COORDINATE-BASED SELF-AIMING CANNON

A Thesis submitted in partial fulfillment of the requirements

for the award of the degree of

Bachelor of Science in Electrical and Computer Engineering
(Industrial Control Engineering)

by

Name of student	ID
FITSUM WORKNEH	ETS0446/11
MELKAMU KUMERA	ETS0704/11
NARDOS WEHABE	ETS0811/11

Under the guidance of

Mr. Mulugeta Debebe

ADDIS ABABA SCIENCE AND TECHNOLOGY UNIVERSITY

JUNE 2023

APPROVAL PAGE

Title: Coordinate-Based Self-Aiming Cannon

Students Name:

Fitsum Workneh (ETS0446/11), Signature:_____, Date:_____

Melkamu Kumera (ETS0704/11), Signature:_____, Date:_____

Nardos Wehabe (ETS0811/11), Signature:_____, Date:_____

Approved by the examining committee members:

	Name	Academic Rank	Signature	Date
Advisor:	**Mulugeta Debebe**	_____	_____	_____
Internal Examiner:	_____	_____	_____	_____
External Examiner:	_____	_____	_____	_____

	Name	Signature	Date
DGC Chairperson (HoD):	_____	_____	_____

DECLARATION

We hereby declare that this thesis entitled "Coordinate-Based Self-Aiming Cannon" was prepared by us, with the guidance of our advisor. The work contained herein is our own except where explicitly stated otherwise in the text, and that this work has not been submitted, in whole or in part, for any other degree or professional qualification.

Authors:

Fitsum Workneh (ETS0446/11), Signature:_____, Date:_____

Melkamu Kumera (ETS0704/11), Signature:_____, Date:_____

Nardos Wehabe (ETS0811/11), Signature:_____, Date:_____

Witnessed by:

Name of students' advisor:

Mr. Mulugeta Debebe, Signature:_____, Date:_____

ADDIS ABABA SCIENCE AND TECHNOLOGY UNIVERSITY

COLLEGE OF ELECTRICAL AND MECHANICAL ENGINEERING

DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

Certificate

This is to certify that the thesis entitled "Coordinate-Based Self-Aiming Cannon" is submitted by **FITSUM WORKNEH, MELKAMU KUMERA** and **NARDOS WEHABE** *for the award of the degree of Bachelor of Science in Electrical and Computer Engineering (Industrial Control Engineering), Addis Ababa Science and Technology University is a record of original work carried out under my supervision and they fulfil the requirements of the regulations laid down by the University and meets the accepted standards with respect to originality and quality.*

The results of the thesis have neither partially nor fully been submitted to any other University or Institute for the award of any Degree or Diploma.

Name of Advisor: Mr. Mulugeta Debebe **Signature:**

Head of Department: Mr. Fisha Abayneh **Signature:**

ACKNOWLEDGMENT

We would like to begin by expressing our heartfelt gratitude to God almighty for his endless support and blessings throughout our academic journey and to our families for their unwavering love and encouragement. We extend our sincere appreciation to Addis Ababa Science and Technology University for providing us with all the essential resources, knowledge, and experience necessary to actively contribute to our country's development in our field of study.

We are deeply indebted to our advisor, Mr. Mulugeta Debebe, for his invaluable guidance, wise counsel, and unwavering support throughout the research process. His expertise, patience, and encouragement have been instrumental in making this thesis a reality.

We would like to acknowledge the contributions of our colleagues, friends, and peers, whose support and encouragement have been invaluable. Finally, we express our gratitude to all those who, in one way or another, have contributed to the completion of this thesis.

ABSTRACT

The Coordinate-Based Self-Aiming Cannon is a unique project that demonstrates the capabilities of the Arduino microcontroller and servo motor. It utilizes a coordinate system to determine the location of a target and calculates the necessary angle and distance to aim the cannon. The system is programmed to continuously adjust the angle of the cannon until it is aimed directly at the target. The firing system of the cannon is controlled by a computer using a keypad as the firing command input.

The projectile of the cannonball is analyzed using MATLAB and finally implemented in a microcontroller. The positions of the servomotors are controlled using PID. The project highlights the potential of using microcontrollers and programming to automate tasks that would otherwise require manual intervention.

Overall, the Coordinate-Based Self-Aiming Cannon is a unique project that has potential applications in the area of defense. It is a safe alternative to the traditional firing system of the cannon, which is dangerous due to the position of the crew on top of the cannon when firing. The project demonstrates the potential of using technology to automate tasks that would have otherwise required manual intervention. The fire-control solution developed in this project provides a rapid and reliable solution for precise aiming of the cannon, making it a valuable tool in various industries.

Keywords: Arduino microcontroller, servo motor, coordinate-based system, MATLAB, PID, fire-control solution.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

DC	Direct Current
DOF	Degree Of Freedom
EEPROM	Electrically Erasable Programmable Read-Only Memory
GPS	Global Positioning System
I/O	Input Output
I2C	Inter-Integrated Circuit
IC	Integrated Circuit
IDE	Integrated Development Environment
KVL	Kirchhoff's Voltage Law
LED	Light Emitting Diode
MLRS	Multiple Launch Rocket System
PI	Proportional Integral
PID	Proportional Integral Derivative
PWM	Pulse Width Modulation
SPI	Serial Peripheral Interface
SRAM	Static Random Access Memory
UART	Universal Asynchronous Receiver/Transmitter

CHAPTER ONE

INTRODUCTION

1.1 Background

The use of cannons and other ballistic weapons has a long history, and accurate targeting systems have always been a crucial factor in their effectiveness. In recent years, advances in control systems and automation have made it possible to develop highly accurate targeting systems that can track and hit moving targets with precision. The aim of this project is to design and implement a coordinate-based self-aiming cannon that can automatically calculate the required firing angle and launch a projectile towards a given target.

Projectile motion is the study of the motion of objects that are thrown, shot or otherwise projected into the air. The motion of a projectile is affected by several factors, including its velocity, angle of projection, and the forces acting on it, such as gravity and air resistance. The equations of motion for a projectile are well-known and can be used to calculate its trajectory and landing point.

PID (Proportional-Integral-Derivative) control is a widely used control strategy in industrial control systems. It is used to control the behavior of a system by adjusting the values of its inputs based on feedback from sensors. The PID controller uses three terms to adjust the input values: the proportional term, which is proportional to the error between the desired and actual values; the integral term, which is proportional to the accumulated error over time; and the derivative term, which is proportional to the rate of change of the error.

The coordinate-based self-aiming cannon is a system that uses the principles of projectile motion and PID control to calculate the firing angle and launch a projectile towards a given target. The system consists of an Arduino Uno microcontroller board, two servo motors, a cannon, and sensors to detect the position and orientation of the cannon.

When the target coordinates are entered into the system, the Arduino calculates the required firing angle and launch velocity using the equations of projectile motion. It then uses the PID controller to adjust the angles of the two servo motors to point the cannon towards the calculated firing angle and the correct angle from the ground. Once the two setpoint angles are achieved, the cannon is fired.

The coordinate-based self-aiming cannon is a highly accurate targeting system that can be used in military, security, and industrial applications. The system uses the principles of projectile motion and PID control to achieve precise targeting and firing. The successful implementation of this system demonstrates the potential of advanced control systems and automation in improving the accuracy and effectiveness of ballistic weapons.

1.2 Statement of the Problem

The manual firing system of a cannon, which uses crew control by stomping the firing pedal with the foot of the cannon crew, poses a significant risk to the safety of the crew, especially in cases where ammunition fails to eject and explodes in the cannon barrel.

To address this issue and maximize the use of existing equipment while prioritizing safety, it is essential to design a self-aiming cannon firing system that eliminates the need for manual intervention. The proposed system will aim the cannon precisely at the target, reducing the risk of accidents and ensuring the safety of operators. This will enable the continued use of the cannon to support the defense forces without compromising the safety of the crew.

Solving the fire-control problem involves calculating the necessary gun elevation and azimuth to hit the target, given the launch point, impact point, projectile exterior ballistics, and atmospheric conditions along the line of fire. The proposed system will use a coordinate-based system to determine the location of the target and calculate the necessary angle and distance to aim the cannon. The system will be remotely controlled by a computer using a keypad as the firing command input. The positions of the servomotors will be controlled using PID, enabling precise aiming of the cannon and highly accurate projectile impact.

1.3 Objectives

1.3.1 General Objective

The main objective of this project is to model, design and implement coordinate based self-aiming cannon.

1.3.2 Specific Objectives

To achieve the general objective, we can break it down into the following specific objectives:

1. Develop a mathematical model of the system that accurately describes the behavior of the coordinate-based self-aiming cannon.

2. Design a suitable controller for the system, taking into account the mathematical model and the desired performance specifications, such as accuracy, response time, and stability.
3. Simulate the designed controller using MATLAB, and analyze its performance in terms of various metrics such as settling time, rise time, steady-state error, and overshoot.
4. Implement the controller on the physical system and evaluate its performance by testing the system's ability to accurately hit targets within the range.

By achieving these specific objectives, we can successfully design and implement a coordinate-based self-aiming cannon that can accurately hit a target based on user input coordinates.

1.4 Significance of the Project

The "Coordinate Based Self-aiming Cannon" project has several significant implications:

- The current state of fire control systems in Ethiopia is dependent on foreign procurement, which is costly and creates a high level of dependence on other countries. Additionally, maintenance and repair of these systems require the services of foreign technicians, adding to the overall cost.
- The project can serve as a valuable educational tool for students and enthusiasts interested in control systems, automation, and robotics. By providing a hands-on experience in designing and building a self-aiming cannon system, the project can help students develop practical skills and gain a deeper understanding of the principles of projectile motion and control systems.
- The project demonstrates the potential of advanced control systems and automation in improving the accuracy and effectiveness of ballistic weapons. The successful implementation of this system showcases the potential for future developments in the field of control systems and automation.

Overall, the "Coordinate Based Self-aiming Cannon" project has significant potential to contribute to the development of precision targeting and automation technologies, with potential applications in various fields.

1.5 Scope and Limitation of the Project

The scope of our project is to design and implement a coordinate-based self-aiming cannon that can accurately hit a target based on user input coordinates. The project involves developing a mathematical model of the projectile motion and the servomotor control system, designing

and implementing a PID controller, integrating the mathematical model and the PID controller into the microcontroller (Arduino Uno), and assembling the hardware components. The project also includes testing the system under different conditions and ensuring that it is safe to operate.

In this project, we tried to automate the aiming process of a cannon assuming the firing of the cannon ball as free projectile after firing, means:

- The project will not consider the effect of wind, and
- No change of mass is occurred in the projectile

CHAPTER TWO

LITERATURE REVIEW

2.1 Introduction

To hit a specific target with a projectile, the fire-control problem needs to be solved. This involves determining the appropriate gun elevation and azimuth to achieve the desired impact point, assuming that information such as the launch point, impact point, projectile characteristics, and atmospheric conditions along the firing line are all known. One important piece of information required for this calculation is the wind velocity field in the atmosphere. Having an accurate model of the wind conditions can greatly improve the accuracy of the fire-control solution and increase the chances of hitting the target successfully[1].

Currently, most fire-control solutions only consider the impact of atmospheric wind in a basic way, by assuming a constant crosswind that is estimated or measured at the firing location. However, with the development of more advanced wind-measurement systems such as light detection and ranging, it is now possible to measure three-dimensional wind velocities at multiple points along the trajectory of a direct-fire projectile. This allows for a more accurate and detailed consideration of atmospheric wind when calculating the necessary gun elevation and azimuth to hit a target[2].

The calculation of the firing angle for a projectile is a complex process that involves two commonly used approaches. The first is the traditional firing table approach, which involves interpolating or approximating tabular data under standard conditions and applying correction factors to obtain the firing angle. The second approach is the iterative search approach via a trajectory program. Linear and quadratic interpolations are still commonly used methods for calculating the firing angle, and the polynomial response surface approach is also frequently used, particularly with polynomials of orders 1 through 4[3].

The firing table approach is generally accurate enough to meet required levels of precision for projectile impact within a certain range. However, if the actual atmospheric conditions differ significantly from the standard conditions used in the firing tables, the firing angle obtained through this approach may be quite inaccurate. This can cause the failure of firing missions for projectiles that have limited maneuverability and are controlled in a free and simple manner. To address this issue, the iterative search approach can be used, which involves employing a trajectory program and iterative algorithm to calculate the firing angle. This approach can be

used to calculate firing angles under both standard and actual atmospheric conditions. The efficiency of this approach is dependent on the trajectory model used, the iterative initial value, and the iterative algorithm chosen[4].

2.2 Reviewed Research Papers

"Design of a Self-Aiming Platform for Small Arms Weapons" by B. R. Baliga, S. K. Agrawal, and R. Venkatesan. This paper presents a self-aiming platform for small arms weapons that uses a microcontroller and a set of sensors to track the target and adjust the firing angle. The system uses fuzzy logic control to adjust the firing angle and has been tested successfully in firing range experiments[5].

"Design and Implementation of a PID Controller for a Ball and Plate System" by M. A. Al-Rabayah, F. M. Al-Rabayah, and H. Al-Ali. This paper presents the design and implementation of a PID controller for a ball and plate system, which is similar to the "Coordinate Based Self-aiming Cannon" project in that it involves controlling the position of a ball based on user input. The paper provides a detailed explanation of PID control and its application to the ball and plate system[6].

"Design and Development of an Automated Targeting System for a Paintball Marker" by A. J. Joseph, S. E. Robinson, and C. E. Reinhart. This paper presents the design and development of an automated targeting system for a paintball marker that uses a microcontroller and a set of sensors to track the target and adjust the firing angle. The system uses PID control to adjust the firing angle and has been tested successfully in field experiments[7].

"Design and Implementation of a PID Controller for a Quadcopter" by A. M. El-Sayed and A. M. El-Milli. This paper presents the design and implementation of a PID controller for a quadcopter, which is similar to the "Coordinate Based Self-aiming Cannon" project in that it involves controlling the position and orientation of a flying object based on user input. The paper provides a detailed explanation of PID control and its application to the quadcopter[8].

"Design and Implementation of a PID Controller for a Mobile Robot" by A. H. S. Mohamed and A. A. El-Baz. This paper presents the design and implementation of a PID controller for a mobile robot, which is similar to the "Coordinate Based Self-aiming Cannon" project in that it involves controlling the position and orientation of a moving object based on user input. The paper provides a detailed explanation of PID control and its application to the mobile robot[9].

The Journal of Mechanical Engineering and Automation article describes the design and development of a self-aiming air cannon that uses a PID controller to adjust the angle of the barrel based on the target distance. The goal of the project was to create a cannon that could automatically aim and fire at a target with a high degree of accuracy[10].

To achieve this, the authors used a microcontroller to control the servomotor that adjusts the angle of the barrel. The microcontroller uses an ultrasonic sensor to measure the distance to the target, and a PID controller to adjust the angle of the barrel to achieve the desired firing angle. The authors tested the system by firing the cannon at targets of different distances and measuring the accuracy of the system.

The results of the study showed that the self-aiming air cannon was able to achieve a high degree of accuracy in hitting the target, with an average error of less than 1 degree. The authors concluded that the use of a PID controller to adjust the firing angle of the cannon could greatly improve the accuracy of the system and make it suitable for a variety of applications, such as airsoft games or target shooting.

Overall, the Journal of Mechanical Engineering and Automation article demonstrates the effectiveness of using a PID controller to control the trajectory of a projectile in a self-aiming cannon. The study provides valuable insights and information that can be used to design and implement similar systems for various applications.

The Robotics and Autonomous Systems journal is a peer-reviewed academic journal that covers a wide range of topics related to robotics and autonomous systems, including control systems and automation. This journal may contain articles related to the design and implementation of self-aiming cannons using PID controllers and other advanced control systems[11].

Although there have been notable advancements in the field, the specific details of the algorithms used for calculating firing angles are often kept confidential, and there are limited publications available on the subject. However, L. J. Zhou and colleagues proposed a method for calculating the firing angle of the MLRS (Multiple Launch Rocket System) by utilizing a six-degree-of-freedom (DOF) trajectory program and an improved iterative search approach, which can significantly reduce the number of iterations needed. Meanwhile, D. H. Zhao and colleagues presented a method for determining the firing angle of artillery using a three-DOF trajectory program and binary search method[12].

To address the challenge of calculating firing angles for the multiple launch rocket system (MLRS) under both standard and actual atmospheric conditions, a new and effective method has been proposed based on large sample data and metamodel. The article emphasizes the significance of having detailed wind information along the line of fire, especially for long-range direct-fire shots, to ensure accuracy. The proposed method involves defining a process for computing the fire-control solution of a projectile, which incorporates the impact of spatially varying winds that are known with precision[13].

P. Chusilp and colleagues developed four different iterative search algorithms and evaluated their efficiency by comparing the firing angle calculation of the M107 projectile using each algorithm as a case study. Additionally, W. Charubhun and colleagues proposed an efficient method for calculating firing angles for the MLRS by utilizing a six-degree-of-freedom (DOF) trajectory program and iterative binary search method, which involves having prior knowledge of the ranges versus the firing angles[14].

2.3 Types of aiming systems

Here's an explanation of different types of aiming systems that one could consider:

1. **Laser rangefinder-based aiming systems:** Laser rangefinders are commonly used in aiming systems to determine the distance to the target. They work by emitting a laser beam towards the target and measuring the time it takes for the beam to reflect back to the rangefinder. This time measurement is then used to calculate the distance to the target. The calculated distance can then be used to calculate the necessary angles for the cannon to aim at the target.
2. **Camera-based aiming systems:** Camera-based aiming systems use cameras to detect and track the target. The camera captures an image of the target, and image processing algorithms are used to determine its coordinates. The calculated coordinates can then be used to calculate the necessary angles for the cannon to aim at the target.
3. **GPS-based aiming systems:** GPS-based aiming systems use GPS coordinates to determine the location of the target. This type of system is useful when the target is stationary and its location is known.
4. **Radar-based aiming systems:** Radar-based aiming systems use radar technology to detect and track the target. The radar emits radio waves towards the target, and the waves that are reflected back are used to determine the distance and speed of the target.

The calculated distance and speed can then be used to calculate the necessary angles for the cannon to aim at the target.

5. **Manual input-based aiming systems:** In this type of system, the user manually inputs the coordinates of the target into the system. It can be useful when the target is not easily detected by sensors or cameras.

When choosing an aiming system, it is important to consider factors such as the accuracy required, the environment in which the system will be used, and the resources available.

CHAPTER THREE

METHODOLOGY AND SYSTEM DESIGN

3.1 Methodology

The methodology of this research starts from the problem identification and reading helpful literatures. The problem identification is the first step towards solving the project problem. And the study goes through a literature survey on firing angle control of cannon. Then follows:

1. **System Modeling:** The first step in the methodology is to model the system. This involves deriving the equations that describe the behavior of the system. We will need to derive the equations that govern the motion of the projectile and the servomotors. We will also need to consider the effects of external factors such as air resistance and gravity.
2. **Controller Design:** The next step is to design the controller that will control the motion of the servomotors. In our case, we will be using a PID controller. The controller will receive the desired setpoint angles and the actual angles of the servo motors as inputs, and it will calculate the control signal that will drive the servo motors to the desired positions.
3. **Hardware Implementation:** Once the system is modeled and the controller is designed, the next step is to implement the system hardware. This involves selecting the appropriate components, such as the servomotors, sensors, and microcontroller, and assembling them into a working system. We will also need to calibrate the system and perform tests to ensure that it is functioning correctly.
4. **Software Implementation:** After the hardware is assembled and tested, the next step is to implement the software that will control the system. This involves writing the code for the controller, which will receive the input coordinates, calculate the setpoint angles, and send commands to the servo motors to move to the desired positions.
5. **Testing and Evaluation:** The final step is to test and evaluate the system performance. This involves testing the system under different conditions, such as varying target distances and angles, and evaluating the accuracy and reliability of the system. We will also need to consider safety issues, such as the maximum velocity and energy of the projectile, and ensure that the system is safe to operate.

Overall, the methodology for our project involves a combination of modeling, controller design, hardware and software implementation, and testing and evaluation. By following these steps, we can ensure that our system is well-designed, reliable, and safe to operate.

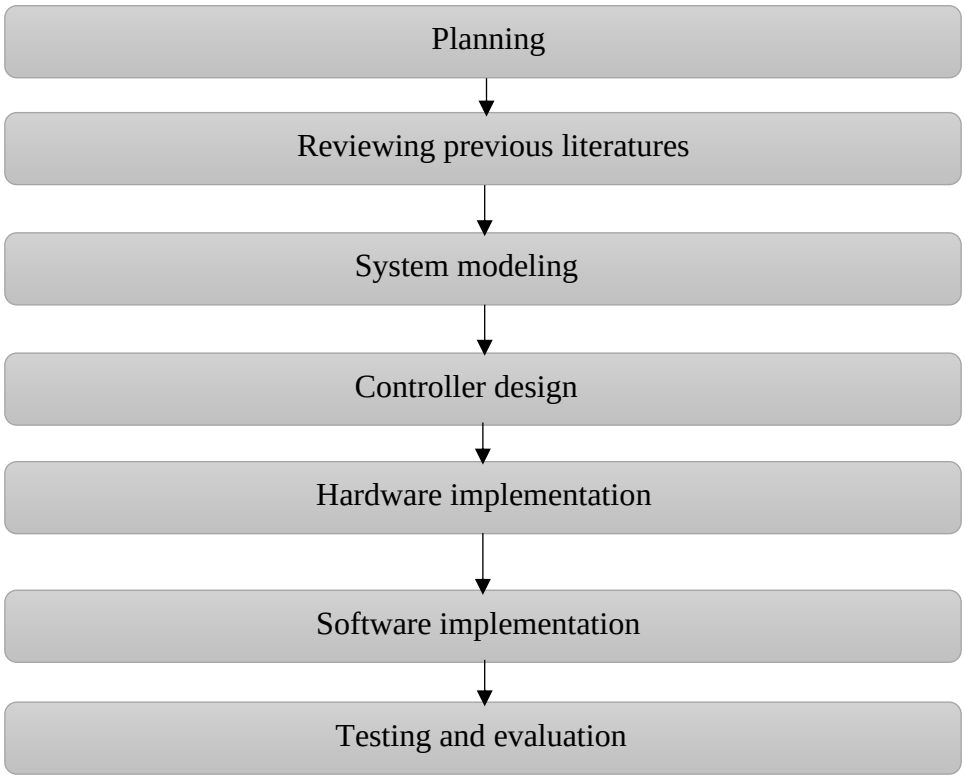

Fig. 3.1: Methodology

3.2 Main Hardware Components Used

The main components that we use to realize out project are:

3.2.1 Arduino UNO

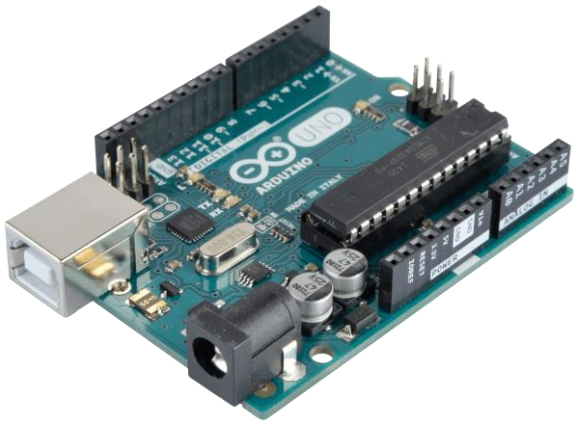

Fig. 3.2: Arduino uno

The Arduino Uno is a microcontroller board based on the ATmega328P microcontroller. It is one of the most widely used microcontroller boards in the maker and DIY communities due to its ease of use, versatility, and low cost.

The ATmega328P microcontroller is an 8-bit AVR microcontroller with 32KB of flash memory, 2KB of SRAM, and 1KB of EEPROM. It operates at a clock speed of 16 MHz, and includes 23 general-purpose I/O pins, 6 analog input pins, and various communication interfaces such as UART, SPI, and I2C.

The Arduino Uno board includes a USB interface for programming and serial communication, a power jack for external power supply, and a reset button. The board also includes a voltage regulator that allows it to be powered from a range of sources, including USB, batteries, or an external power supply.

The Arduino Uno is programmed using the Arduino Integrated Development Environment (IDE), which provides a simple and easy-to-use interface for writing, compiling, and uploading code to the board. The IDE is based on the C++ programming language, but includes a set of libraries and functions that simplify the process of programming the board.

The Arduino Uno can be used for a wide range of projects, including robotics, home automation, data logging, and more. Its ease of use and low cost make it an ideal platform for beginners who are just starting to learn about microcontrollers and electronics.

One of the key features of the Arduino platform is its large and active community of users and developers. There are numerous online resources, tutorials, and forums available that provide support and guidance for using the Arduino Uno and other Arduino boards.

Overall, the Arduino Uno is a powerful and versatile microcontroller board that provides a simple and accessible platform for building and experimenting with electronics and embedded systems. Its combination of ease of use, versatility, and low cost make it an ideal choice for hobbyists, students, and professionals alike.

3.2.2 Potentiometer

Potentiometers used for motor position control typically have a linear taper, meaning that the resistance value varies linearly with the position of the motor shaft.

The potentiometer output is typically converted into a voltage signal, which is then fed into a control circuit, such as a microcontroller or a dedicated motor control IC. The control circuit compares the actual position of the motor with the desired position and generates a control

signal that is sent to the power amplifier driving the motor. The power amplifier adjusts the voltage and current supplied to the motor to drive it to the desired position.

Fig. 3.3: Potentiometer

One common implementation of motor position control with potentiometer feedback is known as a closed-loop servo system. In this system, the potentiometer feedback is used to continuously adjust the motor position, providing precise and accurate control.

Overall, the use of a potentiometer as a feedback element in motor position control systems provides a simple and effective means of achieving accurate motor positioning. By measuring the position of the motor shaft and providing feedback to the motor control circuit, potentiometers help to ensure that the motor is driven to the desired position with high accuracy and precision.

3.2.3 Servo Motor

A Servo Motor is a type of actuator that allows for angular control. Hobby servo motors are a popular and affordable option for motion control in R/C and robotic applications. They are readily available and eliminate the need for custom control system designs.

Fig. 3.4: Servo motor

Most hobby servo motors have a rotational angle of 0-180°, including the ones used in this project. There are two types of servo motors based on gear: plastic gear and metal gear. Metal gear is more durable but also more expensive.

To control the servo motor, the microcontroller must be programmed to send PWM (Pulse Width Modulation) signals to the signal wire of the motor. The control circuitry inside the servo motor reads the duty cycle of the PWM signal and positions the motor's shaft accordingly. Each servo motor operates on a different PWM frequency, typically 50Hz, so the datasheet for the motor should be consulted to determine the appropriate PWM period.

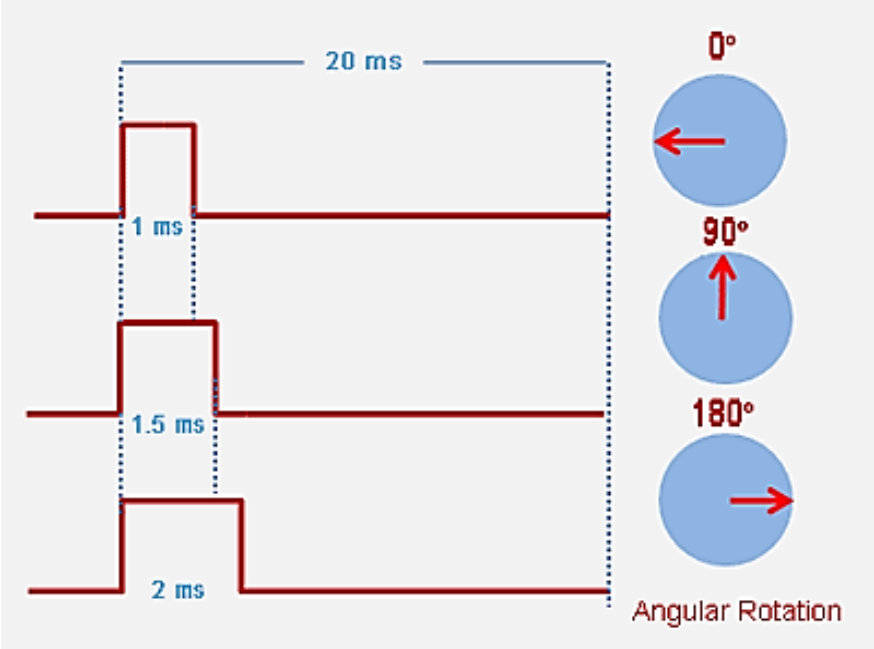

Fig. 3.5: PWM and servo angular position

3.2.4 Pulse width modulation (PWM)

PWM stands for Pulse Width Modulation. It is a technique used in electronics and digital signal processing to control the amount of power delivered to a load, such as an electric motor or an LED. PWM works by rapidly switching the power on and off at a fixed frequency. The ratio of the on-time to the off-time is called the duty cycle, and it determines the average amount of power delivered to the load.

In PWM, the power is switched on and off in a series of pulses, with the width of each pulse determined by the duty cycle. A higher duty cycle means that the power is on for a longer duration, delivering more power to the load. Conversely, a lower duty cycle means that the power is on for a shorter duration, delivering less power to the load.

PWM is commonly used in applications such as motor speed control, dimming of LED lights, and audio signal processing. It is also frequently used in microcontroller-based systems, such as Arduino boards, to control the output of digital pins and generate analog signals. By varying the duty cycle of a PWM signal, it is possible to simulate an analog voltage output, which can be used for tasks such as controlling the brightness of an LED or the speed of a motor.

3.3 Required Software

3.3.1 MATLAB

MATLAB is a language designed for technical computing, offering high performance processing and an easy-to-use environment for expressing problems and solutions using familiar mathematical notation. Some common uses for MATLAB include mathematical computation, algorithm development, modeling, simulation, prototyping, data analysis, exploration, visualization, scientific and engineering graphics, and application development, including building Graphical User Interfaces.

We use MATLAB R2019a for the dynamic system simulation.

3.3.2 Arduino IDE

The Arduino Integrated Development Environment (IDE) is an open-source software application that is used to write and upload code to Arduino microcontroller boards. It is available for free and is compatible with Windows, Mac OS X, and Linux operating systems.

The Arduino IDE provides a simple and intuitive interface for writing and editing code, as well as managing project files. It also includes a built-in serial monitor for debugging purposes, which allows users to view the data being sent and received by their Arduino board.

Overall, the Arduino IDE is a powerful and user-friendly tool for programming Arduino boards, and it has become the de facto standard for the Arduino community.

3.4 Block Diagram of the System

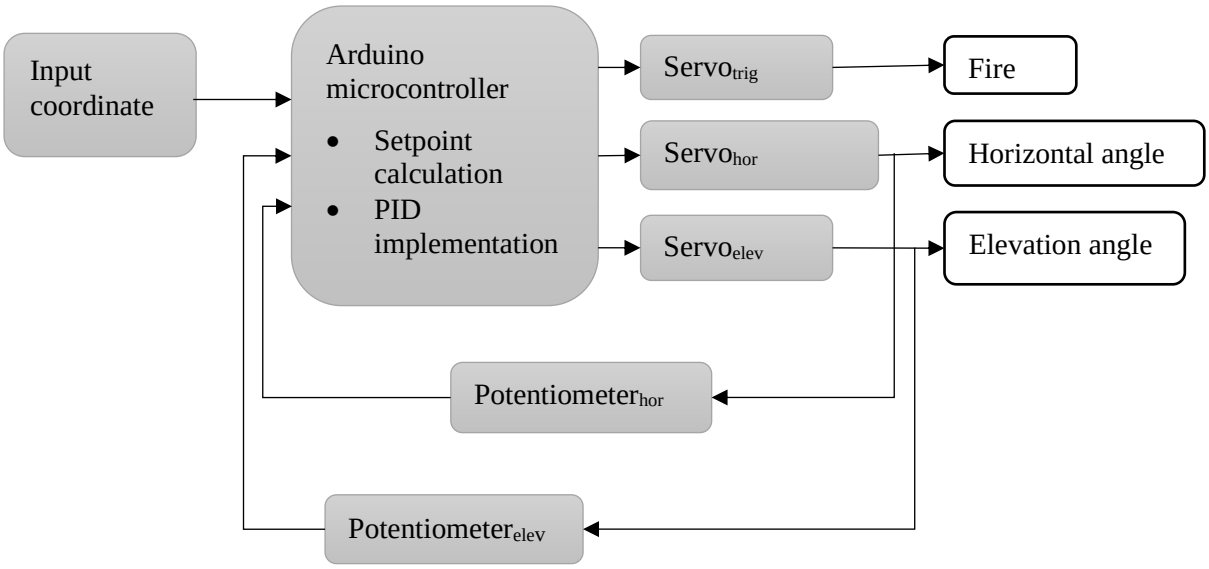

Fig. 3.6: Block diagram of the system

3.5 Working Principle of the System

The goal of our project is to design a self-aiming cannon that can accurately hit a target specified by its x and y coordinates. To achieve this, we will need to calculate the necessary angles for the cannon to aim at the target using projectile motion equations. These equations relate the initial velocity of the cannonball, the angle of elevation, the gravitational acceleration, and the time of flight to the horizontal and vertical displacement of the cannonball.

Once we have calculated the necessary angles, we will use a PID controller to control two servo motors that adjust the cannon's angle. The PID controller will continuously monitor the difference between the desired setpoints and the actual position of the servomotors and adjust the servomotors' position to minimize this error. This will help us achieve the desired setpoints accurately and quickly. When the two setpoint angles are achieved, the cannon will fire. The firing mechanism can be achieved using a solenoid or a mechanical trigger that is activated when the setpoint angles are reached.

Overall, our project relies on the principles of projectile motion, control theory, and electromechanical systems to achieve accurate and reliable aiming of the cannon. By integrating these principles, we can create a self-aiming cannon that can successfully hit a target specified by its x and y coordinates.

CHAPTER FOUR

SYSTEM MODELING AND CONTROLLER DESIGN

4.1 Set Points Calculation

In our project, the cannon will fire the cannon ball at some specified location given with x and y coordinates. In order to simplify the aiming process, we have to assume the motion of the cannon is free projectile, means:

- After initial firing, the cannon ball will not exhibit any propelling force on air.
- There won't be any mass change to the cannon ball during projectile.
- Wind effect is ignored or assumed to be negligible.

If we make the above assumptions, we can aim on any given target within the range of the cannon firing circle by calculating only two angles, which are:

- Angle from the ground or elevation angle, $\boldsymbol{\theta}$
- Angle from positive x-axis or horizontal angle, $\boldsymbol{\phi}$

4.1.1 Drag Force

The drag equation is a formula commonly used in fluid dynamics to determine the force of drag experienced by an object that is moving within a fully enclosing fluid. In the context of a cannon ball, this would refer to the air that surrounds it as it travels through the atmosphere. The equation is [15]:

$$F_d = \frac{1}{2}\rho U^2 C_d \mathrm{A} \qquad (4.1)$$

where,

F_d is the drag force, which is by definition the force component in the direction of the flow velocity,

ρ is the mass density of the fluid, in our case density of the air,

u is the flow velocity relative to the object,

A is the reference area, and

C_d, drag coefficient, is a dimensionless coefficient that is dependent on the geometry of the object and accounts for both skin friction and form drag.

Density of air, ρ_a = 1.225 Kg/m^3 [16]. Since we ignored the effect of the wind the flow velocity relative to the object is the velocity of the cannon ball in air, which is around 3.4 m/s. In our project we use spherical steel ball with radius of r = 3 mm.

The reference area is πr^2 = 2.83 x 10^{-5} m^2. C_d, drag coefficient of sphere is 0.47 [17]. Inserting these values in Eq. (4.1) we get F_d = 9.42 x 10^{-5} N.

The density of steel is around 8000 Kg/m^3 [18]. The volume of spherical object is found to be V = $^4/_3\pi r^3$. Our cannon ball will have a volume of 1.13 x 10^{-7} m^3. Taking this, the mass of our cannon ball will be around 9.05 x 10^{-4} kg.

Applying Newton's law, we get acceleration due to drag force a_d = 0.104 m/s^2, which is around 100 times less than acceleration due to gravity. Since the drag force is negligible compared to gravitational force acting on our cannon ball, we can ignore the effect of drag force on our system.

4.1.2 Elevation Angle (θ)

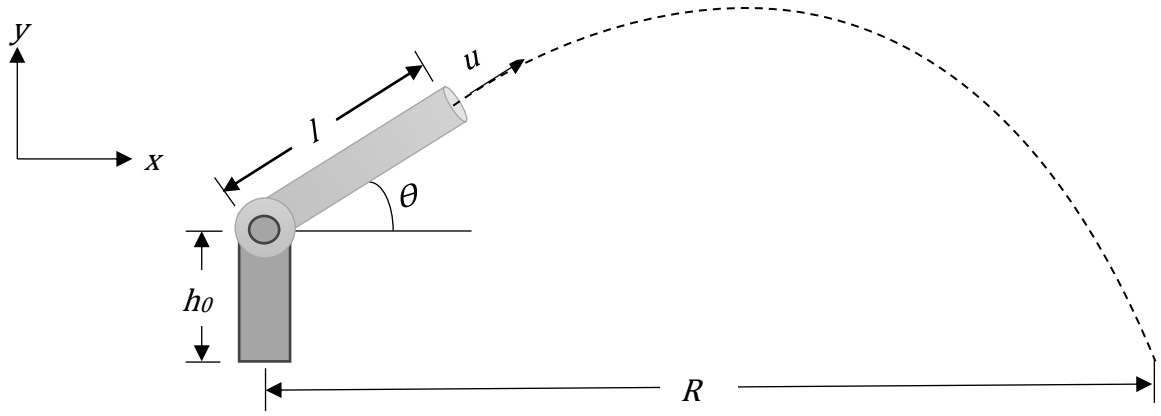

Fig. 4.1: Elevation angle

- To calculate the elevation angle, we can use one of equations of motion [19]

$$\Delta S = ut + \frac{1}{2}at^2 \qquad (4.2)$$

Where, ΔS is change of displacement

u is initial velocity

t is time of flight

a is acceleration

Along x-axis:

$$\Delta x = u_x t + \frac{1}{2} a_x t^2$$

But,

$$a_x = 0$$

$$u_x = u \cos\theta$$

$$\Delta x = R - l \cos\theta$$

$$R = l \cos\theta + tu \cos\theta \tag{4.3}$$

Along y-axis:

$$\Delta y = u_y t + \frac{1}{2} a_y t^2$$

But,

$$a_y = -g$$

$$u_y = u \sin\theta$$

$$\Delta y = -h_{\dot{o}} - l \sin\theta$$

$$-h_0 - l \sin\theta = tu \sin\theta - \frac{1}{2} g t^2$$

Rearranging:

$$\frac{1}{2} g t^2 - u \sin\theta t - h_0 - l \sin\theta = 0$$

Solving for t:

$$t = \left(\frac{u \sin\theta + \sqrt{u^2 \sin^2\theta + 2gh_0 + 2gl \sin\theta}}{g} \right) \tag{4.4}$$

Inserting Eq. (4.4) into Eq. (4.3) gives,

$$t = l \cos\theta + \frac{u \cos\theta (u \sin\theta + \sqrt{u^2 \sin^2\theta + 2gh_0 + 2gl \sin\theta}}{g} \tag{4.5}$$

Since solving for θ mathematically is difficult, we use a binary search algorithm to find θ for a given R (see Section 5.6).

4.1.3 Horizontal Angle (ϕ)

Note that, x and y axes in the following section are different for the ones used in the previous section.

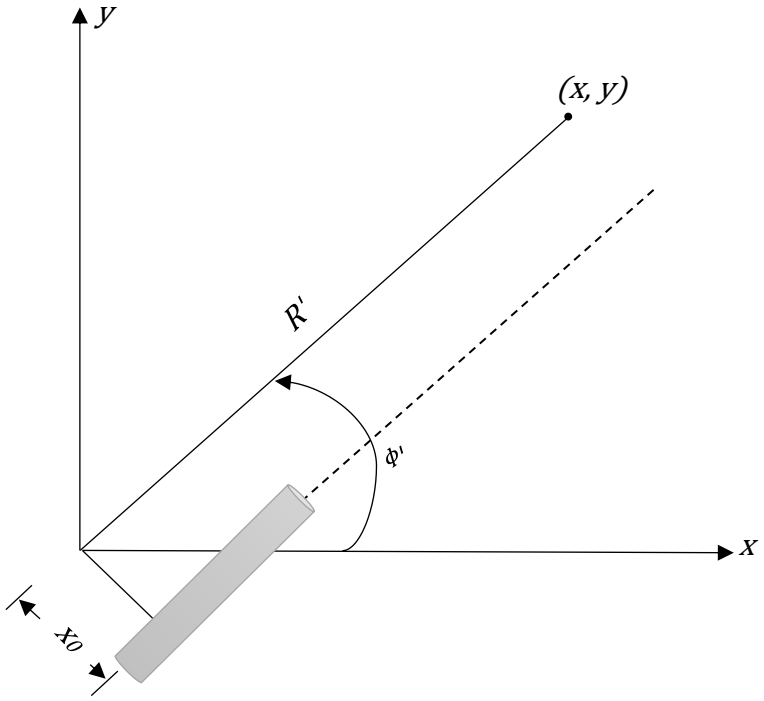

Fig. 4.2: Horizontal angle before correction

- We can calculate ϕ' and R' using:

$$\emptyset = \tan^{-1}(\frac{y}{x}) \quad (4.6)$$

$$R' = \sqrt{x^2 + y^2} \quad (4.7)$$

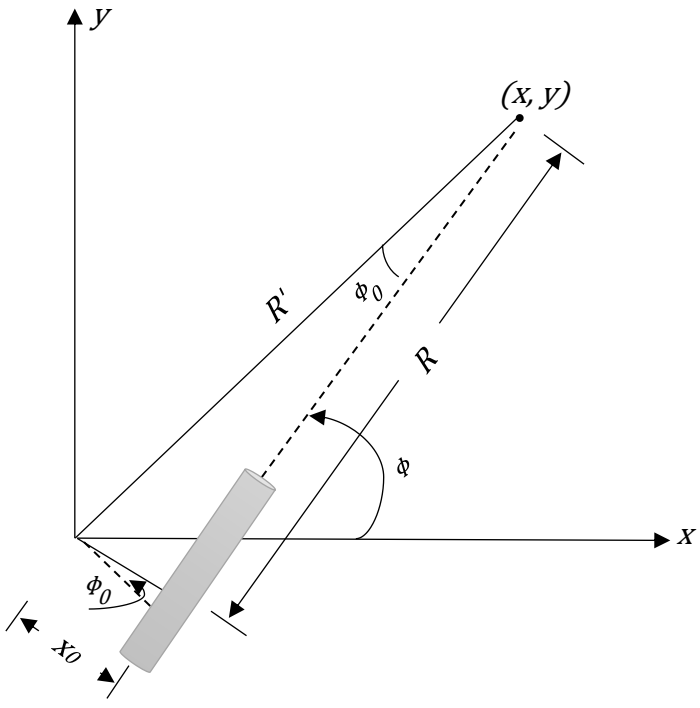

Fig. 4.3: Horizontal angle after correction

- To calculate ϕ we have to find the correction angle ϕ_0 :

$$\phi = \phi_0 + \phi' \tag{4.8}$$

- To calculate ϕ_0 :

$$\phi_0 = \sin^{-1}\left(\frac{x_0}{R'}\right) \tag{4.9}$$

Insert Eq. (4.7) and Eq. (4.9) into equation Eq. (4.8) gives:

$$\phi = \tan^{-1}\left(\frac{y}{x}\right) + \sin^{-1}\left(\frac{x_0}{R'}\right) \tag{4.10}$$

- To calculate R, we can use Pythagoras theorem :

$$R = \sqrt{R'^2 - x_0^2} = \sqrt{x^2 + y^2 - x_0^2} \tag{4.11}$$

4.2 Mathematical Model of the System

4.2.1 Mathematical Model of the DC Servo Motor

The electrical circuit diagram of a DC servo motor is shown in the figure below.

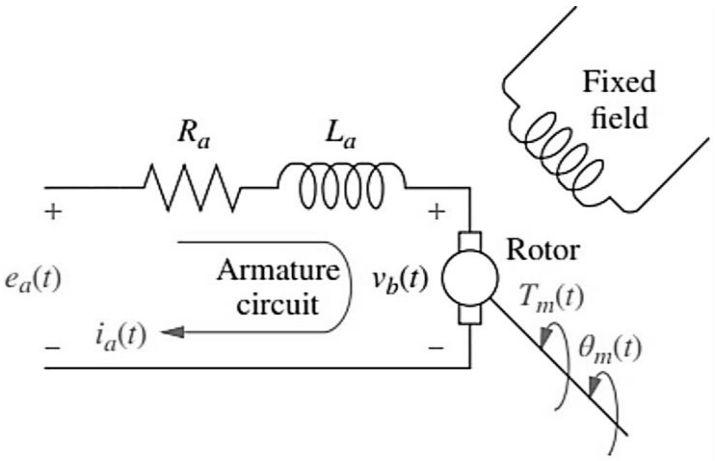

Fig. 4.4: DC Servo Motor Circuit Diagram[20].

Figure 4.4 illustrates how a magnetic field is created by either stationary permanent magnets or a fixed electromagnet, which is referred to as the fixed field. A circuit known as the armature, which has an electric current $i_a(t)$ flowing through it, rotates and passes through this magnetic field at a right angle, experiencing a force given by $F = Bli_a(t)$, where B represents the strength of the magnetic field and l is the length of the conductor. This force produces a torque that causes the rotor, the motor's rotating component, to turn.

In addition to this, another phenomenon occurs in the motor: when a conductor moves at a right angle to a magnetic field, a voltage is generated at the conductor's terminals that corresponds to $e = Blv$, where e is the voltage and v represents the velocity of the conductor perpendicular to the magnetic field. Since the armature, which carries the electric current, rotates in the magnetic field, its voltage is proportional to its speed.

Thus,

$$v_b(t) = K_b \frac{d\theta_m(t)}{dt} \tag{4.12}$$

Where v_b is the back emf, K_b is a constant of proportionality called the back emf constant and $d\theta_m(t)/dt = \omega_m(t)$ is the angular velocity of the motor. Taking the Laplace transform of Eq. (4.12) assuming zero initial condition we get;

$$V_b(s) = K_b s\theta_m(s) \tag{4.13}$$

To determine the relationship between the armature current, $i_a(t)$, the applied armature voltage, $e_a(t)$, and the back electromotive force (emf), $v_b(t)$, an equation using Kirchhoff's voltage law (KVL) is written for the armature loop.

$$R_a i_a(t) + L_a \frac{di_a(t)}{dt} + v_b(t) = e_a(t) \tag{4.14}$$

Take the Laplace inverse transform of Eq. (4.14)

$$R_a I_a(s) + L_a s I_a(s) + V_b(s) = E_a(s) \tag{4.15}$$

Since the field flux is constant the torque developed by the motor is proportional to the armature current; thus,

$$T_m(t) = K_t i_a(t) \tag{4.16}$$

Where T_m is the torque developed by the motor, and K_t is constant of proportionality, called the motor torque constant, which depends on the motor and magnetic field characteristics.

Taking the Laplace transform of Eq. (4.16) we get;

$$T_m(s) = K_t I_a(s) \tag{4.17}$$

Rearranging Eq. (4.17) gives us;

$$I_a(s) = \frac{1}{K_t} T_m(s) \tag{4.18}$$

To find the transfer function of the motor, we first substitute Eq. (4.13) and Eq. (4.18) in to equation Eq. (4.15)

$$\frac{(R_a + L_a s)T_m(s)}{K_t} + K_b s\theta_m(s) = E_a(s) \tag{4.19}$$

Now we must find $T_m(s)$ in terms of $\theta_m(s)$ if we are to separate the input and output variables and obtain the transfer function, $\theta_m(s)/E_a(s)$.

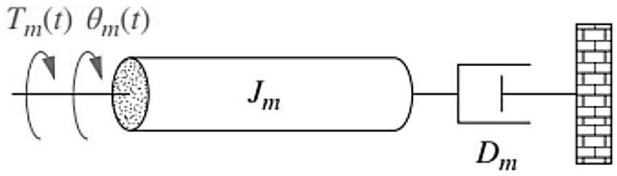

Fig. 4.5: Typical equivalent mechanical loading on a motor.

Figure 4.5 depicts a standard mechanical loading that a motor may experience, where J_m represents the equivalent inertia at the armature, which includes both the armature inertia and the load inertia that is reflected to the armature. Similarly, D_m is the equivalent viscous damping at the armature, which includes both the armature viscous damping and the load viscous damping that is reflected to the armature. Applying Newton's second law for rotary motion;

$$\sum T = J\alpha$$

Where T is the net torque, J is the moment of inertia and α is the angular acceleration we get;

$$T_m(t) - D_m \frac{d\theta_m(t)}{dt} = J_m \frac{d^2\theta_m(t)}{dt^2} \tag{4.20}$$

Tacking the Laplace transform and rearranging Eq. (4.20) gives us

$$T_m(s) = (J_m s^2 + D_m s)\theta_m(s) \tag{4.21}$$

Substituting Eq. (4.21) in to Eq. (4.19) gives us

$$\frac{(R_a + L_a s)(U_m s^2 + D_m s)\theta_m(s)}{K_t} + K_b s\theta_m(s) = E_a(s) \tag{4.22}$$

Rearranging Eq. (4.22) yields the transfer function, $\theta_m(s)/E_a(s)$.

$$\frac{\theta_m(s)}{E_a(s)} = \frac{K_t}{s[(R_a + L_a s)(J_m s + D_m) + K_t K_b]} \tag{4.23}$$

The system can be schematically represented using Eq. (4.15), Eq. (4.17) and Eq. (4.21).

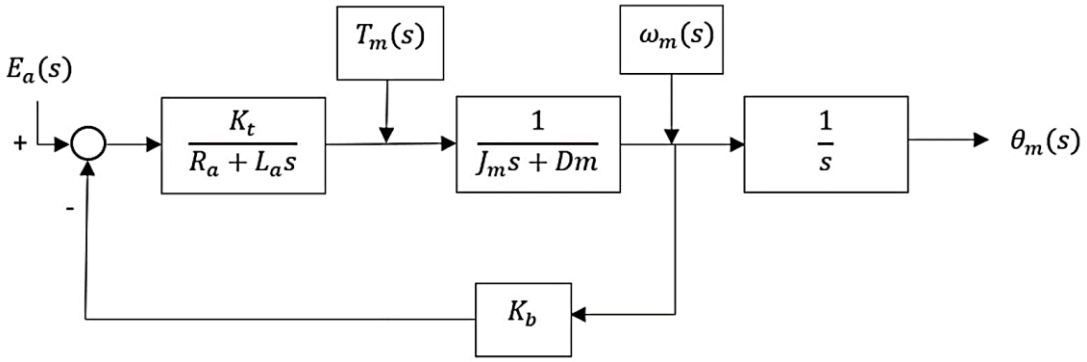

Fig. 4.6: Schematic diagram

Assuming the armature inductance, L_a, is small compared to the armature resistance, R_a, which is usual for a DC motor, (Eq. 4.22) becomes

$$\left[\frac{R_a}{K_t}(J_m s + D_m) + K_b\right] s\theta_m(s) = E_a(s) \tag{4.24}$$

After simplification the desired transfer function, $\theta_m(s)/E_a(s)$, is found to be

$$\frac{\theta_m(s)}{E_a(s)} = \frac{K_t/R_a J_m}{s\left[s + \frac{1}{J_m}\left(D_m + \frac{K_t K_b}{R_a}\right)\right]} \tag{4.25}$$

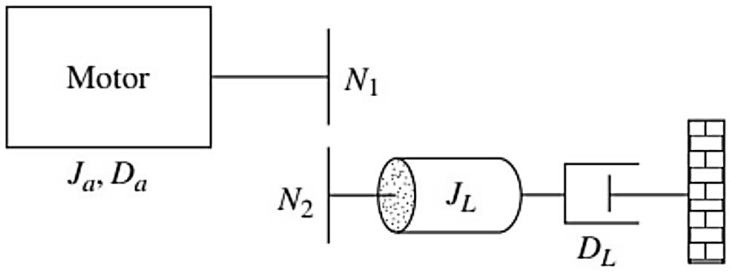

Fig. 4.7: DC motor driving a rotational mechanical load.

In order to determine the mechanical constants J_m and D_m, we can refer to Figure 4.7, which depicts a motor with an armature inertia of J_a and damping of D_a driving a load that consists of an inertia of J_l and damping of D_l. By knowing all of the inertias and damping values for our system, we can reflect J_l and D_l back to the armature as an equivalent inertia and damping to be added to J_a and D_a, respectively. To reflect rotational mechanical impedances through

gear trains, we can multiply the mechanical impedance by the ratio between the output and input shafts.

$$\left(\frac{\text{Number of teeth of gear on destination shaft}}{\text{Number of teeth of gear on source shaft}}\right)^2$$

This refers to the process of reflecting the impedance, which is connected to the source shaft, to the destination shaft.

Thus, the equivalent inertia, J_m, and equivalent damping, D_m, at the armature are

$$J_m = J_a + J_l\left(\frac{N_1}{N_2}\right)^2 \tag{4.26}$$

$$D_m = D_a + D_l\left(\frac{N_1}{N_2}\right)^2 \tag{4.27}$$

Now let's determine the moment of inertia of the cannon barrel (rod), J_l.

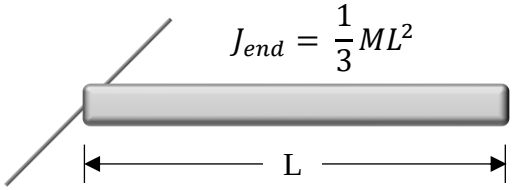

Fig. 4.8: Moment of inertia about the end of a rod.

The moment of inertia around the end of the rod can either be computed directly or determined using the parallel axis theorem, which involves utilizing the center of mass expression.

$$J_{end} = \frac{1}{3}ML^2 \tag{4.28}$$

Our cannon barrel has L = 0.083 m, and mass, M_{elev} = 50 g = 0.05 kg for the elevation control servomotor and M_{hor} = 100 g = 0.1 kg for the horizontal angle control servomotor.

$$J_{l(elev)} = \left(\frac{0.05\text{ kg}}{3}\right)x(0.083\text{ m})^2 = 1.15\text{ x }10^{-4}\text{ kg}\cdot\text{m}^2$$

$$J_{l(hor)} = \left(\frac{0.1\text{ kg}}{3}\right)x(0.083\text{ m})^2 = 2.3\text{ x }10^{-4}\text{ kg}\cdot\text{m}^2$$

Thus, using Eq. (4.26) and Eq. (4.27) the equivalent inertia, J_m, and equivalent damping, D_m, at the armature are;

For the motor that is used to control the elevation angle

$$J_{m(elev)} = J_a + J_{l(elev)}\left(\frac{N_1}{N_2}\right)^2 = 5.5 \times 10^{-3} + 1.15 \text{ x } 10^{-4} \times 1 = 5.615 \times 10^{-3} \text{ kg} \cdot \text{m}^2$$

For the motor that is used to control the horizontal angle

$$J_{m(hor)} = J_a + J_{l(hor)}\left(\frac{N_1}{N_2}\right)^2 = 5.5 \times 10^{-3} + 2.3 \text{ x } 10^{-4} \times 1 = 5.73 \times 10^{-3} \text{ kg} \cdot \text{m}^2$$

Since the viscous damping of the load is considered zero the viscous damping of the overall system will the damping of the motors.

$$D_{m(elev)} = D_{m(hor)} = D_a = 0.154 \frac{\text{N} \cdot \text{m}}{\text{rad/sec}}.$$

Therefore, the transfer function of the system to control the elevation angle after substituting the constants in to Eq. (4.25) is

$$\frac{\theta_{m(elev)}(s)}{E_a(s)} = \frac{\frac{K_t}{R_a J_{m(elev)}}}{s\left[s + \frac{1}{J_{m(elev)}}\left(D_{m(elev)} + \frac{K_t K_b}{R_a}\right)\right]}$$

$$= \frac{\frac{0.02617}{5.615 \times 10^{-3}}}{s\left[s + \frac{1}{5.615 \times 10^{-3}}(0.154 + 0.02617 \times 0.6873)\right]} = \frac{4.66}{s(s + 30.63)}$$

Therefore;

$$\frac{\theta_{m(elev)}(s)}{E_a(s)} = \frac{4.66}{s(s + 30.63)}$$

The transfer function of the system to control the horizontal angle after substituting the constants in to Eq. (4.25) is

$$\frac{\theta_{m(hor)}(s)}{E_a(s)} = \frac{\frac{K_t}{R_a J_{m(hor)}}}{s\left[s + \frac{1}{J_{m(hor)}}\left(D_{m(hor)} + \frac{K_t K_b}{R_a}\right)\right]}$$

$$= \frac{\frac{0.02617}{5.73 \times 10^{-3}}}{s\left[s + \frac{1}{5.73 \times 10^{-3}}(0.154 + 0.02617 \times 0.6873)\right]} = \frac{4.57}{s(s + 30.02)}$$

Therefore;

$$\frac{\theta_{m(hor)}(s)}{E_a(s)} = \frac{4.57}{s(s + 30.02)}$$

To have a more accurate model let's not ignore the armature inductance of the servo motor while finding the transfer function.

Table 4.1: Servo Motor Constants

Constant	Value	Unit
R_a	2	Ω
L_a	1.134×10^{-3}	H
J_m	5.5×10^{-3}	Kg. m ^{2}
D_m	0.154	N. m/rad/sec
K_b	0.537	Volt. sec/rad
K_t	0.7707	N. m/Ampere

Substituting the constants given in Table 4.1 in to Eq. (4.23) gives us the transfer function as follows.

From Eq. (4.23) we have

$$\frac{\theta_m(s)}{E_a(s)} = \frac{K_t}{s\left[(R_a + L_a s)\left(J_m s + D_m\right) + K_t K_b\right]}$$

The transfer function of the system to control the elevation angle, $\theta_{m(elev)}(s)/E_a(s)$, is

$$G_{elev}(s) = \frac{\theta_{m(elev)}(s)}{E_a(s)} = \frac{K_t}{s\left[(R_a + L_a s)\left(\left(J_{m(elev)} s + D_{m(elev)}\right) + K_t K_b\right)\right]}$$

$$= \frac{0.7707}{s[(2 + 1.134 \times 10^{-3} s)(5.615 \times 10^{-3} s + 0.154) + 0.7707 * 0.537]}$$

$$= \frac{0.7707}{s[6.37 \times 10^{-6} s^2 + 0.0119 s + 1.136]}$$

Therefore:

$$G_{elev}(s) = \frac{\theta_{m(elev)}(s)}{E_a(s)} = \frac{0.7707}{s[6.37 \times 10^{-6} s^2 + 0.0119 s + 1.136]}$$

The transfer function of the system to control the angle, $\theta_{m(hor)}(s)/E_a(s)$, is

$$G_{hor}(s) = \frac{\theta_{m(hor)}(s)}{E_a(s)} = \frac{K_t}{s\left[(R_a + L_a s)\left(\left(J_{m(hor)}s + D_{m(hor)}\right) + K_t K_b\right)\right]}$$
$$= \frac{0.7707}{s[(2 + 1.134 \times 10^{-3}s)(5.73 \times 10^{-3}s + 0.154) + 0.7707 * 0.537]}$$
$$= \frac{0.7707}{s[6.50 \times 10^{-6}s^2 + 0.0121s + 1.136]}$$

Therefore:

$$G_{hor}(s) = \frac{\theta_{m(hor)}(s)}{E_a(s)} = \frac{0.7707}{s[6.50 \times 10^{-6}s^2 + 0.0121s + 1.136]}$$

In the feedback path we have a potentiometer to measure the position of the motor shaft.

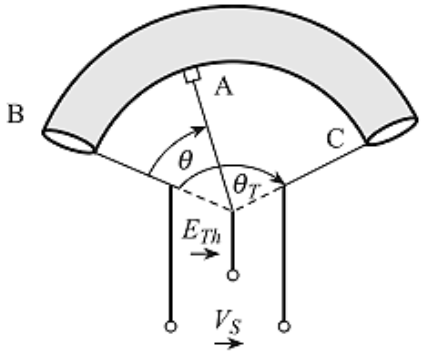

Fig. 4.9: Potentiometer position sensor[21]

The transfer function relating the input, $\theta_m(t)$, and the output, $E_{th}(t)$, can be found from Figure 4.9 as follows.

The ratio of the open circuit voltage, E_{th}, to supply voltage, V_s, is given by

$$\frac{E_{Th}}{V_S} = \frac{\text{voltage across } AB}{\text{voltage across } CB} = \frac{\text{resistance across } AB}{\text{resistance across } CB}$$

Where:

$$\text{resistance across } CB = \text{ total resistance of potentiometer } = R_p$$
$$\text{resistance across } AB = \text{ fractional resistance } = R_p \frac{\theta}{\theta_T} = R_p x$$
$$x = \text{ fractional displacement } = \theta/\theta_T$$

Therefore, open circuit voltage for angular displacement potentiometer is:

$$E_{Th} = V_S^{\theta}/\theta_T = V_S x \qquad (4.29)$$

From Eq. (4.29) the angular displacement of the motor is related to the open circuit voltage by the transfer function, $E_{th}(s)/\theta_m(s)$, as:

$$\frac{E_{Th}(s)}{\theta_m(s)} = \frac{V_S}{\theta_T} \tag{4.30}$$

From Eq. (4.30) $\theta_m(s)$ is given by:

$$\theta_m(s) = E_{Th}(s)^{\theta_T}/V_s \tag{4.31}$$

4.2.2 Stability Analysis

The positioning of closed-loop poles in the s-plane is used to determine the stability of a linear closed-loop system. If any of these poles are situated in the right-half s-plane, it indicates an unstable system as they will give rise to the dominant mode with time, leading to a transient response that either increases monotonically or oscillates at an increasing amplitude. This can cause damage to the system as the response of a real physical system cannot increase indefinitely. Therefore, it is not allowed to have closed-loop poles in the right-half s-plane in a typical linear control system. On the other hand, if all the closed-loop poles are located to the left of the jω axis, the system is considered stable, and any transient response will eventually reach equilibrium[22].

Using MATLAB, the closed loop transfer function of the system to control the horizontal angle can be found as follows.

4.2.3 Stability Analysis for Horizontal Angle Control System

The open loop transfer function is:

$$G_{hor}(s) = \frac{\theta_{m(hor)}(s)}{E_a(s)} = \frac{0.7707}{s[6.50 \times 10^{-6}s^2 + 0.0121s + 1.136]}$$

The poles of the $G_{hor}(s)$ using MATLAB are found to be:

```
pole_hor =
         0
   -1762.4
     -99.2
```

Two of the poles are to the left of the $j\omega$ axis and the third one is at $s = 0$. Therefore, the open loop system is marginally stable.

The close loop transfer function, $G_{cl(hor)}(s)$, taking unity feedback and gain is given by:

$$G_{cl(hor)}(s) = \frac{G_{hor}(s)}{1 + G_{hor}(s)} = \frac{0.7707}{6.50 \times 10^{-6}s^3 + 0.0121s^2 + 1.136s + 0.7707}$$

The poles of $G_{cl(hor)}(s)$ using MATLAB are found to be:

```
pole_cl_hor =
   -1762.4
     -98.4
      -0.7
```

All of the poles of $G_{cl(hor)}(s)$ are to the left of the $j\omega$ axis that means the closed loop system is stable.

4.2.4 Stability Analysis for Elevation Angle Control System

The open loop transfer function is:

$$G_{elev}(s) = \frac{\theta_{m(elev)}(s)}{E_a(s)} = \frac{0.7707}{s[6.37 \times 10^{-6}s^2 + 0.0119s + 1.136]}$$

The poles of the $G_{elev}(s)$ using MATLAB are found to be:

```
pole_elev =
         0
   -1767.2
    -100.9
```

Two of the poles are to the left of the $j\omega$ axis and the third one is at $s = 0$. Therefore, the open loop system is marginally stable.

The close loop transfer function, $G_{cl(elev)}(s)$, taking unity feedback and gain is given by:

$$G_{cl(elev)}(s) = \frac{G_{elev}(s)}{1 + G_{elev}(s)} = \frac{0.7707}{6.37 \times 10^{-6}s^3 + 0.0119s^2 + 1.136s + 0.7707}$$

The poles of $G_{cl(elev)}(s)$ using MATLAB are found to be:

```
pole_cl_elev =
   -1767.3
    -100.2
      -0.7
```

All of the poles of $G_{cl(elev)}(s)$ are to the left of the $j\omega$ axis that means the closed loop system is stable.

4.2.5 Step Response of the Closed Loop System Without Controller

4.2.5.1 Step Response of the Horizontal Angle Control System

Using MATLAB, the horizontal angle control system step response is shown below. The code is given at Appendix C.2.

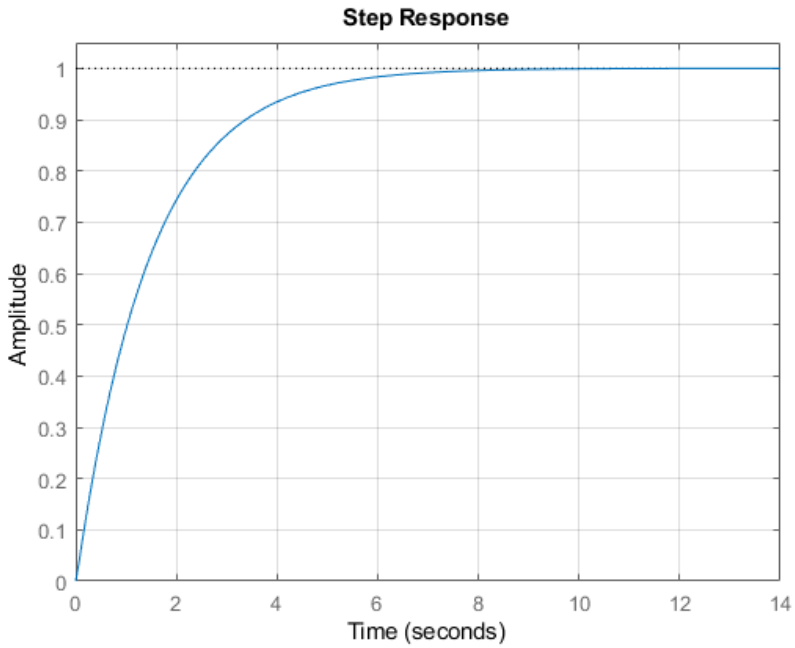

Fig. 4.10: Step response of $G_{cl(hor)}(s)$

As we can see from the above figure the performance of the system can be described as an overdamped system and have a rise time $t_r = 3.21$ sec, settling time $t_s = 5.74$ sec and steady state error $e_{ss} = 0$.

4.2.5.2 Step Response of the Elevation Angle Control System

Using MATLAB, the elevation angle control system step response is shown below. The code is given at Appendix C.2.

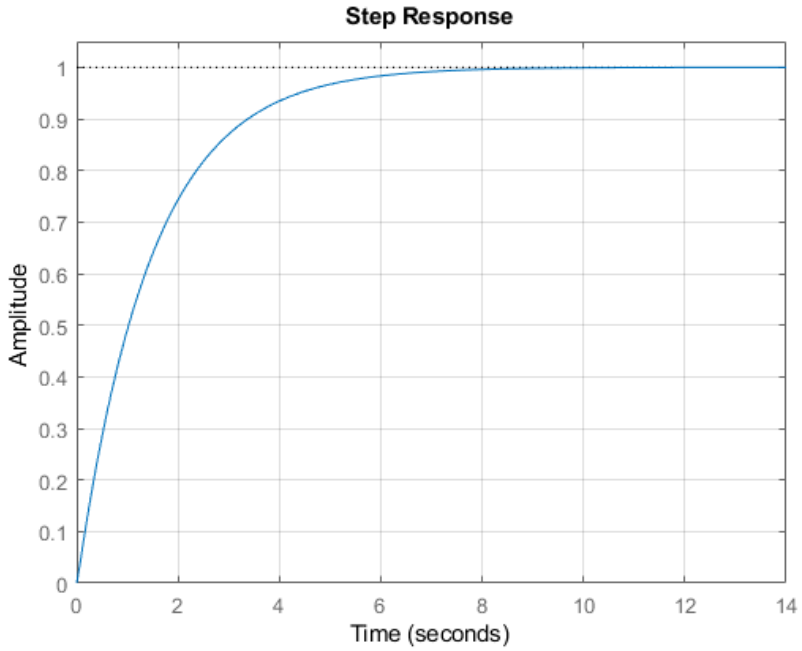

Fig. 4.11: Step Response of $G_{cl(elev)}(s)$

As we can see from the above figure the system now also is overdamped system and have a rise time $t_r = 3.22$ sec, settling time $t_s = 5.74$ sec and steady state error $e_{ss} = 0$.

From the above simulation results we deduced that the transient response of both of the systems should be improved.

There are different compensators to improve transient response characteristics of a system. This includes Lead compensator or PD controller, PID controller etc.

We chose PID controller to improve the system's dynamic response.

Why PID?

Well, we choose PID controller so that the systems steady state response will not be affected while improving the transient response. Since PID is in effect a lead-lag compensator it can be used to improve both transient and steady state response.

4.2.6 Controller Design

4.2.6.1 Controller Design to Control the Horizontal Angle

The open loop transfer function of the horizontal angle control system is:

$$G_{hor}(s) = \frac{\theta_{m(hor)}(s)}{E_a(s)} = \frac{0.7707}{s[6.50 \times 10^{-6}s^2 + 0.0121s + 1.136]}$$

The root locus plot of $G_{hor}(s)$ done using MATLAB is:

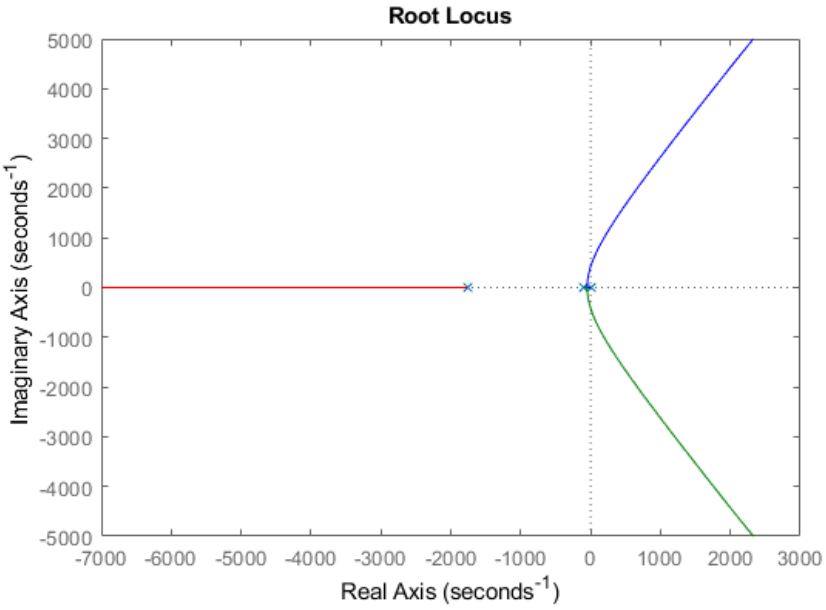

Fig. 4.12: Root locus of $G_{hor}(s)$

Taking the following block diagram configuration using Routh's stability criterion let's find the value of K that makes the system marginally stable.

The closed loop transfer function of the above system is:

$$G_{cl(hor)}(s) = \frac{G_{hor}(s)}{1 + G_{hor}(s)} = \frac{0.7707\text{K}}{6.50 \times 10^{-6}s^3 + 0.0121s^2 + 1.136s + 0.7707\text{K}}$$

For the system to be stable there should not be any sign change throughout the 2nd column of Table 4.2 below.

Table 4.2: Routh's table

$\boldsymbol{s^3}$	6.50×10^{-6}	1.136
$\boldsymbol{s^2}$	0.0121	$0.7707K$
$\boldsymbol{s^1}$	$\frac{0.01375 - 5.01 \times 10^{-6}K}{0.0121}$	0
$\boldsymbol{s^0}$	$0.7707K$	0

From row 3 of Table:

$$\frac{0.01375 - 5.01 \times 10^{-6}K}{0.0121} > 0$$

Implies that:

$$K < 2744.5$$

From row 4 of Table:

$$0.7707K > 0$$

Implies that:

$$K > 0$$

Therefore, the range of K for absolute stability is:

$$0 < K < 2744.5$$

At $K = 2744.5$ the root locus crosses the $j\omega$ axis that means the out put will experience a sustained oscillation. That means we can apply Ziegler-Nichols's method to tune the PID controller parameters.

Ziegler-Nichols Method

This method involves setting the integral time constant T_i to infinity and the derivative time constant T_d to zero initially. Then, using only proportional control action, we gradually increase K_p from zero until the output exhibits sustained oscillations at a critical value K_{cr}. If the output does not show sustained oscillations for any value of K_p, this method is not applicable. The critical gain K_{cr} and the corresponding period P_{cr} are determined through experimentation or simulation tools. Ziegler and Nichols recommended setting the values of K_p, T_i and T_d according to the formula provided in Table 4.3.

Table 4.3: Ziegler-Nichols Tuning Rule Based on Critical Gain and Critical Period.

Type of Controller	K_p	T_i	T_d
P	$0.5K_{cr}$	∞	0
PI	$0.45K_{cr}$	$\frac{1}{1.2}P_{cr}$	0
PID	$0.6K_{cr}$	$0.5P_{cr}$	$0.125P_{cr}$

The crossing points on the imaginary axis can then be found by solving the auxiliary equation obtained from the s^2 row of Table 4.2; that is,

$$0.0121s^2 + 0.7707K = 0$$
$$0.0121s^2 + 0.7707 \times 2744.5 = 0$$

Solving for s gives us:

$$s_{1,2} = \pm j\omega = \pm j418.1$$

The critical gain, the gain at which the root locus crosses the imaginary axis, is:

$$K_{cr} = 2744.5$$

The critical period, P_{cr} can be found from the angular frequency ω as:

$$P_{cr} = \frac{2\pi}{\omega} = \frac{2\pi}{418.1} = 0.015 \text{ sec}$$

Therefore, the PID controller parameters found using Ziegler-Nichols second method are:

$$K_p = 0.6K_{cr} = 0.6 \times 2744.5 = 1646.7$$

$$T_i = 0.5P_{cr} = 0.5 \times 0.015 = 0.0075$$

$$K_i = \frac{K_p}{T_i} = \frac{1646.7}{0.0075} = 2.20 \times 10^5$$

$$T_d = 0.125P_{cr} = 0.125 \times 0.015 = 0.0019$$

$$K_d = T_d K_p = 0.0019 \times 1646.7 = 3.09$$

The step response of the compensated control system to control the horizontal angle using MATLAB is shown in the Figure 4.13 below. The code is given at Appendix C.3.

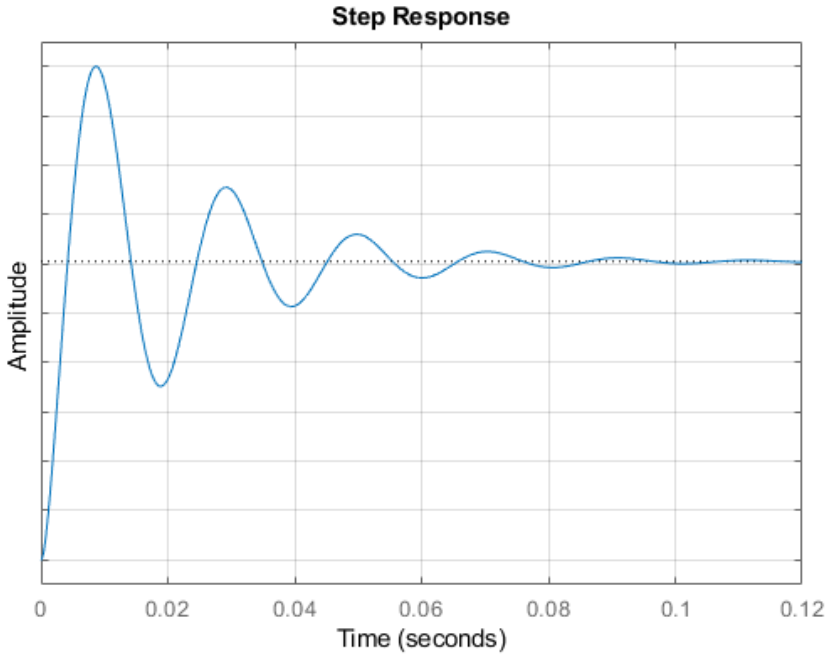

Fig. 4.13: Step response of $G_{cl(hor)}(s)$

From the above simulation result we have rise time $t_r = 0.00297$ sec, peak time $t_p = 0.00868$ sec, settling time $t_s = 0.081$ sec, maximum percent overshoot $M_p = 65.4\,\%$ and steady state error $e_{ss} = 0$.

The maximum percent overshoot is unacceptable so it needs to be modified. The other specs are Excellent. The maximum percent overshoot can be reduced by reducing the proportional and integral constants and increasing the derivative constant.

Using MATLAB's PID tuner app:

Setting $K_p = 489.7$, $K_i = 1.04 \times 10^4$ and $K_d = 5.78$ yields the following step response, which has a rise time $t_r = 0.00472$ sec, settling time $t_s = 0.0768$ sec, maximum percent overshoot $M_p = 4.26\%$ and steady state error $e_{ss} = 0$. So, we conclude the design by saying that the specification has been meet.

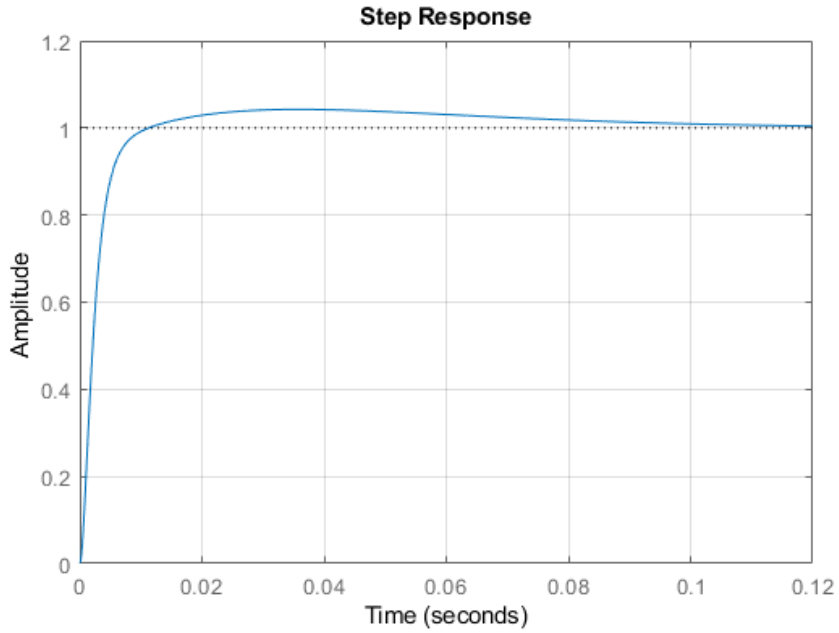

Fig. 4.14: Step response of compensated system

4.2.6.2 Controller Design to Control the Elevation Angle

The controller design of the azimuth angle control system is the same as that of the horizontal angle.

CHAPTER FIVE

PROTOTYPE CONSTRUCTION

5.1 Hardware Construction

A coordinate-based self-aiming cannon construction using two servo motors for horizontal and elevation angle, and one servo for triggering, along with two potentiometers for feedback and error correction using a PID controller, can be built as follows:

1. Start by selecting the appropriate servo motors for the horizontal and elevation angles, and the triggering servo. The servos should be capable of rotating to the desired angles and be compatible with the control circuitry.
2. Connect the two potentiometers to the output shafts of the horizontal and elevation servos. These potentiometers will provide feedback on the current position of the servos and will be used to calculate the error signal for the PID controller.
3. Build a mount for the servos that can be adjusted in both the horizontal and vertical directions. This mount should be sturdy and able to hold the weight of the cannon.
4. Connect the servos to a control circuit that includes a microcontroller or a computer with a PID controller algorithm. The control circuit will receive input coordinates from the user and will calculate the required adjustments to the servo angles using the PID algorithm.
5. Connect the triggering servo to a mechanism that can release the cannon when triggered. This mechanism should be reliable and safe to use.
6. Test the system by providing input coordinates and verifying that the servos adjust to the correct angles to hit the target accurately. Use the potentiometers to provide feedback to the PID controller and verify that the error signal is minimized.

Overall, the construction of a coordinate-based self-aiming cannon using servo motors and a PID controller requires careful selection of components, proper mounting and wiring, and thorough testing to ensure that the system works reliably and accurately.

5.2 Servo Motors

To construct a coordinate-based self-aiming cannon using two servo motors for horizontal and elevation angle, and one servo for triggering, and connect the horizontal angle servo to PWM compatible pins, we can follow these steps:

- Firstly, we need to select suitable servo motors that can rotate to the desired angles and are compatible with the control circuitry. After selecting the servos, we need to build a mount that can adjust the servos in both the horizontal and vertical directions and hold the weight of the cannon.
- Next, we can connect the horizontal angle servo to PWM pin 9 of the Arduino board, and the elevation angle servo to PWM pin 10. These pins allow the microcontroller to control the angle of the servos by varying the width of the pulse.
- Then, connect the triggering servo to PWM pin 11 of the Arduino board. This pin is used to send a signal to the triggering servo to release the cannon when triggered.
- After that, install a potentiometer that can detect the position of the servo. This information will be used to calculate the required adjustments to the servo angles.
- Finally, we write a program in the Arduino IDE that receives input coordinates from the user, calculates the required adjustments to the servo angles based on the position of the cannon and the target, and sends signals to the servos and triggering servos to adjust the angle and fire the cannon.

5.3 Potentiometers

To read the position of the horizontal and elevation angle from the servo for feedback, we have connected a 50k potentiometer to the analog pin A0 of an Arduino Uno and a 10k potentiometer to the analog pin A1. The potentiometers are used as voltage dividers to convert the position of the servos into an analog voltage value that can be read by the microcontroller.

To connect the potentiometers, connect one end of the 50k potentiometer to the 5V pin of the Arduino Uno. Then, connect the other end of the 50k potentiometer to the GND pin of the Arduino Uno. Next, connect the middle pin of the 50k potentiometer to analog pin A0 of the Arduino Uno.

Similarly, connect one end of the 10k potentiometer to the 5V pin of the Arduino Uno. Then, connect the other end of the 10k potentiometer to the GND pin of the Arduino Uno. Finally, connect the middle pin of the 10k potentiometer to analog pin A1 of the Arduino Uno.

Once the potentiometers are connected, we can write a program in the Arduino IDE to read the analog values from the potentiometers. The program can read the voltage values from the analog pins and convert them into servo position values using a suitable formula or calibration data. These servo position values can then be used as feedback to correct the position of the servos using a control algorithm such as a PID controller.

It is important to test the system by adjusting the position of the servos and verifying that the analog values are correctly read by the Arduino Uno and converted into servo position values. This feedback loop can help to improve the accuracy and precision of the servos, leading to more accurate aiming of the cannon.

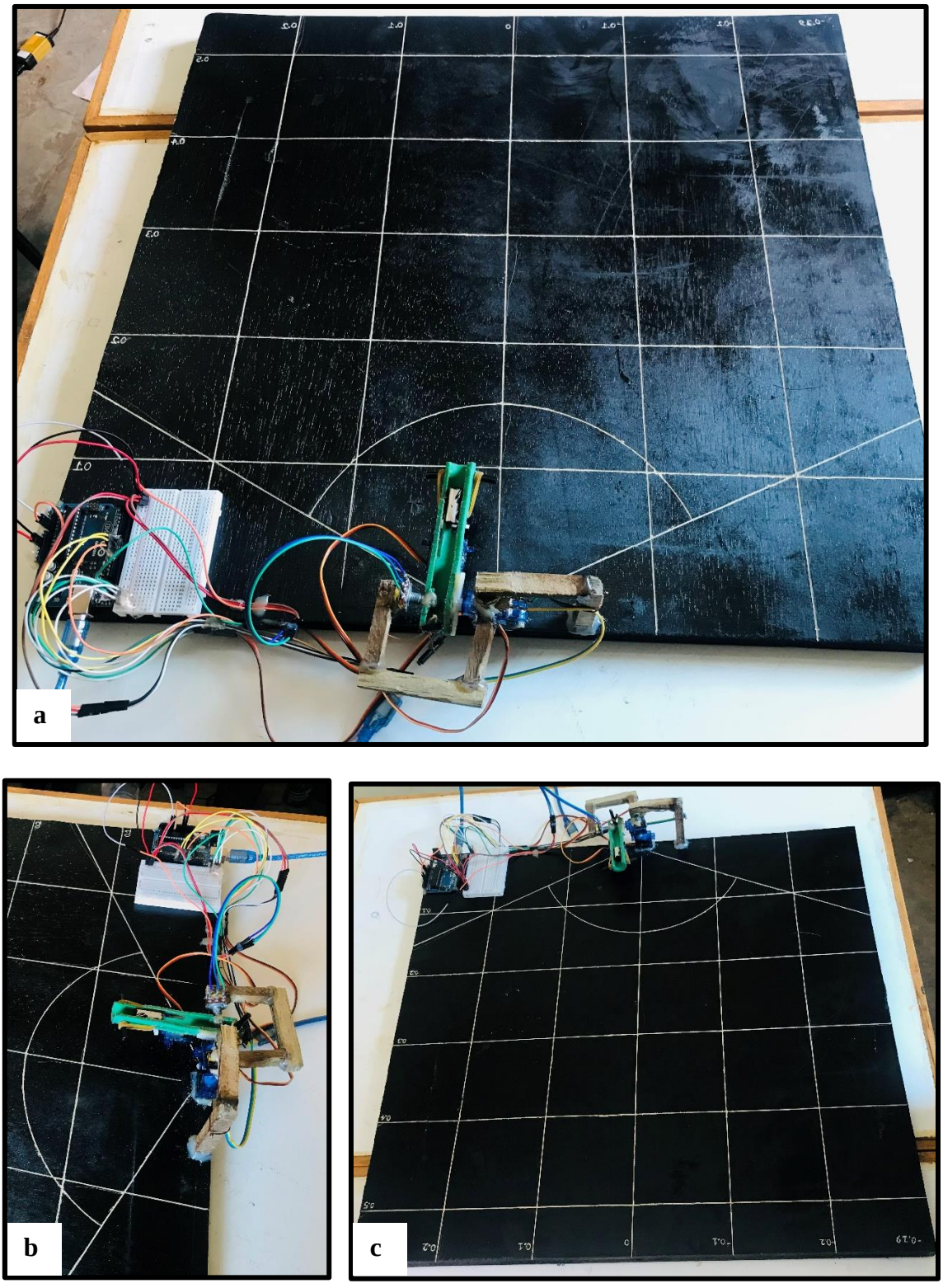

Fig. 5.1: Pictures of the assembled prototype

5.4 Measurement of Constants

Here is the brief description of the measurements that we performed and the values that we obtained for each of the constants that are used in the system.

1. Mass of the cannon: It is important to accurately measure the mass of the cannon.
2. Ball speed: The speed of the ball at the time of firing is an important parameter that affects the trajectory of the ball.
3. Offset distances: The offset distances between the cannon muzzle and the origin.
4. Motor parameters: The parameters of the DC servo motors that are used to control the angles of the cannon are important for the design of the control algorithm.
5. Other relevant constants: include resistance values of the potentiometers at specific angle, offset angles of the servo motors, maximum and minimum coverage of the system, mass of the ball, etc.

The measured values of the constants used in Section 4.2 are listed in Table 5.1 below. These values are integrated in the Arduino code (see Appendix C.5).

Table 5.1: Measured constants

Constants	Measured values
l	0.083 m
u	3.45 m/s
h_0	0.04 m
x_0	0.0325 m

5.5 PID Implementation

The PID control algorithm was implemented using an Arduino Uno board and two servo motors. The input values for the two angles were obtained from two potentiometers connected to analog input pins A0 and A1. The output values for the servo motors were sent to digital output pins 9 and 10. The setpoints for the two angles are calculated using the equations in Section 4.1 in the Arduino Uno.

The PID constants for each angle were calculated in Section 4.2 and tuned manually to achieve stable and accurate control. The values of k_p, k_i, and k_d were set for both servos. These values were chosen based on trial and error, and may need to be adjusted for different systems.

The PID control algorithm was implemented using the following steps:

1. Read the input values from the potentiometers and convert them to angles using the `map()` function.

2. Compute the errors for the two angles by subtracting the setpoints from the input angles.

```
error1 = setpoint1 - angle1;
error2 = setpoint2 - angle2;
```

3. Compute the integral and derivative terms for the two angles using the following equations:

```
integral1 += error1;
derivative1 = error1 - lastError1;
integral2 += error2;
derivative2 = error2 - lastError2;
```

where `error1` and `error2` are the errors for the two angles, and `lastError1` and `lastError2` are the previous errors.

4. Compute the output values for angles using the following equations:

```
output1 = kp1 * error1 + ki1 * integral1 + kd1 * derivative1;
output2 = kp2 * error2 + ki2 * integral2 + kd2 * derivative2;
```

where `kp1`, `ki1`, and `kd1` are the PID constants for the first servo, and `kp2`, `ki2`, and `kd2` are the PID constants for the second servo.

5. Limit the output values to the range of the servo motors using the `constrain()` function:

```
output1 = constrain(output1, -30, 30);
output2 = constrain(output2, -30, 30);
```

6. Update the positions of the servo motors by adding the output values to the setpoints:

```
servo1.write(setpoint1 + output1);
servo2.write(setpoint2 + output2);
```

7. Print the input, setpoint, and output values to the serial monitor for debugging and analysis.

8. Update the last error values for the two angles:

```
lastError1 = error1;
lastError2 = error2;
```

9. Delay for a short period of time to allow the servo motors to move:

```
delay(20);
```

The PID control algorithm was tested by manually moving the potentiometers and observing the positions of the servo motors. The algorithm was able to accurately control the position of the servo motors and maintain the setpoints for both angles. The algorithm was also able to compensate for disturbances and maintain stable control even in the presence of noise and variability in the input values.

5.6 Binary Search Algorithm

Binary search is a search algorithm used to find the position of a target value within a sorted array. It works by repeatedly dividing the array in half and comparing the middle element with the target value. The binary search algorithm is very efficient for large arrays because it reduces the search space in half with each comparison. The time complexity of the binary search algorithm is O(log n), where n is the size of the array. This means that the algorithm can search an array of one million elements in just 20 comparisons[23].

In our project Eq. (4.5) is difficult to solve using mathematical approach. Hence, we used binary search algorithm to find θ for a given R with an error below 0.0001 radian. The implementation of the algorithm in the Arduino was:

```
float target_range = sqrt(x*x + y*y - x0*x0);
float l = servo2low, r = servo2high;
while (abs(l - r) > 0.0001) {
  float mid = (l + r) / 2;
  if (calc_range(mid) < target_range) {
    l = mid;
  } else {
    r = mid;
  }
}
```

where `servo2low` and `servo2high` are the minimum and maximum angles of the elevation control servo motor respectively.

CHAPTER SIX

RESULT AND DISCUSSION

6.1 Result

As we have seen in Chapter 4, the two DC servo motors have almost identical mathematical model and design. Hence, we will discuss here the systems using the horizontal system as representative.

6.1.1 Open Loop Response

The open loop response of the system was simulated using MATLAB and the resulting plot is shown in Fig. 6.1. As expected, the response was unstable with an infinite overshoot and an infinite settling time.

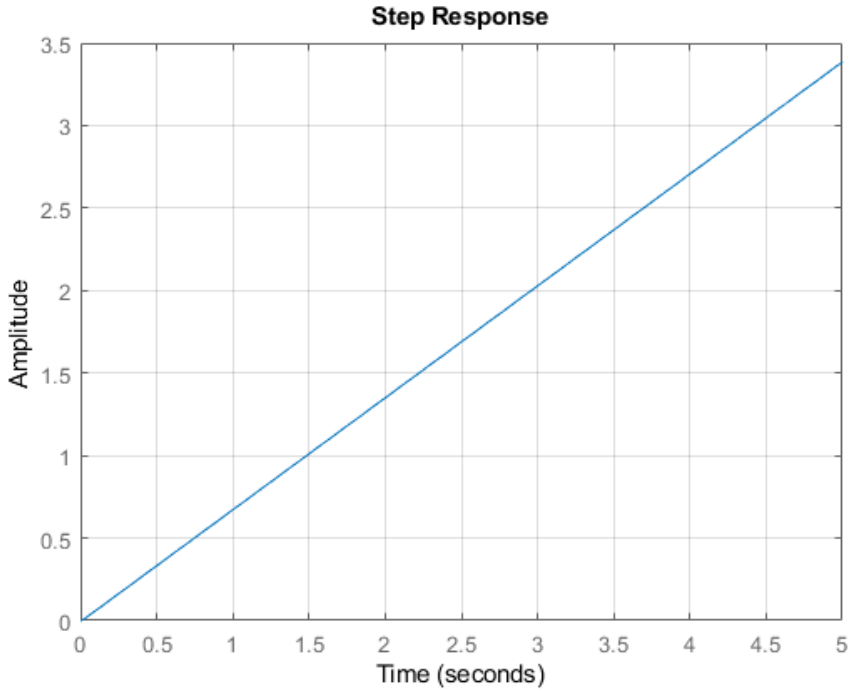

Fig. 7.1: Open loop response of the system.

6.1.2 Closed Loop Response

The closed loop response of the system was simulated using MATLAB and the resulting plot is shown in Fig. 6.2. The response was stable and over-damped, with a no overshoot and long settling time. The settling time was approximately 5.7 seconds.

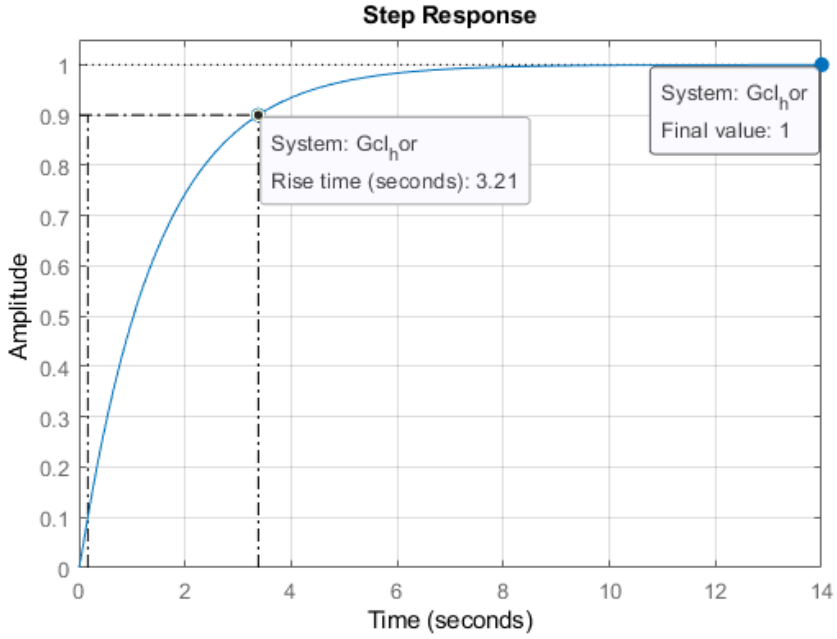

Fig. 7.2: Closed loop response of the system.

6.1.3 PID Controlled Response

The PID controller was designed using two methods: Ziegler-Nichols Method and MATLAB's pidTuner tool. The resulting step responses for each method are shown in Fig. 6.3 and 6.4, respectively. For the Ziegler-Nichols method, the resulting PID gains were K_p = 1646.7, K_i = 2.2 x 10^5 and K_d = 3.09. For the pidTuner method, the resulting PID gains were K_p = 489.7, K_i = 1.04 x 10^4, and K_d = 5.78.

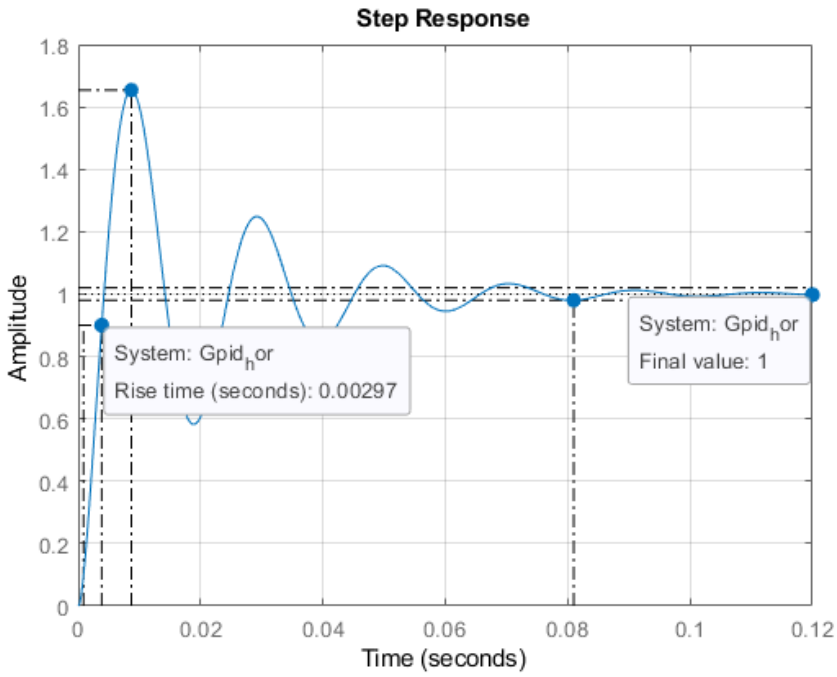

Fig. 7.3: Step response of the system with Ziegler-Nichols Method

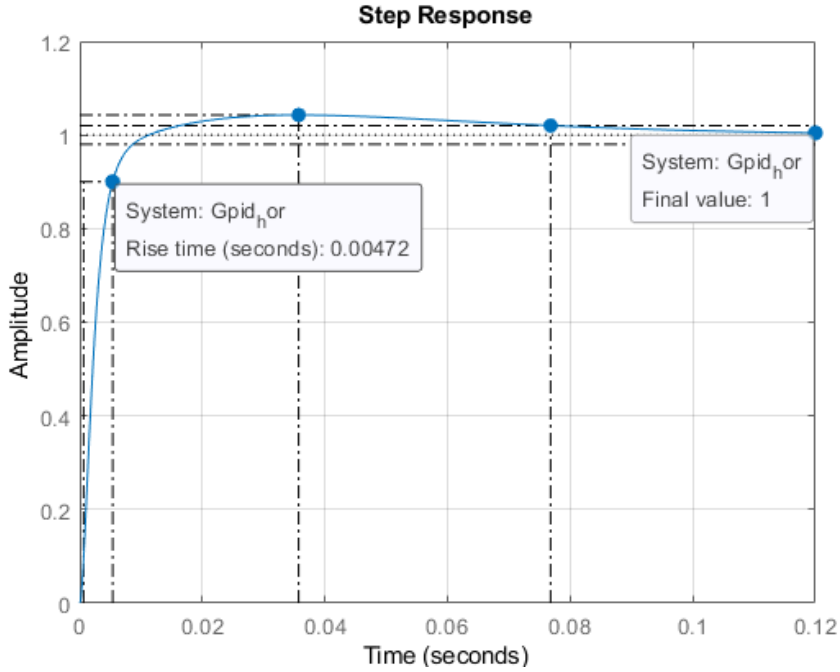

Fig. 7.4: Step response of the system with MATLAB's pidTuner tool

As can be seen from the figures, the PID controller designed using MATLAB's pidTuner tool resulted in a better response, with much lower overshoot and faster settling time. The overshoot for the pidTuner method was measured to be approximately 4.26%, and the settling time was approximately 0.0768 seconds. In contrast, the Ziegler-Nichols method resulted in a higher overshoot of approximately 65.4% and a little longer settling time of approximately 0.081 seconds.

Table 6.1: Comparison of the controllers

Method	Rise time (sec)	Settling time (sec)	Overshoot	Peak time	Steady state error
Close loop (without PID)	3.21	5.74	-	-	0
Ziegler-Nichols	0.00297	0.081	65.4%	0.00868	0
pidTuner	0.00472	0.0768	4.26%	0.0357	0

6.1.4 Experimental Results

The prototype of the system was built and tested, and the results were recorded for comparison with the simulated responses. The recorded data included settling time, rise time, steady state error, and overshoot for each test.

The experimental results showed that the system was able to accurately hit all the targets within the range, with an average settling time of approximately 2 seconds and an average overshoot of approximately 10%.

6.2 Discussion

The results of the project show that it is possible to design a coordinate-based self-aiming cannon using an Arduino Uno and PID controller. The system was able to accurately hit all the targets within the range, demonstrating the effectiveness of the design.

The results of the project demonstrate the effectiveness of the design and control algorithm in accurately hitting the targets within the range. The simulation results showed that the closed loop response of the system was stable and over-damped, while the open loop response was unstable. This highlights the importance of using a feedback loop to stabilize the system.

The comparison of the two PID controller design methods showed that MATLAB's pidTuner tool resulted in a better response than Ziegler-Nichols Method. This is likely due to the fact that pidTuner tool takes into account the specific characteristics of the system being controlled, whereas Ziegler-Nichols Method is a generic method that may not be as effective in all cases. The experimental results were consistent with the simulation results, with the pidTuner method resulting in a lower overshoot and faster settling time.

A comparison of the simulated and experimental results shows that there were some differences between the two. This difference is likely due to the fact that the simulation did not take into account the physical characteristics of the system, such as friction and air resistance. Additionally, the experimental results showed a higher overshoot than the simulated response, which may also be due to the physical characteristics of the system. However, despite these differences, the experimental results were still able to demonstrate the effectiveness of the design and the control algorithm.

Overall, the project demonstrates the importance of control systems in industrial applications, and the effectiveness of using PID controllers to stabilize and control complex systems. Further improvements to the system could include more advanced control algorithms or the use of sensors to improve accuracy.

CHAPTER SEVEN

CONCLUSION AND RECOMMENDATION

7.1 Conclusion

In this project, we have developed a self-aiming cannon that can track and hit a target object based on its x and y coordinates. The system uses projectile formulas to calculate the firing angle and elevation angle required to hit the target, and a PID controller to control the servo motors that adjust the cannon's orientation.

The project successfully demonstrated the design and implementation of a coordinate-based self-aiming cannon using an Arduino Uno and PID controller. The system was able to accurately hit all the targets within the range, and the simulation and experimental results showed the effectiveness of the design and control algorithm.

In conclusion, the "Coordinate Based Self-aiming Cannon" project demonstrates the potential of control systems and automation in industrial applications, particularly in the field of defense and security. With further development and refinement, this technology could have applications in a variety of areas, including military, law enforcement, and industrial settings. Additionally, the project provides a useful application of control systems and automation concepts for students in the field of electrical and computer engineering, particularly in the area of industrial control.

7.2 Recommendation

One limitation of the current design is that it only takes into account the x and y coordinates of the target, and does not consider other factors such as wind speed or direction. Future work could include the addition of sensors to measure these factors, or the use of more advanced control algorithms that can take these factors into account.

Another potential limitation is the range of the cannon. While the current prototype was able to hit all the targets within the range, it may not be effective for longer distances. Future work could include the optimization of the design for longer ranges, or the use of more powerful firing techniques to increase the range.

Here are some recommendations for future improvement of our "Coordinate Based Self-aiming Cannon" project:

1. **Accuracy and precision:** To increase the accuracy and precision of our system, consider using more advanced sensors and algorithms.
2. **Robustness and reliability:** Make sure the system is designed to be robust and reliable. Consider testing our system under various conditions to ensure that it works consistently and reliably. One could also consider implementing redundancy in key components to ensure that the system can continue to function even if one component fails.
3. **Future applications:** Consider the potential applications of our project beyond a self-aiming cannon. The principles and technologies we are using could be applied to other systems that require accurate positioning and aiming, such as robotic arms or autonomous vehicles.

By addressing these considerations, the researcher can create a more robust and reliable system and increase the potential impact of our project beyond its current scope.

REFERENCE

[1] Bucco, D., “An Improved Algorithm for Artillery Fire Control”, Dept. of Defense, Weapons Systems Research Lab., Adelaide, Australia, 1983, pp. 1–69

[2] Costello, M., and Peterson, A., “Linear Theory of a Dual-Spin Projectile in Atmospheric Flight,” Journal of Guidance, Control, and Dynamics, Vol. 23, No. 5, 2000, pp. 789–797. doi:10.2514/2.4639

[3] J. L. Han, L. M. Wang, and X. H. Yang, “Firing Table Theory and Technology of Field Rocket Weapon System”, National Defense Industry Press, Beijing, China, 2015.

[4] J. X. Xu, Z. K. Qi, D. F. Lin et al., “Study on the compilation of fring tables and rapid fring elements binding of rocket gun system,” Journal of Nanjing University of Science and Technology, vol. 28, no. 3, pp. 333–336, 2004.

[5] Baliga, B. R., Agrawal, S. K., & Venkatesan, R. (2012). Design of a Self-Aiming Platform for Small Arms Weapons. International Journal of Engineering and Innovative Technology (IJEIT), 1(4), 7 12.

[6] Al-Rabayah, M. A., Al-Rabayah, F. M., & Al-Ali, H. (2014). Design and Implementation of a PID Controller for a Ball and Plate System. International Journal of Control and Automation, 7(10), 301-312.

[7] Joseph, A. J., Robinson, S. E., & Reinhart, C. E. (2009). Design and Development of an Automated Targeting System for a Paintball Marker. Journal of Applied Sciences, 9(1), 53-59.

[8] El-Sayed, A. M., & El-Milli, A. M. (2015). Design and Implementation of a PID Controller for a Quadcopter. International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering, 4(6), 5265-5272.

[9] Mohamed, A. H. S., & El-Baz, A. A. (2015). Design and Implementation of a PID Controller for a Mobile Robot. International Journal of Computer Applications, 125(1), 1-6.

[10] Ali, M. K. Al-Timemy, M. A. Basheer, and M. M. Hussain, "Design and Development of a Self-Aiming Air Cannon," Journal of Mechanical Engineering and Automation, vol. 6, no. 6, pp. 294 301, 2016

[11] Lee, J. S., Shim, D. H., & Kim, H. J. (2019). Autonomous air cannon system with real-time impact point prediction. Robotics and Autonomous Systems, 118, 12-20.

[12] D. H. Zhao, H. Z. Zhang, S. Q. Guo et al., “Researched on determining fring data of gun based on solving ballistic equations with binary search,” Fire Control & Command Control, vol. 37, no. 12, pp. 182-183, 2012.

[13] M. Yang, H. W. Gao, and Q. Z. Tang, “Research on compilation of fring tables of guided rocket,” Fire Control & Command Control, vol. 38, no. 12, pp. 156–159, 2013.

[14] Hainz, L., and Costello, M., “Modified Projectile Linear Theory for Rapid Trajectory Prediction,” Journal of Guidance, Control, and Dynamics, Vol. 28, No. 5, 2005, pp. 1006–1014. doi:10.2514/1.8027

[15] Nancy Hall, “The Drag Equation,” [Online document] [May 2021], Available at https://www.grc.nasa.gov/www/k-12/airplane/drageq.html

[16] ClueBot NG, “Density of air,” [Online document] [June 2022] , Available at https://en.m.wikipedia.org.wiki/Densityofair

[17] Hellacioussatyr, “Drag Coefficient,” [Online document] [June 2022] , Available at https://en.m.wikipedia.org/wiki/Drag_coefficient

[18] Staff Writer, “Density of Steel,” [Online document] [April 2020], Available at https://www.reference.com/science/density-steel-kg-m3-27e3ee1480071e8c

[19] Raymond A., John W. (2019). “Physics for Scientists and Engineers with Modern Physics, Tenth Edition”. Cangage, United States.

[20] Norman S. Nise, Control System Engineering, 7th edition, John Wiley&sons inc., California State Polytechnique University, Pomona, 2015.

[21] John P. Bentley, Principle of Measurement System, 4th edition, Prentice-Hall, England, 2005.

[22] K. Ogata, Modern Control Engineering, 5th edition, Prentice-Hall, Upper-Saddle River, 2010.

[23] Wikipedia contributors, "Binary search algorithm," Wikipedia, The Free Encyclopedia, Wikimedia Foundation, Inc., accessed June 15, 2023. [Online]. Available: https://en.wikipedia.org/wiki/Binary_search_algorithm.

APPENDIX A

WORK PLAN AND BUDGET

A.1 Work Plan

		No. of weeks from 7/03/2023 up to 20/06/2023						
No	Proposed Task	Week 1 & 2	Week 3	Week 4,5 & 6	Week 7 & 8	Week 9 & 10	Week 11 & 12	Week 13 & 14
1	Proposal submission and presentation	✓						
2	Data collection		✓					
3	Modeling and system analysis		✓	✓				
4	Designing appropriate control strategy		✓	✓				
5	Simulation with appropriate methods and software			✓	✓			
6	Components Collection	✓	✓	✓	✓			
7	Testing circuit and recording of results				✓			
8	Design appropriate improvement methods				✓	✓	✓	
9	Compile the final paper work						✓	
10	Prepare PPT and ready for presentation							✓

A.2 Budget

No	Name of Component	Specifications	Quantity	Cost of each item in ETB	Total Cost in ETB
1	Arduino uno	14-Digital and 6-analog pins	1	1750	1750
2	SG90 servo motor	Stall Torque: 1.2 kg/cm (4.8V), 1.6 kg/cm (6.0V)	3	580	1740
3	Potentiometer	50kΩ and 10kΩ	2	70	140
4	Breadboard	400 Pin	1	350	350
5	Jumper wire	Male-Female and Female-Female	40	6	240
6	Spray paint	Black	1	130	130
7	Wood timber	0.6m x 0.55m	1	280	280
8	Other accessories	Cutting tools, adhesives, lead iron			500
				Grand Total	5130

APPENDIX B

DC MOTOR CONSTANTS CALCULATION

The electrical characteristic of a motor can be determined by conducting a dynamometer test, which involves measuring the torque and speed of the motor while a constant voltage is applied.

$$\frac{R_a}{K_t} T_m(s) + K_b s\theta_m(s) = E_a(s) \tag{B.1}$$

Take the inverse Laplace transform of Eq. (B.1), we get

$$\frac{R_a}{K_t} T_m(t) + K_b \omega_m(t) = e_a(t) \tag{B.2}$$

Where the inverse Laplace transform of $s\theta_m(s)$ is $d\theta_m(t)/dt$ or, alternatively, $\omega_m(t)$.

When a DC voltage, e_a, is supplied, the motor will rotate at a constant angular velocity, ω_m, with a steady torque, T_m. Therefore, by removing the time-based functional relationship from Eq. (B.2), the following equation is obtained for the steady-state operation of the motor with a DC voltage input:

$$\frac{R_a}{K_t} T_m + K_b \omega_m = e_a \tag{B.3}$$

Solving for T_m gives us

$$T_m = -\frac{K_t K_b}{R_a} \omega_m + \frac{K_t}{R_a} e_a \tag{B.4}$$

Equation (B.4) represents a linear relationship between the torque (T_m) and the angular velocity (ω_m), which is commonly referred to as the torque-speed curve. This relationship is illustrated in Figure B.1. The point at which the torque axis intersects the vertical axis corresponds to the stall torque, T_{stall}, which is the maximum torque value that the motor can generate when the angular velocity is zero. Thus,

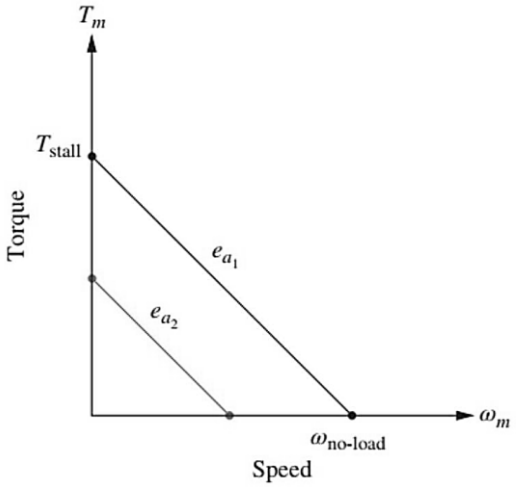

Fig. B.1: Torque-speed curves with an armature voltage, e_a, as a parameter

$$T_{\text{stall}} = \frac{K_t}{R_a} e_a \tag{B.5}$$

The velocity at which the motor rotates without any load or resistance and the torque is zero is referred to as the no-load speed, $\omega_{\text{no-load}}$. Thus,

$$\omega_{no-load} = \frac{e_a}{K_b} \tag{B.6}$$

The transfer function's electrical parameters of the motor can be calculated using Eq. (B.5) and Eq. (B.6), as follows:

$$\frac{K_t}{R_a} = \frac{T_{\text{stall}}}{e_a} \tag{B.7}$$

And

$$K_b = \frac{e_a}{\omega_{no-load}} \tag{B.8}$$

The electrical parameters, namely K_t/R_a and K_b, can be obtained through a dynamometer test of the motor that measures T_{stall} and $\omega_{\text{no-load}}$ for a specified e_a. Alternatively, the motor constants can be obtained from the motor's datasheet.

From the datasheet of SG90 servo motor at $e_a = 6$ V we have;

$$T_{\text{stall}} = 1.6 \text{ Kg} \cdot \text{cm}$$

That means it can support 1.6 Kg of mass placed at 1 cm away from the shaft. Now let's determine the force due to gravity (F_g).

$$F_g = M * g$$

Where M is mass of the object and g is acceleration due to gravity.

$$F_g = 1.6 \text{ Kg} \times 9.81 \text{ m/s}^2 = 15.696 \text{ N}$$

Therefore,

$$T_{\text{stall}} = 15.696 \text{ N} \times 10^{-2} \text{ m} = 0.157 \text{ Nm}.$$

And from the datasheet $\omega_{\text{no-load}}$ is;

$$\omega_{n0-\text{ load}} = 0.12 \text{ s}/60^\circ$$

Changing the unit of $\omega_{\text{no-load}}$ to rad/sec yields

$$\omega_{\text{no-load}} = \frac{60^\circ}{0.12\text{sec}} x \left(\frac{\pi\text{rad}}{180^0}\right) = 8.73 \text{ rad/sec}$$

From Eq. (B.7) we have

$$\frac{K_t}{R_a} = \frac{T_{\text{stall}}}{e_a} = \frac{0.157 \text{ Nm}}{6 \text{ V}} = 0.02617 \text{ Nm/V}$$

And from Eq. (B.8) we have

$$K_b = \frac{e_a}{\omega_{\text{no-load}}} = \frac{6 \text{ V}}{8.73 \text{ rad/sec}} = 0.6873 \text{ V/rad/sec.}$$

The above results are obtained by taking the armature inductance, $L_a = 0$.

APPENDIX C

PROJECT CODES

C.1 MATLAB Code to Find Step Response of the Open Loop System

```
% Clear all variables
clear variables; clc;
num = [0 0.7707];
dnum = [6.5e-6 0.0121 1.136 0];
% Open loop transfer function
G_hor = tf(num, dnum);
step(G_hor); grid on;
```

C.2 MATLAB Code to Find Step Response of the Closed Loop System

```
% Clear all variables
clear variables; clc;
num = [0 0.7707];
dnum = [6.5e-6 0.0121 1.136 0];
% Open loop transfer function
G_hor = tf(num, dnum);
%Closed loop transfer function
Gcl_hor = feedback(G_hor, 1, -1);
step(Gcl_hor); grid on;
```

C.3 MATLAB Code to Find Step Response of the PID Controlled System

```
% Clear all variables
clear variables; clc;
num = [0 0.7707];
dnum = [6.5e-6 0.0121 1.136 0];
% Open loop transfer function
G_hor = tf(num, dnum);
% Closed loop transfer function
Gcl_hor = feedback(G_hor, 1, -1);
% PID parameters
```

```
Kp = 1646.7;
Ki = 2.2e5;
Kd = 3.09;
pid_hor = pid(Kp, Ki, Kd);
% Transfer function with PID
Gpid_hor = feedback(pid_hor*G_hor, 1, -1);
step(Gpid_hor); grid on;
```

C.4 MATLAB Code to Tune the Transfer Function using pidTuner

```
% Clear all variables
clear variables; clc;
num = [0 0.7707];
dnum = [6.5e-6 0.0121 1.136 0];
% Open loop transfer function
G_hor = tf(num, dnum);
%PID parameters
Kp = 1646.7;
Ki = 2.2e5;
Kd = 3.09;
pid_hor = pid(Kp, Ki, Kd);
pidTuner(G_hor, pid_hor);
```

C.5 Arduino Code

```
#include <Servo.h>
#include <math.h>
// The input pins for the potentiometers
int inputPin1 = A0;
int inputPin2 = A1;
// The output pins for the servo motors
int outputPin1 = 9;
int outputPin2 = 10;
int outputPin3 = 11;
// The setpoints for the angles
int setpoint1 = 75;
```

```
int setpoint2 = 97;
// The PID constants for the two degrees of freedom
float kp1 = 489.7;
float ki1 = 1.04e4;
float kd1 = 5.78;
float kp2 = 489.7;
float ki2 = 1.04e4;
float kd2 = 5.78;
// The variables for the PID computation
float error1, lastError1, integral1, derivative1, output1;
int coutput1;
float error2, lastError2, integral2, derivative2, output2;
int coutput2;
//CONSTANTS
const float pi = 3.14159265358979323846264338327950288;
// Initial offset distance from the ground
float h0 = 0.04; // in meter
// Gravitational acceleration
float g = 9.81; // in m/s^2
// Initial velocity
float u = 3.45; // in m/s
// Length of the cannon mouth
float l = 0.083; // in meter
// Offset distance from the vertical axis of rotation
float x0 = 0.0325; // in meter
// Servo motor constants
// Servo1 offset angle
float offset_angle1 = -11;
// Servo2 offset angle
float offset_angle2 = 97.0;
// Servo1 bounds
float servo1low = 25.0 + offset_angle1;
float servo1high = 155.0 + offset_angle1;
```

```
// Servo2 bound
float servo2low = -16.0 * pi / 180.0;
float servo2high = 45.0 * pi / 180.0;
// Potentiometer constants
int pot1min = 155;
int pot1max = 853;
int pot2min = 158;
int pot2max = 876;
// The servo objects
Servo servo1;
Servo servo2;
Servo trigger;
void setup() {
  // Initialize the serial communication
  Serial.begin(9600);
  // Attach the servo motors to their output pins
  servo1.attach(outputPin1);
  servo2.attach(outputPin2);
  trigger.attach(outputPin3);
  // Set the initial positions of the servo motors to their setpoints
  servo1.write(setpoint1);
  servo2.write(setpoint2);
  trigger.write(90);
  // Set the input pins as inputs
  pinMode(inputPin1, INPUT);
  pinMode(inputPin2, INPUT);
  pinMode(LED_BUILTIN, OUTPUT);
}
float calc_setpoint1(float x, float y) {
  float range = sqrt(x*x + y*y);
  float correction_angle = degrees(asin(x0/range)) + offset_angle1;
  if (x == 0) return 90.0 + correction_angle;
  float angle1 = degrees(atan(y / x));
```

```
  if (x < 0) return min(angle1 + 180.0 + correction_angle, servo1high);
  return max(angle1 + correction_angle, servo1low);
}
float calc_range(float th) {
  float sinth = sin(th), costh = cos(th);
  float range = u * costh * (u * sinth + sqrt(u * u * sinth * sinth +
2 * g * h0 + 2 * g * l * sinth)) / g + l * costh;
  return range;
}
float calc_setpoint2(float x, float y) {
  float target_range = sqrt(x*x + y*y - x0*x0);
  float l = servo2low, r = servo2high;
  while (abs(l - r) > 0.001) {
    float mid = (l + r) / 2;
    if (calc_range(mid) < target_range) {
      l = mid;
    } else {
      r = mid;
    }
  }
  return degrees(l) + offset_angle2;
}
void pid(int setpoint1, int setpoint2) {
  // Read the input values from the potentiometers
  int input1 = analogRead(inputPin1);
  int input2 = analogRead(inputPin2);
  // Convert the input values to angles
  int angle1 = map(input1, pot1min, pot1max, 0, 180);
  int angle2 = map(input2, pot2min, pot2max, 0, 180);
  // Initialize the PID variables
  lastError1 = 0;
  integral1 = 0;
  lastError2 = 0;
  integral2 = 0;
```

```
  int cnt = 0;
  while (cnt < 3) {
    if (setpoint1 != angle1 || setpoint2 != angle2){
      cnt = 0;
    }
    else cnt++;
    // Read the input values from the potentiometers
    input1 = analogRead(inputPin1);
    input2 = analogRead(inputPin2);
    // Convert the input values to angles
    angle1 = map(input1, pot1min, pot1max, 0, 180);
    angle2 = map(input2, pot2min, pot2max, 0, 180);
    // Compute the errors for the angles
    error1 = setpoint1 - angle1;
    error2 = setpoint2 - angle2;
    // Compute the integral and derivative terms for the errors
    integral1 += error1;
    derivative1 = error1 - lastError1;
    integral2 += error2;
    derivative2 = error2 - lastError2;
    // Compute the control output values for the angles using PID
control
    output1 = kp1 * error1 + ki1 * integral1 + kd1 * derivative1;
    output2 = kp2 * error2 + ki2 * integral2 + kd2 * derivative2;
    // Limit the output values of the controller
    coutput1 = constrain(output1, -30, 30);
    coutput2 = constrain(output2, -30, 30);
    // Update the positions of the servo motors
    servo1.write(setpoint1 + coutput1);
    servo2.write(setpoint2 + coutput2);
    // Print the input and setpoint to the serial monitor
    Serial.println("");
    Serial.print("angle1: ");
    Serial.print(angle1);
```

```
    Serial.print(", Setpoint1: ");
    Serial.print(setpoint1);
    Serial.print(", angle2: ");
    Serial.print(angle2);
    Serial.print(", Setpoint2: ");
    Serial.print(setpoint2);
    // Update the last error values for the angles
    lastError1 = error1;
    lastError2 = error2;
    // Delay for a short period of time to allow the servo motors to move
    delay(20);
  }
}
void loop() {
  while (Serial.available() == 0);
  float x = Serial.parseFloat();
  float y = Serial.parseFloat();
  while (Serial.available() > 0) Serial.read();
  setpoint1 = round(calc_setpoint1(x, y));
  setpoint2 = round(calc_setpoint2(x, y));
  delay(500);
  pid(setpoint1, setpoint2);
  delay(500);
  trigger.write(60);
  digitalWrite(LED_BUILTIN, HIGH);  // turn the LED on (HIGH is the voltage level)
  delay(2000);
  trigger.write(90);
  digitalWrite(LED_BUILTIN, LOW);  // turn the LED off by making the voltage LOW
}
```