Final Project Justin Garcia

Table of Contents

Instructor: Saeid Bashash Lab Instructor: Nishit Patel Lab Section: Wednesday 10AM-12:45PM 12/14/16

Background

```
Mechatronics is a dynamic field of study due to its necessity for the
practicing engineer to know the fundamentals of many different areas of
engineering. Control Theory is one of the most important functions in
this area, aiming to make closed loop systems that modify their output
by using information of the output provided by onboard sensors. These
controllers must be quick to respond and not overshoot or undershoot
the target output by too much.

% Control theory is used most often in process engineering but has
applicability to any any system that can get feedback from sensors.
As more systems start to include sensors the applications will grow
exponentially, due to new IoT devices and new smart city systems, as
well as drone technology and most modern electronics.
```

Objective

|% The objective of this project was to use the concepts taken from the previous labs and apply them towards making the MinSeg robot stand up on its wheels. In addition to what was previously learned the other objective was to learn how to develop an automatic control system that helps the MinSeg robot stabilize itself through continuous feedback from its onboard sensors. The controller is created by modeling the system dynamics and using that information to adapt a controller for it.

The MinSeg robot that is used in this project is an inverted pendulum that consists of an arduino Mega 2560 that is on a lego NXT base.|

Modeling the Sytem: Hand Calulations

|The following pages are the derivations for the system of equations for the dynamics of the MinSeg.

In order to find the dynamics of the system it was necessary to take the sum of the moments of inertia about the point of contact between the wheels and the ground.

Next by taking the sum of the forces in the x direction of the forces at the point on the pendulum where it contacts the wheel shaft reveals an expression Fx that can be incorporated into EQ1.

To solve for the Torque of the system (given by the motor), the differential equation of the circuit is derived by solving Kirchoff's voltage law.

In order to calculate the second equation of the system it is necessary to take the moment about the same point on the pendulum. These equations then are used to define the State-Space Solution.|

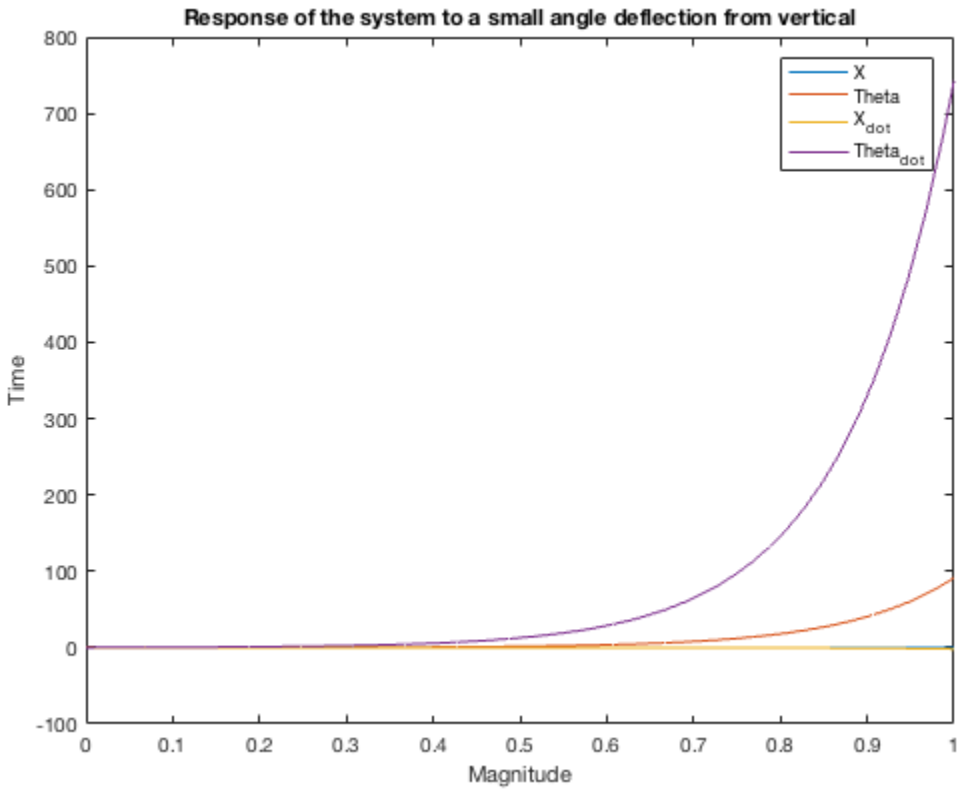
Start of the Matlab portion

```
% Constants of World

g    = 9.81;

% meters per second squared

% Constants of wheel

r_wh = .022;
M_wh = .025;
J_wh = .0013;

% Contants of Pendulum

L_p  = .1;
M_p  = .285;
J_p  = .001;

% Constants of Motor

K_b  = .5;
K_t  = .3;
C    = .0001;
R    = 5; %Ohms

%

System Equation Matrices: M(Xdd)+D(Xd)+K(X)=F(U) These are the matrices
derived from the system equations.

M             = [((J_wh)/(r_wh^2))+M_wh+M_p M_p*L_p;-M_p*L_p -(J_p
+(M_p*(L_p^2)))];
D             = [((K_t*K_b)/(R*r_wh^2))+(C/(r_wh^2)) 0; (C/
r_wh)+(((K_b*K_t)/(R*r_wh))) 0];
K_equation    = [0 0; 0 M_p*g*L_p];
F             = [K_t/(R*r_wh);K_t/R];

% Converting the sytem matrices to state space matirces requires the
use of these matrices following the equation of conversion.

initial    = [1 0;0 1];
A_zero     = [0 0;0 0];
B_zero     = [0;0];
```

```
% Matrices for the State-Space Solution

A    = [A_zero initial; -inv(M)*K_equation -inv(M)*D];
B    = [B_zero;inv(M)*F];
C    = eye(4);
D2   = [0;0;0;0];
```

Part 1.8

```
sim('Project_1_8.slx');
figure(1)
plot(Time_1,X_1,Time_1,Theta_1,Time_1,X_dot_1,Time_1,Theta_dot_1);
xlabel('Magnitude');
ylabel('Time');
title('Response of the system to a small angle deflection from
 vertical');
legend('X','Theta','X_d_o_t','Theta_d_o_t');
eig(A);
```




```
% From this graph we can see the reaction of the system to a small
input of pi/60 degrees. Interestingly the output of x goes towards the
small value of -.25 and x_dot goes to negative 2. Theta goes towards
a hundred and Theta_dot goes towards eight hundred. This implies that
the angle of the system has the most effect on its stability, and that
it is indeed unstable, which all inverted pendulums are. This is backed
up by the fact that there is a zero value and a positive value in the
```

eigenvalues of the matrix A. The transfer function of a system can be determined by the equation G(s) = C*[Is-A]^-1*B + D where A,B,C,D are the matrices of the statespace solution. The key to stability analysis is the denominator of the transfer function, which is given to us by the inverse of [Is-A], due to the calculation of its determinant. In this sense the eigenvalues of matrix A are analogous to the s values of the transfer function. The zero value indicates a asystem that is at best critically stable, while the positive value does show the system to be unstable as expected due to the gravity and the intial inertia, without a couteracting force the pendulum will never return to the starting vertical position.

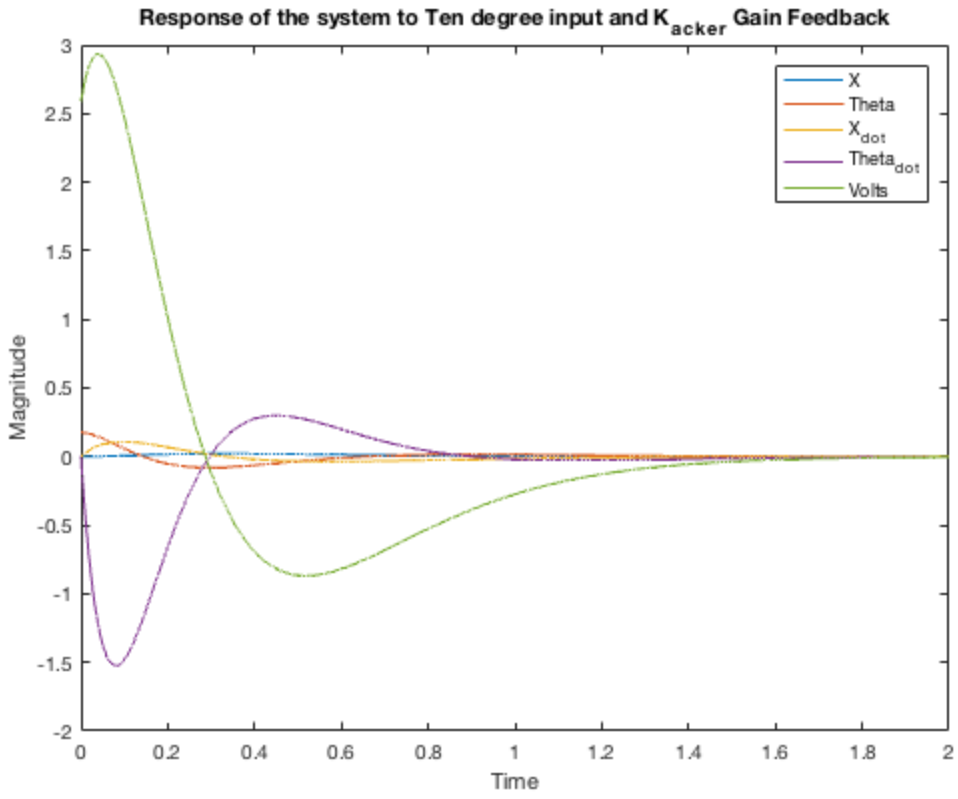
Excercise 2

|% Exercise two introduces two ways to move the system poles into a stable region. The first is Matlab's acker function. The function takes the syetem and finds the correct gain to push the system poles to those defined by user. in this section I chose the system poles to be at -5, -6,-7 and -8. By using the system constant's as provided by the professor the system poles work without saturating the controller which happens at values greater han 5 volts for the Arduino.

Use the place and acker functions|

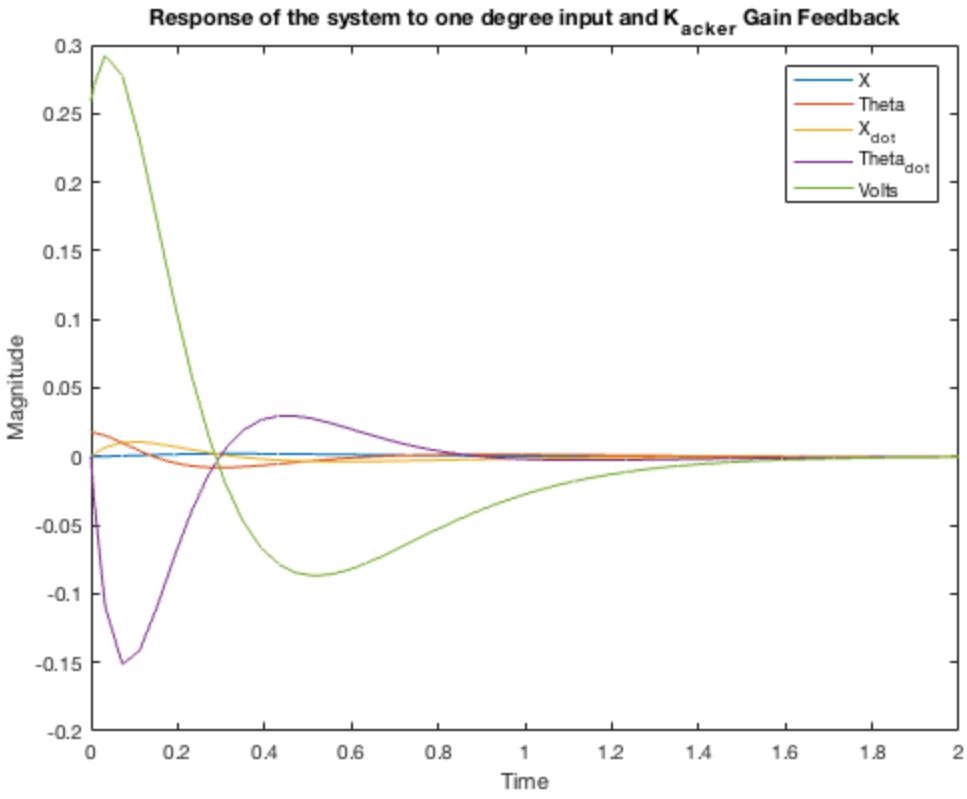
```
p = [-5 -6 -7 -8];
K_place = place(A,B,p);

K_acker = acker(A,B,p)

% both functions give the same gain.
initial = [0;(10*pi)/180;0;0];
sim('Project_2_2.slx');
figure(2);
plot(Time_2,Acker,Time_2,Input);
xlabel('Time');
ylabel('Magnitude');
title('Response of the system to Ten degree input and K_a_c_k_e_r Gain
 Feedback');
legend('X','Theta','X_d_o_t','Theta_d_o_t','Volts');
```

```
K_acker =

  -23.6238  -14.8255  -37.7929   -1.7935
```

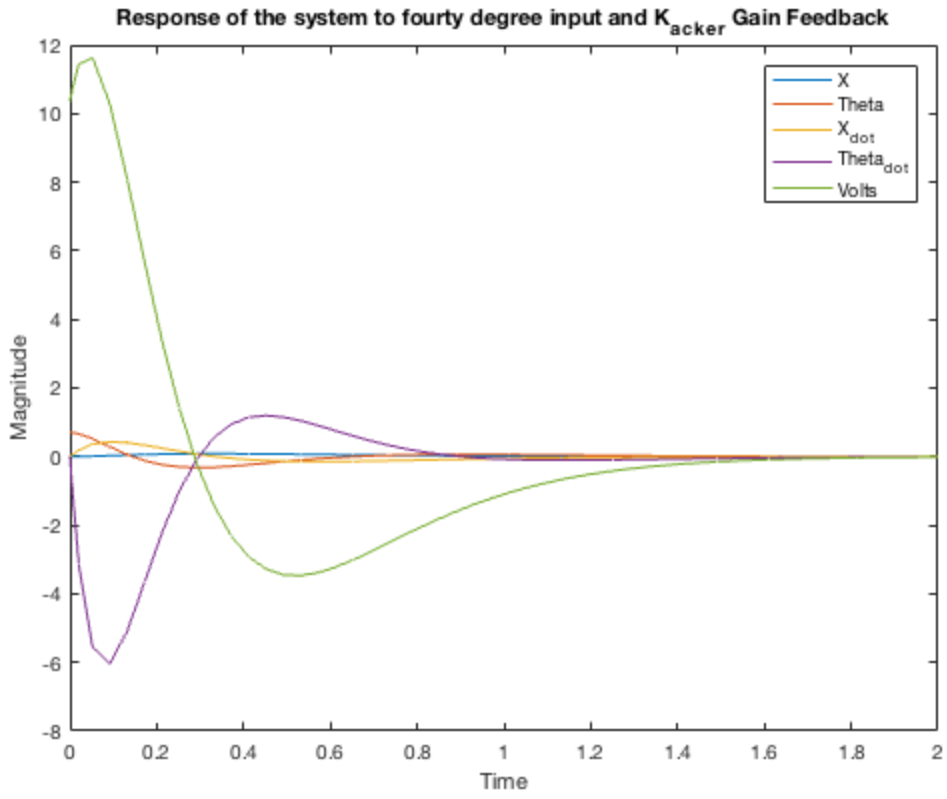


Evaluating different angle inputs in the initial conditions

```
initial = [0;(1*pi)/180;0;0];
sim('Project_2_2_1.slx');
figure(3);
plot(Time_2,Acker_1,Time_2,Input);
xlabel('Time');
ylabel('Magnitude');
title('Response of the system to one degree input and K_a_c_k_e_r Gain
 Feedback');
legend('X','Theta','X_d_o_t','Theta_d_o_t','Volts');
```



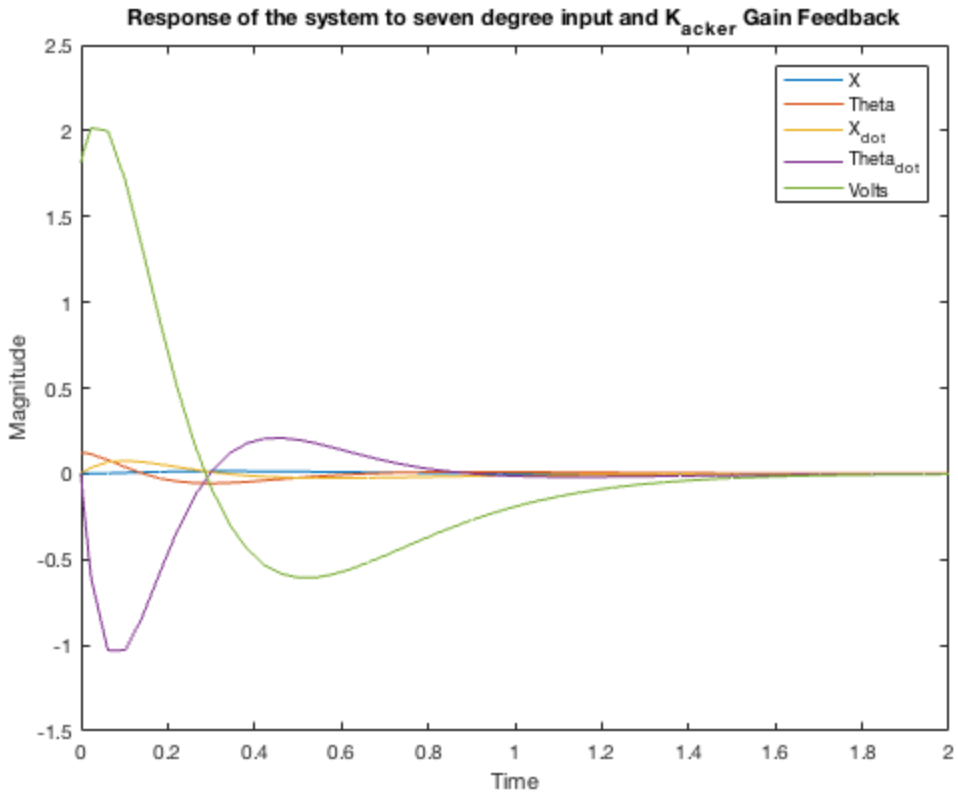

```
initial = [0;(40*pi)/180;0;0];
sim('Project_2_2_2.slx');
figure(4);
plot(Time_2,Acker_2,Time_2,Input);
xlabel('Time');
ylabel('Magnitude');
title('Response of the system to fourty degree input and K_a_c_k_e_r
 Gain Feedback');
legend('X','Theta','X_d_o_t','Theta_d_o_t','Volts');
```




```
% For fun I decided to input a large angle value. We can see that
this saturates the controller and so it should be suspected that our
controller will only be robust over a small angle range. This is both
due to the maximum power that can be provided by the MinSeg and the
simplification of the equations (but almost entirely the motor power).
```

And one last angle close to ten degrees.

```
initial = [0;(7*pi)/180;0;0];
sim('Project_2_2_3.slx');
figure(5);
plot(Time_2,Acker_3,Time_2,Input);
xlabel('Time');
ylabel('Magnitude');
title('Response of the system to seven degree input and K_a_c_k_e_r
 Gain Feedback');
legend('X','Theta','X_d_o_t','Theta_d_o_t','Volts');
```



Section 3: Linear Quadratic Regulator

|% The second controller type provided through the Matlab functions is the Linear Quadratic Regulator. The Linear-quadratic regulator design for state space systems:

[K,S,E] = lqr(SYS,Q,R,N) calculates the optimal gain matrix K such that:

- For a continuous-time state-space model SYS, the state-feedback law u = -Kx minimizes the cost function

J = Integral {x'Qx + u'Ru + 2*x'Nu} dt

subject to the system dynamics dx/dt = Ax + Bu

- For a discrete-time state-space model SYS, u[n] = -Kx[n] minimizes

J = Sum {x'Qx + u'Ru + 2*x'Nu}

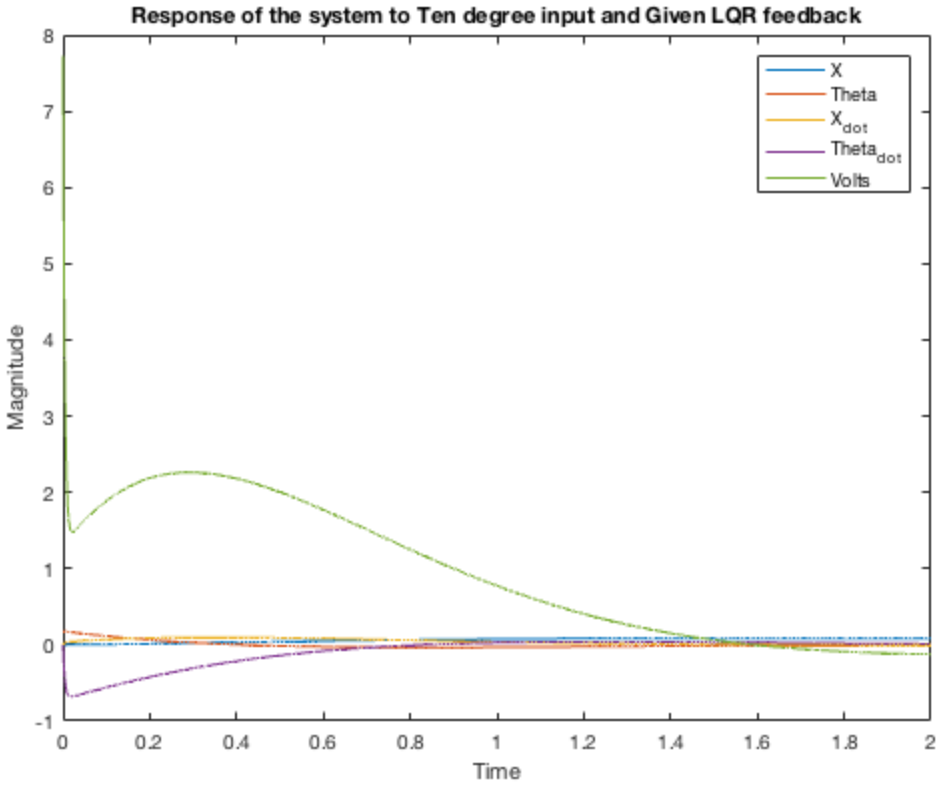
subject to x[n+1] = Ax[n] + Bu[n].

The matrix N is set to zero when omitted. Also returned are the the solution S of the associated algebraic Riccati equation and the closed-loop eigenvalues E = EIG(A-B*K). Q and R represent the weight matrices. Q is a 4X4 matrix and R is a scalar. If R is larger than Q the response will be slower but the effort of the controller will be less, and vice-versa.|

```
Q = [1 0 0 0;0 100 0 0; 0 0 1 0; 0 0 0 100] % as given by Dr. Bashash
```

```
R = 1;
N = 0;
[Klqr,S,e] = lqr(A,B,Q,R,N);
K_acker = Klqr; % for use in the simulink model
Klqr

initial = [0;(10*pi)/180;0;0];
sim('Project_2_2.slx');
figure(6);
plot(Time_2,Acker,Time_2,Input);
xlabel('Time');
ylabel('Magnitude');
title('Response of the system to Ten degree input and Given LQR
 feedback');
legend('X','Theta','X_d_o_t','Theta_d_o_t','Volts');
```

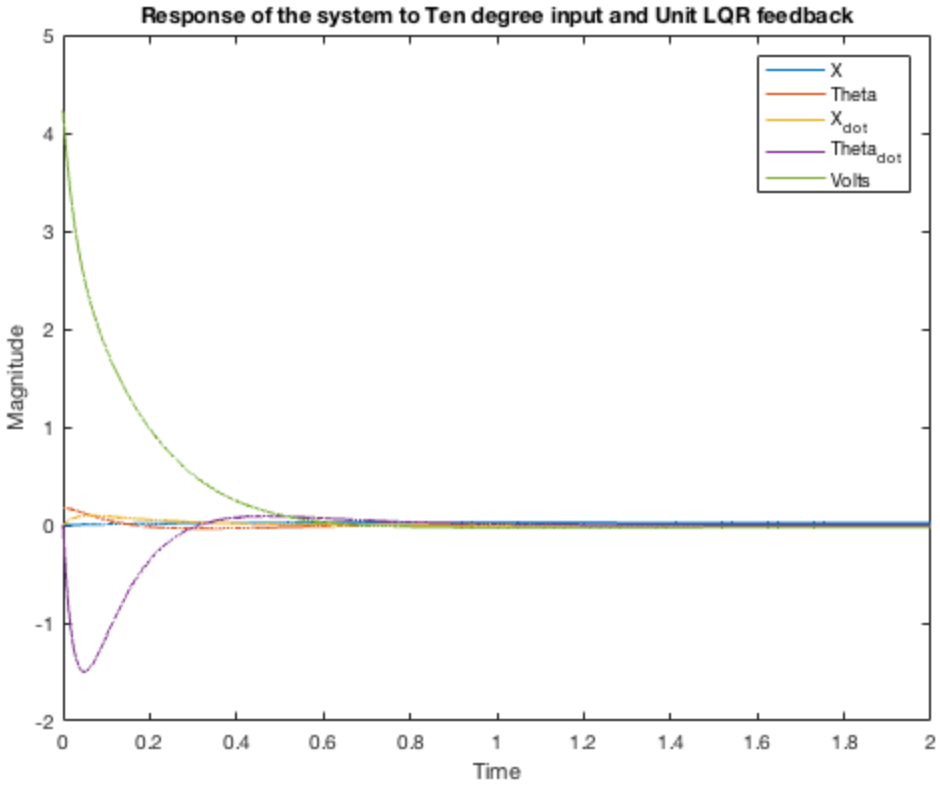
```
Q =

     1     0     0     0
     0   100     0     0
     0     0     1     0
     0     0     0   100


Klqr =

   -1.0000  -44.2547  -46.2267  -11.3505
```

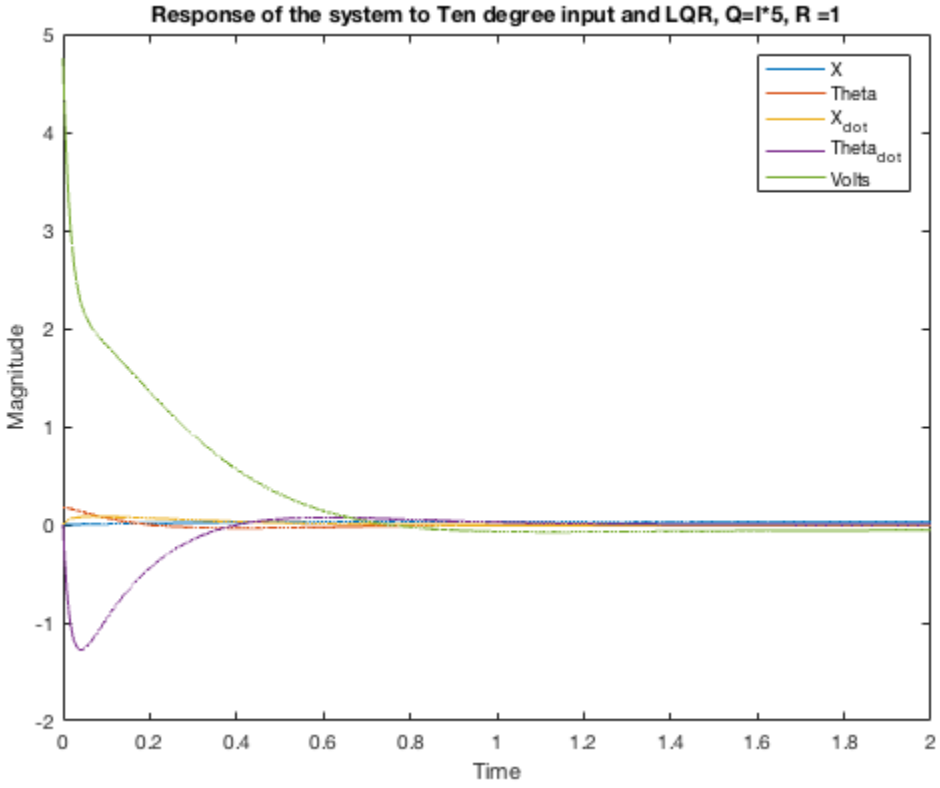


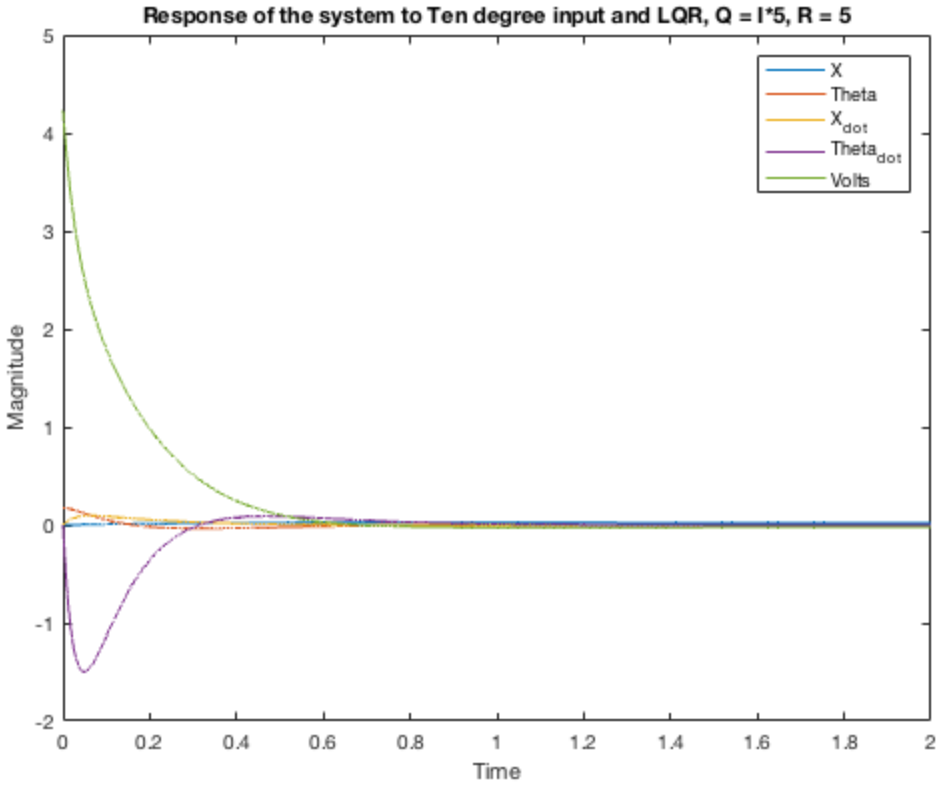
Experimenting with LQR

```
Q = [1 0 0 0;0 1 0 0; 0 0 1 0; 0 0 0 1];
R = 1;
N = 0;
[Klqr,S,e] = lqr(A,B,Q,R,N);
K_acker = Klqr; % for use in the simulink model


initial = [0;(10*pi)/180;0;0];
sim('Project_2_2.slx');
figure(7);
plot(Time_2,Acker,Time_2,Input);
xlabel('Time');
ylabel('Magnitude');
title('Response of the system to Ten degree input and Unit LQR
 feedback');
legend('X','Theta','X_d_o_t','Theta_d_o_t','Volts');

Q = [5 0 0 0;0 5 0 0; 0 0 5 0; 0 0 0 5];
R = 1;
N = 0;
[Klqr,S,e] = lqr(A,B,Q,R,N);
K_acker = Klqr; % for use in the simulink model
```

```
initial = [0;(10*pi)/180;0;0];
sim('Project_2_2.slx');
figure(8);
plot(Time_2,Acker,Time_2,Input);
xlabel('Time');
ylabel('Magnitude');
title('Response of the system to Ten degree input and LQR, Q=I*5, R
 =1');
legend('X','Theta','X_d_o_t','Theta_d_o_t','Volts');

Q = [5 0 0 0;0 5 0 0; 0 0 5 0; 0 0 0 5];
R = 5;
N = 0;
[Klqr,S,e] = lqr(A,B,Q,R,N);
K_acker = Klqr; % for use in the simulink model


initial = [0;(10*pi)/180;0;0];
sim('Project_2_2.slx');
figure(9);
plot(Time_2,Acker,Time_2,Input);
xlabel('Time');
ylabel('Magnitude');
title('Response of the system to Ten degree input and LQR, Q = I*5, R
 = 5');
legend('X','Theta','X_d_o_t','Theta_d_o_t','Volts');
```


Response of the system to Ten degree input and Unit LQR feedback
X
Theta
X_dot
Theta_dot
Volts
Magnitude
Time
5
4
3
2
1
0
-1
-2
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2

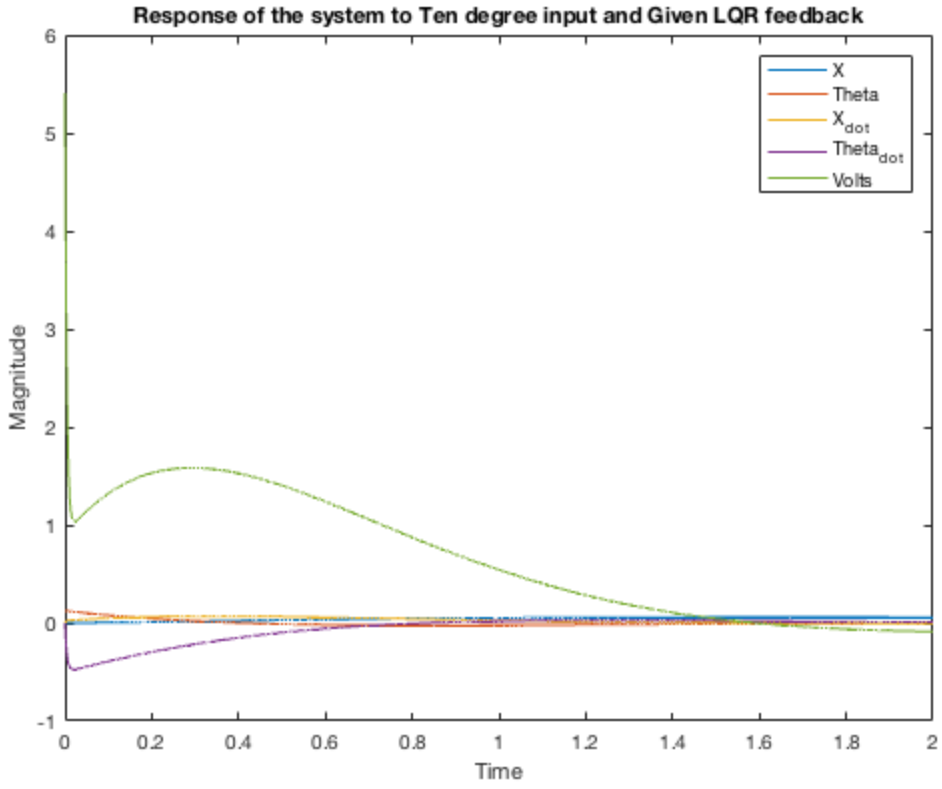

Response of the system to Ten degree input and LQR, Q=I*5, R =1
X
Theta
X_dot
Theta_dot
Volts
Magnitude
Time
5
4
3
2
1
0
-1
-2
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2

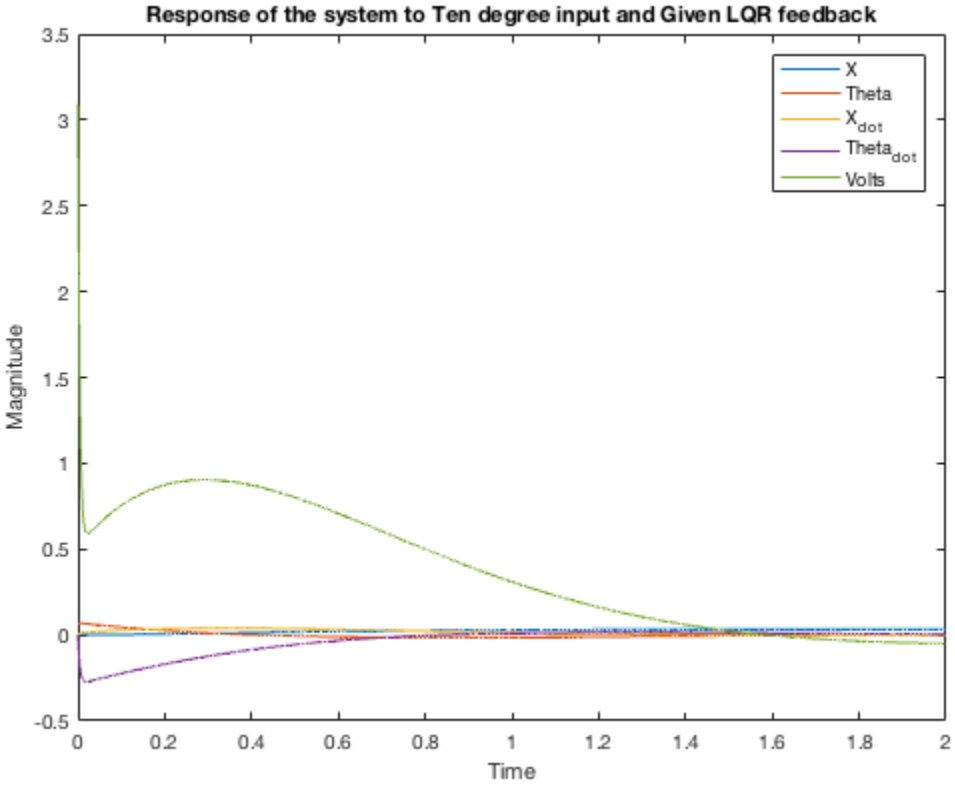


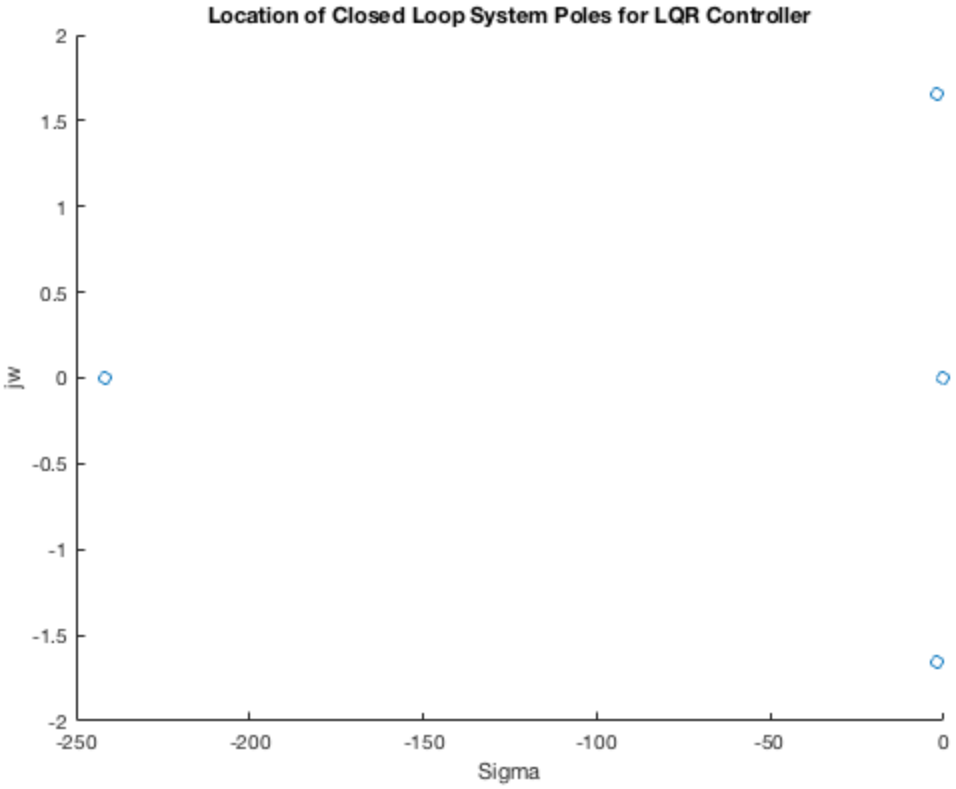
LQR Desirable Balance

```
% Using the desirable values of the LQR the controller was tested for
small angles. As the graphs indicate the controller does stabilize the
MinSeg.

Q = [1 0 0 0;0 100 0 0; 0 0 1 0; 0 0 0 100]; % as given by Dr. Bashash
R = 1;
N = 0;
[Klqr,S,e] = lqr(A,B,Q,R,N);
K_acker = Klqr % for use in the simulink model


initial = [0;(7*pi)/180;0;0];
sim('Project_2_2.slx');
figure(10);
plot(Time_2,Acker,Time_2,Input);
xlabel('Time');
ylabel('Magnitude');
title('Response of the system to Ten degree input and Given LQR
 feedback');
legend('X','Theta','X_d_o_t','Theta_d_o_t','Volts');

Q = [1 0 0 0;0 100 0 0; 0 0 1 0; 0 0 0 100]; % as given by Dr. Bashash
R = 1;
N = 0;
```

```
[Klqr,S,e] = lqr(A,B,Q,R,N);
K_acker = Klqr % for use in the simulink model


initial = [0;(4*pi)/180;0;0];
sim('Project_2_2.slx');
figure(11);
plot(Time_2,Acker,Time_2,Input);
xlabel('Time');
ylabel('Magnitude');
title('Response of the system to Ten degree input and Given LQR
 feedback');
legend('X','Theta','X_d_o_t','Theta_d_o_t','Volts');

figure(12);
sys_eig = eig(A-(B*Klqr));
scatter(sys_eig,[0;0;1.6502;-1.6502]);
title('Location of Closed Loop System Poles for LQR Controller');
xlabel('Sigma');
ylabel('jw');


K_acker =

   -1.0000  -44.2547  -46.2267  -11.3505


K_acker =

   -1.0000  -44.2547  -46.2267  -11.3505

Warning: Using only the real component of complex data.
```


Response of the system to Ten degree input and Given LQR feedback
X
Theta
X_dot
Theta_dot
Volts
Magnitude
Time
6
5
4
3
2
1
0
-1
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2


Response of the system to Ten degree input and Given LQR feedback
X
Theta
X_dot
Theta_dot
Volts
Magnitude
Time
3.5
3
2.5
2
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2



Part 4: Implementation of the feedback controller

```
Friction_bias = 45;
Gyro_bias = -270;
```

|% Accelerometer position constant was equal to pi/2 + 0.05.

Using the above constants in the simulink model the MinSeg was able to balance completely for a long period of time, and hopefully indefinitely if it was left alone and powered. The simulink model used to accomplish this is left in the appendix. For this model our gain was calculated by the LQR function using the values suggested by Dr. Bashash.|

```
Q = [1 0 0 0;0 100 0 0; 0 0 1 0; 0 0 0 100]
R = 1;
```

```
Q =

     1     0     0     0
     0   100     0     0
     0     0     1     0
     0     0     0   100
```

|% Unfortunately I did not save my workspace variables from the lab, so i cannot show the systems performance, but the graph would have had a voltage input oscillating around zero as the motor constantly changed drive direction. Theta dot would also be alternating with a large magnitude, while theta would have had a smaller magnitude and X would have had a smallermagnitude still. This makes sense because the controller would settle around a position and stay there. Interestingly my MinSeg would not center about the origin even when I tried really hard to change its position.

Initially I used the voltage control that is predefined by the MinSeg library and I only changed the output to the PWM blocks as defined int he Lab manual and the constant of 255 to 0. The original set up had an extra absolute value function before the saturation signal and the effect that it produced was a controller that succesfully balanced the MinSeg but did not react strongly to even large changes in angle. It was not robust, but it did make an eerily calm standing MinSeg compared to the others.

I then changed the complimentary filter to exclusively use the accelerometer and found the system lacking in responsiveness. The MinSeg would tilt to far before a reaction allowing for larger angles to generate and ultimately a response that would take the MinSeg too far the other direction. Here we see that the speed of response stops the system from becoming unstable.

When the weights for X and Theta were the same, the controller was somewhat stable, but would not stay up indefinitely. It may have been mostly stable due to the fact that the position does not often change very much.|

Conclusion

```
% This project was a great learning experience and shows the application
from a system equation to a computed system that helps to deliver the
desired output. This project helped in many ways especially since my
senior project will leverage the same types of sensors to hopefully
stabilize an underwter drone and maintain a constant altitude underwater.
In general this course has been a great introduction to the Matlab
software which I want to become more proficient because it is very
useful for data aquisition and system modeling. From the standpoint of
understanding the course material, I can apply the State-space method
but I think I need to review my linear algebra to get a better grasp
on its working.
```

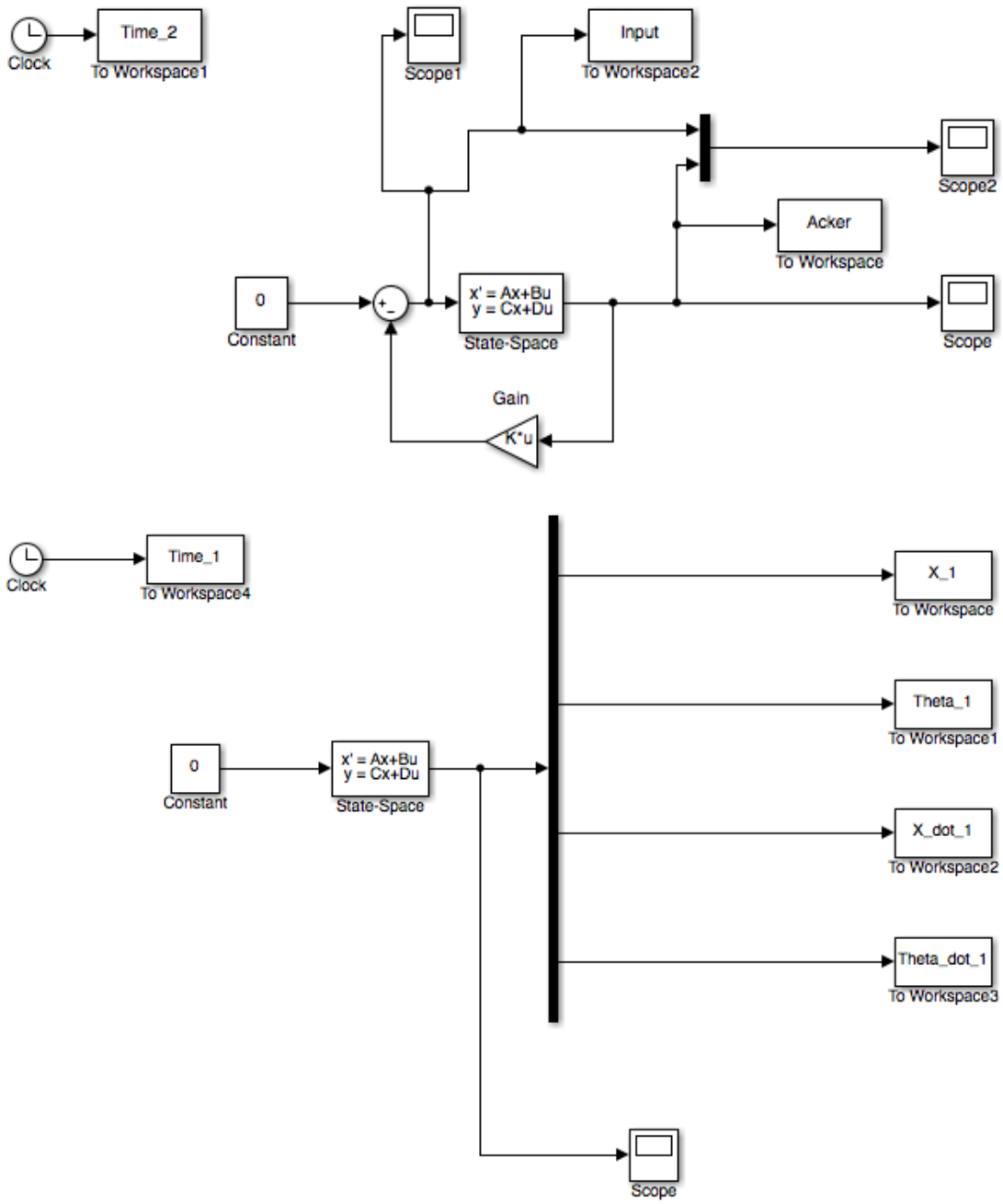
```
% Controller tuning seems to be a very dark art in the sense that even
when a system is well modeled a lot of tuning must still be performed
to get the controller to behave in the ways that are wanted. This is
why much more time would be need to make the controller more robust.
While modeling the system does get you much of the way there, the actual
physical system has many more variables. The friction constant is a
good example. The friction can change due to the surface as well as the
temperature of the materials. Also the consistency of the construction
of the MinSeg robots comes into question. Not all models have the same
center of gravity, the resistance of the motors will change a little
bit, and some sensors may have larger biases than others or may not
be calibrated properly.
```

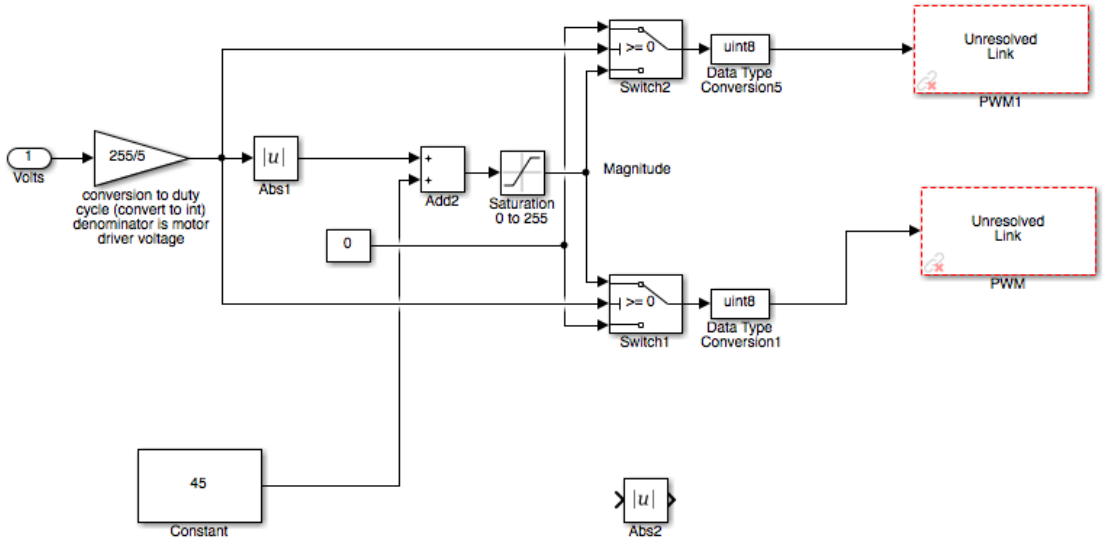
Appendix

```
open_system('Project_1_8.slx');
```

```
open_system('Project_2_2.slx');
open_system('part4_Final_Project.slx');
open_system('part4_Final_Project/Subsystem');
open_system('part4_Final_Project/Subsystem1');
```

```
Warning: Unable to load block diagram 'arduinolib'
```





Published with MATLAB® R2016b