Qiao Gao
ID: A20282211
ECE 531
Final Project: A motion estimate algorithm for video coding
05/03/2013

1. System introduction (A motion estimate algorithm for video coding)

Introduction

Video coding is a technology that is useful in video compression, which will highly improve the efficiency of video transmission and storage. As described in [1], an important method for video coding called motion compensation plays a key role in improving coding efficiency.

The main idea of video compression to achieve compression is to remove spatial and temporal redundancies existing in video sequences [1]. In order to achieve this, firstly we have to detect the motion vector (MV) between the reference frame and current frame. Then, the motion estimation will be generate during the frame scan.

Block matching algorithm

This algorithm is used to test the motion vector of the video. Firstly, we divide the frame area into N*N pixels blocks as the motion element we want to test. Then, we adjust test blocks of (2P+1)*(2P+1) pixels as a test window with the maximum allow displacement value P. The principle is shown in figure 1. From this algorithm we can get motion vector for each test block.

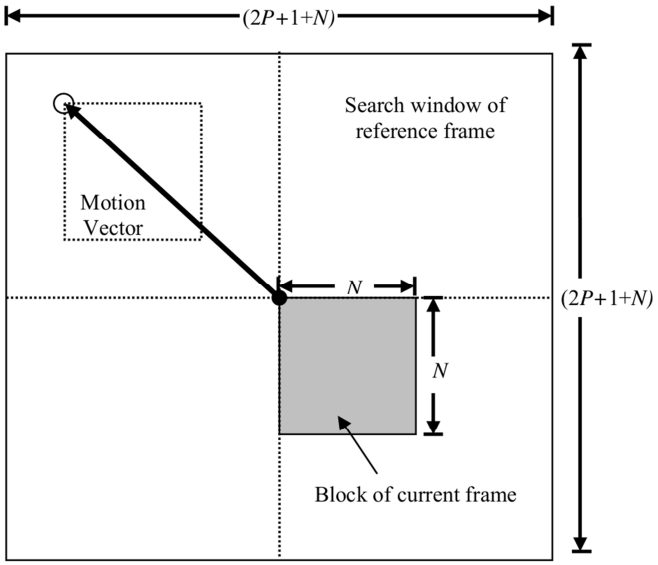


Figure 1

Spatial causal AR models for MV

In the introduction above, we know the motion compensation algorithm is consisted of spatial and temporal causal model. To make it simple, we just consider the spatial causal AR model in this project, which means the MV information of current frame is only generated by its 2D nearby blocks.

Let B(m,n,i) be the block at the location (m,n) in the i_{th} frame, and $V(m,n,i)=[v_x(m,n,i),v_y(m,n,i)]^T$ be the MV of B(m,n,i), where $v_x(m,n,i)$ and $v_y(m,n,i)$ denote the horizontal and vertical components, respectively. Assume that the MV is a random process, and the two components are independent. So we assume v(m,n,i) to be the MV of one of the two directions. In the block matching, the calculation of matching criterion is performed block-by-block in a raster scan manner, i.e., from left to right and top to bottom. Thus we can define the 2-D AR model for a motion vector as

$$v_{i,x} = \sum \sum_{(k,l)\in S^+} a_{kl0} v_{i,x}(m-k, n-l, i) + w_{i,x}(m,n,i)\,,$$

$$v_{i,y} = \sum \sum_{(k,l)\in S^+} a_{kl0} v_{i,y}(m-k, n-l, i) + w_{i,y}(m,n,i)\,,$$

where $S^+ = \{k \geq l,\ \forall l\} \cup \{k = 0,\ l \geq 1\}$ is the model support, and a_{kl0} are the model coefficients. For simplicity, we assume that the model is space invariant. Then, by just choosing blocks nearby the current block, we get the simplified representation below:

$$\begin{aligned} v_{i,x}(m,n,i) &= a_{100} v_{i,x}(m-1,n,i) + a_{-110} v_{i,x}(m+1,n-1,i) \\ &\quad + a_{010} v_{i,x}(m,n-1,i) + a_{110} v_{i,x}(m-1,n-1,i) + w_{i,x}(m,n,i), \end{aligned}$$

$$\begin{aligned} v_{i,y}(m,n,i) &= a_{100} v_{i,y}(m-1,n,i) + a_{-110} v_{i,y}(m+1,n-1,i) \\ &\quad + a_{010} v_{i,y}(m,n-1,i) + a_{110} v_{i,y}(m-1,n-1,i) + w_{i,y}(m,n,i). \end{aligned}$$

The basic principle is shown in figure 2.

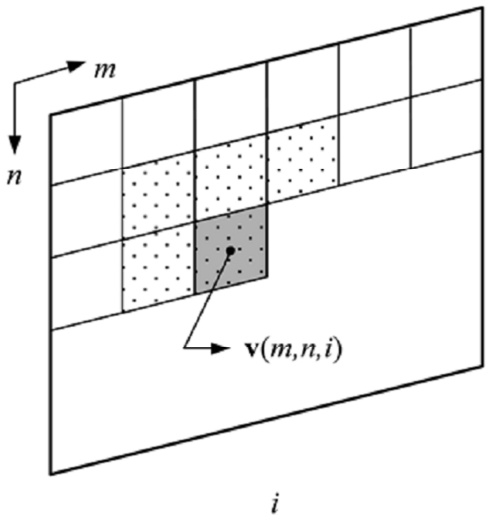


Figure 2

State space representation of spatial local AR model

From the expression above, we can get the state space equations below:

v(m,n,i)=A* v (m-1,n,i)+B*U(m,n,i)+T*w(m,n,i)

z(m,n,i)=C* v (m,n,i)+e(m,n,i)

in which all the values are defined as below:

$$\mathbf{v}(m,n,i) = \begin{bmatrix} v(m,n,i) \\ v(m-1,n,i) \\ v(m+2,n-1,i) \\ v(m+1,n-1,i) \\ v(m,n-1,i) \end{bmatrix}, \mathbf{v}(m-1,n,i) = \begin{bmatrix} v(m-1,n,i) \\ v(m-2,n,i) \\ v(m+1,n-1,i) \\ v(m,n-1,i) \\ v(m-1,n-1,i) \end{bmatrix},$$

$$\mathbf{u}(m,n,i)=\hat{v}(m+2,n-1,i),\qquad \mathbf{w}(m,n)=\begin{bmatrix} w(m,n,i) \\ w(m+2,n-1,i) \end{bmatrix},$$

$$A=\begin{bmatrix} a_{100} & 0 & a_{-110} & a_{010} & a_{110} \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \quad B=\begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \text{ and } \quad T=\begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}.$$

Measurement equation:

z(m,n,i)=C* v (m,n,i)+e(m,n,i)

where C=[1 0 0 0 0].

It can be noticed from the values above that only v(m+2,n-1,i) cannot be expressed by previous state, so we involve $\hat{v}(m+2,n-1,i)$ as the introduced deterministic input which is the most recent estimation of block (m+2,n-1,i) that we have already get before the current prediction.

The w and e are the noise in both prediction and measurement, which can be treated like a constant that we got from the pre-test. z is the measurement value that will be sent to the other modules to achieve more functions.

Since w is a constant noise that can be rejected in our design and e can be ignored, we take y[k]=z[m-1,n,i], c=[a_{100} 0 a_{-110} a_{101} a_{110}], x[k]=v(m-1,n,i),x[k+1]=v(m,n,i) and A=A, b=B. Then we get a standard linear state space expression. Next step is to put general values inside to make an instance. Let a_{100}=7/c0, a_{-110}=2/c0, a_{101}=7/c0, a_{110}=2/c0 where c0=26. These are general values provided in [1].

Finally, the system can be described as below:

x[k+1]=Ax[k]+bu[k];

y[k]=cx[k];

where:

```
A =                                                    b =

    0.2692         0    0.0769    0.2692    0.0769         0
    1.0000         0         0         0         0         0
         0         0         0         0         0         1
         0         0    1.0000         0         0         0
         0         0         0    1.0000         0         0

c =                                                    d =

    0.2692         0    0.0769    0.2692    0.0769         0
```

I use the following code to describe the system in MATLAB:

```
c0=26;
a100=7/c0;
am110=2/c0;
a010=7/c0;
a110=2/c0;
A=[a100 0 am110 a010 a110;1 0 0 0 0;0 0 0 0 0;0 0 1 0 0;0 0 0 1 0];
b=[0;0;1;0;0];
c=[a100 0 am110 a010 a110];
d=0;
```

2. Preparation

Eigenvalue and stability

In order to make a discussion on system stability, the eigenvalues of the system should be calculated first. All the calculations in this paper is based on MATLAB platform. In this paper, I use two methods to generate the eigenvalues and eigenvectors.

Firstly input

[Veig,Deig]=eig(A);

The result is:

```
Veig =

         0    0.2600   -0.0000    0.0000   -0.0000
    1.0000    0.9656    1.0000   -1.0000    1.0000
         0         0         0         0         0
         0         0         0         0         0
         0         0    0.0000   -0.0000    0.0000

Deig =

         0         0         0         0         0
         0    0.2692         0         0         0
         0         0         0         0         0
         0         0         0         0         0
         0         0         0         0         0
```

It is clear that it has four '0' eigenvalues and one 0.2692 eigenvalue. Then, type

[Vj,Jj] = jordan(A)

to get the Jordan form of A and eigenvectors. The result is:

```
Jj =

         0    1.0000         0         0         0
         0         0    1.0000         0         0
         0         0         0    1.0000         0
         0         0         0         0         0
         0         0         0         0    0.2692

Vj =

         0   -0.2857   -2.0612   -7.9417    7.9417
   -0.2857   -2.0612   -7.9417  -29.4977   29.4977
         0         0         0    1.0000         0
         0         0    1.0000         0         0
         0    1.0000         0         0         0
```

which indicates that the '0' eigenvalue has full rank.

In terms of stability, for DT system, because all the magnitude of eigenvalues is less than '1', we can assert that this system is BIBO stable, marginally stable and asymptotically stable.

Transfer function

As this system is an SISO system, we can get its transfer function. We can use the commands below to get all the coefficients in numerator and denominator polynomial of the transfer function and its poles and zeros.

```
[Ne,De]=ss2tf(A,b,c,d);
Zeros=roots(Ne);
Poles=roots(De);
```

The result is:

```
Ne =

         0    0.0769    0.2692    0.0769         0         0

De =

    1.0000   -0.2692         0         0         0         0

Zeros =     Poles =

         0         0
         0         0
   -3.1861         0
   -0.3139         0
                   0
              0.2692
```

So the numerator and denominator of the transfer function are not coprime and two '0's can be cancelled. The final transfer function after cancellation can be generated using the command below:

```
syms z;
I=eye(5);
g=factor(c*inv(z*I-A)*b);
```

The result is:

```
g =

(2*z^2 + 7*z + 2)/(z^2*(26*z - 7))
```

So it is obviously BIBO stable as I have mentioned above.

System's response

In DT system, the system response can be expressed as below:

In Z domain:

$x(z)=(zI-A)^{-1}[x(0)+bu(z)]$

$y(z)=c(zI-A)^{-1}[x(0)+bu(z)]$

In time domain:

$$x(k) = A^k x[0] + \sum_{m=0}^{k-1} A^{k-1-m} bu[m]$$

$$y(k) = cA^k x[0] + \sum_{m=0}^{k-1} cA^{k-1-m} bu[m]$$

In this case the input u is unpredictable, so we just talk about the zero input response. It can be easily noticed that the key to calculate zero input response is how we can get $(zI-A)^{-1}$ and A^k.

$(zI-A)^{-1}$ can be easily calculated directly by using MATLAB. The result is:

```
>> inv(z*I-A)

ans =

[     26/(26*z - 7),   0, (2*z^2 + 7*z + 2)/(z^3*(26*z - 7)), (7*z + 2)/(z^2*(26*z - 7)),   2/(z*(26*z - 7))]
[ 26/(z*(26*z - 7)), 1/z, (2*z^2 + 7*z + 2)/(z^4*(26*z - 7)), (7*z + 2)/(z^3*(26*z - 7)), 2/(z^2*(26*z - 7))]
[                0,   0,                                1/z,                         0,                  0]
[                0,   0,                              1/z^2,                       1/z,                  0]
[                0,   0,                              1/z^3,                     1/z^2,                1/z]
```

We can also generate $c(zI-A)^{-1}$:

```
>> factor(c*inv(z*I-A))

ans =

[ 7/(26*z - 7), 0, (2*z^2 + 7*z + 2)/(z^2*(26*z - 7)), (7*z + 2)/(z*(26*z - 7)), 2/(26*z - 7)]
```

For A^k, it is not easy to get it directly. Let Q=Vj which is the eigenvectors for the Jordan form we have got above, then A^k=Q[0 0 0 0 0; 0 0 0 0 0; 0 0 0 0 0; 0 0 0 0 0; 0 0 0 0 0.2692^k]Q^{-1} according to (3.47) in the textbook[2]. In MATLAB, I use the command below and get the result:

```
>> syms k
>> Vj*[0 0 0 0 0; 0 0 0 0 0;0 0 0 0 0;0 0 0 0 0;0 0 0 0 (7/26)^k]*inv(Vj)

ans =

[         (7/26)^k, 0,    (2724*(7/26)^k)/343,    (101*(7/26)^k)/49,    (2*(7/26)^k)/7]
[  (26*(7/26)^k)/7, 0, (70824*(7/26)^k)/2401, (2626*(7/26)^k)/343, (52*(7/26)^k)/49]
[                0, 0,                     0,                   0,                 0]
[                0, 0,                     0,                   0,                 0]
[                0, 0,                     0,                   0,                 0]
```

Note that this result is correct only when k>4 or k=4 because one of the eigenvalues has rank 4.

Controllability and observability

Firstly, we have to find whether this system is controllable or observable or not. In order to achieve this, the most efficiency method is finding its controllable and observability matrix by using [A, b] and [A, c] respectively. We can get it in MATLAB directly:

```
Cd=ctrb(A,b);
Od=obsv(A,c);
```

The result is:

```
Cd =

         0    0.0769    0.2899    0.1550    0.0417
         0         0    0.0769    0.2899    0.1550
    1.0000         0         0         0         0
         0    1.0000         0         0         0
         0         0    1.0000         0         0

Od =

    0.2692         0    0.0769    0.2692    0.0769
    0.0725         0    0.2899    0.1494    0.0207
    0.0195         0    0.1550    0.0402    0.0056
    0.0053         0    0.0417    0.0108    0.0015
    0.0014         0    0.0112    0.0029    0.0004
```

Then we check if they are of full rank or not to determine the controllability and observability:

```
>> rank(Cd)

ans =

     5

>> rank(Od)

ans =

     3
```

This result clearly indicates that this system is controllable but not observable.

Finding observable subsystem

Because this system is controllable, so we just talk about the procedure of generating observable system. As what have been discussed above, Od has rank 3 and its first three rows are linear independent. So we choose Po by command below:

```
Po=[Od([1:3],[1:5]);0 1 0 0 0;0 0 0 0 1]
```

The result with rank is:

```
Po =

    0.2692         0    0.0769    0.2692    0.0769
    0.0725         0    0.2899    0.1494    0.0207
    0.0195         0    0.1550    0.0402    0.0056
         0    1.0000         0         0         0
         0         0         0         0    1.0000

>> rank(Po)

ans =

     5
```

Then I use code below to get equivalent system:

```
Qo=inv(Po);
Ae=Po*A*Qo;
Be=Pe*b;
Ce=c*Qo;
```

The result is:

```
Ae =                                                Be =

         0    1.0000         0         0         0    0.0769
   -0.0000         0    1.0000         0         0    0.2899
         0   -0.0000    0.2692         0         0    0.1550
    7.2143  -24.2500   41.7857         0   -0.2857         0
   -3.5000   25.2500  -45.5000         0         0         0

Ce =

    1.0000   -0.0000         0         0         0
```

Finally, subsystem can be generated using following commands:

```
Asub=Ae([1:3],[1:3]);
Bsub=Be([1:3]);
Csub=Ce([1:3]);
```

The result is:

```
Asub =                          Bsub =

         0    1.0000         0    0.0769
   -0.0000         0    1.0000    0.2899
         0   -0.0000    0.2692    0.1550
```

```
Csub =

    1.0000   -0.0000         0
```

This subsystem is observable and obviously controllable because the whole system is controllable.

Realization

1. Controllable but not observable form:

Firstly, we can get the controllable form realization from the coefficient we got when calculating transfer function, which was stored in De and Ne. we can use the code below:

```
Ac=[-De([2:6]);1 0 0 0 0;0 1 0 0 0;0 0 1 0 0;0 0 0 1 0];
Bc=[1;0;0;0;0];
Cc=[Ne([2:6])];
```

The result is:

```
Ac =                                                Bc =

    0.2692         0         0         0         0      1
    1.0000         0         0         0         0      0
         0    1.0000         0         0         0      0
         0         0    1.0000         0         0      0
         0         0         0    1.0000         0      0
Cc =

    0.0769    0.2692    0.0769         0         0
```

This is a controllable but not observable realization because D(z) and N(z) are not coprime.

2. Observable but not controllable form:

The observable form realization can be got by simply implementing the following code:

```
Ao=Ac';
Bo=Cc';
Co=Bc';
```

The result is:

```
Ao =                                                Bo =

    0.2692    1.0000         0         0         0      0.0769
         0         0    1.0000         0         0      0.2692
         0         0         0    1.0000         0      0.0769
         0         0         0         0    1.0000           0
         0         0         0         0         0           0
Co =

     1     0     0     0     0
```

This realization is observable but not controllable because D(z) and N(z) are not coprime.

3. Observable and controllable form:

As we know, a minimal realization is both observable and controllable, which we can get from the

coprime form of the transfer function.

Firstly, I get the coefficient of the transfer function after zero pole cancellation by code below:

```
[N,D]=numden(g);
Nm=sym2poly(N);
Dm=sym2poly(D);
Nm=Nm/Dm(1);
Dm=Dm/Dm(1);
```

The result is:

```
Nm =

    0.0769    0.2692    0.0769

Dm =

    1.0000   -0.2692         0         0
```

Then use the same method to get its controllable form:

```
Am=[-Dm([2:4]);1 0 0;0 1 0];
Bm=[1;0;0];
Cm=[Nm([1:3])];
```

The result is:

```
Am =                              Bm =

    0.2692         0         0      1
    1.0000         0         0      0
         0    1.0000         0      0

Cm =

    0.0769    0.2692    0.0769
```

This is the realization that is both controllable and observable.

4. Neither observable nor controllable form:

We can achieve generating a neither observable nor controllable form by attaching the minimal system with a simple system which has no control entry and observe entry such as A=1, B=0, C=0.

State the system as below:

```
Ax=[Am [0;0;0];0 0 0 1];
Bx=[Bm;0];
Cx=[Cm 0];
```

The result is:

```
Ax =                                        Bx =

    0.2692         0         0         0      1
    1.0000         0         0         0      0
         0    1.0000         0         0      0
         0         0         0    1.0000      0

Cx =

    0.0769    0.2692    0.0769         0
```

This is a neither observable nor controllable form. We can check it with function “ctrb” and

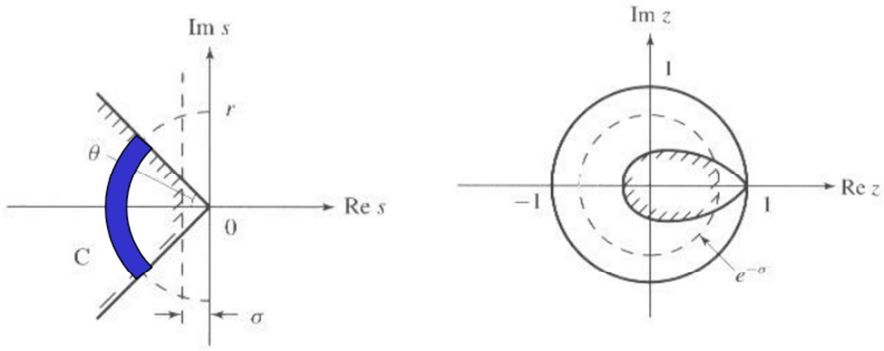
"obsv" and find both control matrix and observe matrix are not nonsingular which implies that it a neither observable nor controllable system.

3. Design

State feedback

I use the minimal realization (Am,Bm,Cm) as the plant system in this design, which is BIBO stable, asymptotically stable, controllable and observable.

The first problem is how to select the desired eigenvalues. As described in [2], a better system should have eigenvalues in the region below:



(a) CT system (b) DT system

A tradeoff has to be made among response speed, actuating signal and overshoot. In the DT condition, all the eigenvalues should be limited in the out light region in (b) so that the system can run faster with less overshoot and keep stable. However the zero point is not the best eigenvalue because it requires a very large actuating signal. As $z=e^{s}=e^{Re}[\cos(Im)+j*\sin(Im)]$, if we assume $\theta=\pi/2$, then by calculating $e^{-\pi}$ and $e^{-\pi/2}$ we can get an approximate range of the area.

$e^{-\pi}=0.0432$, $e^{-\pi/2}=0.2079$

Then I assume the three desired eigenvalues to be [0.3, +0.2j, -0.2j], which are in the region but not too close to zero point.

Next step is to calculate the state feedback vector K which will move the system eigenvalues to our target. Here I use two methods:

Method 1:

Using place function in MATLAB

```
Pt=[0.3 +0.2*j -0.2*j];
K1=place(Am,Bm,Pt);
```

Then get:

```
K1 =

   -0.0308    0.0400   -0.0120
```

Method 2:

Using Lyapunov equation:

Firstly, let $f=\begin{bmatrix} 0.3 & 0 & 0 \\ 0 & 0 & 0.2 \\ 0 & -0.2 & 0 \end{bmatrix}$

Then use the commands below:

```
f=[0.3 0 0; 0 0 0.2; 0 -0.2 0];
kb=[1 0 1];
t=lyap(Am,-f,-Bm*kb)
K2=kb*inv(t);
```

The result is:

```
K2 =

   -0.0308    0.0400   -0.0120
```

We find that K1=K2, so we can conclude the state feedback vector K=K1=[-0.0308 0.0400 -0.0120].

To get the feedforward gain p, in the DT case, the method is slightly different from the one for CT system, because t-> ∞ when z=1. As a result, p=sum(Df)/sum(Nm). Use the following commands:

```
Df=poly(K1);
p=sum(Df)/sum(Nm);
```

The result is:

```
p =

    2.3670
```

With this feedforward gain, the system will track any step inputs.

Observer design

Firstly calculate the full order observer with desired eigenvalue [0.1 +0.1j -0.1j] to make the observer faster because all the desired eigenvalues are closer to zero point. Then use the direct method with the commands below:

```
Fo=[0.1 0 0; 0 0 0.1; 0 -0.1 0];
Io=[1;0;1];
To=lyap(-Fo,Am,-Io*Cm);
```

Then the observer can be written in:

$z' = Fo*z + To*Bm*u + Io*y$

$x_{ob} = To^{-1}*z$

where

```
Fo =                              Io =

    0.1000         0         0      1
         0         0    0.1000      0
         0   -0.1000         0      1

To =

   61.8182  -10.3846   -0.7692
  -18.0201    2.6923    0.7692
  -21.5925    7.6923         0
>> inv(To)

ans =

    0.0450    0.0450    0.0450
    0.1264    0.1264    0.2564
    0.6125    1.9125    0.1575
```

Then calculate reduced order observer with target eigenvalue [0.1+0.1j 0.1-0.1j]. Use commands below to get the result:

```
For=[0.1 0.1; -0.1 0.1];
Ior=[1;0];
Tor=lyap(-For,Am,-Ior*Cm);
```

Then the observer can be written in:

z'=For*z+Tor*Bm*u+Ior*y;

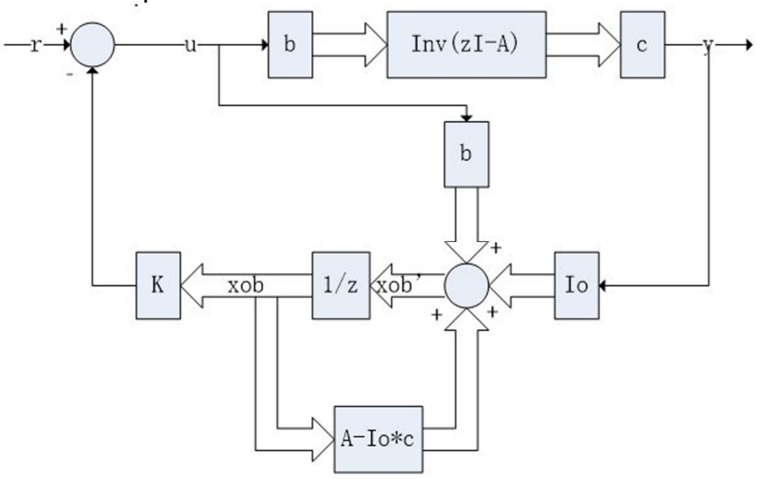
$x_{ob}=\begin{bmatrix} Cm \\ Tor \end{bmatrix}^{-1}\begin{bmatrix} y \\ z \end{bmatrix}$

In which

```
For =                   Ior =

    0.1000    0.1000        1
   -0.1000    0.1000        0

Tor =

   19.6708   -1.3462   -0.3846
   19.0582   -5.1923   -0.3846

>> [Cm;Tor]

ans =

    0.0769    0.2692    0.0769
   19.6708   -1.3462   -0.3846
   19.0582   -5.1923   -0.3846

>> inv(ans)

ans =

    0.2493    0.0499   -0.0000
   -0.0397    0.2521   -0.2600
   12.8897   -0.9321    0.9100
```

System connection

The system connection is shown below, using the observer output as the state feedback signal to make a closed loop.



In this case, I use the full order observer, then the total function can be expressed as:

$$\begin{bmatrix} x' \\ xob' \end{bmatrix} = \begin{bmatrix} Am & -Bm * K1 \\ Io * Cm & Am - Io * Cm - Bm * K1 \end{bmatrix} \begin{bmatrix} x \\ xob \end{bmatrix} + \begin{bmatrix} Bm \\ Bm \end{bmatrix} r$$

$$y = [Cm \quad 0] \begin{bmatrix} x \\ xob \end{bmatrix}$$

In which all the unknown matrices have been calculated above.

Robust tracking and disturbance rejection

Since g has no zero at z=0, we can get the vector Ka=[K3, ka], as what is described in [2], Figure 8.4, with state feedback and unity output feedback so that the system can track any step reference input and reject constant input disturbance.

Firstly, we should choose the eigenvalues for (n+1)*(n+1) matrix because Ka is 1*(n+1). Here we choose Pa=[Pt 0.1] in which I attach a 0.1 to the origin target eigenvalues for state feedback only. The code is shown below:

```
Pa=[Pt 0.1];
syms k1 k2 k3 ka;
K3=[k1 k2 k3];
I=eye(3);
deter=det([z*I-Am-Bm*K3 -ka*Bm; Cm z]);
coef=coeffs(deter,z);
coef=fliplr(coef);
Deltaf=poly(Pa);
```

By doing so:

```
coef =

[ 1, - k1 - 7/26, ka/13 - k2, (7*ka)/26 - k3, ka/13]
```

And

```
Deltaf =

    1.0000   -0.4000    0.0700   -0.0160    0.0012
```

And they should be the same. From this, I finally get:

k1= 0.1308;
k2= -0.0688;
k3= 0.0202;
ka= 0.0156;

Though the method to get Ka is the same in both CT and DT condition, the design is slightly different from that of CT condition. In order to cancel the unstable element that the disturbance brings in, we should use z/z-1 instead of 1/s in the feedforward network.

Compensator design

To get the minimal compensator, set m=n-1=2, the number of desired eigenvalues should be 2n-1=5. In this design, set them to be [0.3 +0.2j -0.2j 0.1+0.1j 0.1-0.1j]. Calculate Acom and Bcom using the method below:

```
Pcom=[0.3  +0.2j -0.2j 0.1+0.1j 0.1-0.1j];
Fcom=poly(Pcom);
Fcom=fliplr(Fcom);
Dcom=fliplr(Dm);
Ncom=fliplr(Nm);
Gcom=[Dcom;Ncom 0];
Sm=[Gcom zeros(2,2);zeros(2,1) Gcom zeros(2,1);zeros(2,2) Gcom];
```

This is the preparation for the calculation and the result is:

```
Fcom =

   -0.0002    0.0032   -0.0260    0.1200   -0.5000    1.0000
Sm =

         0         0   -0.2692    1.0000         0         0
    0.0769    0.2692    0.0769         0         0         0
         0         0         0   -0.2692    1.0000         0
         0    0.0769    0.2692    0.0769         0         0
         0         0         0         0   -0.2692    1.0000
         0         0    0.0769    0.2692    0.0769         0
```

Then get Acom and Bcom using commands below:

Ccom=Fcom/Sm;

Acom=[Ccom(1) Ccom(3) Ccom(5)];

Bcom=[Ccom(2) Ccom(4) Ccom(6)];

The result is:

```
Acom =

    0.1014   -0.2182    1.0000
Bcom =

   -0.0031    0.0525   -0.1639
```

Similarly in DT case, Both Ncom and Bcom do not have root at z=1. So it is possible to get the gain Pm which will make the system track any step reference input. Use the command below:

Pm=sum(Fcom)/sum(Bcom)/sum(Ncom);

The result is:

```
Pm =

  -12.3193
```

Reference

[1] Nai-Chung Yang, Chaur Heh Hsieh, Chung Ming Kuo. "Kalman Filtering Based Motion Estimation for Video Coding". Signal Processing Systems, 2008. SiPS 2008. IEEE Workshop on. Page(s): 129 - 134, 2008

[2] Chi-Tsong Chen. “Linear System Theory and Design(Fourth Edition)”. Oxford University Press. 2013