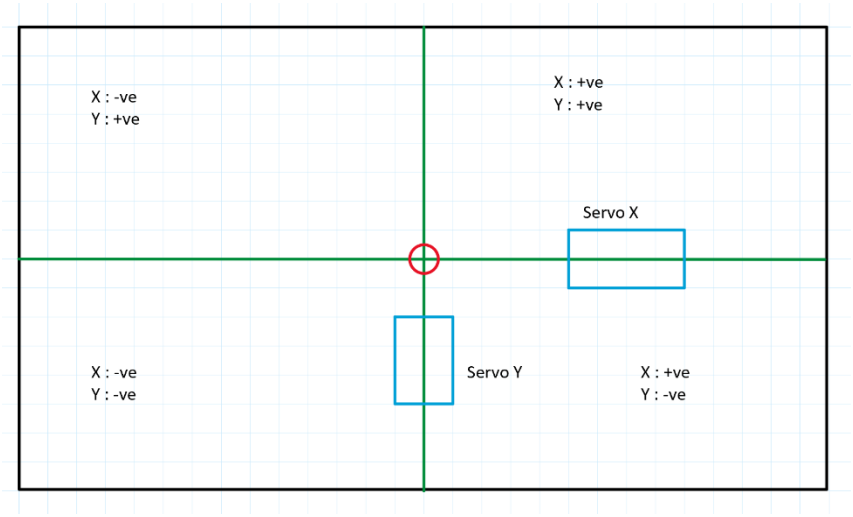

Figure 13: Figure to show the cartesian set up on the plate

OpenCV assigns the pixels of an image starting with 0,0 in the top left corner. The camera is mounted directly above the plate so that the center of the frame aligns with the center of the plate. The center of the frame is treated as (0,0). This helped with visualization of the ball's position relative to the center and therefore the expected movement of the plate. Figure 13 shows the cartesian representation of the plate, with the red circle indicating the center. The upper right quadrant has positive x and y values, while the lower left quadrant has positive x but negative y values, and so on for the remaining quadrants.

The ball's co-ordinates are adjusted to be expressed relative to a center of 0,0. This was done by subtracting half the width of the frame from the ball's x co-ordinate and by subtracting the ball's y co-ordinate from half the frame's height.

```
# adjust to centre
BP_x = ball_x - self.midWidth
BP_y = self.midHeight - ball_y
```

Apply pixel metric:

The derived transfer function, as shown in the System Modelling chapter, takes the ball's position in meters. However, images use pixels.

To convert from pixels into metric units, a pixel metric is needed. A value of the number of pixels that make up one centimetre. Then the pixel coordinate is divided by the pixel metric to get a value in centimetres (cm). To determine the pixel metric, the camera was set into position and then a video feed was started. A measuring tape was used to measure the number of cm's along the width and height of the frame. Then, the width in pixels is divided by the width in cm's, to produce the pixel metric. This value was also checked by dividing the height in pixels by the height in cm's. These values should be the same.

The pixel metric will change with the height of the camera. If the camera is at a different height, then the pixel metric needs to be re-calculated.

```
pxMetric = 7.6 # pixelperMetric for pixels to cm
# apply pixelMetric: pixels to cm
# convert cm to m
BP_x = (BP_x / self.pxMetric) / 100
BP_y = (BP_y / self.pxMetric) / 100
```

As stated, to determine the pixel metric, cm's were used as the units, however, the PID controller requires meters. Therefore, after the pixel metric has been applied, the value is then converted from cm into meters.

6.1.2 Creating a Frame Object

Image Processor gets the frame and uses colour detection to search for a ball. A bool variable for whether a ball was found or not (ballFound), the frame, the ball x and y co-ordinates in pixels and in pixels are passed into a frame object and then added into the queue.

```
# Get the latest frame
ballFound, cameraImage, BP_x, BP_y, pixelX, pixelY, elapsedTime, velocity =
self.getData()

# Append to a new ImageFrame object
imgFrameObj = ImageFrame(ballFound, cameraImage, BP_x, BP_y, pixelX, pixelY,
elapsedTime, velocity)
self.imgQueue.put(imgFrameObj
```

6.2 PID CONTROLLER

The function of the PID controller script is determine the angle that will be sent to the servo motors.

Each servo is treated as its own independent system, so the Director creates an instance of the PID controller for each servo, self.xAxis for the servo on the x axis and self.yAxis for the servo on the y axis. The x co-ordinate and y co-ordinate of the ball are each sent through the PID controller for their axis. This produces an angle output for each servo.

```
self.xAxis = PID.PID(KP, KI, KD, self.setpoint[0], False)
self.yAxis = PID.PID(KP, KI, KD, self.setpoint[1], False)
```

The inputs into the PID controller are the measured position and target position of the ball. The difference (error) between the target position and the actual position is determined and then it calculates the adjustment to the plate angle needed to get the ball to the target using values.

From the Frame Object, the Director checks if a ball was found. If it was, it then takes the ball's position (in meters) and passes it to the PID controller. The PID controller returns the angle for the servo. This can be positive or negative.

```
# If ball is found
        if (nextImg.isBallFound()):
# Get Information
            self.BP_x, self.BP_y = nextImg.getBallPosition()
            timestamp = nextImg.getTimeStamp()

            # Send position data to the PID Controllers and determine the
desired Plate Angles
            self.P_aX = self.xAxis.compute(self.BP_x, timestamp)
            self.P_aY = self.yAxis.compute(self.BP_y, timestamp)
```

When the Director has the angles, it then passes them to the SPAS class.

This section covers the steps the PID controller performs to calculate the output.

Steps:

- Calculate error
- Compare error to deadzone
- Calculate integral and derivative error
- Anti-integrator wind-up
- Calculate output

<u>Calculate the error:</u>
Set point –
This is the desired position for the ball. The position co-ordinates are relative to the center of the plate which is treated as 0,0. By default, on startup, the set point is set to the center position.

Measured value –
This is the ball's position as determined by the Image Processor. The position co-ordinates are also relative to the center of the plate and are expressed in meters as required for the transfer function.

Error –
The error is calculated from the difference between the set point and the measured value. The error is applied to the gain constants of the proportional, integral and derivative factors in order to calculate the output.

Dead zone:

```
DEADZONE = 0.005        # Acceptable Error around the target position

def __calculateError__(self, pos):
''' Compute the error between the current and target ball position with Dead
Zone considerations. '''
        # Calculate the current error of the ball position
        calculatedError = self.setpoint - pos

        # If ball is considered within the deadzone, set error to zero
        if abs(calculatedError) <= PID.DEADZONE:
            calculatedError = 0

        return calculatedError
```

The deadzone compares the calculated error value to the deadzone value. If the error is less than the dead zone value, then the error is set to zero. The deadzone value is currently set an 0.005 meters, which is 0.5 cm's. The reason this was implemented was to prevent the controller from trying to get the servo's to keep working to move the ball with such fine precision, if the ball was already within an attemptable dead zone range. The dead zone value can be changed within the code or commented out.

Calculate integral and derivative error:
Gain –
To calculate the proportional, integral and derivative values, the gain constants are needed. There is a constant value for each of the components of the controller (P, I and D). They determine how the controller will react and are the values that are adjusted when tuning a PID.

There are several methods for tuning a PID. Since the ball-plate system had been modelled and the transfer function derived, using a calculation based tunning method was attempted to save some time over manual tunning. MATLAB's PIDTuner was used to generate some values as a starting point for the tunning. The steps and values obtained are covered in the MATLAB PID Design subsection of the System Modelling chapter. However, when it came to testing these values, they did not provide an ideal starting point for tunning. Manual tunning

was then used. First, the kP value was found by trial by changing the value until the system oscillated back and forth. Then the kD value was adjusted and the system response observed. Tunning of the PID controller was in progress when the August 2021 Covid-19 Lockdown occurred. While, tunning was not able to be completed, it was on track.

```
# PID Specifications
KP = 20
KD = 15
```

Proportional –
This value is calculated by multiplying the P-gain (kP) and the error. The proportional factor gives a large initial reaction to the system. But as the error deduces, the proportional factor has less influence.

Integral –
This value is calculated by multiplying the error and the elapsed time, then adding the total of the integral factor and then multiplying it to the I-gain (kI) value.
The integral does not have much immediate influence on the output, but because it accumulates over time, it helps the ball reach the set point the longer it takes to get there.
The integral factor will get larger and larger the more time that elapses with the error not at zero. The value can very quickly get out of an realistic range for the system. This is known as integrator wind-up. As a part of the integral factor equation, the total integral factor is added in. If that value is very high, it can take a long time for it to come back down.
An anti-integrator wind up was implemented to prevent this and is detailed in the 'Anti-integrator wind-up' subsection below.

Derivative –
This value is calculated by subtracting the previous error from the current error, dividing that by elapsed time and then multiplying it by the D-gain (kD). The derivative factor looks at how fast the measured value is approaching the set value and aims to restrict that approach to limit the overshoot.

```
# Calculate the I, D Components
self.integral_error += self.error * elapsedTime
self.derivative_error = (self.error - self.last_error) / elapsedTime
```

<u>Anti-integrator wind-up:</u>

```
# Anti-Windup for the Integrator
        if self.integral_error > PID.MAX_UI:
            self.integral_error = PID.MAX_UI
```

The anti-windup for the integrator checks the calculated integrated error value against a max integrated value. If it is greater than the max, then it is set to the max. This means that the integrator value will not get larger than its maximum, no matter how long the ball takes to get to the set point. This functionality is available within the code, but the final system uses

a P-D controller. The integral component was not used, therefore the anti-integrator wind-up is not needed, but is there should it required.

Output –
The output is the result of adding up the calculated P and D values.
The Proportional factor is found by multiplying kP by the error, and the derivative factor is found by multiplying kD to the calculated derivative error.

```
 # Determines the appropriate output angle based on the current error
newOutput = (self.kp*self.error) + (self.kd*self.derivative_error)
```

6.3 SPAS

SPAS stands for Servo Plate Augmentation system. It is a thread object.

SPAS was made to apply checks and limitations on the PID output, and transform it into an angle relative to the servo motors.

The Director creates an instance of it and starts the thread.

```
# Initialise the Augmentation System and start Thread
self.augmentation =
ServoPlateAugmentationSystem.ServoPlateAugmentationSystem(serialAddress,
BAUD_RATE, NUM_OF_BITS, 0)
self.augmentation.start()
```

The Director inputs the angle for each servo. But SPAS does not return any value.

```
# Send the desired angle to SPAS for Augmentation and Tx
self.augmentation.setNextAngle(self.P_aX, self.P_aY)
```

When SPAS has prepared the new angles for the servo's, it calls the UART class to send them.

```
# Send the Instruction
self.uart.sendXServo(servoX)
self.uart.sendYServo(servoY)
```

In order to protect the servo motors, several limiting factors were designed and put in place.
Steps:

- Check the hold counter
- Step the angle
- Convert to servo's angle range
- Send to UART

Holder counter:

If the ball is found in the frame, the Image Processor will return its position for every frame. The camera reads 30 frames per second. That means SPAS will receive new angles for the servo's approximately every 0.03 seconds. This is too fast for the servo's.
Instead of altering how many frames were read per second, or how often the PID controller calculates an output, SPAS implements a hold counter on how often new angles are sent to the servos. It was decided to send 5 instructions per second.

When SPAS gets the PID outputs, it checks the hold counter against the hold max value to determine if a new angle can be sent.

```
# If holdCounter is still active, then do not calculate and maintain
self.frameHoldCounter += 1
if self.frameHoldCounter < self.frameHoldMax:
       return self.currXAngle, self.currYAngle
```

If the hold counter is still active, then the current angle is maintained. If the hold max is reached, then the angles are stepped.

Step the angle:

The servo motors will aim to reach the angle they are given immediately. This could mean from flat (0°) to $+20°$. In order to limit this, the angle sent to the servo's will be incrementally stepped until it matches the desired angle. The desired angle may change with each new frame.

The current angle, which in this case is the angle previously sent to the servos, is subtracted from the newly calculated angle. Then that result is compared to max delta angle, which is the maximum degree of angle change (step) allowed per instruction sent.

```
MAX_DELTA_ANGLE = 5     # Maximum Rate of change of Deflection Angle / sec

def __setStepAngle__(desiredAngle, currentAngle):
        # Get delta
        delta = abs(desiredAngle) - abs(currentAngle)

        # If delta is too great, then set the next angle
        if abs(delta) > ServoPlateAugmentationSystem.MAX_DELTA_ANGLE:
            return numpy.sign(delta) *
ServoPlateAugmentationSystem.MAX_DELTA_ANGLE
        else:
            return desiredAngle
```

If delta is greater, then it is set to the max delta value. If delta is not greater than the max delta range, then the angle is unchanged.

Convert to servo:

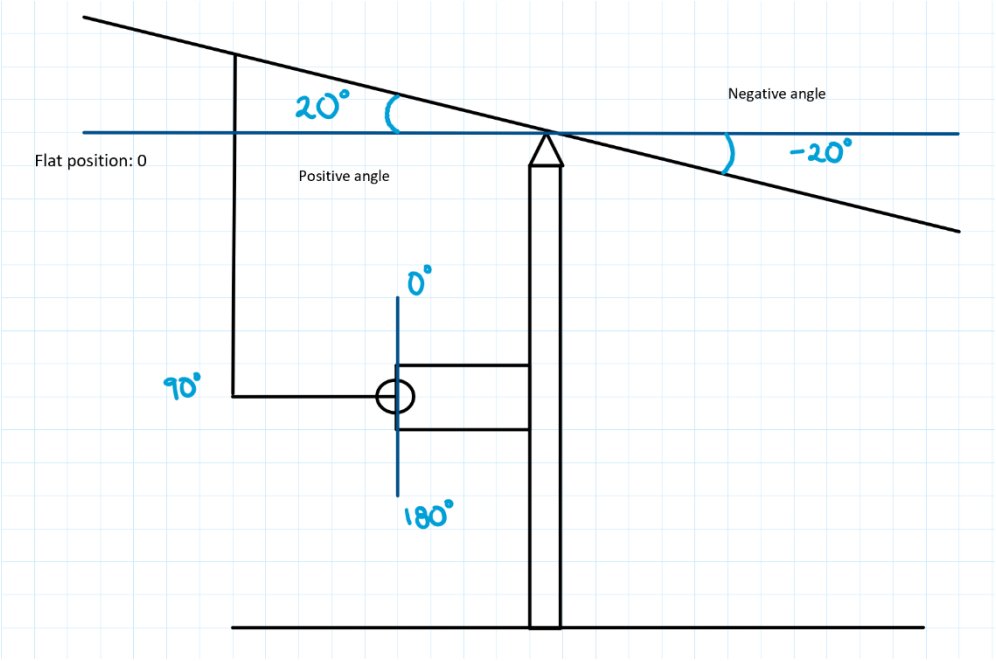

Figure 14: Figure to show the servo angle and the plate angle.

The ball's position is expressed relative to the centre of the plate, as that is the pivot point of the plate. The PID outputs a positive or negative value. An angle of 0° indicates that that axis of the plate be flat. Obviously as one side of the plate goes down, the other goes up, like a see saw. The angle is related to side of the plate that servo arm is connected to. As shown in Figure 14, the servo is connected to one end of the plate. A positive angle indicates the servo should push its side of the plate up, while a negative angle indicates the servo should pull its side of the plate down.

While an angle of 0° indicates that the plate should be parallel with the floor, if the servo was sent 0°, it would try to point to the ceiling. Figure 14 also shows the arm directions of the servo's 0° to 180° range, with its 90° position sitting flat. Therefore, the PID output can't be sent straight to the servo motors. It must first be converted relative to their degree range.

Figure 13 shows the position of each servo relative to the cartesian representation. From that relationship, the following code was determine to adjust the output PID angle to reflect the degree range values of the servos.

```
def __convAngleToServo__(servoXAngle, servoYAngle):
        ''' Convert from plate axis system to servo axis system. '''
        servoXAngle = 90 - servoXAngle
        servoYAngle = 90 + servoYAngle
        return servoXAngle, servoYAngle
```

6.4 UART

The UART class performs final checks on the angle, converts it into binary and then sends it via serial port to the FPGA.

An instance of the UART class is created in the SPAS script.

```
# Create UART Controller
self.uart = UART.UART(devicePath)
```

When SPAS has completed all its tasks, it calls the sendXServo() function and sendYServo() functions from the UART class, which passes in the servo angles. There is one function for each servo, as the FPGA needs to know which servo gets which angle. How this is achieved is discussed in the 'Convert to binary' subsection.
Steps:

- Check if larger than max angle range – set to max
- Convert to binary
- Transmit through serial port

Check max angle range:
The servo has a range of 0 to 180 degrees, however, the plate is not able to move the same range. The ping pong ball is very sensitive to movement. For the mechanical limitation of the plate and for control over the ping pong ball, the maximum range the servos could move is limited to 20. This is done by comparing the new angle to the max angle value. If is it greater, it is set to the max value. If not, it isn't changed.

Convert into Binary representation:
The binary representation and its interpretation by the FPGA was developed by the other group member and further details can be found in the report of Daniel Dymond.
First the angle is converted into a binary representation.

```
# If value is okay, then convert into closest binary representation
binaryPos = ((angle - 70) / 0.5)
binaryPos = int(binaryPos)
return binaryPos
```

Then an 8-bit array is generated. The first bit is used to identify the servo. 1 for x axis and 0 for y axis.

```
def __generateUARTData__(binaryPosition, isYServo):
''' Takes the Binary Position Data and Servo Select and generates the 8-bit
value '''
        # Assembles the byte
        if (isYServo == 1):
            byteInt = binaryPosition + 128
        else:
            byteInt = binaryPosition
```

```
        byte = [byteInt]

        return bytearray(byte)
```

Transmit through serial port:
The array is then transmitted to the FPGA via serial port.

```
UART.__uartTX__(xByte)
UART.__uartTX__(yByte)

def __uartTX__(data):
''' Performs the UART TX for some given data '''
        global serialPort
        size = serialPort.write(data)
```

6.5 Director

The Director is a management script which was designed to manage the following classes: Image Processor, PID controller and SPAS, and to pass the Frame Object into the Frame Collector.

The Director creates instances of the Image Processor, PID controller, SPAS and frame collector and starts the Image Processor and SPAS threads. The Director then checks the queue and fetches the frame object from it when available.

From the information within Frame Object, the Director checks to see is a ball was found. If it was, the ball's position is passed into the PID controller. The PID controller returns the output angle. The Director passes this to SPAS.

The frame object is then sent to the frame collector by the Director, for the GUI to have access to.

The Director is also set up with the ability to implement different modes. Modes mean a different set point or set points (for a pattern).

6.6 Frame Collector

The webcam is accessed by the Image Processor. The graphical user interface displays each frame. The GUI needs to know when a frame is available. QT enables signals and slots to be created, but they can only be sent between QObjects. This is why the Frame Collector script was made. The Frame Collector is a QObject that sends a signal to the GUI, to let it know that there is a frame available to display.

The Director passes the Frame Object, which holds the frame and information about the ball, to the Frame Collector. The frame collector places in into a queue and emits a signal to the GUI object.

```
def addFrame(self, frameToAdd):
        self.frameQueue.put(frameToAdd)
        self.imageUpdate.emit()
```

This signal indicates to the GUI that a new frame is available. When the signal is emitted, it is detected by the GUI and the signal is connected to a slot. In the slot, the action to be taken can be coded.

```
# Link director to GUI
self.frameCollector.imageUpdate.connect(self.ImageUpdateSlot)
def ImageUpdateSlot(self):
# cant access at same time as director img queque
# director will make a separate queue and pass on img object to it,
# after director queue is done with it.
        imageFrame = self.frameCollector.getFrame()

        # Get the image
        image = imageFrame.getCameraFrame()
        self.displayFrame(image, imageFrame)
```

6.7 Graphical User Interface

A Graphical User Interface (GUI) is displayed on the touch screen. It has two purposes; to enable the user to select modes of control of the ball, and to display the feed from the camera.

Modes:

- Balance at centre
- Balance at set position
- Follow a rectangle pattern
- Follow a circle pattern
- Switch to Manual control with joystick

Camera feed display:

- Display the frame with the centre point indicated by a small dot.
- If ball is found in the frame, indicate it by drawing a circle around it and a dot at its centre.
- Display the x and y co-ordinates (in cm) of the ball
- Display the error
- Display the PID output angle

Qt Designer was used to design the GUI. It is a drag and drop design tool for making user interfaces. Figure 30 in Appendix B shows Qt Designer with the main window widget loaded for editing. To place any item onto the main window or any widget, drag and drop the item from the left panel in Qt Designer, the Widget Box. Once placed, it can moved around and resized. On the right panel of Qt Designer is the Property Editor, which is an easy way to find and adjust the editable elements of the widget. Such as size, position, font, state.

Qt Designer produces the designed GUI in a 'ui' script file, written in the user interface markup language. This script then converted into a py file. This converted file is not editable. It just contains the design coding of the GUI in Python.

Front End – Design:

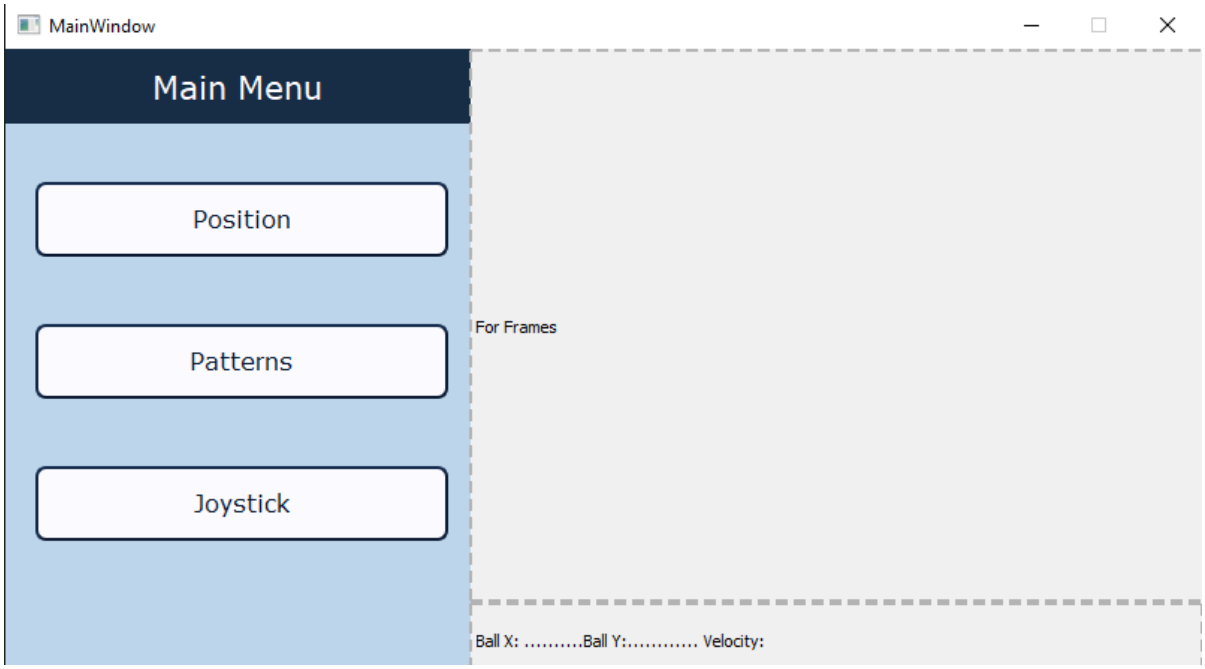

Figure 15: GUI Main Menu screen

The touch screen display area is 800 x 416. The main window was set to this size.
The camera frame size is 640 x 480 pixels. This value has enough pixels that the image is still clear, but not so many that it takes more time and processing power to run detection on the frame. However, we didn't have the space on the central widget to display that frame size.

Figure 15 is the main menu page of the GUI. The space on the right is to display the frames from the camera and is 494 x 370 pixels. The label below the frame display area is to display the values; the ball x and y position, PID output and servo angle. These are both on the central widget.

The panel on the left is a stacked widget. The camera feed is displayed continuously as the user moves through the menu's, but didn't want the camera feed re-loaded on each new page. The stacked widget enables different widgets to be shown in that space without moving away from the central widget.

The stacked widget contains four widgets;

- Main Menu
- Balance at Position
- Follow a Pattern
- Joystick

These can be seen in Appendix B in Figure 31.

The GUI file made in Qt Designer is a .ui file. PyQt offers a method to translate that file into the Python file type. In the terminal,

```
pyuic5 -x BallPlateUI.ui -o BallPlateGUI.py
```

This command creates a python script from the .ui file. 'BallPlate UI.ui' represents the name of the file that was generated from Qt Designer. 'BallPlateGUI.py' represents the name that you want to call the Python file that is created from translating the '.ui' script.
If any design changes are made to the GUI, go back into Qt Designer and make the changes, save it, then run the above command again, and it re-generates the translated Python script.

However, the Python script that is produced is just the design description of the GUI, it has no functionality yet. Also, that script cannot be edited. If it is edited, it will not save any changes. In order to interact with the GUI, another Python script needs to be used. This lead to the creation of the Control script.

6.8 CONTROL

Another Python script was made, BallPlate_run.py, to import the UI GUI from the translated python script (BallPlateGUI.py) and to set up the functionality needed. This script is the main script that displays the GUI, enables its behaviour and integrates with the Director to initiate the rest of the code.

User actions on widgets, such as button presses, are known as events in PyQt. When an event occurs, the widget emits a signal. This enables actions to be performed in response to events. PyQt calls these slots. Most of the widgets have built-in signals and slots, but custom ones can also be made. To set up a slot to respond to an event, the signal needs to be connected to the slot.

```
self.ui.btn_main_position.clicked.connect(self.showPosition_pg)
```

This is an example of a widget built-in signal. The Main Menu page has three buttons, one is labelled 'Position'. Its identifier is btn_main_position. 'Clicked' is one of the signals that can be emitted by the button widget and is being connected to a method. The method is known as a slot. It contains the actions to be performed when that signal occurs. As shown below, that method changes the stacked widget to the 'Position' page.

```
def showPosition_pg(self):
        self.ui.stackedWidget.setCurrentWidget(self.ui.position_pg)
```

Essentially, when the 'position' button on the main menu widget is pressed, a signal is emitted. That signal is connected to a method that changes the stacked widget to the 'Position' page. This how functionality of the GUI was set up. Every action that the user could make, emitted a signal, that connected to a method that performs the required actions.

One custom signal was made in order to display the frames. A method was needed to let the BallPlate_run script to know that a frame is there to be displayed and only Qt items can emit signals. So the 'Frame Collector', a QObject was made. When the Director class receives each frame, it sends it to Frame Collector. When the Frame Collector gets a new frame it emits a signal, it is detected by BallPlate_run and connected to a slot.

```
# Link director to GUI
self.frameCollector.imageUpdate.connect(self.ImageUpdateSlot)
```

This signal lets the main script know that a new frame is available. The slot fetches the frame, prepares it for display and displays it. On the frame, the centre point is indicated with a dot and two lines cut through the centre, indicating the co-ordinate structure used. If the ball was found, a circle is drawn on its outline, and a dot at its centre. The label area is smaller than the frame size, so it is scaled down. OpenCV reads images as BGR, but the Pixmap used to display the image reads images as RGB. Therefore, OpenCV method to convert an image type is use to change it to RGB. It is then formatted into a QImage and converted to QPixmap and then displayed in the label space.

```
# Convert image (np.ndarray) into format for PyQt5 for display
image = cv.cvtColor(image, cv.COLOR_BGR2RGB)
image = qtg.QImage(image, image.shape[1], image.shape[0],
qtg.QImage.Format_RGB888) # format as QImage
image = qtg.QPixmap.fromImage(image) # convert to QPixmap
self.ui.lbl_frames.setPixmap(image)
```

7 RESULTS

The aim of this project is to produce a ball-plate system that can act as a base platform for future projects, capable of controlling the ball's position within the specified project constraints and parameters.

This project produced a functional program that is able detect the ball, and return it to the set point timely if it is pushed off.

Computer vision techniques are used for the colour detection of the orange ping pong ball. The Image Processer searches each incoming frame and is able to detect the ball. From the ball shape detected, a process determines its position in pixels and in meters. A P-D controller has been implemented and control over the ball is achieved, returning it to the centre when it is pushed away. UART communication between the Pi and FPGA was successfully established and the FPGA set up to generate the PWM. When the Pi sends out the angle the FPGA receives it and moves the servos to the required angle. A simple yet elegant GUI was designed and runs on the touch screen. It displays the webcam feed and identifies the ball when it is found. The hardware modification resulted in the production of a sleek, form fitting HMI panel that houses the Pi with touch screen and the joystick. It is mounted onto the existing hardware.

Tasks identified for this project included:

- Design and mount the HMI panel,
- Mount the camera,
- Set up the FPGA,
- Establish UART communication between Pi and FPGA,
- Use computer vision techniques to detect the ball,
- Develop and apply a PID controller,
- Design and create a graphical user interface to offer the user modes of interaction,
- Obtain and demonstrate the system functioning, including control over the ball.

The final outcome of each task is stated in the following sections along with evidence.
In addition, as the 2021 Covid-19 Lockdown has prevented any live demonstration of the final product, a google photo album has been set up for this project at the link provided below.

https://photos.app.goo.gl/27SCimtcfYtEt9jX9

It contains photos that show the successful development and implementation of elements of this project. All pictures shown in this Results section can be found in the google photo album. There is also three short videos taken during the tunning of the PID controller in the album. Further mention of them is covered in the Control over the ball subsection.

7.1 HARDWARE – MECHANICAL

The mechanical modifications for this project are the design, production and mounting of the HMI panel and design and mounting of the camera.

7.1.1 HMI panel

Figure 16: HMI panel mounted onto the existing hardware

The CAD design developed is a sleek and form-fitting HMI housing for the Pi with touchscreen and joystick. It was 3D printed and then the components were then fitted into the housing. The mounting arms already on the existing hardware are used to attach the HMI panel onto the hardware unit. Figure 16 shows the HMI panel mounted into position onto the existing hardware. Three additional images, showing the HMI at different angles are in Appendix C, as Figure 32, Figure 33, and Figure 34.

7.1.2 Camera mount

Figure 35 in Appendix C shows the design for mounting the camera. The height for the camera was determined and plastic piping and an elbow had been purchased. There was the intention to test how sturdy it was at holding the camera and to design methods of fastening it to the hardware. However, testing and further development of the camera mount was halted due to the Covid-19 August 2021 lockdown as the piping and the hardware were at university over lockdown.

7.2 HARDWARE - ELECTRICAL

7.2.1 FPGA

In order to demonstrate the successful set up of the FPGA and UART communications, the other member of this group produced a Python script. This script generates a simulated angle output which is printed into the terminal and sent to the FPGA. FPGA's LEDs light to indicate

the angle received. Full details of this are available in Daniel Dymond's report. Further evidence that the angle is successfully sent from the Pi to the FPGA and that the FPGA has control over the servos can be seen in the videos in the google photo album. The videos show that the servos are moving the plate. The FPGA is receiving an angle from the Pi and driving the servos to that angle.

7.2.2 Joystick

A schematic for the joystick's circuit board was made and the PCB designed. The PCB board arrived during the August 2021 lockdown, but several of the components have not yet arrived. Therefore, full implementation of the joystick could not be realised. All that remains to complete, is to obtain the components and solder them onto the board and then establish SPI communication between the joystick and the Pi. However, the PCB boards and the schematics are available for other groups to continue with.

7.3 SOFTWARE

The program developed for this project contains nine Python classes that each play a role in ensuring the program functions successfully. The Image Processor uses computer vision techniques to successfully detect the ball and determine its position. Control over the ball has been achieved with the design and development of a P-D controller. The outputs from the P-D controller move the plate to the angles required to bring the ball back to the set point after it is pushed away. The graphical user interface has several display panels to offer the user different modes of interaction and displays the camera feed with the ball indicated when it is detected in the frame.

7.3.1 Ball detection

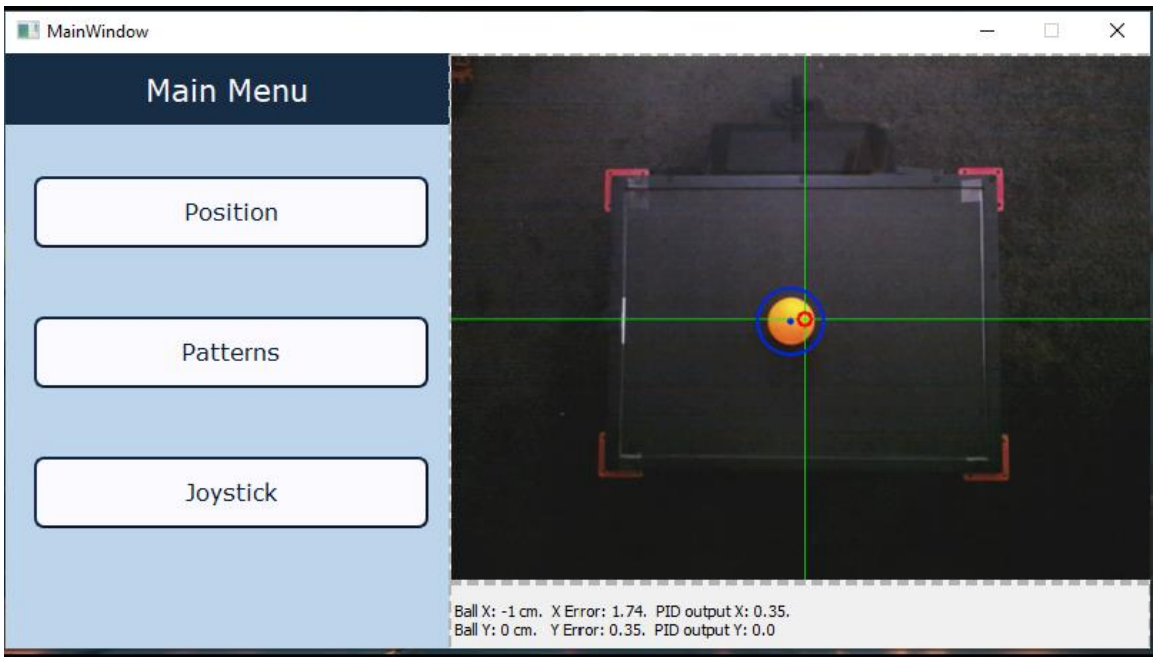

Figure 17: Screen shot of GUI with ball placed near to the centre, showing the ball's position, error value and PID output

The Image Processor is able to detect the ball and determine its position in pixels and in meters. To demonstrate this, the program was run and the ball was placed at the center and at each corner of the plate and a screen shot was taken. On the GUI, below the video display, the X, and Y position of the ball in cm's, the calculated error and the PID controller output is displayed. For the following figures, the position of the ball and the displayed values can be compared to show that program is working and operating as expected.

In Figure 17 the ball has been placed near the center. The Image Processor has determined the ball's position to be -1, 0 cm in the x and y coordinates respectively. As seen in the figure, the ball is sitting in the y axis line.

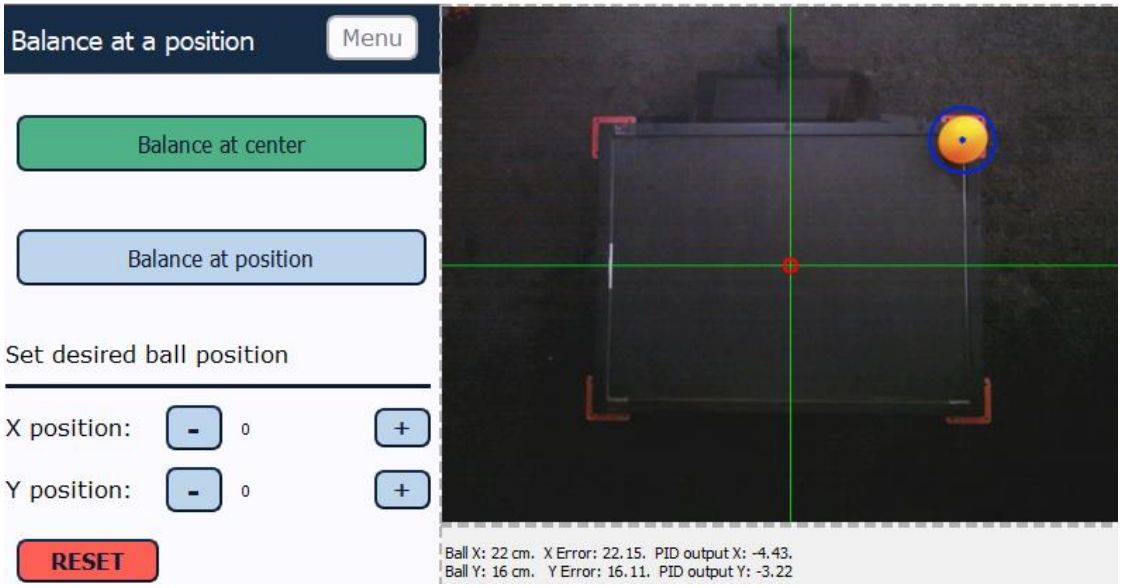

Figure 19: Screen shot of GUI with ball placed in upper right quadrant, showing the ball's position, error value and PID output

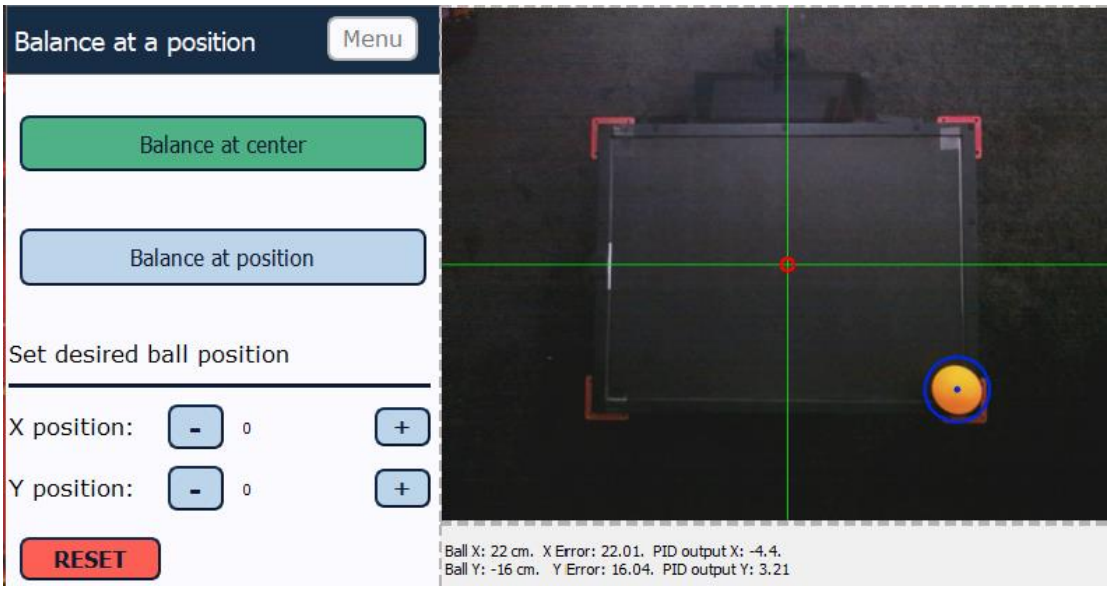

Figure 18: Screen shot of GUI with ball placed in lower right quadrant, showing the ball's position, error value and PID output

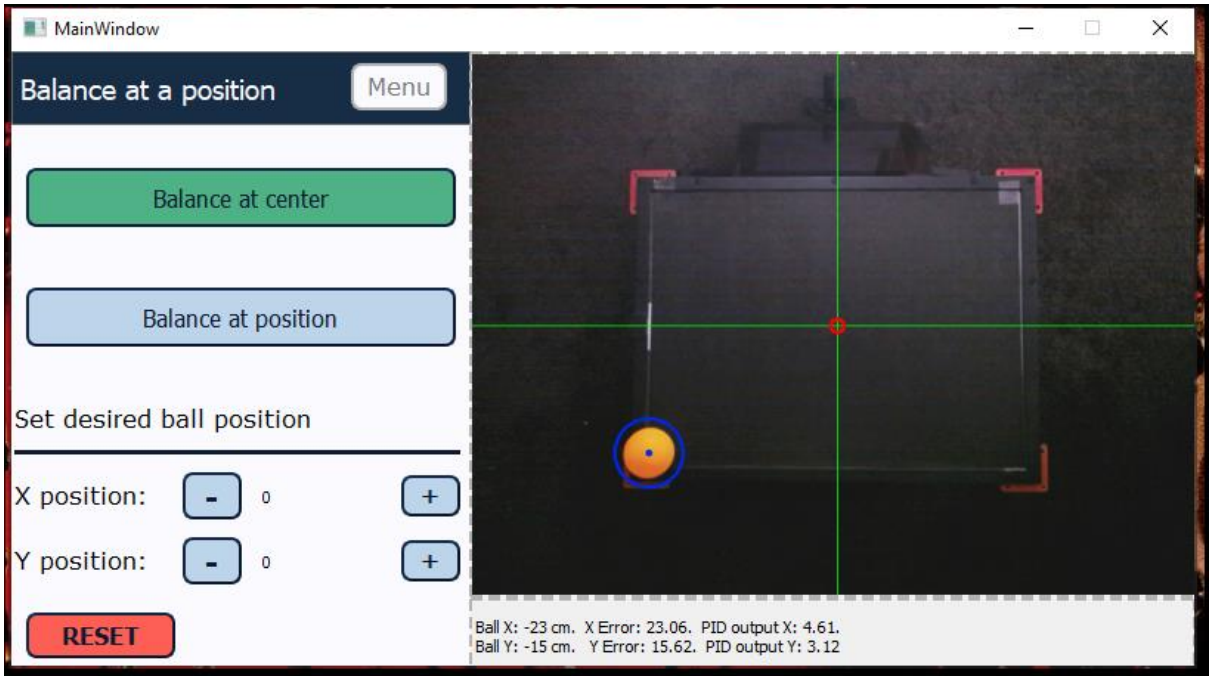

Figure 21: Screen shot of GUI with ball placed in lower left quadrant, showing the ball's position, error value and PID output

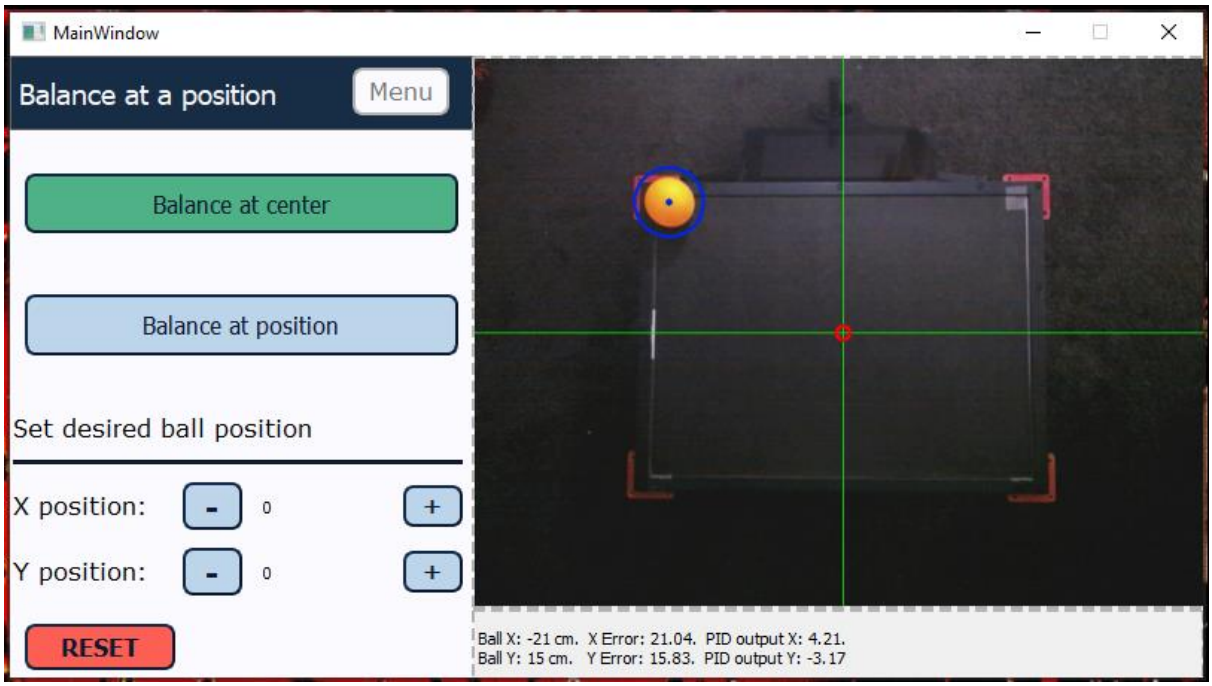

Figure 20: Screen shot of GUI with ball placed in the upper left quadrant, showing the ball's position, error value and PID output

Figure 19 to Figure 20 show the ball placed at each of the four quadrants of the plate. The ball's position, error value and PID output can been seen in the label below the camera display. They show that the Image Processor is able to detect the ball within the frame and determine its position. The position values from the Figures also show that the cartesian coordinate system over the plate, with its center as 0,0, has been successfully established.

7.3.2 Control over the ball

On Figure 17, the error calculated for the y axis is 0.35 cm. This falls within the PID controllers deadzone, meaning the PID controller output should be zero. Figure 17 PID output Y is showing '0', which means the program is performing as expected.

Further evidence of the PID control over the ball can be seen in the three videos in the google photos album. One of the videos shows the tuning of the P constant. The ball oscillates back and forth across the plate. The other two videos show the P-D controller in operation, returning the ball back to the center when it has been pushed off set point.
A few frames have been captured from the videos and are displayed in Figure 22 in order to give a general impression of the video.

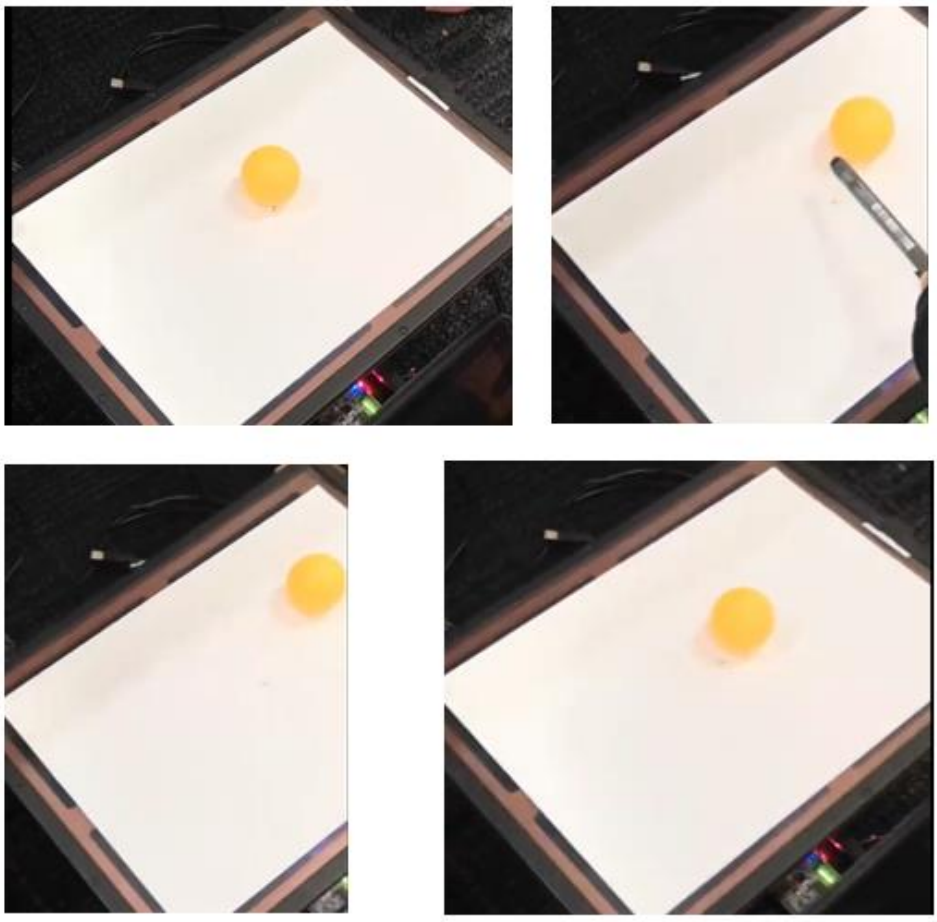

Figure 22: Several frames showing the path of the ball being pushed off from the set point and being rolled back to the set point from the plate's angle

7.3.3 Graphical user interface

A simple and yet elegant GUI was designed and created. It has four menu pages: main menu, balance at position, follow a pattern and joystick control. It displays the video feed and identifies the ball when it has been found. If a ball is found, its x and y coordinates, the error and the PID output are also displayed in the label below the frame window.

The main menu screen buttons are functional. Pressing any of the buttons on the main menu widget shown in Figure 23, changes the stacked widget to the selected menu page.

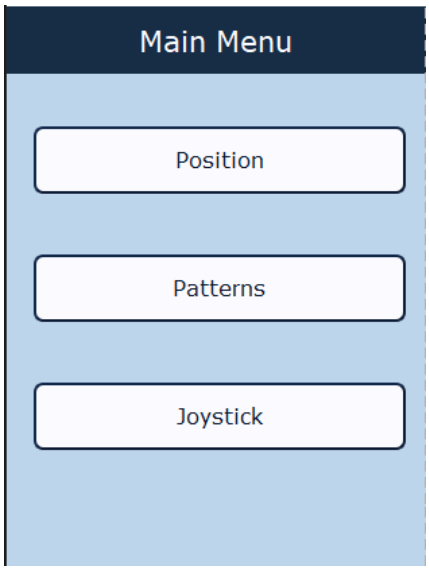

Figure 23: Main Menu stacked widget

On the 'Position' page, two options are available; balance at centre, and balance at position. If the 'balance at position' button is pressed, the X position and Y position plus and minus buttons become active. They are each connected to a label, which displays a value that starts at zero. If the plus or minus button is pressed, the value is altered by ± 2. A yellow dot is displayed on the frame at the x and y coordinates from the X position and Y position labels. As the value is changed by the user pressing the buttons, the yellow dot moves to the entered position. This is shown in the following four images, Figure 24 to Figure 27.

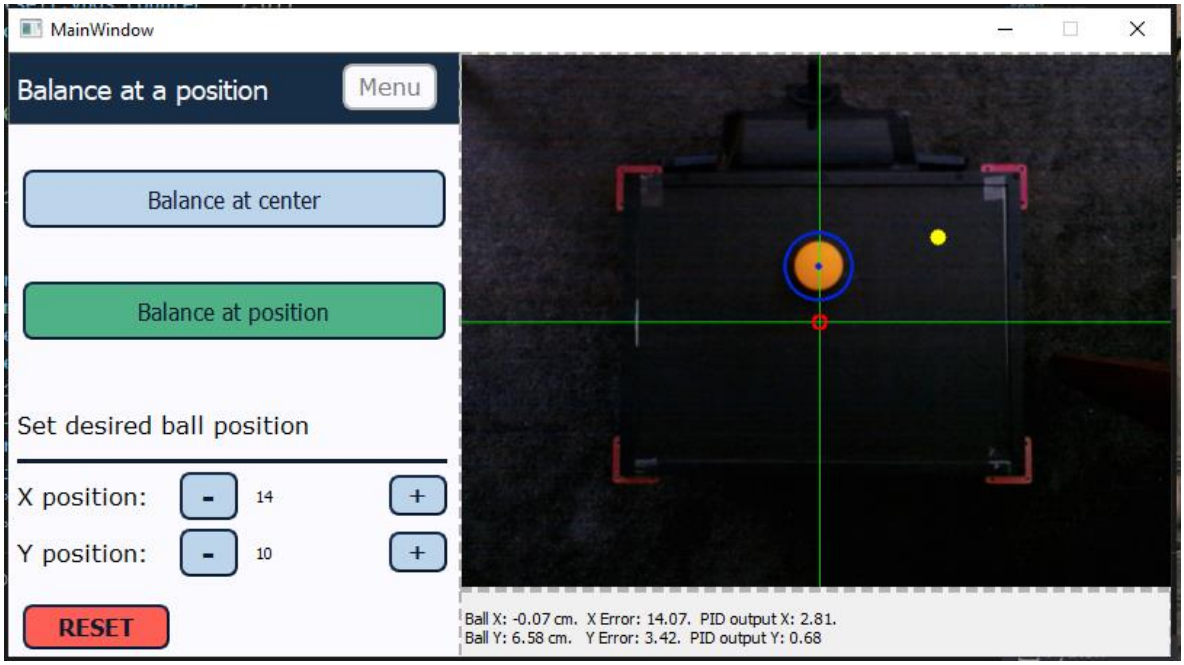

Figure 24: Balance at position mode with yellow dot representing a position entered for the upper right quadrant.

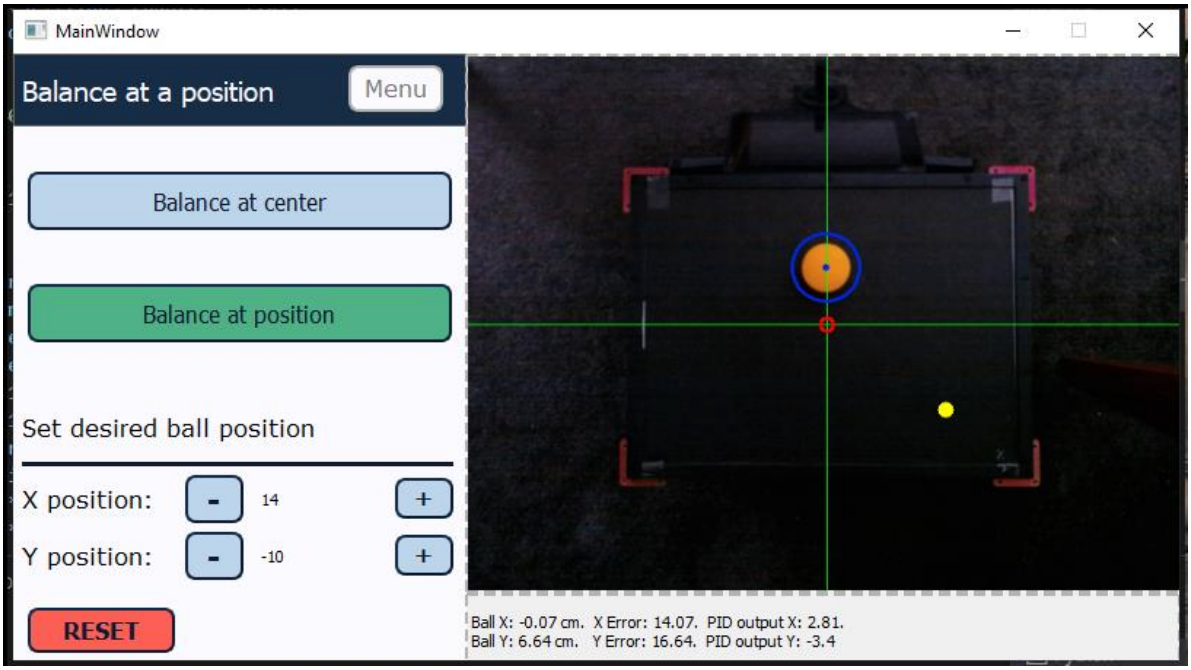

Figure 25: Balance at position mode with yellow dot representing a position entered for the lower right quadrant.

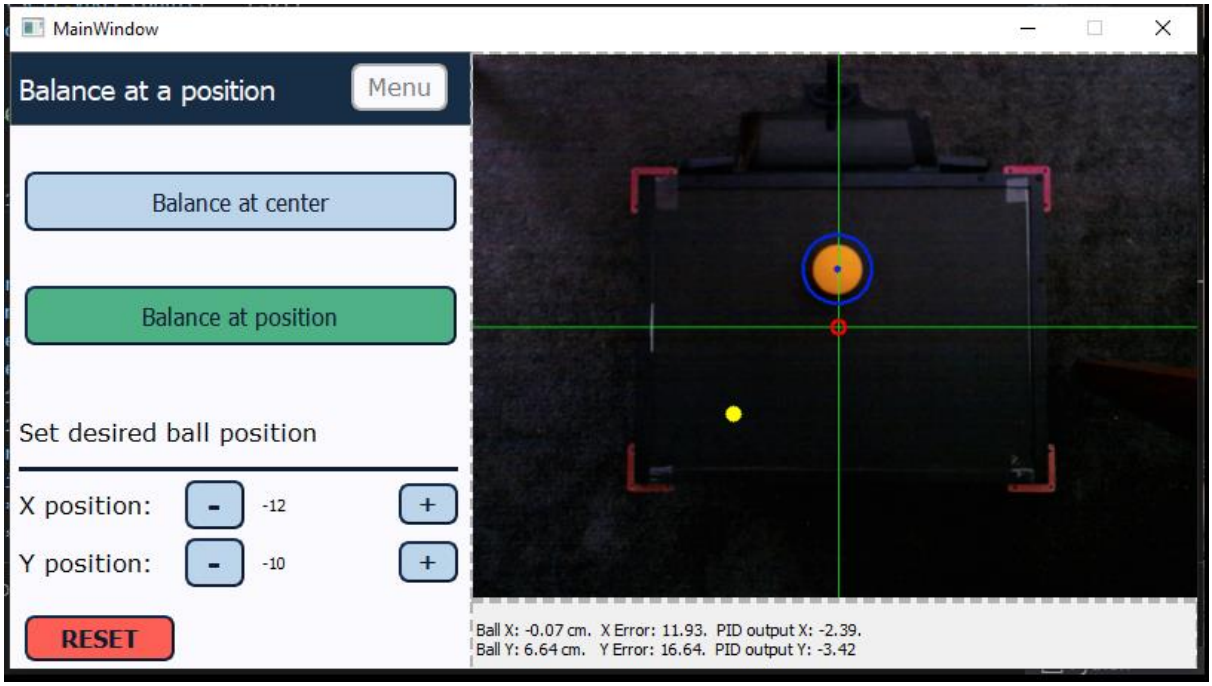

Figure 26: Balance at position mode with yellow dot representing a position entered for the lower left quadrant.

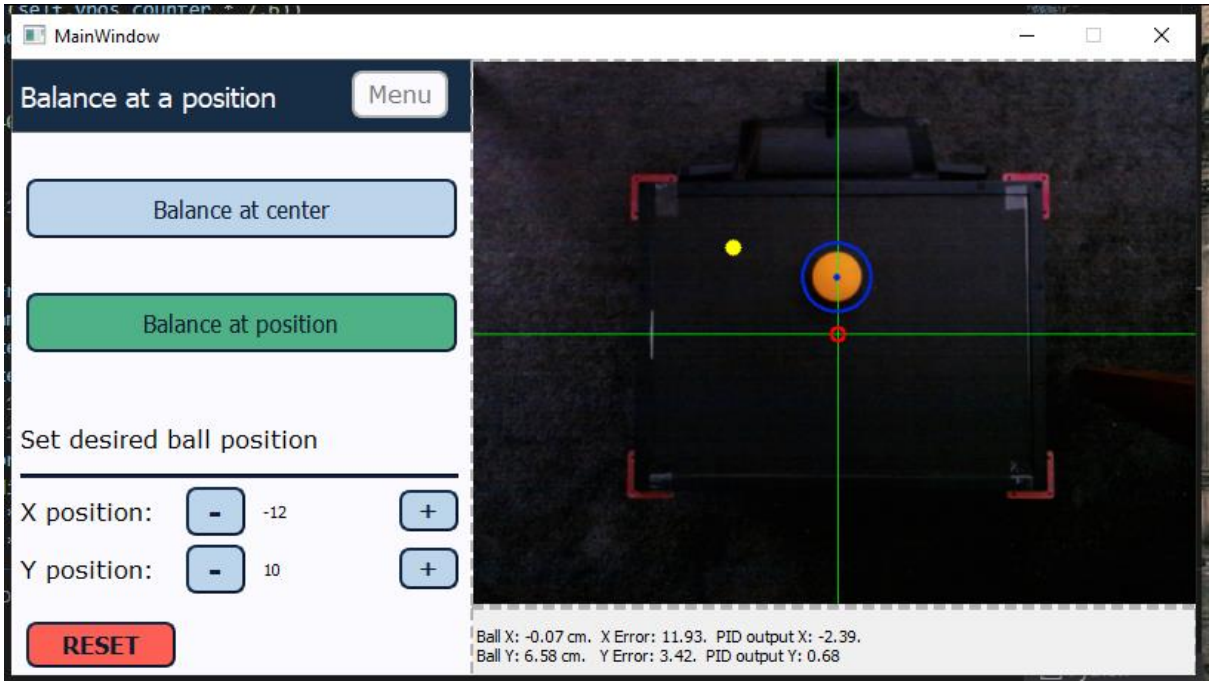

Figure 27: Balance at position mode with yellow dot representing a position entered for the upper left quadrant.

This is to enable the user to enter a set point position. This functionality is set up in the GUI side only. The set point is not passed from the GUI to the Director.
There was the intention to make modes available, other than balancing at the centre. Such as following a rectangle or circle path. The code is set up to allow for this functionality to be activated, however, the timing for the set point changes could not be established. This is the reason why full implementation of the modes was not completed. The physical system was needed in order to test and tune the optimal timing before the next set point is sent to the PID controller. It was considered that a 'time in deadzone' may be required. Time in deadzone would be similar to the set point deadzone but timed. If the ball was determined to be within the deadzone of the set point for a specified time, then it could be considered that the ball had settled at the set point, and the next set point could be sent. This would likely work for a rectangle shape, but possibly not for a circle path. As the hardware was at university during the Covid-19 lockdown, this could not be developed further.

8 CONCLUSIONS AND FUTURE WORK

This project was undertaken to modify an existing ball-plate hardware unit to produce a ball-plate system that can act as a platform for future projects, capable of controlling the ball's position within the specified project constraints and parameters.

This project produced a functional system comprising of hardware modifications and a Python program that detects the ball, and returns it to the set point.

The hardware modifications completed consist of an HMI panel that houses the Pi, touchscreen and joystick.

The program was designed as a modular block platform to provide a structure of the ball-plate for future groups to use. Blocks can be removes and replaces as required. Such as to replace the Image Processor with a new block the uses the resistive touch sensors, or replace the PID controller block and a different controller method block.

The colour detection method used by the Image Processor is successful, as is the PID controller that was tunned to P-D. The results section has shown the success of the Python program and is further supplemented with images and videos available in the google photo album.

Covid-19 did halt further development and final implementation of some aspects of the project. The development of the camera mount was unable to be completed, as was the realisation of the PCB for the joystick. However the joystick PCB schematic is documented, enabling other groups to proceed with implementation should they wish to.

Further tunning of the PID controller and the setup of the patterns could not continue without access to the hardware. The code has been well commented and the reports from each group member supplement information on the program for use by future project groups.

BIBLIOGRAPHY

About. (n.d.). *OpenCV*. Retrieved 4 October 2021, from https://opencv.org/about/

Ahmad, B., & Hussain, I. (2017). Design and hardware implementation of ball amp; beam setup. *2017 Fifth International Conference on Aerospace Science Engineering (ICASE)*, 1–6. https://doi.org/10.1109/ICASE.2017.8374271

Bigharaz, M. H., Safaei, F., Afshar, A., & Suratgar, A. A. (2013). Identification and nonlinear control of a ball-plate system using neural networks. *And Automation The 3rd International Conference on Control, Instrumentation*, 260–262. https://doi.org/10.1109/ICCIAutom.2013.6912845

Chen, W., Sui, X., & Xing, Y. (2012). Modeling and modulation of nonlinear ball-beam system controller based on matlab. *2012 9th International Conference on Fuzzy Systems and Knowledge Discovery*, 2388–2391. https://doi.org/10.1109/FSKD.2012.6234285

Dayala, R. (2020, April 28). Color Filtering/Segmentation/Detection – HSV. *Computer Vision*. https://cvexplained.wordpress.com/2020/04/28/color-detection-hsv/

Dušek, F., Honc, D., & Sharma, K. R. (2017). Modelling of ball and plate system based on first principle model and optimal control. *2017 21st International Conference on Process Control (PC)*, 216–221. https://doi.org/10.1109/PC.2017.7976216

Fan, X., Zhang, N., & Teng, S. (2004). Trajectory planning and tracking of ball and plate system using hierarchical fuzzy control scheme. *Fuzzy Sets and Systems*, *144*(2), 297–312. https://doi.org/10.1016/S0165-0114(03)00135-0

Han, J., & Liu, F. (2016). Positioning Control Research on Ball amp;Plate System Based on Kalman Filter. *2016 8th International Conference on Intelligent Human-Machine*

Systems and Cybernetics (IHMSC), *01*, 420–424. https://doi.org/10.1109/IHMSC.2016.196

He, H., & Chen, J. (2020). Research on Nonlinear Ball-plate System Based on LTR Method. *2020 Chinese Control And Decision Conference (CCDC)*, 5243–5248. https://doi.org/10.1109/CCDC49329.2020.9164066

Isa, A. I., Hamza, M. F., Zimit, A. Y., & Adamu, J. K. (2018). Modelling and Fuzzy Control of Ball and Beam System. *2018 IEEE 7th International Conference on Adaptive Science Technology (ICAST)*, 1–6. https://doi.org/10.1109/ICASTECH.2018.8507132

Kostamo, J., Hyotyniemi, H., & Kuosmanen, P. (2005). Ball balancing system: An educational device for demonstrating optimal control. *2005 International Symposium on Computational Intelligence in Robotics and Automation*, 379–384. https://doi.org/10.1109/CIRA.2005.1554306

Maalini, P. V. M., Prabhakar, G., & Selvaperumal, S. (2016). Modelling and control of ball and beam system using PID controller. *2016 International Conference on Advanced Communication Control and Computing Technologies (ICACCCT)*, 322–326. https://doi.org/10.1109/ICACCCT.2016.7831655

Morales, L., Gordón, M., Camacho, O., Rosales, A., & Pozo, D. (2017). A Comparative Analysis among Different Controllers Applied to the Experimental Ball and Plate System. *2017 International Conference on Information Systems and Computer Science (INCISCOS)*, 108–114. https://doi.org/10.1109/INCISCOS.2017.27

PyQt—Python Wiki. (n.d.). Retrieved 27 September 2021, from https://wiki.python.org/moin/PyQt

Qt Designer and Python: Build Your GUI Applications Faster – Real Python. (n.d.). Retrieved 27 September 2021, from https://realpython.com/qt-designer-python/

Raspberry Pi Camera Board V2. (n.d.). Pi Australia. Retrieved 12 October 2021, from https://raspberry.piaustralia.com.au/products/raspberry-pi-camera-board-v2

Riverbank Computing | Introduction. (n.d.). Retrieved 27 September 2021, from https://riverbankcomputing.com/software/pyqt/intro

Shi, Z., & Liu, F. (2015). An Accuracy Indicator of Control System and Its Application in Ball amp;Plate System. *2015 Fifth International Conference on Instrumentation and Measurement, Computer, Communication and Control (IMCCC)*, 1516–1520. https://doi.org/10.1109/IMCCC.2015.321

Soni, R. & Sathans. (2018). Optimal control of a ball and beam system through LQR and LQG. *2018 2nd International Conference on Inventive Systems and Control (ICISC)*, 179–184. https://doi.org/10.1109/ICISC.2018.8399060

Spacek, L., Bobal, V., & Vojtesek, J. (2017). Digital control of Ball amp; Plate model using LQ controller. *2017 21st International Conference on Process Control (PC)*, 36–41. https://doi.org/10.1109/PC.2017.7976185

Valluru, S. K., Singh, M., & Singh, S. (2016). Prototype design and analysis of controllers for one dimensional ball and beam system. *2016 IEEE 1st International Conference on Power Electronics, Intelligent Control and Energy Systems (ICPEICES)*, 1–6. https://doi.org/10.1109/ICPEICES.2016.7853133

Ward, J. (2017, July 22). Best Programming Language for Raspberry Pi for Beginners. *Raspberry Pi Starter Kits*. https://www.raspberrypistarterkits.com/resources/best-programming-language-raspberry-pi/

What is a Raspberry Pi? (n.d.). *Raspberry Pi*. Retrieved 9 October 2021, from https://www.raspberrypi.org/help/what-is-a-raspberry-pi/

9 APPENDIX

APPENDIX A

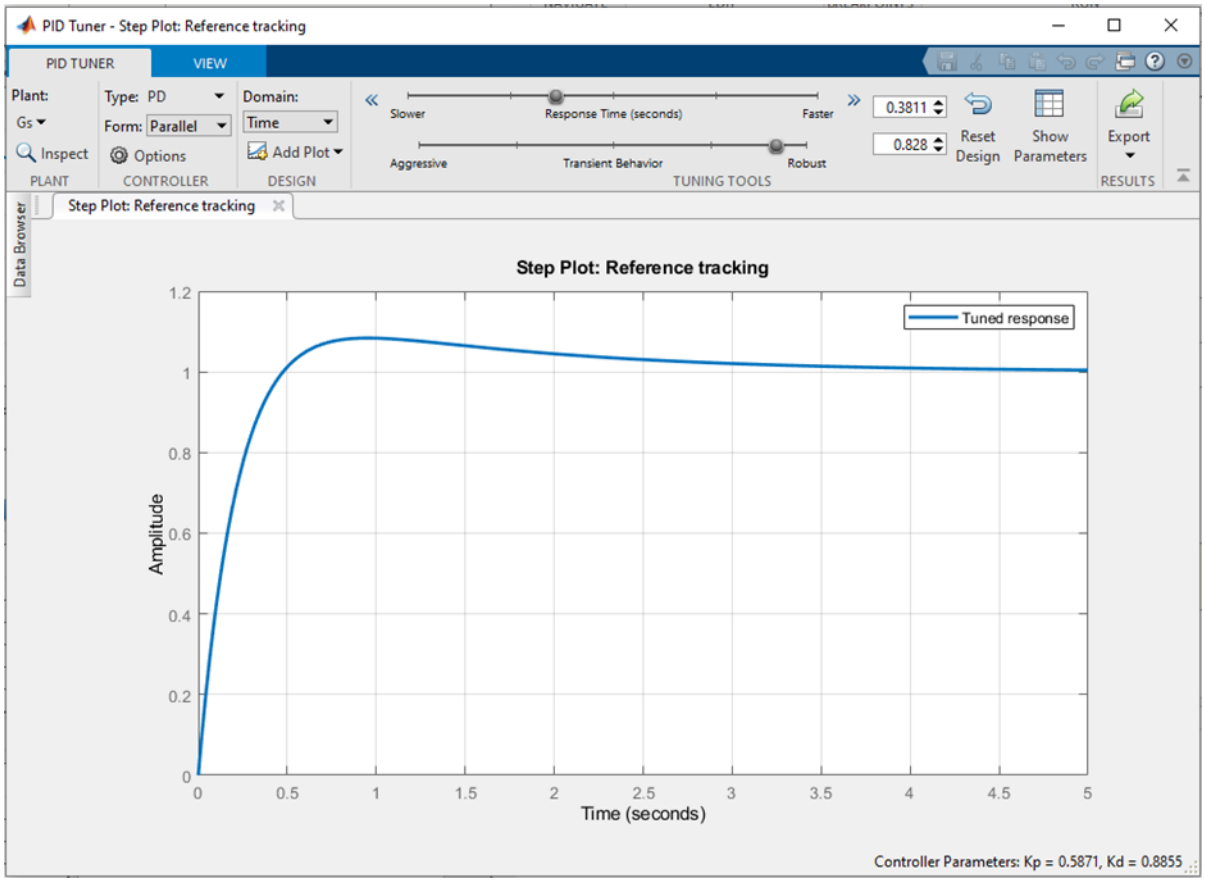

Figure 28: MATLAB's PIDTuner shows the tuned response and initial PID constants.

Controller Parameters

	Tuned
Kp	0.58707
Ki	n/a
Kd	0.88549
Tf	n/a

Performance and Robustness

	Tuned
Rise time	0.325 seconds
Settling time	3.04 seconds
Overshoot	8.38 %
Peak	1.08
Gain margin	-Inf dB @ 0 rad/s
Phase margin	82.8 deg @ 5.25 rad/s
Closed-loop stability	Stable

Figure 29: Dialog box showing the Controller Parameters and performance of the PD controllers current settings.

APPENDIX B

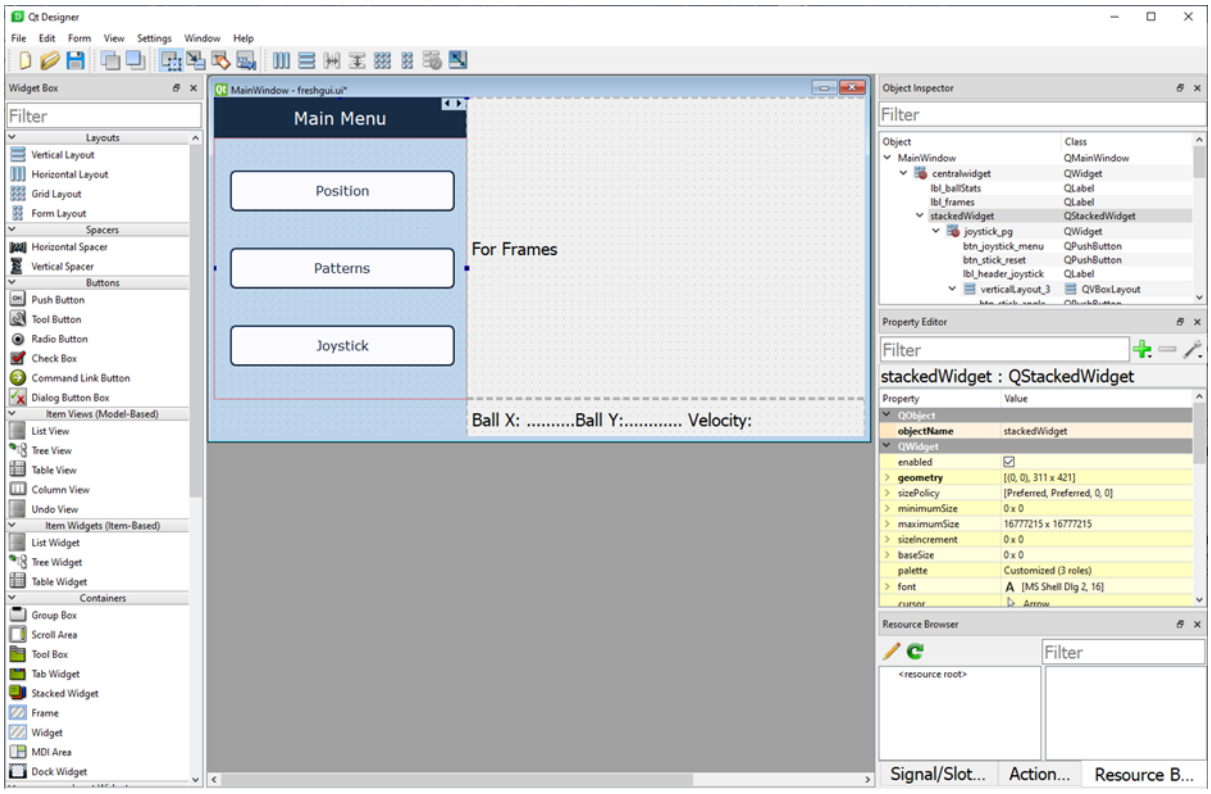

Figure 30: Qt Designer

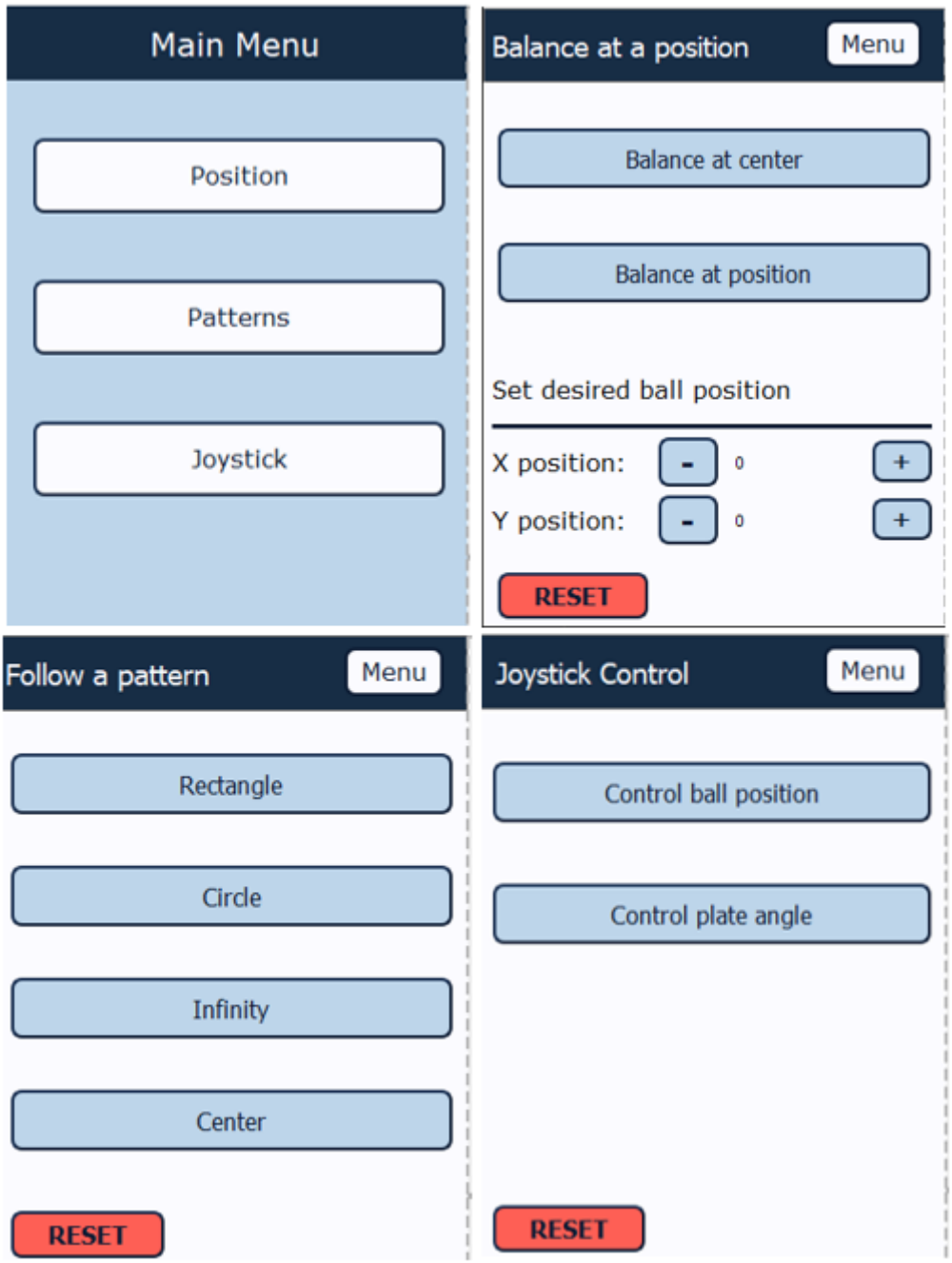

Figure 31: Compilation of the four stacked widget menu screens.

APPENDIX C

Figure 32: Side on view of HMI panel mounted onto hardware

Figure 33: Front view of HMI panel with touch screen displaying the GUI.

Figure 34: Angled view of HMI panel

Figure 35: Proposed design for camera mount.