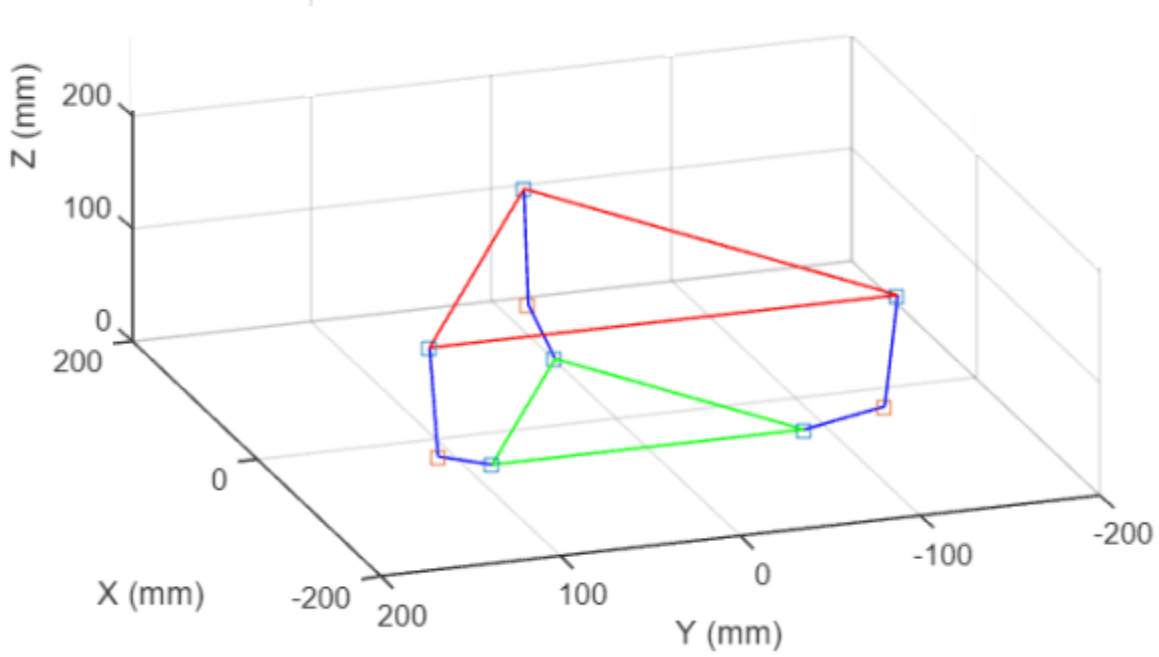
Final Design Report:
Ball Balancing Platform



Final Report Recipient: Dr. Alan Steele (Carleton University Department of Electronics) and **Dr. Hima Dhulipati** (Carleton University Department of Electronics)

Authors: Ayo Owolabi, Ciaran McDonald-Jensen, Ibrahim Shah, Noah Connell, Tristan Laliberté - Electrical Engineering and Engineering Physics students - **Team #3F**

Course: ELEC 3907A

Date Submitted: 10 April 2024

Executive Summary

This final design report for the Ball Balancing Platform project, undertaken by Team 3F from Carleton University's Electronics department, outlines the creation and development of a ball-balancing game. The platform, a practical application of control system principles, consists of a base, ball joints, and a triangular platform. Its motion is controlled by servo motors connected to a microprocessor (Raspberry Pi), with user inputs captured via a joystick.

Throughout the design process, the team explored several approaches and decided on a complex three degrees of freedom design to add a challenging and speedy element to the ball balancing task. Budgetary constraints of $120 necessitated careful consideration of equipment options, leading to the selection of cost-effective yet efficient components such as servo motors and the Raspberry Pi.

In prototyping, 3D printing played a significant role, allowing for quick manufacturing and iterative refinement of components, designed using Fusion 360 software. Challenges in part strength, alignment, and servo performance were addressed through successive prototypes, culminating in an optimised design.

A key feature of the project is the incorporation of an interactive gaming aspect, enhancing user engagement. A camera tracks the ball's position on the platform, and players use a joystick to direct the ball into 'green points' for scoring. This gaming component not only enriches the user experience but also showcases the platform's control and tracking capabilities. The development process utilised the team's prior knowledge in Python programming to create a sophisticated ball tracking system. This system was instrumental in developing the green point game, allowing for real-time tracking of the ball's position and the dynamic placement of scoring points on the platform.

The team's proficiency in software development was further demonstrated in the use of MATLAB for simulation purposes. A MATLAB simulation was created to provide a rough estimate of how the platform would look and function, aiding in the visualisation and planning of the mechanical design.

This project melded theoretical knowledge with practical application, resulting in a sophisticated, interactive ball-balancing platform. The team's adaptability, combined with their software and hardware skills, was key to overcoming challenges and achieving the project goals within budget and time constraints.

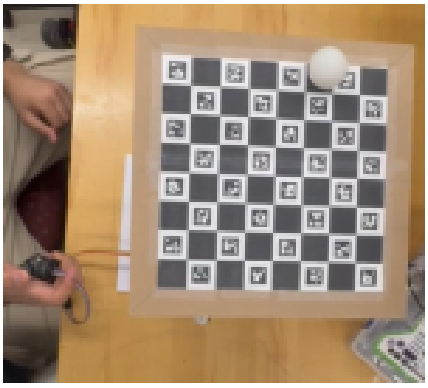
To conclude, this final design report presents the culmination of the Ball Balancing Platform project, executed by Team 3F from Carleton University's Electronics department. The project focused on developing an interactive ball-balancing game platform, leveraging control system principles. The platform is composed of a base, ball joints, and a triangular platform, with its motion controlled by servo motors connected to a microprocessor (Raspberry Pi), and user inputs captured through a joystick.

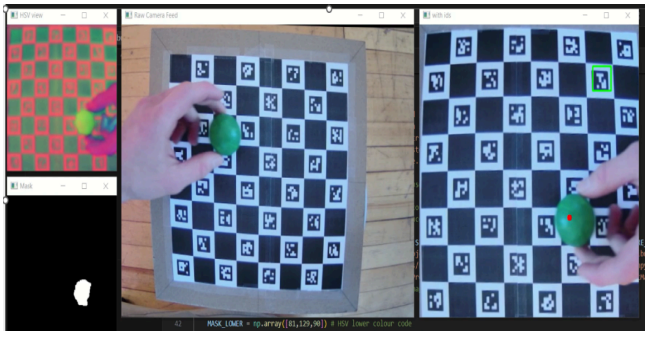
Table of Contents

Introduction

The ball-balancing platform, developed as part of our project, represents a practical application of control system principles. It has three main components: the base, ball joints, and a triangular platform. The ball joints fixed to the corners of the triangle platform are supported by the base. This arrangement allows for controlled movement of the platform.

The platform's motion is controlled by servo motors linked to the ball joints through mechanical connections (linkage connection). The servo motors are controlled by a servo controller, controlled by a microprocessor (Raspberry Pi). User input is then captured through a joystick, and these commands are then processed by the Raspberry Pi to manipulate the movement of the platform, ensuring the ball remains balanced.





We incorporated a game aspect to add an interactive and engaging element to the device. Using a camera, the ball's position on the platform is monitored. Players control the platform using a joystick to get the ball into mini squares (green points) to score points. This gaming feature not only enhances the user experience but also showcases the platform's capabilities in a fun and practical manner.

Alternative Design Details

This section will discuss the unique alternative design details of the ball balancing platform. During the brainstorming phase, our group explored different implementations of the control system for the platform. Our group implemented a platform that can be controlled by using a joystick to balance a ball.

Alternative Platform Design Approaches

There have been many different design approaches created for the ball balancing platform. The easiest implementation is designed with a center joint and a control device titling the end of the slotted rod to balance the ball. The system is considered to be a single degree of freedom that acts like a seesaw. In Figure 1 below, an image depicting this system is shown.

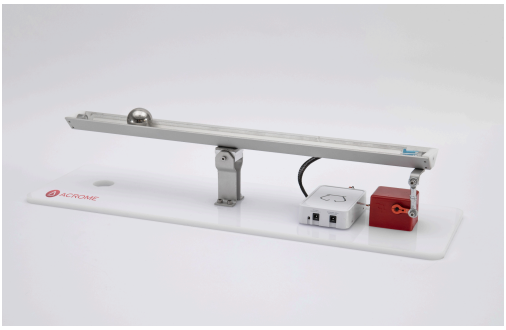


Figure 1. Ball Balancing Platform with one DOF [2]

There is a more difficult implementation that is designed with two control devices titled in the x and y directions. The platform is supported by a joint to allow for stability and maneuverability. Since the system is controlled by two devices, it is considered to be two degrees of freedom (DOF). An illustration of this device is evident below in Figure 2.

Figure 2. Ball Balancing Platform with two DOF [2]

Alternative Position Tracking Methods

As part of the design process, the group looked at multiple tracking methods, including ultrasonic, resistive, and optical. Using an ultrasonic would use at least 2 sensors to measure the position in 2 directions to find the position. On the other hand, a resistive sensor would use a resistive panel with an analog output to find the position. The final method uses a camera to locate the ball. Each method had various advantages and disadvantages; the table below outlines some of them.

Table 1: Advantages and disadvantages of ultrasonic, resistive and optical position sensing

Ultrasonic	Resistive	Optical
Needs multiple sensors	Simple implementation	Complex implementation
Low Accuracy	High accuracy	High accuracy
Cheapest option	Has preset dimensions	Middle price
Possible interference	Highest cost	Multiple camera options
Low field of view	Needs an analog-to-digital converter	High Computation power needed

Table 1 shows the decision to use a camera for position tracking—some of the downsides of the optical method needed to be addressed. A camera is one of the cheaper options and provides the highest flexibility in part options. The high computation power needed is addressed by using a Raspberry Pi 3 A+, which has high computation power and a set of compatible cameras. These cameras simplify the hardware needed to connect to the Raspberry. Implementing a camera-based position-tracking program is still complex, but many resources were available.

The final solution involved taking an image from the camera, undistorting the image from calibration data, locating the platform, applying a perspective transform, and finding the ball's location. The following sections break down the camera calibration, the platform ChArUco pattern, locating the platform and the perspective transformations. Ball tracking is currently being implemented, but this report focuses on the steps to take before tracking the ball.

Assessment of the Design Requirements

After researching the two designs mentioned above, our group agreed to challenge ourselves. The designs above were easy to implement because there are fewer factors to consider such as x and y directions. We decided to implement a platform with three degrees of freedom because it brought another layer of complexity. The design will be controlled by three devices tilting in the roll and pitch directions, as well as adding the ability to move up and down. The three degrees of freedom allowed us to balance the ball at a faster rate of speed.

A big challenge that our group faced was the budget of $120 given by the Instructor. In turn, our options for the type of equipment were limited, therefore with a larger budget our group could make this platform more robust.

Exploring Equipment Options

During the beginning days of the design process, we discussed as a group how we could control the ball well within the budget. We began by choosing between the stepper motor and the servo as the type of device that will move our platform. We considered stepper motors because of the precise movements, but the price is about 16 dollars CAD per motor. Keeping in mind the budget for this project, we could not afford to pay approximately 48 dollars CAD for three motors. Therefore, we discovered that servos are a cheaper option and also provide accurate movements with a quicker reaction speed of about 60 degrees in 0.09sec at 6V. The servos retail at a lower price of 14.38 dollars CAD for a pack of five.

Figure 3. Servo Controller

Our original plan was to use a feedback control loop to balance the ball on the platform using some type of technology for tracking. We could not implement the feedback control loop due to the limited time. We had a long discussion about resistive touch panels, ultrasonic sensors, and a camera to track the ball's position. After performing some research on resistive touch panels we found this application to be ideal, but the high cost eliminated it from consideration. The ultrasonic sensors were very difficult to record the position of the ball because the sensors would interfere with each other. We decided to opt for the camera as our solution to track the ball since it comes at an affordable cost. Our team was worried that using a camera would be too computation-intensive and slow. After some extensive testing, the camera outperformed our expectations. The camera provided us with the exact coordinates of the ball based on the reference of a charuco board. The charuco board was placed on the platform to allow the software to track and measure the location of the ball. Our group also has previous experience using image processing programs, such as OpenCV, which made the use of a camera more feasible.

Figure 4. Camera and the Charuco Board

To power and control all these devices, we required some large computation power. We were choosing between an Arduino or a Raspberry Pi to compute all the information. The Arduino unit seemed too small and required to be programmed using C or C++. Using a procedural programming language is difficult due to features such as pointers. As a result, C or C++ is a faster language for extensive computations. The Raspberry Pi is a unit that our group had previous experience with within a class called mechatronics. To start, we were familiar with the type of language used to program the Raspberry Pi called Python. Using an object-oriented programming language makes it easier to construct code, but is often slower compared to C or C++. As it turned out, the computation power of Python and the Raspberry Pi exceeded our expectations and were able to track the ball very fast and accurately.

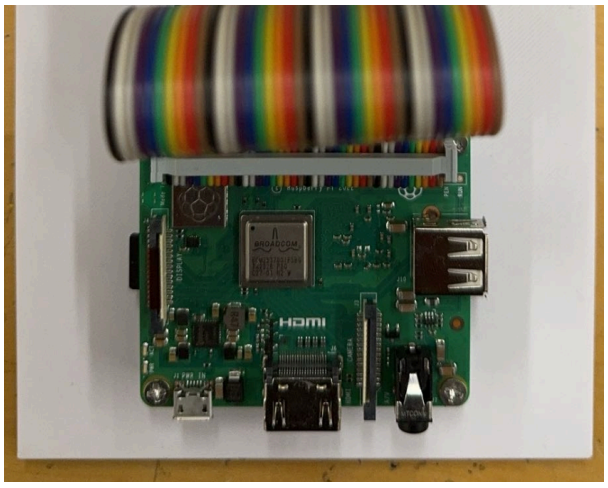


Figure 5. Raspberry Pi 3 Model A+

Prototyping Refinement

Throughout the hardware phases, we agreed as a team to 3D print all the parts since a few members owned a 3D printer. Using our 3D printers allowed for parts to be printed very quickly to optimise part

design. Each part was designed on paper to visualise the finished product. The parts were designed on a computer-aided design software called Fusion 360. Our group designed three prototypes and during each, we found alternative solutions to each part.

There were many issues within the prototype such as low infill, poor servo alignment, poor part quality, etc. Infill is a term used to describe the strength of the part, therefore more infill means more plastic. The infill is adjusted in a software called a slicer that produces a set of commands for the 3D printer. The setting infill was adjusted to 100% to improve the strength of the parts.

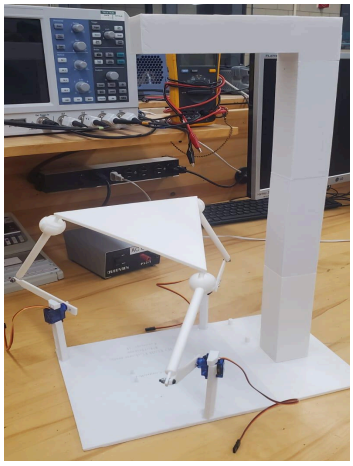


Figure 6. First Prototype

The second prototype had improved significantly by strengthening the servo mounts and ball joints. The servo mounts were improved by adding a triangular support in the direction of the servo forces. Also, two smaller triangles were added on either side to support the mount. The lower ball joint was improved by adding two tabs to add strength to the linkage. As a result a mistake happened, the joint could not go down because it would hit the lower part of the ball joint.

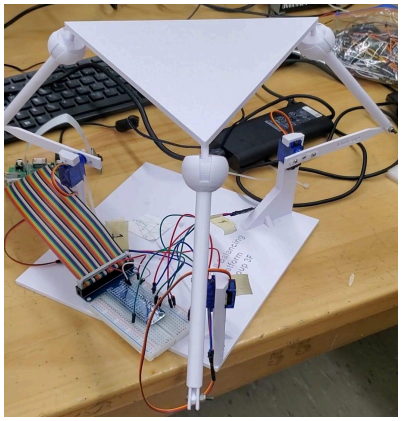


Figure 7. Second Prototype

The final version was improved with shorter linkages from 10cm to 5cm. The reason for this is that it provided the servos with more torque and less leverage on the servo. Once this change was made the testing went very well and each servo performed better.

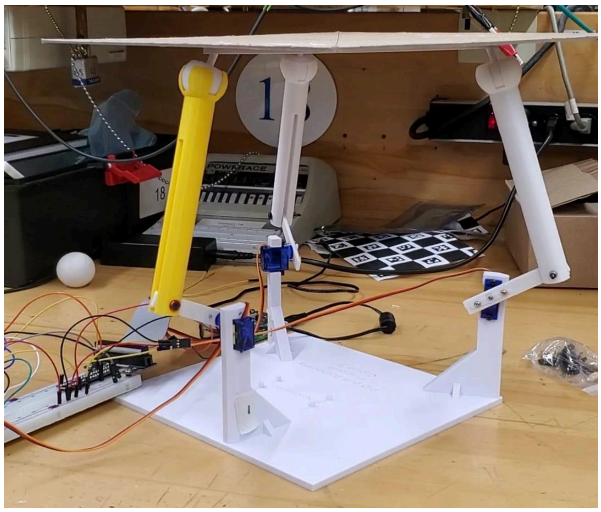


Figure 9. Ball Balancing Platform with three degrees of freedom

Our design approach was affected by the economic downturn due to inflation in housing and an increase in the minimum wage. Therefore it was tougher to afford components because everything costs more. As a group, we tried to collect as many resources as we could from the spare parts in the laboratory. We also kept the environment in mind by using plastics that can be reused and recycled. Future groups can reuse these parts for their project to eliminate waste in our landfills.

Technical details

Platform Motion Calculations (Tristan)

The platform geometry decided upon was a triangular configuration with three servos placed at the apex and connected to the platform with a ball joint at the corresponding apex on the platform. This creates a complicated problem where each servo affects the orientation of the platform independently. To simplify the calculations, a desired platform orientation is first set and used to calculate the angles needed at the servos. This section in the report is only to provide some background to the need for these calculations. A full derivation can be found attached in Appendix ADD LETTER.

The end results from the calculations are a set of matrix and vector operations to convert from the platform's orientation to a vector from the servo to the platform attachment point. This vector to the

platform can then be used with a few trigonometric relations to determine the angle needed at the servo. The calculation lay the groundwork for the simulation explained in the next section to validate platform motion and later the calculations implemented in Python on the Raspberry Pi to move the servos.

Platform Motion Applied (Ciaran)

Finalizing platform calculations and making simulation

There were two options of what platform calculations should be put onto the Raspberry Pi. The calculations done in the above section are very fast to compute but they made the assumption that the center of the platform will be right above the center of the base, and they do not add the constraint of both parts of the leg having to be in line with the center of the platform (due to the type of joint between the two parts of the leg). This means the above calculations have a non-physical assumption and might be slightly off from the real value. The other option is something that Ciaran was working on that involved defining 9 positions (x, y, z of each corner of the platform), and then the 6 constraints based on the types of joints and physical aspects of the system, and then defining the 3 values, pitch, roll, and z to get three more constraining equations. Then numerically solving the system of 9 equations to determine where the positions of the corners of the platform would be. Numerically solving this system was slow but accurate. As this is a very time critical system, speed of the calculation is very important, so the numerical solution was not feasible to run on the pi. This meant that it was necessary to test the above section's calculations to see in what region the approximation is valid, and what range of motion should be maintained to stay in the region the approximation is valid, giving a very close result to the true value. In theory a comparison between the numerical true solution and approximated fast solution could have been done to achieve a quantitative range where the approximation is valid, but instead the quick calculations were implemented in a MATLAB Simulation to be able to visualize what was going on (Tristan started by making the basic simulation and then Ciaran added onto it).

This MATLAB Simulation had three main advantages. As mentioned above it allowed the ability to test where the approximation was valid, and where it was not valid. It also was helpful for checking that the calculations were done correctly without typos (debugging), visualizations are very helpful for debugging. It also was then used for choosing platform parameters, sizes of all of the components (more on this later).

The simulation was created by first turning all of Tristan's above calculations into code, performing the desired calculation for each of the three legs, giving the position of each corner of the platform (the '*l*' vector) based on the selected pitch, roll, and z-offset. The calculations were then added for the positions and angles of the linkage and ball joint arms using basic trigonometry to determine their positions based on the motor angles calculated and the physical constraints. This means the motor angle was computed using the fast calculations that might be slightly off, then the visualized legs used the physical constraints. This meant that if in the visualization the legs did not meet with the platform, then the approximation was causing issues and was not valid, but if visually the legs join with the platform, the approximation is returning the same result as the true physical system. All of the physical things were then plotted on a 3D plot to be able to visualize the system. The full code is well documented walking through what each value

represents and is shown in the github in Appendix D (files calculateRotationMatrix.m and Stewart_Platform.mlx).

Figure C1 shows the usage of the MATLAB Simulation. MATLAB also has the ability to pan and rotate around the plot, making it easy to see.

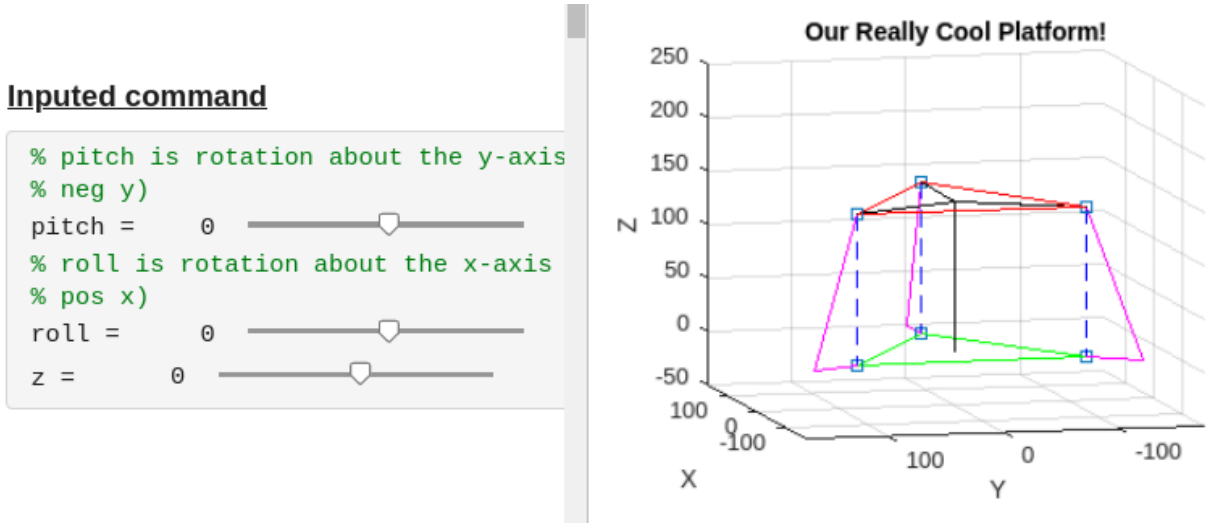


Figure C1: MATLAB Simulation. Left: selection of platform parameters the simulation does calculations based off of. Right: Visualization of the platform's position. Green: solid base, servos are at edges of platform. Red: triangular platform. Black: non-physical reference lines. Dotted Blue: non-physical 'l' vector, displacement from base corner to platform corner. Pink: physical legs composed of linkage and ball joint arm, drawn with physical constraints, if it doesn't meet the platform corner then an assumption that was made is off

As stated, the position of the linkage and ball joint arms (pink lines in Figure 1C) was calculated using the distance of the platform corner (from the above calculations '*l*' vector) and only the physical constraints. This helped identify that for almost every angle that could be desired, the approximation was a really good approximation and the legs matched up almost exactly with the platform. It was found by trying some different platform positions that the approximation only broke down at very extreme angles, an example of this is shown in Figure 2C. This only reinforced the teams idea that the platform should avoid extreme angles as it would get in the region the approximation starts to break down and the calculated angle will not be the actual angle, as well as the fact that for extreme motor angles, the forces on the legs would get more extreme and be more likely to deform or snap part of the leg. Because of this, sizes were chosen such that the desired range of motion could be achieved without exceeding 60 degrees from the horizontal. As well, a hard limit was added in the software so that the motors would throw an error if they were commanded to go to an angle past 60 degrees (Thank you Tristan).

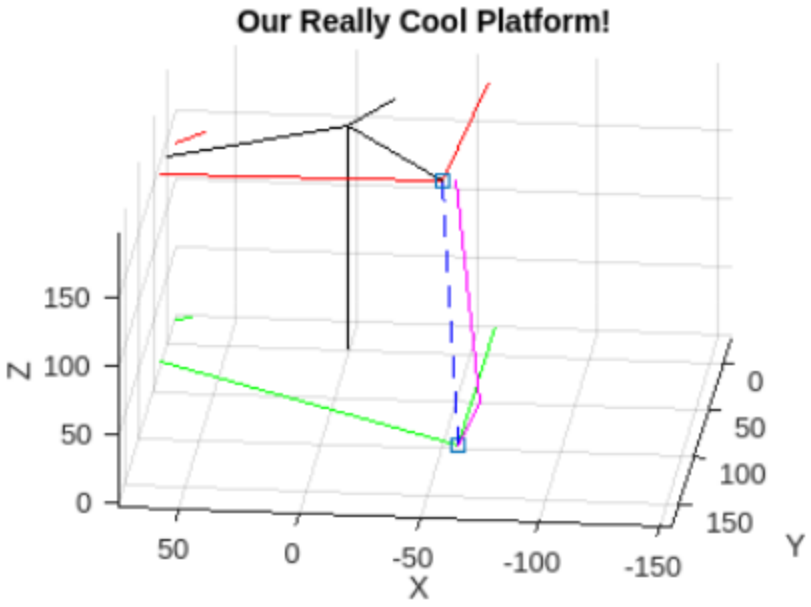


Figure 2C: Zoomed in image of platform at extreme position (roll=20degrees, pitch=6.5degrees, z-offset=20mm). It can be seen that the pink physical arm following physical constraints does not meet exactly the corner of the platform. This indicates that at this extreme angle, assumptions in the calculations are breaking down and caution should be used

Choosing Platform Parameters

Another task that was performed was choosing platform parameters. Full detail of this process can be found as detailed in Ciaran's Individual Technical Report [C1]. Once values were chosen, they could be put into the MATLAB simulation and tested to check that they do indeed give the range of motion that was expected, and that the entire desired range of motion are in positions where the math approximation is valid. After checking the parameters, the parameters were communicated with Noah who then designed the physical components using the selected parameters.

Platform Motion Code to Control Raspberry Pi

There were a few software components that had to be integrated and run together on the pi to have the entire system work together. This section describes the details of the three executable scripts that were the three different 'modes' of the platform. These scripts used three main functions/classes that must be briefly described. All of these files can be found in the github that is linked in Appendix D.

calculateMotorAngle.py was a function made by Ciaran that contained the same calculations that were done in Tristan's original math and that were moved to and checked by the MATLAB simulation. This

calculated the three motor angles to achieve a desired pitch, roll, and z-offset. PlatformController.py was a class made by Tristan that set up the servos and had some helper functions to use calculateMotorAngle.py and communicate to the servos to get them to move to the correct position given a desired pitch, roll, and z-offset. Joystick.py was also a class made by Tristan and Ciaran that made helper functions to read the input from the joystick. These three files were used by the following executable scripts that are described.

The first executable script is cameraControlLoopDemo.py. This script was designed while the plan was to use the camera to use the ball position as feedback and have the platform be self-balancing, maintaining the ball at a desired position. This implements a basic PID control loop using a proportional, integral, and derivative term of the error of the ball's position relative to the desired position in both x and y. Every loop the amount of desired correction to the velocity (AKA the desired acceleration AKA the desired angle) is calculated using the PID terms, then the angle of the platform is set to get this desired correction. All calculations had to be based around the amount of time since the last run of the loop, since the PID terms are based on time and this allowed it to run as fast as it can and avoided slowing it down by having a specific amount of time per loop. This script was never used or finished as a pivot away from the camera to a joystick happened as is discussed later in the report.

The second executable script is joystickDemo.py. This is a very similar control loop to cameraControlLoopDemo.py but instead of using the PID terms based on the ball's position using camera input, it uses the joystick as input. The joystick comprises of two variable resistors, one in the x-direction, one in the y-direction, and it also has a button that can be clicked. The code was set up so the x-direction resistor controlled the roll, the y-direction resistor controlled the pitch, and pushing the button raised the platform up some distance. This script was what was mainly used as our final product, allowing the user to control the platform using the controller, turning the project into a game to try to balance the ball.

The third executable script is platformDemo.py. This did not use any inputs. This script was intended to showcase the movement of the platform and see how it is capable of moving. It was structured with 'behaviours' which is one type of movement that it would follow until some time limit, then it goes to the next 'behaviour'. It was designed so it is easy to quickly add another behaviour to showcase some other sort of movement. It also kept track of the amount of time each loop and each behaviour took so the behaviours could do calculations based on how much time had passed, allowing smooth continuous movement. Behaviour 0 just went to preset angles to show its accuracy and symmetry in each direction. Behaviour 1 showed continuous smooth change of angle, staying at one height and smoothly rotating around. Behaviour 2 showed continuous smooth change of height, staying flat and moving up and down. Behaviour 3 combined 1 and 2 and showed the platform moving up and down while rotating in a circle, it looks really cool.

Details of all of this code can be found in the github linked in Appendix D, it is documented with comments explaining how the code functions.

There was originally concern about how fast the code would run as this is a very time critical system. Python was used as we as a team believed that we would have an easier time working with Python and getting it working, even though it is known that C would run faster. As such we were prepared to switch all the code to C if the Python code was running too slowly. If using the camera, then to balance the ball

the control loop would need to have a quick response to feedback, and if using the joystick, then for it to be an enjoyable smooth experience the control loop must be fast. Doing some tests, our final joystick control loop takes about 2.7-3.5ms per loop, this is about 350 runs every second, where each loop reads joystick input, does the motor angle calculations, and then sends the signal to the motors. Originally the stretch goal was to be able to run at 60 loops per second, as this is a typical computer update rate that feels smooth. This control loop runs incredibly fast, and definitely is fast enough to be integrated with a camera to balance a ball.

Note: It should be noted that the pitch and roll are not quite independent as they are being treated in the code (roll giving x-correction and pitch giving y-correction), the pitch has less and less of an effect given larger roll based on how the calculations are done. But for small angles they are practically independent and as only small angles (less than 20 degrees) are used, treating them as independent is a good approximation.

Platform Design (Noah)

This section will go into depth about the design of the platform. The process began by illustrating each component of the platform on paper along with the dimensions and key features for each design. At the beginning stages, it was important to keep good communication with the team because they provided crucial feedback on my illustrations. This section will also discuss how each component was constructed in the computer-aided design (CAD) software called “Fusion 360”.

Prototype 1

The first prototype allowed the team to visually see design flaws and discuss improvements. The assembly process of the first prototype demonstrated how weak the design was. The servo mounts would not stand vertically because of the low amount of infill. The term infill is described by the amount of material used to construct the inside of an object. The infill amount was changed within a program called a slicer that is used to create a set of instructions for the 3D printer. Also, The forces applied by the servos were an issue because there was no support on the mounts. The joint between the linkage and the ball joint was thin and required more material. Also the alignment of the arms was not referenced to the center of the equilateral triangle. There were many lessons to learn after the first prototype such as fitment, quality and structural issues.

Figure 10. First Prototype

Prototype 2

The second prototype was a better design as it incorporated the improvements from the first prototype. The servo mounts had been redesigned with added support in the direction of the servo force. The triangular supports were designed to distribute some of the weight onto the base. Two smaller triangular supports were added on either side to incorporate any external forces in the horizontal direction. The joint between the linkage and the ball joint was improved by the addition of two bracket supports. The disadvantage to this design had limited moveability in the downward direction due to the ball joint and the linkage colliding. The alignment to the center of the triangle was improved but the measured values were incorrect. There were many lessons to learn such as communication and usability.

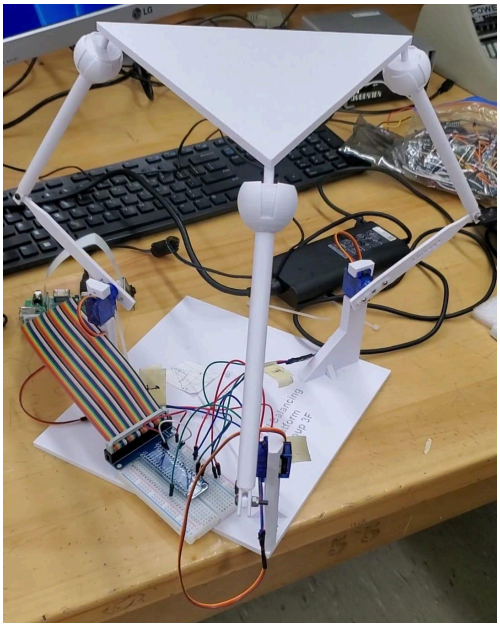


Figure 11. Second Prototype

Final Design

The final design included all of the design considerations into one functional ball balancing platform. The design incorporated the correct alignment to the center of the equilateral triangle. The ball joint was redesigned to make the platform more rigid and reduced the amount of movement within the screw holes. The linkage was modified to create more torque for the servos by the use of shorter links.

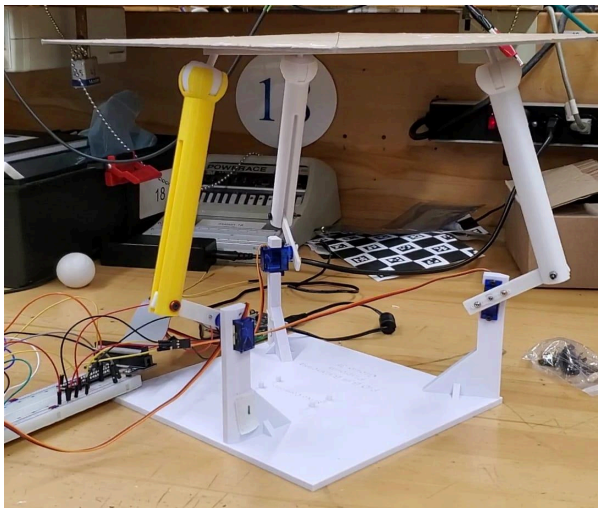


Figure 12. Final Prototype

Base

The design began by drawing an equilateral triangle with sides of 20cm in a 2D sketch. From here, the feature within Fusion called extrude was used to make the 2D sketch become a 3D object. Then, the triangle was extruded by 1cm to make a strong base. The triangle points produced the exact placement for each servo. Next, the servo mounts were created on top of the existing triangle by drawing a square (2 x 1 cm) and extruded it up 10.3cm. Cuts were made on the sides for the servo by drawing a square (2.33 x 1.4cm) and then detruding it. Detrude is the opposite of extrude, therefore removing material from the 3D object. Then, circles (2.8mm diameter) were created for the M3 screw to mount the servo in place. One more cut was made along the side to remove the material hanging off the triangle. This was done by a line created on the top surface of the mount and with the use of the feature called split body. This feature splits the shape into two bodies, then the body can be hidden. The base was made into a square (21x22cm) to account for stability.

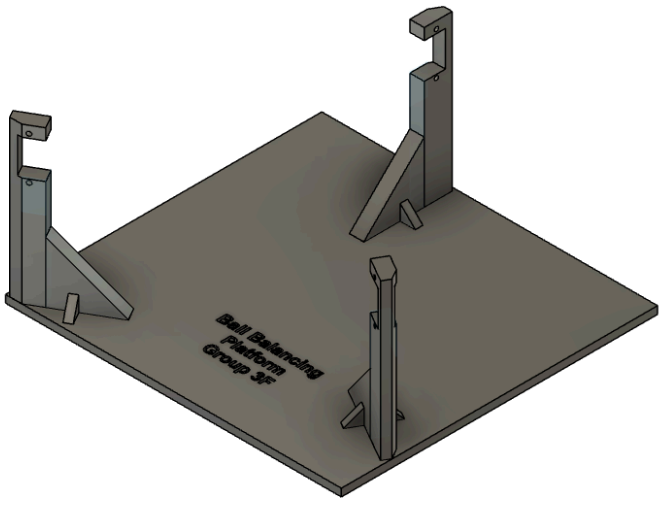


Figure 13. Image of the base

Platform

The platform was created as a triangle to help reduce the use of plastic. The platform was created using a 2D sketch of the equilateral triangle (sides: 20cm). The triangle was extruded by 1cm and three circles were created and also extruded by about 1 cm. Next, a sphere was created above the cylinder by the use of the sphere feature. The calculated values indicated it was optimal to position the sphere at 20cm, the same as the servo. The spheres were slanted with the use of the move and copy feature.

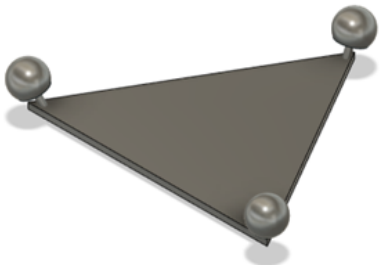


Figure 14. Image of the Platform

Linkage

The linkage was designed to incorporate factors such as torque and mobility. The added support provided by the two screws on the linkage kept it rigid, therefore maximising the torque. The linkage was created with a 5cm line because at either end of the line was the location of the axis. The rectangle was extruded 3mm and then created circles on top of the linkage. The first circle was made to be 3mm and was required to fit the M3 screw at one end of the 5cm line, and the other was required to have a diameter of 7mm circle to fit on the servo arm. Two more circles were drawn approximately 1.1cm away from the centre of the 7mm circle. These two smaller circles have a 2mm diameter and are used to fasten to the servo arm.

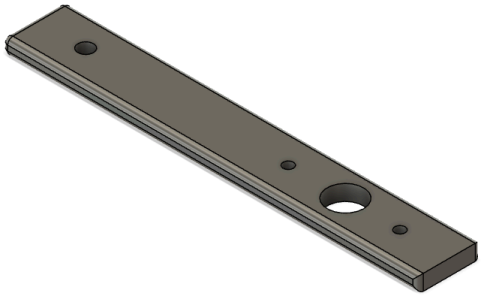


Figure 15. Image of the linkage

Ball Joint

The ball joint was designed to incorporate factors such as support and mobility. With the use of only one screw, the design was strong and flexible. The design of the ball joint was created as a cylinder and then added a sphere on top. Next, Created another plane using the feature called offset plane. This created a plane further up the sphere and made a cut using a split body. Next, A copy of the sphere was created and then was used as a body to split the inside of the sphere. Once this was completed, the joint and ball were two separate components. Therefore, the cut outs of the ball joint were created by a 2D sketch by extruding two rectangles. The circle was extended by extruding the cylinder to 15cm (centre of the ball joint to centre of the screw hole). Also, a slot was created by extruding the center cut out by 3mm. At the position of the lower 15cm a 3mm was created and extruded.

Figure 16. Image of Ball Joint [5]

Camera Tracking (Tristan)

This section will go into details into the implementation of the camera tracking using OpenCV. This section discusses the ChArUco board pattern, calibration, point projection, perspective transformations, and color filtering object tracking. As a reference, see Appendix E for the fully commented code used to implement the tracking.

ChArUco Board

A ChArUco board is a mix between a chessboard pattern and an ArUco board, that allows for the dimension and orientation of the board to be found. An ArUco pattern is quickly detectable and reliable but lacks accuracy when locating the corners of the pattern [T1]. On the other hand, the corners on a chess board pattern can easily be found and allows for accurate subpixel approximations of actual corner position [T1]. The figure below shows the different patterns.

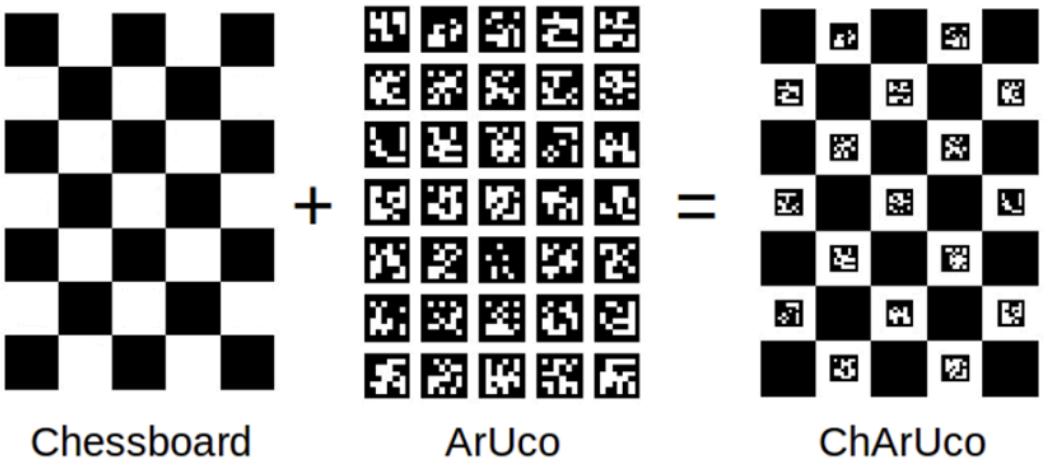


Figure 1: Combination of patterns with a chessboard, ArUco board and a ChArUco board [T1].

The ChArUco board acts as a reference in the real world that can be used to define positions relative to the board.

Calibration

The ChArUco board was also needed to calibrate the camera used for the position tracking. To take a picture of the real world, the three-dimensional information of the world is projected onto the two-dimensional imaging plane of the camera. This 2d image will have some distortions from the camera lenses to project the image onto the sensor and have other sources of distortion. Having an undistorted image is quite important for measurements as the distortions can cause large amounts of error.

Correcting the camera distortion requires a camera matrix and a distortion matrix to convert the image to an undistorted image. There are two transformations that are taking place when taking an image, a rigid transformation and a perspective transformation. The process is shown below.

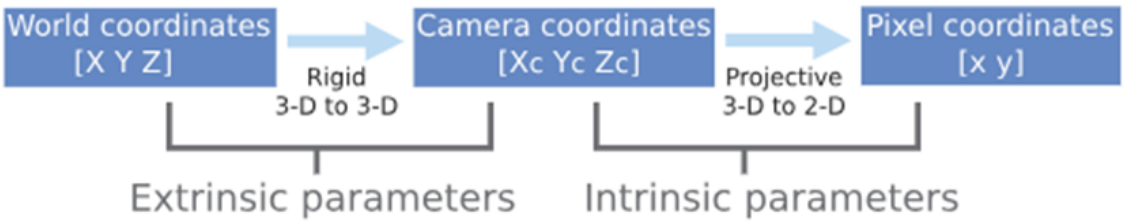


Figure 2: Transformations applied to the real world to an image [T2].

The extrinsic properties are the translation of the camera and the rotation of the camera [T2]. This intrinsic properties are that of the camera and is defined by the following matrix:

$$\boldsymbol{K} = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \quad (1)$$

The components consider the focal lengths of the lens, is the skew of the pixels, and accounts for the center of the sensor in the camera. This matrix considers the properties of the camera but not the distortion caused by the lens itself. The distortion of the lens can come from radial distortion of the lens, or tangential distortion from misalignment of the sensor. Figure 3 shows different types of distortion.

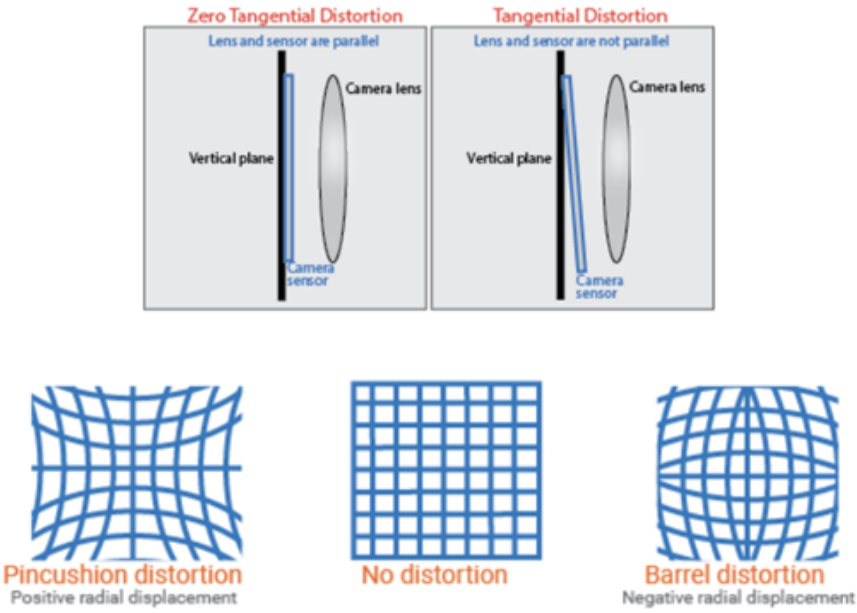


Figure 3: Types of distortion accounted for in the distortion matrix [T2]

Calculation of these parameters is quite complex but necessary to have accurate position measurements. These parameters were calculated with OpenCV functions and multiple images of a calibration object, from multiple perspectives and positions.

3-Dimensional Point projection and Perspective Transformation

To measure the position of the ball on the platform, first the corners need to be found and then the perspective of the image is changed to be looking perpendicular to the platform. The corners are found by projecting points from 3D space onto the platform's ChArUco board pattern. As was explained above, the board's pattern is predefined and the orientation can easily be found using the ArUco markers. A big advantage of the ChArUco board over other images is the need for only 6 markers to locate the corners [T3]. The corners of the board can be measured with a ruler to find the exact positions and then put into OpenCV to project them onto the board. The black dots on Figure 4 show the estimated corner locations

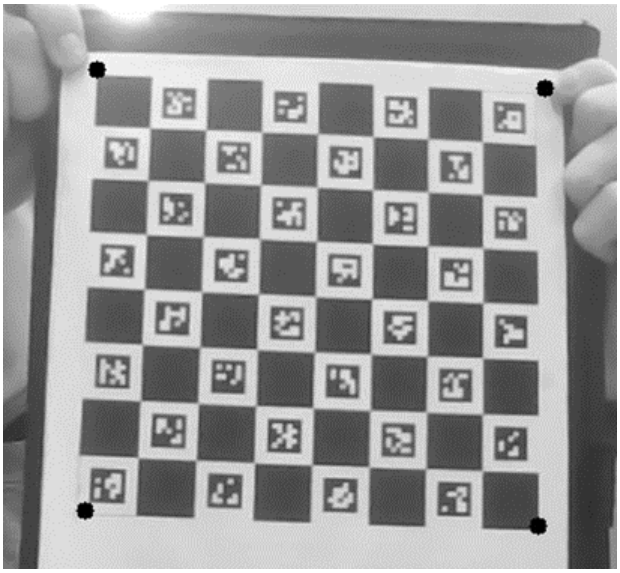


Figure 4: Estimated corner locations using OpenCV projection

The image can then be transformed using the calculated orientation of the board to fit it entirely of the screen. The perspective calculations mapping from a source image to a destination image. For our project, the source image is defined by the corners of the board and the destination is a new square image [4]. The perspective calculations are also done in OpenCV, which calculates the transformation matrix and applies it to the image. Figure 5 shows the corrected image.

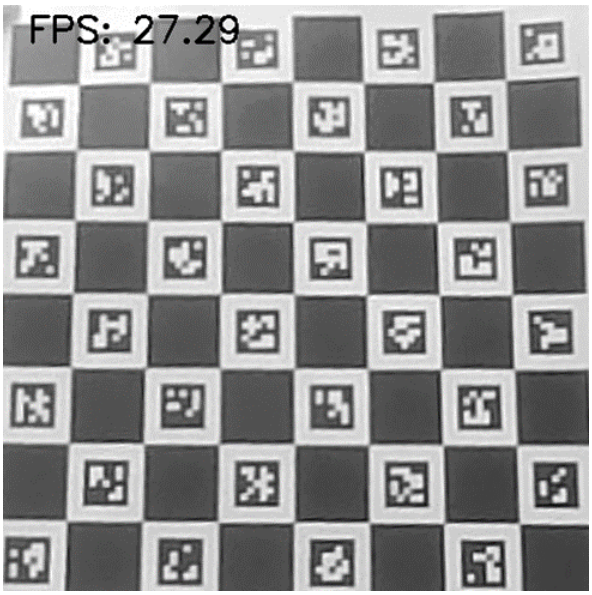


Figure 5: Corrected perspective image

Colour Filtering Feature Tracking

After the perspective of the transformations were performed, the position of the ball could be found. The best method was found to be a colour filtering method, where the background of the image was removed, leaving only the ball in the image. With only the ball left in the image, the centre can easily be found. The steps to locate the ball are as follows: apply a Gaussian blur to the image, convert the image to Hue Saturation Value (HSV) space, mask colours inside specified range, find the centre of the largest area. All these steps can be seen in the commented code in Appendix D.

The first step involved applying a Gaussian filter to the image. This is a common step in image processing in most tracking algorithms. Below is an image where a gaussian blur has been applied to help track a ball.

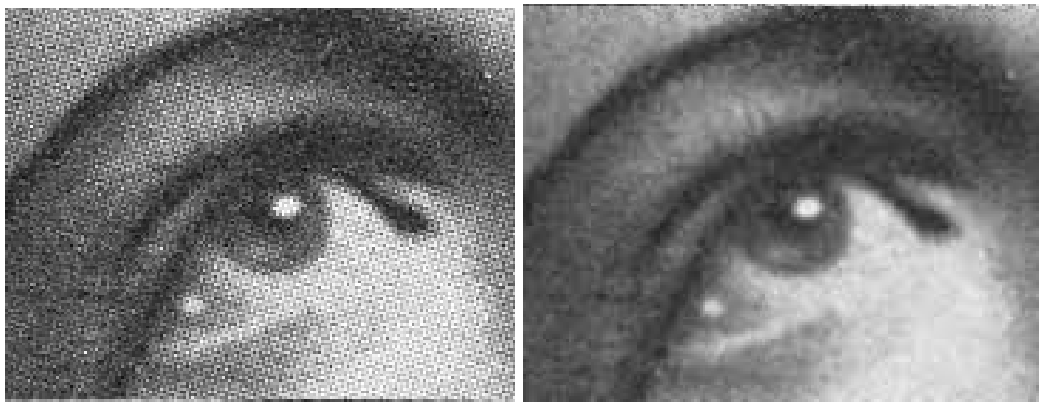


Figure #: Picture of an eye with a gaussian blur applied. The original in on the left and the blurred is on the right[1]

The blurring in this figure has multiple roles, it can be used to remove noise in the image by normalising the pixel intensity distribution. The reduction in noise allows for smoother features that can more easily be tracked. This can allow for a lower quality camera to be used for tracking without the issues of noise from the cheaper sensor affecting the quality of the tracking. The normalisation of pixel intensity also helps to reduce variability from lighting. This allows the algorithm to work in a variety of conditions. Finally, the Gausian blur also helps to create more defined features for tracking. From the Figure above, the right image shows a very noisy image, and it is difficult to find the edge of the pupil. After the Gaussian blur, the edge of the pupil is more defined and easier to locate.

After the gaussian blur, the image is converted to HSV colour space. This space defines each colour with three values, a colour value from 0 to 179, a saturation value from 0 to 255 and an intensity value from 0 to 255. The colour value is a mapping of all the colour hues to a numerical value. Together, the three numbers give a spatial coordinate in HSV space. The HSV space can be pictured as a cylinder in 3d space, with the angle corresponding to the Hue, the radius corresponding to the saturation, and the value corresponding to the height. With this definition, a volume in the cylindrical HSV space can be defined by finding the upper and lower HSV values. For this project, the upper and lower colour values of the ball were found using a helper program to visualise the removed area. The environment has a significant effect on the limits in HSV space. Testing was done in different lighting conditions and bounds set in one condition would not work when the platform was brought to a different room. The Figure below shows the HSV picture from the camera.

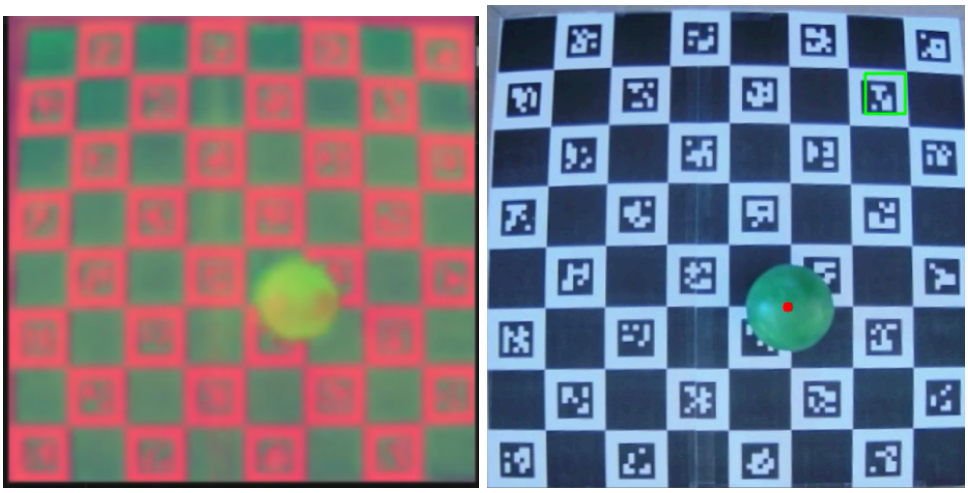


Figure #: (left) HSV colour space image of the ball (bright green) on the ChArUco board pattern. The blurring can also be seen in the chequered pattern. (right) the original image.

This figure also shows how similar colours can be filtered out with the value and saturation. The image on the left shows that the black of the checkerboard shows up as green in HSV space. The filtering is able to distinguish between different shades and intensities of green.

The next step in the process was to apply the colour filtering explained above. The end goal was to generate a binary mask (black and white) with the white region containing the ball and all other features in the black region. There is also a need for post processing to remove background noise. The first iteration of the mask is made by simply assigning a value of 1 to pixels inside the HSV bound and assigning a value of 0 to pixels outside the bound. This first step introduces some noise as individual pixels can fall inside the bounds, even if the surrounding pixels are not. This leads to a “salt and pepper” effect where the image is dotted with black and white spots. This noise will lead to larger computation times in later steps of the program, which will be clear in the following section. This can be removed by tightening the HSV bound, which is not ideal as this could filter out parts of the ball as the lighting changes. To fix this issue, an erode and dilate process was performed. This process helps to remove noise in the background, while keeping the main features. The erode step involves passing a shape over all the pixels in the image and assigning the minimum value to all pixels in the shape. This reduces any white dots in the background of the image. However, the erosion also distorts the sharp edges of features, similar to the natural erosion of rock. To counteract this distortion, a dilation step is performed. Dilation works similarly to the erosion, but instead assigns the maximum value in the brush region. The dilation helps to remove small holes in the object. The dilation process also helps to correct distortions of the edges caused by the erosion. The end product of these operations is a black and white image with a prediction of where the ball is located in the image. The figure below shows the mask with two passes of erosion and dilation versus no erosion and dilation.

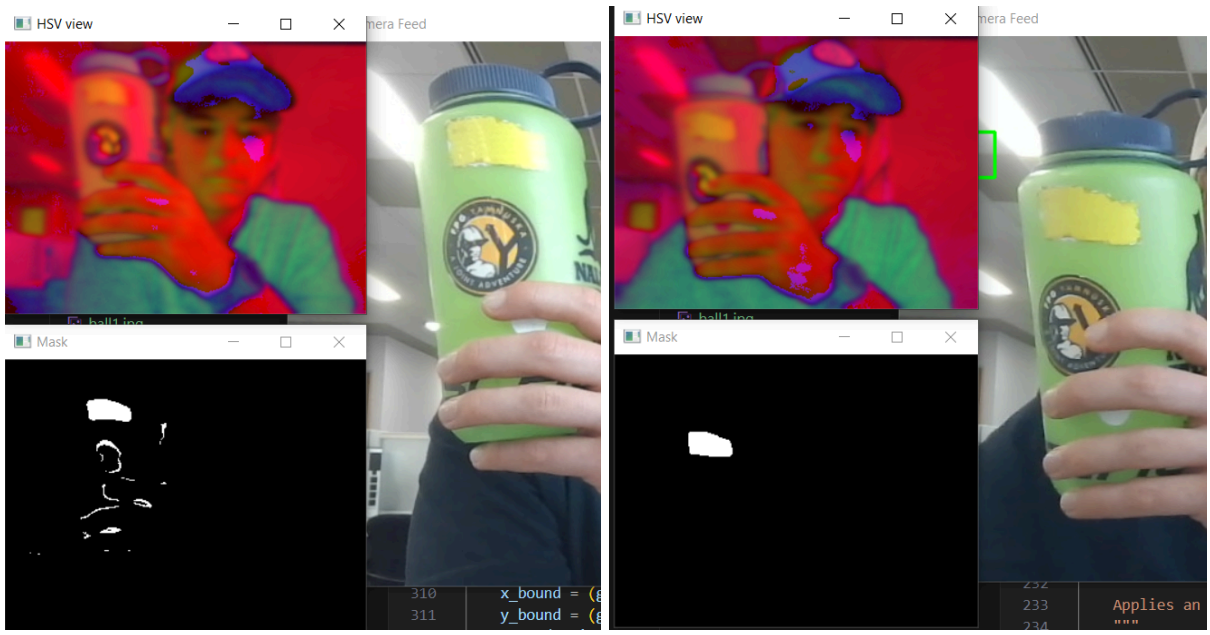


Figure #: Comparison of 2 erosion and dilation passes (Right) and no passes on an image of a water bottle (Left). The colour filter was set to filter the yellow HiVis tape on the water bottle.

The figure shows how the noise from the original filter can be eliminated with multiple passes of erosion and dilation. There is a limit to the number of passes that can be done. With more passes, features can be lost and there is an increased computation time.

The final step was to locate the centre of the largest white region in the mask. This was done by locating a contour for all white features in the image. This operation creates a list of all the shapes in the region. This list can then be filtered to determine the largest contour area. An iterative search is performed to all contours and the area is calculated and compared to the previous largest contour. This step can be computationally difficult if the image is noisy and many white regions are present. The erosion and dilation step explained above helps to reduce the number of contours that need searching. Once the largest contour is located, the centroid of the shape is found using a moment calculation built into OpenCV. The moment calculation returns the total area and the sums of the contributions in the x and y direction. These can easily be used to locate the centre pixel location. The final operation involves returning the contributions in each direction divided by the total area, which gives the pixel location of the centre.

Each step to track the ball's position has been outlined and a complete commented code can be found in Appendix E.

Hardware component Base [Raspberry Pi and Camera] (Ayooluwa)

During the preliminary design and inception of our project, we discussed various dimensions for the platform itself and the base. However, there was a need for an extra base for components such as the Raspberry Pi, the camera, and it’s mount. Using the final dimensions of 21 by 22 cm as a reference, we concluded on the need to create a base for the other components such as the raspberry pi and camera to make up for the already exhausted space available on the platform base.

The hardware component base is specifically designed to accommodate the Raspberry Pi model 3A+. The initial design didnt account for the precision and accuracy of the screw holes and cutouts. However, the base below shows a great deal of precision and accuracy for the screw holes and cutouts to ensure that ariflow surrounding the Raspberry Pi is enhanced and overheating is minimized as much as possible when operating for extended period of times. Furthermore, the cutout and design of the Raspberry Pi mount allows for several connections to its various ports and connectors.

The camera mount base is crafted from the same durable 3D plastic utilized for our platform base, ensuring consistency in stability and durability across our project components. We carefully modified Ibrahim's camera mount design, taking inspiration from its measurements, to produce a base that perfectly meets our unique needs. This meticulous attention to detail guarantees an accurate and precise fit for the camera mount, ensuring optimal functionality. Positioned strategically in the middle of the platform, the base provides a central location for mounting the camera. This placement enables the camera to capture a wide field of view, encompassing the entire board within its frame. By situating the camera mount centrally, we maximize its effectiveness in monitoring and capturing comprehensive visual data of the surrounding environment.

Moving away from the initial concept of the camera mount base, our design trajectory shifted as we opted out of utilizing the camera for real-time ball tracking to balance the platform. Instead, the hardware component base now serves as the central hub for housing not only the Raspberry Pi but also additional components integrated later in the project's development. These additions include the Analog to Digital Converter (ADC) and the protoboard, featuring strategically soldered pin extensions. Furthermore, to accommodate these components, the pin extension and ADC were meticulously affixed to the mount using Gorilla double-sided tape. This strategic placement ensures stability and accessibility while optimizing the overall functionality of the hardware setup. This revised approach reflects our adaptability and commitment to evolving project needs, ultimately enhancing the efficiency and effectiveness of our system. The initial design of the camera mount can be found below in Figure M. Also, the hardware component base with all the hardware components used for the project included on it is evident in Figure P below.

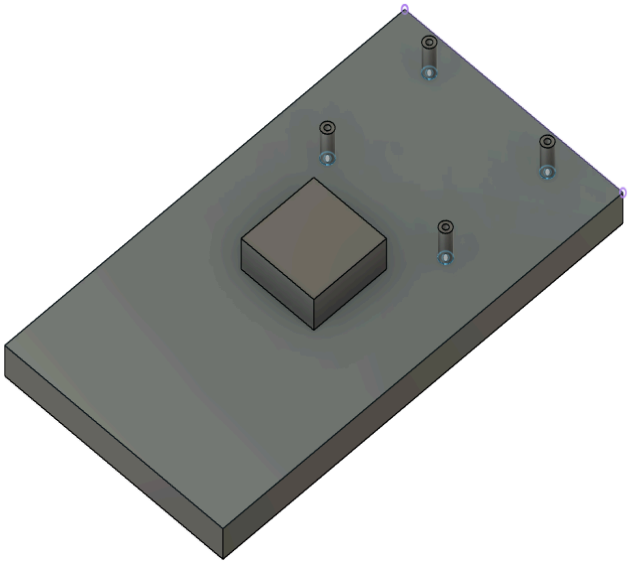


Figure M: The hardware component base without the components

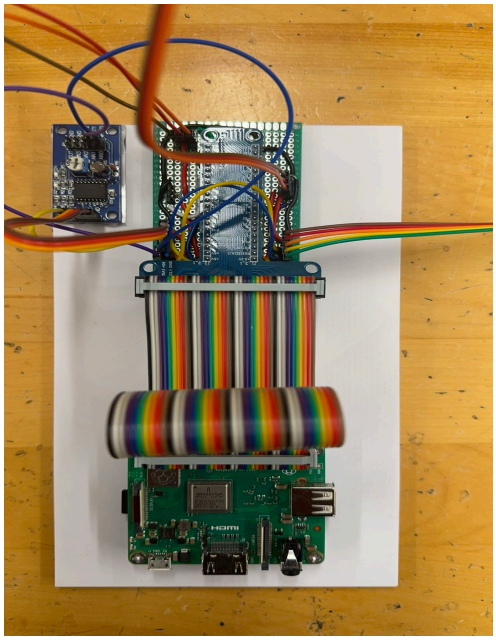


Figure P: The hardware component base with its components mounted on it

Camera Mount (Ibrahim)

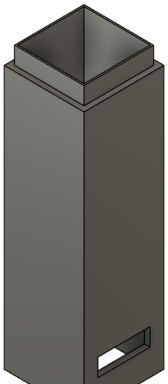
In the final phase of our project, incorporating a camera was essential for transforming our ball-balancing platform into an interactive game controlled by the user. The primary function of the camera is to track the ball's movements across the platform. We designated specific areas on the platform as 'green points'. The player's goal is to navigate the ball to these points using a controller, thereby accumulating points with each successful collection.

The concept for the camera mount drew inspiration from the versatility of LEGO blocks. This approach involved constructing the mount in separate segments, which were then assembled to form the complete structure. This modular design was particularly beneficial as it provided the flexibility to adapt the mount in response to any changes in other hardware components of the project.

The design and development of the camera mount were carried out using Fusion360, a sophisticated AutoCAD software. The mount consists of three primary components: the Base Block, the Connection Block, and the Mount itself. Further details and specifics of these components will be discussed in the section below:

Base Block:

This block serves as the key link connecting the platform's base to the mount. It's designed with a hollow interior running from top to bottom. This design choice was made to facilitate the passage of wires from the microcontroller to the camera, helping to minimize any

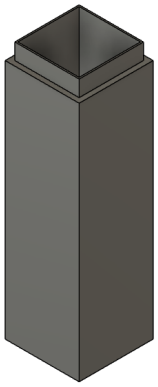


potential damage to these connections. In terms of size, the block measures 160mm in height and 46mm in width.

To ensure the mount remains firmly in place, the base of the components platform features a 40mm square-shaped protrusion. The block complements this design with a corresponding hollow section at its bottom, an inward extrusion measuring 38.06mm. This setup creates a secure, LEGO-like connection between the two surfaces. Additionally, the block has an outward extrusion of 10mm at its top. This feature is designed to accommodate the connection with the next piece in the assembly, the Connection Block.

Regarding wire management, there's a strategically placed incision on the side of the block. This incision measures 10mm in height and 30mm in width, dimensions that were specifically chosen to accommodate the size of the connecting wire.

Connection Block:



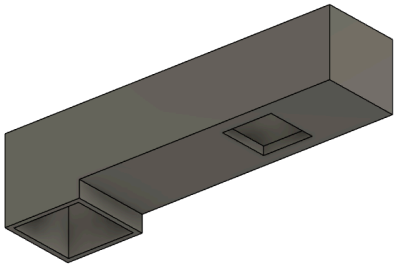
The Connection Block plays a pivotal role in our design, acting as the intermediary between the mount and the Base Block. Its primary function is to add a layer of adaptability to the camera mount, enabling size adjustments. Unlike the other two components, the Connection Block can be duplicated and used as an extension, providing the flexibility to either elongate or shorten the mount as needed.

In design, the Connection Block mirrors the Base Block in many ways, adhering to the same conceptual framework but with variations in its dimensions. This block stands at a height of 215mm and maintains the same width as the Base Block, 46mm. Like its counterpart, it is hollow on the inside, facilitating the passage for the camera’s and maintaining consistency in the design

The bottom of the Connection Block features an inward extrusion measuring 38.06mm, allowing it to seamlessly connect with the 40mm outward extrusion on the top of the Base Block. At the top, this block boasts an outward extrusion of 15mm in height and 40mm in width. This design aspect is critical as it enables a LEGO-like connection either to another Connection Block or directly to the mount itself.

The Mount:

This component is the final piece of the puzzle in our camera mount design and arguably the most critical. It plays a vital role in ensuring that all parts of the camera mount are correctly aligned. Its primary function is to position the camera precisely at the center of the platform where the ball is placed. This central placement is crucial as it allows the camera to capture a comprehensive view of the platform, essential for tracking the ball's movement and identifying the 'green points' for the game.



In terms of dimensions, the mount is 201mm tall and 50mm wide. Consistent with the design of the other components, the mount is also hollow. The bottom of the mount features an inward extrusion, 43mm in size, designed for attachment to the top part of the Connection Block.

A key feature of the mount is an incision at its end, shaped like a square, to accommodate the camera. This incision measures 30x30mm, providing ample space for the camera to pass through easily and be positioned at the very end of the mount. This strategic placement is integral to the functionality of the game, ensuring that the camera has an optimal view of the entire platform.

Specifications

The objective of this project is to develop a system that can actively balance a ball on a flat platform using servo motors for adjustment, guided by real-time motion tracking through a camera. The real-time position data, along with the desired position, gives the needed corrections to keep the ball in the desired positions. The group has decided on some needed specifications to ensure that the project is successful. These specifications were selected based on group discussion and research. We have decided the following specifications are most important:

- Ability to sustain 30 position measurements a second.
- Platform pitch and roll range must have a usable range of 20 degrees in the positive and negative direction.
- Fast platform angle response time.
- Position corrections and calculations need to keep up with the measurement speed.
- Ability to keep the ball balanced stably.

Our team has been allocated a budget of $120 by the university for this project. With a focus on cost-effectiveness, we have carefully selected components that ensure the smooth functioning of our project without exceeding the budget. The initial set of components we have chosen is detailed in the tables provided below.

Items Required		
Part Name	**Specs**	**Price (CAD)**
Servo Motor Driver IIC Module for Arduino Robot	HiLetgo PCA9685; 16-channel 12-Bit PWM (5-10 V supply)	15.99
SG90 Servo Motor	SG90 Servo Motor for RC Robot Helicopter Airplane Car	14.38

	Boat Remote control Blue (0.12sec/ op-voltage: 4.8V-6V)	
Raspberry Pi Camera (Pi 3/3 B+)	1080p, Arducam Adjustable and interchangeable Lens M12 Module, Focus and Angle Enhancement for Raspberry Pi 4/3/3 B+	27.99
Flex Cable for Raspberry Pi Camera	2 m extension	6.95
Raspberry Pi 3 Model A+	Lens Board, Arducam Adjustable and interchangeable Lens M12 Module, Focus and angle enhancement for Raspberry Pi 3	34.95
Raspberry Pi 3 Power supply (Micro USB)	2.5A micro USB	13.95
Platform	3D printed 20x20cm square	-
Ball	ping-pong ball painted green	-
Joystick	A joystick	-
Analog to Digital Converter (ADC)	PCF8591 A/D and D/A Converter Module -Proto Supplies	-
	SUM	**114.21**

In the items table provided above, we have detailed the critical components of the project. The ball-balancing platform system is made up of servo motors arranged in a triangle so that the movements of the platform may be fully controlled. The Raspberry Pi serves as the system's core computing unit, and it communicates with a camera to collect data and track motion in real time. However, we deviated from this idea because of the insufficient time it would have taken for the feedback loop using the camera to be running on an optimum level. In coming to this decision as a team to deviate from the camera feedback loop, we then took the choice of using a joystick as the feedback loop, which not only provides a formidable feedback loop but also allows for a user interface which then elevates our device.

Using the joystick (specifically the ADIY 2-axis), we were able to provide analog input with precise position detection from its potentiometers for each x and y axes. It has a comfortable grip and cup-type knob which automatically returns to its center position as its resting mode with its easy accessibility and adaptability making it user-friendly and tempting to give it a go. Furthermore, the joystick comes with pins; **GND (Ground)**, **+5V (Power from the raspberry Pi)**, **VRx (AIN 1 from the Joystick)**, **VRy (AIN 0 from the Joystick)**, and **SW (Pin 24 from the Raspberry Pi)**, which will serve as connections to the analog to digital converter (ADC) and pin extension which ultimately is connected to the Raspberry Pi. Finally, a mount for the Joystick was created to appease the aesthetic feature while also keeping in mind the comfortability, adaptability, and accessibility of the device. In Figure x below, an illustration of the joystick is provided.

Figure x: The ADIY 2-axis joystick [1]

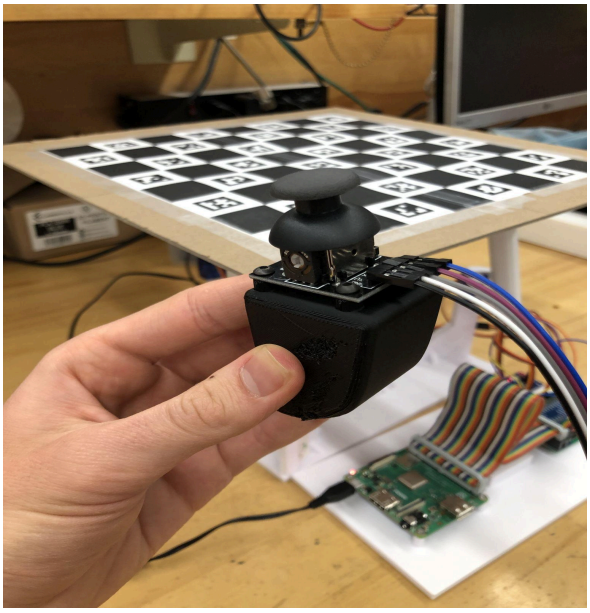


Figure x (ii): The ADIY 2-axis joystick with its mount

The PCF8591ADC (Analog to Digital Converter) plays a crucial role in enhancing user interaction. This ADC utilizes six signals: **GND (Ground), Vcc (+5V from the Raspberry Pi), SCL (Clock data from pin extension, P03), SDA (Data signal from pin extension, P02), AIN 1 (Analog input 1 for the x-axis from the joystick), and AIN 0 (Analog input 0 for the y-axis from the joystick)**. It takes analog input from the joystick, converting it into a digital signal. This digital signal is then used to control the joystick, allowing users to fully immerse themselves in engaging and captivating gameplay experiences. There is a visual representation of the ADC provided in Figure Y below.

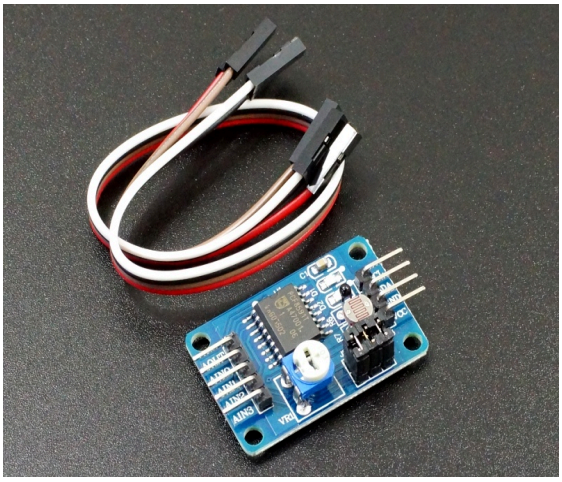


Figure Y: The PCF8591 Analog to Digital Converter (ADC) [3]

Using sophisticated algorithms, the Raspberry Pi interprets incoming data to dynamically change the platform's location and converts it into commands that the servo motors can follow. The system's mix of cutting-edge hardware and software components assures accuracy in preserving the ball's position. The servo motors are crucial for actuating the platform and will be configured in a triangular pattern to ensure comprehensive control of its movements. These motors' function is directly influenced by the instructions received from our central processing unit, the Raspberry Pi.

Finally, the overview of the specifications for our project incorporates user-controlled input through a joystick for real-time data acquisition, essential for our control strategy. The Raspberry Pi serves as the backbone, processing data from the analog joystick and dynamically adjusting the platform's position accordingly. This interaction is facilitated by converting analog joystick data into a digital signal. Additionally, a camera captures the ball's position on the board, contributing to the system's feedback loop for precise control. The integration of these components underscores our commitment to seamless hardware and software interaction for optimal movement and control.

Sustainability Assessment

This project's 'product', the ball balancing platform, is not intended for large scale manufacturing. It is not a useful device on its own, its purpose was to showcase the use of control systems and show how control systems can be used. It could in theory be made large scale as a game, by adding a housing for the electrical components. This section dives into the sustainable design aspects of this project and what its intended life cycle would be were it to be manufactured on a large scale.

The Ball Balancing Platform will reach its end of life when it is no longer wanted or used. This could happen if say it is used as a demo of control systems and better control system demos are made that replace this project. If it is used as a game it should not become obsolete, as it stays entertaining, and if someone no longer wants it it could be passed on to someone else. If it gets damaged then any piece is easily replaceable, as the servo motors and joystick are common components that are easy to replace, and all the 3D printed pieces are attached with screws and a replacement piece could easily be printed and swapped out for the broken piece.

All parts of this project were attached using screws instead of adhesive so that parts are easily replaceable as well as for disassembling the product at end of life. As well, the wires are attached using wire mounts so no new soldering would be required if an electrical component needs to be replaced as wires can easily be unplugged and plugged in. At end of life this product can be disassembled and the parts can be reused. The servos and joystick can easily be reused and the 3D printed parts could be melted down to make new filament. Only a few wires and wire mounts are soldered onto a proto board, if wanted for reuse these would need to be unsoldered. The ping pong ball can be reused for ping pong.

The product uses minimal electricity as all of its processing is efficiently done on the raspberry pi. The only environmental impacts would be the loss of solder used, energy used for 3D printing and melting down the 3D filament, and the small amount of electricity used to run the servos and pi.

Conclusion & further development

Conclusion (Tristan)

Conclusion (Ciaran)

This project was a huge success and we as a group achieved much more than we thought we would be able to when starting out the term. We made a plan of what needed to be done and we got through it, checking things off of our todo list. We of course came into a lot of hurdles, but we overcame them, fixing problem after problem until it all worked. This project has quite a bit of room for improvement. The camera can be connected to the pi and the full feedback control loop can be implemented. This then extends to adding a UI to control where the desired ball position is, not only requiring it to stay in the middle. A casing could also be made for the electrical components to hide with the base so it is all one connected thing, easier to transport around. A button could also be added to switch between different modes, either joystick control, camera control, or just smooth demo movement (Or this could be a toggle on the UI). This could become a really fun and polished demo to showcase the electrical and engineering physics program at Carleton.

Conclusion (Noah)

The ball balancing platform incorporated the skills and knowledge throughout my engineering degree. The ball balancing platform used knowledge from the class visual communications to teach me how to use Fusion and to further develop components. These components were 3D printed and tested to observe the type of stress, strain, and forces that were taught in the courses statics and mechanics. I learned to keep good communication, work as a team, and meet important deadlines. Moving forward, I can continue to expose myself to engineering societies and speak with mature engineers to further enhance my skills and knowledge.

Conclusion (Ayooluwa)

The ball balancing project is evidence of our team's strong teamwork, perseverance, and everlasting dedication. We faced and overcame several obstacles at every turn, from the beginning to the end, constantly going above and beyond with our creative solutions. Digital models and circuit schematics were made in order to provide a concrete representation of the ideas taught in class, thus bridging the gap between theory and practice. This initiative not only perfectly captures the spirit of engineering education, but it also emphasises how crucial it is to combine theoretical ideas with practical application. This journey has been an incredible educational experience that has reinforced the significance of time management, cooperation, and interacting with mentors—including our esteemed professors, teaching assistants, and engineering societies. Looking ahead, I am eager to further refine my skills and broaden my horizons by deepening my engagement with diverse engineering societies and mentors from various backgrounds. Moreover, I recognize the significance of meticulous documentation, understanding its pivotal role in project management and success.

Conclusion (Ibrahim)

This project brought together basic engineering ideas from the various stages of my studies and put them into practice. A digital model of a camera mount was made using AutoCAD, turning classroom concepts into something real. A circuit schematic was also developed to link the microcontroller with its parts, and this was tested on a breadboard to make sure it all worked well. Picking up new skills like soldering was key, especially for moving the circuit from the breadboard to a protoboard, making the signals stronger and more reliable. Overall, this project was a great mix of using what was already known and learning new things, blending book learning with real-world applications.

References

[1] “Dual Axis Analog Joystick Module With Push Button Function,” UNIVERSAL SOLDER. https://universal-solder.ca/product/dual-axis-analog-joystick-module/ (accessed Apr. 09, 2024).

[2]“ERM Automatismes Industriels,” www.erm-automatismes.com. https://www.erm-automatismes.com/p469-eng-ball-balancing-table.html?g=1 (accessed Apr. 09, 2024).

[3] “PCF8591 A/D and D/A Converter Module,” ProtoSupplies. https://protosupplies.com/product/pcf8591-a-d-and-d-a-converter-module/ (accessed Apr. 10, 2024).

[4] “What are the Basics of Ball and Beam | Acrome Robotics,” acrome.net. https://acrome.net/post/what-are-the-basics-of-ball-and-beam

[5] CmdrZIn. “Ball Joint With Fusion 360”. AUTODESK Instructables. https://www.instructables.com/Ball-Joint-With-Fusion-360/ (retrieved: Jan 22, 2024)

Tristan’s Citations

[1] "OpenCV: Detection of ChArUco Boards." Available: https://docs.opencv.org/3.4/df/d4a/tutorial_charuco_detection.html. [Accessed: Mar. 06, 2024]

[2] "What Is Camera Calibration? - MATLAB & Simulink." Available: https://www.mathworks.com/help/vision/ug/camera-calibration.html. [Accessed: Mar. 06, 2024]

[3] "OpenCV: Aruco markers, module functionality was moved to objdetect module." Available: https://docs.opencv.org/5.x/d9/d6a/group__aruco.html#ga366993d29fdddd995fba8c2e6ca811ea. [Accessed: Mar. 07, 2024]

[4] R. Shaikh, "OpenCv Perspective Transformation," Analytics Vidhya, Feb. 06, 2024. Available: https://medium.com/analytics-vidhya/opencv-perspective-transformation-9edffefb2143. [Accessed: Mar. 07, 2024]

[1] “Gaussian Blur.” Available: https://www.w3.org/Talks/2012/0125-HTML-Tehran/Gaussian.xhtml. [Accessed: Apr. 10, 2024]

Ciaran’s Citation

[C1] C. McDonald-Jensen, “Individual Design Report: Parameter Selection for the Ball Balancing Platform.” Carleton University, Mar. 08, 2024. Available: https://brightspace.carleton.ca/d2l/lms/dropbox/user/folders_history.d2l?db=212476&grpid=0&isprv=0&bp=0&ou=221577. [Accessed: April 10, 2024]

Appendix A: Division of Work

Below is the division of work decided on by all the group members. Signatures for the agreed upon division can also be found below.

Percentage Breakdown	
Tristan Laliberté	24%
Ciaran McDonald-Jensen	19%
Noah Connell	19%
Ayooluwa Owolabi	19%
Ibrahim Shah	19%
TOTAL	100%

Tristan Laliberté 

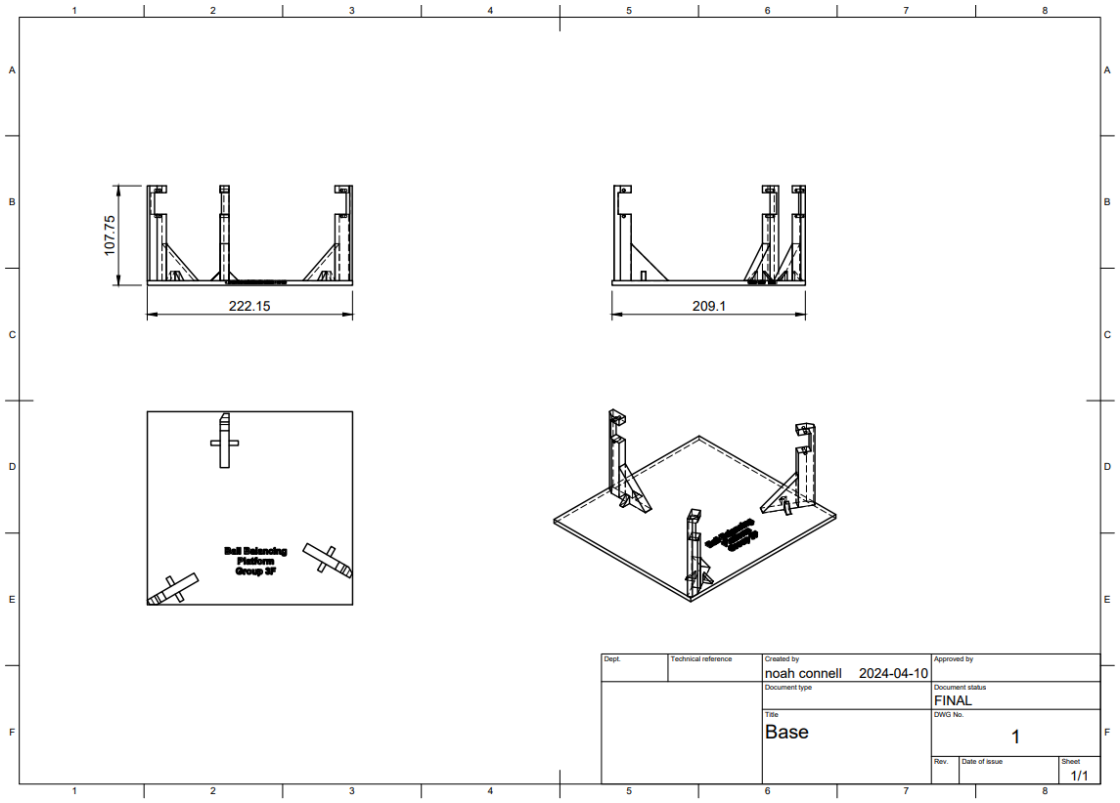
Ciaran McDonald-Jensen 

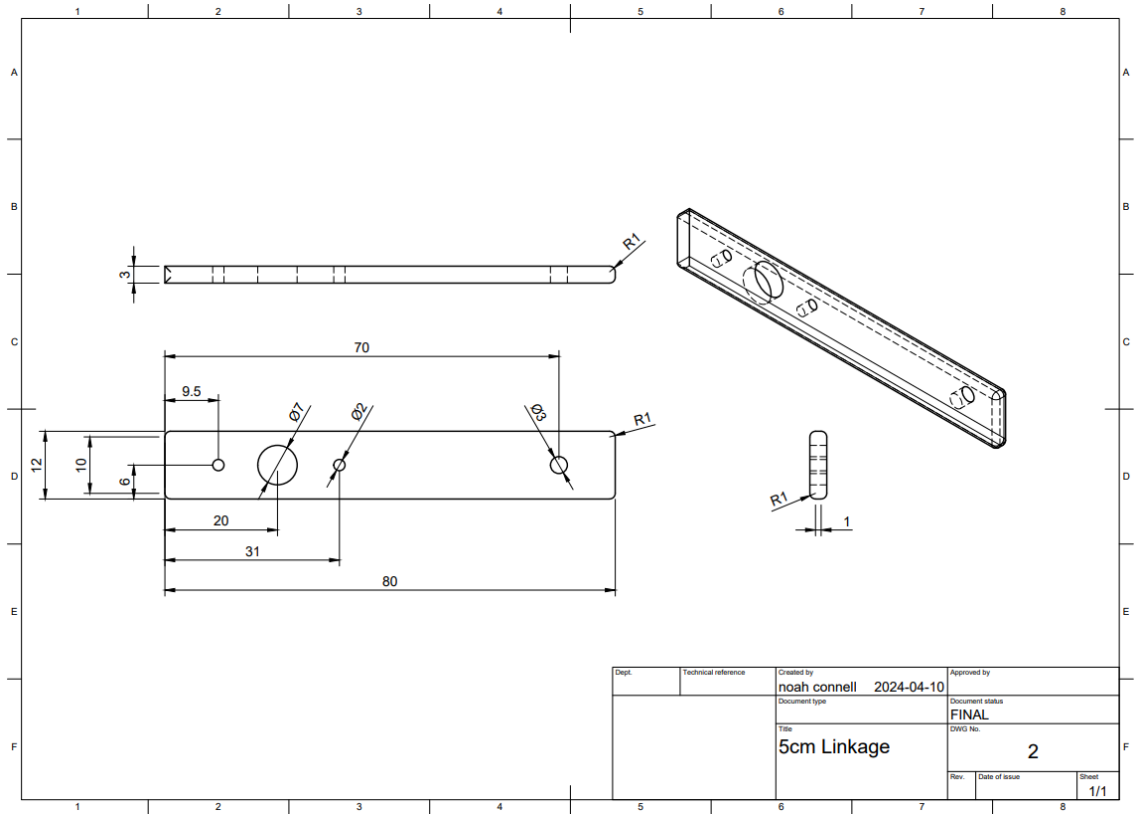
Noah Connell 

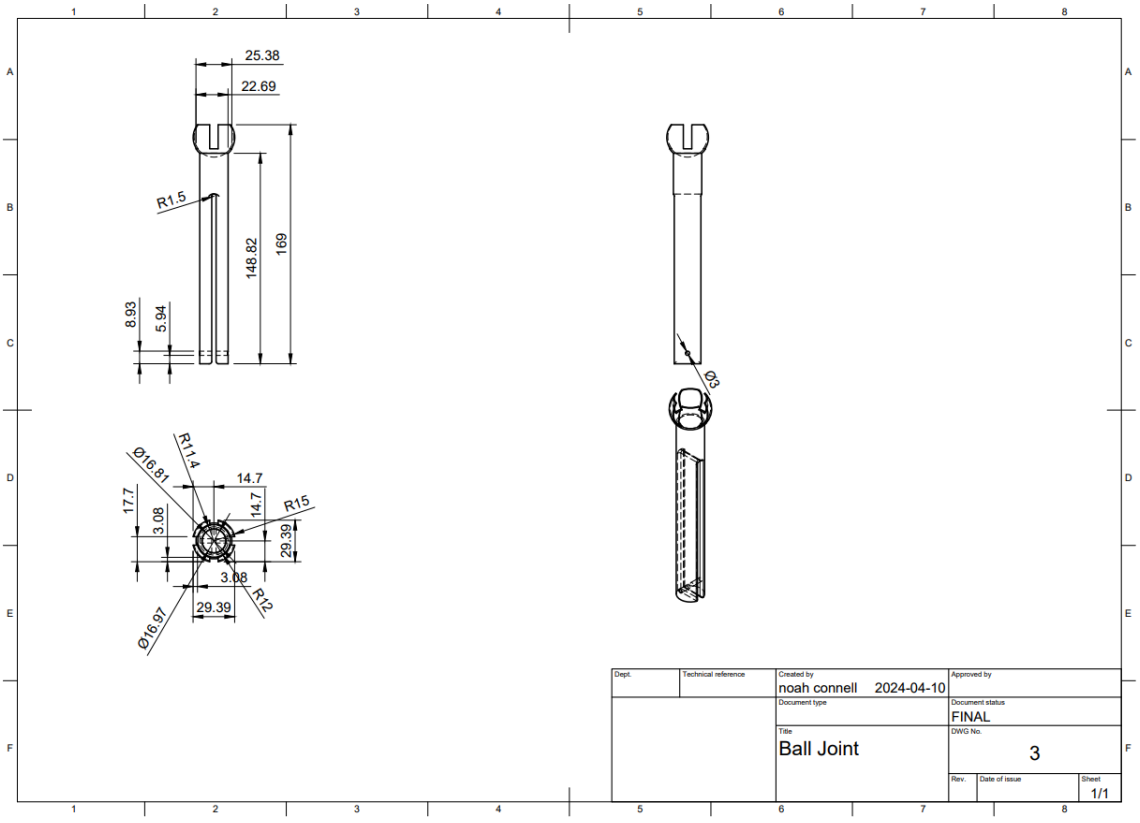
Ayooluwa Owolabi 

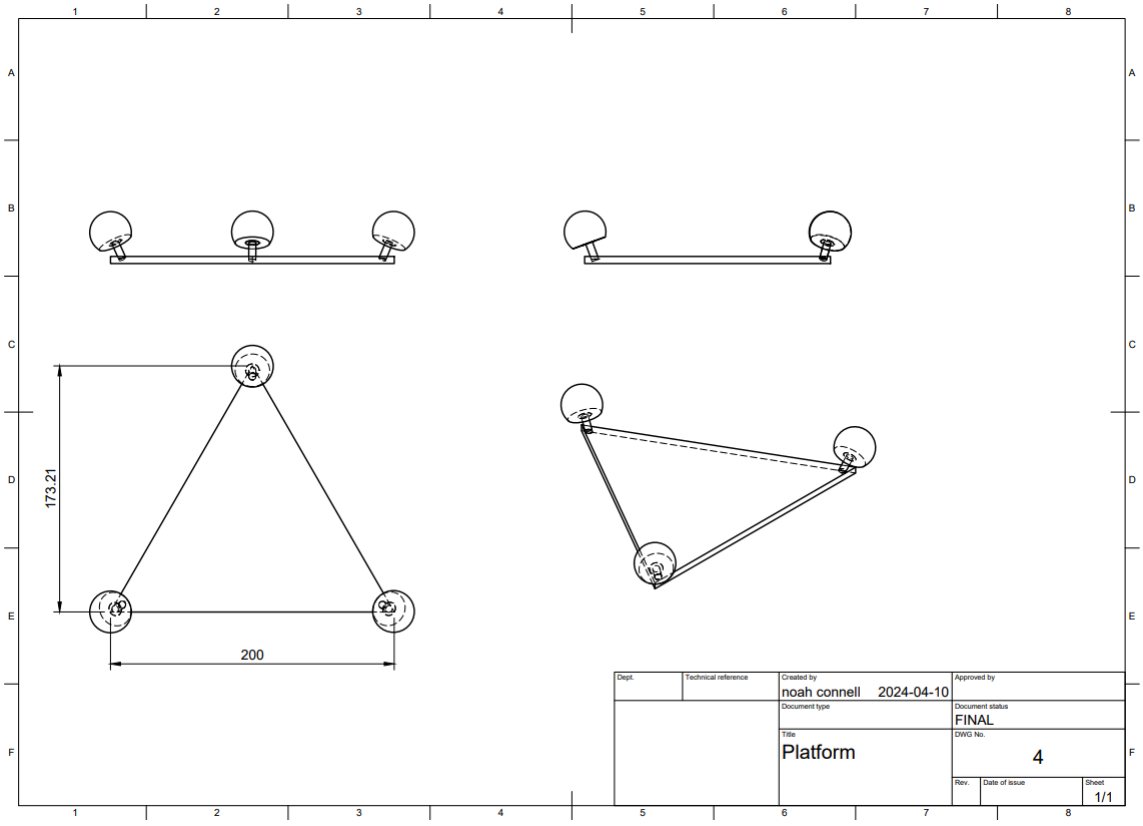
Ibrahim Shah 

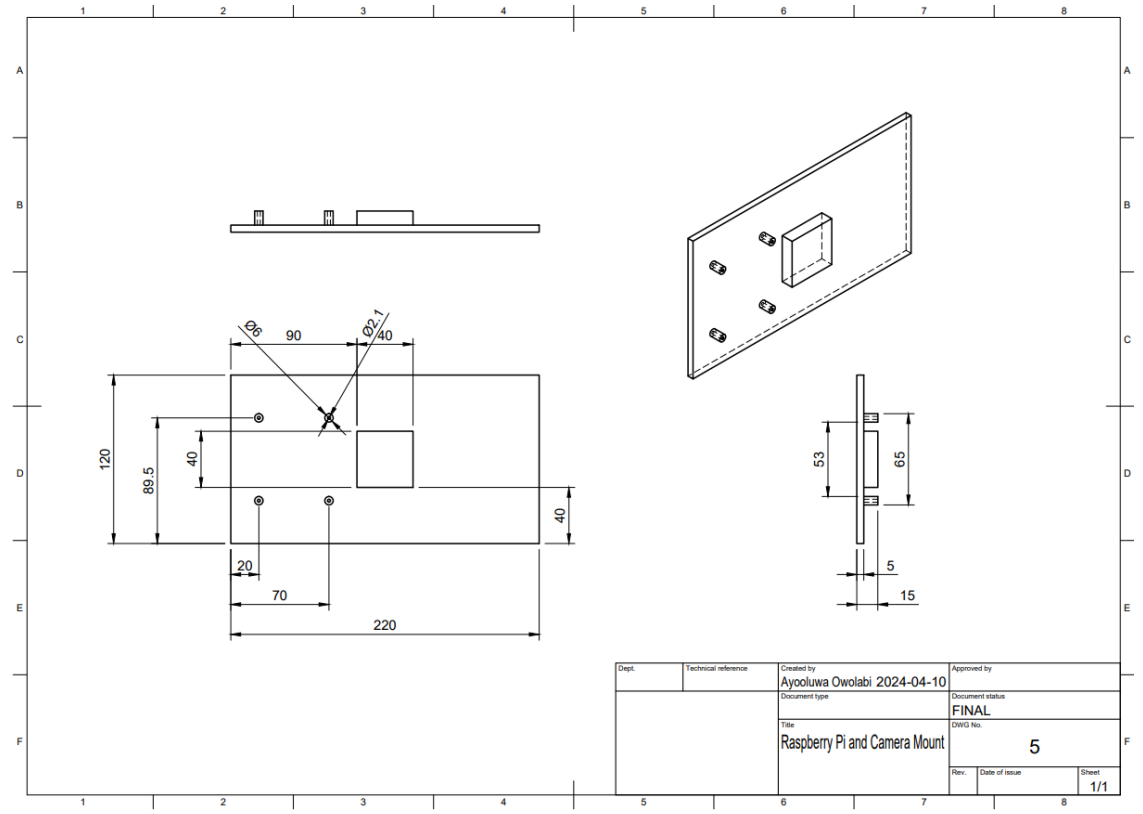
Appendix C: Part Schematics

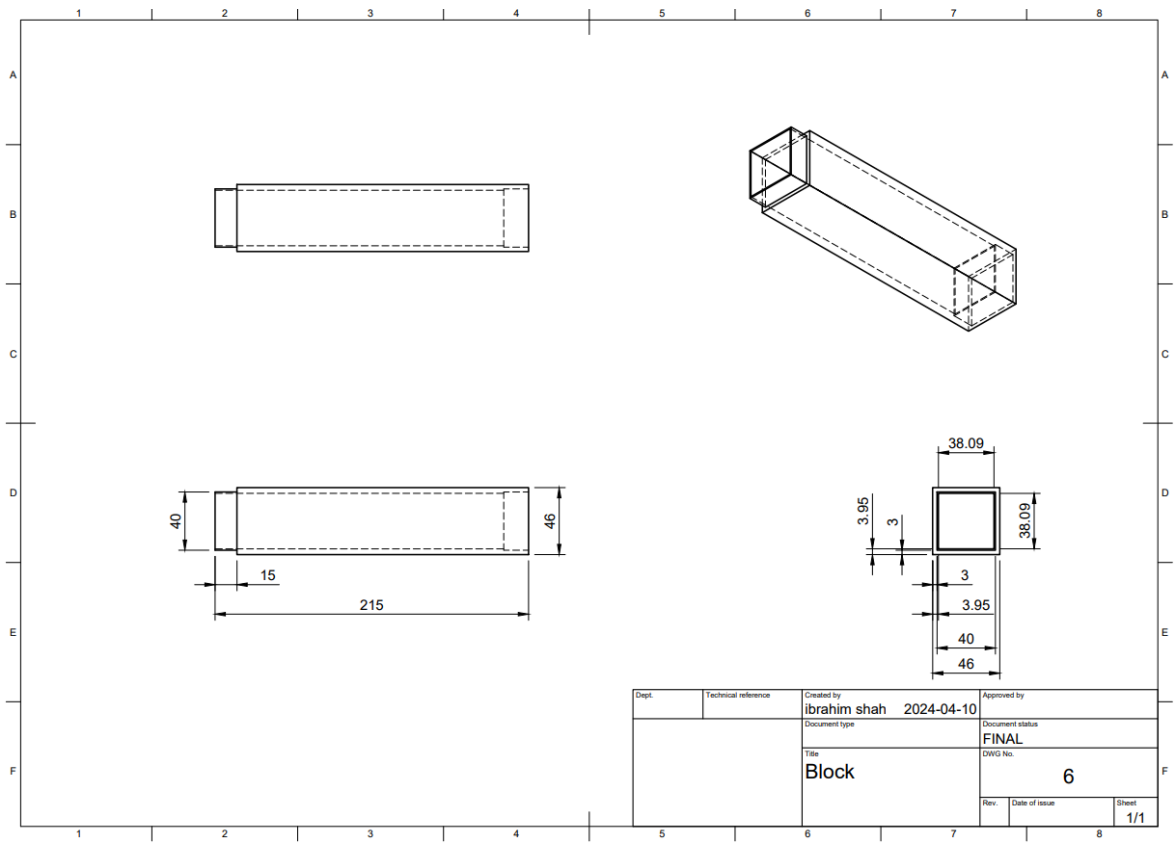


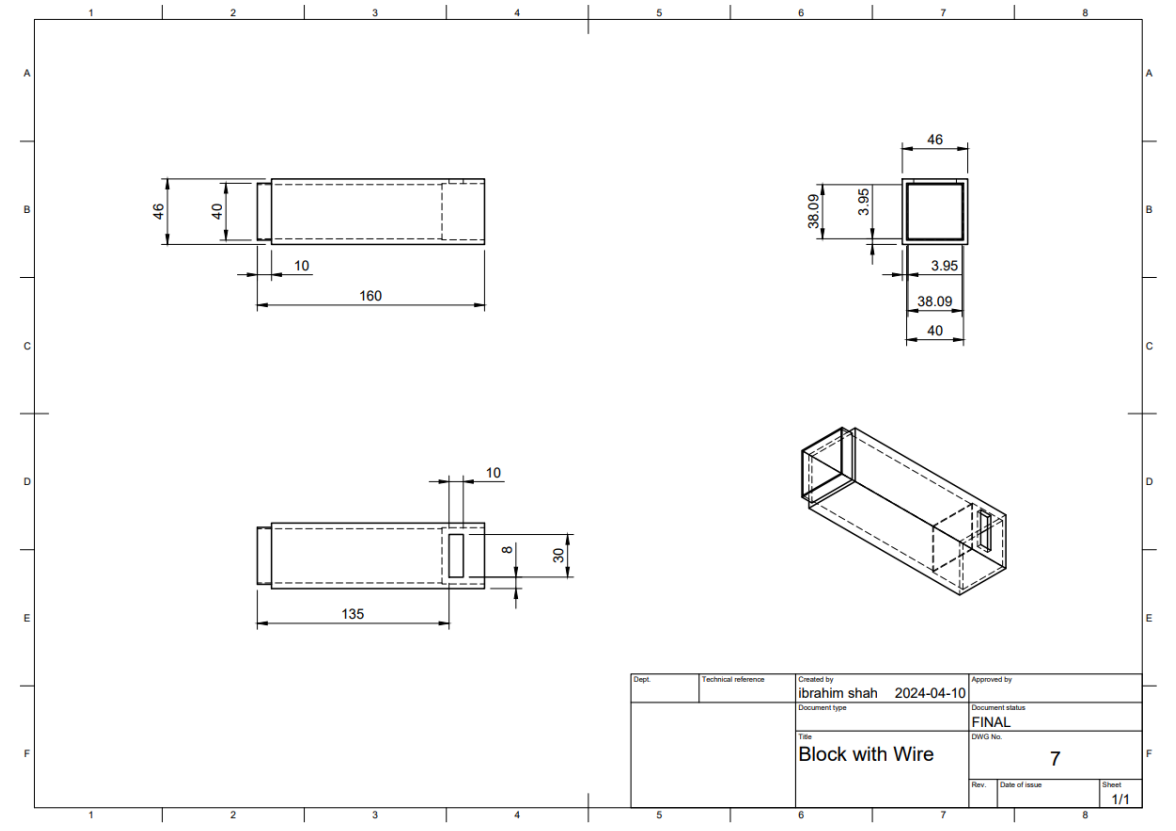

3
R1
70
9.5
Ø7
Ø2
Ø3
R1
12
10
6
20
31
80
R1
1
Dept.
Technical reference
Created by
noah connell 2024-04-10
Approved by
Document type
Document status
FINAL
Title
5cm Linkage
DWG No.
2
Rev.
Date of issue
Sheet
1/1

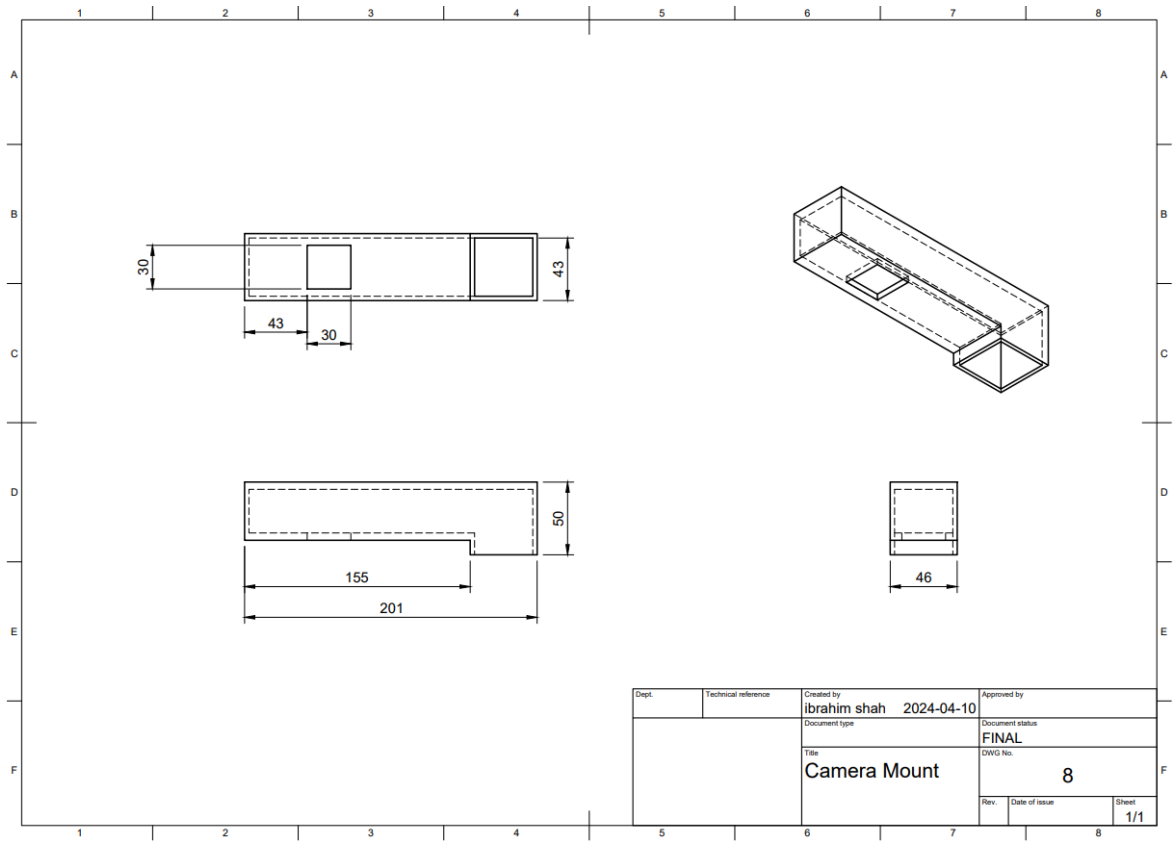

25.38
22.69
R1.5
148.82
169
8.93
5.94
Ø3
Ø16.81
R11.4
14.7
14.7
R15
17.7
3.08
29.39
3.08
R12
29.39
Ø16.97
Dept.
Technical reference
Created by
noah connell 2024-04-10
Approved by
Document type
Document status
FINAL
Title
Ball Joint
DWG No.
3
Rev.
Date of issue
Sheet
1/1

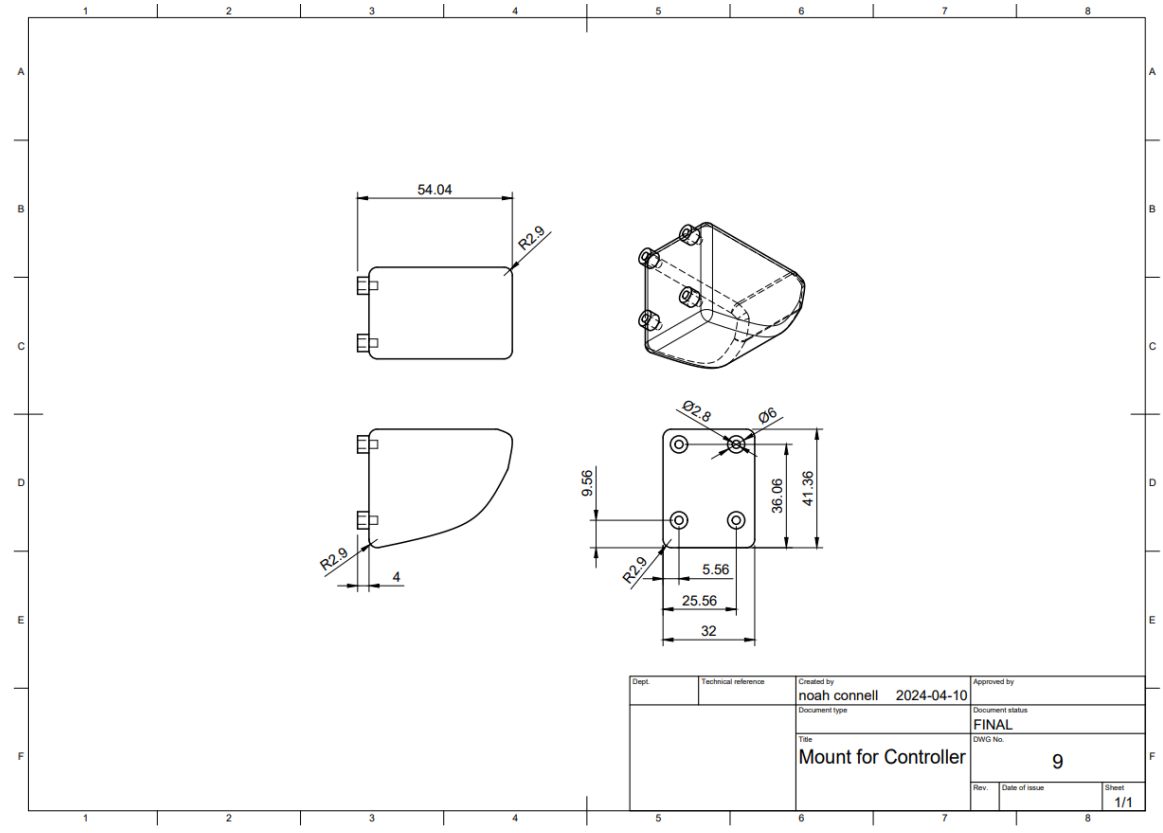

173.21
200
Dept.
Technical reference
Created by
noah connell 2024-04-10
Approved by
Document type
Document status
FINAL
Title
Platform
DWG No.
4
Rev.
Date of issue
Sheet
1/1

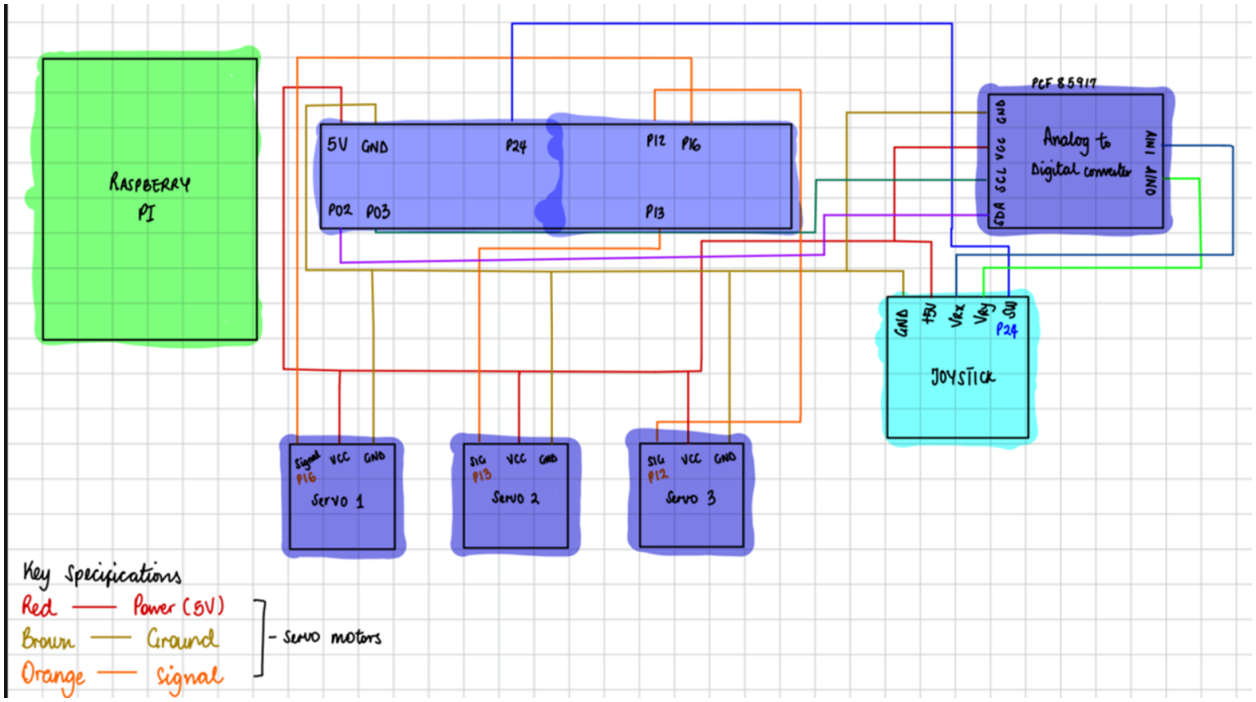

Ø6
90
Ø2.1
40
120
89.5
40
40
20
70
220
53
65
5
15
Dept.
Technical reference
Created by
Ayooluwa Owolabi 2024-04-10
Approved by
Document type
Document status
FINAL
Title
Raspberry Pi and Camera Mount
DWG No.
5
Rev.
Date of issue
Sheet
1/1


40
46
15
215
38.09
3.95
3
38.09
3
3.95
40
46
Dept.
Technical reference
Created by
ibrahim shah 2024-04-10
Approved by
Document type
Document status
FINAL
Title
Block
DWG No.
6
Rev.
Date of issue
Sheet
1/1


46
40
10
160
46
38.09
3.95
3.95
38.09
40
10
8
30
135
Dept.
Technical reference
Created by
ibrahim shah 2024-04-10
Approved by
Document type
Document status
FINAL
Title
Block with Wire
DWG No.
7
Rev.
Date of issue
Sheet
1/1


30
43
43
30
50
155
201
46
Dept.
Technical reference
Created by
ibrahim shah 2024-04-10
Approved by
Document type
Document status
FINAL
Title
Camera Mount
DWG No.
8
Rev.
Date of issue
Sheet
1/1


54.04
R2.9
R2.9
4
Ø2.8
Ø6
9.56
36.06
41.36
R2.9
5.56
25.56
32
Dept.
Technical reference
Created by
noah connell 2024-04-10
Approved by
Document type
Document status
FINAL
Title
Mount for Controller
DWG No.
9
Rev.
Date of issue
Sheet
1/1


Raspberry Pi
5V GND P24 P12 P16
P02 P03 P13
PCF 8591P
Analog to Digital converter
GND VCC SCL SDA AIN0 AIN1
GND +5V Vrx Vry SW P24
Joystick
Signal P16 VCC GND
Servo 1
SIG P13 VCC GND
Servo 2
SIG P12 VCC GND
Servo 3
Key Specifications
Red — Power (5V)
Brown — Ground
Orange — Signal
- Servo motors

Appendix D: Source Code

A history of the code's development can be found on our Github page:

https://github.com/Ciaran-McDJ/ELEC3907A3-F-Ball-Balancing-Platform

Appendix E: OpenCV Camera Tracking Algorithm

```
import cv2 as cv
import numpy as np
import os
from typing import Optional
import random

"""
Module for ball tracking on a Charuco board.

This module provides functions for generating and manipulating Charuco boards, saving camera
calibration parameters, drawing corners on images, transforming perspective, sharpening
images, and getting the position of a ball in an image. The ball is defined by setting an
upper and lower HSV colour limit, which is used to mask out the background.

"""



class Config:
    """
    The `Config` class represents the configuration parameters for the Charuco imaging
application.

    Attributes:
    ARUCO_DICT (int): The ARUCO dictionary type.
    SIZE (tuple): The number of squares in the Charuco board.
    SQUARE_LENGTH (float): The length of each square in meters.
    MARKER_LENGTH (float): The length of each marker in meters.
    IMAGE_SIZE (tuple): The size of the generated Charuco board image.
    CALIBRATION_LOCATION (str): The location of the calibration images folder.
    CAM_MATRIX_PATH (str): The file path to save the camera matrix.
    DISTORTION_MATRIX_PATH (str): The file path to save the distortion matrix.
    TRANSFORMED_SIZE (tuple): The size of the transformed image.
    """
    ARUCO_DICT = cv.aruco.DICT_5X5_250  # ArUco dictionary to use
    SIZE = (8, 8)  # Size of the Charuco board
    SQUARE_LENGTH = 0.03  # Length of the squares in the Charuco board
    MARKER_LENGTH = 0.018  # Length of the markers in the Charuco board
    IMAGE_SIZE = (800, 800)  # Size of the image to generate
    BOARD_POINTS = (0,0,0), (SIZE[0]*SQUARE_LENGTH, 0, 0), (0, SIZE[1]*SQUARE_LENGTH, 0),
(SIZE[0]*SQUARE_LENGTH, SIZE[1]*SQUARE_LENGTH, 0)
    CALIBRATION_LOCATION =
'C:/Users/trist/OneDrive/Desktop/Projects/ELEC3907A3-F-Ball-Balancing-Platform/Ball
balancer/CamCalibrationOld'  # Location of the calibration images
    CAM_MATRIX_PATH =
'C:/Users/trist\OneDrive/Desktop/Projects/ELEC3907A3-F-Ball-Balancing-Platform/Ball
balancer/3camMatrix.npy'  # Path to save the camera matrix
    DISTORTION_MATRIX_PATH =
'C:/Users/trist\OneDrive/Desktop/Projects/ELEC3907A3-F-Ball-Balancing-Platform/Ball
balancer/3distMatrix.npy' # Path to save the distortion matrix
```

```
    TRANSFORMED_SIZE = (480, 480)  # Size of the transformed image
    BUFFER = 50 # pixels
    GOAL_SIZE = 40 # pixels
    MASK_LOWER = np.array([71,131,78]) # HSV lower colour code
    MASK_UPPER = np.array([90, 245, 171]) # HSV Upper colour code
    SCALE_FACTOR = 0.5 # Image scaling factor
    DISPLAY_MODE = False
    SHOW_CENTER = False
    MODES = [ord('1'),ord('2'),ord('3'),ord('4'),ord('5')]
    DIFFICULTIES = {49:100, 50:80, 51:60, 52:40, 53:20}


def load_charuco_board() -> cv.aruco.CharucoBoard:
    """
    Load the Charuco board.

    Returns:
        cv.aruco.CharucoBoard: The loaded Charuco board.
    """
    dictionary = cv.aruco.getPredefinedDictionary(Config.ARUCO_DICT)
    return cv.aruco.CharucoBoard(Config.SIZE, Config.SQUARE_LENGTH, Config.MARKER_LENGTH,
dictionary)


def generateCharucoBoard() -> None:
    """
    Generate a Charuco board image.
    """
    board = load_charuco_board()
    image = board.generateImage(Config.IMAGE_SIZE)

    cv.imshow("charuco", image)
    np.save('charucoBoard.npy', board)

    # Wait for the user to press 'q' to quit
    while True:
        if cv.waitKey(1) & 0xFF == ord('q'):
            Break
    cv.destroyAllWindows()


def saveCalibrationCameraParameters() -> None:
    """
    Save the camera calibration parameters.
    """
    dictionary = cv.aruco.getPredefinedDictionary(Config.ARUCO_DICT)
    board = load_charuco_board()
    params = cv.aruco.DetectorParameters()

    # Get a list of all the .jpg images in the calibration location
    images = [os.path.join(Config.CALIBRATION_LOCATION, filename) for filename in
```

```
os.listdir(Config.CALIBRATION_LOCATION) if filename.endswith('.jpg')]

    all_charuco_corners = []
    all_charuco_ids = []

    # For each image, detect the markers and interpolate the Charuco corners
    for image_file in images:
        image = cv.imread(image_file)
        marker_corners, marker_ids, _ = cv.aruco.detectMarkers(image, dictionary,
parameters=params)
        if len(marker_corners) > 0:
            charuco_retval, charuco_corners, charuco_ids =
cv.aruco.interpolateCornersCharuco(marker_corners, marker_ids, image, board)
            if charuco_retval:
                all_charuco_corners.append(charuco_corners)
                all_charuco_ids.append(charuco_ids)

    # Calibrate the camera using the Charuco corners and ids
    retval, camera_matrix, dist_coeffs, rvecs, tvecs =
cv.aruco.calibrateCameraCharuco(all_charuco_corners, all_charuco_ids, board,
image.shape[:2], None, None)

    # Save the camera matrix and distortion coefficients
    np.save(Config.CAM_MATRIX_PATH, camera_matrix)
    np.save(Config.DISTORTION_MATRIX_PATH, dist_coeffs)
    print("Calibration complete and saved")


def drawCorners(image: np.ndarray, dictionary: cv.aruco.Dictionary, board:
cv.aruco.CharucoBoard, camMatrix: np.ndarray, distMatrix: np.ndarray) -> tuple[bool,
np.ndarray]:
    """
    Draw corners on an image.

    Args:
        image (np.ndarray): The input image.
        dictionary (cv.aruco.Dictionary): The ArUco dictionary.
        board (cv.aruco.CharucoBoard): The Charuco board.
        camMatrix (np.ndarray): The camera matrix.
        distMatrix (np.ndarray): The distortion matrix.

    Returns:
        Tuple[bool, np.ndarray]: A tuple containing a boolean indicating if the corners were
successfully drawn and the resulting image.

    """
    charucoPoints3D = Config.BOARD_POINTS
    corners, ids, _ = cv.aruco.detectMarkers(image, dictionary)


if len(corners) == 0:
```

```
        return False, image

    charucoCorners = cv.aruco.interpolateCornersCharuco(corners, ids, image, board)
    if charucoCorners is None or len(charucoCorners) == 0:
        return False, image

    charucoCornerLocations = charucoCorners[1]
    charucoIds = charucoCorners[2]

    if charucoCornerLocations is None or len(charucoCornerLocations) == 0 or len(charucoIds)
< 6:
        return False, image

    # Estimate the pose of the Charuco board
    ret, rvec, tvec = cv.aruco.estimatePoseCharucoBoard(charucoCornerLocations, charucoIds,
board, camMatrix, distMatrix, None, None, useExtrinsicGuess=False)
    cornerArray = []
    if ret:
        for point_3d in charucoPoints3D:
            point_2d, _ = cv.projectPoints(point_3d, rvec, tvec, camMatrix, distMatrix)
            cornerArray.append(tuple(point_2d[0][0]))

        cornerArray = np.array(cornerArray)
        transformedImage = transformPerspective(image, cornerArray)
        return True, transformedImage

    return False, image


def transformPerspective(image: np.ndarray, cornerList: np.ndarray) -> np.ndarray:
    """
    Transform the perspective of an image.

    Args:
        image (np.ndarray): The input image.
        cornerList (np.ndarray): The list of corners.

    Returns:
        np.ndarray: The transformed image.
    """
    x_length, y_length = Config.TRANSFORMED_SIZE
    destination = np.array([(0,0), (x_length,0), (0, y_length), (x_length, y_length)])
    transform, mask = cv.findHomography(cornerList, destination, cv.RANSAC, 5.0)
    return cv.warpPerspective(image, transform, (x_length, y_length))


def draw_center(image: np.ndarray, x: float, y: float) -> None:
    """
    Draw the center of the circle on an image.

    Args:
```

```
        image (np.ndarray): The input image.
        x (float): The x position of the circle.
        y (float): The y position of the circle.
    """
    if x is not None and y is not None:
        cv.circle(image, (int(x), int(y)), 5, (0, 0, 255), -1)


def generate_random_point()->tuple:
    """
    Creates a random set of x and y coordinates inside the transformed size with a  pixel
buffer.
    """
    Config.BUFFER
    x = random.randint(Config.BUFFER, Config.TRANSFORMED_SIZE[0]-Config.BUFFER)
    y = random.randint(Config.BUFFER, Config.TRANSFORMED_SIZE[0]-Config.BUFFER)
    return (x,y)


def draw_square(image, center, side_length=Config.GOAL_SIZE, color=(0, 255, 0),
thickness=2):
    """
    Draw a square on the given image.

    Parameters:
        image: numpy.ndarray
            The image on which the square will be drawn.
        center: tuple (int, int)
            The coordinates (x, y) of the center of the square.
        side_length: int, optional
            The length of each side of the square (default is 100).
        color: tuple (int, int, int), optional
            The color of the square in BGR format (default is green).
        thickness: int, optional
            The thickness of the square's edges (default is 2).
    """
    half_length = side_length // 2
    top_left = (center[0] - half_length, center[1] - half_length)
    bottom_right = (center[0] + half_length, center[1] + half_length)
    cv.rectangle(image, top_left, bottom_right, color, thickness)


def img_process(frame)-> None:
    """
    Takes in an image and rescales the image to a smaller scale and returns the image in hsv
    """
    height, width, _ = frame.shape
    frame = cv.resize(frame, (int(width*Config.SCALE_FACTOR),
int(height*Config.SCALE_FACTOR)))
    blurred = cv.GaussianBlur(frame, (11, 11), 0)
    hsv = cv.cvtColor(blurred, cv.COLOR_BGR2HSV)
```

```
    return hsv


def generate_mask(hsv):
    """
    Applies an HSV filter with upper and lower range defined in config class
    """
    mask = cv.inRange(hsv, Config.MASK_LOWER, Config.MASK_UPPER)
    mask = cv.erode(mask, None, iterations=2)
    mask = cv.dilate(mask, None, iterations=2)
    return mask


def find_centroid(mask):
    # Find contours in the mask image
    contours, _ = cv.findContours(mask, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)

    # Initialize variables to store the maximum contour area and its corresponding contour
    max_contour_area = 0
    max_contour = None

    # Iterate through all contours
    for contour in contours:
        # Calculate the area of the contour
        area = cv.contourArea(contour)
        # If the area is larger than the maximum area found so far, update the maximum area
and contour
        if area > max_contour_area:
            max_contour_area = area
            max_contour = contour

    # If a contour is found, calculate its centroid and draw a point on the original frame
    if max_contour is not None:
        # Calculate centroid (center) of the contour
        M = cv.moments(max_contour)
        if M["m00"] != 0:
            center_x = int(M["m10"] / M["m00"])
            center_y = int(M["m01"] / M["m00"])

            return center_x, center_y
    return -1,-1


def get_ball_position(frame):
    """
    Takes in an image, scales the image down and applies a contour search using an HSV mask.
The mask limits are defied in config
    """
    hsv = img_process(frame)
    mask = generate_mask(hsv)
    unscaled_x, unscaled_y = find_centroid(mask)
```

```
    if Config.DISPLAY_MODE:
        cv.imshow("HSV view", hsv)
        cv.imshow("Mask", mask)
        cv.moveWindow("HSV view", 0, 0)
        cv.moveWindow("Mask", 0, 280)

    return unscaled_x/Config.SCALE_FACTOR, unscaled_y/Config.SCALE_FACTOR


# Main function
def main() -> None:  # sourcery skip: low-code-quality
    """
    The main function.

    Raises:
        Exception: If loading calibration files fails.

    """
    try:
        camMatrix = np.load(Config.CAM_MATRIX_PATH)
        distMatrix = np.load(Config.DISTORTION_MATRIX_PATH)
    except:
        print('Loading calibration files failed')
        return

    # Initalizing cam and board params
    dictionary = cv.aruco.getPredefinedDictionary(Config.ARUCO_DICT)
    board = load_charuco_board()
    cam = cv.VideoCapture(0)
    print("Camera Running: " + str(cam.isOpened()))

    # Game set up
    score = 0
    goal_coord = generate_random_point()
    x_bound = (goal_coord[0] - Config.GOAL_SIZE/2, goal_coord[0] + Config.GOAL_SIZE/2)
    y_bound = (goal_coord[1] - Config.GOAL_SIZE/2, goal_coord[1] + Config.GOAL_SIZE/2)
    scored_point = False

    # Main loop
    while True:
        ret, frame = cam.read()

        if ret:
            # Handel key escapes
            key = cv.waitKey(1) & 0xFF
            if key == ord('q'):
                break
            elif key == ord('r'):
                score = 0
            elif key == 32: #space bar press
                Config.DISPLAY_MODE = not Config.DISPLAY_MODE
```

```
                if not Config.DISPLAY_MODE:
                    cv.destroyWindow("Raw Camera Feed")
                    cv.destroyWindow("HSV view")
                    cv.destroyWindow("Mask")
                print("Display mode: " + str(Config.DISPLAY_MODE))
            elif key ==ord('c'):
                Config.SHOW_CENTER = not Config.SHOW_CENTER
                print("Display Center: ", str(Config.SHOW_CENTER))
            elif key in Config.MODES:
                Config.GOAL_SIZE = Config.DIFFICULTIES[key]
                x_bound = (goal_coord[0] - Config.GOAL_SIZE/2, goal_coord[0] +
Config.GOAL_SIZE/2)
                y_bound = (goal_coord[1] - Config.GOAL_SIZE/2, goal_coord[1] +
Config.GOAL_SIZE/2)
                score = 0

            # Image perspective transform
            frame = cv.undistort(frame, camMatrix, distMatrix)
            corner_ret, perspective = drawCorners(frame, dictionary, board, camMatrix,
distMatrix)

            if perspective is not None:
                # Get the balls position
                x, y = get_ball_position(perspective.copy())

                if Config.SHOW_CENTER:
                    draw_center(perspective, x, y)

                if not scored_point:
                    draw_square(perspective, goal_coord, side_length= Config.GOAL_SIZE)
                    if x_bound[0] <= x <= x_bound[1] and y_bound[0] <= y <= y_bound[1]:
                        scored_point = True
                if scored_point:
                    score += 1
                    print(f"You Score a Point!! \n Current Score: {score}")
                    goal_coord = generate_random_point()
                    x_bound = (goal_coord[0] - Config.GOAL_SIZE/2, goal_coord[0] +
Config.GOAL_SIZE/2)
                    y_bound = (goal_coord[1] - Config.GOAL_SIZE/2, goal_coord[1] +
Config.GOAL_SIZE/2)
                    scored_point = False

                cv.imshow("Perspective", perspective)
                if Config.DISPLAY_MODE:
                    cv.imshow("Raw Camera Feed", frame)
                    cv.moveWindow("Raw Camera Feed", 250, 0)

    print("Camera Released")
    cam.release()
    cv.destroyAllWindows()
```

```
if __name__ == '__main__':
    main()
```