非线性模型预测控制

1 概览

MPC实际上是一种最优控制方法。线性MPC是把目标函数表达成输入序列 的二次型表达式，把约束表示成输入序列 的线性等式和不等式约束，这样利用二次规划求解出来输入序列，然后施加到被控系统，得到反馈结果以后，再不断重复这个过程。

线性MPC的特点就是：模型预测，滚动优化，反馈校正。

它擅长处理多输入和约束。

如果需要控制非线性的被控系统，那么需要先将非线性系统进行线性化。

非线性MPC是一种针对非线性系统并且不需要线性化的模型预测控制。

非线性MPC通常把状态量，输入量合并到一起作为优化问题的决策变量，然后把动力学模型（状态方程）作为非线性约束，再结合实际问题定义目标函数，最后用非线性规划求解器进行求解。求解结果施加到被控系统，得到反馈结果以后，再不断重复这个过程。

- Nonlinear prediction model

$$\begin{cases} x_{k+1} & = & f(x_k, u_k) \\ y_k & = & g(x_k, u_k) \end{cases}$$

- Nonlinear constraints $h(x_k, u_k) \leq 0$
- Nonlinear performance index $\min \ \ell_N(x_N) + \sum_{k=0}^{N-1} \ell(x_k, u_k)$
- Optimization problem: **nonlinear programming problem (NLP)**

$$\begin{array}{ll} \min_z & F(z, x(t)) \\ \text{s.t.} & G(z, x(t)) \leq 0 \\ & H(z, x(t)) = 0 \end{array} \qquad z = \begin{bmatrix} u_0 \\ \vdots \\ u_{N-1} \\ x_1 \\ \vdots \\ x_N \end{bmatrix}$$

非线性主要体现在：

1）预测模型是非线性的

2）约束条件是非线性的

3）目标函数是非线性的

这样带约束的非线性问题，一般用非线性规划（NLP）来求解。常见的NLP求解器有：

fmincon(matlab), ipopt(c++)。

和线性模型预测控制一样，非线性模型预测控制也是一种最优控制方法，它每一次求解能够得出预测时域内最佳的决策序列，因此它也常被用来做轨迹优化。

2 平行泊车轨迹规划

这个是轨迹优化的例子。

它直接通过最优化的方法得到平行车位泊入的最佳路径。

2.1 车辆运动学模型

由于场景是低速泊车，因此使用运动学模型，这是一个非线性模型。

$$\begin{aligned}\dot{x} &= v\cos(\psi)\\ \dot{y} &= v\sin(\psi)\\ \dot{\psi} &= \frac{v}{b}\tan(\delta)\end{aligned}$$

2.2 设计非线性模型预测控制

状态变量 (x, y, ψ)

输入变量 $\delta \in \left(-\frac{\pi}{4}, \frac{\pi}{4}\right), v \in (-2, 2)m/s$

目标函数

$$J = \int_0^d \left(s(t) - s_{ref}\right)^T Q_p \left(s(t) - s_{ref}\right) + u(t)^T R_p u(t) dt + \left(s(d) - s_{ref}\right)^T Q_t \left(s(d) - s_{ref}\right) + u(d)^T R_t u(d)$$

其中

约束条件

与障碍车的最小距离大于安全距离 $dist_{min} \geq dist_{safe}$

2.3 使用matlab工具箱实现nmpc

这里使用mpc工具箱中的nlmpcmove函数

```
% 车辆参数
% 轴距
egoWheelbase    = 2.8;
distToCenter    = 0.5*egoWheelbase;
% 初始位姿
egoInitialPose = [7,3.1,0];
% 目标位姿
egoTargetPose = [-distToCenter,0,0];

% nmpc参数
% 采样时间
Ts = 0.1;
% 预测步长
p = 70;
% 控制步长
c = 70;
% 权重矩阵
Qp = diag([0.1 0.1 0]);
Rp = 0.01*eye(2);
Qt = diag([1 5 100]);
Rt = 0.1*eye(2);
% 安全距离
safetyDistance = 0.1;
```

```matlab
% 最大迭代次数
maxIter = 20;

% 定义nmpc对象
% 状态数目
nx = 3;
% 输出数目
ny = 3;
% 输入数目
nu = 2;
% 调用nlmpc库函数
nlobj = nlmpc(nx,ny,nu);
nlobj.Ts = Ts;
nlobj.PredictionHorizon = p;
nlobj.ControlHorizon = c;
nlobj.MV(1).Min = -2;
nlobj.MV(1).Max = 2;
nlobj.MV(2).Min = -pi/4;
nlobj.MV(2).Max = pi/4;
% 状态转换函数
nlobj.Model.StateFcn = "parkingVehicleStateFcn";
% 状态转换函数雅可比
nlobj.Jacobian.StateFcn = "parkingVehicleStateJacobianFcn";
% 目标函数
nlobj.Optimization.CustomCostFcn = "parkingCostFcn";
nlobj.Optimization.ReplaceStandardCost = true;
% 目标函数雅可比
nlobj.Jacobian.CustomCostFcn = "parkingCostJacobian";
% 不等式约束
nlobj.Optimization.CustomIneqConFcn = "parkingIneqConFcn";
% 不等式约束雅可比
nlobj.Jacobian.CustomIneqConFcn = "parkingIneqConFcnJacobian";
% 求解器选项
nlobj.Optimization.SolverOptions.FunctionTolerance = 0.01;
nlobj.Optimization.SolverOptions.StepTolerance = 0.01;
nlobj.Optimization.SolverOptions.ConstraintTolerance = 0.01;
nlobj.Optimization.SolverOptions.OptimalityTolerance = 0.01;
nlobj.Optimization.SolverOptions.MaxIter = maxIter;
nlobj.Optimization.SolverOptions.Display = 'iter';
opt = nlmpcmoveopt;
% 猜测初始解
opt.X0 = [linspace(egoInitialPose(1),egoTargetPose(1),p)', ...
          linspace(egoInitialPose(2),egoInitialPose(2),p)'...
          zeros(p,1)];
opt.MV0 = zeros(p,nu);
paras = {egoTargetPose,Qp,Rp,Qt,Rt,distToCenter,safetyDistance}';
nlobj.Model.NumberOfParameters = numel(paras);
opt.Parameters = paras;

% 仿真过程
x0 = egoInitialPose';
u0 = [0;0];
[mv,nloptions,info] = nlmpcmove(nlobj,x0,u0,[],[],opt);
% 可视化
timeLength = size(info.Xopt,1);
for ct = 1:timeLength
    % 可视化
    helperSLVisualizeParking(info.Xopt(ct,:), info.MVopt(ct,2));
    pause(0.05);
end
```

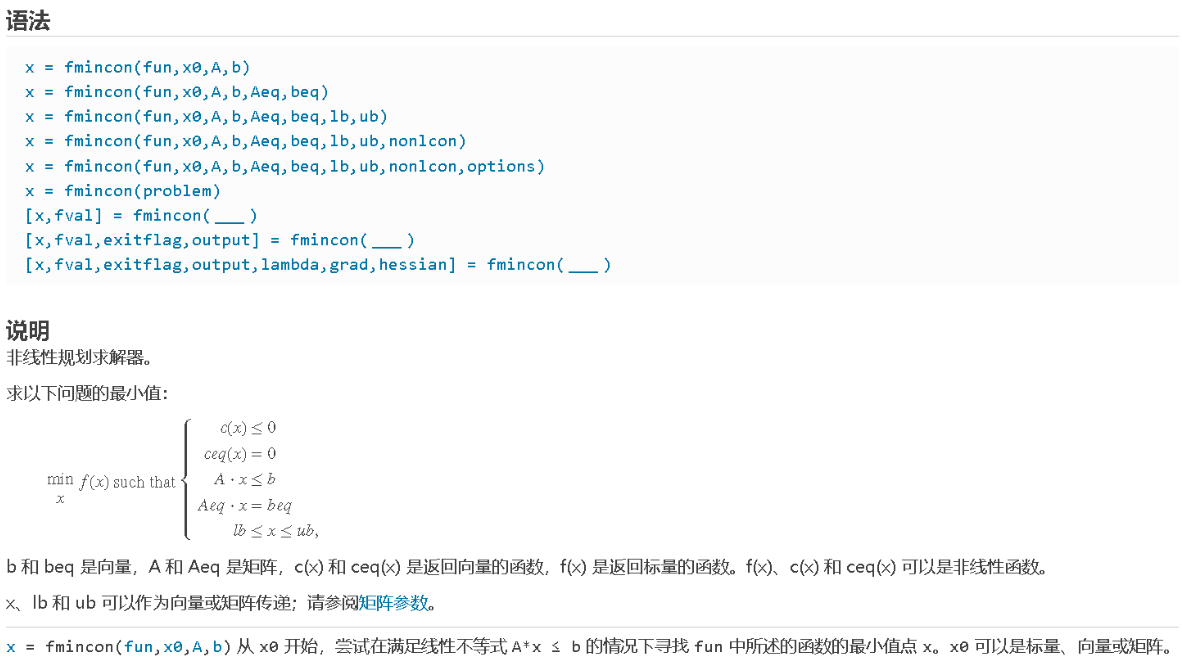
这里面，优化只求解一次，用于路径规划。

mpc工具箱里面的 nlmpcmove把算法框架全部封装好了，并且留出来目标函数和约束函数的接口，这样不需要自己从头构建优化问题并求解。

2.4 自己动手实现nmpc

自己写nmpc的代码，要构建目标函数，约束函数。同时，还有雅可比函数，虽然它不是必需的，但是如果显式提供雅可比函数的话是可以加速求解过程的。

要实现nmpc，关键还是求解过程，matlab里面的非线性规划（NLP）求解器是fmincon，看一下这个函数的说明



语法

```
x = fmincon(fun,x0,A,b)
x = fmincon(fun,x0,A,b,Aeq,beq)
x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub)
x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon)
x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon,options)
x = fmincon(problem)
[x,fval] = fmincon(___)
[x,fval,exitflag,output] = fmincon(___)
[x,fval,exitflag,output,lambda,grad,hessian] = fmincon(___)
```

说明

非线性规划求解器。

求以下问题的最小值：

$$\min_{x} f(x) \text{ such that } \begin{cases} c(x) \leq 0 \\ ceq(x) = 0 \\ A \cdot x \leq b \\ Aeq \cdot x = beq \\ lb \leq x \leq ub, \end{cases}$$

b 和 beq 是向量，A 和 Aeq 是矩阵，c(x) 和 ceq(x) 是返回向量的函数，f(x) 是返回标量的函数。f(x)、c(x) 和 ceq(x) 可以是非线性函数。

x、lb 和 ub 可以作为向量或矩阵传递；请参阅矩阵参数。

`x = fmincon(fun,x0,A,b)` 从 x0 开始，尝试在满足线性不等式 `A*x ≤ b` 的情况下寻找 fun 中所述的函数的最小值点 x。x0 可以是标量、向量或矩阵。

目标函数通过第一个参数fun给出

初始值通过第二个参数x0给出

线性不等式约束通过第三四个参数A,b给出

线性等式约束通过第五六个参数Aeq, beq给出

决策变量的上下限通过第七八个参数lb, ub给出

非线性等式和不等式约束通过第九个参数nonlcon给出

目标函数就是前面的表达式， 实现在函数parkingCostFcn中，parkingCostFcn本来是给nlmpcmove用的，要直接给fmincon用，还需要包装了一层，在CostFcnWrapper函数中

2.4.1 目标函数

$$J=\int_0^d \left(s(t)-s_{ref}\right)^T Q_p \left(s(t)-s_{ref}\right)+u(t)^T R_p u(t) dt+\left(s(d)-s_{ref}\right)^T Q_t \left(s(d)-s_{ref}\right)+u(d)^T R_t u(d)$$

matlab封装好的目标函数是parkingCostFcn

```
function cost =
 parkingCostFcn(X,U,e,data,ref,Qp,Rp,Qt,Rt,distToCenter,safetyDistance)
% Cost function for parking.
```

```
    p = data.PredictionHorizon;
    % process cost
    cost = 0;
    for idx = 1:p
        runningCost = (X(idx+1,:)-ref)*Qp*(X(idx+1,:)-ref)' +
U(idx,:)*Rp*U(idx,:)';
        cost = cost + runningCost;
    end
    % terminal cost
    terminal_cost = (X(p+1,:)-ref)*Qt*(X(p+1,:)-ref)' + U(p,:)*Rt*U(p,:)';
    % total cost
    cost = cost + terminal_cost;
end
```

但是它本来是给nlmpcmove用的，要直接给fmincon用，还需要包装了一层。

主要是把从决策变量z中把状态变量和输入变量提取出来：

```
function f = CostFcnWrapper(z, data,ref,Qp,Rp,Qt,Rt,distToCenter,safetyDistance)
% 预测补偿
p = 70;
% 状态变量个数
nx = 3;
% 输入变量个数
nu = 2;
nmv = 2;
p1 = p+1;
% 从z中提取出来输入变量和状态变量
uz = z(p*nx+1:end-1);
X = zeros(p1,nx);
U = zeros(p1,nu);
Xz = reshape(z(1:p*nx), nx, p)';
Uz = reshape(uz,nmv,p)';
X(1,:) = data.state;
X(2:p1,:) = Xz;
U(1:p1-1,:) = Uz;
e = z(end);
f =  parkingCostFcn(X,U,e,data,ref,Qp,Rp,Qt,Rt,distToCenter,safetyDistance);
end
```

从这个函数的实现，能看得出来决策变量包含了状态变量和输入变量。

初始值还是通过猜测（直线连接起点与终点）获得。

线性不等式约束主要是对输入变量（操纵变量）的范围约束线性等式约束没有

决策变量的上下限约束主要是对状态变量的范围约束，状态变量的范围约束不放在这里的话就要放到非线性约束里面去了，对于NLP求解，非线性约束越少越好解。

最终传递给fmincon求解器的目标函数是一个匿名函数：

```
CostFcn =
@(z)CostJacFcnWrapper(z,data,ref,Qp,Rp,Qt,Rt,distToCenter,safetyDistance);
```

2.4.2 约束函数

约束函数包括非线性等式约束和不等式约束，分别实现在parkingVehicleStateFcn和parkingIneqConFcn中，最后通过包装层ConFcnWrapper合并提供给fmincon

等式约束实际上就是状态方程约束

$\frac{dx}{dt} = f(x)$

2边积分：

$$\int_a^b f(x) = x(b) - x(a)$$
$$\frac{f(a)+f(b)}{2}T = x(b) - x(a)$$

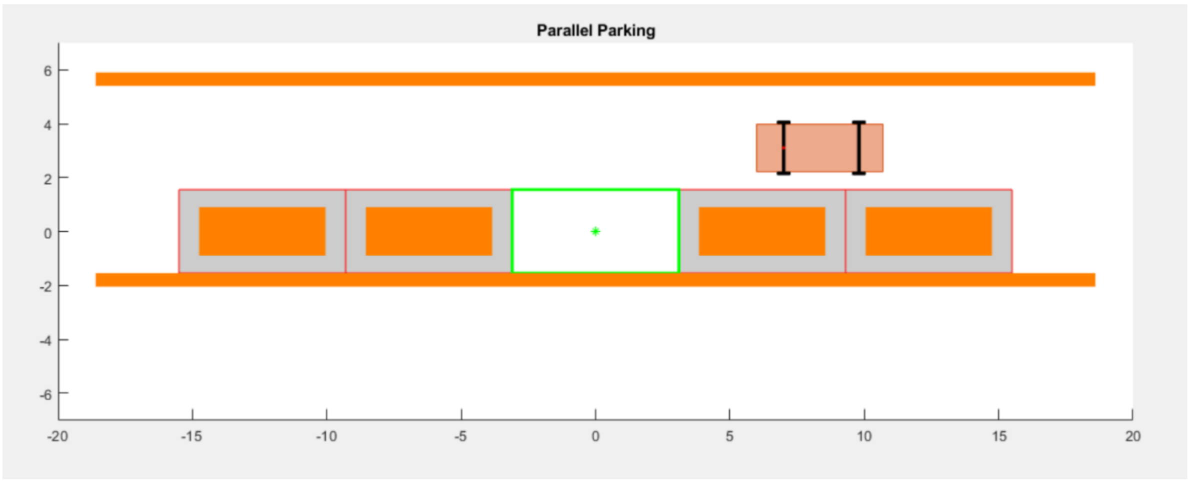
因此，约束函数是：

$$x_k + \left(f\left(x_k\right) + f\left(x_{k+1}\right)\right)\frac{T}{2} - x_{k+1} = 0$$

函数$f(x)$实现在parkingVehicleStateFcn：

```
function dxdt = parkingVehicleStateFcn(x, u,
ref,Qp,Rp,Qt,Rt,distToCenter,safetyDistance)
% State equations for parking:
% state variables x, y and yaw angle theta.
% control variables v and steering angle delta.

%%
% Parameters
wb = 2.8;

% Variables
theta = x(3);
v = u(1);
delta = u(2);

% State equations
dxdt = zeros(3,1);
dxdt(1) = v*cos(theta);
dxdt(2) = v*sin(theta);
dxdt(3) = v/wb*tan(delta);
```

不等式约束函数的实现需要把最短距离求出来，然后对它施加一个约束。



程序中构建了6个collisionBox来表示环境信息，然后ego车辆也用1个collisionBox来表示，之后用checkCollision函数来计算出最短距离

```
function cineq =
parkingIneqConFcn(X,U,e,data,ref,Qp,Rp,Qt,Rt,distToCenter,safetyDistance)
%#codegen
% Inequality constraints for parking.

% Copyright 2019 The MathWorks, Inc.

%% Ego and obstacles
persistent obstacles ego

if isempty(obstacles)
    %% Ego car
    vdims = vehicleDimensions;
    egoLength = vdims.Length;
    egoWidth = vdims.Width;
    ego = collisionBox(egoLength,egoWidth,0);

    %% Obstacles (occupied parking lots)
    obsLength = 6.2; % parking lot length

    obs1 = collisionBox(egoLength,egoWidth,0);
    T1 = trvec2tform([-2*obsLength,0, 0]);
    obs1.Pose = T1;

    obs2 = collisionBox(egoLength,egoWidth,0);
    T2 = trvec2tform([-obsLength,0, 0]);
    obs2.Pose = T2;

    obs3 = collisionBox(egoLength,egoWidth,0);
    T3 = trvec2tform([obsLength,0, 0]);
    obs3.Pose = T3;

    obs4 = collisionBox(egoLength,egoWidth,0);
    T4 = trvec2tform([2*obsLength,0, 0]);
    obs4.Pose = T4;

    obs5 = collisionBox(6*obsLength,0.5,0);
    T5 = trvec2tform([0,-1.8, 0]);
    obs5.Pose = T5;
    obs6 = collisionBox(6*obsLength,0.5,0);
    T6 = trvec2tform([0,5.65, 0]);
    obs6.Pose = T6;

    obstacles = {obs1,obs2,obs3,obs4,obs5,obs6};
end

%% constraints
p = data.PredictionHorizon;

numObstacles = numel(obstacles);
allDistances = zeros(p*numObstacles,1);

for i =1:p
    % Update ego positions
    x = X(i,1) + distToCenter*cos(X(i,3));
    y = X(i,2) + distToCenter*sin(X(i,3));
    T = trvec2tform([x,y,0]);
    H = axang2tform([0 0 1 X(i,3)]);
    ego.Pose = T*H;
    % Calculate distances from ego to obstacles
    distances = zeros(numObstacles,1);
    for ct = 1:numObstacles
        [~, dist, ~] = checkCollision(ego,obstacles{ct});
        distances(ct) = max(dist,-10); % dist is nan when collided.
        allDistances((1+(i-1)*numObstacles):numObstacles*i,1) = distances;
    end
end
cineq = -allDistances + safetyDistance;
end
```

同样，这里也需要进行包装。这里把等式约束和不等式约束包装到一起（nlmpcmove内部也是这样包装的），传递给fmincon

```
function [cineq, ceq] = ConFcnWrapper(z,
data,ref,Qp,Rp,Qt,Rt,distToCenter,safetyDistance)
p = 70;
nx = 3;
pnx = p*nx;
nu = 2;
nmv = 2;
Ts = 0.1;
p1 = p+1;
uz = z(p*nx+1:end-1);
X = zeros(p1,nx);
U = zeros(p1,nu);
Xz = reshape(z(1:p*nx), nx, p)';
Uz = reshape(uz,nmv,p)';
X(1,:) = data.state;
X(2:p1,:) = Xz;
U(1:p1-1,:) = Uz;
e = z(end);
cineq =
parkingIneqConFcn(X,U,e,data,ref,Qp,Rp,Qt,Rt,distToCenter,safetyDistance);
ceq = zeros(pnx,1);
ic = 1:nx;
Ui = U';
```

```
Xi = X';
h = Ts/2;
for i = 1:p
    uk  = Ui(:,i);
    xk  = Xi(:,i);
    xk1 = Xi(:,i+1);
    fk  = parkingVehicleStateFcn(xk,  uk,
ref,Qp,Rp,Qt,Rt,distToCenter,safetyDistance);
    fk1 = parkingVehicleStateFcn(xk1, uk,
ref,Qp,Rp,Qt,Rt,distToCenter,safetyDistance);
    ceq(ic) = xk + h*(fk + fk1) - xk1;
    ic = ic + nx;
end
end
```

2.4.3 主函数

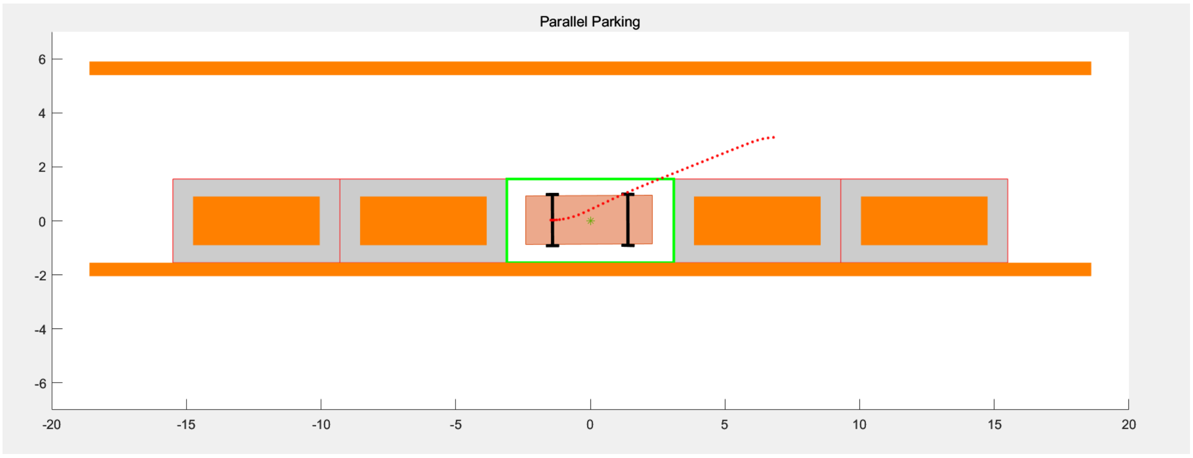
目标函数，约束函数和fmincon的接口写好了以后，剩下的主函数比较简单：

```
% 车辆参数
% 轴距
egoWheelbase    = 2.8;
distToCenter    = 0.5*egoWheelbase;
% 初始位姿
egoInitialPose = [7,3.1,0];
% 目标位姿
egoTargetPose = [-distToCenter,0,0];
% 参考状态
ref = egoTargetPose;

% nmpc参数
% 采样时间
Ts = 0.1;
% 预测步长
p = 70;
% 控制步长
c = 70;
% 权重矩阵
Qp = diag([0.1 0.1 0]);
Rp = 0.01*eye(2);
Qt = diag([1 5 100]);
Rt = 0.1*eye(2);
% 安全距离
safetyDistance = 0.1;

% NMPC信息
% 状态数目
nx = 3;
% 输出数目
ny = 3;
% 输入数目
nu = 2;
% 约束边界
MV1Min = -2;
MV1Max = 2;
MV2Min = -pi/4;
MV2Max = pi/4;
```

```
% 猜测初始解
X0 = [linspace(egoInitialPose(1),egoTargetPose(1),p)', ...
          linspace(egoInitialPose(2),egoInitialPose(2),p)'...
          zeros(p,1)];
MV0 = zeros(p,nu);
xz = reshape(X0',p*nx,1);
uz = reshape(MV0,p*nu,1);
z0 = [xz; uz; 0];

% 计算线性不等式约束(A,B)
% 这里是把输入变量（也叫操纵变量）的范围约束放在这里表示
% 在决策变量的上下限中就不需要再对输入变量进行约束了
A1  = [zeros(p*nu,p*nx), eye(p*nu), zeros(p*nu,1)];
B1  = zeros(p*nu,1);
for i = 1:p
    B1((i-1)*2+1) = MV1Max;
    B1((i-1)*2+2) = MV2Max;
end
A2  = [zeros(p*nu,p*nx), -eye(p*nu), zeros(p*nu,1)];
B2  = zeros(p*nu,1);
for i = 1:p
    B2((i-1)*2+1) = -MV1Min;
    B2((i-1)*2+2) = -MV2Min;
end
A = [A1;A2];
B = [B1;B2];

% 计算zLB, zUB
StateMin = -inf*ones(p,nx);
StateMax = inf*ones(p,nx);
xLB = reshape(StateMin', p*nx, 1);
xUB = reshape(StateMax', p*nx, 1);
nzu = 2*p;
uLB = -Inf(nzu,1);
uUB =  Inf(nzu,1);
zLB = [xLB; uLB; 0];
zUB = [xUB; uUB; Inf];
zUB(end) = 0;

% 设置data
data.PredictionHorizon = p;
data.state = egoInitialPose';
% 设置CostFcn和ConFcn,
CostFcn = @(z)CostFcnWrapper(z,data,ref,Qp,Rp,Qt,Rt,distToCenter,safetyDistance);
ConFcn  = @(z)ConFcnWrapper(z,data,ref,Qp,Rp,Qt,Rt,distToCenter,safetyDistance);
% 求解NLP问题
options = optimoptions(@fmincon,'Algorithm','sqp','MaxIterations',20,...
    'StepTolerance',0.01,...
    'ConstraintTolerance',0.01,...
    'OptimalityTolerance',0.01,...
    'FunctionTolerance',0.01, ...
    'Display', 'iter');
[cineq, ceq] = ConFcn(z0);
[z, cost, ExitFlag, Out] = fmincon(CostFcn, z0, A, B, [], [], zLB, zUB,
ConFcn,options);

% 从求得的解中还原出状态和输入
```

```
Xopt = reshape(z(1:210),3,70)';
MVopt = reshape(z(211:350),2,70)';
% 可视化
timeLength = size(Xopt,1);
for ct = 1:timeLength
    helperSLVisualizeParking(Xopt(ct,:), MVopt(ct,2));
    pause(0.05);
end
```

运行结果:



2.4.4 雅可比函数

前面提到，使用雅可比函数可以提高求解的速度和稳定性。

使用雅可比矩阵的代码实际上就是多提供了雅可比函数。

以目标函数为例，它的返回多了1个维度，就是雅可比函数值

带雅可比函数的目标函数

```
function [f, gf] = CostJacFcnWrapper(z,
data,ref,Qp,Rp,Qt,Rt,distToCenter,safetyDistance)
p = 70;
nx = 3;
nu = 2;
nmv = 2;
p1 = p+1;
uz = z(p*nx+1:end-1);
X = zeros(p1,nx);
U = zeros(p1,nu);
Xz = reshape(z(1:p*nx), nx, p)';
Uz = reshape(uz,nmv,p)';
X(1,:) = data.state;
X(2:p1,:) = Xz;
U(1:p1-1,:) = Uz;
e = z(end);
f =  parkingCostFcn(X,U,e,data,ref,Qp,Rp,Qt,Rt,distToCenter,safetyDistance);
[Gfxu, Gfuu, gfeu] =
parkingCostJacobian(X,U,e,data,ref,Qp,Rp,Qt,Rt,distToCenter,safetyDistance);
Gfxu = Gfxu';
Gfuu = Gfuu';
gfu = [Gfxu(:);Gfuu(:);gfeu];
gf = gfu(:);
end
```

带雅可比函数的约束函数

```
function [cineq, ceq, Jcineq, Jceq] = ConJacFcnWrapper(z,
data,ref,Qp,Rp,Qt,Rt,distToCenter,safetyDistance)
p = 70;
nx = 3;
pnx = p*nx;
nu = 2;
nmv = 2;
Ts = 0.1;
p1 = p+1;
uz = z(p*nx+1:end-1);
X = zeros(p1,nx);
U = zeros(p1,nu);
Xz = reshape(z(1:p*nx), nx, p)';
Uz = reshape(uz,nmv,p)';
X(1,:) = data.state;
X(2:p1,:) = Xz;
U(1:p1-1,:) = Uz;
e = z(end);
cineq =
parkingIneqConFcn(X,U,e,data,ref,Qp,Rp,Qt,Rt,distToCenter,safetyDistance);
[Jx, Ju, Je] =
parkingIneqConFcnJacobian(X,U,e,data,ref,Qp,Rp,Qt,Rt,distToCenter,safetyDistance)
;
Jx = permute(Jx, [3,2,1]);
Ju = permute(Ju, [3,2,1]);
Je = Je(:);
[nc,nx,p] = size(Jx);
Jcineq = [reshape(Jx,nc,p*nx)';(reshape(Ju,nc,p*nmv))';Je'];
ceq = zeros(pnx,1);
ic = 1:nx;
Ui = U';
Xi = X';
h = Ts/2;
Jx = zeros(pnx,nx,p);
Jmv = zeros(pnx,nmv,p);
Je = zeros(pnx,1);
for i = 1:p
    uk  = Ui(:,i);
    xk  = Xi(:,i);
    xk1 = Xi(:,i+1);
    fk  = parkingVehicleStateFcn(xk,  uk,
ref,Qp,Rp,Qt,Rt,distToCenter,safetyDistance);
    fk1 = parkingVehicleStateFcn(xk1, uk,
ref,Qp,Rp,Qt,Rt,distToCenter,safetyDistance);
    ceq(ic) = xk + h*(fk + fk1) - xk1;
    [Ak,  Bk]  = parkingVehicleStateJacobianFcn(xk,  uk,
ref,Qp,Rp,Qt,Rt,distToCenter,safetyDistance);
    [Ak1, Bk1] = parkingVehicleStateJacobianFcn(xk1, uk,
ref,Qp,Rp,Qt,Rt,distToCenter,safetyDistance);
    if i > 1
        for k=1:nx
            Jx(ic,k,i-1) = h.*Ak(:,k);   % scale by h and assign
            Jx(ic(k),k,i-1) = Jx(ic(k),k,i-1) + 1;  % + Identity
        end
    end
    for k=1:nx
        Jx(ic,k,i) = h.*Ak1(:,k);   % scale by h and assign
        Jx(ic(k),k,i) = Jx(ic(k),k,i) - 1;  % - Identity
    end
    val = h.*(Bk + Bk1);
    for k=1:nmv
        Jmv(ic,k,i) = val(:,k);
    end
    ic = ic + nx;
end
[nc,nx,p] = size(Jx);
Jceq = [reshape(Jx,nc,p*nx)';(reshape(Jmv,nc,p*nmv))';Je'];
end
```

相应的主函数代码

```
% 车辆参数
% 轴距
egoWheelbase    = 2.8;
distToCenter    = 0.5*egoWheelbase;
% 初始位姿
egoInitialPose = [7,3.1,0];
% 目标位姿
egoTargetPose = [-distToCenter,0,0];
% 参考状态
ref = egoTargetPose;

% nmpc参数
% 采样时间
Ts = 0.1;
% 预测步长
p = 70;
% 控制步长
c = 70;
% 权重矩阵
Qp = diag([0.1 0.1 0]);
Rp = 0.01*eye(2);
Qt = diag([1 5 100]);
Rt = 0.1*eye(2);
% 安全距离
safetyDistance = 0.1;

% NMPC信息
% 状态数目
nx = 3;
% 输出数目
ny = 3;
% 输入数目
nu = 2;
% 约束边界
MV1Min = -2;
MV1Max = 2;
MV2Min = -pi/4;
MV2Max = pi/4;

% 猜测初始解
```

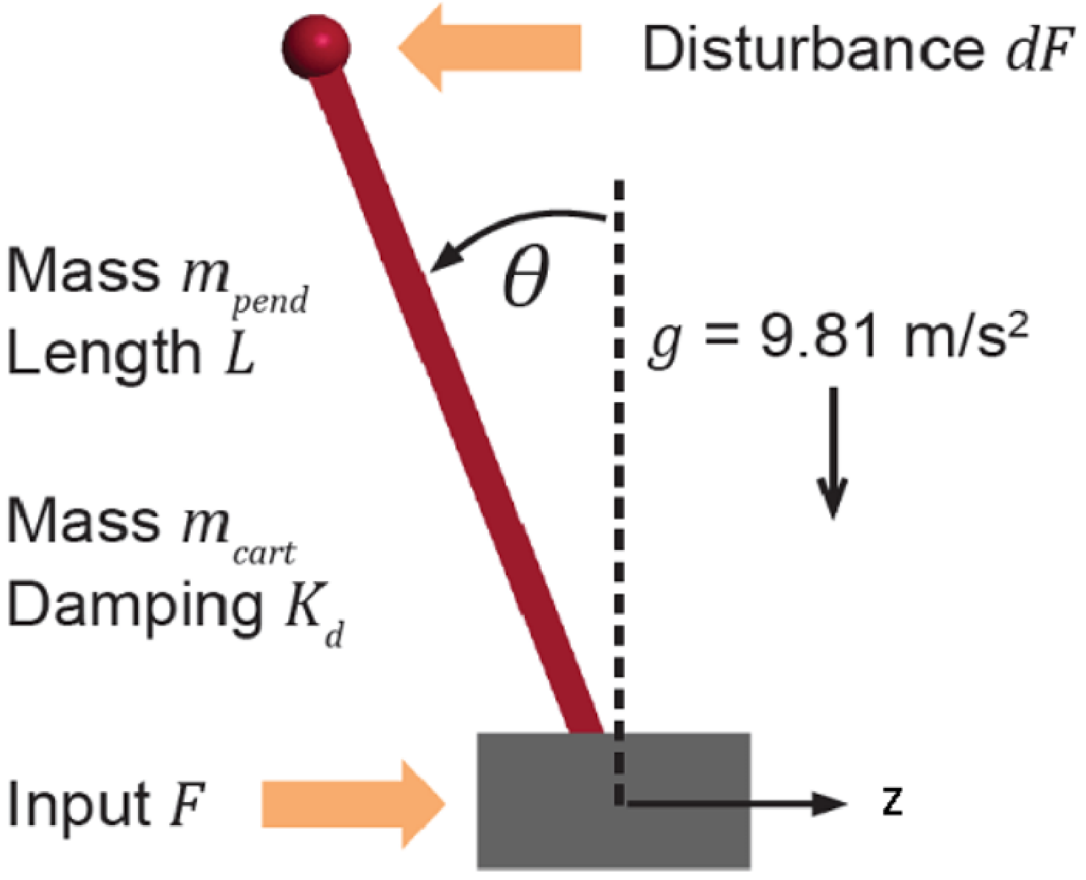
```matlab
X0 = [linspace(egoInitialPose(1),egoTargetPose(1),p)', ...
          linspace(egoInitialPose(2),egoInitialPose(2),p)'...
          zeros(p,1)];
MV0 = zeros(p,nu);
xz = reshape(X0',p*nx,1);
uz = reshape(MV0,p*nu,1);
z0 = [xz; uz; 0];

% 计算线性不等式约束(A,B)
% 这里是把输入变量（也叫操纵变量）的范围约束放在这里表示
% 在决策变量的上下限中就不需要再对输入变量进行约束了
A1  = [zeros(p*nu,p*nx), eye(p*nu), zeros(p*nu,1)];
B1  = zeros(p*nu,1);
for i = 1:p
    B1((i-1)*2+1) = MV1Max;
    B1((i-1)*2+2) = MV2Max;
end
A2  = [zeros(p*nu,p*nx), -eye(p*nu), zeros(p*nu,1)];
B2  = zeros(p*nu,1);
for i = 1:p
    B2((i-1)*2+1) = -MV1Min;
    B2((i-1)*2+2) = -MV2Min;
end
A = [A1;A2];
B = [B1;B2];

% 计算zLB, zUB
StateMin = -inf*ones(p,nx);
StateMax = inf*ones(p,nx);
xLB = reshape(StateMin', p*nx, 1);
xUB = reshape(StateMax', p*nx, 1);
nzu = 2*p;
uLB = -Inf(nzu,1);
uUB =  Inf(nzu,1);
zLB = [xLB; uLB; 0];
zUB = [xUB; uUB; Inf];
zUB(end) = 0;

% 设置data
data.PredictionHorizon = p;
data.state = egoInitialPose';
data.NumOfStates = 3;
data.MVIndex = [1,2];

% 设置CostFcn和ConFcn
CostFcn =
@(z)CostJacFcnWrapper(z,data,ref,Qp,Rp,Qt,Rt,distToCenter,safetyDistance);
ConFcn  =
@(z)ConJacFcnWrapper(z,data,ref,Qp,Rp,Qt,Rt,distToCenter,safetyDistance);

% 求解NLP问题
options = optimoptions(@fmincon,'Algorithm','sqp','MaxIterations',20,...
    'StepTolerance',0.01,...
    'ConstraintTolerance',0.01,...
    'OptimalityTolerance',0.01,...
    'FunctionTolerance',0.01, ...
    'Display', 'off',...
    'SpecifyConstraintGradient', true);
```

```
[cineq, ceq] = ConFcn(z0);
[z, cost, ExitFlag, Out] = fmincon(CostFcn, z0, A, B, [], [], zLB, zUB,
ConFcn,options);

% 从求得的解中还原出状态和输入
Xopt = reshape(z(1:210),3,70)';
MVopt = reshape(z(211:350),2,70)';
% 可视化
timeLength = size(Xopt,1);
for ct = 1:timeLength
    helperSLVisualizeParking(Xopt(ct,:), MVopt(ct,2));
    pause(0.05);
end
```

3 倒立摆Swingup控制

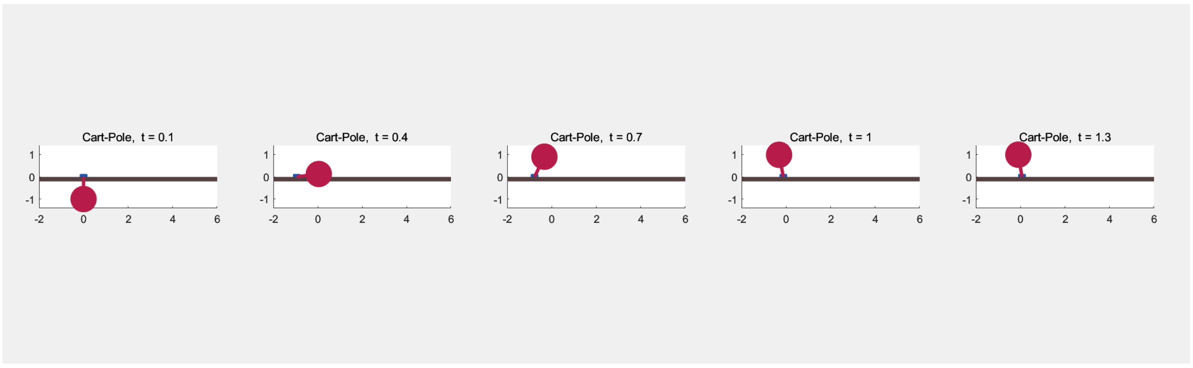
3.1 概览

倒立摆：

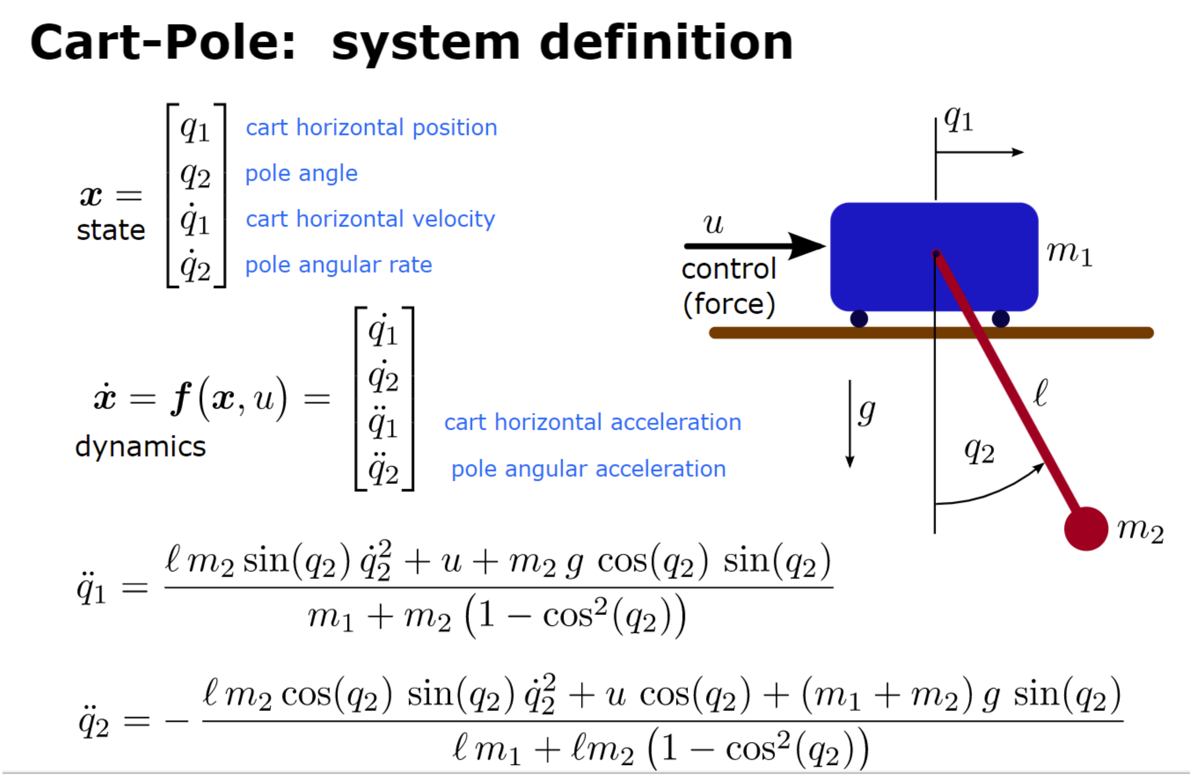


倒立摆自然状态下是不稳定的，但是可以通过控制让他稳定。

这里不是要控制杆子在上面保持稳定，而是要从杆子在下面的状态变化到杆子在上面的状态，并且保持稳定。



动力学模型:



Cart-Pole: system definition

$$\underset{\text{state}}{\boldsymbol{x}} = \begin{bmatrix} q_1 \\ q_2 \\ \dot{q}_1 \\ \dot{q}_2 \end{bmatrix} \begin{array}{l} \text{cart horizontal position} \\ \text{pole angle} \\ \text{cart horizontal velocity} \\ \text{pole angular rate} \end{array}$$

$$\underset{\text{dynamics}}{\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, u)} = \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \ddot{q}_1 \\ \ddot{q}_2 \end{bmatrix} \begin{array}{l} \\ \\ \text{cart horizontal acceleration} \\ \text{pole angular acceleration} \end{array}$$

$$\ddot{q}_1 = \frac{\ell\, m_2 \sin(q_2)\, \dot{q}_2^2 + u + m_2\, g\, \cos(q_2)\, \sin(q_2)}{m_1 + m_2\left(1 - \cos^2(q_2)\right)}$$

$$\ddot{q}_2 = -\frac{\ell\, m_2 \cos(q_2)\, \sin(q_2)\, \dot{q}_2^2 + u\, \cos(q_2) + (m_1 + m_2)\, g\, \sin(q_2)}{\ell\, m_1 + \ell m_2\left(1 - \cos^2(q_2)\right)}$$

注意:

这一页ppt是从网上找到的,这里的坐标系统与我们用的坐标系统有一点儿区别,就是角度q****

我们的角度是0表示在上面, π表示在下面,与ppt上刚好相差了角度ϕ

控制目标和约束

控制目标: $y = y_{ref}$

约束: 1 动力学约束 2 输入量范围约束

3.2 使用matlab工具箱实现nmpc

```
% NLMPC参数
% 状态数目
nx = 4;
% 输出数目
ny = 2;
% 输入变量数目
nu = 1;
% 定义nlmpc对象
nlobj = nlmpc(nx, ny, nu);
% 采样时间
```

```
Ts = 0.1;
nlobj.Ts = Ts;
% 预测步长
nlobj.PredictionHorizon = 10;
% 控制步长
nlobj.ControlHorizon = 5;
% 状态函数
nlobj.Model.StateFcn = "pendulumDT0";
nlobj.Model.IsContinuousTime = false;
nlobj.Model.NumberOfParameters = 1;
nlobj.Model.OutputFcn = 'pendulumOutputFcn';
nlobj.Jacobian.OutputFcn = @(x,u,Ts) [1 0 0 0; 0 0 1 0];
nlobj.Weights.OutputVariables = [3 3];
nlobj.Weights.ECR = eps;
nlobj.OV(1).Min = -10;
nlobj.OV(1).Max = 10;
nlobj.MV.Min = -100;
nlobj.MV.Max = 100;
nlobj.Weights.ManipulatedVariablesRate = 0.1;
nlobj.Optimization.SolverOptions.Display = 'off';
nlobj.Optimization.SolverOptions.SpecifyObjectiveGradient = false;
nlobj.Optimization.SolverOptions.SpecifyConstraintGradient = false;

% 卡尔曼滤波
EKF = extendedKalmanFilter(@pendulumStateFcn, @pendulumMeasurementFcn);

%% 闭环仿真
x = [0;0;-pi;0];
y = [x(1);x(3)];
EKF.State = x;
mv = 0;
yref1 = [0 0];
yref2 = [5 0];

nloptions = nlmpcmoveopt;
nloptions.Parameters = {Ts};

% 循环运行20s
Duration = 20;
xHistory = x;
for ct = 1:(20/Ts)
    disp(ct)
    % 设置参考输出
    % 参考输出在10s左右的时候发生了变化，需要控制跟随
    if ct*Ts<10
        yref = yref1;
    else
        yref = yref2;
    end
    % 从输出值来得到状态值，卡尔滤波估计
    xk = correct(EKF, y);
    % 计算最优控制输入
    [mv,nloptions,info] = nlmpcmove(nlobj,xk,mv,yref,[],nloptions);
    % 预测状态，卡尔曼滤波估计
    predict(EKF, [mv; Ts]);
    % 把输入施加到被控对象仿真模型
    x = pendulumDT0(x,mv,Ts);
    % 模拟传感器得到带噪声的观测数据
    y = x([1 3]) + randn(2,1)*0.01;
    % 保存数据以便可视化
    xHistory = [xHistory x]; %#ok<*AGROW>
end

%%
dyn.l = 1;
for i = 1:size(xHistory,2)
    drawCartPole(i*Ts,xHistory(:,i),dyn);
    drawnow;
    pause(0.1);
end

% Plot the closed-loop response.
figure
subplot(2,2,1)
plot(0:Ts:Duration,xHistory(1,:))
xlabel('time')
ylabel('z')
title('cart position')
subplot(2,2,2)
plot(0:Ts:Duration,xHistory(2,:))
xlabel('time')
ylabel('zdot')
title('cart velocity')
subplot(2,2,3)
plot(0:Ts:Duration,xHistory(3,:))
xlabel('time')
ylabel('theta')
title('pendulum angle')
subplot(2,2,4)
plot(0:Ts:Duration,xHistory(4,:))
xlabel('time')
ylabel('thetadot')
title('pendulum velocity')
```

这里用EKF（扩展卡尔曼滤波）来估计状态量。

如果状态量都能够用传感器测量，那么不需要卡尔曼滤波的。

如果有部分状态量不能用传感器直接测量，需要根据状态方程和输出观测值，对状态进行估计，那么则需要EKF。

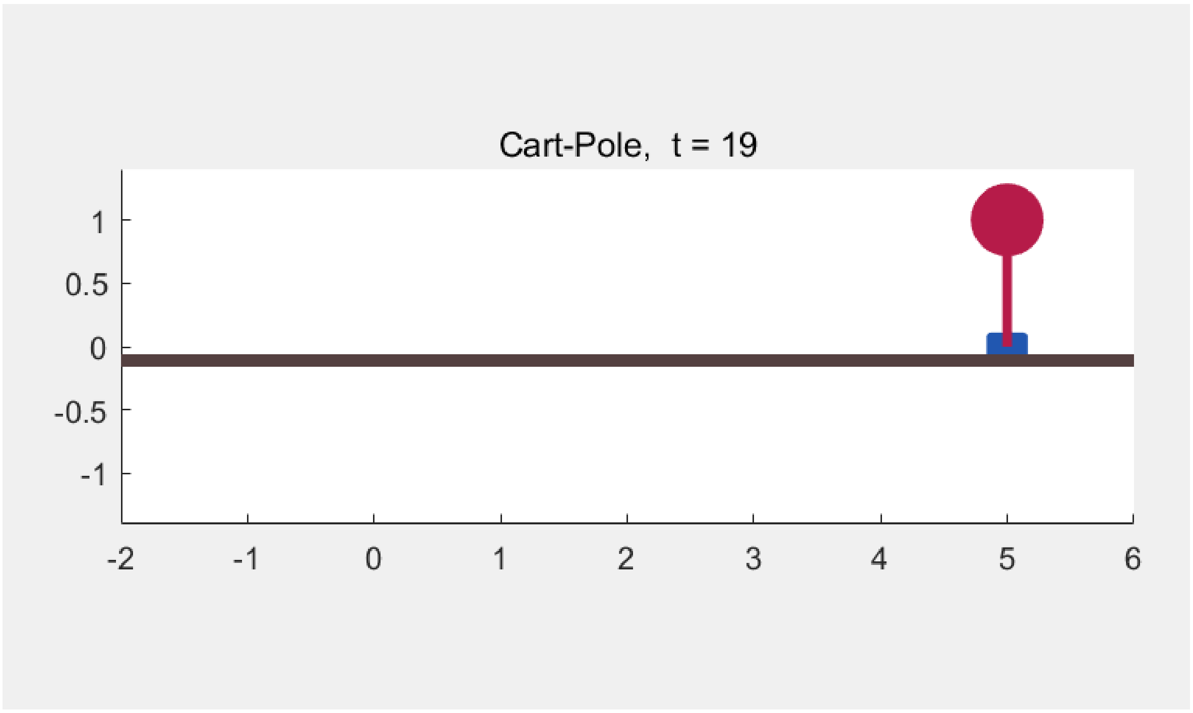
如果只研究NMPC的时候可以不考虑使用EKF

去除EKF后的代码：

```
nx = 4;
ny = 2;
nu = 1;
nlobj = nlmpc(nx, ny, nu);
Ts = 0.1;
nlobj.Ts = Ts;
nlobj.PredictionHorizon = 10;
nlobj.ControlHorizon = 5;
nlobj.Model.StateFcn = "pendulumDT0";
nlobj.Model.IsContinuousTime = false;
nlobj.Model.NumberOfParameters = 1;
nlobj.Model.OutputFcn = 'pendulumOutputFcn';
nlobj.Jacobian.OutputFcn = @(x,u,Ts) [1 0 0 0; 0 0 1 0];
nlobj.Weights.OutputVariables = [3 3];
nlobj.Weights.ManipulatedVariablesRate = 0.1;
nlobj.Weights.ECR = eps;
nlobj.OV(1).Min = -10;
nlobj.OV(1).Max = 10;
nlobj.MV.Min = -100;
nlobj.MV.Max = 100;

%% Closed-Loop Simulation in MATLAB(R)
x = [0;0;-pi;0];
y = [x(1);x(3)];
EKF.State = x;
mv = 0;
yref1 = [0 0];
yref2 = [5 0];
nloptions = nlmpcmoveopt;
nloptions.Parameters = {Ts};
% Run the simulation for |20| seconds.
Duration = 20;
xHistory = x;
for ct = 1:(20/Ts)
    % Set references
    if ct*Ts<10
        yref = yref1;
    else
        yref = yref2;
    end
    xk = x;
    % Compute optimal control moves.
    [mv,nloptions,info] = nlmpcmove(nlobj,xk,mv,yref,[],nloptions);
    % Implement first optimal control move and update plant states.
    x = pendulumDT0(x,mv,Ts);
    % Save plant states for display.
    xHistory = [xHistory x]; %#ok<*AGROW>
end

%%
figure
dyn.l = 1;
for i = 1:size(xHistory,2)
    drawCartPole(i*Ts,xHistory(:,i),dyn);
    drawnow;
    pause(0.1);
end

figure
subplot(2,2,1)
plot(0:Ts:Duration,xHistory(1,:))
xlabel('time')
ylabel('z')
title('cart position')
subplot(2,2,2)
plot(0:Ts:Duration,xHistory(2,:))
xlabel('time')
ylabel('zdot')
```

```
title('cart velocity')
subplot(2,2,3)
plot(0:Ts:Duration,xHistory(3,:))
xlabel('time')
ylabel('theta')
title('pendulum angle')
subplot(2,2,4)
plot(0:Ts:Duration,xHistory(4,:))
xlabel('time')
ylabel('thetadot')
title('pendulum velocity')
```

运行结果



3.3 自己动手实现nmpc

现在不使用mpc工具箱，我们自己来实现Swing-up最优控制（NMPC）的代码

```
% NLMPC参数
% 状态数目
nx = 4;
% 输出数目
ny = 2;
% 输入变量数目
nu = 1;
% 采样时间
Ts = 0.1;
% 预测步长
p = 10;
% 控制步长
c = 5;
% 输入变量的范围
MVMin = -100;
MVMax = 100;

%% 闭环仿真
```

```
x = [0;0;-pi;0];
y = [x(1);x(3)];
mv = 0;
yref1 = [0 0];
yref2 = [5 0];

% 循环运行20s
Duration = 20;
xHistory = x;
mvs = [];
z = [];
data.lastMV = 0;
for ct = 1:(20/Ts)
    disp(ct)
    % 设置参考输出
    % 参考输出在10s左右的时候发生了变化，需要控制跟随
    if ct*Ts<10
        yref = yref1;
    else
        yref = yref2;
    end
    xk = x;

    % 猜测初始值
    if isempty(z)
        X0 = repmat(xk,1,p);
    else
        X0 = z(nx+1:p*nx);
        X0 = [X0;X0(end-3:end)];
    end
    MV0 = zeros(c,nu);
    xz = reshape(X0,p*nx,1);
    uz = reshape(MV0,c*nu,1);
    z0 = [xz; uz; 0];

    %计算线性不等式约束(A,B)
    A1  = [zeros(p*nu,p*nx), [eye(c*nu);[zeros(5,4),ones(5,1)]], zeros(p*nu,1)];
    B1  = zeros(p*nu,1);
    for i = 1:p
        B1(i) = MVMax;
    end
    A2  = [zeros(p*nu,p*nx), -[eye(c*nu);[zeros(5,4),ones(5,1)]], zeros(p*nu,1)];
    B2  = zeros(p*nu,1);
    for i = 1:p
        B2(i) = -MVMin;
    end
    A = [A1;A2];
    B = [B1;B2];

    % 计算zLB, zUB
    StateMin = -inf*ones(p,nx);
    StateMax = inf*ones(p,nx);
    xLB = reshape(StateMin', p*nx, 1);
    xUB = reshape(StateMax', p*nx, 1);
    nzu = c;
    uLB = -Inf(nzu,1);
    uUB =  Inf(nzu,1);
    zLB = [xLB; uLB; 0];
    zUB = [xUB; uUB; Inf];

    % 设置CostFcn和ConFcn,
    data.state = x;
    CostFcn = @(z)CostFcnWrapper(z,data,yref);
    ConFcn  = @(z)ConFcnWrapper(z,data,yref);


    % 求解NLP问题
    options = optimoptions(@fmincon,'Algorithm','sqp','MaxIterations',400,...
        'StepTolerance',1e-6,...
        'ConstraintTolerance',1e-6,...
        'OptimalityTolerance',1e-6,...
        'FunctionTolerance',0.01, ...
        'Display', 'off');
    [z, cost, ExitFlag, Out] = fmincon(CostFcn, z0, A, B, [], [], zLB, zUB,
ConFcn,options);
    % 从求得的解中获取当前循环的输入
    mv = z(41);
    data.lastMV = mv;
    mvs = [mvs,mv];
    % 施加到被控对象仿真模型中
    x = pendulumDT0(x,mv,Ts);
    % 保存数据以便可视化
    xHistory = [xHistory x]; %#ok<*AGROW>
end

dyn.l = 1;
for i = 1:size(xHistory,2)
    drawCartPole(i*Ts,xHistory(:,i),dyn);
    drawnow;
    pause(0.1);
end

%%
figure
subplot(2,2,1)
plot(0:Ts:Duration,xHistory(1,:))
xlabel('time')
ylabel('z')
title('cart position')
subplot(2,2,2)
plot(0:Ts:Duration,xHistory(2,:))
xlabel('time')
ylabel('zdot')
title('cart velocity')
subplot(2,2,3)
plot(0:Ts:Duration,xHistory(3,:))
xlabel('time')
ylabel('theta')
title('pendulum angle')
subplot(2,2,4)
plot(0:Ts:Duration,xHistory(4,:))
xlabel('time')
ylabel('thetadot')
title('pendulum velocity')
```

运行结果与工具箱函数版本一致。

4 车辆运动学模型轨迹跟踪

4.1 非线性模型预测跟踪函数

车辆低速大转角运动时（比如：泊车场景）车辆运动规律是遵循运动学模型的。此时并不适合动力学模型。

运动学模型：

$$\begin{aligned}\dot{x} &= v\cos(\psi)\\ \dot{y} &= v\sin(\psi)\\ \dot{\psi} &= \frac{v}{b}\tan(\delta)\end{aligned}$$

由于运动学模型是非线性模型，因此可以考虑进行线性化，参考《MPC模型预测控制从原理到代码》。

但是线性化的时候是沿着参考路径和参考输入量进行展开的，一旦实际轨迹偏离参考轨迹或者参考输入量过大的话就会导致模型不准确，从而控制不稳定。

还有一种办法就是采用非线性模型预测控制。

设计目标函数：

$$J = \sum_{i=1}^{i=p} Q_x(x_i - x_{i,ref})^2 + Q_y(y_i - y_{i,ref})^2 + Q_\theta(\theta_i - \theta_{i,ref})^2$$

目标函数是使得预测时域内，预测点离参考点偏差最小。

```
% 目标函数
function cost = CostFcuntion(X, refxs, refys, refths, p)
% X 车辆实际状态
% refxs  预测时域内的参考点x坐标
% refys  预测时域内的参考点y坐标
% refths 预测时域内的参考点航向角
% p      预测步长数
    xs   = X(1:p);
    ys   = X(p+1:2*p);
    ths  = wrapTo2Pi(X(2*p+1:3*p));
    us   = X(3*p+1:4*p);
    Qx   = 1;
    Qy   = 1;
    Qth  = 0.5;
    Qu   = 0.01;
    cost = 0;
    for i = 1:p
        cost = cost + Qx*(xs(i) - refxs(i)).^2;
        cost = cost + Qy*(ys(i) - refys(i)).^2;
        dth =  angdiff(ths(i), refths(i));
        cost = cost + Qth*(dth).^2;
        cost = cost + Qu*us(i).^2;
    end
end
```

设计约束函数：

$$\begin{bmatrix} x \\ y \\ \theta \end{bmatrix}_{k+1} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}_{k} + ds \begin{bmatrix} cos(\theta_k) \\ sin(\theta_k) \\ \frac{tan(\delta)}{L} \end{bmatrix}$$

约束函数是使得决策变量中的状态变量和输入变量满足车辆运动学模型约束。

```
% 约束函数
function [cineq, ceq] = ConFunction(X, X0, ds, p)
% X    决策变量实际状态
% X0   决策变量初始值
% ds   行驶距离间隔
% p    预测步长数

% 车辆轴距参数
L = 2.6;
% 不等式约束
cineq = [];
% 等式约束
ceq = zeros(3*p,1);

Ui  = X(3*p+1:4*p);
Xi  = zeros(p+1, 3);
Xi(1,:) = X0';
Xi(2:end,1) = X(1:p);
Xi(2:end,2) = X(p+1:2*p);
Xi(2:end,3) = X(2*p+1:3*p);
Xi = Xi';
ic = 1:3;
for i = 1:p
    uk  = Ui(i);
    xk  = Xi(:,i);
    xk1 = xk;

    xk1(1) = xk(1) + ds*cos(xk(3));
    xk1(2) = xk(2) + ds*sin(xk(3));
    xk1(3) = xk(3) + (tan(uk)/L)*ds;
    ceq(ic) = Xi(:,i+1) - xk1(:);
    ic = ic + 3;
end
end
```

整个非线性模型预测跟踪控制的函数NMPCTrack.m

```
function steer = NMPCTrack(x, y, phi, refxs, refys, refths, ff, ds, p)
% 输入参数
% x        当前车辆的x坐标
% y        当前车辆的y坐标
% phi      当前车辆的航向角
% refxs    参考路径的x坐标
% refys    参考路径的y坐标
% refths   参考路径的航向角
% ff       前轮转角的前馈两，作为热启动参数
% ds       预测时域点集的距离间隔
% p        预测步长数

% 输出参数
```

```matlab
% steer 前轮转角

% 目标函数
CostFcn = @(X)CostFcuntion(X, refxs, refys, refths, p);

% 约束函数
ConFcn = @(X)ConFunction(X, [x;y;phi], ds, p);

% 猜测初始决策变量,热启动
x0  = refxs;
y0  = refxs;
th0 = refths;
u0  = ones(p,1)*ff;
z0 = [x0; y0; th0; u0];

% 计算决策变量的上下限
StateMin = -inf*ones(p,3);
StateMax =  inf*ones(p,3);
xLB = reshape(StateMin', p*3, 1);
xUB = reshape(StateMax', p*3, 1);
uLB = -ones(p,1)*0.8;
uUB =  ones(p,1)*0.8;
zLB = [xLB; uLB];
zUB = [xUB; uUB];

% 求解最优化
options = optimoptions(@fmincon,'Algorithm','sqp','MaxIterations',40,...
    'StepTolerance',1e-6,...
    'ConstraintTolerance',1e-6,...
    'OptimalityTolerance',1e-6,...
    'FunctionTolerance',0.01, ...
    'Display', 'off');
[z, ~, ~, ~] = fmincon(CostFcn, z0, [], [], [], [], zLB, zUB, ConFcn,options);
steer = z(3*p+1);
end

% 约束函数
function [cineq, ceq] = ConFunction(X, X0, ds, p)
% X    决策变量实际状态
% X0   决策变量初始值
% ds   行驶距离间隔
% p    预测步长数

% 车辆轴距参数
L = 2.6;
% 不等式约束
cineq = [];
% 等式约束
ceq = zeros(3*p,1);

Ui  = X(3*p+1:4*p);
Xi  = zeros(p+1, 3);
Xi(1,:) = X0';
Xi(2:end,1) = X(1:p);
Xi(2:end,2) = X(p+1:2*p);
Xi(2:end,3) = X(2*p+1:3*p);
Xi = Xi';
ic = 1:3;
```

```
for i = 1:p
    uk  = Ui(i);
    xk  = Xi(:,i);
    xk1 = xk;

    xk1(1) = xk(1) + ds*cos(xk(3));
    xk1(2) = xk(2) + ds*sin(xk(3));
    xk1(3) = xk(3) + (tan(uk)/L)*ds;
    ceq(ic) = Xi(:,i+1) - xk1(:);
    ic = ic + 3;
end
end

% 目标函数
function cost = CostFcuntion(X, refxs, refys, refths, p)
% X 决策变量实际状态
% refxs  预测时域内的参考点x坐标
% refys  预测时域内的参考点y坐标
% refths 预测时域内的参考点航向角
% p      预测步长数
    xs   = X(1:p);
    ys   = X(p+1:2*p);
    ths  = wrapTo2Pi(X(2*p+1:3*p));
    us   = X(3*p+1:4*p);
    Qx   = 1;
    Qy   = 1;
    Qth  = 0.5;
    Qu   = 0.01;
    cost = 0;
    for i = 1:p
        cost = cost + Qx*(xs(i) - refxs(i)).^2;
        cost = cost + Qy*(ys(i) - refys(i)).^2;
        dth =  angdiff(ths(i), refths(i));
        cost = cost + Qth*(dth).^2;
        cost = cost + Qu*us(i).^2;
    end
end
```

4.2 案例代码

4.2.1 路径生成

通过Roadgen界面生成路径数据。

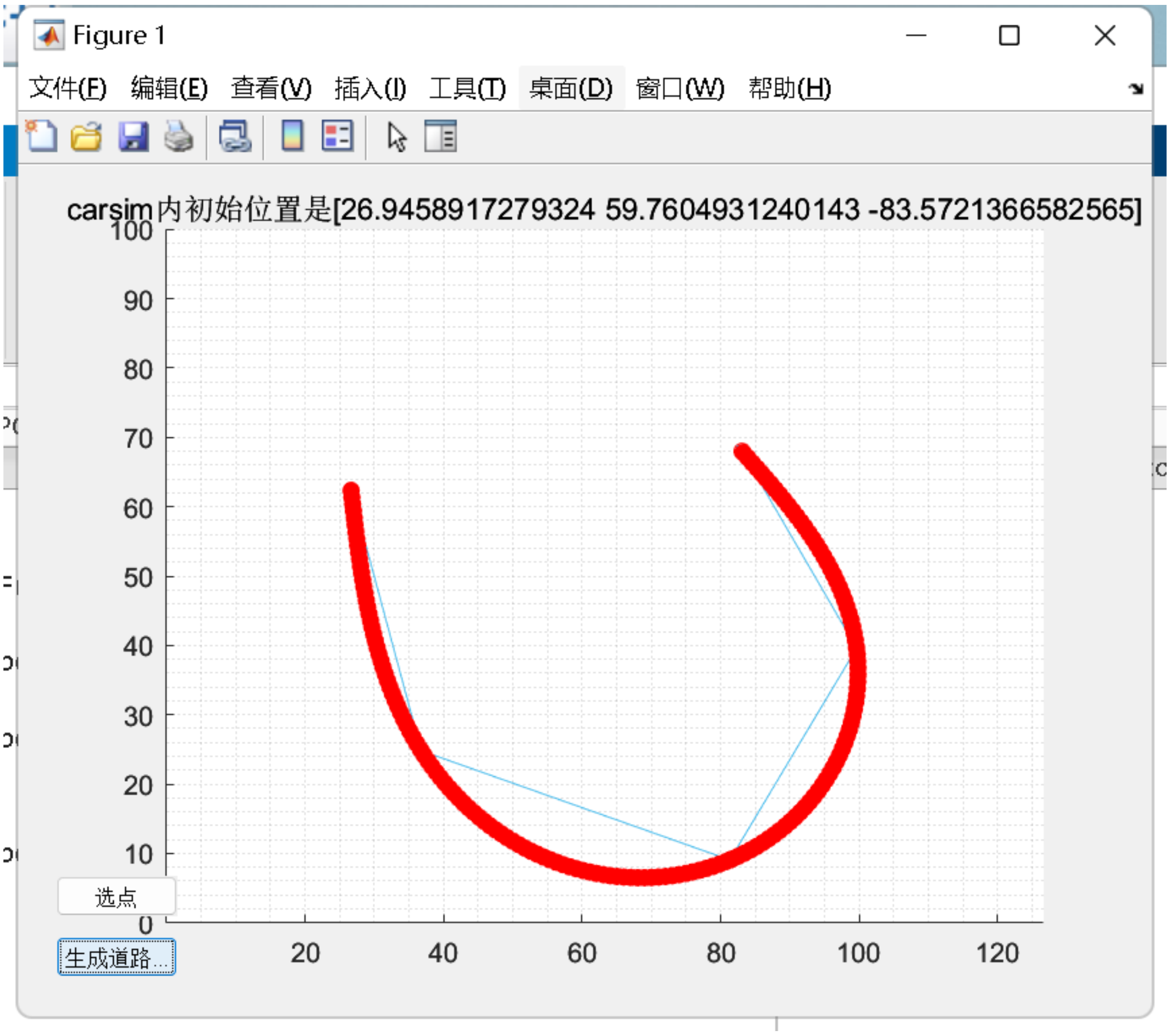


Figure 1
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
carsim内初始位置是[26.9458917279324 59.7604931240143 -83.5721366582565]
100
90
80
70
60
50
40
30
20
10
0
20
40
60
80
100
120
选点
生成道路...

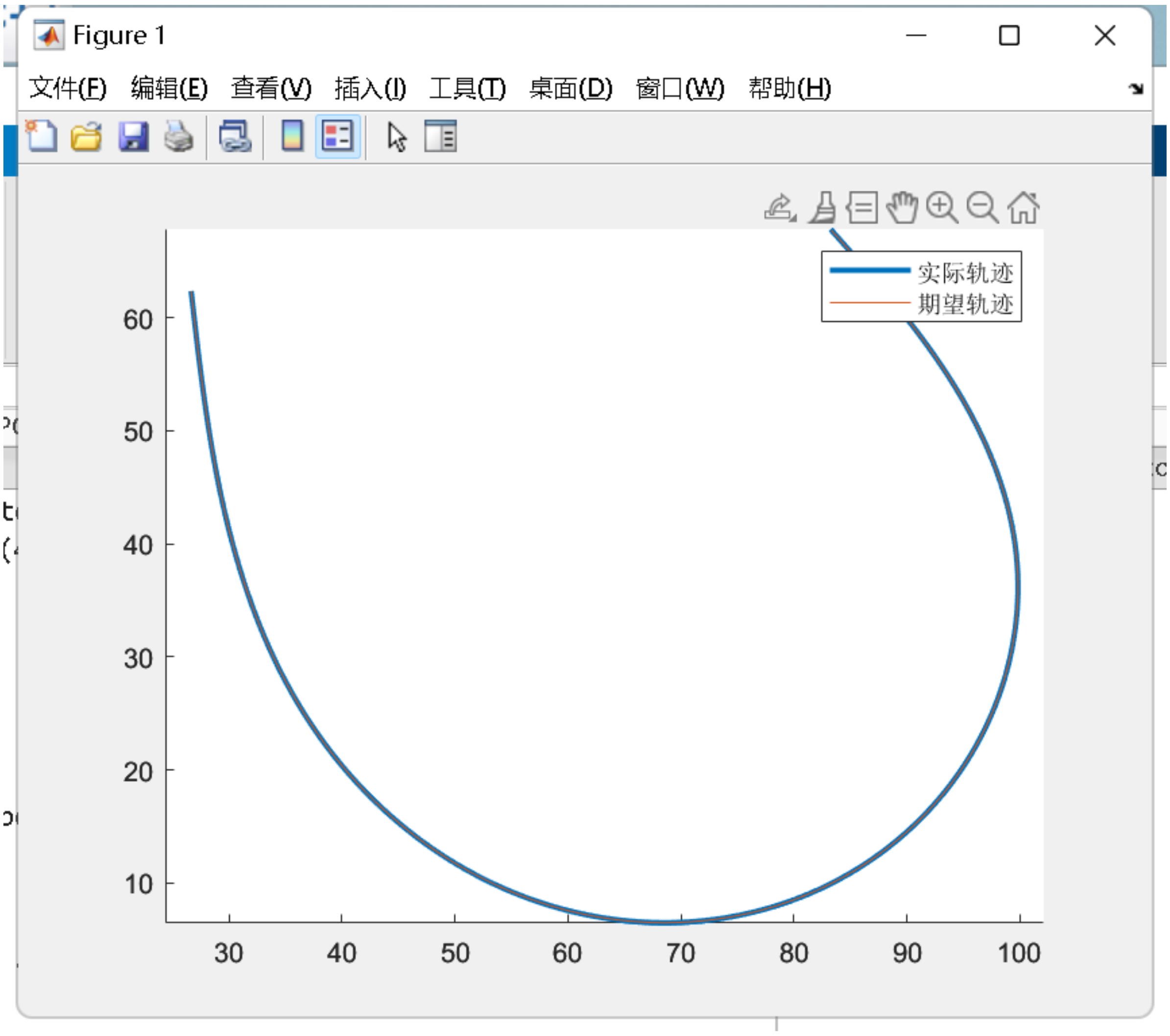
4.2.2 路径跟踪

接下来使用上面的NMPCTrack函数对路径进行跟踪：

```
% 加载路径
load pathpoint.mat;
refPath       = referencePathFrenet(waypoints);
% 第一个点
startp        = refPath.interpolate(0);
% 最后一个点
endp          = refPath.interpolate(realmax);
maxdist       = endp(6);
% 待跟踪路径
pp            = refPath.interpolate(0:0.1:maxdist);
xs            = pp(:,1)';
ys            = pp(:,2)';
ths           = pp(:,3)';
dt            = 0.1;

% 车辆初始状态
L = 2.4;
drivedist    = 0;
vehx     = startp(1);
vehy     = startp(2);
vehth    = startp(3);
vehspd   = 2;
```

```
actxs = vehx;
actys = vehy;

for i = 1:1000
    disp(['loop ', num2str(i)])
    if drivedist > maxdist
        break;
    end
    % 计算前馈转角
    curvaturep = refPath.interpolate(drivedist);
    curvature  = curvaturep(4);
    disp(curvature)
    ff = atan(curvature*L);

    % NMPC跟踪
    p = 10;
    ds = 0.1;
    refdata = refPath.interpolate(drivedist + (1:p)*ds);
    refxs  = refdata(:,1);
    refys  = refdata(:,2);
    refths = refdata(:,3);
    steer = NMPCTrack(vehx, vehy, vehth, refxs, refys, refths, ff, ds, 10);
    % 运动学模型仿真
    ds        = dt*vehspd;
    drivedist = drivedist + ds;
    vehx      = vehx  + ds*cos(vehth);
    vehy      = vehy  + ds*sin(vehth);
    vehth     = vehth + ds*tan(steer)/L;
    actxs     = [actxs, vehx];%#ok
    actys     = [actys, vehy];%#ok
end

%% 绘图
hold on;
plot(actxs, actys, 'LineWidth',2);
plot(xs, ys);
axis equal;
```

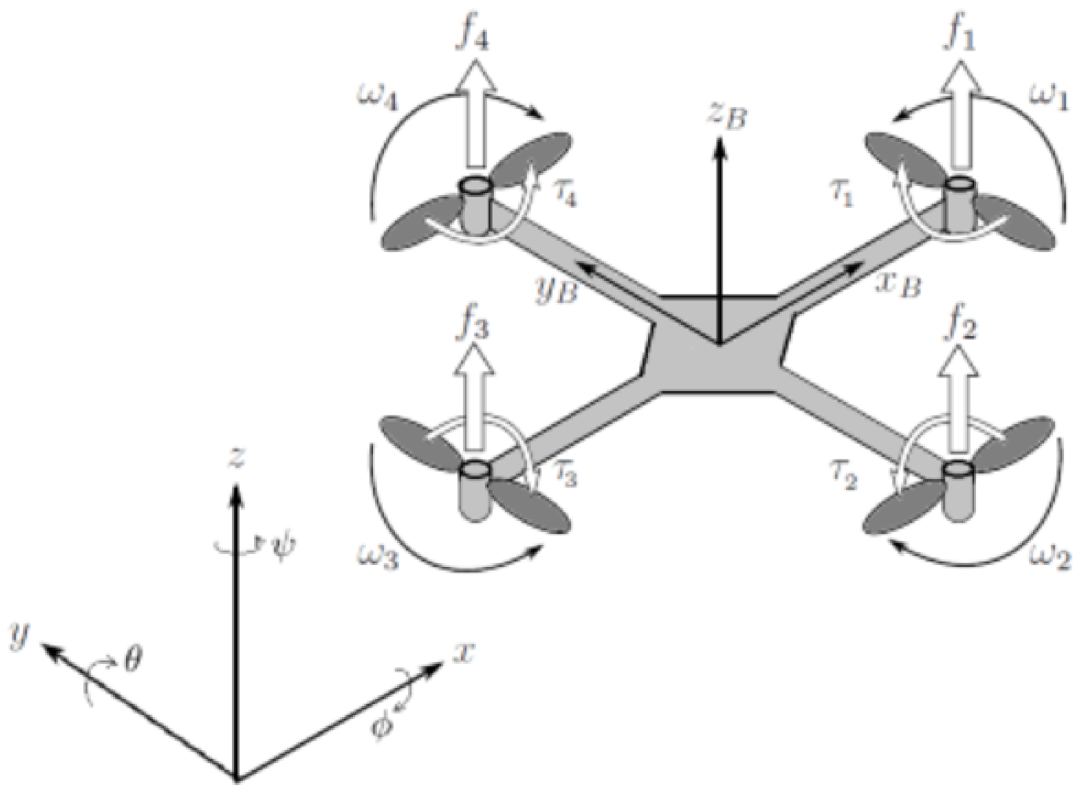
运行结果:



5 四旋翼无人机轨迹跟踪

5.1 四旋翼无人机动力学模型

参考论文《An integral predictive/nonlinear Hinf control structure for a quadrotor helicopter》



12个状态变量分别是6自由度位移和6自由度速度

$(x, y, z, \phi, \theta, \psi, \dot{x}, \dot{y}, \dot{z}, \dot{\phi}, \dot{\theta}, \dot{\psi})^T$

其中

$\xi = (x, y, z)^T$表示位置

$\eta = (\phi, \theta, \psi)^T$表示姿态

$(\dot{x}, \dot{y}, \dot{z}, \dot{\phi}, \dot{\theta}, \dot{\psi})^T$表示速度

$(u_1, u_2, u_3, u_4) = \left(\omega_1^2, \omega_2^2, \omega_3^2, \omega_4^2\right)$表示四旋翼的角速度平方

(I_{xx}, I_{yy}, I_{zz})表示xyz3个轴的转动惯量

(k, l, m, b, g)分别表示升力系数，转轴距离，质量，阻力系数，重力系数

接下来用拉格朗日方程来推导动力学方程

$$\tau = \frac{d}{dt}\frac{\partial L}{\partial \dot{\eta}} - \frac{\partial L}{\partial \eta}$$

L包含转动能量。

$E_{rot} = \frac{1}{2}\dot{\eta}^T J \dot{\eta} + \frac{1}{2}\dot{\xi}^T M \dot{\xi}$

其中：

$J = W_\eta^T I W_\eta$

$$I = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix}$$

$$W_\eta = \begin{bmatrix} 1 & 0 & -sin(\theta) \\ 0 & cos(\phi) & cos(\theta)sin(\phi) \\ 0 & -sin(\phi) & cos(\theta)cos(\phi) \end{bmatrix}$$

W是从惯性系到机体系的转换矩阵

$$\tau = \frac{d}{dt}(J\dot{\eta}) - \frac{1}{2}\frac{\dot{\eta}^T J\dot{\eta}}{d\eta}$$

$$\tau = (\frac{dJ}{dt}\dot{\eta} + J\ddot{\eta}) - \frac{1}{2}\frac{\dot{\eta}^T J\dot{\eta}}{d\eta}$$

$$\tau = J\ddot{\eta} + (\frac{dJ}{dt} - \frac{1}{2}\frac{\partial}{\partial\eta}(\dot{\eta}^T J)) * \dot{\eta}$$

记$C = (\frac{dJ}{dt} - \frac{1}{2}\frac{\partial}{\partial\eta}(\dot{\eta}^T J))$

$\tau = J\ddot{\eta} + C\dot{\eta}$

这样，能得到J和C

$$J = \begin{bmatrix} I_{xx} & 0 & -I_{xx}S\theta \\ 0 & I_{yy}C^2\phi + I_{zz}S^2\phi & (I_{yy} - I_{zz})\,C\phi S\phi C\theta \\ -I_{xx}S\theta & (I_{yy} - I_{zz})\,C\phi S\phi C\theta & I_{xx}S^2\theta + I_{yy}S^2\phi C^2\theta + I_{zz}C^2\phi C^2\theta \end{bmatrix}$$

$$\begin{aligned}
C_{11} =& 0 \\
C_{12} =& (I_{yy} - I_{zz})\left(\dot{\theta}C\phi S\phi + \dot{\psi}S^2\phi C\theta\right) + (I_{zz} - I_{yy})\,\dot{\psi}C^2\phi C\theta \\
& - I_{xx}\dot{\psi}C\theta \\
C_{13} =& (I_{zz} - I_{yy})\,\dot{\psi}C\phi S\phi C^2\theta \\
C_{21} =& (I_{zz} - I_{yy})\left(\dot{\theta}C\phi S\phi + \dot{\psi}S^2\phi C\theta\right) + (I_{yy} - I_{zz})\,\dot{\psi}C^2\phi C\theta \\
& + I_{xx}\dot{\psi}C\theta \\
C_{22} =& (I_{zz} - I_{yy})\,\dot{\phi}C\phi S\phi \\
C_{23} =& -I_{xx}\dot{\psi}S\theta C\theta + I_{yy}\dot{\psi}S^2\phi C\theta S\theta + I_{zz}\dot{\psi}C^2\phi S\theta C\theta \\
C_{31} =& (I_{yy} - I_{zz})\,\dot{\psi}C^2\theta S\phi C\phi - I_{xx}\dot{\theta}C\theta \\
C_{32} =& (I_{zz} - I_{yy})\left(\dot{\theta}C\phi S\phi S\theta + \dot{\phi}S^2\phi C\theta\right) + (I_{yy} - I_{zz})\,\dot{\phi}C^2\phi C\theta \\
& + I_{xx}\dot{\psi}S\theta C\theta - I_{yy}\dot{\psi}S^2\phi S\theta C\theta - I_{zz}\dot{\psi}C^2\phi S\theta C\theta \\
C_{33} =& (I_{yy} - I_{zz})\,\dot{\phi}C\phi S\phi C^2\theta - I_{yy}\dot{\theta}S^2\phi C\theta S\theta - I_{zz}\dot{\theta}C^2\phi C\theta S\theta \\
& + I_{xx}\dot{\theta}C\theta S\theta.
\end{aligned}$$

这里是用matlab的符号运算来推导的，代码如下：

```
%% 推导J和C矩阵
% 定义角度
% phi: roll angle
% theta: pitch angle
% psi: yaw angle
syms phi(t) theta(t) psi(t)

% 惯性坐标系到机体坐标系的转换矩阵
W = [ 1,  0,        -sin(theta);
      0,  cos(phi),  cos(theta)*sin(phi);
      0, -sin(phi),  cos(theta)*cos(phi) ];
% R_ZYX机体坐标系到惯性坐标系的转换矩阵
R = rotationMatrixEulerZYX(phi,theta,psi);
% 定义惯量符号
syms Ixx Iyy Izz
% Jacobian that relates body frame to inertial frame velocities
I = [Ixx, 0, 0; 0, Iyy, 0; 0, 0, Izz];
J = W.'*I*W;
% Coriolis matrix
dJ_dt = diff(J);
h_dot_J = [diff(phi,t), diff(theta,t), diff(psi,t)]*J;
grad_temp_h = transpose(jacobian(h_dot_J,[phi theta psi]));
C = dJ_dt - 1/2*grad_temp_h;
C = subsStateVars(C,t);
```

$$\tau = \begin{bmatrix} l*k*(-u_2+u_4) \\ l*k*(-u_1+u_3) \\ b*(-u_1+u_2-u_3+u_4) \end{bmatrix}$$
$$T = k(u_1+u_2+u_3+u_4)$$

定义状态变量

$$X = [\xi; \eta; \dot{\xi}; \dot{\eta}]$$

状态变量的导数：

$$f = [\dot{\xi}; \dot{\eta}; \ddot{\xi}; \ddot{\eta}]$$

现在目的是要用X来表示f，即求出$f(X)$

$$\ddot{\xi} = -(0,0,g)^T + R(0,0,T)^T/m$$
$$\ddot{\eta} = J^{-1}(\tau - C\dot{\eta})$$

这样最终能够求出状态转换方程，这里使用matlab符号计算工具箱直接得到状态转换方程的表达式，并且生成代码

```
%% 求状态方程表达式并生成代码
% 定义固定参数和输入变量
% k: 升力系数
% l: 转轴距离
% m: 质量
% b: 阻力常数
% g: 重力常数
% ui: 4个输入量
syms k l m b g u1 u2 u3 u4
% phi, theta, psi3个方向的姿态扭矩
tau_beta = [l*k*(-u2+u4); l*k*(-u1+u3); b*(-u1+u2-u3+u4)];
% 总推力
T = k*(u1+u2+u3+u4);
syms x(t) y(t) z(t)
% 状态变量
state = [x; y; z; phi; theta; psi; diff(x,t); diff(y,t); ...
    diff(z,t); diff(phi,t); diff(theta,t); diff(psi,t)];
state = subsStateVars(state,t);
f = [ % Set time-derivative of the positions and angles
      state(7:12);
      % Equations for linear accelerations of the center of mass
      -g*[0;0;1] + R*[0;0;T]/m;
      % Euler–Lagrange equations for angular dynamics
      inv(J)*(tau_beta - C*state(10:12))
];
f = subsStateVars(f,t);
% Replace fixed parameters with given values here
IxxVal = 1.2;
IyyVal = 1.2;
IzzVal = 2.3;
kVal = 1;
lVal = 0.25;
mVal = 2;
bVal = 0.2;
gVal = 9.81;
f = subs(f, [Ixx Iyy Izz k l m b g], ...
    [IxxVal IyyVal IzzVal kVal lVal mVal bVal gVal]);
```

```
f = simplify(f);
% 生成代码
matlabFunction(f,"File","QuadrotorStateFcn", ...
    "Vars",{state,control});
```

最终生成的状态转换函数代码：

```
function f = QuadrotorStateFcn(in1,in2)
%QuadrotorStateFcn
%    F = QuadrotorStateFcn(IN1,IN2)

%    This function was generated by the Symbolic Math Toolbox version 9.2.
%    07-Mar-2023 15:39:12

phi_t = in1(4,:);
phi_dot_t = in1(10,:);
psi_t = in1(6,:);
psi_dot_t = in1(12,:);
theta_t = in1(5,:);
theta_dot_t = in1(11,:);
u1 = in2(1,:);
u2 = in2(2,:);
u3 = in2(3,:);
u4 = in2(4,:);
x_dot_t = in1(7,:);
y_dot_t = in1(8,:);
z_dot_t = in1(9,:);
t2 = cos(phi_t);
t3 = cos(psi_t);
t4 = cos(theta_t);
t5 = sin(phi_t);
t6 = sin(psi_t);
t7 = sin(theta_t);
t8 = phi_t.*2.0;
t9 = psi_dot_t.^2;
t10 = theta_t.*2.0;
t15 = u1+u2+u3+u4;
t11 = t2.^2;
t12 = t4.^2;
t13 = sin(t8);
t14 = sin(t10);
t16 = 1.0./t12;
mt1 = [x_dot_t;y_dot_t;z_dot_t;phi_dot_t;theta_dot_t;psi_dot_t;(t15.*
(t5.*t6+t2.*t3.*t7))./2.0;t15.*(t3.*t5-t2.*t6.*t7).*(-1.0./2.0);
(t2.*t4.*t15)./2.0-9.81e+2./1.0e+2];
mt2 = [(t16.*(u2.*-1.15e+2+u4.*1.15e+2-t7.*u1.*9.2e+1+t7.*u2.*9.2e+1-
t7.*u3.*9.2e+1+t7.*u4.*9.2e+1+t11.*u2.*5.5e+1-
t11.*u4.*5.5e+1+phi_dot_t.*t14.*theta_dot_t.*2.3e+1+psi_dot_t.*t4.*theta_dot_t.*1
.058e+3+t7.*t11.*u1.*4.4e+1-t7.*t11.*u2.*4.4e+1+t7.*t11.*u3.*4.4e+1-
t7.*t11.*u4.*4.4e+1-t11.*t12.*u2.*5.5e+1+t11.*t12.*u4.*5.5e+1-
psi_dot_t.*t4.*t11.*theta_dot_t.*5.06e+2-t2.*t5.*t9.*t12.*5.06e+2-
psi_dot_t.*t4.^3.*t11.*theta_dot_t.*5.06e+2+t2.*t5.*t12.*theta_dot_t.^2.*5.06e+2+
phi_dot_t.*t4.*t7.*t11.*theta_dot_t.*5.06e+2-
t2.*t4.*t5.*t7.*u1.*5.5e+1+t2.*t4.*t5.*t7.*u3.*5.5e+1+phi_dot_t.*psi_dot_t.*t2.*t
5.*t7.*t12.*5.06e+2))./5.52e+2];
```

```
mt3 = [((t4.*u1.*6.0e+1-t4.*u3.*6.0e+1+t13.*u1.*2.2e+1-
t13.*u2.*2.2e+1+t13.*u3.*2.2e+1-
t13.*u4.*2.2e+1+phi_dot_t.*psi_dot_t.*t12.*5.52e+2+t4.*t11.*u1.*5.5e+1-
t4.*t11.*u3.*5.5e+1-
phi_dot_t.*psi_dot_t.*t11.*t12.*5.06e+2+t7.*t9.*t11.*t12.*5.06e+2+t2.*t5.*t7.*u2.
*5.5e+1-t2.*t5.*t7.*u4.*5.5e+1+phi_dot_t.*t2.*t4.*t5.*theta_dot_t.*5.06e+2-
psi_dot_t.*t2.*t4.*t5.*t7.*theta_dot_t.*5.06e+2).*(-1.0./5.52e+2))./t4];
mt4 = [(t16.*(u1.*-9.2e+1+u2.*9.2e+1-u3.*9.2e+1+u4.*9.2e+1-
t7.*u2.*1.15e+2+t7.*u4.*1.15e+2+t11.*u1.*4.4e+1-t11.*u2.*4.4e+1+t11.*u3.*4.4e+1-
t11.*u4.*4.4e+1+phi_dot_t.*t4.*theta_dot_t.*4.6e+1+psi_dot_t.*t14.*theta_dot_t.*5
.29e+2+t7.*t11.*u2.*5.5e+1-
t7.*t11.*u4.*5.5e+1+phi_dot_t.*t4.*t11.*theta_dot_t.*5.06e+2-
t2.*t4.*t5.*u1.*5.5e+1+t2.*t4.*t5.*u3.*5.5e+1+phi_dot_t.*psi_dot_t.*t2.*t5.*t12.*
5.06e+2-psi_dot_t.*t4.*t7.*t11.*theta_dot_t.*5.06e+2-
t2.*t5.*t7.*t9.*t12.*5.06e+2))./5.52e+2];
f = [mt1;mt2;mt3;mt4];
```

由于在进行最优化求解过程还需要用到雅克比矩阵（加速，增加稳定性）

因此还需要生成雅克比矩阵函数：

```
%% 计算状态转换函数的雅克比矩阵函数，并生成代码
% Calculate Jacobians for nonlinear prediction model
A = jacobian(f,state);
control = [u1; u2; u3; u4];
B = jacobian(f,control);
matlabFunction(A,B,"File","QuadrotorStateJacobianFcn", ...
    "Vars",{state,control});
```

生成的雅克比函数代码如下：

```
function [A,B] = QuadrotorStateJacobianFcn(in1,in2)
%QuadrotorStateJacobianFcn
%    [A,B] = QuadrotorStateJacobianFcn(IN1,IN2)

%    This function was generated by the Symbolic Math Toolbox version 9.2.
%    07-Mar-2023 16:16:13

phi_t = in1(4,:);
phi_dot_t = in1(10,:);
psi_t = in1(6,:);
psi_dot_t = in1(12,:);
theta_t = in1(5,:);
theta_dot_t = in1(11,:);
u1 = in2(1,:);
u2 = in2(2,:);
u3 = in2(3,:);
u4 = in2(4,:);
t2 = cos(phi_t);
t3 = cos(psi_t);
t4 = cos(theta_t);
t5 = sin(phi_t);
t6 = sin(psi_t);
t7 = sin(theta_t);
t8 = phi_t.*2.0;
t9 = psi_dot_t.^2;
```

```
t10 = theta_t.*2.0;
t11 = theta_dot_t.^2;
t21 = u1+u2+u3+u4;
t12 = cos(t8);
t13 = t2.^2;
t14 = cos(t10);
t15 = t4.^2;
t16 = t4.^3;
t17 = sin(t8);
t18 = t5.^2;
t19 = sin(t10);
t20 = t7.^2;
t22 = t4.*6.0e+1;
t23 = 1.0./t4;
t26 = t7.*9.2e+1;
t27 = t7.*1.15e+2;
t32 = (t2.*t4)./2.0;
t33 = (t3.*t5)./2.0;
t34 = (t5.*t6)./2.0;
t41 = t2.*t4.*t5.*5.5e+1;
t42 = t2.*t5.*t7.*5.5e+1;
t47 = (t2.*t3.*t7)./2.0;
t50 = (t2.*t6.*t7)./2.0;
t24 = 1.0./t15;
t25 = 1.0./t16;
t28 = -t26;
t29 = t13.*4.4e+1;
t30 = t13.*5.5e+1;
t31 = t17.*2.2e+1;
t37 = -t33;
t43 = t7.*t13.*-4.4e+1;
t44 = t7.*t13.*-5.5e+1;
t52 = t7.*t41;
t53 = t4.*t13.*theta_dot_t.*5.06e+2;
t54 = t4.*t13.*u3.*-5.5e+1;
t59 = phi_dot_t.*psi_dot_t.*t13.*t15.*5.06e+2;
t60 = psi_dot_t.*t4.*t7.*t13.*theta_dot_t.*-5.06e+2;
t62 = t7.*t9.*t13.*t15.*5.06e+2;
t63 = t34+t47;
t35 = -t29;
t36 = -t30;
t38 = t4.*t30;
t39 = t7.*t29;
t40 = t7.*t30;
t51 = t15.*t30;
t55 = t44.*u4;
t56 = phi_dot_t.*t53;
t57 = t7.*t53;
t61 = -t59;
t64 = t37+t50;
t45 = t38.*u1;
t46 = t38.*u3;
t48 = t40.*u2;
```

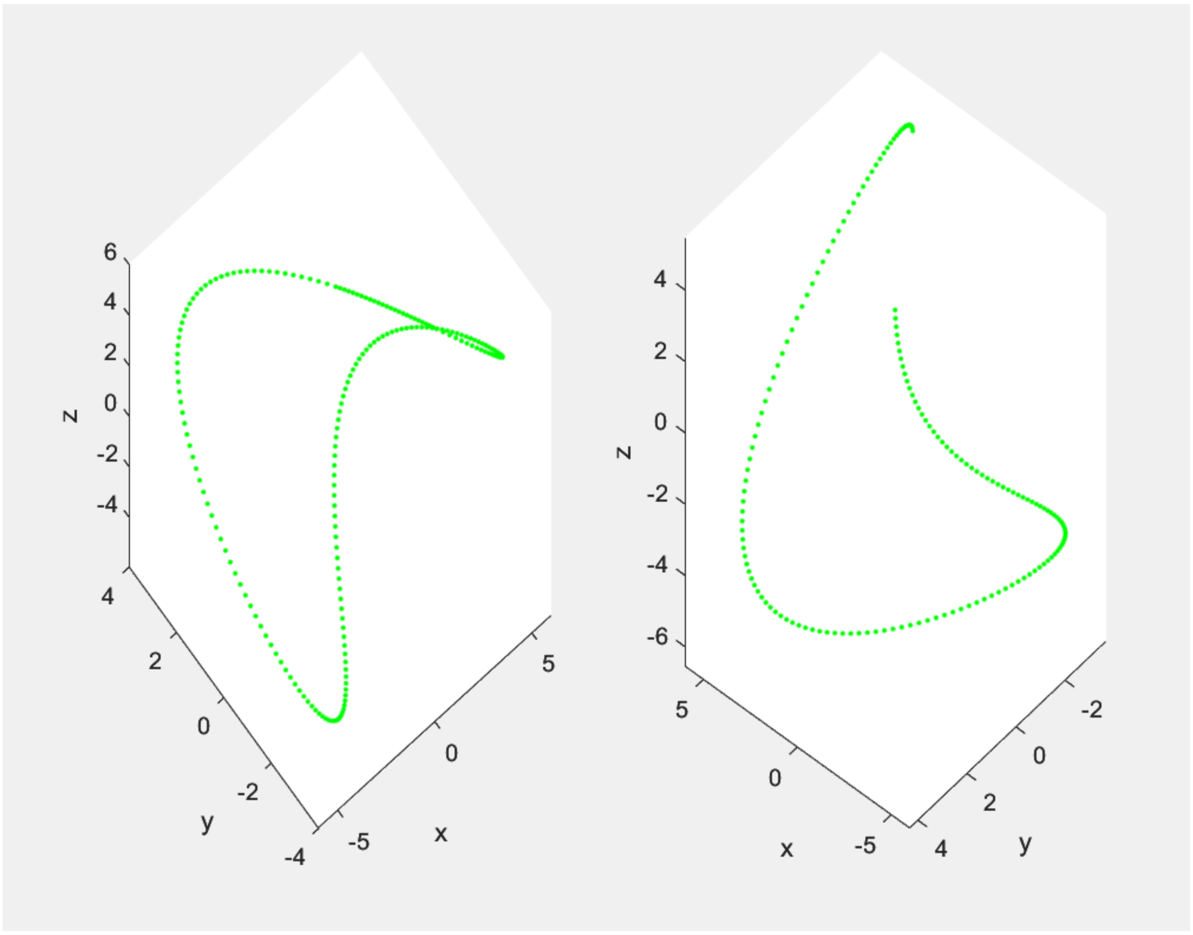
```
et1 = (t24.*(t4.*u1.*-9.2e+1+t4.*u2.*9.2e+1-
t4.*u3.*9.2e+1+t4.*u4.*9.2e+1+phi_dot_t.*t14.*theta_dot_t.*4.6e+1-
psi_dot_t.*t7.*theta_dot_t.*1.058e+3-t4.*t13.*u2.*4.4e+1-
t4.*t13.*u4.*4.4e+1+t4.*t29.*u1+t4.*t29.*u3+phi_dot_t.*t13.*t15.*theta_dot_t.*5.0
6e+2-
phi_dot_t.*t13.*t20.*theta_dot_t.*5.06e+2+psi_dot_t.*t7.*t13.*theta_dot_t.*5.06e+
2-
t2.*t5.*t15.*u1.*5.5e+1+t2.*t5.*t15.*u3.*5.5e+1+t4.*t7.*t13.*u2.*1.1e+2+t2.*t5.*t
20.*u1.*5.5e+1-t4.*t7.*t13.*u4.*1.1e+2-
t2.*t5.*t20.*u3.*5.5e+1+phi_dot_t.*psi_dot_t.*t2.*t5.*t16.*5.06e+2+psi_dot_t.*t7.
*t13.*t15.*theta_dot_t.*1.518e+3+t2.*t4.*t5.*t7.*t9.*1.012e+3-
t2.*t4.*t5.*t7.*t11.*1.012e+3-
phi_dot_t.*psi_dot_t.*t2.*t4.*t5.*t20.*1.012e+3))./5.52e+2;
et2 = (t7.*t25.*(u2.*-1.15e+2+u4.*1.15e+2+t7.*t56-t7.*u1.*9.2e+1-t7.*u3.*9.2e+1-
t13.*u4.*5.5e+1+t26.*u2+t26.*u4+t30.*u2+t39.*u1+t39.*u3+t43.*u2+t43.*u4+t51.*u4+t
52.*u3+phi_dot_t.*t19.*theta_dot_t.*2.3e+1+psi_dot_t.*t4.*theta_dot_t.*1.058e+3-
t13.*t15.*u2.*5.5e+1-psi_dot_t.*t4.*t13.*theta_dot_t.*5.06e+2-
psi_dot_t.*t13.*t16.*theta_dot_t.*5.06e+2-
t2.*t5.*t9.*t15.*5.06e+2+t2.*t5.*t11.*t15.*5.06e+2-
t2.*t4.*t5.*t7.*u1.*5.5e+1+phi_dot_t.*psi_dot_t.*t2.*t5.*t7.*t15.*5.06e+2))./2.76
e+2;
et3 = t24.*(t4.*u2.*1.15e+2-
t4.*u4.*1.15e+2+t38.*u4+t42.*u3+phi_dot_t.*t7.*theta_dot_t.*4.6e+1-
psi_dot_t.*t14.*theta_dot_t.*1.058e+3-
t4.*t13.*u2.*5.5e+1+phi_dot_t.*t7.*t13.*theta_dot_t.*5.06e+2+psi_dot_t.*t13.*t15.
*theta_dot_t.*5.06e+2-
psi_dot_t.*t13.*t20.*theta_dot_t.*5.06e+2+t2.*t5.*t9.*t16.*5.06e+2-
t2.*t5.*t7.*u1.*5.5e+1-
t2.*t4.*t5.*t9.*t20.*1.012e+3+phi_dot_t.*psi_dot_t.*t2.*t4.*t5.*t7.*1.012e+3).*
(-1.0./5.52e+2);
et4 = (t7.*t25.*(t48+t55+t56+t60-u1.*9.2e+1+u2.*9.2e+1-u3.*9.2e+1+u4.*9.2e+1-
t7.*u2.*1.15e+2-t13.*u2.*4.4e+1-
t13.*u4.*4.4e+1+t29.*u1+t27.*u4+t29.*u3+t41.*u3+phi_dot_t.*t4.*theta_dot_t.*4.6e+
1+psi_dot_t.*t19.*theta_dot_t.*5.29e+2-
t2.*t4.*t5.*u1.*5.5e+1+phi_dot_t.*psi_dot_t.*t2.*t5.*t15.*5.06e+2-
t2.*t5.*t7.*t9.*t15.*5.06e+2))./2.76e+2;
mt1 =
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0
.0,0.0,(t21.*(t2.*t6-t3.*t5.*t7))./2.0,t21.*(t2.*t3+t5.*t6.*t7).*
(-1.0./2.0),t4.*t5.*t21.*(-1.0./2.0)];
mt2 = [(t24.*(t7.*t46+t7.*t59-
t9.*t13.*t15.*5.06e+2+t11.*t13.*t15.*5.06e+2+t9.*t15.*t18.*5.06e+2-
t11.*t15.*t18.*5.06e+2-t2.*t5.*u2.*1.1e+2+t2.*t5.*u4.*1.1e+2+t4.*t44.*u1-
t2.*t5.*t7.*u1.*8.8e+1+t2.*t5.*t7.*u2.*8.8e+1-
t2.*t5.*t7.*u3.*8.8e+1+t2.*t5.*t7.*u4.*8.8e+1+t2.*t5.*t15.*u2.*1.1e+2-
t2.*t5.*t15.*u4.*1.1e+2+t4.*t7.*t18.*u1.*5.5e+1-t4.*t7.*t18.*u3.*5.5e+1-
phi_dot_t.*psi_dot_t.*t7.*t15.*t18.*5.06e+2+psi_dot_t.*t2.*t4.*t5.*theta_dot_t.*1
.012e+3+psi_dot_t.*t2.*t5.*t16.*theta_dot_t.*1.012e+3-
phi_dot_t.*t2.*t4.*t5.*t7.*theta_dot_t.*1.012e+3))./5.52e+2];
mt3 = [t23.*(t48+t55+t56+t60+t12.*u1.*4.4e+1-t12.*u2.*4.4e+1+t12.*u3.*4.4e+1-
t12.*u4.*4.4e+1-t7.*t18.*u2.*5.5e+1+t7.*t18.*u4.*5.5e+1-
phi_dot_t.*t4.*t18.*theta_dot_t.*5.06e+2-
t2.*t4.*t5.*u1.*1.1e+2+t2.*t4.*t5.*u3.*1.1e+2+phi_dot_t.*psi_dot_t.*t2.*t5.*t15.*
1.012e+3+psi_dot_t.*t4.*t7.*t18.*theta_dot_t.*5.06e+2-
t2.*t5.*t7.*t9.*t15.*1.012e+3).*(-1.0./5.52e+2)];
```

```
mt4 = [t24.*(t45+t54+t61+t62+t2.*t5.*u1.*8.8e+1-
t2.*t5.*u2.*8.8e+1+t2.*t5.*u3.*8.8e+1-t2.*t5.*u4.*8.8e+1-
t4.*t18.*u1.*5.5e+1+t4.*t18.*u3.*5.5e+1+phi_dot_t.*psi_dot_t.*t15.*t18.*5.06e+2-
t7.*t9.*t15.*t18.*5.06e+2+t2.*t5.*t7.*u2.*1.1e+2-
t2.*t5.*t7.*u4.*1.1e+2+phi_dot_t.*t2.*t4.*t5.*theta_dot_t.*1.012e+3-
psi_dot_t.*t2.*t4.*t5.*t7.*theta_dot_t.*1.012e+3).*
(-1.0./5.52e+2),0.0,0.0,0.0,0.0,0.0,0.0,t3.*t21.*t32,t6.*t21.*t32,t2.*t7.*t21.*
(-1.0./2.0),et1+et2];
mt5 = [(t23.*(t7.*u1.*6.0e+1-t7.*u3.*6.0e+1+t40.*u1+t41.*u4+t44.*u3-
t9.*t13.*t16.*5.06e+2+phi_dot_t.*psi_dot_t.*t4.*t7.*1.104e+3+t4.*t9.*t13.*t20.*1.
012e+3-t2.*t4.*t5.*u2.*5.5e+1-
phi_dot_t.*psi_dot_t.*t4.*t7.*t13.*1.012e+3+phi_dot_t.*t2.*t5.*t7.*theta_dot_t.*5
.06e+2+psi_dot_t.*t2.*t5.*t15.*theta_dot_t.*5.06e+2-
psi_dot_t.*t2.*t5.*t20.*theta_dot_t.*5.06e+2))./5.52e+2-(t7.*t24.*
(t45+t54+t61+t62-t4.*u3.*6.0e+1-t17.*u2.*2.2e+1-
t17.*u4.*2.2e+1+t22.*u1+t31.*u1+t31.*u3+t42.*u2+phi_dot_t.*psi_dot_t.*t15.*5.52e+
2-t2.*t5.*t7.*u4.*5.5e+1+phi_dot_t.*t2.*t4.*t5.*theta_dot_t.*5.06e+2-
psi_dot_t.*t2.*t4.*t5.*t7.*theta_dot_t.*5.06e+2))./5.52e+2,et3+et4,0.0,0.0,0.0,0.
0,0.0,0.0];
mt6 = [(t21.*(t3.*t5-t2.*t6.*t7))./2.0,(t21.*
(t5.*t6+t2.*t3.*t7))./2.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,
0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,(t24.*
(t57+t19.*theta_dot_t.*2.3e+1+psi_dot_t.*t2.*t5.*t7.*t15.*5.06e+2))./5.52e+2,t23.
*(psi_dot_t.*t15.*5.52e+2-
psi_dot_t.*t13.*t15.*5.06e+2+t2.*t4.*t5.*theta_dot_t.*5.06e+2).*(-1.0./5.52e+2),
(t24.*
(t53+t4.*theta_dot_t.*4.6e+1+psi_dot_t.*t2.*t5.*t15.*5.06e+2))./5.52e+2,0.0,0.0,0
.0,0.0,1.0,0.0,0.0,0.0,0.0];
mt7 = [(t24.*(phi_dot_t.*t19.*2.3e+1+psi_dot_t.*t4.*1.058e+3-
psi_dot_t.*t4.*t13.*5.06e+2-
psi_dot_t.*t13.*t16.*5.06e+2+phi_dot_t.*t4.*t7.*t13.*5.06e+2+t2.*t5.*t15.*theta_d
ot_t.*1.012e+3))./5.52e+2,t23.*(phi_dot_t.*t2.*t4.*t5.*5.06e+2-
psi_dot_t.*t2.*t4.*t5.*t7.*5.06e+2).*(-1.0./5.52e+2),(t24.*
(phi_dot_t.*t4.*4.6e+1+psi_dot_t.*t19.*5.29e+2+phi_dot_t.*t4.*t13.*5.06e+2-
psi_dot_t.*t4.*t7.*t13.*5.06e+2))./5.52e+2,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,t2
4.*(t53-
t4.*theta_dot_t.*1.058e+3+t13.*t16.*theta_dot_t.*5.06e+2+psi_dot_t.*t2.*t5.*t15.*
1.012e+3-phi_dot_t.*t2.*t5.*t7.*t15.*5.06e+2).*(-1.0./5.52e+2)];
mt8 = [t23.*(phi_dot_t.*t15.*5.52e+2-
phi_dot_t.*t13.*t15.*5.06e+2+psi_dot_t.*t7.*t13.*t15.*1.012e+3-
t2.*t4.*t5.*t7.*theta_dot_t.*5.06e+2).*(-1.0./5.52e+2),(t24.*
(t19.*theta_dot_t.*5.29e+2+phi_dot_t.*t2.*t5.*t15.*5.06e+2-
t4.*t7.*t13.*theta_dot_t.*5.06e+2-
psi_dot_t.*t2.*t5.*t7.*t15.*1.012e+3))./5.52e+2];
A = reshape([mt1,mt2,mt3,mt4,mt5,mt6,mt7,mt8],12,12);
if nargout > 1
    mt9 = [0.0,0.0,0.0,0.0,0.0,0.0,t63,t64,t32,t24.*(t26+t43+t52).*
(-1.0./5.52e+2),t23.*(t22+t31+t38).*(-1.0./5.52e+2),t24.*(t35+t41+9.2e+1).*
(-1.0./5.52e+2),0.0,0.0,0.0,0.0,0.0,0.0,t63,t64,t32,t24.*
(t28+t36+t39+t51+1.15e+2).*(-1.0./5.52e+2),(t23.*(t31-t42))./5.52e+2,t24.*
(t27+t29+t44-9.2e+1).*(-1.0./5.52e+2),0.0,0.0,0.0,0.0,0.0,0.0,t63,t64,t32,(t24.*
(t28+t39+t52))./5.52e+2,(t23.*(t22-t31+t38))./5.52e+2,(t24.*(t29+t41-
9.2e+1))./5.52e+2,0.0,0.0,0.0,0.0,0.0,0.0,t63,t64,t32,(t24.*
(t26+t36+t43+t51+1.15e+2))./5.52e+2,(t23.*(t31+t42))./5.52e+2];
    mt10 = [(t24.*(t27+t35+t44+9.2e+1))./5.52e+2];
    B = reshape([mt9,mt10],12,4);
end
```

5.2 四旋翼无人机轨迹跟踪

有了无人机的动力学模型以后，考虑对一段轨迹进行跟踪。这个动力学模型是非线性的，因此用非线性模型预测控制。

期望的轨迹：



5.1 mpc工具箱实现nmpc

关键函数是nlmpcmove

1 定义mpc对象，然后把状态函数和状态转换雅克比函数传入即可。

2 循环内获取当前实际状态，预测时域内的参考状态，用nlmpcmove求出最优输入变量。

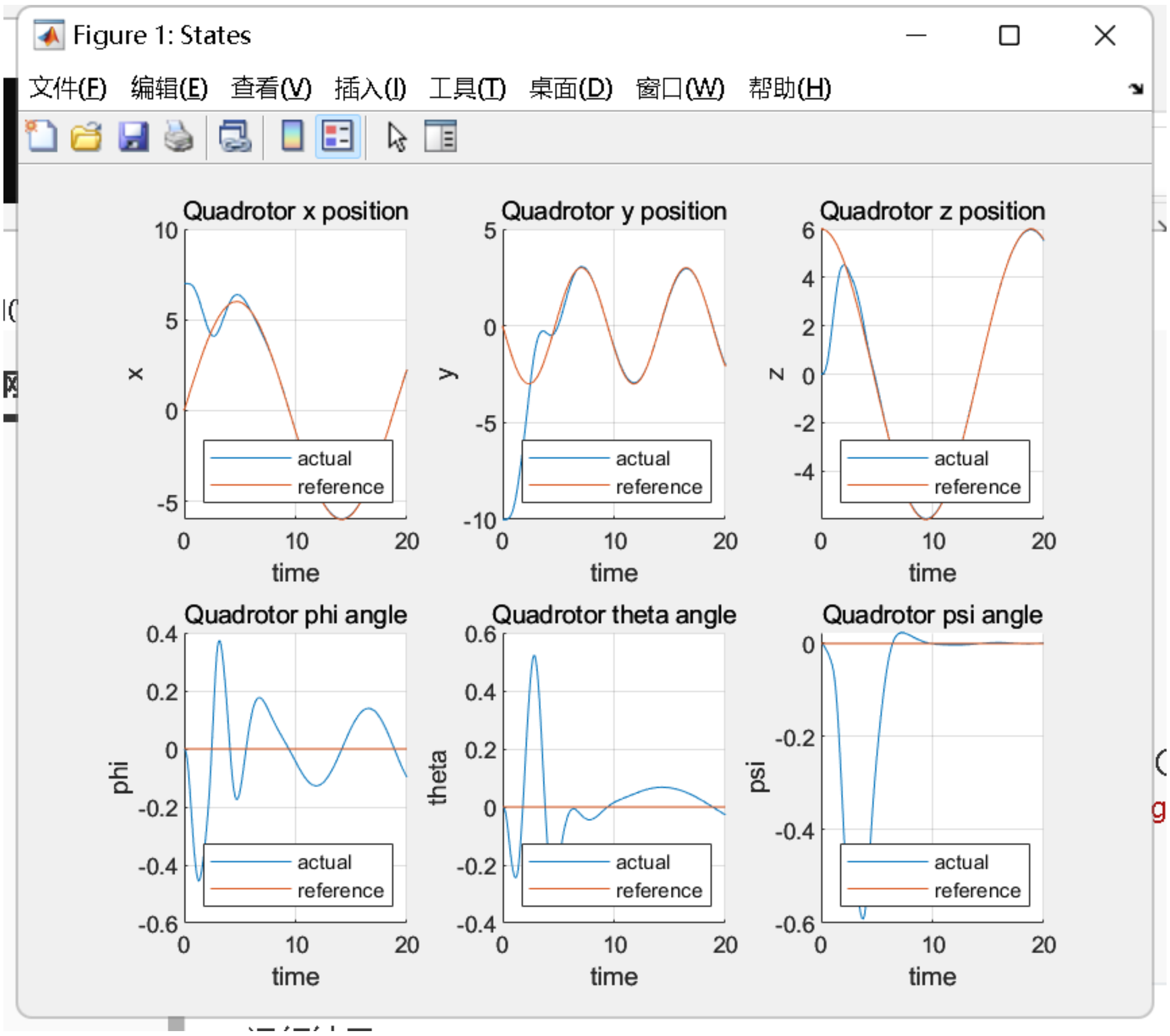
这里面，目标函数和约束函数都是nlmpcmove函数框架自己定义好的，下一节我们要自己写代码来实现非线性模型预测控制。

代码：

```
% 初始化
clear;clc;
% 状态变量个数
nx = 12;
% 输出变量个数
ny = 12;
% 输入变量个数
nu = 4;
% 时间不长
Ts = 0.1;
```

```matlab
% 预测时域和控制时域
p = 18;
m = 2;
% 定义mpc对象
nlmpcobj = nlmpc(nx, ny, nu);
nlmpcobj.Model.StateFcn = "QuadrotorStateFcn";
nlmpcobj.Jacobian.StateFcn = @QuadrotorStateJacobianFcn;
nlmpcobj.Ts = Ts;
nlmpcobj.PredictionHorizon = p;
nlmpcobj.ControlHorizon = m;
nlmpcobj.MV = struct( ...
    Min={0;0;0;0}, ...
    Max={10;10;10;10}, ...
    RateMin={-2;-2;-2;-2}, ...
    RateMax={2;2;2;2} ...
    );
nlmpcobj.Weights.OutputVariables = [1 1 1 1 1 1 0 0 0 0 0 0];
nlmpcobj.Weights.ManipulatedVariables = [0.1 0.1 0.1 0.1];
nlmpcobj.Weights.ManipulatedVariablesRate = [0.1 0.1 0.1 0.1];
% 初始状态
x = [7;-10;0;0;0;0;0;0;0;0;0;0];
% 目标状态
nloptions = nlmpcmoveopt;
nloptions.MVTarget = [4.9 4.9 4.9 4.9];
mv = nloptions.MVTarget;
Duration = 20;
% 记录上一步长的控制量
lastMV = mv;
% 保存状态数据和输入数据
xHistory = x';
uHistory = lastMV;
% 主循环
for k = 1:(Duration/Ts)
    disp(['loop: ', num2str(k), '/',num2str((Duration/Ts))])
    % 设定预测时域的参考轨迹
    t = linspace(k*Ts, (k+p-1)*Ts,p);
    yref = QuadrotorReferenceTrajectory(t);
    % 获取实际状态
    xk = xHistory(k,:);
    % mpc计算输入量
    [uk,nloptions,info] = nlmpcmove(nlmpcobj,xk,lastMV,yref',[],nloptions);
    lastMV = uk;
    % 仿真模型
    ODEFUN = @(t,xk) QuadrotorStateFcn(xk,uk);
    [TOUT,XOUT] = ode45(ODEFUN,[0 Ts], xHistory(k,:)');
    % 保存输入量和状态量
    uHistory(k+1,:) = uk';
    xHistory(k+1,:) = XOUT(end,:);
end
%% 绘图
plotQuadrotorTrajectory;
figure;
hold on;
for i = 1:size(xHistory, 1)
    cla;
    hold on;
    plot3(xHistory(1:i,1),xHistory(1:i,2),xHistory(1:i,3),'r.');
    plot3(yreftot(:,1),yreftot(:,2),yreftot(:,3),'g.');
    drone_Animation(xHistory(i, 1:6));
    drawnow;
end
```

运行结果：



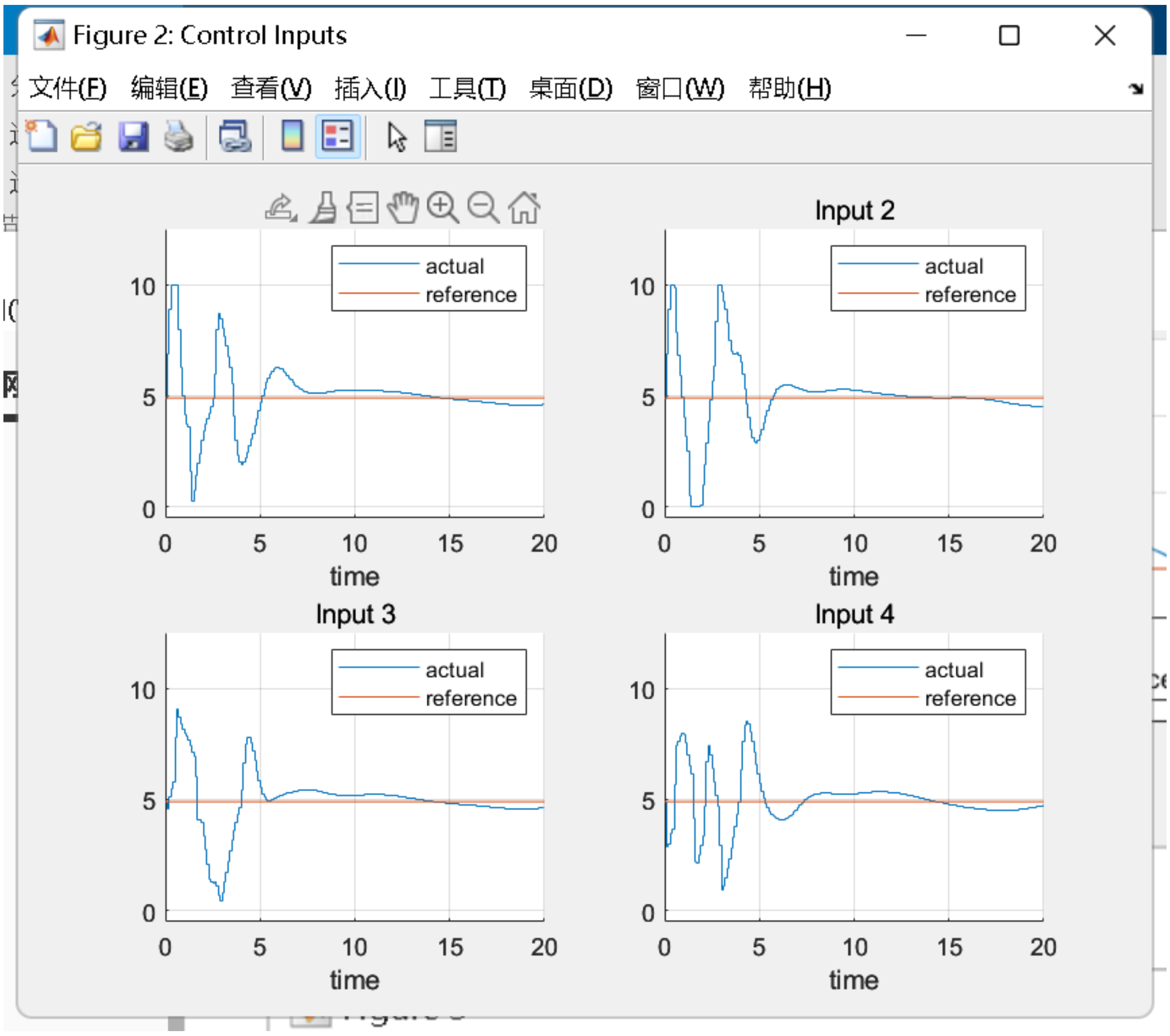
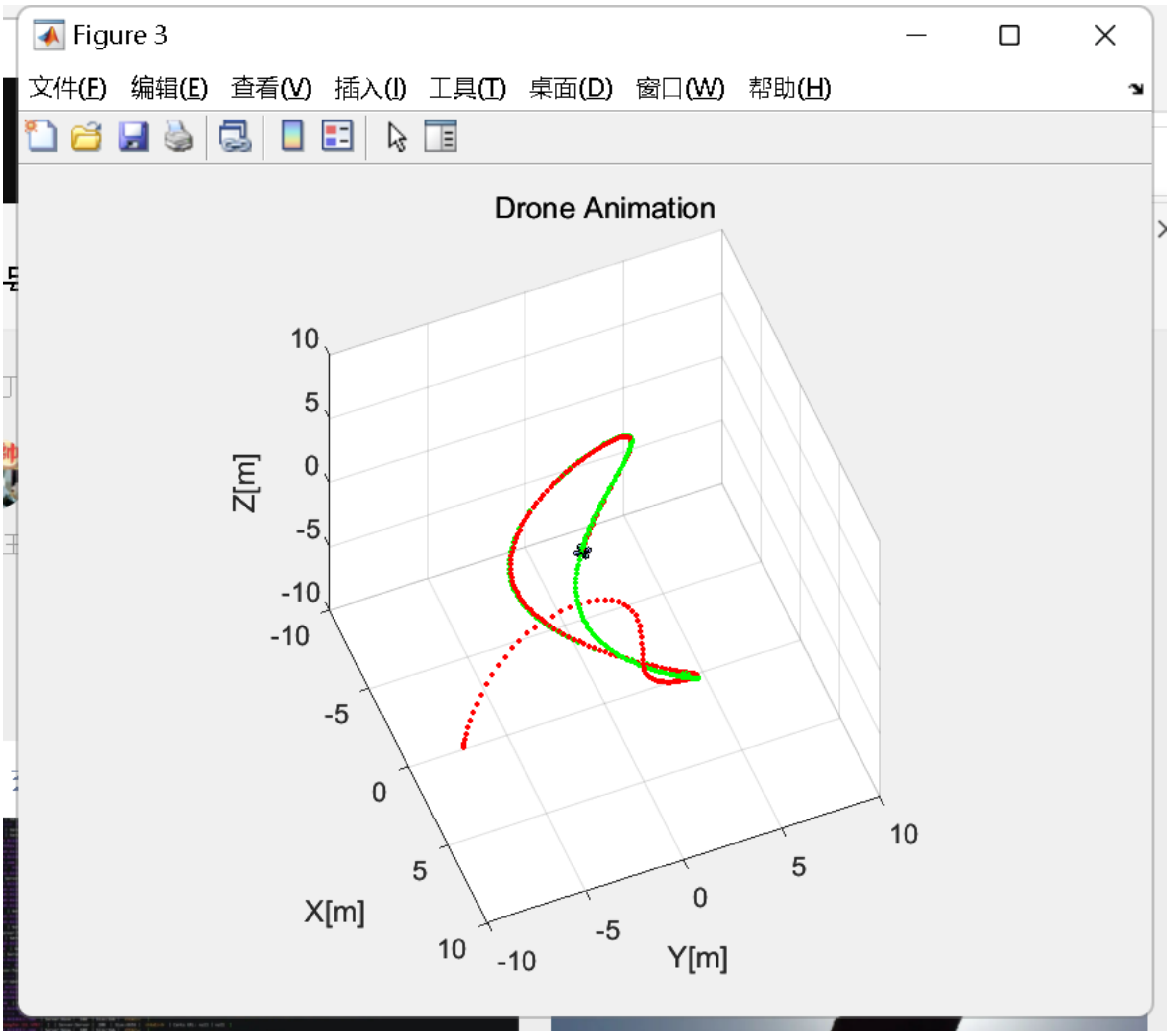


Figure 2: Control Inputs
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
Input 2
Input 3
Input 4
actual
reference
time
0
5
10
15
20



5.2.2 自己实现非线性模型预测控制

5.2.2.1 目标函数

$$J=\sum_{i=1}^{i=p} Q_x(x_i-x_{i,ref})^2+Q_y(y_i-y_{i,ref})^2+Q_z(z_i-z_{i,ref})^2$$

实现包装函数:

```
function f = CostFcnWrapper(z, data,yref)
p = 18;
nx = 12;
pnx = p*nx;
nu = 4;
nmv = 4;
Ts = 0.1;
p1 = p+1;
uz = z(p*nx+1:end-1);
Iz2u = [eye(2*4);[zeros(64,4),repmat(eye(4),16,1)]];
X = zeros(p1,nx);
U = zeros(p1,nu);
Xz = reshape(z(1:p*nx), nx, p)';
Uz = reshape(Iz2u*uz,nmv,p)';
X(1,:) = data.state;
X(2:p1,:) = Xz;
U(1:p1-1,:) = Uz;
e = z(end);
```

```
Xi = X';
Ui = U';
f = 0;
OutputVariables = [1 1 1 1 1 1 0 0 0 0 0 0];
ManipulatedVariables = [0.1 0.1 0.1 0.1];
ManipulatedVariablesRate = [0.1 0.1 0.1 0.1];
for i = 1:p
    % Calculate plant outputs
    xk = Xi(:,i+1);
    uk = Ui(:,i+1);
    yk = xk;
    yerr = yk - yref(:,i);
    Qy = OutputVariables';
    wtYerr = Qy.*yerr;
    f = f + wtYerr'*wtYerr;

    umvk = Ui(:,i);
    if i == 1
        duk = umvk - data.lastMV;
    else
        umvk1 = Ui(:,i-1);
        duk = umvk - umvk1;
    end
    Qu = ManipulatedVariables';
    wtu = Qu.*uk;
    f = f +wtu'*wtu;

%     size(wtu'*wtu)
    Qdu = ManipulatedVariablesRate';
    wtDu = Qdu.*duk;
    f = f + wtDu'*wtDu;
end
end
```

5.2.2.2 约束函数

设状态转换函数为

$$\frac{dx}{dt} = f(x)$$

那么约束函数为:

$$x_k + \left(f\left(x_k\right) + f\left(x_{k+1}\right)\right)\frac{T}{2} - x_{k+1} = 0$$

实现包装函数:

```
function [cineq, ceq] = ConFcnWrapper(z, data, yref)
p = 18;
nx = 12;
pnx = p*nx;
nu = 4;
nmv = 4;
Ts = 0.1;
p1 = p+1;
uz = z(p*nx+1:end-1);
Iz2u = [eye(2*4);[zeros(64,4),repmat(eye(4),16,1)]];
```

```
X = zeros(p1,nx);
U = zeros(p1,nu);
Xz = reshape(z(1:p*nx), nx, p)';
Uz = reshape(Iz2u*uz,nmv,p)';
X(1,:) = data.state;
X(2:p1,:) = Xz;
U(1:p1-1,:) = Uz;
ceq = zeros(pnx,1);
ic = 1:nx;
Ui = U';
Xi = X';
h = Ts/2;
cineq = [];
for i = 1:p
    uk  = Ui(:,i);
    xk  = Xi(:,i);
    xk1 = Xi(:,i+1);
    fk  = QuadrotorStateFcn(xk, uk);
    fk1 = QuadrotorStateFcn(xk1, uk);
    ceq(ic) = xk+h*(fk+fk1) - xk1;
    ic = ic + nx;
end
end
```

主函数:

```
% 初始化
clear;clc;
% 状态变量个数
nx = 12;
% 输出变量个数
ny = 12;
% 输入变量个数
nu = 4;
% 时间不长
Ts = 0.1;
% 预测时域和控制时域
p = 18;
m = 2;
% MPC参数
% 约束
MVMin=[0;0;0;0];
MVMax=[10;10;10;10];
RateMin={-2;-2;-2;-2};
RateMax={2;2;2;2};
% 权重
OutputVariables = [1 1 1 1 1 1 0 0 0 0 0 0];
ManipulatedVariables = [0.1 0.1 0.1 0.1];
ManipulatedVariablesRate = [0.1 0.1 0.1 0.1];
% 初始状态
x = [7;-10;0;0;0;0;0;0;0;0;0;0];
% 目标状态
MVTarget = [4.9 4.9 4.9 4.9];
nloptions.MVTarget = MVTarget;
Duration = 20;
```

```
% 记录上一步长的控制量
lastMV = MVTarget;
% 保存状态数据和输入数据
xHistory = x';
uHistory = lastMV;
mvs = [];
z = [];
data.lastMV = MVTarget';
% 主循环
for k = 1:(Duration/Ts)
    disp(['loop: ', num2str(k), '/',num2str((Duration/Ts))])
    % 设定预测时域的参考轨迹
    t = linspace(k*Ts, (k+p-1)*Ts,p);
    yref = QuadrotorReferenceTrajectory(t);
    % 获取实际状态
    xk = xHistory(k,:);
    % mpc计算输入量

    % 猜测初始值
    if isempty(z)
        X0 = repmat(xk,1,p);
    else
        X0 = z(nx+1:p*nx);
        X0 = [X0;X0(end-11:end)];
    end
    MV0 = repmat(MVTarget,m,1);
    xz = reshape(X0,p*nx,1);
    uz = reshape(MV0,m*nu,1);
    z0 = [xz; uz; 0];

    %计算线性不等式约束(A,B)
    A1  = [zeros(p*nu,p*nx), [eye(m*nu);[zeros(64,4),repmat(eye(nu),16,1)]],
zeros(p*nu,1)];
    B1  = zeros(p*nu,1);
    B1(1:p*nu) = repmat(MVMax,p,1);
    A2  = [zeros(p*nu,p*nx), -[eye(m*nu);[zeros(64,4),repmat(eye(nu),16,1)]],
zeros(p*nu,1)];
    B2  = zeros(p*nu,1);
    B2(1:p*nu) = repmat(-MVMin,p,1);
    A = [A1;A2];
    B = [B1;B2];

    % 计算zLB, zUB
    StateMin = -inf*ones(p,nx);
    StateMax = inf*ones(p,nx);
    xLB = reshape(StateMin', p*nx, 1);
    xUB = reshape(StateMax', p*nx, 1);
    nzu = m*nu;
    uLB = -Inf(nzu,1);
    uUB =  Inf(nzu,1);
    zLB = [xLB; uLB; 0];
    zUB = [xUB; uUB; Inf];

    % 设置CostFcn和ConFcn,
    data.state = xk;
    CostFcn = @(z)CostFcnWrapper(z,data,yref);
    ConFcn  = @(z)ConFcnWrapper(z,data,yref);
    % 求解NLP问题
    options = optimoptions(@fmincon,'MaxIterations',400,...
        'StepTolerance',1e-6,...
        'ConstraintTolerance',1e-6,...
        'OptimalityTolerance',1e-6,...
        'FunctionTolerance',0.01, ...
        'Display', 'off');
    [z, cost, ExitFlag, Out] = fmincon(CostFcn, z0, A, B, [], [], zLB, zUB,
ConFcn,options);
    uk = z(217:220);
    lastMV = uk;
    data.lastMV = lastMV;
    % 仿真模型
    ODEFUN = @(t,xk) QuadrotorStateFcn(xk,uk);
    [TOUT,XOUT] = ode45(ODEFUN,[0 Ts], xHistory(k,:)');
    % 保存输入量和状态量
    uHistory(k+1,:) = uk';
    xHistory(k+1,:) = XOUT(end,:);
end
%% 绘图
plotQuadrotorTrajectory;
figure;
hold on;
for i = 1:size(xHistory, 1)
    cla;
    hold on;
    plot3(xHistory(1:i,1),xHistory(1:i,2),xHistory(1:i,3),'r.');
    plot3(yreftot(:,1),yreftot(:,2),yreftot(:,3),'g.');
    drone_Animation(xHistory(i, 1:6));
    drawnow;
end
```