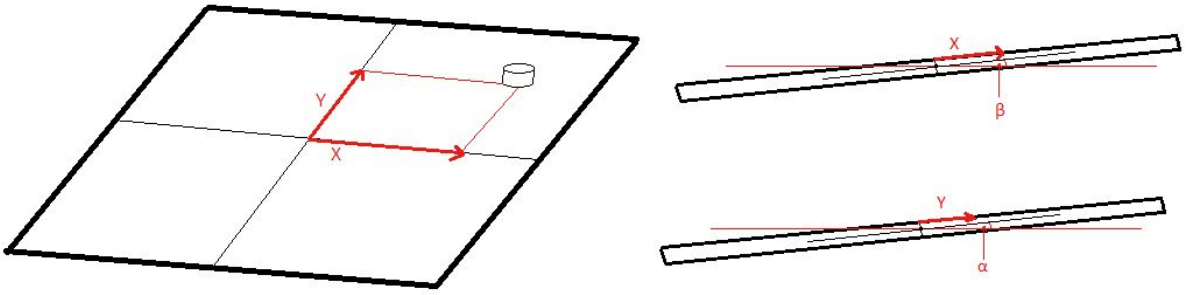
MSE 483: Slider and Plate State Space Control System

Marko Jovanovic
301306143
Group #18

Introduction

The goal of this project is to design a control system for a simulated slider moving across a simulated tilt plate. The tilt plate will be torque-actuated, and a continuous-domain control system will be created. The position X and Y axes will be along the tilt plate, and as such will require the consideration of illusory forces that occur when in a non-inertial reference frame, such as euler and coriolis forces.



Labelling of Axes and Angles of Simulated Slider-Plate System

The design objectives of this project will be to be able to control the path the slider takes, and compare the performance of a few different control schemes.

System Identification and Analysis

The dynamics equations were formulated by calculating the net forces, and net torques acting on the system, in which

$$F_{puck} = F_{gravity} + F_{centrifugal} + F_{loss}$$

Since the system could be decoupled in the X and Y directions along the plate, we can formulate our systems separately for each direction.

Eq(1)

$$m\ddot{x} = -mgsin(\beta) - m(\dot{\beta}^2)x - K_F\dot{x}$$
$$m\ddot{y} = -mgsin(\alpha) - m(\dot{\alpha}^2)y - K_F\dot{y}$$

In which m is the mass of the puck being moved, and K_F is a loss coefficient.

Next, we formulate the moments equation about a given tilt axis.

$$M_{system} = M_{motor} + M_{gravity} + M_{coriolis} + M_{loss}$$

As before, we put this into specific terms:

Eq (2)

$$(J_{plate-y} + mx^2)\ddot{\beta} = T_y - mgxcos(\beta) - 2m\dot{x}\dot{\beta}x - K_{py}\dot{\beta}$$
$$(J_{plate-x} + my^2)\ddot{\alpha} = T_x - mgycos(\alpha) - 2m\dot{y}\dot{\alpha}y - K_{px}\dot{\alpha}$$

In which J_{plate} is the mass moment of inertia about the center axis, and K_p is the rotational motor losses. The moment of inertia of the slider was assumed to be negligible.

If we rearrange eq 1 and 2, and linearize about the origin, with an operating point of (in the case of the x direction), $x = 0,\ \dot{x} = 0,\ \beta = 0, \dot{\beta} = 0$, and using sin(β) = β we can reformulate our system equations as follows:

$$\ddot{x} = -g\beta - \frac{K_F}{m}\dot{x}$$
$$\ddot{y} = -g\alpha - \frac{K_F}{m}\dot{y}$$
$$\ddot{\beta} = \frac{1}{J_{plate-y}}T_y - \frac{mg}{J_{plate-y}}x - \frac{K_{py}}{J_{plate-y}}\dot{\beta}$$
$$\ddot{\alpha} = \frac{1}{J_{plate-x}}T_x - \frac{mg}{J_{plate-x}}y - \frac{K_{px}}{J_{plate-x}}\dot{\alpha}$$

Table 1: Dynamics Values

Mass of slider	m	0.1 kg
Sliding loss coefficient	K_F	0.1 N/(m/s)
Rotational Loss Coefficient	K_{px}, K_{py}	0.1 Nm/(rad/s)
Plate Mass Moment of Inertia	$J_{plate-x}$, $J_{plate-y}$	0.33 kg*m^2
Gravitational Acceleration	g	9.81 m/s^2

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\beta} \\ \ddot{\beta} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & \frac{-Kf}{m} & -g & 0 \\ 0 & 0 & 0 & 1 \\ \frac{-mg}{Jplate-y} & 0 & 0 & \frac{-Kpy}{Jplate-y} \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \beta \\ \dot{\beta} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ \frac{1}{Jplate-y} \end{bmatrix} Ty$$

$$\begin{bmatrix} \dot{y} \\ \ddot{y} \\ \dot{\alpha} \\ \ddot{\alpha} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & \frac{-Kf}{m} & -g & 0 \\ 0 & 0 & 0 & 1 \\ \frac{-mg}{Jplate-x} & 0 & 0 & \frac{-Kpx}{Jplate-x} \end{bmatrix} \begin{bmatrix} y \\ \dot{y} \\ \alpha \\ \dot{\alpha} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ \frac{1}{Jplate-x} \end{bmatrix} Tx$$

With the values of table 1, our system matrices are:

A B C

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -1 & -9.81 & 0 \\ 0 & 0 & 0 & 1 \\ -\frac{0.981}{0.33} & 0 & 0 & -\frac{0.1}{0.33} \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ \frac{1}{0.33} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}$$

With matrix D being zero.

Controllability and Observability

Using matlab, we can determine the controllability, observability, and open-loop stability of our system.

First, we check the controllability of our system.

```
>> rank(ctrb(A,B))

ans =

     4
```

Our system is fully controllable.

Next, we check observability,

```
>> rank(obsv(A,C))

ans =

     4
```

Our system is fully observable.

Note: Because of duality, we could also use

```
>> rank(ctrb(A',C'))

ans =

     4
```

Which is an equivalent statement to that of the observability.

Finally, we check the eigenvalues of our open-loop zero input system

```
>> eig(A)

ans =

   2.0306 + 0.0000i
  -0.3221 + 2.2883i
  -0.3221 - 2.2883i
  -2.6895 + 0.0000i
```

This result, with a positive real root, tells us that our system is unstable, and will require a controller in order to stabilize it. Luckily, our system is both controllable and observable.

Controller and Observer Design

Our controller design will be taking the form of

$$u = -K(\hat{x} - x_d)$$

In

$$\dot{x} = Ax + Bu$$
$$y = Cx$$

Where x_d is the desired state location, and $\hat{x}$ is the current state, estimated by our observer. K is a feedback gain matrix.

Due to the separation principle, we can choose the eigenvalues of our A-BK matrix, and our A-LC matrix independently. First, we will choose our eigenvalues through some trial and error, to be located at s = -4, -5, -6, -7. Eigenvalues with no imaginary part were chosen so as to minimize overshoot and oscillation. K values were found using the K = place(A,B,[-4,-5,-6,-7]) matlab function, and are listed below.

```
K =

  -29.2657  -16.1633   52.1400    6.8300
```

To confirm our results, we test the eigenvalues of the full state feedback system.

```
eig(A-B*K)

ans =

   -7.0000
   -6.0000
   -5.0000
   -4.0000
```

The results are good.

Next, we'll look at an observer. For our A-LC eigenvalues, we chose values 10 times that of our controller, so as to have a much faster observer response than controller. We used the same process, except using L = place(A',C',[-40,-50,-60,-70])' in matlab (the transposes of the matrices were used in this case). Again, we tested the eigenvalues. Note that the eigenvalues of a matrix are the same, regardless of if it was transposed or not.

```
eig(A-L*C)

ans =

  -70.0000
  -60.0000
  -50.0000
  -40.0000
```

For the actual L values, due to the high coefficient values (1.0e+05), small changes in lower-order sig-figs had a large impact on the performance of the system. Long decimal format was used to get the non-truncated L values for more accuracy. The long L values were the ones used in our observer, since they provided better results.

```
L =

  1.0e+05 *

   0.0022
   0.1761
  -0.6455
  -8.3759
```

```
L =

  1.0e+05 *

   0.002186969696867
   0.176147281894239
  -0.645506052740793
  -8.375850372119285
```

Lastly, we were also able to design K gains based on an LQR controller. Our Q and R gains (listed below) were chosen such that actuation was cheap, and we wanted to penalize the position and velocity of our state matrix more heavily than the rest of the system, with position being penalized the most in our error function.

```
R =

    0.1000

Q =

   100     0     0     0
     0    10     0     0
     0     0     1     0
     0     0     0     1
```

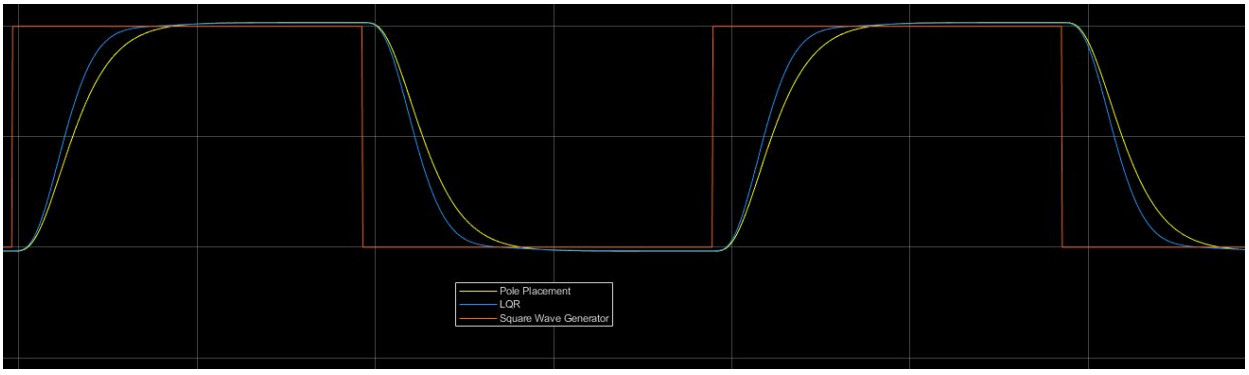
The resulting K gains from the lqr(A,B,Q,R) command in matlab were:

```
K =

  -32.6180  -16.0550   43.0520    6.0987
```

With feedback eigenvalues of

```
ans =

  -9.9829 + 0.0000i
  -3.3718 + 4.4060i
  -3.3718 - 4.4060i
  -3.0575 + 0.0000i
```

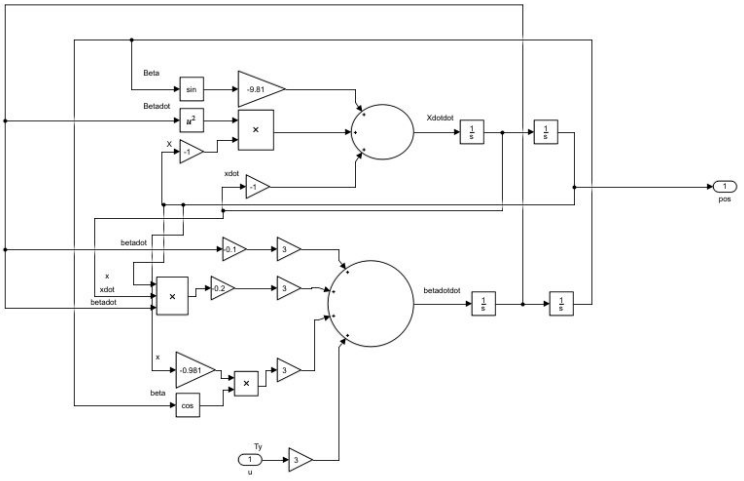
To compare the performance of both our pole placement and LQR controllers, they were tested on our system with a square wave input, with the observer being our lunenburg observer with the L gains calculated from before. Below, we have the step response of our Pole Placement controller (yellow), and that of our LQR controller (blue). We can see that, although they have similar performance, the LQR slightly wins out in rise time. For this reason, the LQR controller gains were used for the rest of our simulations.



Comparison of LQR and Pole Placement Controller on Square Wave Input

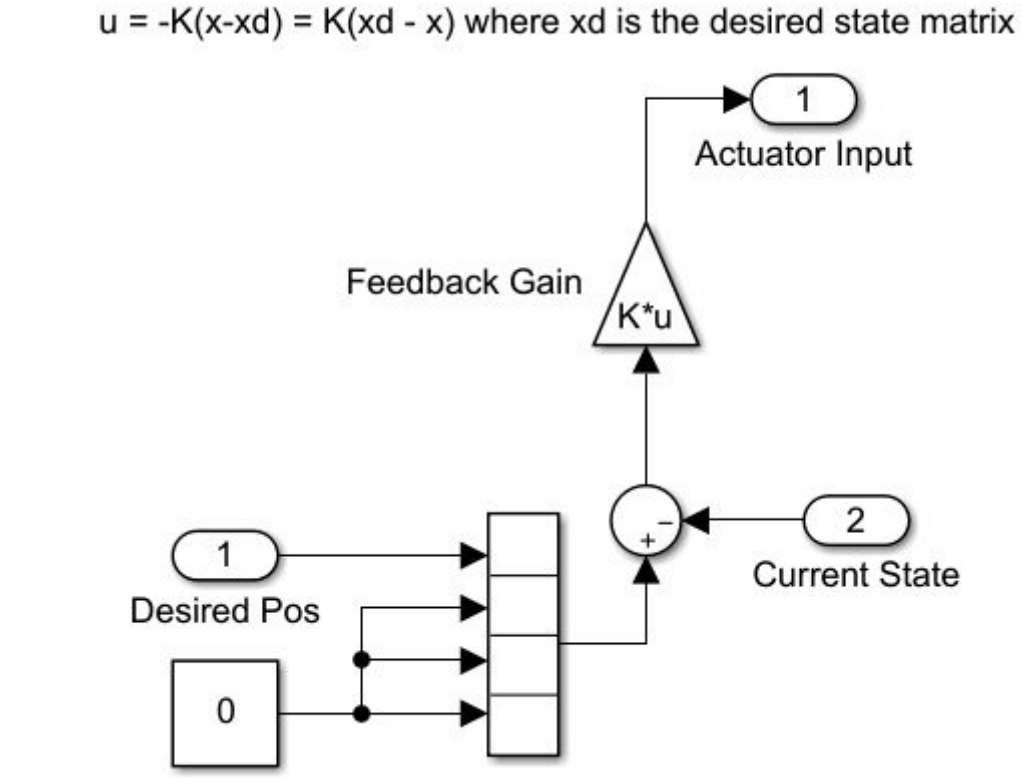
Simulation Results

For our simulation, a continuous block diagram form of our system equations (eq 1 and eq 2) were used as our system plant, decoupled in the x and y directions, so for each Simulation, two plants were used.

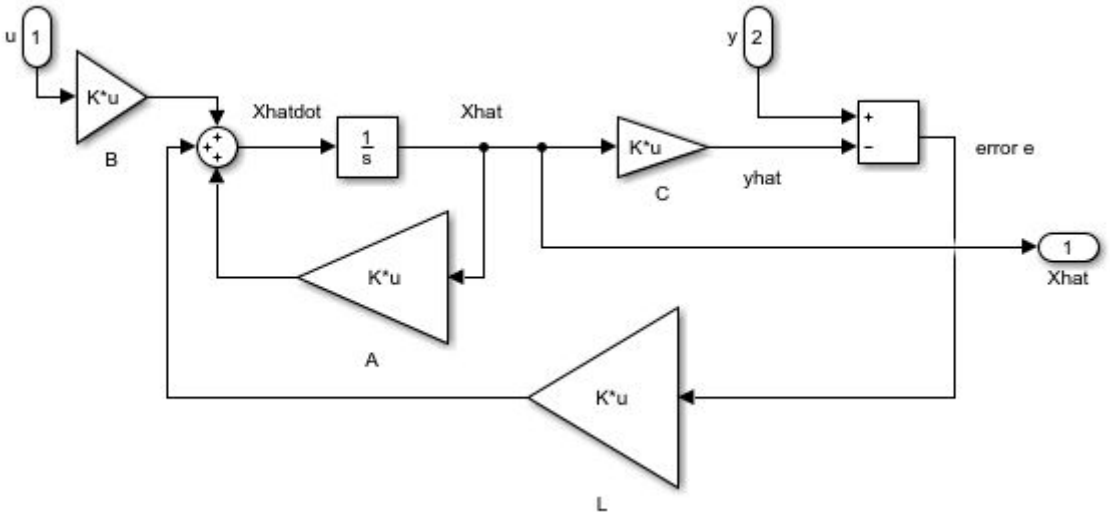


Block Diagram Representation of Nonlinear System Dynamics

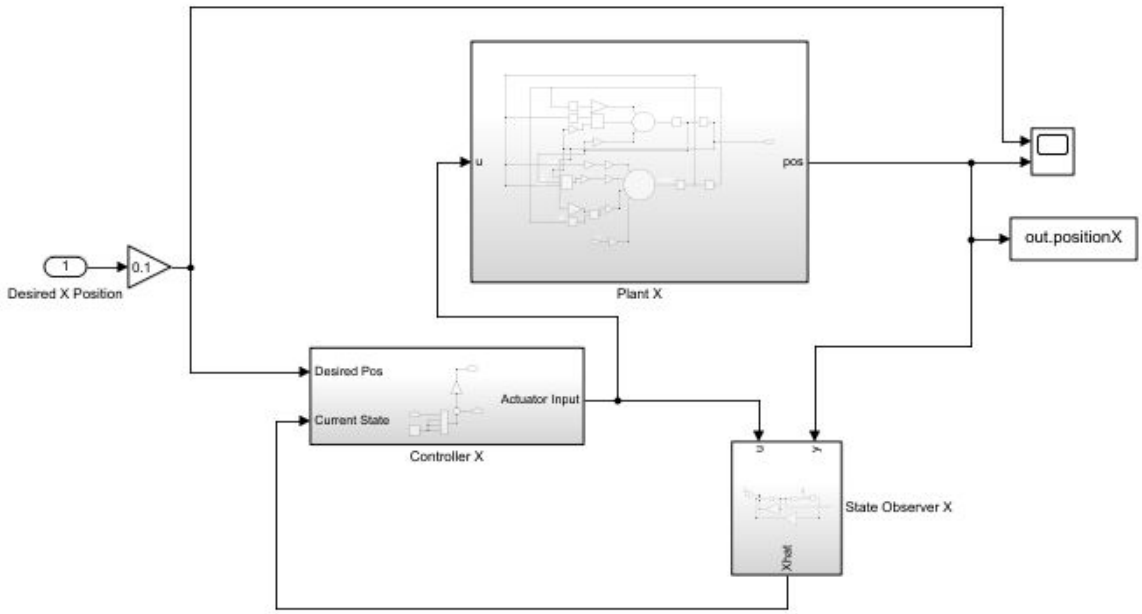
Our controller and observer were standard continuous time feedback controllers, in which the output ($\widehat{x}$) of our observer subsystem was the current state to our controller.



Full State Feedback Controller Block Diagram



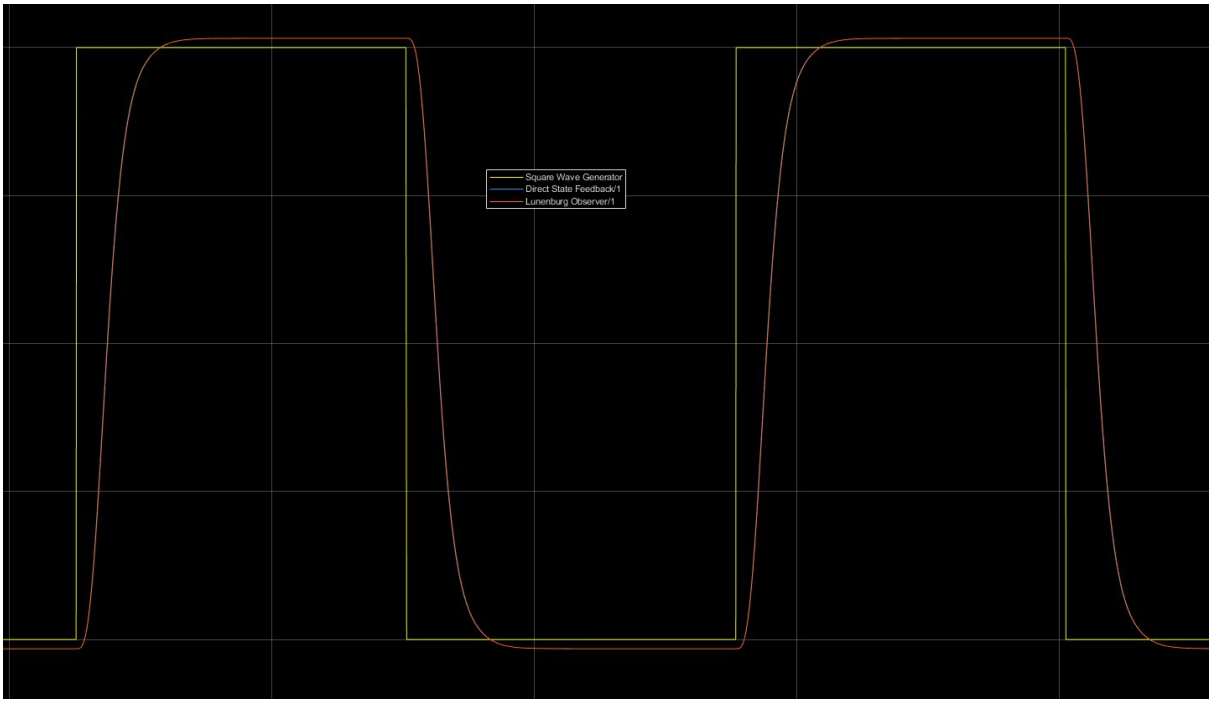
Lunenburg Observer Block Diagram



Fully realized “X direction” Combined System

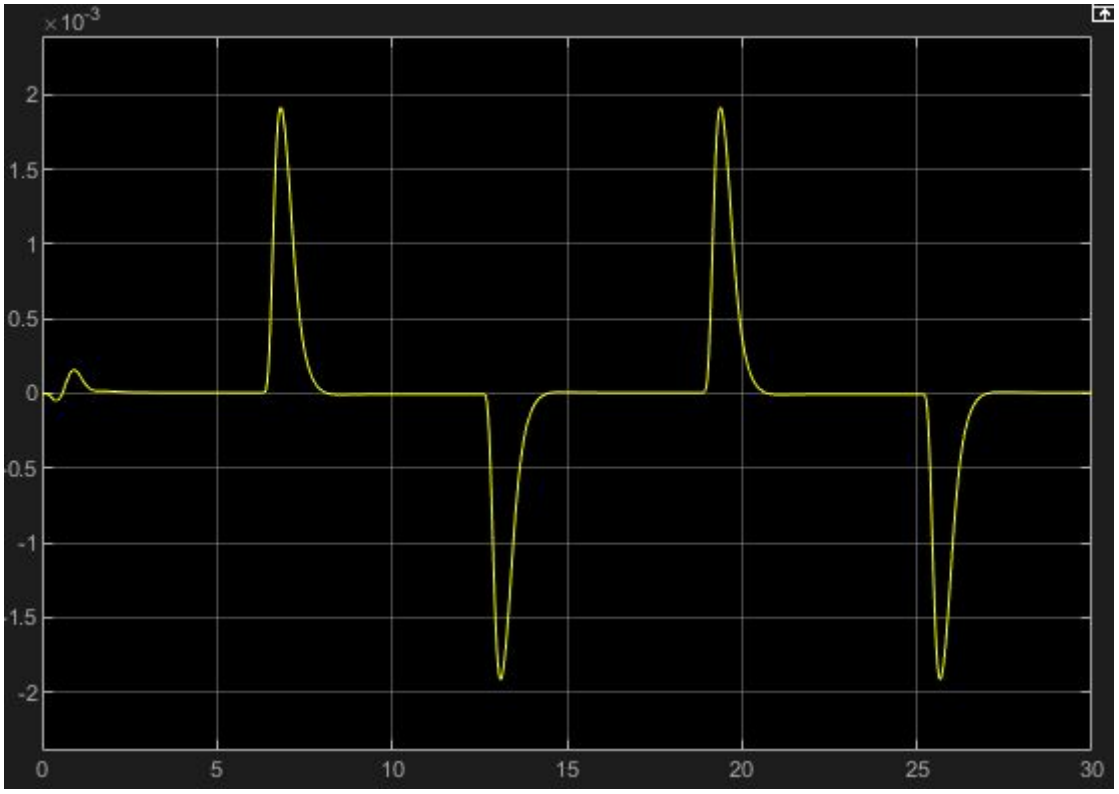
Note that our position input has a gain of 0.1, this is so that the position inputs remain relatively small (±10cm from the origin) so as to keep our operation around our linearization point. The X and Y positions were separately controlled using parametric equations as our inputs, parameterized to time.

Also, a comparison of having no observer with direct state feedback, versus using our lunenburg observer was done.



Comparison of No-observer vs Lunenburg Observer Full State Feedback Controller

As is seen in the graph above, there is almost no difference in performance between using our designed observer, and directly feeding the true states back into the controller.



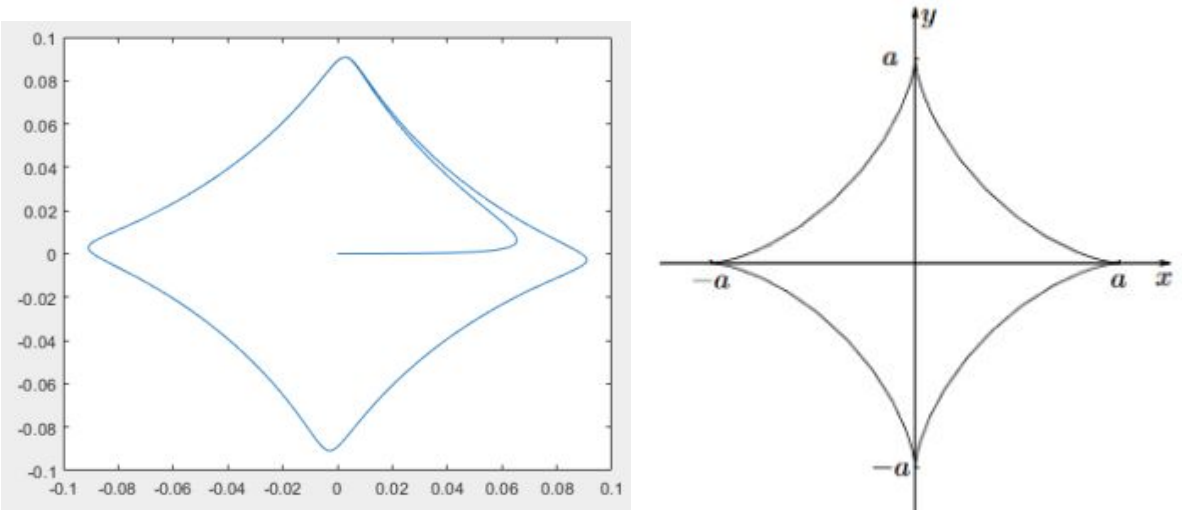
Difference Between Estimated Position and Actual Position

Note that the difference between the real position of the state, and our estimated position was at most 0.002m, and even then it quickly minimized to zero.

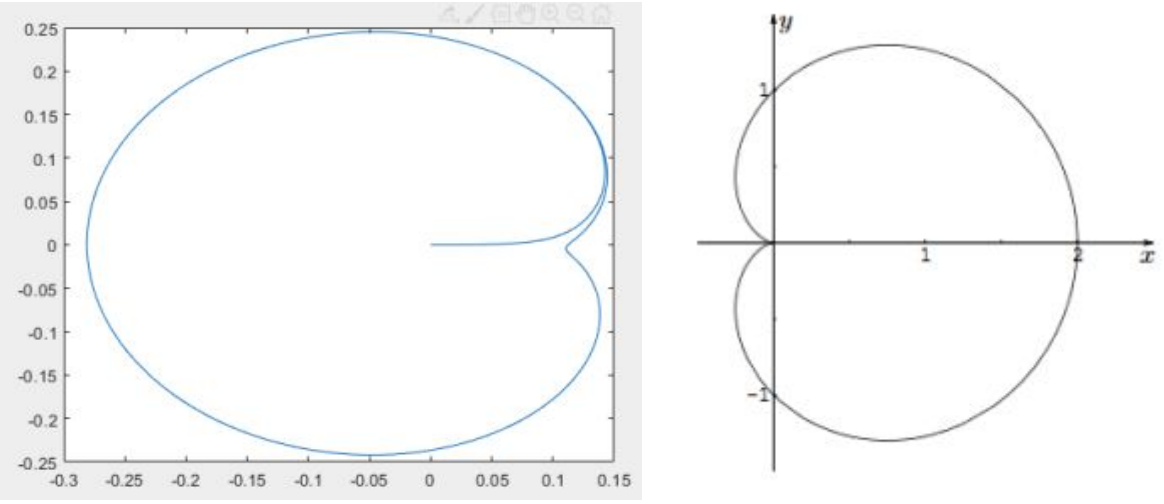
We tested several inputs, with equations and graph comparison taken from [2]. They were graphed using the matlab command:

```
>>  t = out.tout; Y = out.positionY; X = out.positionX; plot(X,Y)
```

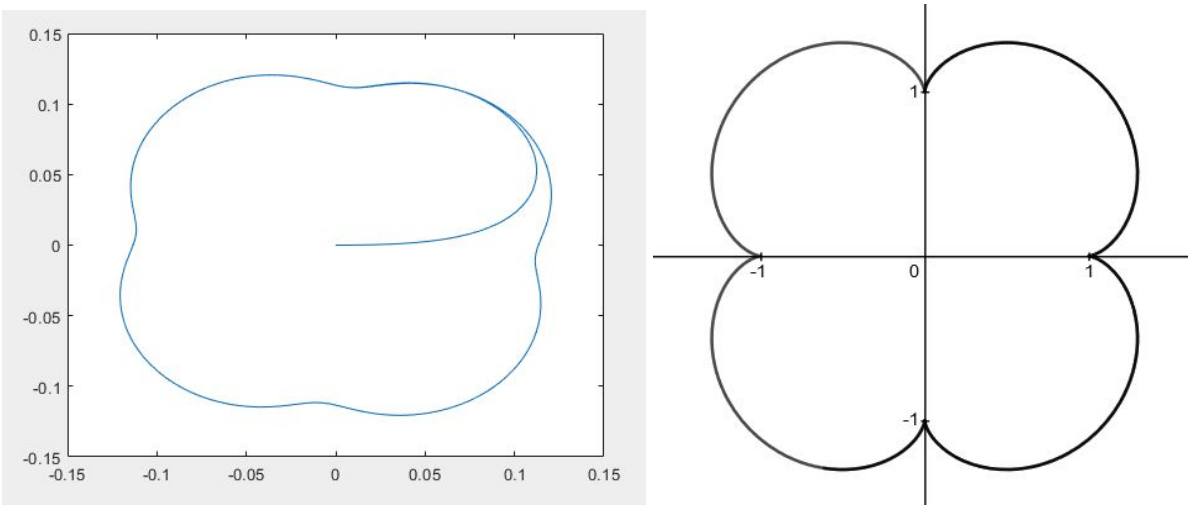
The simulated position graphs are shown below.



X-Y Path of Slider, Given Astroid Path



X-Y Path of Slider Given Cardioid Path



X-Y Path of Slider Given Epicycloid Path

Note that for some input curves, such as the Epicycloid path, the system cannot match the path perfectly, and would need further tuning and response time improvement in order to do so. All curves were plotted on a 10.0 second simulation time.

Conclusions

We were successfully able to control a simulated slider on plate system, by actuating the plate rotation axis torques, using both an LQR, and a pole-placement controller, as well as a standard Lunenburg observer.

Sources

[1] F. Dušek, D. Honc and K. R. Sharma, "Modelling of ball and plate system based on first principle model and optimal control," *2017 21st International Conference on Process Control (PC)*, Strbske Pleso, 2017, pp. 216-221.

[2] S. Kokoska, *Fifty Famous Curves, Lots of Calculus Questions, And a Few Answers*, Accessed on: April 4, 2020. [Online]. Available: https://elepa.files.wordpress.com/2013/11/fifty-famous-curves.pdf

[3] R. L. Williams II and D. A. Lawrence, *Linear State-Space Control Systems*. Hoboken NJ, simultaneously published in Canada: John Wiley & Sons Inc., 2007.