Ain Shams University
Faculty of Engineering
(ICHEP)

MCT211 – Automatic Control
Phase 2

Submitted to:

Dr Mohamed Omar

Eng Mina Maher

Eng Dina Zakaria

Submitted by Team 3:

Mahmoud Eldwakhly 21P0017

Yousef Ahmed Hesham 21P0088

Omar ElShawaf 21P0410

Mohamed Taha 21P0137

9 May 2024

TABLE OF CONTENTS

TABLE OF CONTENTS ii
LIST OF FIGURES iii
Contribution List iv
1. INTRODUCTION 1
2. MATHEMATICAL MODEL 2
3. ROOT LOCUS 4
3.1 Inner Loop 4
3.2 Outer Loop 5
4. SIMULATION 7
4.1 The Whole System 7
4.2 H-bridge Subsystem 7
4.3 MATLAB Function inside H-bridge Subsystem 8
4.4 Encoder Subsystem 8
4.5 Inner Loop PID 9
4.6 Outer loop PID 10
5. VALIDATION FOR SIMULATION & HARDWARE 11
5.1 At 30 cm position with 210 rpm 11
5.2 At 30 cm position with 150 rpm 12
5.3 At 70 cm position with 190 rpm 13
5.4 Test Holding The Cart 14
6. CONCLUSION 15

LIST OF FIGURES

Figure 1 Electrical System 2
Figure 2 Mechanical System 3
Figure 3 - Inner Loop steps 4
Figure 4: MATLAB Inner Loop Root Locus 5
Figure 5: Outer Loop Steps 6
Figure 6: MATLAB Outer Loop Root Locus.....6
Figure 7 Simulation. 7
Figure 8 H-bridge Subsystem. 7
Figure 9 MATLAB Function inside H-bridge Subsystem. 8
Figure 10 Encoder Subsystem 8
Figure 11 Inner Loop PID.....9
Figure 12 Outer loop PID.....10
Figure 13- 30 cm & 210 RPM Simulation 11
Figure 14- 30 cm & 210 Hardware. 11
Figure 15- 30 cm & 150 RPM Simulation 12
Figure 16 - 30 cm & 150 RPM Hardware 12
Figure 17- 70 cm & 190 RPM Simulation 13
Figure 18- 70 cm & 190 RPM Hardware.....13
Figure 20 Holding Test Graph.....14
Figure 19 Holding Cart Photo 14

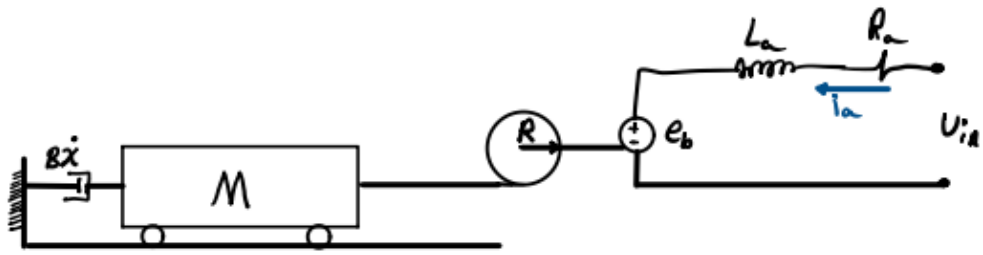
Contribution List

Name	ID	Contribution
Mohamed Mohamed Taha	21P0137	Parameter Estimation +Hardware Implementation+ Validation(Dead Zone + Reverse direction in speed &position)
Mahmoud Eldwakhly	21P0017	Cascaded PID Simulation+ Hardware Implementation+ Calculations+ Validation (Various positions limits in phase 1)
Yousef Ahmed Hesham	21P0088	Root Locus for Outer loop +Hardware Implementation+ Mathematical Model+ Validation(Various RPM limits in Cascaded PID)
Omar Elshawaf	21P0410	Root Locus for Inner loop Hardware Implementation + Manual PID Tuning + Mathematical Model + Validation + Simulation (MATLAB Function for direction reversing)

1. INTRODUCTION

The project entails a linear cart moving along a V-slot aluminum rod. This cart is linked to a motor, which rotates via a belt mechanism. Additionally, the motor is outfitted with an encoder to monitor its movement. We proceeded by creating a simulation using MATLAB Simulink to replicate real-world conditions accurately. This involved calculating parameters, establishing a mathematical model, and conducting parameter estimations to ensure the simulation's fidelity to the actual system. Our primary goal was to align the simulation's behavior with real-world scenarios, which required accurately estimating various factors influencing the cart's speed, such as different types of friction. This involved integrating input signals, like PWM, measuring the cart's output velocity, and utilizing Simulink/Simscape for system modeling and simulation. By employing analytical tools and leveraging MATLAB Simulink's Parameter Estimation feature, we aimed to determine precise values for system parameters, ensuring the virtual model closely resembled real-world performance. This endeavor demanded both technical proficiency and critical evaluation of simulation expertise, enabling us to compare responses of physical and simulated systems under different input signals. Additionally, we were tasked with implementing a PID controller to precisely control the linear distance traveled by the cart, ensuring it accurately reached specified points. Subsequently, we focused on controlling the cart's velocity, enabling it to reach specific points either at maximum velocity or at user-defined speeds. Thorough testing, including load testing, and graphical analysis confirmed the correctness and robustness of our implemented model.

2. MATHEMATICAL MODEL



Givens:

$R = 1.1\ cm = 0.011\ m$

$L_a = 0.01\ H$

$V_{in} = 12\ V$

$R_a = \frac{V_{in}}{I_{stall}} = \frac{12}{1.5} = 8\ \Omega$

$M = 107\ gm = 0.107\ Kg$

$J_r = 0.0002\ Kg.m^2$

$K_t = 0.494\ N.m/A$

$b = 5 \times 10^{-6}\ \frac{N.m}{r/s}$

$Friction = 7.92\ N$

For electrical system:

$(R_a + \cancel{L_a \times S})\, I(s) + e_b(s) = V_{in}(s)$

neglect

$e_b(s) = K_b \times S \times \theta_m(s)$

$T_m = K_t\, I(s)$

$\frac{R_a}{K_t} \times T_m(s) + K_b S\, \theta_m(s) = V_{in}(s)$

$$\boxed{16.19\, T_m(s) + 0.494 \times S \times \theta_m(s) = V_{in}(s)} \rightarrow (1)$$

Figure 1 Electrical System

For mechanical system:

$$J_{eq}\ddot{\theta}_2 = T_{meq} - b_{eq}\dot{\theta}_2$$

$$0.0002\,\ddot{\theta}_2 = T_m - 5\times10^{-6}\dot{\theta}_2$$

$$\boxed{T_m = 2\times10^{-4}\ddot{\theta}_2 + 5\times10^{-6}\dot{\theta}_2} \rightarrow ②$$

Sub 2 in 1

$$V_{in}(s) = 3.24\times10^{-3}\ddot{\theta}_2 + 8.1\times10^{-5}\dot{\theta}_2 + 0.494\times\dot{\theta}_m$$

$\rightsquigarrow$ Assume $\frac{N_1}{N_2} = 1$, $\theta_1 = \theta_2$

$$V_{in}(s) = 3.24\times10^{-3}\ddot{\theta}_m + 0.494\dot{\theta}_m$$

$$V_{in}(s) = \left[3.24\times10^{-3}s^2 + 0.494s\right]\theta_m(s)$$

$$\boxed{\frac{\theta_m(s)}{V_{in}(s)} = \frac{1}{3.24\times10^{-3}s^2 + 0.494s}}$$

$$\boxed{\frac{\omega(s)}{V_{in}(s)} = \frac{1}{3.24\times10^{-3}s + 0.494}}$$

Figure 2 Mechanical System

3. ROOT LOCUS

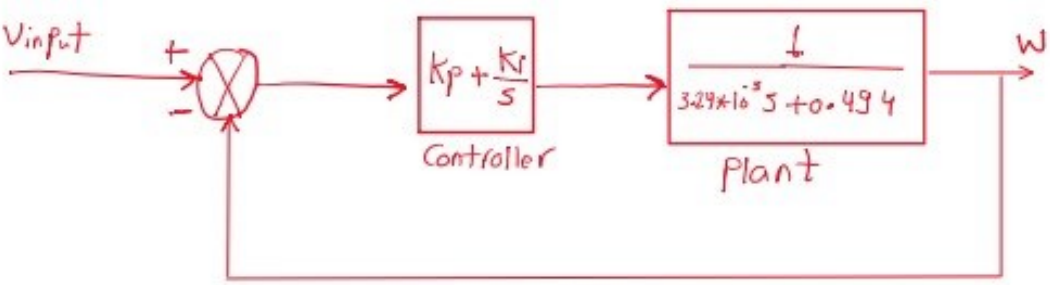
3.1 Inner Loop

$$\frac{\theta_m(s)}{V_{in}(s)} = \frac{1}{3.24\times10^{-3}s^2 + 0.494s}$$

$$\frac{\omega(s)}{V_{in}(s)} = \frac{1}{3.24\times10^{-3}s + 0.494}$$

plant.

*Inner loop [velocity] :-



open loop :- $\left[K_p + \frac{K_i}{s}\right] * \frac{1}{\tau s + 1}$

$$\frac{K_p\left[s + \frac{K_i}{K_p}\right]}{s} * \frac{K}{\tau s + 1}$$

poles : $0, \frac{-1}{\tau_c} \Rightarrow 0, \frac{-1}{6.5\times10^{-3}}$

zeros : $-\frac{K_i}{K_p} = \frac{-1}{T_c}$, $\boxed{\frac{K_i}{K_p} \simeq 152}$

Figure 3 - Inner Loop steps

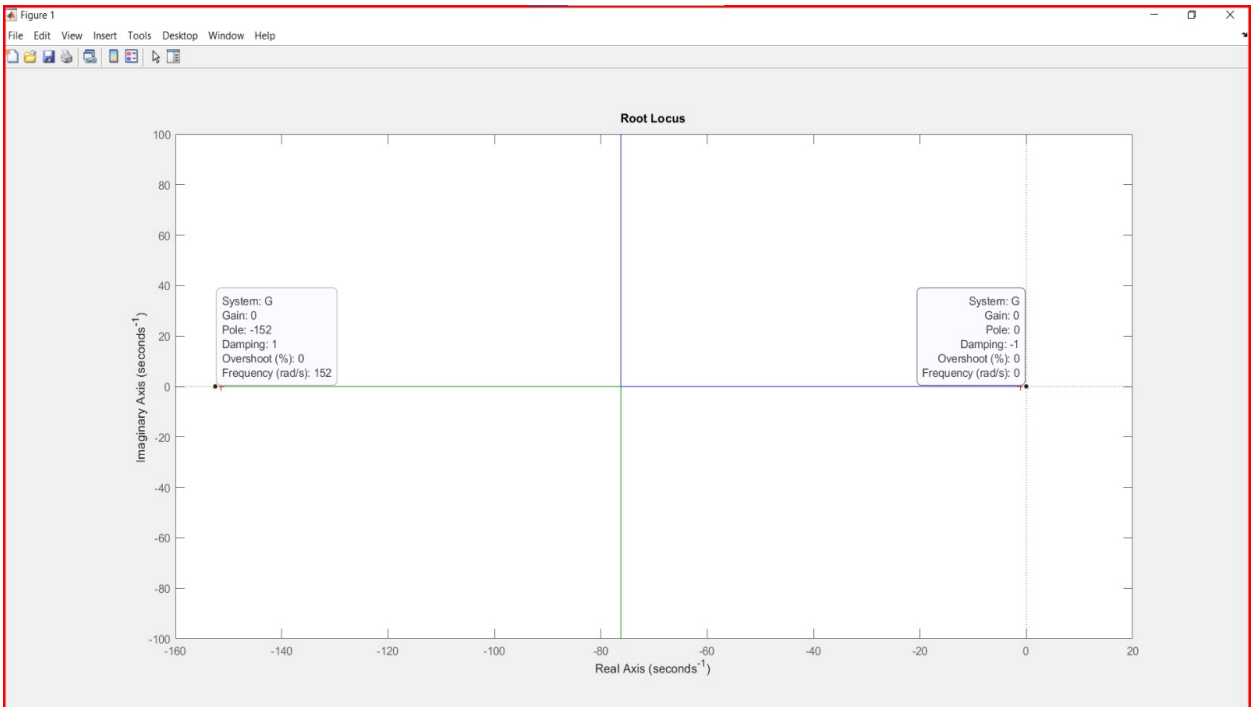
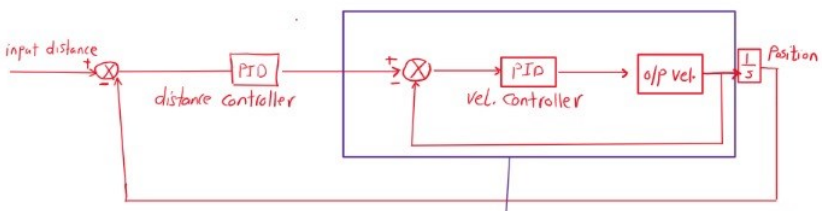


Figure 4: MATLAB Inner Loop Root Locus

3.2 Outer Loop



Approximate to first order.

Outer loop :-

$$K_p * \left[\frac{1}{\tau s + 1}\right] * \frac{1}{s} \simeq \frac{\Delta k}{\tau s + 1} = 1 \quad \boxed{\text{as o/p = I/p}}$$

from velocity graph $\tau = \frac{T_{settle\ time}}{4} = 6$ ms

No. zeros , poles :- 0, $\frac{-1}{\tau_c}$

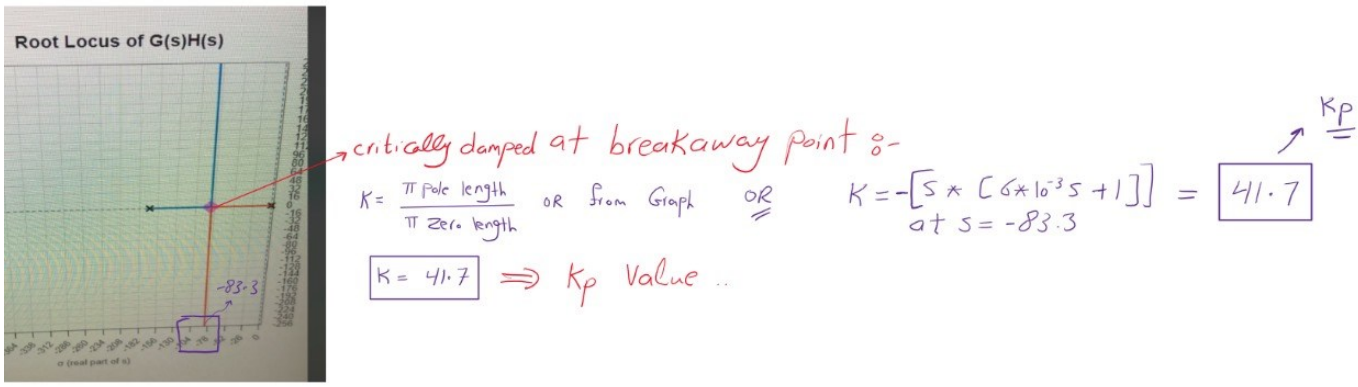


Figure 5: Outer Loop Steps

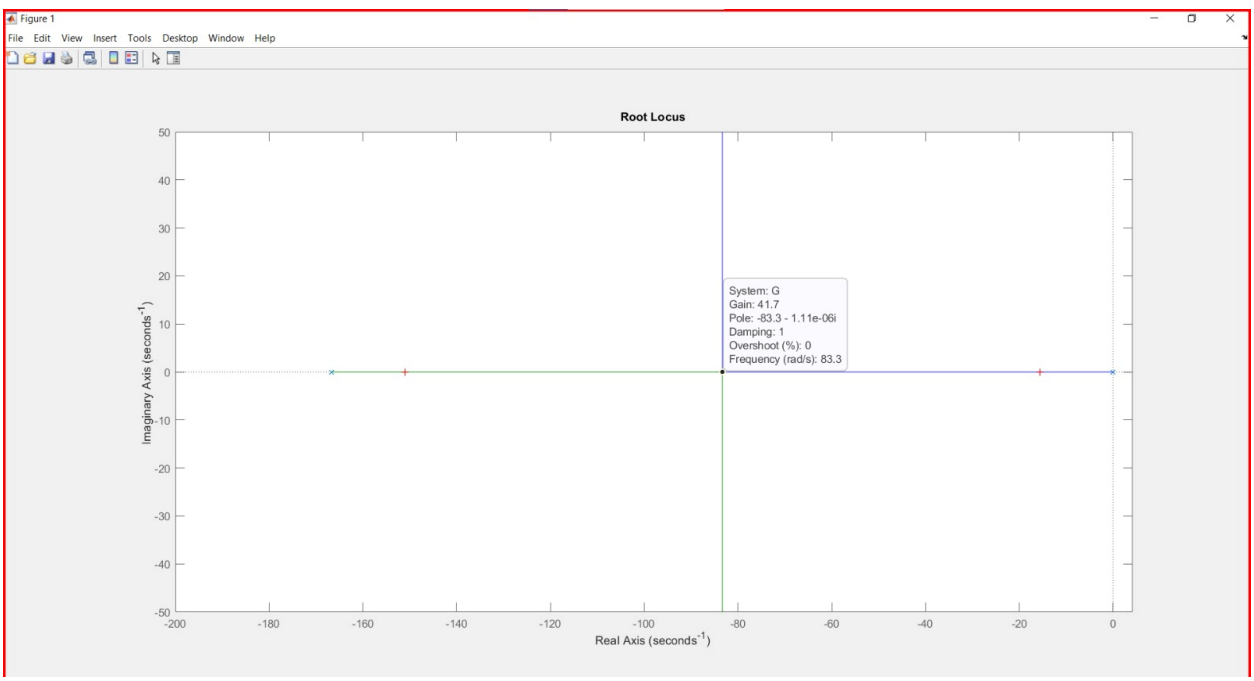


Figure 6: MATLAB Outer Loop Root Locus

4. SIMULATION

4.1 The Whole System

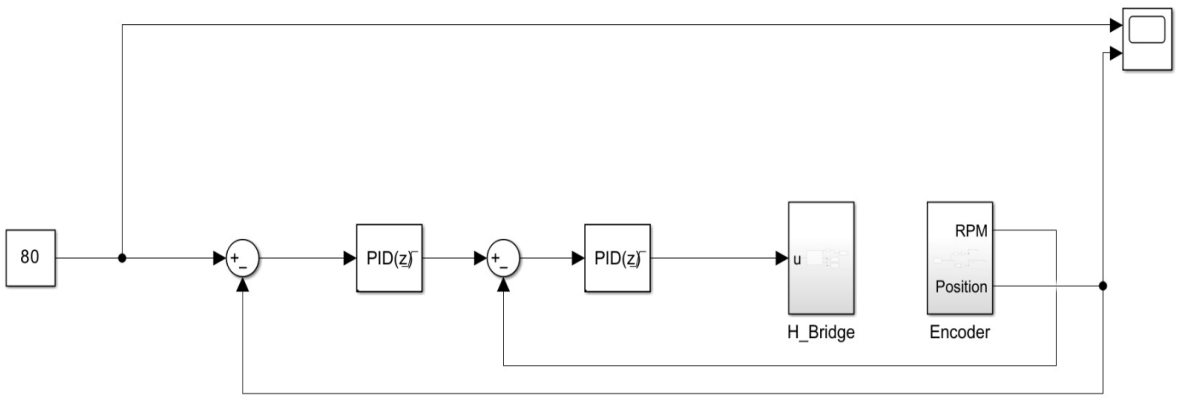


Figure 7 Simulation.

4.2 H-bridge Subsystem

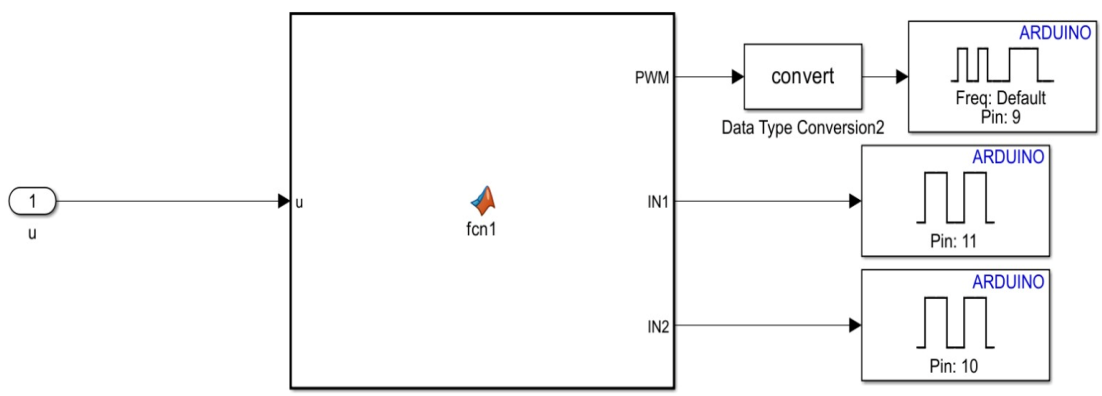


Figure 8 H-bridge Subsystem.

4.3 MATLAB Function inside H-bridge Subsystem

```
function [PWM,IN1,IN2]=fcn1(u)
PWM= abs(u);
if u<0
IN1=1; IN2=0;
elseif u>0
IN1=0; IN2=1;
else
IN1=0; IN2=0;
end
```

Figure 9 MATLAB Function inside H-bridge Subsystem.

4.4 Encoder Subsystem

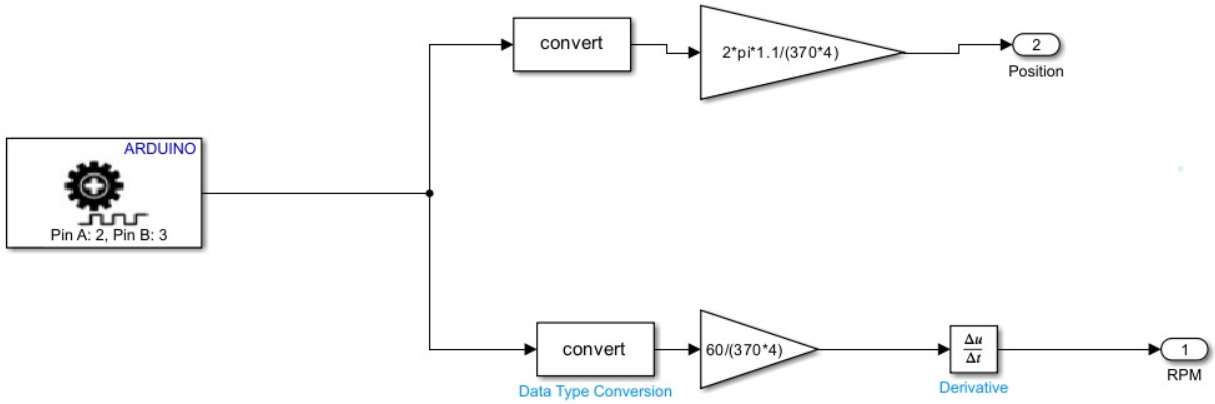


Figure 10 Encoder Subsystem

4.5 Inner Loop PID

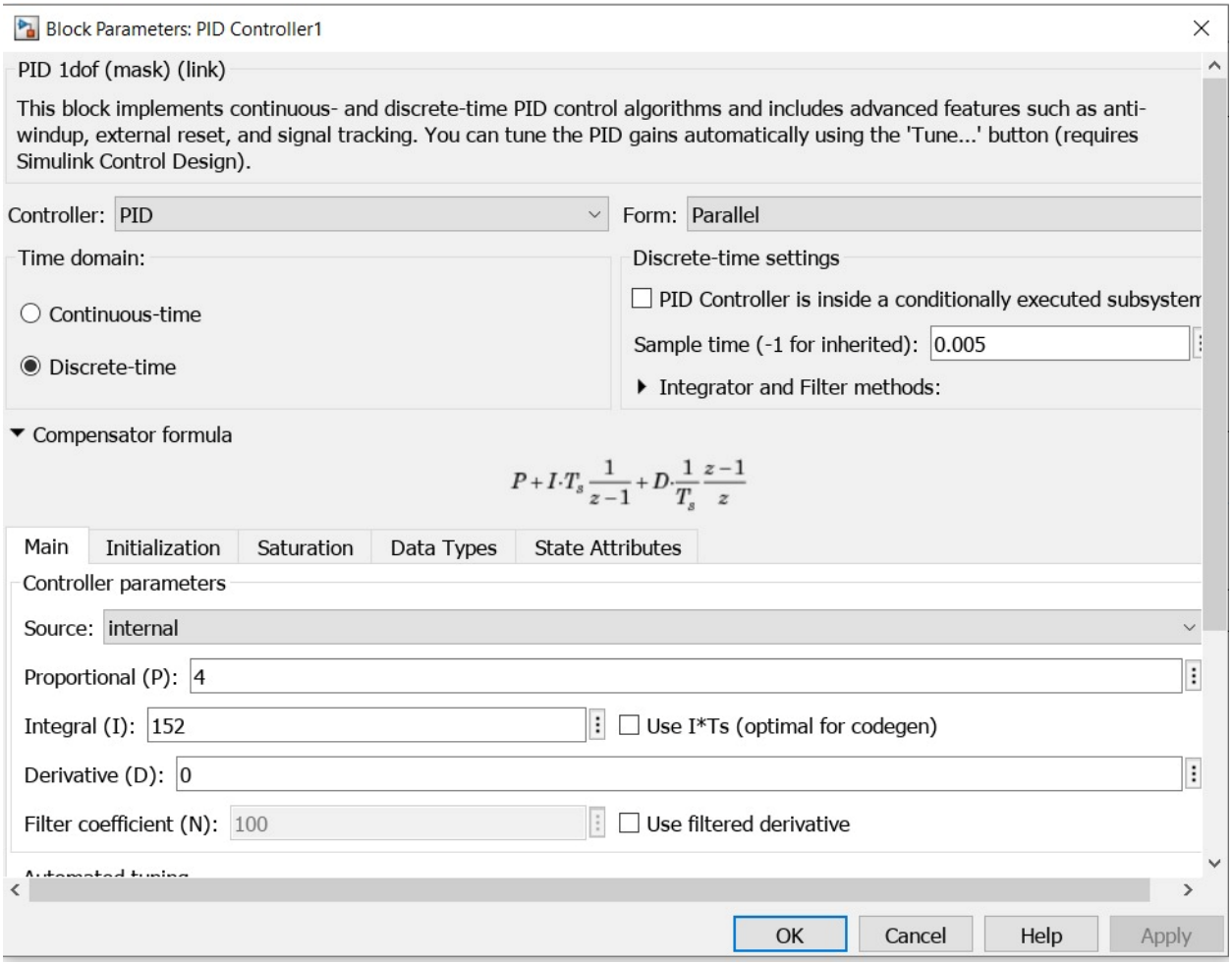


Figure 11 Inner Loop PID

4.6 Outer loop PID

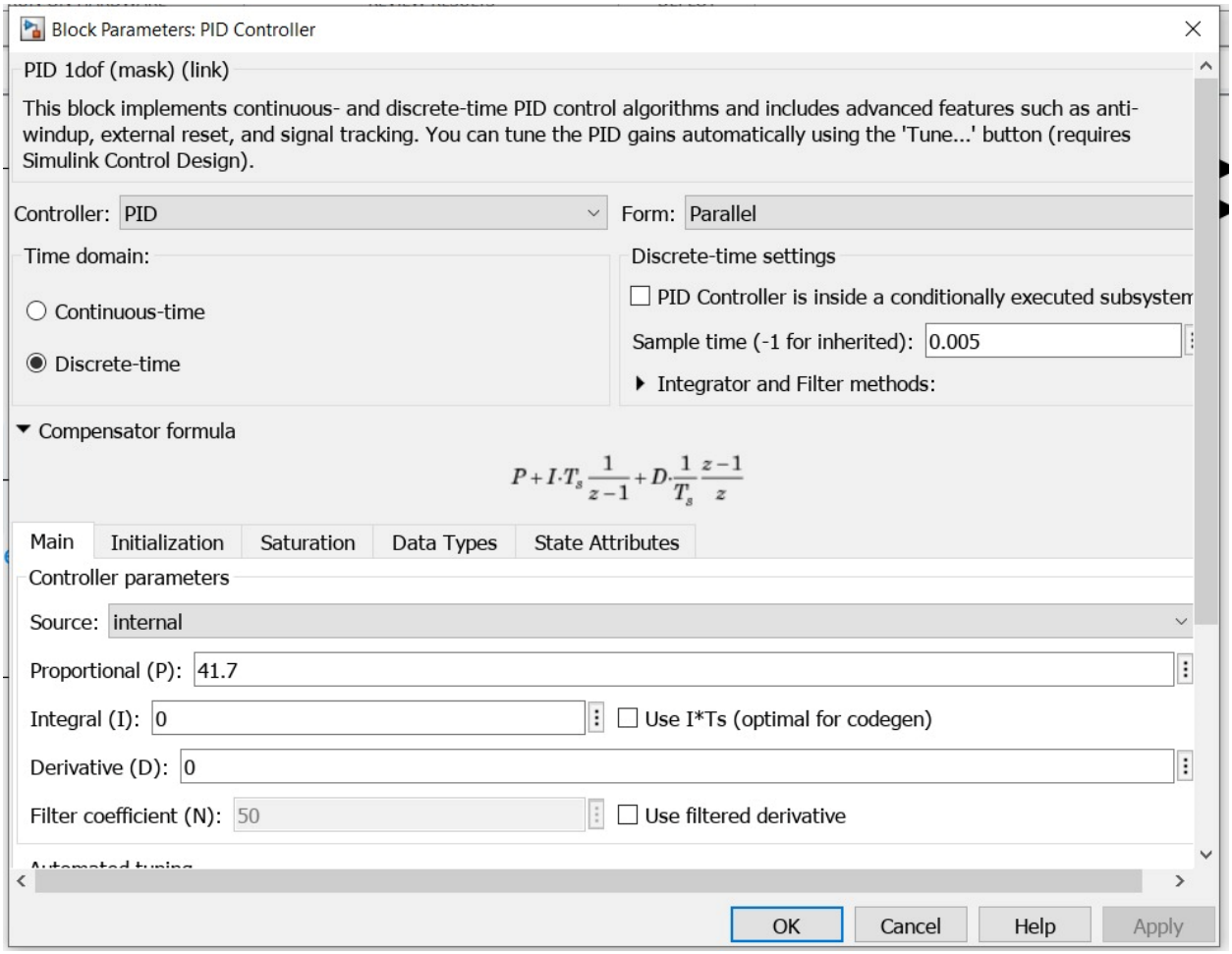


Figure 12 Outer loop PID.

5. VALIDATION FOR SIMULATION & HARDWARE

5.1 At 30 cm position with 210 rpm

Simulation:

Figure 13- 30 cm & 210 RPM Simulation

Hardware:

Figure 14- 30 cm & 210 Hardware.

5.2 At 30 cm position with 150 rpm

Simulation:

Hardware:

Figure 15- 30 cm & 150 RPM Simulation

Figure 16 - 30 cm & 150 RPM Hardware

5.3 At 70 cm position with 190 rpm

Simulation:

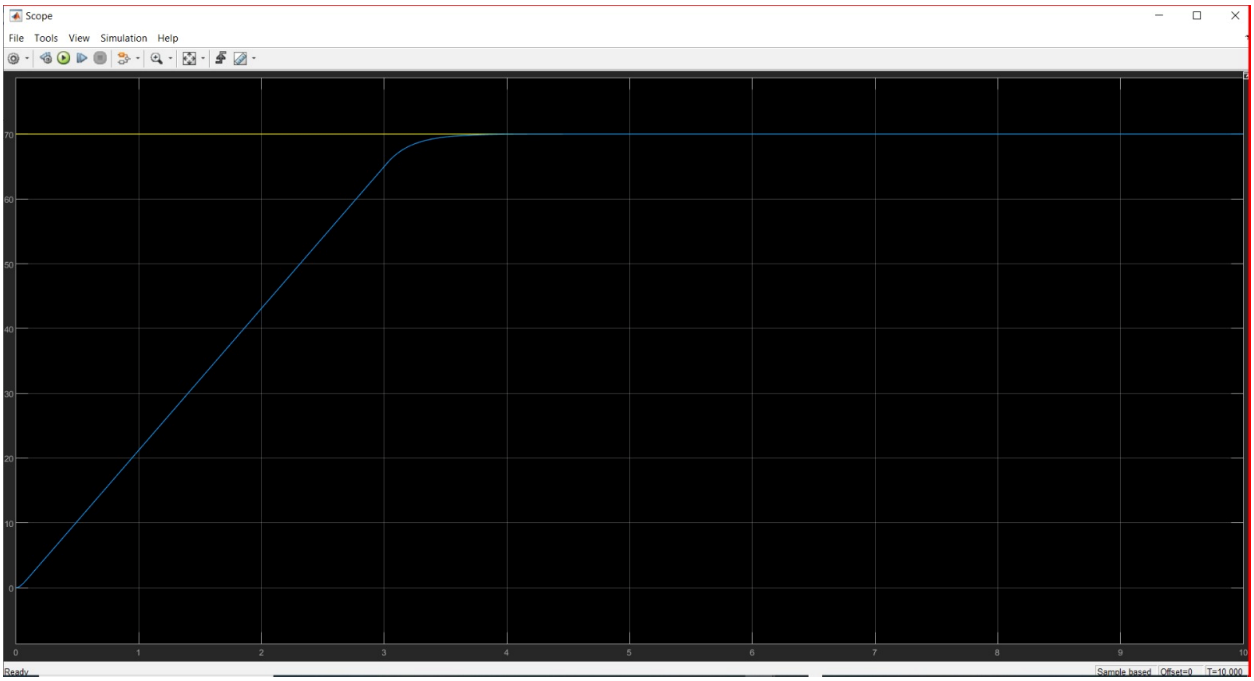


Figure 17- 70 cm & 190 RPM Simulation

Hardware:

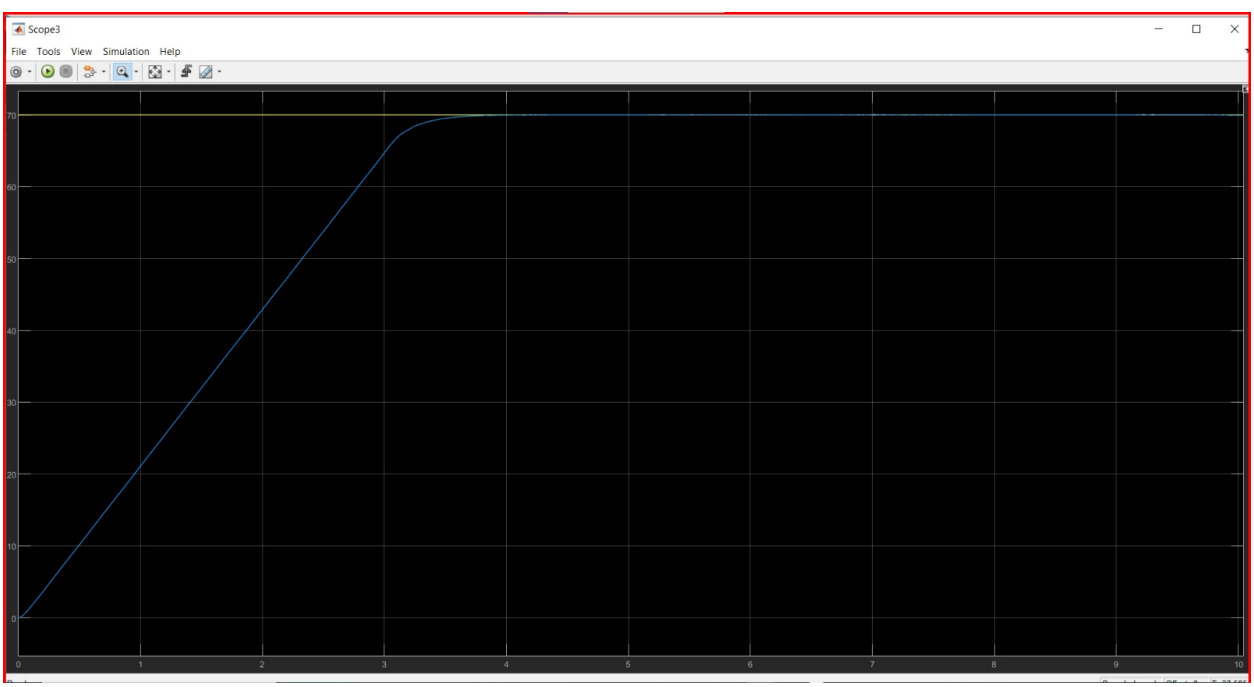


Figure 18- 70 cm & 190 RPM Hardware

5.4 Test Holding The Cart

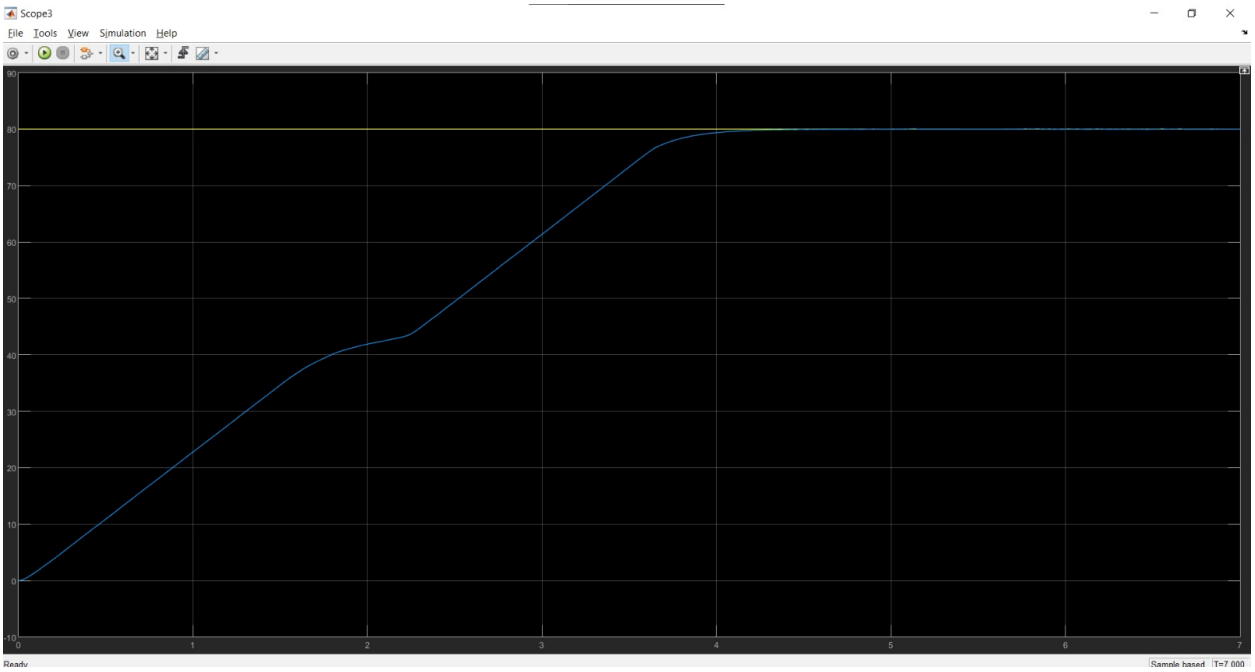


Figure 20 Holding Test Graph

After holding the cart to observe its response to reach the final position or not and it reaches the final position effectively.

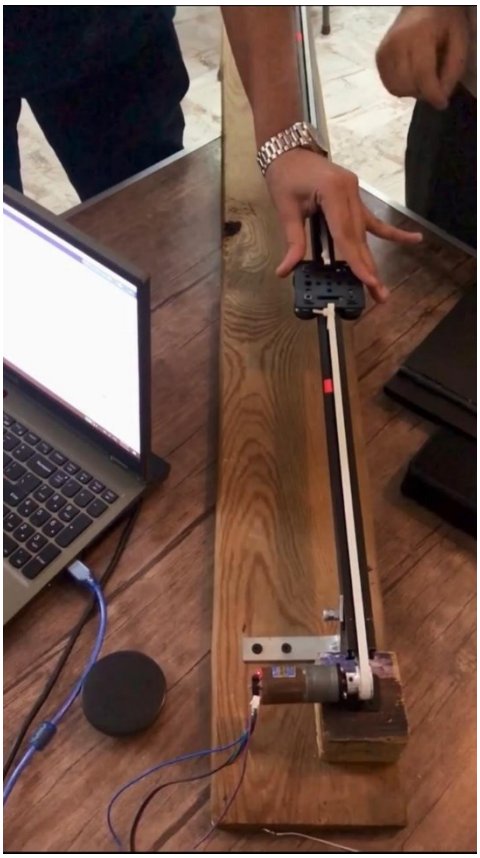


Figure 19 Holding Cart Photo

6. CONCLUSION

All in all, our project involved creating a simulation of a linear cart moving along an aluminum rod using MATLAB Simulink and control its position and speed using cascaded PID Controllers. This project helped us to gain hands-on experience of controlling a system by using hardware in the loop and observe the effect by changing the variables inside MATLAB.

We've applied the control system design steps starting from identifying a plant and modelling it until verifying and testing the control loop. Our main goal was to have the cart reach the position entered as fast as possible, while not passing a certain limit of speed (which we decide). Root locus method was exploited to design the PID controller. It gave us an overview of the values of the PID instead of just manually tuning and not knowing if this is close or not.

Eventually, this goal was reached and has been validated by testing out different positions and speeds and checking out the graph that was output. Checks were making sure that there was no overshoot, no steady-state error, response is quick and that the final value reached is correct.

AIN SHAMS UNIVERSITY

FACULTY OF ENGINEERING

International Credit Hour Programs

(ICHEP)

MCT211 – Automatic Control

Major Task Report (Phase 1)

Submitted to:

Dr Mohamed Omar & Eng Mina Maher & Eng Dina Zakaria

Team 3 Group Members:

Name:	ID:
Mahmoud Eldwakhly	21P0017
Yousef Ahmed Hesham Hammad	21P0088
Mohamed Mohamed Taha	21P0137
Omar ElShawaf	21P0410

Contents

Introduction: 3
Mathematical model 4
General Steps to perform project: 6
Comparing results of physical model & simulation 26
Controlling Distance (Phase 1) 27
Hardware Model 28
H-Bridge subsystem 28
MATLAB Function 29
Encoder Subsystem 29
Simulation 30
Simulated Model Subsystem 30
Model using Transfer function made manually 31
..... 31
Tuning the PID values in Simulation 32
Tuning the PID values in Hardware 33
..... 35
Final Values of PID 36

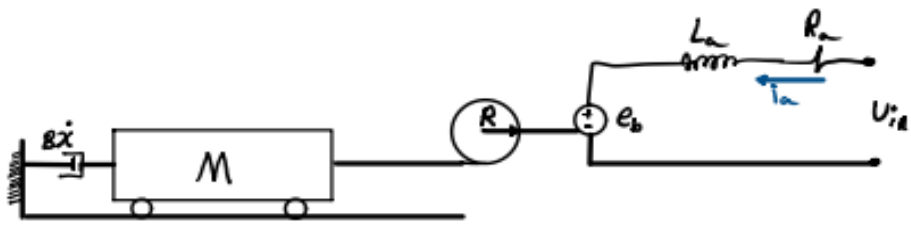
Introduction:

The project simply is a linear cart that moves linearly on a V-slot aluminium rod. The cart is attached to the motor that rotates by a belt, and the motor is equipped with an encoder to track its movement. To control the motor's speed, we've used tools like MATLAB, Arduino, and to control the motor's speed, an H-bridge is used.After fixing the physical system completely. Using MATLAB, A simulation and mathematical model is made, The Task is to make the simulation behave like the model in daily life. This will be done by estimating parameters like different types of friction affecting the real-life speed of the cart. So, it should be put it in simulation and so on. This process involves applying input signals which will be PWM in our project, measuring the cart's output velocity, and using Simulink/Simscape to model and simulate the system. Through analytical tools and the Parameter Estimation feature in MATLAB Simulink, we aim to pinpoint the values of system parameters, ensuring our virtual model closely mirrors the real-world behavior. This endeavor not only calls for technical skills but also prompts reflection on the acquired simulation expertise, evaluating and comparing responses of both the physical and simulated systems under different input signals.

After that, we are ready to begin Control journey, implement and validate the system. Using PID controller that regulates the position of the cart given desired set points. That is for phase 1.

Mathematical model

Mathematical model of the system



Givens:

$R = 1.1\ cm = 0.011\ m$

$L_a = 0.01\ H$

$V_{in} = 12\ V$

$R_a = \frac{V_{in}}{I_{stall}} = \frac{12}{1.5} = 8\ \Omega$

$M = 107\ gm = 0.107\ Kg$

$J_r = 0.0002\ Kg.m^2$

$K_t = 0.494\ N.m/A$

$b = 5 \times 10^{-6}\ \frac{N.m}{r/s}$

$Friction = 7.92\ N$

For electrical system:

$$(R_a + L_a \times s)\, I(s) + e_b(s) = V_{in}(s)$$

$$e_b(s) = K_b \cdot s \cdot \theta_m(s)$$

$$T_m = K_t\, I(s)$$

$$\frac{(R_a + L_a s)\, T_m(s)}{K_t} + K_b s\, \theta_m(s) = V_{in}(s)$$

$$\boxed{\left(\frac{8 + 0.01 s}{0.494}\right) T_m(s) + 0.494 \times s \cdot \theta_m(s) = V_{in}(s)} \rightarrow (1)$$

For mechanical system:

$$J_{eq}\ddot{\theta}_2 = T_{meq} - b_{eq}\dot{\theta}_2$$

$$0.0002\,\ddot{\theta}_2 = T_m - 5\times10^{-6}\dot{\theta}_2$$

$$\boxed{T_m = 2\times10^{-4}\ddot{\theta}_2 + 5\times10^{-6}\dot{\theta}_2} \rightarrow (2)$$

Sub 2 in 1

$$V_{in}(s) = \left(\frac{8+0.01s}{0.494}\right)\times\left[2\times10^{-4}\ddot{\theta}_2 + 5\times10^{-6}\dot{\theta}_2\right] + 0.494\times\dot{\theta}_m$$

$$V_{in}(s) = 3.24\times10^{-3}\ddot{\theta}_2 + 8.1\times10^{-5}\dot{\theta}_2 + 4.05\times10^{-6}\times s\times\ddot{\theta}_2 + 1.01\times10^{-7}\times s\times\dot{\theta}_2 + 0.494\dot{\theta}_m$$

$$V_{in}(s) = (3.24\times10^{-3} + 4.05\times10^{-6}\times s)\ddot{\theta}_2 + (8.1\times10^{-5} + 1.01\times10^{-7}\times s)\dot{\theta}_2 + 0.494\dot{\theta}_m$$

$$V_{in}(s) = (3.24\times10^{-3} + 4.05\times10^{-6}\times s)\ddot{\theta}_m + (0.4941 + 1.01\times10^{-7}\times s)\dot{\theta}_m \rightarrow \text{Assume } \frac{N_1}{N_2} = 1$$

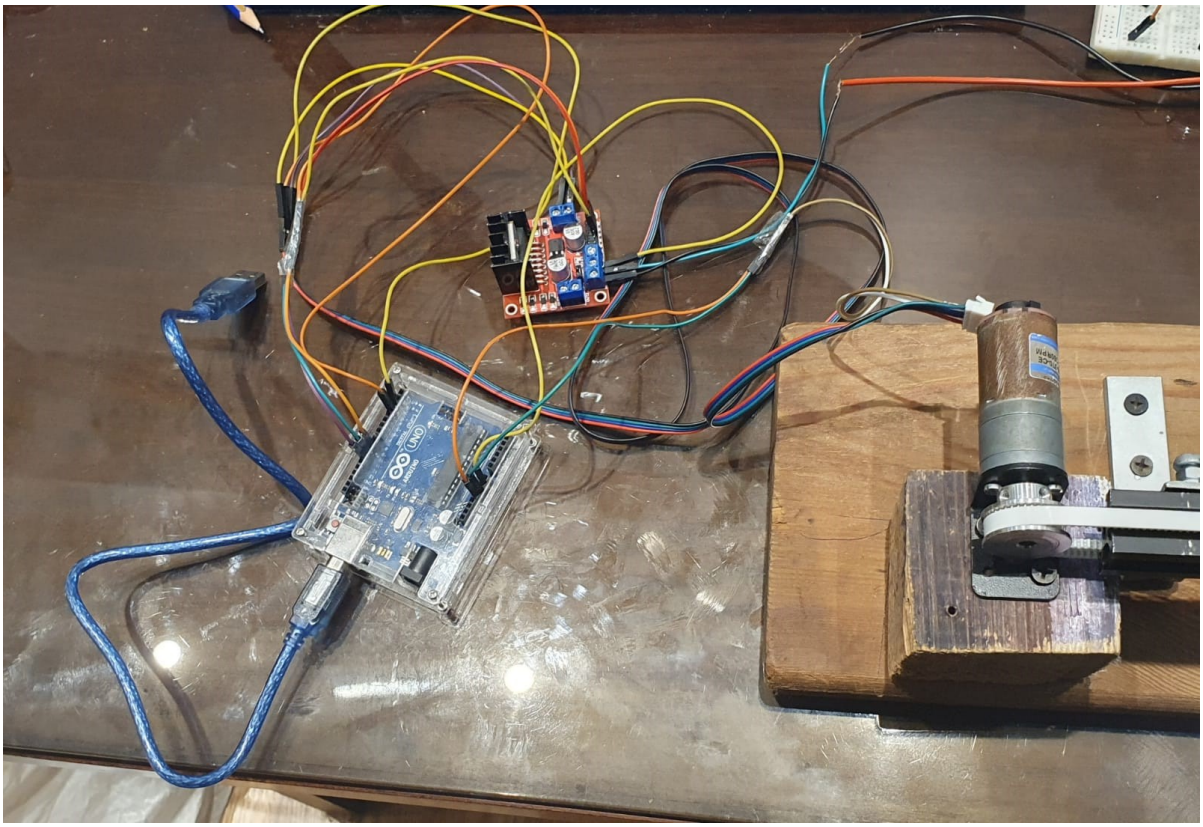
$$V_{in}(s) = (3.24\times10^{-3} + 4.05\times10^{-6}\times s)\times s^2\times\theta_m + (0.4941 + 1.01\times10^{-7}\times s)\cdot s\times\theta_m$$

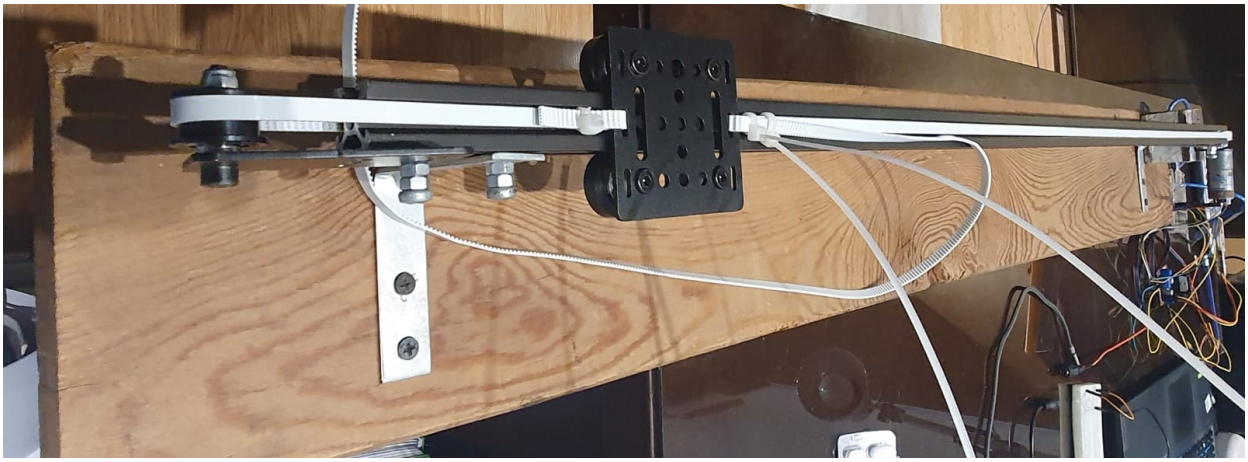
$$V_{in}(s) = \left[4.05\times10^{-6}\times s^3 + 3.24\times10^{-3}s^2 + 0.4941s\right]\theta_m$$

$$\boxed{\frac{\theta_m(s)}{V_{in}(s)} = \frac{1}{\left[4.05\times10^{-6}\times s^3 + 3.24\times10^{-3}s^2 + 0.4941s\right]}}$$

General Steps to perform project:

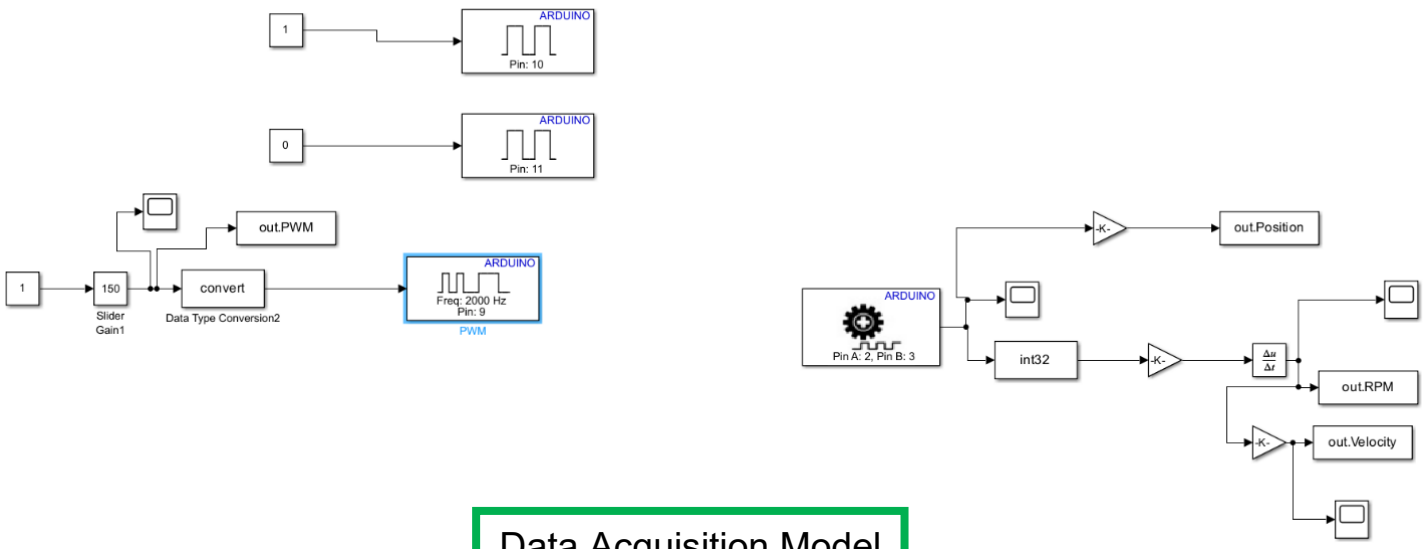
1. Build & connect our system hardware components





2. Build a data acquisition model in Simulink

- The aim of this step is to get readings (pulses) from the encoder and then convert it into whatever I want (RPM, Linear Velocity, Linear Position, etc…). Below is our data acquisition model:



Data Acquisition Model

- With the aid of the Simulink Support Package for Arduino, the above data acquisition model was created. To start with, a PWM block was used, and this is the block which will send the PWM signal from the Arduino to the H-bridge Enable pin. A slider gain is connected to this block (or instead a constant block could've been used) to change the PWM.
- 2 digital output blocks have been used and are connected to 2 constant blocks of values 1 & 0. These digital blocks are just for the direction of rotation of motor. By using values of 1 & 0, the motor is made to rotate in a certain direction. These pins are 11 & 12 in our circuit in Arduino that'll be connected to IN1 & IN2 pins in Arduino.
- An encoder block is used to get readings from our physical encoder. This block the pins at which the phases of encoder are connected on Arduino are defined.

- To get RPM out of the encoder, we used the derivative block & a gain block of 60/(370*4). Derivative makes the output pulses/s. To convert pulses/s into RPM, multiply by 60 and divide by the encoder's pulses per revolution (ppr) multiplied by 4 since our encoder is a quadrature encoder (It uses 2 output channels to indicate the position of the motor)
- There are 2 To Workspace blocks (PWM & RPM). The data will be sent to Matlab Workspace as 2D arrays. These will be used to run the simulated model.

3. Create a simulation model similar to the real-life dynamic system (Include dead zones)

- The simulation model must behave exactly like the real-life model. For this to occur, the dead zones in our system must be identified. Dead zone is the range of PWM at which the motor won't rotate when given these values.

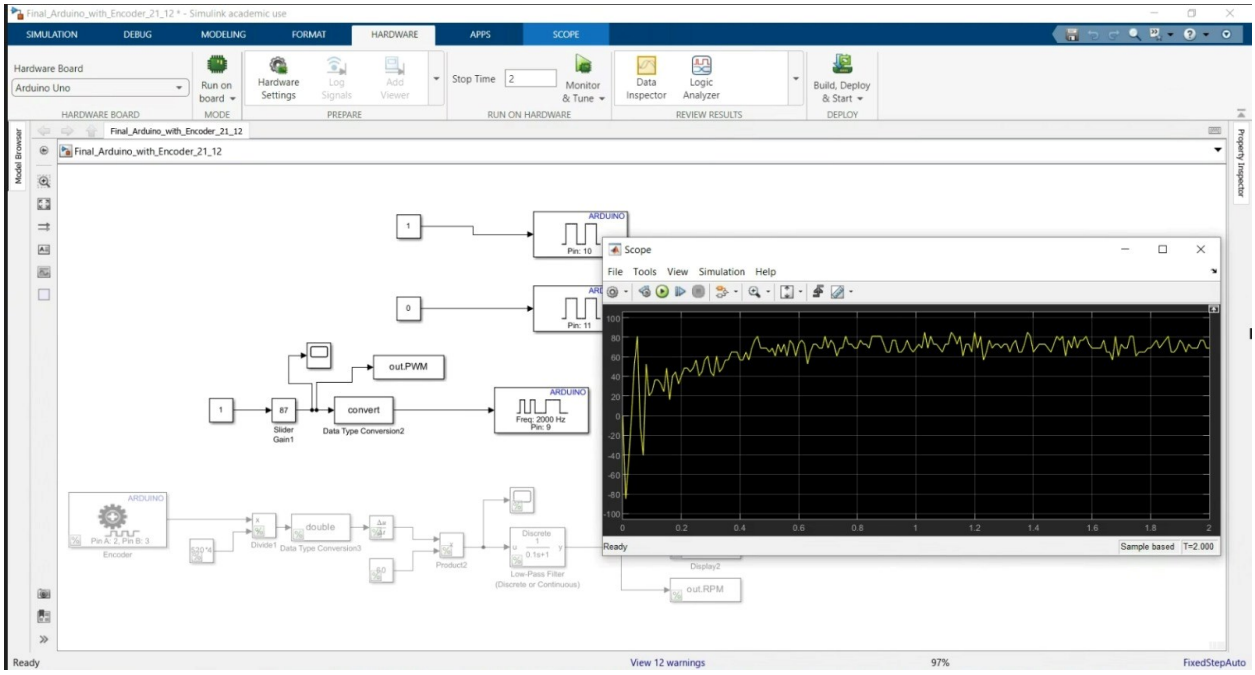
DEAD Zones:

- There are two dead zones in our model. The greater value of dead zone will be inserted to our model, but we'll calculate both since their difference will be used to bring an initial value for Coulomb & Breakaway Friction forces which are inside the Translational Friction Block.

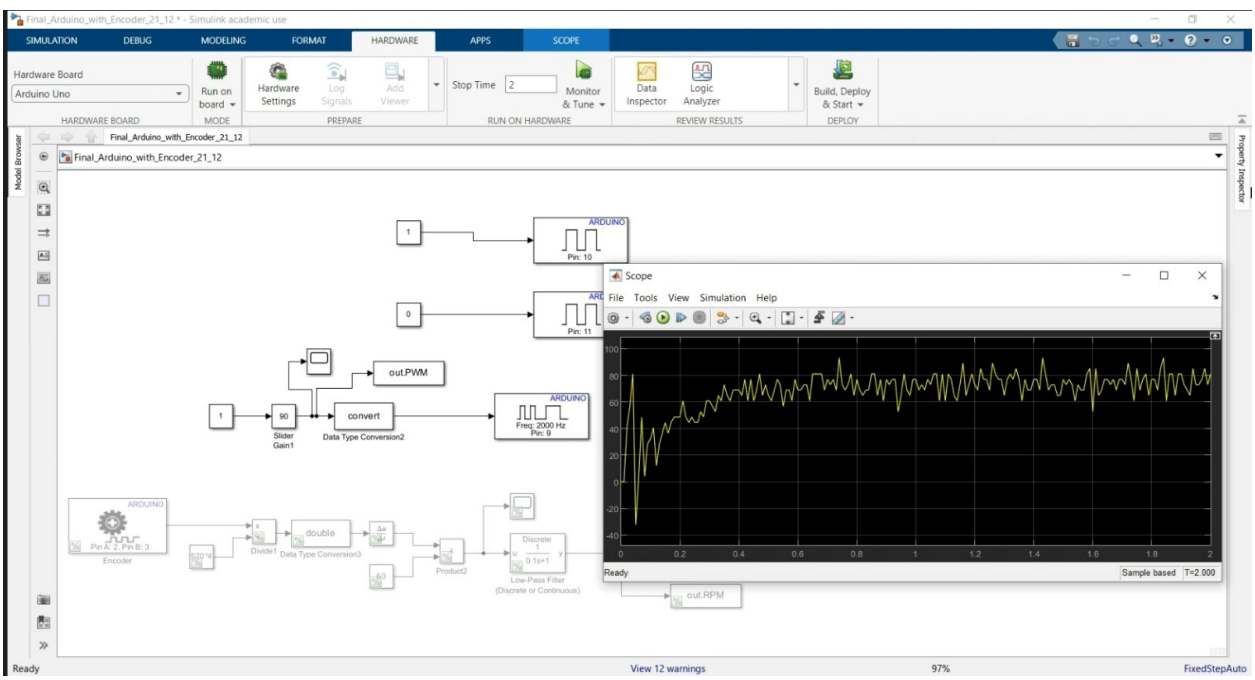
1. One will be calculated when the motor has no load nor friction. It is required to know the voltage needed to make the motor start rotation with no-load (Only connecting the H-bridge & Motor). We calculated it by applying different PWM on motor via H-bridge. Consequently, the value of PWM at which the motor will start rotation can be determined. The motor starts rotation at **81** PWM.

➢ Our sequence of testing:

- 90 PWM without load:



- 87 PWM without load:

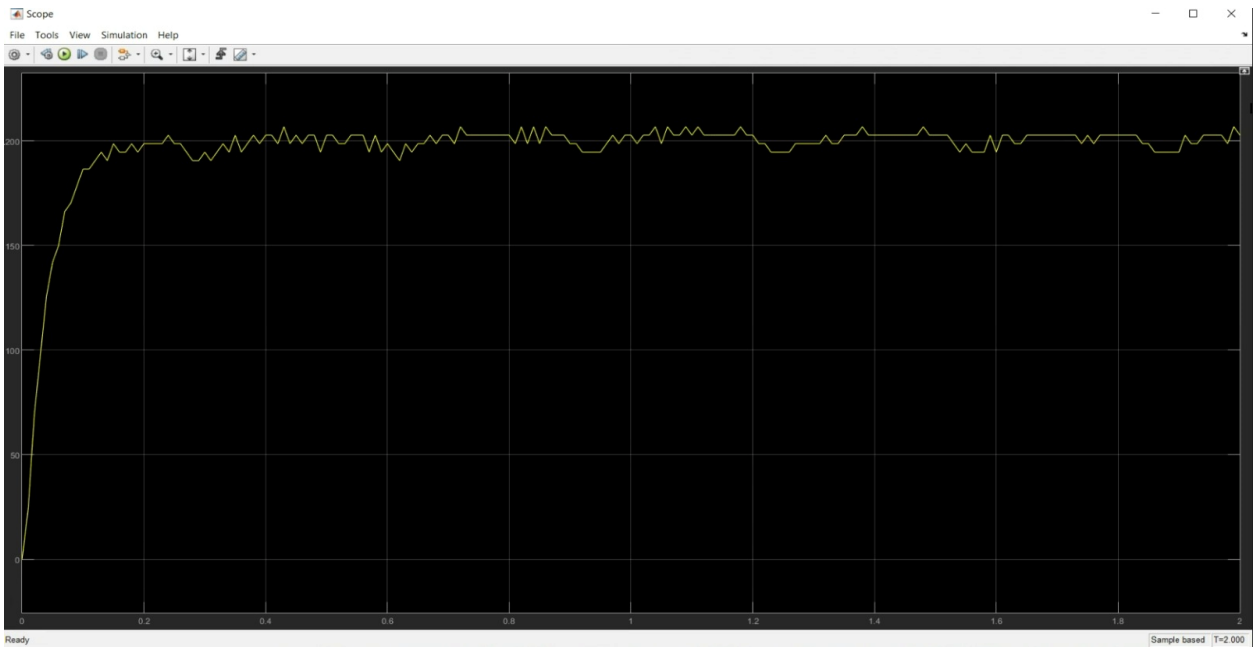


- At 86 PWM the motor rotated but, at 85 PWM it did not rotate and the same at 84 PWM. We thought that the right value at 85 PWM., but we

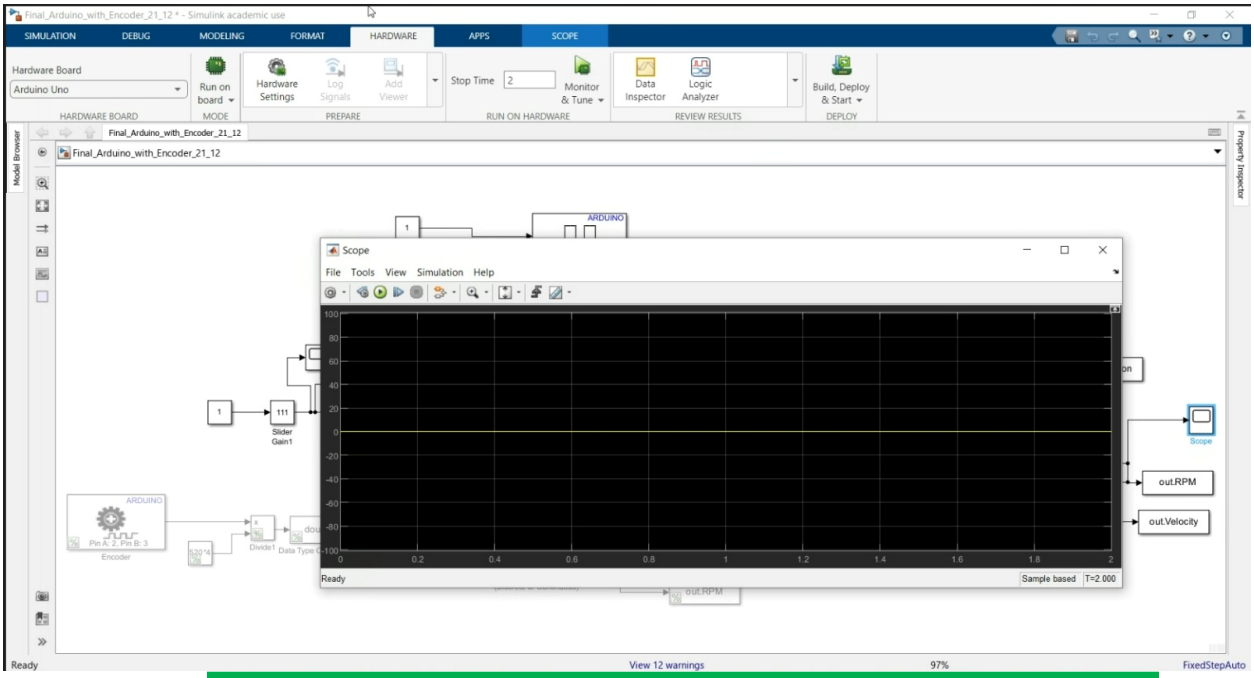
found out that the motor rotated at PWM 83 and PWM 82 but did not rotate at PWM 81. Eventually, we decided that 81 PWM is our first dead zone value

2. The other dead zone is to calculate friction, and we calculated it after putting the load on motor which is belt with the cart. We tested like the other dead zone by applying different input voltage. We found that it is at PWM **111**.

- At 255 PWM:



- At 111 PWM:



RPM=0, so this is the value of the second dead zone

- By subtracting the 2 dead zones, this is the change in PWM the friction caused. We will use that to calculate coulomb and breakaway friction forces and put them in transitional friction force block.

Calculations:

- We need to bring initial values for R_a (armature resistance) & K_b (Back-emf Coefficient). A K.V.L is done (Inductance is very small so it's negligible). The following equation is reached:

$$\boxed{E = IR + Kb * w}$$

- This will be calculated @ 2 conditions: stall & no-load to bring R_a & K_b

$$E = IR + 0$$

$$12 = 1.5 * R$$

$$\boxed{Ra = 8}$$ @ Stall Conditions (Stall Current=1.5 A & w=0)

$$E = IR + kb * w$$

$$12 = 0.1 * 8 + kb * 26.2$$

$$\boxed{Kb = 0.493}$$ @ Stall Conditions (No-load Current=0.1 A & No-load Speed = 250 Rpm = 26.2 rad/s

- Now, the difference in PWM in the dead zones is 30. This will be converted into voltage and will be substituted in the following equation to bring the friction force:

$$\boxed{E = \frac{Ra}{Kb} * F * r}$$

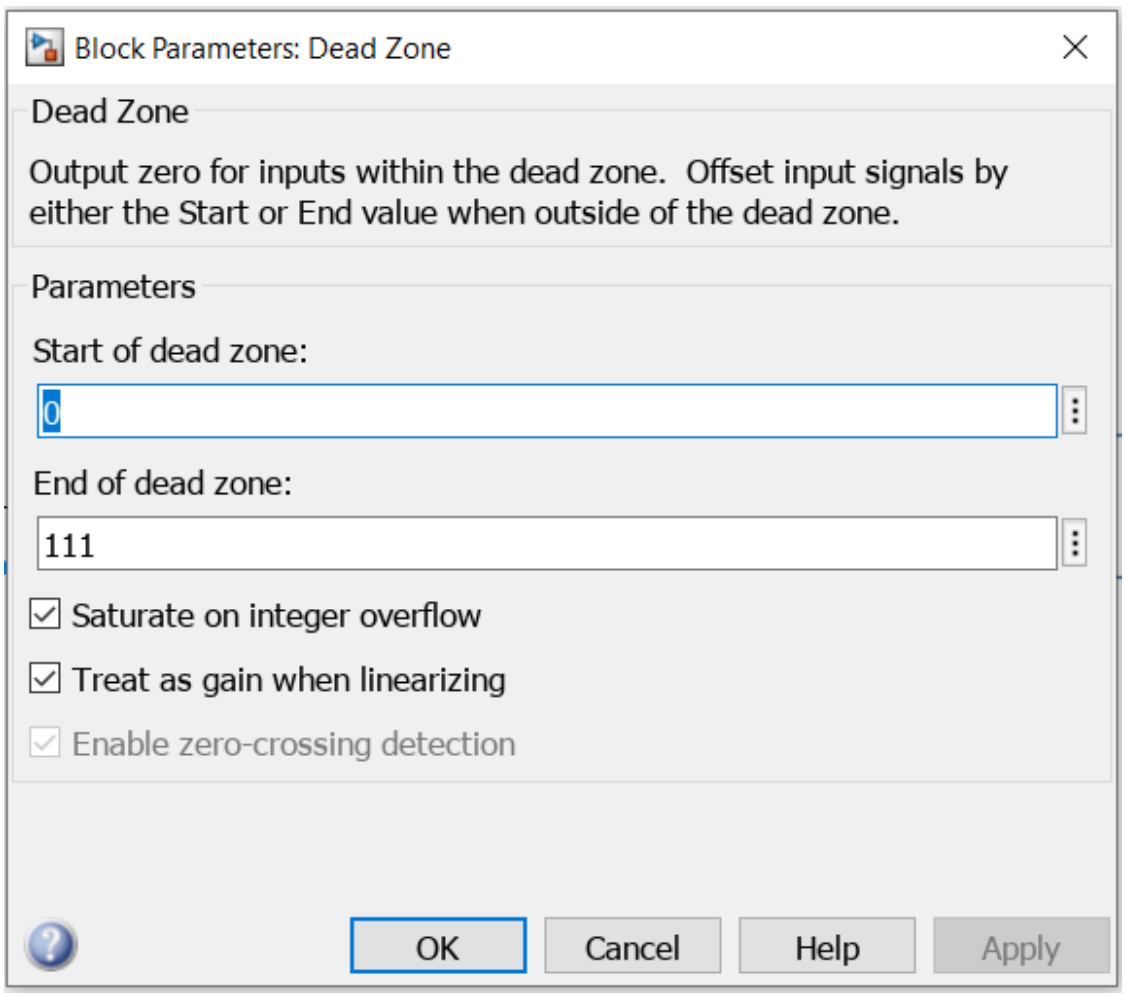
@ r=radius of pulley (0.011 m) & E=difference in PWM but in voltage

$$\frac{30}{255} * 12 = \frac{8}{0.493} * F * 0.011$$

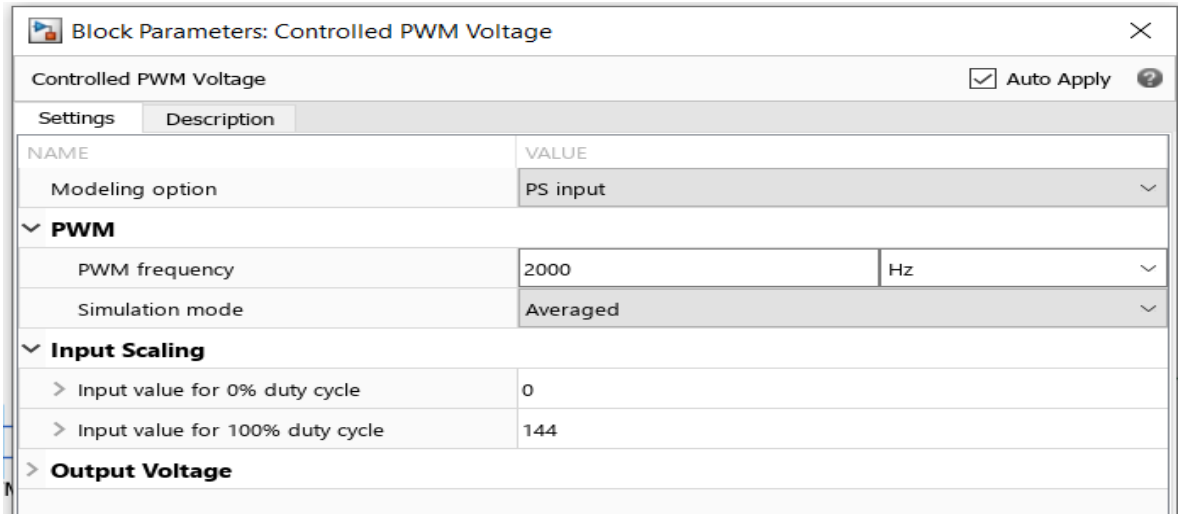
$$\boxed{F = 7.92}$$

Notes about model:

- The input signal to this simulation is a constant PWM signal
- The dead zone block has an end of 111 PWM

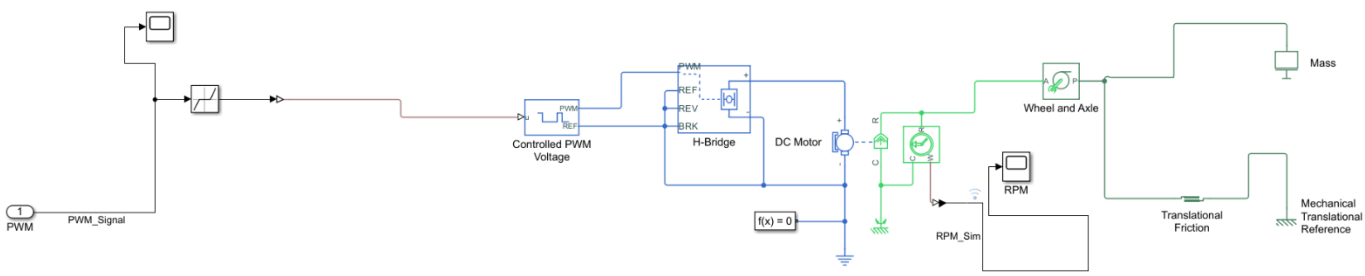


- The controlled PWM Voltage Block will have an input of 144 PWM for 100 % duty cycle (255-111)



- Wheel and axle block is to simulate the pulley and we determined the pulley's radius (1.1 cm)
- Mass block is the mass of the cart (107 g)
- We defined variables for the circuit's parameters and gave them initial values

Final Simulation Model



4. Use parameter estimation to estimate the parameters of the system at one PWM signal

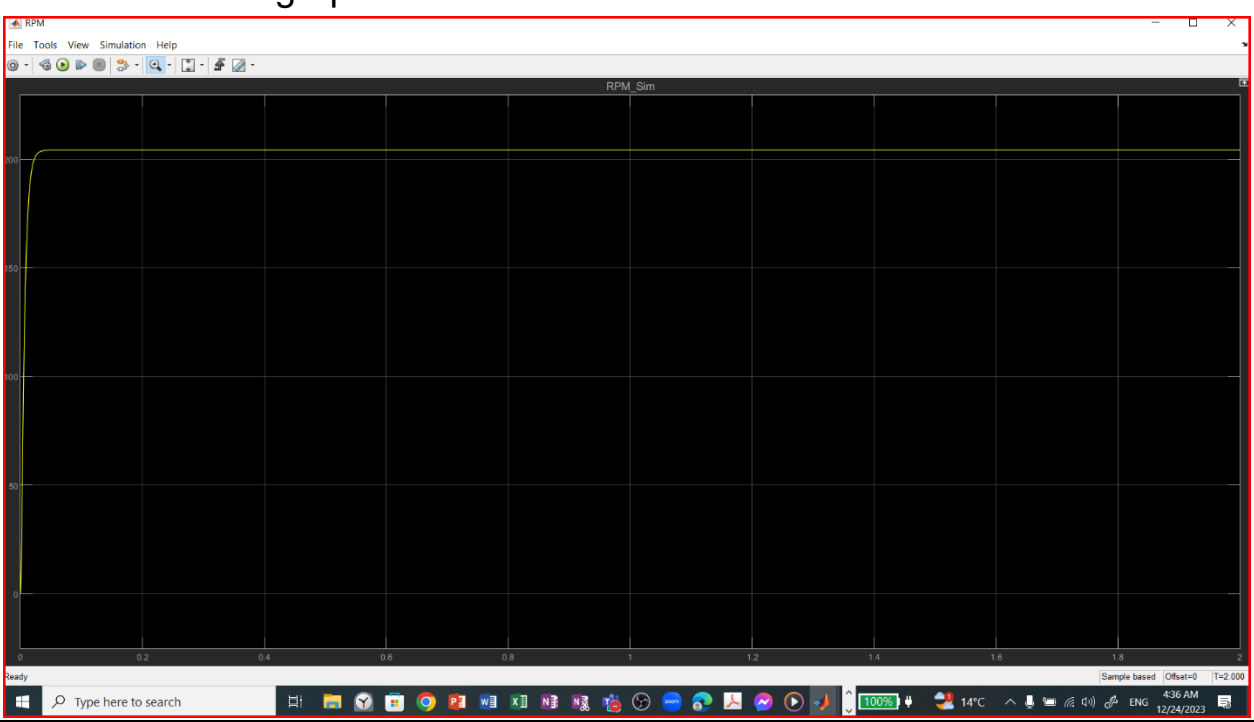
1. Firstly, we monitor & tune the Data Acquisition Model. PWM & RPM signals will be sent to the simulation model.
2. Next, convert the PWM & RPM 2D arrays into timeseries and store them in a new variable. Moreover, create a time array with 0.01 step (Our model step-size) & a final value of the stop time beginning with 0 (This is to be able to convert the 2-D array into timeseries)

3. Thirdly, in simulation model, put and determine your input signal that'll operate the model (PWM signal). Also, initialize your variables.
4. Log the output signal to RPM signal generated from data acquisition model (So, simulation knows what the graph of RPM should look like)
5. Run the simulation (take care, that the stop time of simulation and data acquisition model must be identical)
6. Cut the 2D arrays of RPM & PWM to 0.4 seconds (as this is what we only want to parameter estimate on). Furthermore, create a new time array with the same step size of model but a final value of 0.4 (So later, the 2-D array can coincide with the time)
7. Open parameter estimator app and identify your input and output signals (data & time arrays)
8. Click plot & simulate to plot the simulated & measured graphs
9. Now, select the parameters you want to estimate, then click estimate
10. Eventually, the measured graph will coincide on the simulated graph

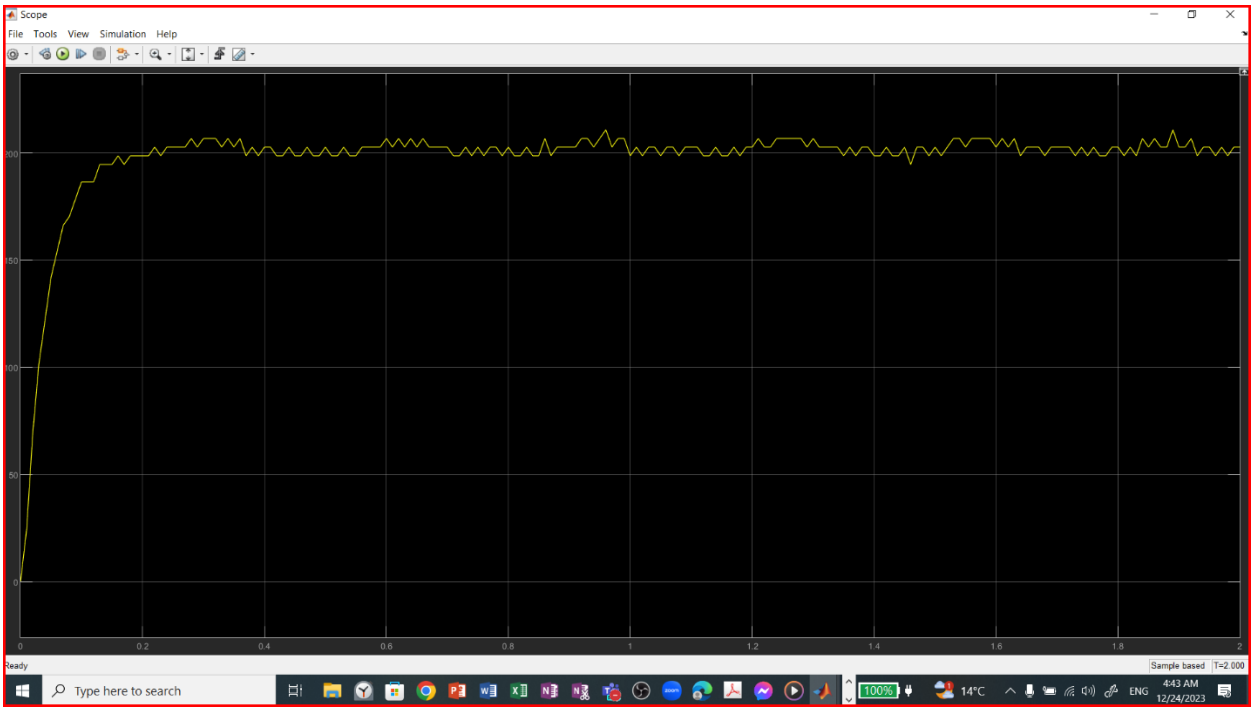
We performed our parameter estimation on 255 PWM, and here are our results:

255 PWM

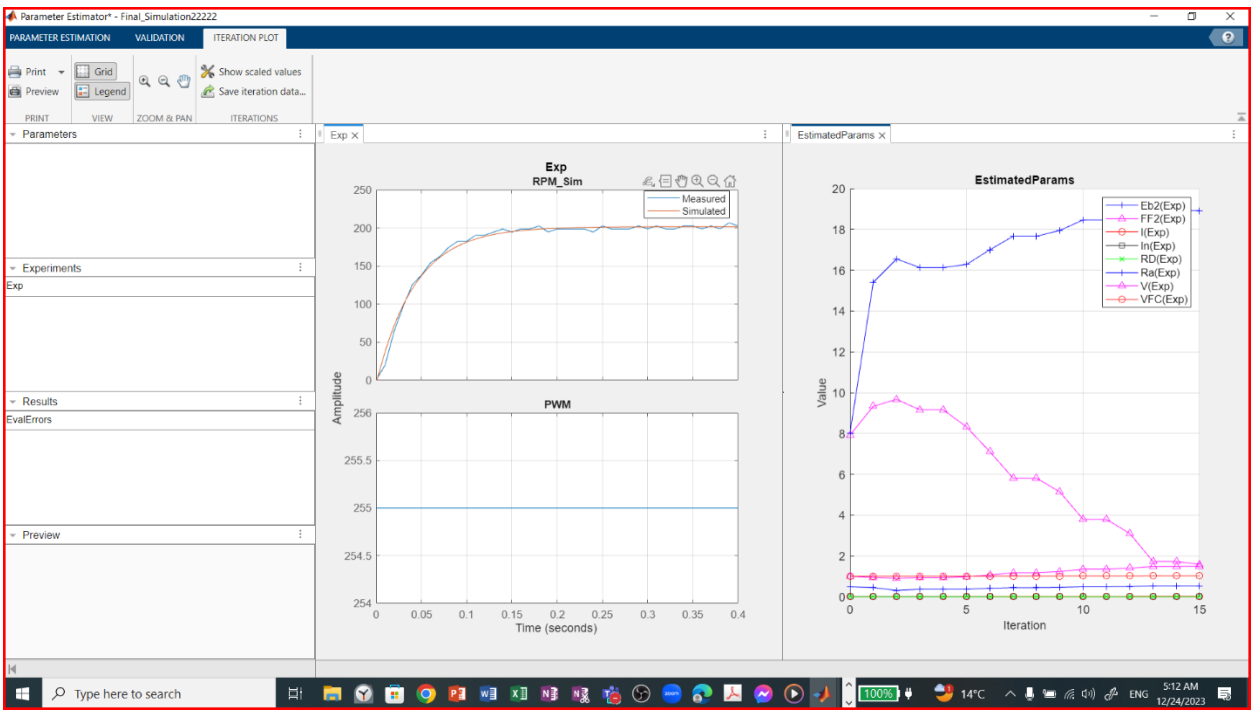
- Simulated graph



- Measured Graph

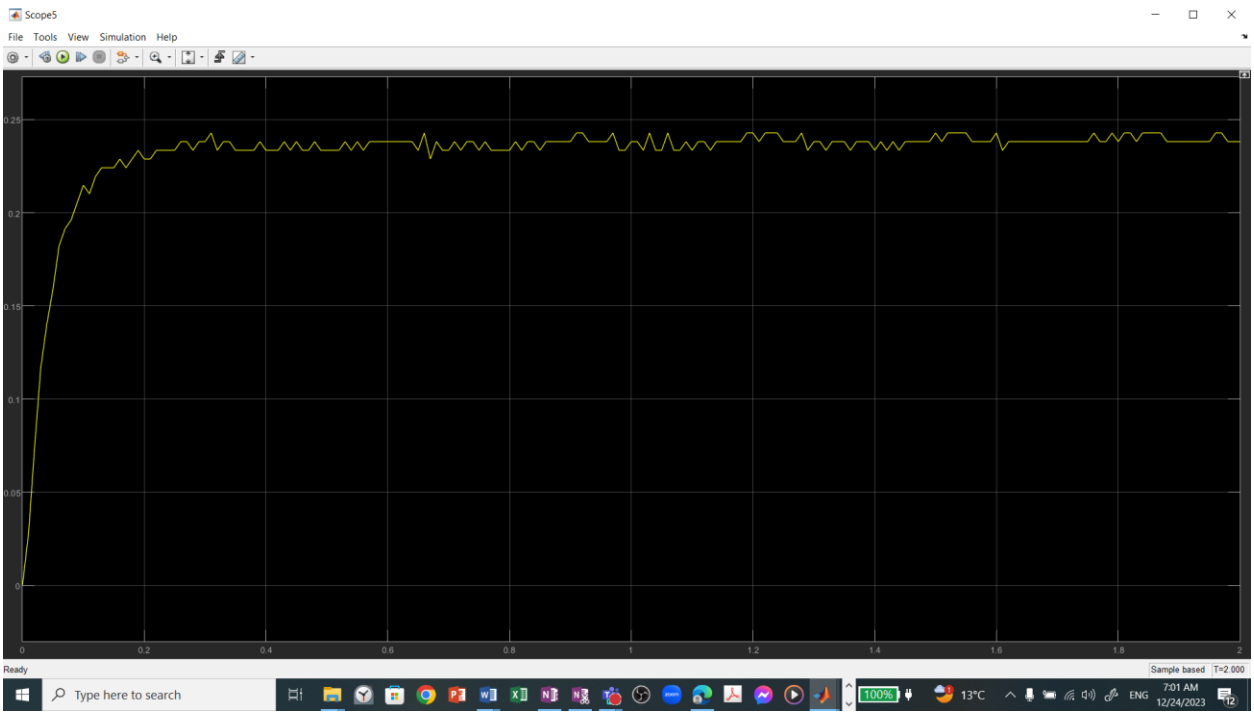


- Result after parameter estimation



- Since this is a 1 degree of freedom, by knowing the RPM or by making the simulation an exact copy of the measured, so will the linear velocity of the cart as well. Below is the result of the measured graph of the cart's linear velocity:

- Velocity



Velocity graph looks the same like the RPM graph (Multiply RPM by constant to bring linear velocity)

- Parameters estimated:

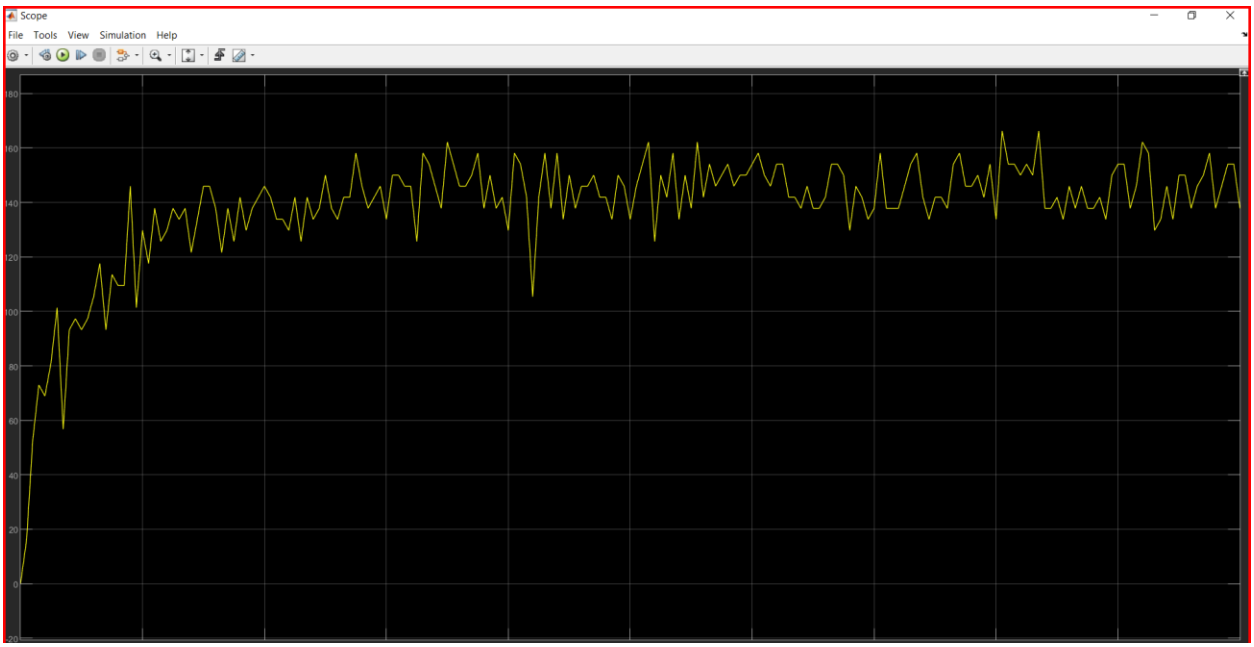
Parameter	Initial Value
Eb2	0.536753678380987
FF2	1.5907845922581
I	0.0235727851988216
In	0.000681256412882049
RD	5.01200021054277e-06

5. Try at different PWMs, the measured graph and simulated graph should coincide on each other

- Now, the parameters are determined, so when trying out different PWMs, the measured & simulated graphs should be the same like each other
- Here are some of our results:

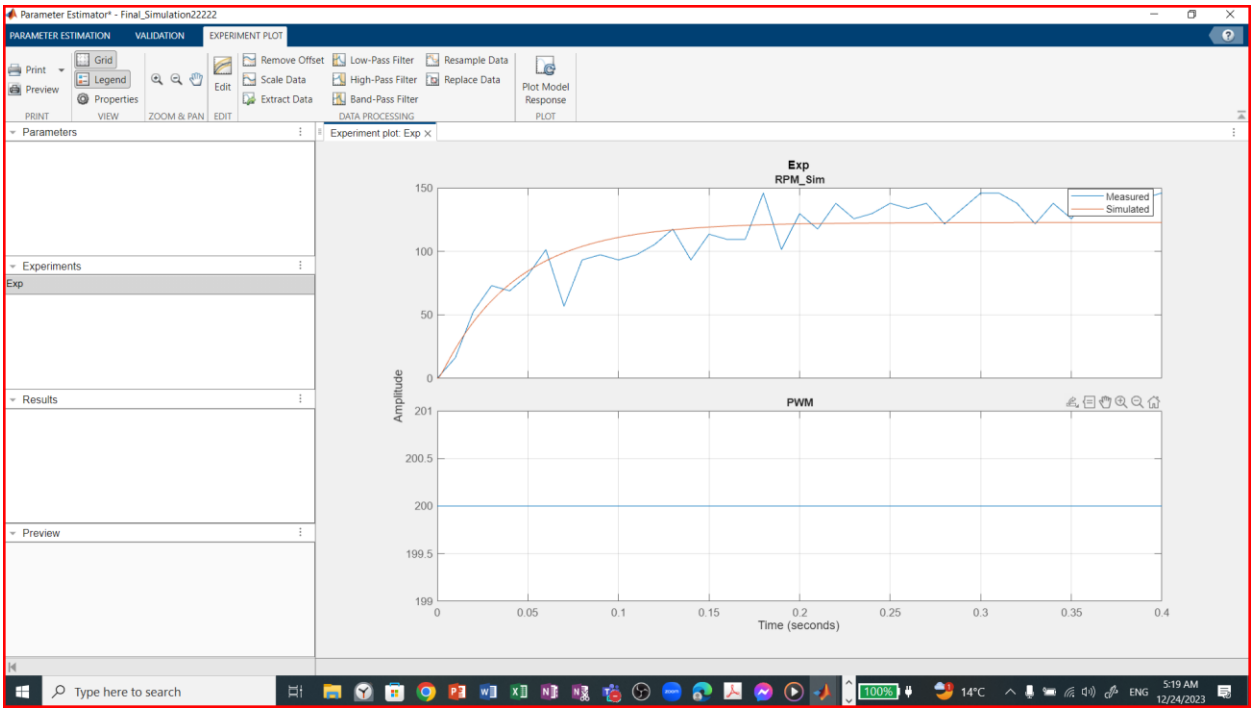
❖ 200 PWM

- Measured Graph

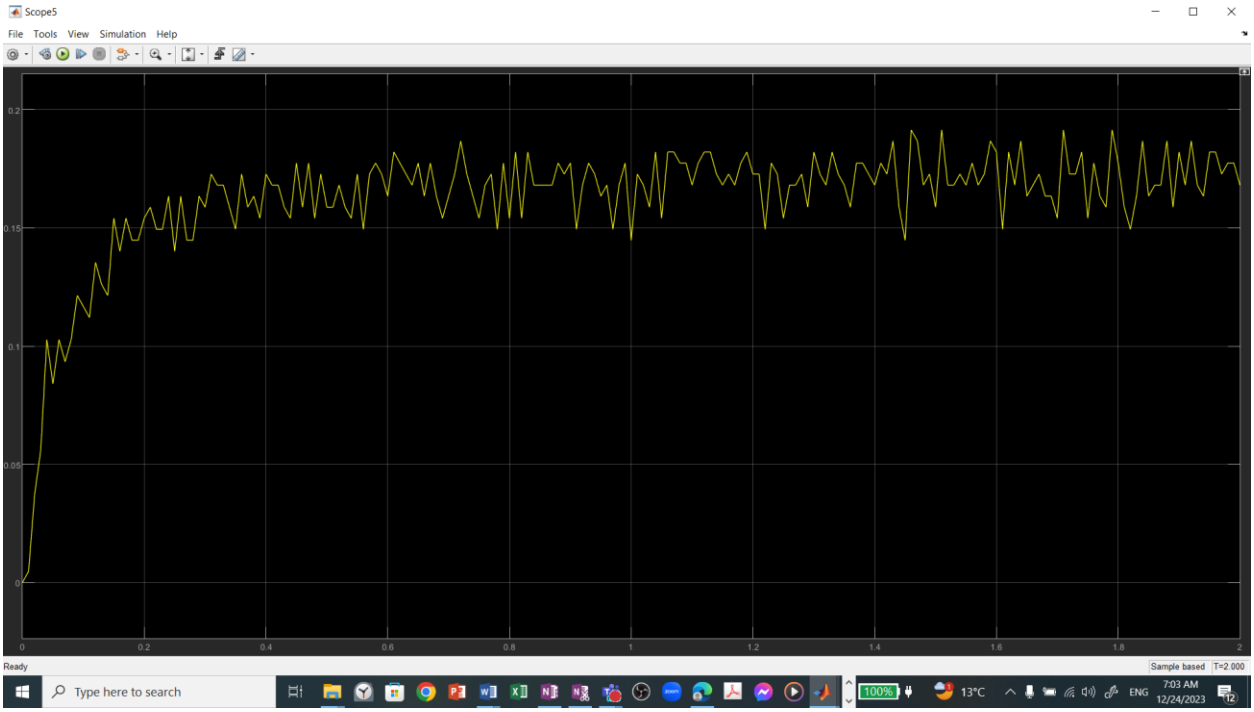


- Simulated Graph

- Measured & Simulated Graphs (200 PWM)

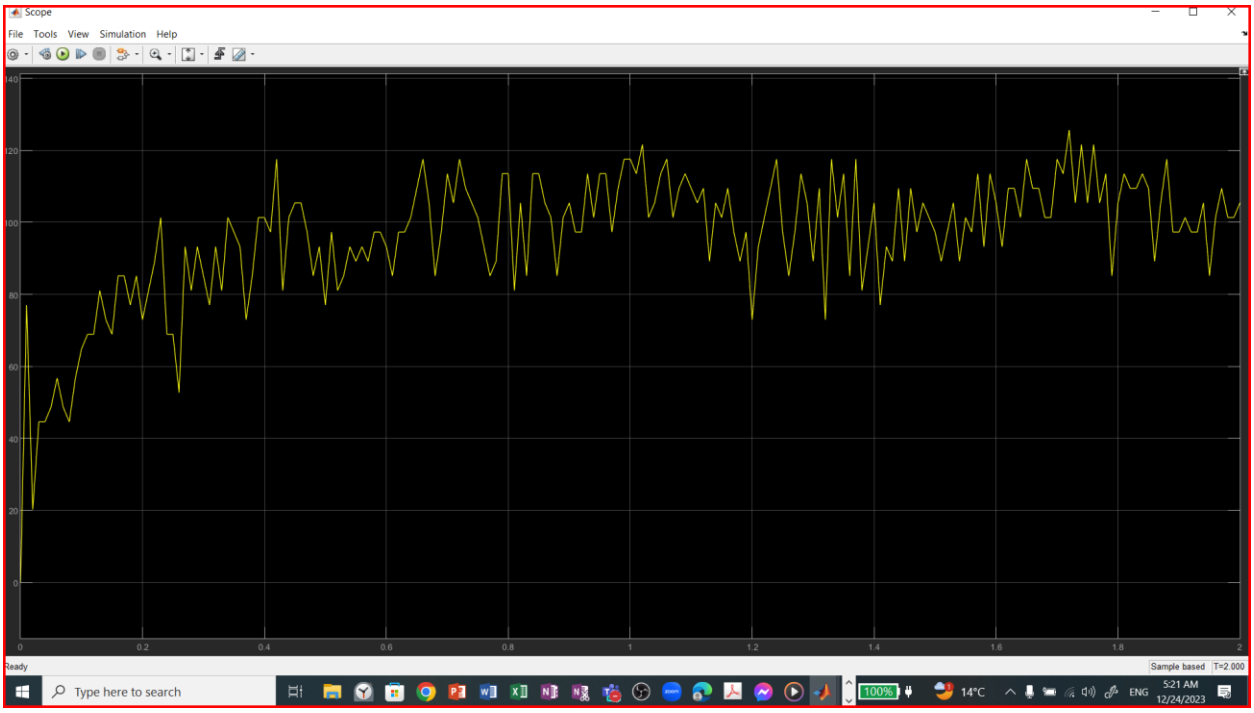


- Velocity



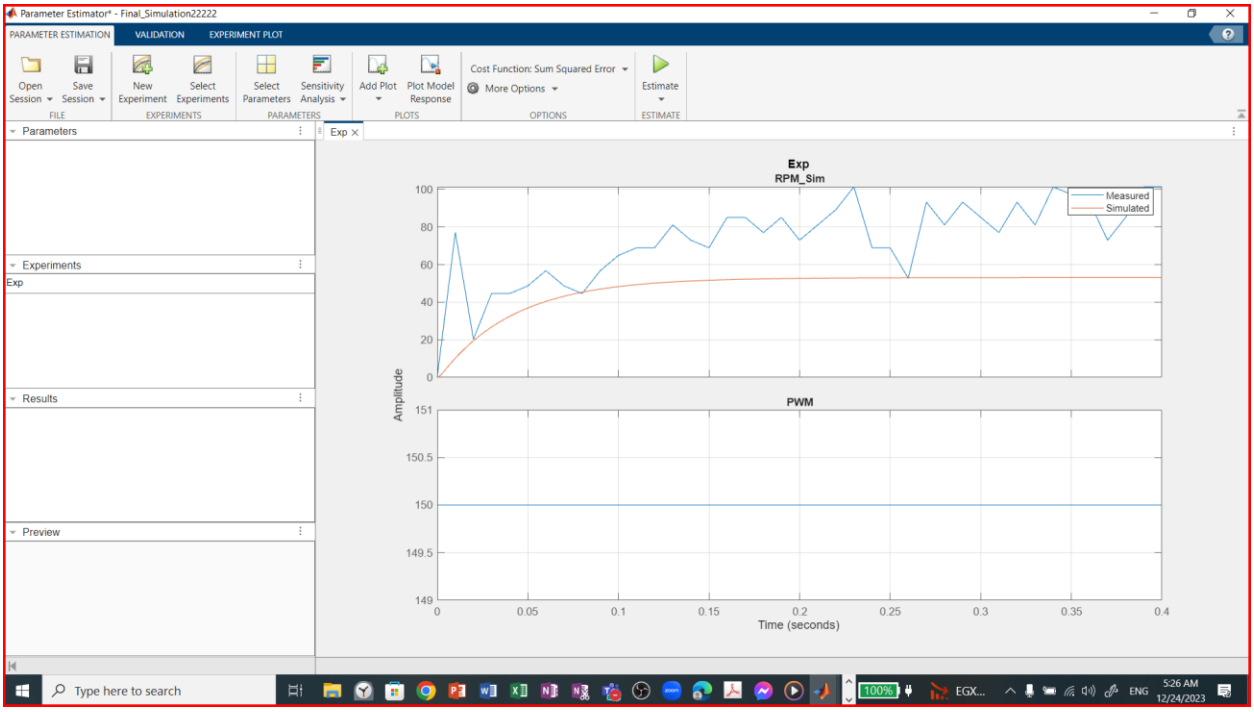
❖ 150 PWM

- Measured Graph



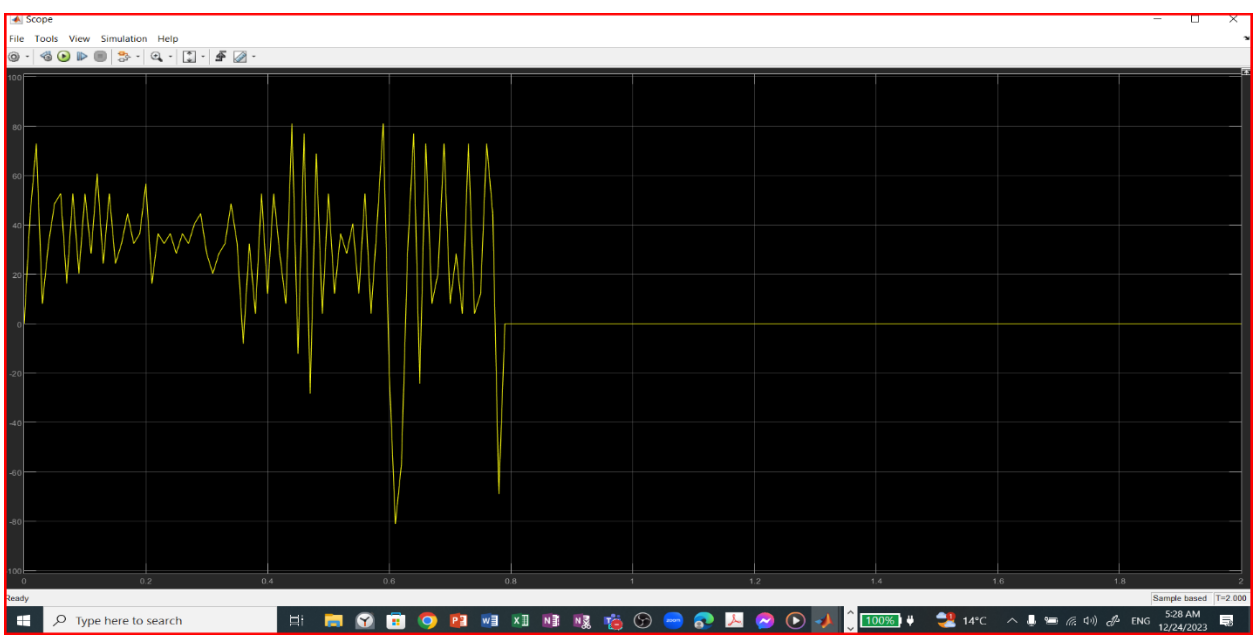
- Simulated Graph

- Graphs without parameter estimation



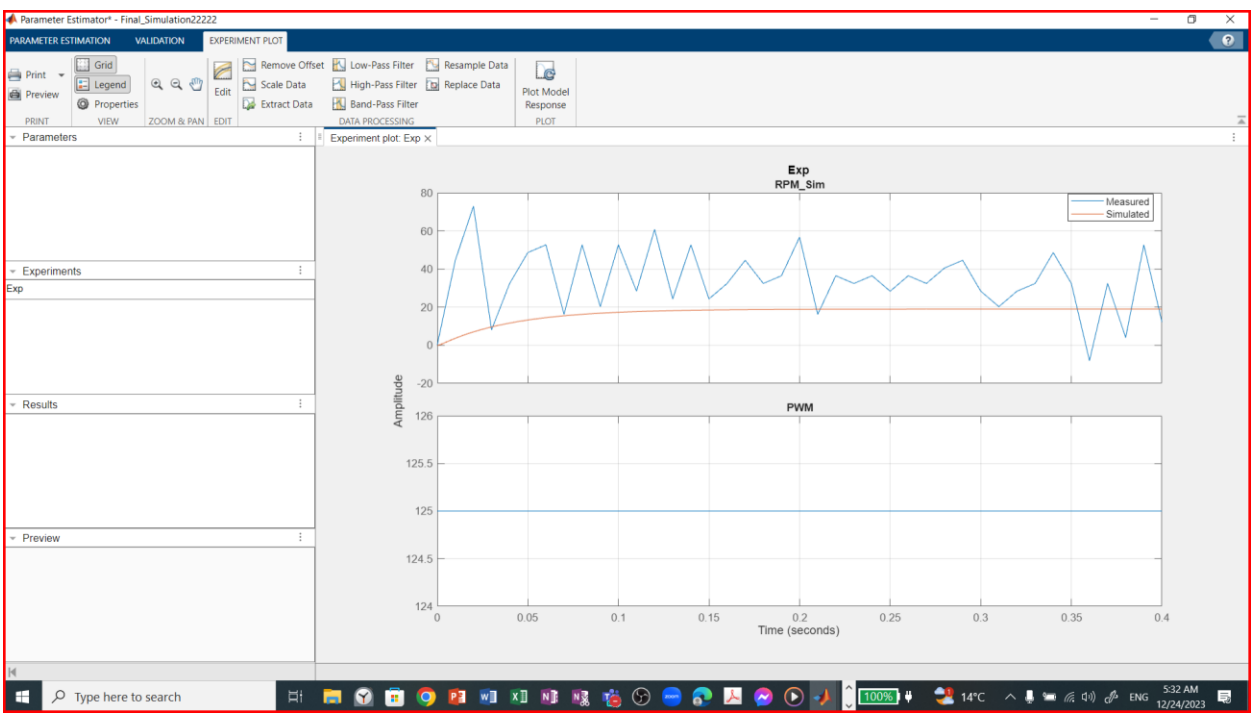
❖ 125 PWM

- Measured Graph



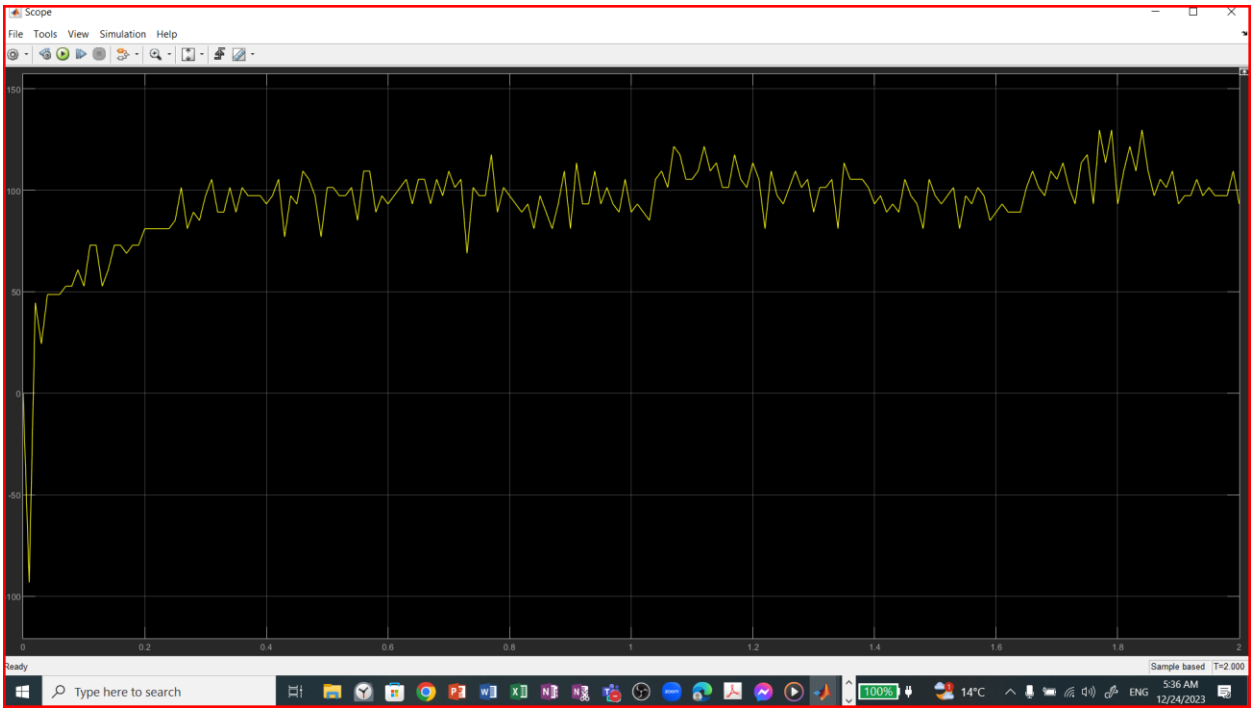
- Simulated Graph

- Graphs without parameter estimation:

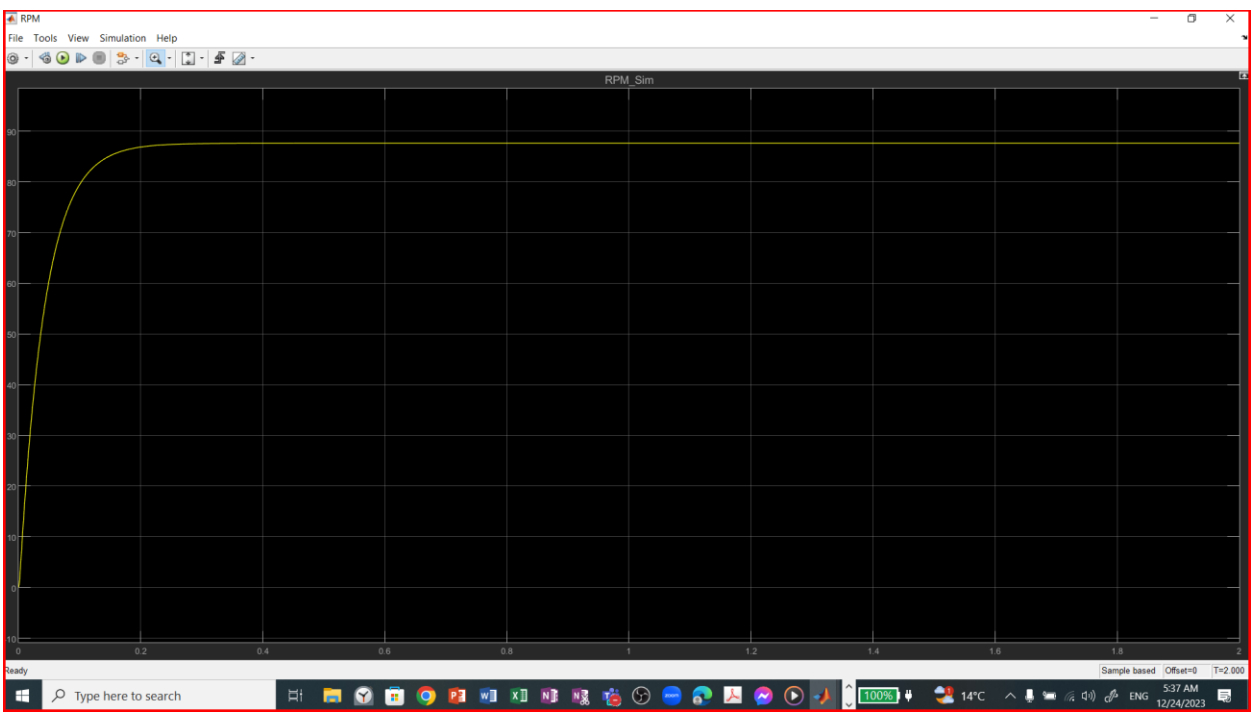


❖ 175 PWM

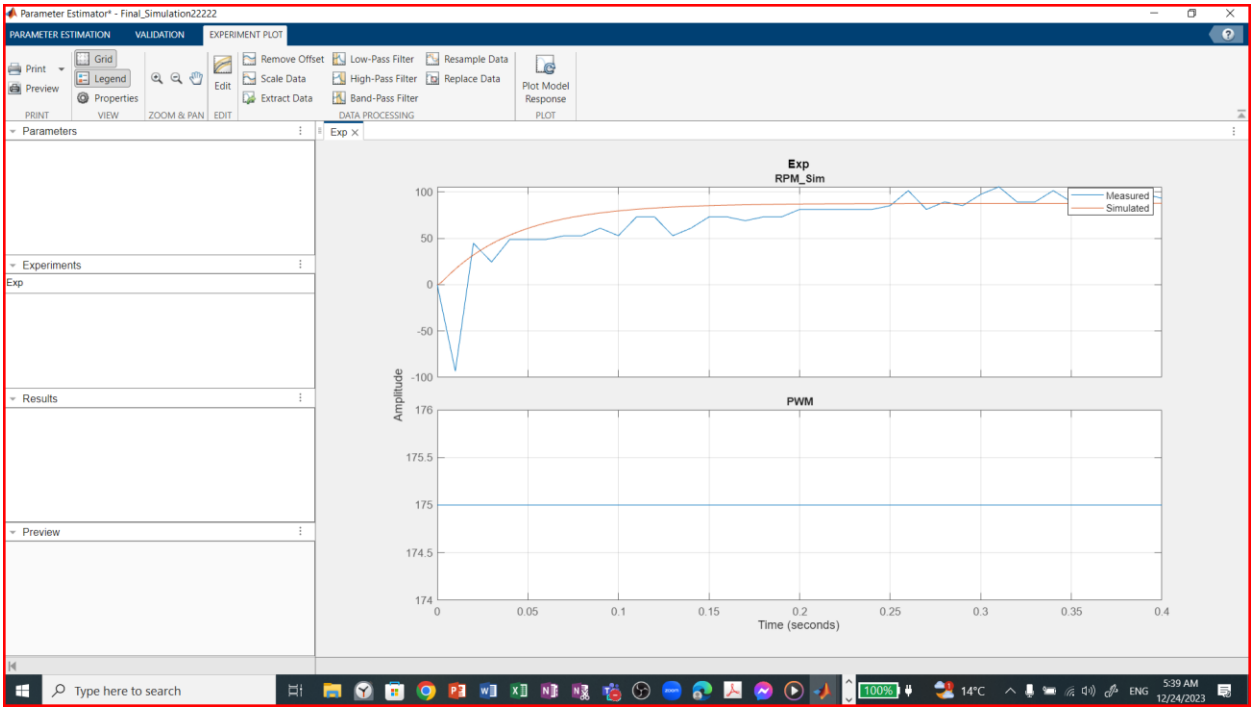
- Measured Graph



- Simulated Graph

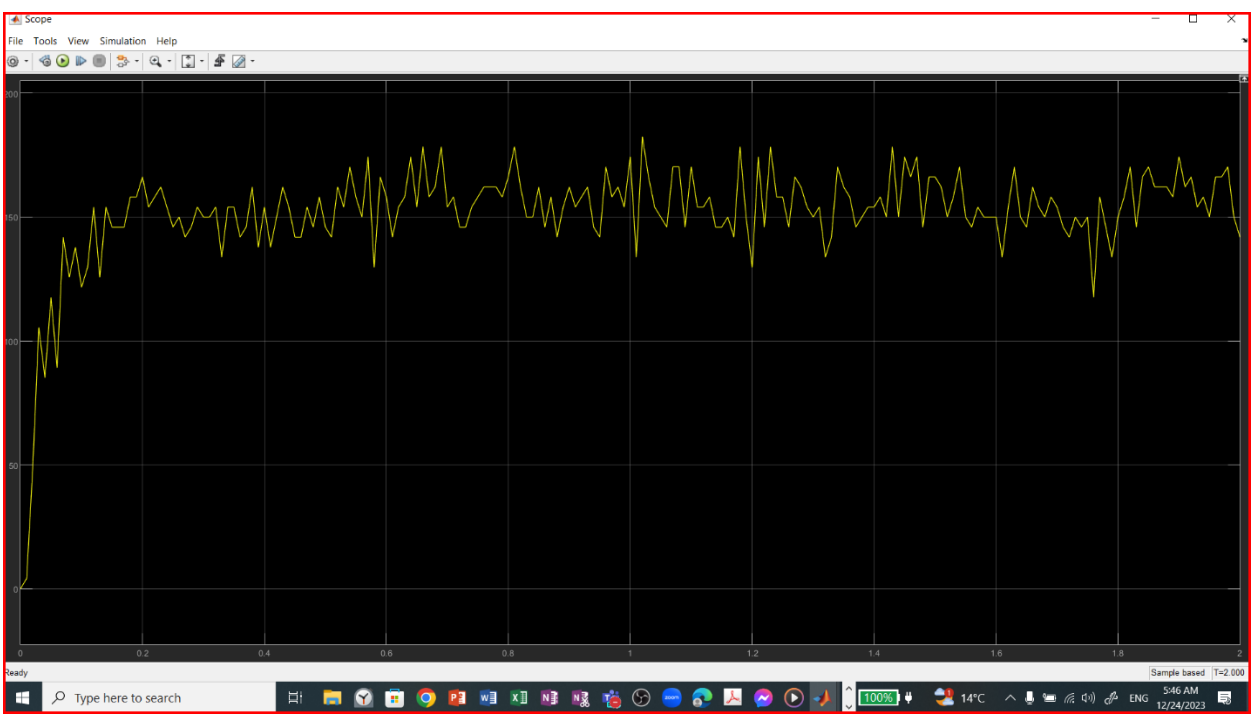


- Graphs without estimation:

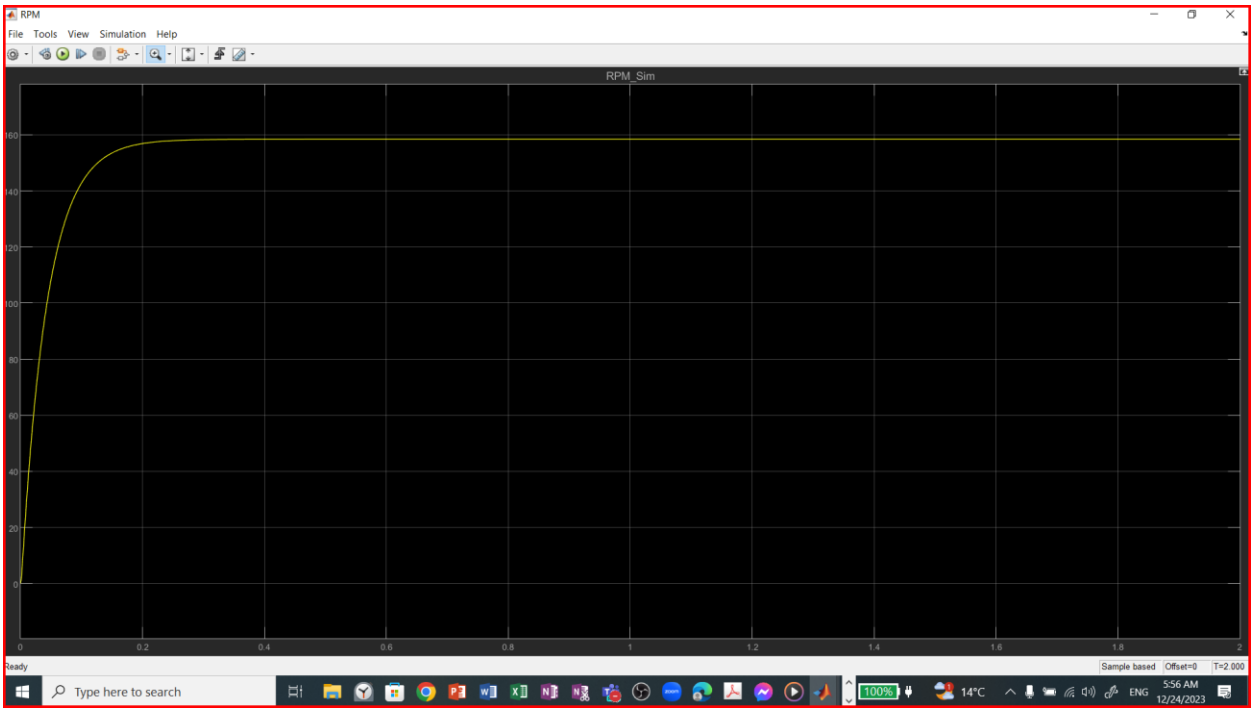


❖ 225 PWM

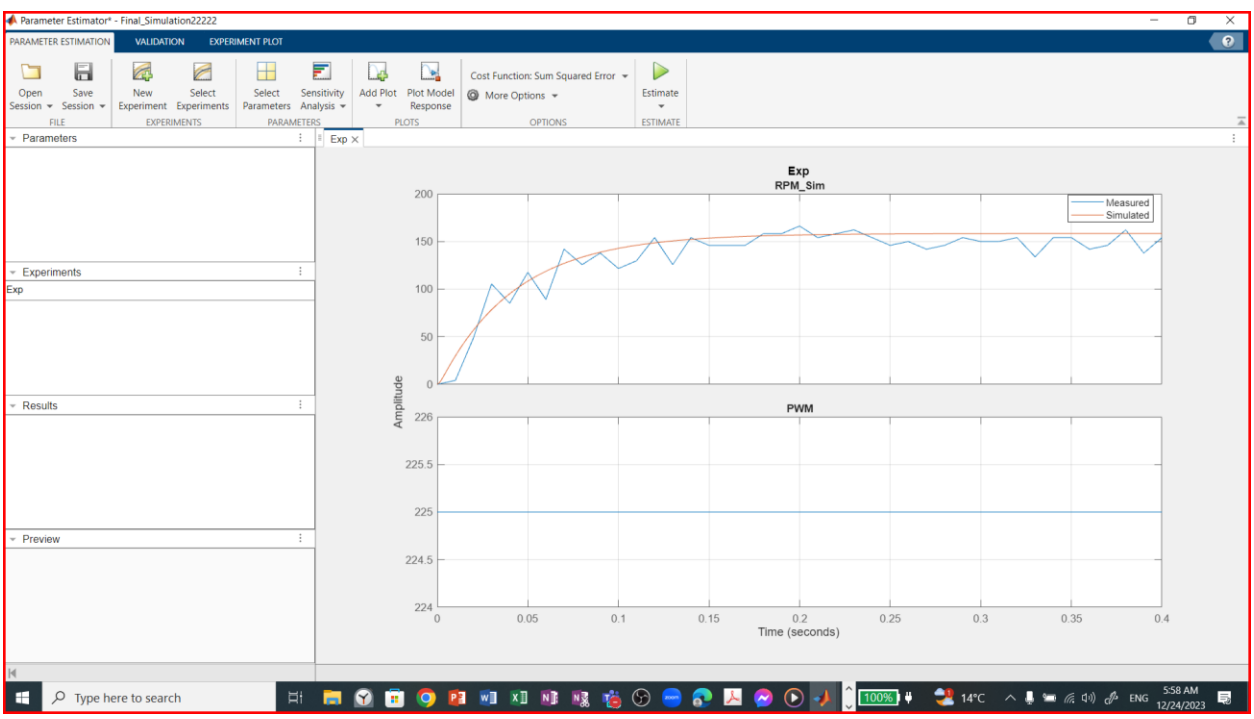
- Measured Graph



- Simulated Graph:



- Graphs without estimation:



Comparing results of physical model & simulation

Overall, at high PWMs, both graphs look identical to each other with the same system parameters. But at low PWMs, they don't exactly coincide to each other, but the graphs are very close. This maybe be to at low PWMs, noise has greater effect in the readings. Eventually, the parameter estimation succeeded in making the simulation model resemble our physical model.

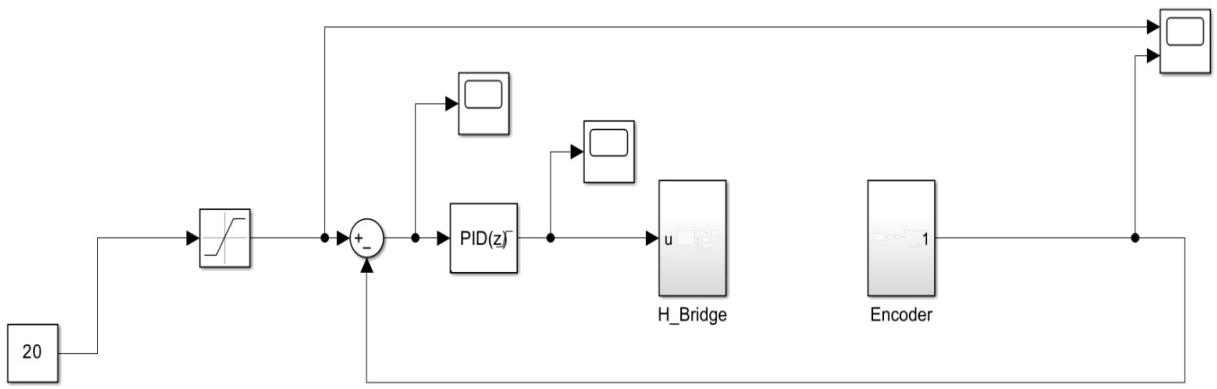
Now, After finishing system modeling, Starting to control the system.

Controlling Distance (Phase 1)

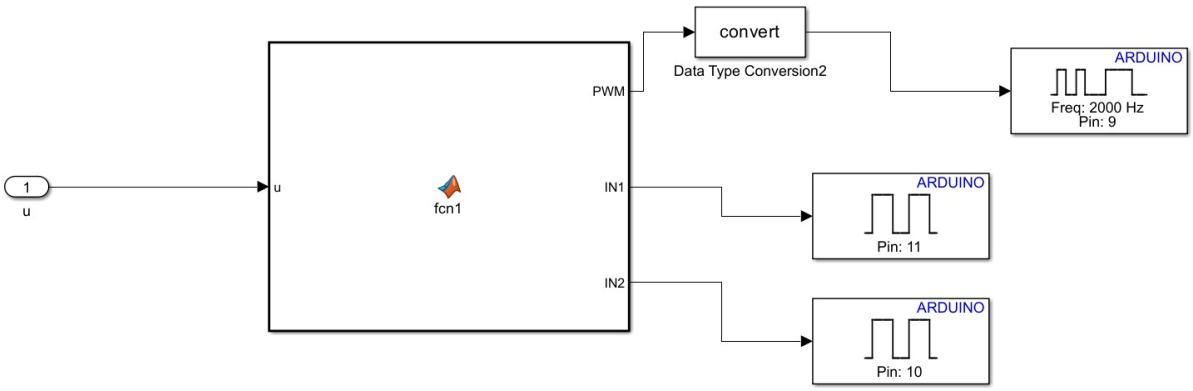
We require precise movement of the linear cart to reach the specified position provided to the system, accomplished through implementation of a PID controller.

- Initially, in the main model and simulation setup, we integrated the PID block.
- We established a unity feedback loop where the desired position served as the input.
- Our objective was for the output to precisely reach this input by fine-tuning the PID values.
- Through iterative processes, we systematically adjusted the PID parameters, guided by graphical analysis.
- Our aim was to optimize the output graph to closely match the input signal.
- Additionally, we developed a MATLAB function to invert the cart's direction when the error value was negative.
- Furthermore, we implemented input saturation to limit the input range, considering our system dimensions at hardware.

Hardware Model



H-Bridge subsystem



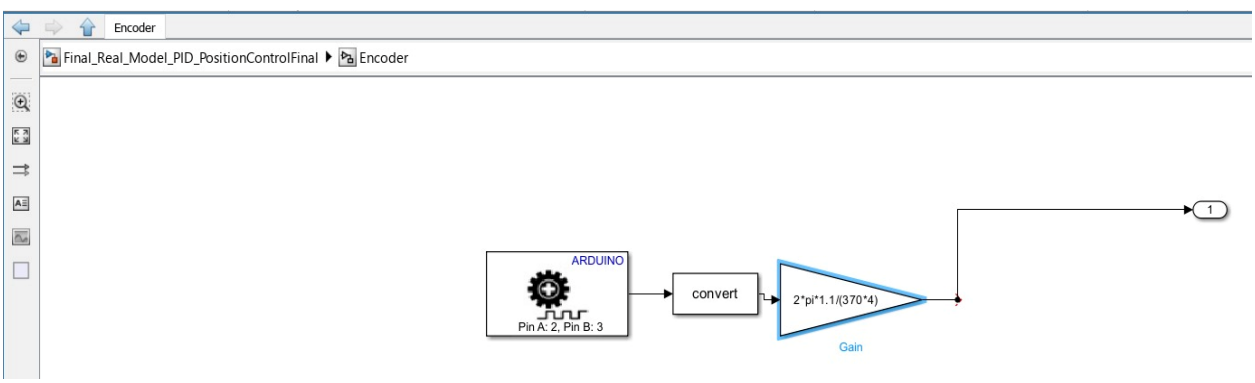
MATLAB Function

MATLAB Function

Final_Real_Model_PID_PositionControlFinal ▸ H_Bridge ▸ MATLAB Function

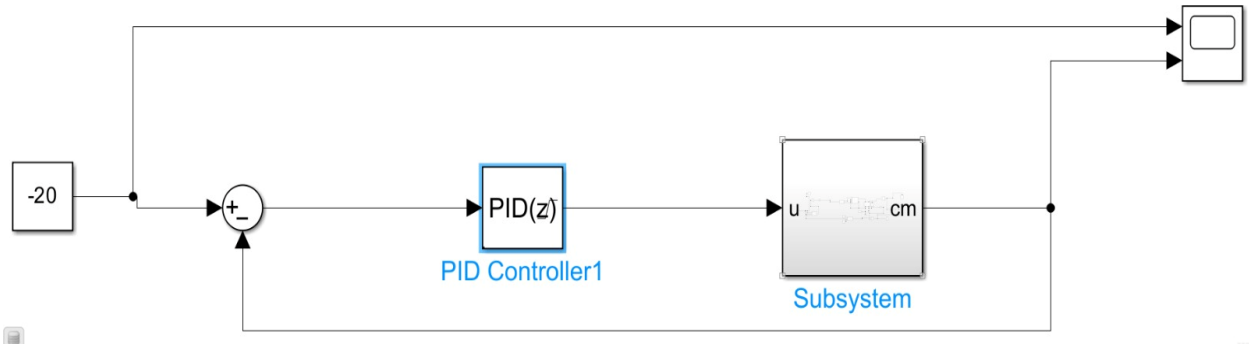
```
function [PWM,IN1,IN2]=fcn1(u)
PWM= abs(u);
if u<0
IN1=1; IN2=0;
elseif u>0
IN1=0; IN2=1;
else
IN1=0; IN2=0;
end
```

Encoder Subsystem

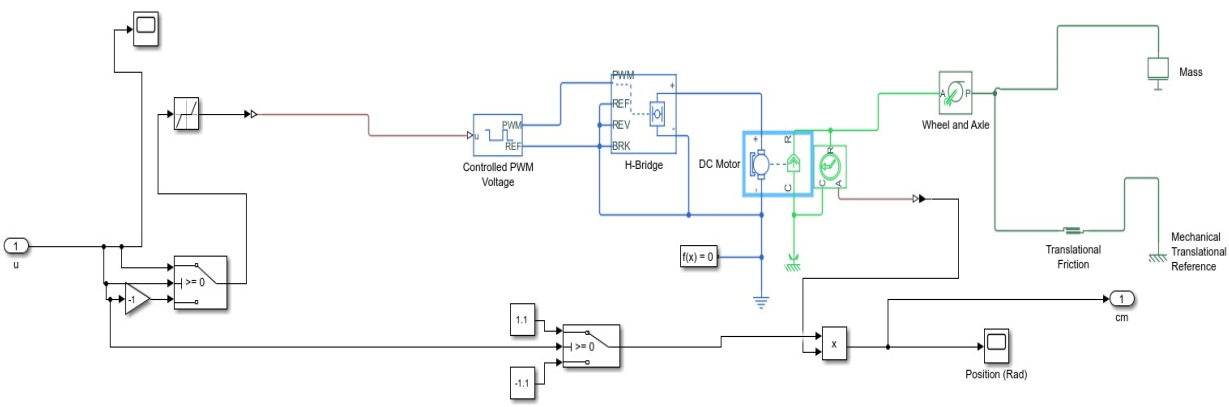


- The Gain to make the output position, The output from the encoder is pulses divide it by the encoder's pulses per revolution (ppr) multiplied by 4 since our encoder is a quadrature encoder (It uses 2 output channels to indicate the position of the motor) then multiplying by the pulley's radius* 2π .

Simulation



Simulated Model Subsystem

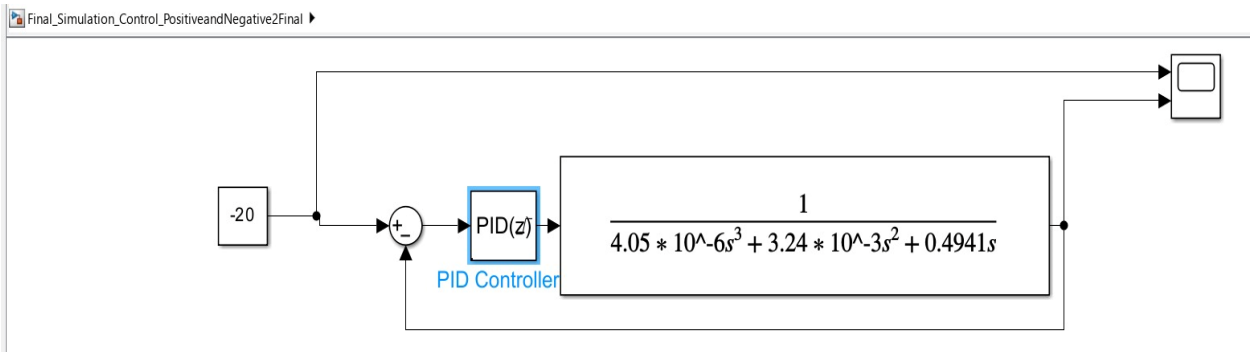


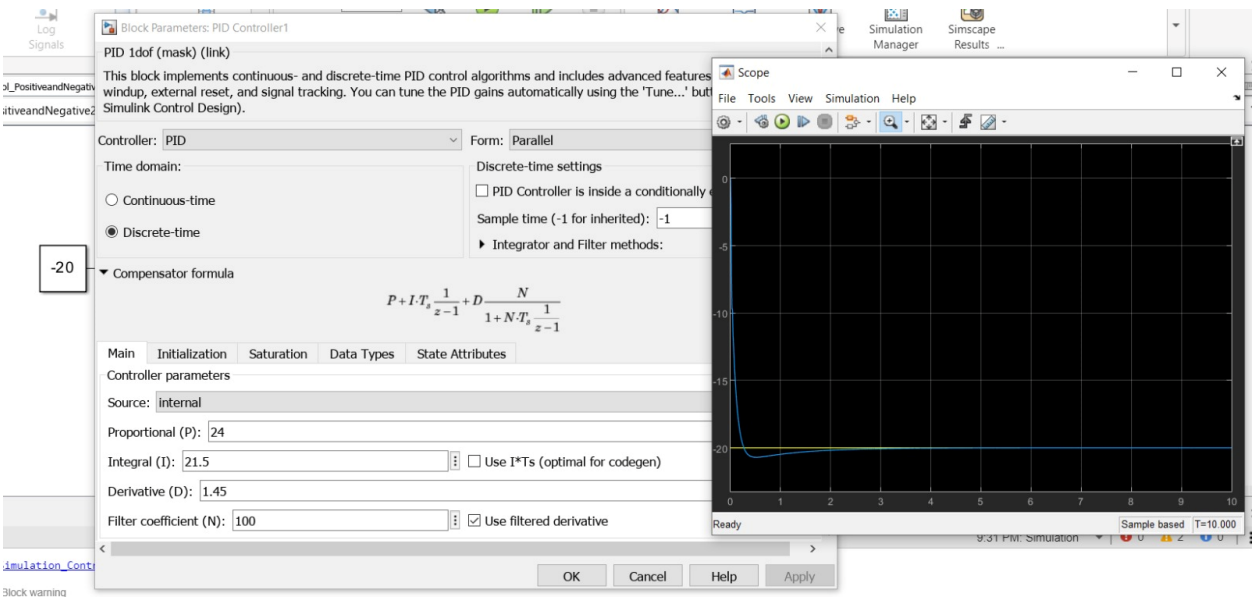
To ensure the simulation closely mirrored the behavior of the real system, we implemented two switches.

- The first switch, placed at the input, effectively transformed any negative input values into positive ones.
- Following this, another switch was positioned to account for negative inputs. When triggered, it depicted the graph in the negative region. This methodology was devised to maintain the accuracy of the dead zone block's functionality.

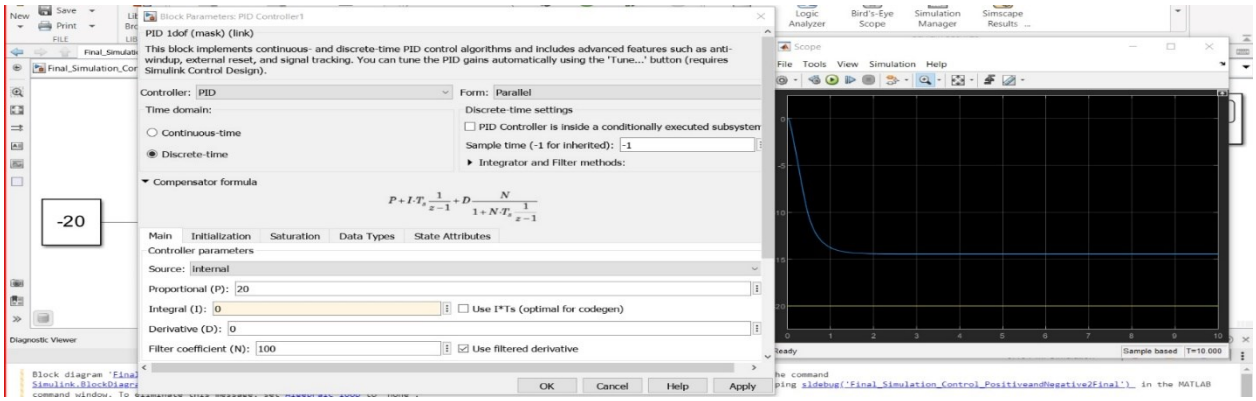
- Furthermore, within the simulation, we introduced a mechanism to reverse the system's direction. This reversal was reflected in the simulation's output graph, ensuring a faithful representation of the system's behavior.

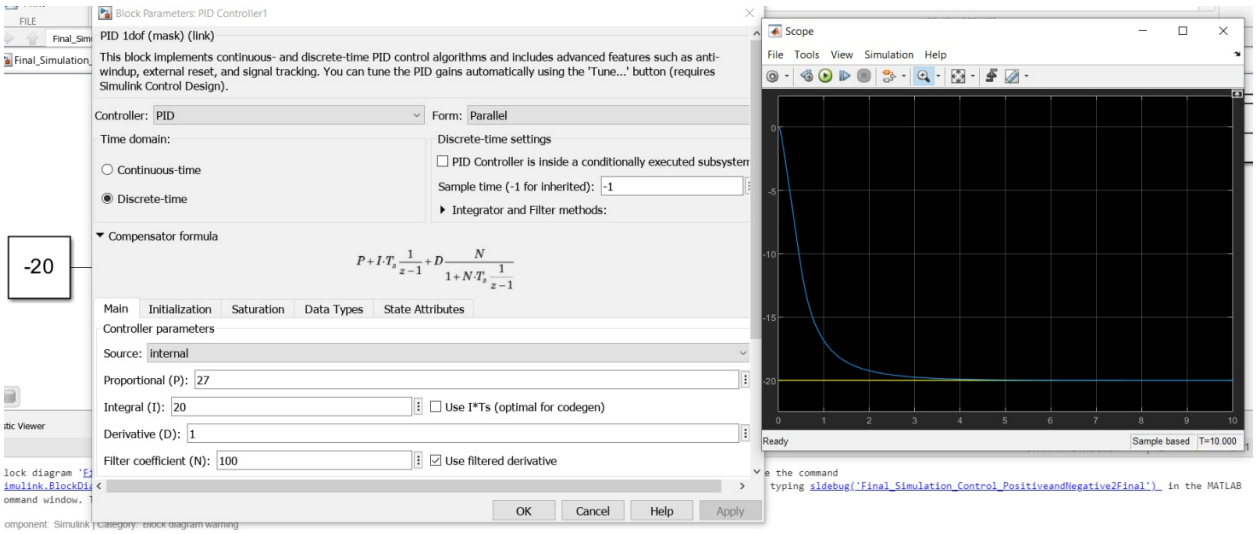
Model using Transfer function made manually

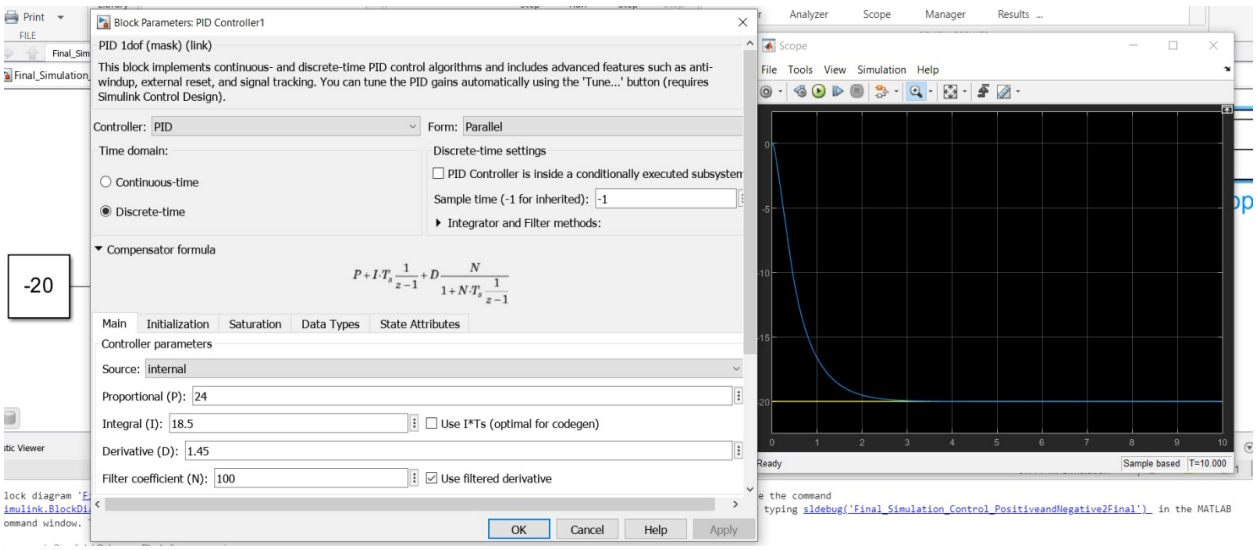




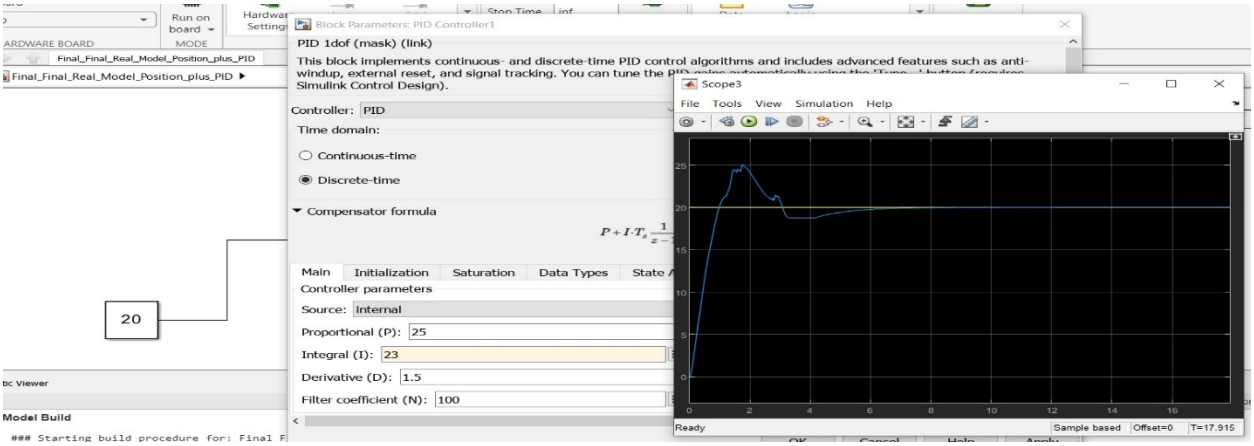
Tuning the PID values in Simulation

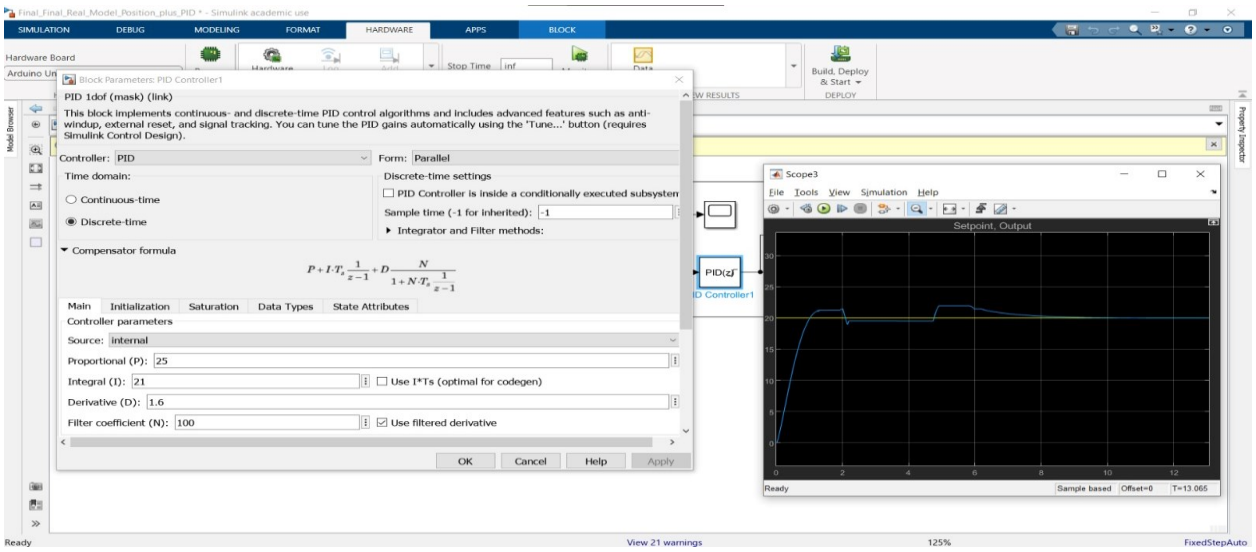


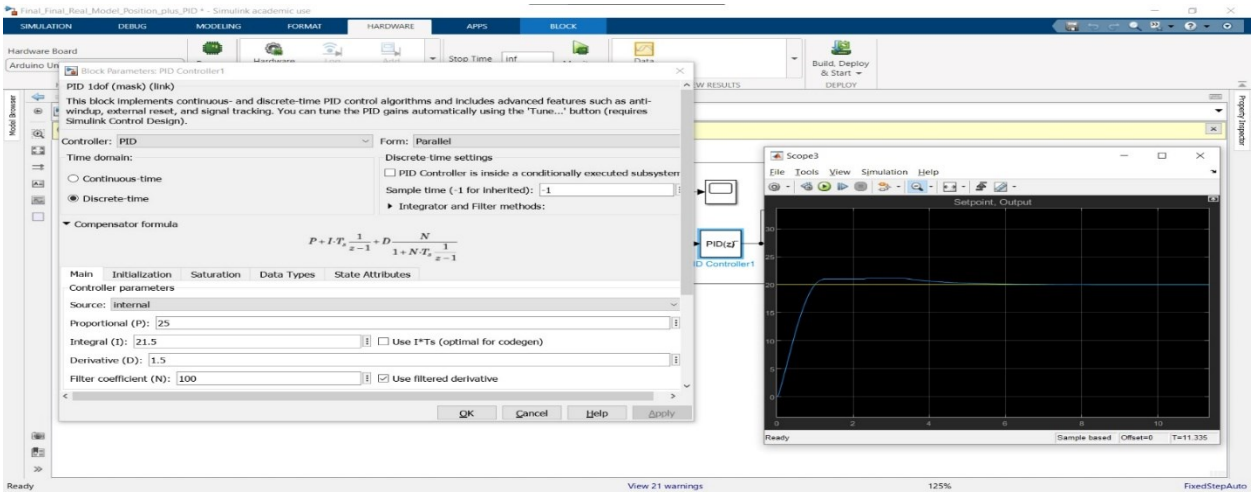


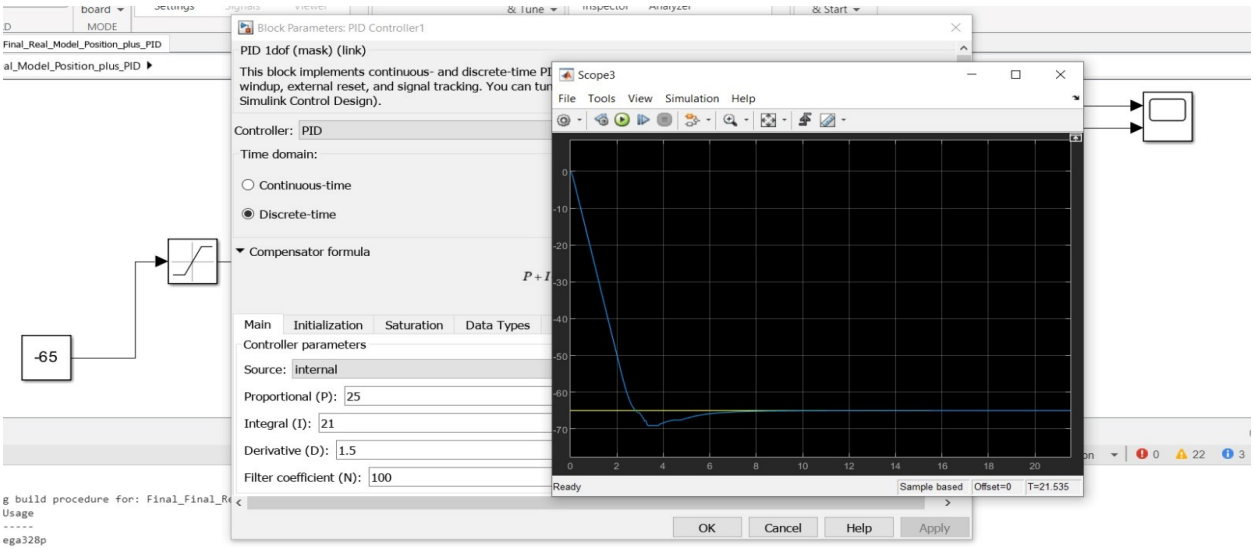


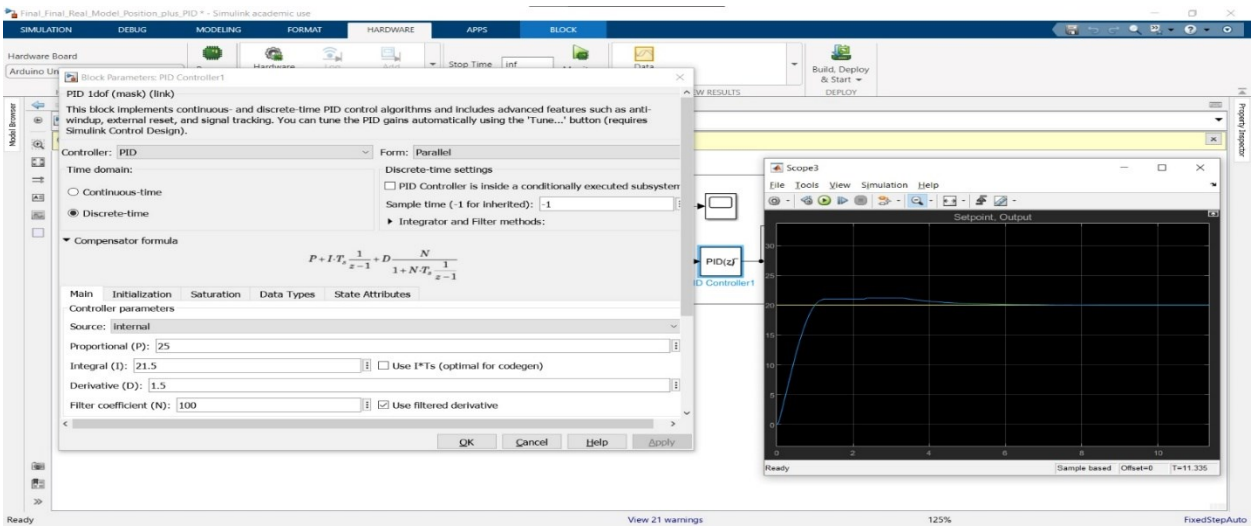
Tuning the PID values in Hardware

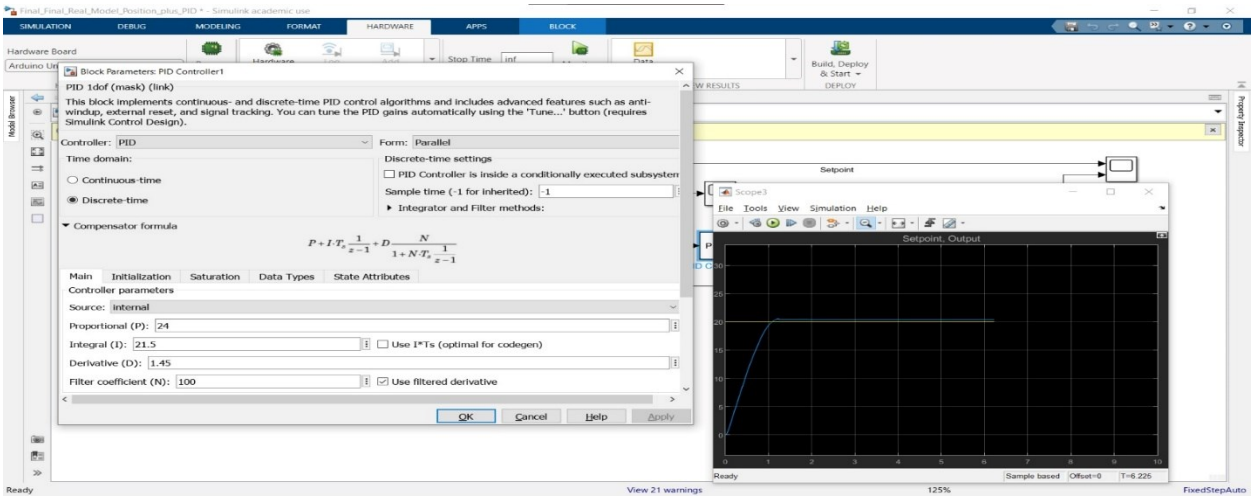


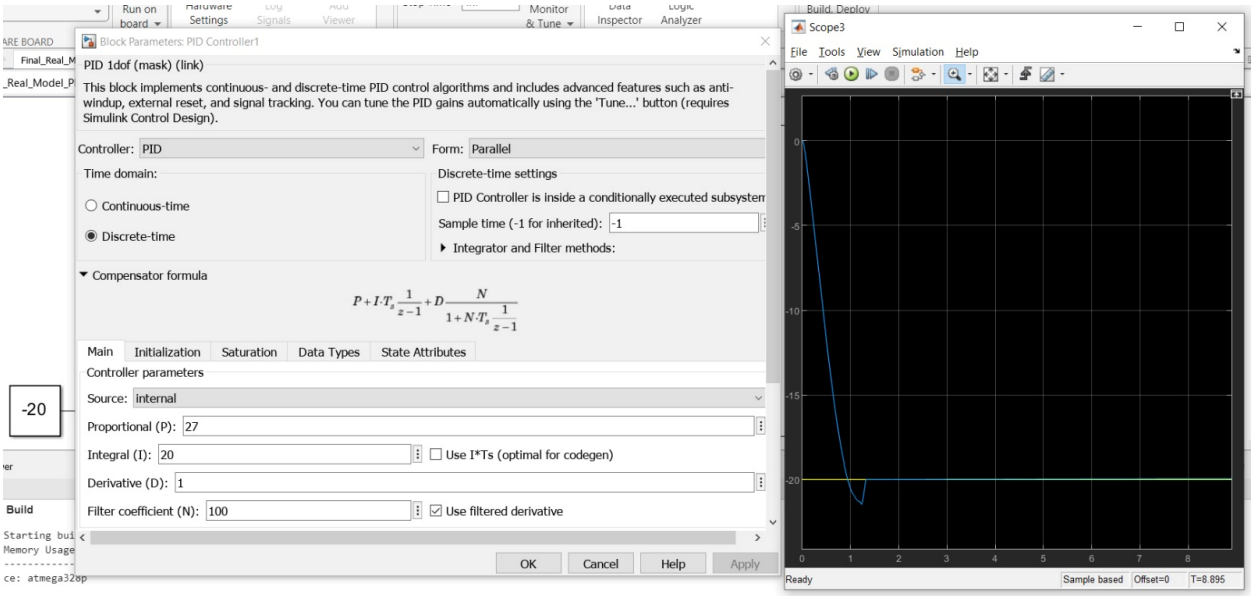


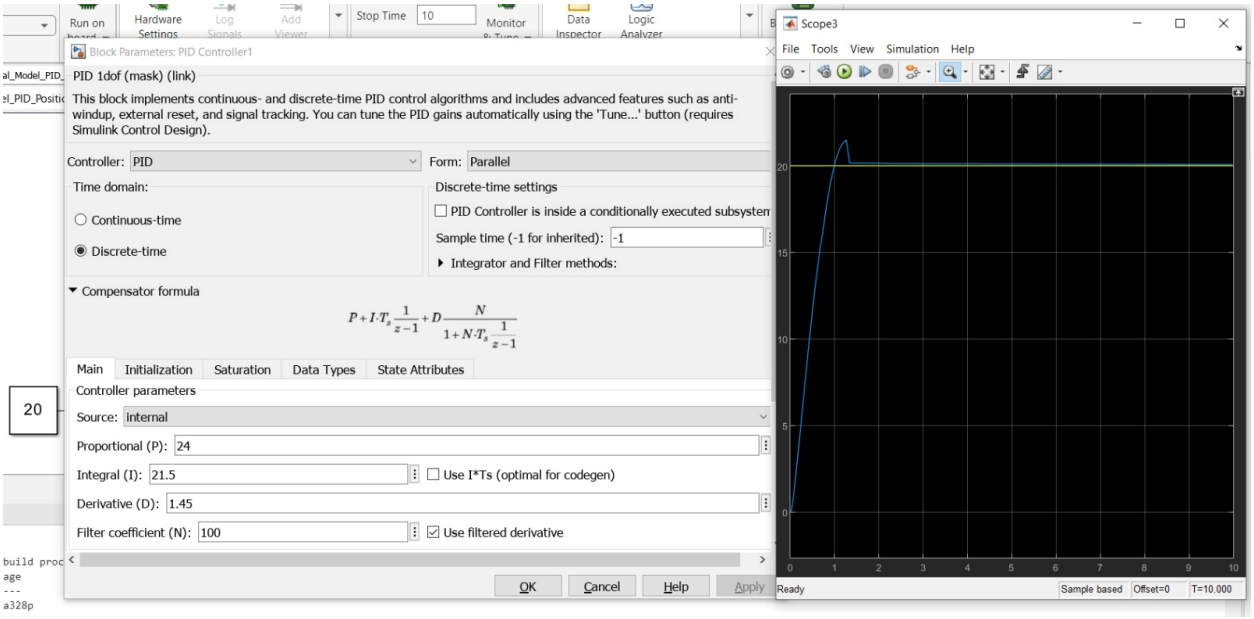



Block Parameters: PID Controller1
PID 1dof (mask) (link)
This block implements continuous- and discrete-time PI
windup, external reset, and signal tracking. You can tun
Simulink Control Design).
Controller: PID
Time domain:
Continuous-time
Discrete-time
Compensator formula
Main
Initialization
Saturation
Data Types
Controller parameters
Source: internal
Proportional (P): 25
Integral (I): 21
Derivative (D): 1.5
Filter coefficient (N): 100
OK
Cancel
Help
Apply
Scope3
File Tools View Simulation Help
Ready
Sample based
Offset=0
T=21.535
-65


Block Parameters: PID Controller1
PID 1dof (mask) (link)
This block implements continuous- and discrete-time PID control algorithms and includes advanced features such as anti-windup, external reset, and signal tracking. You can tune the PID gains automatically using the 'Tune...' button (requires Simulink Control Design).
Controller: PID
Form: Parallel
Time domain:
Continuous-time
Discrete-time
Discrete-time settings
PID Controller is inside a conditionally executed subsystem
Sample time (-1 for inherited): -1
Integrator and Filter methods:
Compensator formula
Main
Initialization
Saturation
Data Types
State Attributes
Controller parameters
Source: internal
Proportional (P): 25
Integral (I): 21.5
Use I*Ts (optimal for codegen)
Derivative (D): 1.5
Filter coefficient (N): 100
Use filtered derivative
OK
Cancel
Help
Apply
Scope3
File Tools View Simulation Help
Setpoint, Output
Ready
Sample based
Offset=0
T=11.335
View 21 warnings
125%
FixedStepAuto

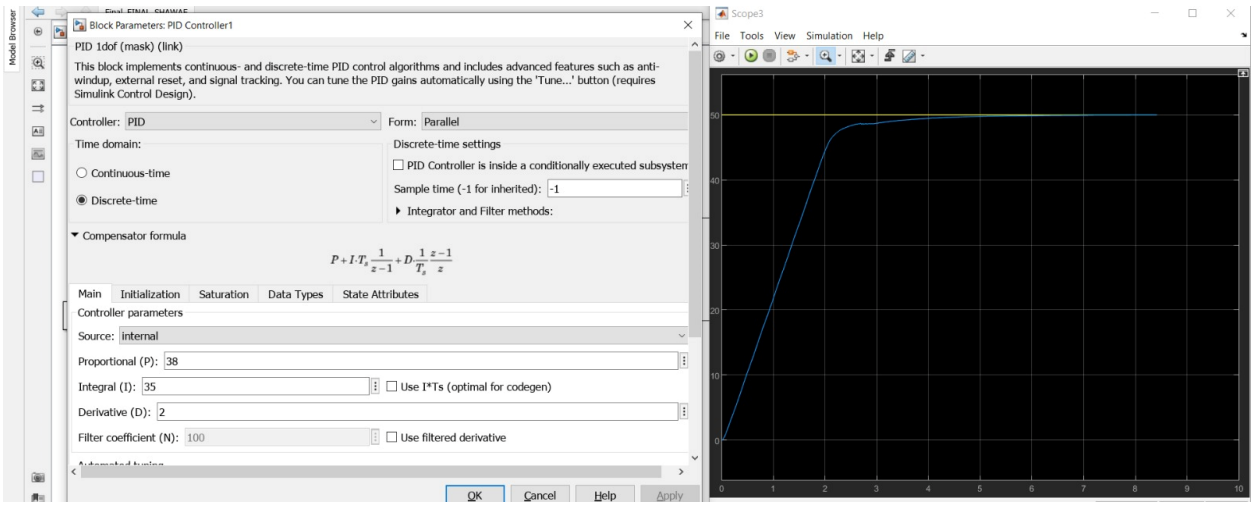
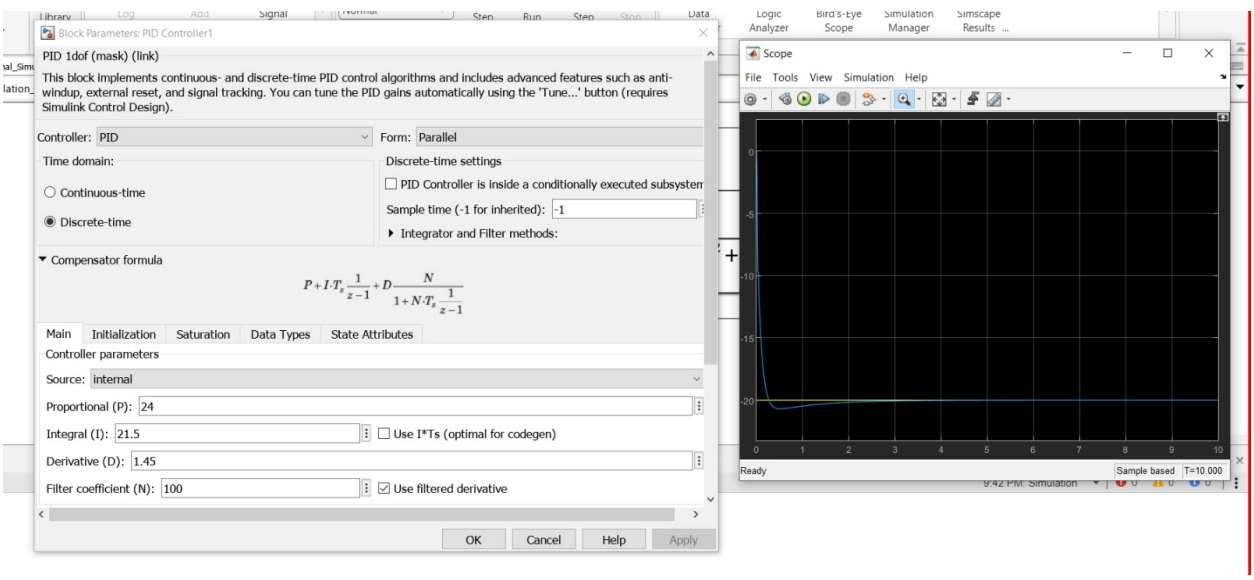

Block Parameters: PID Controller1
PID 1dof (mask) (link)
This block implements continuous- and discrete-time PID control algorithms and includes advanced features such as anti-windup, external reset, and signal tracking. You can tune the PID gains automatically using the 'Tune...' button (requires Simulink Control Design).
Controller: PID
Form: Parallel
Time domain:
Continuous-time
Discrete-time
Discrete-time settings
PID Controller is inside a conditionally executed subsystem
Sample time (-1 for inherited): -1
Integrator and Filter methods:
Compensator formula
Main
Initialization
Saturation
Data Types
State Attributes
Controller parameters
Source: internal
Proportional (P): 24
Integral (I): 21.5
Use I*Ts (optimal for codegen)
Derivative (D): 1.45
Filter coefficient (N): 100
Use filtered derivative
OK
Cancel
Help
Apply
Setpoint
Scope3
File Tools View Simulation Help
Setpoint, Output
Ready
Sample based
Offset=0
T=6.225
View 21 warnings
125%
FixedStepAuto


Block Parameters: PID Controller1
PID 1dof (mask) (link)
This block implements continuous- and discrete-time PID control algorithms and includes advanced features such as anti-windup, external reset, and signal tracking. You can tune the PID gains automatically using the 'Tune...' button (requires Simulink Control Design).
Controller: PID
Form: Parallel
Time domain:
Continuous-time
Discrete-time
Discrete-time settings
PID Controller is inside a conditionally executed subsystem
Sample time (-1 for inherited): -1
Integrator and Filter methods:
Compensator formula
Main
Initialization
Saturation
Data Types
State Attributes
Controller parameters
Source: internal
Proportional (P): 27
Integral (I): 20
Use I*Ts (optimal for codegen)
Derivative (D): 1
Filter coefficient (N): 100
Use filtered derivative
OK
Cancel
Help
Apply
-20
Scope3
File Tools View Simulation Help
Ready
Sample based
Offset=0
T=8.895


Block Parameters: PID Controller1
PID 1dof (mask) (link)
This block implements continuous- and discrete-time PID control algorithms and includes advanced features such as anti-windup, external reset, and signal tracking. You can tune the PID gains automatically using the 'Tune...' button (requires Simulink Control Design).
Controller: PID
Form: Parallel
Time domain:
Continuous-time
Discrete-time
Discrete-time settings
PID Controller is inside a conditionally executed subsystem
Sample time (-1 for inherited): -1
Integrator and Filter methods:
Compensator formula
Main
Initialization
Saturation
Data Types
State Attributes
Controller parameters
Source: internal
Proportional (P): 24
Integral (I): 21.5
Use I*Ts (optimal for codegen)
Derivative (D): 1.45
Filter coefficient (N): 100
Use filtered derivative
OK
Cancel
Help
Apply
20
Scope3
File Tools View Simulation Help
Ready
Sample based
Offset=0
T=10.000

Final Values of PID

Hardware:



Simulation: