深蓝学院
shenlanxueyuan.com

运动规划第六章作业分享

主讲人 Travis

主要内容

1. Implement MPC of tracking the reference trajectory in C++;
2. Implement MPC with delays in C++;

MPC

深蓝学院
shenlanxueyuan.com

本章用的是自行车模型求解线性时变 MPC 问题。首先根据给定的运动模型构造矩阵 Ac，Bc，gc。

$$\begin{bmatrix} \dot{p}_x \\ \dot{p}_y \\ \dot{\phi} \\ \dot{v} \end{bmatrix} = \underbrace{\begin{pmatrix} 0 & 0 & -\bar{v}\sin\bar{\phi} & \cos\bar{\phi} \\ 0 & 0 & \bar{v}\cos\bar{\phi} & \sin\bar{\phi} \\ 0 & 0 & 0 & \frac{\tan\bar{\delta}}{L} \\ 0 & 0 & 0 & 0 \end{pmatrix}}_{A_c} \underbrace{\begin{bmatrix} p_x \\ p_y \\ \phi \\ v \end{bmatrix}}_{x} + \underbrace{\begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & \frac{\bar{v}}{L}\frac{1}{\cos^2\bar{\delta}} \\ 1 & 0 \end{pmatrix}}_{B_c} \underbrace{\begin{bmatrix} a \\ \delta \end{bmatrix}}_{u} + \underbrace{\begin{pmatrix} \bar{v}\bar{\phi}\sin\bar{\phi} \\ -\bar{v}\bar{\phi}\cos\bar{\phi} \\ -\frac{\bar{v}}{L}\frac{\bar{\delta}}{\cos^2\bar{\delta}} \\ 0 \end{pmatrix}}_{g_c}$$

然后根据 k+1 时刻的状态方程与 k 时刻的关系，可以得到所需的 Ad，Bd，gd。

$$x_{k+1} = (I + T_s A_c)x_k + T_s B_c u_k + T_s g_c$$

也即有以下关系：

$$A_d = I + T_s A_c \quad B_d = T_s B_c \quad g_d = T_s g_c$$

MPC

```
// x_{k+1} = Ad * x_{k} + Bd * u_k + gd
// TODO: set values to Ad_, Bd_, gd_
Ad_<<0,0,-v*sin(phi),cos(phi),
    0,0,v*cos(phi),sin(phi),
    0,0,0,tan(delta)/ll_,
    0,0,0,0;
Ad_ = MatrixA::Identity()+dt_*Ad_;

Bd_<<0,0,
    0,0,
    0,v/(ll_*pow(cos(delta),2)),
    1,0;
Bd_ = dt_*Bd_;

gd_<<v*phi*sin(phi),
    -v*phi*cos(phi),
    -v*delta/(ll_*pow(cos(delta),2)),
    0;
gd_ = dt_*gd_;

return;
}
```

MPC

深蓝学院 shenlanxueyuan.com

不等式约束 a delta delta，主要是对各个矩阵C的构造。

```
// set lower and upper boundaries
for (int i = 0; i < N_; ++i) {
  // TODO: set stage constraints of inputs (a, delta, ddelta)
  // -a_max <= a <= a_max for instance:
  Cu_.coeffRef(i * 3 + 0, i * m + 0) = 1;
  lu_.coeffRef(i * 3 + 0, 0) = -a_max_;
  uu_.coeffRef(i * 3 + 0, 0) = a_max_;
  // ...
  Cu_.coeffRef(i * 3 + 1, i * m + 1) = 1;
  lu_.coeffRef(i * 3 + 1, 0) = -delta_max_;
  uu_.coeffRef(i * 3 + 1, 0) = delta_max_;

  if(i>0){Cu_.coeffRef(i * 3 + 2, (i-1) * m + 1) = -1;}
  Cu_.coeffRef(i * 3 + 2, i * m + 1) = 1;
  lu_.coeffRef(i * 3 + 2, 0) = -dt_*ddelta_max_;
  uu_.coeffRef(i * 3 + 2, 0) = dt_*ddelta_max_;

  // TODO: set stage constraints of states (v)
  // -v_max <= v <= v_max
  Cx_.coeffRef(i,i*n+3)=1;
  // lx_.coeffRef(i,0)=-0.1;
  lx_.coeffRef(i,0)=-v_max_;
  ux_.coeffRef(i,0)=v_max_;
}
```

MPC

深蓝学院 shenlanxueyuan.com

设置完约束后，我们就可以考虑求解二次规划的问题。之前我们已经得到了状态方程，将其离散化后每个时刻构造成一个大的矩阵X，则可以得到新的关于矩阵X的状态方程。如下推导所示：

- Linear prediction model: $\begin{cases} \boldsymbol{x}_{k+1} = \boldsymbol{A}\boldsymbol{x}_k + \boldsymbol{B}\boldsymbol{u}_k \\ \boldsymbol{y}_k = \boldsymbol{C}\boldsymbol{x}_k \end{cases}$
- Relation between input and states: $\boldsymbol{x}_k = \boldsymbol{A}^k \boldsymbol{x}_0 + \sum_{j=0}^{k-1} \boldsymbol{A}^j \boldsymbol{B}\boldsymbol{u}_{k-1-j}$
- Forward simulation as matrix equation:

$$\begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{x}_{N-1} \\ \boldsymbol{x}_N \end{bmatrix} = \begin{bmatrix} \boldsymbol{B} & 0 & \cdots & 0 \\ \boldsymbol{A}\boldsymbol{B} & \boldsymbol{B} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{A}^{N-1}\boldsymbol{B} & \boldsymbol{A}^{N-2}\boldsymbol{B} & \cdots & \boldsymbol{B} \end{bmatrix} \begin{bmatrix} \boldsymbol{u}_0 \\ \boldsymbol{u}_1 \\ \vdots \\ \boldsymbol{u}_{N-1} \end{bmatrix} + \begin{bmatrix} \boldsymbol{A} \\ \boldsymbol{A}^2 \\ \vdots \\ \boldsymbol{A}^N \end{bmatrix} \boldsymbol{x}_0$$

MPC

值得注意的是，由于我们这里的系统是跟随时间变化的，因此A,B矩阵事实上都是随时间改变的，所以此处正确的形式应该是如下形式。
这里参考了此帖子：
https://blog.csdn.net/qq_42286607/article/details/124972866

- **Extended System State**:

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}$$

- **Extended** A_0:

$$A_0 = \begin{bmatrix} A_0 \\ A_1 A_0 \\ \vdots \\ \prod_{k=0}^{N-1} A_k \end{bmatrix}$$

- **Extended B**, which contains all the control matrices in discrete system equations, stacked:

$$B = \begin{bmatrix} B_0 & 0 & \dots & 0 & 0 \\ A_1 B_0 & B_1 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \prod_{k=1}^{N-1} A_k B_0 & \prod_{k=2}^{N-1} A_k B_1 & \dots & A_{N-1} B_{N-2} & B_{N-1} \end{bmatrix}$$

- **Extended G**, which contains all the constant terms in discrete system equations, stacked:

$$G = \begin{bmatrix} g_0 \\ A_1 g_0 + g_1 \\ \vdots \\ \sum_{n=0}^{N-2} (\prod_{k=n+1}^{N-1} A_k) g_n + g_{N-1} \end{bmatrix}$$

MPC

深蓝学院
shenlanxueyuan.com

```cpp
for (int i = 0; i < N_; ++i) {
  calLinPoint(s0, phi, v, delta);
  if (phi - last_phi > M_PI) {
    phi -= 2 * M_PI;
  } else if (phi - last_phi < -M_PI) {
    phi += 2 * M_PI;
  }
  last_phi = phi;
  linearization(phi, v, delta);

  if (i == 0) {
    BB.block(0, 0, n, m) = Bd_;
    AA.block(0, 0, n, n) = Ad_;
    gg.block(0, 0, n, 1) = gd_;
  } else {
    // TODO: set BB AA gg
    // ...
    BB.block(n*i,m*i,n,m)=Bd_;
    for(int j =i-1; j>=0; j--){
      BB.block(n*i,m*j,n,m)=Ad_*BB.block(n*(i-1),m*j,n,m);
    }
    AA.block(n*i,0,n,n)=Ad_*AA.block(n*(i-1),0,n,n);
    gg.block(n*i,0,n,1)=Ad_*gg.block(n*(i-1),0,n,1)+gd_;
  }
```

MPC

深蓝学院
shenlanxueyuan.com

由于最终我们是依靠 osqp 来求解二次凸优化问题，而其接口形式为

$$J = 0.5z^TQz + q^Tz$$

其中 Q 和 q 分别为 X 的二次项和一次项系数矩阵。

因此我们就需要将 cost function J 写成如下形式：

$$J = 0.5X^TQX + qX + const$$

我们首先求解 Q，由于我们的 cost function J 定义为：

$$J = (x - x_r)^2 + (y - y_r)^2 + \rho\,(\emptyset - \emptyset_r)^2$$

则我们可以把 J 的二次项表达为如下式的形式

$$J = (x - x_r)^TQ(x - x_r)$$

对比两式就可以得到 Q，且 Q 是一个稀疏矩阵。形式如下：

$$Q = \begin{pmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \rho & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} & & & \\ & \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \rho & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} & & \\ & & \ddots & \\ & & & \begin{bmatrix} \rho_N & 0 & 0 & 0 \\ 0 & \rho_N & 0 & 0 \\ 0 & 0 & \rho * \rho_N & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{pmatrix}$$

MPC

深蓝学院
shenlanxueyuan.com

我们之前已经得到了 X 的形式

$$\mathrm{X = BB * U + AA * x_0 + gg}$$

将其代入 J 计算，并将输入 U 写成 Z 就又可得到如下关于 J 的式子

$$\mathrm{J = 0.5Z^T BB^T QBBZ + (AAx_0 + gg - x_r)^T QBBZ + const}$$

则不难得到 q：

$$q^T = \mathrm{(AAx_0 + gg - x_r)^T QBB}$$

则有

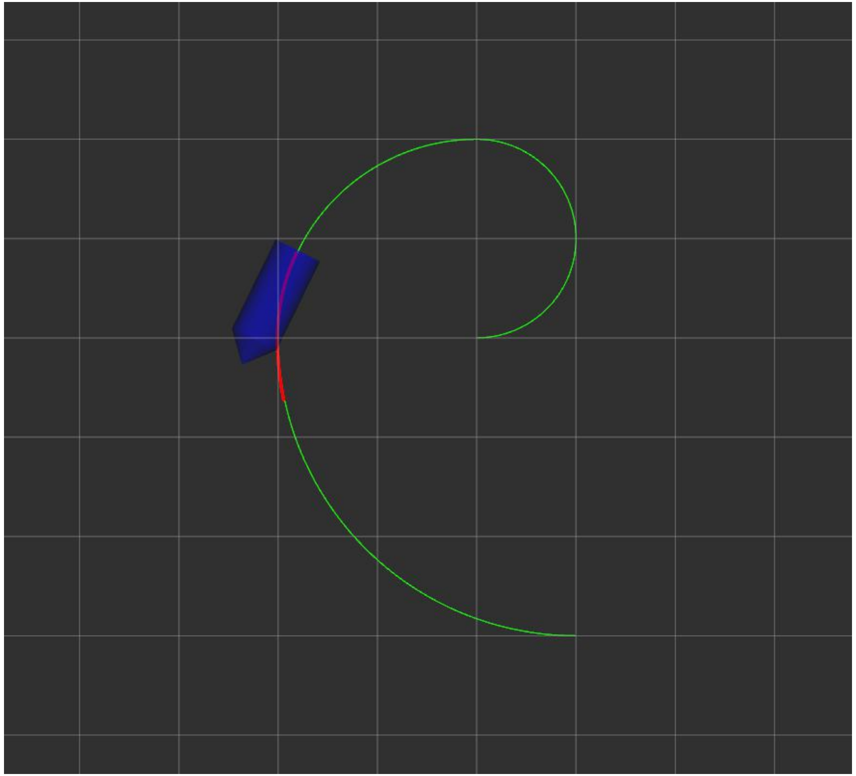
$$q = BB^T Q\mathrm{(AAx_0 + gg - x_r)}$$

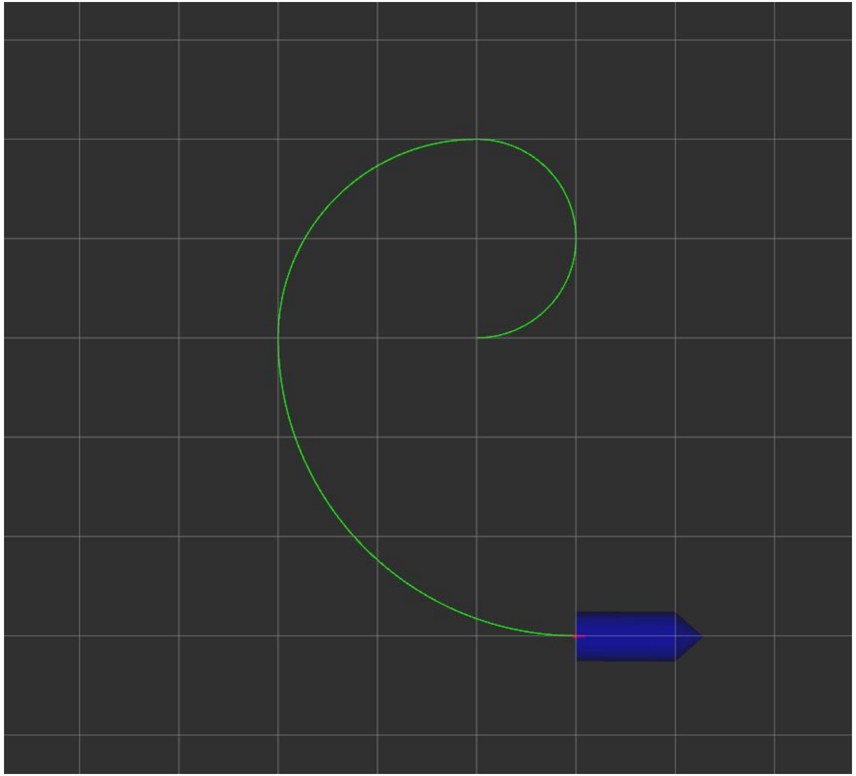
MPC

深蓝学院
shenlanxueyuan.com

```
// TODO: set qx
Eigen::Vector2d xy = s_(s0);  // reference (x_r, y_r)

// cost function should be represented as follows:
/* *
 *          /  x1  \T       /  x1  \         /  x1  \
 *          |  x2  |        |  x2  |         |  x2  |
 *  J =  0.5 |  x3  |   Qx_  |  x3  | + qx^T  |  x3  | + const.
 *          | ...  |        | ...  |         | ...  |
 *          \  xN  /        \  xN  /         \  xN  /
 * */

// qx.coeffRef(...
// ...
qx.coeffRef(n*i+0, 0) = -Qx_.coeffRef(n*i+0, n*i+0)*xy(0);
qx.coeffRef(n*i+1, 0) = -Qx_.coeffRef(n*i+1, n*i+1)*xy(1);
qx.coeffRef(n*i+2, 0) = -Qx_.coeffRef(n*i+2, n*i+2)*phi;
// qx.coeffRef(n*i+3, 0) = -Qx_.coeffRef(n*i+3, n*i+3)*v;
qx.coeffRef(n*i+3, 0) = 0;
```

MPC运行结果



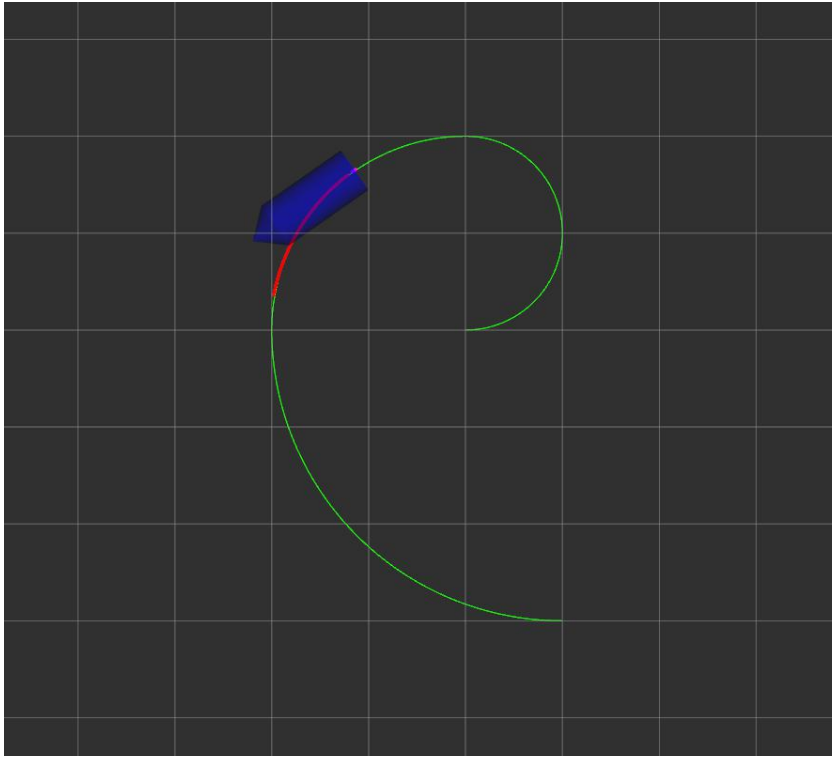


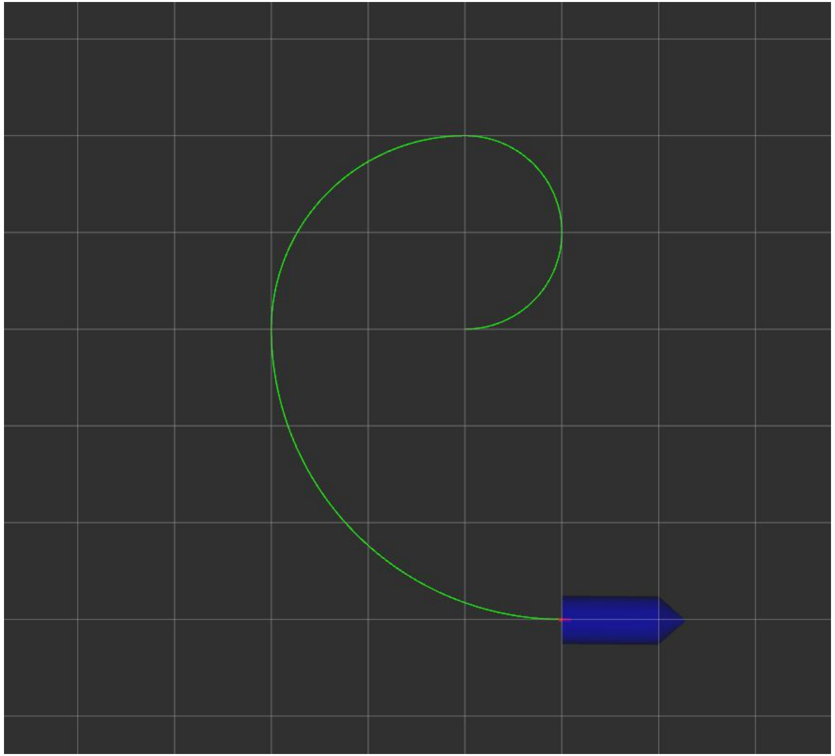
MPC with delays

深蓝学院
shenlanxueyuan.com

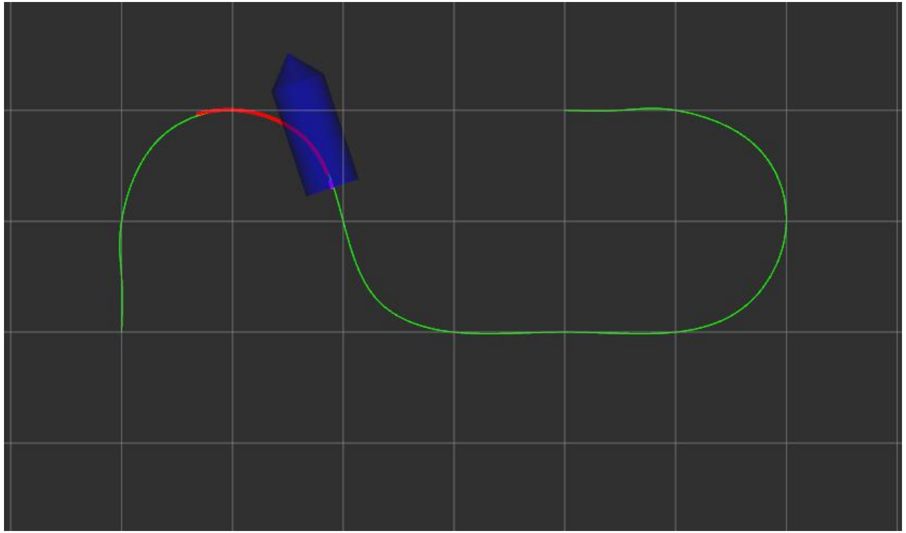
```
inline void step(VectorX& state, const VectorU& input, const double dt) const {
  // Runge–Kutta
  VectorX k1 = diff(state, input);
  VectorX k2 = diff(state + k1 * dt / 2, input);
  VectorX k3 = diff(state + k2 * dt / 2, input);
  VectorX k4 = diff(state + k3 * dt, input);
  state = state + (k1 + k2 * 2 + k3 * 2 + k4) * dt / 6;
}

VectorX compensateDelay(const VectorX& x0) {
  VectorX x0_delay = x0;
  // TODO: compensate delay
  // ...
  double dt = 1e-3;
  for (double t =delay_;t>0;t-=dt){
    int i =std::ceil(t/dt_);
    VectorU input = historyInput_[history_length_-i];
    step(x0_delay, input, dt);
  }

  return x0_delay;
}
```

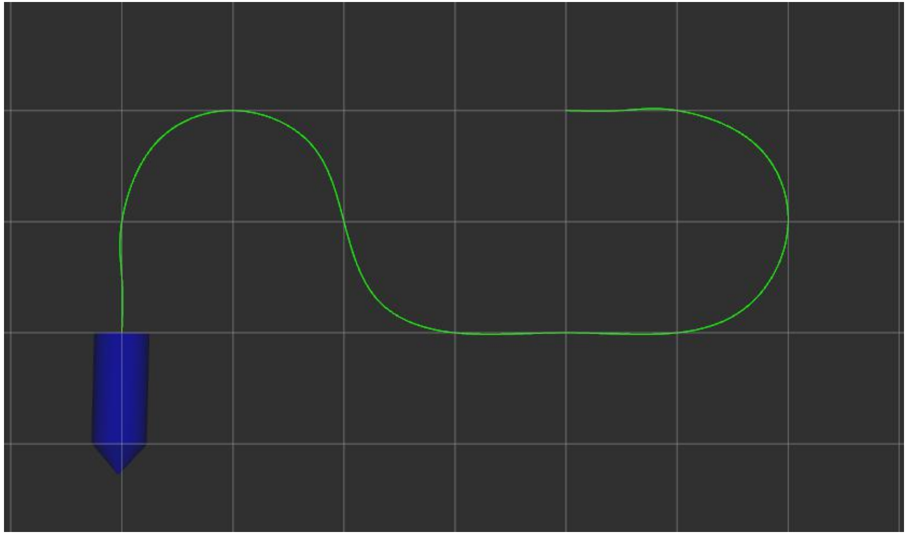
MPC with delays 运行结果



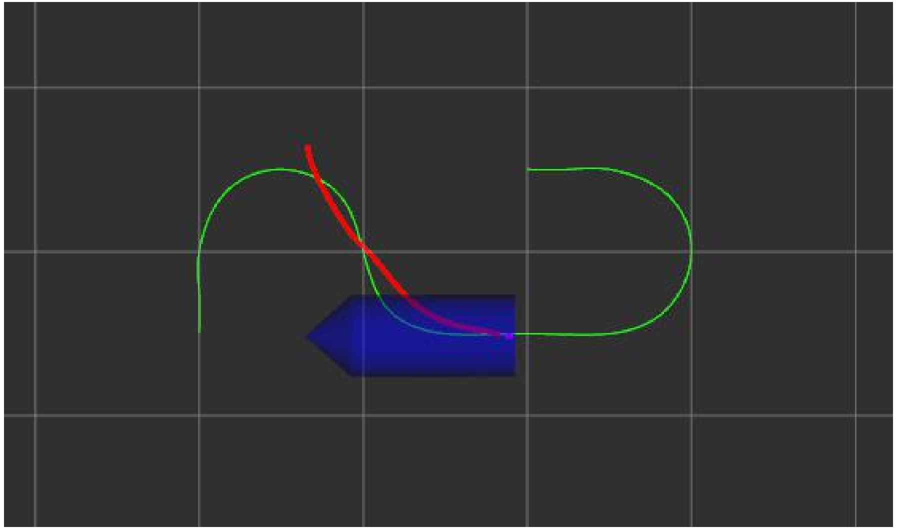


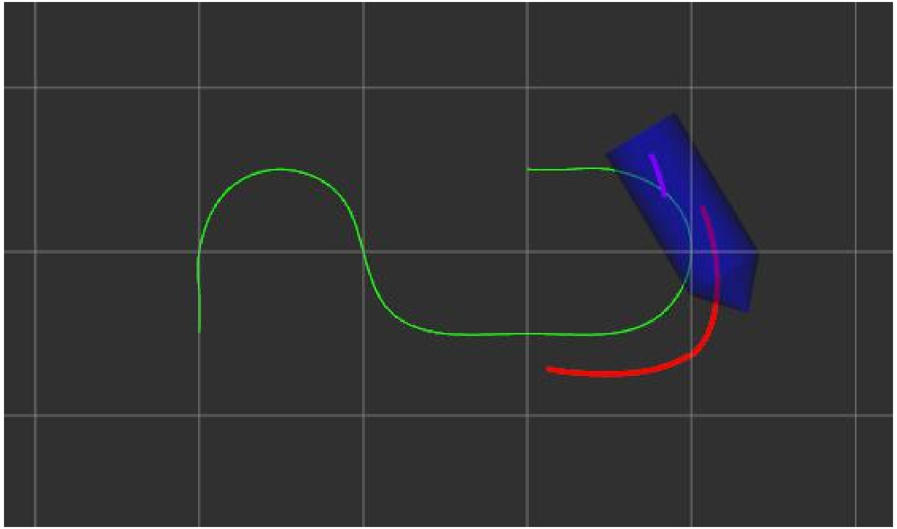
轨迹1



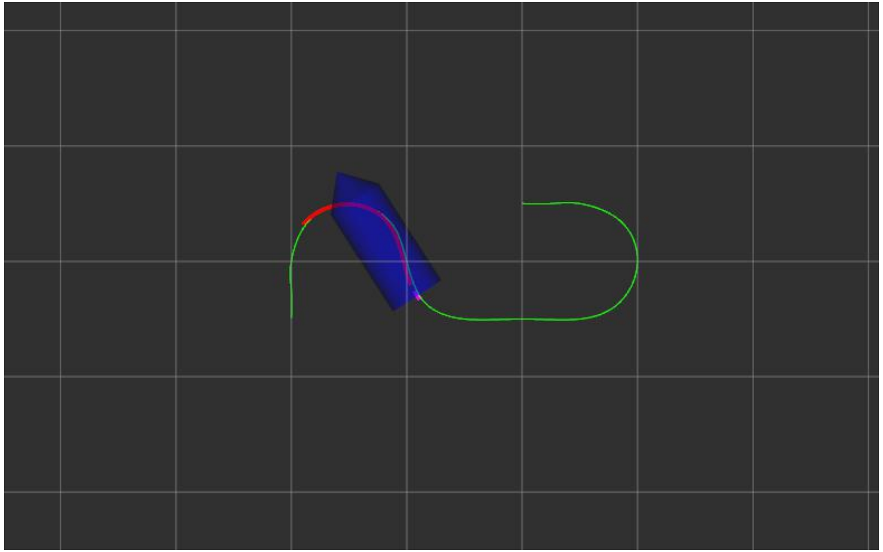


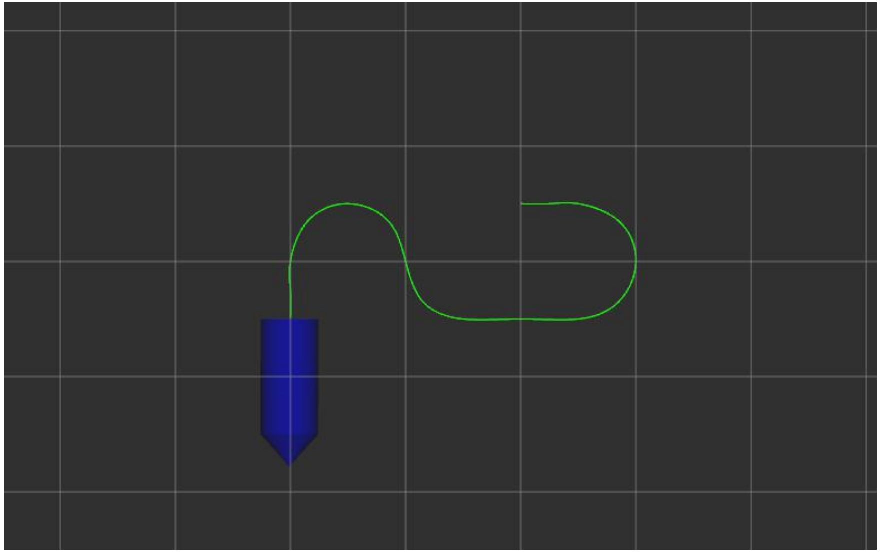
轨迹2 弯道半径过小





轨迹2 修改参数后





在线问答

Q&A

深蓝学院
shenlanxueyuan.com

感谢各位聆听！

Thanks for Listening