SSY280 Model Predictive Control

Assignment 1
Inverted pendulum

$\boxed{\text{Due February 9 at 12:00.}}$

Automatic Control

Department of Signals and Systems

Chalmers University of Technology

January 2015

1 Objectives

The purpose of this assignment is to balance an inverted pendulum using Model Predictive Control. The control problem is not very difficult and can be solved using conventional design techniques. The focus of the assignment is instead to get familiar with the details of a simple MPC algorithm by actually writing the Matlab code and validate the code by simulating the pendulum in closed loop.

2 Pendulum model

The pendulum is modeled as an ideal mathematical pendulum with length l and point mass m. The pendulum is subject to gravity force and is controlled by a motor at the pivot point, giving a steering torque τ. The task is to use the steering torque to stabilize the pendulum in the upward position without using too much power to the motor. The pendulum is described by the following differential equation:

$$ml^2\ddot{\theta} = \tau + mgl\sin\theta$$

where θ is the angle relative to the vertical, upward direction. Using the state variables $x_1 = \theta$ and $x_2 = \dot{\theta}$, the following approximate state model is obtained for small angles:

$$\begin{aligned}\dot{x}_1(t) &= x_2(t)\\ \dot{x}_2(t) &= g/l \cdot x_1(t) + 1/ml^2 \cdot \tau(t) = \alpha x_1(t) + \beta\tau(t)\end{aligned}$$

Since MPC is usually based on discrete time models, we need to convert this model into discrete time. Using a short sampling interval h, we get the following approximate model:

$$\begin{aligned}x(k+1) &= Ax(k) + Bu(k) = \begin{bmatrix}1 & h\\ \alpha h & 1\end{bmatrix} x(k) + \begin{bmatrix}\beta h^2/2\\ \beta h\end{bmatrix} u(k)\\ y(k) &= Cx(k) = \begin{bmatrix}1 & 0\end{bmatrix} x(k)\end{aligned}$$

where $u(k) = \tau(k)$. This will from now on be considered to be the true model of the system. The following parameters will be used: $\alpha = 0.5$, $\beta = 1$ (i.e. we let the parameter β be 'absorbed' into the control u), and $h = 0.1$.

3 Control design

The control of the pendulum will be based on the minimization of a quadratic cost function with a receding horizon N, i.e. at each time instant k, the following optimization problem should be solved:

$$
\begin{aligned}
\min_{u(k:k+N-1)} & q \cdot \|x(k+N)\|^2 + \sum_{i=0}^{N-1} \big(q \cdot y^2(k+i) + r \cdot u^2(k+i)\big) \\
\text{s.t. } & x(k+i+1) = Ax(k+i) + Bu(k+i), \quad i = 0, 1, \ldots, N-1
\end{aligned}
\tag{1}
$$

We have thus chosen to have only scalar weights q and r to simplify the tuning, and the stage cost contains penalties on the control input and the output angle.

4 Tasks

In the following tasks we ask you to implement a model predictive controller for the pendulum, based on the quadratic optimization problem given above. You are also expected to experiment with the algorithm and to reflect on the experience gained. The implementation of the algorithm should be made in Matlab, and the Appendix gives a suggested skeleton for your code. There are also some hints on how to implement the system matrices.

You should provide a brief written report with the answers to the questions, including short motivations, your observations from the simulations, and the Matlab program code. Try to be as clear and concise as possible! Figures included in the report should have legends, and axes should be labelled. The report should be uploaded to your project's document area in PingPong. At the same time, please notify Faisal via email. Please respect the naming conventions for files and email subjects!

4.1 Without actuator constraints

We will begin by considering the control problem without any state or control constraints.

(a) Rewrite the optimization problem (1) on the form

$$
\begin{aligned}
\min_{z} & \frac{1}{2} z^T H z \\
\text{s.t. } & A_{eq} z = b_{eq}
\end{aligned}
$$

where $z = \big(x^T(k+1)\ \cdots\ x^T(k+N)\ u(k)\ \cdots\ u(k+N-1)\big)^T$. Note that $x(k)$ is the current state, which is known and therefore not an optimization variable. Also note that the right hand side b_{eq} depends on the current state $x(k)$ and must be updated at each iteration of the MPC algorithm; see the skeleton code. Complete the Matlab code so that you can simulate your MPC algorithm with the pendulum. *Include a listing of the code with your report, as well as a documentation of the mathematical reformulation of the problem.*

(b) Simulate the pendulum, controlled by your MPC algorithm, with the initial condition $x(0) = (0.5\ 1)^T$ and with the following design parameters:

1. $N = 5$, $q = 3.8$, $r = 1$
2. $N = 10$, $q = 3.8$, $r = 1$
3. $N = 10$, $q = 10$, $r = 1$

Investigate the performance of the closed-loop system in the different cases. *Briefly summarize your findings, and provide an interpretation of what you see in the graphs. Include a couple of graphs (showing both input and output) illustrating your points with the report.*

(c) In the case without state or control constraints, there is no difference between the receding horizon controller and the corresponding controller obtained from solving a finite time LQ control problem and repeatedly applying this solution, as explained in the lectures. Verify your results from (b) by computing the controller directly from the (time-varying) Riccati equation and simulating the closed-loop system (e.g. by using the Matlab command `lsim`).

4.2 With actuator constraints

In reality, the actuator used to control the pendulum has limitations on the maximum torque available. Hence it is useful to take these constraints into account in the formulation of the MPC algorithm. The revised optimization problem becomes

$$\begin{aligned} \min_{u(k:k+N-1)} \; & q \cdot \|x(k+N)\|^2 + \sum_{i=0}^{N-1} \big(q \cdot y^2(k+i) + r \cdot u^2(k+i)\big) \\ \text{s.t.} \; & x(k+i+1) = Ax(k+i) + Bu(k+i), \quad i = 0, 1, \ldots, N-1 \\ & -1 \le u(k+i) \le 1, \quad i = 0, 1, \ldots, N-1 \end{aligned}$$

(d) Rewrite the new optimization problem on the form

$$\begin{aligned} &\min_z \frac{1}{2} z^T H z \\ &\text{s.t. } A_{eq} z = b_{eq} \\ &\qquad A_{in} z \leq b_{in} \end{aligned}$$

and apply the MPC with the initial condition $x(0) = (0.5\ 1)^T$ and with the following design parameters:

1. $N = 5$, $q = 3.8$, $r = 1$
2. $N = 10$, $q = 3.8$, $r = 1$
3. $N = 10$, $q = 10$, $r = 1$

Briefly summarize your findings with the report, and include a couple of graphs (*both inputs and outputs!*) illustrating your points.

4.3 Optimization algorithm performance

The Matlab implementation of the optimization routines carry with them quite a bit of overhead, which means that the computational efficiency is not what you could expect from a state-of-the-art algorithm, exploiting the structure of the problem and programmed directly in C, for example. Nevertheless, to get an idea of what is going on in the optimization step, go back to tasks 4.1 and 4.2. Use the Matlab functions `tic` and `toc` to log the execution times (*of the optimization only*) for the unconstrained and the constrained case, respectively—it is enough to look at one of the parameter settings. Plot the execution times as function of time and include a graph with your report.

In the code skeleton, there is an assignment of the variable `options`, which instructs the function `quadprog` to apply a so called interior point method. Change this statement by replacing `'interior-point-convex'` to `'active-set'` and repeat the logging of execution times as above. Compare the results and summarize your findings in the report.

4.4 Reducing the number of optimization variables

The vector of decision variables z contains both predicted states and control variables. However, the predicted states can be expressed in terms of the control variables $u(k)\ \ldots\ u(k+N-1)$. Hence, the optimization problem can be expressed in terms of the reduced vector of decision variables $u_N = \big(u(k)\ \ldots\ u(k+N-1)\big)^T$.

(e) Formulate the optimization problem on the form

$$\begin{aligned} \min_{u_N} & \frac{1}{2} u_N^T H u_N + f^T u_N \\ \text{s.t. } & A_{eq} u_N = b_{eq} \\ & A_{in} u_N \leq b_{in} \end{aligned}$$

Include your formulation in the report. You don't need to implement this in Matlab!

Matlab skeleton

```
%
%---Basic system model
%
clear;
h=0.1;
A=[1 h;0.5*h 1];
B=[h^2/2; h];
C=[1 0];
n=size(A,1);
m=size(B,2);
%
%---Parameters
%
q=3.8;
r=1;
N=5;
x0=[0.5 1]';
%
%---Define matrices for the QP
%

For you to do!

%
%---For the case with actuator constraints
%
Ain=[];   % Use empty matrices for the first case without actuator...
bin=[];    % ...constraints and change for the case with constraints!
%
%---Cost function
%

For you to do!

%
%---MPC algorithm
%
M=100;   % simulation time
```

```
xk=x0;   % initialize state vector
yvec=[];
uvec=[];

options = optimset('Algorithm','interior-point-convex','Display','off');

for k=1:M,
beq=AA*xk;      % The matrix AA defines how the last measured state xk
                % determines the right hand side of the inequality condition.
z=quadprog(H,f,Ain,bin,Aeq,beq,[],[],[],options);
uk=z(n*N+1);
xk=A*xk+B*uk;
yvec=[yvec; C*xk];
uvec=[uvec; uk];
end

tvec=h*(1:1:M);
subplot(3,1,1)   % For the other two sets of parameters you should change
                 % the third index to 2 and 3, respectively.

plot(tvec,yvec,'-'tvec,uvec,'--'); grid
```

Hints for the implementation

The following hints simplify the implementation of block matrices in Matlab:

- Block diagonal matrices can be created using the command `blkdiag(A,B)`.
- The command `kron(eye(N),M)` generates the block matrix

$$\begin{bmatrix} M & 0 & \cdots & 0 \\ 0 & M & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & M \end{bmatrix}$$

  and, more generally, `kron(A,M)` generates the block matrix

$$\begin{bmatrix} a_{11}M & a_{12}M & \cdots & a_{1n}M \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1}M & a_{n2}M & \cdots & a_{nn}M \end{bmatrix}$$