EBO780

Assignment 5
Theme 5: Introduction to Unconstrained MPC

MATLAB implementation of unconstrained linear model predictive control

Short memorandum

Assignment

The linear model for your model predictive control and plant is given by:

$$G(s) = \begin{bmatrix} \dfrac{12.8}{16.7s+1}e^{-1s} & \dfrac{-18.9}{21s+1}e^{-3s} \\ \dfrac{6.6}{10.9s+1}e^{-7s} & \dfrac{-19.4}{14.4s+1}e^{-3s} \end{bmatrix}$$

Questions

- **Formulate and define the MPC**

1. Define a discrete-time linear unconstrained MPC by giving the objective function and the optimization problem. Define all the variables/parameters involved. [10]

Optimization problem:

$$\min_{u(k+N_C|k),\dots,u(k|k)} J\big(u, x(k|k)\big)$$

Subject to:
(The system model.)

$$x(k+i+1|k) = f\big(x(k+i|k), u(k+i|k)\big) \qquad \forall i = 1, \dots, N_C$$

$$x(k+i+1|k) = f\big(x(k+i|k), u(k+N_C|k)\big) \qquad \forall i = N_C+1, \dots, N_P$$

$$y(k+i|k) = h\big(x(k+i|k), u(k+i|k)\big) \qquad \forall i = 1, \dots, N_C$$

$$y(k+i|k) = h\big(x(k+i|k), u(k+N_C|k)\big) \qquad \forall i = N_C+1, \dots, N_P$$

(The objective function.)

$$J(u,x) = \sum_{i=1}^{N_C} \Big(Y_{sp} - h\big(x(k+i|k), u(k+i|k)\big)\Big)^T Q \Big(Y_{sp} - h\big(x(k+i|k), u(k+i|k)\big)\Big) +$$
$$\sum_{i=N_C+1}^{N_P} \Big(Y_{sp} - h\big(x(k+i|k), u(k+N_C|k)\big)\Big)^T Q \Big(Y_{sp} - h\big(x(k+i|k), u(k+N_C|k)\big)\Big) +$$
$$\sum_{i=1}^{N_C} \Delta u(k+i|k)^T R \Delta u(k+i|k)$$

(The variable dimensions.)

$u \in \mathbb{R}^{N_C \times N_U}$	System input
$x \in \mathbb{R}^{N_P \times N_X}$	System states
$y \in \mathbb{R}^{N_P \times N_Y}$	System output

$x(k|k) \triangleq$ Initial state

(The MPC design variables.)

N_C	Control moves
N_P	Prediction samples

$Q \triangleq$ Positive definite weighting matrix for output

$R \triangleq$ Positive definite weighting matrix for change in input

(Note, blocking is not explicitly defined above.)

- **Create the discrete state-space model**

2. Convert the continuous-time model to a discrete-time model with sampling time of 1 second and document the model. (Use *c2d* in Matlab.) [5]

$$G_d(z) = \begin{bmatrix} z^{-1}\dfrac{0.744}{z-0.9419} & z^{-3}\dfrac{-0.8789}{z-0.9535} \\ z^{-7}\dfrac{0.5786}{z-0.9123} & z^{-3}\dfrac{-1.302}{z-0.9329} \end{bmatrix}$$

3. Convert the discrete-time model to a state-space representation and document the model. (Use *ss* in Matlab.) [5]

```
G_sys_d_ss =

  A =
         x1      x2      x3      x4
   x1  0.9419       0       0       0
   x2       0  0.9123       0       0
   x3       0       0  0.9535       0
   x4       0       0       0  0.9329

  B =
      u1  u2
   x1   1   0
   x2   1   0
   x3   0   1
   x4   0   1

  C =
          x1       x2       x3       x4
   y1   0.744        0  -0.8789        0
   y2       0   0.5786        0   -1.302

  D =
      u1  u2
   y1   0   0
   y2   0   0

  (values computed with all internal delays set to zero)
  Input delays (sampling periods): 1  3
  Internal delays (sampling periods): 6

Sample time: 1 seconds
Discrete-time state-space model.
```

4. The model in Question 3 does not model the time delays explicitly. Therefore, create a discrete-time state-space model with the time-delay expressed as explicit states, i.e. replace the delays of the discrete-time model by poles at zero. (See *absorbDelay* in Matlab.) This model will be used by the MPC. Document the discrete-time state-space model with delays as explicit states.
   Using this model, create a second discrete model which outputs all the states in the model, i.e., use an identity matrix as the *C*-matrix. This model will be used to represent the plant. See question 9 below. Document the discrete-time state-space model. [5]

If you use "absorbDelay" on *G_sys_d_ss* above, you get the system below. It has 14 states where each delay is modelled explicitly as a state. The model has 2 inputs and 2 outputs and will be used by the MPC controller:

```
G_sys_d_ss_td =

  A =
          x1      x2      x3      x4     x5    x6    x7    x8    x9   x10   x11   x12   x13   x14
   x1  0.9419       0       0       0     0     0     0     0     0     0     1     0     0     0
   x2       0  0.9123       0       0     0     0     0     0     0     0     1     0     0     0
   x3       0       0  0.9535       0     0     0     0     0     0     0     0     1     0     0
   x4       0       0       0  0.9329     0     0     0     0     0     0     0     1     0     0
   x5       0       0       0       0     0     1     0     0     0     0     0     0     0     0
   x6       0       0       0       0     0     0     1     0     0     0     0     0     0     0
   x7       0       0       0       0     0     0     0     1     0     0     0     0     0     0
   x8       0       0       0       0     0     0     0     0     1     0     0     0     0     0
   x9       0       0       0       0     0     0     0     0     0     1     0     0     0     0
   x10      0  0.5786       0       0     0     0     0     0     0     0     0     0     0     0
   x11      0       0       0       0     0     0     0     0     0     0     0     0     0     0
   x12      0       0       0       0     0     0     0     0     0     0     0     0     1     0
   x13      0       0       0       0     0     0     0     0     0     0     0     0     0     1
   x14      0       0       0       0     0     0     0     0     0     0     0     0     0     0

  B =
        u1  u2
   x1    0   0
   x2    0   0
   x3    0   0
   x4    0   0
   x5    0   0
   x6    0   0
   x7    0   0
   x8    0   0
   x9    0   0
   x10   0   0
```

```
x11  1  0
x12  0  0
x13  0  0
x14  0  1

C =
      x1     x2     x3     x4     x5     x6     x7     x8     x9    x10    x11    x12    x13    x14
y1  0.744     0 -0.8789     0      0      0      0      0      0      0      0      0      0      0
y2      0     0      0 -1.302      1      0      0      0      0      0      0      0      0      0

D =
    u1  u2
y1   0   0
y2   0   0

Sample time: 1 seconds
Discrete-time state-space model.
```

We want a model which outputs all 14 states. The reason we want a system to output 14 states is for state-feedback to the MPC. To achieve this, we can simply create a new state-space model where the C matrix is an identity matrix:

G_sys_states = ss(G_sys_d_ss_td.A, G_sys_d_ss_td.B, eye(size(G_sys_d_ss_td.A,1)), 0, Ts);

- **Create the MPC Controller.**

5. Document the prediction horizon that will lead to 99% of steady-state for the slowest model to a unit step. (You can use *step* in Matlab to get a step-response of the LTI model.) [5]

The transfer function with the longest settling time is G12. It reaches 99% of steady-state at approximately 100s. Therefore, $N_P = \frac{100}{T_S}$, where $T_S = 1$ s.

6. Document the control horizon that you have chosen. [5]

You can choose a control horizon that makes sense to you. Note, the control horizon should be shorter than the prediction horizon. I considered how you combined the control moves with the control blocking to get the correct control horizon.
The less moves, you allow the controller to make, i.e. the smaller you make N_C, the more aggressive the control moves will be. The more moves you allow the controller to make, the more calculations you have to make, but the less aggressive the control actions will be.

7. Code the Matlab function *ObjFunc* that calculates the objective value that will be optimized by *fminunc*. Your MPC controller must implement blocking to allow for a smooth closed-loop MV trajectory. [10]

8. Show the code to simulate the model predictions in your objective function using the *A, B, C* and *D* matrices of the discrete time state-space model of Question 4 explicitly. Do not use *lsim* to simulate the system. [5]

Below is a short example of an ObjFunc in Matlab. Here are a few comments.

- Please see the line marked in red. The variable u0 represents the current control move at the plant. The MPC will choose a new value to where the control should move. Thus, there is a delta between u0 and the next new move the MPC calculates. It is important to include this change in u.

- I marked how you implemented the model predictions. The model predictions should work with the first discrete model shown above in Q4.

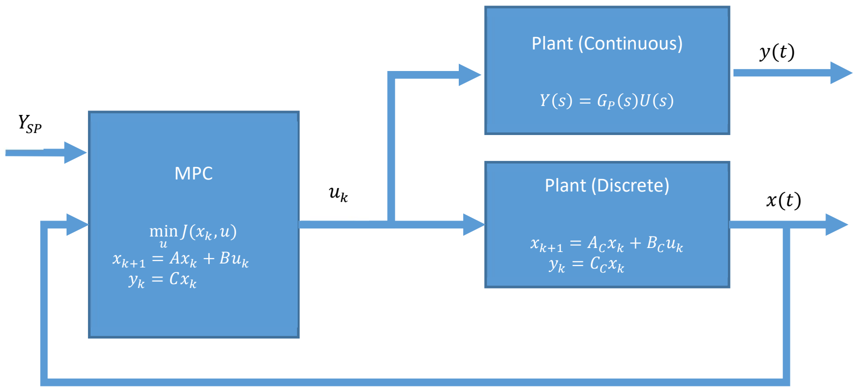
```
function [obj] = ObjFunc(u, x0, u0, Ysp, Np, Nc, Nb, Q, R, Model)
        [y, Usim] = PredictSim(u, x0, Np, Nb, Model);
        obj_output = 0;
        for (i = 1:Np)
                obj_output = obj_output + (Ysp-y(:,i))'*Q*(Ysp-y(:,i));
        end;

        obj_moves = (u(:,1) - u0)'*R*(u(:,1) - u0);
        for (i = 2:Nc)
                obj_moves = obj_moves + (u(:,i) - u(:,i-1))'*R*(u(:,i) - u(:,i-1));
        end;
        obj = obj_output+obj_moves;
```

- **Create the Simulation environment in Simulink**

9. Code a MATLAB Simulink s-function m-file to call *fminunc* on your function *ObjFunc*. This s-function will represent your MPC controller in Figure 1 above. You will connect the controller to a continuous version of the plant and to a discrete representation of the plant. The discrete version is necessary to feed all the states back to the controller. If you do not feed the states back, the control is open-loop. Remember to specify the sampling time of your s-function m-file to be 1 second. [10]

The general layout of your Simulink file should look more or like the figure below. This is an example of *closed-loop* MPC with *full-state feedback*. The reason for the *closed-loop* is that we measure the output of the plant and feed it back to the MPC. The reason for *the full-state feedback* is that the output that we measure and feed back to the MPC is all 14 states.

Figure 1: Control configuration in Simulink



The model for the *MPC* block is the first one created in Q4 with 14 states, 2 inputs and 2 outputs. We use this to predict the movement of y.
The model for the *Plant (Discrete)* block is the second one created in Q4 with 14 states, 2 inputs and 14 outputs. We use this model to output the 14 states after the input u as calculated by the MPC routine was implemented. These 14 states are fed back to the MPC. Remember, the MPC algorithm as defined in Q1 requires an initial value x(k|k). Thus, at every sampling instant, we are measuring the initial value x(k|k) and feeding it back to the MPC.
(The feedback of states will be different in Assignment 7 where we use a Kalman Filter to estimate the states rather than measuring them directly as above.)

- **Run the Simulation**

10. Run the Simulink model for 200 seconds and7 document the plot for the controlled variables and the manipulated variables. Use the fixed time step solver (*ode4*) with fundamental time-step of 1 second. This is set under the *Simulation->Configuration Parameters* menu. On the main screen under *Solver options* choose *Type* as *Fixed-step* and *Solver* as *ode4* (Runge-Kutta) and set *fixed-step size (fundamental sample time)* to 1 second. Start with the initial condition of all the states at 0 and then make a setpoint change from $[0, 0]^T$ to $[5, -5]^T$ at time 20 seconds and then to $[-5; 5]^T$ at time 100 seconds. [10]

Since each submission has a slightly different configuration, I marked each result separately.

11. Attach your MATLAB code to the end of the report. [-10 if not present]

- **A question on weighting matrices**

12. Assume you have to setup an MPC controller for a 2-by-2 system with $N_P = 20$ and $N_C = 3$. Determine what Q_1 and Q_2 should be in order for a 1% deviation of y_1 from its steady-state of 150 to contribute 5 times as much to the objective function as a 7% deviation of y_2 from its steady-state value of 320. Further, determine what R_1 and R_2 should be so that a 10% change relative to the range of u_1 of $0 - 400$ contributes the same to the objective function as a 20% change relative to the range of u_2 of $25 - 68$. Lastly, the contributions of y_1 and y_2 should be 60 times larger for the stated deviations than the contributions of u_1 and u_2, after the objective function was corrected for prediction and control horizon. Document your calculations in you report. [5]

(See the following paper for an example of how to tune MPC matrices:
Le Roux, J.D., Padhi, R., Craig, I.K., 2014, "Optimal Control of Grinding Mill Circuit using Model Predictive Static Programming: A New Nonlinear MPC Paradigm", J. Process Control, 24, pp. 29-40.)

Total: 75 marks