Inverted Pendulum on MPC

目录

0 准备工作 1
1 模型参数 1
2 控制器设计 1
2.1 状态预测 1
2.2 代价函数 3
2.3 选取控制量 4
3 Simulink 仿真 4

0 准备工作

系统建模

```
run("Inv_Pendulum_modeling.mlx");
```

1 模型参数

其中 I 应该根据杆子质量和长度计算得出，这里直接给出计算结果，如更新了质量和长度属性，则应该进行参数更新。

```
param = struct;

param.M = 0.5;
param.m = 0.2;
param.b = 0.1;
param.I = 0.018;
param.L = 0.3;

param.x_0 = 0;
param.y_0 = 0.15;
param.q_0 = 10; %degree
param.ref = [2;0;0;0];

param.Np = 10;
param.lower = -100;
param.upper = 100;

param.wheel_damping = 1e-5;
param.joint_damping = 1e-5;
```

2 控制器设计

2.1 状态预测

采用 MPC 控制器，下面对控制过程进行简要说明。

通常来说，我们列出的状态空间方程描述的是连续系统的状态变化：

$$\begin{cases} \dot{\boldsymbol{x}} = A\,\boldsymbol{x} + B\,\boldsymbol{u} \\ \boldsymbol{y} = C\,\boldsymbol{x} + D\,\boldsymbol{u} \end{cases}$$

但是 MPC 对系统采用离散化控制，即将时域分为一个个控制周期，在每个控制周期开头给系统输入恒定的控制量。因此，我们需要状态转移方程对状态空间方程做离散化处理，这个过程有两种方法：第一直接调用 MATLAB 函数 ss，但是这种方法在 simulink 的 MATLAB FUNCTION 模块中会失效；第二，是比较推荐的方法，是使用离散化的近似数值计算，在控制周期比较短的情况下采用这种计算方法可以有比较好的效果。

得到的离散状态空间方程，这里的系数矩阵 A、B 已经不是之前的系数矩阵了：

$$\boldsymbol{x}[k+1] = A\,\boldsymbol{x}[k] + B\,\boldsymbol{u}[k]$$

```
model = struct;

model.A = double(subs( ...
    Inpend.A, ...
    {sym("g"),Inpend.M,Inpend.m,Inpend.b,Inpend.I,Inpend.L}, ...
    {9.81,param.M,param.m,param.b,param.I,param.L}));

model.B = double(subs( ...
    Inpend.B, ...
    {sym("g"),Inpend.M,Inpend.m,Inpend.b,Inpend.I,Inpend.L}, ...
    {9.81,param.M,param.m,param.b,param.I,param.L}));

model.C = Inpend.C;
model.D = Inpend.D;

sys = ss(model.A,model.B,model.C,model.D);
sys_d = c2d(sys,0.1);
model.A = sys_d.a;
model.B = sys_d.b;
```

我们的目标是用初始状态 $x[0]$ 以及控制输入序列 $u[1], u[2], \ldots, u[n]$ 来预测接下来的状态序列 $x[1], x[2], \ldots, x[n]$.

我们有：

$$\boldsymbol{x}[1] = A\,\boldsymbol{x}[0] + B\,\boldsymbol{u}[1]$$

$$\boldsymbol{x}[2] = A\,\boldsymbol{x}[1] + B\,\boldsymbol{u}[2]$$

$$\cdots$$

$$\boldsymbol{x}[n] = A\,\boldsymbol{x}[n-1] + B\,\boldsymbol{u}[n]$$

每次将前式带入后式，我们可以有：

$$x[2] = A^2 x[0] + \mathrm{ABu}[1] + \mathrm{Bu}[2]$$

$$x[3] = A^3 x[0] + A^2\mathrm{Bu}[1] + \mathrm{ABu}[2] + \mathrm{Bu}[3]$$

...

$$x[n] = A^n x[0] + A^{n-1}\mathrm{Bu}[1] + \cdots + \mathrm{ABu}[n-1] + \mathrm{Bu}[n]$$

我们可以把这一系列式子写成矩阵表达式：

$$X = \begin{bmatrix} x[1] \\ x[2] \\ \vdots \\ x[n] \end{bmatrix}, U = \begin{bmatrix} u[1] \\ u[2] \\ \vdots \\ u[n] \end{bmatrix}$$

$$X = \begin{bmatrix} A \\ A^2 \\ \vdots \\ A^n \end{bmatrix} * x[0] + \begin{bmatrix} B & & & \\ AB & B & & \\ \vdots & & & \\ A^{n-1}B & A^{n-2}B & \cdots & B \end{bmatrix} * U = E * x[0] + FU$$

通过这个矩阵表达式，我们可以利用系统的初始状态以及我们给出的定长的控制序列来预测之后指定长度的系统状态。

```
[m1, n1] = size(model.A);
[n1, n_in] = size(model.B);

%% calculate M
n = m1 + n1;
h(1:m1,:) = eye(m1);
E(1:m1,:) = eye(m1) * model.A;

for kk = 2 : param.Np
    h((kk - 1) * m1 + 1 : kk * m1, :) = h((kk - 2) * m1 + 1 : (kk - 1) *
m1,:) * model.A;
    E((kk - 1) * m1 + 1 : kk * m1, :) = E((kk - 2) * m1 + 1 : (kk - 1) *
m1, :) * model.A;
end

%% Calculate C
% first column of C
v = h * model.B;
F = zeros(param.Np * m1, param.Np * n_in);
F(:,1 : n_in) = v;

% Other columns of \Phi
for i = 2 : param.Np
    F(:,(i-1) * n_in + 1 : i * n_in) = [zeros((i-1) * m1, n_in);v(1:
(param.Np - i + 1) * m1, 1 : n_in)];
end
model.E = E; % M
model.F = F; % THETA
```

2.2 代价函数

代价函数有很多种定义方式，这里我们将其定义为：

$$J = \|X - X_{ref}\|_Q + \|U\|_R = (X - X_{ref})^\top Q(X - X_{ref}) + U^\top RU$$

其中矩阵 Q,R 被我们称为权重矩阵，用以表示对应的量在我们的代价函数中所占的比重。

当然，目前的这种代价函数的形式很难让我们进行计算，所以我们用第一部中通过控制序列预测得到的状态序列的公式对代价函数进行推演，将其化为只关于控制序列 U 的函数：

$$J = \left(\mathrm{Ex}[0] + \mathrm{FU} - X_{\mathrm{ref}}\right)^\top Q\left(\mathrm{Ex}[0] + \mathrm{FU} - X_{\mathrm{ref}}\right) + U^\top \mathrm{RU}$$

对其进行进一步化简，我们可以得到一个很棒的二次型：

$$J = \frac{1}{2}U^\top \mathrm{HU} + U^\top g$$

$$H = 2\left(F^\top \mathrm{QF} + R\right)$$
$$g = 2F^\top Q\left(\mathrm{Ex}[0] - X_{\mathrm{ref}}\right)$$

```
% Q,R
model.Q = [10 0 0 0
     0 10 0 0
     0 0 1 0
     0 0 0 1];
model.R = 0.001;
model.q = [];
model.r = [];

for i = 1:param.Np
    model.q = blkdiag(model.q, model.Q);  % 从三个不同大小的矩阵创建一个分块对角矩
阵
    model.r = blkdiag(model.r, model.R);  % 从三个不同大小的矩阵创建一个分块对角矩
阵
end

model.ll = repmat(param.lower, param.Np, 1);  % 创建一个所有元素的值均为 10 的
3×2 矩阵。A = repmat(10,3,2)
model.uu = repmat(param.upper, param.Np, 1);

% 二次型应该在仿真控制器中计算，因为与系统的实时状态有关
% 后面的实现在 simulink 中完成
```

2.3 选取控制量

我们选取得到的最优控制序列中的第一项：

$$u = [1 \quad 0 \quad \cdots \quad 0]U_{opt}$$

3 Simulink 仿真

```
run("Cart_Pole_MPC_doc.slx");
```