A REVIEW OF THE UNKNOWN INPUT OBSERVER WITH EXAMPLES

SAM NAZARI

Abstract. We consider the Unknown Input Observer and focus on three examples found in the literature.

1. INTRODUCTION

This memo considers the design of observers for a class of linear dynamic systems in which system uncertainty can be modeled as an additive unknown disturbance term in the dynamic equation:

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t) + Ed(t) \\ y(t) &= Cx(t) \end{aligned} \tag{1.1}$$

where $x(t) \in \mathbb{R}^n$ is the state vector, $y(t) \in \mathbb{R}^m$ is the output vector, $u(t) \in \mathbb{R}^r$ is the known input vector and $d(t) \in \mathbb{R}^q$ is the unknown input or disturbance vector. If the unknown input distribution matrix, E, is not full column rank, then the rank decomposition:

$$Ed(t) = E_1 E_2 d(t)$$

can be applied where, E_1, is a full column rank matrix and $E_2 d(t)$ can be considered as a new unknown input. The term, $Ed(t)$, may be used to model additive disturbances or model uncertainties that are not known *a priori*. Typical uncertainties are noise, plant nonlinearities, model reduction errors, parametric uncertainties and difficult to model interconnection terms in large scale systems. While not considered in this memo, a disturbance term may also appear in the output equation:

$$y(t) = Cx(t) + E_y d(t)$$

Additionally, this memo does not consider the $Du(t)$ term in the system output equations. For the sake of brevity and without loss of generality, we will not treat systems with this term. Note that the development of the theoretical material in this memo closely follows Chapter 3 of [1].

2. Unknown Input Observer Theory

The problem of designing observers for a linear system with both known and unknown inputs has been studied since the 1970's [1], [2], [4]. The motivation for studying this problem is that in practice, for a variety of reasons, many plants are modeled with disturbance terms as shown in Eq 1.1. When modeled in such a way, traditional observers with the Luenberger structure become difficult to implement since they use all of the input signals in order to compute an estimate of the state vector. This limitation renders them ineffective for many practical applications. Contrastingly, the observer structure considered in this memo assumes no *a priori* knowledge about the unknown inputs, rendering it more amenable for use in fault detection applications. Many investigators have addressed this problem with varying success and we refer the reader to Section 3.2 of [1] for a broad literature review. The class of observers examined in this memo is based on the *Unknown Input Observer* (UIO) scheme proposed by Chen, Patton and Zhang.

Definition 2.1. An observer is defined as an *unknown input observer* [3, 2] for the system described by 1.1, if its state estimation error vector, $e(t)$, defined as:

$$e(t) = x(t) - \hat{x}(t) \tag{2.1}$$

approaches zero asymptotically regardless of the presence of unknown inputs, $d(t)$, in the system. Furthermore, the structure for a full order UIO is given by the dynamic system:

$$\begin{aligned} \dot{z}(t) &= Fz(t) + TBu(t) + Ky(t) \\ \hat{x}(t) &= z(t) + Hy(t) \end{aligned} \tag{2.2}$$

Key words and phrases. Unknown Input Observer, Disturbance Estimation.

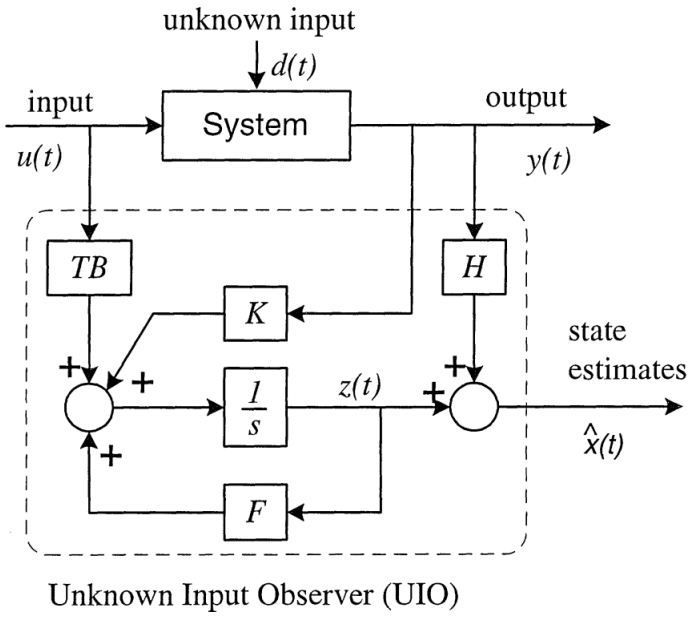

FIGURE 1. UIO Structure where $K = K_1 + K_2$.

where $\hat{x} \in \mathbb{R}^n$ is the state estimate, $z \in \mathbb{R}^n$ is the state of the full-order dynamic observer, and F, T, K, H are matrices to be designed for achieving unknown input decoupling. A block diagram of the UIO is shown in Figure 1. From the block diagram, one can see that the UIO is essentially a dynamic system that decouples the state estimation dynamics from the disturbance term in the original system. By expanding $\dot{e}(t)$ one obtains:

$$\begin{aligned}\dot{e}(t) = (A - HCA - K_1C)e(t) + [F - (A - HCA - K_1C)]\, z(t) \\ + [K_2 - (A - HCA - K_1C)]y(t) \\ + [T - (I - HC)]Bu(t) + (HC - I)Ed(t)\end{aligned}$$

and it is easy to see that in order to make the estimation error a function of $Fe(t)$:

$$\dot{e}(t) = Fe(t) \tag{2.3}$$

the equations:

$$0 = (HC - I)E \tag{2.4}$$

$$T = I - HC \tag{2.5}$$

$$F = A - HCA - K_1C \tag{2.6}$$

$$K_2 = FH \tag{2.7}$$

must hold. If all the eigenvalues in F are stable, then $e(t)$ will approach zero asymptotically. Note that Eq 2.3 is not a function of E or d. Therefore, the estimation error approaches zero independently of the disturbance terms, achieving the desired decoupling of the state estimate from the unknown disturbance inputs.

Theorem 2.2. *The necessary and sufficient conditions for the system in Eq 2.2 to be an UIO for the system defined in 1.1 are:*

(1) $rank(CE) = rank(E)$
(2) (C, A_1) *is a detectable pair, where* $A_1 = A - E[(CE)^T CE]^{-1}(CE)^T CA$

Proof. Please see [1] □

According to [4], the disturbance estimate can now be obtained, if needed, by the relation:

$$\hat{d} = (CE)^{\dagger}[\dot{\hat{y}} - CA\hat{x} - CBu] \tag{2.8}$$

Next we provide a step by step design procedure for the UIO that will be utilized in the examples that ensue.

2.1. **Design Procedure for UIOs.** We will now describe a procedure that can lead to effective UIO design. It can be shown that Eqn 2.4 is solvable if and only if the condition $rank(CE) = rank(E)$ is met [1]. Therefore, the rank condition $rank(CE) = rank(E)$ must first be checked. If this condition is not met, then a UIO does not exist. If the condition is met, then we may compute the following observer matrices:

$$\begin{aligned} H &= E[(CE)^T CE]^{-1}(CE)^T \\ T &= I - HC \\ A_1 &= TA \end{aligned}$$

Next, the observability of the pair (C, A_1) must be checked. As shown in Eq 2.3, an important step in designing a UIO is to stabilize $F = A_1 - K_1 C$. If the pair (C, A_1) is detectable, then this can be achieved by using pole placement to choose the appropriate gain matrix K_1. It can be shown that the observability of the pair (C, A_1) is equivalent to the observability of the pair (C, A) [3]. Therefore, if the pair (C, A_1) is observable or at least detectable then the gain, K_1, in Eq 2.6 can be obtained for the UIO. Hence, if (C, A_1) is observable then a UIO exists and K_1 should be computed using pole placement.

On the other hand, if the pair (C, A_1) is not observable, then one must construct a transformation matrix, P, for the observable canonical decomposition of the system:

$$\begin{aligned} PA_1P^{-1} &= \begin{bmatrix} A_{11} & 0 \\ A_{12} & A_{22} \end{bmatrix} & A_{11} &\in \mathbb{R}^{n_1 \times n_1} \\ CP^{-1} &= [C^* \;\; 0] & C^* &\in \mathbb{R}^{m \times n_1} \end{aligned}$$

where n_1 is the rank of the observability pair (C, A_1), and (C^*, A_{11}) is observable. If any of the eigenvalues of A_{22} are unstable, then a UIO does not exist. If the eigenvalues of A_{22} are stable, then (C, A_1) is detectable and we may select n_1 desirable eigenvalues and assign them to $A_{11} - K_p^1 C^*$ using pole placement to obtain K_p^1. Next compute

$$K_1 = P^{-1}K_p = P^{-1}[(K_p^1)^T \;\; (K_p^2)^T]^T$$

where K_p^2 can be any $(n - n1) \times m$ matrix, because it does not affect the eigenvalues of F. Finally, compute the observer matrices:

$$\begin{aligned} F &= A_1 - K_1 C \\ K &= K_1 + K_2 = K_1 + FH. \end{aligned}$$

In order to clarify the design procedure we consider some numerical examples next.

3. Numerical Examples

3.1. **Third order system with no inputs.** The first example considered is taken from page 76 of [1]. There, the following parameter matrices are used:

$$A = \begin{bmatrix} -1 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & -1 & -1 \end{bmatrix}, \; C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \; E = \begin{bmatrix} -1 \\ 0 \\ 0 \end{bmatrix} \tag{3.1}$$

First we check that $rank(CE) = rank(E) = 1$:

```
rank(C*E)
rank(E)
```

```
ans =       ans =

     1            1
```

The rank is equal to 1 as needed. The observer matrices are computed next:

```
H = E*inv((C*E)'*(C*E))*(C*E)'
T = eye(3)-H*C
A1 = T*A
```

```
H =                 T =                 A1 =

     1     0           0     0     0           0     0     0
     0     0           0     1     0          -1     0     0
     0     0           0     0     1           0    -1    -1
```

Next, the observability of the pair (A_1, C) must be checked. If it is full rank ($rank(obsv(A_1, C)) = 3$), then we may apply pole placement:

```
rank(obsv(A1,C))
```

```
ans =

     3
```

Now we proceed to apply pole placement in order to place the observer poles at $[-2, -10, -5]$.

```
K1 = place(A',C',[-2,-10,-5])'
```

```
K1 =

    6.7231   -3.7816
    7.5581  -11.3458
   -3.0543    8.2769
```

Finally, the observer F and K matrices are computed:

```
F = A1-K1*C
K = K1 + F*H
```

```
F =                                   K =

   -6.7231         0    3.7816              0    -3.7816
   -8.5581         0   11.3458             -1   -11.3458
    3.0543   -1.0000   -9.2769              0     8.2769
```

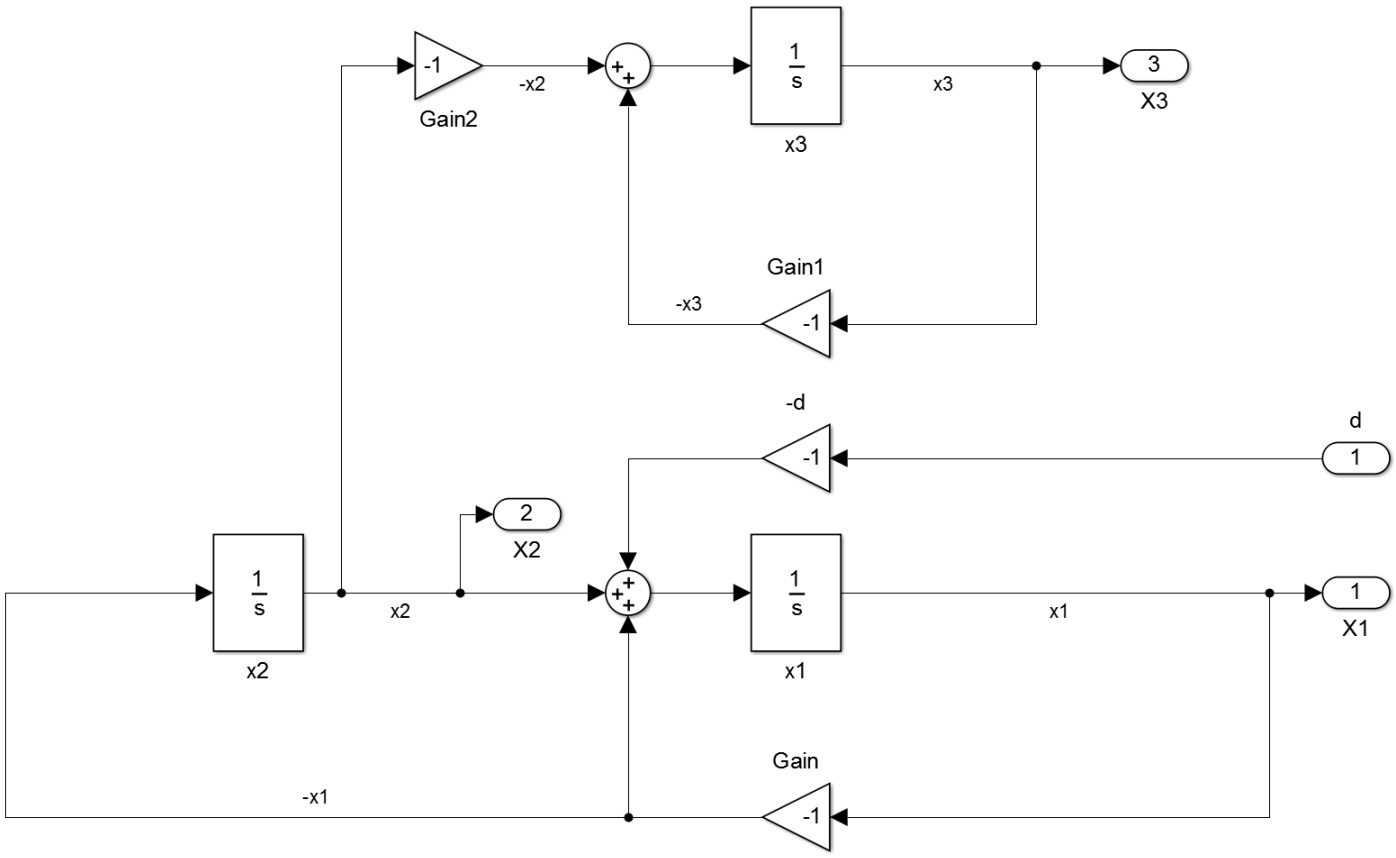

FIGURE 2. Simulink block diagram of the dynamics subsystem.

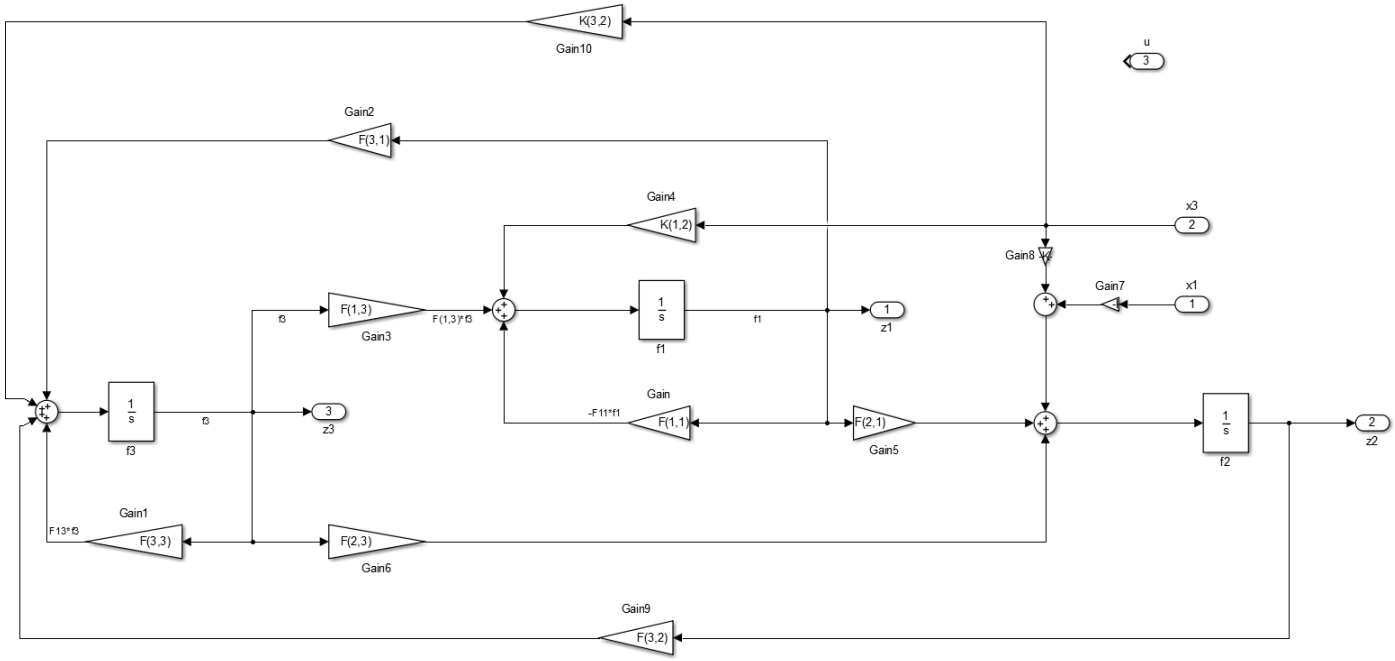

FIGURE 3. Simulink block diagram of the UIO.

Let us examine the Simulation results with a disturbance occurring at $t = 10$ seconds. This is showin in Figure 4. The initial conditions of the dynamic system as:

```
x1_0 = 100;
x2_0 = -100;
x3_0 = 1;
```

In Figure 4, one is able to see the state trajectories in the solid lines. The dotted 'x' represent the estimate from the UIO. For the given disturbance and the chosen observer pole locations, the observer seems to converge (remove all estimation error) within five seconds. Figures 2 and 3 show the block diagram in detail. Note that we have chosen not to estimate the disturbance itself. All the files for this simulation can be found in the

`C:\Users\sqn4594\Documents\FDIR`

folder. Specifically, the file

`bookEx.m`

and the Simulink model

`exOne.xsl`

contain everything.

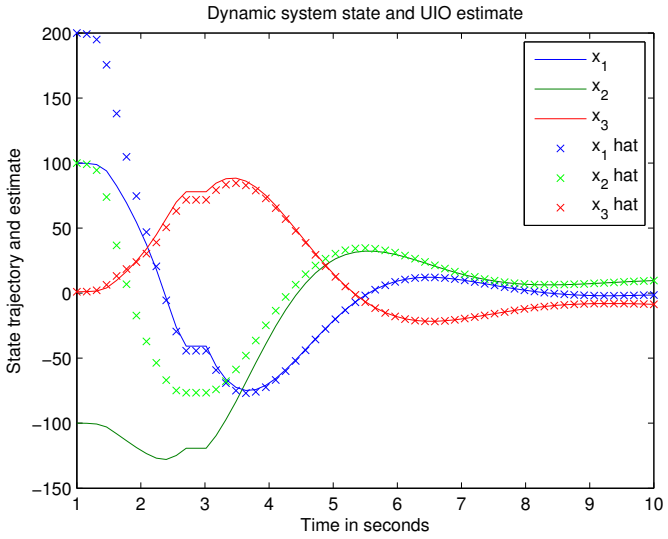

FIGURE 4. State response and UIO estimate for example

3.2. **Car Suspension System.** The next example is a quarter car model. The system matrices are:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{-ks}{mb} & \frac{-bs}{mb} & \frac{ks}{mb} & \frac{bs}{mb} \\ 0 & 0 & 0 & 1 \\ \frac{ks}{mw} & \frac{bs}{mw} & \frac{-ks-kt}{mw} & \frac{-bs}{mw} \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 \\ 0 & \frac{1000}{mb} \\ 0 & 0 \\ \frac{kt}{mw} & \frac{-1000}{mw} \end{bmatrix} \quad C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad E = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \tag{3.2}$$

where m_b, represents the car body and the mass, m_w, represents the wheel assembly. The spring ,k_s, and damper, b_s, represent the passive spring and shock absorber placed between the car body and the wheel assembly. The spring, k_t, models the compressibility of the pneumatic tire. The variables x_b, x_w and r are the car body travel, the wheel travel, and the road disturbance, respectively. The force, f_s, which is applied between the body and the wheel assembly, is controlled by feedback. We assume that the disturbance force may enter at any of the state variables, not just the road disturbance. In MATLAB, we assign the following values to the parameters:

```
%----System Physical Constants----%
mb = 300;     % kg
mw = 60;      % kg
bs = 1000;    % N/m/s
ks = 16000 ; % N/m
kt = 190000; % N/m
```

Therefore, the system matrices become:

```
% --------System Matrices(It is a SIMO system)---------%
A = [ 0 1 0 0; [-ks -bs ks bs]/mb ; ...
      0 0 0 1; [ks bs -ks-kt -bs]/mw];
B = [0 0; 0 10000/mb ; 0 0; [kt -10000]/mw];
C = eye(4);
E=[1;1;1;1];
```

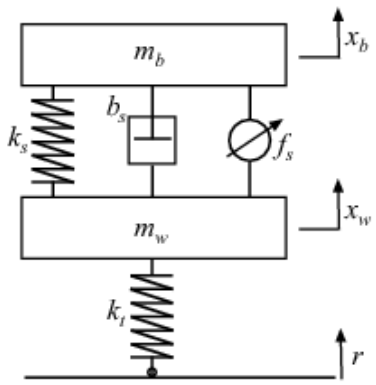

FIGURE 5. Free Body Diagram of the Quarter Car Suspension model (idealization).

```
A =

   1.0e+03 *

         0    0.0010         0         0
   -0.0533   -0.0033    0.0533    0.0033
         0         0         0    0.0010
    0.2667    0.0167   -3.4333   -0.0167

B =

    1.0e+03 *

         0         0
         0         0.0333
         0         0
    3.1667   -0.1667

C =

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

E =

     1
     1
     1
     1
```

Now we can obtain an optimal gain for control of the suspension system by using LQR below. We will not dwell too much on this part of the design work since our main focus is the UIO design.

```
%---------LQR design----------%
Q=[.25 0 0 0;0 4 0 0;0 0 1 0;0 0 0 4];
R=50*eye(2);
[K1 l s]=lqr(A,B,Q,R);
```

In order to design the UIO, we first check to see if the $rank(CE) = rank(E) = 1$

```
rank(C*E)
rank(E)
```

```
ans =       ans =

     1             1
```

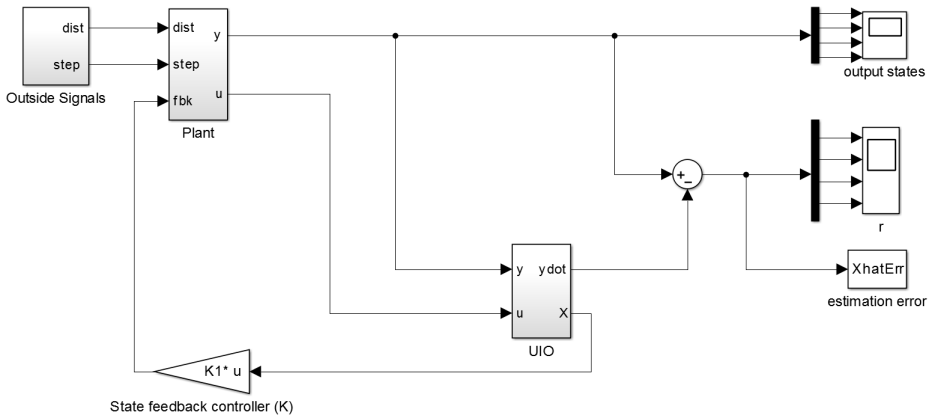

FIGURE 6. Root leve Simulink Block diagram architecture of the car suspension system in example two.

Next we compute the observer matrices

```
H = E*inv((C*E)'*(C*E))*(C*E)'
T = eye(4)-H*C
A1 = T*A

f=zeros(4,4)
v=[-4 -4 -4 -4]
F=diag(v)

k1=inv(C)*(A-F-H*C*A)
k2=F*H
k=zeros
k=k1+k2
```

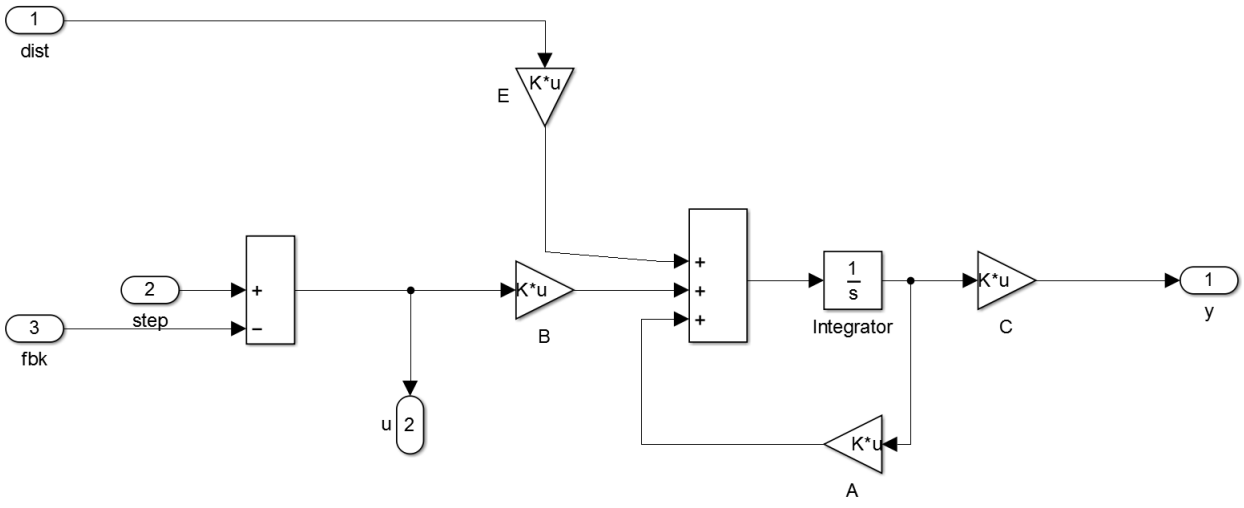

FIGURE 7. Quarter car suspension system dynamics implementation in Simulink.

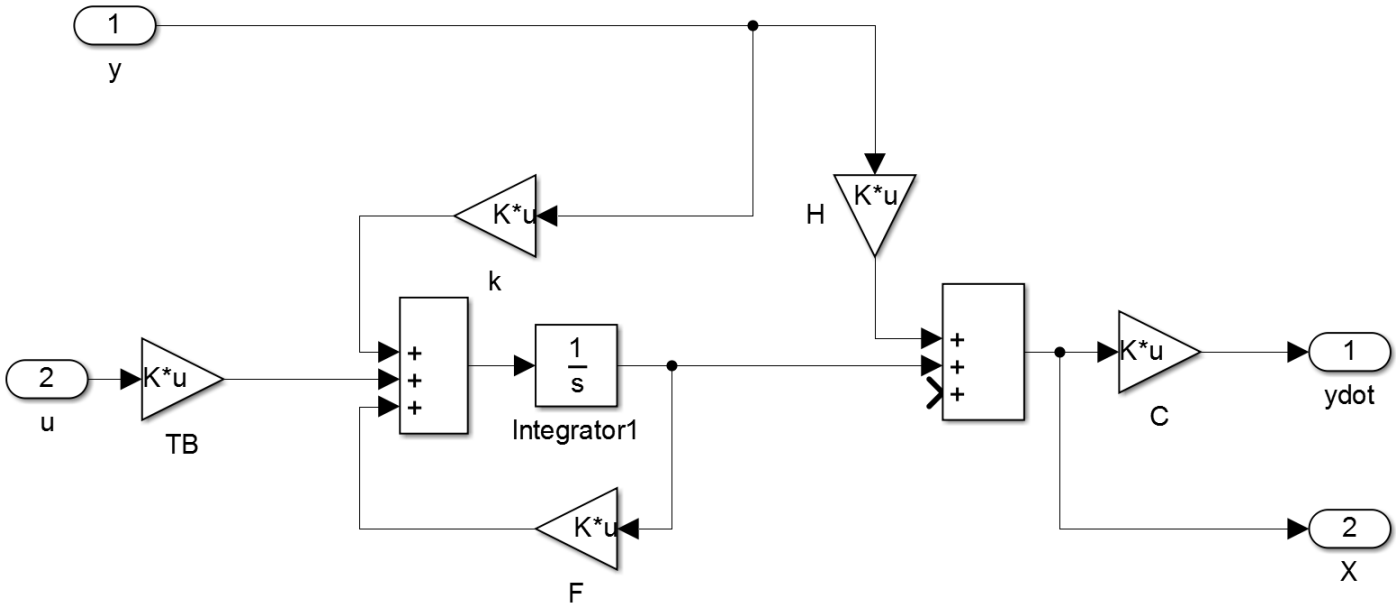

FIGURE 8. Simulink implementation of the Unknown Input Observer for the quarter car suspension model.

```
H =

    0.2500    0.2500    0.2500    0.2500
    0.2500    0.2500    0.2500    0.2500
    0.2500    0.2500    0.2500    0.2500
    0.2500    0.2500    0.2500    0.2500

A1 =

   1.0e+03 *

   -0.0533   -0.0026    0.8450    0.0031
   -0.1067   -0.0069    0.8983    0.0064
   -0.0533   -0.0036    0.8450    0.0041
    0.2133    0.0131   -2.5883   -0.0136

v =

    -4    -4    -4    -4


k1 =

   1.0e+03 *

   -0.0493   -0.0026    0.8450    0.0031
   -0.1067   -0.0029    0.8983    0.0064
   -0.0533   -0.0036    0.8490    0.0041
    0.2133    0.0131   -2.5883   -0.0096
```

```
T =

    0.7500   -0.2500   -0.2500   -0.2500
   -0.2500    0.7500   -0.2500   -0.2500
   -0.2500   -0.2500    0.7500   -0.2500
   -0.2500   -0.2500   -0.2500    0.7500

f =

         0    0    0    0
         0    0    0    0
         0    0    0    0
         0    0    0    0

F = eye(-4)


k2 = 4X4 matrix '-1'
```

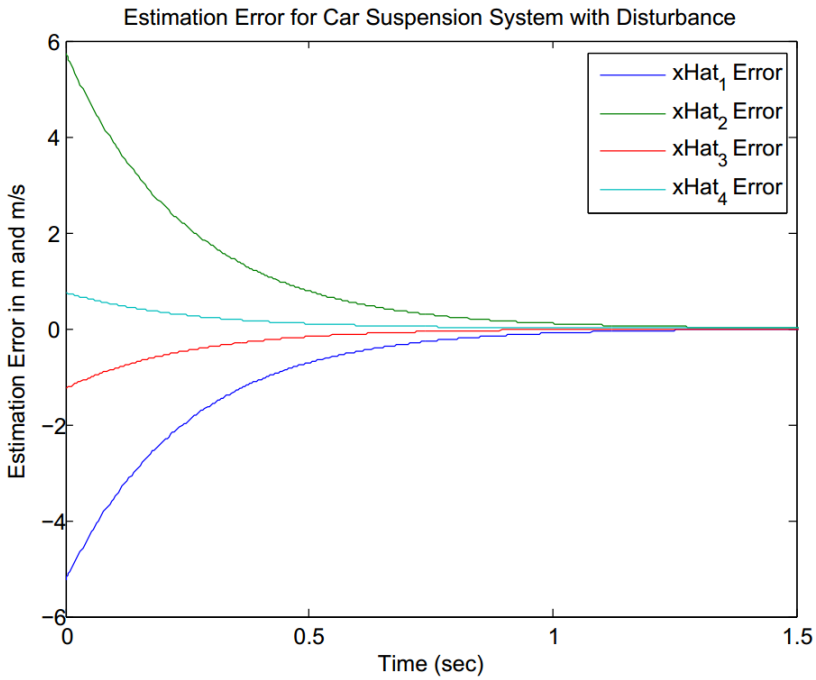

FIGURE 9. UIO results for the car suspension model.

```
k =

   1.0e+03 *

   -0.0503   -0.0036    0.8440    0.0021
   -0.1077   -0.0039    0.8973    0.0054
   -0.0543   -0.0046    0.8480    0.0031
    0.2123    0.0121   -2.5893   -0.0106
```

Next we form $A1$ and check observability

```
rank(obsv(A1,C))
```

```
ans =

     4
```

Since the observability matrix has full rank, we can go on and simulate the system to obtain the dynamic estimator performance. The initial conditions are set to be:

```
X_0 = [-1;10;3;5]
```

```
X_0 =

    -1
    10
     3
     5
```

Figure 9 shows the state estimation error for the UIO used in the car suspension model. The observer seems to converge within 1.5 seconds. Figures 8 shows the implementation of the UIO in Simulink. All the files for this simulation can be found in the FDIR folder. Specifically, the file

`carsus.m`

and the Simulink model

`car_sus.xsl`

contain everything.

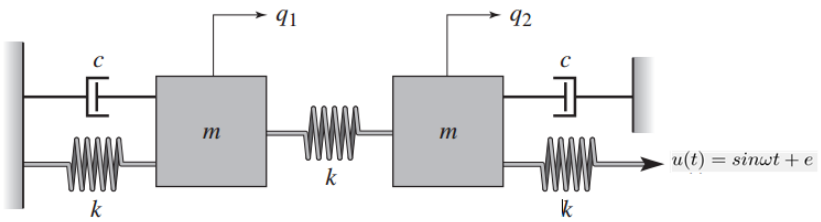

FIGURE 10. The free body diagram for the two mass and spring system.

3.3. **Coupled Mass-Spring System.** Consider the coupled spring mass system shown in Figure 10. The coupled mass spring system is a well studied mechanical system in control theory. The input to this system is the sinusoidal motion of the end of the rightmost spring, and the output is the position of each mass, q_1 and q_2. The disturbance input is also shown in the figure as e. In state space form, the equations of motion are:

$$x = \begin{bmatrix} q_1 \\ q_2 \\ \dot{q}_1 \\ \dot{q}_2 \end{bmatrix}, \quad A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ \frac{-2k}{m} & \frac{k}{m} & \frac{-c}{m} & 0 \\ \frac{k}{m} & \frac{-2k}{m} & 0 & \frac{-c}{m} \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \frac{k}{m} \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}, \quad E = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{3.3}$$

In MATLAB, we assign the following values to the parameters:

```
%----------System Parameters---------%
k   = 1   %
m   = 1   % kg
c   = 1 %
```

```
k =          m =          c =

     1            1                1
```

to obtain the following state space formulation:

```
%----------State Space Formulation---------%
A   = [0 0 1 0;
       0 0 0 1;
       -2*k/m k/m -c/m 0;
       k/m -2*k/m 0 -c/m]

B   = [0;0;0;k/m]
C   = [1 0 0 0;0 1 0 0;0 0 1 0;0 0 0 1]
D   = 0
E   = [0;0;0;1]
```

```
A =                          B =             C =                   D =        E =

     0     0     1     0              0                1     0     0     0        0              0
     0     0     0     1              0                0     1     0     0                       0
    -2     1    -1     0              0                0     0     1     0                       0
     1    -2     0    -1              1                0     0     0     1                       1
```

In order to control the placement of the masses in the system, we utilized LQR. The following MATLAB code accomplishes this objective:

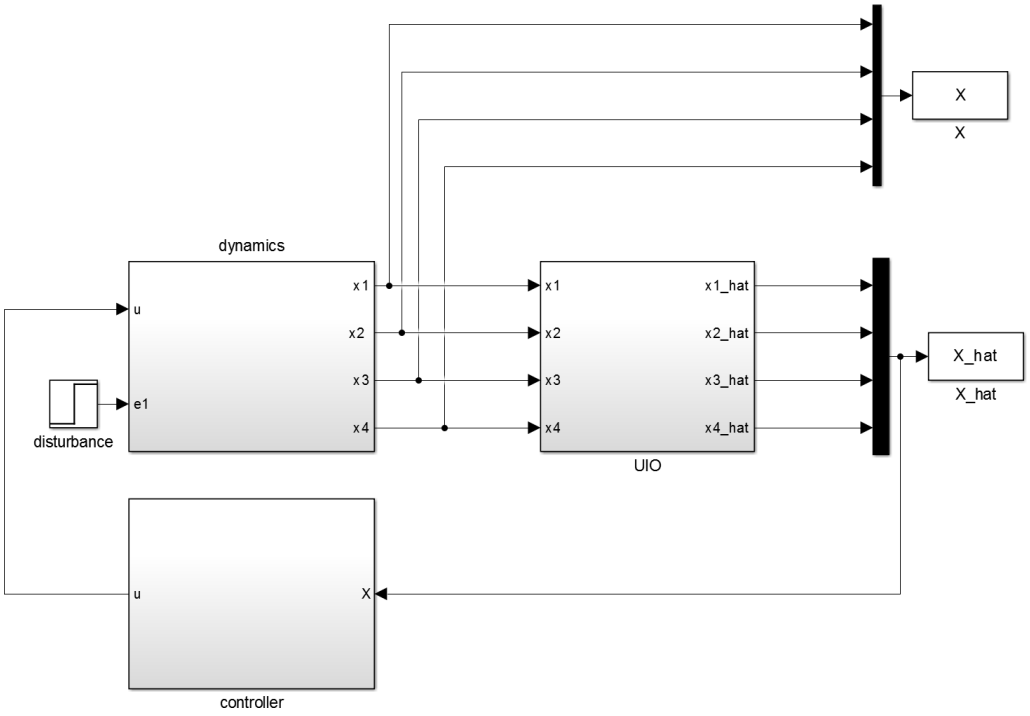

FIGURE 11. The root level Simulink block diagram for the coupled mass spring dynamic system.

```
%----------LQR Control---------%
Q=[100 0 0 0;
    0 100 0 0;
    0 0 100 0;
    0 0 0 100]
R=1
[Klqr ll ss]=lqr(A,B,Q,R)
```

```
Q =                     R =       Klqr =

   100   0     0     0         1           -1.3620   10.4615    2.0165   10.0419
     0   100   0     0
     0   0     100   0
     0   0     0     100
```

Now we are ready to begin the UIO design. The first step is to check that $rank(CE) = rank(E)$:

```
%----------Step 1: rank(CE) = rank(E) ?= 1---------%
rank(C*E)
rank(E)
```

```
ans =     ans =

     1          1
```

Next we are ready to compute the first set of observer matrices:

```
%----------Step 2: Compute observer matrices---------%
H = E*inv((C*E)'*(C*E))*(C*E)'
T = eye(4)-H*C
A1 = T*A
```

```
H =                 T =                     A1 =

     0   0   0   0       1   0   0   0              0   0   1   0
     0   0   0   0       0   1   0   0              0   0   0   1
     0   0   0   0       0   0   1   0             -2   1  -1   0
     0   0   0   1       0   0   0   0              0   0   0   0
```

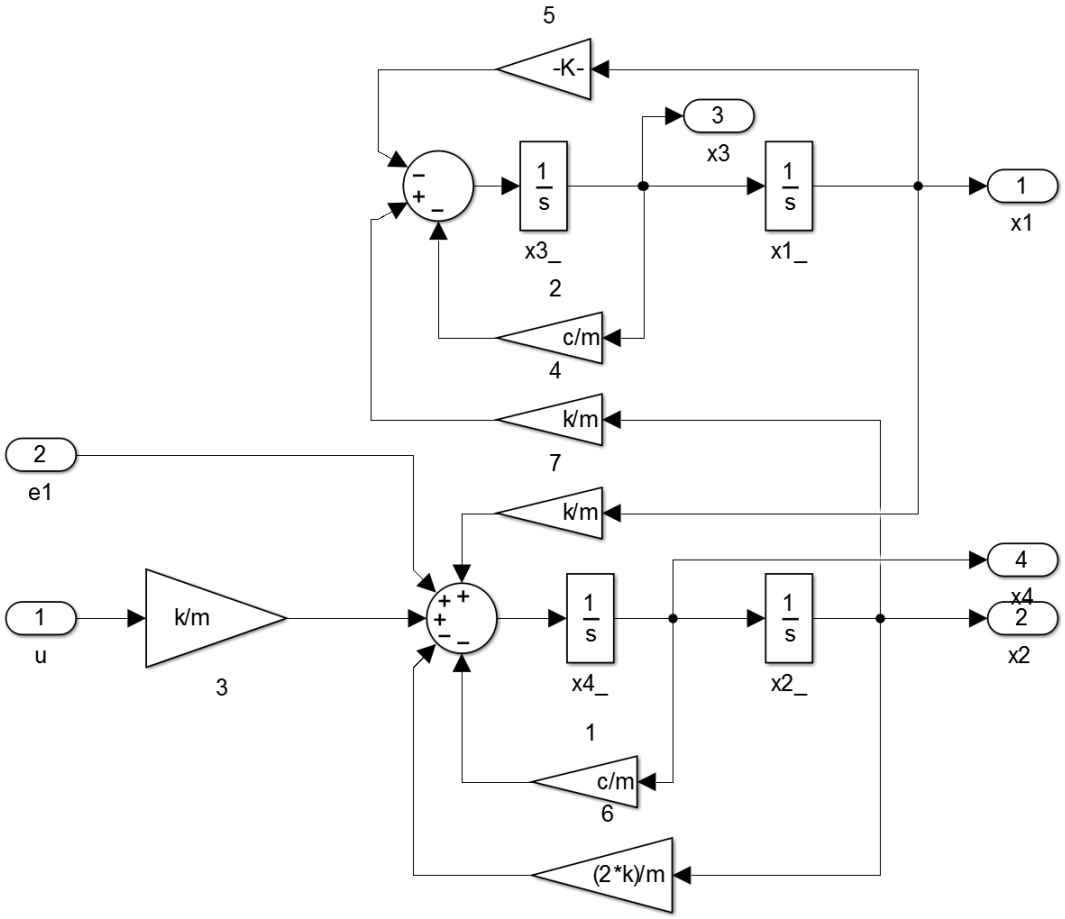

FIGURE 12. The mass spring system dynamics implemented in linearized form in Simulink.

Now that A_1 has been obtained, we must check the observability of the pair (C, A_1):

```
%----------Step 3:Check (C,A1) rank---------%
rank(obsv(A1,C))
```

```
ans =

     4
```

Since the pair (C, A_1) is full rank, it is observable. Therefore, we can use pole placement for the observer gain K_1:

```
%----------Step 4: Choose observer poles---------%
K1 = place(A',C',[-2,-10,-5,-3])'
```

```
K1 =

     2     0     1     0
     0    10     0     1
    -2     1     4     0
     1    -2     0     2
```

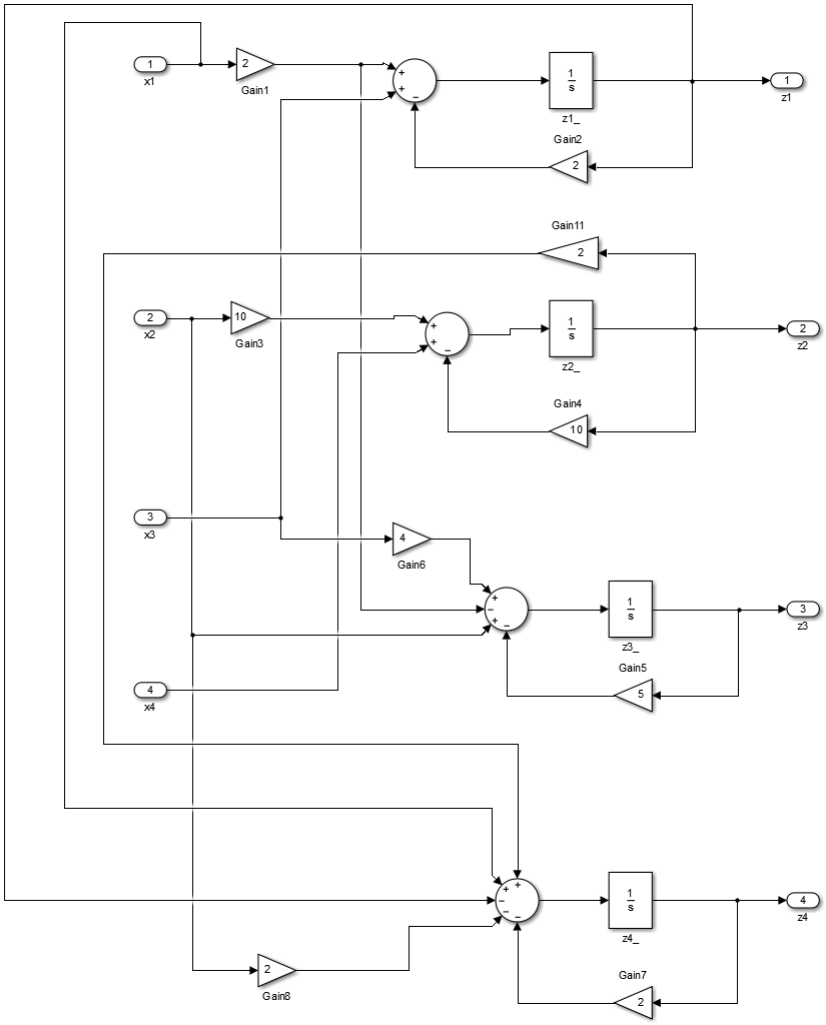

FIGURE 13. The mass spring system dynamics implemented in linearized form in Simulink.

and to conclude the observer design we compute the F and K matrices:

```
%----------Step 5: Finish observer design---------%
F = A1-K1*C
K = K1 + F*H
```

```
F =                         K =

    -2   0    0    0            2    0    1    0
     0 -10    0    0            0   10    0    1
     0   0   -5    0           -2    1    4    0
    -1   2    0   -2            1   -2    0    0
```

The simulation results are considered next. The top level Simulink block diagram is shown in Figure 11. The mass spring system dynamics are implemented as shown in Figure 12. The UIO implementation is shown in Figure 13. The state estimation error is shown in Figure 14. The following initial conditions were used for the simulation.

```
x1_0        = -1;
x2_0        = -5;
x3_0        = 1;
x4_0        = 1;
```

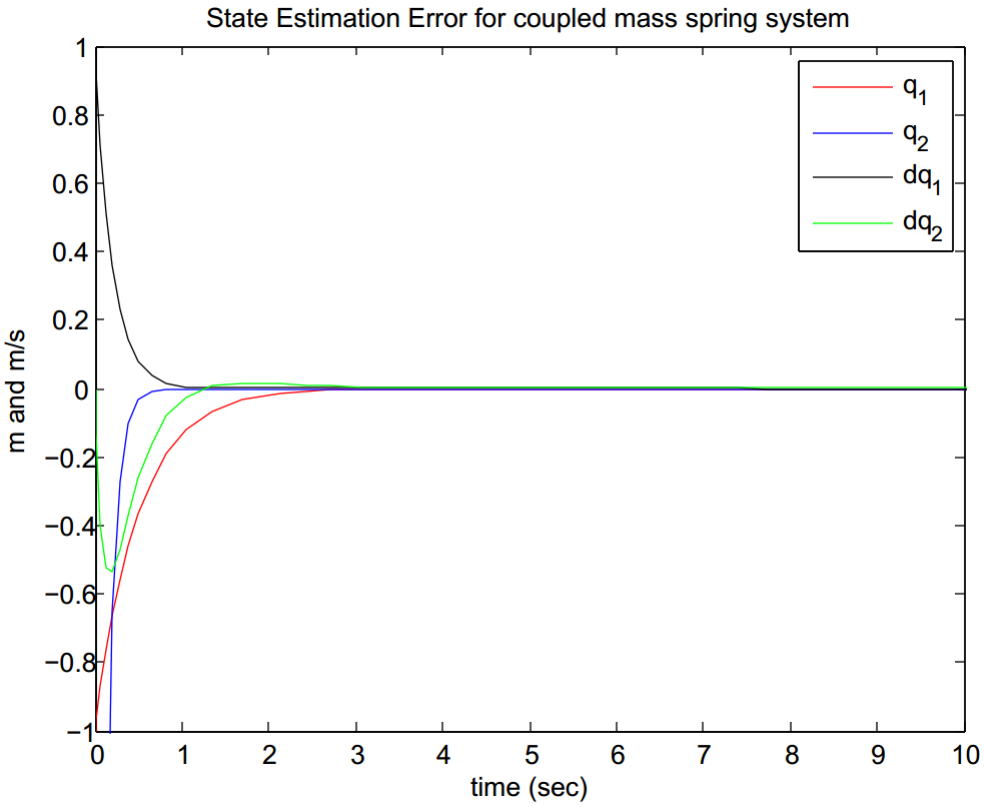

FIGURE 14. The state estimation error for the mass spring system using the UIO.

4. Conclusion

This memo begins by motivating the UIO paradigm for state estimation. A main advantage of this approach is to decouple the plant disturbance inputs from the state estimation process. The UIO design procedure was reviewed and illustrated with three common examples found in control theory. The estimation error for the coupled mass spring system, the car suspension system and the linear system with no inputs was considered. In each example, it was shown that this error asymptotically approaches zero by utilization of a UIO.

References

[1] Ron J. Patton Jie Chen, *Robust model-based fault diagnosis for dynamic systems*, Springer, New York, 1999.
[2] R. Clark R. Patton, P. Frank, *Fault diagnosis in dynamic systems: Theory and applications*, Prentice Hall, New York, 1989.
[3] Thoralf A. Schwarz, *Uncertainty Analysis of a Fault Detection and Isolation Scheme for Multi-Agent Systems*, Master's thesis, KTH, Sweden, 2012.
[4] Bahram Shafai, *Proportional integral observer in robust control, fault detection and decentralized control of dynamic systems*, To appear in book **21** (2015), 1087–1091.

Department of Electrical & Computer Engineering, Northeastern University, Boston, MA,02210
E-mail address: nazari.s@neu.edu