FunWork #2

Victoria Nagorski

Lyapunov Stability - Open-Loop

The Lyapunov matrix equation is $A^TP + PA = -Q$. For the system to be asymptotically stable, the matrix must be real, symmetric, and positive definite. Theorem 4.1, pg 155, gives some proofs by contradiction to show that the matrix A must be asymptotically stable for the solution to work.

Setting up the problem, the Q matrix will be defined as $Q = I_6$. We will be using Matlab's built in function 'lyap()'. It is important to note that they define their Lyapunov equation as $AP + PA^T = -Q$. That means to use their equation we simply need to put $lyap(A', Q)$ into Matlab.

What we find is that Matlab cannot solve for the P matrix since the solution either does not exist or is not unique. This would mean that system, and thus matrix A is not asymptotically stable. $\dot{x} = Ax$ would go to 0 as $x \to \infty$ in an asymptotically stable solution.

$$\frac{V(x(t))}{dt} = \dot{V}(t) = \dot{x}^T(x)Tx(t) + x^T(t)Px(t)$$

If the system were asymptotically stable, we could expect $\dot{V}(t)$ to be decreasing towards 0 as well as $x \to \infty$. Not because $\dot{x} = Ax$ does not have a unique solution. Which allows us to look strictly at the $A^TP + PA$ part of the equation. A negative times a positive is negative. To maintain a decreasing rate if A is negative trending, then P must be positive definite and provide a unique solution.

Linear State-Feedback Controller

The purpose of this problem is to solve for the gain vector that could be used to create the closed-loop system. Define the following poles: $p_1 = -2$, $p_2 = -3$, $p_3 = -6$, $p_4 = -7$, $p_5 = -1$, and $p_6 = -8$.

In Funwork 1, we proved that pair (A, B) is reachable. So there exists a solution to place the poles where desired. The poles can be manipulated using the controller form, but we will use Matlab's function 'place()' we get:

$$K = \begin{bmatrix} 4.7134 & -283.4683 & 348.3459 & 10.6893 & -14.9053 & 61.5009 \end{bmatrix}$$

All the eigenvalues of A are now on the left side of the complex plane.

$$A_c = A - BK = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ -3.1423 & 180.8039 & -232.2306 & -7.1262 & 9.9369 & -41.0006 \\ 6.2845 & -312.5577 & 435.0311 & 14.2524 & -19.8738 & 82.0013 \\ 0 & -32.7 & 32.7 & 0 & 0 & 0 \end{bmatrix}$$

Transfer Function of Closed-Loop System

From the previous hw, we know the transfer function can be derived with the following equation:

$$H(s) = \frac{Y(s)}{U(s)} = C(sI - A)^{-1}B$$

We can also use Matlab's function 'ss2tf()' to get the transfer functions. Using the new closed-loop A matrix we get:

$$x : H_1(s) = \frac{0.6667s^4 - 54.5s^2 + 427.716}{s^6 + 27s^5 + 283s^4 + 1449s^3 + 3748s^2 + 4572s + 2016}$$

$$\theta_1 : H_2(s) = \frac{-1.3333s^4 + 43.6s^2}{s^6 + 27s^5 + 283s^4 + 1449s^3 + 3748s^2 + 4572s + 2016}$$

$$\theta_2 : H_3(s) = \frac{43.6s^2}{s^6 + 27s^5 + 283s^4 + 1449s^3 + 3748s^2 + 4572s + 2016}$$

It should be noted that numerator stayed the same, but the denominator changed. This makes sense since it is the denominator of the transfer function form which we pull the characteristic equation from. The characteristic equation defines were the poles of the system are.

Lyapunov Stability - Closed-Loop

In a previous step, we put all the eigenvalues on the left-hand side of the imaginary complex plane. This means we should have an asymptotically stable matrix. To test, we use Matlab's function 'lyap()' to test for a real symmetric positive definite matrix. The P matrix we get:

$$P = \begin{bmatrix} 2.1122 & -1.5874 & 9.8259 & 1.7187 & 0.7798 & 2.3307 \\ -1.5878 & 49.9909 & -84.5715 & -3.8876 & -0.4804 & -16.8878 \\ 9.8259 & -84.5715 & 201.1701 & 17.9226 & 6.3656 & 42.5826 \\ 1.7187 & -3.8876 & 17.9226 & 2.7488 & 1.2187 & 4.1388 \\ 0.7798 & -0.4804 & 6.3656 & 1.2187 & 0.6104 & 1.544 \\ 2.3307 & -16.8878 & 42.5826 & 4.1388 & 1.544 & 9.2094 \end{bmatrix}$$

We note that the matrix is symmetric and real. Then we calculate the principal minors:

$$\begin{vmatrix} 2.1122 \end{vmatrix} = 2.1122$$

$$\begin{vmatrix} 2.1122 & -1.5874 \\ -1.5878 & 49.9909 \end{vmatrix} = 103.0735$$

$$\begin{vmatrix} 2.1122 & -1.5874 & 9.8259 \\ -1.5878 & 49.9909 & -84.5715 \\ 9.8259 & -84.5715 & 201.1701 \end{vmatrix} = 3.4394e+03$$

$$\begin{vmatrix} 2.1122 & -1.5874 & 9.8259 & 1.7187 \\ -1.5878 & 49.9909 & -84.5715 & -3.8876 \\ 9.8259 & -84.5715 & 201.1701 & 17.9226 \\ 1.7187 & -3.8876 & 17.9226 & 2.7488 \end{vmatrix} = 676.6665$$

$$\begin{vmatrix} 2.1122 & -1.5874 & 9.8259 & 1.7187 & 0.7798 \\ -1.5878 & 49.9909 & -84.5715 & -3.8876 & -0.4804 \\ 9.8259 & -84.5715 & 201.1701 & 17.9226 & 6.3656 \\ 1.7187 & -3.8876 & 17.9226 & 2.7488 & 1.2187 \\ 0.7798 & -0.4804 & 6.3656 & 1.2187 & 0.6104 \end{vmatrix} = 23.1826$$

$$\begin{vmatrix} 2.1122 & -1.5874 & 9.8259 & 1.7187 & 0.7798 & 2.3307 \\ -1.5878 & 49.9909 & -84.5715 & -3.8876 & -0.4804 & -16.8878 \\ 9.8259 & -84.5715 & 201.1701 & 17.9226 & 6.3656 & 42.5826 \\ 1.7187 & -3.8876 & 17.9226 & 2.7488 & 1.2187 & 4.1388 \\ 0.7798 & -0.4804 & 6.3656 & 1.2187 & 0.6104 & 1.544 \\ 2.3307 & -16.8878 & 42.5826 & 4.1388 & 1.544 & 9.2094 \end{vmatrix} = 2.0736$$

Since we find that all the principal minors have a positive determinant, we conclude that the P matrix is positive definite. Therefore the system is asymptotically stable.

Added Input to System

To add another input we go back to our original set of equations:
Equation 1:

$$(M+m_1+m_2)\ddot{x}+(m_1+m_2)l_1\ddot{\theta}_1\cos\theta_1+m_2l_2\ddot{\theta}_2\cos\theta_2-(m_1+m_2)l_1\dot{\theta}_1^2\sin\theta_1-m_2l_2\dot{\theta}_2^2\sin\theta_2=F$$

Equation 2:

$$(m_1+m_2)l_1\ddot{x}\cos\theta_1+(m_1+m_2)\ddot{\theta}_1l_1^2+m_2l_1l_2\ddot{\theta}_2\cos(\theta_1-\theta_2)+m_2l_1l_2\dot{\theta}_2^2\sin(\theta_1-\theta_2)$$
$$-m_1l_1g\sin\theta_1-m_2l_1g\sin\theta_1=0$$

Equation 3:

$$m_2l_2\ddot{x}\cos\theta_2+m_2l_1l_2\ddot{\theta}_1\cos(\theta_1-\theta_2)+m_2\ddot{\theta}_2l_2^2-m_2l_1l_2\dot{\theta}_1^2\sin(\theta_1-\theta_2)-m_2gl_2\sin\theta_2=0$$

The right side of the Lagrangian uses generalized forces. The generalized forces were derived using the equation $Q_i=\frac{\partial\dot{W}}{\partial\dot{q}_i}$ - where $\dot{W}$ is derived using the equation $\dot{W}=F\cdot v+\tau\cdot\omega$. Using the equation for Q, we find that $Q_2=\tau$. τ is torque.
We can re-write equation 2 as:

$$(m_1+m_2)l_1\ddot{x}\cos\theta_1+(m_1+m_2)\ddot{\theta}_1l_1^2+m_2l_1l_2\ddot{\theta}_2\cos(\theta_1-\theta_2)+m_2l_1l_2\dot{\theta}_2^2\sin(\theta_1-\theta_2)$$
$$-m_1l_1g\sin\theta_1-m_2l_1g\sin\theta_1=\tau$$

The H matrix will change to become:

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

We can now linearize the model (using the same method as Funwork 1) to now have the following system:

$$A=\begin{bmatrix}0&0&0&1&0&0\\0&0&0&0&1&0\\0&0&0&0&0&1\\0&-8.1759&0&0&0&0\\0&65.4&-29.43&0&0&0\\0&-32.7&32.7&0&0&0\end{bmatrix}$$

$$B=\begin{bmatrix}0&0\\0&0\\0&0\\0.6667&-1.333\\-1.3333&10.6667\\0&-5.333\end{bmatrix}$$

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

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

We can then design a linear state-feedback controller with the linearized model using pole placement. The poles were first found using a trial and error method that worked, but I was given a better set of poles. (Issues with this problem were not due to choosing wrong poles, and I actually had guessed similar poles after lots and lots of trial and error). The dynamics provided by these pole were better since they were derived using a cost function as to minimize deviations, so I decided to keep using them. The cost function was probably:

$$J = \int_0^\infty (x^T Q x + u^T R u) dt$$

Solving this cost function leads to the Algebraic Riccoti Equation (ARE). We end up with two equations of importance:

$$A^T P + PA + Q - PBR^{-1}B^T P = 0$$

$$u^* = -R^{-1}B^T P x = -Kx$$

The weight matrix R would have required some trial and error to solve for P. Trial and error would have been used on the weight matrix R since we do not know where we want to specifically put the poles in the system. However, it would still place them better than when I was using trial and error specifically on just the poles themselves. Once R and P were solved, one could solve for the optimal control law - allowing us to solve for K. Matlab's 'lqr()' or 'care()' commands are available for solving these equations quickly based on given weighted matrices Q and R.
Using the following poles: $p_1 = -1.6720 + 4.6295i$, $p_2 = -1.6720 - 4.6295i$, $p_3 = -1.1206$, $p_4 = -0.7333$, $p_5 = -0.5301 + 1.0487i$, and $p_6 = -0.5301 - 1.0487i$.

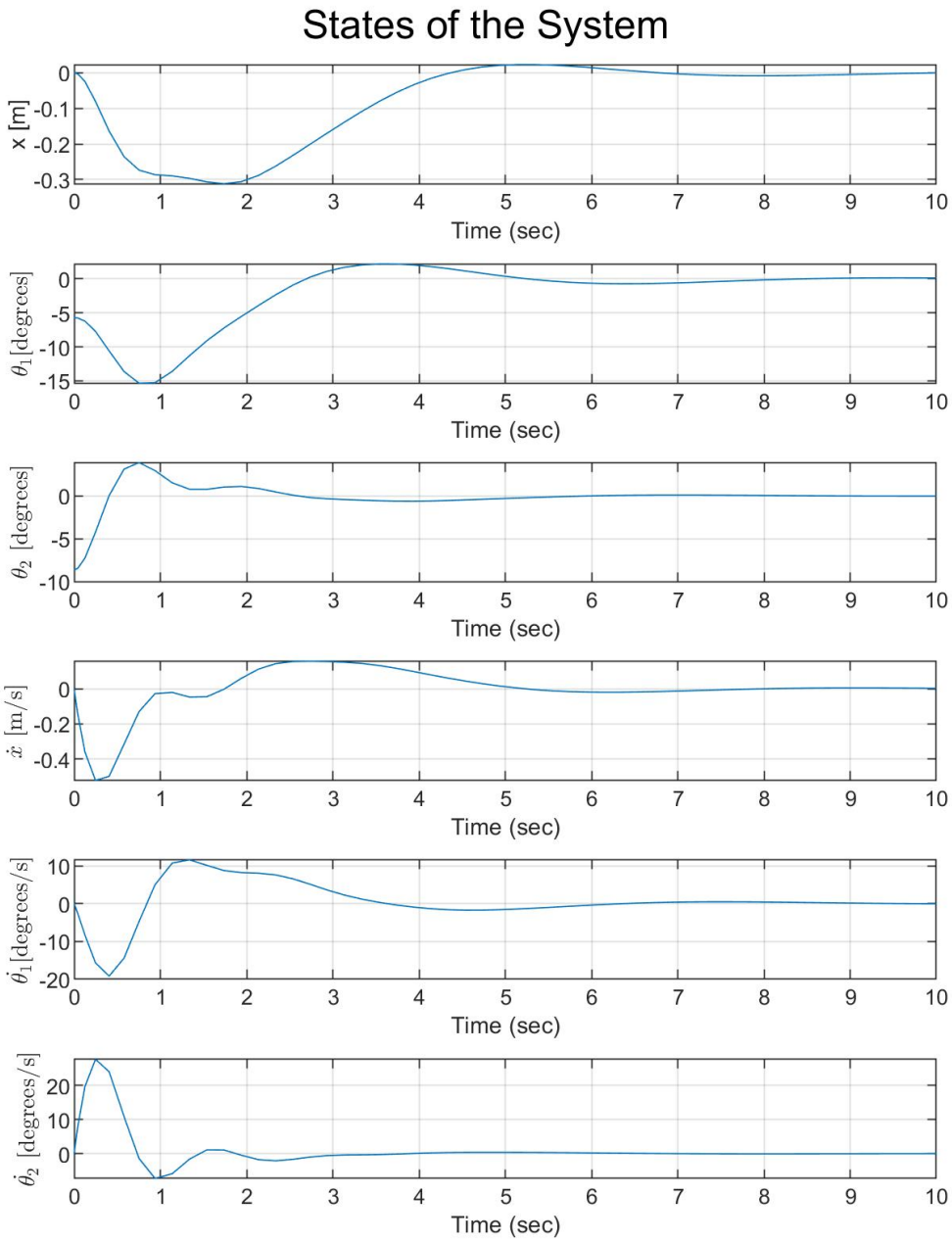
$$K = \begin{bmatrix} -2.681 & 1.114 & -57.416 & -4.791 & -1.082 & -10.644 \\ -0.354 & 6.425 & -10.646 & -0.663 & 0.072 & -1.192 \end{bmatrix}$$

Closed-loop A is:

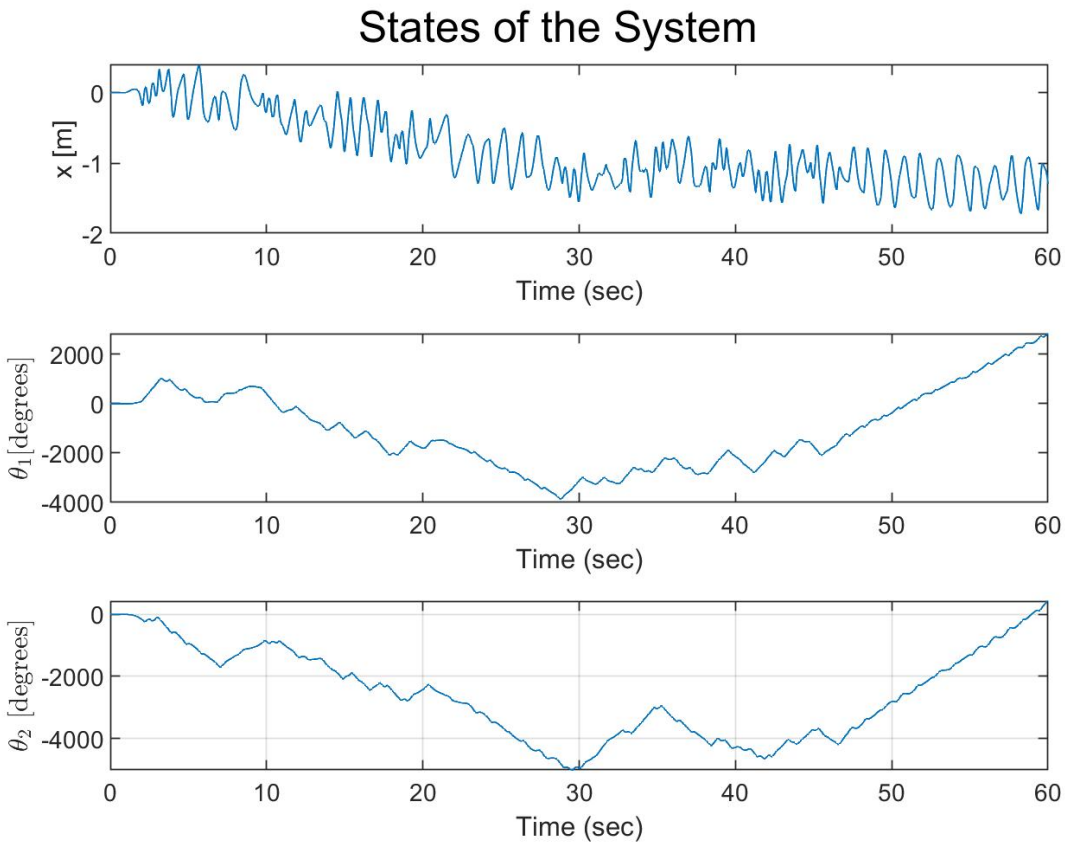
$$A_c = A - BK = \begin{bmatrix} 0 & 0 & 0 & 1.0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1.0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1.0 \\ 1.315 & -0.351 & 24.081 & 2.31 & 0.817 & 5.506 \\ 0.201 & -1.652 & 7.578 & 0.682 & -2.209 & -1.473 \\ -1.887 & 1.569 & -24.081 & -3.535 & 0.383 & -6.359 \end{bmatrix}$$

The next few pages will show the diagrams for linear and non-linear states utilizing controllers.

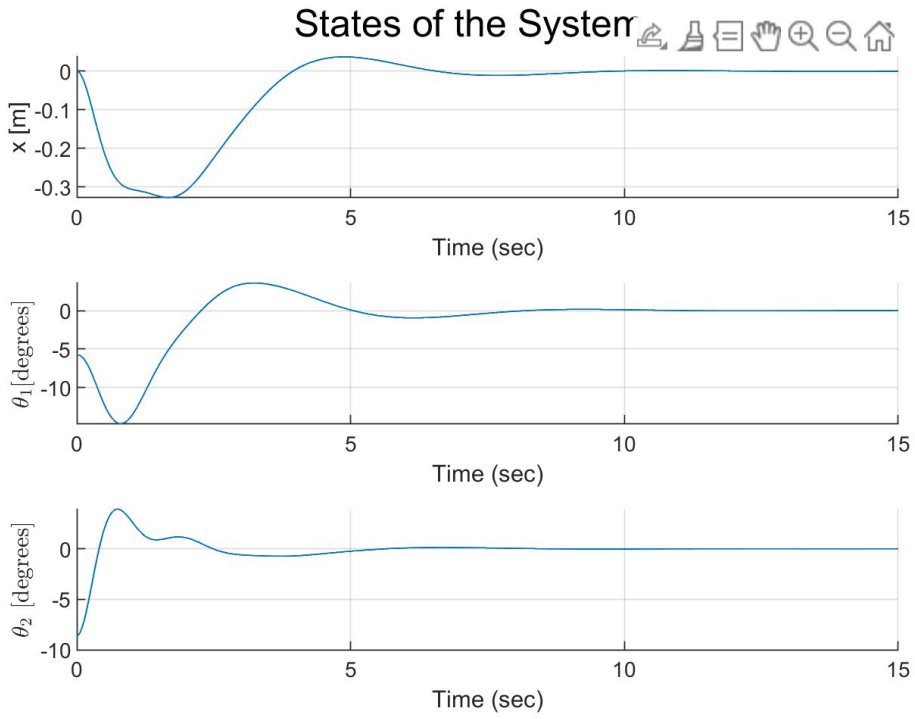
The linear states with controller:

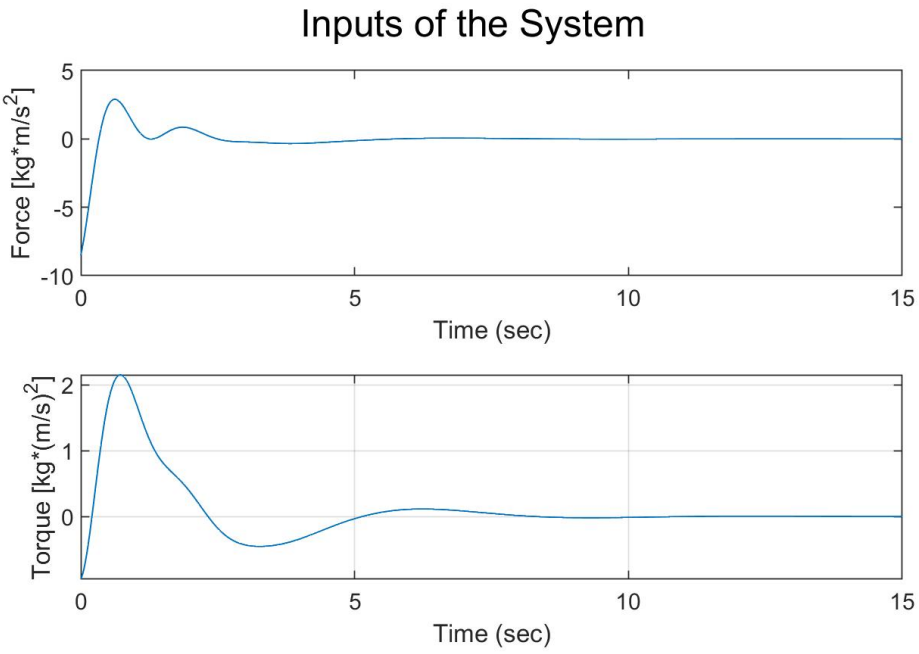


We can compare the open-loop states to show that the closed-loop states perform better.
Old states:



New states:





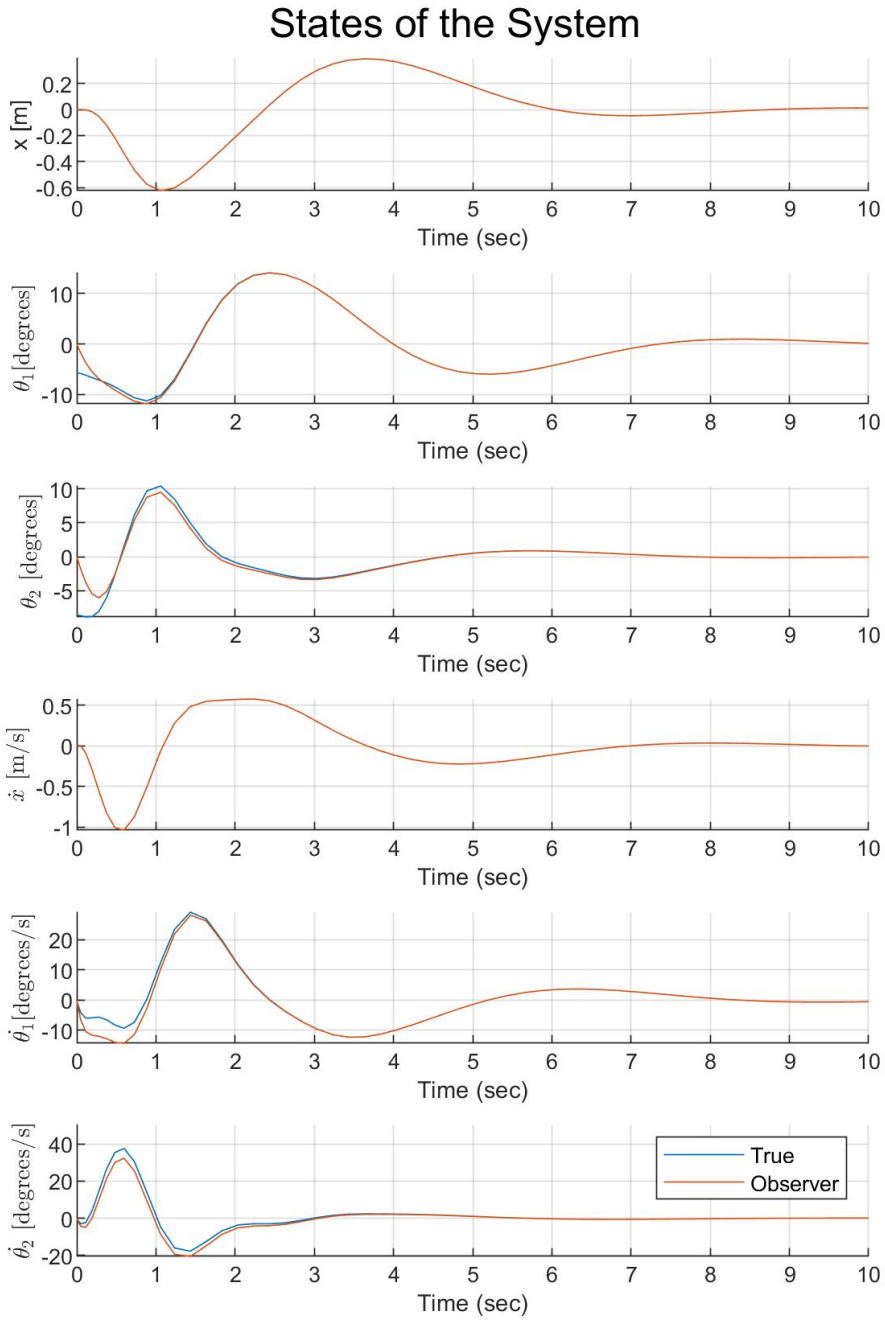
The video for the non-linear dynamics integrated with the gains can be found here:
`https://youtu.be/VrpQKy5UZNg`

Design Luenberger Observer

We first note that the pair (A, C) was observable from Funwork 1. This means that the pair (A^T, C^T) is also reachable. Therefore, there exists a solution a to place the observer poles where we want them. The Luenberger observer can be designed in a similar way to that of the controller by using the observer form. However, the Matlab function 'place()' was used again. Except this time L was solved by doing $L = place(A', C', poles)'$ instead. Used the following poles: $l_1 = real(p_1) * 2$, $l_2 = real(p_2) * 2$, $l_3 = real(p_3) * 2$, $l_4 = real(p_4) * 62$, $l_5 = real(p_5) * 2$, and $l_6 = real(p_6) * 2$. These poles were found through trial and error. The resulting L matrix:

$$L = \begin{bmatrix} 4.4042 & 0 & 0 \\ 0 & 6.641 & 0 \\ 0 & 0 & 4.4042 \\ 3.545 & -8.175 & 0 \\ 0 & 75.261 & -29.43 \\ 0 & -32.7 & 36.245 \end{bmatrix}$$

Then next page shows how well the observer tracks the linear states.


States of the System
x [m]
Time (sec)
θ_1 [degrees]
Time (sec)
θ_2 [degrees]
Time (sec)
$\dot{x}$ [m/s]
Time (sec)
$\dot{\theta}_1$ [degrees/s]
Time (sec)
$\dot{\theta}_2$ [degrees/s]
Time (sec)
True
Observer

Lyapunov Stability for Controller-Observer Compensator

This first part will derive down to the closed-loop system of the controller-observer compensator. Starting with:

$$\dot{x}(t) = Ax(t) + Bu(t)$$

$$y(t) = Cx(t)$$

Then we can define an observer that is a 'duplicate' of the controller:

$$\dot{\tilde{x}}(t) = A\tilde{x}(t) + Bu(t)$$

The error can be defined as (after some manipulation):

$$\dot{e}(t) = A(\tilde{x}(t) - x(t))$$

However, $\dot{\tilde{x}}(t)$ cannot be controlled as is. So we modify to:

$$\dot{\tilde{x}}(t) = A\tilde{x}(t) + Bu(t) + L(y(t) - \tilde{y}(t))$$

where $\tilde{y}(t) = C\tilde{x}(t)$. We can then begin to re-write $\dot{\tilde{x}}(t)$:

$$\dot{\tilde{x}}(t) = A\tilde{x}(t) + Bu(t) + L(Cx(t) - C\tilde{x}(t))$$

$$\dot{\tilde{x}}(t) = A\tilde{x}(t) + Bu(t) + LC(x(t) - \tilde{x}(t)) = (A - LC)\tilde{x}(t) + LCx(t) + Bu(t)$$

With the equations $\dot{\tilde{x}}(t)$ and $\dot{x}(t)$, we can begin to write in matrix form:

$$\begin{bmatrix} \dot{x}(t) \\ \dot{\tilde{x}}(t) \end{bmatrix} = \begin{bmatrix} A & 0 \\ LC & A - LC \end{bmatrix} \begin{bmatrix} x(t) \\ \tilde{x}(t) \end{bmatrix} + \begin{bmatrix} B \\ B \end{bmatrix} u(t)$$

$$y(t) = \begin{bmatrix} C & 0 \end{bmatrix} \begin{bmatrix} x(t) \\ \tilde{x}(t) \end{bmatrix}$$

$$u(t) = -K\tilde{x}(t) + v(t)$$

Rewrite:

$$\begin{bmatrix} \dot{x}(t) \\ \dot{\tilde{x}}(t) \end{bmatrix} = \begin{bmatrix} A & 0 \\ LC & A - LC \end{bmatrix} \begin{bmatrix} x(t) \\ \tilde{x}(t) \end{bmatrix} + \begin{bmatrix} B \\ B \end{bmatrix} (-K\tilde{x}) + \begin{bmatrix} B \\ B \end{bmatrix} v(t)$$

$$\begin{bmatrix} \dot{x}(t) \\ \dot{\tilde{x}}(t) \end{bmatrix} = \begin{bmatrix} A & -BK \\ LC & A - LC - BK \end{bmatrix} \begin{bmatrix} x(t) \\ \tilde{x}(t) \end{bmatrix} + \begin{bmatrix} B \\ B \end{bmatrix} v(t)$$

$$y(t) = \begin{bmatrix} C & 0 \end{bmatrix} \begin{bmatrix} x(t) \\ \tilde{x}(t) \end{bmatrix}$$

We now have our closed-loop system that we can take to Matlab. Define A and Q:

$$A_{co} = \begin{bmatrix} A & -BK \\ LC & A - LC - BK \end{bmatrix}$$

The closed-loop matrix was too big for Latex to handle. So refer to Matlab publish for matrix.

$$Q = I_6$$

P is also a 12x12 matrix and cannot be handled by Latex. Refer to Matlab's publish for matrix. We will solve for the principal minors after noting P is symmetric and real.

- $1st = 6.4$
- $2nd = 561.001$
- $3rd = 5.217e + 04$
- $4th = 3.205e + 05$
- $5th = 1.088e + 07$
- $6th = 1.451e + 08$
- $7th = 3.291e + 07$
- $8th = 5.932e + 07$
- $9th = 5.936e + 08$
- $10th = 1.293e + 08$
- $11th = 1.713e + 07$
- $12th = 2.877e + 06$

It is to be noted that all of the principal minors are greater than zero. Therefore, we can conclude that P is positive definite, and therefore the closed-loop system is asymptotically stable.

TF of System with Controller-Observer Compensator

Starting with the previously found closed-loop system;

$$\begin{bmatrix}\dot{x}(t)\\ \dot{\tilde{x}}(t)\end{bmatrix} = \begin{bmatrix}A & -BK\\ LC & A-LC-BK\end{bmatrix}\begin{bmatrix}x(t)\\ \tilde{x}(t)\end{bmatrix} + \begin{bmatrix}B\\ B\end{bmatrix} v(t)$$

$$y(t) = \begin{bmatrix}C & 0\end{bmatrix}\begin{bmatrix}x(t)\\ \tilde{x}(t)\end{bmatrix}$$

We can do a change of variable to make it easier to work with the system to solve for the transfer function. Define the following:

$$\begin{bmatrix}x(t)\\ \tilde{x}(t)-x(t)\end{bmatrix} = \begin{bmatrix}I_n & 0\\ -I_n & I_n\end{bmatrix}\begin{bmatrix}x(t)\\ \tilde{x}(t)\end{bmatrix}$$

Then:

$$\dot{\tilde{x}}(t) - \dot{x}(t) = LCx(t) + A\tilde{x}(t) - LC\tilde{x}(t) - BK\tilde{x}(t) - Ax(t) + BK\tilde{x}(t) + Bv(t) - Bv(t)$$

$$\dot{\tilde{x}}(t) - \dot{x}(t) = A(\tilde{x}(t) - x(t)) - LC(\tilde{x}(t) - x(t))$$

And:

$$\dot{x}(t) = Ax(t) - BK\tilde{x}(t) - BKx(t) + BKx(t)$$

$$\dot{x}(t) = (A - BK)x(t) - BK(\tilde{x}(t) - x(t))$$

Then:

$$\begin{bmatrix}\dot{x}(t)\\ \dot{\tilde{x}}(t)-\dot{x}(t)\end{bmatrix} = \begin{bmatrix}A-BK & -BK\\ 0 & A-LC\end{bmatrix}\begin{bmatrix}x(t)\\ \tilde{x}(t)-x(t)\end{bmatrix} + \begin{bmatrix}B\\ 0\end{bmatrix} v(t)$$

$$y(t) = \begin{bmatrix}C & 0\end{bmatrix}\begin{bmatrix}x(t)\\ \tilde{x}(t)-x(t)\end{bmatrix}$$

From the previous homework, we know:

$$H(s) = \frac{Y(s)}{U(s)} = C(sI - A)^{-1}B$$

We are going to do some re-writing:

$$Y(s) = \begin{bmatrix}C & 0\end{bmatrix}\begin{bmatrix}sI - A + BK & -BK\\ 0 & sI - A + LC\end{bmatrix}^{-1}\begin{bmatrix}B\\ 0\end{bmatrix} V(s)$$

$$Y(s) = \begin{bmatrix}C & 0\end{bmatrix}\begin{bmatrix}sI - A + BK\\ 0\end{bmatrix}^{-1} BV(s)$$

$$Y(s) = C(sI - A + BK)^{-1}BV(s)$$

Which then we can use this equation to solve for the transfer function in Matlab. However, I used 'ss2tf()' since it is easier to use. It should be noted the characteristic equation in the denominator does not change with the input, and that the observer is not visible from the input-output view of the transfer function.

For input 1:

$$H_1(s) = \frac{Y_1(s)}{U_1(s)} = \frac{s^{10} + 15s^9 + 152s^8 + 902s^7 + 3253s^6 + 6874s^5 + 6951^4 - 974s^3 - 9651s^2 - 8652s - 2543}{20s^{11} + 230s^{10} + 1560s^9 + 7410s^8 + 24800s^7 + 58930s^6 + 100090s^5 + 121970s^4 + 105270s^3 + 61330s^2 + 21530s + 3140}$$

$$H_2(s) = \frac{Y_2(s)}{U_1(s)} = \frac{-s^{10} - 26s^9 - 233s^8 - 1374s^7 - 5700s^6 - 16552s^5 - 32574s^4 - 41944s^3 - 33719s^2 - 15393s - 3060}{20s^{11} + 230s^{10} + 1560s^9 + 7410s^8 + 24800s^7 + 58930s^6 + 100090s^5 + 121970s^4 + 105270s^3 + 61330s^2 + 21530s + 3140}$$

$$H_3(s) = \frac{Y_3(s)}{U_1(s)} = \frac{-3s^9 - 48s^8 - 324s^7 - 1161s^6 - 2365s^5 - 2734^4 - 1667s^3 - 415s^2}{20s^{11} + 230s^{10} + 1560s^9 + 7410s^8 + 24800s^7 + 58930s^6 + 100090s^5 + 121970s^4 + 105270s^3 + 61330s^2 + 21530s + 3140}$$

For input 2:

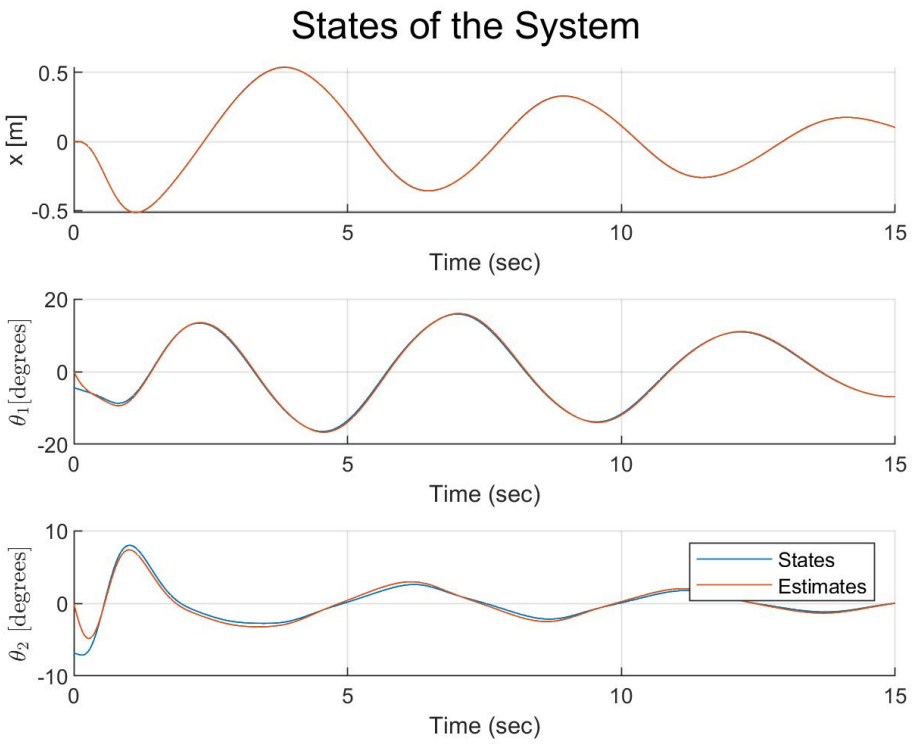
$$H_4(s) = \frac{Y_1(s)}{U_2(s)} = \frac{-50s^9 - 790s^8 - 6080s^7 - 26950s^6 - 70860s^5 - 107630s^4 - 82590s^3 - 12770s^2 + 21070s + 9630}{20s^{11} + 230s^{10} + 1560s^9 + 7410s^8 + 24800s^7 + 58930s^6 + 100090s^5 + 121970s^4 + 105270s^3 + 61330s^2 + 21530s + 3140}$$

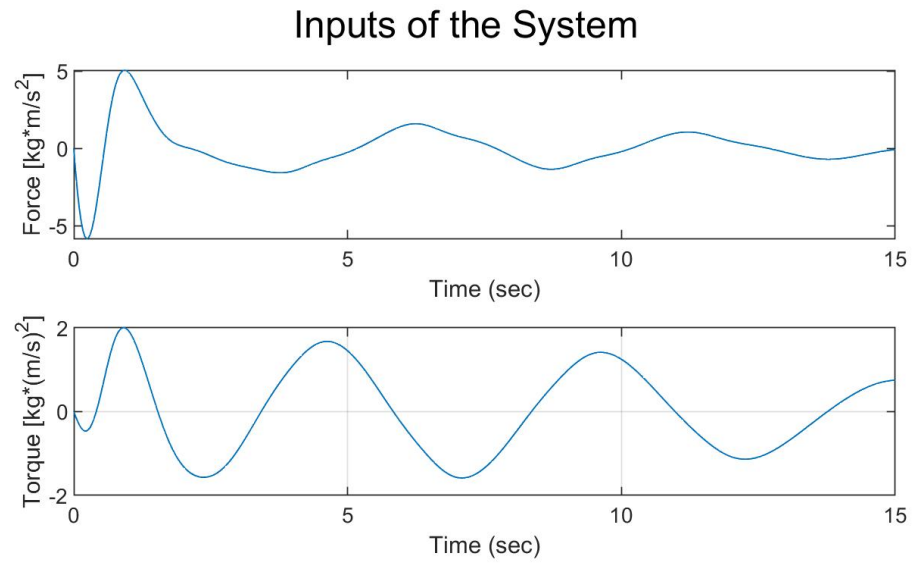
$$H_5(s) = \frac{Y_2(s)}{U_2(s)} = \frac{10s^{10} + 210s^9 + 1990s^8 + 11350s^7 + 44400s^6 + 122320s^5 + 234200s^4 + 301110s^3 + 245840s^2 + 114610s + 23180}{20s^{11} + 230s^{10} + 1560s^9 + 7410s^8 + 24800s^7 + 58930s^6 + 100090s^5 + 121970s^4 + 105270s^3 + 61330s^2 + 21530s + 3140}$$

$$H_6(s) = \frac{Y_3(s)}{U_2(s)} = \frac{-10s^{10} - 70s^9 - 340s^8 - 410s^7 + 1840s^6 + 7540s^5 + 11390s^4 + 7990s^3 + 2160s^2}{20s^{11} + 230s^{10} + 1560s^9 + 7410s^8 + 24800s^7 + 58930s^6 + 100090s^5 + 121970s^4 + 105270s^3 + 61330s^2 + 21530s + 3140}$$

Animate the System

The final video for this animation can be found at: `https://youtu.be/xXaQXk__SGU`





Contents


```
% Victoria Nagorski - ECE 680
% Version 1.0 - 9/12/2021
% FunWork 2
```

Start Script

Load Variable From Previous Part

```
clear; close all; clc;
load('Values.mat');
```

Problem 1

Solve for Lyapunov matrix for linearized open-loop

```
try                % Still publish despite error
Q = eye(6);        % Define
P = lyap(A',Q);
end
```

Problem 2

Design a linear state-feedback controller

```
p1 = -2;
p2 = -3;
p3 = -6;
p4 = -7;
p5 = -1;
p6 = -8;
K = place(A,B,[p1,p2,p3,p4,p5,p6])
A_c = A - B*K
eig(A_c)
```

```
K =

    4.7134 -283.4683  348.3459   10.6893  -14.9053   61.5009


A_c =

         0         0         0    1.0000         0         0
         0         0         0         0    1.0000         0
         0         0         0         0         0    1.0000
   -3.1423  180.8039 -232.2306   -7.1262    9.9369  -41.0006
    6.2845 -312.5577  435.0311   14.2524  -19.8738   82.0013
         0  -32.7000   32.7000         0         0         0


ans =

   -8.0000
   -7.0000
   -6.0000
   -3.0000
   -2.0000
   -1.0000
```

Problem 3

Transfer function of the closed-loop system

```
[b,a] = ss2tf(A_c,B,C,D)
```

```
b =

         0         0    0.6667   -0.0000  -54.5000    0.0000  427.7160
         0         0   -1.3333   -0.0000   43.6000         0         0
         0         0         0         0   43.6000         0         0


a =

   1.0e+03 *

    0.0010    0.0270    0.2830    1.4490    3.7480    4.5720    2.0160
```

Problem 4

Solve for Lyapunov matrix for linearized closed-loop

```
P = lyap(A_c',Q)

% Check principal matrices
princ1 = det(P(1,1))
princ2 = det(P(1:2,1:2))
princ3 = det(P(1:3,1:3))
princ4 = det(P(1:4,1:4))
princ5 = det(P(1:5,1:5))
princ6 = det(P)
```

```
P =

    2.1122   -1.5874    9.8259    1.7187    0.7798    2.3307
   -1.5874   49.9909  -84.5715   -3.8876   -0.4804  -16.8878
    9.8259  -84.5715  201.1701   17.9226    6.3656   42.5826
    1.7187   -3.8876   17.9226    2.7488    1.2187    4.1388
    0.7798   -0.4804    6.3656    1.2187    0.6104    1.5440
    2.3307  -16.8878   42.5826    4.1388    1.5440    9.2094


princ1 =

    2.1122


princ2 =

  103.0735


princ3 =

   3.4394e+03


princ4 =

  676.6665


princ5 =

   23.1826


princ6 =

    2.0736
```

Problem 5

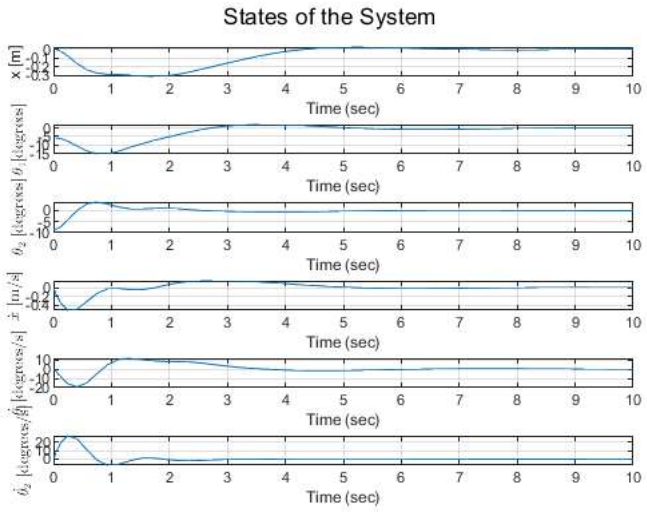
Add Torque to Equations Problem contants

```
m1 = 0.5;                          % kg
l1 = 0.5;                          % m
m2 = 0.75;                         % kg
l2 = 0.75;                         % m
M = 1.5;                           % kg
g = 9.81;                          % m/sec^2

% Solve for matrix contants
r1 = M + m1 + m2;
r2 = (m1 + m2) * l1;
```

```
r3 = m2*l2;
r4 = (m1 + m2) * l1^2;
r5 = m2 * l1 * l2;
r6 = m2 * l2^2;
f1 = (m1 * l1 + m2 * l1) * g;
f2 = m2 * l2 * g;

% Define Non-Linear Matrices
syms x1 x2 x3 x4 x5 x6 u1 u2
M = [  r1        r2*cos(x2)     r3*cos(x3);
    r2*cos(x2)     r4         r5*cos(x2-x3);
    r3*cos(x3) r5*cos(x2-x3)       r6];
C = [0 -r2*x5*sin(x2)    -r3*x6*sin(x3);
     0       0            r5*x6*sin(x2-x3);
     0 -r5*x5*sin(x2-x3)       0];
G = [      0;
    -f1 * sin(x2);
    -f2 * sin(x3)];
H = [1 0;0 1;0 0];

% Solve for the Non-Linear Functions
matrix1 = [zeros(3,3)   eye(3);     % Break up equation
           zeros(3,3) -M^-1 * C];
matrix2 = [zeros(3,1); -M^-1 * G];  % Break up equation
matrix3 = [zeros(3,2); M^-1 * H];   % Break up equation
x = [x1; x2; x3; x4; x5; x6];       % Define x vector
u = [u1; u2];                       % Define state vector
f = simplify(matrix1 * x + matrix2 + matrix3 * u);

% Linearize with the Jacobian
equil = [0;0;0;0;0;0;0;0];          % Equilibrium point (about orgin)
a = jacobian(f,x);                  % Create Jacobian matrix for A
b = jacobian(f,u);                  % Create Jacobian matrix for B
A = double(subs(a,[x;u],equil))     % Plug in equilibrium points
B = double(subs(b,[x;u],equil))     % Plug in equilibrium points
C = [1 0 0 0 0 0;                   % Define the C matrix for Sys
     0 1 0 0 0 0;
     0 0 1 0 0 0]
D = zeros(3,2)                      % Define the D matrix for Sys

% Design a linear state-feedback controller
p1 = -1.6720+4.6295i;
p2 = -1.6720-4.6295i;
p3 = -1.1206;
p4 = -0.7333;
p5 = -0.5301 + 1.0487i;
p6 = -0.5301-1.0487i;
K = place(A,B,[p1,p2,p3,p4,p5,p6])
A_c = A - B*K
sys_s = ss(A_c,B,C,D);
damp(sys_s)

% Simulate the Controller
% Initial States
x0.x = 0;
x0.theta1 = -.1;
x0.theta2 = -.15;
x0.x_dot = 0;
x0.theta1_dot = 0;
x0.theta2_dot = 0;

% Reference Input
v.one = 0;
v.two = 0;
sim('Linear_Controller_Design.slx',10);

% Pull out Data
data = ans.logsout{1}.Values.Data;
time = ans.logsout{1}.Values.Time';

% Plot State vs Time
figure
hold on
sgtitle('States of the System')
subplot(6,1,1)
plot(time,data(:,1)')
xlabel('Time (sec)')
ylabel('x [m]')
grid
subplot(6,1,2)
plot(time,data(:,2)'*180/pi)
xlabel('Time (sec)')
ylabel('$\theta_1$[degrees]','Interpreter','latex')
grid
subplot(6,1,3)
plot(time,data(:,3)'*180/pi)
xlabel('Time (sec)')
ylabel('$\theta_2$ [degrees]','Interpreter','latex')
grid
subplot(6,1,4)
plot(time,data(:,4)')
```

```
xlabel('Time (sec)')
ylabel('$\dot{x}$ [m/s]','Interpreter','latex')
grid
subplot(6,1,5)
plot(time,data(:,5)'*180/pi)
xlabel('Time (sec)')
ylabel('$\dot{\theta_1}$[degrees/s]','Interpreter','latex')
grid
subplot(6,1,6)
plot(time,data(:,6)'*180/pi)
xlabel('Time (sec)')
ylabel('$\dot{\theta_2}$ [degrees/s]','Interpreter','latex')
grid
hold off

% Save Matrices for Easy Access
save('Values_New.mat','A','B','C','D','f','x','K','A_c')
```

```
A =

         0         0         0    1.0000         0         0
         0         0         0         0    1.0000         0
         0         0         0         0         0    1.0000
         0   -8.1750         0         0         0         0
         0   65.4000  -29.4300         0         0         0
         0  -32.7000   32.7000         0         0         0


B =

         0         0
         0         0
         0         0
    0.6667   -1.3333
   -1.3333   10.6667
         0   -5.3333


C =

     1     0     0     0     0     0
     0     1     0     0     0     0
     0     0     1     0     0     0


D =

     0     0
     0     0
     0     0


K =

   -2.6806    1.1142  -57.4157   -4.7906   -1.0815  -10.6440
   -0.3539    6.4254  -10.6465   -0.6627    0.0719   -1.1924


A_c =

         0         0         0    1.0000         0         0
         0         0         0         0    1.0000         0
         0         0         0         0         0    1.0000
    1.3152   -0.3507   24.0819    2.3101    0.8169    5.5062
    0.2008   -1.6515    7.5779    0.6815   -2.2089   -1.4733
   -1.8874    1.5686  -24.0811   -3.5345    0.3834   -6.3593


         Pole              Damping       Frequency      Time Constant
                                      (rad/seconds)      (seconds)

 -7.33e-01                1.00e+00       7.33e-01         1.36e+00
 -1.12e+00                1.00e+00       1.12e+00         8.92e-01
 -5.30e-01 + 1.05e+00i    4.51e-01       1.18e+00         1.89e+00
 -5.30e-01 - 1.05e+00i    4.51e-01       1.18e+00         1.89e+00
 -1.67e+00 + 4.63e+00i    3.40e-01       4.92e+00         5.98e-01
 -1.67e+00 - 4.63e+00i    3.40e-01       4.92e+00         5.98e-01
```



Problem 6

Construct the Observer

```
close all;
% Construct Observer Poles
l1 = real(p1) * 2;
l2 = real(p2) * 2;
l3 = real(p3) * 2;
l4 = real(p4) * 6;
l5 = real(p5) * 2;
l6 = real(p6) * 2;
L = place(A',C',[l1,l2,l3,l4,l5,l6])'

sim('Linear_Control_Observer.slx',10)
% Pull out Data
data = ans.logsout{1}.Values.Data;
data2 = ans.logsout{2}.Values.Data;
time = ans.logsout{1}.Values.Time';

% Plot State vs Time
figure
hold on
sgtitle('States of the System')
subplot(6,1,1)
hold on
plot(time,data(:,1)')
plot(time,data2(:,1)')
hold off
xlabel('Time (sec)')
ylabel('x [m]')
grid
subplot(6,1,2)
hold on
plot(time,data(:,2)'*180/pi)
plot(time,data2(:,2)'*180/pi)
hold off
xlabel('Time (sec)')
ylabel('$\theta_1$[degrees]','Interpreter','latex')
grid
subplot(6,1,3)
hold on
plot(time,data(:,3)'*180/pi)
plot(time,data2(:,3)'*180/pi)
hold off
xlabel('Time (sec)')
ylabel('$\theta_2$ [degrees]','Interpreter','latex')
grid
subplot(6,1,4)
hold on
plot(time,data(:,4)')
plot(time,data2(:,4)')
hold off
xlabel('Time (sec)')
ylabel('$\dot{x}$ [m/s]','Interpreter','latex')
grid
subplot(6,1,5)
hold on
plot(time,data(:,5)'*180/pi)
plot(time,data2(:,5)'*180/pi)
hold off
xlabel('Time (sec)')
ylabel('$\dot{\theta_1}$[degrees/s]','Interpreter','latex')
```

```
grid
subplot(6,1,6)
hold on
plot(time,data(:,6)'*180/pi)
plot(time,data2(:,6)'*180/pi)
hold off
xlabel('Time (sec)')
ylabel('$\dot{\theta_2}$ [degrees/s]','Interpreter','latex')
grid
legend('True','Observer')
hold off

close all;
% Save Matrices for Easy Access
save('Values_New.mat','A','B','C','D','f','x','K','A_c','L')
```

```
L =

    4.4042         0         0
         0    6.6410         0
         0         0    4.4042
    3.5453   -8.1750         0
         0   75.2608  -29.4300
         0  -32.7000   36.2453


ans =

  Simulink.SimulationOutput:
               logsout: [1x1 Simulink.SimulationData.Dataset]
                  tout: [60x1 double]
                  yout: [1x1 Simulink.SimulationData.Dataset]

     SimulationMetadata: [1x1 Simulink.SimulationMetadata]
           ErrorMessage: [0x0 char]
```

Problem 7

Solve the Lyapunov Equation for closed-loop

```
A_co = [A    -B*K;
      L*C A-L*C-B*K]            % Closed-loop matrix
Q = eye(12);                    % Define Q matrix
P = lyap(A_co',Q)
one = det(P(1,1))
two = det(P(1:2,1:2))
three = det(P(1:3,1:3))
four = det(P(1:4,1:4))
five = det(P(1:5,1:5))
six = det(P(1:6,1:6))
seven = det(P(1:7,1:7))
eight = det(P(1:8,1:8))
nine = det(P(1:9,1:9))
ten = det(P(1:10,1:10))
eleven = det(P(1:11,1:11))
twelve = det(P)
```

```
A_co =

  Columns 1 through 7

         0         0         0    1.0000         0         0         0
         0         0         0         0    1.0000         0         0
         0         0         0         0         0    1.0000         0
         0   -8.1750         0         0         0         0    1.3152
         0   65.4000  -29.4300         0         0         0    0.2008
         0  -32.7000   32.7000         0         0         0   -1.8874
    4.4042         0         0         0         0         0   -4.4042
         0    6.6410         0         0         0         0         0
         0         0    4.4042         0         0         0         0
    3.5453   -8.1750         0         0         0         0   -2.2301
         0   75.2608  -29.4300         0         0         0    0.2008
         0  -32.7000   36.2453         0         0         0   -1.8874

  Columns 8 through 12

         0         0         0         0         0
         0         0         0         0         0
         0         0         0         0         0
    7.8243   24.0819    2.3101    0.8169    5.5062
  -67.0515   37.0079    0.6815   -2.2089   -1.4733
   34.2686  -56.7811   -3.5345    0.3834   -6.3593
         0         0    1.0000         0         0
   -6.6410         0         0    1.0000         0
         0   -4.4042         0         0    1.0000
    7.8243   24.0819    2.3101    0.8169    5.5062
```

```
  -76.9124   37.0079    0.6815   -2.2089   -1.4733
   34.2686  -60.3264   -3.5345    0.3834   -6.3593


P =

  Columns 1 through 7

    6.4006   -4.3698   26.1427   -0.5000  -10.5080   -9.7292   -2.7248
   -4.3698   90.6320  -28.2650   10.5080   -0.5000   55.5655    4.3797
   26.1427  -28.2650  201.0181    9.7292  -55.5655   -0.5000  -13.3123
   -0.5000   10.5080    9.7292    9.1718    2.1558   32.2496    3.1246
  -10.5080   -0.5000  -55.5655    2.1558   56.6141   12.2993    6.5061
   -9.7292   55.5655   -0.5000   32.2496   12.2993  140.3674   13.6003
   -2.7248    4.3797  -13.3123    3.1246    6.5061   13.6003    2.8529
    3.8168  -81.1556   30.4210   -7.3532    4.8743  -43.2613   -3.3524
  -21.8844   42.3917 -158.9206    3.9163   66.6610   38.9505   14.9478
    3.2439   -8.4396    7.7884   -5.6870   -6.8497  -24.2691   -3.4314
   10.2599    8.9743   59.3746    1.2718  -52.5727    0.3252   -5.5496
   12.0911  -52.8943   22.8407  -27.5092  -15.1349 -126.6657  -13.6589

  Columns 8 through 12

    3.8168  -21.8844    3.2439   10.2599   12.0911
  -81.1556   42.3917   -8.4396    8.9743  -52.8943
   30.4210 -158.9206    7.7884   59.3746   22.8407
   -7.3532    3.9163   -5.6870    1.2718  -27.5092
    4.8743   66.6610   -6.8497  -52.5727  -15.1349
  -43.2613   38.9505  -24.2691    0.3252 -126.6657
   -3.3524   14.9478   -3.4314   -5.5496  -13.6589
   75.6040  -38.2734    6.2743  -11.2912   42.3399
  -38.2734  162.2507  -11.8851  -62.7200  -49.2618
    6.2743  -11.8851    5.6458    4.7753   23.0128
  -11.2912  -62.7200    4.7753   50.7851    4.6380
   42.3399  -49.2618   23.0128    4.6380  117.5367


one =

    6.4006


two =

  561.0016


three =

   5.2174e+04


four =

   3.2046e+05


five =

   1.0881e+07


six =

   1.4510e+08


seven =

   3.2907e+07


eight =

   5.9315e+07


nine =

   5.9363e+08


ten =

   1.2931e+08


eleven =

   1.7126e+07
```

```
twelve =

   2.8772e+06
```

Problem 8

Transfer function of closed-loop

```
syms 's'
Y = C * (s*eye(6) - A + B*K)^-1 * B
% Method 2
[b1,a1] = ss2tf(A_co,[B;B],[C zeros(3,6)],D,1)      % Input 1
[b2,a2] = ss2tf(A_co,[B;B],[C zeros(3,6)],D,2)      % Input 2
```

```
Y =

[(4503599627370496*(5070602400912917605986812821504000*s^4 + 4344603426296280586326554705920000*s^3 + 2045710717419803303138095252959892225*s^2 + 319956418859944134
[         - (9007199254740992*(- 1727896870025471258850469347328000*s^3 - 2469970417608399942995712568341675*s^2 + 160093970830031615847837970783711 2*s + 2400449232
[                                                 (4503599627370496*(- 38884452581274262645830183813120*s^3 + 22355326029771730252537910380978 1*s^2 + 449276415981010

b1 =

   1.0e+04 *

  Columns 1 through 7

         0         0    0.0001    0.0015    0.0152    0.0902    0.3253
         0         0   -0.0001   -0.0026   -0.0233   -0.1374   -0.5700
         0         0         0   -0.0003   -0.0048   -0.0324   -0.1161

  Columns 8 through 13

    0.6874    0.6951   -0.0974   -0.9651   -0.8652   -0.2543
   -1.6552   -3.2574   -4.1944   -3.3719   -1.5393   -0.3060
   -0.2365   -0.2734   -0.1667   -0.0415    0.0000    0.0000


a1 =

   1.0e+05 *

  Columns 1 through 7

    0.0000    0.0002    0.0023    0.0156    0.0741    0.2480    0.5893

  Columns 8 through 13

    1.0009    1.2197    1.0527    0.6133    0.2153    0.0341


b2 =

   1.0e+05 *

  Columns 1 through 7

         0         0   -0.0000   -0.0005   -0.0079   -0.0608   -0.2695
         0         0    0.0001    0.0021    0.0199    0.1135    0.4440
         0         0   -0.0001   -0.0007   -0.0034   -0.0041    0.0184

  Columns 8 through 13

   -0.7086   -1.0763   -0.8259   -0.1277    0.2107    0.0963
    1.2232    2.3420    3.0111    2.4584    1.1461    0.2318
    0.0754    0.1139    0.0799    0.0216    0.0000    0.0000


a2 =

   1.0e+05 *

  Columns 1 through 7

    0.0000    0.0002    0.0023    0.0156    0.0741    0.2480    0.5893

  Columns 8 through 13

    1.0009    1.2197    1.0527    0.6133    0.2153    0.0341
```

Problem 9

Next published code

Contents

Pre-Load Before Starting

```
load('Values_New.mat')
Bus_Architecture()
```

Begin Script

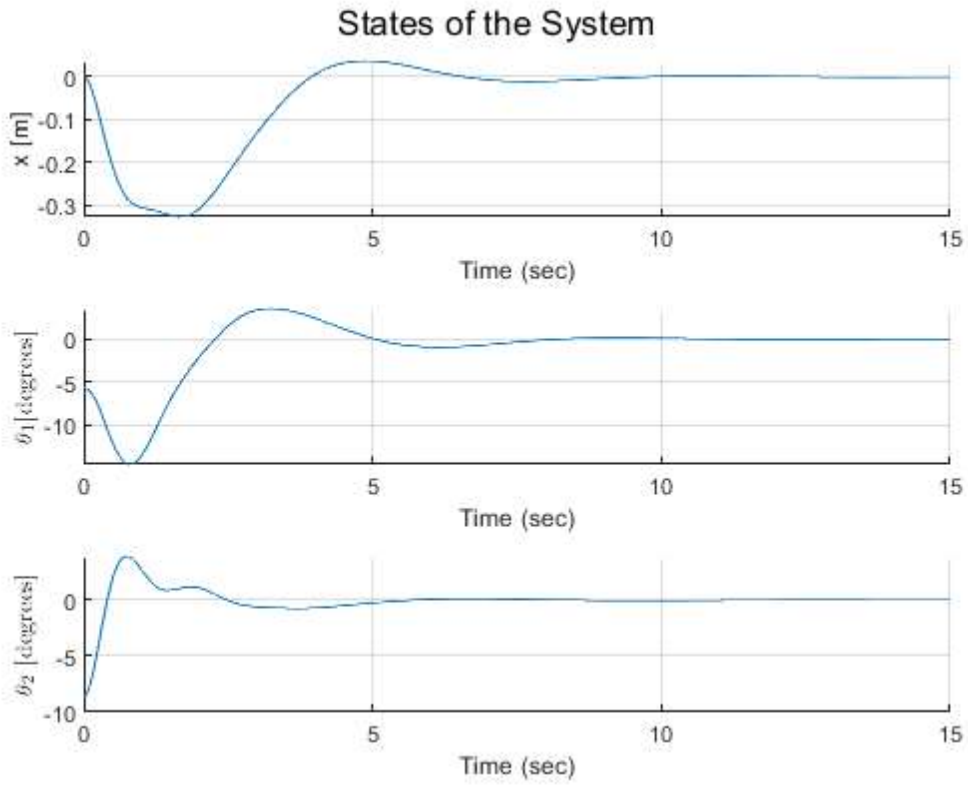
```
Sim_Time = 15;                              % Initialize Time

% Initial States
x0.x = 0;
x0.theta1 = -.1;
x0.theta2 = -.15;
x0.x_dot = 0;
x0.theta1_dot = 0;
x0.theta2_dot = 0;

% Reference Input
v.one = 0;
v.two = 0;
```

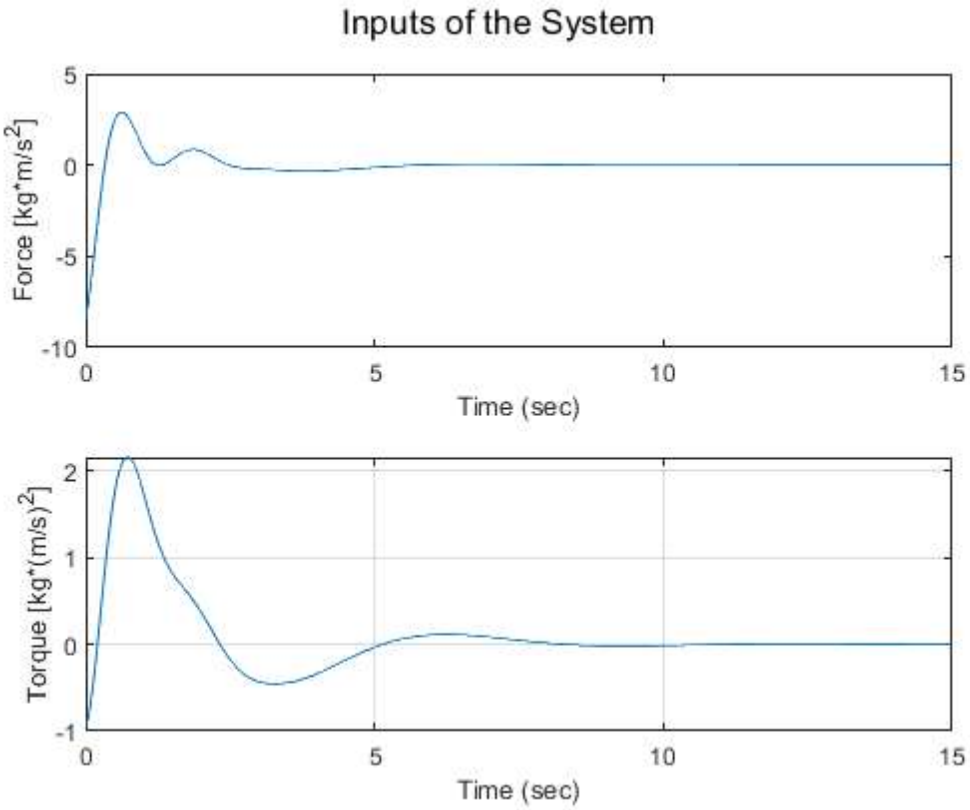
Just Controller

```
sim('Two_Input_Model.slx',Sim_Time)
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(2,1,1)
plot(logsout{4}.Values.Time',logsout{4}.Values.Data)
xlabel('Time (sec)')
ylabel('Force [kg*m/s^2]')
subplot(2,1,2)
plot(logsout{5}.Values.Time',logsout{5}.Values.Data)
xlabel('Time (sec)')
ylabel('Torque [kg*(m/s)^2]')
grid
hold off
```

```
ans =

  Simulink.SimulationOutput:
                logsout: [1x1 Simulink.SimulationData.Dataset]
                   tout: [459x1 double]

     SimulationMetadata: [1x1 Simulink.SimulationMetadata]
           ErrorMessage: [0x0 char]
```



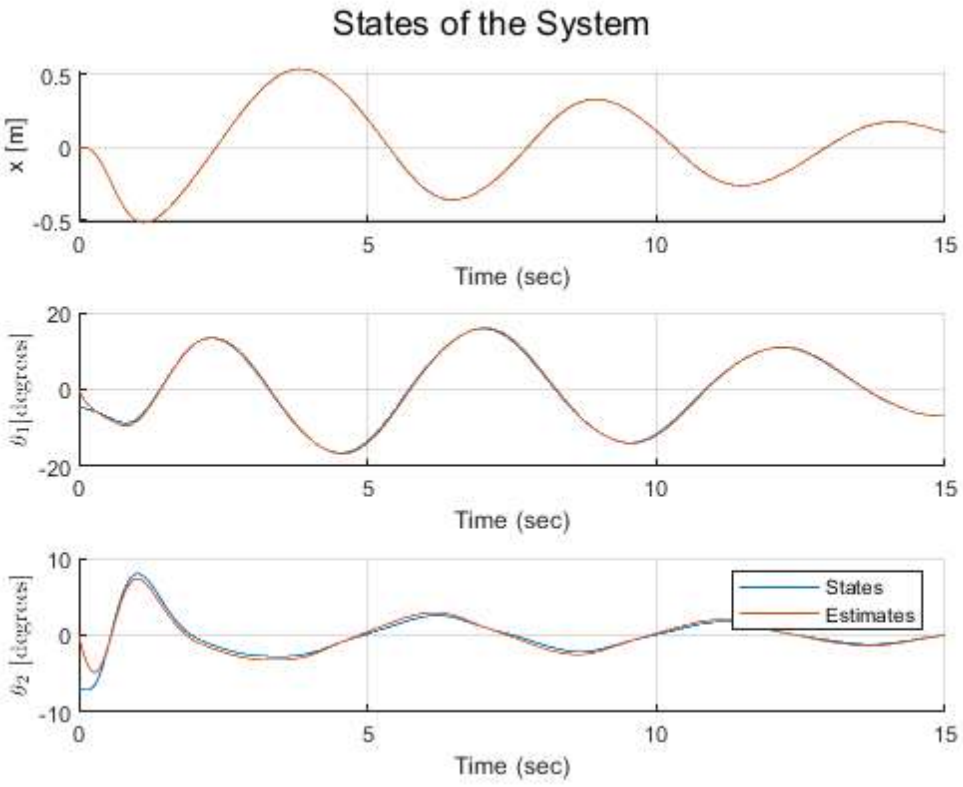


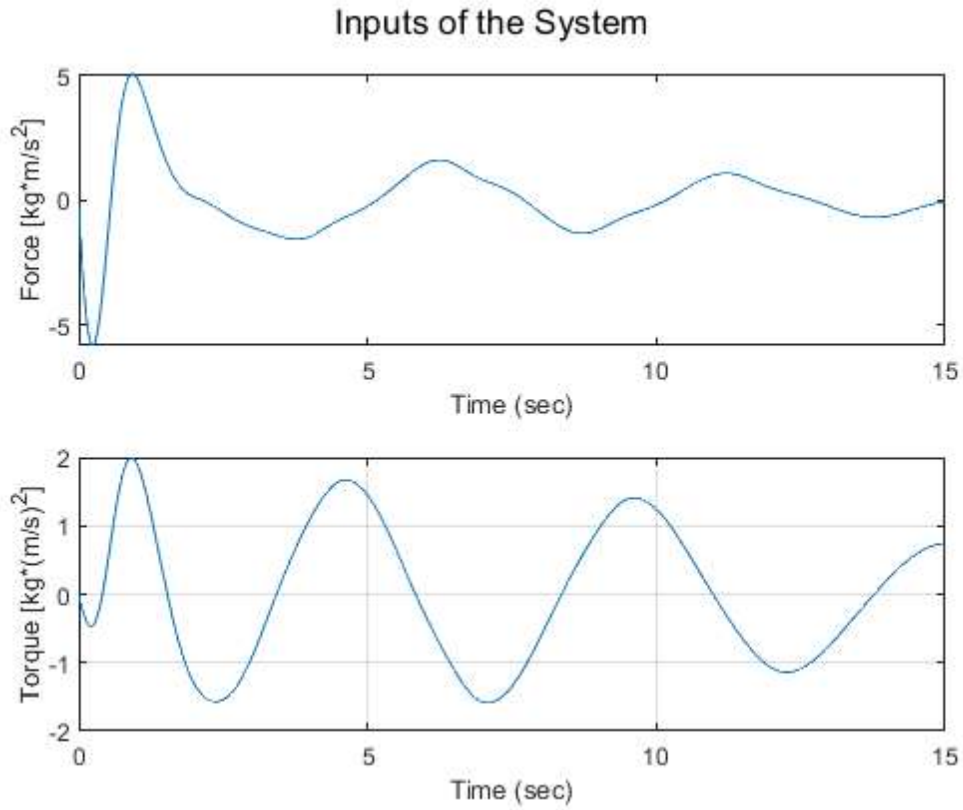
Controller-Observer

```
x0.theta1 = -.08;
x0.theta2 = -.12;
sim('Two_Input_Model_CO.slx',Sim_Time)
```

```
logsout = ans.logsout;

close all;
% 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{1}.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{2}.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{3}.Values.Time',logsout{8}.Values.Data*180/pi)
hold off
xlabel('Time (sec)')
ylabel('$\theta_2$ [degrees]','Interpreter','latex')
grid
legend('States','Estimates')
hold off

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

```
ans =

  Simulink.SimulationOutput:
                logsout: [1x1 Simulink.SimulationData.Dataset]
                   tout: [460x1 double]

    SimulationMetadata: [1x1 Simulink.SimulationMetadata]
          ErrorMessage: [0x0 char]
```





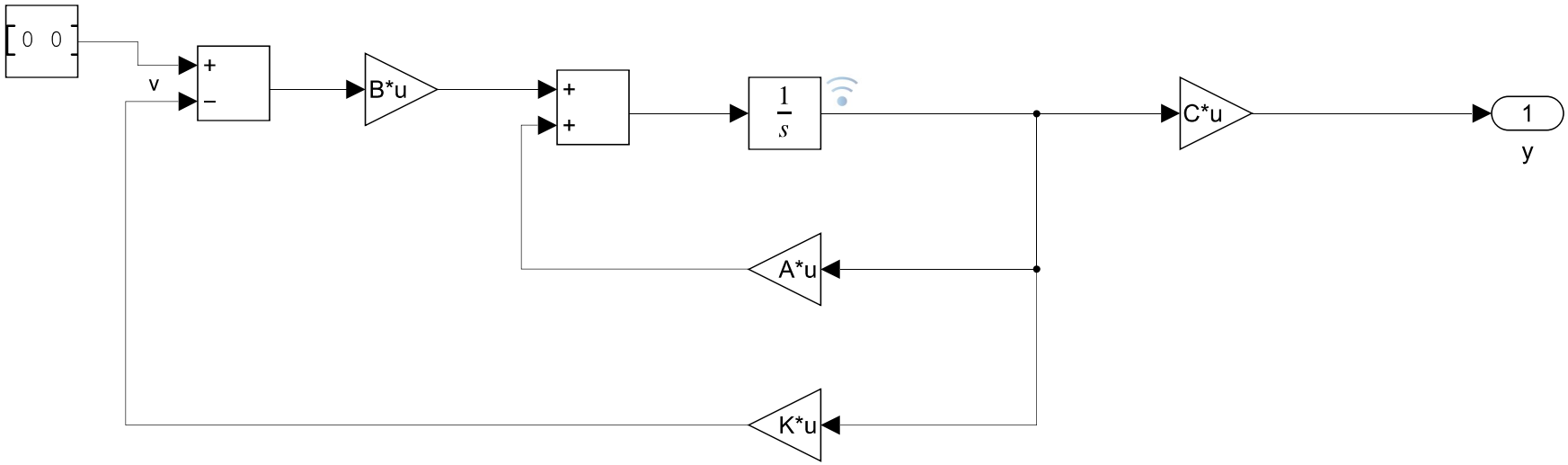
Published with MATLAB® R2021a

```
function Bus_Architecture()
    % 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 Bus of Refence
    ref_names = {'one','two',};
    ref_types = {'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
end
```

Published with MATLAB® R2021a

THIS IS FOR THE LINEAR CONTROLLER DESIGN

```
x0.x = 0;
x0.theta1 = -.01;
x0.theta2 = -.02;
x0.x_dot = 0;
x0.theta1_dot = 0;
x0.theta2_dot = 0;
```



Sample Times for 'Linear_Controller_Design'

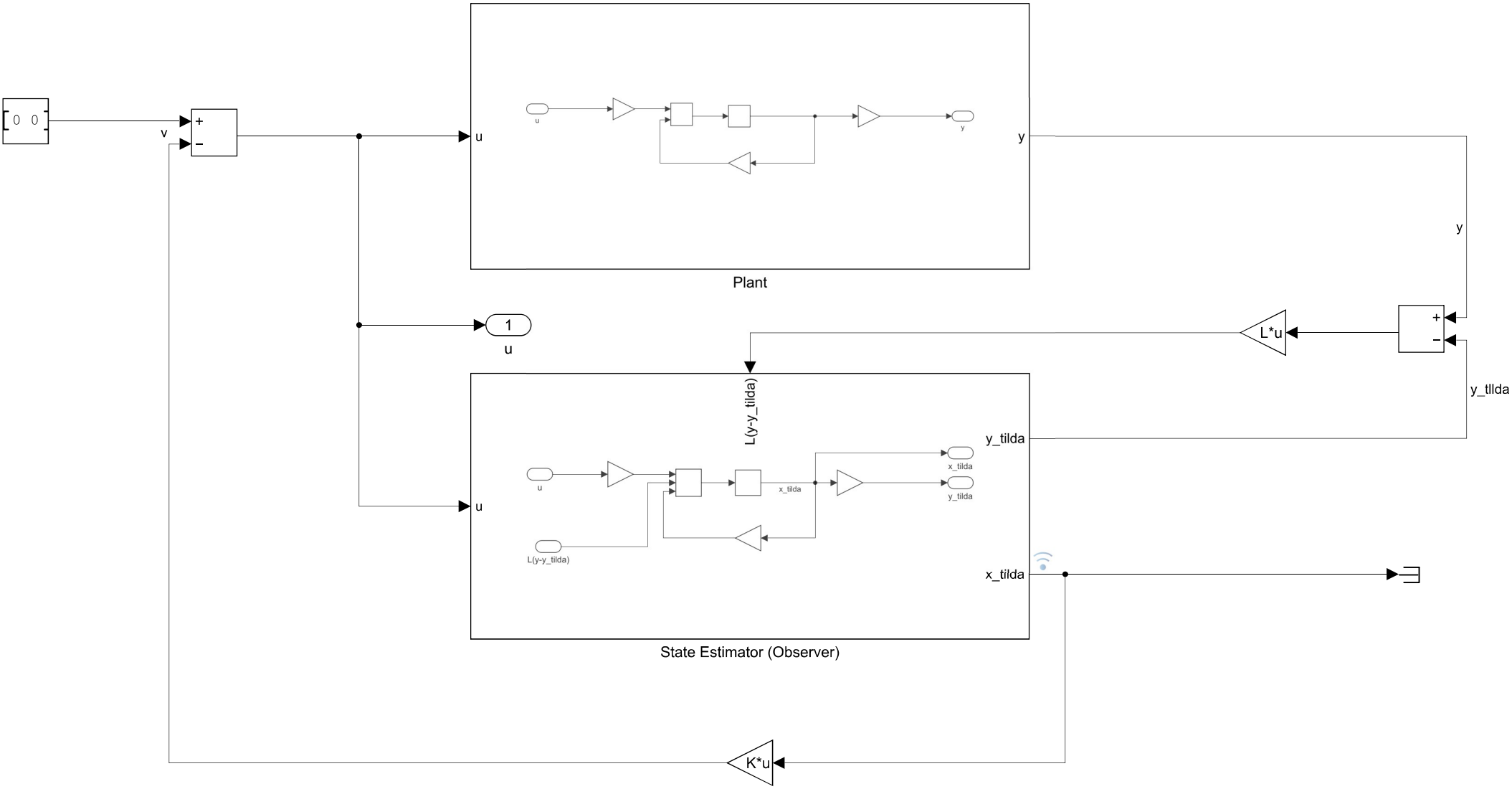
Color	Annotation	Description	Value
	Cont	Continuous	0
	FiM	Fixed in Minor Step	[0,1]
	Inf	Constant	Inf

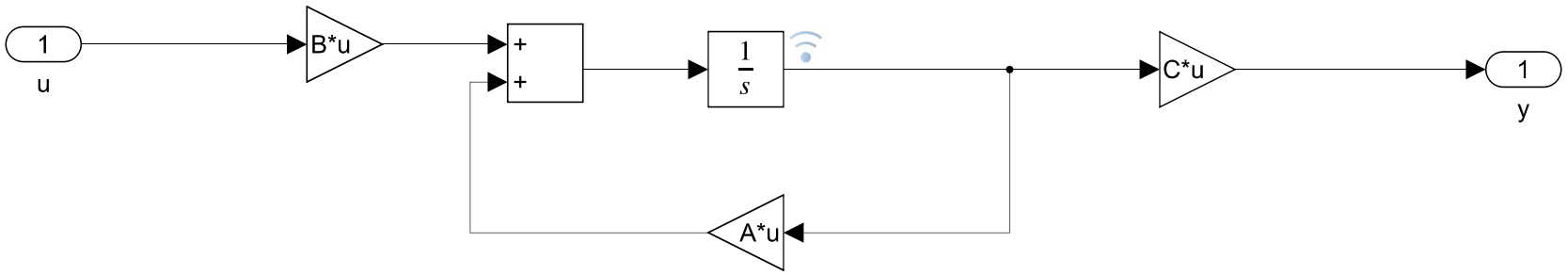
Page	System Name
1	Linear_Controller_Design

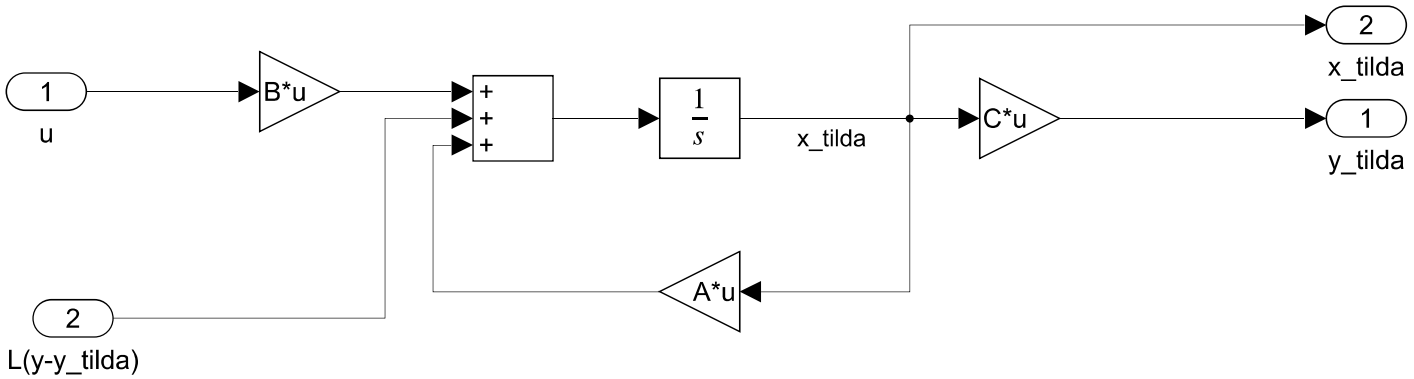
Other

Page	
1	Sample Time Legend

THIS IS TO DESGIN THE LINEAR CONTROLLER-OBSERVER COMPENSATOR




1
u
B*u
+
+
1
s
C*u
1
y
A*u


1
u
B*u
+
+
+
1
s
x_tilda
C*u
2
x_tilda
1
y_tilda
A*u
2
L(y-y_tilda)

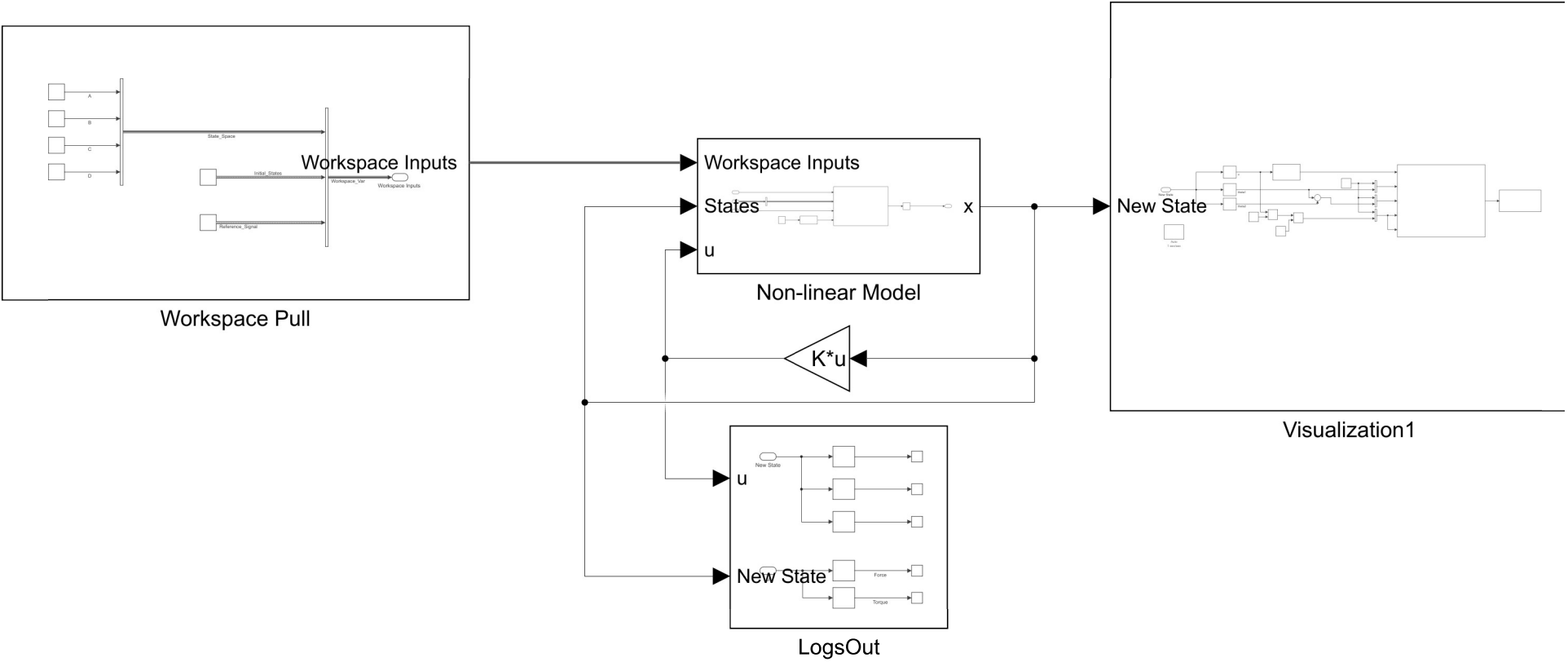
Sample Times for 'Linear_Control_Observer'

Color	Annotation	Description	Value
	Cont	Continuous	0
	FiM	Fixed in Minor Step	[0,1]
	Inf	Constant	Inf

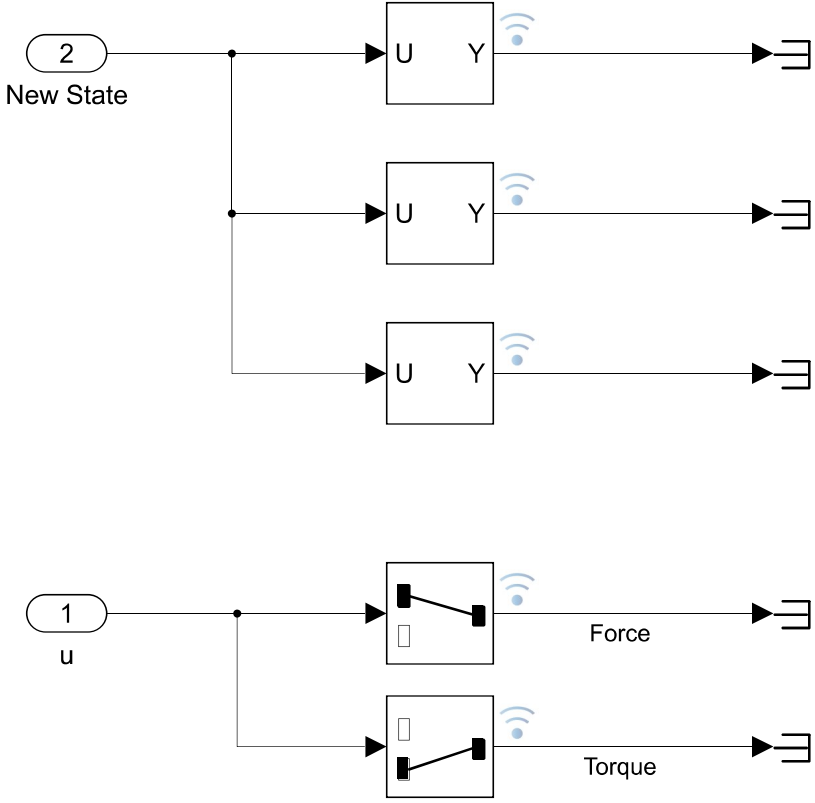
Page	System Name
1	Linear_Control_Observer
2	Linear_Control_Observer/Plant
3	Linear_Control_Observer/State Estimator (Observer)

Other

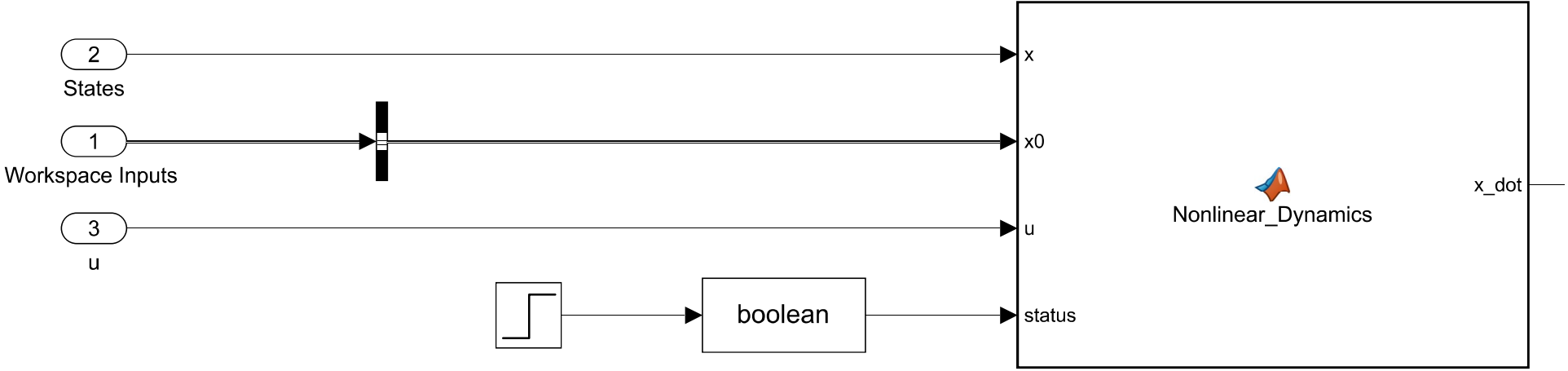
1	Sample Time Legend

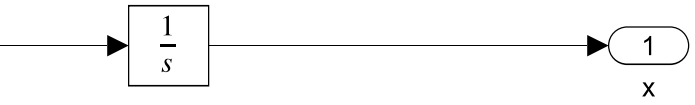


THIS IS FOR THE NON-LINEAR INTEGRATION WITH CONTROLLER ONLY

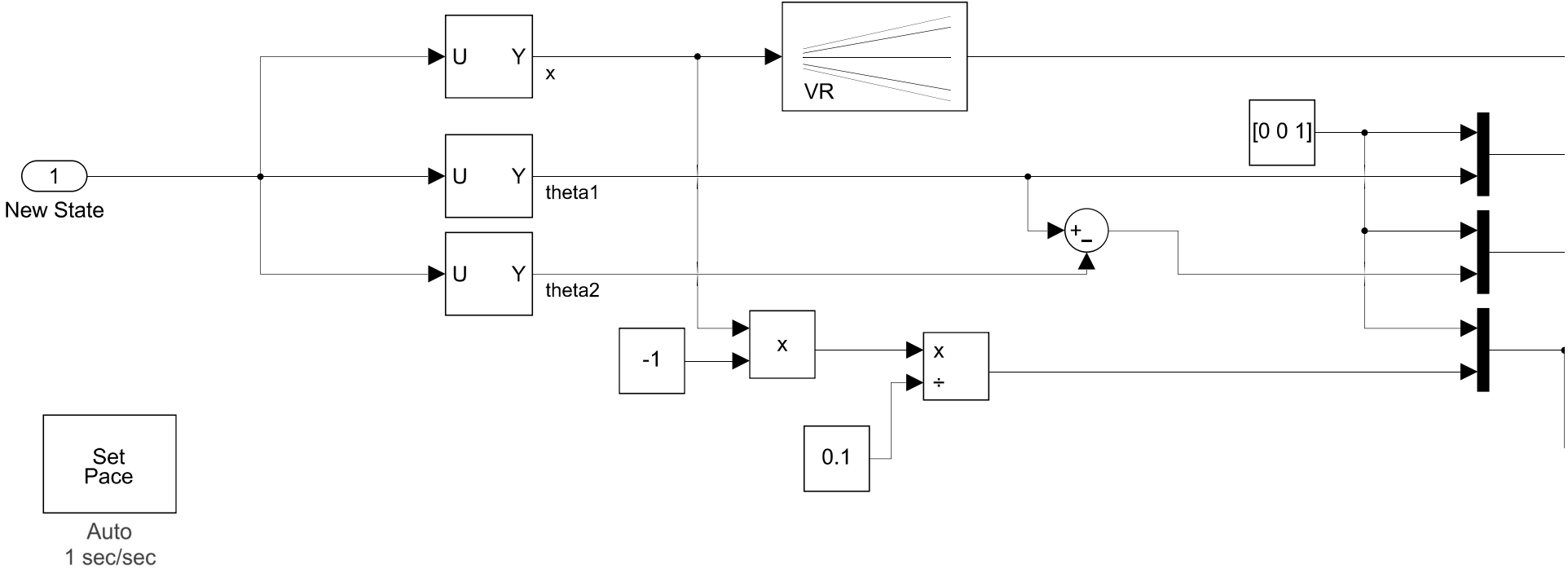

2
New State
U
Y
U
Y
U
Y

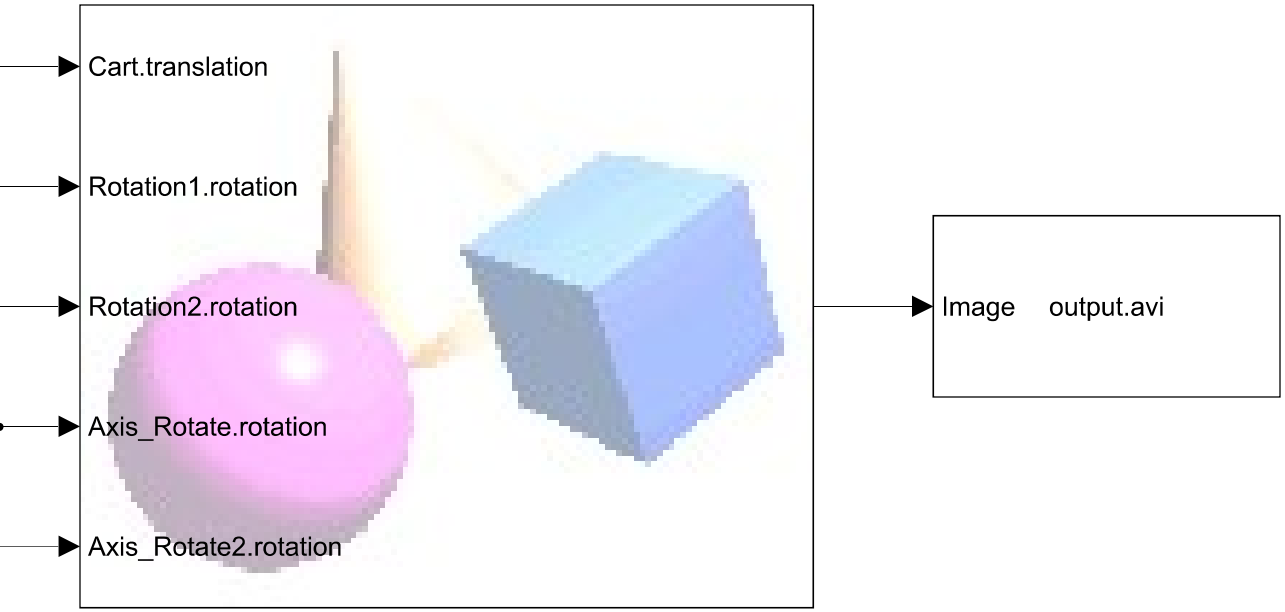
1
u
Force
Torque

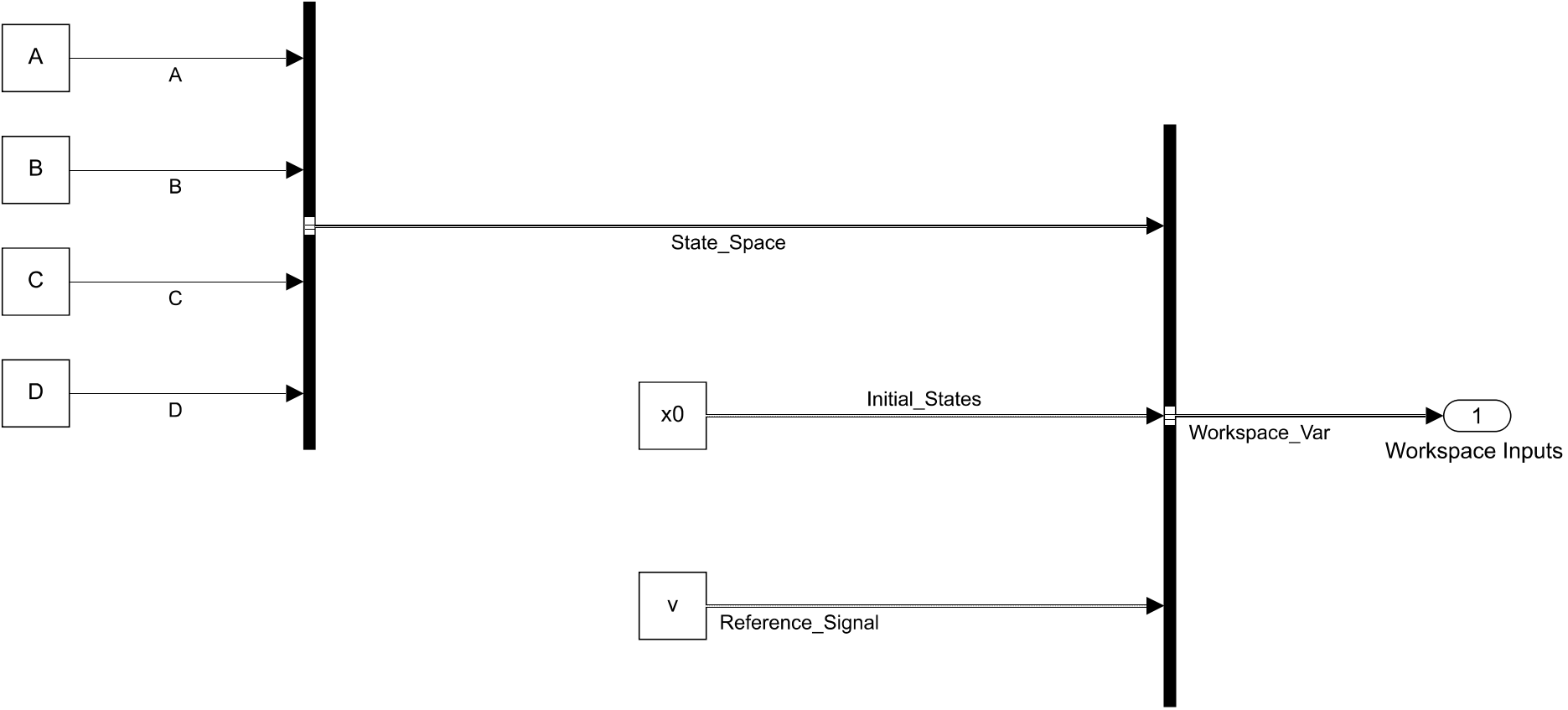

2
States
1
Workspace Inputs
3
u
x
x0
u
status
boolean
Nonlinear_Dynamics
x_dot

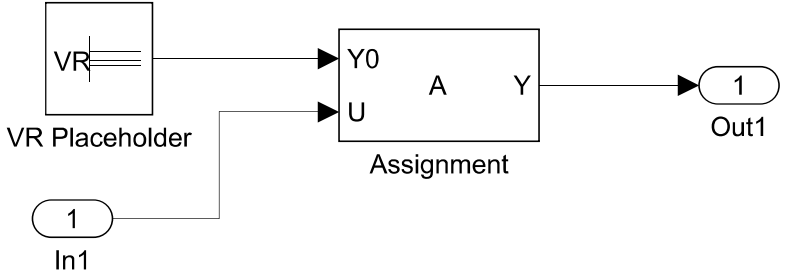

1
s
1
x

```
function x_dot  = Nonlinear_Dynamics(x,x0,u,status)
    if status == false
        x2 = x0.theta1;
        x3 = x0.theta2;
        x4 = x0.x_dot;
        x5 = x0.theta1_dot;
        x6 = x0.theta2_dot;
        u1 = u(1);
        u2 = u(2);
    else
        x2 = x(2);
        x3 = x(3);
        x4 = x(4);
        x5 = x(5);
        x6 = x(6);
        u1 = u(1);
        u2 = u(2);
    end
    % Insert Equations
    x_dot = double([
                x4;
                x5;
                x6;
                x5*((15*x5*sin(x2 - x3)*(cos(x3) - cos(x2 - x3)*cos(x2)))/(2*(25*cos(x2)^2 + 15*cos(x3)^2 + 33*cos(x2 - x3)^2 - 30*c
                x5*((5*x5*sin(x2)*(5*cos(x2) - 3*cos(x2 - x3)*cos(x3)))/(25*cos(x2)^2 + 15*cos(x3)^2 + 33*cos(x2 - x3)^2 - 30*cos(x2
                x6*((15*x6*sin(x3)*(cos(x3) - cos(x2 - x3)*cos(x2)))/(25*cos(x2)^2 + 15*cos(x3)^2 + 33*cos(x2 - x3)^2 - 30*cos(x2 -

end
```


U
Y
x
VR
1
New State
U
Y
theta1
U
Y
theta2
[0 0 1]
+
_
-1
x
x
÷
0.1
Set
Pace
Auto
1 sec/sec


Cart.translation
Rotation1.rotation
Rotation2.rotation
Axis_Rotate.rotation
Axis_Rotate2.rotation
Image
output.avi


A
A
B
B
C
C
D
D
State_Space
x0
Initial_States
Workspace_Var
1
Workspace Inputs
v
Reference_Signal

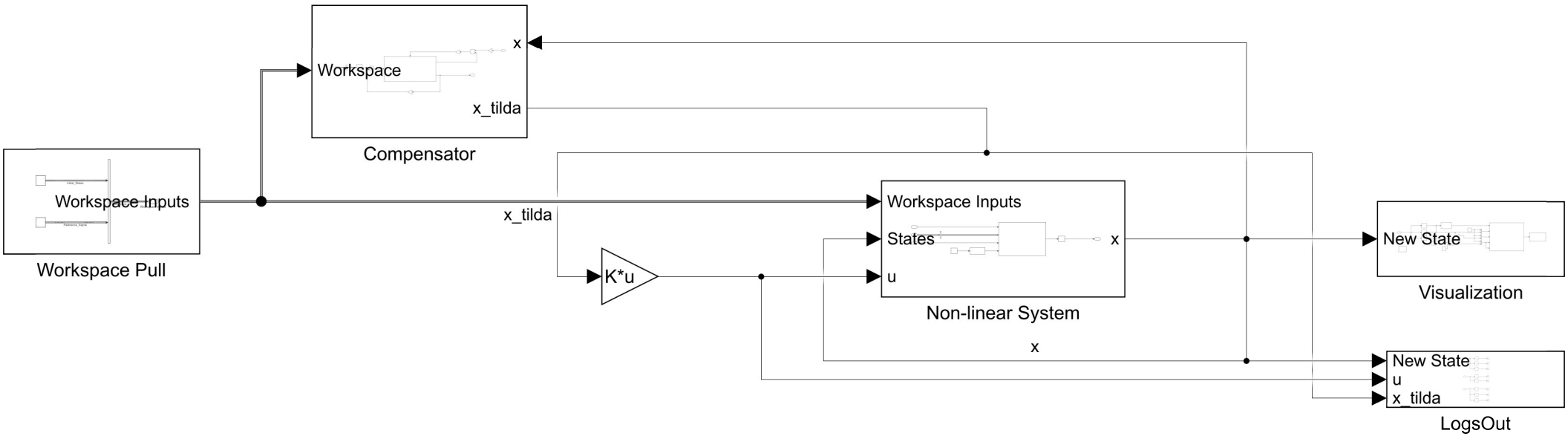

VR
VR Placeholder
Y0
U
A
Y
Assignment
1
Out1
1
In1

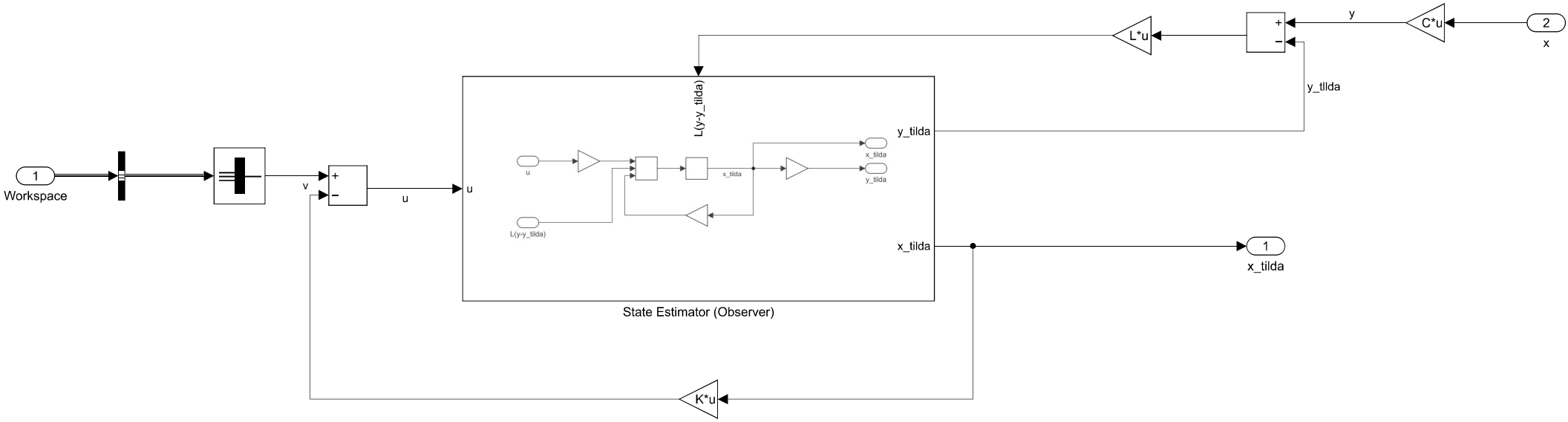
Sample Times for 'Two_Input_Model'

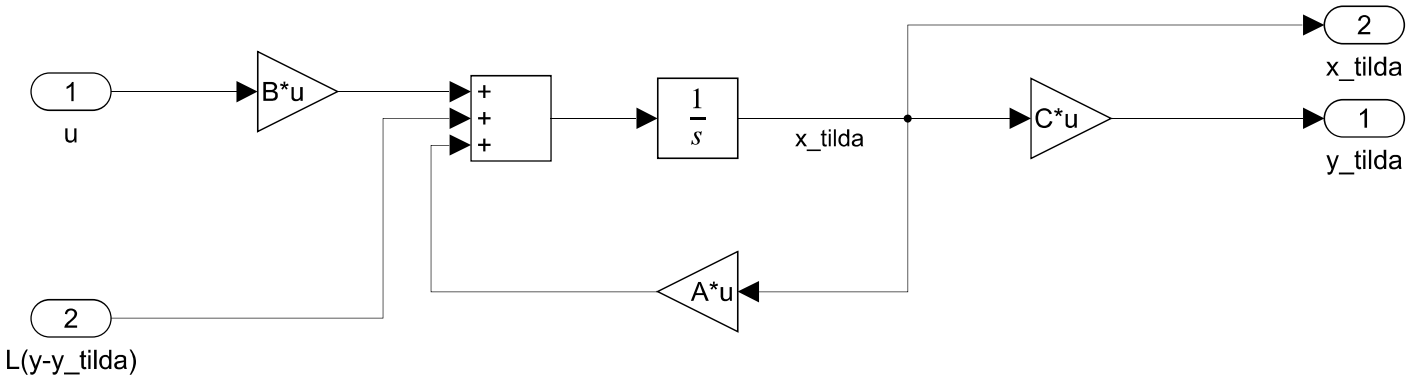
Color	Annotation	Description	Value
	Cont	Continuous	0
	FiM	Fixed in Minor Step	[0,1]
	D1	Discrete 1	0.033333
	D2	Discrete 2	0.1
	Inf	Constant	Inf
	M	Multirate	N/A

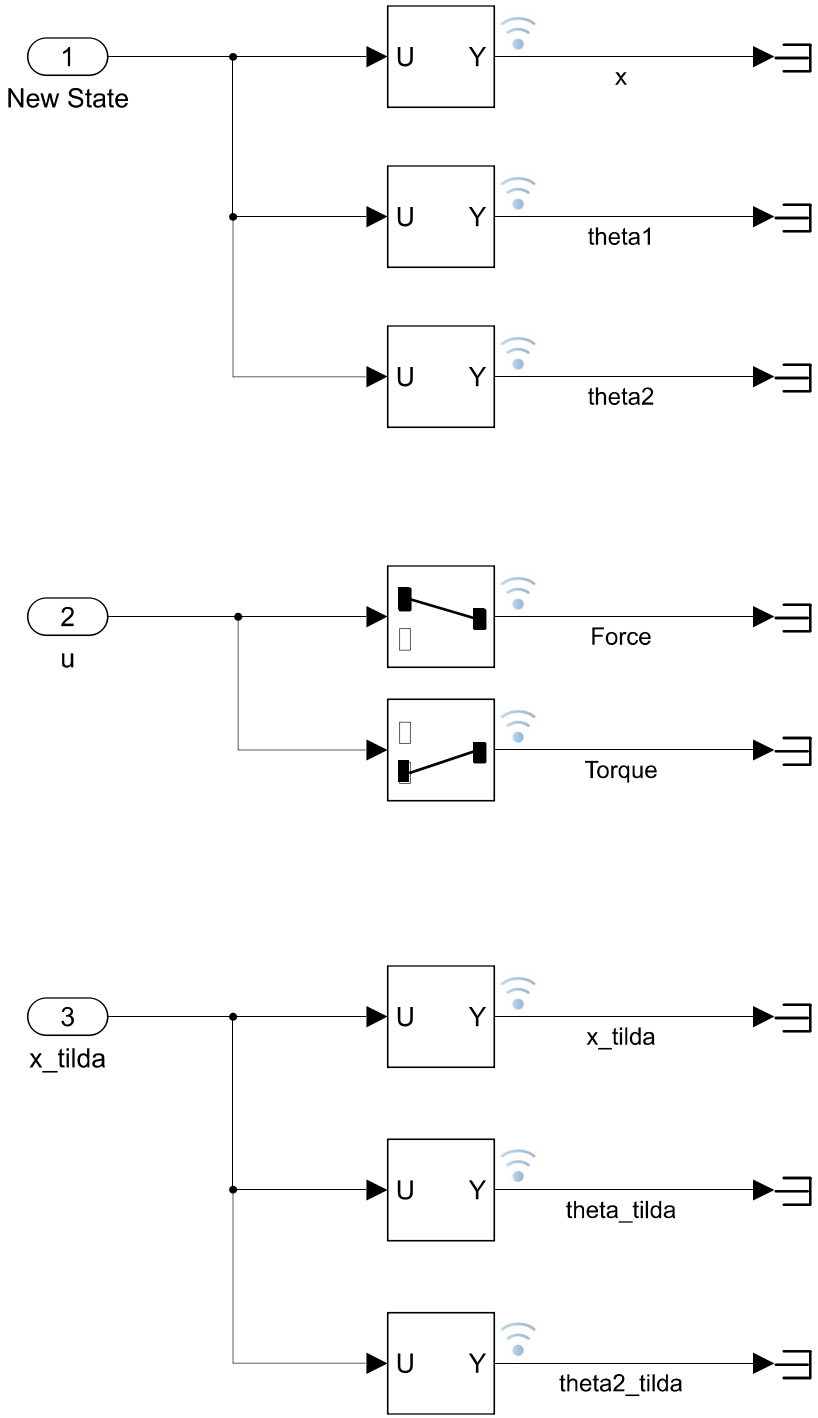
```
  Page      System Name
-----------------------------------------------------------
     1      Two_Input_Model
     2      Two_Input_Model/LogsOut
     3      Two_Input_Model/Non-linear Model
     4 (SF) Two_Input_Model/Non-linear Model/MATLAB Function
     5      Two_Input_Model/Visualization1
     6      Two_Input_Model/Workspace Pull


  Page      Unique Resolved Library Links
       (Page and Block Name referencing link)
-----------------------------------------------

     7      vrlib/VR Signal Expander
       (  5  VR Signal Expander)



Other
------
     1      Sample Time Legend
```

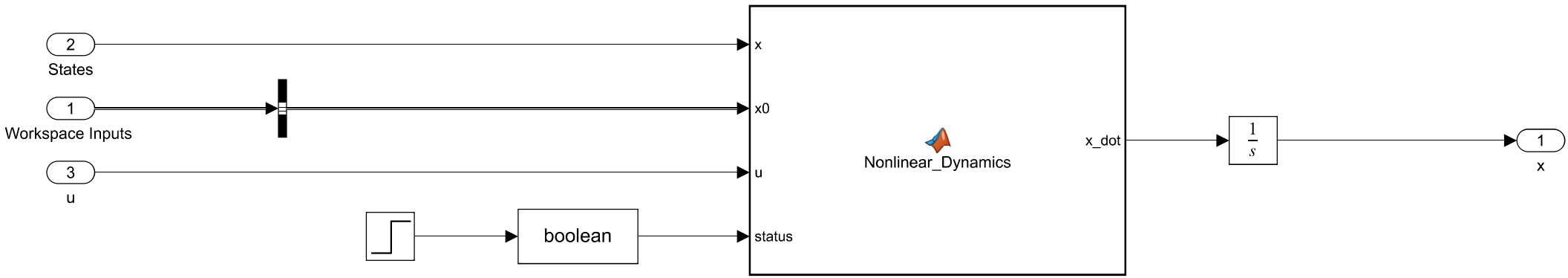
NON-LINEAR MODEL INTEGRATED WITH CONTROLLER-OBSERVER COMPENSATOR



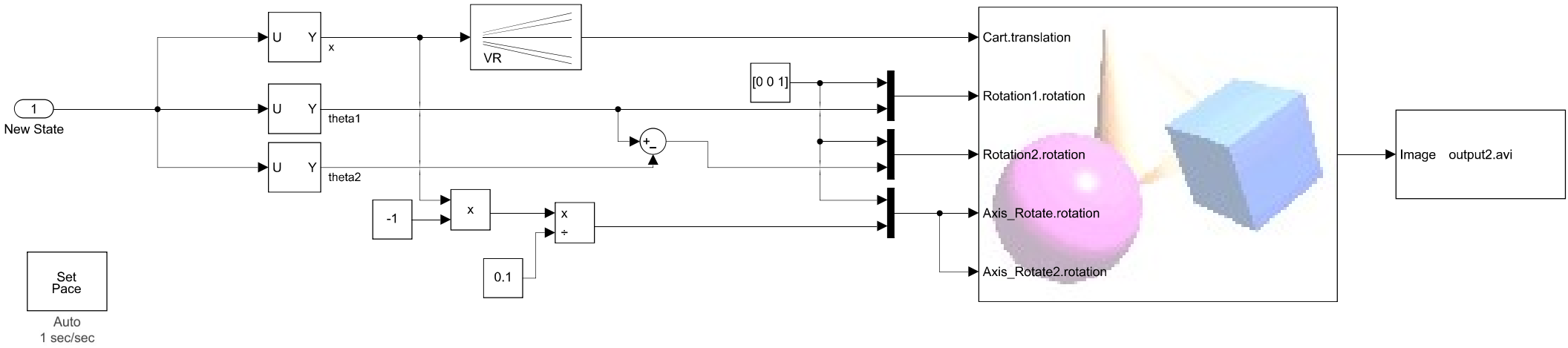

1
Workspace
v
+
−
u
u
L(y-y_tilda)
u
x_tilda
x_tilda
y_tilda
L(y-y_tilda)
y_tilda
x_tilda
State Estimator (Observer)
L*u
+
−
y
C*u
2
x
y_tllda
1
x_tilda
K*u

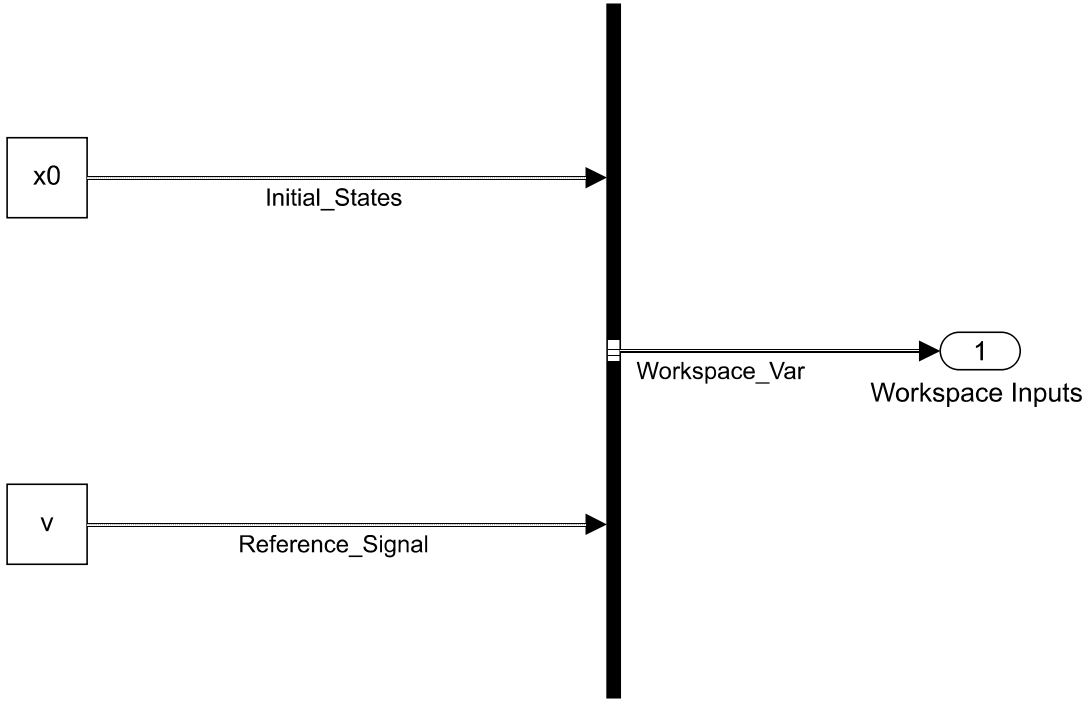

1
u
B*u
+
+
+
1
s
x_tilda
C*u
2
x_tilda
1
y_tilda
A*u
2
L(y-y_tilda)

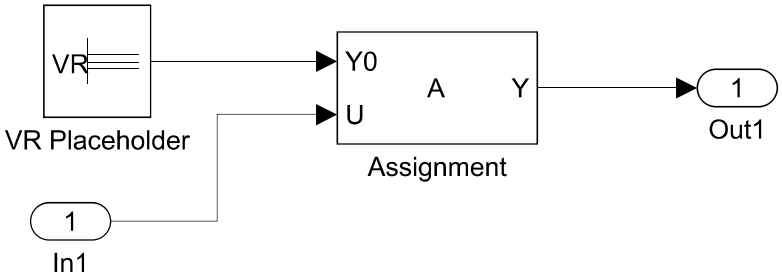

1
New State
U
Y
x
U
Y
theta1
U
Y
theta2
2
u
Force
Torque
3
x_tilda
U
Y
x_tilda
U
Y
theta_tilda
U
Y
theta2_tilda


2
States
1
Workspace Inputs
3
u
x
x0
u
status
Nonlinear_Dynamics
x_dot
1
s
1
x
boolean

```
function x_dot  = Nonlinear_Dynamics(x,x0,u,status)
    if status == false
        x2 = x0.theta1;
        x3 = x0.theta2;
        x4 = x0.x_dot;
        x5 = x0.theta1_dot;
        x6 = x0.theta2_dot;
        u1 = u(1);
        u2 = u(2);
    else
        x2 = x(2);
        x3 = x(3);
        x4 = x(4);
        x5 = x(5);
        x6 = x(6);
        u1 = u(1);
        u2 = u(2);
    end
    % Insert Equations
    x_dot = double([
                x4;
                x5;
                x6;
                x5*((15*x5*sin(x2 - x3)*(cos(x3) - cos(x2 - x3)*cos(x2)))/(2*(25*cos(x2)^2 + 15*cos(x3)^2 + 33*cos(x2 - x3)^2 - 30*c
                x5*((5*x5*sin(x2)*(5*cos(x2) - 3*cos(x2 - x3)*cos(x3)))/(25*cos(x2)^2 + 15*cos(x3)^2 + 33*cos(x2 - x3)^2 - 30*cos(x2
                x6*((15*x6*sin(x3)*(cos(x3) - cos(x2 - x3)*cos(x2)))/(25*cos(x2)^2 + 15*cos(x3)^2 + 33*cos(x2 - x3)^2 - 30*cos(x2 -

end
```


1
New State
U
Y
x
U
Y
theta1
U
Y
theta2
VR
[0 0 1]
+
-
-1
x
x
÷
0.1
Cart.translation
Rotation1.rotation
Rotation2.rotation
Axis_Rotate.rotation
Axis_Rotate2.rotation
Image
output2.avi
Set
Pace
Auto
1 sec/sec


x0
Initial_States
Workspace_Var
1
Workspace Inputs
v
Reference_Signal


VR
VR Placeholder
Y0
U
A
Y
Assignment
1
Out1
1
In1

Sample Times for 'Two_Input_Model_CO'

Color	Annotation	Description	Value
	Cont	Continuous	0
	FiM	Fixed in Minor Step	[0,1]
	D1	Discrete 1	0.033333
	D2	Discrete 2	0.1
	Inf	Constant	Inf
	M	Multirate	N/A

Page	System Name
1	Two_Input_Model_CO
2	Two_Input_Model_CO/Compensator
3	Two_Input_Model_CO/Compensator/State Estimator (Observer)
4	Two_Input_Model_CO/LogsOut
5	Two_Input_Model_CO/Non-linear System
6 (SF)	Two_Input_Model_CO/Non-linear System/MATLAB Function
7	Two_Input_Model_CO/Visualization
8	Two_Input_Model_CO/Workspace Pull

Page	Unique Resolved Library Links (Page and Block Name referencing link)
9	vrlib/VR Signal Expander (7 VR Signal Expander)

Other

1	Sample Time Legend