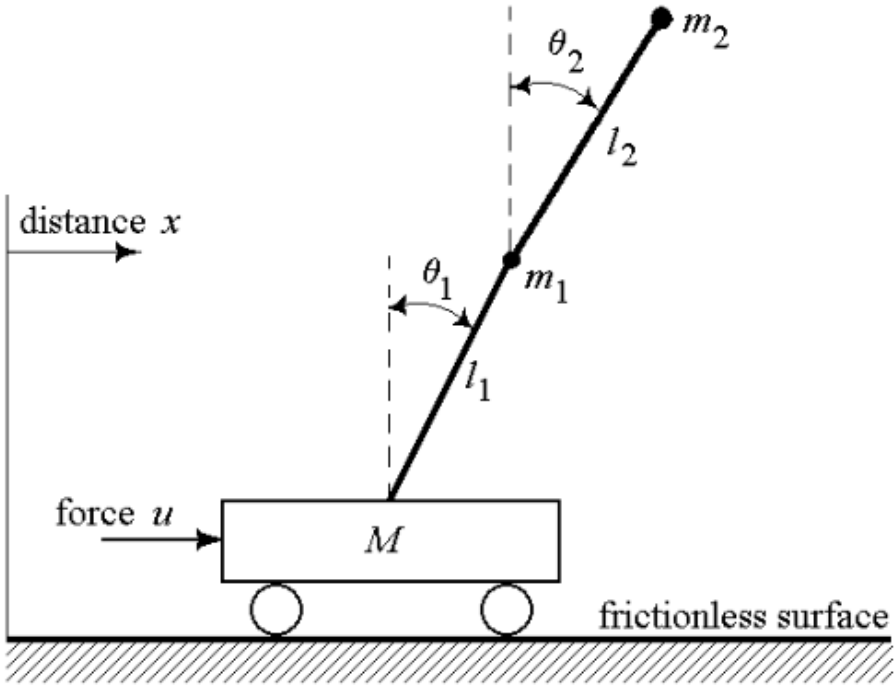
FunWork #4

Victoria Nagorski

Prelude

In Funwork #1 the problem statement was defined with the following diagram:



With the given properties of the system:

- $m_1 = 0.5kg$
- $l_1 = 0.5m$
- $m_2 = 0.75kg$
- $l_2 = 0.75m$
- $M = 1.5kg$
- $g = 9.81m/sec^2$

We were also given that the state variables follow the following definition: $x_1 = x$, $x_2 = \theta_1$, $x_3 = \theta_2$, $x_4 = \dot{x}$, $x_5 = \dot{\theta_1}$, $x_6 = \dot{\theta_2}$. These were substituted much later since it was easier to think in terms of x, θ_1, and θ_2.

With the problem statement, the following assumptions were made:

- The body is rigid
- The rods are mass-less
- We are working with a simplified point-mass system

The matrices that came from Funwork 3:

$$A = \begin{bmatrix} 0 & 0 & 0 & 1.0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1.0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1.0 \\ 0 & -0.534 & -1.126 & 0 & 0 & 0 \\ 0 & 22.505 & -17.804 & 0 & 0 & 0 \\ 0 & -13.988 & 21.776 & 0 & 0 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0.425 & -0.174 & -0.114 \\ -0.174 & 7.341 & -11.904 \\ -0.289 & -4.563 & 10.144 \end{bmatrix}$$

We are also going to define matrices C and D at this point. We only have 3 outputs, so we should only have 3 rows within C. D will be a zero matrix. In most instances (at least in aerospace engineering), D will be a zero matrix.

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

$$D = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

These were solved for using the following equilibrium points (for report towards page is positive, while in code out of page is positive):

$$x_e = \begin{bmatrix} 0.1 & 60^\circ & 45^\circ & 0 & 0 & 0 \end{bmatrix}^T$$

$$u_e = \begin{bmatrix} 0 & -9.2117 & -3.9019 \end{bmatrix}^T$$

That that the torques point outward from the paper, and hence are positive in the code. The system was tested to be both controllable and observable in the previous Funwork.

Discretize the System

For the first problem, we are asked to discretize the continuous system. To discretize a system, the following equations can be used:

$$\Phi = e^{Ah}$$

$$\Gamma = \int_0^h e^{A\eta} B d\eta$$

I will define the sampling frequency to be at 100 Hz since that tends to be the (aerospace) industry standard. The sampling interval is $h = 1/f$, which is 0.01 (will take a sample every hundredth of a second). Sampling rates, if slow enough, can cause a system that is stable to fall into instability. I wanted to choose an industry standard to represent how this technique might be used in real life. In general, 0.01 is a pretty fast sampling rate.
Matlab has a command 'c2d()' that will be used to discretize the model. We now have the following discrete system that will be used for the rest of the model (in code):

$$\Phi = \begin{bmatrix} 1.0 & 0 & 0 & 0.01 & 0 & 0 \\ 0 & 1.001 & -0.0009 & 0 & 0.01 & 0 \\ 0 & -0.0007 & 1.0012 & 0 & 0 & 0.01 \\ 0 & -0.0053 & -0.0113 & 1.0 & 0 & 0 \\ 0 & 0.2252 & -0.1782 & 0 & 1.0011 & -0.0009 \\ 0 & -0.1400 & 0.2179 & 0 & -0.0007 & 1.0011 \end{bmatrix}$$

$$\Gamma = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0.0004 & -0.0006 \\ 0 & -0.0002 & 0.0005 \\ 0.0043 & -0.0017 & -0.0011 \\ -0.0017 & 0.0735 & -0.119 \\ -0.0029 & -0.0457 & 0.1015 \end{bmatrix}$$

The C and D matrices do not change when discretized.

```
%% Problem 1
% Discretize the model
clear; clc; close all;                  % Close everything
load('Values_Rev3.mat')                 % Load variables

% Solve
sys_c = ss(A,B,C,D);                    % Create continous system
h = 0.01;                               % Define sampling interval
sys = c2d(sys_c,h);                     % Discritize system
```

Design a MPC w/o Constraints (Augmented)

The first step of this question is to derive the augmented Φ and Γ equations. We start with the following system:

$$x[k+1] = \Phi x[k] + \Gamma u[k]$$

$$y[k] = Cx[k]$$

The backwards difference operator, $\Delta x[k+1] = x[k+1] - x[k]$, can be used to re-write the equations:

$$\Delta x[k+1] = \Phi \Delta x[k] + \Gamma \Delta u[k]$$

$$\Delta y[k+1] = Cx[k+1] - Cx[k] \rightarrow \Delta y[k+1] = C\Phi \Delta x[k] + C\Gamma \Delta u[k]$$

$$y[k+1] = y[k] + C\Phi \Delta x[k] + C\Gamma \Delta u[k]$$

Combining the equations we get:

$$\begin{bmatrix} \Delta x[k+1] \\ y[k+1] \end{bmatrix} = \underbrace{\begin{bmatrix} \Phi & 0 \\ C\Phi & I_p \end{bmatrix}}_{\Phi_a} \underbrace{\begin{bmatrix} \Delta x[k] \\ y[k] \end{bmatrix}}_{x_a[k]} + \underbrace{\begin{bmatrix} \Gamma \\ C\Gamma \end{bmatrix}}_{\Gamma_a} \Delta u[k]$$

$$y[k] = \underbrace{\begin{bmatrix} 0 & I_p \end{bmatrix}}_{C_a} \begin{bmatrix} \Delta x[k] \\ y[k] \end{bmatrix}$$

The next part can be derived by expanding out these equations out for N_p steps ahead. (Plug in $x[1]$ into $x[2]$ and so on for example.) N_p represents our prediction horizon for which we are looking ahead to make a proper control decision. After we expand the equations out, we end up with the following matrices:

$$\begin{bmatrix} x_a[k+1|k] \\ x_a[k+2|k] \\ \vdots \\ x_a[k+N_p|k] \end{bmatrix} = \begin{bmatrix} \Phi_a \\ \Phi_a^2 \\ \vdots \\ \Phi_a^{N_p} \end{bmatrix} x_a[k] + \begin{bmatrix} \Gamma_a & 0 & \dots & 0 \\ \Phi_a\Gamma_a & \Gamma_a & \dots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ \Phi_a^{N_p-1}\Gamma_a & \dots & \dots & \Gamma_a \end{bmatrix} \begin{bmatrix} \Delta u[k] \\ \Delta u[k+1] \\ \vdots \\ \Delta u[k+N_p-1] \end{bmatrix}$$

$$\begin{bmatrix} y[k+1|k] \\ y[k+2|k] \\ \vdots \\ y[k+N_p|k] \end{bmatrix} = \begin{bmatrix} C_a x_a[k+1|k] \\ C_a x_a[k+2|k] \\ \vdots \\ C_a x_a[k+N_p|k] \end{bmatrix}$$

$$\underbrace{\begin{bmatrix} y[k+1|k] \\ y[k+2|k] \\ \vdots \\ y[k+N_p|k] \end{bmatrix}}_{Y} == \underbrace{\begin{bmatrix} C_a\Phi_a \\ C_a\Phi_a^2 \\ \vdots \\ C_a\Phi_a^{N_p} \end{bmatrix}}_{W} x_a[k] + \underbrace{\begin{bmatrix} C_a\Gamma_a & 0 & \dots & 0 \\ C_a\Phi_a\Gamma_a & C_a\Gamma_a & \dots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ C_a\Phi_a^{N_p-1}\Gamma_a & \dots & \dots & C_a\Gamma_a \end{bmatrix}}_{Z} \underbrace{\begin{bmatrix} \Delta u[k] \\ \Delta u[k+1] \\ \vdots \\ \Delta u[k+N_p-1] \end{bmatrix}}_{\Delta U}$$

Next, we will define a cost function ($Q = Q^T \succeq 0$ and $R = R^T \succ 0$):

$$J(\Delta U) = \frac{1}{2}(r_p - Y)^T Q(r_p - Y) + \frac{1}{2}\Delta U^T R \Delta U$$

We then take the partial w.r.t ΔU:

$$\frac{\partial J}{\partial \Delta U} = 0^T = -r_p^T QZ + x_a^T[k]W^T QZ + \Delta U^T Z^T QZ + \Delta U^T R$$

Do some rearranging:

$$0 = (R + Z^T QZ)\Delta U + Z^T Q(-r_p + Wx_a)$$

$$\Delta U^\star = (R + Z^T QZ)^{-1} Z^T Q(r_p - Wx_a)$$

We can then solve for $\Delta u[k]$ using:

$$\Delta u[k] = \underbrace{\begin{bmatrix} I_m & 0 & \dots & 0 \end{bmatrix}}_{N_p \text{ block matrices}} (R + Z^T QZ)^{-1} Z^T Q(r_p - Wx_a)$$

We will break up this equation into:

$$K_{Unconstrained} = (R + Z^T QZ)^{-1} Z^T Q$$

$$\Delta u[k] = K_{Unconstrained}(r_p - Wx_a)$$

This will allow the Simulink model to run faster and not have to recalculate a constant every time. The next parts will show and discuss the results. The design values I used in my controller design are:

$$N_p = 23$$

$$Q0 = \begin{bmatrix} 5 & 0 & 0 \\ 0 & 30 & 0 \\ 0 & 0 & 25 \end{bmatrix}$$

$$R0 = \begin{bmatrix} 0.01 & 0 & 0 \\ 0 & 0.01 & 0 \\ 0 & 0 & 0.0075 \end{bmatrix}$$

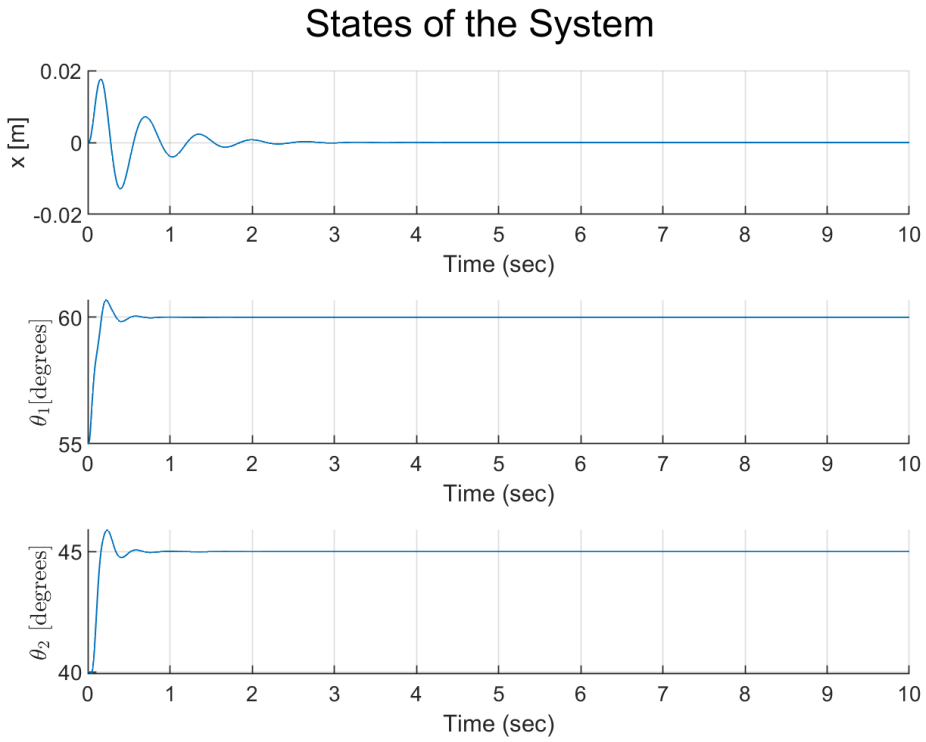
$N_p = 185$ did work better for this specific controller. It was the last value before I had an ill-conditioned matrix. I used 23 instead since it was much faster and worked with the constrained controller in the next section. For this design, I am tracking the following:

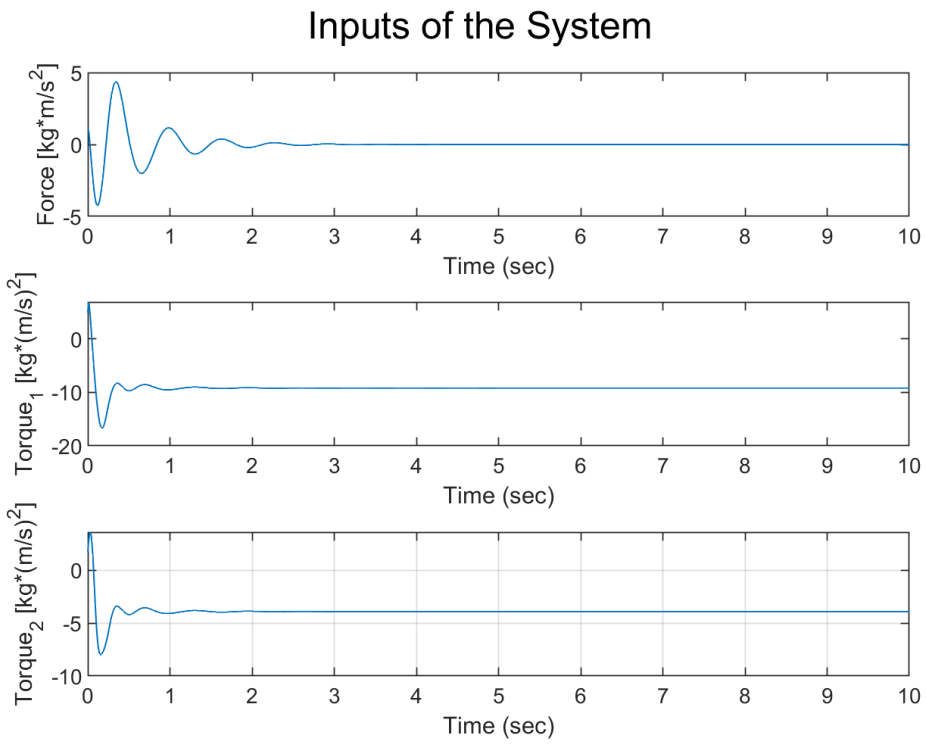
$$v = \begin{bmatrix} 0 & 60^\circ & 45^\circ \end{bmatrix}^T$$

The YouTube video of the animation:

`https://youtu.be/sczaU4yhu3s`

The next page will show graphs of the results. Looking at the results, we see that the two theta states converge in less than a second. This rapid convergence is desirable for these two states. The distance, x, takes a little longer than three seconds. This is a little slower, but we do not care about this state as much as the thetas. In this problem, we do not particularly care about the inputs yet. However, I will reference this problem when first setting my constraints on the next problem.


States of the System
x [m]
0.02
0
-0.02
Time (sec)
θ_1 [degrees]
60
55
Time (sec)
θ_2 [degrees]
45
40
Time (sec)

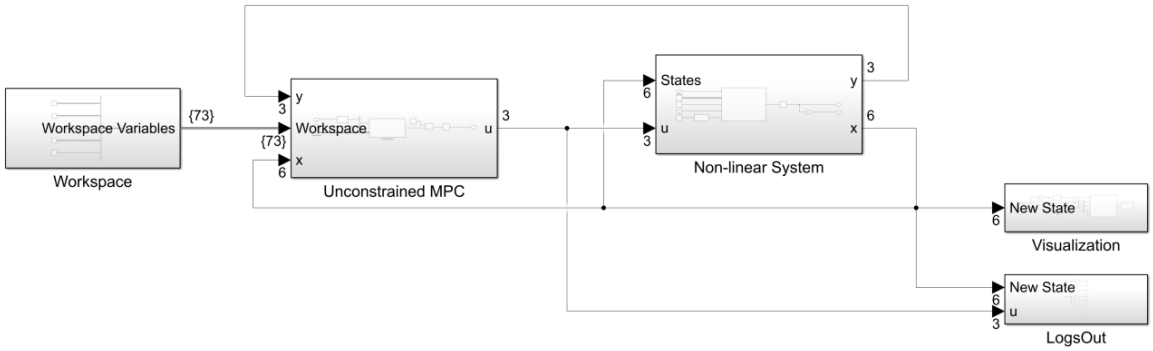

Inputs of the System
Force [kg*m/s^2]
5
0
-5
Time (sec)
Torque$_1$ [kg*(m/s)^2]
0
-10
-20
Time (sec)
Torque$_2$ [kg*(m/s)^2]
0
-5
-10
Time (sec)

I created a section between questions 4 and 5 to discuss the overlapping architecture of the model. I tried to create as few deltas between problems- hopefully allowing me to only show the deltas in the main sections. The below script initializes and runs the Simulink model.

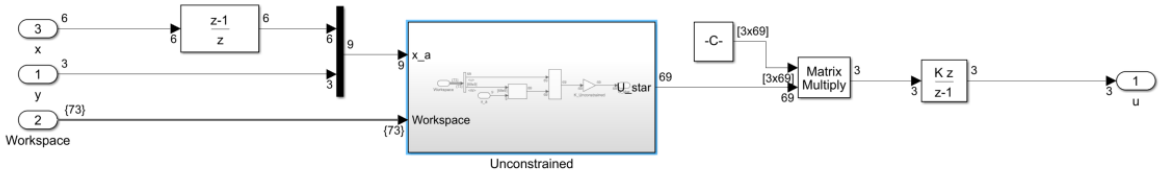
```
%% Problem 2
% Design an MPC for augmented model w/o contraints
Np = 23;                                % Define prediction horizon

% Initialize the problem
[matrices,constraints,r_p,m,x0,int] = Initialize_MPC(sys,Np,1);

% Define the gain matrices:
R0 = [.01 0 0;
      0 .01 0;
      0 0 .0075];
matrices.R = kron(eye(Np),R0);          % Gain Matrix - Control
Q0 = [5 0 0;
      0 30 0;
      0 0 25];
matrices.Q = kron(eye(Np),Q0);          % Gain Matrix - States

% Edit initial conditions from 0
x0.theta1 = -55*pi/180;
x0.theta2 = -40*pi/180;

% Lump together unchanging parts
K_Uncons = (matrices.R + matrices.Z'*matrices.Q*matrices.Z)^-1*↩
    matrices.Z'*matrices.Q;

sim('Aug_MPC_UnCon.slx',10);            % Run simulation
open_system('Aug_MPC_UnCon.slx')        % Open for publish
Graphing(ans)                           % Graph the results
```

On the next page, I will quickly cover the deltas in the augmented model specific to the unconstrained problem. I also want to make some clarifications on what v is. v is what is being tracked. r_p is that v variable being tracked over the prediction horizon.

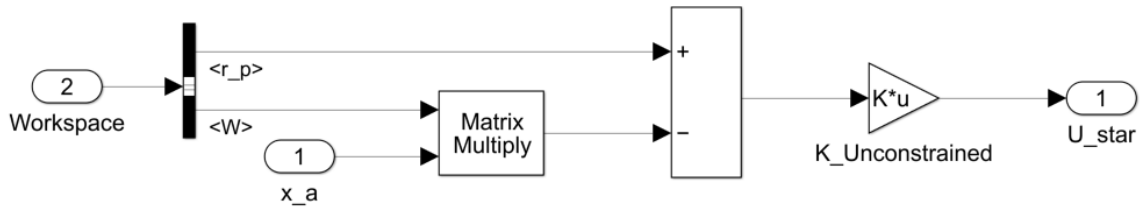
The top-level of the unconstrained model:



When designing this part of the model, I tried to keep parts that would not overlap with the other models contained into its own box. Everything outside of the Unconstrained block will be used again.



The easy calculation that happens with the Unconstrained block:



Design a MPC w/ Constraints (Augmented)

An overall statement of our goal for this section is:

$$\text{Minimize } J(\Delta U)$$

$$\text{subject to } g(\Delta U) \leq 0$$

A first-order Lagrangian algorithm for the optimization problem involving inequality constraints can be written as:

$$x^{[k+1]} = x^{[k]} - \alpha_{[k]}(\nabla f(x^{[k]}) + Dg(x^{[k]})^T \mu^{[k]})$$

$$\mu^{[k+1]} = [\mu^{[k]} = \beta_{[k]} g(x^{[k]})]_+$$

For my design, I decided to keep α and β a constant. I am now going to start defining different parts of this equation. From the overall goal, we can see that we are trying to minimize the cost function w.r.t. ΔU. Starting with $\nabla f(x^{[k]})$:

$$\nabla f(x^{[k]}) = \frac{\partial J}{\partial \Delta U} = (R + Z^T Q Z)\Delta U + Z^T Q(-r_p + W x_a)$$

When coding the model, I defined the following two variables to reduce computations within the for-loop:

$$fun\_grad = Z^T Q(-r_p + W x_a)$$

$$fun\_grad2 = R + Z^T Q Z$$

The function of constraints, $g(x^{[k]})$, depend on what we are constraining. I will derive the three different equations now for output, input, and input rate. First output:

$$Y^{min} \leq Y \leq Y^{max}$$

$$\begin{bmatrix} -Y \\ Y \end{bmatrix} \leq \begin{bmatrix} -Y^{min} \\ Y^{max} \end{bmatrix}$$

Using the equation $Y = W x_a[k] + Z\Delta U$:

$$g(\Delta U) = \begin{bmatrix} -Z \\ Z \end{bmatrix} \Delta U - \begin{bmatrix} -Y^{min} + W x_a[k] \\ Y^{max} - W x_a[k] \end{bmatrix} \leq 0$$

The Jacobian matrix of this constraint is:

$$Dg(\Delta U) = \begin{bmatrix} -Z \\ Z \end{bmatrix}$$

We now want to create constraints for the inputs. We are going to start expanding out $u[k]$ over the prediction horizon.

$$u[k] = u[k-1] + \Delta u[k] \rightarrow u[k] = u[k-1] + \begin{bmatrix} I_m & 0 & \dots & 0 \end{bmatrix} \Delta u[k]$$

$$u[k+1] = u[k] + \Delta u[k+1] \rightarrow u[k+1] = u[k-1] + \begin{bmatrix} I_m & I_m & 0 & \dots & 0 \end{bmatrix}$$

Following this pattern, we can write:

$$\underbrace{\begin{bmatrix} u[k] \\ u[k+1] \\ \vdots \\ u[k+N_p-1] \end{bmatrix}}_{U} = \underbrace{\begin{bmatrix} I_m \\ I_m \\ \vdots \\ I_m \end{bmatrix}}_{E} u[k-1] + \underbrace{\begin{bmatrix} I_m & 0 & \dots & 0 \\ I_m & I_m & \dots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ I_m & I_m & \dots & I_m \end{bmatrix}}_{H} \underbrace{\begin{bmatrix} \Delta u[k] \\ \Delta u[k+1] \\ \vdots \\ \Delta u[k+N_p-1] \end{bmatrix}}_{\Delta U}$$

Moving onto creating the function of constraint for the inputs:

$$U^{min} \leq U \leq U^{max}$$

$$\begin{bmatrix} -U \\ U \end{bmatrix} \leq \begin{bmatrix} -U^{min} \\ U^{max} \end{bmatrix}$$

Using the equation $U = Eu[k-1] + H\Delta U$:

$$g(\Delta U) = \begin{bmatrix} -H \\ H \end{bmatrix} \Delta U - \begin{bmatrix} -U^{min} + Eu[k-1] \\ U^{max} - Eu[k-1] \end{bmatrix} \leq 0$$

The Jacobian matrix of this constraint is:

$$Dg(\Delta U) = \begin{bmatrix} -H \\ H \end{bmatrix}$$

Next we derive the function of constraints for rate of change of the control action. We want to get this in terms of ΔU as well:

$$\Delta u^{min} \leq \Delta u \leq \Delta u^{max}$$

$$\begin{bmatrix} -I_m \\ I_m \end{bmatrix} \Delta u[k] \leq \begin{bmatrix} -\Delta u[k]^{min} \\ \Delta u[k]^{max} \end{bmatrix}$$

Then expanding so it is in terms of ΔU:

$$g(\Delta U) = \underbrace{\begin{bmatrix} -I_m & 0 & \dots & 0 & 0 \\ I_m & 0 & \dots & 0 & 0 \\ 0 & -I_m & 0 & \dots & 0 \\ 0 & I_m & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & -I_m \\ 0 & \dots & \dots & \dots & I_m \end{bmatrix}}_{S} \underbrace{\begin{bmatrix} \Delta u[k] \\ \Delta u[k+1] \\ \vdots \\ \Delta u[k+N_p-2] \\ \Delta u[k+N_p-1] \end{bmatrix}}_{\Delta U} - \underbrace{\begin{bmatrix} -\Delta u^{min} \\ \Delta u^{max} \\ \vdots \\ -\Delta u^{min} \\ \Delta u^{max} \end{bmatrix}}_{P} \leq 0$$

I would like to note that S and P are not formally noted in the notes, but I wanted to give them variables so I could easily call it out later in my model. Next we will define the Jacobian matrix of this function of constraint to be:

$$Dg(\Delta U) = S$$

When starting with the iterative design process, I first started with constraints that were larger than the numerical values presented in the graphs for model without constraints. I wanted to

first make sure that I had a working system. From there, I began defining requirements for my model. I started off by defining the input constraint to:

$$u^{min} = \begin{bmatrix} -10 & -5.9 & -5.9 \end{bmatrix}^T$$

$$u^{max} = \begin{bmatrix} 10 & 5.9 & 5.9 \end{bmatrix}^T$$

I chose $5.9N \cdot m$ because that was the typical torque a stepper motor could provide. I also decided that I wanted 0 overshoot within my response, so I set the output requirements to:

$$y^{min} = \begin{bmatrix} -1 & -60^\circ & -45^\circ \end{bmatrix}^T$$

$$y^{max} = \begin{bmatrix} 1 & 60^\circ & 45^\circ \end{bmatrix}^T$$

I quickly found out that these were not realistic requirements for my system. It was time to loosen up the requirements. When I looked at the past unconstrained response, the input for the first torque shot past $10N \cdot m$. This probably meant that a $5.9N \cdot m$ producing stepper motor was not a realistic design. So I decided to increase that limit to $9N \cdot m$- another common size. The new constraints:

$$u^{min} = \begin{bmatrix} -10 & -9 & -5.9 \end{bmatrix}^T$$

$$u^{max} = \begin{bmatrix} 10 & 9 & 5.9 \end{bmatrix}^T$$

$$y^{min} = \begin{bmatrix} -10 & -65^\circ & -50^\circ \end{bmatrix}^T$$

$$y^{max} = \begin{bmatrix} 10 & 65^\circ & 50^\circ \end{bmatrix}^T$$

This was still a little too much so I loosened requirements to utilize a special torque motor:

$$u^{min} = \begin{bmatrix} -20 & -10 & -5.9 \end{bmatrix}^T$$

$$u^{max} = \begin{bmatrix} 20 & 10 & 5.9 \end{bmatrix}^T$$

The requirements for output remained the same. After struggling with balancing requirements on constraints and performance, I decided not to add input rate constraints. That would make the problem much harder, and it was not required. The design parameters used:

$$N_p = 23$$

$$Q0 = \begin{bmatrix} 5 & 0 & 0 \\ 0 & 30 & 0 \\ 0 & 0 & 25 \end{bmatrix}$$

$$R0 = \begin{bmatrix} 0.01 & 0 & 0 \\ 0 & 0.01 & 0 \\ 0 & 0 & 0.0075 \end{bmatrix}$$

$$\alpha = 0.01$$

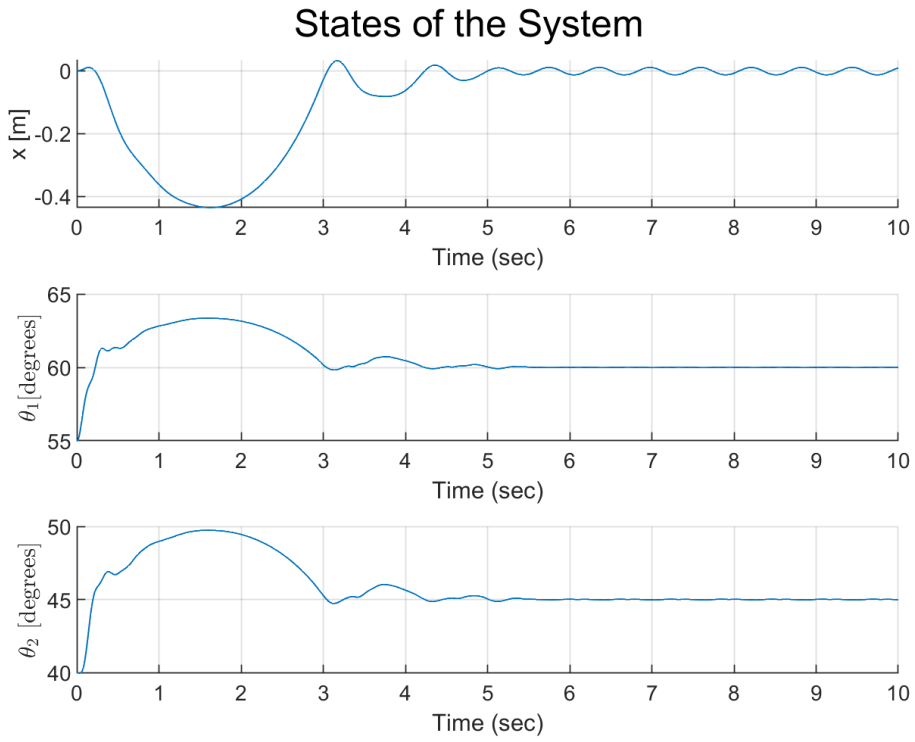
$$\beta = 0.01$$

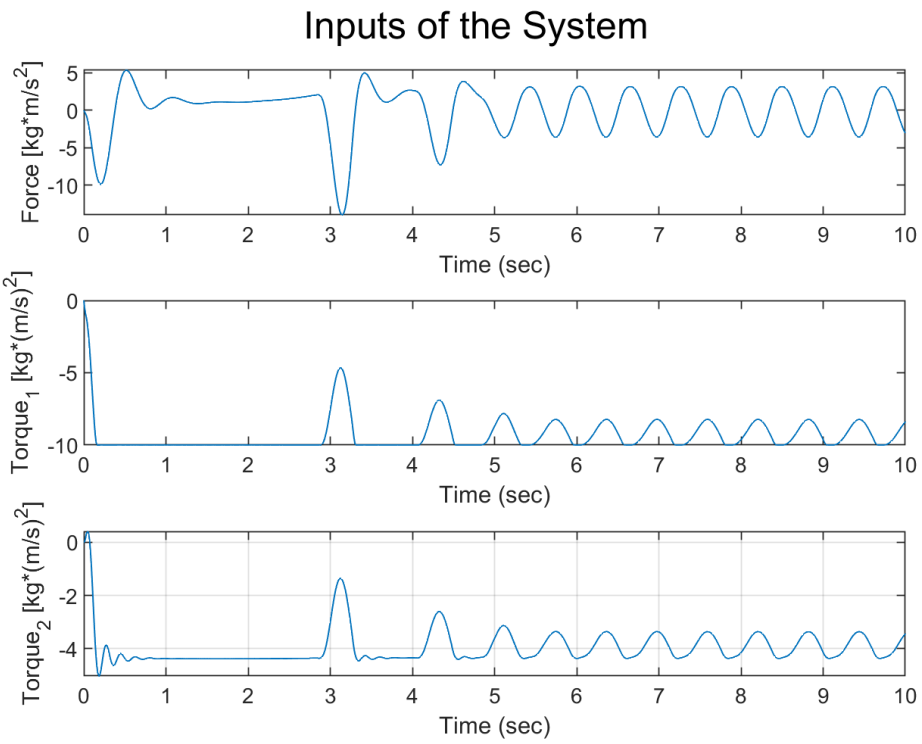
$$iter = 1000$$
$$v = \begin{bmatrix} 0 & 60^\circ & 45^\circ \end{bmatrix}^T$$

The YouTube video of the animation:

`https://youtu.be/B3Cs5VZsosk`

Graphs of the system are on the next page. If we inspect the inputs of the system, we see that they do not cross the requirements. Moving onto the outputs of the system, we see that it takes the theta states much longer to converge than the unconstrained model. This convergence happens around the five second mark. Looking at the x state, we see that the system does not completely settle within 10 seconds, and oscillated about $x = 0$ while slowly decaying. As stated before, I care more for the theta values than the distance, x.

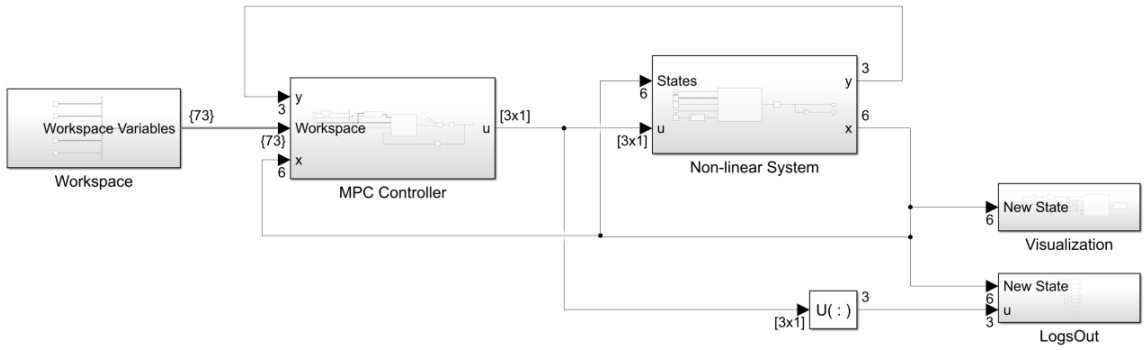

States of the System
x [m]
Time (sec)
θ_1 [degrees]
Time (sec)
θ_2 [degrees]
Time (sec)


Inputs of the System
Force [kg*m/s^2]
Time (sec)
Torque$_1$ [kg*(m/s)^2]
Time (sec)
Torque$_2$ [kg*(m/s)^2]
Time (sec)

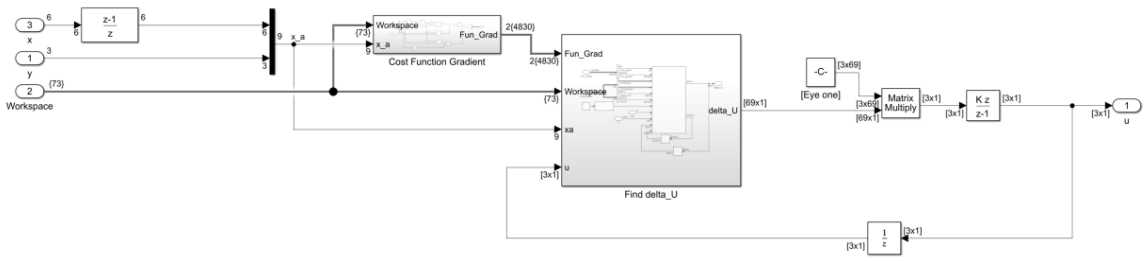
The code to initialize the Simulink model:

```
%% Problem 3
% Impose contraints on the inputs and outputs
close all;
cnt = 2;                                % Set cnt = 2 for contraints in and out
int.mu0 = zeros(cnt*6*Np,1);            % Re-initilaize mu
Bus_Architecture(3,3,6,Np,cnt)          % Run bus-architecture for different set up
R0 = [.01 0 0;
      0 .01 0;
      0 0 .0075];
R = kron(eye(Np),R0);                   % Gain Matrix - Control
Q0 = [5 0 0;
      0 30 0;
      0 0 25];
Q = kron(eye(Np),Q0);                   % Gain Matrix - States

sim('Aug_MPC_Con.slx',10);              % Run simulation
open_system('Aug_MPC_Con.slx')          % Open for publish

Graphing(ans)                           % Graph the results
```

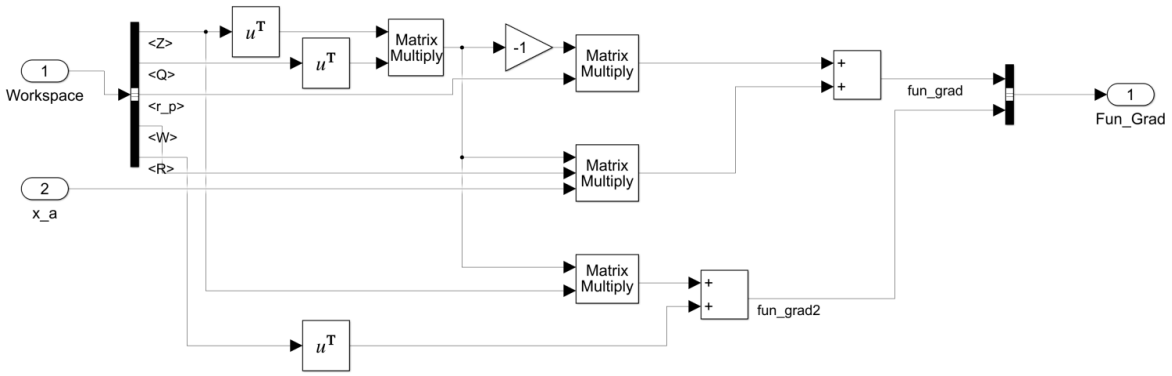
The top-level view of the model:



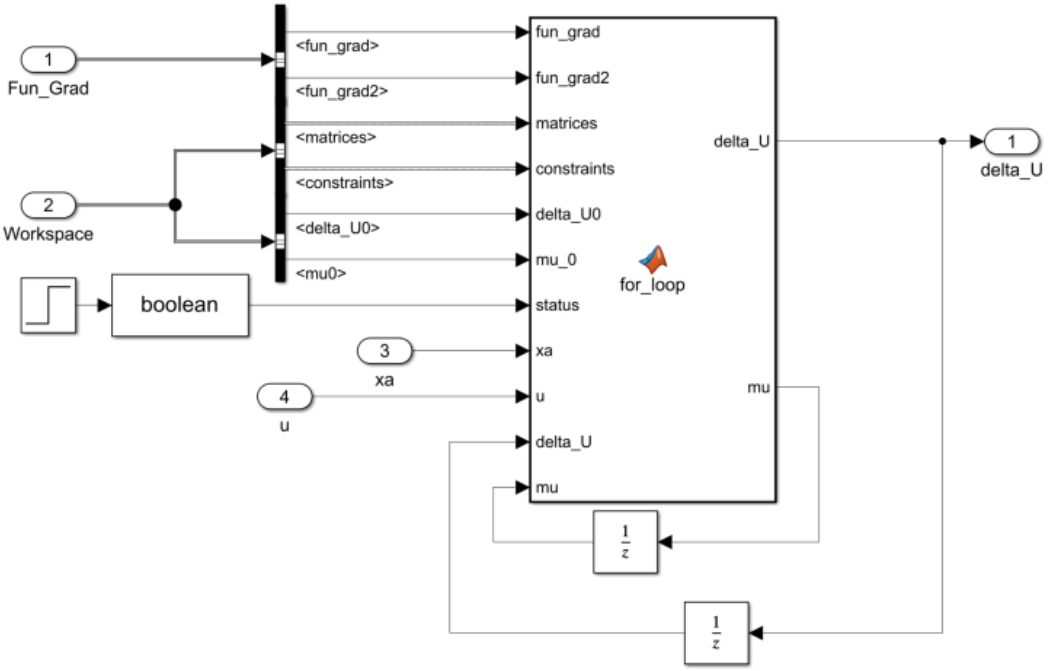
Going down to the MPC Controller block. We see the parts re-used from the unconstrained model on the outside.



The cost function gradient variables are calculated here. These variables stay constant within the for-loop.



The for-loop that iterates over everything:



Implement MPC Controller-Observer Compensator (Augmented)

In the previous sections, we used the state vector from the non-linear equations to feed into the MPC. However, this is not proper, so will create an observer that feeds the MPC estimated states. I will use a LMI solver to find the L matrix. Using the LMI solver added another design parameter. I changed α to η in my LMI equation code to differentiate from the already existing parameter used in the for-loop. The L matrix used:

$$L = \begin{bmatrix} 1.428 & -0.099 & -0.114 \\ -0.106 & 4.150 & -1.528 \\ -0.259 & -1.137 & 3.879 \\ 1.997 & -0.475 & -1.834 \\ 0.295 & 48.804 & -37.478 \\ -1.888 & -34.092 & 43.545 \end{bmatrix}$$

Going onto the changes in the model. The observer is derived from the linear model, and therefore deals with perturbations. Therefore the observer should really be written as:

$$\frac{d}{dt}\delta\tilde{x} = A\delta\tilde{x} + B\delta u + L(\delta y - C\delta\tilde{x})$$

$$\delta\tilde{x} = \tilde{x} - x_e$$

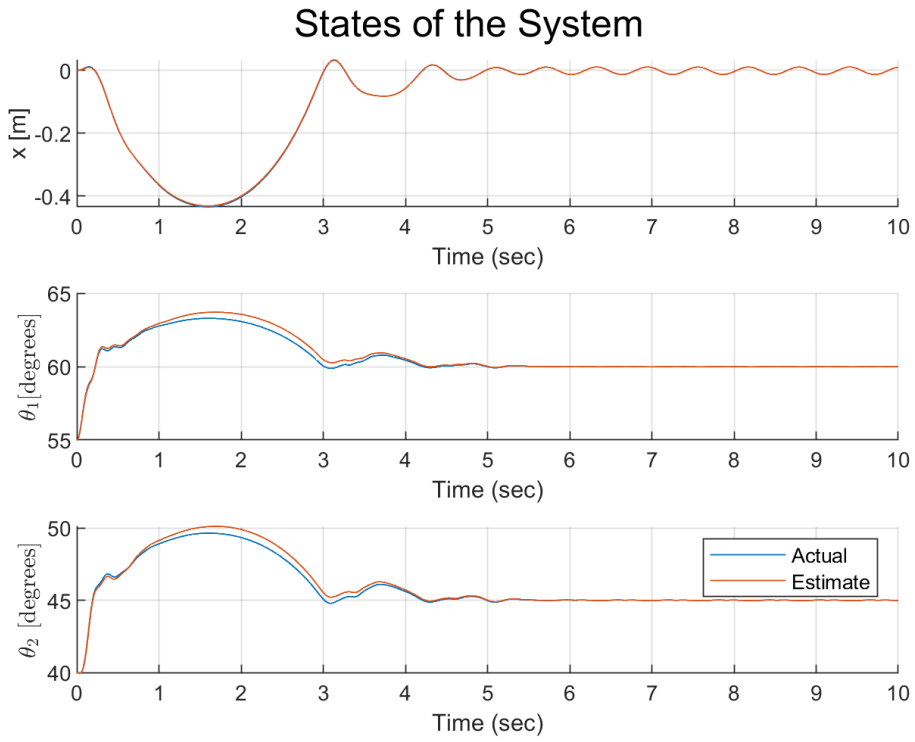
$$\delta y = C(x - x_e)$$

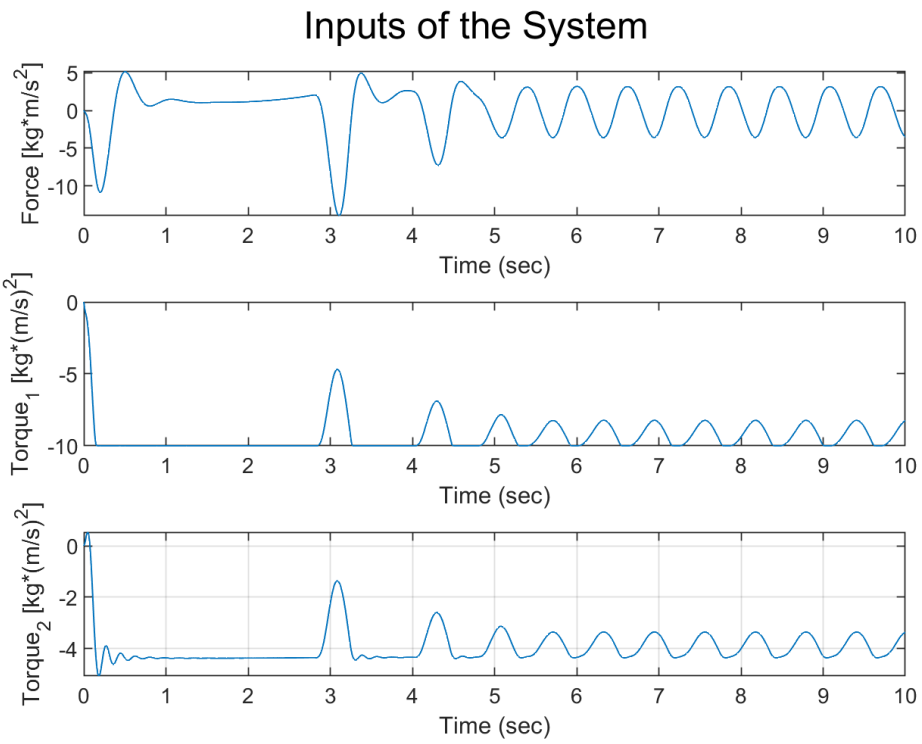
$$\delta u = u - u_e$$

When going to calculate $\tilde{x}_a$, it is important to note that the x_e terms will cancel out. They will not cancel out as so with the non-augmented model. The design parameters used were mostly the same from the previous problem. Only η was added:

$$\eta = \frac{1}{2}$$

I found that you wanted your observer to actually converge very slowly for the best performance out of the controller in terms of requirements and responses. This makes sense since quick convergence between real and estimated states normally results in a large overshoot of the estimated states- which would make the inputs work harder. The initial value for the observer was set to equal initial state- just because this controller would pretty easily “blow up".
Looking at the graphs, I find that my controller performs just as well as it did in the previous problem. The inputs stay within their bounds, and the outputs for the theta converge around five seconds. There is still some oscillation in the x state- which is fine.
The YouTube video:
`https://youtu.be/nFfOmlkBKmg`

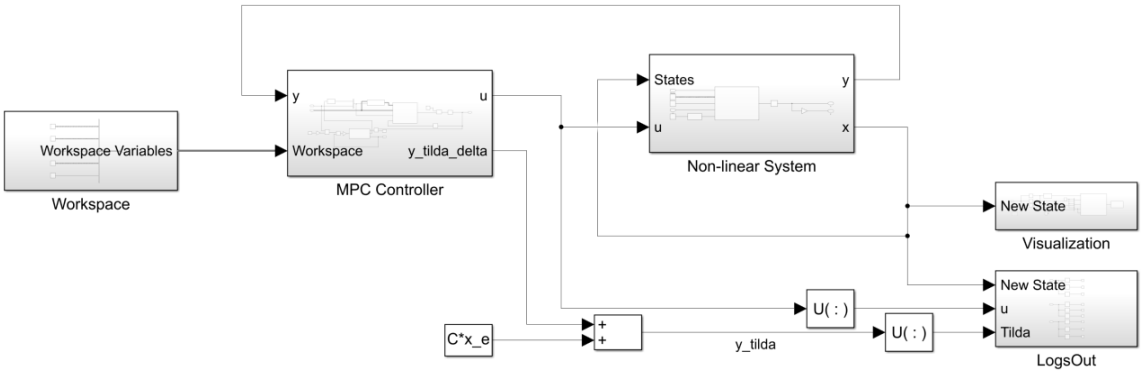

States of the System
x [m]
Time (sec)
θ_1 [degrees]
Time (sec)
θ_2 [degrees]
Actual
Estimate
Time (sec)


Inputs of the System
Force [kg*m/s^2]
Time (sec)
Torque$_1$ [kg*(m/s)^2]
Time (sec)
Torque$_2$ [kg*(m/s)^2]
Time (sec)

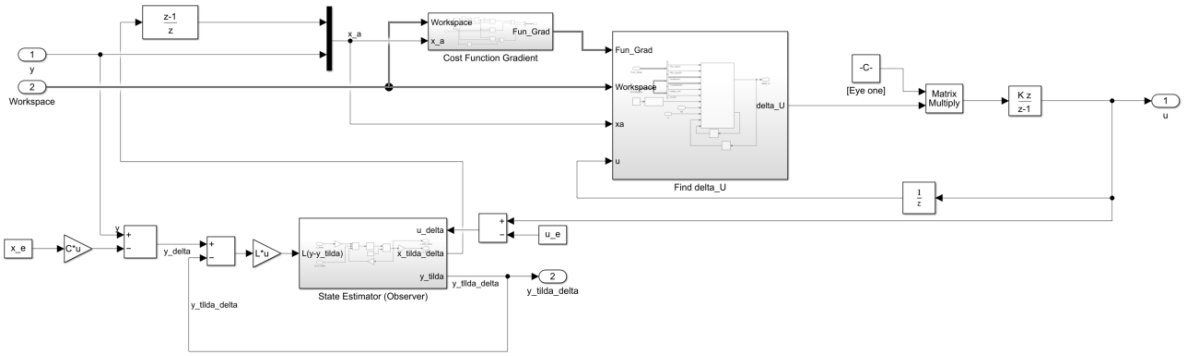
Code to initialize the Simulink model:

```
%% Problem 4
% Combined MPC controller-observer compensator
close all;
m = 3;
n = 6;
p = 3;
cvx_begin sdp quiet

%Variable Definitions
variable P(n,n) symmetric
variable Y(n,p)

% LMIs
eta = 1/2;
P*A + A'*P - Y*C - C'*Y' + eta * P <= 0
P >= eps*eye(n)

cvx_end
L = P^-1*Y                          % Solve for L

sim('Aug_MPC_CO.slx',10);           % Run simulation
open_system('Aug_MPC_CO.slx')       % Open for publish
Graphing_Obs(ans)                   % Graph the results
```

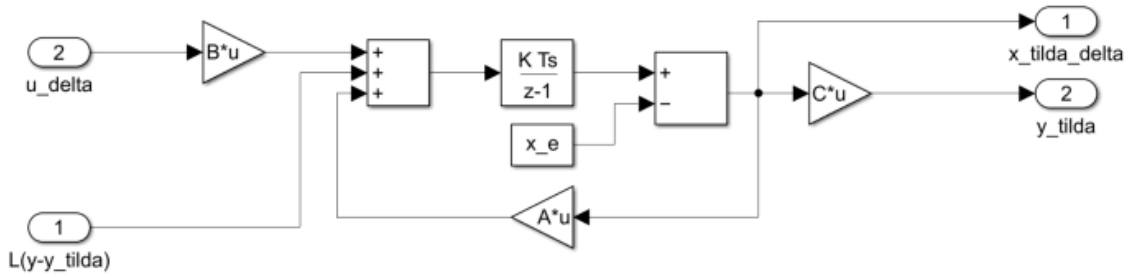
The top-level of the model:



As one can see, there are few changes from the previous problem. However, the observer now has been added to feed in the states to the MPC controller.



The innards of the observer:



Augmented Model Architecture

The following is the code used to initialize every augmented model in the previous sections. Initializes all the needed variables- which can be changed in the main script.

```
function [matrices,constraints,r_p,m,x0,int] = Initialize_MPC(sys,Np,cnt)
    % Function builds necessary matrices to perform MPS

    % Start Code Here::
    % Determine Sizes Matrices
    [p,~] = size(sys.C);
    [~,m] = size(sys.B);
    [n,~] = size(sys.A);


    % Pull Out Variables from Sys
    phi = sys.A;
    gamma = sys.B;
    C = sys.C;

    % Solve for Augmented System
    phi_a = [phi  zeros(n,p);
             C*phi  eye(p)];
    gamma_a = [gamma;
               C*gamma];
    C_a = [zeros(p,n) eye(p)];

    % Solve for W and Z
    W = zeros(p*Np,p+n);            % Initialize W
    Z = zeros(p*Np,m*Np);           % Initialize Z
    index = 1;                      % Initialize var
    % Loop through
    for i = 1:p:p*Np
        W((i:(i+2)),:) = C_a * phi_a^index;
        index = index + 1;
    end
    index = 0;                      % Zeroize for next
    for i =1:p:p*Np
        index2 = i;
        index3 = 0;
        while index2 > 0
            Z(i:(i+2),index2:(index2+2)) = C_a * phi_a^index3 * gamma_a;
            index2 = index2 - p;
            index3 = index3 + 1;
```

```
        end
        index = index + 1;
    end

    % Solve for H and E
    H_eye = zeros(Np);
    for i = 1:Np
        j = i;
        while j > 0
            H_eye(i,j)  = 1;
            j = j - 1;
        end
    end
    H = kron(H_eye,eye(m));          % Solve H
    E = ones(Np,1);
    E = kron(E,eye(m));              % Solve E

    % Initialize Bus
    Bus_Architecture(p,m,n,Np,cnt)  % Call bus architecure

    % Define Reference Signal
    v_one = 0;
    v_two = -60*pi/180;
    v_three = -45*pi/180;
    r_p = zeros(m*Np,1);
    for i = 1:m:m*Np-2
        r_p(i,1) = v_one;
        r_p(i+1,1) = v_two;
        r_p(i+2,1) = v_three;
    end

    % Constraint Bus Initialize
    y_min = [-10;-65*pi/180;-50*pi/180];
    y_max = [10;65*pi/180;50*pi/180];
    u_min = [-20;-10;-5.9];
    u_max = [20;10;5.9];
    constraints.Y_min = kron(ones(Np,1),y_min);
    constraints.Y_max = kron(ones(Np,1),y_max);
    constraints.U_min = kron(ones(Np,1),u_min);
    constraints.U_max = kron(ones(Np,1),u_max);

    % Create Matrix Bus
    matrices.W = W;
    matrices.Z = Z;
    matrices.phi_a = phi_a;
    matrices.gamma_a = gamma_a;
    matrices.C_a = C_a;
```

```
    matrices.H = H;
    matrices.E = E;
    matrices.Z_aug = [-Z;Z];
    matrices.H_aug = [-H;H];

    % Initialize x0
    x0.x = 0;
    x0.theta1 = 0;
    x0.theta2 = 0;
    x0.x_dot = 0;
    x0.theta1_dot = 0;
    x0.theta2_dot = 0;

    % Initialization Bus
    int.delta_U0 = zeros(m*Np,1);
    int.mu0 = zeros(cnt*n*Np,1);

end
```

Bus architecture used for the augmented system:

```
function Bus_Architecture(p,m,n,Np,cnt)
    % Give Definition to Bus of States
    state_names = {'x','theta1','theta2',...
        'x_dot','theta1_dot','theta2_dot'};
    state_types = {'double','double','double',...
        'double','double','double'};
    for i = 1:length(state_names)
        temp(i) = Simulink.BusElement;
        temp(i).Name = state_names{i};
        temp(i).SampleTime = -1;
        temp(i).Complexity = 'real';
        temp(i).Dimensions = 1;
        temp(i).DataType = state_types{i};
    end
    State_bus = Simulink.Bus;
    State_bus.Elements = temp;
    assignin('base','State_bus',State_bus)
    clear temp state_names state_types

    % Give Definition to Reference Bus
    ref_names = {'one','two','three'};
    ref_types = {'double','double','double'};
    for i = 1:length(ref_names)
        temp(i) = Simulink.BusElement;
        temp(i).Name = ref_names{i};
```

```
        temp(i).SampleTime = -1;
        temp(i).Complexity = 'real';
        temp(i).Dimensions = 1;
        temp(i).DataType = ref_types{i};
    end
    Reference_bus = Simulink.Bus;
    Reference_bus.Elements = temp;
    assignin('base','Reference_bus',Reference_bus)
    clear temp ref_names ref_types

    % Give Definition to Contrain Bus
    ref_names = {'Y_min','Y_max','U_min',...
                 'U_max'};
    ref_types = {'double','double','double',...
                 'double'};
    ref_dim = {[Np*p],[Np*p],[Np*m],...
               [Np*m]};
    for i = 1:length(ref_names)
        temp(i) = Simulink.BusElement;
        temp(i).Name = ref_names{i};
        temp(i).SampleTime = -1;
        temp(i).Complexity = 'real';
        temp(i).Dimensions = ref_dim{i};
        temp(i).DataType = ref_types{i};
    end
    Contraint_bus = Simulink.Bus;
    Contraint_bus.Elements = temp;
    assignin('base','Contraint_bus',Contraint_bus)
    clear temp ref_names ref_types

    % Give Definition to Matrix Bus
    ref_names = {'W','Z','phi_a',...
                 'gamma_a','C_a',...
                 'H','E','Z_aug','H_aug',...
                 'R','Q'};
    ref_types = {'double','double','double',...
                 'double','double','double','double',...
                 'double','double','double','double',...
                 'double','double'};
    ref_dim = {[p*Np,p+n],[p*Np,p*Np],[p+n,p+n],...
               [p+n,m],[m,p+n],...
               [p*Np,p*Np],[Np*m,m],[2*p*Np,p*Np],...
               [2*p*Np,p*Np],[m*Np,m*Np],[m*Np,m*Np]};
    for i = 1:length(ref_names)
        temp(i) = Simulink.BusElement;
        temp(i).Name = ref_names{i};
        temp(i).SampleTime = -1;
```

```
        temp(i).Complexity = 'real';
        temp(i).Dimensions = ref_dim{i};
        temp(i).DataType = ref_types{i};
    end
    Matrix_bus = Simulink.Bus;
    Matrix_bus.Elements = temp;
    assignin('base','Matrix_bus',Matrix_bus)
    clear temp ref_names ref_types

    % Give Definition to Initialization Bus
    ref_names = {'delta_U0','mu0'};
    ref_types = {'double','double'};
    ref_dim = {[p*Np,1],[cnt*n*Np,1]};
    for i = 1:length(ref_names)
        temp(i) = Simulink.BusElement;
        temp(i).Name = ref_names{i};
        temp(i).SampleTime = -1;
        temp(i).Complexity = 'real';
        temp(i).Dimensions = ref_dim{i};
        temp(i).DataType = ref_types{i};
    end
    Initial_bus = Simulink.Bus;
    Initial_bus.Elements = temp;
    assignin('base','Initial_bus',Initial_bus)
    clear temp ref_names ref_types
end
```

Graphing without the observer (works both types of models):

```
function Graphing(ans)
    % For graphing without observer
    logsout = ans.logsout;

    % Plot state vs time
    figure(1)
    hold on
    sgtitle('States of the System')
    subplot(3,1,1)
    hold on
    plot(logsout{1}.Values.Time',logsout{1}.Values.Data)
    hold off
    xlabel('Time (sec)')
    ylabel('x [m]')
    grid
    subplot(3,1,2)
    hold on
```

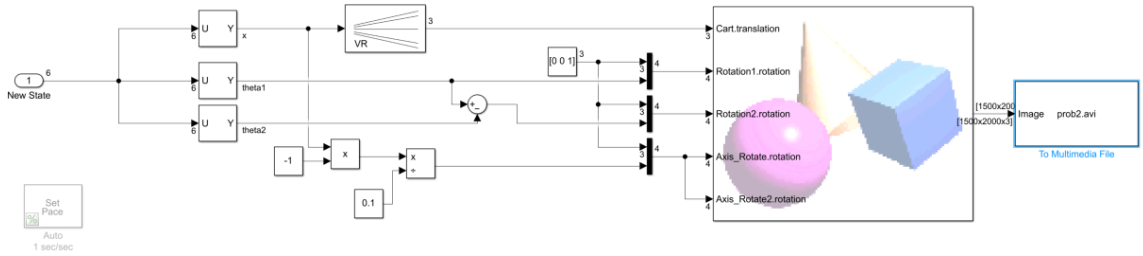
```
    plot(logsout{2}.Values.Time',-logsout{2}.Values.Data*180/pi)
    hold off
    xlabel('Time (sec)')
    ylabel('$\theta_1$[degrees]','Interpreter','latex')
    grid
    subplot(3,1,3)
    hold on
    plot(logsout{3}.Values.Time',-logsout{3}.Values.Data*180/pi)
    hold off
    xlabel('Time (sec)')
    ylabel('$\theta_2$ [degrees]','Interpreter','latex')
    grid
    hold off

    % Plot u vs time
    figure(2)
    hold on
    sgtitle('Inputs of the System')
    subplot(3,1,1)
    plot(logsout{4}.Values.Time',logsout{4}.Values.Data)
    xlabel('Time (sec)')
    ylabel('Force [kg*m/s^2]')
    subplot(3,1,2)
    plot(logsout{5}.Values.Time',-logsout{5}.Values.Data)
    xlabel('Time (sec)')
    ylabel('Torque_1 [kg*(m/s)^2]')
    subplot(3,1,3)
    plot(logsout{6}.Values.Time',-logsout{6}.Values.Data)
    xlabel('Time (sec)')
    ylabel('Torque_2 [kg*(m/s)^2]')
    grid
    hold off

end
```

Graphing with the observer (works both types of models):

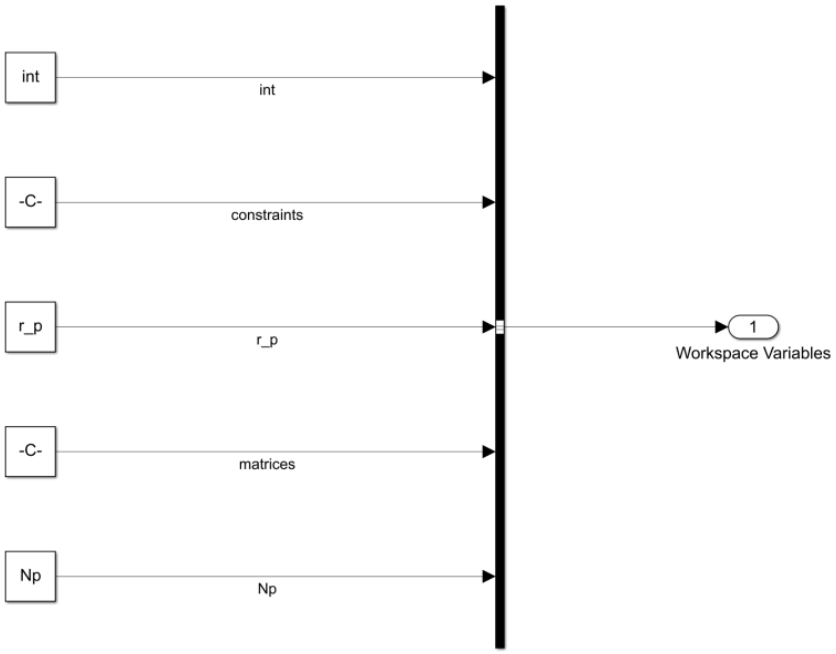
```
function Graphing_Obs(ans)
    % For graphing without observer
    logsout = ans.logsout;

    % Plot state vs time
    figure(1)
    hold on
    sgtitle('States of the System')
    subplot(3,1,1)
```

```
    hold on
    plot(logsout{1}.Values.Time',logsout{1}.Values.Data)
    plot(logsout{6}.Values.Time',logsout{6}.Values.Data)
    hold off
    xlabel('Time (sec)')
    ylabel('x [m]')
    grid
    subplot(3,1,2)
    hold on
    plot(logsout{2}.Values.Time',-logsout{2}.Values.Data*180/pi)
    plot(logsout{7}.Values.Time',-logsout{7}.Values.Data*180/pi)
    hold off
    xlabel('Time (sec)')
    ylabel('$\theta_1$[degrees]','Interpreter','latex')
    grid
    subplot(3,1,3)
    hold on
    plot(logsout{3}.Values.Time',-logsout{3}.Values.Data*180/pi)
    plot(logsout{8}.Values.Time',-logsout{8}.Values.Data*180/pi)
    legend('Actual','Estimate')
    hold off
    xlabel('Time (sec)')
    ylabel('$\theta_2$ [degrees]','Interpreter','latex')
    grid
    hold off

    % Plot u vs time
    figure(2)
    hold on
    sgtitle('Inputs of the System')
    subplot(3,1,1)
    plot(logsout{4}.Values.Time',logsout{4}.Values.Data)
    xlabel('Time (sec)')
    ylabel('Force [kg*m/s^2]')
    subplot(3,1,2)
    plot(logsout{5}.Values.Time',-logsout{5}.Values.Data)
    xlabel('Time (sec)')
    ylabel('Torque_1 [kg*(m/s)^2]')
    subplot(3,1,3)
    plot(logsout{6}.Values.Time',-logsout{9}.Values.Data)
    xlabel('Time (sec)')
    ylabel('Torque_2 [kg*(m/s)^2]')
    grid
    hold off

end
```

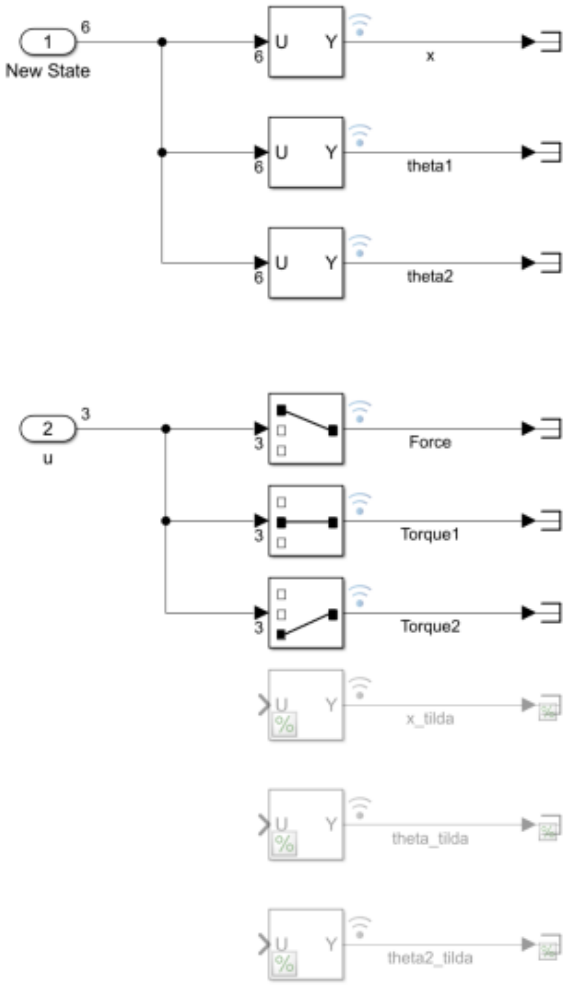
The visualization block:



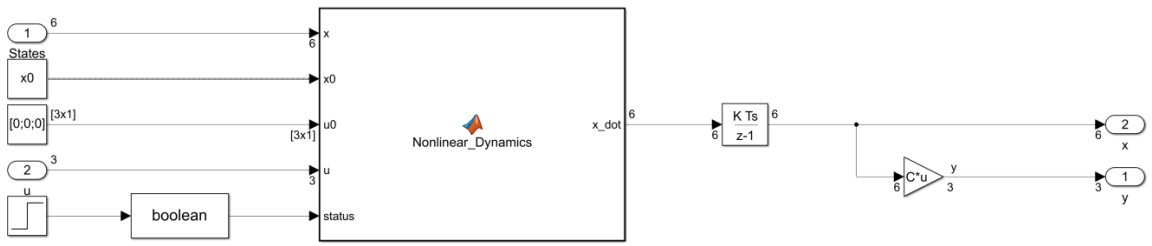
The workspace block:



The logsout block is below. This block changes a little for the observer model. Bottom three blocks get uncommented to record the estimated values.



The nonlinear equations of the system:



Re-Do Design with Non-Augmented Model

The two methods (augmented and non-augmented) are pretty similar, and are derived in the same manner after augmenting the matrices. Since they are derived in much of the same manner, I will skip some work. You will notice the equations are the same other than the non-augmented matrices are used to construct Y, W, Z, and U.

$$\begin{bmatrix} x[k+1|k] \\ x[k+2|k] \\ \vdots \\ x[k+N_p|k] \end{bmatrix} = \begin{bmatrix} \Phi \\ \Phi^2 \\ \vdots \\ \Phi^{N_p} \end{bmatrix} x[k] + \begin{bmatrix} \Gamma & 0 & \dots & 0 \\ \Phi\Gamma & \Gamma & \dots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ \Phi^{N_p-1}\Gamma & \dots & \dots & \Gamma \end{bmatrix} \begin{bmatrix} u[k] \\ u[k+1] \\ \vdots \\ u[k+N_p-1] \end{bmatrix}$$

$$\begin{bmatrix} y[k+1|k] \\ y[k+2|k] \\ \vdots \\ y[k+N_p|k] \end{bmatrix} = \begin{bmatrix} Cx[k+1|k] \\ Cx[k+2|k] \\ \vdots \\ Cx[k+N_p|k] \end{bmatrix}$$

$$\underbrace{\begin{bmatrix} y[k+1|k] \\ y[k+2|k] \\ \vdots \\ y[k+N_p|k] \end{bmatrix}}_{Y} == \underbrace{\begin{bmatrix} C\Phi \\ C\Phi^2 \\ \vdots \\ C\Phi^{N_p} \end{bmatrix}}_{W} x[k] + \underbrace{\begin{bmatrix} C\Gamma & 0 & \dots & 0 \\ C\Phi\Gamma & C\Gamma & \dots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ C\Phi^{N_p-1}\Gamma & \dots & \dots & C\Gamma \end{bmatrix}}_{Z} \underbrace{\begin{bmatrix} u[k] \\ u[k+1] \\ \vdots \\ u[k+N_p-1] \end{bmatrix}}_{U}$$

The cost function is built with U instead of ΔU.

$$J(U) = \frac{1}{2}(r_p - Y)^T Q(r_p - Y) + \frac{1}{2}U^T RU$$

Skipping forward (same exact process as augmented system):

$$U^\star = (R + Z^T QZ)^{-1} Z^T Q(r_p - Wx[k])$$

We can then solve for $u[k]$ using:

$$u[k] = \underbrace{\begin{bmatrix} I_m & 0 & \dots & 0 \end{bmatrix}}_{N_p \text{ block matrices}} (R + Z^T QZ)^{-1} Z^T Q(r_p - Wx[k])$$

We will break up this equation into:

$$K_{Unconstrained} = (R + Z^T QZ)^{-1} Z^T Q$$

$$u[k] = K_{Unconstrained}(r_p - Wx[k])$$

The following are the design parameters used:

$$N_p = 23$$

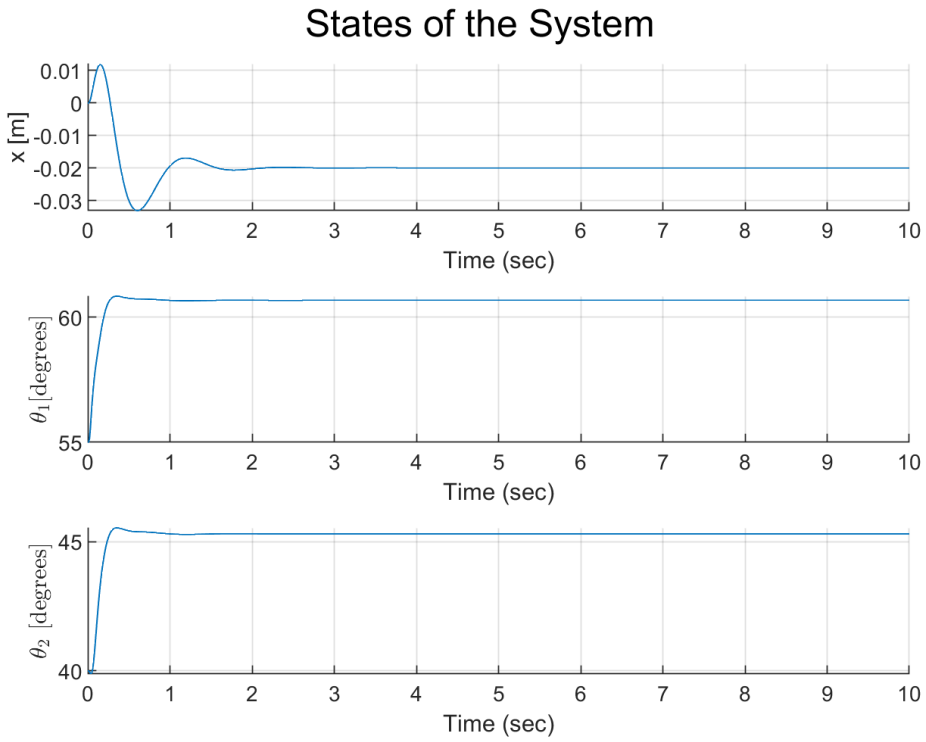
$$Q0 = \begin{bmatrix} 15 & 0 & 0 \\ 0 & 28 & 0 \\ 0 & 0 & 25 \end{bmatrix}$$

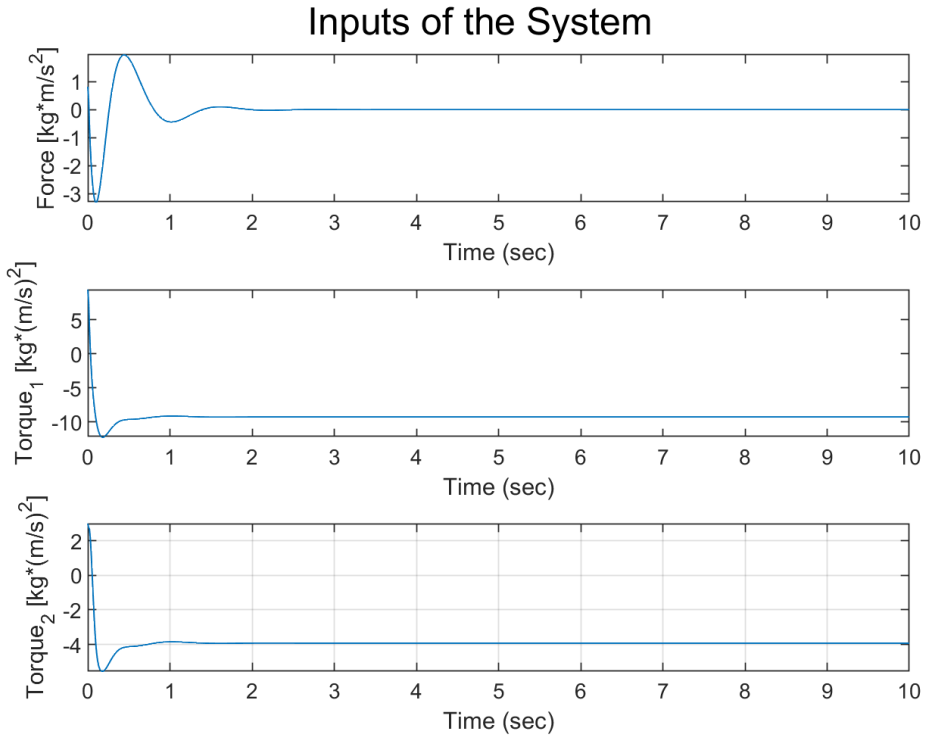
$$R = 0.001 * eye(3 * N_p)$$

For this design, I am tracking the following:

$$v = \begin{bmatrix} 0 & 60^\circ & 45^\circ \end{bmatrix}^T$$

The Youtube video of the results for the unconstrained controller can be found here:
`https://youtu.be/MEpjVk2CclE`
Next page contains graphs of the unconstrained controller response. One will notice the tiny steady state errors in the system. The states all settle in less than two seconds, but I could not get rid of the tiny steady state error. The x state settles at -0.02 instead of 0. θ_1 settles at 60.6696° instead of 60°. θ_2 settles at 45.3008° instead of 45°. Such small steady state error are usually not of concern and can normally be fixed with an integrator.
The inputs of the system are not yet of concern, but it is encouraging that they are so far looking reasonable.

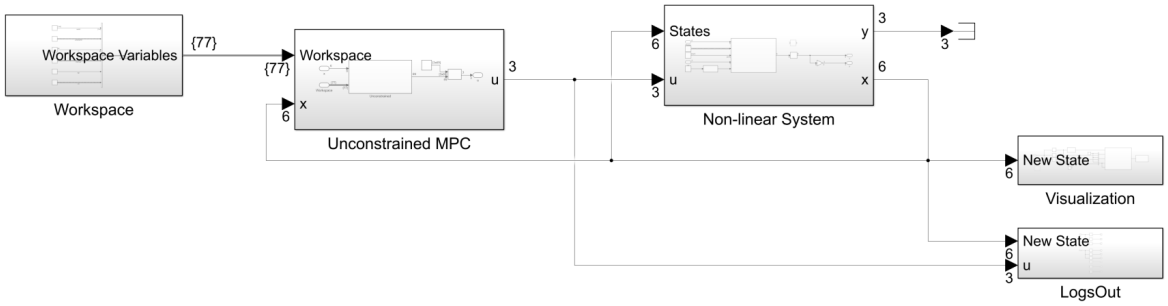

States of the System
x [m]
0.01
0
-0.01
-0.02
-0.03
Time (sec)
θ_1 [degrees]
60
55
Time (sec)
θ_2 [degrees]
45
40
Time (sec)


Inputs of the System
Force [kg*m/s^2]
1
0
-1
-2
-3
Time (sec)
Torque$_1$ [kg*(m/s)^2]
5
0
-5
-10
Time (sec)
Torque$_2$ [kg*(m/s)^2]
2
0
-2
-4
Time (sec)

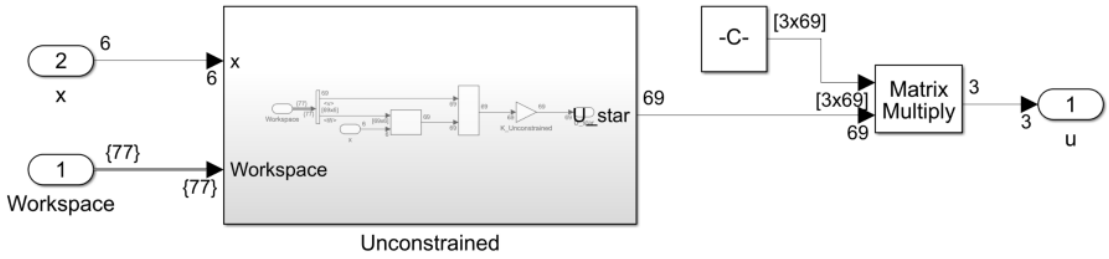
Code to initialize variables to run model:

```
%% Problem 5a
% Design an MPC for non-augmented model w/o contraints
% Begin Prelininaries
clear; clc; close all;
load('Values_Rev3.mat')

% Create continuous system
sys_c = ss(A,B,C,D);
h = 0.01;

% Discritize system
sys = c2d(sys_c,h);

% Start Actual::
Np = 23;                              % Define the prediction horizon
% Initialize the problem
[matrices,constraints,r_p,m,x0,int] = Initialize_MPC_non(sys,Np,1);

% Define weight matrices
matrices.R = 0.001*eye(3*Np);         % Gain Matrix - Control
Q0 = [15 0 0;
      0 28 0;
      0 0 25];
matrices.Q = kron(eye(Np),Q0);        % Gain Matrix - Input

% Edit initial conditions from 0
x0.theta1 = -55*pi/180;
x0.theta2 = -40*pi/180;

% Lump together unchanging parts
K_Uncons = (matrices.R + matrices.Z'*matrices.Q*matrices.Z)^-1*↩
    matrices.Z'*matrices.Q;

sim('NonAug_MPC_UnCon.slx',10);       % Run simulation
open_system('NonAug_MPC_UnCon.slx')% For publishing later
Graphing(ans)                         % Graph the results
```

The top-level view of the model is below. For the sake of keeping as much as possible the same, you will notice the y vector still outputs but goes to the termination block. The y vector is no longer needed for the MPC block to work.



The difference worth noting from the augmented model is the fact that we are no longer accumulating the inputs. What we solve for is the actual u and not the change of u.



We are now going to derive the deltas between the augmented and non-augmented constrained system. First, we will start with the output:

$$Y^{min} \leq Y \leq Y^{max}$$

$$\begin{bmatrix} -Y \\ Y \end{bmatrix} \leq \begin{bmatrix} -Y^{min} \\ Y^{max} \end{bmatrix}$$

Using the equation $Y = Wx[k] + ZU$:

$$g(U) = \begin{bmatrix} -Z \\ Z \end{bmatrix} U - \begin{bmatrix} -Y^{min} + Wx[k] \\ Y^{max} - Wx[k] \end{bmatrix} \leq 0$$

The Jacobian matrix of this constraint is:

$$Dg(U) = \begin{bmatrix} -Z \\ Z \end{bmatrix}$$

We now want to create the constraints for the inputs. We are going to start expanding out $u[k]$ over the prediction horizon.

$$u[k]^{min} \leq u[k] \leq u[k]^{max}$$

$$\begin{bmatrix} -I_m \\ I_m \end{bmatrix} u[k] \leq \begin{bmatrix} -u[k]^{min} \\ u[k]^{max} \end{bmatrix}$$

With this stated, we really want U- which is $u[k]$ over the prediction horizon.

$$g(U) = \underbrace{\begin{bmatrix} -I_m & 0 & \dots & 0 & 0 \\ I_m & 0 & \dots & 0 & 0 \\ 0 & -I_m & 0 & \dots & 0 \\ 0 & I_m & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & -I_m \\ 0 & \dots & \dots & \dots & I_m \end{bmatrix}}_{S} U - \underbrace{\begin{bmatrix} -u^{min} \\ u^{max} \\ -u^{min} \\ u^{max} \\ \vdots \\ -u^{min} \\ u^{max} \end{bmatrix}}_{F} \leq 0$$

The Jacobian matrix of this constraint is:

$$Dg(U) = S$$

The last thing to be derived is the constraint on the rate of change of the control action:

$$\Delta u[k] = u[k] - u[k-1]$$

$$\Delta u[k+1] = u[k+1] - u[k]$$

$$\vdots$$

$$\Delta u[k+N_p-1] = u[k+N_p-1] - u[k+N_p-2]$$

Following this pattern, we can write:

$$\underbrace{\begin{bmatrix} \Delta u[k] \\ \Delta u[k+1] \\ \vdots \\ \Delta u[k+N_p-2] \\ \Delta u[k+N_p-1] \end{bmatrix}}_{\Delta U} = \underbrace{\begin{bmatrix} I_m & 0 & \dots & 0 & 0 \\ -I_m & I_m & \dots & 0 & 0 \\ \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & I_m & 0 \\ 0 & 0 & \dots & -I_m & I_m \end{bmatrix}}_{G} \underbrace{\begin{bmatrix} u[k] \\ u[k+1] \\ \vdots \\ u[k+N_p-2] \\ u[k+N_p-1] \end{bmatrix}}_{U} - \underbrace{\begin{bmatrix} I_m \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}}_{K} u[k-1]$$

Moving onto creating the function of constraint for the inputs:

$$\Delta U^{min} \leq \Delta U \leq \Delta U^{max}$$

$$\begin{bmatrix} -\Delta U \\ \Delta U \end{bmatrix} \leq \begin{bmatrix} -\Delta U^{min} \\ \Delta U^{max} \end{bmatrix}$$

Using $\Delta U = GU - Ku[k-1]$, we can then write the equation as:

$$g(U) = \begin{bmatrix} -G \\ G \end{bmatrix} U - \begin{bmatrix} -\Delta U^{min} - Ku[k-1] \\ \Delta U^{max} + Ku[k-1] \end{bmatrix} \leq 0$$

The Jacobian matrix of this constraint is:

$$Dg(U) = \begin{bmatrix} -G \\ G \end{bmatrix}$$

When going to do the constrained version of the non-augmented model, I found that I had to increase the N_p and iterations to get it to work. This increase made the model very slow. The design variables were:

$$N_p = 43$$

$$\alpha = 0.01$$

$$\beta = 0.01$$

$$iter = 2000$$

$$Q0 = \begin{bmatrix} 20 & 0 & 0 \\ 0 & 30 & 0 \\ 0 & 0 & 30 \end{bmatrix}$$

$$R0 = \begin{bmatrix} 0.01 & 0 & 0 \\ 0 & 0.001 & 0 \\ 0 & 0 & 0.005 \end{bmatrix}$$

The constraints placed on the model:

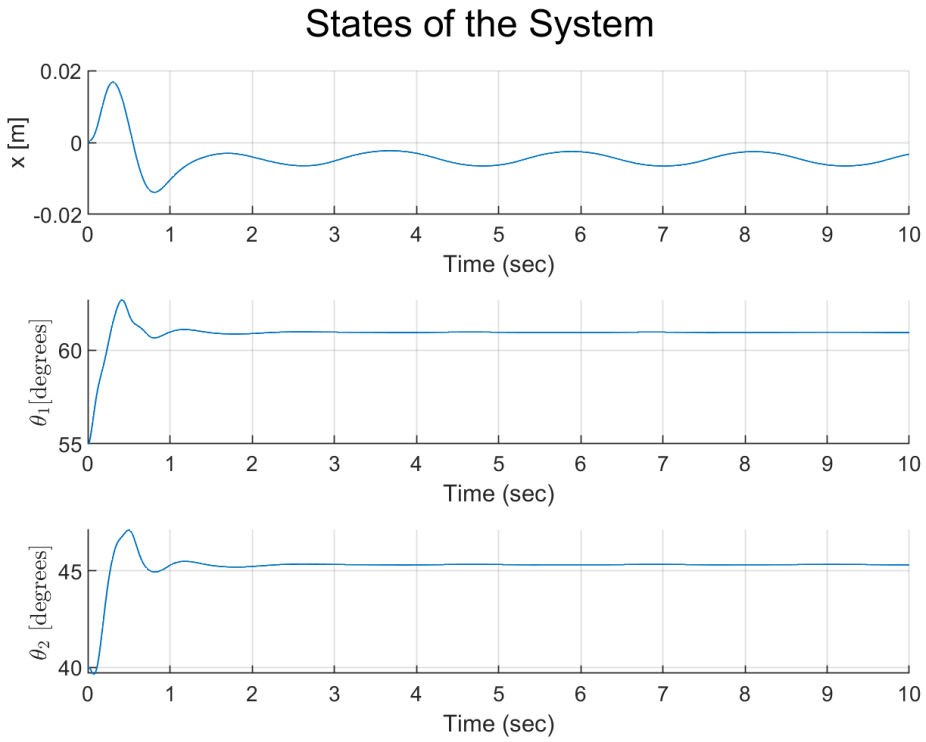
$$u^{min} = \begin{bmatrix} -50 & -12 & -5.9 \end{bmatrix}^T$$

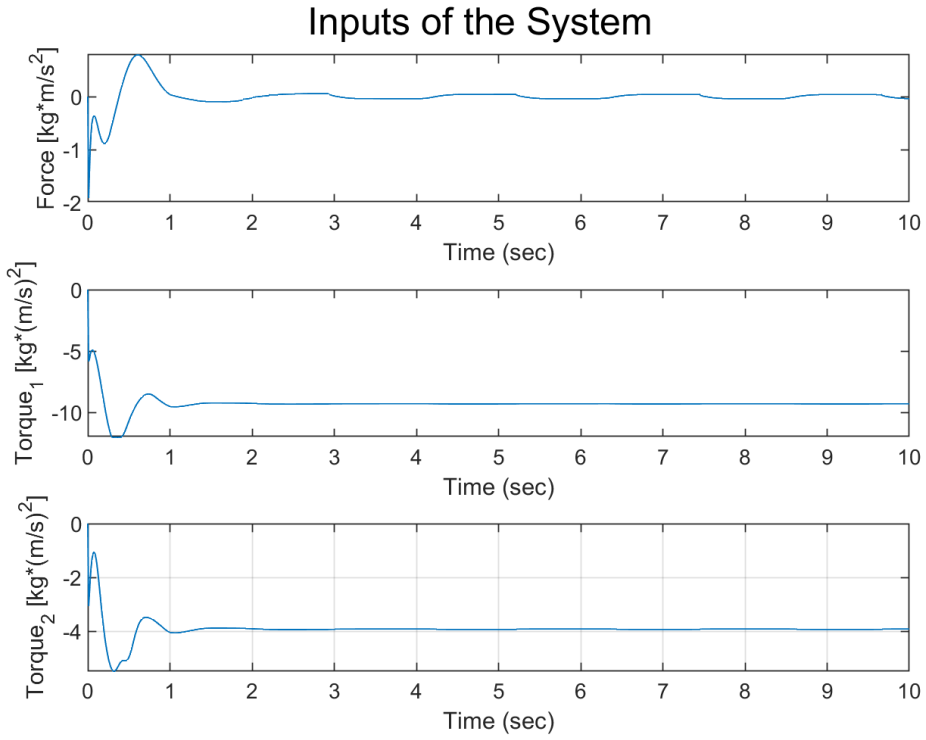
$$u^{max} = \begin{bmatrix} 50 & 12 & 5.9 \end{bmatrix}^T$$

$$y^{min} = \begin{bmatrix} -20 & -62^\circ & -47^\circ \end{bmatrix}^T$$

$$y^{max} = \begin{bmatrix} 20 & 62^\circ & 47^\circ \end{bmatrix}^T$$

The Youtube video of the results for the constrained controller can be found here:
`https://youtu.be/bu8FDz0dHpI`
First, I would like to talk about how slow this simulation is. Compared to the augmented model, it is much slower, and harder to work with. As a consequence,I am not sure this type of controller would be effective in a high performance aircraft. Maybe in a commercial aircraft.
I will now start commenting on the inputs into the system. Glancing at the input graphs, one can see that the inputs stayed within the boundaries of the constraints without any weird noise. I had to increase the iterations from 1000 to 2000 to get rid of the original noise. I would like to note that the inputs change rapidly in the beginning before settling. Those fast changes could make adding a rate change constraint difficult to the problem. Fortunately, it is not required for this assignment.
Moving onto the states, one will notice that the tiny steady state errors persist to follow me into the constrained model. The angle states all settle in less than two seconds, but the state x oscillates about it settling value. The x state oscillates around -0.005 instead of 0. θ_1 settles at 60.9506° instead of 60°. θ_2 settles at 45.3093° instead of 45°. Such small steady state error are usually not of concern and can normally be fixed with an integrator.

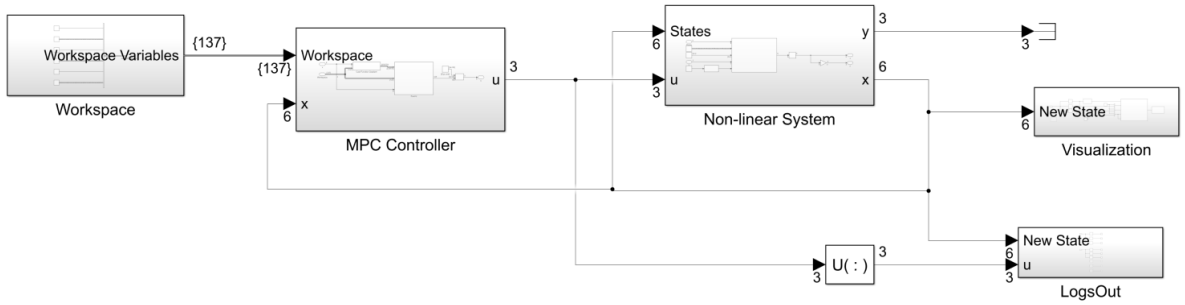

States of the System
x [m]
0.02
0
-0.02
Time (sec)
θ_1 [degrees]
60
55
Time (sec)
θ_2 [degrees]
45
40
Time (sec)


Inputs of the System
Force [kg*m/s^2]
Time (sec)
Torque$_1$ [kg*(m/s)^2]
Time (sec)
Torque$_2$ [kg*(m/s)^2]
Time (sec)

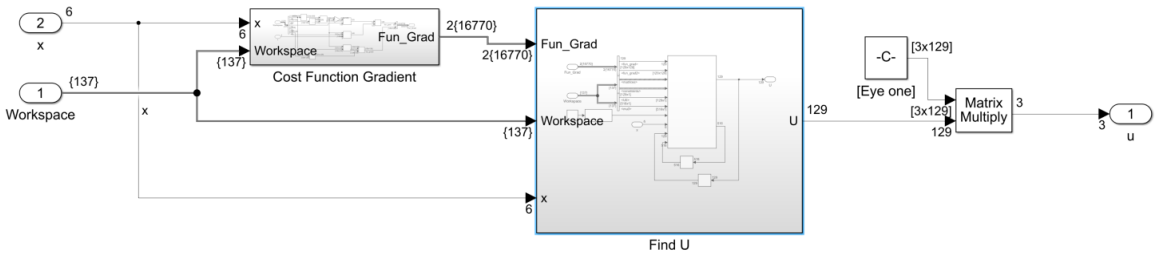
Code to initialize variables to run model:

```
%% Problem 5b
% Impose contraints on the inputs and outputs
close all;
Np = 43;                              % Define the prediction horizon
cnt = 2;                              % cnt = 2 for contrained in and out
[matrices,constraints,r_p,m,x0,int] = Initialize_MPC_non(sys,Np,cnt);
R0 = [.01 0 0;
      0 .001 0;
      0 0 .005];
matrices.R = kron(eye(Np),R0);        % Gain Matrix - Control
Q0 = [20 0 0;
      0 30 0;
      0 0 30];
matrices.Q = kron(eye(Np),Q0);        % Gain Matrix - Input

% Edit initial conditions from 0
x0.theta1 = -55*pi/180;
x0.theta2 = -40*pi/180;

sim('NonAug_MPC_Con.slx',10);         % Run simulation
open_system('NonAug_MPC_Con.slx')     % For publishing later
Graphing(ans)                         % Graph the results
```

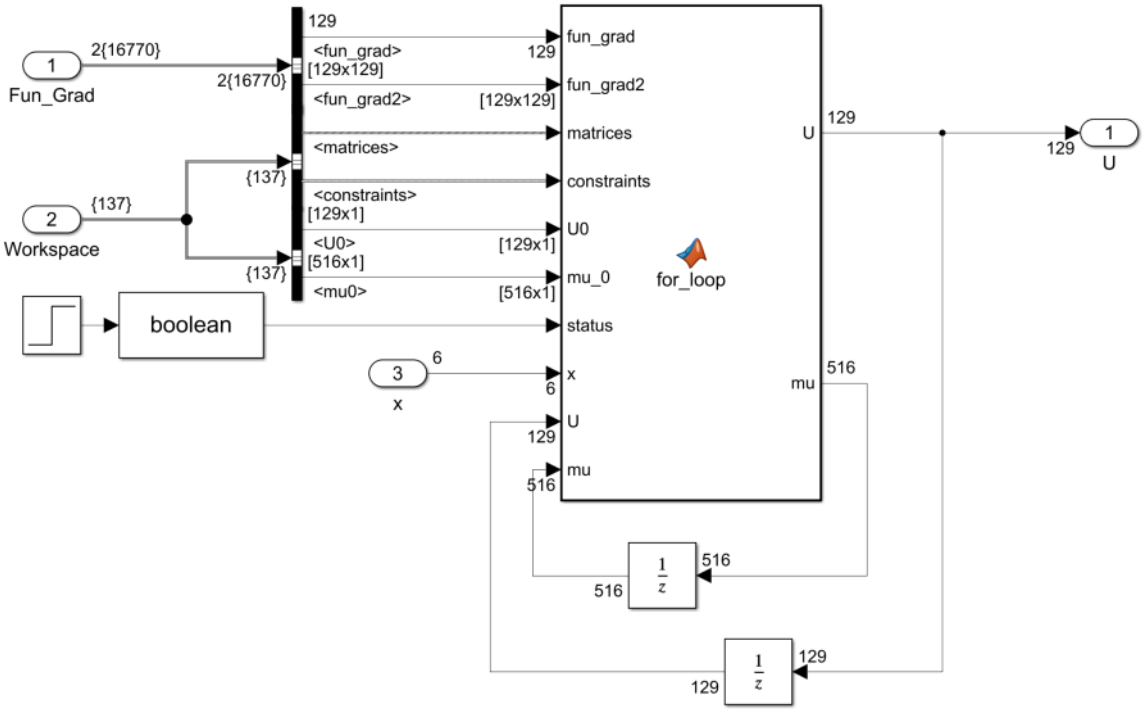
The top-level view of the model is below:



The difference worth noting from the augmented model is the fact that we are no longer accumulating the inputs. What we solve for is the actual u and not the change of u.



The differences worth noting is that it is not x_a feeding into the for-loop, but x. Then just noting we are solving for U instead of ΔU.



The last part is into integrate the observer to the model. The L matrix for this section is the same as the augmented model section. $\eta = \frac{1}{2}$ was the best design variable for the controller. L matrix used:

$$L = \begin{bmatrix} 1.428 & -0.099 & -0.114 \\ -0.106 & 4.150 & -1.528 \\ -0.259 & -1.137 & 3.879 \\ 1.997 & -0.475 & -1.834 \\ 0.295 & 48.804 & -37.478 \\ -1.888 & -34.092 & 43.545 \end{bmatrix}$$

Going onto the changes in the model. The observer is derived from the linear model, and therefore deals with perturbations. Therefore the observer should really be written as:

$$\frac{d}{dt}\delta\tilde{x} = A\delta\tilde{x} + B\delta u + L(\delta y - C\delta\tilde{x})$$

$$\delta\tilde{x} = \tilde{x} - x_e$$

$$\delta y = C(x - x_e)$$

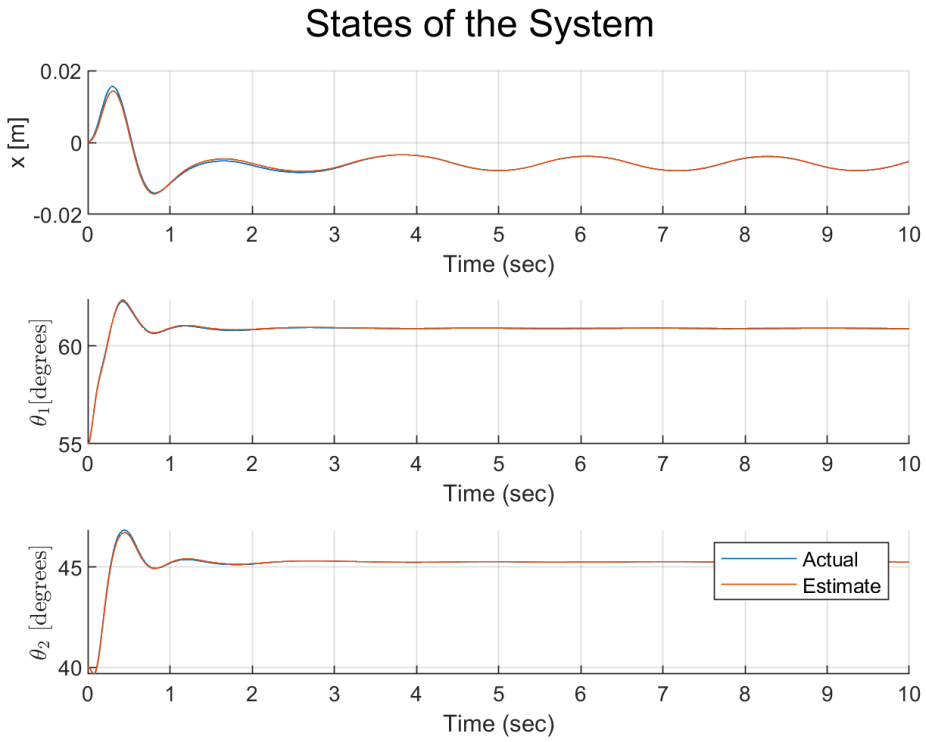
$$\delta u = u - u_e$$

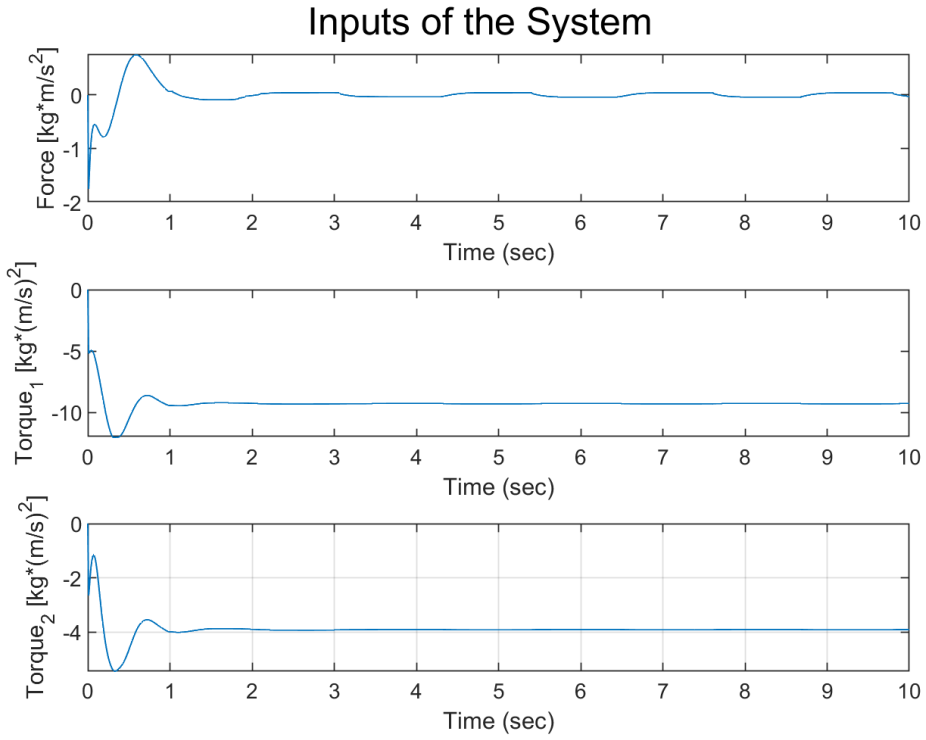
Initial value for the observer was set to equal initial state- just because this controller would pretty easily “blow up". There were no changes to the design parameters from the previous problem. The constraints also remained the same since the last problem.

The Youtube video of the results for the constrained controller-observer compensator can be found here:

`https://youtu.be/fvbCMfStxGE`

Overall, there was not a lot of difference between this and the controller without the observer. They all experienced the same steady-state error, and all settled (or oscillated) around the same point. The estimated state begins to stop becoming noticeable around the three second mark.

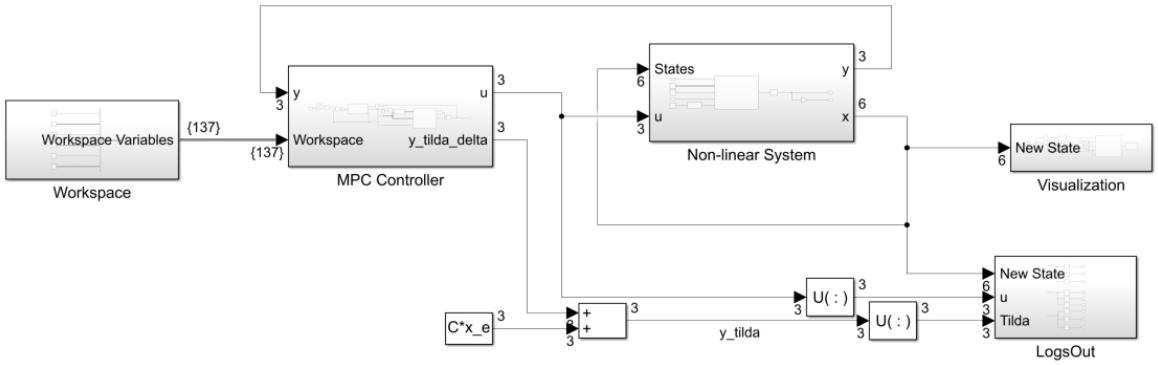

States of the System
x [m]
0.02
0
-0.02
Time (sec)
θ_1 [degrees]
60
55
Time (sec)
θ_2 [degrees]
45
40
Actual
Estimate
Time (sec)


Inputs of the System
Force [kg*m/s^2]
Time (sec)
Torque$_1$ [kg*(m/s)^2]
Time (sec)
Torque$_2$ [kg*(m/s)^2]
Time (sec)

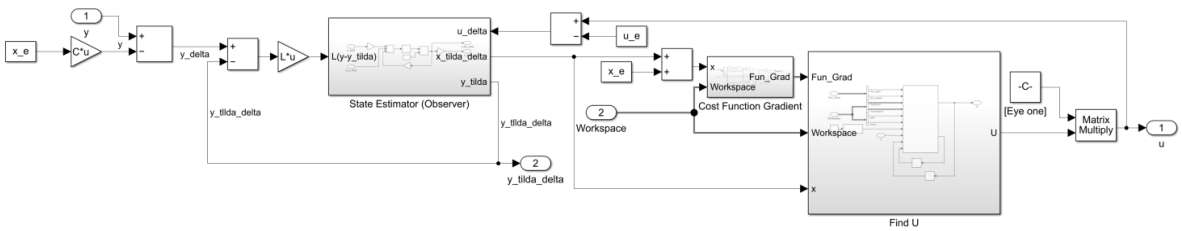
The code to initialize the model:

```
%% Problem 5c
% Combined MPC controller-observer compensator
close all;
m = 3;
n = 6;
p = 3;
cvx_begin sdp quiet

%Variable Definitions
variable P(n,n) symmetric
variable Y(n,p)

% LMIs
eta = 1/2;
P*A + A'*P - Y*C - C'*Y' + eta * P <= 0
P >= eps*eye(n)

cvx_end
L = P^-1*Y                          % Solve for L

sim('NonAug_MPC_CO.slx',10);        % Run simulation
open_system('NonAug_MPC_CO.slx')    % For publishing later
Graphing_Obs(ans)                   % Graph the results
```

Top-level view:



Some changes to outside blocks:



Deltas from Augmented Model Architecture

Reviewing changes to initializing the problem:

```
function [matrices,constraints,r_p,m,x0,int] = Initialize_MPC_non(sys,Np,cnt)
    % Function builds necessary matrices to perform MPS

    % Start Code Here::
    % Determine Sizes Matrices
    [p,~] = size(sys.C);
    [~,m] = size(sys.B);
    [n,~] = size(sys.A);


    % Pull Out Variables from Sys
    phi = sys.A;
    gamma = sys.B;
    C = sys.C;

    % Solve for W and Z
    W = zeros(p*Np,n);              % Initialize W
    Z = zeros(p*Np,m*Np);           % Initialize Z
    index = 1;                      % Initialize var
    % Loop through
    for i = 1:p:p*Np
        W((i:(i+2)),:) = C * phi^index;
        index = index + 1;
    end
    index = 0;                      % Zeroize for next
    for i =1:p:p*Np
        index2 = i;
        index3 = 0;
        while index2 > 0
            Z(i:(i+2),index2:(index2+2)) = C * phi^index3 * gamma;
            index2 = index2 - p;
            index3 = index3 + 1;
        end
        index = index + 1;
    end

    % Initialize Bus
    Bus_Architecture_non(p,m,n,Np,cnt)

    % Define Reference Signal
    v_one = 0;
    v_two = -60*pi/180;
```

```
43      v_three = -45*pi/180;
44      r_p = zeros(3*Np,1);
45      for i = 1:m:m*Np-2
46          r_p(i,1) = v_one;
47          r_p(i+1,1) = v_two;
48          r_p(i+2,1) = v_three;
49      end
50
51     % Constraint Bus Initialize
52      y_min = [-20;-62*pi/180;-47*pi/180];
53      y_max = [20;62*pi/180;47*pi/180];
54      u_min = [-50;-12;-5.9];
55      u_max = [50;12;5.9];
56      constraints.Y_min = kron(ones(Np,1),y_min);
57      constraints.Y_max = kron(ones(Np,1),y_max);
58      constraints.U_min = kron(ones(Np,1),u_min);
59      constraints.U_max = kron(ones(Np,1),u_max);
60
61      % Solve for S and F
62      S_eye = eye(Np);
63      S = kron(S_eye,[-eye(m);eye(m)]);
64      F = zeros(2*m*Np,1);
65      for i = 1:2*m:2*m*Np
66         F((i:i+5),1) = [-u_min;u_max];
67      end
68
69      % Create Matrix Bus
70      matrices.W = W;
71      matrices.Z = Z;
72      matrices.S = S;
73      matrices.F = F;
74      matrices.Z_aug = [-Z;Z];
75
76      % Initialize x0
77      x0.x = 0;
78      x0.theta1 = 0;
79      x0.theta2 = 0;
80      x0.x_dot = 0;
81      x0.theta1_dot = 0;
82      x0.theta2_dot = 0;
83
84      % Initialization Bus
85      int.U0 = zeros(m*Np,1);
86      int.mu0 = zeros(cnt*n*Np,1);
87
88  end
```

Reviewing changes to the bus:

```
function Bus_Architecture_non(p,m,n,Np,cnt)
    % Give Definition to Bus of States
    state_names = {'x','theta1','theta2',...
        'x_dot','theta1_dot','theta2_dot'};
    state_types = {'double','double','double',...
        'double','double','double'};
    for i = 1:length(state_names)
        temp(i) = Simulink.BusElement;
        temp(i).Name = state_names{i};
        temp(i).SampleTime = -1;
        temp(i).Complexity = 'real';
        temp(i).Dimensions = 1;
        temp(i).DataType = state_types{i};
    end
    State_bus = Simulink.Bus;
    State_bus.Elements = temp;
    assignin('base','State_bus',State_bus)
    clear temp state_names state_types

    % Give Definition to Reference Bus
    ref_names = {'one','two','three'};
    ref_types = {'double','double','double'};
    for i = 1:length(ref_names)
        temp(i) = Simulink.BusElement;
        temp(i).Name = ref_names{i};
        temp(i).SampleTime = -1;
        temp(i).Complexity = 'real';
        temp(i).Dimensions = 1;
        temp(i).DataType = ref_types{i};
    end
    Reference_bus = Simulink.Bus;
    Reference_bus.Elements = temp;
    assignin('base','Reference_bus',Reference_bus)
    clear temp ref_names ref_types

    % Give Definition to Contrain Bus
    ref_names = {'Y_min','Y_max','U_min',...
                 'U_max'};
    ref_types = {'double','double','double',...
                 'double'};
    ref_dim = {[Np*p],[Np*p],[Np*m],...
               [Np*m]};
    for i = 1:length(ref_names)
        temp(i) = Simulink.BusElement;
        temp(i).Name = ref_names{i};
```

```
46          temp(i).SampleTime = -1;
47          temp(i).Complexity = 'real';
48          temp(i).Dimensions = ref_dim{i};
49          temp(i).DataType = ref_types{i};
50      end
51      Contraint_bus = Simulink.Bus;
52      Contraint_bus.Elements = temp;
53      assignin('base','Contraint_bus',Contraint_bus)
54      clear temp ref_names ref_types
55
56      % Give Definition to Matrix Bus
57      ref_names = {'W','Z',...
58                   'S','F','Z_aug',...
59                   'R','Q'};
60      ref_types = {'double','double',...
61                   'double','double','double',...
62                   'double','double'};
63      ref_dim = {[p*Np,n],[p*Np,m*Np],...
64                 [2*m*Np,m*Np],[2*Np*m,1],...
65                 [2*p*Np,p*Np],[m*Np,m*Np],[m*Np,m*Np]};
66      for i = 1:length(ref_names)
67          temp(i) = Simulink.BusElement;
68          temp(i).Name = ref_names{i};
69          temp(i).SampleTime = -1;
70          temp(i).Complexity = 'real';
71          temp(i).Dimensions = ref_dim{i};
72          temp(i).DataType = ref_types{i};
73      end
74      Matrix_bus = Simulink.Bus;
75      Matrix_bus.Elements = temp;
76      assignin('base','Matrix_bus',Matrix_bus)
77      clear temp ref_names ref_types
78
79      % Give Definition to Initialization Bus
80      ref_names = {'U0','mu0'};
81      ref_types = {'double','double'};
82      ref_dim = {[p*Np,1],[cnt*n*Np,1]};
83      for i = 1:length(ref_names)
84          temp(i) = Simulink.BusElement;
85          temp(i).Name = ref_names{i};
86          temp(i).SampleTime = -1;
87          temp(i).Complexity = 'real';
88          temp(i).Dimensions = ref_dim{i};
89          temp(i).DataType = ref_types{i};
90      end
91      Initial_bus = Simulink.Bus;
92      Initial_bus.Elements = temp;
```

```
93      assignin('base','Initial_bus',Initial_bus)
94      clear temp ref_names ref_types
95  end
```