ADDIS ABABA SCIENCE AND TECHNOLOGY UNIVERSITY

Coordinate-Based Self-Aiming Cannon

Semester Project

By

FITSUM WORKNEH (ETS0446/11)

MELKAMU KUMERA (ETS0704/11)

NARDOS WEHABE (ETS0811/11)

Adviser: BIRUK T.

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

CONTROL STREAM

COLLEGE OF ELECTRICAL AND MECHANICAL ENGINEERING

JUNE 2022

Approval Page

Title: Coordinate-Based Self-Aiming Cannon

Student 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:	**Biruk T.**	**MSc**	_____	_____
Internal Examiner:	_____	_____	_____	_____
External Examiner:	_____	_____	_____	_____

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

Declaration

I hereby declare that this thesis entitled "Coordinate-Based Self-Aiming Cannon" was prepared by me, with the guidance of my advisor. The work contained herein is my 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 student advisor: Biruk T. **Signature_____, Date:_____**

Abstract

The firing system on cannon uses the foot of the cannon crew, which is very dangerous with the position of the crew on top of the cannon when firing. So, a firing system that can be remotely controlled by a computer is required. The design of the cannon firing control system uses a keypad as the firing command input. The fire-control solution is an important element of any modern weapon system, providing precise aiming of the gun to enable highly accurate projectile impact. To be practical, the fire-control solution must be computed rapidly and reliably while simultaneously including all pertinent physical effects that can alter the trajectory and impact point. In this project the projectile of cannon ball is analyzed and controlled using MATLAB and microcontroller respectively.

Keyword: Microcontroller, Trajectory

Acknowledgments

First and foremost, we would like to thank God almighty for his limitless support thought our academic life, our family without their help we wouldn't have been here, we want to thanks Addis Ababa Science and Technology University for giving us all the necessary knowledge and experience we need to actively participate and help our countries development in our field of study. We also want to express our sincere gratitude to our teacher and advisor Biruk T.

Table of Contents

Abbreviations and Acronyms

ADC	Analog Digital Converter
DC	Direct Current
DOF	Degree of Freedom
I2C	Inter-Integrated Circuit
LCD	Liquid Crystal Display
LED	Light Emitting Diode
MCU	Micro Controller Unit
MLRS	Multiple Launch Rocket System
PIC	Programmable Interface Controller
PWM	Pulse Width Modulation
SPI	Serial Peripheral Interface

List of Figures

Chapter 1

1. Introduction

1.1. Background

The development of increasingly advanced technology today, has encouraged the growing use of technology to support the military. One of its applications is the use of computers as weapons control.

Projectile motion is defined as the flight of an object near the Earth's surface under the action of gravity alone. Understanding the factors that affect the motion of projectiles has led to several major milestones in human history.

The ability to make and then use a spear to hunt large animals allowed our ancestors to gather more food. Later on, as people developed a better understanding of the factors that govern projectile motion, they were able to build new tools for launching projectiles that could travel farther in the air and hit targets with great accuracy. These tools included such things as bows and arrows, trebuchets, and cannons.

When a projectile is in flight, we assume that gravity is the sole force acting on it. Although scientists recognize that air resistance does affect the flight of a projectile, under most circumstances the effect of air resistance can be ignored. When we ignore air resistance, the initial velocity of the projectile governs the horizontal component of the projectile's velocity and the force of gravity governs its vertical component.

People often want to be able to predict how long a projectile will stay in the air (i.e., the hang time) after it is launched. There are a number of variables that may, or may not, affect the hang time of a projectile. These variables include the launch angle (denoted as θ) the initial velocity of the projectile, and the mass of the projectile. Some of these variables may also interact with each other, so the effect of any one variable may differ depending on the value of another variable. People therefore need to understand not only how these three variables affect the motion of a projectile but also how they interact with each other to predict how long a projectile will remain in the air after it is launched.

In this project we tried to model and control the projectile of a cannon ball using MATLAB and PIC18F43K22 microcontroller respectively.

1.2. Statement of the Problem

The need for fire control system in Ethiopia has been fulfilled by overseas procurement or purchase. Thus, the level of dependence on other countries is very high and the purchase also costs a lot. In repairing the system in case of damage, the service of foreign technicians is more likely to be used, where the time and cost depend on the company. Therefore, the maintenance costs are also high.

The manual firing system of a cannon uses crew control by stomping the firing pedal with the foot of the cannon crew, where the gun crew is on the gun platform. It is very risky for the cannon crew, e.g., accidents might occur when the ammunition fails to eject and explodes in the cannon barrel. Therefore, to maximize the use of existing equipment while prioritizing zero accidents at the same time, it is necessary to design a self-aiming cannon firing system so that the safety of crew or operators will not be compromised so that the use of this cannon can continue to be maximized to support the defense forces.

Solving the fire-control problem amounts to aiming a gun so that a projectile hits a desired target. If the launch point, impact point, projectile exterior ballistics, and atmospheric conditions along the line of fire are all known, then the necessary gun elevation and azimuth to hit the target can be calculated.

1.3. Objectives of the Project

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 Objective

The specific objective of this project includes:

- To model the projectile of cannon ball
- To design control of DC motor.
- To effectively use the limited memory of the microcontroller to accomplish the main objective.
- To test the system with physics simulation.

1.4. Significance of the Project

The fire-control solution is an important element of any modern weapon system, providing precise aiming of the gun to enable highly accurate projectile impact. To be practical, the fire-control solution must be computed rapidly and reliably while simultaneously including all pertinent physical effects that can alter the trajectory and impact point.

1.5. Delimitations of the Project

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

1.6. Definition of Basic Terms Used in the Project

Pulse width modulation(PWM): or Pulse duration modulation(PDM), is a method of reducing the average power delivered by an electrical signal , by effectively chopping it up in to discrete parts. The average value of voltage(and current) fed to the load is controlled by turning the switch is on compared to the off periods, the higher the total power supplied to the load.

Servo motor: is a rotary actuator or linear actuator that allows for precise control of angular or linear position, velocity and acceleration. It consists of a suitable motor coupled to a sensor for position feedback.

Microcontroller: is a compact integrated circuit designed to govern a specific operation in an embedded system. A typical microcontroller includes a processor, memory and input/output (I/O) peripherals on a single chip.

PIC: is an abbreviation used for peripheral interface controller. PIC microcontroller is the smallest microcontroller in the world and are programmed to execute large number of operations. These were initially designed to support PDP (programmed data processor)computers, for controlling the peripheral devises. It is based on RISC architecture.

Chapter 2

2. Literature Review

2.1. Introduction

Solving the fire-control problem amounts to aiming a gun so that a projectile hits a desired target. If the launch point, impact point, projectile exterior ballistics, and atmospheric conditions along the line of fire are all known, then the necessary gun elevation and azimuth to hit the target can be calculated. A part of the required input information is the atmospheric wind velocity field. An accurate model of the spatially varying wind can be crucial in accurately finding the correct solution and hitting the target[1].

Current fire-control solutions account for the effect of atmospheric wind in a rudimentary manner, typically assuming a constant crosswind that is estimated in the field or measured at the firing site. With the advent of advanced wind-measurement systems (light detection and ranging, for example), it is now possible to accurately measure three-dimensional wind velocities at numerous points approximately along the path of a direct-fire projectile[2].

The process of calculating the firing angle, however, is relatively complicated. There are two commonly used approaches for obtaining the firing angle: the traditional firing table approach and the iterative search approach via a trajectory program, of which the former can be realized by interpolating or approximating the tabular data under standard conditions then applying correction factors. Currently, the linear and quadratic interpolations are still two commonly used interpolation approaches for calculating the firing angle, and one of commonly used approximation approaches is the polynomial response surface, particularly the polynomials of orders 1 through 4[3].

The calculation result by using the firing table approach can meet the accuracy requirement within a certain range. However, when the actual atmospheric conditions are quite different from the standard atmospheric conditions, the firing angle obtained by using the firing table approach is rather inaccurate, which will lead to the firing mission failure of the free and simple controlled rockets with limited maneuverability. The iterative search approach is another method for calculating the firing angle by employing the trajectory program and iterative algorithm, which is applicable to the

firing angle calculation under both standard and actual atmospheric conditions. The calculation efficiency depends strongly on the trajectory model, the iterative initial value, and the iterative algorithm[4].

2.2. Related Works

Due to the classified and confidential nature of the project, we weren't able to find literatures that are closely related to our project. Below, we tried to mention the literatures we found.

Despite the significant improvements that have been achieved, the details of the algorithms are mostly confidential and limited publications are currently available. By employing the six DOF trajectory program and an improved iterative search approach, L. J. Zhou et al. proposed a method for calculating the firing angle of the MLRS, which can reduce the number of iterations substantially. D. H. Zhao et al. presented a method to determine the firing angle of the artillery based on the three- DOF trajectory program and binary search method[5].

Aiming at solving the problem of firing angle calculation for the multiple launch rocket system (MLRS) under both standard and actual atmospheric conditions, an efficient method based on large sample data and metamodel is proposed. This article first shows the importance of wind knowledge along the line of fire for accuracy, particularly for long-range direct-fire shots. Then, a method to compute the fire-control solution of a projectile is defined, including the effect of exactly known spatially varying winds[6].

P. Chusilp et al. came up with four iterative search algorithms and compared the calculation efficiency of four algorithms with the firing angle calculation of M107 projectile as a case study. In addition, W. Charubhun et al. put forward an efficient method of firing angle calculation for the MLRS with the six DOF trajectory program and iterative binary search method by having prior knowledge of the ranges versus the firing angles[7].

Chapter 3

3. Methodology

3.1. The Project Methodology

The methodology of this project 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.

- First we mathematically model the system.
- Then we analyzed the system and designed a controller as control engineer.
- We implemented the designed system.
- Finally, we tested the system.

3.2. Method of Simulation

- MATLAB R2019a is used for physics simulation. See Appendix II for the codes used in the physics simulation.
- Proteus 8.13 is used for simulation of the microcontroller and the whole modeled system.
- MPLAB and XC8 are used for building the microcontroller program. See Appendix I for the codes used for the microcontroller.

Chapter 4

4. Modeling and Design of The Control System

4.1 Set Point (Firing Angles) 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, $\boldsymbol{\theta}$
- Angle from positive x-axis, $\boldsymbol{\phi}$

4.1.1 Drag Force

In fluid dynamics, the drag equation is a formula used to calculate the force of drag experienced by an object due to movement through a fully enclosing fluid, in our case the air enclosing the cannon ball. The equation is[8]:

$$F_d = \frac{1}{2}\rho U^2 C_d \mathrm{A}$$

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 is the drag coefficient, a dimensionless coefficient related to the object's geometry and taking into account both skin friction and form drag.

Density of air, ρ_a = 1.225 Kg/m^3[9]. 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 m/s. In our project we use spherical steel ball with radius of r = 5mm. The reference

area is πr^2 = 7.85 x 10^{-5} m^2. C_d, drag coefficient of sphere is 0.47[10]. Inserting this values in the above equation we get F_d = 2.26 x 10^{-5} N.

The density of steel is around 8000 Kg/m^3[11]. The volume of spherical object is found to be V = $^4/_3\pi r^3$. Our cannon ball will have a volume of 5.236 x 10^{-7} m^3. Taking this, the mass of our cannon ball will be around 4.19 x 10^{-3} kg.

Applying Newton's law, we get acceleration due to drag force a_d = 0.0054 m/s^2, which is around 2000 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 Angle from the Ground

Using equations of motion[12]

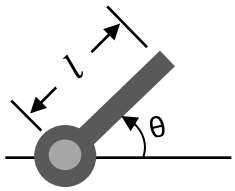
$$\Delta S = Ut + \frac{1}{2}at^2 \qquad (1)$$

Where, ΔS is change of displacement

U is initial velocity

t is time of flight

a is acceleration



$l\cos\theta$ acceleration along X-axis

$$\Delta X = U_x t$$

But $\Delta X = X - X_0$ and

$$U_x = U\cos\theta$$

$$X = X_0 + U\cos\theta t = l\cos\theta + Ut\cos\theta \qquad (2)$$

To get time of flight, t ;

$$S = Ut + \frac{1}{2}at^2$$

$$\Delta Y = U_y t - \frac{1}{2}gt^2$$

$$-l\sin\theta = U\sin\theta\, t - \frac{1}{2}gt^2$$

Solving for t, we get

$$t = \left(\frac{u\sin\theta \pm \sqrt{u^2\sin^2\theta + 2gl\sin\theta}}{g}\right) \qquad (3)$$

Inserting (3) in (2) we get

$$x = l\cos\theta + u\cos\theta \cdot \left(\frac{u\sin\theta + \sqrt{u^2\sin^2\theta + 2gl\sin\theta}}{g}\right) \qquad (3)$$

Rearranging (3) we get ;

$$gx^2 - 2gxlcos\theta - 2u^2xsin\theta cos\theta + gl^2cos^2\theta = 0 \qquad (4)$$

Since solving this equation for θ and implementing it is difficult, we approximated this function to degree 3 polynomial function. We will discuss it in next chapter.

4.1.3 Angle from positive X-axis

Calculation of angle from positive X-axis

$$\emptyset = \tan^{-1}\left(\frac{Y}{X}\right) \qquad (5)$$

But we need to be careful on angles in 2nd and 3rd quadrants. The $\tan^{-1}()$ function only gives us angles with range between -90 and 90. We can get the exact angle by locating y and x on the cartesian axis.

4.2 Controller Design

4.2.1 Mathematical model of DC motor [13]

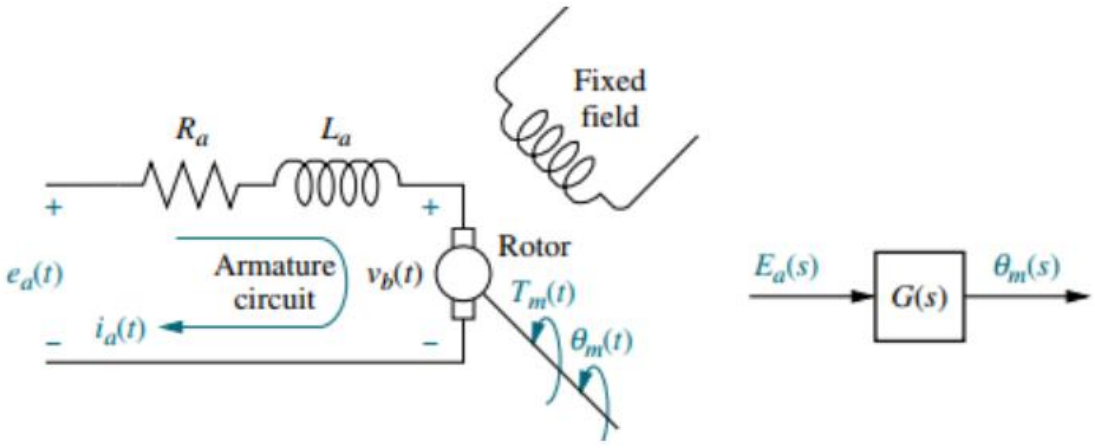


Figure 4.1: Electrical circuit diagram of a separately exited DC motor

$$Ea(s) = (Ra + LaS)Ia(s) + Vb(s)$$

Applying KVL on the armature circuit:

$$e_a = Raia + La\frac{dia}{dt} + Vb. \quad (6)$$

Take the Laplace transform of equation 1 assuming zero initial conditions

$$Ea(s) = (Ra + LaS)Ia(s) + Vb(s) \quad (7)$$

$$Vb(s) = K_b\omega(s), K_b$$

is constant that depends on the motor components.

Where $\omega(s) = S\theta_m(s)$

$$Vb(s) = K_bS\theta_m(s). \quad (8)$$

The motor torque, Tm, is given by:

$$Tm \propto \Phi Ia$$

For separately exited DC motor:

$$\Phi \propto \text{If}$$

When constant field current is established is the field coil the motor torque is given by:

$$\text{Tm}\ (s) = (K1KfIf)Ia(s) = KmIa(s) \quad (9)$$

When permanent magnet is used, we have:

$$Tm(s) = KmIa(s) \quad (10)$$

From Newton's second law of motion:

$$T_m(s) = bS\theta_m(s) + J_mS^2\theta_m(s). \quad (11)$$

From equation (10) and (11)

$$Ia(s) = \frac{bs\theta_m(s) + J_ms^2\theta_m(s)}{Km} \quad (12)$$

Substituting equation (8) and (12) into equation (7) we get:

$$E_a(s) = \frac{(R_a + L_aS)(J_ms^2 + bs)\theta_m(s)}{K_m} + K_bS\theta_m(s) \quad (13)$$

However, for many DC motors time constant of the armature, $\tau = \frac{L_a}{R_a}$, is negligible

$$\text{i.e. } L_a \approx 0$$

Therefore, the transfer function relating input $Ea(s)$ and output $\theta m(s)$ is:

$$G(s) = \frac{\theta_m(s)}{E_a(s)} = \frac{\frac{K_m}{R_a Jm}}{s\left[s + \frac{1}{J_m}\left(b + \frac{K_m K_b}{R_a}\right)\right]}$$

For geared DC motor: Assuming linear gear system; We have the following relations

$$r_1\theta_1 = r_2\theta_2$$
$$\frac{\theta_2}{\theta_1} = \frac{r_1}{r_2} = \frac{N_1}{N_2}$$

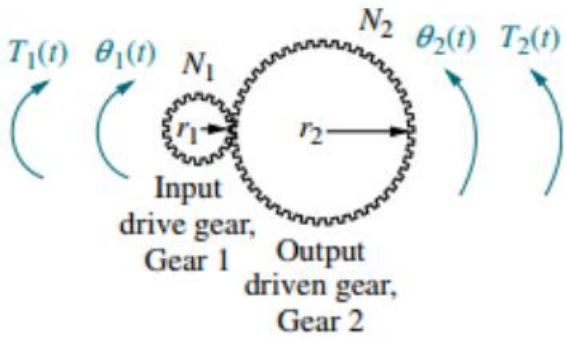


Figure 4.2: Gear box

Assuming the gears are lossless, that is they do not absorb or store energy, the energy into Gear 1 equals the energy out of Gear 2.

$$T_1\theta_1 = T_2\theta_2$$

Reflecting the Mechanical impedances from load side to input side of the system:

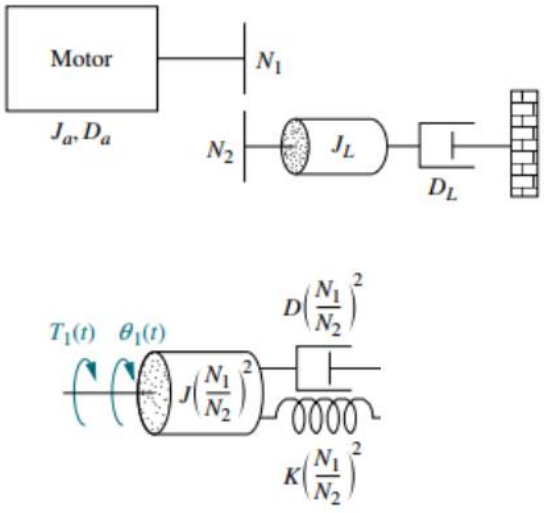


Figure 4.3: Equivalent reflected system

The effective moment of inertia and damping can be calculated as:

$$J_{equ} = J_m + J_l\left(\frac{N_1}{N_2}\right)^2, b_{equ} = b + b_l\left(\frac{N_1}{N_2}\right)^2$$

If there is no gear involved (i.e. direct coupling between load and motor):

$$J_{equ} = J_m + J_l, b_{equ} = b + b_l \text{ and } \theta_m = \theta_l$$

Where J_m *is* motor inertia *and* J_l is load inertia, b *is motor damping constant and* b_l is load damping constant θ_m is angular position of the motor and θ_l is angular position of the load.

Therefore ,the transfer function relating the load position and the input armature terminal voltage is:

$$G(s) = \frac{\theta_l(s)}{E_a(s)} = \frac{\frac{K_m}{R_a J_{equ}}}{s\left[s + \frac{1}{J_{equ}}\left(b_{equ} + \frac{K_m K_b}{R_a}\right)\right]}, \text{ where } \theta_m(s) = \theta_l(s)\frac{N_2}{N_1} = \theta_l(s)$$

From the motor specifications we have:

$$R_a = 25\Omega$$
$$K_b = 0.25\text{Vrad}^{-1}\text{s}^{-1}$$
$$K_m = 0.25\text{NmA}^{-1}$$
$$J_{equ} = 0.003\text{Kgm}^2$$
$$b_{equ} = 0.02410\text{Nmsrad}^{-1}$$

Substituting the values, we get the transfer function of the DC motor together with the plant to be:

$$G(s) = \frac{3.3}{S(S + 8.87)}$$

4.2.2 Stability Analysis of the Closed Loop System Without a Controller

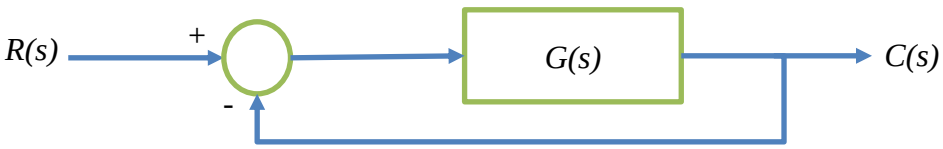


Figure 4.4: Block diagram of closed loop system without controller

Let the transfer function of the closed loop transfer function be $T(s)$,

$$T(s) = \frac{G(s)}{1 + G(s)} = \frac{\frac{3.3}{S(S + 8.87)}}{1 + \frac{3.3}{S(S + 8.87)}} = \frac{3.3}{S^2 + 8.87S + 3.3}$$

For the closed loop system to be absolutely stable the closed loop poles should be located to the left of the jw axis.

The poles of the open loop system can be determined using MATLAB as follows:

```
%open loop step response
n=3.3;
```

```
d=[1 8.87 0];
Gopen=tf(n,d);
step(Gopen);
p=[1 8.87 0];
pole=roots(p);
grid on;
Gopen =
      3.3
  ------------
  s^2 + 8.87 s
 Continuous-time transfer function.
pole =
  0
 -8.8700
```

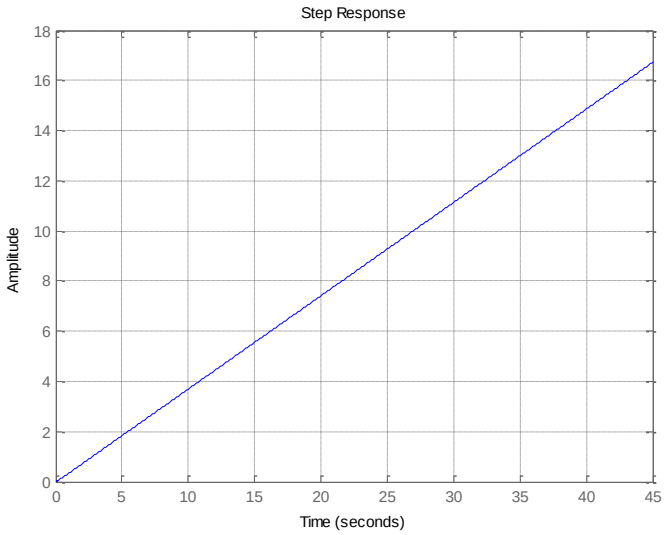


Figure 4.5: Step response of the uncompensated open System

The step response of the uncompensated system is determined using MATLAB as follow. Also, the poles of the open loop system can be determined using MATLAB as follows:

```
%closed loop step response
n=3.3;
d=[1 8.87 0];
Gopen=tf(n,d);
Gloop=feedback(Gopen,1,-1)
p=[1 8.87 3.3];
pole=roots(p)
step(Gloop);
grid on;
>> closedloop
Gloop =
        3.3
  ------------------
  s^2 + 8.87 s + 3.3
Continuous-time transfer function.
pole =
-8.4809
-0.3891
```

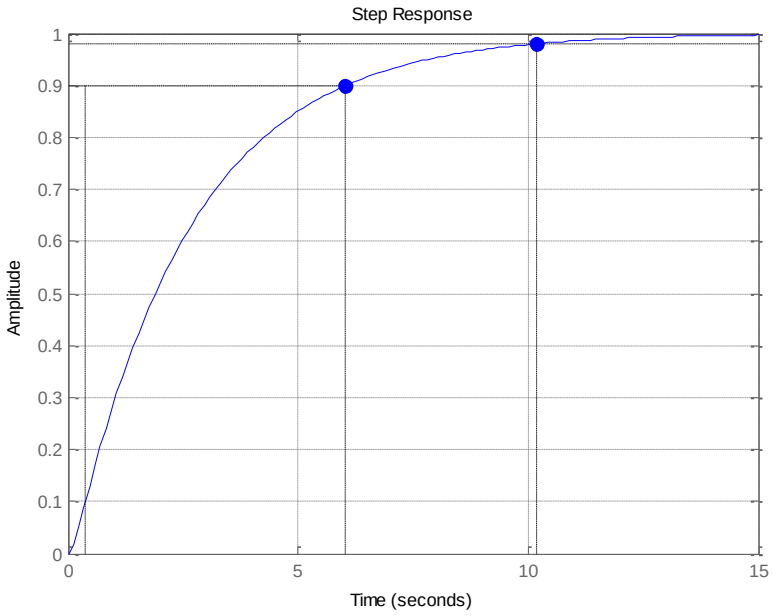


Figure 4.6: Step response of uncompensated closed system

Since the system is Type1 system as expected the steady state error for step input is zero. The final value is 1 implies

$$e_{ss} = 0$$

From this we deduce that the transient response of the system should be improved. There are different compensators to improve transient response characteristics of a system. This includes Lead compensator PD controller, PID controller etc. We choose PID controller to improve the system's dynamic response.

We choose PID in order not to affect the steady state response 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.

4.2.3 Controller

A controller is a mechanism that seeks to minimize the difference between the actual value and the desired value (set point) of the controlled variable (the process variable). Controllers are a fundamental concern of Control Engineering and are used in all complex control systems. There are different types of controllers that can be used to control the response of the system to a certain input, this includes:

Lead Compensator: adds high frequency component that their effect will damp out the oscillation in the transient response. But this controller may cause the steady state

response to deviate from the desired response if not selected carefully. PD controller is an example lead compensator.

Lag Compensator: this compensator tries to reduce the steady state error without changing the root locus of the original system (i.e. without changing the dominant closed loop poles as much as possible so that the transient response is not affected much). Example PI controller. We use this compensator when the transient response is satisfactory and need to correct the steady state response. PI controller is an example of this group

Lag-Lead Compensator: this compensator is the combination of the above two compensators. We use it if both the transient and steady state response needs modification or when modifying the transient response affects the steady state or vice versa. Example PID controller.

We use PID controller so that modifying the transient will not degrade the steady state response. Science the response of the system without controller is S shaped Ziegler–Nichols first method can used to start tuning the PID controller to determine its constants, i.e. Kp, Ti and Td as follows:

Ziegler–Nichols first method:

In this method, if the mathematical model of the plant is not known we obtain experimentally the response of the plant to a unit-step input, since we have the mathematical model, we obtain the response graph to unit step input using MATLAB as:

We take our specifications to be $t_s = 0.23\text{sec}, \%M_p < 2\%$, and $e_{ss} = 0$.

The S-shaped curve may be characterized by two constants, delay time L and time constant T For our case the value of L and T is:

$$L \approx 0.1675\text{sec}, T = 5.165 - 0.1675 = 4.99\text{sec}$$

Using first method the PID controller constants are given by:

$$K_p = 1.2\frac{T}{L}, T_i = 2L, T_d = 0.5L$$

Therefore:

$$K_p = 1.2\left(\frac{4.99}{0.1675}\right) = 35.8$$
$$T_i = 2L = 2(0.1675) = 0.335$$
$$T_d = 0.5L = 0.5(0.1675) = 0.084$$

Ki and Kd can be determined from Ti and Td respectively as:

$$K_i = K_p T_i = 35.8 * 0.335 = 12$$
$$K_d = K_p T_d = 35.8 * 0.084 = 2.99$$

Now let's check whether the determined parameter value of the PID controller give as the desired response. Using MATLAB for the simulation purpose we found the following results. MATLAB code

```
%after compensation
%closed loop step response
n=3.3;
d=[1 8.87 0];
Gopen=tf(n,d);
Gloop=feedback(Gopen,1,-1);
step(Gloop);
grid on;
hold on;
Gc=pid(35.8,11.993,2.99);
Gpid=feedback(Gloop*Gc,1,-1)
p=[1 18.74 121.4 39.58];
pole=roots(p)
step(Gpid);
legend('without pid controller','with pid controller');
>> withpid
Gc =Kp + Ki *1 /s + Kd * s
with Kp = 35.8, Ki = 12, Kd = 2.99
Continuous-time PID controller in parallel form.
Gpid =
     9.867 s^2 + 118.1 s + 39.58
  ---------------------------------
  s^3 + 18.74 s^2 + 121.4 s + 39.58
 Continuous-time transfer function.
pole =
  -9.1980 + 5.5199i
  -9.1980 - 5.5199i
  -0.3440 + 0.0000i
```

Simulation result:

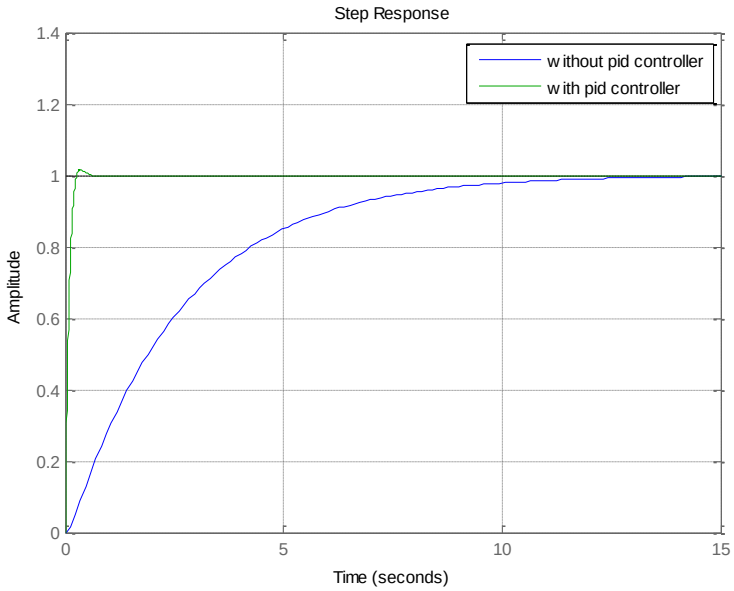


Figure 4.7: Step response of PID compensated system

SIMULINK block diagram of the compensated system

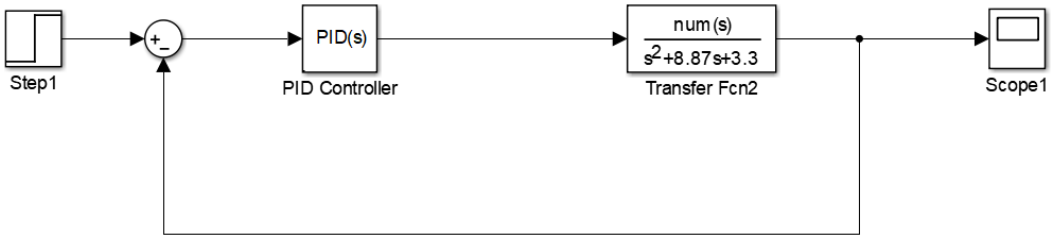


Figure 4.8: SIMULINK block diagram

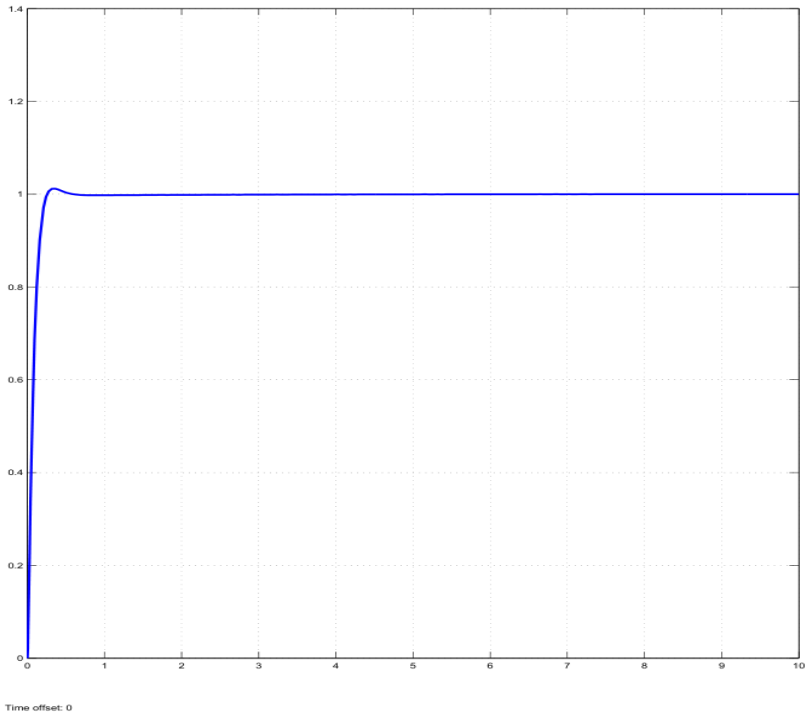


Figure 4.9: SIMULINK simulation result

As we can see from Figure 4.9 above the raise time tr is reduced from

$$t_r = 5.165 \text{ sec to } t_r = 0.1675 \text{ sec}$$

and the settling time is also reduced from ts is also reduced from

$$t_s = 10.2 \text{ sec to } t_s = 0.23 \text{ sec}$$

There is also an overshot of about $\%M_p \approx 1.99\%$,

Chapter 5

5. Implementation

5.1 Microcontroller

In this project we have used PIC18F43K22 microcontroller, due to its availability and low cost. But there are plenty of microcontrollers capable of performing the project's functionality.

Figure 5.1: PIC18F43K22

5.2 Servo Motor

A Servo Motor is a type of actuator (mostly circular) that allows angular control. There are many types of Servo motors available but in this project, we used hobby servo motors. Hobby servos are a popular because they are the inexpensive method of motion control. They provide an off-the-shelf solution for most of the R/C and robotic hobbyist's needs. They also eliminate the need to custom design a control system for each application.

Figure 5.2: Servo motor.

Most of the hobby servo motors have a rotational angle of 0- 180° but we can also get 360° servo motor if we're interested. This project uses a 0- 180° and -180 – 180 servo

motors. There are two types of Servo motors based on the gear, one is the Plastic Gear Servo Motor and the other is Metal Gear Servo Motor. Metal gear is used in places where the motor is subjected to more wear and tear, but it comes only at a high price.

Before we can start programming for the servo motor we should know what type of signal is to be sent for controlling the servo motor. We should program the MCU to send PWM signals to the signal wire of the Servo motor. There is a control circuitry inside the servo motor which reads the duty cycle of the PWM signal and positions the servo motors shaft in the respective place as shown in the picture below.

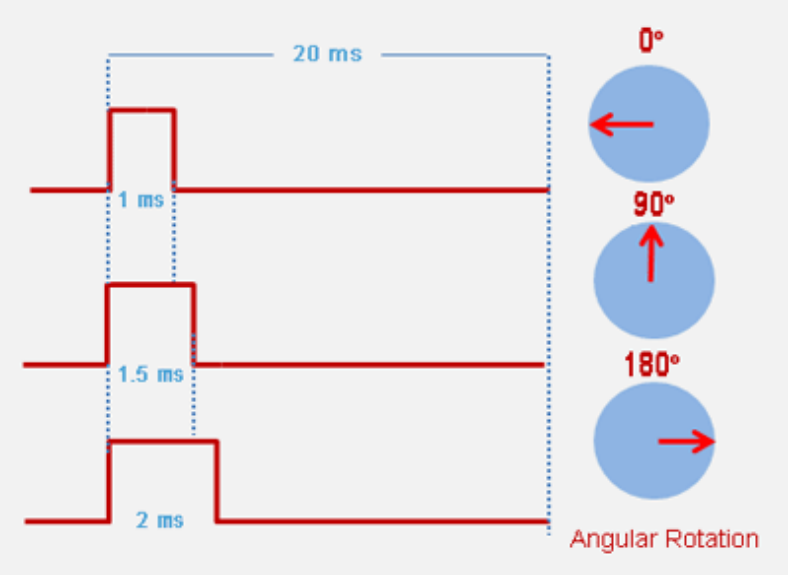


Figure 5.3: PWM to control servo motor.

Each servo motor operates on a different PWM frequencies (most common frequency is 50HZ) so we saw the datasheet of our motor to check the on which PWM period our servo motor works.

With the help of Proteus simulation, we can verify the PWM signal using an oscilloscope and also check the rotating angel of the Servo motor. Few snapshots of the simulation are shown on Chapter 6.

5.3 Pulse Width Modulation (PWM)

Our PIC MCU has a special module called Compare Capture module (CCP) which can be used to generate PWM signals. Here, we will generate a PWM of 5Hz with a variable duty cycle from 0% to 100%.

Pulse Width Modulation (PWM) is a digital signal which is most commonly used in control circuitry. This signal is set high (5v) and low (0v) in a predefined time and speed. The time during which the signal stays high is called the "on time" and the time during which the signal stays low is called the "off time". There are two important parameters for a PWM as discussed below:

Duty cycle of the PWM

The percentage of time in which the PWM signal remains HIGH (on time) is called as duty cycle. If the signal is always ON it is in 100% duty cycle and if it is always off it is 0% duty cycle.

Duty Cycle =Turn ON time/ (Turn ON time + Turn OFF time)

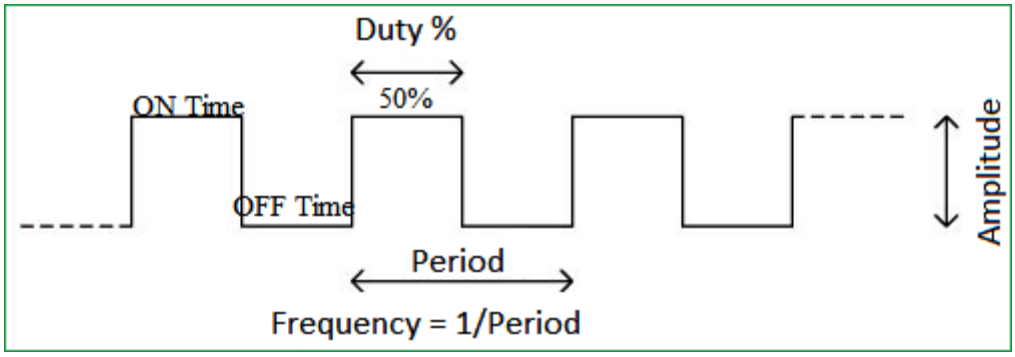


Figure 5.4: Duty cycle of PWM

Frequency of a PWM

The frequency of a PWM signal determines how fast a PWM completes one period. One Period is complete ON and OFF of a PWM signal as shown in the above figure. In our project we set a frequency of 5Hz.

PWM signals can be generated in our PIC Microcontroller by using the CCP (Compare Capture PWM) module. The resolution of our PWM signal is 10-bit, that is for a value of 0 there will be a duty cycle of 0% and for a value of 1024 (2^10) there be a duty cycle of 100%. There are two CCP modules in our PIC MCU (CCP1 And CCP2), this means we can generate two PWM signals on two different pins (pin 17 and 16) simultaneously, in our project we are using CCP1 to generate PWM signals on pin 17.

The following registers are used to generate PWM signals using our PIC MCU:

- CCP1CON (CCP1 control Register)
- T2CON (Timer 2 Control Register)

- PR2 (Timer 2 modules Period Register)
- CCPR1L (CCP Register 1 Low)

The following steps should be taken when configuring the CCP module for PWM operation:

- Set the PWM period by writing to the PR2 register.
- Set the PWM duty cycle by writing to the CCPR1L register and CCP1CON<5:4> bits.
- Make the CCP1 pin an output by clearing the TRISC<2> bit.
- Set the TMR2 prescale value and enable Timer2 by writing to T2CON.
- Configure the CCP1 module for PWM operation.

The timer module's prescaler is set by making the bit T2CKPS0 as high and T2CKPS1 as low the bit TMR2ON is set to start the timer.

Now, we have to set the Frequency of the PWM signal. The value of the frequency has to be written to the PR2 register. The desired frequency can be set by using the below formula[14]:

$$\text{PWM Period} = [(PR2) + 1] * 4 * TOSC * (\text{TMR2 Prescale Value})$$

Rearranging these formulae to get PR2 will give

$$PR2 = (\text{Period} / (4 * Tosc * \text{TMR2 Prescale })) - 1$$

We know that Period = (1/PWM_freq) and Tosc = (1/_XTAL_FREQ). Therefore:

$$PR2 = (\_XTAL\_FREQ/ (PWM\_freq*4*TMR2PRESCALE)) – 1$$

Once the frequency is set this function need not be called again unless and until we need to change the frequency again.

Our PWM signal has 10-bit resolution hence this value cannot be stored in a single register since our PIC has only 8-bit data lines. So, we have to use the other two bits of CCP1CON<5:4> (CCP1X and CCP1Y) to store the last two LSB and then store the remaining 8 bits in the CCPR1L Register.

The PWM duty cycle time can be calculated by using the below formulae:

$$\text{PWM Duty Cycle} = (CCPRIL:CCP1CON<5:4>) * Tosc * (\text{TMR2 Prescale Value})$$

Rearranging these formulae to get value of CCPR1L and CCP1CON will give:

CCPRIL:CCP1Con<5:4> = PWM Duty Cycle / (Tosc * TMR2 Prescale Value)

We also know that Tosc = (1/PWM_freq), hence..

Duty = (((float)duty/1023) * (1/PWM_freq)) / ((1/_XTAL_FREQ) *TMR2PRSCL)

Resolving the above equation will give us:

Duty = ((float)duty/1023) * (_XTAL_FREQ / (PWM_freq*TMR2PRESCALE));

5.4 Keypad

Keypads are widely used input devices being used in various electronics and embedded projects. They are used to take inputs in the form of numbers and alphabets, and feed the same into system for further processing. In this project we are going to interface a 4x3 matrix keypad with PIC18F43K22.

Typically, we use single I/O pin of a microcontroller unit to read the digital signal, like a switch input. In few applications where 9, 12, 16 keys are needed for input purposes, if we add each key in a microcontroller port, we will end up using 16 I/O ports. These 16 I/O ports are not only for reading I/O signals, but they can be used as peripheral connections too, like ADC supports, I2C, SPI connections are also supported by those I/O pins. As those pins are connected with the switches/keys, we can't use them but only as I/O ports. So we can reduce pin counts, which associate 4x3 matrix keys. It will use 8 pins out of which 4 connected in rows and 3 connected in columns, therefore saving 5 pins of the microcontroller.

Figure 5.5: 4x3 keypad

If we see the port there are 7 pins, first 4 from left to right are X1, X2, X3, and X4 are the rows, and last 3 from left to right are Y1, Y2, and Y3 are four columns. If we make

4 rows or X side as output and make them logic low or 1, and make the 4 columns as input and read the keys we will read the switch press when correspondent Y gets 1. Same thing will happen in n x n matrix where n is the number. That can be 3x3, 6x6 etc.

Now just think that 1 is pressed. Then the 1 is situated at X1 row and Y1 column. If X1 is 1, then the Y1 will be 1. By the same way we can sense each key in the X1 row, by sensing column Y1, Y2 and Y3. This thing happens for every switch and we will read the position of the switches in the matrix.

When we press key there are spikes or noise are generated with switch contacts, and due to this multiple switch press happens which is not expected. So, we will first detect the switch press, wait for few milliseconds, again check whether the switch is still pressed or not and if the switch is still pressed, we will accept the switch press finally otherwise not. This is called as de-bouncing of the switches. We will implement this all in our code. The delay is used for the debounce effect, when the switch is still pressed, we will return the value associated with it.

5.5 16x2 Alphanumeric LCD

An LCD (Liquid Crystal Display) screen is an electronic display module and has a wide range of applications. A 16x2 LCD display is very basic module and is very commonly used in various devices and circuits. A 16x2 LCD means it can display 16 characters per line and there are 2 such lines. In this LCD each character is displayed in 5x7 pixel matrix. The 16 x 2 intelligent alphanumeric dot matrix display is capable of displaying 224 different characters and symbols. This LCD has two registers, namely, Command and Data.

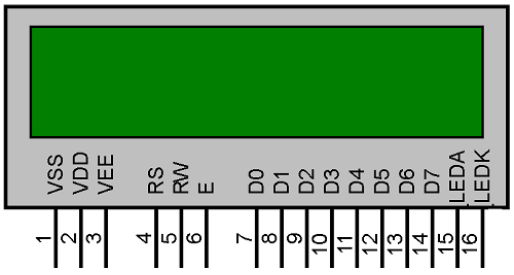


Figure 5.6: LCD 16x2 Display

Command register stores various commands given to the display. Data register stores data to be displayed. The process of controlling the display involves putting the data

that form the image of what we want to display into the data registers, then putting instructions in the instruction register. Contrast of the display can be adjusted by adjusting the potentiometer to be connected across VEE pin.

5.6 Approximation of the Functions

Most microcontrollers have very limited memory, so we have to use this memory very efficiently. One of the significant techniques we have used in our project is approximating non polynomial functions to polynomial of sufficient degree.

Without the approximation of this functions the project would never be possible.

5.6.1 Approximation of θ(x) Function

$$gx^2 - 2gxlcos\theta - 2u^2xsin\theta cos\theta + gl^2cos^2\theta = 0 \qquad (5)$$

g = 9.81 m/s^2 l = 0.1 m u = 3 m/s

Approximating this function was not only for the purpose of efficient memory utilization but also the equation is very difficult to solve for θ.

We took 4 points carefully within the range between 0° and 18°, which are the minimum and maximum values θ by looking the graph eqn. (5) with in the range of firing circle of the cannon. Using these points we approximated eqn.(5) to degree 3 polynomial:

$$\theta(x) = -43.54261x^3 + 72.347453x^2 - 3.9841x - 0.2815$$

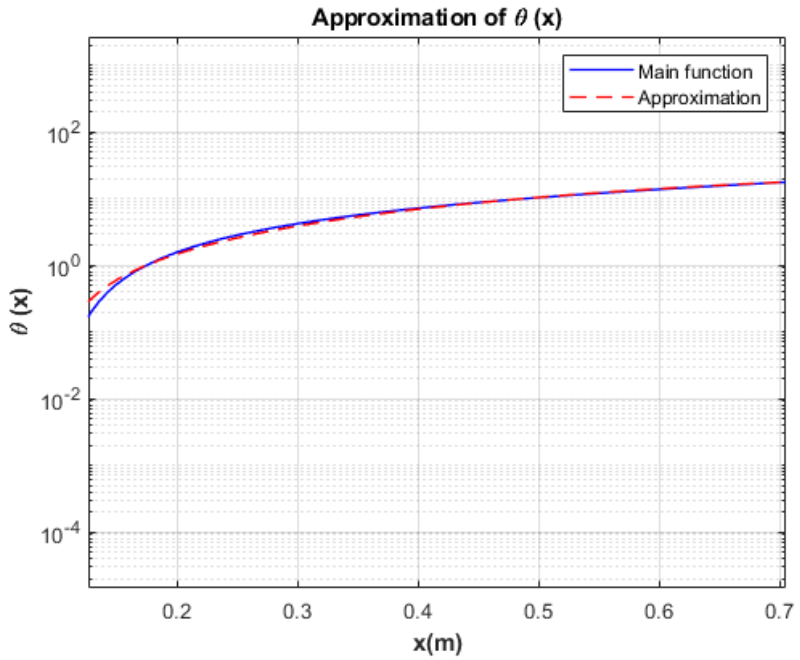


Figure 5.7: Approximation of θ(x) function

As we can see on the figure the two graphs are almost identical except at the beginning, which is do-not-fire zone of the cannon, since it is close distance to the cannon itself. Hence, the approximation can be implemented with very small error.

5.5.2 Approximation of Square Root Function

$$\text{Range}=\sqrt{x^2+y^2}$$

Even if this equation simple it is a huge burden for the microcontroller to solve it with high degree of accuracy. We need to approximate the function with acceptable amount of error.

Again, we carefully chose 4 points within a range between 0.02 and 0.5, which are the minimum and maximum values of the part inside the square root within the firing circle of the cannon. Using these points, we approximated the function to degree 3 polynomial:

$$Y = 3.9939x^3 + 4.7126x^2 - 2.5891x - 0.0915$$

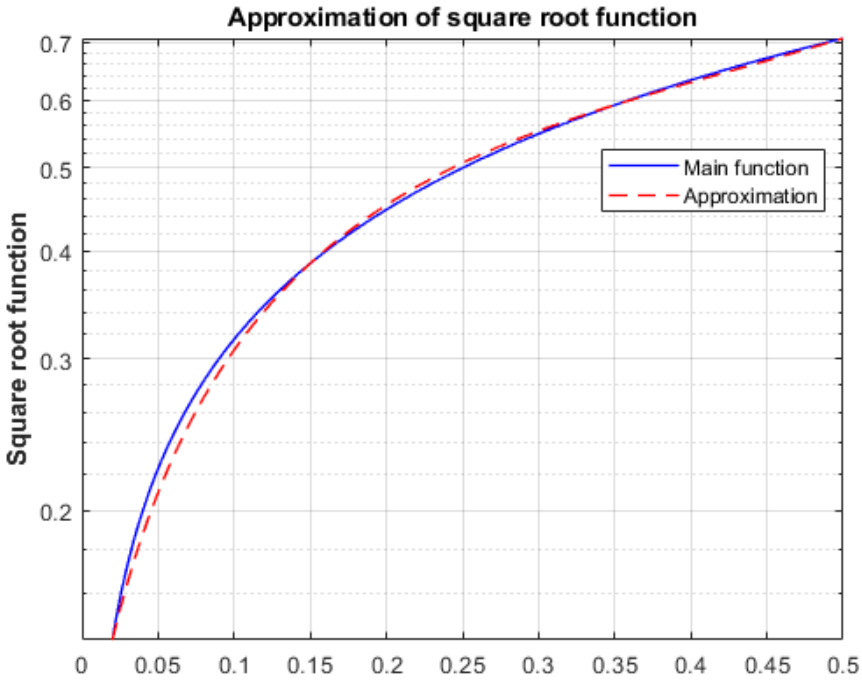


Figure 5.8: Approximation of square root function

As we can see on the graph the approximate function is sufficient for our purpose of calculation. The error is slightly higher at the start of the graph but this would not cause that much of a problem, since the distance is in close range to the cannon there won't be much firing in that range.

5.5.3 Approximation of Tan Inverse Function

We need whole range of tan inverse function in our project so approximating it with polynomial will not be a good idea to address whole function from negative infinity to positive infinity.

We used interpolation technique for this function. First, we took 9 points carefully and for any value between this value we used linear interpolation. For values above the range, we gave the value of 90°. For negative values just multiply it with -1.

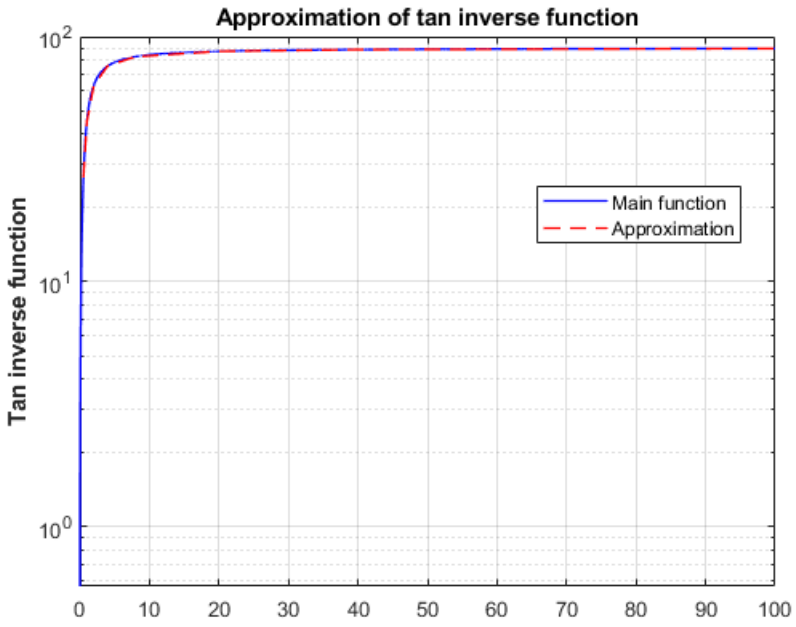


Figure 5.9: Approximation of tan inverse function.

As we can see from the graph the two functions are almost identical. The error introduced by the approximation within the acceptable range.

Chapter 6

6. Results and Discussion

6.1 Results

We implemented the project on Proteus 8.13 and for the physics simulation of the project we used MATLAB R2019a. The code for the microcontroller and physics simulation is found on the Appendix.

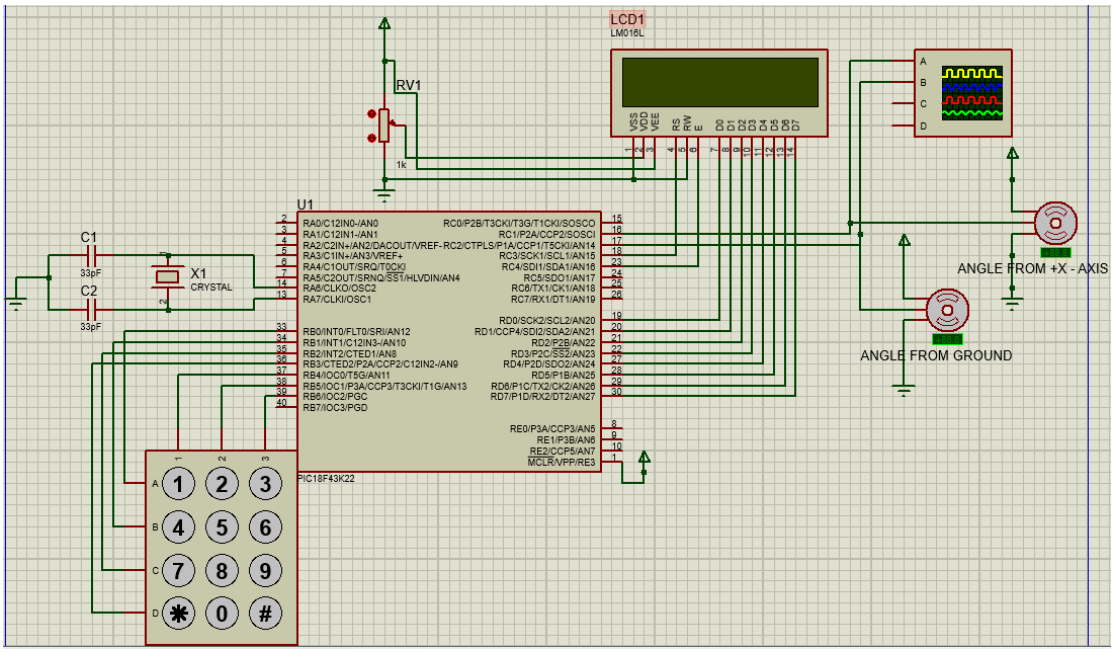


Figure 6.1: Proteus simulation block diagram of the project.

Then we tested the project on different coordinates, but for discussion purpose let's take 4 points from the 4 quadrants of the system.

Target on 1st Quadrant: x = 0.5 m and y = 0.5 m

We inserted the coordinates using the keypad interface and the microcontroller calculated the angles and their respective PWM duty cycle. Then using this duty cycle and the period 0.2 sec it gives the servos their respective PWM signal. Then the servos gave angle of rotation as output.

Then to test how much accurate is the result, we tested the result using physics simulation on MATLAB as shown on Figure 6.3.

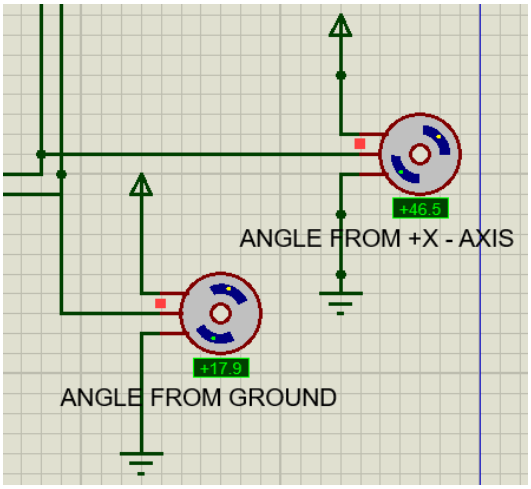


Figure 6.2: Firing angles for x = 0.5m and y = 0.5m.

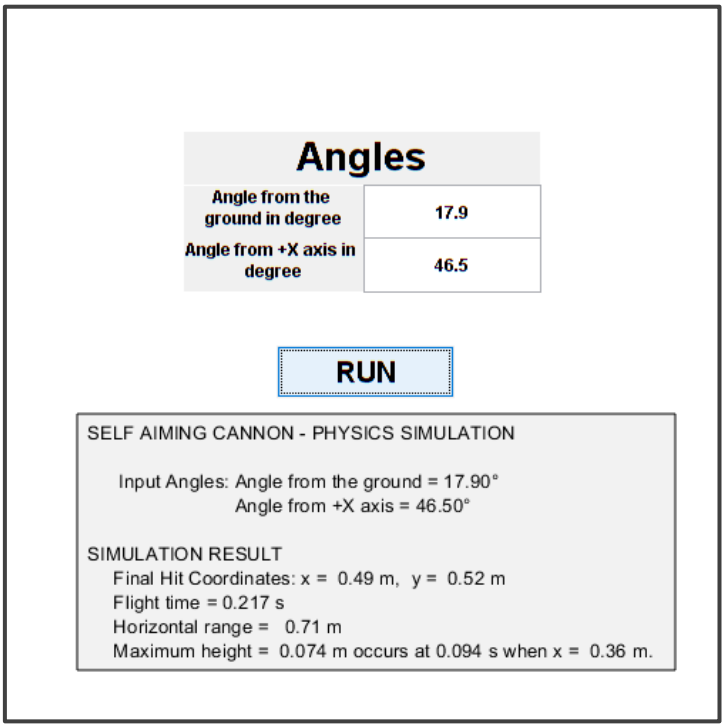


Figure 6.3: Physics simulation result for x = 0.5 m and y = 0.5 m.

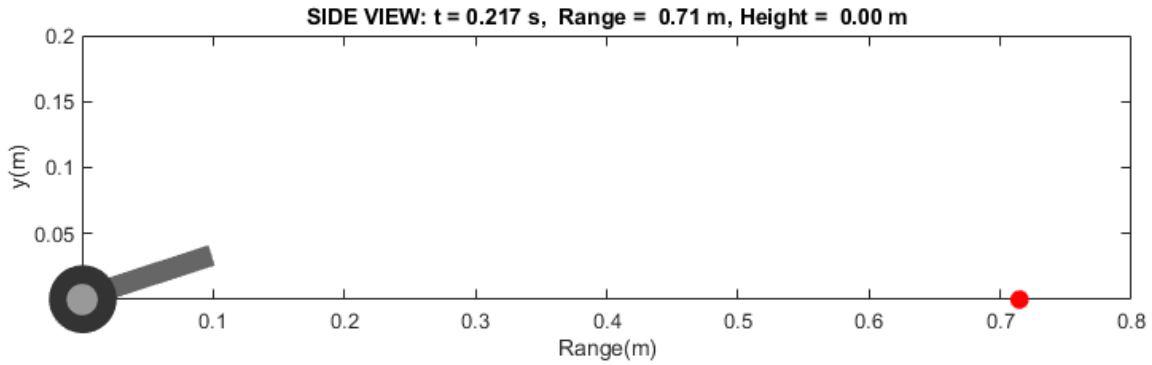


Figure 6.4: Side view of the physics simulation.

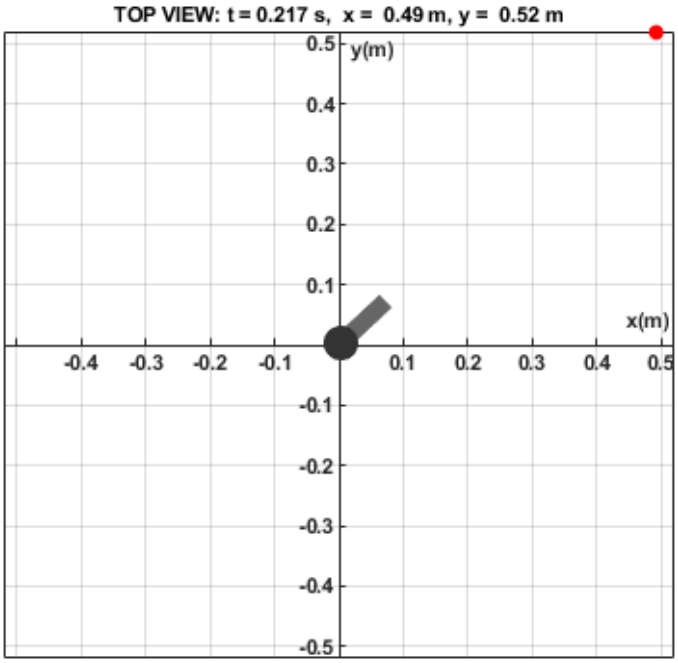


Figure 6.5: Top view of physics simulation for x = 0.5m and y = 0.5m.

Both the top view graph and final hit coordinate output shows the hit is very near to the target. To check if the error is due to approximation of the functions or the system itself we prepared another MATLAB physics simulation using the approximated function.

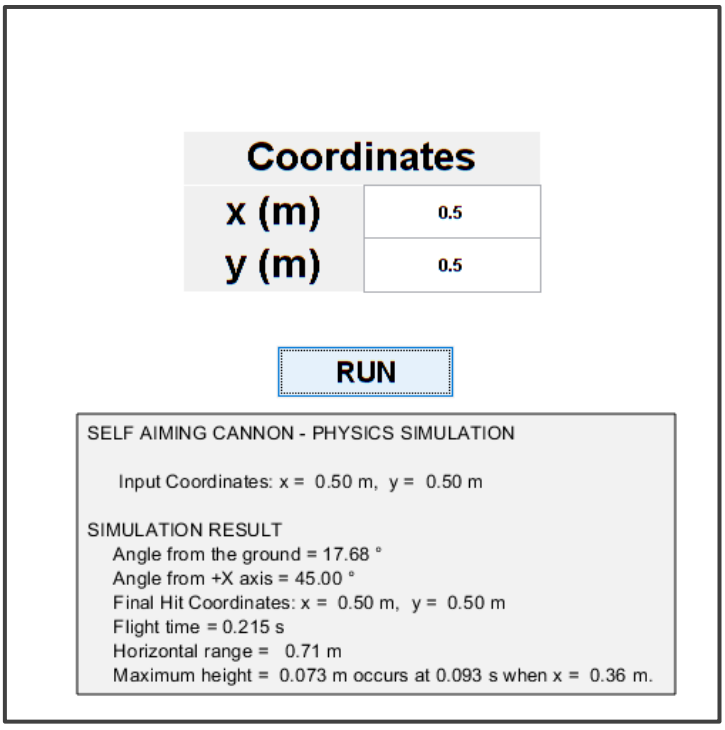


Figure 6.6: Physics simulation using approximated functions (x = 0.5m and y = 0.5m).

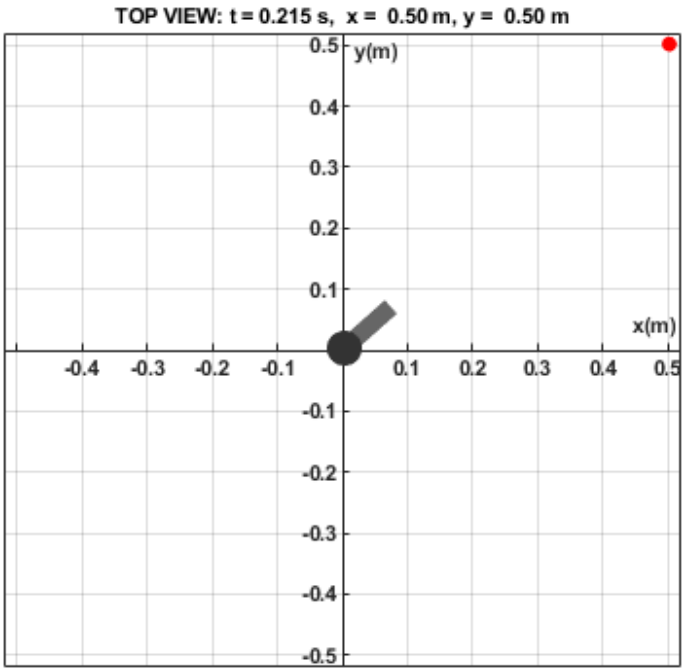


Figure 6.7: Top view simulation of the approximated functions.

From the above figures, we can see that the approximation of the function has made no significant error on the calculation of the firing angles.

Target on 2nd Quadrant: x = -0.5 m and y = 0.3 m

We inserted the coordinates using the keypad interface and the servos gave angle of rotation as output.

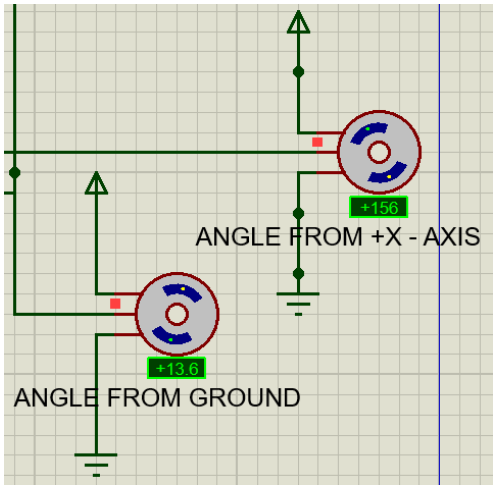


Figure 6.8: Firing angles for x = -0.5m and y = 0.3m.

Then to test how much accurate is the result, we tested the result using physics simulation on MATLAB.

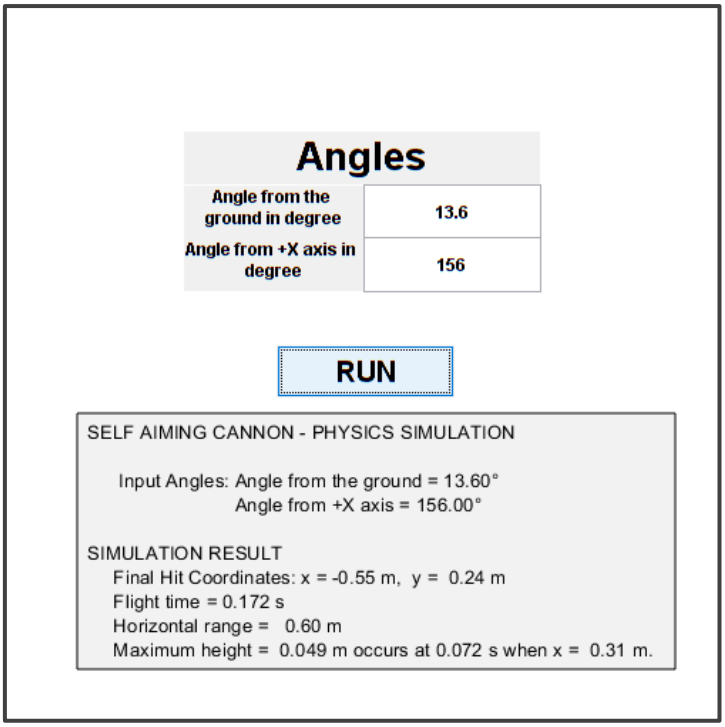


Figure 6.9: Physics simulation result for x = -0.5 m and y = 0.3 m.

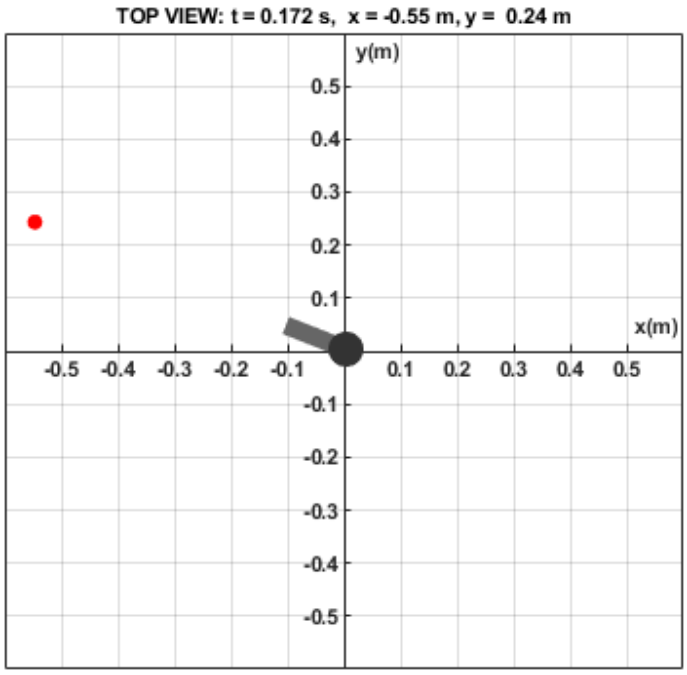


Figure 6.10: Top view of physics simulation for x = -0.5m and y = 0.3m.

This test shows a slightly higher error than the first one. Again, to check the error if it is because or of the approximated functions of the system itself, we tested it with another physics simulation.

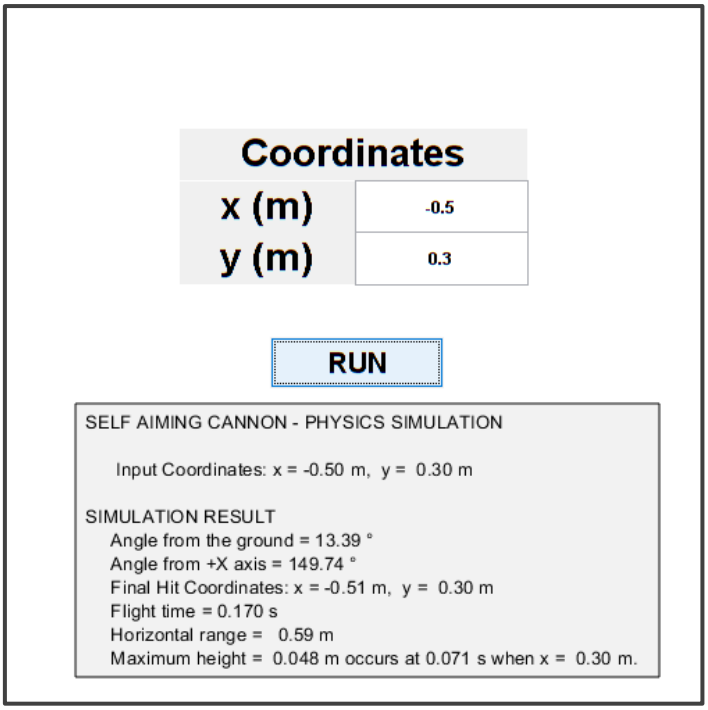


Figure 6.11: Physics simulation using approximated functions (x=-0.5m and y=0.3m).

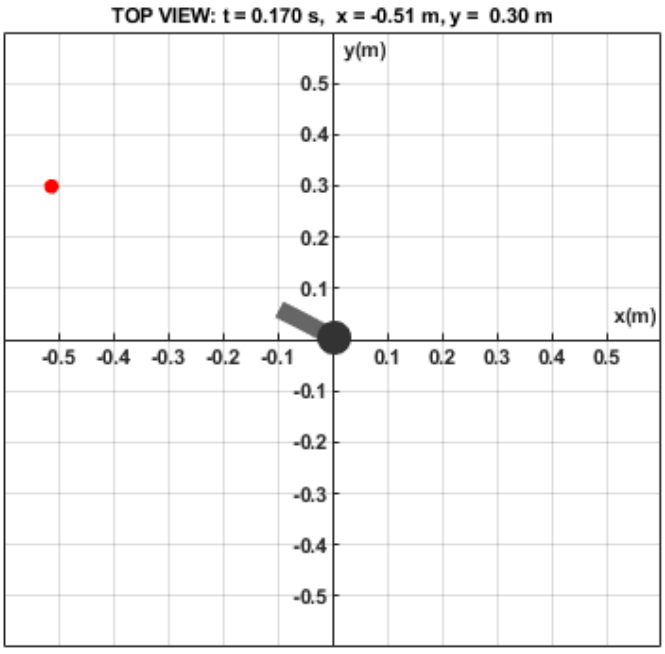


Figure 6.12: Top view simulation of the approximated functions(x=-0.5m & y=0.3m).

Again, from the above figures, we can see that the approximation of the function has made no significant error on the calculation of the firing angles.

Target on 3rd Quadrant: x = -0.2 m and y = -0.5 m

We inserted the coordinates using the keypad interface and the servos gave angle of rotation as output.

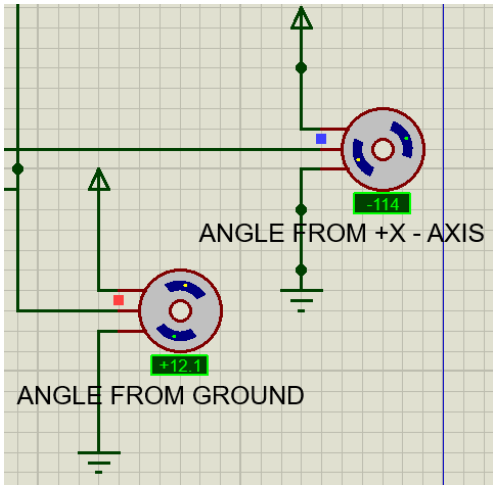


Figure 6.13: Firing angles for x = -0.2m and y = 0.5m.

Then to test how much accurate is the result, we tested the result using physics simulation on MATLAB.

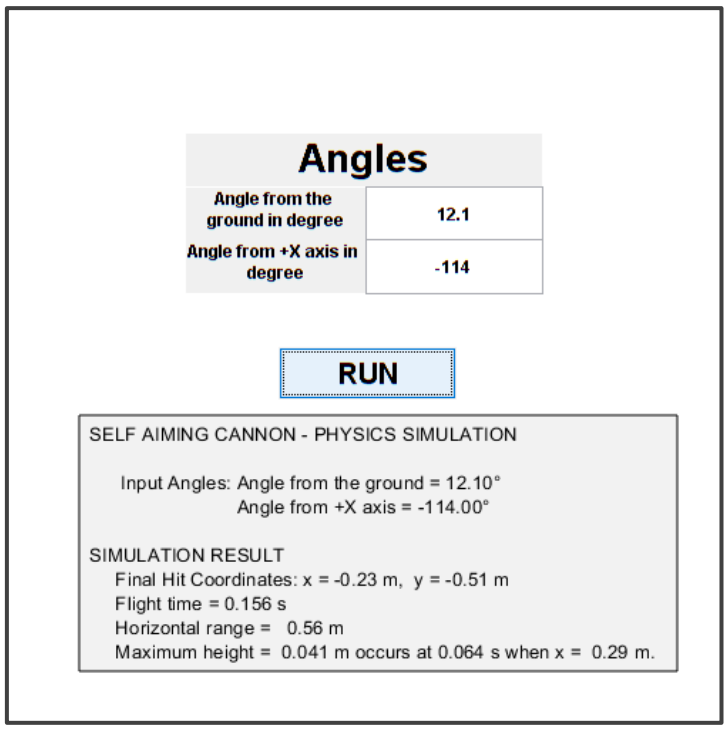


Figure 6.14: Physics simulation result for x = -0.2 m and y = -0.5 m.

The figure below shows the hit is very near to the target. Again, to check the error if it is because of the approximated functions or of the system itself, we tested it with another physics simulation.

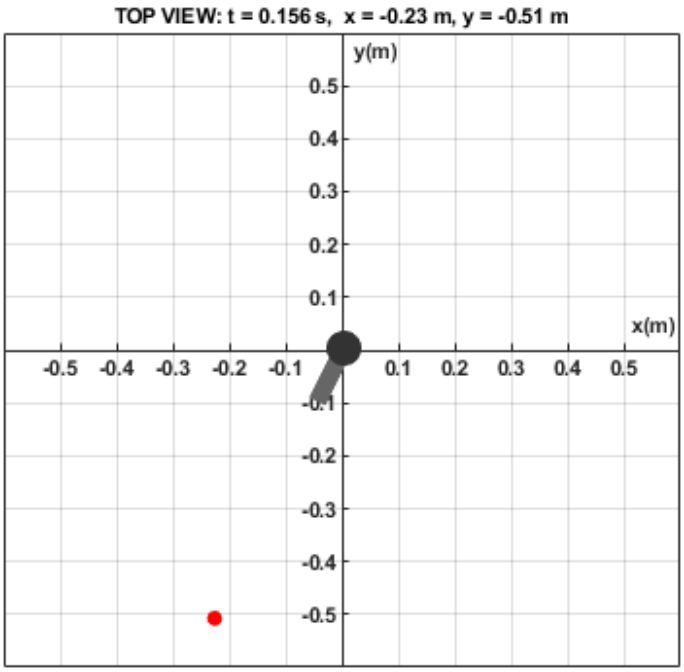


Figure 6.15: Top view of physics simulation for x = -0.2m and y = -0.5m.

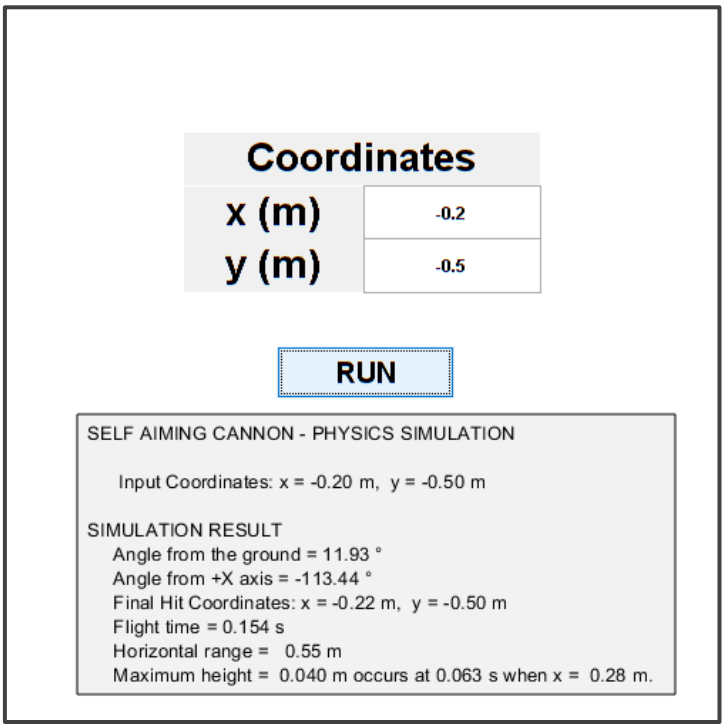


Figure 6.16: Physics simulation using approximated functions (x=-0.2m & y= 0.5m).

Again, from Figure 6.16 and 6.17, we can see that the approximation of the function has made no significant error on the calculation of the firing angles, means, the error is due to the system itself, in fact, due to the servo motor.

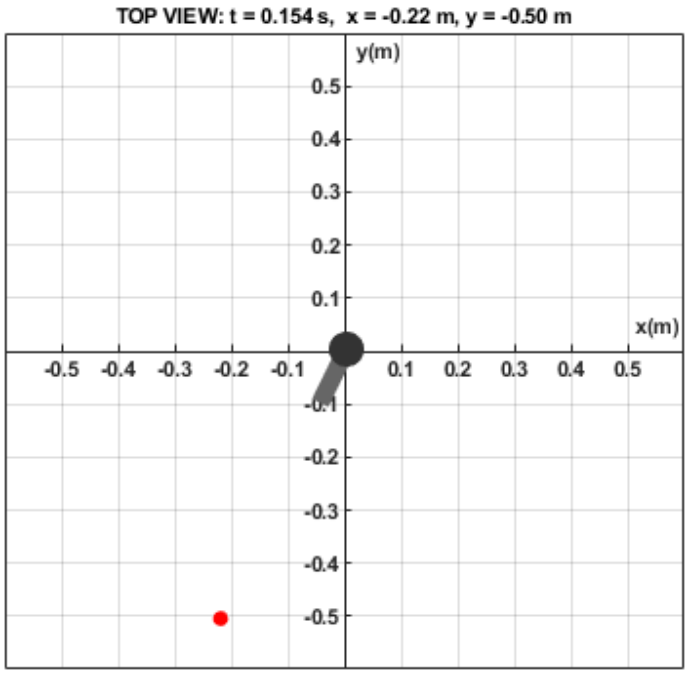


Figure 6.17: Top view simulation of the approximated functions(x=-0.2m & y= 0.5m)

Target on 4th Quadrant: x = 0.5 m and y = -0.5 m

We inserted the coordinates using the keypad interface and the servos gave angle of rotation as output.

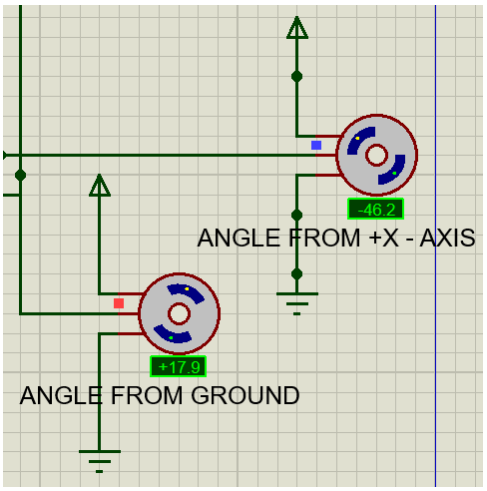


Figure 6.18: Firing angles for x = 0.5m and y = -0.5m.

Then to test how much accurate is the result, we tested the result using physics simulation on MATLAB. In which we will just enter the angles we got from the microcontroller and the physics simulator will simulate how the projectile will hit it's it target.

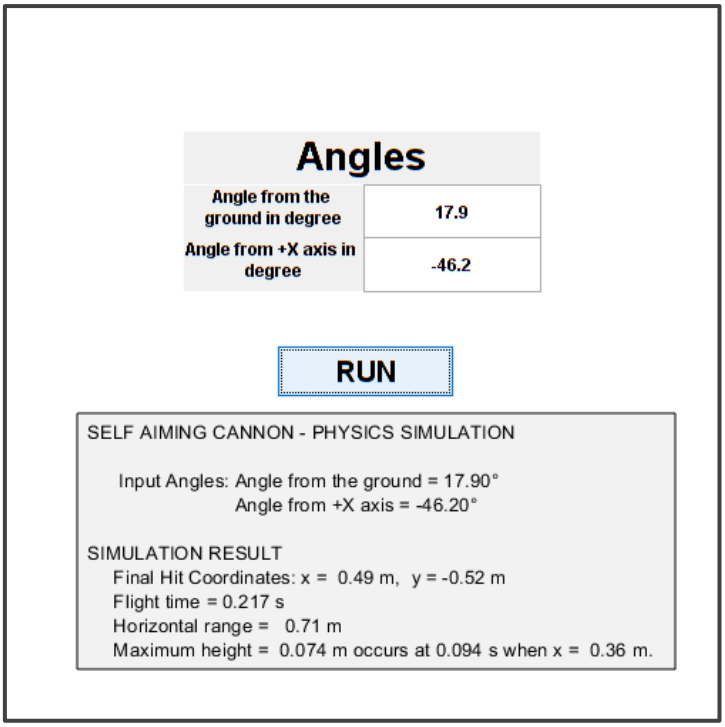


Figure 6.19: Physics simulation result for x = 0.5 m and y = -0.5 m.

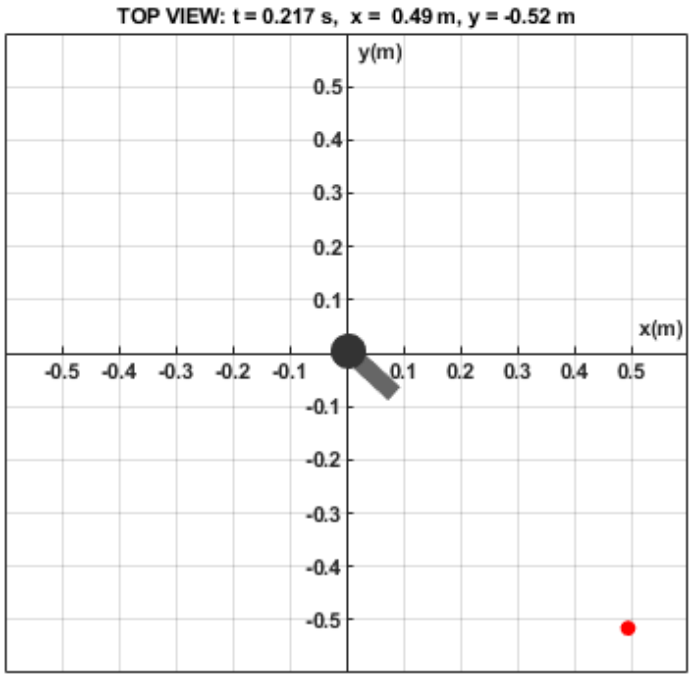


Figure 6.20: Top view of physics simulation for x = 0.5m and y = -0.5m.

The figure shows the hit is very near to the target. Again, to check the error if it is because of the approximated functions or of the system itself, we tested it with another physics simulation.

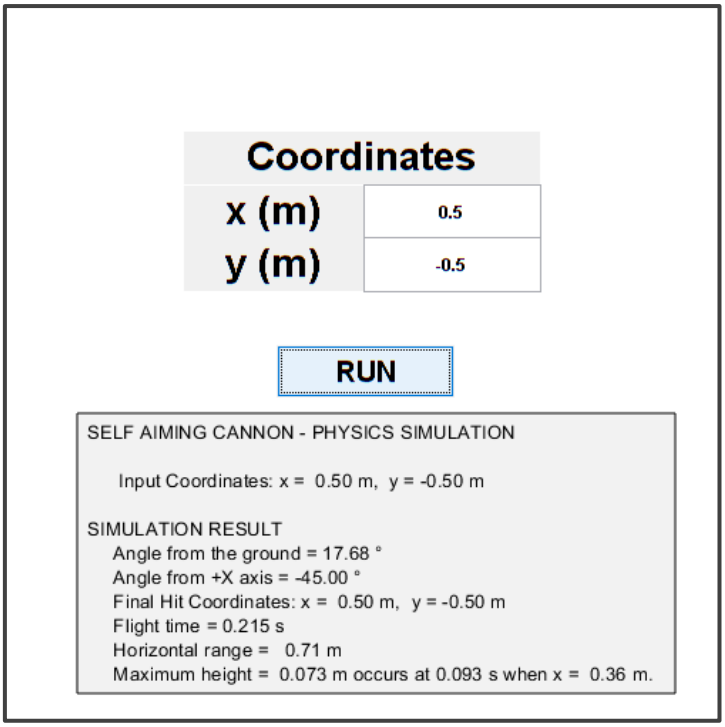


Figure 6.21: Physics simulation using approximated functions (x=0.5m and y=-0.5m).

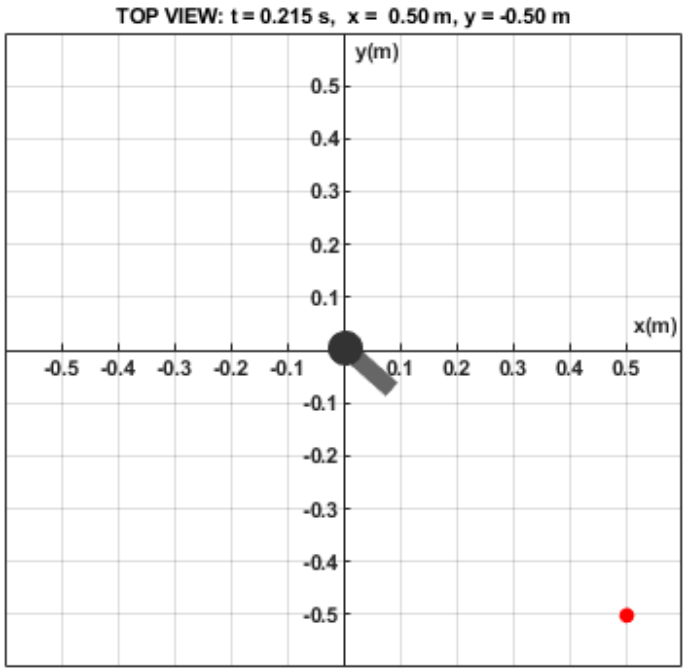


Figure 6.22: Top view simulation of the approximated functions(x=0.5m & y=-0.5m).

Again, from the above figures, we can see that the approximation of the function has made no significant error on the calculation of the firing angles.

6.2 Discussion

We have tested the project on all 4 quadrants and we have found it very satisfactory. There are some acceptable errors between the actual hit point and target point. We can conclude that these errors are not mainly because of the approximation of the functions in the system modeling. In fact, the errors are introduced due to the stepwise movement of the servo motor, which means there are some angle ranges which are not covered by the servo motor.

The approximation of the functions was very helpful in the development of the project. Without it our microcontroller could have not take the function that, now, it is performing. By using this method and others we were able to utilize the microcontroller effectively or 100%, as all microcontroller project should be.

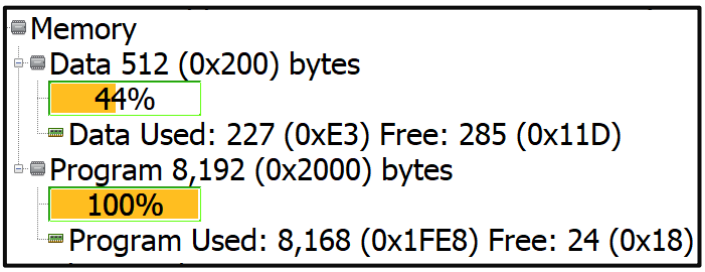


Figure 6.23: Memory usage of the microcontroller after the final compilation of the project.

Chapter 7

7. Conclusions and Recommendations

7.1. Conclusion

The use of technology is necessary in military. Especially, if the technology has any possibility of saving one soldier's life. The firing system on cannon uses the foot of the cannon crew, which is very dangerous with the position of the crew on top of the cannon when firing. So, a firing system that can be remotely controlled by a computer is required. The design of the cannon firing control system uses a keypad as the firing command input. The project was able to hit it's targets within acceptable range by taking just two coordinate inputs. The project is proved to be accurate on all 4 quadrants.

The project utilized the PIC microcontroller effectively. To accomplish this approximation of some non-polynomial functions was necessary.

7.2. Recommendations

Since we would like someone to reach our delimitations. We recommend more projects to be done:

- Including the effect of wind
- On non-flat surface
- Variable mass cannon firing
- Forced projectile

References

[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] 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.

[6] 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.

[7] 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

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

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

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

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

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

[13] Norman S. Nise. (2011). "CONTROL SYSTEMS ENGINEERING,Sixth Edition". John Wiley & Sons, Inc.

[14] Microchip Technology Inc. (2016). "PIC Microcontroller Datasheet".

Appendices

Appendix I – MATLAB Physics Simulation Code

```
% Self Aiming Cannon Simulaton
function CannonDeg()
clc
figure(1);
h = gcf;
set(h,'Color',[0.5 0.5 0.65],'Name',...
    'Self Aiming Cannon - Physics Simulaton', ...
    'NumberTitle','off','Position',[10 100 500 500]);
ha = annotation('textbox','String',...
    sprintf('Self Aiming Cannon\n Physics Simulaton'), ...
        'Color','w','Position',[0.23 0.98 0.9 0.01]);
ha.FontSize=20; ha.FontWeight='bold'; ha.EdgeColor='none';
handles = generateUI(h);
xc = 0.38;  yc=0.88;  yc=yc-0.43;  wt=0.25; ht=0.075;
uicontrol(h,'Style','PushButton','String','RUN', ...
    'Units','normalized','Position',[xc yc wt ht], ...
    'FontWeight','bold','FontSize',16, ...
    'Callback',{@runSimulation,handles});
function runSimulation(hObj,event,handles)
hOut = handles.hOut;
h_theta = handles.h_theta;     h_phi = handles.h_phi;
sTheta = get(h_theta,'String');   theta = str2double(sTheta);
sPhi = get(h_phi,'String');      phi = str2double(sPhi);
g = 9.81; l = 0.1; V0 = 3; h = l*sind(theta);
tmax = 1;   dt = 0.001;     tp = (0:dt:tmax);   nt=length(tp);
vx = zeros(size(tp));       vx(1) = V0*cosd(theta);
vy = zeros(size(tp));       vy(1) = V0*sind(theta);
sx = zeros(size(tp));       sx(1) = l*cosd(theta);
sy = zeros(size(tp));       sy(1) = h;
tx = zeros(size(tp));       tx(1) = sx(1)*cosd(phi);
ty = zeros(size(tp));       ty(1) = sx(1)*sind(phi);
htop=0; ttop=0; xtop=0;
for i=2:nt
    im = i-1;
    dvy = -g*dt;
    vx(i) = vx(im);
    vy(i) = vy(im) + dvy;
    sx(i) = sx(im) + 0.5*(vx(im)+vx(i))*dt;
    sy(i) = sy(im) + 0.5*(vy(im)+vy(i))*dt;
    tx(i) = sx(i)*cosd(phi);
    ty(i) = sx(i)*sind(phi);
    if sy(i)>htop   %saves the max height values
        htop = sy(i);
        ttop = tp(i);
        xtop = sx(i);
    end
    if sy(i)<=0.0001  %check if it touches the ground
        sy(i) = 0;
        iend = i;
        break
    end
end
vx = vx(1:iend);    vy = vy(1:iend);
sx = sx(1:iend);    sy = sy(1:iend);        tp = tp(1:iend);
```

```
tx = tx(1:iend);    ty = ty(1:iend);

if (sx(iend) < 0.2) || (abs(tx(iend))>0.6) || (abs(ty(iend))>0.6)
    figure(2); oorf = gcf; clf(oorf); set (oorf,...
     'Name','Out of Range','NumberTitle','off','Color','w','Position',...
     [725 475 400 150]);
    oor = annotation(oorf,'textbox','String','Out of Range!', ...
        'Color','r','Position',[0.3 0.45 0.9 -2]);
    oor.FontSize=20; oor.FontWeight='bold'; oor.EdgeColor='none';
else
sOut = 'SELF AIMING CANNON - PHYSICS SIMULATION';
sOut = sprintf('%s\n\n      Input Angles: Angle from the ground = %5.2f%c'...
    ,sOut,theta,char(176));
sOut = sprintf('%s\n       Angle from +X axis = %5.2f%c'...
    ,sOut,phi,char(176));
sOut = sprintf('%s\n\nSIMULATION RESULT',sOut);
sOut = sprintf('%s\n Final Hit Coordinates: x = %5.2f m, y = %5.2f m'...
    ,sOut,tx(iend),ty(iend));
sOut = sprintf('%s\n     Flight time = %5.3f s',sOut,tp(end));
sOut = sprintf('%s\n     Horizontal range = %6.2f m',sOut,sx(end));
sOut = sprintf('%s\n     Maximum height = %6.3f m',sOut,htop);
sOut = sprintf('%s occurs at %5.3f s when x = %5.2f m.'...
,sOut,ttop,xtop);
set(hOut,'String',sOut);
figure(3); topView = gcf; set(topView,'Name','Top View','NumberTitle','off');
% animate the trajectory
for i =0:10
    clf(topView);
    set (topView,'Color','w','Position',[700 -20 450 400]);
    axis([-0.6 0.6 -0.6 0.6]);
    ax = gca;
    ax.XAxisLocation = 'origin';
    ax.YAxisLocation = 'origin';
    ax.YTick = -0.6:0.1:0.6;
    ax.XTick = -0.6:0.1:0.6;
    ax.FontSize = 6;
    ax.FontWeight = 'bold';
    th = phi*i/10;
    tcx = 0.52; tcy = 0.52; tcdx = 0.075; tcdy = tcdx;
    annotation(topView,'line',[tcx tcx+tcdx*cosd(th)],...
[tcy tcy+tcdy*sind(th)],...
        'Color',[0.4 0.4 0.4],'LineWidth',7);
    annotation(topView,...
'ellipse',[tcx-0.02 tcy-0.02*450/400 0.04 0.04*450/400]...
        ,'FaceColor',[.2 .2 .2],'LineStyle','none');
    pause(0.001)
end
figure(2); sideView = gcf;
set(sideView,'Name','Side View','NumberTitle','off');
set (sideView,'Color','w','Position',[510 465 850 225]);
axis([0 0.8 0 0.2]);
for i = 0:10
clf(sideView);
th = theta*i/10;
cx = 0.13; cy = 0.18; cdx = 0.1; cdy = cdx*850/225; cd = 0.2;
annotation(sideView,'line',[cx cx+cdx*cosd(th)],[cy cy+cdy*sind(th)],...
    'Color',[0.4 0.4 0.4],'LineWidth',10);
annotation(sideView,'ellipse',[cx-0.025 cy-0.1 0.05 0.05*850/225],...
    'FaceColor',[.2 .2 .2],'LineStyle','none');
```

```
annotation(sideView,'ellipse',[cx-0.012 cy-0.05 0.023 0.023*850/225],...
    'FaceColor',[.6 .6 .6],'LineStyle','none');
axis([0 0.8 0 0.2]);
xlabel('x, m'); ylabel('y, m');
pause(0.001)
end
pause(0.5)
for j=1:10:length(tp)+10
    i = j;
    if i > length(tp)
        i = length(tp);
    end
    figure(2);
    clf(sideView);
    annotation(sideView,'line',[cx cx+cdx*cosd(theta)],...
        [cy cy+cdy*sind(theta)],'Color',[0.4 0.4 0.4],'LineWidth',10);
    annotation(sideView,'ellipse',[cx-0.025 cy-0.1 0.05 0.05*850/225],...
        'FaceColor',[.2 .2 .2]...
    ,'LineStyle','none');
    annotation(sideView,'ellipse',[cx-0.012 cy-0.05 0.023 0.023*850/225]...
        ,'FaceColor',[.6 .6 .6]...
    ,'LineStyle','none');
%     plot(sx,sy,'b-'); hold on % on/off trajectary
    plot(sx(i),sy(i),'r.','MarkerSize',30);
    axis([0 0.8 0 0.2]);
    xlabel('Range(m)'); ylabel('y(m)');
    title(sprintf('SIDE VIEW: t =%6.3f s,  Range =%6.2f m, Height =%6.2f m'...
        ,tp(i),sx(i),sy(i)));
    figure(3);
    clf(topView);
    annotation(topView,'line',[tcx tcx+tcdx*cosd(phi)],...
        [tcy tcy+tcdy*sind(phi)],'Color',[0.4 0.4 0.4],'LineWidth',7);
    annotation(topView,'ellipse',...
    [tcx-0.02 tcy-0.02*450/400 0.04 0.04*450/400]...
        ,'FaceColor',[.2 .2 .2],'LineStyle','none');
%     plot(tx,ty,'b-'); hold on; %on/off trajectary
    plot(tx(i),ty(i),'r.','MarkerSize',20);
    axis([-0.6 0.6 -0.6 0.6]);
    ax = gca;
    ax.XAxisLocation = 'origin';
    ax.YAxisLocation = 'origin';
    ax.YTick = -0.6:0.1:0.6;
    ax.XTick = -0.6:0.1:0.6;
    ax.FontSize = 6;
    ax.FontWeight = 'bold';
    xlabel('x(m)'); ylabel('y(m)');
    title(sprintf('TOP VIEW: t =%6.3f s,  x =%6.2f m, y =%6.2f m'...
        ,tp(i),tx(i),ty(i)));
    pause(0.001)
end
grid
end
function handles = generateUI(h
xc = 0.25;  yc=0.75;  wt=0.5; ht=0.075;
uicontrol(h,'Style','text','String','Angles', ...
    'FontWeight','bold','FontSize',20, ...
    'Units','normalized','Position',[0.25 yc wt ht]);
xc = 0.25;  yc=yc-0.075;  wt=0.25; ht=0.075;
uicontrol(h,'Style','text','String',...
```

```
'Angle from the ground in degree', ...
    'FontWeight','bold','FontSize',9, ...
    'Units','normalized','Position',[xc yc wt+0.002 ht]);
handles.h_theta = uicontrol(h,'Style','edit','String','15', ...
    'FontWeight','bold','FontSize',9, ...
    'Units','normalized','Position',[xc+wt+0.002 yc-0.001 wt ht+0.003]);
xc = 0.25;  yc=yc-0.075;  wt=0.25; ht=0.075;
uicontrol(h,'Style','text','String','Angle from +X axis in
degree','FontWeight','bold','FontSize',9, ...
    'Units','normalized','Position',[xc yc wt+0.002 ht]);
handles.h_phi = uicontrol(h,'Style','edit','String','45', ...
    'FontWeight','bold','FontSize',9, ...
    'Units','normalized','Position',[xc+wt+0.002 yc-0.001 wt ht+0.003]);
handles.hOut = annotation('TextBox',[0.1 0.08 0.8 0.35], ...
        'Interpreter','tex', ...
        'Background',[0.95 0.95 0.95], ...
        'String',' ','FitBoxToText','on');
```

Appendix II – MATLAB Physics Simulation by Approximate Functions

```
% Self Aiming Cannon Simulaton
function CannonXY()
clc
figure(1);
h = gcf;
set(h,'Color',[0.5 0.5 0.65],'Name','Self Aiming Cannon - Physics Simulaton',
...
    'NumberTitle','off','Position',[10 100 500 500]);
ha = annotation('textbox','String',sprintf(...
    'Self Aiming Cannon\n Physics Simulaton'), ...
        'Color','w','Position',[0.23 0.98 0.9 0.01]);
ha.FontSize=20; ha.FontWeight='bold'; ha.EdgeColor='none';
handles = generateUI(h);
xc = 0.38;  yc=0.88;  yc=yc-0.43;  wt=0.25; ht=0.075;
uicontrol(h,'Style','PushButton','String','RUN', ...
    'Units','normalized','Position',[xc yc wt ht], ...
    'FontWeight','bold','FontSize',16, ...
    'Callback',{@runSimulation,handles});
function runSimulation(hObj,event,handles)
hOut = handles.hOut;
h_x = handles.h_x;     h_y = handles.h_y;
sX = get(h_x,'String');             x = str2double(sX);
sY = get(h_y,'String');             y = str2double(sY);
inr = (x^2 + y^2); % range
rng = 3.9939*inr^3 - 4.7126*inr^2 + 2.5891*inr + 0.0915;% approximated function
theta = -43.54261*rng^3 + 72.347453*rng^2 - 3.9841*rng - 0.2815218; %
approximated function
g = 9.81; l = 0.1; V0 = 3; h = l*sind(theta);
d = y/x;
if d<0
    phi = -1*tani(-1*d);
else
    phi = tani(d);
end
if x < 0 && y >= 0
    phi = phi + 180;
```

```
elseif x < 0 && y < 0
    phi = phi - 180;
end
tmax = 1;   dt = 0.001;     tp = (0:dt:tmax);       nt=length(tp);
vx = zeros(size(tp));       vx(1) = V0*cosd(theta);
vy = zeros(size(tp));       vy(1) = V0*sind(theta);
sx = zeros(size(tp));       sx(1) = l*cosd(theta);
sy = zeros(size(tp));       sy(1) = h;
tx = zeros(size(tp));       tx(1) = sx(1)*cosd(phi);
ty = zeros(size(tp));       ty(1) = sx(1)*sind(phi);
htop=0; ttop=0; xtop=0;
for i=2:nt
    im = i-1;
  % dvx = K1*real(vx(im)^n)*dt; drag force is negligible
    dvy = -g*dt;
    vx(i) = vx(im);
    vy(i) = vy(im) + dvy;
    sx(i) = sx(im) + 0.5*(vx(im)+vx(i))*dt;
    sy(i) = sy(im) + 0.5*(vy(im)+vy(i))*dt;
    tx(i) = sx(i)*cosd(phi);
    ty(i) = sx(i)*sind(phi);
    if sy(i)>htop   %saves the max height values
        htop = sy(i);
        ttop = tp(i);
        xtop = sx(i);
    end
    if sy(i)<=0.0001  %check if it touches the ground
        sy(i) = 0;
        iend = i;
        break
    end
end
vx = vx(1:iend);    vy = vy(1:iend);
sx = sx(1:iend);    sy = sy(1:iend);        tp = tp(1:iend);
tx = tx(1:iend);    ty = ty(1:iend);

if (rng < 0.2) || (abs(x)>0.5) || (abs(y)>0.5)
    figure(2); oorf = gcf; clf(oorf); set (oorf,'Name','Out of
Range','NumberTitle','off','Color','w','Position',[725 475 400 150]);
    oor = annotation(oorf,'textbox','String','Out of Range!', ...
        'Color','r','Position',[0.3 0.45 0.9 -2]);
    oor.FontSize=20; oor.FontWeight='bold'; oor.EdgeColor='none';
else
sOut = 'SELF AIMING CANNON - PHYSICS SIMULATION';
sOut = sprintf('%s\n\n      Input Coordinates: x = %5.2f m,  y = %5.2f
m',sOut,x,y);
sOut = sprintf('%s\n\nSIMULATION RESULT',sOut);
sOut = sprintf('%s\n     Angle from the ground = %5.2f %c\n     Angle from +X
axis = %5.2f %c',...
    sOut,theta,char(176),phi,char(176));
sOut = sprintf('%s\n     Final Hit Coordinates: x = %5.2f m,  y = %5.2f
m',sOut,tx(iend),ty(iend));
sOut = sprintf('%s\n     Flight time = %5.3f s',sOut,tp(end));
sOut = sprintf('%s\n     Horizontal range = %6.2f m',sOut,sx(end));
sOut = sprintf('%s\n     Maximum height = %6.3f m',sOut,htop);
sOut = sprintf('%s occurs at %5.3f s when x = %5.2f m.',sOut,ttop,xtop);
set(hOut,'String',sOut);
figure(3); topView = gcf; set(topView,'Name','Top View','NumberTitle','off');
for i =0:10
```

```
    clf(topView);
    set (topView,'Color','w','Position',[700 -20 450 400]);
    axis([-0.6 0.6 -0.6 0.6]);
    ax = gca;
    ax.XAxisLocation = 'origin';
    ax.YAxisLocation = 'origin';
    ax.YTick = -0.6:0.1:0.6;
    ax.XTick = -0.6:0.1:0.6;
    ax.FontSize = 6;
    ax.FontWeight = 'bold';
    th = phi*i/10;
    tcx = 0.52; tcy = 0.52; tcdx = 0.075; tcdy = tcdx;
    annotation(topView,'line',[tcx tcx+tcdx*cosd(th)],[tcy
tcy+tcdy*sind(th)],'Color',[0.4 0.4 0.4],'LineWidth',7);
    annotation(topView,'ellipse',[tcx-0.02 tcy-0.02*450/400 0.04
0.04*450/400],'FaceColor',[.2 .2 .2],'LineStyle','none');
    pause(0.001)
end
figure(2); sideView = gcf; set(sideView,'Name','Side View','NumberTitle','off');
set (sideView,'Color','w','Position',[510 465 850 225]);
axis([0 0.8 0 0.2]);
for i = 0:10
clf(sideView);
th = theta*i/10;
cx = 0.13; cy = 0.18; cdx = 0.1; cdy = cdx*850/225; cd = 0.2;
annotation(sideView,'line',[cx cx+cdx*cosd(th)],[cy cy+cdy*sind(th)],...
    'Color',[0.4 0.4 0.4],'LineWidth',10);
annotation(sideView,'ellipse',[cx-0.025 cy-0.1 0.05
0.05*850/225],'FaceColor',[.2 .2 .2]...
    ,'LineStyle','none');
annotation(sideView,'ellipse',[cx-0.012 cy-0.05 0.023
0.023*850/225],'FaceColor',[.6 .6 .6]...
    ,'LineStyle','none');
axis([0 0.8 0 0.2]);
xlabel('x, m'); ylabel('y, m');
pause(0.001)
end
pause(0.5)
for j=1:10:length(tp)+10
    i = j;
    if i > length(tp)
        i = length(tp);
    end
    figure(2);
    clf(sideView);
    annotation(sideView,'line',[cx cx+cdx*cosd(theta)],[cy
cy+cdy*sind(theta)],'Color',[0.4 0.4 0.4],'LineWidth',10);
    annotation(sideView,'ellipse',[cx-0.025 cy-0.1 0.05
0.05*850/225],'FaceColor',[.2 .2 .2],'LineStyle','none');
    annotation(sideView,'ellipse',[cx-0.012 cy-0.05 0.023
0.023*850/225],'FaceColor',[.6 .6 .6],'LineStyle','none');
%     plot(sx,sy,'b-'); hold on %on/off trajectary
    plot(sx(i),sy(i),'r.','MarkerSize',30);
    axis([0 0.8 0 0.2]);
    xlabel('Range(m)'); ylabel('y(m)');
    title(sprintf('SIDE VIEW: t =%6.3f s,  Range =%6.2f m, Height =%6.2f
m',tp(i),sx(i),sy(i)));
    figure(3);
    clf(topView);
```

```
    annotation(topView,'line',[tcx tcx+tcdx*cosd(phi)],[tcy
tcy+tcdy*sind(phi)],...
        'Color',[0.4 0.4 0.4],'LineWidth',7);
    annotation(topView,'ellipse',[tcx-0.02 tcy-0.02*450/400 0.04
0.04*450/400],'FaceColor',[.2 .2 .2],'LineStyle','none');
%     plot(tx,ty,'b-'); hold on; %on/off trajectary
    plot(tx(i),ty(i),'r.','MarkerSize',20);
    axis([-0.6 0.6 -0.6 0.6]);
    ax = gca;
    ax.XAxisLocation = 'origin';
    ax.YAxisLocation = 'origin';
    ax.YTick = -0.6:0.1:0.6;
    ax.XTick = -0.6:0.1:0.6;
    ax.FontSize = 6;
    ax.FontWeight = 'bold';
    xlabel('x(m)'); ylabel('y(m)');
    title(sprintf('TOP VIEW: t =%6.3f s,  x =%6.2f m, y =%6.2f m'...
        ,tp(i),tx(i),ty(i)));
    pause(0.001)
end
grid
end
function handles = generateUI(h)
xc = 0.25;  yc=0.75;  wt=0.5; ht=0.075;
uicontrol(h,'Style','text','String','Coordinates', ...
    'FontWeight','bold','FontSize',20, ...
    'Units','normalized','Position',[0.25 yc wt ht]);
xc = 0.25;  yc=yc-0.075;  wt=0.25; ht=0.075;
uicontrol(h,'Style','text','String','x (m)', ...
    'FontWeight','bold','FontSize',20, ...
    'Units','normalized','Position',[xc yc wt+0.002 ht]);
handles.h_x = uicontrol(h,'Style','edit','String','0.5', ...
    'FontWeight','bold','FontSize',9, ...
    'Units','normalized','Position',[xc+wt+0.002 yc-0.001 wt ht+0.003]);
xc = 0.25;  yc=yc-0.075;  wt=0.25; ht=0.075;
uicontrol(h,'Style','text','String','y (m)', ...
    'FontWeight','bold','FontSize',20, ...
    'Units','normalized','Position',[xc yc wt+0.002 ht]);
handles.h_y = uicontrol(h,'Style','edit','String','0.5', ...
    'FontWeight','bold','FontSize',9, ...
    'Units','normalized','Position',[xc+wt+0.002 yc-0.001 wt ht+0.003]);
handles.hOut = annotation('TextBox',[0.1 0.08 0.8 0.35],'Interpreter','tex', ...
        'Background',[0.95 0.95 0.95], ...
        'String',' ','FitBoxToText','on');
function x = tani(d)
tnd = [0 0.5 1 2 4 8 20 40 100];
atnd = [0 26.57 45 63.43 75.96 82.87 87.14 88.57 89.43];
for q=1:9
        if tnd(q) > d
            break;
        end
end
    if(d>=100)
        x = 90;
    else
    x = (d-tnd(q-1))/(tnd(q)-tnd(q-1))*(atnd(q)-atnd(q-1))+atnd(q-1);
    return;
    end
end
```

Appendix III – Microcontroller Code

```
#include <xc.h>
#pragma config FOSC = INTIO7
#define _XTAL_FREQ 16000000
#define X_1         PORTBbits.RB0
#define X_2         PORTBbits.RB1
#define X_3         PORTBbits.RB2
#define X_4         PORTBbits.RB3
#define Y_1         PORTBbits.RB4
#define Y_2         PORTBbits.RB5
#define Y_3         PORTBbits.RB6
#define Keypad_PORT PORTB
#define Keypad_PORT_Direction TRISB
#define rs LATC3
#define en LATC4
#define lcd LATD
float atand[] = {0,26.57,45,63.43,75.96,82.87,87.14,88.57,89.43};
float tand[] = {0,0.5,1,2,4,8,20,40,100};
float tani(float d){ //tan inverse approximation
    int j;
    for(j = 0; j < 9;++j){
        if(tand[j]>d) break;
    }
    if(j==9) return 90;
    float ans;
    ans = (d-tand[j-1])/(tand[j]-tand[j-1])*(atand[j]-atand[j-1])+atand[j-1];
    return ans;
}
void pulse(){
    en = 1;
    NOP();
       en = 0;
}
void command(unsigned char cmd ){
       lcd = cmd;
       rs = 0;
       pulse();
}
void data(unsigned char dt){
       lcd = dt;
       rs = 1;
       pulse();
}
PWM_Duty(unsigned int duty,char c){
      if(duty<=20000)
  {
    duty = (float)duty/51.2;
    if(c == 1){DC1B0 = duty & 1; //Store the 1st bit
    DC1B1 = duty & 2; //Store the 0th bit
    CCPR1L = duty>>2;}// Store the remaining 8 bit
    else{DC2B0 = duty & 1; //Store the 1st bit
    DC2B1 = duty & 2; //Store the 0th bit
    CCPR2L = duty>>2;}// Store the remaining 8 bit
  }
}
char keypad_scanner(void){
    while(1){
          Y_1 = 1; Y_2 = 0; Y_3 = 0;
```

```
            if (X_1 == 1) {while (X_1 == 1);return '1';}
            if (X_2 == 1) {while (X_2 == 1);return '4';}
            if (X_3 == 1) {while (X_3 == 1);return '7';}
            if (X_4 == 1) {while (X_4 == 1);return '.';}
          Y_1 = 0; Y_2 = 1;
            if (X_1 == 1) {while (X_1 == 1);return '2';}
            if (X_2 == 1) {while (X_2 == 1);return '5';}
            if (X_3 == 1) {while (X_3 == 1);return '8';}
            if (X_4 == 1) {while (X_4 == 1);return '0';}
          Y_2 = 0; Y_3 = 1;
            if (X_1 == 1) {while (X_1 == 1);return '3';}
            if (X_2 == 1) {while (X_2 == 1);return '6';}
            if (X_3 == 1) {while (X_3 == 1);return '9';}
            if (X_4 == 1) {while (X_4 == 1);return 'E';}
    }}
float scanner(char c){
    command(0x01);
    command(0x0c);
      if(c == 2){
        data('y');data('(');data('m');data(')');data(':');}
      else {
        data('x');data('(');data('m');data(')');data(':');}
    command(0x85);
    command(0x0f);
      char ch;
      float x = 0;
      float s  = 1;
      ch = keypad_scanner();
      if(ch=='.'){
          data('-');
          ch = keypad_scanner();
          s = -1;}
      while(ch != 'E'){
      if(ch=='.'){
        data('.');
        float i = 1;
        ch = keypad_scanner();
        while(ch != 'E' && ch != '.'){
        data(ch);
        i /=10;
        x += (ch-'0')*i;
        ch = keypad_scanner();
        }
        return s*x;
      }
      data(ch);
      x = x*10 + ch - '0';
      ch = keypad_scanner();
      }
      return s*x;
      }
unsigned int dcCalcT(float x,float y){
    float rng;
    unsigned int dc;
    rng = x*x + y*y;
    float rng2 = rng*rng;
    float rng3 = rng*rng2;
    rng = 3.9939*rng3-4.7126*rng2 + 2.5891*rng + 0.0915; // square root
approximation
```

```
    rng2 = rng*rng;
    rng3 = rng*rng2;
    rng = -43.5426*rng3 + 72.3475*rng2 - 3.9841*rng - 0.2815; // function
approximation
    dc = rng*1116.6667;
    return dc;
}
unsigned int dcCalcP(float x,float y){
    float phi;
    unsigned int dc;
    phi = y/x;
    if(phi < 0)phi = -1*tani(-1*phi);
    else phi = tani(phi);
    if(x < 0 && y >= 0){
        phi = phi + 180;}
    else if (x < 0 && y < 0){
        phi = phi - 180;}
    dc = (180+phi)*56.1111;
    return dc;
}
void main(){
    OSCCON &= 0b10000100;
    OSCCON |= 0b00000010;
    ANSELB = 0;
    TRISD = 0;
    TRISC &= 0b11100111;
    NOP();
    command(0x38);
    command(0x01);
    command(0x0c);
    command(0x06);
    Keypad_PORT         = 0x00;
    Keypad_PORT_Direction = 0x0f;
    PR2 = 97; //Setting the PR2 formulae using Data sheet
    PR4 = 97;
    CCP1M3 = 1; CCP1M2 = 1;  //make CCP1 PWM
    CCP2M3 = 1; CCP2M2 = 1;  //make CCP2 PWM
    T2CKPS1 = 1; TMR2ON = 1; //Configure the Timer module
    T4CKPS1 = 1; TMR4ON = 1;
    TRISC2 = 0; // make port pin on C2 (CCP1 output pin) as output
    TRISC1 = 0; // make pin C1 (CCP2 output pin) as output
    float x,y;
  do{
    x = scanner(1);
    y = scanner(2);
    PWM_Duty(dcCalcT(x,y),1);
    PWM_Duty(dcCalcP(x,y),2);
  }while(1);
}
```