ROCO218: Control Engineering
Dr Ian Howard

Lecture 7

Hints for the coursework

2. Write down the state space model of the system

```
% consider equation
% d2theta/dt2 - a1*dtheta/dt -a2*theta + b0* ndu/dt + b1*u
% this equation captures the dynamics of the inverted pendulum if
b0 = params.m*params.lh/(params.I+params.m*params.lh^2);
b1 = 0;
a0 = 1;
a1 = params.mu/(params.I+params.m* params.lh^2);
a2 = params.direction * params.m*params.g*params.lh/(params.I+params.m*params.lh^2);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% build linearized state space matrices from differential equation
% include theta, thedaDot and cart position x as states
A = [0 1; -a2  -a1; ];
B = [b0; (b1-a1*b0);];
C = [1 0;];
D = 0;
```

```
% get inverted configuation
A_inverted = [0 1; -a2  -a1; ];
```

```
A_inverted =

         0    1.0000
   23.5440   -0.0270
```

```
% get non-inverted configuation
A_nonInverted = [0 1; a2  -a1; ];
```

```
A_nonInverted =

         0    1.0000
  -23.5440   -0.0270
```

3. Observability, controllability and stability

[10 marks]

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% build linearized state space matrices from differential equation
% include theta, thedaDot and cart position x as states
A = [0 1; -a2  -a1; ];
B = [b0; (b1-a1*b0);];
C = [1 0;];
D = 0;
```

```
A =

         0    1.0000
   23.5440   -0.0270


B =

    2.4000
   -0.0648
```

```
disp('observability matrix for 2x2 A system matrix system');
CA = C*A;
MXo=[C; CA;]
rank(MXo)
```

```
MXo =

     1     0
     0     1
```

rank = 2 therefore observable

```
% estimate the controllability
disp('controllablity matrix for 2x2 A system matrix system');
AB = A * B;
MXc=[B AB ]
rank(MXc)
```

```
MXc =

    2.4000   -0.0648
   -0.0648   56.5073
```

rank = 2 therefore controllable

3. Observability, controllability and stability

[10 marks]

```
% get inverted configuation
A_inverted = [0 1; -a2  -a1; ];

% eigenvalues of non-inverted configuration
eig(A_inverted)
```

```
ans =

    4.8387
   -4.8657
```

Includes a +ve eigenvalue
Therefore unstable

```
% get non-inverted configuation
A_nonInverted = [0 1; a2  -a1; ];

% eigenvalues of non-inverted configuration
eig(A_nonInverted)
```

```
ans =

  -0.0135 + 4.8522i
  -0.0135 - 4.8522i
```

Eigenvalues almost on imaginary axis
Therefore stable (would be marginally stable without damping term)

4. Simulate your state space module using the Matlab ode45 function

- state space dynamics equation is **[10 marks]**

$$\dot{X} = AX + BU$$

- write a Matlab function and pass it the parameters: A, B and u and return xDot

```
function xDot = SSSimulate(X, A, B, u)
% state space model
% x is state
% A,B are state space matrices
% u is ther control input
% xdot is the returned 1st time derivative of state

% implement model
xDot = A * X + B * u;
```

- The Matlab command ode45 will then integrate the state vector for you!

```
% compute output of linearized state space system
% ssmP.A is A matrix
% ssmP.B is B matrix
% x0 are initial conditions
% note input to state space model is full state feedback
[tSS, xDirectK] = ode45(@(t,y)SSSimulate(y, ssmP.A, ssmP.B,  -ssmP.K * y), params.t, x0);
```

5. Design a state feedback controller

[10 marks]

- Designing a state feedback controller merely involves calculating a suitable feedback gain vector K

```
% get inverted configuation
A_inverted = [0 1; -a2  -a1; ];
```

```
% compute SFC gains to set eigenvalues
PX=8 * [-1 -1.1 ];
ssm.K = place(ssm.A,ssm.B,PX);
ssm.K
```

```
ssm.K

ans =

    7.0336    1.6626
```

6. Implement the controller system using Euler integration

[10 marks]

- We need to solve the equations

$$\dot{X} = AX + BU \qquad Y = CX + DU \quad \text{Where} \qquad U = -KX$$

- This time we need to iteratively estimate the state vector X
- Thus starting by setting the initial conditions

$$X = X_0$$

- We then have to calculate the recurrence relationships in a loop

$$U(k) = -KX(k)$$

$$X(k+1) = X(k) + h\left(AX(k) + BU(k)\right)$$

- Where state X and input U are now described by a step index to make the recurrence clear
- Again we will need t define the time step h
- NB: In your implementation use element-wise multiplication because you to avoid using a matrix multiplication library on the Arduino!
- See example of this on next slide

Using Euler method to perform integration for SFC

```
function [y, t, xout] = SimulateSFCArduino2x2(A, B, C, D, K, t, x0)
% simulate state space feedback control using C-language compatible formulation
% performs integration by Euler's method
% A,B,C,D are state space matrices and k is state feeeback gain
% t is time at each sample,  x0 is initial state
% retuns outout y, each time t and full state at each update
% Author: Dr. Ian Howard% all rights reserved

% get signal length
len = length(t);

% init outout
y = zeros(1,len);
xout = zeros(2,len);

% record the initial state
xout(:, 1) = x0;
x = x0;

% calculate the command
u(1)= C(1) * x(1) + C(2) * x(2);

% calculate output from theta and thetaDot states
y(1)= C(1) * x(1) + C(2) * x(2) + D(1) * u(1);

% for all remaining data points, simulate state-space model using C-language compatible formulation
for idx = 2:len

    % state feedback rule
    u(idx) = - K(1) * x(1) - K(2) * x(2);

    % get the duration between updates
    h = t(idx) - t(idx-1);

    % calculate state derivative
    xdot(1) = A(1,1) * x(1) + A(1,2) * x(2) + B(1) * u(idx);
    xdot(2) = A(2,1) * x(1) + A(2,2) * x(2) + B(2) * u(idx);

    % update the state
    x(1) = x(1) + h * xdot(1);
    x(2) = x(2) + h * xdot(2);

    % record the state
    xout(:, idx) = x;

    % calculate output from theta and thetaDot states only
    y(idx)= C(1) * x(1) + C(2) * x(2) + D(1) * u(idx);
end
```

Initialize

Calculate initial state feedback

Calculate initial output

Loop over all data

Calculate control input

Get time step

xDot = AX + BU

Update the state

Record the state

Calculate output

7. Add a Luenberger observer to your state feedback controller

[10 marks]

- Designing a Luenberger observer merely involves calculating a suitable feedback gain vector L!

```
% get inverted configuation
A_inverted = [0 1; -a2  -a1; ];
```

```
% observer gain - use more agressive poles than for controller
PX=20 * [-1 -1.2];
ssm.L = place(ssm.A, ssm.C',PX);
ssm.L
```

```
ssm.L

ans =

   43.9730   21.3370
```

- Observer structure calculates a state estimate that can be used for state feedback control, rather than using the actual system state
- This is implemented using the estimated state update equation

$$\dot{\hat{X}} = A\hat{X} + BU + L(y - C\hat{X})$$

Pseudocode for implementing simulation with observer

Your Matlab Euler integration code now needs to do the two following estimations:

For each time point

For the simulated system

1 Compute state feedback control variable u using the SFC gain K and the estimated observer state xHat:

$$U = -K * XHat$$

2 Update the simulated system state X using Euler integration over time step h

$$X = X + h * (A * X + B * U)$$

3 Simulate the real output

$$y_{real} = C * X + B * U$$

For the observer

5 Calculate the observer correction term

$$y_{corr} = L*(y – C * XHat)$$

6 Update the observer state Xhat using Euler integration with correction term

$$Xhat = Xhat + h * (A * X + B * U + y_{corr})$$

NB: Using appropriate A and B values here will implicitly update the estimate of position of cart x_3 by integrating the input control U

8. Augment positional state into your state space model

[10 marks]

Remember: The state space representation of the velocity controlled inverted pendulum was

$$\frac{d}{dt}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -a_2 & -a_1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} b_0 \\ -a_1 b_0 \end{bmatrix} v_c$$

$$\Rightarrow \frac{d}{dt}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ \dfrac{mgl}{\left(I+ml^2\right)} & -\dfrac{\mu}{\left(I+ml^2\right)} \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} \dfrac{ml}{\left(I+ml^2\right)} \\ \dfrac{-\mu ml}{\left(I+ml^2\right)^2} \end{bmatrix} v_c$$

$$y = \begin{bmatrix} 1 & 0 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Where output y is the pendulum angle θ

Remember: *Augmented* velocity control inverted pendulum

- We can add a third state x_3 to the state space mode to represent the cart position
- Since the control signal is cart velocity, the differential of x_3 is simply given by the input velocity control signal
- Therefore we can write

$$\frac{d}{dt}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ -a_2 & -a_1 & 0 \\ 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_0 \\ -a_1 b_0 \\ 1 \end{bmatrix} v_c$$

$$y = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Now when we integrate the xDot vector, the new element x_3 will correspond to the cart position

9. Implement the augmented state feedback controller

The Matlab Euler integration code again needs to do the following:

[10 marks]

For each time point

For the simulated system

1 Compute state feedback control variable u using the SFC gain K and the estimated observer state xHat:

$$U = -K * xHat$$

2 Update the simulated system state X using Euler integration over time step h

$$X = X + h * (A * X + B * U)$$

3 Simulate the real output

$$y_{real} = C * X + B * U$$

For the observer

5 Calculate the observer correction term

$$y_{corr} = L * (y - C * xHat)$$

6 Update the observer state Xhat using Euler integration with correction term

$$Xhat = Xhat + h * (A * X + B * U + y_{corr})$$

NB: Using appropriate A and B values here will implicitly update the estimate of position of cart x_3 by integrating the input control U

11. Implement the augmented state feedback controller on the Arduino Mega

[10 marks]

- In the main Arduino program SFCIPROCO218 you first need to enter your own values for the system matrices A, B, C and D.
- The enter the SFC gains K
- Then enter the Luenberger observer gains L

```
// ENTER YOUR VALUES FOR THE SYSTEM MATRICES HERE
// system matrix definitions
double A[3][3] = {{xxx, xxx, xxx}, {xxx, xxx, xxx},  {xxx, xxx, xxx}};
double B[3] = {xxx, xxx, xxx};
double C[3] = {xxx, xxx, xxx};

// ENTER YOUR VALUES FOR THE SFC GAINS HERE
// SFC gains
double K[3] = {xxx, xxx, xxx};

// ENTER YOUR VALUES FOR THE OBSERVER GAINS HERE
// observer gain just for theta and thetadot
double L[2] = {xxx, xxx};
```

Implement SFC in the class CSFCROCO218.cpp

- In the Arduino the cpp class CSFCROCO218.cpp you now need to enter your implementation of state feedback control using a Luenberger observer within the empty ComputeSFC function

```
/////////////////////////////////////////////////////////////////
// compute the SFC
// given output angle of pendulum y and the current time
// returns the motor command u
double CSFCROCO218::ComputeSFC(double y, unsigned long theTime)
{
  // control value - stepper motor speed
  double    u = 0.0;

  // PUT YOUR OWN ComputeSFC FUNCTIONALITY IN HERE

  // Calculate time since last update

  // compute control variable u

  // calculate observer correction term

  // update the state estimates for theta and thetaHat

  // record variables

  //  use control velocity from input to update position of cart


  // return motor command
 return (u);
}
```

Implement SFC in the class CSFCROCO218.cpp

- For operation on the Arduino, your Euler integration code now only needs to implement the observer and use its state estimate xHat to generate the control command u:

For each time point

1 Compute state feedback control variable u on basis of SFC gain K and the estimated observer state xHat:

$$U = -K * xHat$$

2 Calculate the observer correction term using the real pendulum output

$$y_{corr} = L * (y - C * xHat)$$

3 Update the observer state Xhat using Euler integration with correction term

$$Xhat = Xhat + h *(A * X \ + B * U + \ y_{corr})$$

NB: Using appropriate A and B values here will implicitly update the estimate of position of cart x_3 by integrating the input control U