Feedback Control Systems CEP

Evaluation of System parameters, Control Analysis and Revision in Dynamics of a DC motor

Report prepared by:

Ali Aman EE-19136
Hammad Ali Khan EE-19139
Syeda Rujab Hussani EE-19154
Ifra Sohail EE-19161
Moeed Alam EE-19170

Dated:

1st August, 2022

Course title:

Feedback Control System (EE-374)

Submitted to:

Dr. Muhammad Javed

Introduction

Motor is an electrical machine that **converts electrical energy into mechanical energy**.
The working of DC motor is based on the principle that when a current carrying conductor is placed in a magnetic field, it experiences a mechanical force.
The direction of the mechanical force is given by **Fleming's Left-hand Rule** and its magnitude is given by:

$$F = BIL\ (Newtons)$$

DC Motor Principle
A machine that converts DC electrical power into mechanical power is known as a Direct Current motor.

DC motor working is based on the principle that when a current carrying conductor is placed in a magnetic field, the conductor experiences a mechanical force.
The direction of this force is given by **Fleming's left-hand rule** and magnitude is given by:

$$F = BIL\ (Newtons)$$

According to Fleming's left-hand rule when an electric current passes through a coil in a magnetic field, the magnetic force produces a torque that turns the DC motor.
The direction of this force is perpendicular to both the wire and the magnetic field.

The objective of this CEP was to determine the motor transfer function and to evaluate the physical parameters of our DC motor. Transfer Function relates the output or response of the system to its input applied. Here we are evaluating transfer function for motor when at output we take the speed of rotation of motor and at input we have applied DC voltage. The physical parameters are first assigned initial values and then are estimated to get their final and actual values based on the data provided by motor's rotation. These parameters then help us to calculate the required transfer function.

Abstract

We interfaced our DC motor first with Arduino UNO, to which the speed sensor was also interfaced. Executing the Arduino code, we were able to collect the data for the rotation of motor or its rpm. This data was then exported to a Microsoft Excel file.

Excel sheet was imported using MATLAB. The data was just filtered for accurate step response of motor and to get a smooth response curve of the motor. The filtered data was then used on a Dummy Model on Simulink. Dummy because we are itinually giving the program our initial values and based on repetitive iterations it estimates the actual values of the parameters based on the data we have recorded using Arduino.

On Simulink we perform parameters estimation. Once the parameters are estimated to our recorded data from motor, we just simply updated the same parameters on MATLAB workspace defined by the same names and plotted the transfer function using MATLAB command.

Procedure

RPM Calculation using Arduino Uno

Firstly, we have connected the motor to dc supply 5V. The motor's shaft is connected to an encoder which has 20 holes. IR Sensor has two outputs i.e., digital and analog. We have used digital output pin. DC motor was interfaced along with motor driver to Arduino.

To measure the speed of rotation of DC motor speed sensor was also interfaced with Arduino. Physical readings to measure speed was done by placing the motor itself into transmitter and receiver of the sensor. Sensor works on the principle that the no of times it sees interruption it counts that.

Our motor has a disc with holes (encoder disc) to block the infrared beam, thus by counting the number of times the sensors go from Low to High we can calculate the number of revolutions for a given time period. We will count the number of times the speed sensor goes from Low too High in a second and then divide that number by 20 (number of holes in the encoder disc) to get the number of revolutions per second attached to its shaft.

So, whenever holes pass through the sensor's transmitter receiver the sensor records a value equivalent to 0 meaning it doesn't see any interruption but when holes are in transition the sensor counts a 1 as it sees an interruption.
So, this series of interruptions per second give us the RPS that is Revolutions Per Second. To get rpm that is revolutions per minute we multiply RPS by 60.
The speed sensor uses only 1 pin that goes from Low to High to detect holes in the encoder disc.

Since we will be using Interrupt zero to read the speed sensor, we need to connect it to Pin 2 (interrupt 0 pin) on the UNO. We're are also using the L9110 motor driver to control the speed and direction of the geared DC motor.

Through Arduino code we have also recorded the time corresponding to the certain value of RPS and rpm. This value of time will be used to plot this data for step response of motor that is speed vs time plot.

Procedure

Importing Data on MATLAB and Filtering

Once this data was recorded it was exported to excel sheet. We imported this excel sheet on MATLAB where we simply have applied different techniques to filter and smooth this data

```
var = xlsread ( 'C:\Users\Hammad\Desktop\data123.csv' ) ;
RPS_temp = nonzeros ( var ( : , 1 ) ) ;
RPS = zeros ( length ( RPS_temp ) + 1 , 1 ) ;
RPS ( 2 : end , 1 ) = RPS_temp ;
RPM_temp = nonzeros ( var ( : , 2 ) ) ;
RPM = zeros ( length ( RPS_temp ) + 1 , 1 ) ;
RPM ( 2 : end , 1 ) = RPM_temp ;
TIME_temp = nonzeros ( var ( : , 3 ) ) ;
TIME = zeros ( length ( TIME_temp ) + 1 , 1 ) ;
TIME( 2 : end , 1 ) = TIME_temp ;
data_temp = table ( RPS , RPM , TIME ) ;
[ temp , index ] = unique ( data_temp.TIME , 'stable' ) ;
data = data_temp ( index , : ) ;
temp8 = smooth ( data.RPS ) ;
temp9 = smooth ( data.RPM ) ;
for i=1:3
    temp8 = smooth(temp8);
    temp9 = smooth(temp9);
end
temp1 = temp8 ( 1 : 35 , 1 ) ;
temp_rpm = temp9 ( 1 : 35 , 1 ) ;
temp_time = data.TIME ( 1 : 35 , 1 ) ;
plot ( temp_time , temp1 )
grid on , xlabel ( "Time(sec)" ) , ylabel ( 'RPS' )
title ( "RESPONSE OF MOTOR (AFTER DATA CLEANING)" )
unit = data.TIME >= 0 ;
unit = double ( unit ) ;
var1 = ones ( length ( RPS ) , 1 ) ;
var1 ( 1 : length ( unit ) , 1 ) = unit ( : ) ;
unit = unit ( 1 : 35 , 1 ) * 6 ;
temp3 = [ temp_time unit ] ;
figure ,
plot ( temp_time , temp_rpm )
grid on , xlabel ( "Time(sec)" ) , ylabel ( 'RPM' )
title ( "RESPONSE OF MOTOR (AFTER DATA CLEANING)" )
```

Procedure

The measurement of data required one person of the group to hold motor steady on to the sensor which obviously is erroneous hence causing the values of the data to be varying making the graph look something like this:

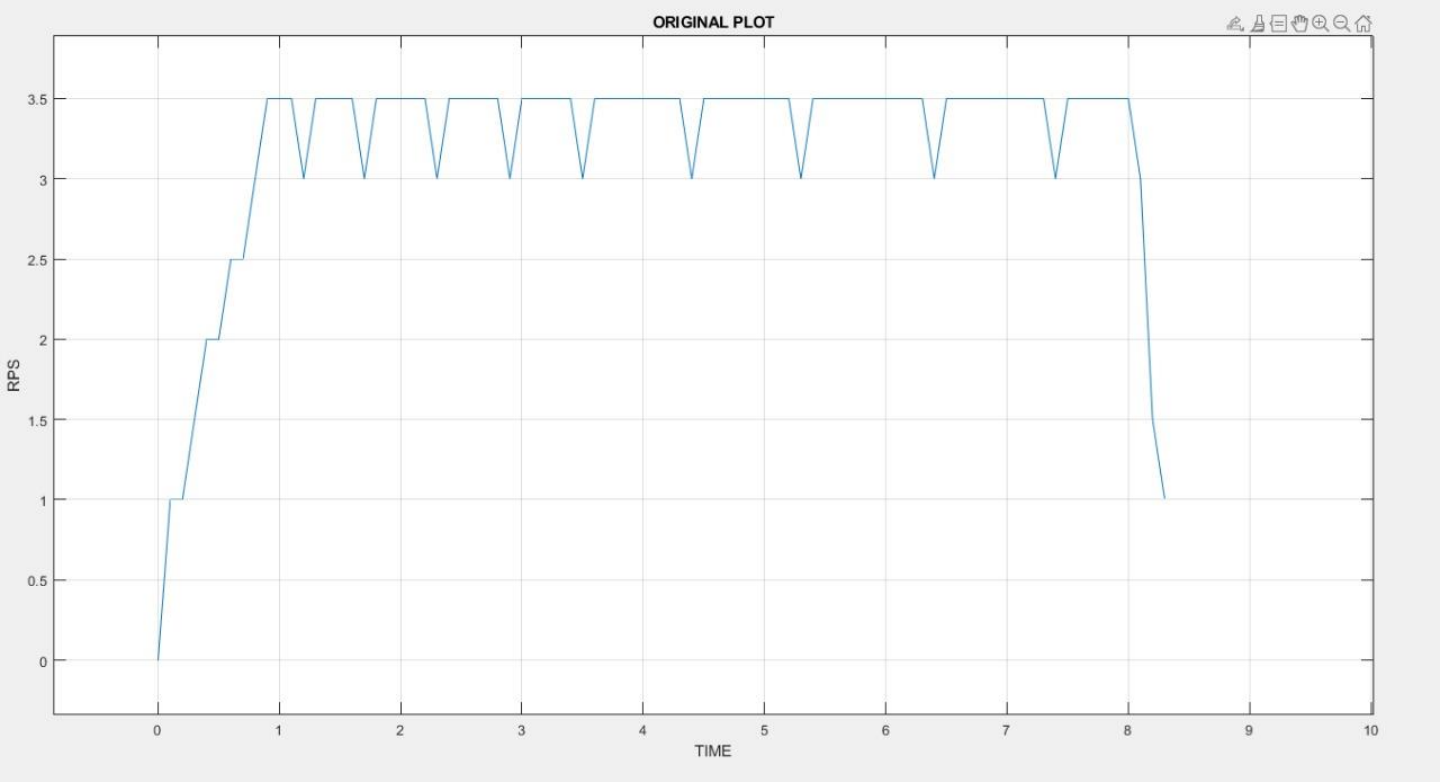


Figure 1 Raw data obtained

This clearly isn't the curve we desire. So, in order to get the required curve, we have written code lines for smoothing process of the data.

Procedure

After processing data through filtering and smoothing process we obtained plot below which is far more accurate and what we desire:

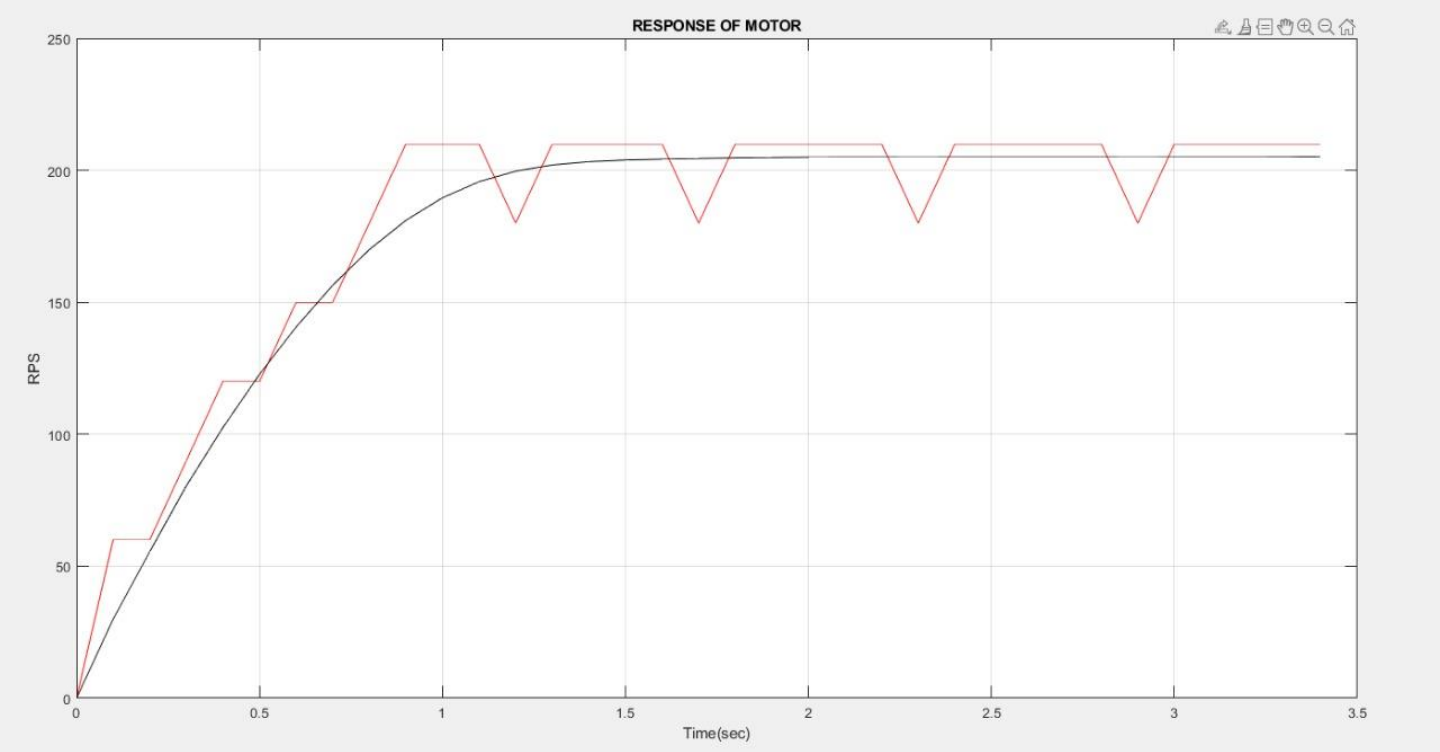


Figure 2 Plot showing comparison between raw data and filtered data

Once the data was smoothed, we now move towards the next phase and perhaps the most important one of this CEP that is determining and estimating the physical parameters of the DC motor.

Procedure

Estimating Parameters on Simulink

On Simulink environment we have designed a "dummy" model. Dummy because for this initial model we have used initial guessed values for each parameter.

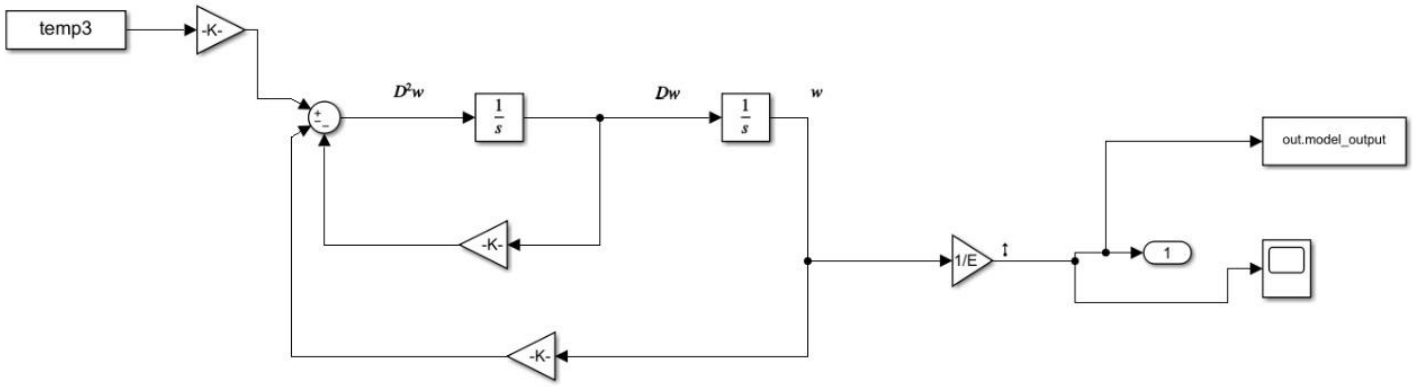


Figure 3 Simulink Dummy Model

Mainly this model was developed using the Differential Equation of DC motor obtained through mathematical modelling which is discussed already. It's understood from the equation that to get output the as rotation of motor it requiring the input and the feedback from B and K which forms a simple block diagram and we have used the same logic behind our Simulink dummy model. At output we have simply attached a scope to observe the response of the motor which is coming out as we desire after estimation.

Procedure

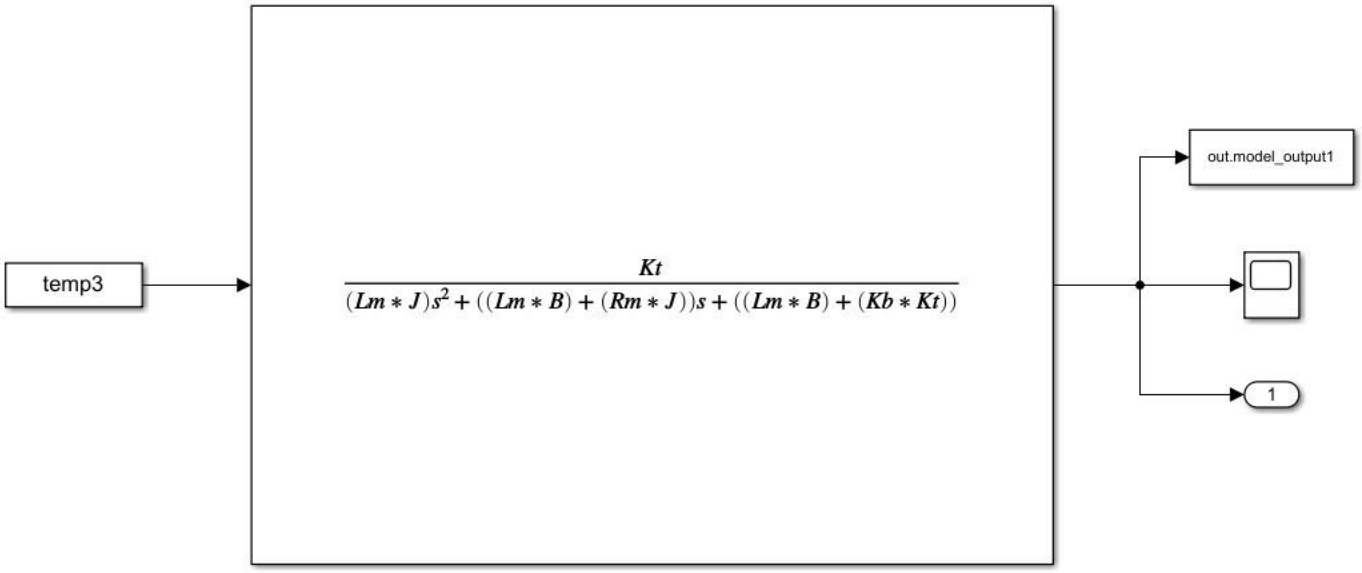


Figure 4 Same Simulink Model but with everything enclosed in a sub-circuit

Parameters were assigned arbitrary values, all of them, initially and they provided estimated values with reference to the plot obtained from the cleaned, filtered smoothed data from physical measurement of RPS.

```
clc,clear,close all
B = 0.1 ; E = 6 ; J = 0.1 ; Kb = 0.1 ; Kt = 0.1 ; Rm = 0.1 ; Lm = 0.1 ;
```

Using Parameter Estimation on Simulink we through repetitive iterations were able to successfully estimate the values of parameters based on the data we recorded from Arduino:

Procedure

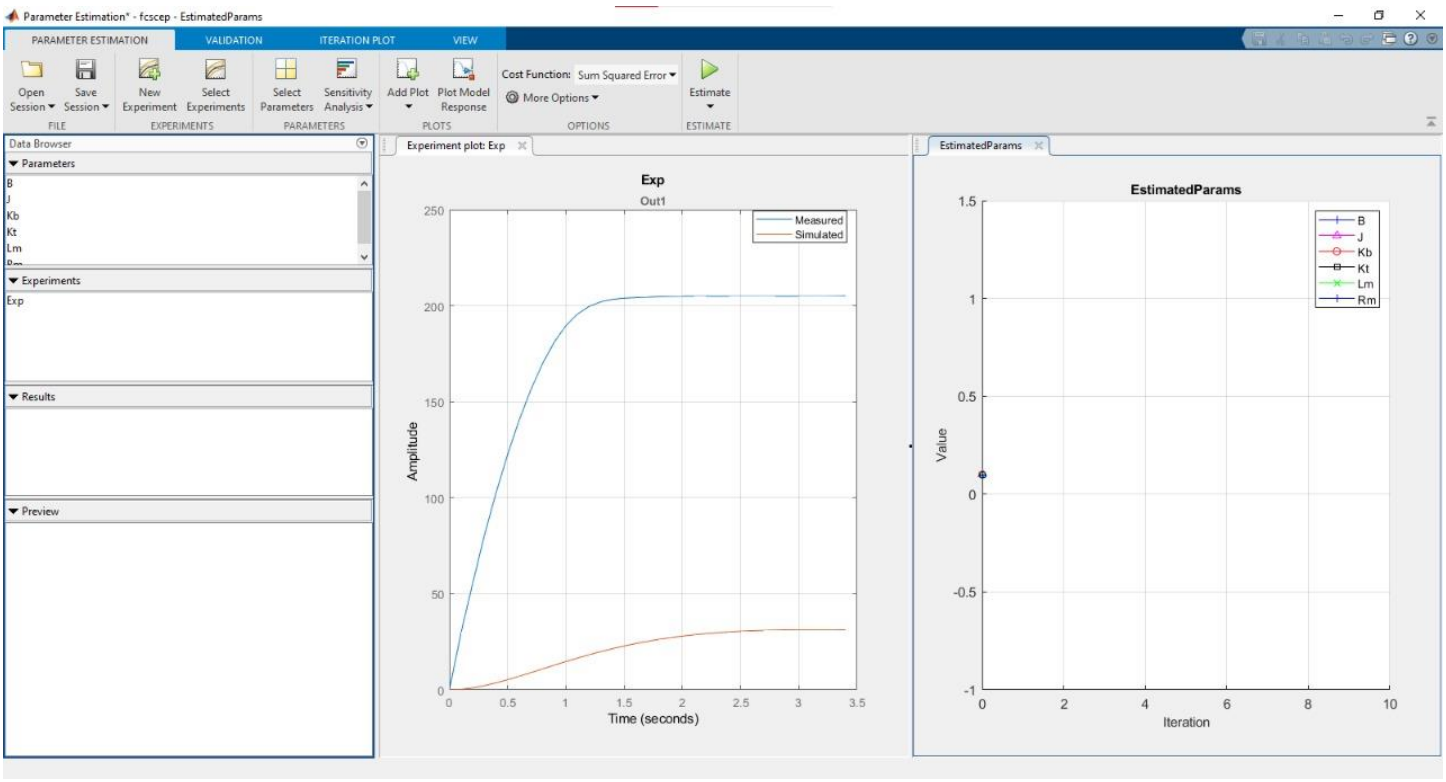


Figure 5 Pre-Parameter Estimation

Procedure

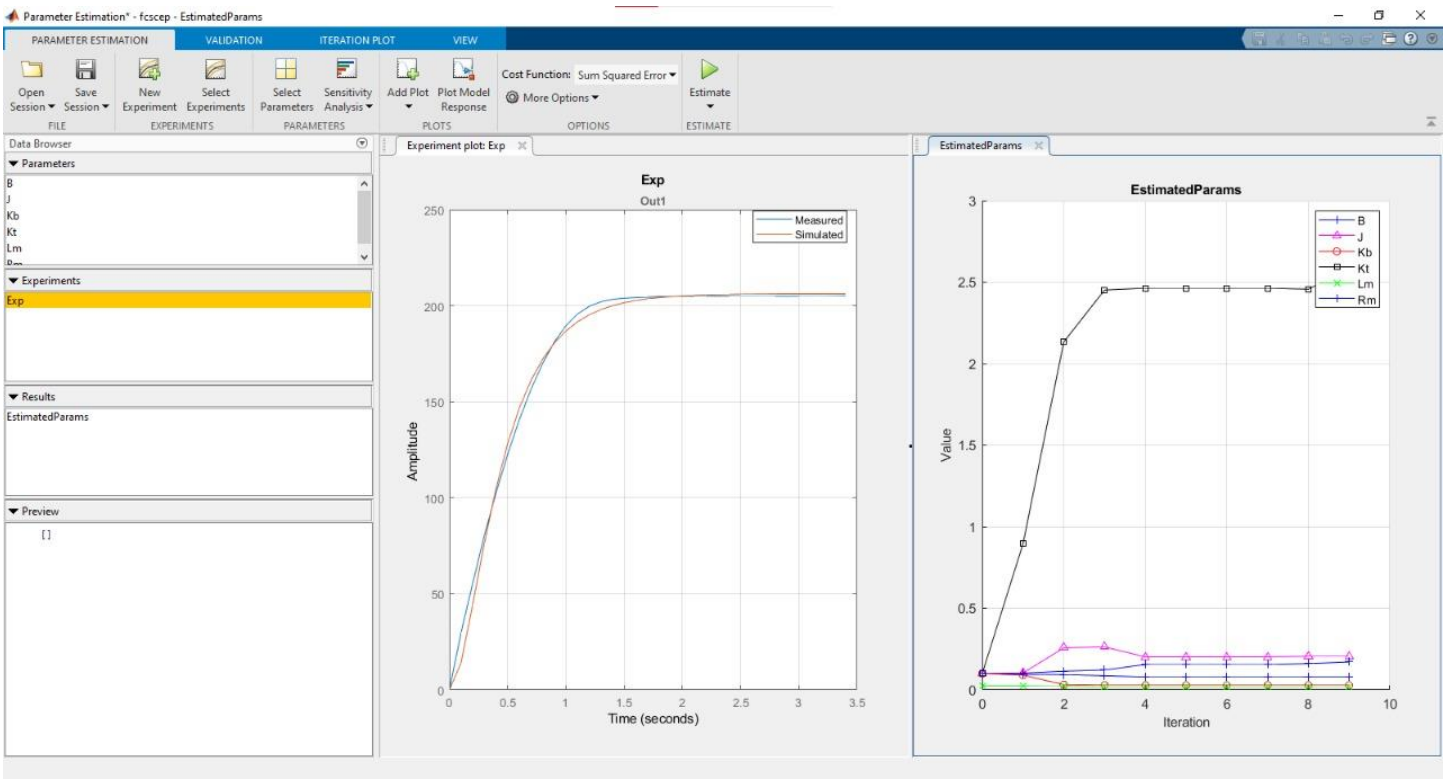


Figure 6 Post-Parameter Estimation

Estimation iterated for 10 iterations and it converged at 10th iteration. This whole optimization process took 4 min 11 sec for Simulink to solve for. After the convergence we got an error of just 1.37%. the parameter estimation just simply estimated our assigned random values to the parameters to originally recorded data. After estimation it matched the original plot from the original data which we obtained after filtering and smoothing process.

After this process the updated and estimated values of the parameters came out to be

$$B = 0.0797\ Kgm^2,\ \ J = 0.2061\ Nm - s$$
$$Kb = 0.0286,\qquad Kt = 2.5891\ Nm/A$$
$$Lm = 0.0175\ H,\qquad Rm = 0.1713\ \Omega$$

Procedure

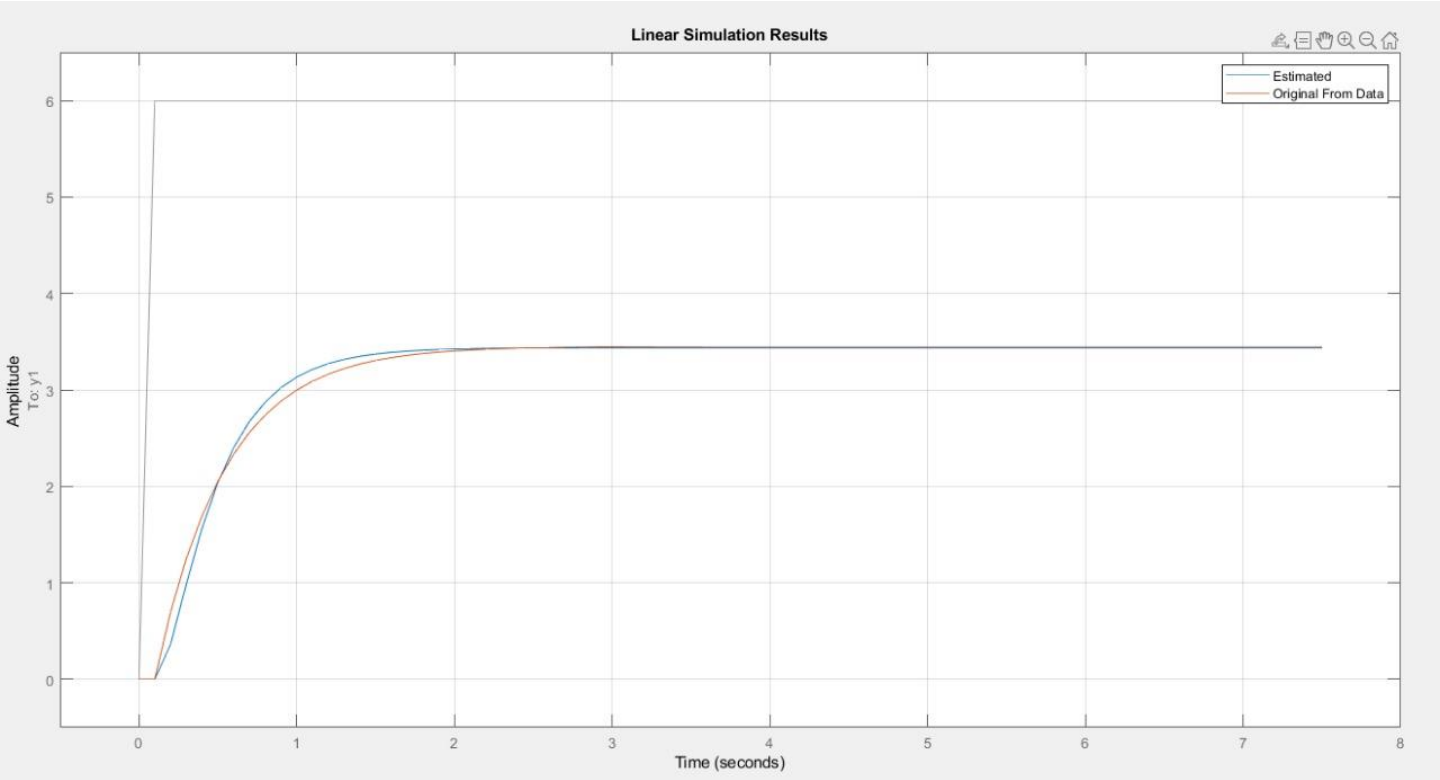


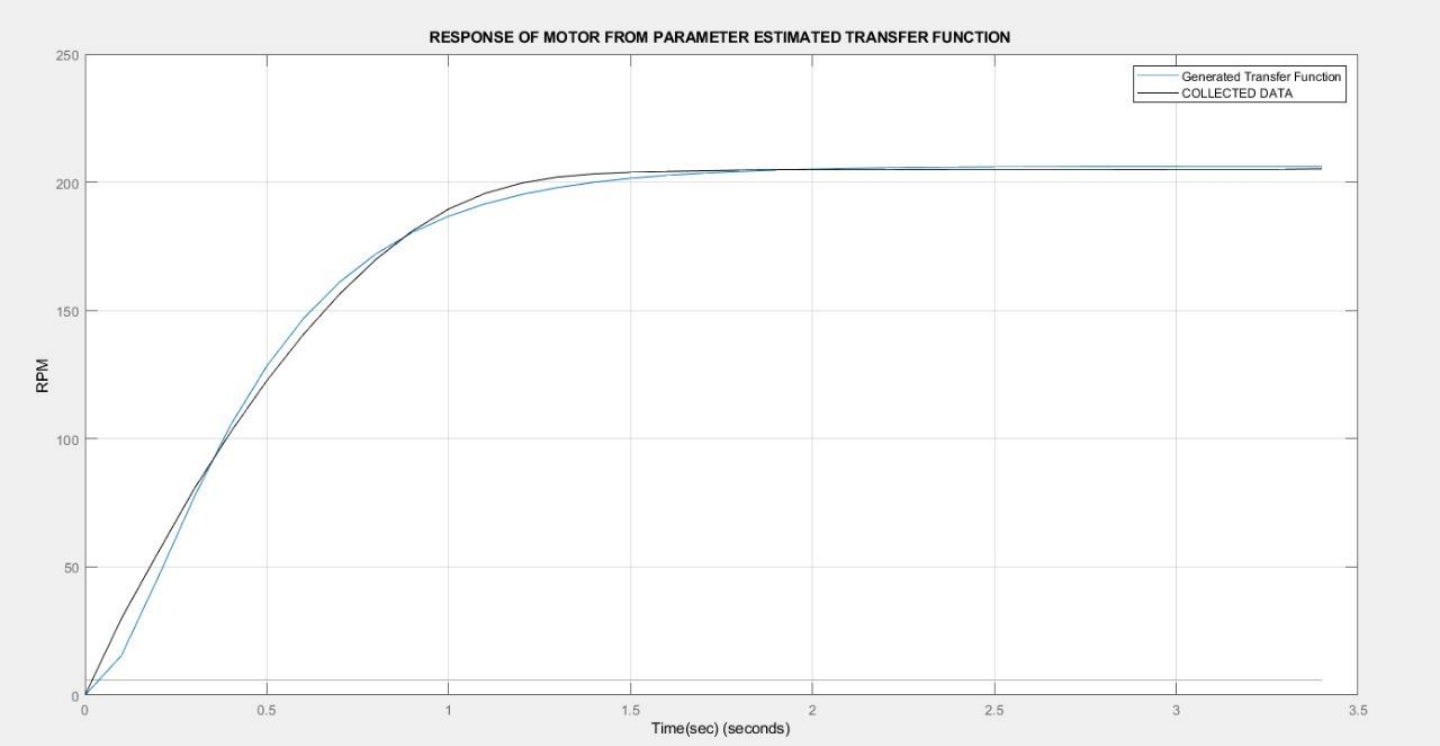
Figure 7 Plot showing parameter estimated plot and raw data plot

Procedure

Determining the Transfer Function

Lastly the values when estimated were updated on MATLAB workspace. Using the updated variables and the equation of transfer function we found using mathematical modelling of DC motor we were able to determine the transfer function of our DC motor.

```
num = (Kt);
den = [(Lm*J) ((Lm*B)+(Rm*J)) ((Lm*B)+(Kb*Kt))];
sys = tf(num,den);
lsim(sys2,unit,temp_time)
hold on
plot (temp_time,temp_rpm,'color','k'),legend("Generated Transfer
Function",'COLLECTED DATA')
grid on , xlabel ( "Time(sec)" ) , ylabel ( 'RPM' )
title ( "RESPONSE OF MOTOR FROM PARAMETER ESTIMATED TRANSFER FUNCTION" )
rlocus(sys);
```



Procedure

```
sys =

                2.589
  ---------------------------------
  0.003609 s^2 + 0.0367 s + 0.07533

Continuous-time transfer function.
```

Figure 8 Transfer Function obtained from estimated system

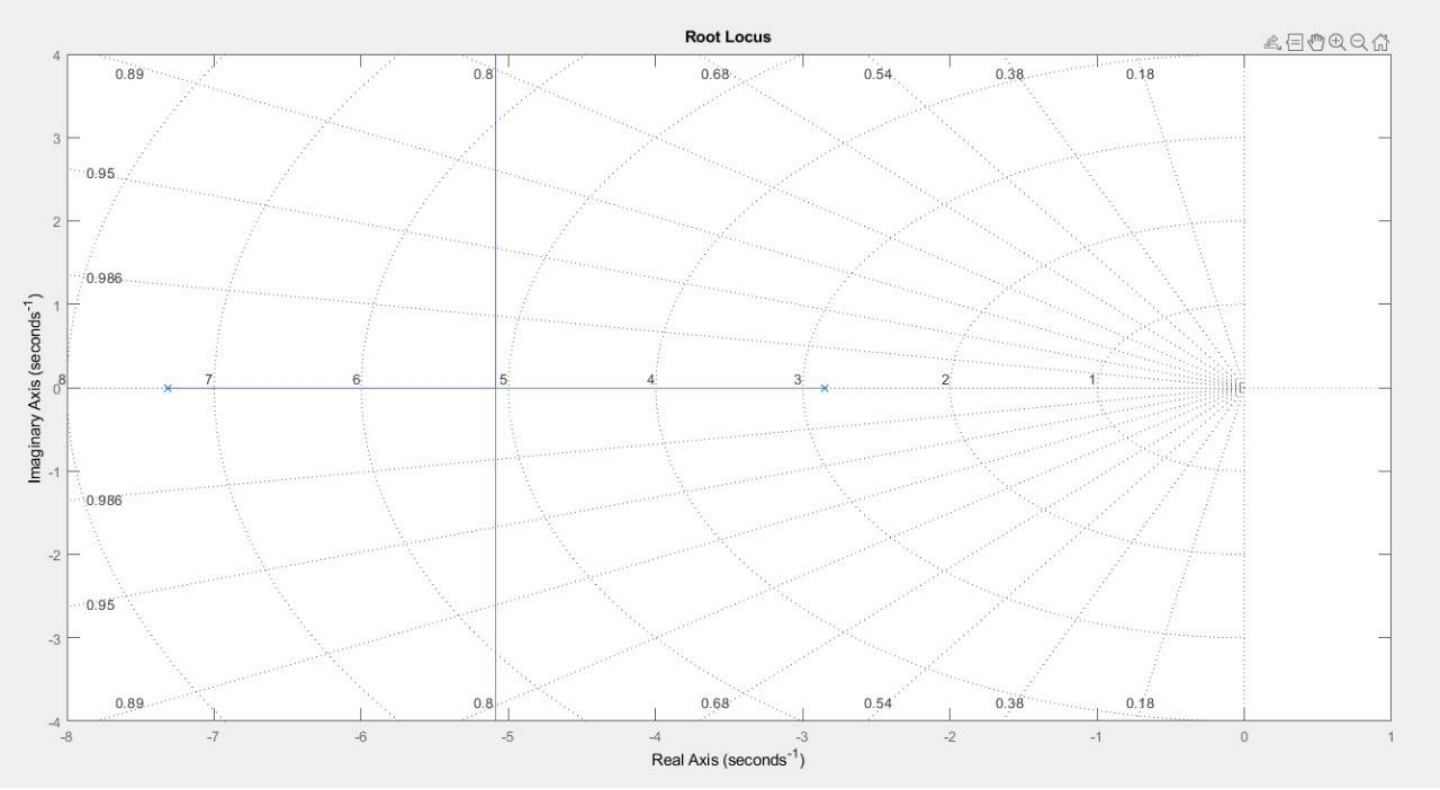


Figure 9 Root Locus plot for the system

Plotting & obtaining different stable responses by adding a control parameter

To obtain different stable resposnes we just added a PID block to our existing Simulink model. Adjusting the values of P, I and D we are able to obtain 3 different stable responses i.e., Overdamped, Critically Damped and Undamped responses for the same system.

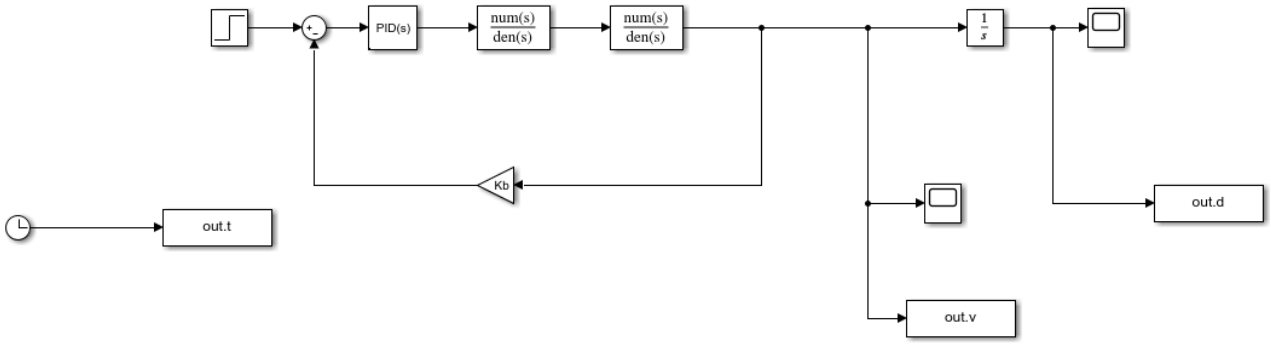


Figure 10 Simulink model for Various Stable Responses

Stable Responses

1) Overdamped Response:

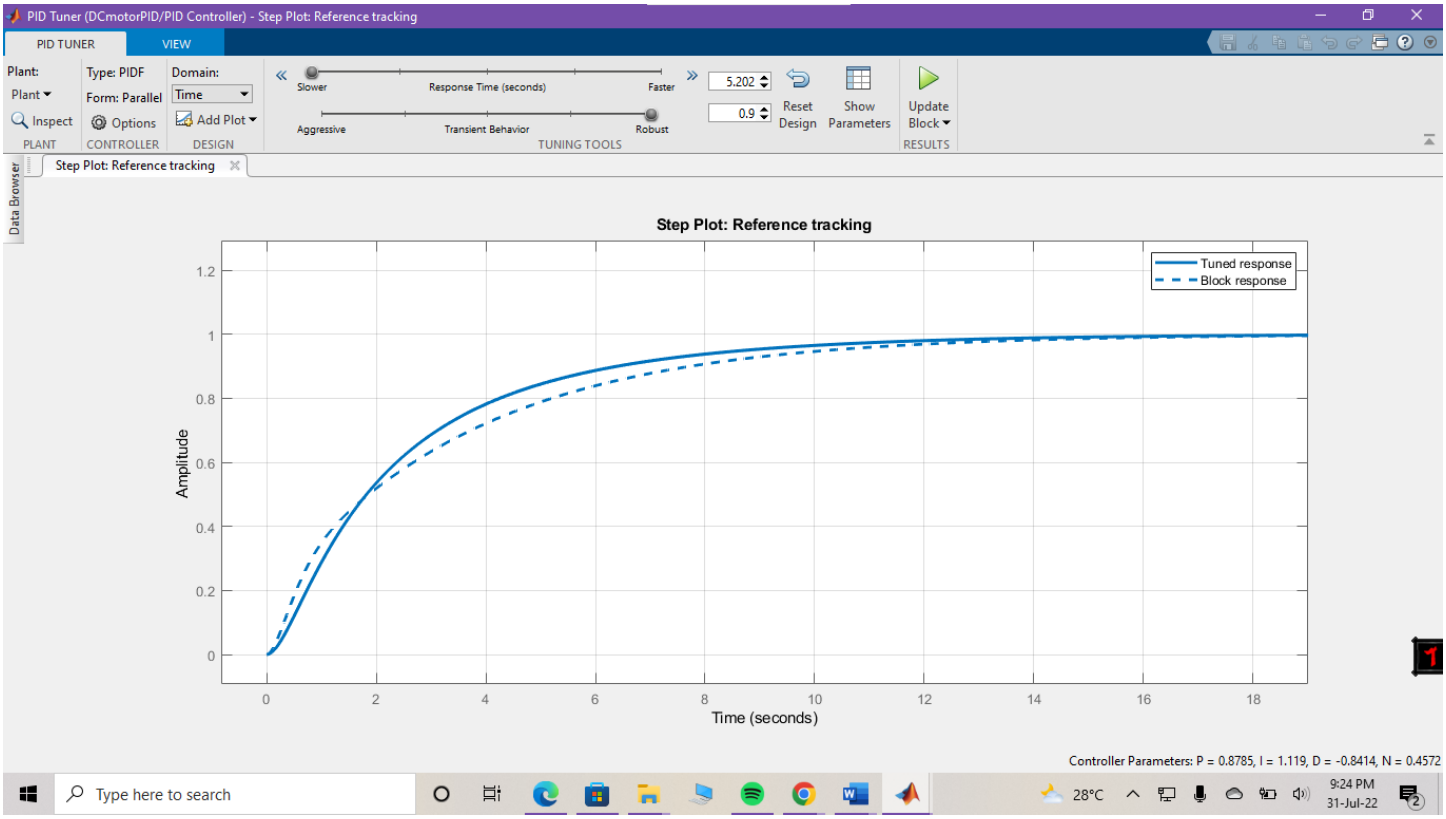


Figure 11 Overdamped Response

Stable Responses

Controller Parameters

	Tuned	Block
P	0.8785	1
I	1.1186	1
D	-0.84142	0
N	0.45725	100

Performance and Robustness

	Tuned	Block
Rise time	5.96 seconds	7.42 seconds
Settling time	12 seconds	13.6 seconds
Overshoot	0 %	0 %
Peak	0.999	0.999
Gain margin	Inf dB @ Inf rad/s	Inf dB @ Inf rad/s
Phase margin	90 deg @ 0.384 rad/s	97.1 deg @ 0.332 rad/s
Closed-loop stability	Stable	Stable

Figure 12 Different values for obtaining Overdamped Response

2) Critically Damped Response:

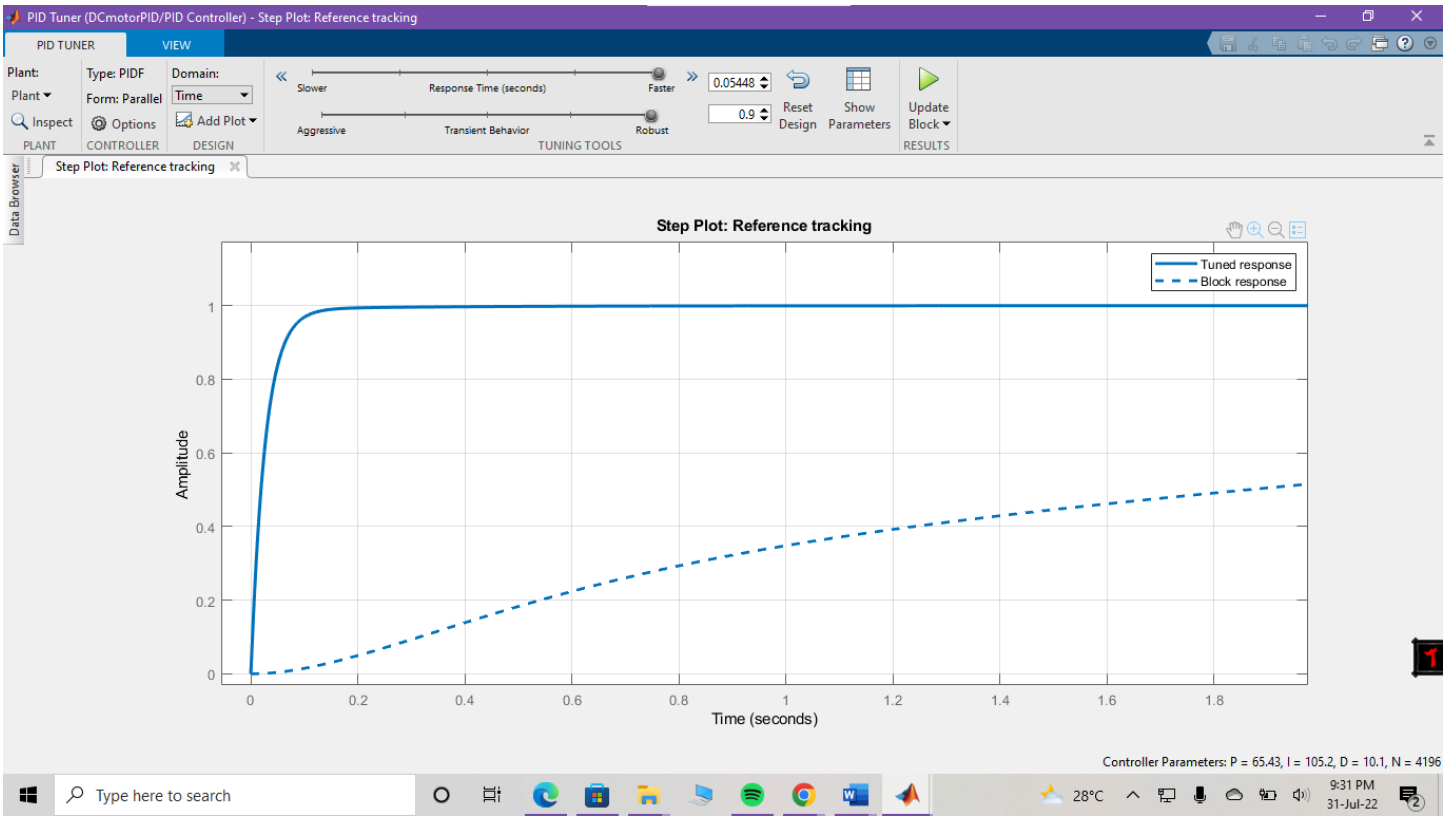


Figure 13 Critically Damped Response

Controller Parameters

	Tuned	Block
P	65.4349	1
I	105.1526	1
D	10.1034	0
N	4195.7003	100

Performance and Robustness

	Tuned	Block
Rise time	0.0608 seconds	7.42 seconds
Settling time	0.116 seconds	13.6 seconds
Overshoot	0 %	0 %
Peak	0.998	0.999
Gain margin	Inf dB @ Inf rad/s	Inf dB @ Inf rad/s
Phase margin	90 deg @ 36.7 rad/s	97.1 deg @ 0.332 rad/s
Closed-loop stability	Stable	Stable

Figure 14 Different values for obtaining Critical damped Response

3) Undamped Response:

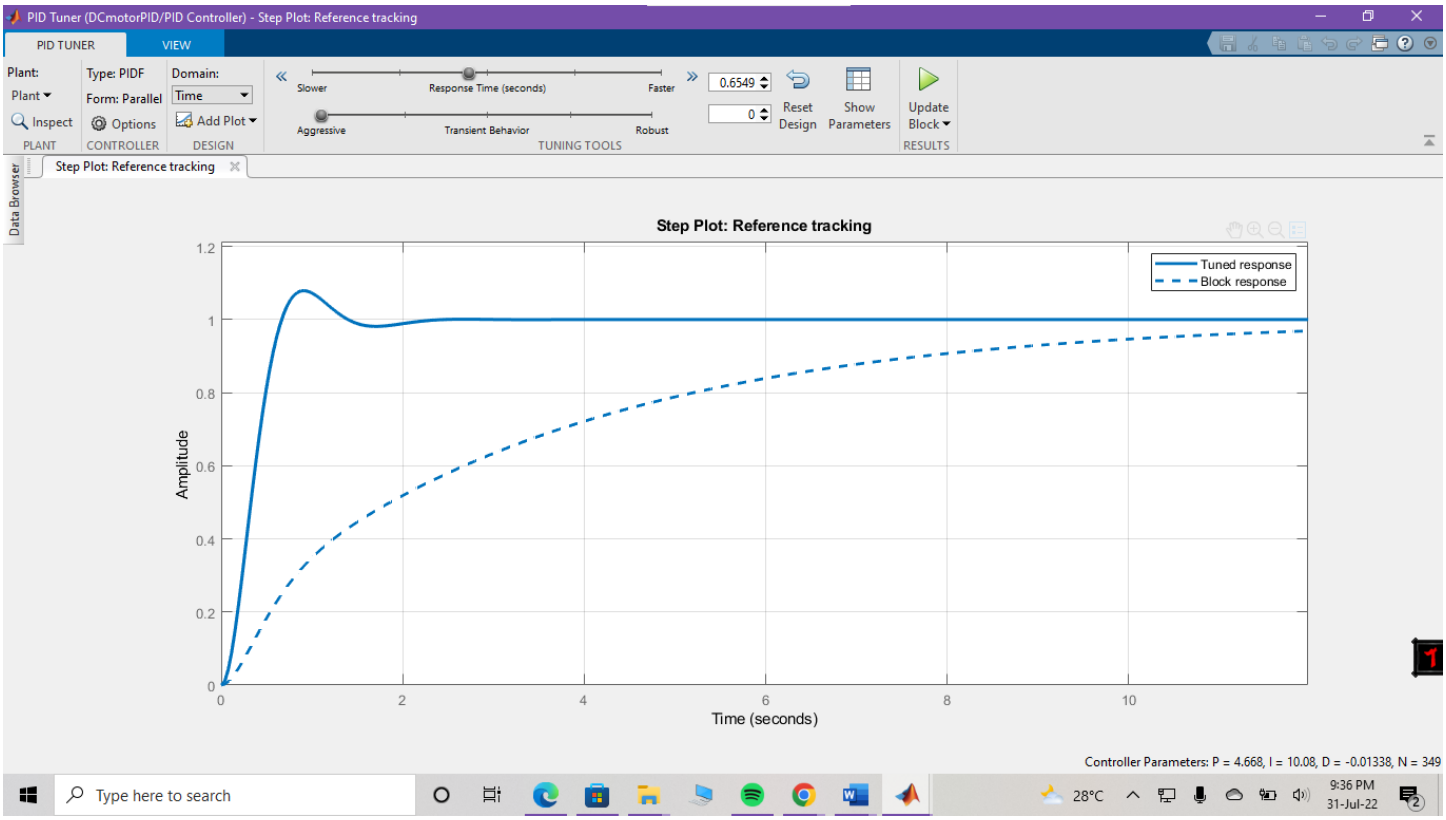


Figure 15 Underdamped Response

Controller Parameters

	Tuned	Block
P	4.6678	1
I	10.0841	1
D	-0.013375	0
N	348.9831	100

Performance and Robustness

	Tuned	Block
Rise time	0.441 seconds	7.42 seconds
Settling time	1.27 seconds	13.6 seconds
Overshoot	7.9 %	0 %
Peak	1.08	0.999
Gain margin	39.7 dB @ 40.3 rad/s	Inf dB @ Inf rad/s
Phase margin	60.1 deg @ 3.05 rad/s	97.1 deg @ 0.332 rad/s
Closed-loop stability	Stable	Stable

Figure 16 Different values for obtaining Underdamped Response

Arduino Code

```
#include "TimerOne.h"
unsigned int counter=0;
float t=0;
int enB=9;
int in3 = 8;
int in4 = 7;

void count()
{
  counter++;
}

void timer()
{if(counter==0)
   {t=0;}
else
   {t=t;}
  Timer1.detachInterrupt();  //stop the timer
  float rps = (counter / 20)*10;
  float rpm = rps*60;
  Serial.print(rps);
  Serial.print(",");
  Serial.print(rpm);
  Serial.print(",");
  Serial.println(t);
  counter=0;
  Timer1.attachInterrupt( timer );  //enable the timer
  t=t+0.1;
}
```

MATLAB Code

```
void setup()
{
 Serial.begin(9600);

 pinMode(enB, OUTPUT);
 pinMode(in3, OUTPUT);
 pinMode(in4, OUTPUT);

  Timer1.initialize(100000); // set timer for 0.1sec
  attachInterrupt(0, count, RISING);
  Timer1.attachInterrupt( timer ); // enable timer

}

void loop()
{
  int potvalue = analogRead(A1);
  int motorspeed = map(potvalue, 0, 680, 255, 0);
  // set speed of motor (0-255)
  analogWrite(enB, motorspeed);
  //rotation direction
  digitalWrite(in3, 0);
  digitalWrite(in4, 1);}
```

MATLAB Code

```
clc,clear,close all
B = 0.1 ; E = 6 ; J = 0.1 ; Kb = 0.1 ; Kt = 0.1 ; Rm = 0.1 ; Lm = 0.1
;
%%
var = xlsread ( 'C:\Users\Hammad\Desktop\data123.csv' ) ;
RPS_temp = nonzeros ( var ( : , 1 ) ) ;
RPS = zeros ( length ( RPS_temp ) + 1 , 1 ) ;
RPS ( 2 : end , 1 ) = RPS_temp ;
RPM_temp = nonzeros ( var ( : , 2 ) ) ;
RPM = zeros ( length ( RPS_temp ) + 1 , 1 ) ;
RPM ( 2 : end , 1 ) = RPM_temp ;
TIME_temp = nonzeros ( var ( : , 3 ) ) ;
TIME = zeros ( length ( TIME_temp ) + 1 , 1 ) ;
TIME( 2 : end , 1 ) = TIME_temp ;
data_temp = table ( RPS , RPM , TIME ) ;
[ temp , index ] = unique ( data_temp.TIME , 'stable' ) ;
data = data_temp ( index , : ) ;
temp8 = smooth ( data.RPS ) ;
temp9 = smooth ( data.RPM ) ;
for i=1:3
    temp8 = smooth(temp8);
    temp9 = smooth(temp9);
end
temp1 = temp8 ( 1 : 35 , 1 ) ;
temp_rpm = temp9 ( 1 : 35 , 1 ) ;
temp_time = data.TIME ( 1 : 35 , 1 ) ;
plot ( temp_time , temp1 )
grid on , xlabel ( "Time(sec)" ) , ylabel ( 'RPS' )
title ( "RESPONSE OF MOTOR (AFTER DATA CLEANING)" )
unit = data.TIME >= 0 ;
unit = double ( unit ) ;
var1 = ones ( length ( RPS ) , 1 ) ;
var1 ( 1 : length ( unit ) , 1 ) = unit ( : ) ;
unit = unit ( 1 : 35 , 1 ) * 6 ;
temp3 = [ temp_time unit ] ;
figure ,
plot ( temp_time , temp_rpm )
```

MATLAB Code

```
grid on , xlabel ( "Time(sec)" ) , ylabel ( 'RPM' )
title ( "RESPONSE OF MOTOR (AFTER DATA CLEANING)" )
%%
num = (Kt);
den = [(Lm*J) ((Lm*B)+(Rm*J)) ((Lm*B)+(Kb*Kt))];
sys = tf(num,den);
lsim(sys2,unit,temp_time)
hold on

plot (temp_time,temp_rpm,'color','k'),legend("Generated Transfer
Function",'COLLECTED DATA')
grid on , xlabel ( "Time(sec)" ) , ylabel ( 'RPM' )
title ( "RESPONSE OF MOTOR FROM PARAMETER ESTIMATED TRANSFER
FUNCTION" )
rlocus(sys);
%%
```

Conclusions

Through this CEP we successfully developed the mathematical model of a DC motor and analyzed it. We identified the different parameters of DC motor.

Starting off we determined the RPM value of motor itself. We had applied 6V DC step input and we got 210 RPM of motor

We determined the parameters of the motor through parameters estimation.

We also calculated the transfer function of the DC motor and through that we performed further analysis which included developing the root locus for the motor, developing Routh's Array through which we commented on the absolute stability of the motor and also determined the range of stability which came out to be $-0.1845 < K < +\infty$.

Furthermore, we also obtained different stable responses of the motor through a PID controller.

The different objectives that were assigned to us as a part of this CEP were successfully achieved and we were able to apply our theoretical understanding from the course to the practical implementation.