EE-361: Principles of Feedback Control – Fall 2023

Milestone 2 Report

Group Members

Afsah Hyder
Ilsa Shariff
Shameer Masroor

Supervisor

Dr. Shafayat Abrar

December 14st, 2023

Habib University

Contents

1 Introduction

In the quest to master the art of precision and balance through control engineering, our project takes a critical step forward by focusing on the fundamental aspects of self-balancing systems.

Problem Statement Designing an effective solution for creating a self-sustaining beam-and-ball balance system presents a challenge. The goal is to employ a camera to monitor the ball's position on the beam and implement an actuator capable of swiftly repositioning the ball to the center of the beam when disturbances occur.

Control Objective We aim to design a self-balancing beam that will keep a ball at its center with unwavering stability.

2 Real-World Application

Our ball-beam system is a mechatronic device consisting of a ball rolling on a beam tilted by the stepper motor. The ball-beam system are used in variety of applications, including:

- Control System Design: Ball-Beam systems are a classic example of a non-linear, unstable control systems. This means that they are difficult to control, making them a popular example to allow students to learn about concepts such as feedback control, stability analysis and PID control.
- Robotics: The principles learned from controlling ball-beam systems can be applied to a wide range of robotic tasks. For example, same techniques can be used to balance a two wheeled robot or to track a moving object with robotic arm.
- Active Vibration Control: The ball-beam system's principles can be extended to active vibration control in large structures. By applying similar control techniques, one can mitigate vibrations in buildings or bridges.
- Gimbal Stabilization Systems: The control principles and techniques used in ball-beam can be used in this system. Just like ball-beam systems, gimbals display non-linear behavior influenced by factors such as inertia, friction, and external disturbances. This intricacy underscores the importance of developing effective control algorithms for optimal performance.

This report presents our mechanical structure in detail, clarifies our system's

block diagram, and derives relevant transfer functions for each of its constituent parts. When obtaining a transfer function is too complex or impractical, ordinary differential equations (ODEs) for the relevant components are presented as a substitute method. The system is simulated in great detail, the design of the controller is also presented covering both the compensated and uncompensated systems. Following that, we explore the nuances of putting our system into practice and clarify the main features of our code. The difficulties that arose during the implementation stage are carefully considered and examined. An examination of the practical applications of this self-balancing mechanism concludes the report.

3 Methods and Materials

3.1 Block Diagram

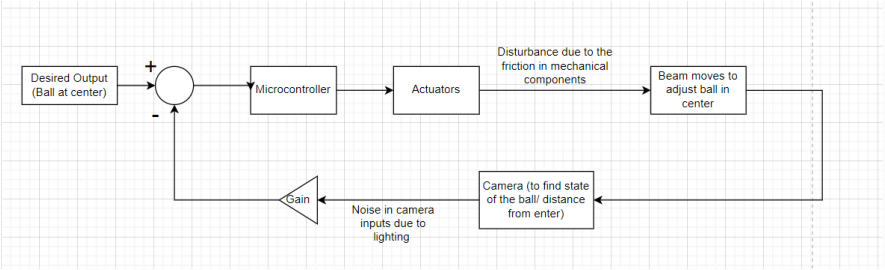

Figure 1: Block Diagram

- Plant: Beam on which the ball is placed
- Feedback Mechanism: Output of the camera determining state
- Controller: Micro controller/ PID
- Control Signals: Output of the PID controller to the actuators
- Manipulated Input: Position of the ball
- Reference Input: Position of the ball in the middle of the beam.
- Sensor: Camera
- Disturbance: Friction in the motor, friction in the ball bearings join beam to pivot unequal weight distribution in the beam,

Discussion on Block Diagram Our control variable is the position of the ball. The sensor (camera) feeds the current position of the ball with a gain to the computer + micro-controller which in turn generates a control signal to control the actuator. The actuator moves the beam and thus moves the ball. The camera keeps a track of the position of the ball.

3.2 Control Flow

The following section aims to explain the block diagram block by block in order to achieve a deeper understanding.

The camera sends real-time positional data of the ball to the controller. This controller determines the offset of the ball from the desired position, and generates a control signal to control the actuator to apply a torque on the beam. This torque rotates the beam through an angle; the ball which rests on the beam starts rolling towards the desired position. As the ball gets closer to the desired setpoint, the angle decreases.

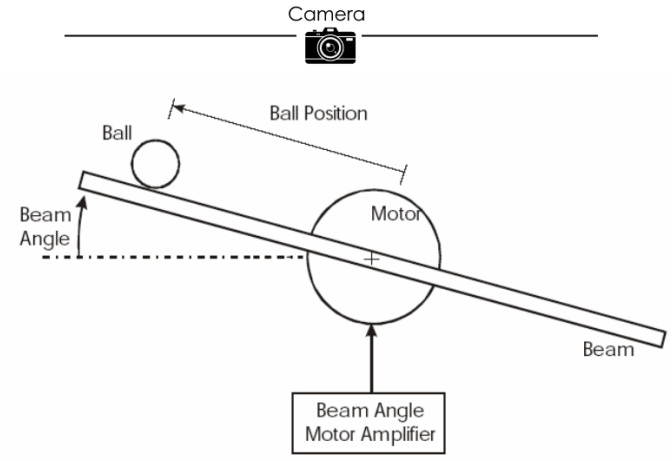

Figure 2: Expected Setup

3.3 What we did and how we did it

We started off with searching the internet for OpenCV code. OpenCV is a popular library of Python used for computer vision based projects. Our objective was to find code that is fast and reliable in its ball tracking abilities.

Once we identified the correct code, we set out to modify it to meet our needs. The code produced a camera window on which the ball of our choice of colour was encircled. We first modified the code such that the centre of the camera frame was marked on the window. This point would later serve as the reference position.

We then set out to divide the camera input frames into an upper and lower region; the upper region was the space on the frame above the equatorial line and above the marked frame center, and lower region where the ball was below the centre line of the frame, below the frame center.

Using Numpy, we then calculated the distance of the circle's centre enclosing the ball from the marked reference point on the frame. This was essentially the error from the reference position. When the ball was in the upper region of the frame, we marked the error as positive, and when the ball in the lower region, the error was negative.

Next came the part where we had to control the actuator using the error we had. We explored methods to control an Arduino via Python and found that Python's PySerial library could work for us. Using this library, we sent the control signal to the Arduino Uno via the serial port. However, before this control signal was sent, we performed a number of calculations on the error to produce a meaningful control signal. Our choice of controller was a PID (*proportional plus integral plus derivative*) controller. Tuning the PID constants to values that yielded a meaningful response, we sent this control signal to the Arduino which simply actuated the stepper motor, moving it's shaft, and thus the beam, to the required position. The library for stepper control used by the Arduino was MobaTools by MicroBahner, an independent developer.

This was all done at home using a make-shift setup to see the response we would get from our system.

Work on the actual hardware started in Week 7 of Fall 2023. The engineering workshop proved to be a valuable resource for us, allowing us to use the various kinds of wooden sheets and metal parts to materialize our project.

Construction was complete by Week 8. We then set out to find the correct camera for our setup (*we were using our laptop's camera for very early testing*).

We initially used a smartphone camera mounted to a ringlight on the hardware; this setup, however, proved ineffective due to the lag the interfacing between the computer and the smartphone camera introduced into the system. We then switched to a Konika 8100 series Smart TV camera but then did not use it due to low resolution. We finally settled on using the Intel RealSense cameras present in the project lab due to their good resolution.

Work done in parallel

While the selection of the appropriate code for ball tracking was made, it was important to select the correct actuator for our system and the associated drivers. After much deliberation and discussion, we settled on using stepper motors. We started learning how to control them using commands from Arduino's serial monitor.

Once the system hardware was made, work started on the simulation of our system. Using Simulink and the equations we had derived for our plant, we were able to produce a reasonable model of our project (results shown at the end of this report).

4 Mathematical Modelling

The following section presents the transfer functions of the sensor, actuator and plant in our project.

4.1 Actuator

Our project makes use of a NEMA-17 Hybrid Bi-polar stepper motor, model 17PM-K406V, as an actuator. This motor is connected to the main beam via a shaft coupling. A rotation of the motor shaft by a certain degree translates to a rotation of the beam by the same amount.

$$\frac{di_d}{dt} = \frac{V_d - Ri_a + N_r\omega L i_q}{L} \tag{4.1}$$

$$\frac{di_q}{dt} = \frac{V_q - Ri_q + K_m\omega - N_r\omega i_d}{L} \tag{4.2}$$

$$\frac{dw}{dt} = [K_m i_q - B\omega] \tag{4.3}$$

$$\frac{d\theta}{dt} = \omega \tag{4.4}$$

Derivation present in Appendix I

These equations were borrowed from S. S. Harish, K. Barkavi, C. S. Boopathi, and K. Selvakumar [2] "Modelling and Control of Hybrid Stepper Motor" International Science Press, 2016.

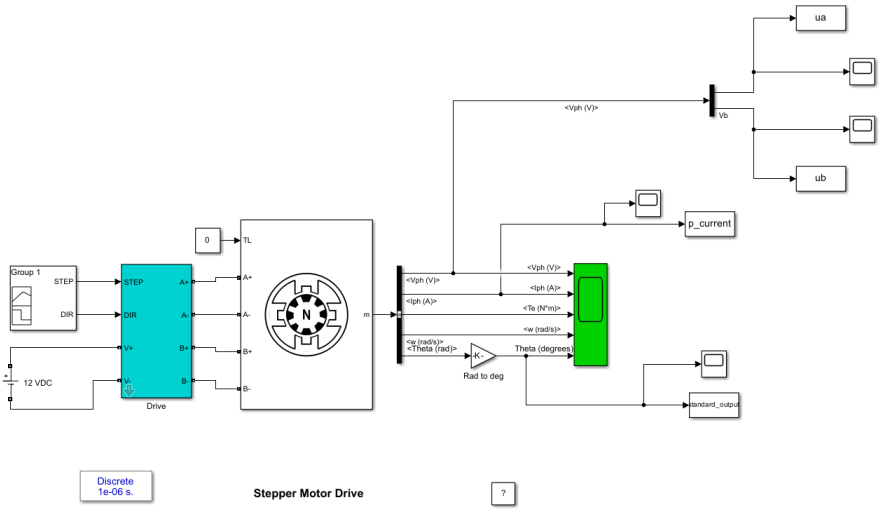

Figure 3: Simulink model using the Stepper Motor Module

4.2 Sensor

Our camera simply imparts a gain to the system.

The value determined for gain is 0.25.

Explanation in Appendix I

4.3 Plant Transfer function

$$P(s) = \frac{X(s)}{\Theta(s)} = -\frac{m_{ball}g}{(\frac{J}{r_{ball}^2} + m_{ball})}\frac{1}{s^2} \quad [\frac{m}{rad}] \tag{4.5}$$

All parameters involved in the above transfer function have their values listed in the Parameter evaluation section.
Derivation in Appendix I

4.4 Mechanical Structure

This sections presents the images of our mechanical structure from different angles. The setup is made from 5-ply wood and held together by nails and hand-made L-brackets.

The beam is made of 5mm acrylic. It is attached to the stepper motor shaft using a shaft coupler.

The ball is made of light-weight plastic ball painted red.

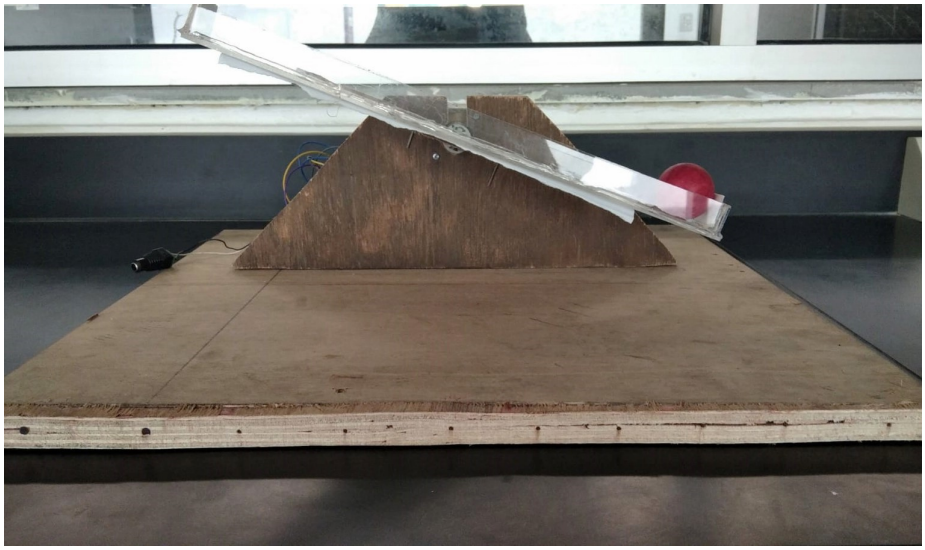

Figure 4: Front View

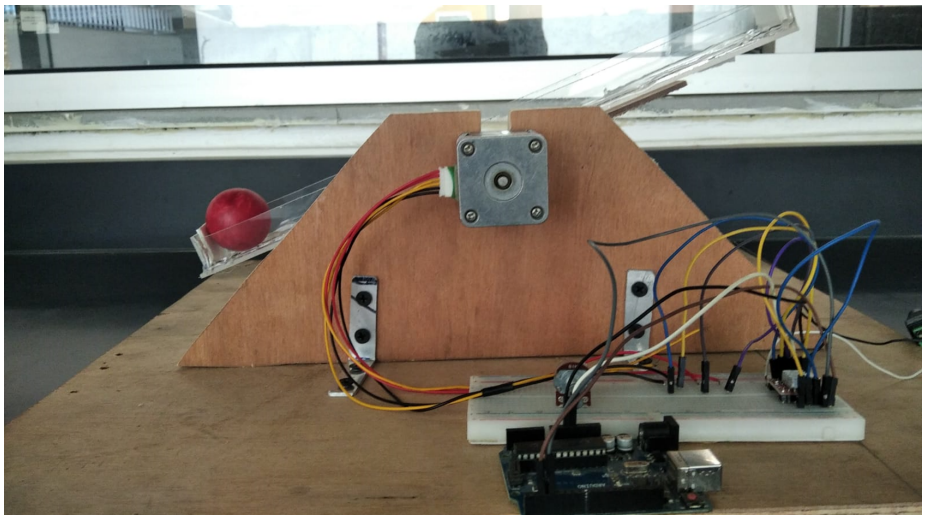

Figure 5: Back View

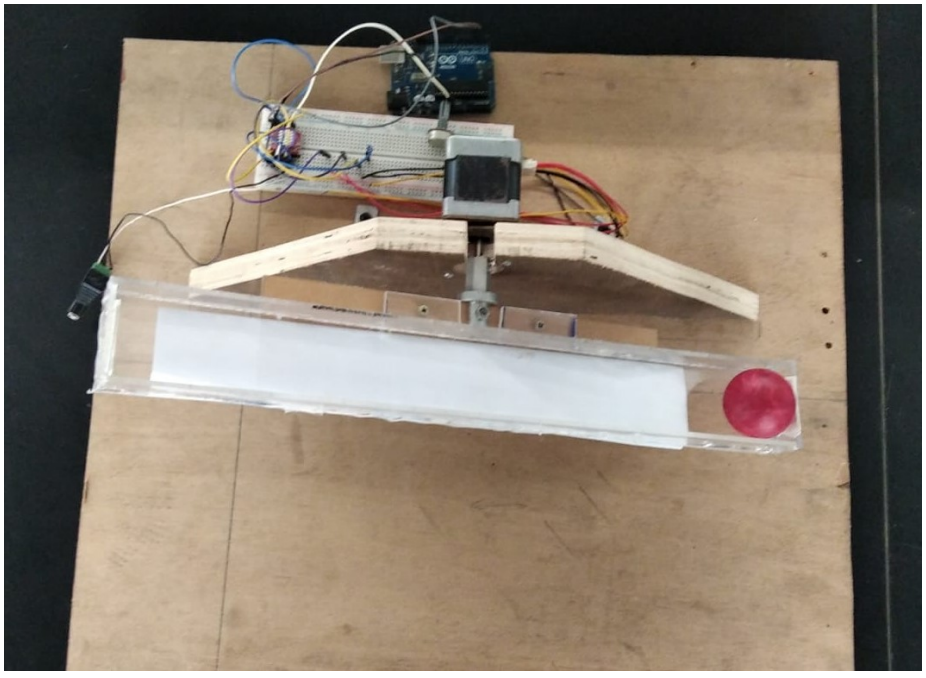

Figure 6: Top View

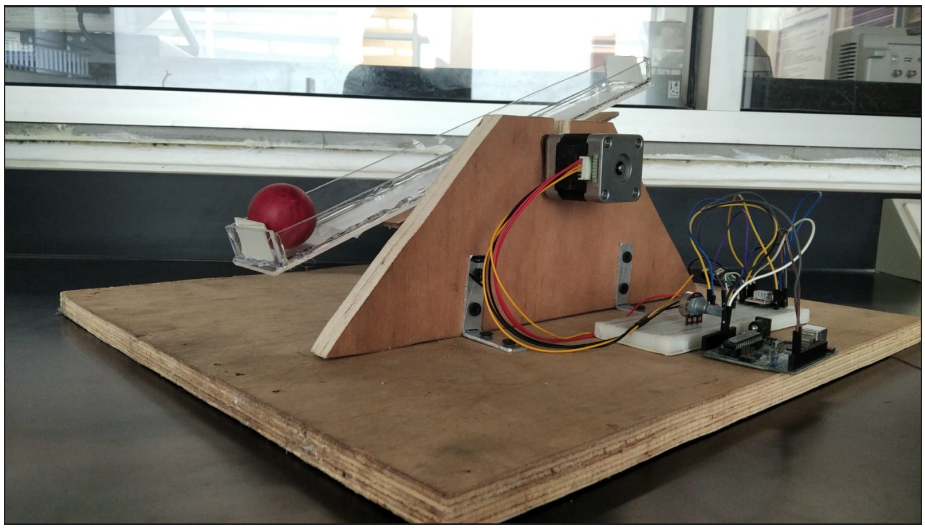

Figure 7: Angular View

5 Parameter evaluation

We experimentally determined the electrical and mechanical parameters where possible. For parameters which we could not obtain due to unavailablity of measuring instruments, we used values from the datasheet of the stepper motor.

Stepper Motor Parameter Evaluation

Parameter	Value
Inductance	5.7 mH (from datasheet)
Winding resistance (Red-wired coil)	12.2Ω (averaged from 3 readings)
Winding resistance (black-wired coil)	12.9Ω (avg of 3 readings of ohm-meter)
Detent torque	0.0147 kg-cm^2 (datasheet)
Rotor Inertia	8×10^{-6} (datasheet)
Rated Current	1.4 A (datasheet)
Rotor teeth	$N_r = 50$ (datasheet)
Holding Torque	$T_{holding} = 0.490$ Nm (datasheet)
Torque Constant	$K_m = T_{holding}/I_{rated} = \frac{0.490}{1.4} = 0.35$
Detent Torque Constant	$K_d = T_{detent}/\Delta\theta = \frac{0.0147}{1.8} = 8.16 \times 10^{-3}$
Coefficient of Viscous friction	$B = 0.1$ (assumed reasonable value)
Inertia Constant	$J = Torque/AngularAcc. = 0.001$ kg $\cdot$ m^2 (ds.)
Load Torque	$T_L = J\frac{d^2\theta}{dt} = (r_{beam})^2 m_{beam}\ \frac{d^2\theta}{dt}$

The datasheet for the stepper motor can be found here.

Explantion of parameters Here, we shall give a description of parameters

- Inductance: Opposes changes in current flow, affecting coil response time.
- Winding resistance: Limits current flow, impacting motor efficiency and heating.
- Detent torque: Tiny force holding rotor in place between steps, influencing positioning accuracy.
- Rotor inertia: Resists changes in rotor speed, affecting acceleration and braking.
- Rated current: Maximum recommended current for continuous operation without overheating.
- Rotor teeth: Number of teeth on the rotor, determining steps per revolution and torque generation.
- Holding torque: Maximum force the motor can maintain without losing its position.
- Torque constant: Conversion factor between current and generated torque.

- Detent torque constant: Ratio of detent torque to step angle, influencing positioning precision.
- Coefficient of viscous friction: Opposes rotation due to internal friction, affecting motor efficiency.
- Inertia constant: Ratio of torque to angular acceleration, determining acceleration/braking response.
- Load torque: Variable force acting against the motor, depending on the attached object's weight and movement.

Plant Parameter Evaluation

Parameter	Value
Mass of beam + shaft coupling	0.193 kg (avg. of 3 readings)
Length of beam	$l_{beam} = 0.381$ m (avg. of 3 readings)
Mass of ball	$m_{ball} = 1.91$ g $= 1.91 \times 10^{-3}$ kg (avg. of 3 readings)
Radius of ball	$r_{ball} = 19.92$ mm $= 1.99$ cm (avg. of 3 readings)
Gravitational acceleration	$g = 9.81$ m/s^2 (from theory)

Description of Parameters

- Combined Mass: Total weight of the beam and shaft coupling, influencing inertia and load on the motor.
- Beam Length: Distance between the fixed end and the point of attachment, affecting moment of inertia and lever arm for torque calculation.
- Ball Mass: Weight of the attached ball, contributing to the total inertia and load on the motor shaft.
- Ball Radius: Distance from the ball's center to its edge, determining its moment of inertia and lever arm for torque calculation.
- Gravitational Acceleration: Constant downward force acting on the ball, influencing its weight and potential energy.

6 Approach for Controller Design

6.1 Model Simulation

Our microcontroller has the primary task of controlling the logic levels of the STEP and DIR pin on the stepper motor driver, which drives it clockwise and anti-clockwise. The PWM input to the STEP pin controls the speed of the motor.

Let us delve deeper into the Simulink model presented earlier. Our model takes two inputs, STEP and DIR, just like our real setup takes.

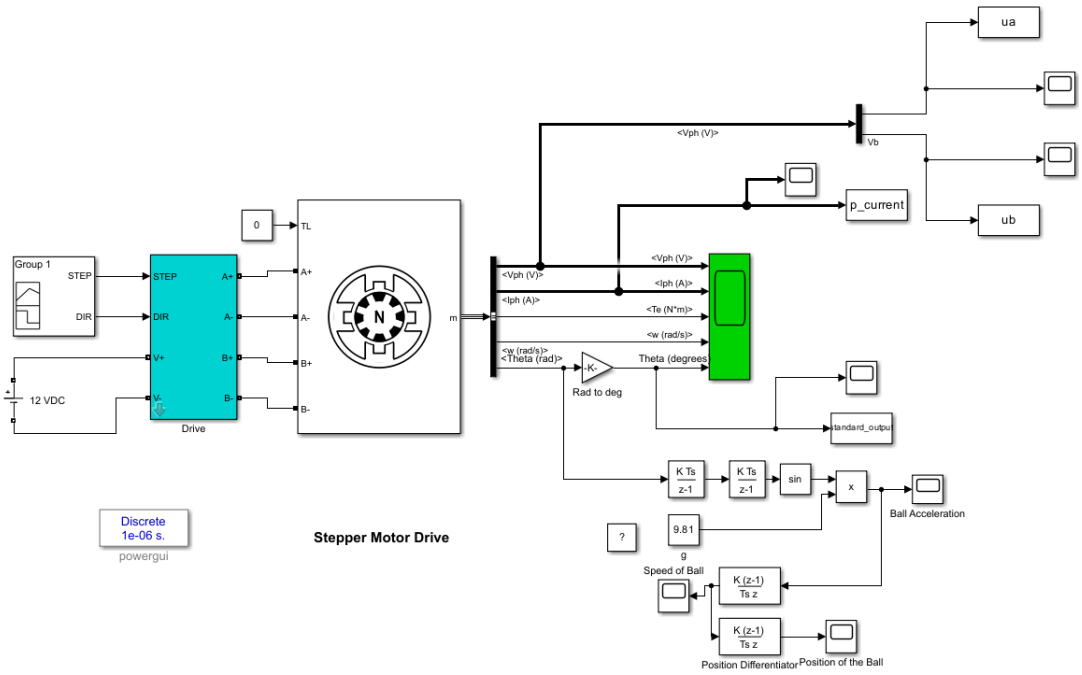

Figure 8: Model with ball accounted for

By toggling the values of DIR, we can change the direction of rotation of the stepper.

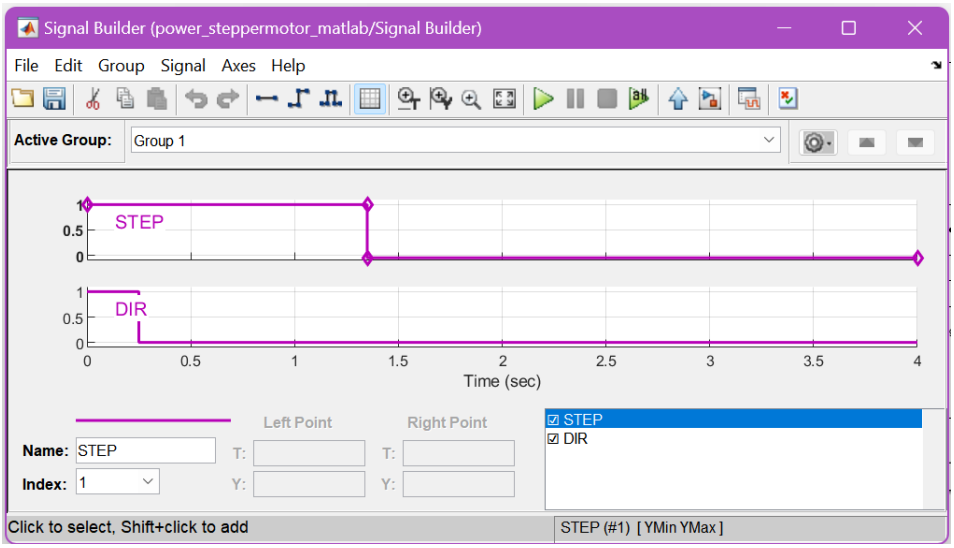

Figure 9: Signal Builder window

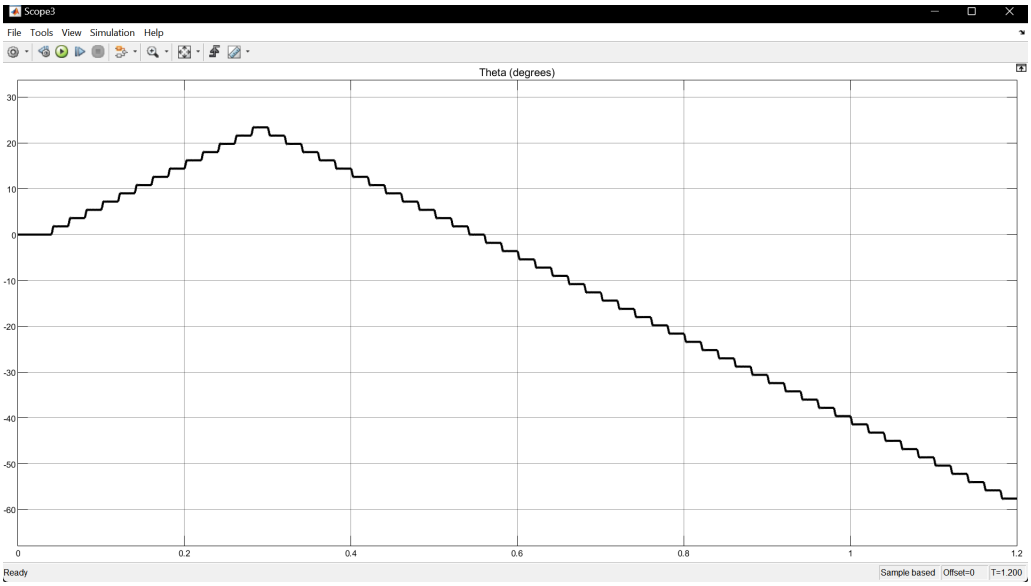

Figure 10: Scope Output in degrees rotated

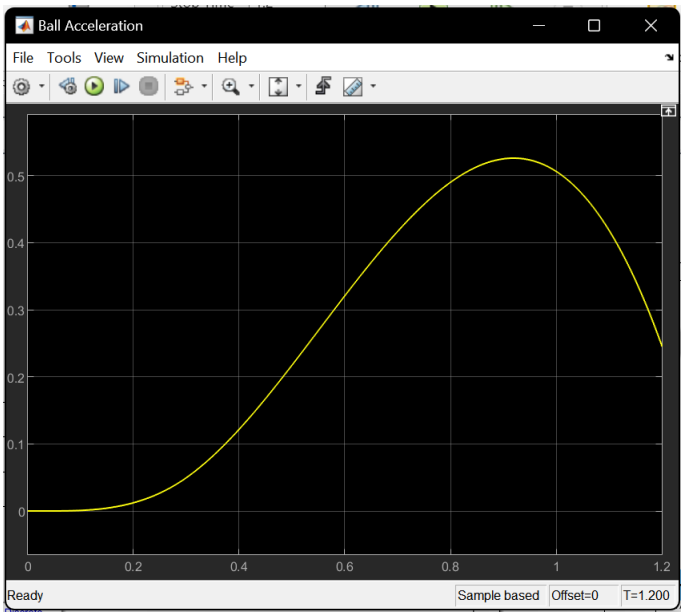

Figure 11: Acceleration of the Ball

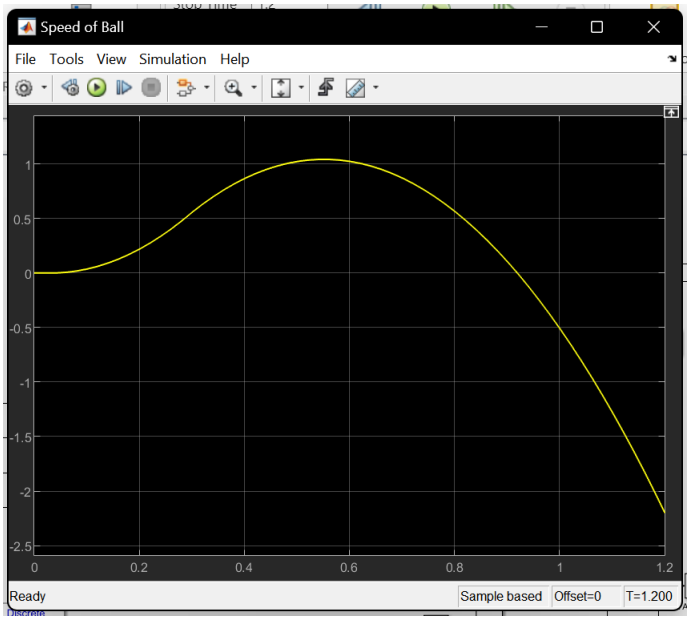

Figure 12: Speed of the Ball

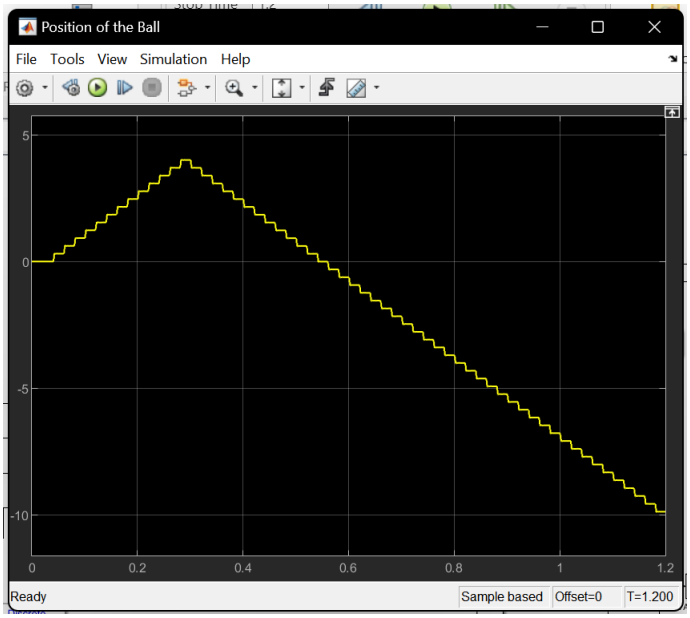

Figure 13: Position of the Ball

The graphical results obtained makes sense and can be related to the actual movement of the ball.

A practical demonstration can be viewed via this YouTube video.

6.2 Designing the Controller

The ball-beam balancing system is inherently unstable, requiring the implementation of a controller to achieve stabilization. Due to its inherent instability, a controller becomes essential to counteract deviations and maintain equilibrium. In this context, the utilization of a derivative controller proves valuable for enhancing transient response. The derivative controller anticipates the future behavior of the system based on the current rate of change, effectively mitigating overshoot and improving the system's ability to respond promptly to disturbances.

Moreover, the introduction of an integral controller becomes imperative to address steady-state errors and enhance the steady-state response of the ball-beam system. The integral controller continually accumulates error over time and applies corrective actions, ensuring that any persistent deviations from the desired state are gradually minimized. By combining derivative and integral controllers, the overall control strategy becomes well-rounded, effectively addressing both transient and steady-state aspects to achieve stability and accurate control in the ball-beam system.

We first have to observe the natural behavior of the system to truly appreciate the need for the controller. The system can be implemented simply as follows.

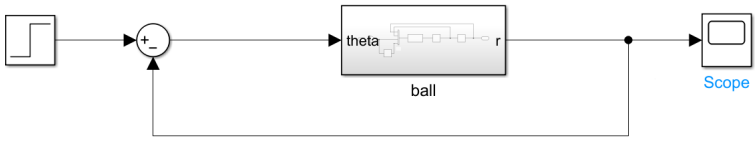

Figure 14: Simulated system

The system is implemented with a step input representing the desired distance of the ball from the side of the beam, set at 0.5 for center alignment. The subsystem labeled "ball" is responsible for tracking the motion of the ball and determining the changing values of r, representing the distance of the ball from the end of the beam. Figure 17 explains the subsystem.

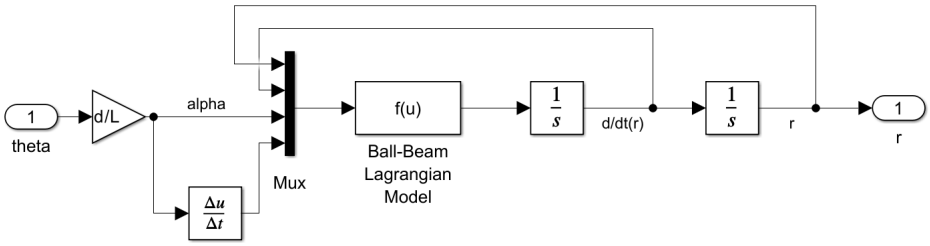

Figure 15: Subsystem

The Ball-Beam Lagrangian model incorporates the equation obtained during the derivation of the transfer function.

$$\ddot{r} = \left(-1/\left(J/R^2 + m\right)\right)\left(mg\sin(\alpha) - mr\dot{\alpha}^2\right) \tag{6.1}$$

The output of this subsystem, denoted as r, provides the distance of the ball from the end of the beam. This distance calculation takes into account the influences of inertial, gravitational, and centrifugal forces. To represent a disturbance in the system that needs overcoming, the initial value of r is set to a non-zero value. The constants utilized in this equation have been previously defined, with the main distinction being that m represents the sum of the masses of both the ball and the beam.

The following images show the uncompensated responses for different initial values of r.

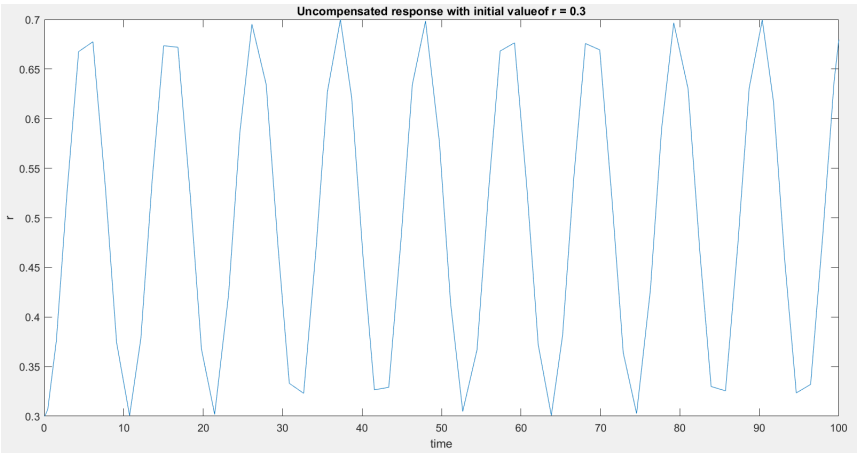

Figure 16: Response of the uncompensated system with the ball initial value of $r = 0.3$

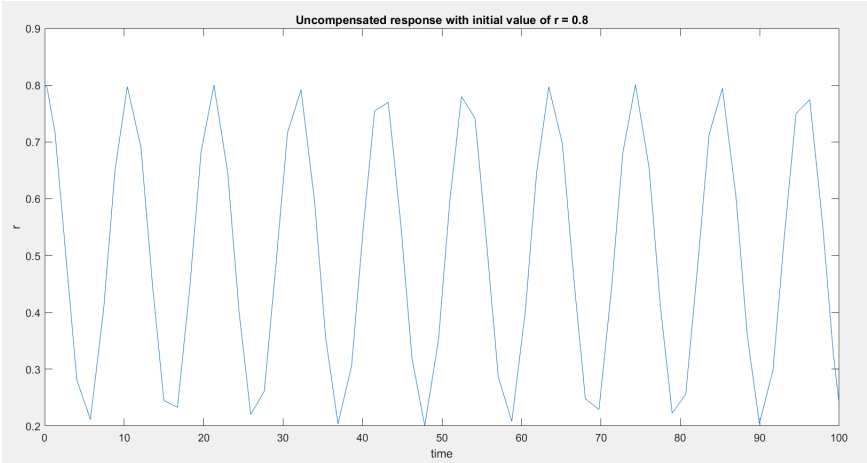

Figure 17: Response of the uncompensated system with the ball initial value of $r = 0.8$

The above response show the need for a controller to drive the ball to the center of the beam, and stabilise it. Hence, we use a PID controller to improve the transient and steady-state response to drive the ball to the center quickly. Figure 20 shows the simulated system with the PID controller.

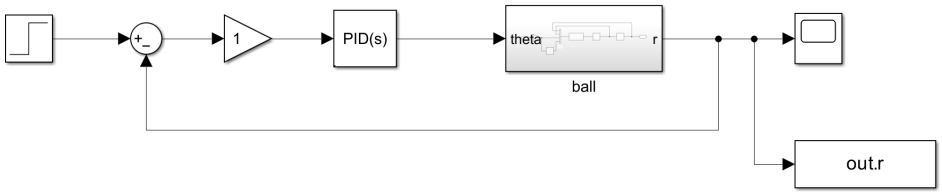

Figure 18: Simulated system with controller

The following images show the compensated responses for different initial values of r.

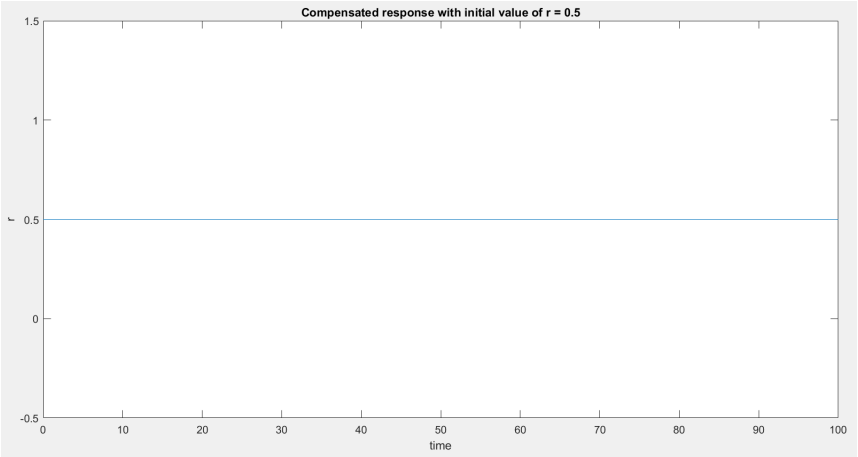

Figure 19: Response of the uncompensated system with the ball initial value of $r = 0.5$, that is the ball is at the center

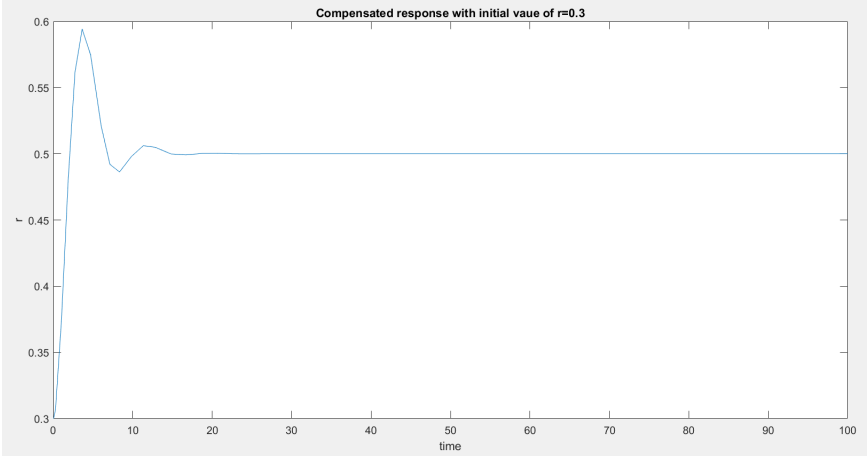

Figure 20: Response of the compensated system with the ball initial value of $r = 0.3$

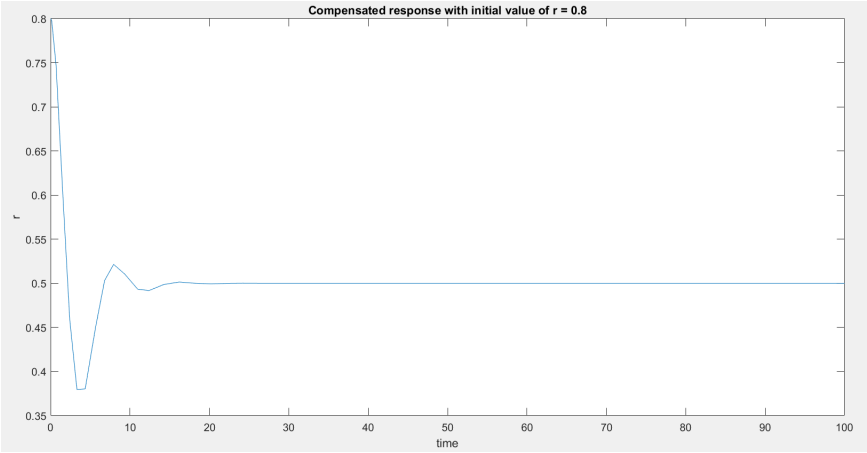

Figure 21: Response of the compensated system with the ball initial value of $r = 0.8$

These responses were obtained through trial and error, utilizing different PID constant values. The final responses were achieved with PID constants set to Kp=0.7, Ki=0.2, and Kd=2, which closely align with the values used in the hardware implementation (Kp=0.7, Ki=0.001, Kd=0.1). This iterative process in determining PID constants demonstrates the precision required to achieve optimal system performance.

The disparity between the PID constants used in simulation and the actual hardware implementation can be attributed to several challenges encountered during the real-world testing phase. Factors such as air resistance, variations in light intensities affecting input readings from the camera, jerks in the beam's movement, and the ball's tendency to almost bounce off the beam (given its lightweight nature) for higher values of Ki and Kd contributed to this difference. These challenges highlight the complexities involved in translating simulation results to a physical system, emphasizing the need for fine-tuning PID constants to account for real-world dynamics and ensure effective control in the ball and beam balancing system.

7 Implementing the System

The following section explains the implementation of this idea, it talks about the main components and functionalities of our code and how we achieved our desired results.

7.1 Video Processing and Display

- Camera capture and frame reading are handled by OpenCV.
- Distance and control signal are printed to the console for monitoring.
- A circle is drawn around the detected ball in the frame.
- The frame is displayed, and pressing 'q' exits the loop.

7.2 Ball Tracking

- **HSV Color Space:** The script utilizes the HSV color space to identify the red ball in the captured frame.
- **Red Color Ranges:** Two distinct red color ranges are defined to accommodate variations in lighting conditions.
- **Mask Creation:** Masks are created based on the defined red color ranges. These masks isolate the pixels corresponding to the red color of the ball.
- **Noise Reduction:** The created masks undergo an erosion and dilation process to remove noise and enhance the accuracy of the color segmentation.
- **Contour Detection:** Contours of the red object are found using the processed masks.
- **Largest Contour Assumption:** The largest contour is assumed to represent the ball, considering it as the main object of interest.
- **Minimum Enclosing Circle:** The minimum enclosing circle for the assumed ball contour is calculated to determine the center of the ball.
- **Distance Calculation:** The distance from the center of the frame to the center of the ball is computed using the calculated coordinates. This distance represents the measure of proximity of the ball to the center.

7.3 PID Control

- PID gains (kp, kd, ki) are defined.
- Error is calculated as the difference between the ball's center and the frame's center (converted to another unit by dividing by 4).

- Integral (`integral`) accumulates the sum of errors over time.
- Difference (`difference`) is the change in error compared to the previous frame.
- Control signal is computed using the PID formula: $kp \cdot \texttt{error} + kd \cdot \texttt{difference} + ki \cdot \texttt{integral}$.
- `kd` adjusts for the rate of change of error, becoming 1 only if the error is small and stable (within ± 15), effectively acting as a dead zone to avoid noise.
- The control signal is sent to the motor controller via the serial port.

This control signal is then used to turn the stepper motor by the Arduino. It's functionality is as follows.

7.4 Stepper Motor control

- The code uses the "MobaTools" library, which facilitates stepper motor control.
- The code implements a state machine with two states: IDLE and MOVING. The initial state is IDLE.
- Continuous monitoring of the serial port for incoming data to control the stepper motor's movement.
- Reading the next position value from the serial port and updating the 'nextPos' variable.
- Limiting the target position ('targetPos') to a range of -40 to 40 to prevent extreme movements.
- In the IDLE state, if the stepper motor is not currently in motion, setting a new target position ('targetPos') and transitioning to the MOVING state.
- In the MOVING state, checking for the completion of the stepper motor's movement. Upon completion, transitioning back to the IDLE state.
- The state machine continuously updates the stepper motor's position based on the received target positions. This process continues until the ball is positioned at the center of the beam.

8 Challenges faced

Development of a setup that works as well as ours required time, iteration, and a bit of intuition.

Our group faced a number of challenges before our project could work as expected.

8.1 Hardware Challenges

1. **Under-powered Stepper Motor Drivers** Our initial selection of A4988 as our stepper driver was met with difficulties; despite of the driver's capability of delivering 1A of current to each coil of the stepper motor, the IC on-board the driver could not cool itself fast enough to prevent overheating, despite of having a dedicated heat-sink. Overheating caused 3 out of 4 of our drivers to fail.

   As a last resort, and to ensure longevity, our group decided to invest in a high-powered stepper motor driver, the TB6600, capable of delivering current up to 4A per coil with a large heat-sink to allow for effective cooling.

2. **Beam free-play and jitter** Using half-stepping for the stepper motor, we faced the issue of excessive beam shake and jitter (rapid vibrations and jerky movement). We were able to identify the step size of the stepper motor as the source of this vibration. We were able to reduced jitter to a great extent using a smaller step size of the stepper, down to 1/8th of a step.

8.2 Software Difficulties

1. **Surrounding Colour Disturbance** Our system relies heavily on the input to the system from the camera. We wrote our OpenCV code such that it only tracks red objects. An issue was that our surrounding contained hues of red, and this interfered with the ball tracking code. We were able to fix this issue using MATLAB's Colour Thresholder and narrow out the hues of red within which our ball's red hue lied and used it to tune our ball tracking code so that it would only detect the hue of red we wanted it to.

2. **Arduino Serial Buffer Overflow** Our laptops and computing devices are vastly more powerful and fast in comparison to the micro-controller on the Arduino Uno. In the initial stages of the project, sending data as fast as our computers could filled up the serial buffer of the Arduino, causing it to stop responding. The Arduino would simply stop accepting step position inputs from the serial port and would eject itself from the computer's serial port. A solution to this was to periodically close the Serial buffer of the Arduino using Serial.flush(), allowing it to execute the commands sent earlier, allowing the buffer to clear.

9 Conclusion

Our project turned out as expected. We succeeding in creating the beam-and-ball balance system that performs satisfactorily.

The concepts that we put to play revolved around designing a good controller to stabilize our otherwise inherently unstable system. The entire process of deciding PID constants and relating it to theory was incredibly exciting as we could see how the system behavior changed with different constant values.

10 Team Work

Our group is happy with our progress and the overall functionality of the project. We will now work on tuning the PD controller to obtain a better response.

Work done by Shameer Shameer worked on sourcing the parts for the project which includes the stepper motor, motor driver, and the microcontroller. He also worked on writing the OpenCV Python code and interfacing the microcontroller to run directly off of the Python code output.

Work done by Afsah Afsah worked on improving the OpenCV code and implemented the PID controller in the code. She also helped in the formation of the relevant transfer functions for the plant and actuator as well as the Simulink Model. She worked on simulating and designing the controller.

Work done by Ilsa Ilsa worked on hardware of the project which involves making the motor mounts, base, beam, and shaft coupling. Ilsa also provided the camera for the project. She worked on simulating and designing the controller.

References

[1] M. Virseda, “Modeling and control of the ball and beam process,” Master’s thesis, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden, 2004.

[2] S. S. Harish, K. Barkavi, C. S. Boopathi, and K. Selvakumar "Modelling and Control of Hybrid Stepper Motor" International Science Press, 2016.

[3] https://ctms.engin.umich.edu/CTMS/index.php?example=BallBeam&section=SimulinkModeling

11 Appendix I - Derivation of Model

11.1 Actuator

A closed-form transfer function for a stepper motor is generally not straightforward to derive because stepper motors are typically nonlinear and exhibit discontinuous behavior. Transfer functions are commonly used to represent linear time-invariant (LTI) systems, and stepper motors are inherently nonlinear systems.

It is possible to linearize the transfer function but such models are limited in their application. Since our Beam-and-ball balance system makes use of microstepping down to 1/8th of a step, and PWM signals to control the position, speed, and acceleration of the stepper, relying on a simple linear transfer function is not enough as we are not providing input voltage as a function of time as in the case of a generic DC motor, but rather it is in the form of discontinuous pulses. As an acceptable alternative, we shall here present the differential equations for the stepper motor and model them using the appropriate stepper motor model in Simulink.

Our stepper motor has two coils, which can be represented as shown below:

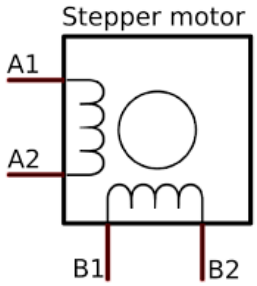

Figure 22: Coils of the stepper motor

We shall use Kirchoff's Voltage Law for obtaining the ODEs for the coils of the stepper.

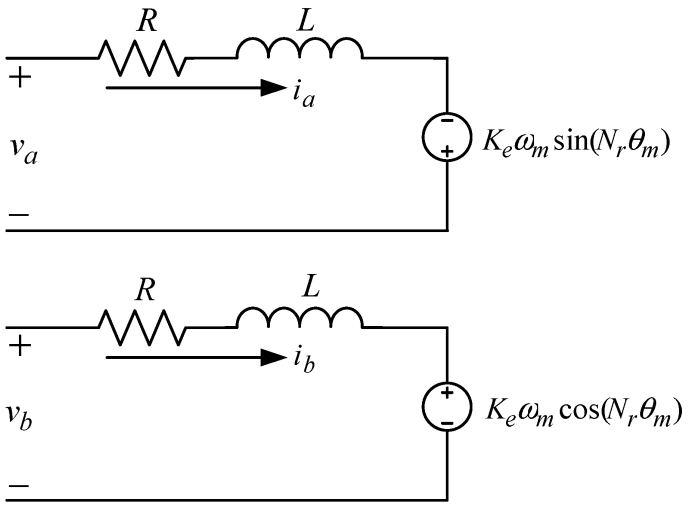

Figure 23: Coils of the stepper motor

$$\frac{di_a}{dt} = \frac{V_a - Ri_a + E_a}{L} \tag{11.1}$$

$$\frac{di_b}{dt} = \frac{V_a - Ri_b + E_b}{L} \tag{11.2}$$

From our analysis of DC Motors, the induced voltage is equal to

$$Ea = K_m w sin(N_r\theta) \tag{11.3}$$

It is worth pointing out that, in the control of the stepper motor, the input voltages to the coils are 90 degrees out of phase with each other. As a result, the current in the two coils i_a and i_b also exhibit a 90-degree phase difference, which plays a crucial role in the motor's operation.

$$Eb = K_m w cos(N_r\theta) \tag{11.4}$$

We can model this system as a mechanical circuit. The motor torque ODE will be written as

$$J\frac{dw}{dt} = -K_m i_a sin(N_r\theta) + K_m i_b cos(N_r\theta) - Bw - T_L + K_d sin(4N_r\theta) \tag{11.5}$$

$$\frac{dw}{dt} = [-K_m i_a sin(N_r\theta) + K_m i_b cos(N_r\theta) - Bw - T_L + K_d sin(4N_r\theta)]\frac{1}{J} \tag{11.6}$$

where v_a, v_b and i_a, i_b are the voltages and currents in phases A and B, respectively, w is the rotor (angular) speed, θ is the rotor (angular) position, R and L

are the resistance and inductance of the phase windings, N_r is number of rotor teeth, K_m, K_d is torque constant and detent torque constant, B is the coefficient of viscous friction, J is inertia constant and T_L represents load torque, which is assumed to be constant. In our project, the load torque is the torque exerted by the beam and ball on the rotor shaft.

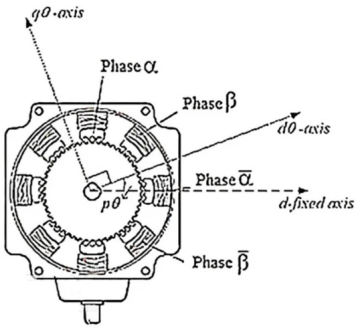

Figure 24: d-q representation of a stepper motor

Modelling of stepper motors is done in the q-d frame of reference where the frame of reference is changed from the fixed phase axes to the axes moving with the rotor. The dq axis (direct-quadrature axis) is a rotating reference frame that simplifies the mathematical analysis of stepper motors. By transforming the voltage equations from the axes to the dq axes, the model becomes more compact and easier to understand. This is achieved by means of a transformation.

$$M = \begin{pmatrix} cos(N_r\theta) & sin(N_r\theta) \\ -sin(N_r\theta) & cos(N_r\theta) \end{pmatrix} \tag{11.7}$$

The application of d-q transformation to the original system yields the following system of equations

$$\frac{di_d}{dt} = \frac{V_d - Ri_a + N_r\omega Li_q}{L} \tag{11.8}$$

$$\frac{di_q}{dt} = \frac{V_q - Ri_q + K_m\omega - N_r\omega i_d}{L} \tag{11.9}$$

$$\frac{dw}{dt} = [K_m i_q - B\omega] \tag{11.10}$$

$$\frac{d\theta}{dt} = \omega \tag{11.11}$$

These equations were borrowed from S. S. Harish, K. Barkavi, C. S. Boopathi, and K. Selvakumar [2] "Modelling and Control of Hybrid Stepper Motor" International Science Press, 2016.

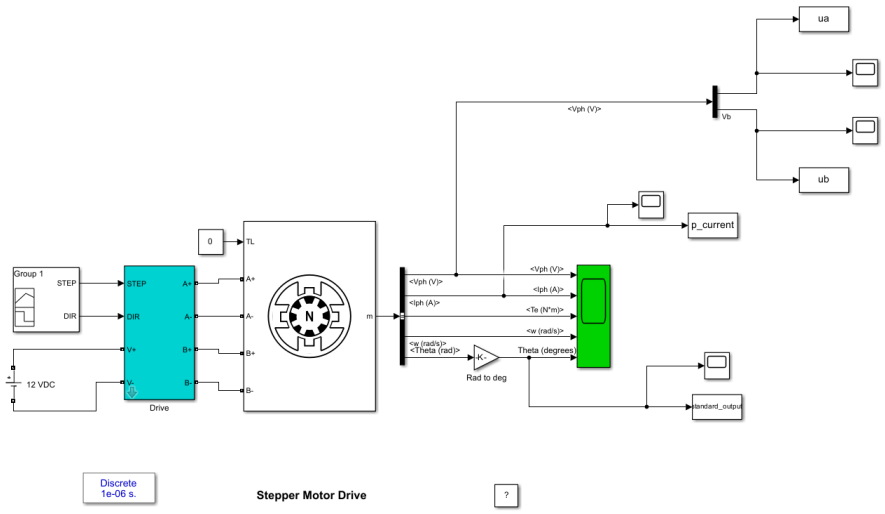

Figure 25: Simulink model using the Stepper Motor Module

11.2 Sensor

In this system, the camera serves as the primary sensor responsible for assessing the position of the ball on the beam. Specifically, it calculates the distance between the ball and the center of the beam, which is regarded as the "error" – signifying the deviation of the ball from the desired central position.

The error value derived from the camera's observations is subsequently utilized in our code to govern the operation of the stepper motor. We have employed the MobaTools library to compute the precise angle by which the stepper motor must rotate. This angle is determined based on the magnitude of the error, essentially quantifying the deviation of the ball from the beam's center. This process recurs until the error reaches zero, indicating the ball's arrival at the center and the successful balancing of the beam.

Furthermore, the camera's functionality in this context is notable. It takes as its input the initial ball-to-center distance and provides an output that effectively maps this distance onto the camera's screen. In essence, it serves as a conversion factor, transforming the real-world position of the ball into a corresponding position on the camera's screen. This mapping procedure is related to the vertical pixels of the screen and their relationship with the ball's distance from the center. So essentially, the camera's transfer function simplifies to a gain, translating the input ball distance it captures into a corresponding distance on the camera screen.

We experimentally determined the gain the camera provides to the input signal.

A value obtained experimentally was 0.25 to convert distances in pixels on the screen to cm. This was dependent on the distance the camera was from the set up. If the distance of the camera from the setup is changed, the gain will change.

It is worth pointing out that since the distance of the ball from the desired position was returned in pixels, it needed to be scaled down to control the step input to the stepper motor accordingly. This scaling was of $\frac{1}{4}$*, which is actually the gain the camera impacts to the system.*

In summary, the camera plays a pivotal role in the conversion of the ball's physical position to a screen representation. This conversion process is pivotal for ensuring effective feedback and operation within our system.

11.3 Plant Transfer function

The plant consists of a beam with a ball. This beam is connected to the motor shaft via a rigid shaft coupling. The angular acceleration of the motor becomes the angular acceleration of the beam. In order to determine the transfer function between the acceleration of the ball and the acceleration of the beam, we shall use the following diagram:

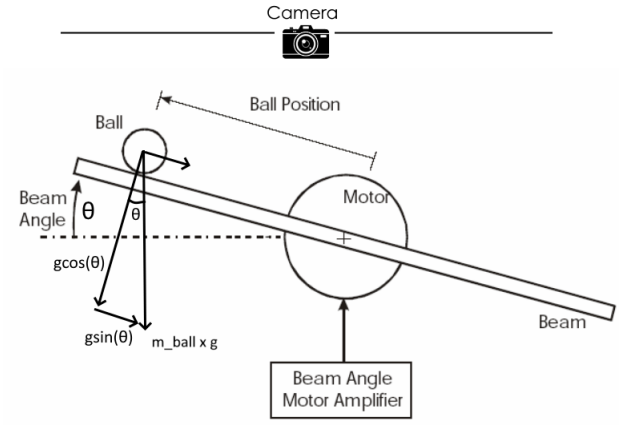

Figure 26: Forces on the ball

The working below is based on the assumption that the angular movement of the beam corresponds directly to the angular movement of the stepper motor's shaft. This assumption is considered valid because the beam is firmly and directly linked to the stepper motor's rotor, and the mechanical connection is highly secure, resulting in minimal losses.

Hence, the equation for the acceleration of the ball is simply

$$a = gsin\theta \tag{11.12}$$

where 'a' is the linear acceleration of the ball and θ is the angular displacement of the beam. The equation can be re-written as

$$a = gsin(\iint_T \alpha dt) \tag{11.13}$$

This equation links the linear acceleration of the ball with the angular acceleration of the rotor α.

Writing the Langrangian equations for this system with x being the position of the ball measured from the desired position, we get

$$0 = (\frac{J}{r_{ball}^2} + m_{ball})\frac{d^2x}{dt^2} + mgsin(\theta) - mx_{ball}\dot{\theta}^2 \tag{11.14}$$

Here the first term represents the inertial force acting on the ball, the second term is the component of the gravitational force acting parallel to the beam, and the third term is the centrifugal force acting on the ball (not shown on free-body diagram). Linearization of this equation about the beam angle, $\theta = 0$, gives us the following linear approximation of the system:

$$(\frac{J}{r_{ball}^2} + m_{ball})\ddot{x_{ball}} = -mg\theta \tag{11.15}$$

For $\theta = 0$, the ball remains stationary, and thus, the centrifugal force on the ball is negligible. Since this linearization pertains to very small values of θ, we approximate $\sin(\theta)$ to be approximately equal to θ.

To obtain the transfer function of the equation above, we perform the Laplace transform to the equation

$$(\frac{J}{r_{ball}^2} + m_{ball})X(s)s^2 = -m_{ball}g\Theta(s) \tag{11.16}$$

Rearranging, we find the transfer function of the rotor angle $\Theta(s)$ to the ball position $X(s)$.

$$P(s) = \frac{X(s)}{\Theta(s)} = -\frac{m_{ball}g}{(\frac{J}{r_{ball}^2} + m_{ball})}\frac{1}{s^2} \quad [\frac{m}{rad}] \tag{11.17}$$

All parameters involved in the above transfer function have their values listed in the Parameter evaluation section.


Fin