

Ball Balancing Table

Berkay Ataçoğlu

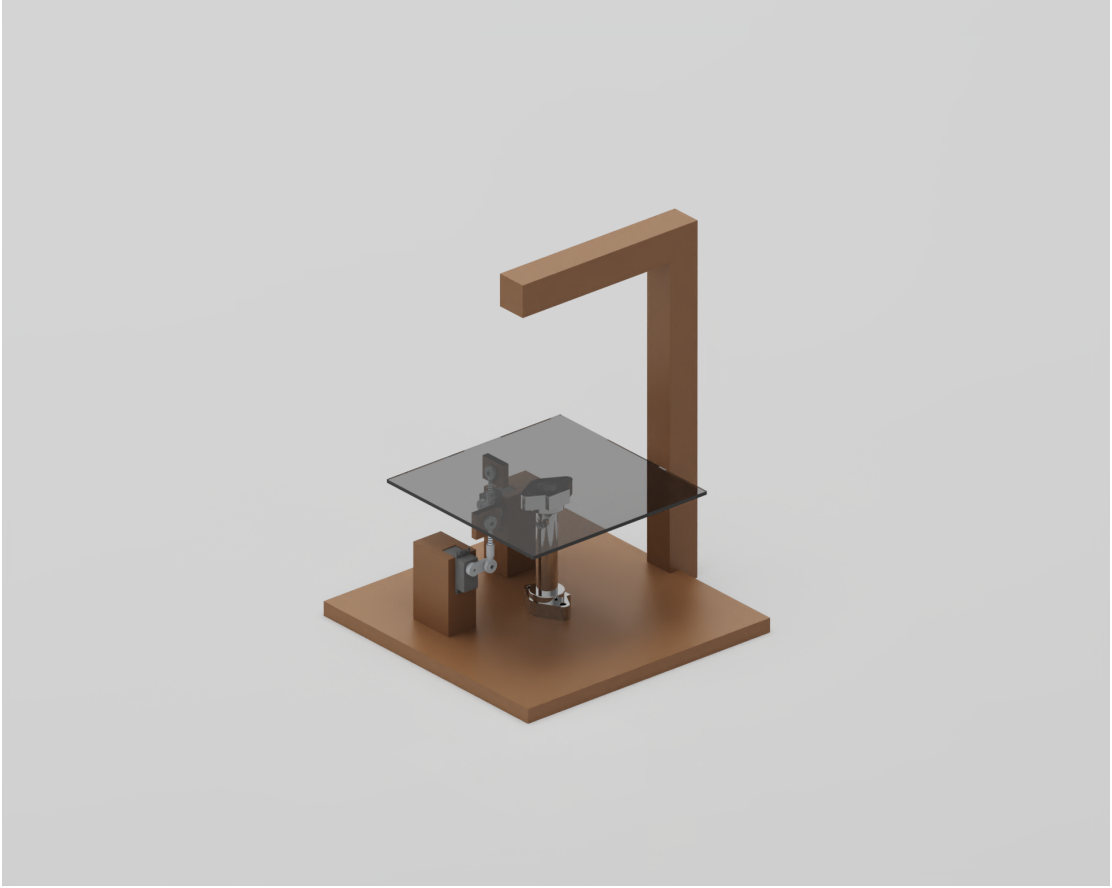


Figure 1: Solid Edge render of the ball balancing table. This is based on the actual physical machine I've built.

1 Introduction

Ball balancing tables are widely used in laboratories to demonstrate the complexities of controlling nonlinear systems. These systems are inherently unstable and require continuous adjustments to maintain equilibrium, making them a popular tool for teaching and researching control theory. The ball balancing table is conceptually similar to an inverted pendulum, showcasing intricate dynamic behaviors that pose challenges for real-world implementation. In the real world, the base principles of the “ball balancing table” are used in aerospace, robotics, and various fields. By simulating this kind of system in MATLAB Simulink or Python environments, we aim to get knowledge and experience in solving similar balancing nonlinear problems by using advanced control techniques such as PID. This project aims to design and simulate a ball balancing table capable of maintaining a ball at its center, regardless of external disturbances or initial conditions. The system uses simulated servomotors to tilt the table and a virtual position detection system to track the ball’s movement. Inspired by models like the ACROME Ball Balancing Table, this study explores the design, modeling, and control of a two-degree-of-freedom system, emphasizing the use of PID control for stabilization. Through this project, we aim to understand and address the challenges involved in controlling nonlinear systems while developing skills applicable to broader engineering problems. A major component of this project involves deriving the mathematical model of the system, considering the ball’s dynamics on a tilting surface. Using Newton’s second law and incorporating the small-angle approximation for simplification, the equations of motion are formulated. These equations serve as the foundation for simulating the system’s behavior in MATLAB Simulink/Python. The simulation provides a platform to test and refine the control system before physical implementation, reducing development time and cost.

The concept of ball balancing systems has gained a lot of interest in the academic as well as industrial platforms because of its complex dynamics and its usability in various fields. ACROME BBT (ball balancing table) is a well-known and a good example of a real-world application of this problem because of this in this project, as a baseline model we will use the ACROME BBT and we will use a similar approach to create a simulation of this model in Python. The system uses one camera tracking system and two servo motors linked to the table frame with rotary joints via rod end bearings.

The ACROME Ball Balancing Table is one of the most commonly used benchmarks, most systems use a camera display and servo motor real-life connected example by of rotary nonlinear joints control to and manipulate it the is tilting an apparatus. In academia, works have been devoted to the development of the control algorithms for such systems with the use of PID-regulators, state-space approach, and machine learning. Designed an observer-based feedback controller for ball balancing tables and showed that the proposed controller improves the system stability and robustness for different conditions. Some research works have also focused on the development of the hardware, for instance, by using high-speed cameras as well as enhanced actuation systems with an aim of enhancing the response time as well as accuracy.

Besides, being used in educational environments, ball balancing principles have been applied to robotics and aerospace as well. For instance, humanoid robots use the same control equations to manage the balance during the movement, and in the aerospace field, these concepts are used for stabilizing the payloads and platforms.

2 Methods

We aim to control the ball's position on a tilting table with two degrees of freedom. We want the ball to always be on the table and not fall off. The table should react to the outside disturbances and various initial conditions of the ball.

We can divide this problem into smaller problems.

- Modeling the system and deriving the equations of motion.
- Extraction of the ball's position, velocity, and acceleration data.
- Conditioning the data so that we can manipulate it using our desired programming environment.
- Tilting the table using actuators to influence the ball to its desired position.

2.1 Actuators

With 2 DOF, the plate can be tilted around the X and Y axes. Accuracy is crucial because the error in the actuator will cause an error in the plate angle. The most suitable actuators are the stepper motor and servo motor. Because the servo motor has a more flat and direct relationship.

The servo systems are more suitable for high speed, high torque applications, for our case high speed actuation is a must. Therefore our choice is a servo motor.

2.2 Sensors

In this project, the position sensor is one of the most crucial components, as the position of the ball has to be determined at all times in order to manage the system and hence, guide the ball to the desired location. Finally, a camera has been selected to be the link between the mechanical design and the control system. The reasons will be explained in the following sections.

- **Touchscreen:** Uses a capacitive or resistive touch interface to detect the position of the ball directly on the table surface. It offers accurate and fast data. No need for external components, easy to calibrate. But it picks up more noise, high-resolution ones quickly get expensive, and they're not as durable.
- **Position Sensors:** Use sensors such as potentiometers, encoders, or infrared/ultrasonic sensors to track the ball's position. They're inexpensive but not as precise. They require more setup and calibration. That's mainly why we didn't use them.
- **External Camera:** Uses a camera placed above the table to track the ball's position via image processing. Offers great tracking capabilities. Though significantly more costly than a set of position sensors, not as expensive as a high-resolution touchscreen. Requires an image processing algorithm. Introduces computational delay.

It is advantageous to select a camera over using resistive and capacitive touchscreens for our project, particularly in the matter of tracking and setting the position of the ball. The camera enables accurate, non-contact, and real-time positioning and can therefore follow changes in the position of the ball making it suitable for the ball balancing table that requires feedback.

2.3 Embedded Computer

Arduino is an open-source electronics platform designed for building interactive projects. Basically, Arduino is made of microcontroller boards through which you can connect with sensors, motors and other devices. While traditional computing systems can run full-fledged operating systems and complex applications, Arduino operates on much simpler code that is compiled and run on the microcontroller itself, without the need for an operating system, which makes it perfect for control applications in real time. The Arduino Integrated Development Environment or Arduino IDE is very easy to use, and one can write and upload code in C/C++ without much hassle. This makes it particularly accessible for both beginners and experienced developers looking to create diverse projects. One of the main advantages of Arduino is its real-time processing capabilities. Since Arduino runs without an operating system, it directly executes the code uploaded to it, providing minimal delay between input and output. This is particularly crucial for applications like our ball-balancing table, where the system must respond quickly to changes in the ball's position and adjust the table accordingly. Arduino's ability to handle real-time control of sensors and actuators with minimal latency makes it an excellent choice for your project, where precision and speed are key factors for success.

3 Design

To extract the position, velocity, and acceleration data of the ball relative to the table's center, we've decided to use a high-speed camera rigidly attached to the tilting table. The camera rotates with the table itself and always looks toward the center of the table from above. Since the distance between the camera and the table is constant, and not very relevant, one single camera is sufficient to capture both x and y coordinates of the ball.

The diameter of the ball is 1.5 cm, and the width and length of the table are 20 cm. The greater the difference between the table dimensions and the ball's radius, the less error the camera will have in capturing the ball's position. The ball must be made of a heavy material. The camera will be fixed to the table on which the ball rests, not the base plate. In this way, the loss of information on one axis will be prevented when converting the three-dimensional environment encountered in camera use into a two-dimensional image. Therefore, the camera will be held by arms attached to the corners of the table and aligned exactly with the center point of the table. The distance of the camera to the plate is also one of the important measurements. To minimize errors in camera measurements, the camera should not be too close to the table. This distance is determined as 25 cm. Likewise, to reduce the error rate and enable the camera to detect the ball better, the table and the ball should be in contrasting tones that are easily distinguishable.

We will choose two contrasting colors for the ball and the table's surface. This will aid in the segmentation process. We will assume the position of the ball to be the center of the segmented area corresponding to the ball. This also introduces an issue. While the position error of the ball at $x,y=0$ is 0, it increases as the ball moves towards the ends of the table. But this is a small error and is negligible for our situation. Also, statistically, the ball would not be toward the ends of the table for our operation range. If it did, the ball would most likely fall off anyway. We would then need to condition this data and be able to import it into our desired programming environment for designing a controller. It is Python for our case.

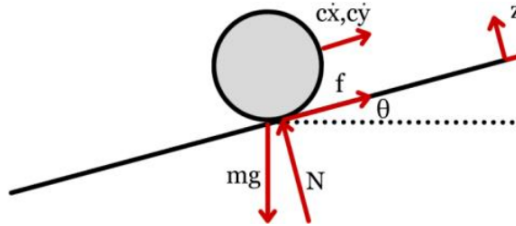


Figure 2: FBD of the ball

Figure 2 shows the free body diagram for our ball. W is the weight, N is the normal force the table exerts on the ball, and f is the friction force that will cause the ball to rotate. θ is the table tilt angle with the vertical.

Writing Newton's 2nd law for the x,y directions based on the standard coordinate assignment where a positive tilt yields a downward acceleration toward the negative axis:

$$m\ddot{x} = -W \sin(\theta) - f - c\dot{x} \quad (1)$$

Now, writing the moment equilibrium equation with respect to the ball's center, where α represents the rotational angle of the sphere:

$$I\ddot{\alpha} = fr \quad (2)$$

for a uniform solid sphere, the moment of inertia is;

$$I = \frac{2}{5}mr^2 \quad (3)$$

Also, assuming no-slip rolling condition:

$$\ddot{\alpha} = \frac{\ddot{x}}{r} \quad (4)$$

Plugging in Equations 2, 3 and 4 into Equation 1 becomes;

$$\ddot{x} + \frac{5c}{7m}\dot{x} = -\frac{5}{7}g \sin(\theta) \quad (5)$$

Now, we need to model the relation between the servo motors, and the table. Later, we'll assume that our table will operate between tilt angles $-15^\circ < \theta < 15^\circ$. We can also assume that link 2 will stay vertical throughout this whole range of motion. This will make our model way simpler.

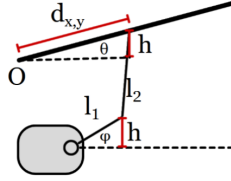


Figure 3: FBD of the linkages

$$\sin \theta = \frac{l_1}{d_x} \sin \phi \quad (6)$$

We can directly plug Equation 6 into Equation 5.

$$\ddot{x} + \frac{5c}{7m}\dot{x} = -\frac{5gl_1}{7d_x} \sin(\phi) \quad (7)$$

Finally, using small-angle approximation, our equation of motion becomes;

$$\ddot{x} + \frac{5c}{7m}\dot{x} = -\frac{5gl_1}{7d_x}\phi \quad (8)$$

3.1 Modeling

The conceptual design of the mechanism with the camera is not connected to the plate.

1. Camera
2. Ball
3. Plate

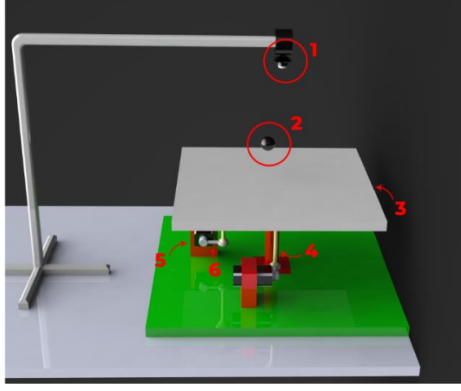


Figure 4: Assembly render

- 4. Long arm
- 5. Servo motor
- 6. Short arm

3.2 Sensing

We’ve decided to use two servo motors attached to the table with the help of two links. They’re both only responsible for the tilt of the table in their direction alone (One motor causes tilt in the x-direction, the other in the y-direction). We could’ve instead used magnetic actuators, linear actuators, etc. But for our problem, they wouldn’t bring any meaningful improvement in performance and greatly drive the cost.

The servo motor of our choice was TowerPro MG996R. It’s an inexpensive but capable servo motor. The specifications for the motor are given below.

Table 1: Specs of TowerPro MG996R.

Weight	55 g
Dimension	$40.7 \times 19.7 \times 42.9$ mm
Stall Torque	11 kg cm
Operating Speed	0.15 sec/60°
Operating Voltage	4.8 V to 6.6 V
Operating Range	180°
Temperature Range	0 °C to 55 °C

TowerPro MG996R does not include an internal encoder. It’s not able to output its motor angle. That means we can’t directly and specifically set its motor angle. That said, we can still calibrate the motor before use. TowerPro MG996R supports PWM signals and is compatible with Arduino and its own servo motor library. For our project, we’ve decided to use an external camera module based on the high-speed OV9281 sensor framework, complete with integrated on-board visual processing features. Specifications of the camera configuration are given below.

Looking at the specs for OV9281, one crucial design consideration is the lens field-of-view. For the camera to be able to see the whole table surface with 120° field of view, distance between the camera and the table should be sufficient.

Table 2: Specs of OV9281 global shutter camera.

Max Image Transfer Rate	1200 × 800 px at 120 fps
Dimension	24 × 25 × 12.7 mm
Temperature Range	0 °C to 50 °C
Field of View	120°
Power Requirements	max 156 mW

$$h = \frac{d_{\text{table}}}{2 \cdot \tan(35^\circ)} = 142.8 \text{ mm} \quad (9)$$

Given the geometry of the system, to capture the whole table surface with OV9281, the camera should be located at least 14.3 cm from the table surface vertically.

3.3 Controller

Equation 8 described our linearized equation of motion. We'll use transfer function approach to design a PID controller.

$$\frac{X(s)}{\Phi(s)} = \frac{-\frac{5gl_1}{7d_x}}{s^2 + \frac{5c}{7m}s} \quad (10)$$

Table 3: Our design parameters.

Gravitational Constant (g)	9.807 m/s ²
Motor Arm Length (l_1)	5.25 cm
Offset of the Actuation Point from the Center (d_x)	2.1 cm
Mass (m)	0.5 kg
Damping Coefficient (c)	0.05 kg s ⁻¹

Plugging in our design parameters into Equation 10 yields the final physical transfer function layout:

$$\frac{X(s)}{\Phi(s)} = \frac{-17.51}{s^2 + 0.0714s} \quad (11)$$

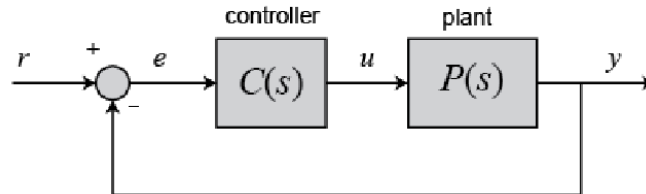


Figure 5: Block diagram of our system

For the design and tuning of our PID controller, we'll use a systematic approach. We'll first only add a proportional controller and tune its constant till we reach a system response that overshoots a little but quickly subsides.

PID control in Laplace domain;

$$C(s) = \frac{K_d s^2 + K_p s + K_i}{s} \quad (12)$$

Combining the controller transfer function with our plant transfer function;

$$C(s) \cdot P(s) = -17.51 \cdot \frac{K_d s^2 + K_p s + K_i}{s(s^2 + 0.0714s)} \quad (13)$$

Let's start our design by simulating our plant without any feedback loop.

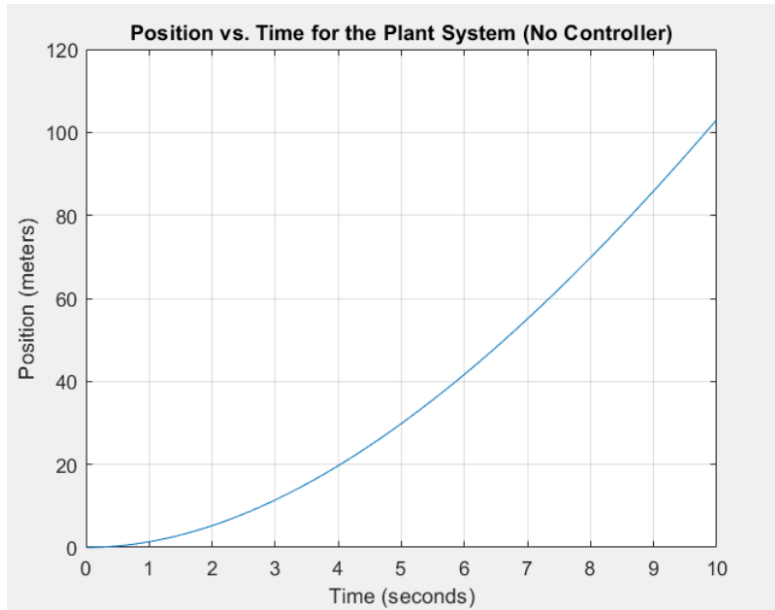


Figure 6: Position vs. Time for the System (No Controller)

As you can see from Figure 6, the system is initially not stable. We could already guess it since the system has no energy storage k term.

Now, let's start by introducing K_p and simulate the system on a step input.

$$K_p \cdot \frac{-17.51}{s^2 + 0.0714s} \quad (14)$$

When $K_p = 0.05$;

Now, we introduce the K_i term.

$$\frac{K_i}{s} \cdot K_p \cdot \frac{-17.51}{s^2 + 0.0714s} \quad (15)$$

When $K_p = 0.05$ and $K_i = 0.001$;

Finally, introducing the K_d term;

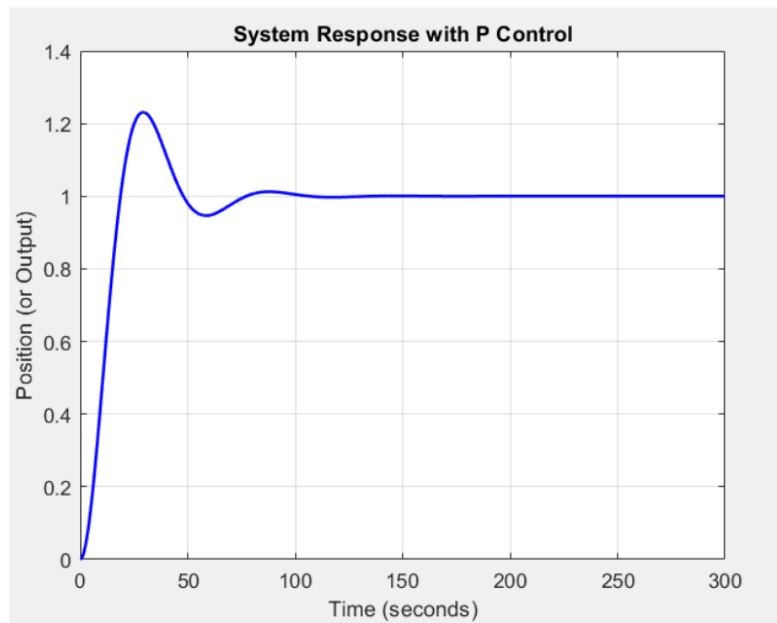


Figure 7: System response with proportional control.

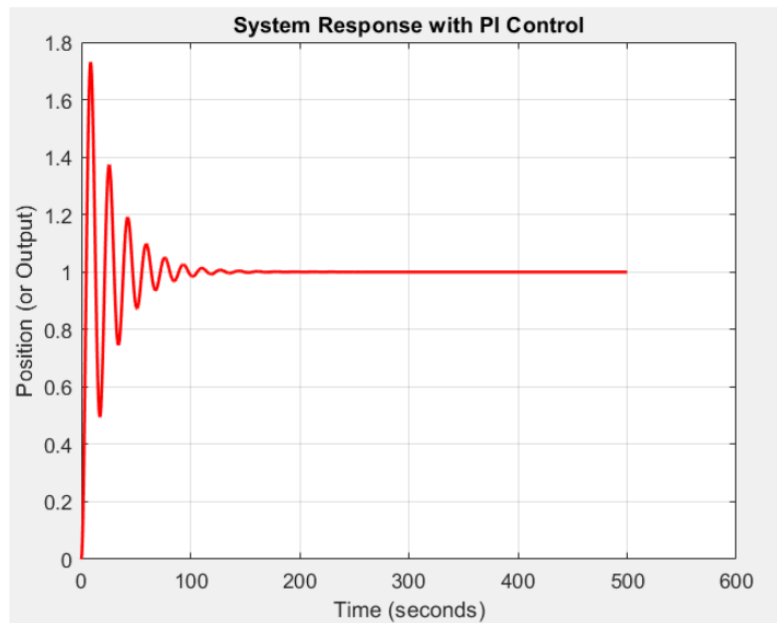


Figure 8: System response with proportional and integral control.

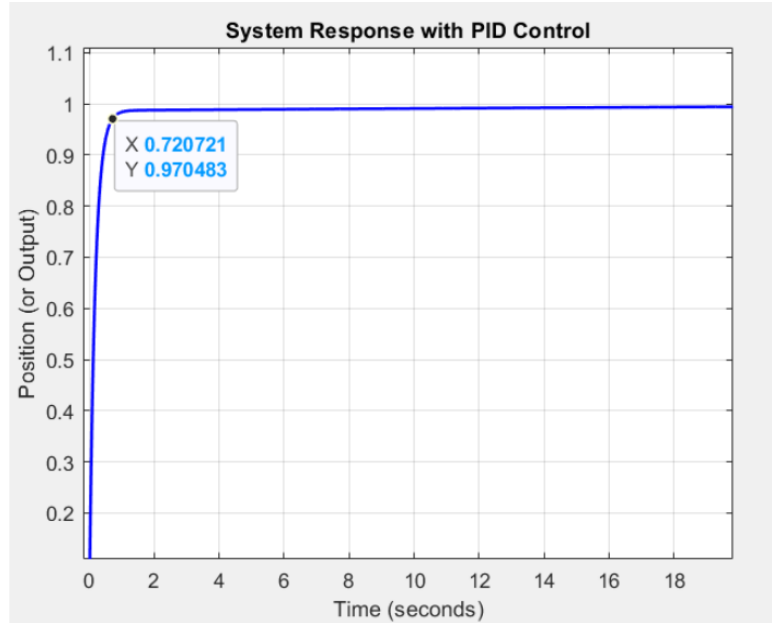


Figure 9: System response with proportional, integral and derivative control.

Now, we have a fully created PID controller. Now, let's give it some initial conditions, and check its response. To do this, I will use Python. The file bbt.py will be sent alongside this report.

After realizing that our initial PID parameters are not optimal and the system response is too slow, I've adjusted the parameters manually to get satisfactory system responses. Response of the system for various initial conditions are plotted below. Note that the units of both axes are centimeters. If a value in any given point crosses $[-2, 2]$ cm range, the ball falls off the table.

For the initial conditions;

- $x_0 = 0.5 \text{ cm}$
- $\dot{x}_0 = 5 \text{ cm s}^{-1}$
- $y_0 = -1.75 \text{ cm}$
- $\dot{y}_0 = 3 \text{ cm s}^{-1}$

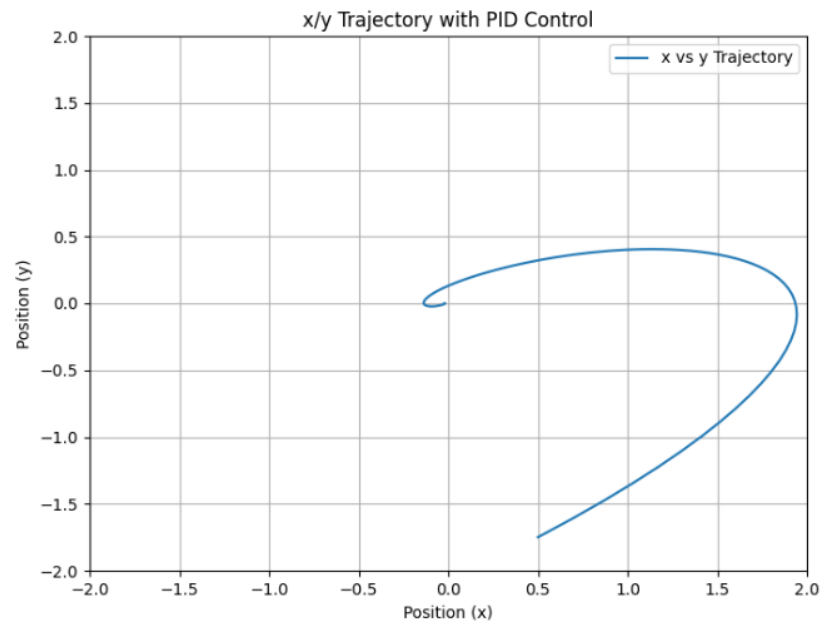


Figure 10: System response with above initial conditions.

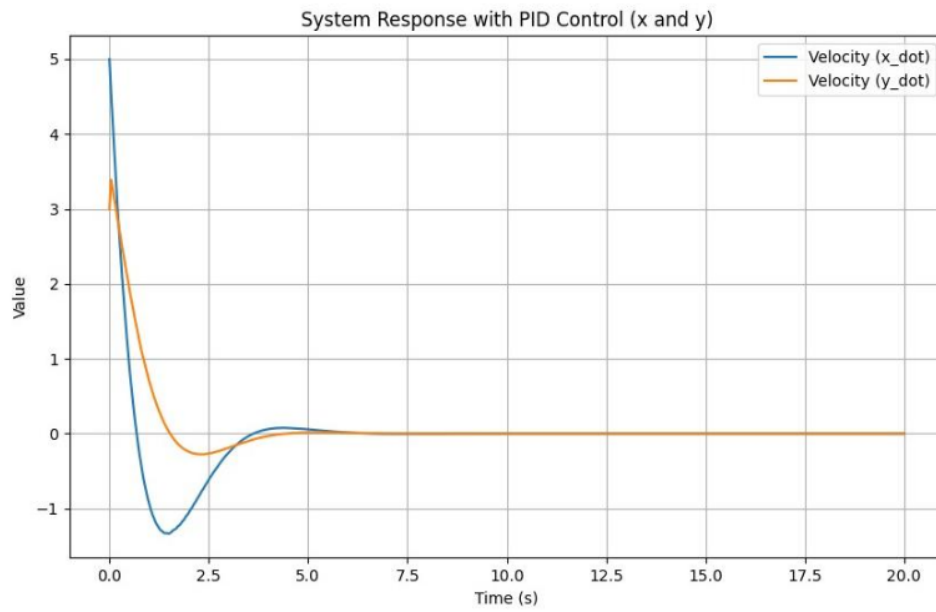


Figure 11: Velocity of X and Y vs Time - For the first initial conditions.

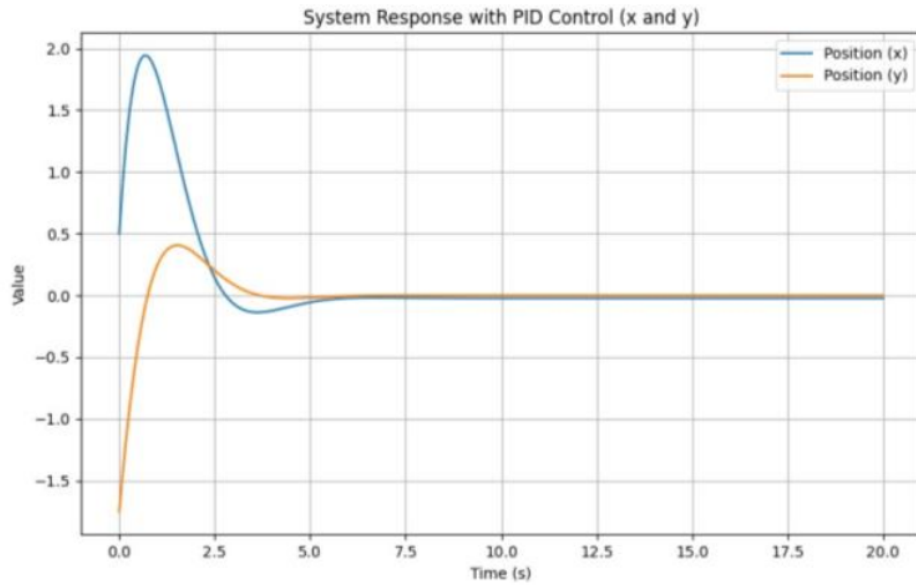


Figure 12: Position of X and Y vs Time - For the first initial conditions.

4 Conclusion

The aim of this project was to create a small passive table that can support a ball placed on it and keep it balanced for at least 10 seconds. The system systematically combined various mechatronics elements such as sensors, actuators, and control. An OV9281 camera was used as the main position sensor because of the capabilities of the camera to handle visual information and output position information to the controller at high speeds. This information was then supplied to an Arduino controller that implemented a PID control algorithm to vary the slope of the table with the help of servomotors. The system was able to balance the ball, and it was brought to a balanced or near-balanced position within the desired time.

5 Appendix

5.1 Python Code

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from scipy.integrate import solve_ivp
4
5 class PIDController:
6     def __init__(self, Kp, Ki, Kd):
7         self.Kp = Kp
8         self.Ki = Ki
9         self.Kd = Kd
10        self.prev_error = 0
11        self.integral = 0
12
13    def compute(self, setpoint, current_value, dt):
14        error = setpoint - current_value
15        self.integral += error * dt
16        derivative = (error - self.prev_error) / dt if dt > 0 else
17        0
18        self.prev_error = error
19        # Note: The controller output sign automatically adapts to
20        the negative plant gain
21        return self.Kp * error + self.Ki * self.integral + self.Kd
22        * derivative
23
24 def system(t, state, pid_controller_x, pid_controller_y, setpoint)
25 :
26     x, x_dot, y, y_dot = state
27     dt = 0.01
28     # System physical parameters mapping l_1 and d_x safely with
29     5/7 dynamics
30     g, l_1, d_x, c, m = 9.807, 0.0525, 0.021, 0.05, 0.5
31     phi_x = pid_controller_x.compute(setpoint, x, dt)
32     phi_y = pid_controller_y.compute(setpoint, y, dt)
33
34     # Implementing the physically accurate negative sign and 5c/7m
35     damping ratio
36     x_ddot = -(5 * g * l_1 / (7 * d_x) * phi_x) - (5 * c / (7 * m)
37     ) * x_dot
38     y_ddot = -(5 * g * l_1 / (7 * d_x) * phi_y) - (5 * c / (7 * m)
39     ) * y_dot
40     return [x_dot, x_ddot, y_dot, y_ddot]
41
42 setpoint = 0.0
43 initial_conditions_x = [1.50, 0]
44 initial_conditions_y = [0, 5]
45 initial_conditions = initial_conditions_x + initial_conditions_y
46
47 t_span = (0, 20)
48 t_eval = np.linspace(0, 20, 500)
49
50 # Negative gains are applied here to cancel out the plant's
51 negative gain loop
52 Kp = -0.75
53 Ki = -0.001
54 Kd = -2.0
```

```

47 pid_controller_x = PIDController(Kp, Ki, Kd)
48 pid_controller_y = PIDController(Kp, Ki, Kd)
49
50 solution = solve_ivp(system, t_span, initial_conditions,
51                      args=(pid_controller_x, pid_controller_y,
52                            setpoint),
53                          t_eval=t_eval)
54
55 plt.figure(figsize=(10, 6))
56 plt.plot(solution.t, solution.y[0], label='Position (x)')
57 plt.plot(solution.t, solution.y[2], label='Position (y)')
58 plt.plot(solution.t, solution.y[1], label='Velocity (x_dot)')
59 plt.plot(solution.t, solution.y[3], label='Velocity (y_dot)')
60 plt.xlabel('Time (s)')
61 plt.ylabel('Value')
62 plt.title('System Response with PID Control (x and y)')
63 plt.legend()
64 plt.grid(True)
65 plt.show()
66
67 plt.figure(figsize=(8, 6))
68 plt.plot(solution.y[0], solution.y[2], label='x vs y Trajectory')
69 plt.xlim(-2, 2)
70 plt.ylim(-2, 2)
71 plt.xlabel('Position (x)')
72 plt.ylabel('Position (y)')
73 plt.title('x/y Trajectory with PID Control')
74 plt.legend()
75 plt.grid(True)
76 plt.show()

```

Listing 1: PID Controller Simulation for x and y Axes

References

- [1] M. Hamdoun, M. B. Abdallah, M. Ayadi, F. Rotella and I. Zambettakis, "Functional observer-based feedback controller for ball balancing table," *SN Applied Sciences*, vol. 3, 2021.
- [2] "Ball Balancing Table," ACROME, [Online]. Available: <https://acrome.net/product/ball-balancing-table>. [Accessed 14 November 2024].
- [3] B. Appleton, "Ball Balancing Project," Fontys University of Applied Sciences.
- [4] J. Alsamarji, "Design and Control of a Ball and," p. 61, 2019.
- [5] A. H. FRANK and M. TJERNSTRÖM, "Construction and theoretical study of a ball balancing platform," p. 70, 2019.
- [6] M. Gräber, "PID Tuning Guide," TLK ENERGY, 16 3 2021. [Online]. Available: <https://tlk-energy.de/blog-en/practical-pid-tuning-guide>. [Accessed 5 1 2025].