机器人路径规划

作业六 - 轨迹优化

张 文泰

学号：21009101463

MATLAB 作业部分

一、 程序分析

```
function Q = getQ(n_seg, n_order, ts)
    Q = [];
    fac = @(x) x*(x-1)*(x-2)*(x-3);
    for k = 1:n_seg
        Q_k = zeros(n_order+1,n_order+1);
        for i = 0:n_order
            for j = 0:n_order
                if (i < 4) || (j < 4)
                    continue;
                else
                    Q_k(i + 1,j + 1) = fac(i) * fac(j)/(i + j - 7) * ts(k) ^ (i + j -7);
                end
            end
        end
        Q = blkdiag(Q, Q_k);
    end
end
```

getQ 函数

函数 getQ 用于计算最小加速度规划的矩阵 Q，函数输入路径被分成的段数 n_seg、每个路径段上用于拟合轨迹的多项式的阶数 n_order、每个路径段的时间长度 ts。

函数中通过 fac 函数定义了一个简单的阶乘表达式，通过循环为每个路径段计算 Q_k 矩阵：对于每个路径段使用一个双重循环来填充 Q_k 矩阵。因为只有多项式的阶数大于等于 4 时，才能保证拟合的多项式具有至少 4 次连续可导性，所以在这个循环中，只有当 i 和 j 都大于或等于 4 时，才会计算 Q_k 的元素。在计算完每个 Q_k 矩阵后，使用 blkdiag 函数将它们放置在一个大的块对角矩阵 Q 中，并最终返回该矩阵。

```
function M = getM(n_seg, n_order, ts)
    coeff = [1,  1,  1,  1,  1,  1,  1,  1,;
             0,  1,  2,  3,  4,  5,  6,  7,;
             0,  0,  2,  6,  12, 20, 30, 42;
             0,  0,  0,  6,  24, 60, 120,210];
    M = [];
    for k = 1:n_seg
        M_k = zeros(8,8);
        t = ts(k);
        for i = 0:3
            M_k(i+1, i+1) = coeff(i+1, i+1);
        end
        for i = 0:3
            for j = i:n_order
                if (i == j)
                    M_k(i+4+1, j+1) = coeff(i+1, j+1) ;
                else
                    M_k(i+4+1, j+1) = coeff(i+1, j+1) * t^(j - i);
                end
            end
        end
        M = blkdiag(M, M_k);
    end
      disp(M)
end
```

getM 函数

getM 函数的输入参数与 getQ 函数相同，包括：路径被分成的段数 n_seg、每个路径段上用于拟合轨迹的多项式的阶数 n_order、每个路径段的时间长度 ts。函数通过 coeff 矩阵定义了一个系数矩阵，用于计算 M_k 矩阵的元素。然后，函数通过循环为每个路径段计算 M_k 矩阵：使用一个双重循环来填充 M_k 矩阵。在这个循环中，根据 coeff 矩阵的定义，M_k 的对角线上的元素直接复制，而非对角线上的元素根据给定的时间间隔 t 进行计算。

与上一个函数相同，在计算完每个 M_k 矩阵后，使用 blkdiag 函数将它们放置在一个大的块对角矩阵 M 中，并最终返回该矩阵。

```
function Ct = getCt(n_seg, n_order)
        row = 4 * 2 * n_seg;
    col = 4 * (n_seg + 1);
    Ct = zeros(row, col);
    n_fix = 4 + 4 + n_seg - 1;
    Ct(1:4, 1:4) = eye(4);
    idx_fix = 5;
    idx_free = n_fix + 1;
    for i = 1 : n_seg - 1
        start = 4 + 8 * (i - 1);
        for j = start + 1 : start + 4
            if mod(j, 4) == 1
                Ct(j, idx_fix) = 1;
                Ct(j + 4, idx_fix) = 1;
                idx_fix = idx_fix + 1;
            else
                Ct(j, idx_free) = 1;
                Ct(j + 4, idx_free) = 1;
                idx_free = idx_free + 1;
            end
        end
    end
    Ct(row - 3 : row , n_fix - 3 : n_fix) = eye(4);
end
```

getCT 函数

函数 getCt 用来生成约束矩阵 Ct，即用于描述轨迹端点（起点和终点）和轨迹段之间连接约束的矩阵。这个函数的输入参数包括路径被分成的段数 n_seg、每个路径段上用于拟合轨迹的多项式的阶数 n_order。函数中初始化 Ct 为一个零矩阵，其行数和列数根据给定的路径段数和阶数来确定。然后，函数通过循环为 Ct 矩阵的每个元素赋值。

1. 起始约束（Start Constraint）：前 4 行设置为单位矩阵，以确保起始点的位置和前三阶导数被固定。
2. 中间约束（Intermediate Constraints）：循环中的第二部分用于处理路径段之间的约束。它通过在每个路径段内部设置固定和自由变量来连接不同路径段之间的约束。
3. 终止约束（Terminal Constraint）：最后 4 行设置为单位矩阵，以确保终点的位置和前三阶导数被固定。

```
function [Aeq beq]= getAbeq(n_seg, n_order, waypoints, ts, start_cond, end_cond)
    n_all_poly = n_seg*(n_order+1);
    Aeq_start = zeros(4, n_all_poly);
    beq_start =  start_cond';
    Aeq_start(1:4,1:8) = getCoeff(0);
    Aeq_end = zeros(4, n_all_poly);
    beq_end =  end_cond';
    t = ts(end);
    Aeq_end(1:4, end-7:end) = getCoeff(t);
    Aeq_wp = zeros(n_seg-1, n_all_poly);
    beq_wp = zeros(n_seg-1, 1);
    for k = 0:1:n_seg-2
        beq_wp(k+1, 1) = waypoints(k+2);
        coeff = getCoeff(ts(k+1));
        Aeq_wp(k+1, 1+k*8:8+k*8) = coeff(1, :);
    end
    Aeq_con_p = zeros(n_seg-1, n_all_poly);
    beq_con_p = zeros(n_seg-1, 1);
    Aeq_con_v = zeros(n_seg-1, n_all_poly);
    beq_con_v = zeros(n_seg-1, 1);
    Aeq_con_a = zeros(n_seg-1, n_all_poly);
    beq_con_a = zeros(n_seg-1, 1);
    Aeq_con_j = zeros(n_seg-1, n_all_poly);
    beq_con_j = zeros(n_seg-1, 1);
    Aeq_con = [Aeq_con_p; Aeq_con_v; Aeq_con_a; Aeq_con_j];
    beq_con = [beq_con_p; beq_con_v; beq_con_a; beq_con_j];
    for k = 0:1:n_seg-2
            Aeq_con(1+4*k:4+4*k,1+8*k:8+8*k) = getCoeff(ts(k+1));
            Aeq_con(1+4*k:4+4*k,1+8*(k+1):8+8*(k+1)) = -getCoeff(0);
    end
    Aeq = [Aeq_start; Aeq_end; Aeq_wp; Aeq_con];
    beq = [beq_start; beq_end; beq_wp; beq_con];
end
```

getAbeq 函数

函数 getAbeq 用于生成等式约束矩阵 Aeq 和等式约束向量 beq 以描述最小加速度规划问题中的起始、终止条件、中间路点、以及路径段之间的连续性约束。函数主要步骤包括：

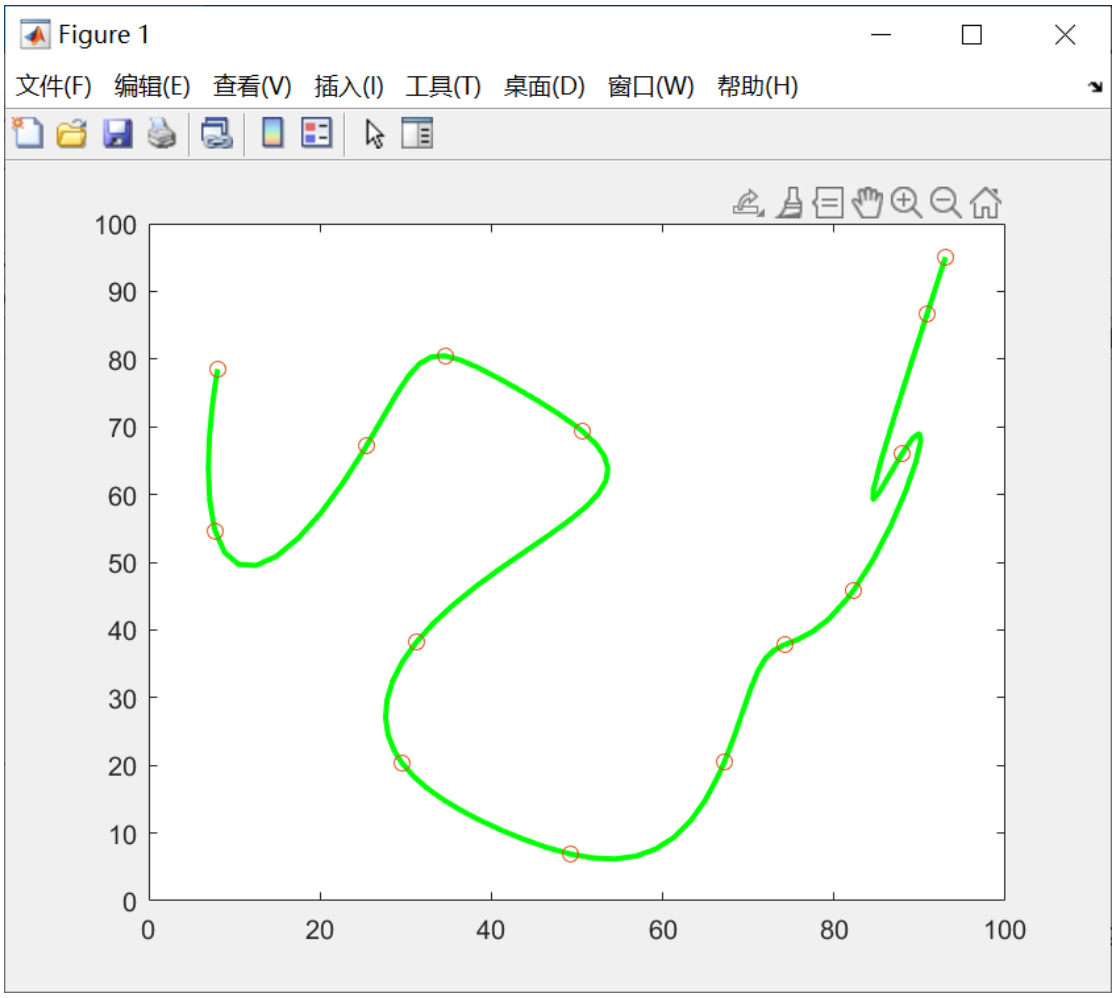
1. 起始条件约束：通过调用 getCoeff 函数得到起始条件所对应的多项式系数，并将其放置在 Aeq 和 beq 中。
2. 终止条件约束：同样地，通过调用 getCoeff 函数得到终止条件所对应的多项式系数，并将其放置在 Aeq 和 beq 中。
3. 中间路点约束：将中间路点的位置信息放置在 Aeq 和 beq 中。

4. 路径段连续性约束：对于每两个相邻的路径段，设置位置、速度、加速度和 jerk 的连续性约束。

5. 组合所有约束：将所有约束组合成 Aeq 和 beq。

二、 实验结果

运行 hw1_1.m 与 hw1_2.m，使用鼠标左键在图中选点，随机选取几个点后按回车，会出现规划好的路径，如图所示。



hw1_1.m 运行结果

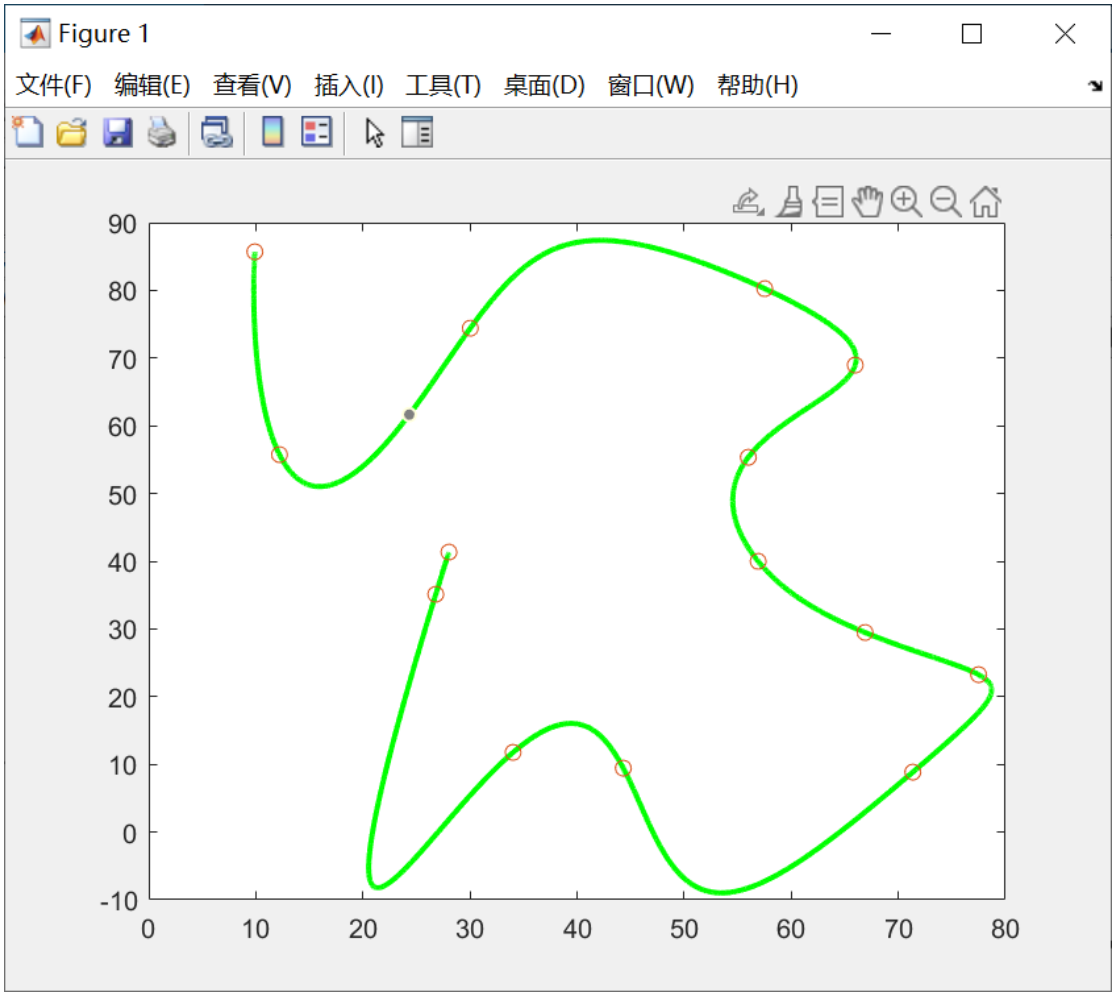


Figure 1
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
90
80
70
60
50
40
30
20
10
0
-10
0
10
20
30
40
50
60
70
80

hw1_2.m 运行结果

ROS 作业部分

一、 程序分析

```
void minimumJerkTrajGen(
    const int pieceNum,
    const Eigen::Vector3d &initialPos,
    const Eigen::Vector3d &initialVel,
    const Eigen::Vector3d &initialAcc,
    const Eigen::Vector3d &terminalPos,
    const Eigen::Vector3d &terminalVel,
    const Eigen::Vector3d &terminalAcc,
    const Eigen::Matrix3Xd &intermediatePositions,
    const Eigen::VectorXd &timeAllocationVector,
    Eigen::MatrixX3d &coefficientMatrix)
{
    Eigen::MatrixXd M = Eigen::MatrixXd::Zero(6 * pieceNum, 6 *pieceNum);
    Eigen::MatrixXd b = Eigen::MatrixXd::Zero(6 * pieceNum, 3);
    Eigen::MatrixXd F_0(3,6);
    F_0 << 1, 0, 0, 0, 0, 0,
           0, 1, 0, 0, 0, 0,
           0, 0, 2, 0, 0, 0;
    M.block(0, 0, 3, 6) = F_0;
    b.block(0, 0, 3, 3) << initialPos(0), initialPos(1), initialPos(2),
                           initialVel(0), initialVel(1), initialVel(2),
                           initialAcc(0), initialAcc(1), initialAcc(2);
    for (int i=1; i < pieceNum ; i++)
    {
        double ti = timeAllocationVector(i-1);
        Eigen::MatrixXd F_i(6,6), E_i(6,6);
        Eigen::Vector3d D_i = intermediatePositions.transpose().row(i - 1);
        E_i <<1, ti, pow(ti,2), pow(ti,3), pow(ti,4), pow(ti,5),
              1, ti, pow(ti,2), pow(ti,3), pow(ti,4), pow(ti,5),
              0, 1, 2*ti, 3*pow(ti,2), 4*pow(ti,3), 5*pow(ti,4),
              0, 0, 2, 6*ti, 12*pow(ti,2), 20*pow(ti,3),
              0, 0, 0, 6, 24*ti, 60*pow(ti,2),
              0, 0, 0, 0, 24, 120*ti;
        F_i <<0, 0, 0, 0, 0, 0,
             -1, 0, 0, 0, 0, 0,
              0, -1, 0, 0, 0, 0,
              0, 0, -2, 0, 0, 0,
              0, 0, 0, -6, 0, 0,
              0, 0, 0, 0, -24, 0;
        M.block(3 + 6 * (i - 1), 6 * (i - 1), 6, 6) =  E_i;
        M.block(3 + 6 * (i - 1), 6 + 6 * (i - 1), 6, 6) = F_i;
        b.block(3 + 6 * (i - 1), 0, 6, 3)  <<  D_i(0), D_i(1), D_i(2),
                                               0, 0, 0,
                                               0, 0, 0,
                                               0, 0, 0,
                                               0, 0, 0,
                                               0, 0, 0;
    }
    double tm = timeAllocationVector(pieceNum-1);
    Eigen::MatrixXd E_M(3,6);
    E_M << 1, tm, pow(tm,2),   pow(tm,3),    pow(tm,4),   pow(tm,5),
           0,  1,      2*tm, 3*pow(tm,2),  4*pow(tm,3),  5*pow(tm,4),
           0,  0,         2,        6*tm, 12*pow(tm,2), 20*pow(tm,3);
    M.block(pieceNum * 6 -3, pieceNum * 6 - 6, 3, 6) << E_M;
    b.block(pieceNum * 6 -3, 0, 3, 3) << terminalPos(0), terminalPos(1), terminalPos(2),
                                         terminalVel(0), terminalVel(1), terminalVel(2),
                                         terminalAcc(0), terminalAcc(1), terminalAcc(2);
    coefficientMatrix = M.inverse() * b;
}
```

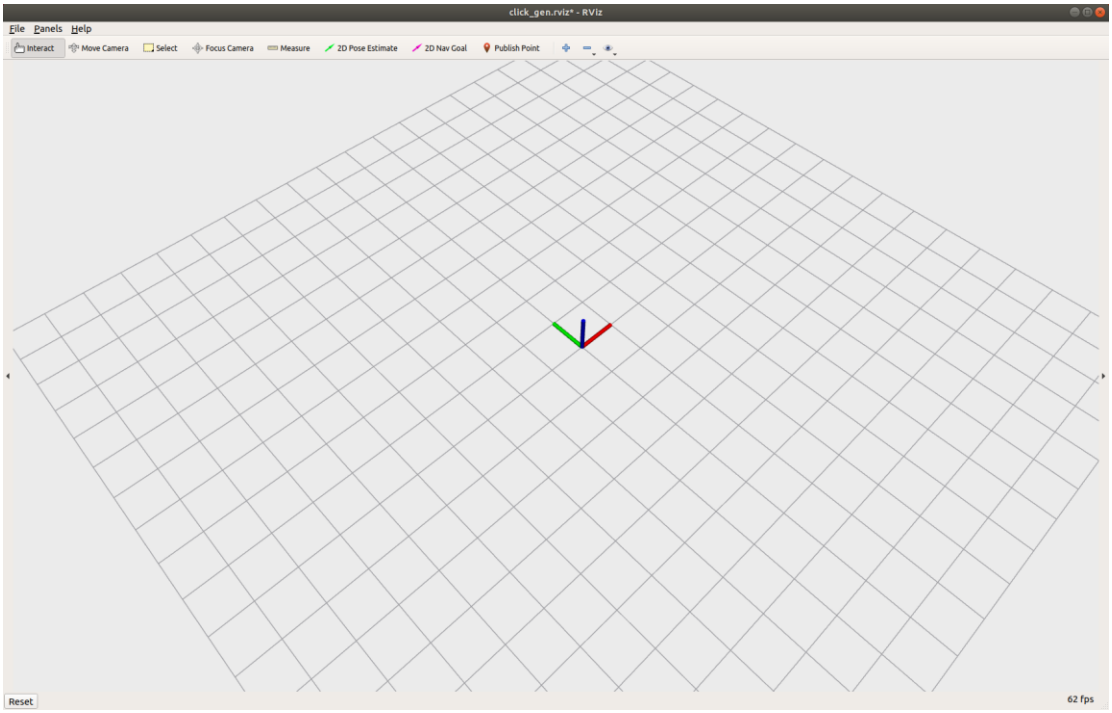
minimumJerkTrajGen 函数

这段代码的主要思路是利用多项式的最小加速度（minimum jerk）轨迹生成算法来计算给定初始和终点状态下的光滑轨迹。代码思路如下：

1. 初始化矩阵和向量：首先，初始化了两个矩阵 M 和 b，它们分别用于构建系数矩阵和常数矩阵。
2. 处理初始状态：将初始位置、速度和加速度放入常数矩阵 b 的前三行。
3. 循环处理每个时间段：对于每个时间段，生成了一个$6x6$的矩阵 E_i 和一个$6x6$的矩阵 F_i，它们分别代表了当前时间段内位置、速度和加速度的关系，然后将这些矩阵填入系数矩阵 M 的相应位置。
4. 处理终点状态：将终点的位置、速度和加速度放入常数矩阵 b 的最后三行。
5. 求解系数矩阵：通过求解线性方程组 $M * coefficientMatrix = b$，得到了系数矩阵 $coefficientMatrix$。

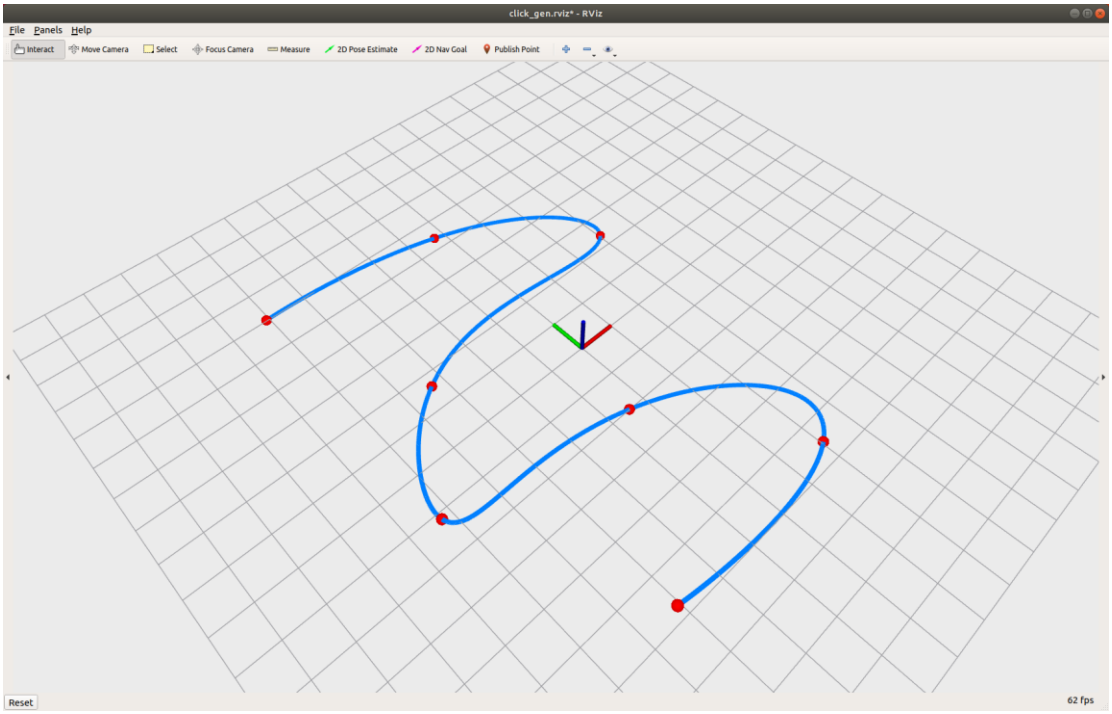
简单来讲就是，这段代码通过循环处理每个时间段，构建了一个线性方程组，然后通过求解该方程组得到了最小加速度轨迹的系数矩阵。

二、 实验结果



RViz 运行成功

如图所示，程序成功运行，地图显示在 RViz 中。接下来使用 2D nav goal 工具在图中选取出发点并依次选取路径点，可以看到在图中显示出了平滑的轨迹，运行结果如图所示：



轨迹运行结果

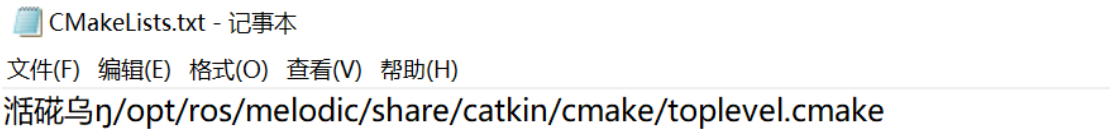
三、 实验中遇到的问题及解决方案

```
ros@ros-virtual-machine:~/catkin_ws_hw4/src$ cd ..
ros@ros-virtual-machine:~/catkin_ws_hw4$ catkin_make
Base path: /home/ros/catkin_ws_hw4
Source space: /home/ros/catkin_ws_hw4/src
Build space: /home/ros/catkin_ws_hw4/build
Devel space: /home/ros/catkin_ws_hw4/devel
Install space: /home/ros/catkin_ws_hw4/install
####
#### Running command: "cmake /home/ros/catkin_ws_hw4/src -DCATKIN_DEVEL_PREFIX=
/home/ros/catkin_ws_hw4/devel -DCMAKE_INSTALL_PREFIX=/home/ros/catkin_ws_hw4/in
stall -G Unix Makefiles" in "/home/ros/catkin_ws_hw4/build"
####
CMake Error at CMakeLists.txt:1:
  Parse error.  Expected a command name, got unquoted argument with text
  "IntxLNK[illegible]".


-- Configuring incomplete, errors occurred!
Invoking "cmake" failed
ros@ros-virtual-machine:~/catkin_ws_hw4$
```

报错现象

如图所示，我在进行工作空间编译时遇到了报错，根据报错内容可以得知是 CMakeLists.txt 存在问题，于是我去查看该文件内容，如图所示：



CMakeLists.txt 内容

可以发现该文件出现编码问题，开头并非使用 UTF-8 编码导致 Ubuntu 无法识别，在修正之后依然出现报错情况，于是我新建工作空间，在将作业包中的内容导入 Ubuntu 工作空间时，没有导入 CMakeLists.txt，而是使用工作空间初始化后自动生成的 CMakeLists.txt，编译之后程序成功运行，问题得到解决。