机器人路径规划

第四次作业

张 文泰

学号：21009101463

一、代码补充与分析

```
for(int step=0 ; step<=_time_step ; step ++){
    pos(0) = pos(0) + vel(0)*delta_time + 0.5*acc_input(0)*delta_time*delta_time;
    pos(1) = pos(1) + vel(1)*delta_time + 0.5*acc_input(1)*delta_time*delta_time;
    pos(2) = pos(2) + vel(2)*delta_time + 0.5*acc_input(2)*delta_time*delta_time;

    vel(0) = vel(0) + acc_input(0)*delta_time;
    vel(1) = vel(1) + acc_input(1)*delta_time;
    vel(2) = vel(2) + acc_input(2)*delta_time;

    Position.push_back(pos);
    Velocity.push_back(vel);
    double coord_x = pos(0);
    double coord_y = pos(1);
    double coord_z = pos(2);
    //check if if the trajectory face the obstacle
    if(_homework_tool->isObsFree(coord_x,coord_y,coord_z) != 1){
                        collision = true;
    }
}
```

STEP1 补充代码

其中作业要求补充的代码部分是欧拉积分法的实现，用于在给定的时间步长内更新位置和速度，其逻辑如下：

1. pos(0)这一行代码更新了位置向量 pos 的第一个分量（x 分量）。根据欧拉积分法，位置的更新公式为：

新位置=旧位置+速度*时间步长+1/2*加速度*时间步长^2

2. pos(1)、pos(2)两行使用了同样的逻辑，用于位置向量的第二、三个分量（y、z 分量）的更新。

3. vel(0)这行代码更新了速度向量 vel 的第一个分量（x 分量）。根据欧拉积分法，速度的更新公式为：

新速度=旧速度+加速度*时间步长

4. vel(1)、vel(2)两行使用了同样的逻辑，用于速度向量的第二、三个分量（y、z 分量）的更新。

```cpp
double Homeworktool::OptimalBVP(Eigen::Vector3d _start_position,Eigen::Vector3d
_start_velocity,Eigen::Vector3d _target_position)
{
    double optimal_cost = 100000;
    double pxf = _target_position(0);
    double pyf = _target_position(1);
    double pzf = _target_position(2);
    double px0 = _start_position(0);
    double py0 = _start_position(1);
    double pz0 = _start_position(2);
    double vx0 = _start_velocity(0);
    double vy0 = _start_velocity(1);
    double vz0 = _start_velocity(2);
    double vxf = 0,vyf = 0, vzf = 0;
    Eigen::Matrix<double,4,4> m;
    double c0 = -36*( (pxf-px0)*(pxf-px0) + (pyf-py0)*(pyf-py0) + (pzf-pz0)*(pzf-pz0)) ;
    double c1 = 24*((pxf-px0)*(vxf+vx0) + (pyf-py0)*(vyf+vy0) + (pzf-pz0)*(vzf+vz0));
    double c2 = -4*(vx0*vx0 + vx0*vxf + vxf*vxf + vy0*vy0 + vy0*vyf + vyf*vyf + vz0*vz0 + vz0*vzf + vzf*vzf);
    double c3 = 0.0;
    m << 0,0,0,-c0,
         1,0,0,-c1,
         0,1,0,-c2,
         0,0,1,c3;
    double J;
    Eigen::Matrix<complex<double>,Eigen::Dynamic,Eigen::Dynamic> eigenValules;
    eigenValules = m.eigenvalues();
    for(int i=0; i<4; ++i){
        double T = std::real(eigenValules(i));
        double img = std::imag(eigenValules(i));
        if(T <= 0 || std::abs(img) >= 1e-16){
            continue;
        }
        Eigen::Vector3d alpha,beta;
        alpha(0) = 12*(px0-pxf+T*vx0)/T/T/T - 6*(vx0-vxf)/T/T;
        alpha(1) = 12*(py0-pyf+T*vy0)/T/T/T - 6*(vy0-vyf)/T/T;
        alpha(2) = 12*(pz0-pzf+T*vz0)/T/T/T - 6*(vz0-vzf)/T/T;
        beta(0) = 2*(vx0-vxf)/T -6*(px0-pxf+T*vx0)/T/T;
        beta(1) = 2*(vy0-vyf)/T -6*(py0-pyf+T*vy0)/T/T;
        beta(2) = 2*(vz0-vzf)/T -6*(pz0-pzf+T*vz0)/T/T;
        J = T + 1.0/3.0*alpha.dot(alpha)*std::pow(T,3) + alpha.dot(beta)*std::pow(T,2) + beta.dot(beta)*T;
        if(J < optimal_cost){
            optimal_cost = J;
        }
    }
    return optimal_cost;
}
```

STEP2 补充代码

作业 STEP2 需要补充的代码是用于计算 OBVP 的，在函数中给定起始点的位置和速度以及目标点的位置，补充代码主要用于计算出最小的代价，其大致逻辑如下：

1.从传入的参数_start_position、_start_velocity 和_target_position 中提取出各个坐标分量的数值。

2.定义一个 4x4 的矩阵 m 用来求解伴随矩阵的特征值，用于确定最小代价对应的时间 T。

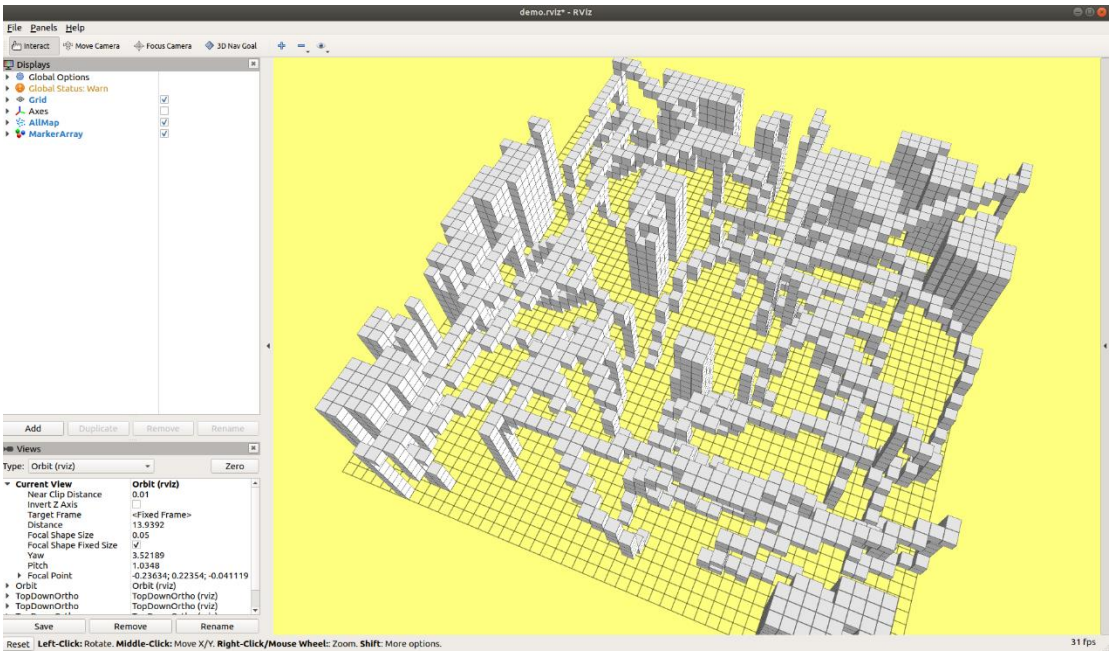
3.通过 m.eigenvalues()计算特征值，并进行循环遍历。

4.在循环中检查特征值是否为负数或者虚数，否则计算对应的时间 T，然后根据公式计算出 alpha 和 beta，并根据这些值计算出代价 J。

5. 如果 J 小于当前的 optimal_cost，则更新 optimal_cost。

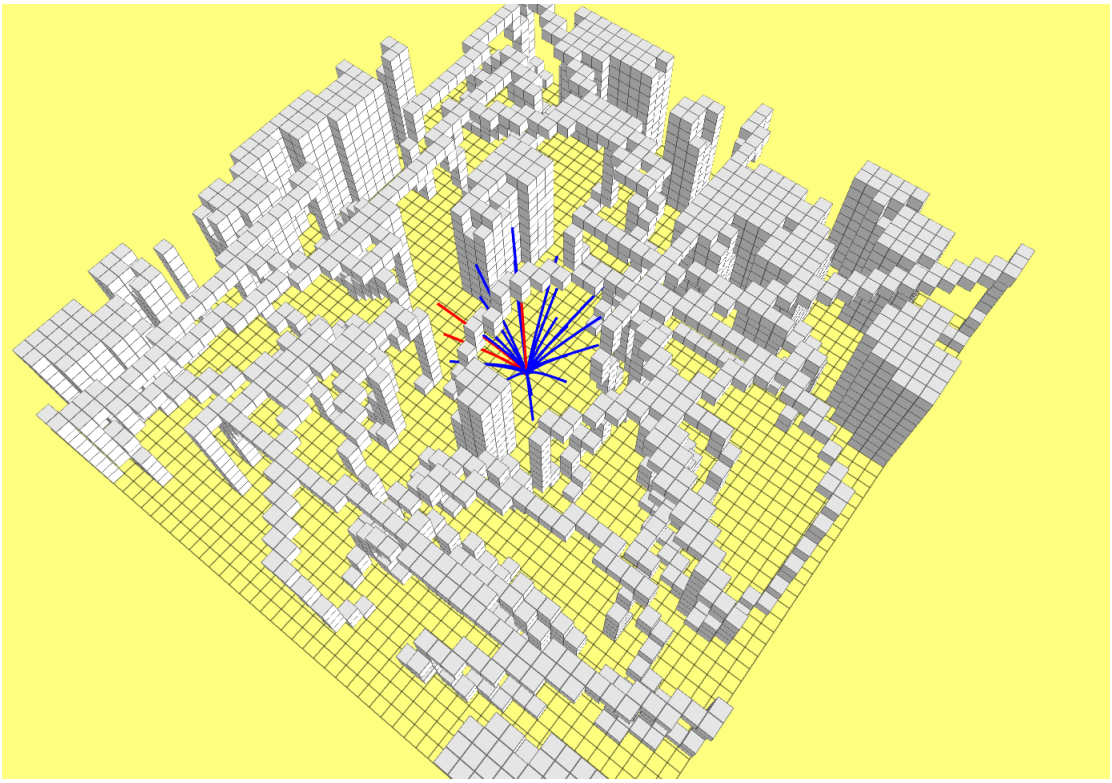
6. 最后返回 optimal_cost。

二、实验过程



程序运行成功

三、实验结果



程序运行结果