SPC 428 : Mechatronics and Robotics

Servo base unit

Aya Sayed 202000279

Mohamed Moustafa 201800271

Mohamed Ghoraba 2019000795

Muhammad Ayman 201801356

Nouran Abdelmageed 201901755

Work Distribution

P.S: All the team members contributed in every aspect of this project, however, these were the main functionalities of each member.

Name	Contribution
Nouran Abdelmageed 201901755	Task: Mechanical setup and parameter estimation Report: mechanical setup, analysis and flowchart
Aya Sayed 202000279	Task : Circuit wiring and Current sensor code, Position and velocity “without PID controller” code, Parameter estimation Report: Result, Circuit wiring and Current sensor wiring.
Mohamed Moustafa 201800271	Report: Introduction, Part of Simulink model,Data Collection. Task: tried some tuning and Parameter estimation. Have worked on A reduced model for the Motor(We decided to move with the standard model)
Mohamed Ghoraba 2019000795	Task: Mechanical setup and parameter estimation Report: parameter estimation and code
Muhammad Ayman 201801356	The control task on simulink and the fine tuning on the physical system.

Introduction

Servo based applications are widely used in different industrial fields with different working mechanisms but the same principle which comprises a motor (AC or DC) and moving guide for the moving object. Servo applications always require a robust control system to manage different situations with minimum divergence from the default performance. In production lines, Servo units are used to maintain the movement of the objects via controlling the position of the objects referenced to a certain frame of reference and speed control to manage the rate of change in the positions of the objects to give another degree of freedom as certain application require certain speed for the moving objects to prevent overloading or deal with any liquids in case of any bottles filling stages. The velocity profile varies from application to another because of different requirements for each project but in general, the profile include four main stages, acceleration and deceleration stages, constant speed stage and dwel stage. The dwell stage is the main application dependent stage.

We have used a belt drive approach for the system to convert the rotational motion of the rotor inside the motor into transnational motion in the belt as it's designed to move horizontally. Some physical assessments have been applied on the motor to measure some properties such as the Breakaway friction torque and using the Motor data sheet to extract some properties like the stall torque and stall current.

Workflow

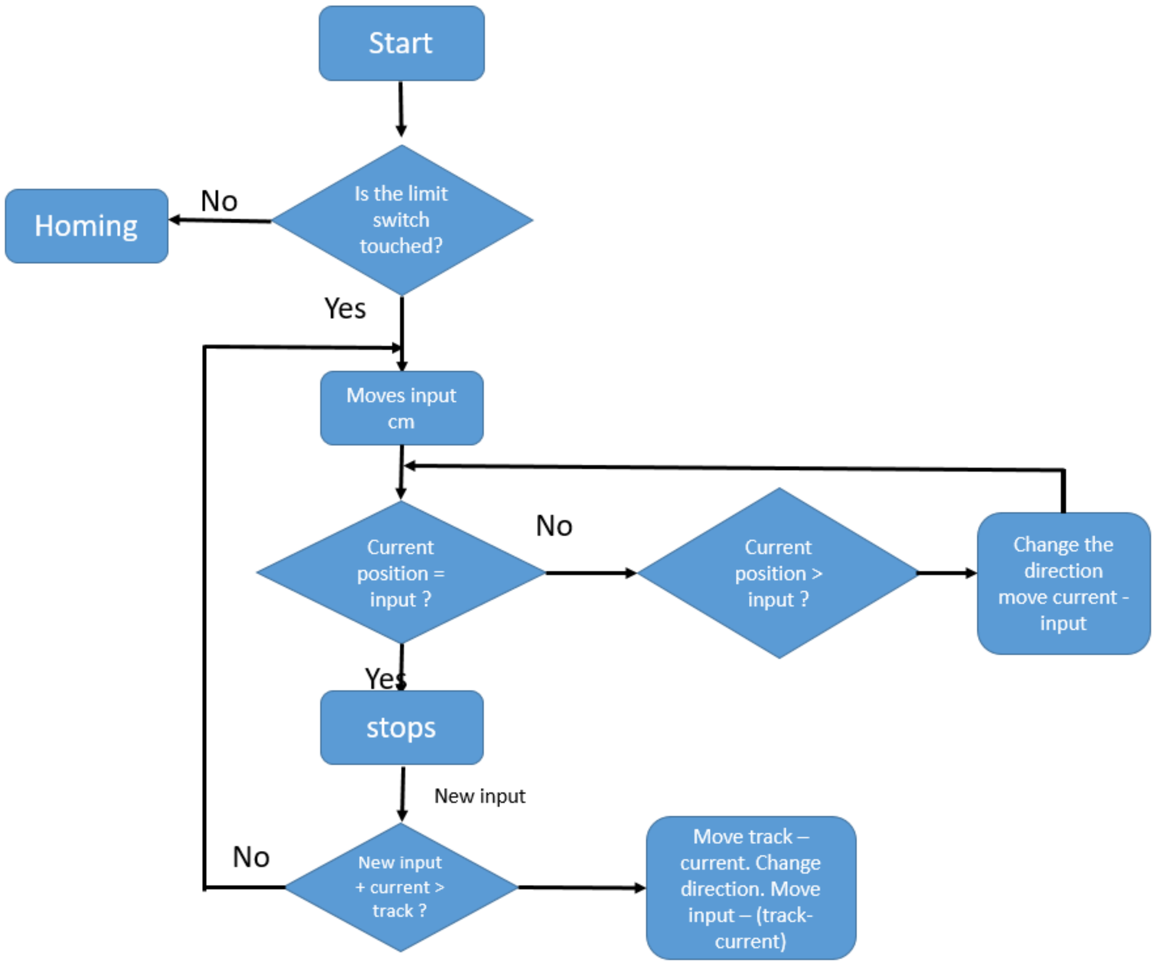

Start
Is the limit switch touched?
No
Homing
Yes
Moves input cm
Current position = input ?
No
Current position > input ?
Change the direction move current - input
Yes
stops
New input
No
New input + current > track ?
Move track – current. Change direction. Move input – (track-current)

figure 1 flowchart

Mechanical design

For the mechanical setup, our design was inspired by OpenBuilds linear actuator design attached below:

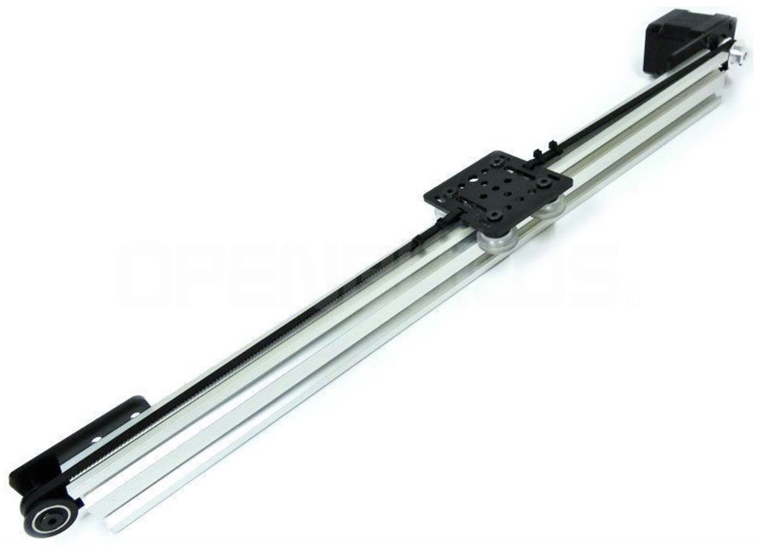


figure 2 setup

The main differences between our design and this one is that we are using an SG555123000 DC motor instead of a stepper motor and that our setup includes a limit switch.

The system consists of :

1. 1*A 73 rpm geared dc motor with encoder (SG555123000.pdf (ram-e-shop.com))
2. 1* limit switch
3. 1*1 m 20*20 V-slot aluminum extrusion bar.
4. 1*Mini-V gantry Plate.
5. 1*Idler Pulley Plate.
6. 1*Motor mount plate.
7. 4*Mini V-wheel bearing kit.
8. 1*Timing Pulley GT2.
9. 1*GT2 timing belt.
10. 1*Smooth idler pulley kit.

Main flow of the program

Circuit wiring

Encoder :

VCC → +5 v on motor driver L298

GND → GND on motor driver L298

Channel A → Arduino pin 2

Channel B → Arduino pin 4

Motor :

M + → Connected to OUT 1 on motor driver L298

M - → Connected to OUT 2 on motor driver L298

Arduino :

Pin 2 → Connected to Channel A on Encoder

Pin 3 → Connected to Limit Switch

Pin 4 → Connected to Channel B on Encoder

Pin 5 → Connected to ENA on motor driver L298

Pin 6 → Connected to IN1 on motor driver L298

Pin 7 → Connected to IN2 on motor driver L298

GND → Connected to GND on motor driver L298

Limit Switch :

Normally Open → Connected to Pin 3 on Arduino

Common terminal → Connected to GND on motor driver L298

motor driver L298 :

OUT 1 → Connected to M + on Motor

OUT 2 → Connected to M - on Motor

ENA → Connected to Pin 5 on Arduino

IN1 → Connected to Pin 6 on Arduino

IN2 → Connected to Pin 7 on Arduino

+5 V → Connected to VCC on Encoder

+12 V → Connected to +ve power supply

GND → Common ground for : 1- Common terminal on Limit Switch 2- GND on Arduino 3- GND on Encoder and 4- Connected to -ve power supply

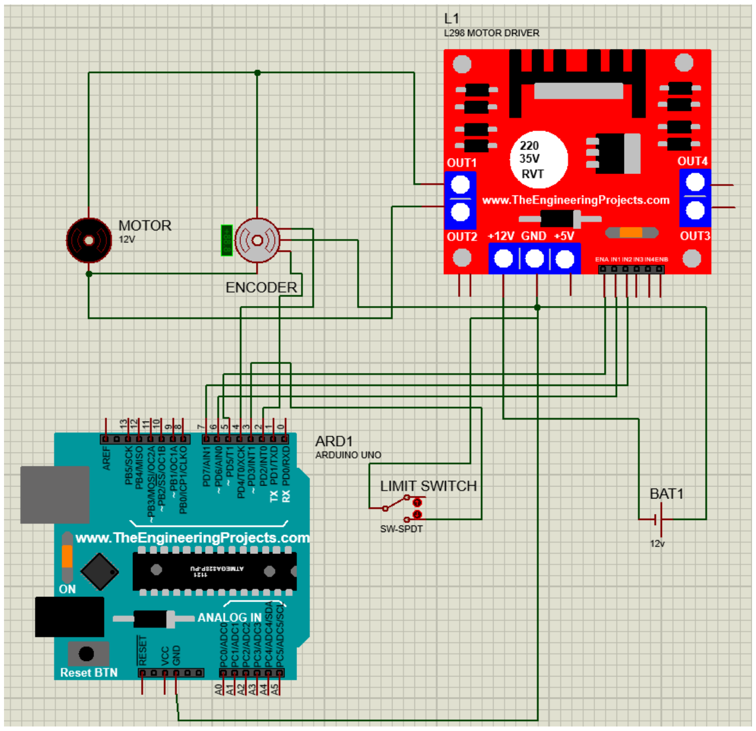


Figure 3 circuit wiring diagram

Current Circuit wiring

Encoder :

VCC → +5 v on motor driver L298

GND → GND on motor driver L298

Channel A → Arduino pin 2

Channel B → Arduino pin 3

Motor :

M + → Connected to +ve Ammeter

M - → Connected to OUT 1 on motor driver L298

Arduino :

Pin 2 → Connected to Channel A on Encoder

Pin 3 → Connected to Channel B on Encoder

Pin 5 → Connected to ENA on motor driver L298

Pin 6 → Connected to IN1 on motor driver L298

Pin 7 → Connected to IN2 on motor driver L298

A0 → Connected to OUT on Current Sensor

GND → Connected to GND on motor driver L298

Current sensor ACS712 :

IP+ → Connected to M - on Motor

IP- → Connected to GND on motor driver L298

VCC → Connected to +5 v on Arduino

OUT → Connected to A0 on Arduino

GND → Connected to GND on Arduino

motor driver L298 :

OUT 1 → Connected to M - on Motor

OUT 2 → Connected to IP+ on Current sensor ACS712

ENA → Connected to Pin 5 on Arduino

IN1 → Connected to Pin 6 on Arduino

IN2 → Connected to Pin 7 on Arduino

+5 V → Connected to VCC on Encoder and +5 V in Arduino

+12 V → Connected to +ve power supply

GND → Common ground for : 1- GND on Arduino 2- GND on Encoder 3- Connected to -ve power supply

Capacitor :

I also added a Capacitor 330µF to maintain the voltage level.

+ve → Connected to VCC on Current sensor ACS712 and VCC on Arduino

-ve → Connected to GND on motor driver L298 and GND on Arduino

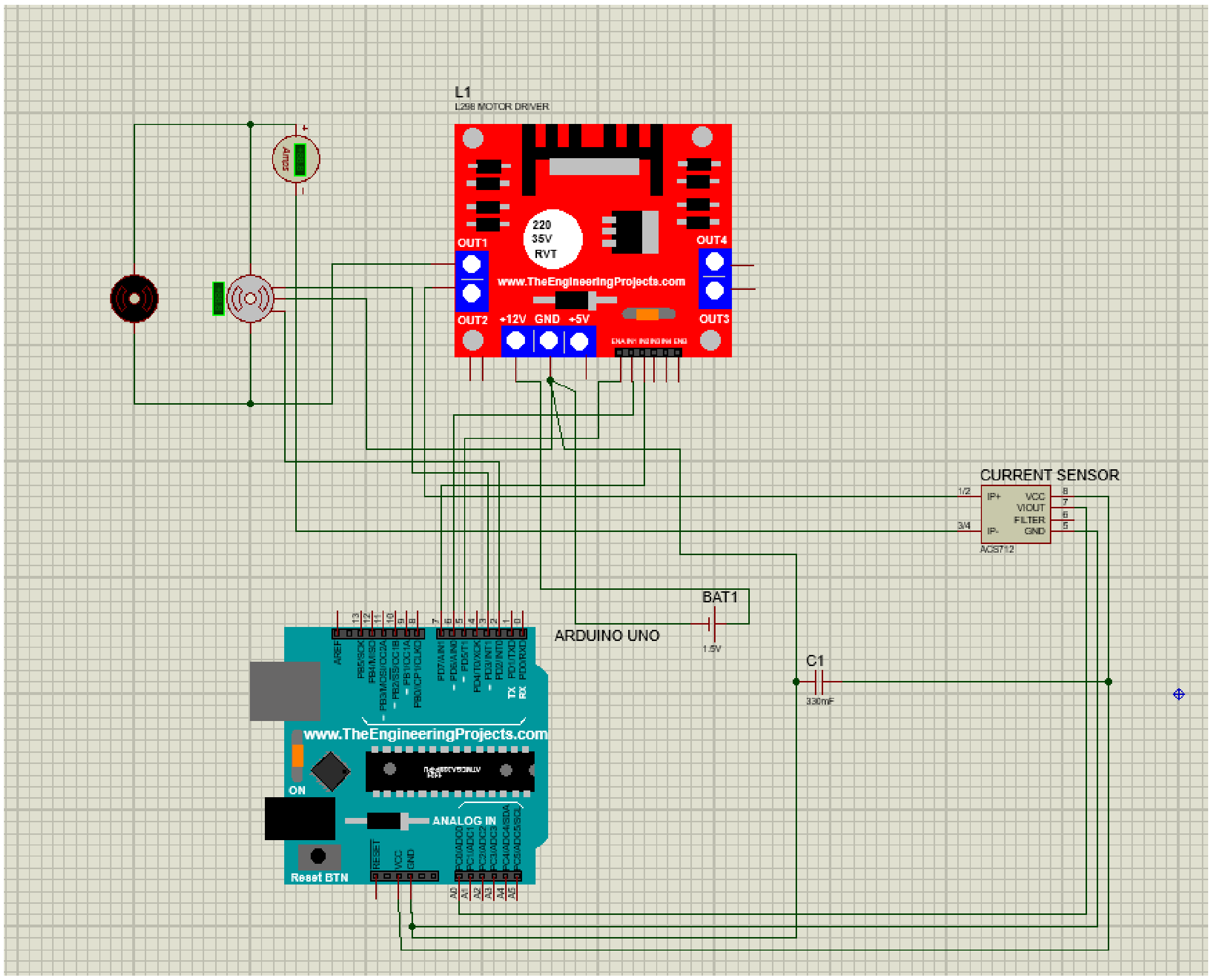


Figure 4 circuit diagram for the current sensor

Data Collection

For the best representation of the system, performance data has been collected from the motor by setting the input power for the motor to the maximum and measuring the position and the speed of the cart over time. This has helped in having intuitions about the performance of a motor once it is given power.

The velocity profile of the motor:

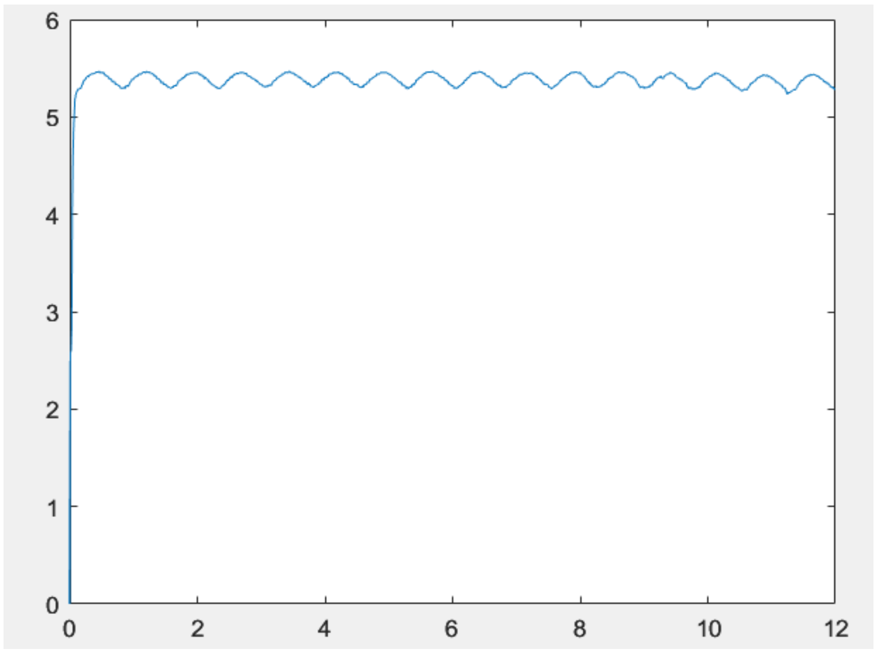


Figure 5

This plot shows velocity of the motor as an open loop response and using this graph, the time constant of the system could be calculated vai calculating the settling time, which is 0.12 s and so the time constant is 0.03s . The velocity of the motor is about 5.36 cm/s on average with a maximum value of 5.47 cm/s. A mathematical representation of the velocity profile could be used to predict the velocity profile for longer time period using this formula:

$$\text{Speed}(x) = 0.001077x^3 - 0.02291x^2 + 0.14x + 5.612$$

while x represents time in seconds

The open-loop response of the system has given us a guiding for the controllers gains as the time constant of the open-loop response must be the same time constant of the simulated model and so this is the criteria of validating the model design . $\tau = \frac{settling\ time}{4} = \frac{0.1212}{4} = 0.0303$ this value is accurate to the fourth decimal digits.

Simulink Model

The simulink model used for this project was the model introduced in this course namely linear servo based unit. This model takes as input PWM and its output is position, velocity and current. Our model for the system is more accurate and representative of the system's dynamics as we have managed to include the gear ratio of the motor before the output of the system as well the dead zone of the motor as we have done some experiments to measure the starting point for the motor in terms of the PWM and we have concluded that the dead zone for the motor is [-67: 67] PWM and so the motor will start work only once the PWM is greater than the absolute of 67. The gear ratio of our motor is 30. Some motor properties are calculated manually from the data sheet and others have been calculated using the optimization process in SIMULINK using the Parameter Estimation tool in SIMULINK.

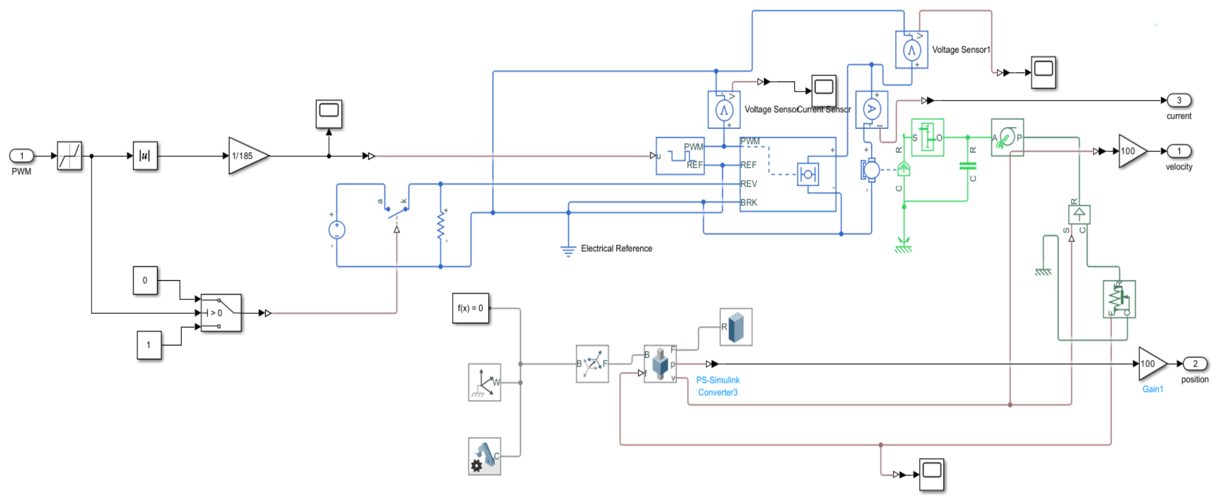


Figure 6 simulink model

Parameter Estimation

In order to correctly model our system, some parameters need to be determined first.

Some parameters were calculated accurately, other parameters were initially approximated and then estimated and others were totally estimated.

- Mass m: the mass of the cart was measured to be 0.11 Kg, however, the mass was included in the parameter estimation with a minimum of 0.1 KG and maximum of 0.2 Kg.
- Breakaway friction torque: to get this value the following was done:
  A very small volt was directly applied on the motor and it was raised until the point the cart starting moving and the current was recorded at this exact point to be 0.26 A, then this current was multiplied by the torque constant which was obtained from the data sheet for the geared motor to be 0.425 N/A. the final T_{st} value is 0.1105 N.
- Armature resistance Ra: to get this value, a volt of 5 V was applied on the motor while preventing the shaft from moving and the current was recorded to be 1.2 A, making $R_a = \frac{V_{cc}}{I_a} = 4.13\ \Omega$. This value was included in the parameter estimation with lower limit of 3.8 and upper limit of 4.4 ohm.

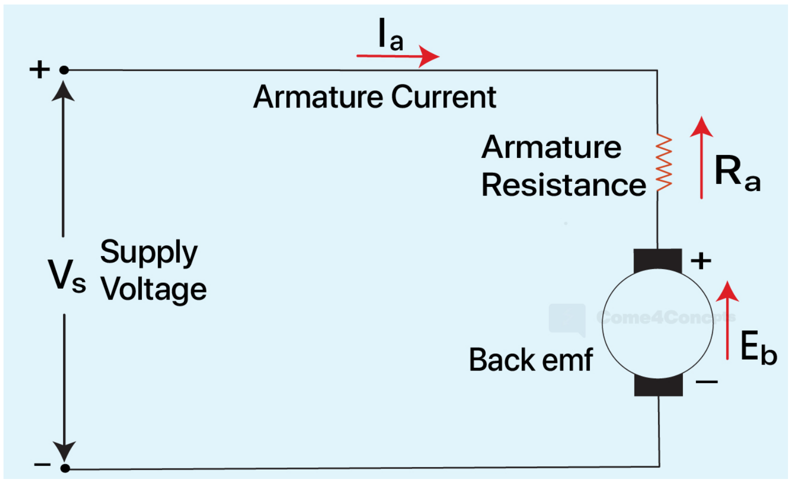


figure 7

- Motor inductance L: this value was estimated in the parameter estimation. An initial value was obtained to be 0.12 H by approximating the time constant to be L/R.
- Motor inertia J: this value was estimated in the parameter estimation.
- Viscous friction coefficient b: this value was estimated in the parameter estimation.
- Back-emf constant K_b: this value was initially calculated using the following equation:

$$K_b = \frac{V_{rated} - R_a * I_{noload}}{\omega_{rated}} = \frac{12 - 4 * 0.14}{3000 * 2\pi/60} = 0.0364$$

Results

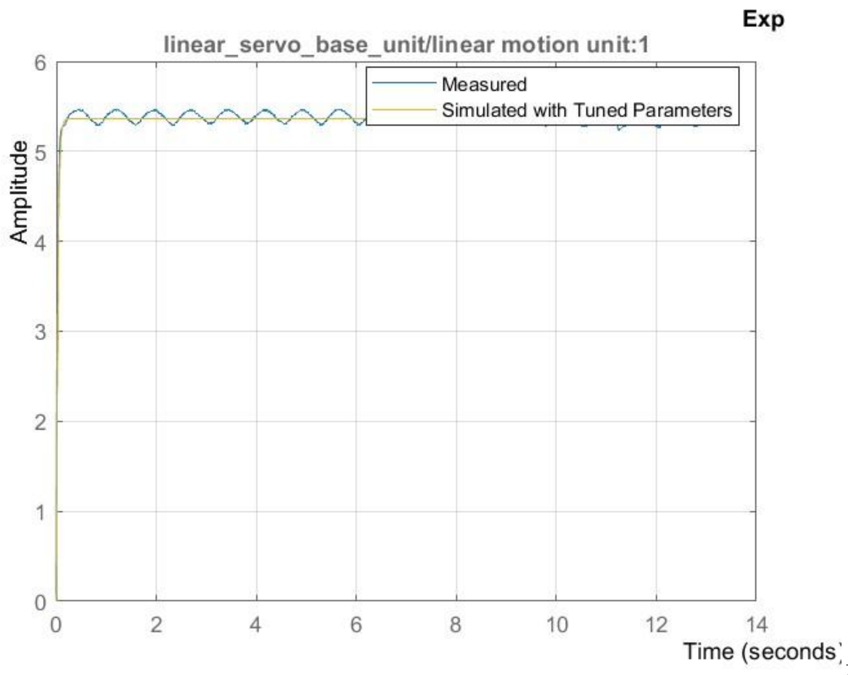


figure 8

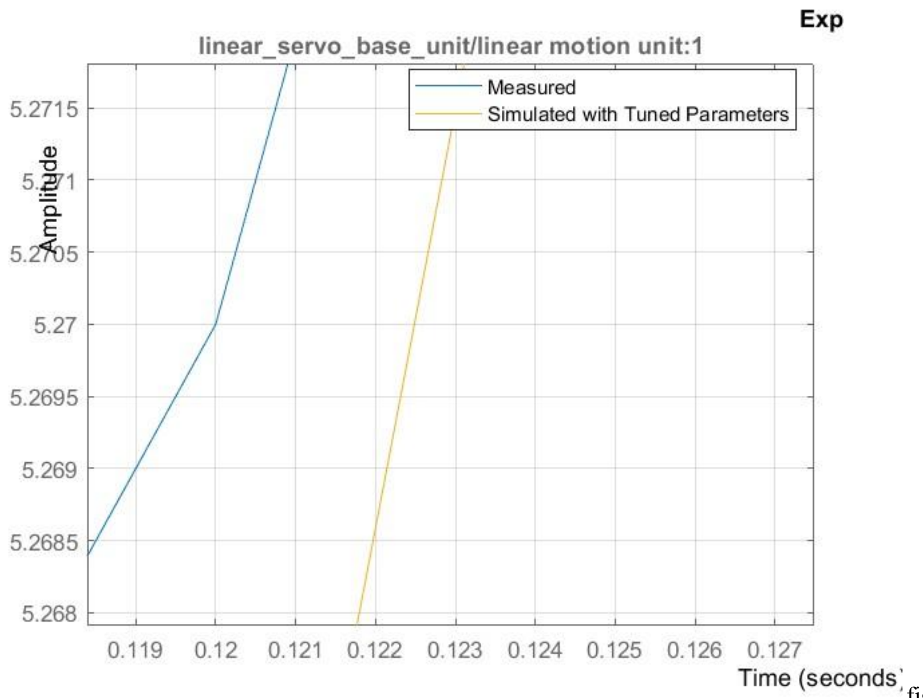
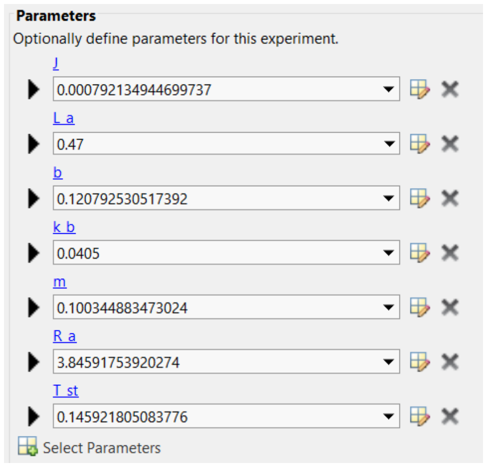


figure 9

Plots showing the measured data plot vs the simulated plot obtained with values from the parameter estimation. The time constant of the two responded in the same with 4*tau=0.12 s.

Final parameters values:



Parameters

Optionally define parameters for this experiment.

J	0.000792134944699737
L a	0.47
b	0.120792530517392
k b	0.0405
m	0.100344883473024
R a	3.84591753920274
T st	0.145921805083776

figure10

Model Controller Tuning

Controller Code

The code is written in Arduino and is used to control the position and velocity of a DC motor that is connected to a cart loaded on a belt. The system uses two PID controllers, one for position control and one for velocity control, to ensure precise and accurate movement of the cart. The system also utilizes an encoder and a limit switch to determine the position of the cart. The encoder is connected to the motor shaft and is used to measure the rotation of the motor in pulses. These pulses are then converted to centimeters of movement for the cart using a conversion ratio. The limit switch is used as a home position for the cart, allowing the system to zero the position at startup. The code also uses a TimerOne library to acquire data at a set interval, and it allows the user to input a new setpoint position for the cart to move to. This is done through the serial communication and it allows for real-time control of the system.

The cascaded control system in this code utilizes two PID controllers, the first one is the PosPID controller and the second one is VelPID controller. The PosPID controller takes as input the current position of the cart measured by the encoder and compares it to the desired setpoint position that is entered by the user. The output of the PosPID controller is the setpoint velocity that the cart should reach. This output is then used as the input for the VelPID controller. The VelPID controller takes as input the current velocity of the cart and compares it to the setpoint velocity outputted by the PosPID controller. The output of the VelPID controller is the power that should be applied to the motor in order to reach the desired setpoint velocity. This power is then applied to the motor through the use of PWM signals.

The code uses saturation limits for both the PosPID and VelPID controllers. The saturation limits are used to ensure that the output of the controllers stays within a specific range. For the PosPID controller, the saturation limit is set using the function PosPID.SetOutputLimits(-2, 2). This means that the output of the PosPID controller is limited to a range of -2 to 2. This output is the setpoint velocity that the cart should reach. For the VelPID controller, the saturation limit is set using the function VelPID.SetOutputLimits(-255, 255). This means that the output of the VelPID controller is limited to a range of -255 to 255.

An important issue to keep in mind is the motor driver deadzone, it is the range of inputs where the motor does not move. The line

if (pwr < 67 && pwr > -67){pwr = 0;}

accounts for the motor driver deadzone. This line of code checks the value of the output from the VelPID controller, if it falls within a certain range, it sets the output to zero. This range is set between 67 and -67. This means that if the output of the VelPID controller is between these two values, the motor will not move.

Control results

Open-Loop Response,Plotting the open loop response of the model in Simulink

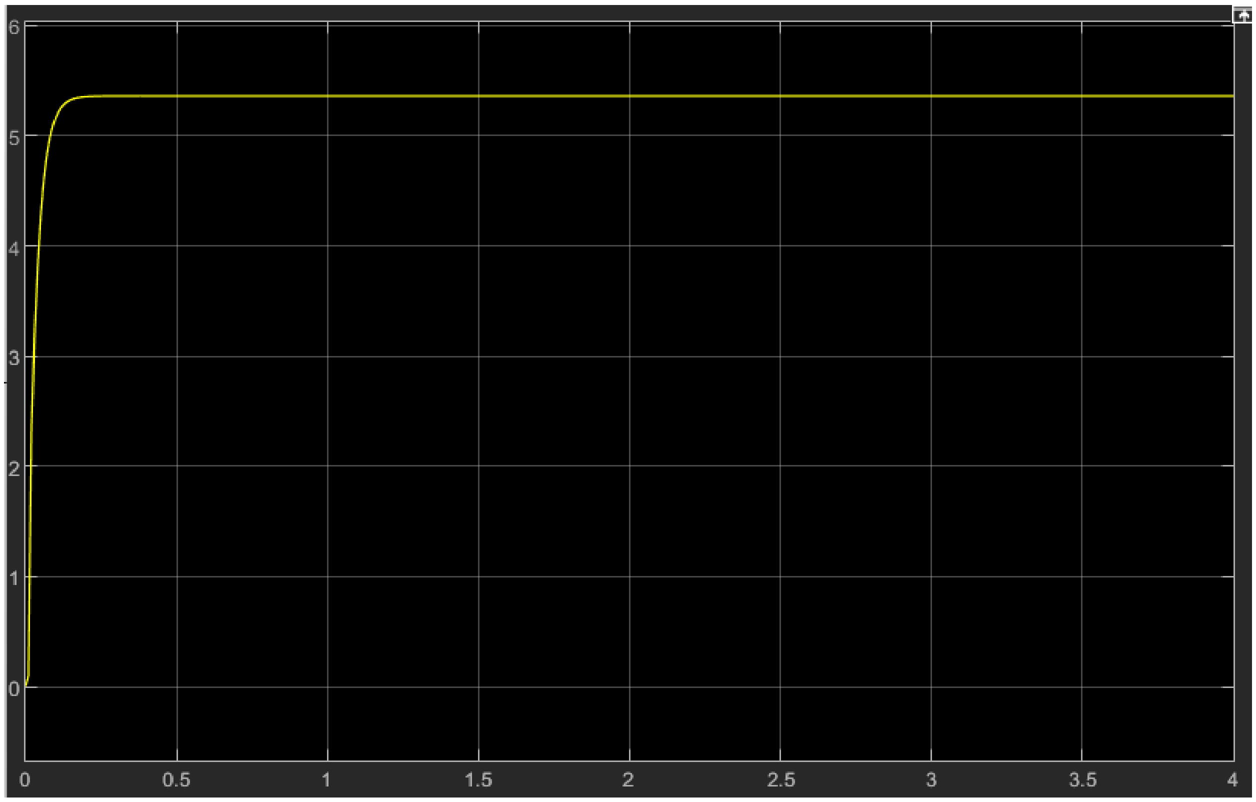


figure11

The settling time of the model is almost 0.12 as the velocity reaches 98% of the steady state value(5.4 cm/s). The time constant $\tau = \frac{settling\ time}{4} = \frac{0.12}{4} = 0.03$. the value of the time constant must be the same as the value calculated from the open-loop response of the physical system. The time constant gives indicator about the ratio between the controller gains $K_p and\ K_i \Rightarrow \frac{K_p}{K_i} = \tau$.

Velocity Loop

Controller for this loop is a PI controller with the following transfer function:

$$PI(s) = K_p + \frac{K_i}{S}$$

The sample time of the controller is set to 10 ms to be consistent with the arduino code.

The output saturation of the Controller must be set to be within the limits of [-255,255] and turn the Anti-windup feature to clamping for better error management.

Trial 1

Choose gains that match the ratio of the time constant

$K_p = 10,\ K_i = 300$, the velocity response is:

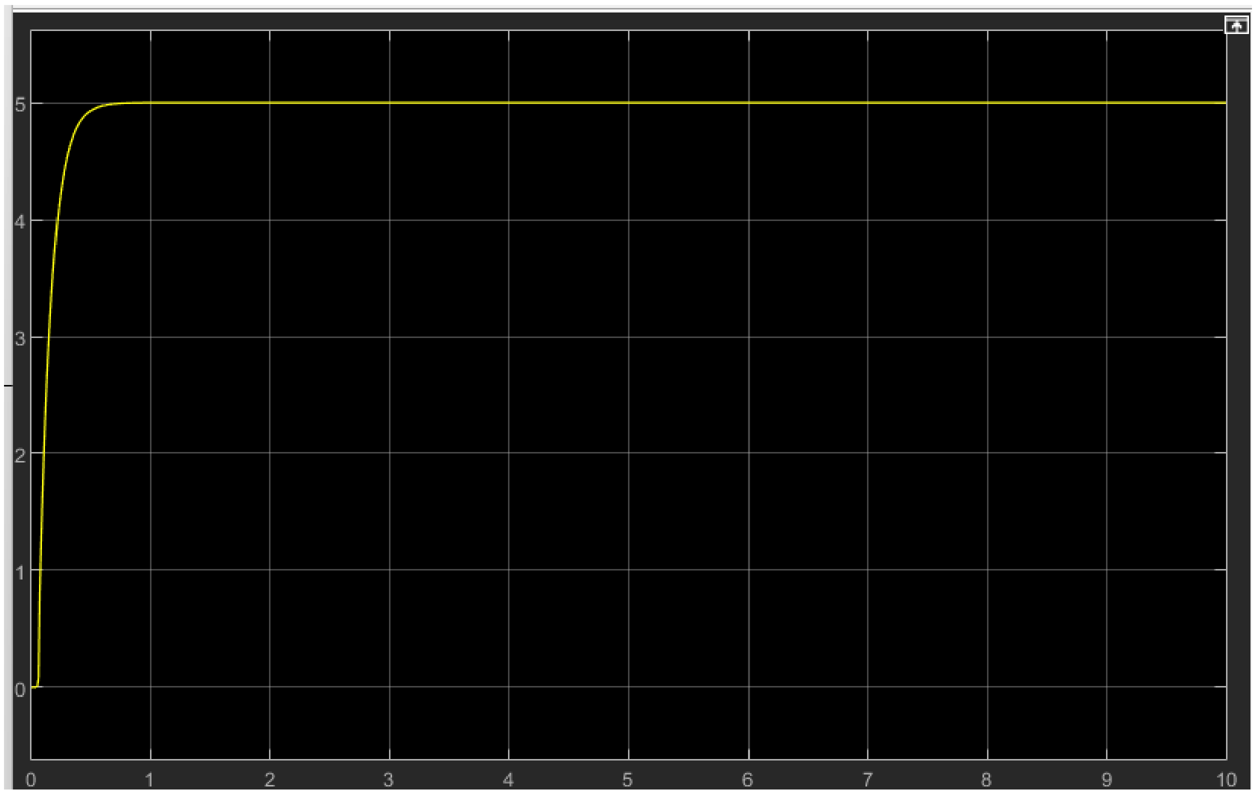


figure 12

The model takes time to start working as the velocity is zero for some time at the beginning and so the system needs to be accelerated with higher pattern but at the steady state, the system converges to the set point 5 cm/s.

Trial 2

$K_p = 30,\ K_i = 1000$, the velocity response is:

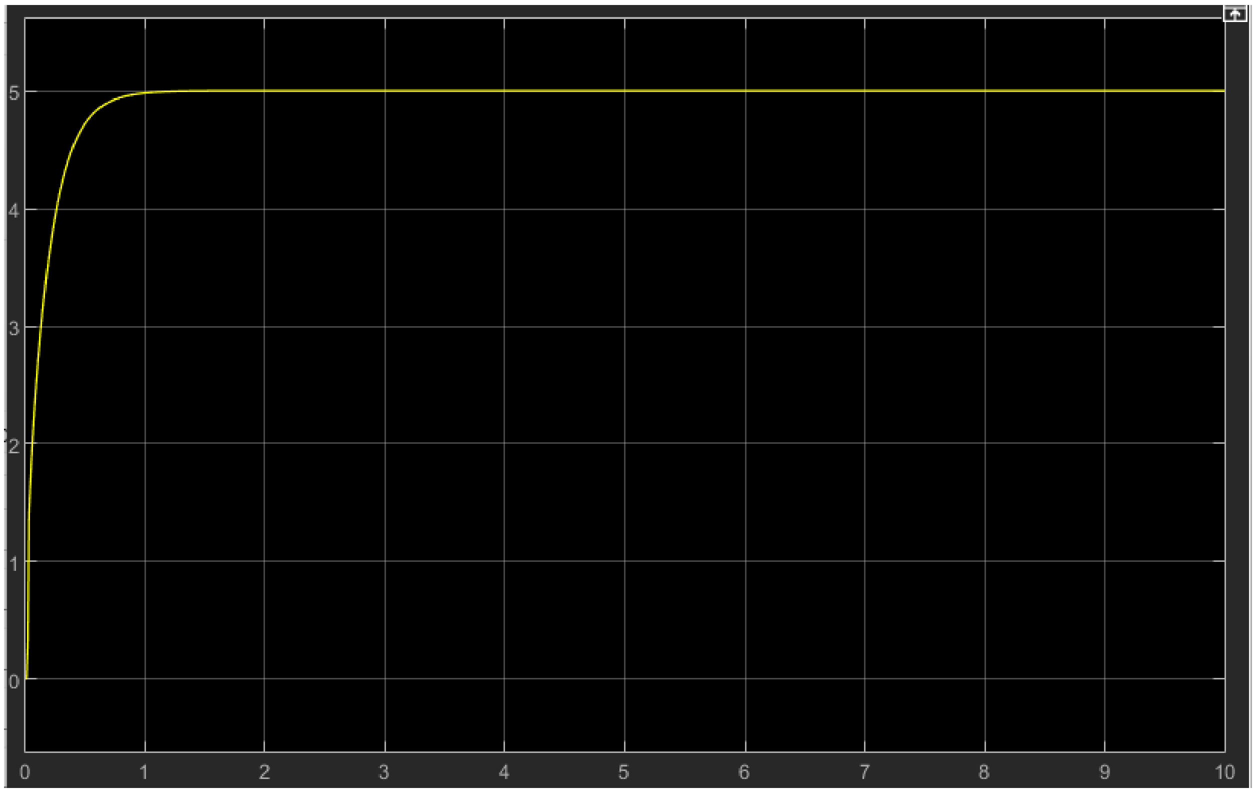


Figure 13

The model velocity response is better with these gains as the problem at the very beginning has been fixed and the system performance is perfect, but these gains do not fit the time constant value and so the integral gain will be changed accordingly in the third trial.

Trial 3

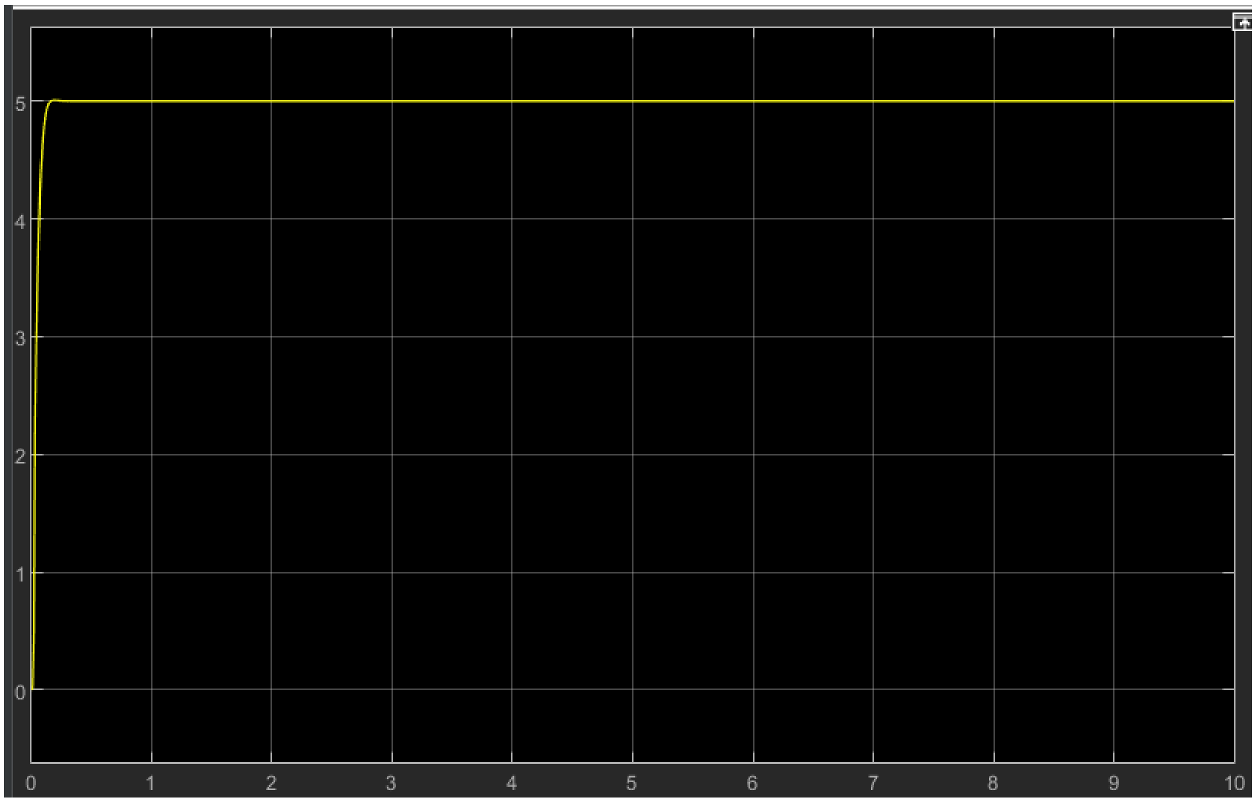


Figure 14

The model performance is perfect with minimum settling time and fast growing speed.

Position Loop

After tuning the most inner loop(the velocity loop), it's time to tune the outer loop, the Position loop with P

Controller for this loop is a P controller with the following transfer function:

$$PI(s) = K_p$$

The sample time of the controller is set to 10 ms to be consistent with the arduino code.

The output saturation of the Controller must be set to be within the limits of [-5,5] and turn the Anti-windup feature to clamping for better error management.

Trial 1

$K_p = 10$; the speed response of the system is as follows:

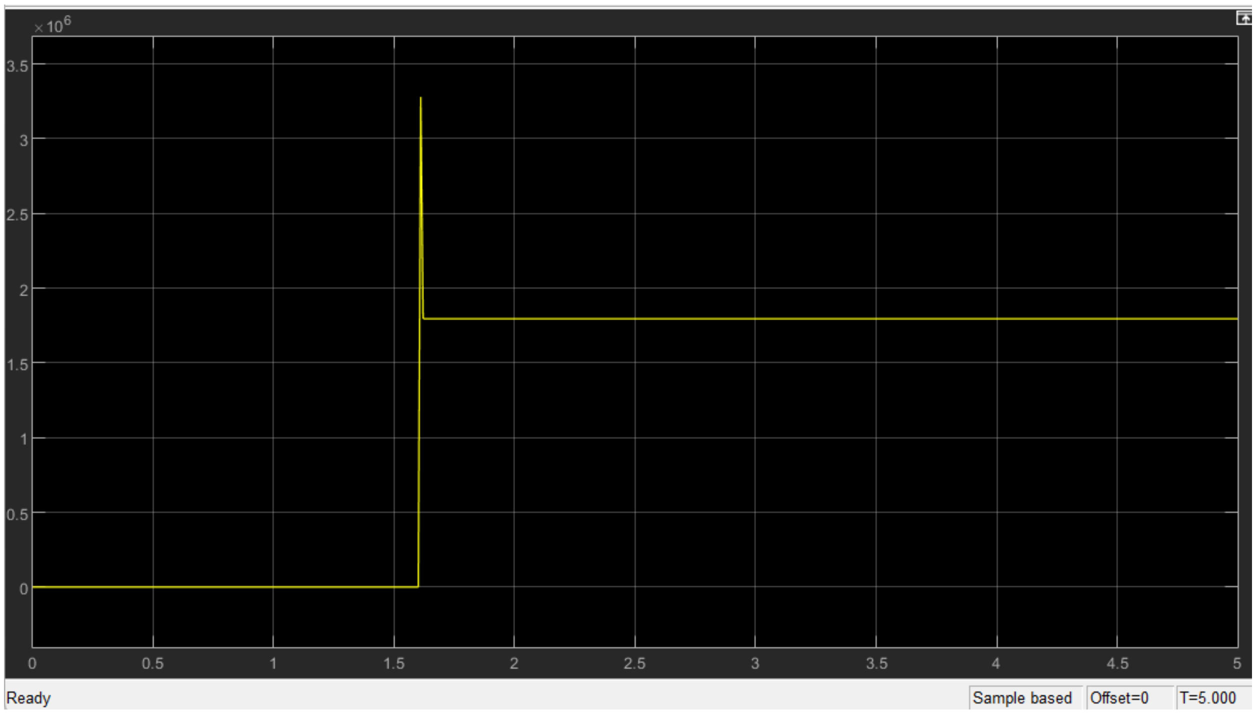


Figure 15

This error pops up after the system reaches its settling time (i.e. for position set point of 5 cm and a speed limit of 5. The system reached the set point after 1 sec, and the error showed up at approx. 1.5 sec. So we reduced the simulation time to 1.4 s in the second trial.

This error as discussed with the instructor is due to some block in the model itself but it is totally safe to ignore it by reducing the simulation time.

Trial 2

$K_p = 10$; the speed response of the system is as follows:

The model response is very good as it reaches the distance set point (was 5cm in this simulation) in about 1 second which is acceptable.

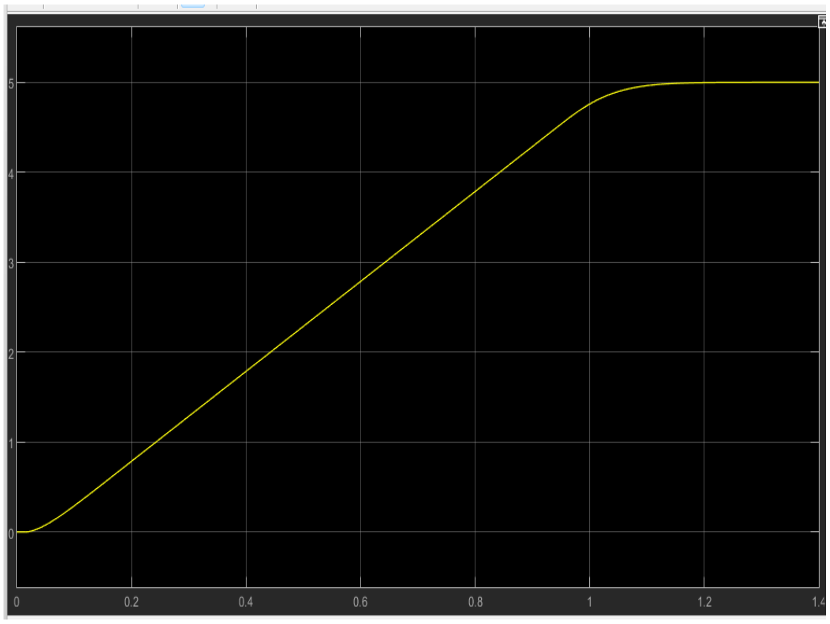


Figure 16

The velocity response after tuning the position controller:

The response is good with no overshoot or oscillations. The graph of the velocity response is very smooth as well as the position response.

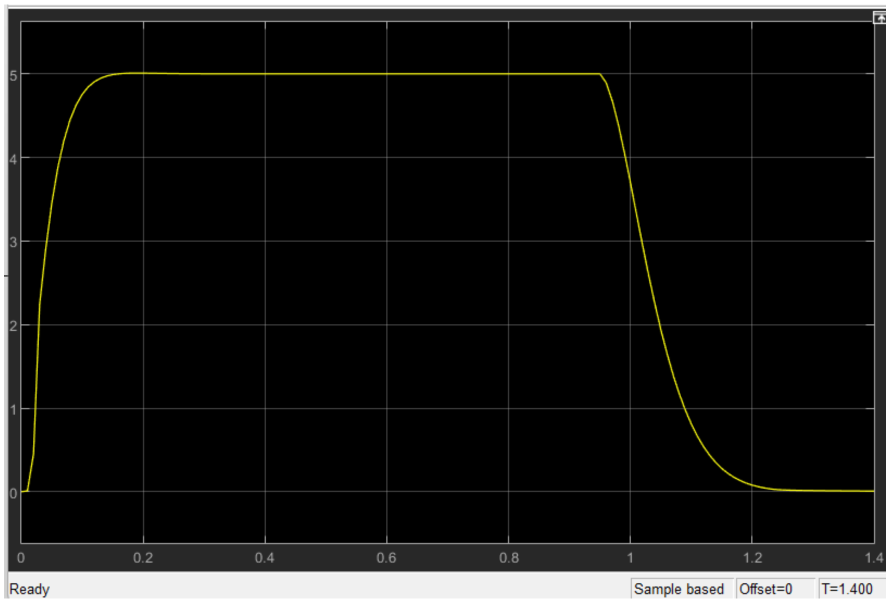


Figure 17

Controller results analysis

The physical setup was controlled using different PID libraries. The first library we used caused the power to oscillate after changing the library using the same gains the oscillations stopped. However, some parameters in the system caused a small overshoot that did not happen after in simulink model despite using the same gains we concluded that this might be to :

1. The low speed of the motor
2. Not lubricating the system
3. Loose belt

Result

Controlling movement of the linear cart using cascaded PID the controller was able to effectively reduce the overshoot and settling time of the system, achieving an overshoot of 5% and a settling time of 0.12 seconds. These results indicate that the cascaded PID controller is an effective control strategy for movement control of linear carts, as it allows for precise and accurate positioning of the cart. The results of the experimental data showed that the linear cart was able to reach and maintain the desired position with a high degree of accuracy.

Conclusion

Problems faced:

Many problem were faced in this project, either in the setup, the control task or the parameter estimation, some of the problems can be summarized as following:

1. Incorrect and inconsistent reading from the motor: in the first stages of the project we needed to get the PPR rate for the motor, however, the reading were never consistent and it took us a long time to make sure that the code is correct and the wiring is correct to finally replace the encoder with another working one.

2. Timing pulley radius: the pulley inner diameter sizes available were 4 mm and 6mm, however, the shaft diameter was 5 mm so we needed to perform machining on the 4 mm diameter pulley.
3. Parameter estimation: in this step, the model response did not have the same time constant of the experimental plot, after a little search, we included the motor inductance and inertia as estimated parameters which helped change the time constant very effectively.
4. Importing accurate experimental data: in the first stages of the project, we needed to import experimental data for the parameter estimation process, but the time steps were not consistent. So we used a library time TimerOne that allows us to acquire the data at a fixed time interval which we desire.

Table of figures