MPC模型预测控制从原理到代码实现

1 MPC原理

模型预测控制是依据模型对未来预测步长内的输出进行预测，根据预测目标来求解二次规划问题，从而得到未来的决策序列。

离散系统状态空间方程

$$X_{i+1}=AX_i+Bu_i+C$$

未来N个步长内

$$\begin{gathered}X_{i+2}=AX_{i+1}+Bu_{i+1}+C=A^2X_i+ABu_i+Bu_{i+1}+AC+C\\X_{i+3}=AX_{i+2}+Bu_{i+2}+C=A^3X_i+A^2Bu_i+ABu_{i+1}+Bu_{i+2}+A^2C+AC+C\\\dots\end{gathered}$$

那么

$$\begin{bmatrix}X_{i+1}\\X_{i+2}\\X_{i+3}\\\dots\\X_{i+N}\end{bmatrix}=\begin{bmatrix}AX_i\\A^2X_i\\A^3X_i\\\dots\\A^NX_i\end{bmatrix}+\begin{bmatrix}Bu_i\\ABu_i+Bu_{i+1}\\A^2Bu_i+ABu_{i+1}+Bu_{i+2}\\\dots\\A^{N-1}Bu_i+A^{N-2}Bu_{i+1}+\dots+Bu_{i+N-1}\end{bmatrix}+\begin{bmatrix}C\\AC+C\\A^2C+AC+C\\\dots\\A^{N-1}C+A^{N-2}C+\dots+AC+C\end{bmatrix}$$

记

$$X_{all}=\begin{bmatrix}X_{i+1}\\X_{i+2}\\X_{i+3}\\\dots\\X_{i+N}\end{bmatrix}$$

则

$$
\begin{aligned}
X_{all} = \begin{bmatrix} A \\ A^2 \\ A^3 \\ \dots \\ A^N \end{bmatrix} X_i + \begin{bmatrix} B & 0 & 0 & \dots & 0 \\ AB & B & 0 & \dots & 0 \\ A^2B & AB & B & \dots & 0 \\ \dots \\ A^{N-1}B & A^{N-2}B & A^{N-3}B & \dots & B \end{bmatrix} * \begin{bmatrix} u_i \\ u_{i+1} \\ u_{i+2} \\ \dots \\ u_{i+N-1} \end{bmatrix} + \\
\begin{bmatrix} C \\ AC+C \\ A^2C+AC+C \\ \dots \\ A^{N-1}C+A^{N-2}C+\dots+AC+C \end{bmatrix} \\
= \begin{bmatrix} B & 0 & 0 & \dots & 0 \\ AB & B & 0 & \dots & 0 \\ A^2B & AB & B & \dots & 0 \\ \dots \\ A^{N-1}B & A^{N-2}B & A^{N-3}B & \dots & B \end{bmatrix} * \begin{bmatrix} u_i \\ u_{i+1} \\ u_{i+2} \\ \dots \\ u_{i+N-1} \end{bmatrix} + \\
\begin{bmatrix} AX_i+C \\ A^2X_i+AC+C \\ A^3X_i+A^2C+AC+C \\ \dots \\ A^NX_i+A^{N-1}C+A^{N-2}C+\dots+AC+C \end{bmatrix} \\
== \begin{bmatrix} B & 0 & 0 & \dots & 0 \\ AB & B & 0 & \dots & 0 \\ A^2B & AB & B & \dots & 0 \\ \dots \\ A^{N-1}B & A^{N-2}B & A^{N-3}B & \dots & B \end{bmatrix} * \begin{bmatrix} u_i \\ u_{i+1} \\ u_{i+2} \\ \dots \\ u_{i+N-1} \end{bmatrix} + \\
\begin{bmatrix} AX_i+C \\ A(AX_i+C)+C \\ A(A(AX_i+C)+C)+C \\ \dots \\ A(A(A(\dots(AX_i+C)\dots)+C)+C)+C \end{bmatrix}
\end{aligned}
$$

记作

$$X_{all} = KU + M$$

构造目标函数：

$$
\begin{aligned}
f(U) = (X_{all} - X_{ref})^T Q (X_{all} - X_{ref}) + U^T RU \\
= (M + KU - X_{ref})^T Q (M + KU - X_{ref}) + U^T RU \\
f(U) = (KU)^T Q (KU) + U^T RU + 2(M - X_{ref})(KU)^T + \text{常数项}
\end{aligned}
$$

令

$$
\begin{aligned}
M_1 = K^T QK + R \\
M_2 = K^T Q (M - X_{ref})
\end{aligned}
$$

则$f(U)$可以用二次型表示

$$J = \frac{1}{2} U^T M_1 U + U^T M_2$$

2 代码实现

2.1 cpp实现

根据第一章的原理来分析apollo中mpc_solver的实现

函数声明：

```
bool SolveLinearMPC(
    const Eigen::MatrixXd &matrix_a, const Eigen::MatrixXd &matrix_b,
    const Eigen::MatrixXd &matrix_c, const Eigen::MatrixXd &matrix_q,
    const Eigen::MatrixXd &matrix_r, const Eigen::MatrixXd &matrix_lower,
    const Eigen::MatrixXd &matrix_upper,
    const Eigen::MatrixXd &matrix_initial_state,
    const std::vector<Eigen::MatrixXd> &reference, const double eps,
    const int max_iter, std::vector<Eigen::MatrixXd> *control);
```

函数定义

`matrix_t` 对应X_{ref},把N个参考状态组装成一个列向量即可。

`matrix_v` 对应U

`matrix_a_power` 对应$A^1\ A^2 \dots A^N$

`matrix_k` 对应矩阵K

$$\begin{bmatrix} B & 0 & 0 & \dots & 0 \\ AB & B & 0 & \dots & 0 \\ A^2B & AB & B & \dots & 0 \\ \dots \\ A^{N-1}B & A^{N-2}B & A^{N-3}B & \dots & B \end{bmatrix}$$

`matrix_m` 对应矩阵M

$$\begin{bmatrix} AX_i + C \\ A(AX_i + C) + C \\ A(A(AX_i + C) + C) + C \\ \dots \\ A(A(A(\dots(AX_i + C)\dots) + C) + C) + C \end{bmatrix}$$

```
// discrete linear predictive control solver, with control format
// x(i + 1) = A * x(i) + B * u (i) + C
bool SolveLinearMPC(const Matrix &matrix_a, const Matrix &matrix_b,
                    const Matrix &matrix_c, const Matrix &matrix_q,
                    const Matrix &matrix_r, const Matrix &matrix_lower,
                    const Matrix &matrix_upper,
                    const Matrix &matrix_initial_state,
                    const std::vector<Matrix> &reference, const double eps,
                    const int max_iter, std::vector<Matrix> *control) {
  if (matrix_a.rows() != matrix_a.cols() ||
      matrix_b.rows() != matrix_a.rows() ||
      matrix_lower.rows() != matrix_upper.rows()) {
    AERROR << "One or more matrices have incompatible dimensions. Aborting.";
    return false;
  }
  // 参考状态的个数，即预测步长数
  unsigned int horizon = reference.size();
```

```
  // 更新每一个参考状态到总的参考状态列向量中去
  // Update augment reference matrix_t
  Matrix matrix_t = Matrix::Zero(matrix_b.rows() * horizon, 1);
  for (unsigned int j = 0; j < horizon; ++j) {
    matrix_t.block(j * reference[0].size(), 0, reference[0].size(), 1) =
        reference[j];
  }

  // 初始化决策变量
  // Update augment control matrix_v
  Matrix matrix_v = Matrix::Zero((*control)[0].rows() * horizon, 1);
  for (unsigned int j = 0; j < horizon; ++j) {
    matrix_v.block(j * (*control)[0].rows(), 0, (*control)[0].rows(), 1) =
        (*control)[j];
  }

  // 求A^k序列
  std::vector<Matrix> matrix_a_power(horizon);
  matrix_a_power[0] = matrix_a;
  for (unsigned int i = 1; i < matrix_a_power.size(); ++i) {
    matrix_a_power[i] = matrix_a * matrix_a_power[i - 1];
  }

  //求矩阵K,这是低版本apollo的代码，有一个BUG，就是多乘以了一个矩阵A,新版本已经修复
//  Matrix matrix_k =
//      Matrix::Zero(matrix_b.rows() * horizon, matrix_b.cols() * horizon);
//  for (unsigned int r = 0; r < horizon; ++r) {
//    for (unsigned int c = 0; c <= r; ++c) {
//      matrix_k.block(r * matrix_b.rows(), c * matrix_b.cols(),
matrix_b.rows(),
//                     matrix_b.cols()) = matrix_a_power[r - c] * matrix_b;
//    }
//  }
    // 修复的版本
    Matrix matrix_k =
            Matrix::Zero(matrix_b.rows() * horizon, matrix_b.cols() * horizon);
    matrix_k.block(0, 0, matrix_b.rows(), matrix_b.cols()) = matrix_b;
    for (size_t r = 1; r < horizon; ++r) {
        for (size_t c = 0; c < r; ++c) {
            matrix_k.block(r * matrix_b.rows(), c * matrix_b.cols(),
matrix_b.rows(),
                           matrix_b.cols()) = matrix_a_power[r - c - 1] *
matrix_b;
        }
        matrix_k.block(r * matrix_b.rows(), r * matrix_b.cols(),
matrix_b.rows(),
                       matrix_b.cols()) = matrix_b;
    }
  // 初始化矩阵M , Q, R, ll, uu
  // Initialize matrix_k, matrix_m, matrix_t and matrix_v, matrix_qq, matrix_rr,
  // vector of matrix A power
  Matrix matrix_m = Matrix::Zero(matrix_b.rows() * horizon, 1);
  Matrix matrix_qq = Matrix::Zero(matrix_k.rows(), matrix_k.rows());
  Matrix matrix_rr = Matrix::Zero(matrix_k.cols(), matrix_k.cols());
  Matrix matrix_ll = Matrix::Zero(horizon * matrix_lower.rows(), 1);
  Matrix matrix_uu = Matrix::Zero(horizon * matrix_upper.rows(), 1);

  // 计算矩阵M
```

```cpp
// 用迭代的方式求M就行了
matrix_m.block(0, 0, matrix_a.rows(), 1) =
    matrix_a * matrix_initial_state + matrix_c;
for (unsigned int i = 1; i < horizon; ++i) {
  matrix_m.block(i * matrix_a.rows(), 0, matrix_a.rows(), 1) =
      matrix_a *
          matrix_m.block((i - 1) * matrix_a.rows(), 0, matrix_a.rows(), 1) +
      matrix_c;
}
// 接下来就是矩阵Q, R
// 以及二次规划的约束条件ll和rr
// Compute matrix_ll, matrix_uu, matrix_qq, matrix_rr
for (unsigned int i = 0; i < horizon; ++i) {
  matrix_ll.block(i * (*control)[0].rows(), 0, (*control)[0].rows(), 1) =
      matrix_lower;
  matrix_uu.block(i * (*control)[0].rows(), 0, (*control)[0].rows(), 1) =
      matrix_upper;
  matrix_qq.block(i * matrix_q.rows(), i * matrix_q.rows(), matrix_q.rows(),
                  matrix_q.rows()) = matrix_q;
  matrix_rr.block(i * matrix_r.rows(), i * matrix_r.rows(), matrix_r.cols(),
                  matrix_r.cols()) = matrix_r;
}
// 计算求解二次规划问题的矩阵M1和M2
// Update matrix_m1, matrix_m2, convert MPC problem to QP problem done
Matrix matrix_m1 = matrix_k.transpose() * matrix_qq * matrix_k + matrix_rr;
Matrix matrix_m2 = matrix_k.transpose() * matrix_qq * (matrix_m - matrix_t);

// Format in qp_solver
/**
 * *           min_x  : q(x) = 0.5 * x^T * Q * x  + x^T c
 * *           with respect to:  A * x = b (equality constraint)
 * *                             C * x >= d (inequality constraint)
 * **/

// TODO(QiL) : change qp solver to box constraint or substitute QPOASES
// Method 1: QPOASES
Matrix matrix_inequality_constrain_ll =
    Matrix::Identity(matrix_ll.rows(), matrix_ll.rows());
Matrix matrix_inequality_constrain_uu =
    Matrix::Identity(matrix_uu.rows(), matrix_uu.rows());
Matrix matrix_inequality_constrain =
    Matrix::Zero(matrix_ll.rows() + matrix_uu.rows(), matrix_ll.rows());
matrix_inequality_constrain << matrix_inequality_constrain_ll,
    -matrix_inequality_constrain_uu;
Matrix matrix_inequality_boundary =
    Matrix::Zero(matrix_ll.rows() + matrix_uu.rows(), matrix_ll.cols());
matrix_inequality_boundary << matrix_ll, -matrix_uu;
Matrix matrix_equality_constrain =
    Matrix::Zero(matrix_ll.rows() + matrix_uu.rows(), matrix_ll.rows());
Matrix matrix_equality_boundary =
    Matrix::Zero(matrix_ll.rows() + matrix_uu.rows(), matrix_ll.cols());

std::unique_ptr<QpSolver> qp_solver(new ActiveSetQpSolver(
    matrix_m1, matrix_m2, matrix_inequality_constrain,
    matrix_inequality_boundary, matrix_equality_constrain,
    matrix_equality_boundary));
auto result = qp_solver->Solve();
if (!result) {
```

```
    AERROR << "Linear MPC solver failed";
    return false;
  }
  matrix_v = qp_solver->params();

  for (unsigned int i = 0; i < horizon; ++i) {
    (*control)[i] =
        matrix_v.block(i * (*control)[0].rows(), 0, (*control)[0].rows(), 1);
  }
  return true;
}
```

2.2 matlab实现

根据前面的原理和cpp代码实现，来用matlab实现自己的SolveLinearMPC函数

```
function control = SolveLinearMPC(a, b, c, q, r, lower, upper, x0, refs,N)
    % 预测步长是N
    % 设状态量个数是Xn
    Xn = length(x0);
    % refs是N个参考状态组合成的参考状态合集
    % refs的维度是[N*Xn, 1]

    % 求矩阵k
    k = cell(N,N);
    for i = 1:N
        for j = 1:N
            if i < j
                k{i, j} =  b*0;
            else
                k{i,j} = a^(i-j)*b;
            end
        end
    end
    K = cell2mat(k);

    % 求矩阵M
    m = a*x0 + c;
    M = cell(N,1);
    for i = 1:N
        M{i} = m;
        m = a*m +c;
    end
    M = cell2mat(M);

    % Q,R
    Q = [];
    R = [];
    for i = 1:N
        Q = blkdiag(Q, q);
        R = blkdiag(R, r);
    end

    ll = repmat(lower, N, 1);
    uu = repmat(upper, N, 1);
    M1 = (K.')*Q*K+R;
```

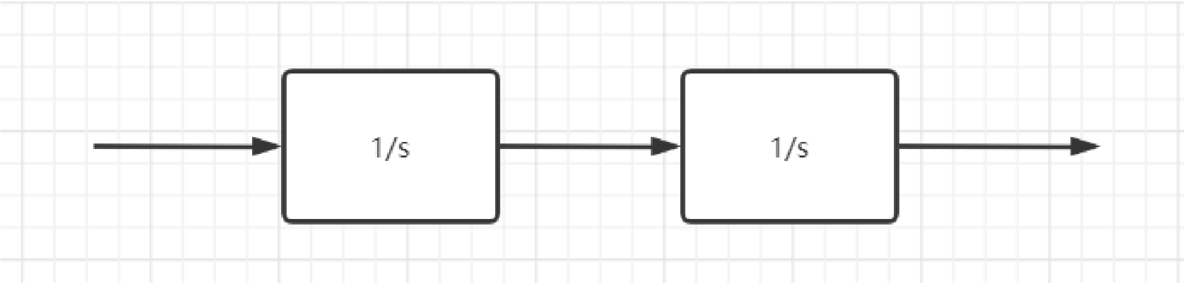
```
    M2 = (K.')*Q*(M-refs);
    [control,~,~,~,~] = quadprog(M1,M2,[],[],[],[],ll, uu);
end
```

3 MPC控制实践

3.1 双积分系统

3.1.1 模型分析

双积分系统就是由2个积分环节组成的系统



现实中典型的例子如从加速度控制位移。

它的状态空间方程如下:

$$\frac{dX}{dt} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} X + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u$$
$$Y = \begin{bmatrix} 1 & 0 \end{bmatrix} X + 0u$$

控制任务:

初始状态

$X = [0, 0]^T$

控制步长

0.1s

输出参考值

$Y_{ref} = 1$

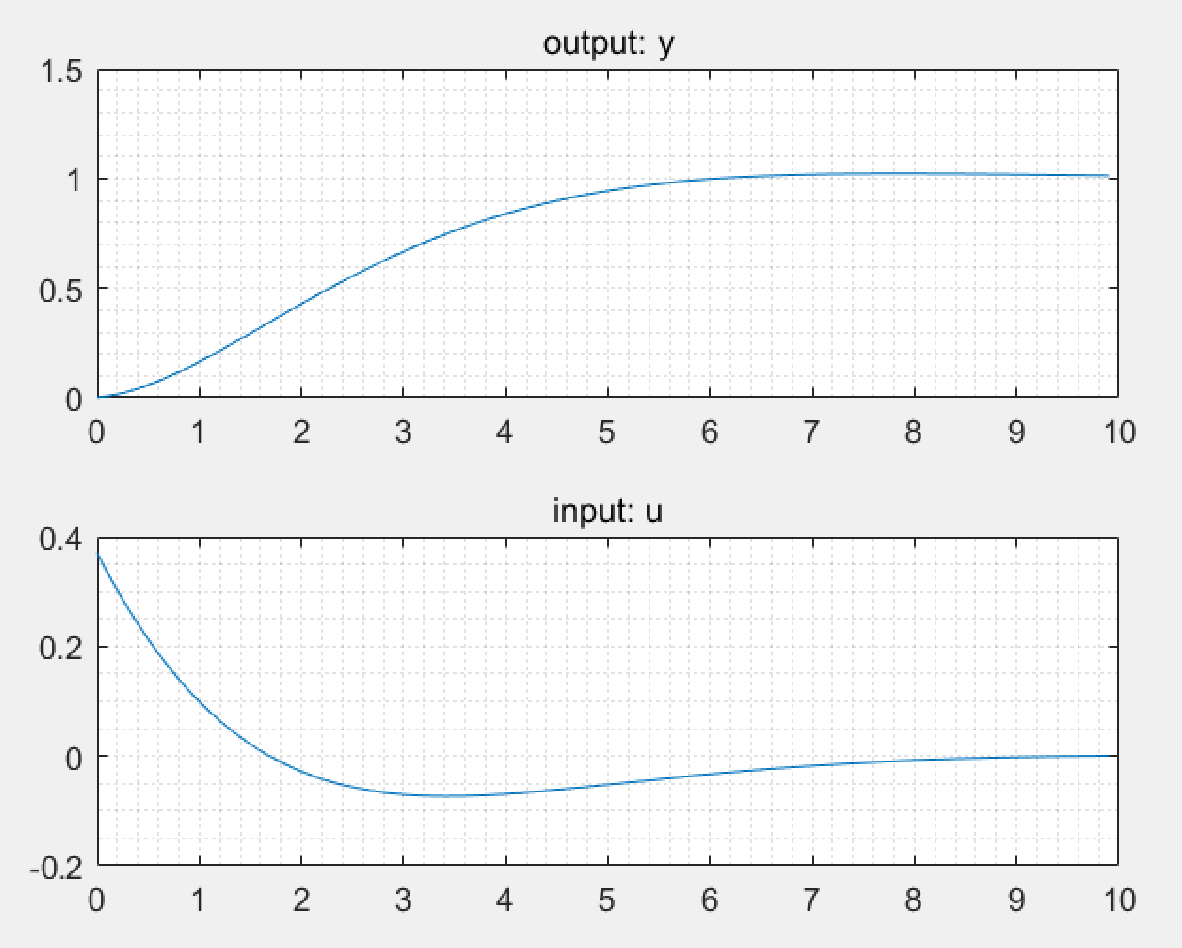
3.1.2 matlab实现MPC控制

主函数`double_int_mpc.m`

```
% doubleint system tf
% state space
% dx/dt = Ax + Bu
% y     = Cx + Du

% 离散化
clear;
Ts = 0.1;
s = ss([0,1;0,0],[0;1],[1,0],0);
s = c2d(s,Ts);
```

```
A = s.A;
B = s.B;

% 初始状态
x0 = [0;0];
ref = [1;0];
N = 10;
refs = repmat(ref,N,1);

% 保存数据
ys = [];
ts = [];
us = [];

% 初始状态
x = x0;
t = 0;

Q = [1 0;
     0 1];
R = 1;
low = -100;
hi = 100;
% 仿真循环
for i = 1:100
    % mpc求解
    z = SolveLinearMPC(A, B, x*0, Q, R, low, hi, x, refs, N);
    u = z(1);
    x = A*x+B*u;
    % 保存数据
    us = [us, u];
    ys = [ys; x(1)];
    ts = [ts; t];
    t = t + Ts;
end
% 绘图
subplot(2,1,1)
plot(ts, ys);
title('output: y')
grid minor
subplot(2,1,2)
plot(ts, us);
title('input: u')
grid minor
```

运行结果



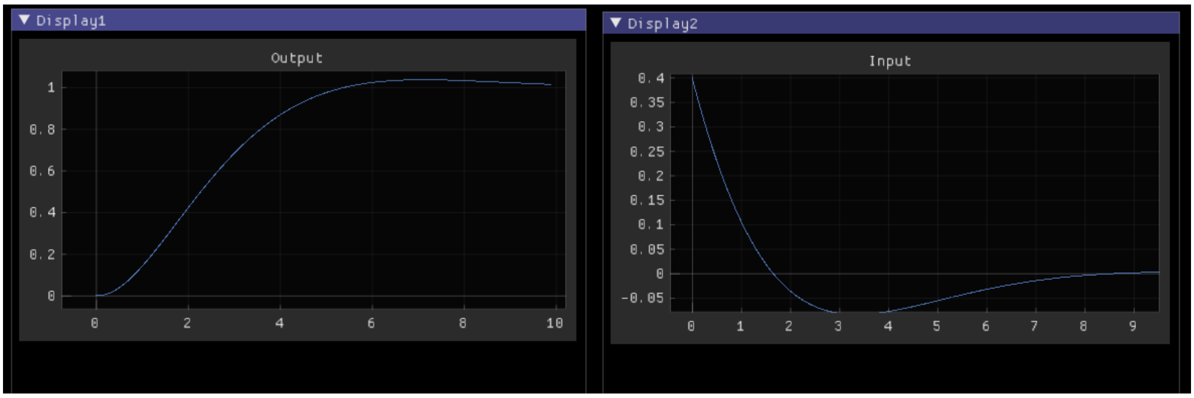
3.1.3 cpp实现MPC控制

根据前面的理论分析，同时调用apollo中的mpc_solver

```
// 状态量数目
    const int STATES = 2;
    // 控制量数目
    int CONTROLS = 1;
    // 预测步长
    const int HORIZON = 10;
    const double EPS = 0.01;
    const int MAX_ITER = 100;
    const double Ts = 0.1;
    Eigen::MatrixXd A(STATES, STATES);
    A << 0, 1, 0, 0;
    Eigen::MatrixXd B(STATES, CONTROLS);
    B << 0, 1;
    Eigen::MatrixXd C(STATES, 1);
    C << 0, 0;
    Eigen::MatrixXd Q(STATES, STATES);
    Q << 1, 0, 0, 1;
    Eigen::MatrixXd R(CONTROLS, CONTROLS);
    R << 1;
    // 离散化
    Eigen::MatrixXd Ad = Eigen::MatrixXd::Identity(STATES, STATES) + A*Ts;
    Eigen::MatrixXd Bd = B * Ts;
    Eigen::MatrixXd lower_bound(CONTROLS, 1);
    lower_bound << -100;
    Eigen::MatrixXd upper_bound(CONTROLS, 1);
    upper_bound << 100;
```

```
    // 初始状态
    Eigen::MatrixXd initial_state(STATES, 1);
    initial_state << 0, 0;
    // 参考状态
    Eigen::MatrixXd reference_state(STATES, 1);
    reference_state << 1, 0;

    std::vector<Eigen::MatrixXd> reference(HORIZON, reference_state);
    Eigen::MatrixXd control_matrix(CONTROLS, 1);
    control_matrix << 0;
    std::vector<Eigen::MatrixXd> control(HORIZON, control_matrix);

    // 记录数据
    std::vector<double> ts;
    std::vector<double> ys;

    // 仿真循环
    double t = 0.0;
    Eigen::MatrixXd X = initial_state;
    for(int i = 0; i < 100; ++i)
    {
        ys.push_back(X(0));
        ts.push_back(t);
        // 求解MPC问题
        SolveLinearMPC(Ad, Bd, C, Q, R, lower_bound, upper_bound, X,
                       reference, EPS, MAX_ITER, &control);
        double u = control[0](0, 0);

        // 被控对象仿真
        X = Ad*X +Bd*u;
        t += Ts;
    }
```

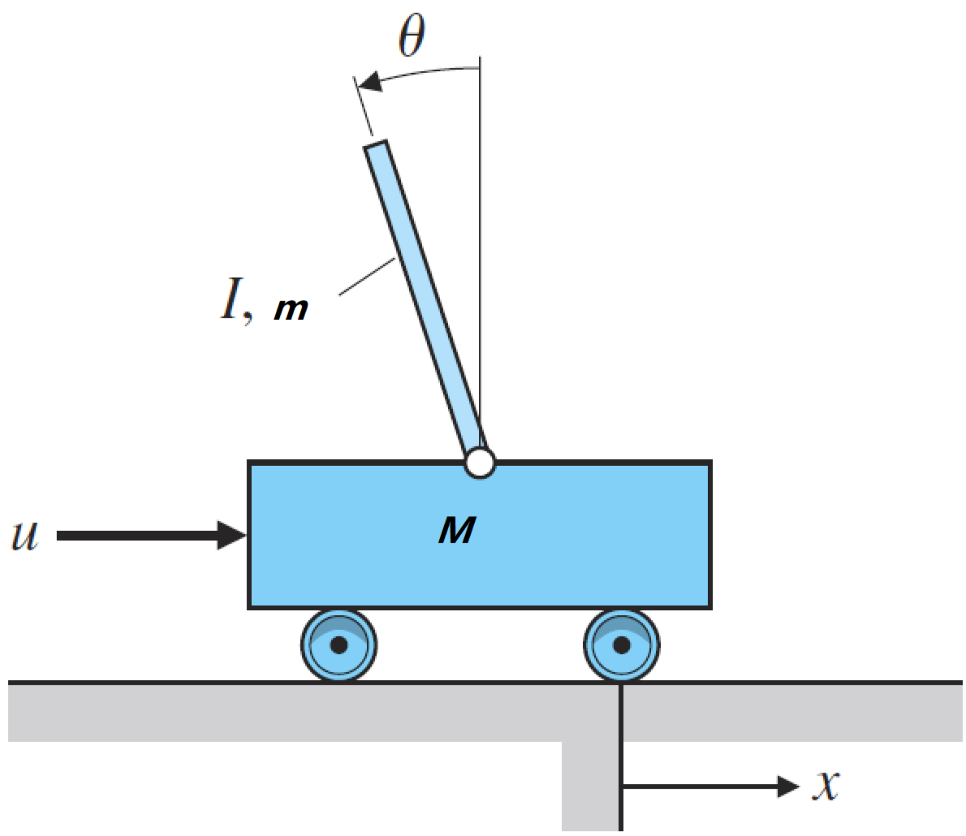
运行结果：



和matlab仿真结果一致。

3.2 倒立摆系统

3.2.1 模型分析



1 对小车水平手里方向进行分析

$$M\ddot{x} = u - b\dot{x} - N$$

N为杆对小车的水平力，杆仅由此水平力提供水平加速度

$$N = m\frac{d^2}{dt^2}(x - l\sin\theta)$$
$$N = m\ddot{x} - ml\ddot{\theta}\cos\theta + ml\dot{\theta}^2\sin\theta$$

代入得到

$$(M+m)\ddot{x} + b\dot{x} - ml\ddot{\theta}\cos\theta + ml\dot{\theta}^2\sin\theta = u$$

2 对杆进行手里分析

垂直方向受力分析：

$$P - mg = m\frac{d^2}{dt^2}(l\cos\theta)$$
$$P = mg - ml\ddot{\theta}\sin\theta - ml\dot{\theta}^2\cos\theta$$

P为小车对杆的支撑力。

杠绕重心力矩平衡：

$$Pl\sin\theta + Nl\cos\theta = I\ddot{\theta}$$

P，N带入：

$$
\begin{gathered}
mgl\sin\theta - ml\ddot{\theta}\sin\theta l\sin\theta - ml\dot{\theta}^2\cos\theta l\sin\theta \\
+ m\ddot{x}l\cos\theta - ml\ddot{\theta}\cos\theta l\cos\theta + ml\dot{\theta}^2\sin\theta l\cos\theta \\
= mgl\sin\theta - ml^2\ddot{\theta} + m\ddot{x}l\cos\theta = I\ddot{\theta}
\end{gathered}
$$

得到

$$
(I+ml^2)\ddot{\theta} - mgl\sin\theta = ml\ddot{x}\cos\theta
$$

联列2式:

$$
\begin{gathered}
(M+m)\ddot{x} + b\dot{x} - ml\ddot{\theta}\cos\theta + ml\dot{\theta}^2\sin\theta = u \\
(I+ml^2)\ddot{\theta} - mgl\sin\theta = ml\ddot{x}\cos\theta
\end{gathered}
$$

线性化

$$
\begin{gathered}
(M+m)\ddot{x} + b\dot{x} - ml\ddot{\theta} = u \\
(I+ml^2)\ddot{\theta} - mgl\theta = ml\ddot{x}
\end{gathered}
$$

状态空间变量定义为

$$
\begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix}
$$

改写方程组:

$$
\begin{gathered}
\text{记}\, p = I(M+m) + Mml^2 \\
\dot{x} = \dot{x} \\
\ddot{x} = -\tfrac{b(I+ml^2)}{p}\dot{x} + \tfrac{m^2gl^2}{p}\theta + \tfrac{(I+ml^2)}{p}u \\
\dot{\theta} = \dot{\theta} \\
\dot{\theta} = \tfrac{-bml}{p}\dot{x} + \tfrac{mgl(M+m)}{p}\theta + \tfrac{ml}{p}u
\end{gathered}
$$

$$
\frac{d}{dt}\begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -\frac{b(I+ml^2)}{p} & \frac{m^2gl^2}{p} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & -\frac{bml}{p} & \frac{mgl(M+m)}{p} & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{(I+ml^2)}{p} \\ 0 \\ \frac{ml}{p} \end{bmatrix} u
$$

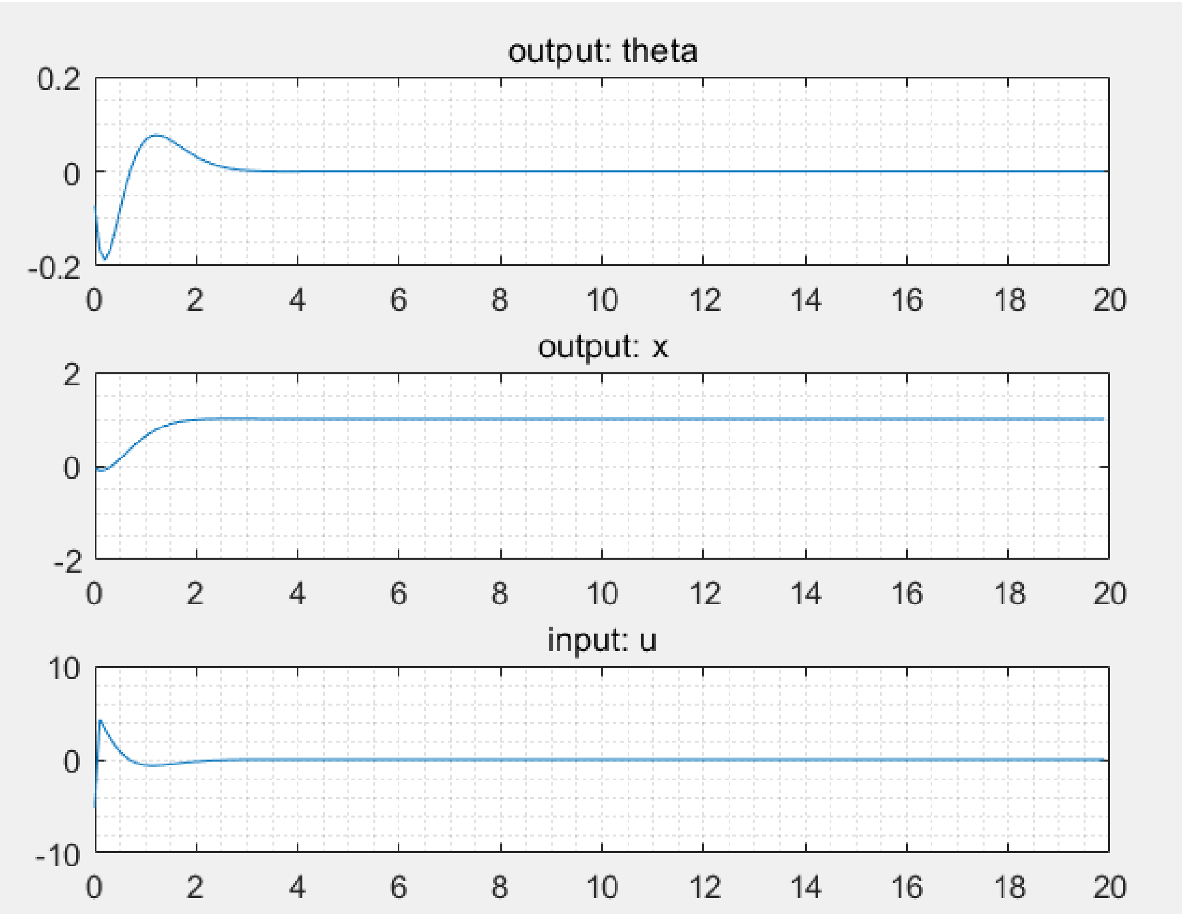
$$
\begin{gathered}
A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -\frac{b(I+ml^2)}{p} & \frac{m^2gl^2}{p} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & -\frac{bml}{p} & \frac{mgl(M+m)}{p} & 0 \end{bmatrix} \\
B = \begin{bmatrix} 0 \\ \frac{(I+ml^2)}{p} \\ 0 \\ \frac{ml}{p} \end{bmatrix} \\
C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \\
D = \begin{bmatrix} 0 \\ 0 \end{bmatrix}
\end{gathered}
$$

控制目标：控制倒立摆从初始位置0，移动到1。

3.2.2 matlab实现MPC控制

```
M = 0.5;
m  = 0.2;
b = 0.1;
I = 0.018;
g = 9.8;
L = 0.3;
q = (M+m )*(I+m *L^2)-(m *L)^2;
p = I*(m +M)+M*m *L^2;
A = [0  1  0  0
     0,  -(I+m *L^2)*b/p, m ^2*g*L^2/p, 0
     0  0  0  1
     0,  -m *L*b/p, m *g*L*(m +M)/p, 0];
B = [0;
     (I+m *L^2)/p;
     0;
     m *L/p];
C = [1  0  0  0
     0  0  1  0];
D = [0; 0];

Ts = 0.1;
A = eye(4)+A*Ts;
B = B*Ts;

% 初始状态 [x; dx; theta; dtheta]
x0 = [0;0;0;0];
ref = [1;0;0;0];
N = 10;
refs = repmat(ref,N,1);

% 保存数据
xs     = []; % x
thetas = []; % theta
ts     = []; % time
fs     = []; % F

% 初始状态
x = x0;
t = 0;

% Q矩阵,x和theta权重稍大一点
Q = [1 0 0 0
     0 1 0 0
     0 0 1 0
     0 0 0 1];
R = 0.1;
low = -100;
hi = 100;

% 仿真循环
for i = 1:200
    % mpc求解
    z = SolveLinearMPC(A, B, x*0, Q, R, low, hi, x, refs, N);
```

```
    u = z(1);
    x = A*x+B*u;
    % 保存数据
    fs = [fs, u];
    thetas = [thetas; x(3)];
    xs = [xs, x(1)];
    ts = [ts; t];
    t = t + Ts;
end
% 绘图
subplot(3,1,1)
plot(ts, thetas);
title('output: theta')
grid minor
subplot(3,1,2)
plot(ts, xs);
title('output: x')
grid minor
subplot(3,1,3)
plot(ts, fs);
title('input: u')
grid minor
```

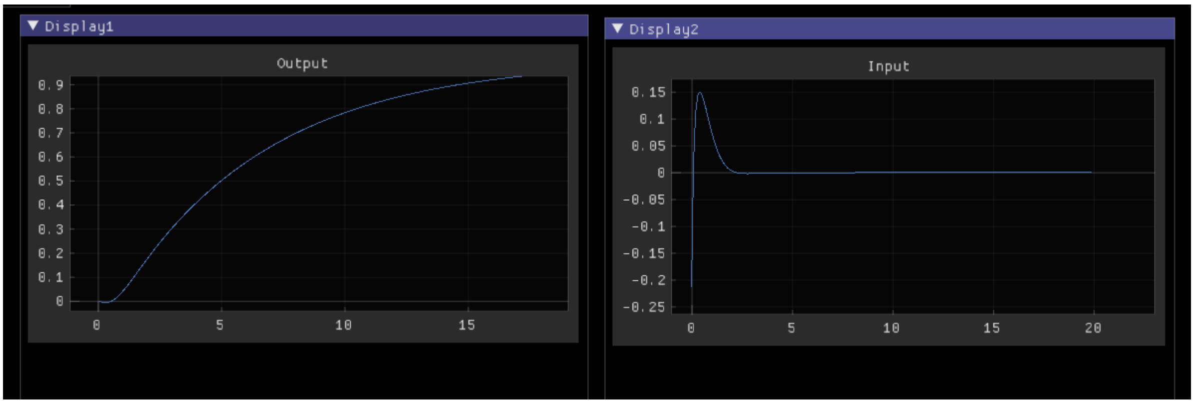


3.2.3 cpp实现MPC控制

```
#include "mpc_solver/mpc_solver.h"
#include <iostream>
// 倒立摆系统
void display(double* x_data, double* y_data, double* x_data1, double* y_data1,
int len);
```

```
int main() {
    double M = 0.5;
    double m  = 0.2;
    double b = 0.1;
    double I = 0.018;
    double g = 9.8;
    double L = 0.3;
    double q = (M+m )*(I+m *L*L)-(m *L)*(m *L);
    double p = I*(m +M)+M*m*L*L;
    double A22 = -(I+m*L*L)*b/p;
    double A23 = m*m*g*L*L/p;
    double A42 =  -m *L*b/p;
    double A43 = m *g*L*(m +M)/p;
    double B2 = (I+m *L*L)/p;
    double B4 = m*L/p;
    // 状态量数目
    const int STATES = 4;
    // 控制量数目
    int CONTROLS = 1;
    // 预测步长
    const int HORIZON = 10;
    const double EPS = 0.001;
    const int MAX_ITER = 1000;
    const double Ts = 0.1;
    Eigen::MatrixXd A(STATES, STATES);
    A << 0, 1, 0, 0, 0, A22, A23, 0, 0, 0, 0, 1, 0, A42, A43, 0;
    Eigen::MatrixXd B(STATES, CONTROLS);
    B << 0, B2, 0, B4;
    Eigen::MatrixXd C(STATES, 1);
    C << 0, 0, 0, 0;
    Eigen::MatrixXd Q(STATES, STATES);
    Q << 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0 ,0, 1;
    Eigen::MatrixXd R(CONTROLS, CONTROLS);
    R << 0.1;
    // 离散化
    Eigen::MatrixXd Ad = Eigen::MatrixXd::Identity(STATES, STATES) + A*Ts;
    Eigen::MatrixXd Bd = B * Ts;
    Eigen::MatrixXd lower_bound(CONTROLS, 1);
    lower_bound << -100;
    Eigen::MatrixXd upper_bound(CONTROLS, 1);
    upper_bound << 100;


    // 初始状态
    Eigen::MatrixXd initial_state(STATES, 1);
    initial_state << 0, 0, 0, 0;
    // 参考状态
    Eigen::MatrixXd reference_state(STATES, 1);
    reference_state << 1, 0, 0, 0;

    std::vector<Eigen::MatrixXd> reference(HORIZON, reference_state);
    Eigen::MatrixXd control_matrix(CONTROLS, 1);
    control_matrix << 0;
    std::vector<Eigen::MatrixXd> control(HORIZON, control_matrix);

    // 记录数据
    std::vector<double> ts;
    std::vector<double> ys;
```

```
    std::vector<double> us;
    // 仿真循环
    double t = 0.0;
    Eigen::MatrixXd X = initial_state;
    for(int i = 0; i < 200; ++i)
    {
        // 求解MPC问题
        SolveLinearMPC(Ad, Bd, C, Q, R, lower_bound, upper_bound, X,
                       reference, EPS, MAX_ITER, &control);
        double u = control[0](0, 0);
        us.push_back(u);
        // 被控对象仿真
        X = Ad*X +Bd*u;
        ys.push_back(X(0));
        ts.push_back(t);
        t += Ts;
    }

    double x_data[5000];
    double y_data[5000];
    double x_data1[5000];
    double y_data1[5000];
    for(int i = 0; i < ts.size(); ++i)
    {
        x_data[i] = ts[i];
        y_data[i] = ys[i];
        x_data1[i] = ts[i];
        y_data1[i] = us[i];
    }
    display(x_data,y_data,x_data1,y_data1,ts.size());
    return 0;
}
```



3.3 车辆运动学跟踪控制

3.3.1 模型分析

车辆运动学模型

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\varphi} \end{bmatrix} = \begin{bmatrix} \cos\varphi \\ \sin\varphi \\ \frac{\tan\delta}{l} \end{bmatrix} v$$

这个模型是一个非线性模型，需要将它线性化

$$
\dot{X}_r = f(X_r, U_r)
$$
$$
\dot{X}_k = f(X_k, U_k)
$$
$$
\text{在} X_r, U_r \text{处展开}
$$
$$
\dot{X}_k - f(X_r, U_r) = \frac{\partial f}{\partial X}(X_k - X_r) + \frac{\partial f}{\partial U}(U_k - U_r)
$$

记

$$
E_x = \begin{bmatrix} x_k - x_r \\ y_k - y_r \\ \theta_k - \theta_r \end{bmatrix}
$$
$$
E_u = \begin{bmatrix} v_k - v_r \\ \delta_k - \delta_r \end{bmatrix}
$$
$$
\dot{E}_x = \begin{bmatrix} 0 & 0 & -v_r sin\theta_r \\ 0 & 0 & v_r cos\theta_r \\ 0 & 0 & 0 \end{bmatrix} E_x + \begin{bmatrix} cos\theta_r & 0 \\ sin\theta_r & 0 \\ \frac{tan\delta_r}{l} & \frac{v_r}{lcos^2\delta_r} \end{bmatrix} E_u
$$

离散化

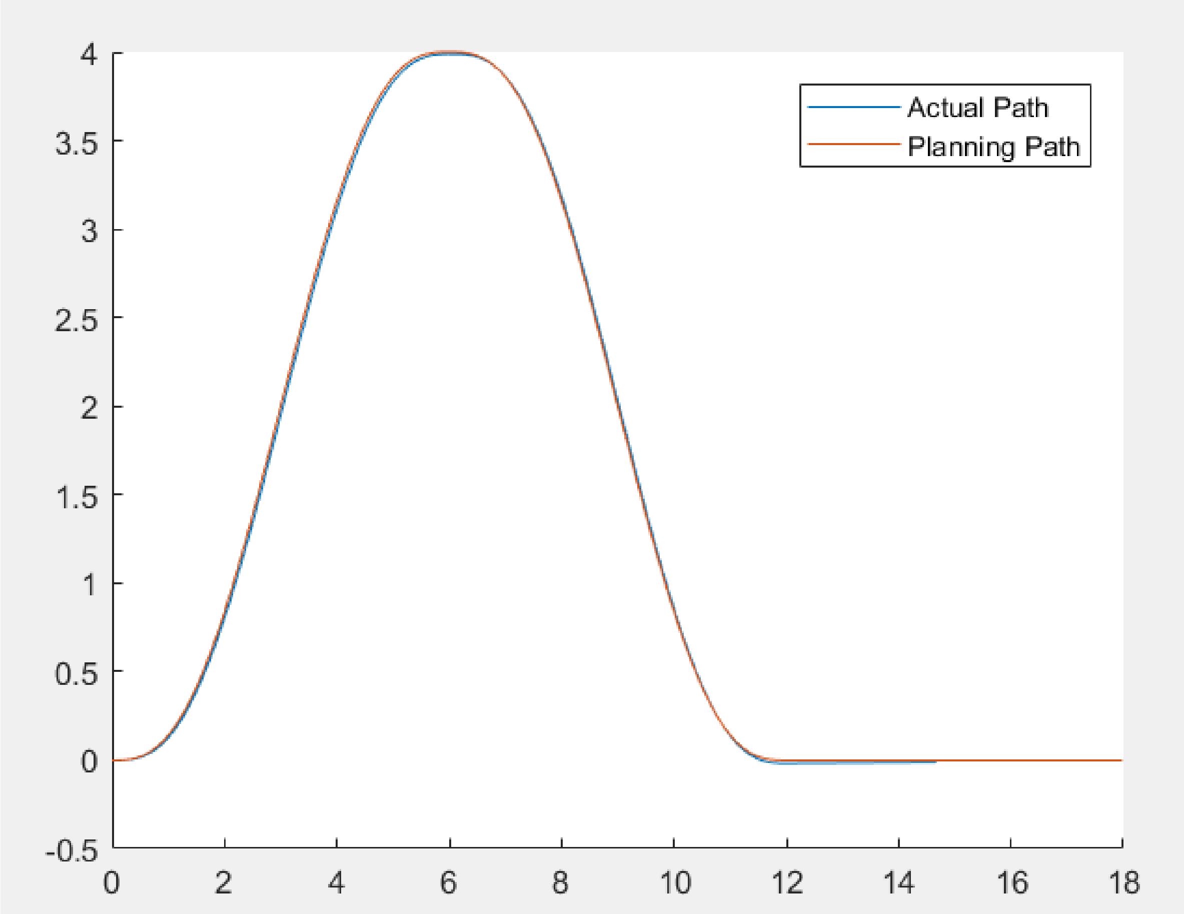
$$
A = \begin{bmatrix} 1 & 0 & -v_r sin\theta_r T \\ 0 & 1 & v_r cos\theta_r T \\ 0 & 0 & 1 \end{bmatrix}
$$
$$
B = \begin{bmatrix} cos\theta_r T & 0 \\ sin\theta_r T & 0 \\ \frac{tan\delta_r T}{l} & \frac{v_r T}{lcos^2\delta_r} \end{bmatrix}
$$

3.3.2 matlab实现mpc控制

```
% 车辆配置参数
VehConf.m  = 1573;
VehConf.Cf = 80000;
VehConf.Cr = 80000;
VehConf.Lf = 1.1;
VehConf.Lr = 1.58;
VehConf.Iz = 2873;
% 初始状态
x0     = 0;
y0     = 0;
phi0   = 0;
v0     = 2;
dT     = 0.01;
vofs   = 0;
% 生成期望轨迹
[x1, y1, v1, phi1] = GenRefLineSegment(x0,      y0,      v0,  v0,   'left', dT);
[x2, y2, v2, phi2] = GenRefLineSegment(x1(end), y1(end), v0,  v0+vofs, 'right',
  dT);
[x3, y3, v3, phi3] = GenRefLineSegment(x2(end), y2(end), v0+vofs,v0,   'center',
 dT);
x_ref = [x1(1:end-1);x2(1:end-1);x3(1:end-1)];
y_ref = [y1(1:end-1);y2(1:end-1);y3(1:end-1)];
v_ref = [v1(1:end-1);v2(1:end-1);v3(1:end-1)];
phi_ref = [phi1(1:end-1);phi2(1:end-1);phi3(1:end-1)];
s_ref = cumsum([0;sqrt(diff(x_ref).^2+diff(y_ref).^2)]);
len    = length(x_ref);
```

```
simT  = (0:(len-1))*dT;
% 初始状态空间变量
XCur = [y0; 0; phi0; 0];
VCur = v0;
Pos  = [x0; y0];
% 输入
delta   = 0;
acc     = 0;
Inputs  = [];
Outputs = [];
Ts      = [];

for i = 1:(length(simT)-10)
    t       = simT(i);
    Inputs  = [Inputs;delta, acc];
    Outputs = [Outputs;Pos(1),Pos(2),XCur(3),VCur];
    Ts      = [Ts; t];
    % 根据距离查找最近点
    idx = find(s_ref > v0*t,1);
    % 计算偏差
    x_err   = Pos(1) - x_ref(idx);
    y_err   = Pos(2) - y_ref(idx);
    phi     = XCur(3);
    phi_err = phi - phi_ref(idx);

    % 道路曲率
    if idx < 5
        a = [x_ref(idx),  y_ref(idx)];
        b = [x_ref(idx+1),y_ref(idx+1)];
        c = [x_ref(idx+2),y_ref(idx+2)];
    else
        a = [x_ref(idx-1),y_ref(idx-1)];
        b = [x_ref(idx),  y_ref(idx)];
        c = [x_ref(idx+1),y_ref(idx+1)];
    end
    kDes = -getCurvature(a,b,c);
    ErrState = [x_err;y_err;phi_err];
    % 得到运动学MPC计算相关的A, B, C矩阵
    [Ad, Bd, Cd] = GetKineMPCControlMatrix(VehConf, VCur, phi_ref(idx), kDes*
(VehConf.Lf+VehConf.Lr), dT);
    % 求解MPC问题
    u = SolveLinearMPC(Ad,Bd,Cd,diag([10,10,1]),diag([1,1]),[-1;-1],
[1;1],ErrState,zeros(3*5,1),5);
    delta = u(2)+kDes*(VehConf.Lf+VehConf.Lr);
    acc   = 0;
    % 施加控制到仿真模型
    [Pos, XCur, VCur] = GetNextPosition(VehConf, Pos, XCur, VCur, delta, acc,
dT);
end
hold on;
plot(Outputs(:,1),Outputs(:,2),'DisplayName','Actual Path')
plot(x_ref, y_ref,'DisplayName','Planning Path')
legend(gca,'show');
```

运行结果



3.3.3 cpp实现mpc控制

主函数

```
//
// Created by Lenovo on 2021/9/9.
//

#include "mpc_solver/mpc_solver.h"
#include <iostream>
#include <planning/DesireTrajectory.h>
#include <vehcontroller/KineMPCController.h>
#include <vehiclemodel/Vehicle.h>
// 车辆动力学控制
void display(double* x_data, double* y_data, int len, double* x_data1, double*
y_data1, int len1);
int main() {
    DesireTrajectory traj;
    Vehicle veh;

    KineMPCController controller;
    VehicleState vstate;
    ControlCommand cmd;
    std::vector<double> xs;
    std::vector<double> ys;
    double dT = 0.01;
    // 设置期望轨迹，这里是给定一个多项式变道的轨迹
    traj.SetDemoTrajData();
    // 设置仿真时间步长,初始状态
    veh.setSimDt(dT);
    veh.setInitPose(0, 0, 0);
```

```
    veh.setInitSpeed(3.0);
    double t = 0.0;
    for(int i = 0; i < 1150; ++i)
    {
        // Get Vehicle State
        vstate.xpos = veh.getX();
        vstate.ypos = veh.getY();
        vstate.speed = veh.getSpeed();
        vstate.phi = veh.getHeading();
        vstate.dphi = veh.getHeadingRate();
        // Compute control command
        controller.ComputeControlCommand(t, &vstate, &traj, &cmd);
        // Save data
        xs.push_back(vstate.xpos);
        ys.push_back(vstate.ypos);
        // Simulate
        veh.setAcc(cmd.acc);
        veh.setSteer(cmd.steer);
        veh.update();
        // update time
        t += dT;
    }
    double x_data[5000];
    double y_data[5000];
    double x_data1[5000];
    double y_data1[5000];
    for(int i = 0; i < xs.size(); ++i) {
        x_data[i] = xs[i];
        y_data[i] = ys[i];
    }
    for(int i = 0; i < traj.path.size(); ++i)
    {
        x_data1[i] = traj.path[i].x;
        y_data1[i] = traj.path[i].y;
    }
    display(x_data,y_data,xs.size(),x_data1,y_data1,traj.path.size());
    return 0;
}
```

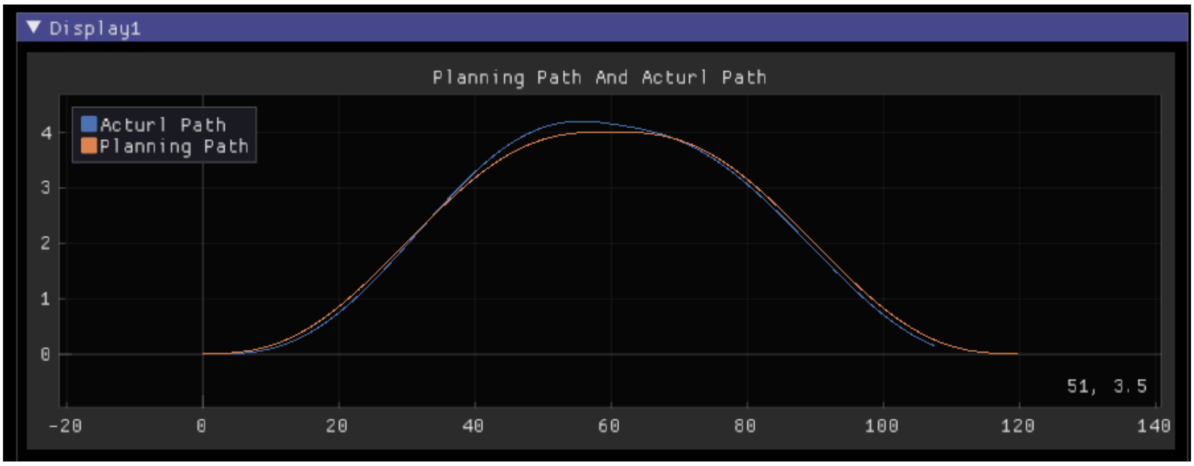
运动学模型MPC求解封装在KineMPCController类中

```
//
// Created by yuanyancheng on 2020/11/11.
//

#include "KineMPCControler.h"
#include "mpc_solver/mpc_solver.h"
#include <cmath>
#include "Eigen/LU"
#include "common/log.h"
#include <iostream>
using Matrix = Eigen::MatrixXd;
KineMPCController::KineMPCController()
{
    double thetar = 0;
    double vr = 0;
```

```cpp
    double deltar = 0;

    wheelbase_ = lf_ + lr_;
    // Matrix init operations.
    matrix_ad_ = Matrix::Zero(basic_state_size_, basic_state_size_);
    matrix_ad_(0, 0) = 1.0;
    matrix_ad_(0, 2) = -vr*sin(thetar)*ts_;
    matrix_ad_(1, 1) = 1.0;
    matrix_ad_(1, 2) = vr*cos(thetar)*ts_;
    matrix_ad_(2, 2) = 1.0;
    matrix_bd_ = Matrix::Zero(basic_state_size_, controls_);
    matrix_bd_(0, 0) = cos(thetar)*ts_;
    matrix_bd_(1, 0) = sin(thetar)*ts_;
    matrix_bd_(2, 0) = tan(deltar)*ts_/wheelbase_;
    matrix_bd_(2, 1) = vr*ts_/wheelbase_/cos(deltar)/cos(deltar);
    matrix_cd_ = Matrix::Zero(basic_state_size_, 1);
    matrix_state_ = Matrix::Zero(basic_state_size_, 1);
    matrix_r_ = Matrix::Identity(controls_, controls_);
    matrix_q_ = Matrix::Identity(basic_state_size_, basic_state_size_);
    double qarray[] = {5.0, 5.0, 1.0};
    for (int i = 0; i < 3; ++i) {
        matrix_q_(i, i) = qarray[i];
    }
}
void KineMPCController::ComputeStateErrors(double t, VehicleState* state,
DesireTrajectory* traj)
{
    reference_point = traj->QueryPathPointAtDistance(state->speed*t);
    double xTarget = reference_point.x;
    double yTarget = reference_point.y;
    //std::cout<<state->speed*t<<","<<reference_point.x<<","
<<reference_point.y<<std::endl;
    x_error   = state->xpos - xTarget;
    y_error   = state->ypos - yTarget;
    phi_error = state->phi  - reference_point.phi;
}
void KineMPCController::UpdateStateAnalyticalMatching()
{
    matrix_state_(0, 0) = x_error;
    matrix_state_(1, 0) = y_error;
    matrix_state_(2, 0) = phi_error;
}
void KineMPCController::UpdateMatrix(VehicleState* state) {
    double vr = state->speed;
    double deltar = reference_point.Curvature*wheelbase_;
    double thetar = reference_point.phi;
    matrix_ad_(0, 2) = -vr*sin(thetar)*ts_;
    matrix_ad_(1, 2) = vr*cos(thetar)*ts_;
    matrix_bd_(0, 0) = cos(thetar)*ts_;
    matrix_bd_(1, 0) = sin(thetar)*ts_;
    matrix_bd_(2, 0) = tan(deltar)*ts_/wheelbase_;
    matrix_bd_(2, 1) = vr*ts_/wheelbase_/cos(deltar)/cos(deltar);
}
void KineMPCController::ComputeControlCommand(double t, VehicleState* state,
DesireTrajectory* traj, ControlCommand *cmd)
{
    ComputeStateErrors(t, state, traj);
    UpdateStateAnalyticalMatching();
```

```
    UpdateMatrix(state);
    Eigen::MatrixXd control_matrix(controls_, 1);
    control_matrix << 0, 0;

    Eigen::MatrixXd reference_state(basic_state_size_, 1);
    reference_state << 0, 0, 0;

    std::vector<Eigen::MatrixXd> reference(horizon_, reference_state);

    Eigen::MatrixXd lower_bound(controls_, 1);
    lower_bound << -100, -10;

    Eigen::MatrixXd upper_bound(controls_, 1);
    upper_bound << 100, 10;

    std::vector<Eigen::MatrixXd> control(horizon_, control_matrix);

    if (SolveLinearMPC(
            matrix_ad_, matrix_bd_, matrix_cd_, matrix_q_,
            matrix_r_, lower_bound, upper_bound, matrix_state_, reference,
            mpc_eps_, mpc_max_iteration_, &control) != true) {
        AERROR << "MPC solver failed";
    } else {
        //AINFO << "MPC problem solved! ";
    }
    cmd->steer = control[0](1, 0) + reference_point.Curvature*wheelbase_;
    cmd->acc = 0;
}
```

运行结果



3.4 车辆动力学跟踪控制

3.4.1 模型分析

动力学跟踪控制模型采用误差状态方程模型

$$
\frac{d}{dt}\begin{bmatrix} e_1 \\ \dot{e}_1 \\ e_2 \\ \dot{e}_2 \\ e_3 \\ e_4 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & -\frac{2C_{af}+2C_{ar}}{mV_x} & \frac{2C_{af}+2C_{ar}}{m} & \frac{-2C_{af}\ell_f+2C_{ar}\ell_r}{mV_x} & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & -\frac{2C_{af}\ell_f-2C_{ar}\ell_r}{I_xV_x} & \frac{2C_{af}\ell_f-2C_{ar}\ell_r}{I_x} & -\frac{2C_{af}\ell_f^2+2C_a\ell_r^2}{I_xV_x} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} e_1 \\ \dot{e}_1 \\ e_2 \\ \dot{e}_2 \\ e_3 \\ e_4 \end{bmatrix}
$$

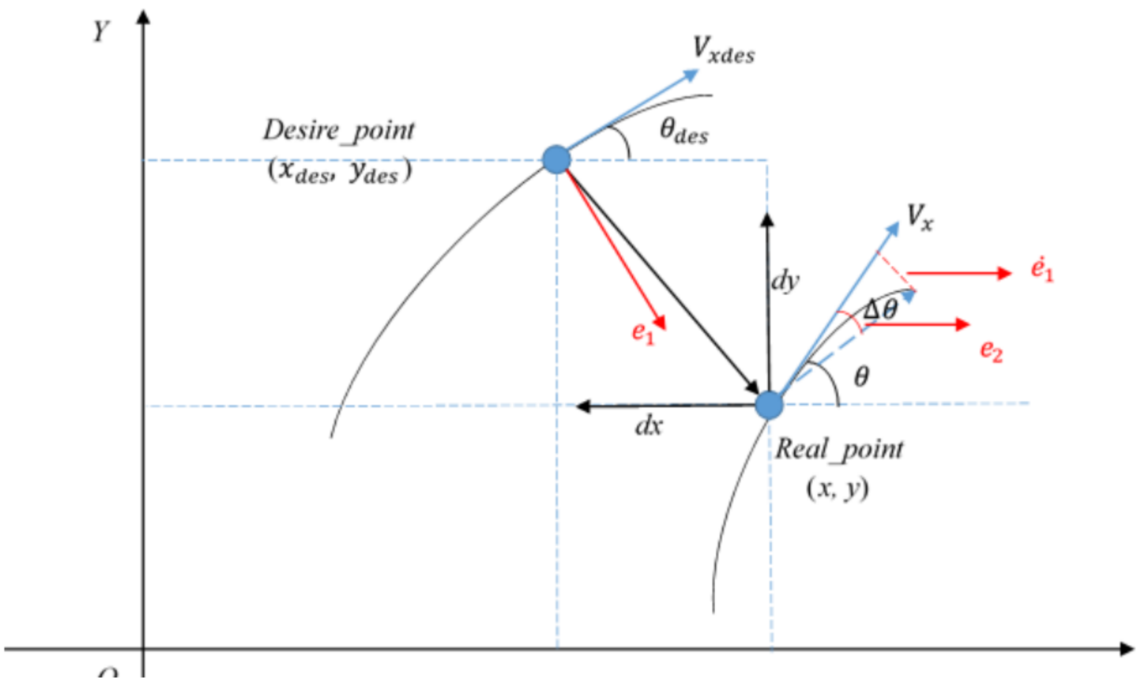
$$
+\begin{bmatrix} 0 & 0 \\ \frac{2C_{\alpha f}}{m} & 0 \\ 0 & 0 \\ \frac{2C_{\alpha f \ell_f}}{I_z} & 0 \\ 0 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} \delta \\ a \end{bmatrix} + \begin{bmatrix} 0 \\ -\frac{2C_{af}\ell_f-2C_a\ell_r}{mV_x} - V_x \\ 0 \\ -\frac{2C_{af}\ell_f^2+2C_{ar}\ell_r^2}{I_zV_x} \\ 0 \\ 1 \end{bmatrix} \dot{\varphi}_{des}
$$

推导过程见《车辆动力学与控制》

误差状态变量分别是：

- 横向误差 lateral_error, e_1
- 横向误差率 lateral_error_rate, $\dot{e_1}$
- 航向误差 heading_error, e_2
- 航向误差率 heading_error_rate, $\dot{e_2}$
- 速度误差speed_error
- 位置误差station_error

横向误差的计算：



$$
\begin{cases} e_1 = dy * \cos\theta_{\text{des}} - dx * \sin\theta_{\text{des}} \\ \dot{e_1} = V_x * \sin\Delta\theta = V_x * \sin e_2 \\ e_2 = \theta - \theta_{\text{des}} \\ \dot{e_2} = \dot{\theta} - \dot{\theta}_{\text{des}} \end{cases}
$$

纵向误差的计算

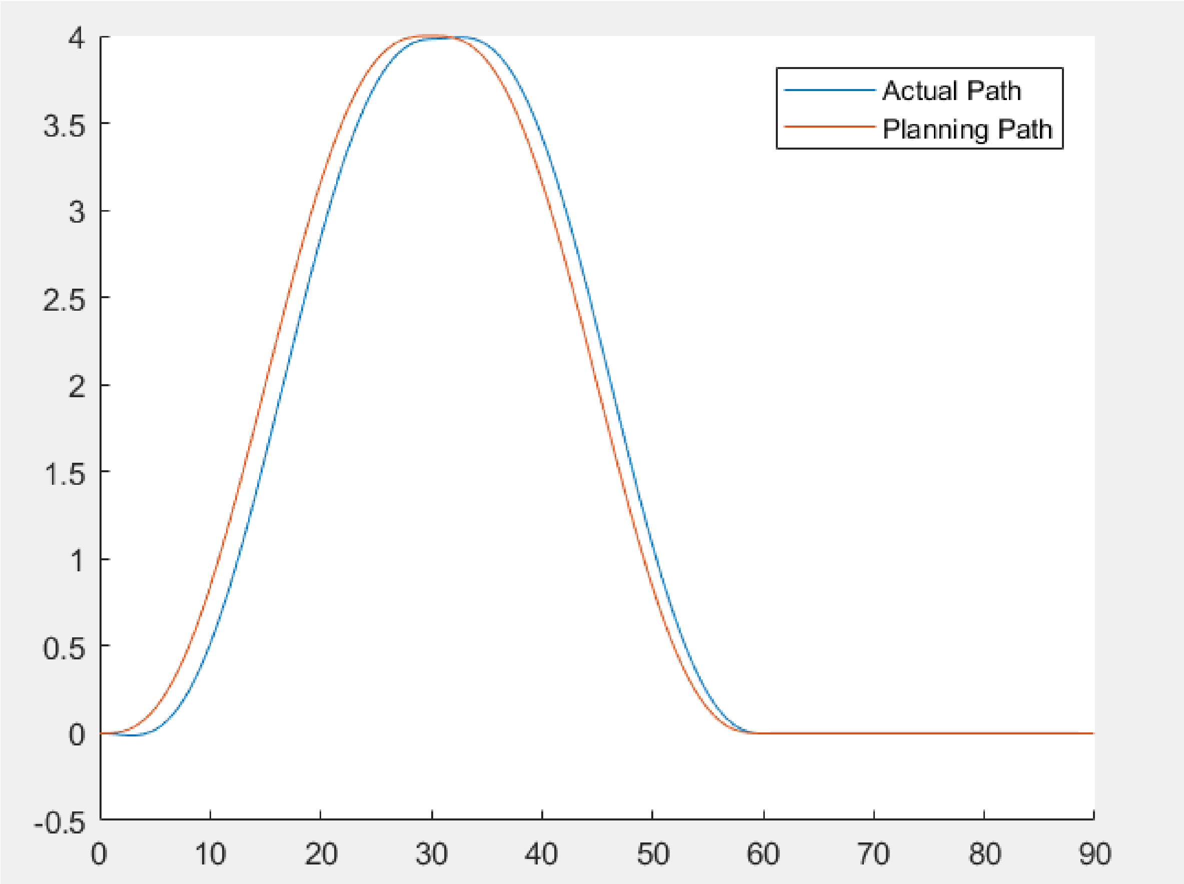
$$
\begin{aligned} \text{station\_error} &= -(dx * \cos\theta_{\text{des}} + dy * \sin\theta_{\text{des}}) \\ \text{speed\_error} &= V_{\text{des}} - V * \cos\Delta\theta / k \end{aligned}
$$

3.4.2 matlab实现MPC控制

```
% 车辆配置参数
VehConf.m  = 1573;
VehConf.Cf = 80000;
VehConf.Cr = 80000;
VehConf.Lf = 1.1;
VehConf.Lr = 1.58;
VehConf.Iz = 2873;
% 初始状态
x0    = 0;
y0    = 0;
phi0  = 0;
v0    = 10;
dT    = 0.01;
vofs  = 0;
% 生成期望轨迹
[x1, y1, v1, phi1] = GenRefLineSegment(x0,      y0,      v0,  v0,   'left', dT);
[x2, y2, v2, phi2] = GenRefLineSegment(x1(end), y1(end), v0,  v0+vofs, 'right',
  dT);
[x3, y3, v3, phi3] = GenRefLineSegment(x2(end), y2(end), v0+vofs,v0,   'center',
 dT);
x_ref = [x1(1:end-1);x2(1:end-1);x3(1:end-1)];
y_ref = [y1(1:end-1);y2(1:end-1);y3(1:end-1)];
v_ref = [v1(1:end-1);v2(1:end-1);v3(1:end-1)];
phi_ref = [phi1(1:end-1);phi2(1:end-1);phi3(1:end-1)];
len   = length(x_ref);
simT  = (0:(len-1))*dT;
% 初始状态空间变量
XCur = [y0; 0; phi0; 0];
VCur = v0;
Pos  = [x0; y0];
% 输入
delta   = 0;
acc     = 0;
Inputs  = [];
Outputs = [];
Ts      = [];

for i = 1:(length(simT)-10)
    t       = simT(i);
    Inputs  = [Inputs;delta, acc];
    Outputs = [Outputs;Pos(1),Pos(2),XCur(3),VCur];
    Ts      = [Ts; t];

    % 计算偏差
    phi      = XCur(3);
    dphi     = XCur(4);
    vDes     = v_ref(i);
    phiDes   = phi_ref(i);

    % 道路曲率
    if i < 5
        a = [x_ref(i),  y_ref(i)];
        b = [x_ref(i+1),y_ref(i+1)];
        c = [x_ref(i+2),y_ref(i+2)];
    else
```

```
        a = [x_ref(i-1),y_ref(i-1)];
        b = [x_ref(i),  y_ref(i)];
        c = [x_ref(i+1),y_ref(i+1)];
    end
    kDes = -getCurvature(a,b,c);
    dphiDes = vDes*kDes;
    xTarget = x_ref(i);
    yTarget = y_ref(i);
    dxTarget = Pos(1) - xTarget;
    dyTarget = Pos(2) - yTarget;

    heading_error      = phi - phiDes;
    heading_error_rate = dphi - dphiDes;
    lateral_error      = dyTarget*cos(phiDes) - dxTarget*sin(phiDes);
    lateral_error_rate = VCur*sin(heading_error);

    kParam = 1;
    speed_error =  vDes - VCur*cos(heading_error)/kParam;
    station_error = -(dxTarget*cos(phiDes) + dyTarget*sin(phiDes));
    ErrState =
[lateral_error;lateral_error_rate;heading_error;heading_error_rate;
station_error; speed_error];
    % 计算横纵向控制指令
    % 得到MPC计算相关的A, B, C矩阵
    [A, B, C] = GetMPCControlMatrix(VehConf, VCur);
    Ad = eye(6) + dT*A;
    Bd = dT*B;
    Cd = dT*C*heading_error_rate;
    % 求解MPC问题
    u = SolveLinearMPC(Ad,Bd,Cd,diag([1,1,1,1,1,1]),diag([1,1]),[-1;-1],
[1;1],ErrState,zeros(6*10,1),10);
    delta = u(1);
    acc   = u(2);
    % 施加控制到仿真模型
    [Pos, XCur, VCur] = GetNextPosition(VehConf, Pos, XCur, VCur, delta, acc,
dT);
end
hold on;
plot(Outputs(:,1),Outputs(:,2),'DisplayName','Actual Path')
plot(x_ref, y_ref,'DisplayName','Planning Path')
legend(axes1,'show');
```

运行结果



3.4.3 cpp实现MPC控制

主函数

```
#include "mpc_solver/mpc_solver.h"
#include <iostream>
#include <planning/DesireTrajectory.h>
#include <vehcontroller/MPCControler.h>
#include <vehiclemodel/Vehicle.h>
// 车辆动力学控制
void display(double* x_data, double* y_data, double* x_data1, double* y_data1,
int len);
int main() {
    DesireTrajectory traj;
    Vehicle veh;
    MPCController controller;
    VehicleState vstate;
    ControlCommand cmd;
    std::vector<double> xs;
    std::vector<double> ys;
    double dT = 0.01;
    // 设置期望轨迹，这里是给定一个多项式变道的轨迹
    traj.SetDemoTrajData();
    // 设置仿真时间步长,初始状态
    veh.setSimDt(dT);
    veh.setInitPose(0, 0, 0);
    veh.setInitSpeed(10.0);
    double t = 0.0;
    for(int i = 0; i < 1200; ++i)
    {
        // Get Vehicle State
        vstate.xpos = veh.getX();
```

```
        vstate.ypos = veh.getY();
        vstate.speed = veh.getSpeed();
        vstate.phi = veh.getHeading();
        vstate.dphi = veh.getHeadingRate();
        // Compute control command
        controller.ComputeControlCommand(t, &vstate, &traj, &cmd);
        // Save data
        xs.push_back(vstate.xpos);
        ys.push_back(vstate.ypos);
        // Simulate
        veh.setAcc(cmd.acc);
        veh.setSteer(cmd.steer);
        veh.update();
        // update time
        t += dT;
    }
    double x_data[5000];
    double y_data[5000];
    double x_data1[5000];
    double y_data1[5000];
    for(int i = 0; i < xs.size(); ++i)
    {
        x_data[i] = xs[i];
        y_data[i] = ys[i];
        x_data1[i] = traj.path[i].x;
        y_data1[i] = traj.path[i].y;
    }
    display(x_data,y_data,x_data1,y_data1,xs.size());
    return 0;
}
```

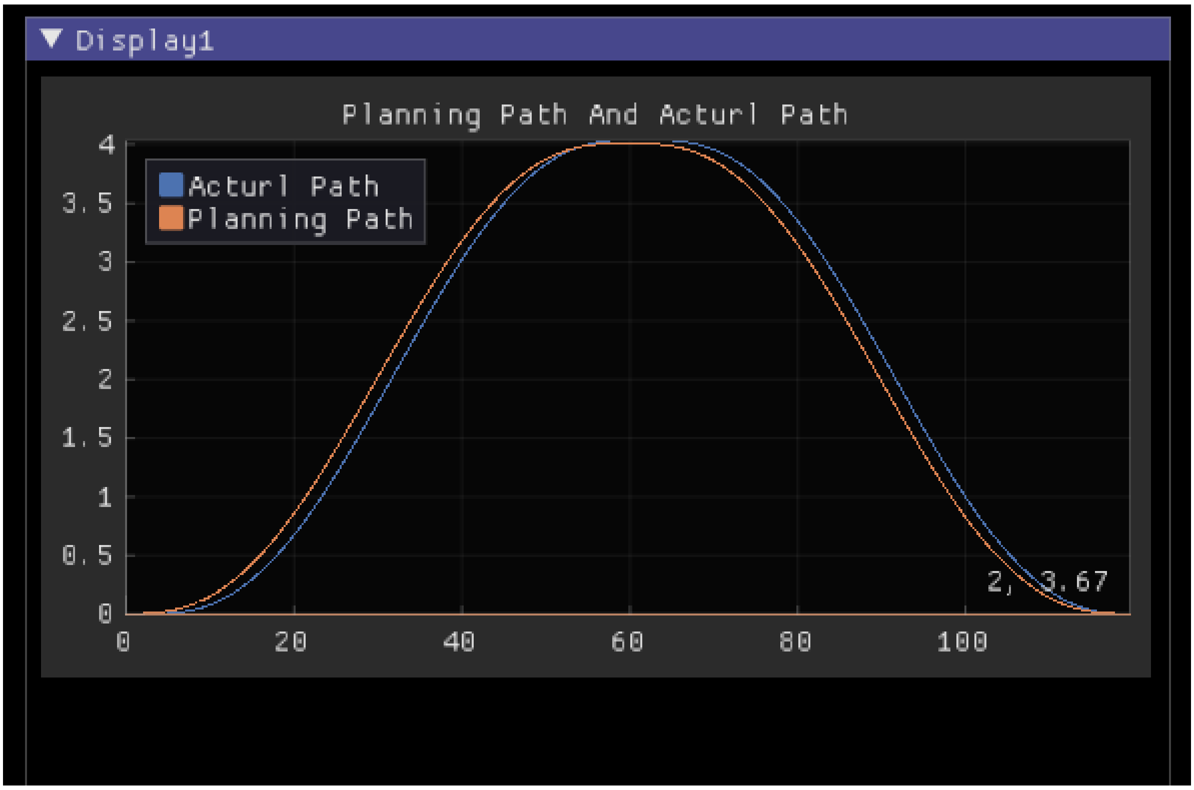
运动学模型MPC求解封装在MPCController类中

```
//
// Created by yuanyancheng on 2020/11/11.
//

#include "MPCControler.h"
#include "mpc_solver/mpc_solver.h"
#include <cmath>
#include "Eigen/LU"
#include "common/log.h"

using Matrix = Eigen::MatrixXd;
//构造函数
MPCController::MPCController()
{
    wheelbase_ = lf_ + lr_;
    // Matrix init operations.
    matrix_a_  = Matrix::Zero(basic_state_size_, basic_state_size_);
    matrix_ad_ = Matrix::Zero(basic_state_size_, basic_state_size_);
    matrix_a_(0, 1) = 1.0;
    matrix_a_(1, 2) = (cf_ + cr_) / mass_;
    matrix_a_(2, 3) = 1.0;
    matrix_a_(3, 2) = (lf_ * cf_ - lr_ * cr_) / iz_;
```

```
    matrix_a_(4, 5) = 1.0;
    matrix_a_(5, 5) = 0.0;
    matrix_a_coeff_ = Matrix::Zero(basic_state_size_, basic_state_size_);
    matrix_a_coeff_(1, 1) = -(cf_ + cr_) / mass_;
    matrix_a_coeff_(1, 3) = (lr_ * cr_ - lf_ * cf_) / mass_;
    matrix_a_coeff_(2, 3) = 1.0;
    matrix_a_coeff_(3, 1) = (lr_ * cr_ - lf_ * cf_) / iz_;
    matrix_a_coeff_(3, 3) = -1.0 * (lf_ * lf_ * cf_ + lr_ * lr_ * cr_) / iz_;
    matrix_b_ = Matrix::Zero(basic_state_size_, controls_);
    matrix_bd_ = Matrix::Zero(basic_state_size_, controls_);
    matrix_b_(1, 0) = cf_ / mass_;
    matrix_b_(3, 0) = lf_ * cf_ / iz_;
    matrix_b_(4, 1) = 0.0;
    matrix_b_(5, 1) = -1.0;
    matrix_bd_ = matrix_b_ * ts_;
    matrix_c_ = Matrix::Zero(basic_state_size_, 1);
    matrix_c_(5, 0) = 1.0;
    matrix_cd_ = Matrix::Zero(basic_state_size_, 1);
    matrix_state_ = Matrix::Zero(basic_state_size_, 1);
    matrix_r_ = Matrix::Identity(controls_, controls_);
    matrix_q_ = Matrix::Zero(basic_state_size_, basic_state_size_);
    double qarray[] = {2.0, 2.0, 2.0, 2.0, 0.0, 0.0};
    for (int i = 0; i < 6; ++i) {
        matrix_q_(i, i) = qarray[i];
    }
}
// 计算横纵向偏差
void MPCController::ComputeLongitudinalLateralErrors(double t, VehicleState*
state, DesireTrajectory* traj)
{
    auto reference_point = traj->QueryPathPointAtTime(t);
    double xTarget = reference_point.x;
    double yTarget = reference_point.y;
    double dxTarget = state->xpos - xTarget;
    double dyTarget = state->ypos - yTarget;
    double KParam = 1.0;
    speed_error = reference_point.v - state->speed*cos(heading_error)/KParam;
    station_error   = -(dxTarget*cos(reference_point.phi) +
dyTarget*sin(reference_point.phi));
    heading_error      = state->phi - reference_point.phi;
    double dphiDes = reference_point.v*reference_point.Curvature;
    heading_error_rate = state->dphi - dphiDes;
    lateral_error      = dyTarget*cos(reference_point.phi) -
dxTarget*sin(reference_point.phi);
    lateral_error_rate = state->speed*sin(heading_error);

}
// 组合误差状态
void MPCController::UpdateStateAnalyticalMatching()
{
    matrix_state_(0, 0) = lateral_error;
    matrix_state_(1, 0) = lateral_error_rate;
    matrix_state_(2, 0) = heading_error;
    matrix_state_(3, 0) = heading_error_rate;
    matrix_state_(4, 0) = station_error;
    matrix_state_(5, 0) = speed_error;
}
// 更新ABC矩阵
```

```
void MPCController::UpdateMatrix(VehicleState* state) {
    double v = state->speed;
    matrix_a_(1, 1) = matrix_a_coeff_(1, 1) / v;
    matrix_a_(1, 3) = matrix_a_coeff_(1, 3) / v;
    matrix_a_(3, 1) = matrix_a_coeff_(3, 1) / v;
    matrix_a_(3, 3) = matrix_a_coeff_(3, 3) / v;

    Matrix matrix_i = Matrix::Identity(matrix_a_.cols(), matrix_a_.cols());
    matrix_ad_ = (matrix_i + ts_ * 0.5 * matrix_a_) *
                 (matrix_i - ts_ * 0.5 * matrix_a_).inverse();

    matrix_c_(1, 0) = (lr_ * cr_ - lf_ * cf_) / mass_ / v - v;
    matrix_c_(3, 0) = -(lf_ * lf_ * cf_ + lr_ * lr_ * cr_) / iz_ / v;
    matrix_cd_ = matrix_c_ * heading_error_rate * ts_;
}
// 求解MPC问题
void MPCController::ComputeControlCommand(double t, VehicleState* state,
DesireTrajectory* traj, ControlCommand *cmd)
{
    ComputeLongitudinalLateralErrors(t, state, traj);
    UpdateStateAnalyticalMatching();
    UpdateMatrix(state);
    Eigen::MatrixXd control_matrix(controls_, 1);
    control_matrix << 0, 0;

    Eigen::MatrixXd reference_state(basic_state_size_, 1);
    reference_state << 0, 0, 0, 0, 0, 0;

    std::vector<Eigen::MatrixXd> reference(horizon_, reference_state);

    Eigen::MatrixXd lower_bound(controls_, 1);
    lower_bound << -100, -10;

    Eigen::MatrixXd upper_bound(controls_, 1);
    upper_bound << 100, 10;

    std::vector<Eigen::MatrixXd> control(horizon_, control_matrix);

    if (SolveLinearMPC(
            matrix_ad_, matrix_bd_, matrix_cd_, matrix_q_,
            matrix_r_, lower_bound, upper_bound, matrix_state_, reference,
            mpc_eps_, mpc_max_iteration_, &control) != true) {
        AERROR << "MPC solver failed";
    } else {
        //AINFO << "MPC problem solved! ";
    }
    double steer_angle_feedback = control[0](0, 0);
    double steer_angle_feedforwardterm_updated_ = 0.0;
    double steer_angle =
            steer_angle_feedback + steer_angle_feedforwardterm_updated_;
    double acceleration_reference = 0.0;
    double acceleration_cmd = control[0](1, 0);
    cmd->steer = steer_angle;
    cmd->acc = acceleration_cmd;
}
```

运行结果:


Display1
Planning Path And Acturl Path
Acturl Path
Planning Path
4
3.5
3
2.5
2
1.5
1
0.5
0
0
20
40
60
80
100
2, 3.67