MCT 411: HYBRID CONTROL

Project Milestone 1

Rotary Inverted Pendulum

Team 3

Omar Khaled Mahmoud El Naggar	*18P8881*
Mahmoud Magdy Elasmar	*18P8983*
Amr Issa El-Sayed	*18P8601*
Ahmed Ossama El-Sayed	*18P8506*
Mark Awad Helmy	*18P1692*

Table of Contents

Table Of Figures 3

Abstract 5

Introduction 6

Linearizing equations of motion and modelling 7

  STATE-SPACE MODEL 7

  SIMSCAPE MODEL 8

Controlling methods 9

  PID CONTROL 9

  LQR CONTROL 10

CAD model 11

Integration and implementation 13

Simscape LQR Control MATLAB Model 14

Simscape PID Control MATLAB Model 18

Steady state equations with LQR Control Matlab Model 20

Hardware in the loop 23

  Using MATLAB m files 23

  Using Simulink 23

Video Links 25

  Simscape Multibody simulation 25

  real time performance 25

  video of project explanations 25

Appendix 26

  Motor parameter estimation code 26

  Model linearizer 27

  lqr.m 28

  Some function files 28

    Get_feedback.m 28

saturation.m 29

Table Of Figures

Figure 1 The Furuta Pendulum 7
Figure 2 3d model of our furuta pendulum 8
Figure 3 speed of motor in PID Simscape model 9
Figure 4 General LQR control 10
Figure 5 Parameter Estimation Setup 13
Figure 6 Parameter Estimation Output 13
Figure 7 Full LQR Simscape model 14
Figure 8 all_system subsystem 14
Figure 9 Simscape subsystem 15
Figure 10 motor subsystem 15
Figure 11 LQR Simscape Linearization script 16
Figure 12 LQR Simscape K gain script 16
Figure 13 Motor Torque and Voltage Input 17
Figure 14 Pendulum Angle 17
Figure 15 Simscape PID Full model 18
Figure 16 Simscape PID subsystem 18
Figure 17 PID motor subsystem 18
Figure 18 PID Simscape Pendulum angle and motor speed 19
Figure 19 Steady State LQR Model 20
Figure 20 Steady State Parameter Initialization Script 20
Figure 21 Steady State LQR calculations 21

Figure 22 State Space LQR Torque 21

Figure 23 State Space LQR Pendulum Angle 22

Figure 24 State Space LQR Motor Angle 22

Figure 25: hardware in the loop code 23

Abstract

An inverted pendulum is a pendulum that has its center of mass above its pivot point. It is unstable and without additional help will fall over. It can be suspended stably in this inverted position by using a control system to monitor the angle of the pole and move the pivot point horizontally back under the center of mass when it starts to fall over, keeping it balanced.

The Furuta pendulum is a version of the inverted pendulum that depends on rotary motion instead of linear motion.It consists of a driven arm which rotates in the horizontal plane and a pendulum attached to that arm which is free to rotate in the vertical plane.

It was invented in 1992 at Tokyo Institute of Technology by Katsuhisa Furuta and his colleagues. It is an example of a complex nonlinear oscillator of interest in control system theory.

The pendulum is underactuated and extremely non-linear due to the gravitational forces and the coupling arising from the Coriolis and centripetal forces. Since then, dozens, possibly hundreds of papers and theses have used the system to demonstrate linear and non-linear control laws

Introduction

In this project, we aimed to achieve a couple of goals.

First: we aimed to model and simulate a *rotary inverted pendulum,* also known as a ***Furuta Pendulum****.* For that, we needed to understand and apply multiple things, starting with finding the pendulum's equations of motion, and by *linearizing* them, we would be able to simulate our system by using *steady state equations in* ***Simulink****.* There was also another option, which is to try and model the system as accurately as possible using 3d models and make a fully detailed model using ***Simscape*** and applying appropriate parameters as accurately as possible to real life.

Second: we aimed to apply different control methods on those models to keep the pendulum balanced vertically, without it swaying to the sides.
We tried different control methods including:

- ***PID control***
- ***LQR control***

Third, and finally: we aimed to integrate these simulations with our in real life model and apply these control methods to achieve the same results we got from our simulations and estimations by using matlab scripts and matlab HIL.

At first, we will list out all of the methods and techniques used to obtain our results, then at the end, all of the models and different outputs showcased.

Linearizing equations of motion and modelling

In this section, we will specify how we manually linearized the equations of the inverted pendulum and also how we used matlab scripts to automatically linearize it for us.

STATE-SPACE MODEL

By using this publication as a reference, we can see that the equation of moions for the pendulum are:

$$\begin{aligned}(\alpha + \beta \sin^2 \theta)\, \ddot{\phi} + \gamma \cos\theta \ddot{\theta} + 2\beta \cos\theta \sin\theta \dot{\phi}\dot{\theta} - \gamma \sin\theta \dot{\theta}^2 = \tau_\phi \\ \gamma \cos\theta \ddot{\phi} + \beta \ddot{\theta} - \beta \cos\theta \sin\theta \dot{\phi}^2 - \delta \sin\theta = \tau_\theta\end{aligned}$$

And by rewriting that in Matrix form, we can get the following equation:

$$D(\phi, \theta) \begin{pmatrix} \ddot{\phi} \\ \ddot{\theta} \end{pmatrix} + C(\phi, \theta, \dot{\phi}, \dot{\theta}) \begin{pmatrix} \dot{\phi} \\ \dot{\theta} \end{pmatrix} + g(\phi, \theta) = \tau$$

And by rewriting the equation and introducing the state variable:

$$x \triangleq \begin{pmatrix} \phi \\ \dot{\phi} \\ \theta \\ \dot{\theta} \end{pmatrix}$$

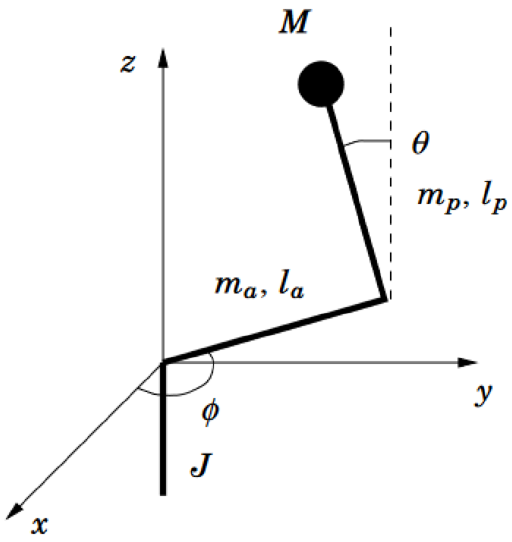


Figure 1 The Furuta Pendulum

And having x = [0, 0, 0, 0], We get:

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -\frac{\delta\gamma}{\alpha\beta-\gamma^2} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{\alpha\delta}{\alpha\beta-\gamma^2} & 0 \end{pmatrix}, \; B = \begin{pmatrix} 0 & 0 \\ \frac{\beta}{\alpha\beta-\gamma^2} & -\frac{\gamma}{\alpha\beta-\gamma^2} \\ 0 & 0 \\ -\frac{\gamma}{\alpha\beta-\gamma^2} & \frac{\alpha}{\alpha\beta-\gamma^2} \end{pmatrix}$$

We also tried using a linear inverted pendulum model to linearize, but in the end, the performance wasn't quite up to bar or accurate as the rotary version, so we scrapped that.

We could use the equations we deduced and inserted it into the state-space Simulink block to model our systems.

SIMSCAPE MODEL

We also designed our system on inventor and imported our inventor model into MATLAB and turned it into a Simscape model on Simulink that includes all of the correct mass and other parameters for all parts of the system.

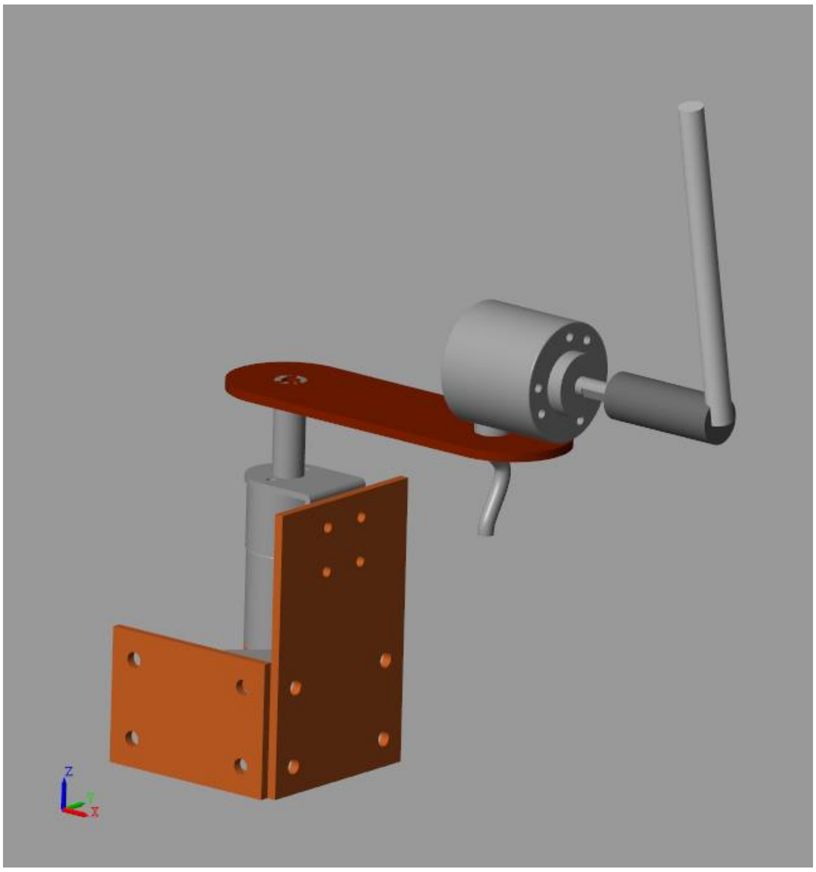


Figure 2 3d model of our furuta pendulum

After that, we used a ***linearization matlab function*** in a script to automatically calculate the system's equations.

We could then take the A matrix and B matrix elements from the linearized system object to use in our calculations for ***LQR control*** down the line.

Controlling methods

PID CONTROL

We applied PID control on our Simscape model by taking the angle of the pendulum as feedback, and manually tuning the PID parameters to keep the pendulum angle vertical

However, when we only took 1 state into consideration, being the angle of the pendulum, the motor kept generally moving at a constant speed each time a disturbance happened, instead of overshooting and coming to a stop, it kept endlessly rotating, which is not ideal.

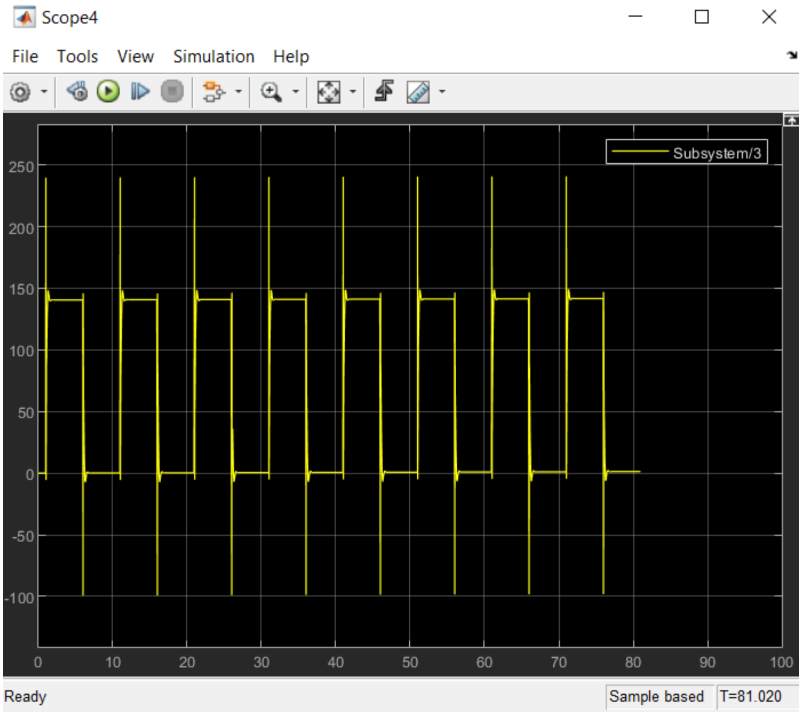


Figure 3 speed of motor in PID Simscape model

In the figure shown above, a disturbance happens every 10 seconds at 1 second delay, and another disturbance in the opposite direction happens every 10 seconds at 6 second delay. As we can see, the motor stayed rotating at 140 deg/s until the second disturbance happened that made the motor stop.

If both disturbances were in the same direction, the motor would keep gaining more and more speed, infinitely increasing. By limiting the speed and adding a threshold to it, the pendulum would simply fall down when it couldn't obtain that extra speed.

LQR CONTROL

We applied LQR (Linear Quadratic Regulation) control on both the state space model, and the Simscape model.

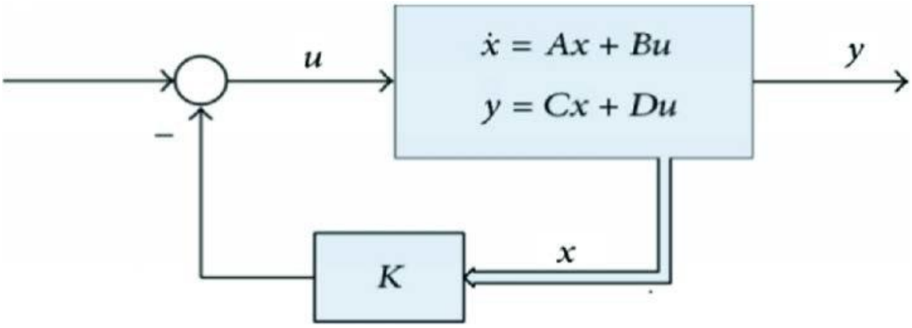


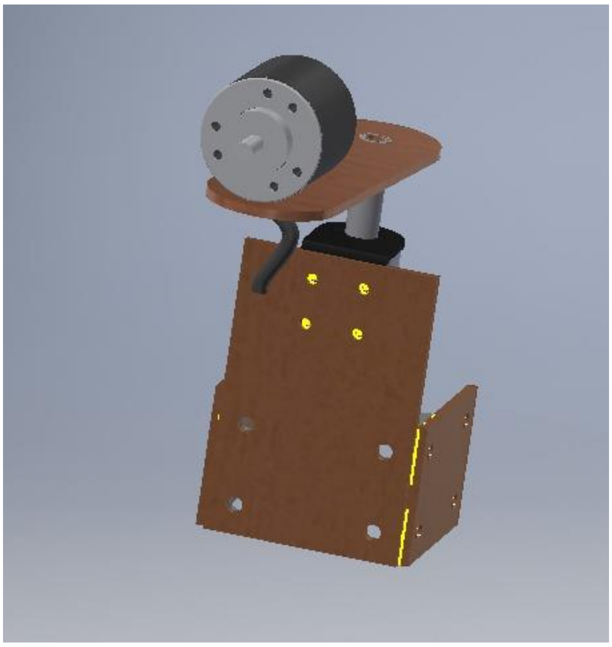
Figure 4 General LQR control

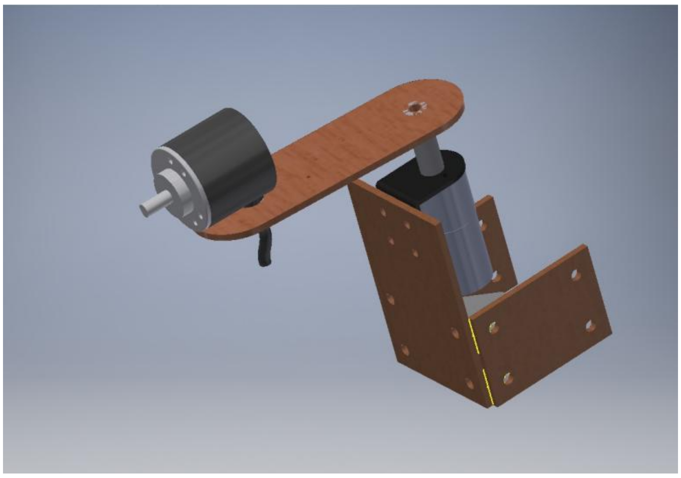
To apply LQR, we used the LQR MATLAB function to calculate the K gain matrix in both cases.

Both models provided great output and performance.

CAD model

This is our CAD model that we have designed.



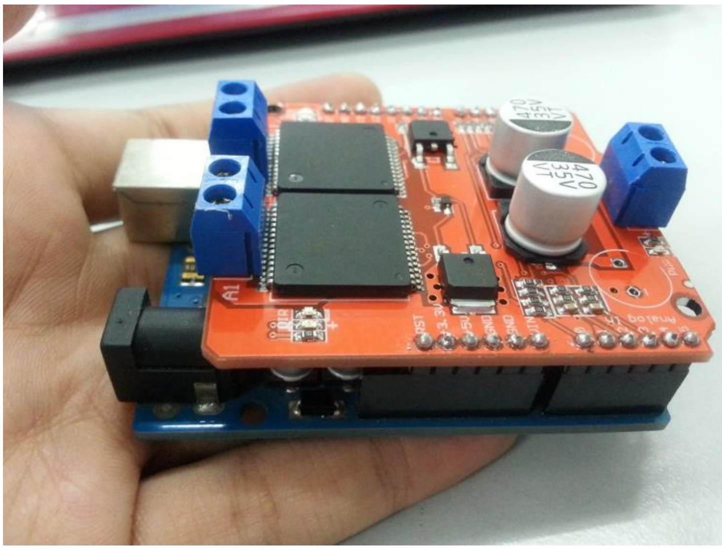


We used the following components:

- 12V DC Motor with Encoder

- Rotary Encoder 360 pulse per rev, 2 phase so may be multiplexed and pulses turn to 1440 pulse per revolution.

- Motor Shield



Integration and implementation

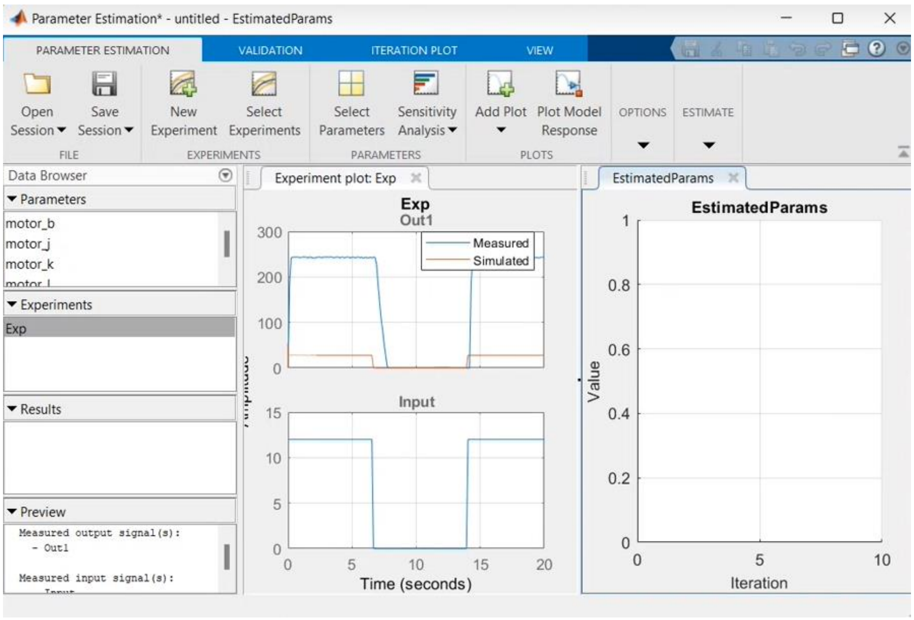


Figure 5 Parameter Estimation Setup

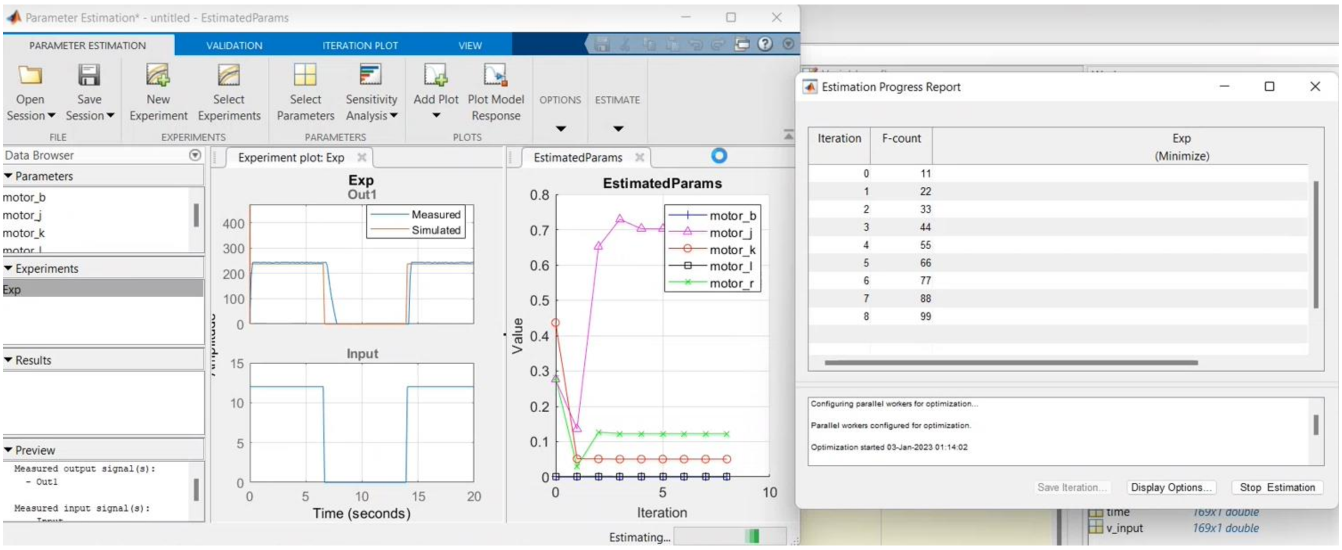


Figure 6 Parameter Estimation Output

Simscape LQR Control MATLAB Model

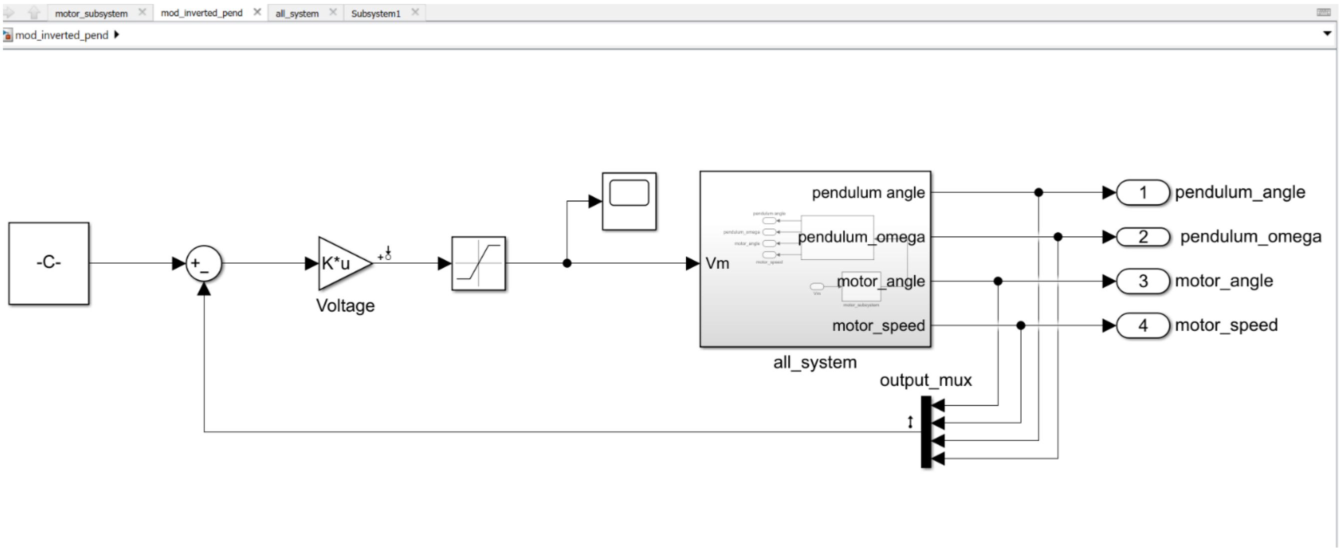


Figure 7 Full LQR Simscape model

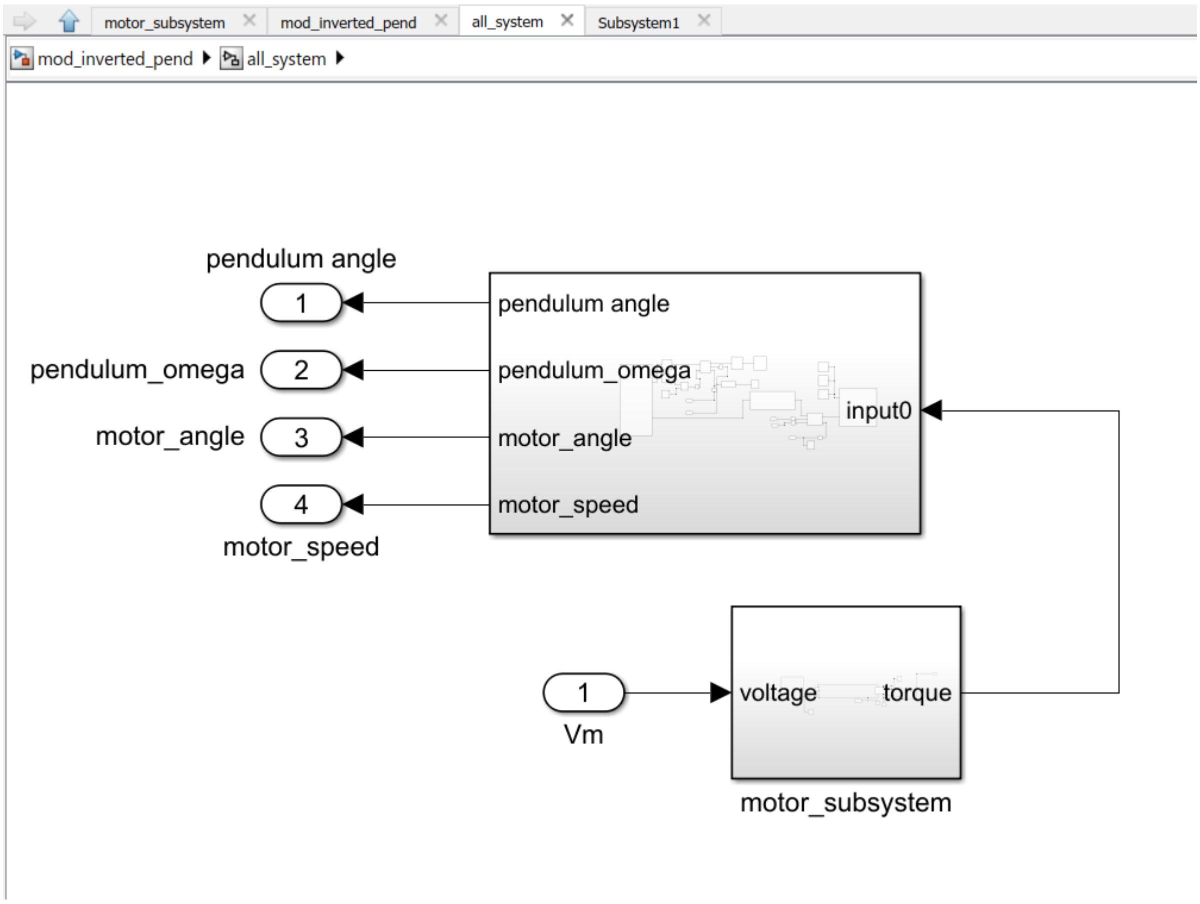


Figure 8 all_system subsystem

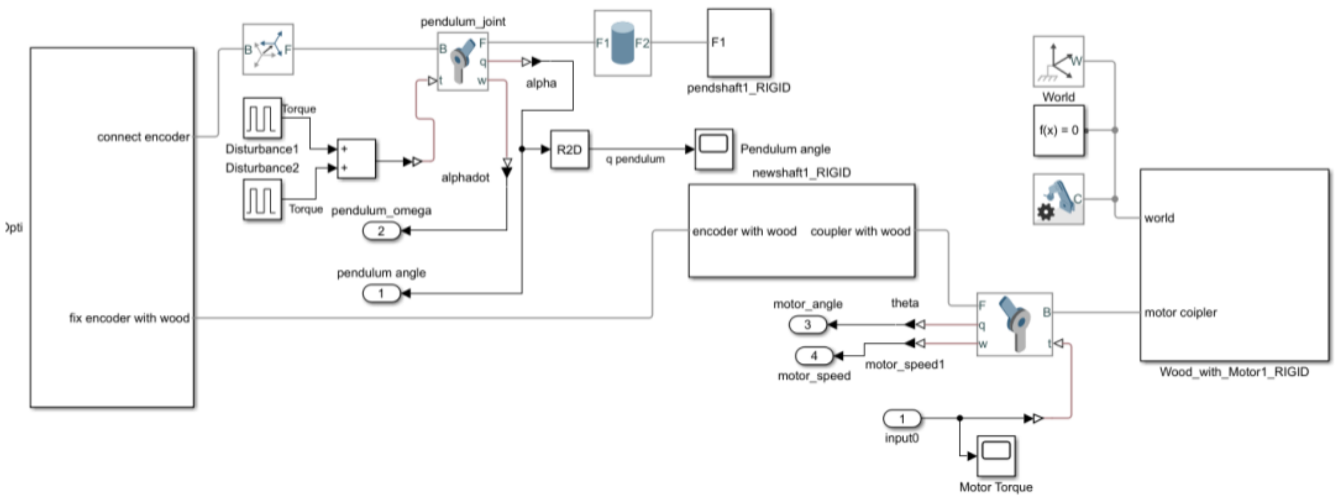


Figure 9 Simscape subsystem

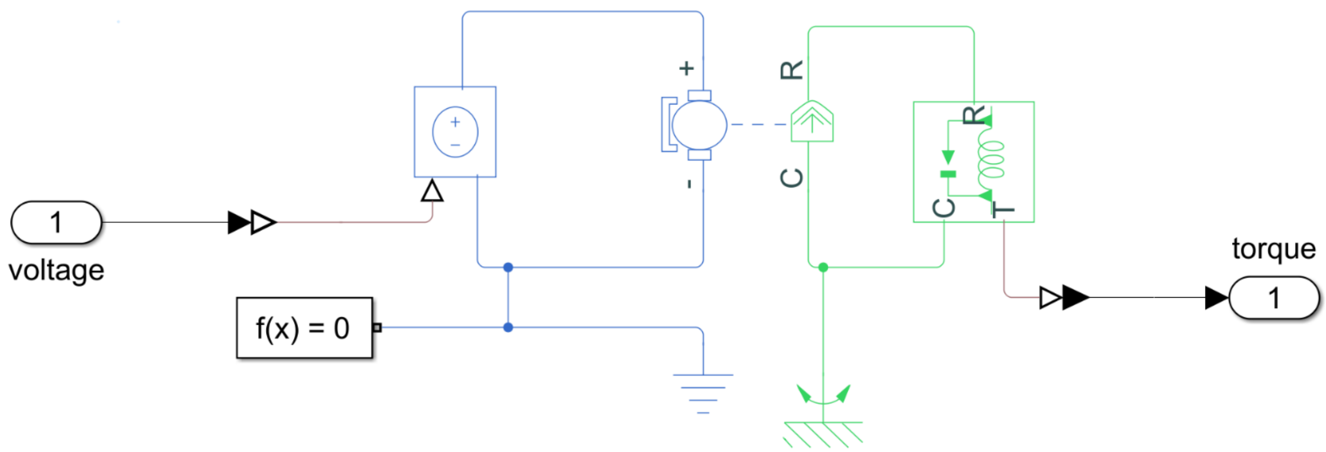


Figure 10 motor subsystem

Editor - C:\Solar\College stuff\term 9\Hybrid control\project\final\lin_mod.m

tfparameter.m | ssparameter.m | ssparameter_futura.m | all2_DataFile.m | lin_mod.m

```matlab
%% Linearized System Matrices
% K-Matrix Initialization
K=[0 0 0 0];
refernce_values=[(3.14/6) 0 0 0];
K_final = [0 0 0 0];

mdl='mod_inverted_pend'
linsystem=[mdl '/all_system']
io(1) = linio('mod_inverted_pend/Voltage',1,'input');
io(2) = linio('mod_inverted_pend/output_mux',1,'openoutput');
linsys1 = linearize(mdl,io)

```

Figure 11 LQR Simscape Linearization script

Editor - C:\Solar\College stuff\term 9\Hybrid control\project\final\lqr_sus.m

tfparameter.m | ssparameter.m | ssparameter_futura.m | all2_DataFile.m | lin_mod.m | lqr_sus.m

```matlab
refernce_values=[(3.14/6) 0 0 0];
A=linsys1.A;
B=linsys1.B

Q=[10    0    0    0     0 ;
   0     1    0    0     0;
   0     0    10   0     0;
   0     0    0    1     0;
   0     0    0    0     1; ]
R=1;

[K,eigen,s]=lqr(A,B,Q,R)
K_final=K
```

Figure 12 LQR Simscape K gain script

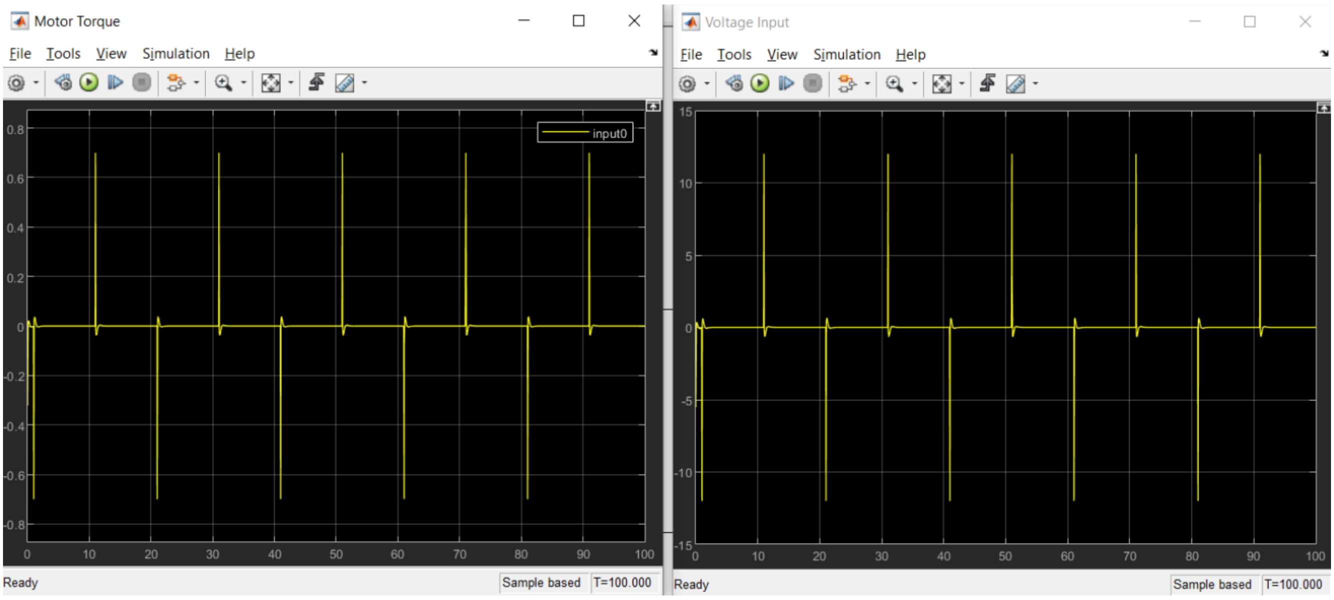


Figure 13 Motor Torque and Voltage Input

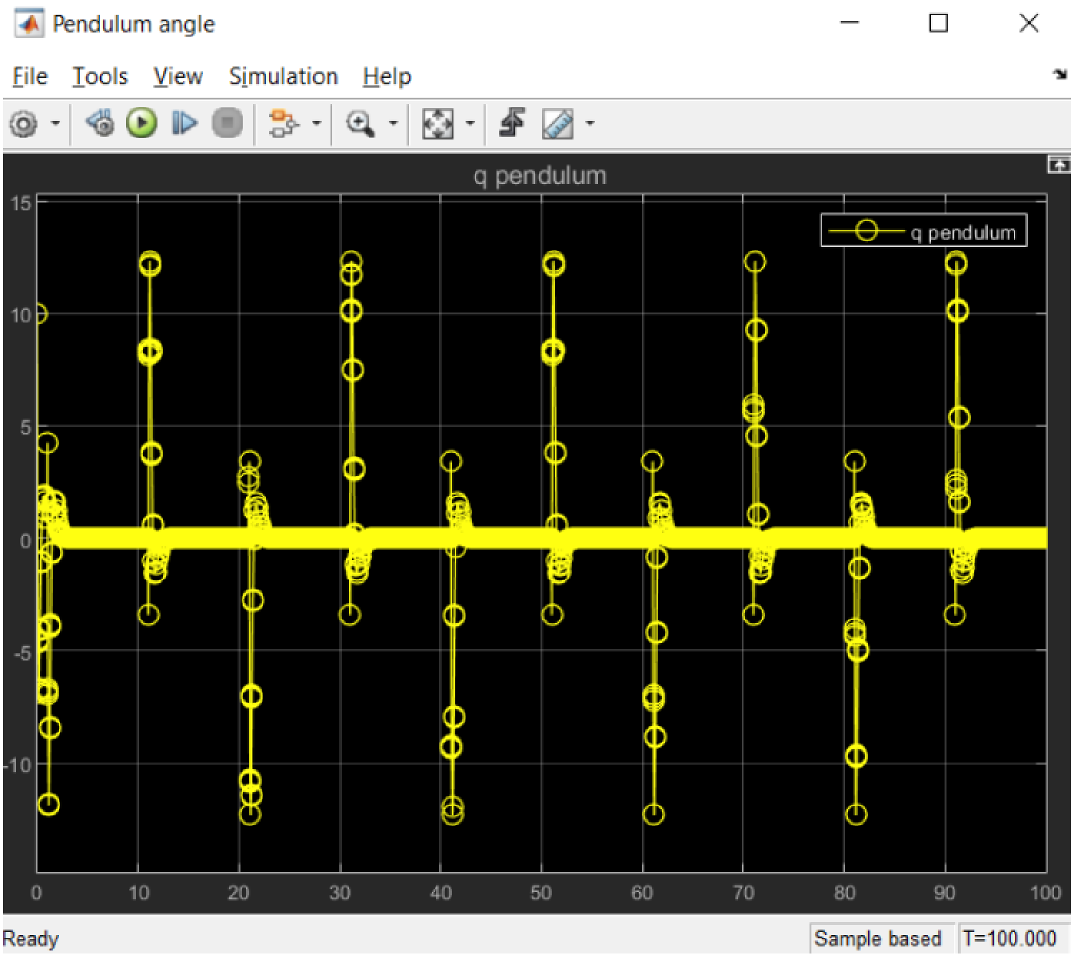


Figure 14 Pendulum Angle

Simscape PID Control MATLAB Model

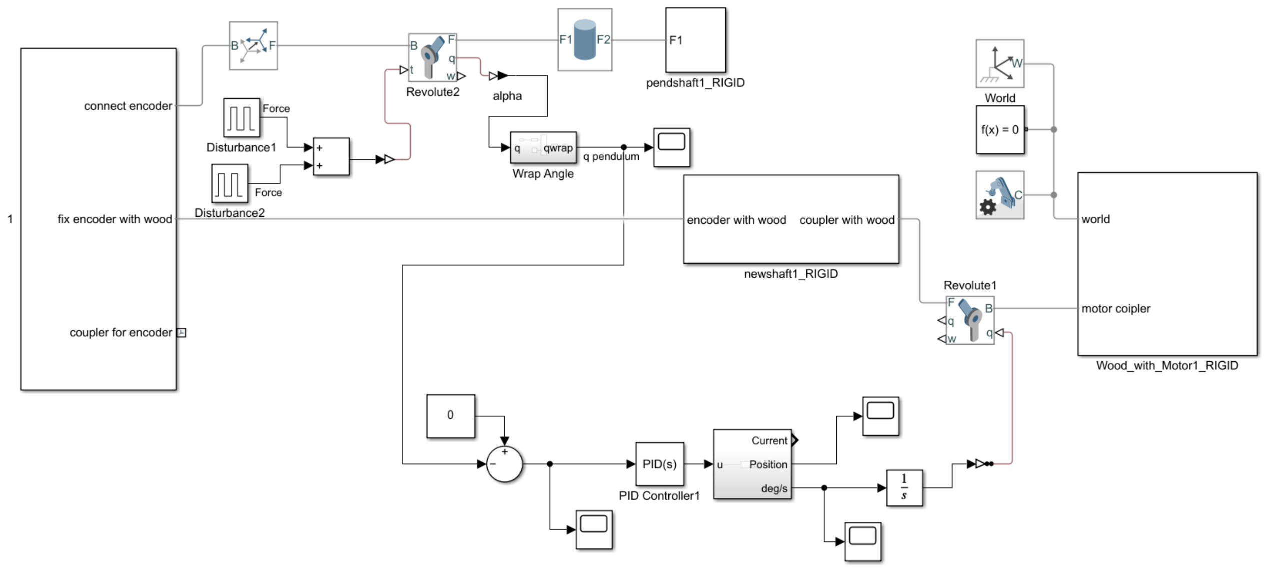


Figure 15 Simscape PID Full model

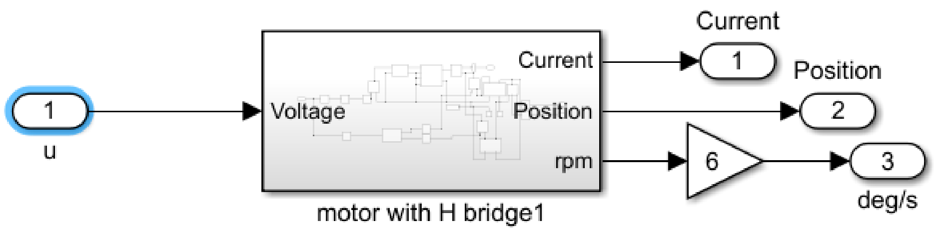


Figure 16 Simscape PID subsystem

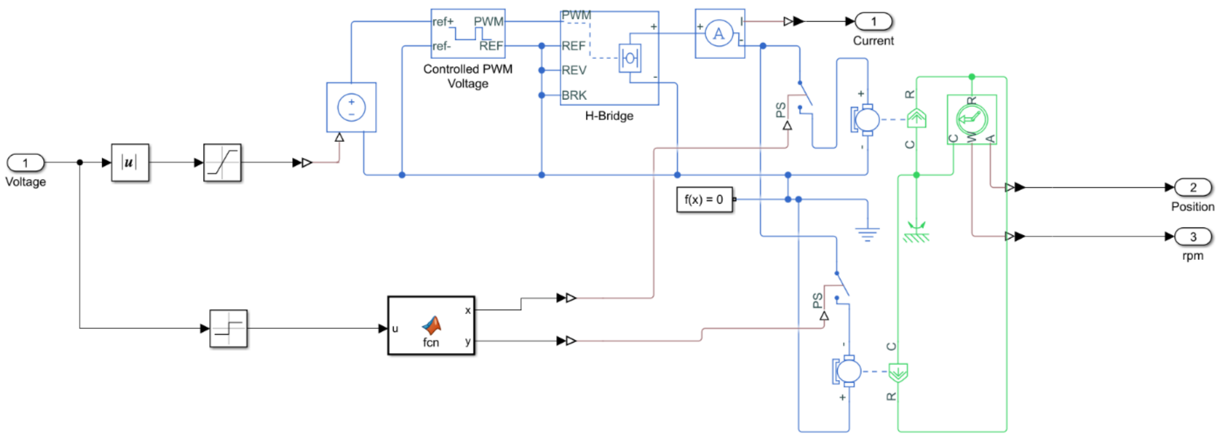


Figure 17 PID motor subsystem

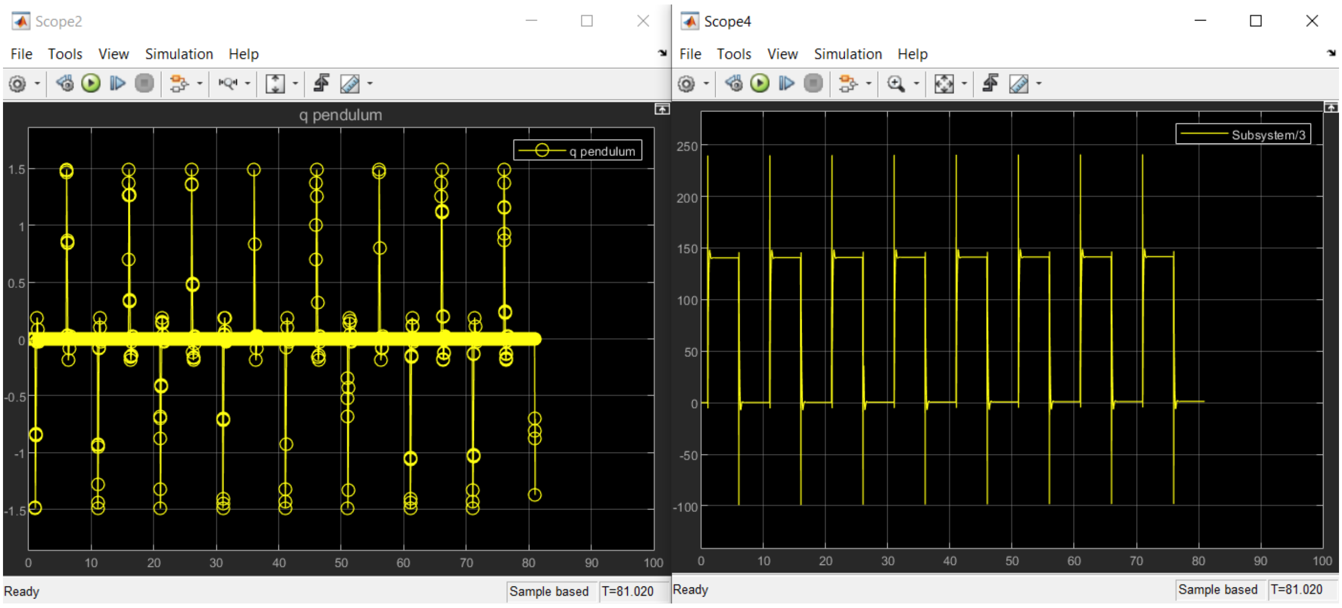


Figure 18 PID Simscape Pendulum angle and motor speed

Steady state equations with LQR Control Matlab Model

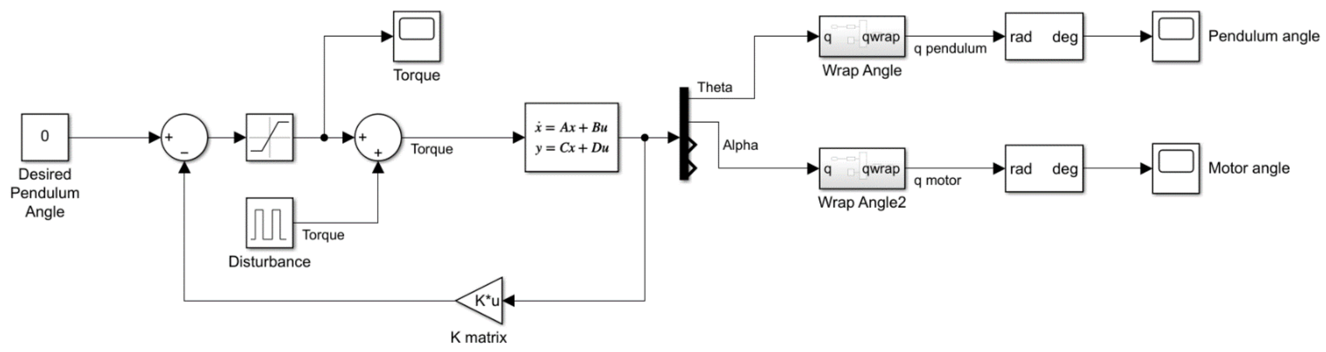
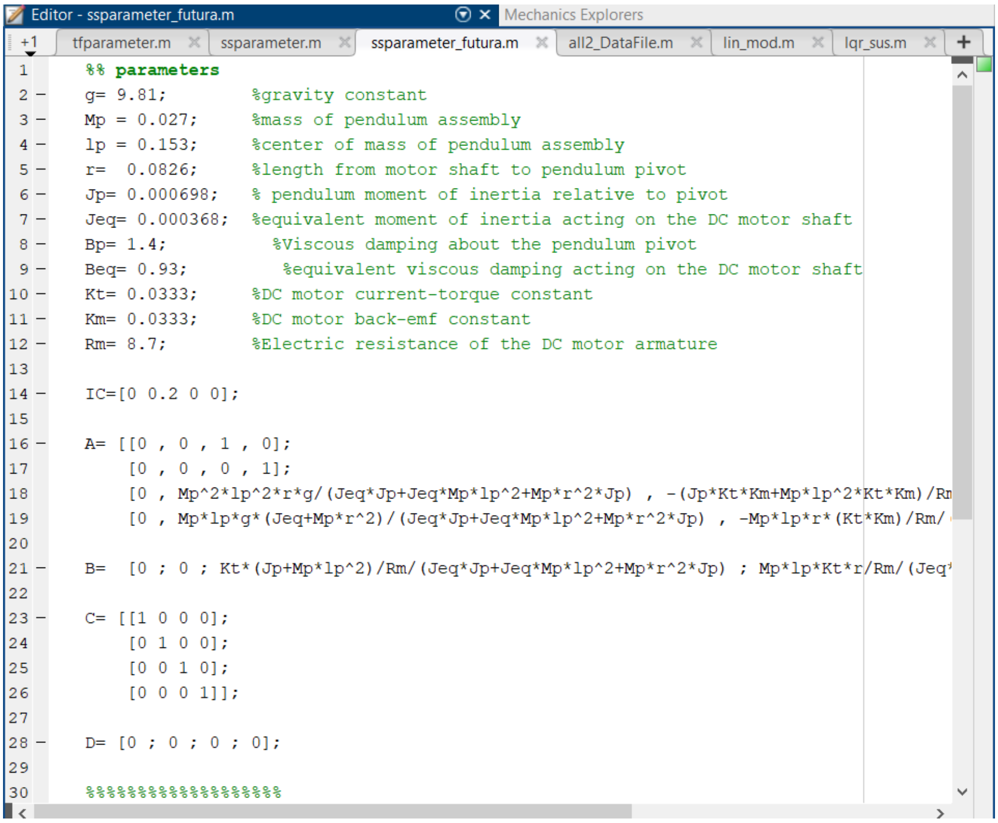


Figure 19 Steady State LQR Model




```matlab
%% parameters
g= 9.81;        %gravity constant
Mp = 0.027;     %mass of pendulum assembly
lp = 0.153;     %center of mass of pendulum assembly
r=  0.0826;     %length from motor shaft to pendulum pivot
Jp= 0.000698;   % pendulum moment of inertia relative to pivot
Jeq= 0.000368;  %equivalent moment of inertia acting on the DC motor shaft
Bp= 1.4;          %Viscous damping about the pendulum pivot
Beq= 0.93;         %equivalent viscous damping acting on the DC motor shaft
Kt= 0.0333;     %DC motor current-torque constant
Km= 0.0333;     %DC motor back-emf constant
Rm= 8.7;        %Electric resistance of the DC motor armature

IC=[0 0.2 0 0];

A= [[0 , 0 , 1 , 0];
    [0 , 0 , 0 , 1];
    [0 , Mp^2*lp^2*r*g/(Jeq*Jp+Jeq*Mp*lp^2+Mp*r^2*Jp) , -(Jp*Kt*Km+Mp*lp^2*Kt*Km)/Rm
    [0 , Mp*lp*g*(Jeq+Mp*r^2)/(Jeq*Jp+Jeq*Mp*lp^2+Mp*r^2*Jp) , -Mp*lp*r*(Kt*Km)/Rm/

B=  [0 ; 0 ; Kt*(Jp+Mp*lp^2)/Rm/(Jeq*Jp+Jeq*Mp*lp^2+Mp*r^2*Jp) ; Mp*lp*Kt*r/Rm/(Jeq*

C= [[1 0 0 0];
    [0 1 0 0];
    [0 0 1 0];
    [0 0 0 1]];

D= [0 ; 0 ; 0 ; 0];

%%%%%%%%%%%%%%%%%%%
```

Figure 20 Steady State Parameter Initialization Script

```
%% lqr

states = {'theta' 'alpha' 'theta_dot' 'alpha_dot'};
inputs = {'u'};
outputs = {'theta'; 'alpha'; 'theta_dot'; 'alpha_dot'};

sys_ss = ss(A,B,C,D,'statename',states,'inputname',inputs,'outputname',outputs);
co = ctrb(sys_ss);
controllability = rank(co);

Q = C'*C;

Q(1,1) = 5000;
Q(2,2) = 100;
R = 1;
K = lqr(A,B,Q,R)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Figure 21 Steady State LQR calculations

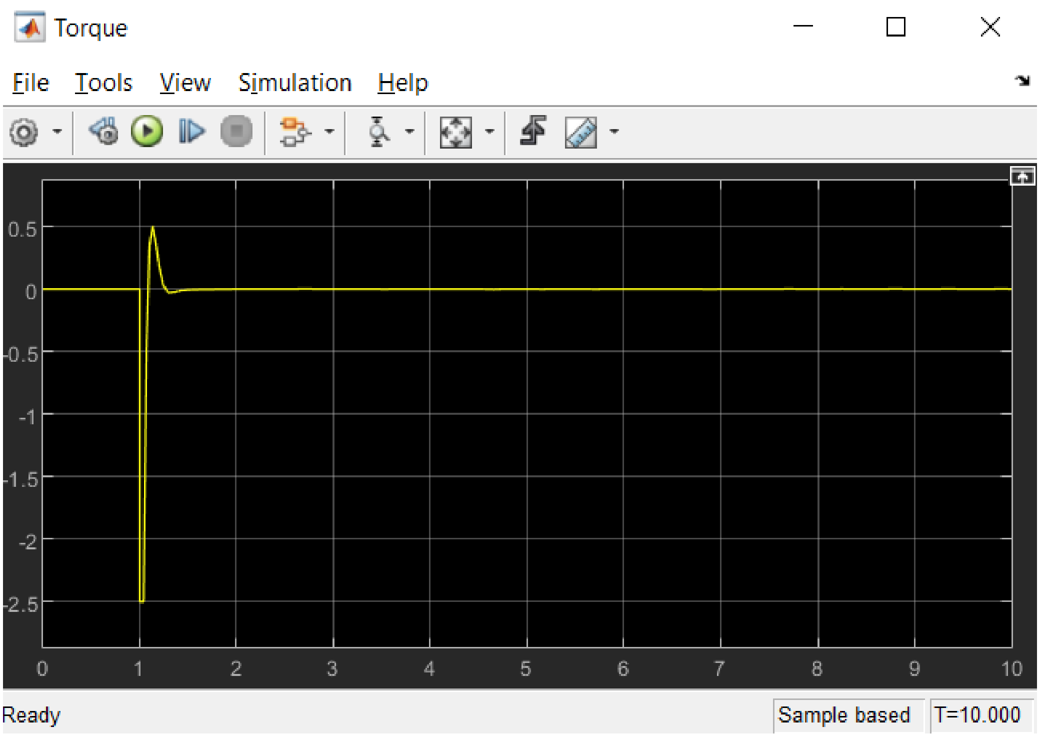


Figure 22 State Space LQR Torque

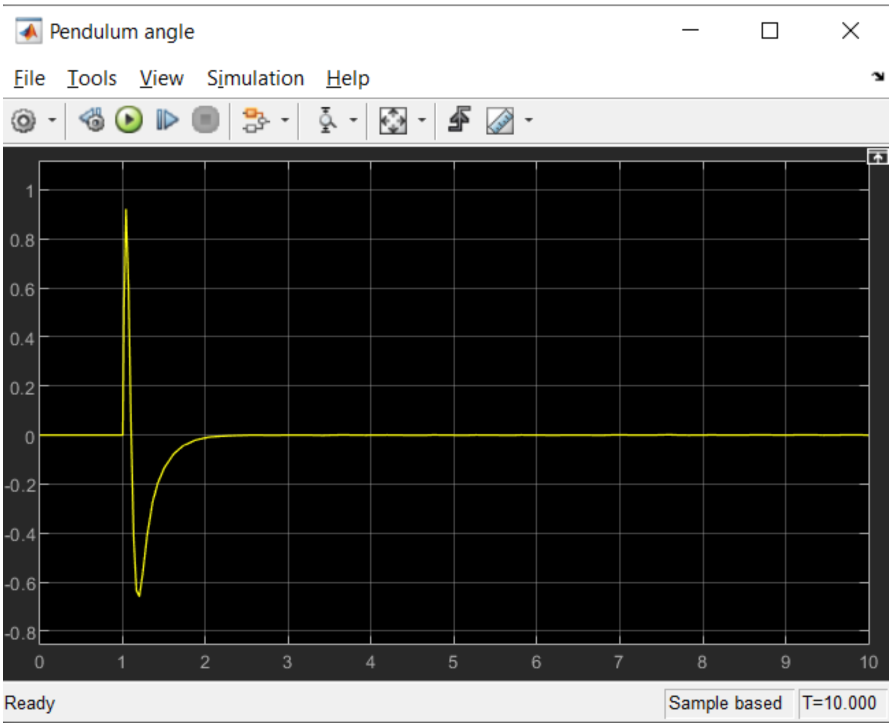


Figure 23 State Space LQR Pendulum Angle

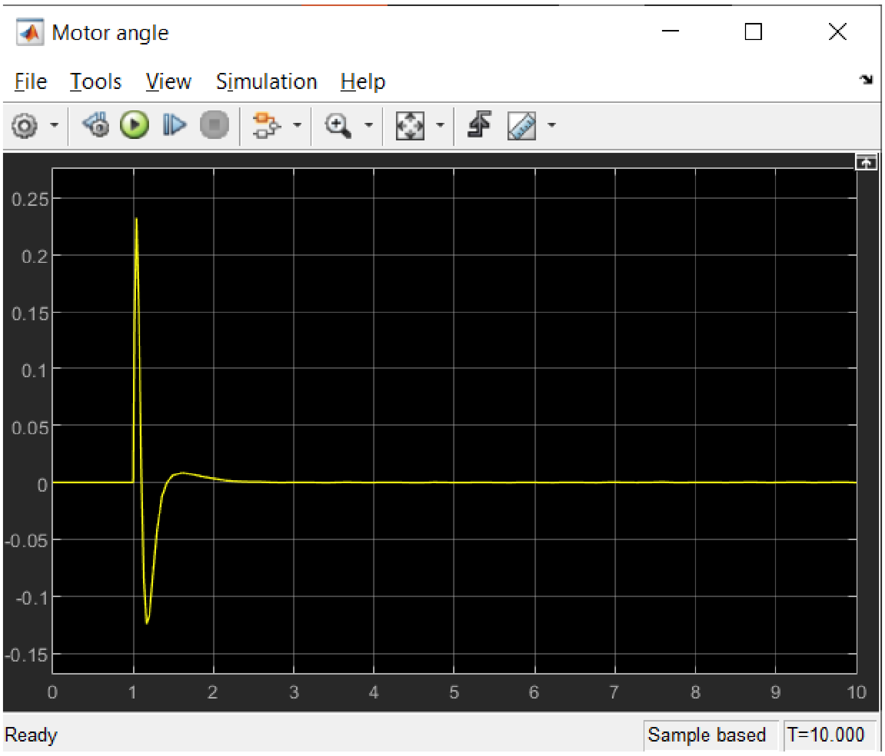


Figure 24 State Space LQR Motor Angle

Hardware in the loop

Using MATLAB m files

Using Arduino Io library in MATLAB we were able to use two Arduino uno and created two encoder object one attached to each Arduino.

The code is attached in the appendix

Using Simulink

We used the Simulink Hardware Support Package for Arduino to write code that reads the pendulum encoder and the motor encoder and apply out control algorithm

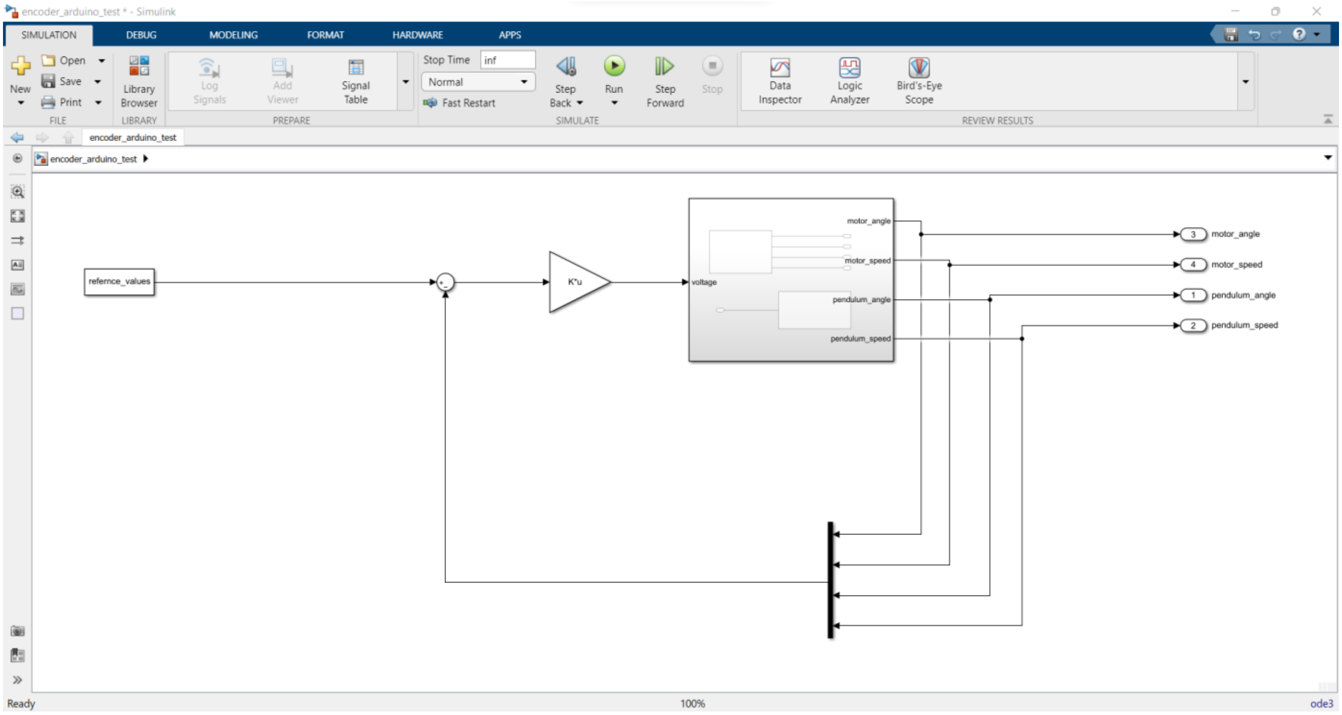
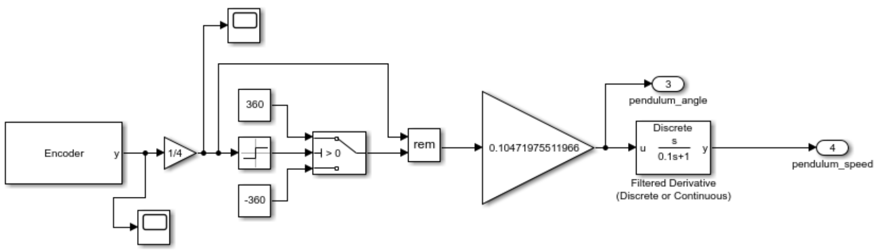
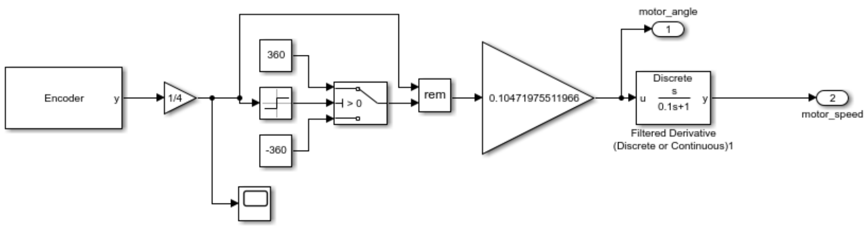
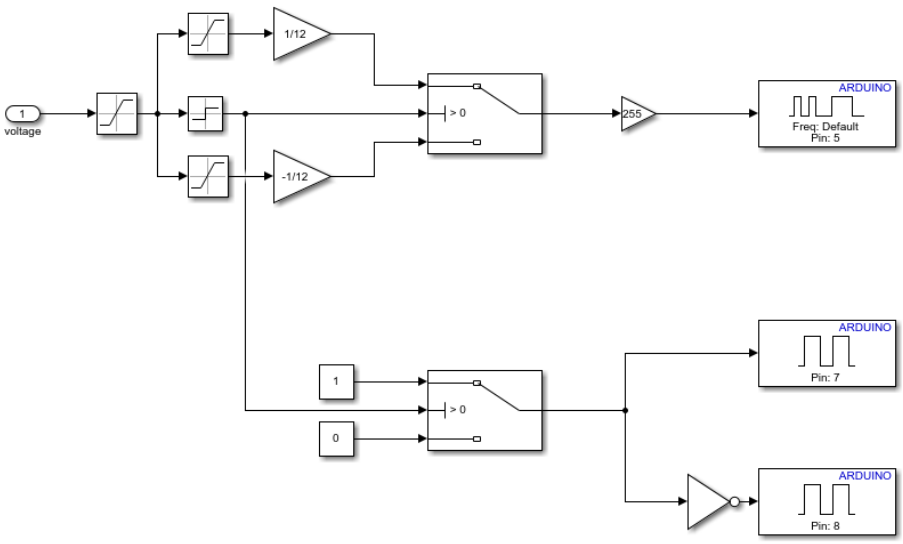


Figure 25: hardware in the loop code


Encoder
y
1/4
360
> 0
-360
rem
0.10471975511966
3
pendulum_angle
Discrete
s
0.1s+1
u
y
Filtered Derivative
(Discrete or Continuous)
4
pendulum_speed


Encoder
y
1/4
360
> 0
-360
rem
0.10471975511966
motor_angle
1
Discrete
s
0.1s+1
u
y
Filtered Derivative
(Discrete or Continuous)1
2
motor_speed


1
voltage
1/12
> 0
-1/12
255
ARDUINO
Freq: Default
Pin: 5
1
0
ARDUINO
Pin: 7
ARDUINO
Pin: 8

Video Links

Simscape Multibody simulation

https://drive.google.com/drive/folders/1zdPZ2cXnkS0gApTIgiaa04q0Ty0_LqDH?usp=share_link

real time performance

https://drive.google.com/drive/folders/1UCLyvGeGcu2_uVO_NML1-lcAwYkvcN68?usp=share_link

video of project explanations

https://drive.google.com/file/d/1jxREDyPpEztUKR24KXZmhzqwfPtvkhEn/view?usp=share_link

Appendix

Motor parameter estimation code

```
%%
global a
a = arduino('COM3','Uno','libraries','RotaryEncoder');
%%
global b
b=arduino('COM6','Mega2560','libraries','RotaryEncoder');
%%
y(b,1);
%%
writeDigitalPin(b,'D13',0);

%%
encoder_a = rotaryEncoder(a,"D2",'D3',1440);
%%
motor1SpeedPin = 'D5';
motor1Direction_cw_Pin = 'D7';
motor1Direction_ccw_Pin = 'D8';
motor_1_enable='A0';

%%
  direction =0;
  enable=0
  initialPWMVoltage =1;
  duty=1;
  writeDigitalPin(a,motor_1_enable,enable);
  writeDigitalPin(a,motor1Direction_cw_Pin,direction);
  writeDigitalPin(a,motor1Direction_ccw_Pin,~(direction));
  %writePWMVoltage(a,motor1SpeedPin,initialPWMVoltage);
  writePWMDutyCycle(a,motor1SpeedPin,duty)
  pause(1);
 %%
executionTime = 20;
period = 0.1;
rpm=zeros(1,200);
i=1;
%%
prev_count=0;
new_count=0;
%%
```

```
tic;
while toc < executionTime
 new_count=readCount(encoder_a);
 fprintf('new: %.2f\n',new_count);
 rpm(i)=(60*(new_count-prev_count)/2880)/(period);
 prev_count=new_count;
 fprintf('old: %.2f\n',prev_count);
 if toc<0.35*executionTime
     enable=1;
     writeDigitalPin(a,motor_1_enable,enable);
 end
 if toc>0.35*executionTime && toc<0.6*executionTime
     enable=0;
     writeDigitalPin(a,motor_1_enable,enable);
 end
 if toc>0.7*executionTime && toc<executionTime
     enable=1;
     writeDigitalPin(a,motor_1_enable,enable);
 end
%rpm

 %count = readCount(encoder_a);
 %fprintf('Current motor speed is: %.2f\n',rpm);
 i=i+1;
 pause(period)
end
resetCount(encoder_a);
enable=0;
writeDigitalPin(a,motor_1_enable,enable);
%%
plotted_rpm=rpm(1:120);
t_ploted=t(1:120);
%%
plot(t_ploted,plotted_rpm)
```

Model linearizer

```
%% Linearized System Matrices
% K-Matrix Initialization
K=[0 0 0 0];
K_final=[0 0 0 0];
refernce_values=[0; 0 ;0; 0];
```

```
mdl='mod_inverted_pend'
linsystem=[mdl '/all_system']%plant
io(1) = linio('mod_inverted_pend/Voltage',1,'input');
io(2) = linio('mod_inverted_pend/output_mux',1,'openoutput');
linsys1 = linearize(mdl,io)
```

---

lqr.m

```
refernce_values=[0; 0;0;0 ];
A=linsys1.A;
B=linsys1.B

Q=[1    0   0   0     0 ;
   0    1   0   0     0;
   0    0   500  0     0;
   0    0   0   1     0;
   0    0   0   0     1; ]
R=120;

[K,eigen,s]=lqr(A,B,Q,R)

K_final=K
```

Some function files

Get_feedback.m

```
function out=get_feedback()
global alpha dalpha theta dtheta;
global encoder_motor encoder_enc

%global prev_count_motor now_count_motor prev_count_encoder
now_count_encoder;

alpha=(readCount(encoder_enc)/4);
if alpha>0
    alpha=rem(alpha,360)*(3.14/180);
else
    alpha=rem(alpha,-360)*(3.14/180);
end
dalpha=readSpeed(encoder_enc)*0.104719755;
theta=readCount(encoder_motor)/4;
```

```
if theta>0
    theta=rem(theta,360)*(3.14/180);
 else
    theta=rem(theta,-360)*(3.14/180);
end
dtheta=readSpeed(encoder_motor)*0.104719755;
out=[ theta; dtheta;alpha; dalpha]
```

---

saturation.m

```
function satu_=sat_(inp_val,max,min)
if inp_val>max
    satu_=max;
    return
end
if inp_val<min
    satu_=min;
    return
else
    satu_=inp_val;
    return
end
```