深蓝学院
shenlanxueyuan.com

第四章作业思路分享

主讲人 JT

纲要

- 第一部分：算法
- 第二部分：仿真实现
- 第三部分：Q&A

算法

- Build an ego-graph of the linear modeled robot

$$x = x_0 + \int v dt$$

$$v = v_0 + \int a dt$$

$$x = x_0 + v_0 t + \frac{1}{2} a t^2$$

算法

- Select the best trajectory closest to the planning target
  - 参数求解公式：

$$\begin{pmatrix}\alpha_1\\ \alpha_2\\ \alpha_3\\ \beta_4\\ \beta_5\\ \beta_6\end{pmatrix}=\begin{pmatrix}\frac{12}{T^3} & 0 & 0 & \frac{6}{T^2} & 0 & 0\\ 0 & \frac{12}{T^3} & 0 & 0 & \frac{6}{T^2} & 0\\ 0 & 0 & \frac{12}{T^3} & 0 & 0 & \frac{6}{T^2}\\ \frac{6}{T^2} & 0 & 0 & -\frac{2}{T} & 0 & 0\\ 0 & \frac{6}{T^2} & 0 & 0 & -\frac{2}{T} & 0\\ 0 & 0 & \frac{6}{T^2} & 0 & 0 & -\frac{2}{T}\end{pmatrix}\begin{pmatrix}\Delta p_x\\ \Delta p_y\\ \Delta p_z\\ \Delta v_x\\ \Delta v_y\\ \Delta v_z\end{pmatrix}$$

  - 代价函数：

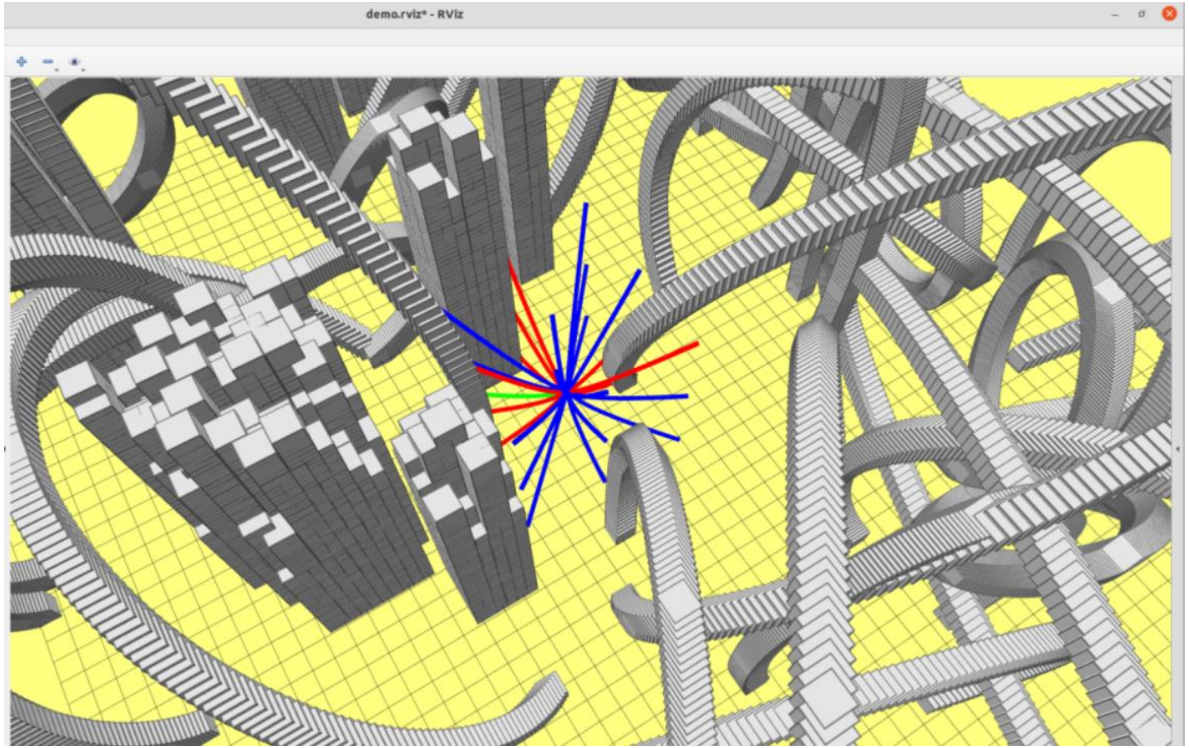
$$J=T+\left(\frac{1}{3}\alpha_1^2T^3+\alpha_1\beta_1T^2+\beta_1^2T\right)+\left(\frac{1}{3}\alpha_2^3T^3+\alpha_2\beta_2T^2+\beta_2^2T\right)+\left(\frac{1}{3}\alpha_3^3T^3+\alpha_3\beta_3T^2+\beta_3^2T\right)$$

纲要

- 第一部分：算法
- 第二部分：仿真实现
- 第三部分：Q&A

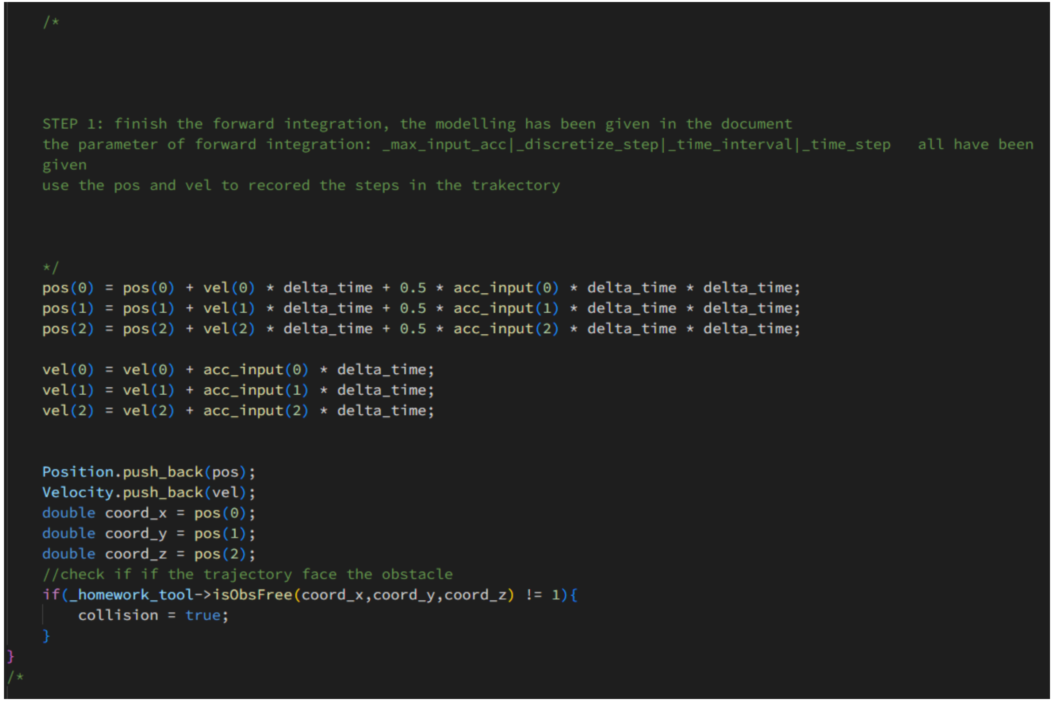
仿真实现

- 运行结果



仿真实现

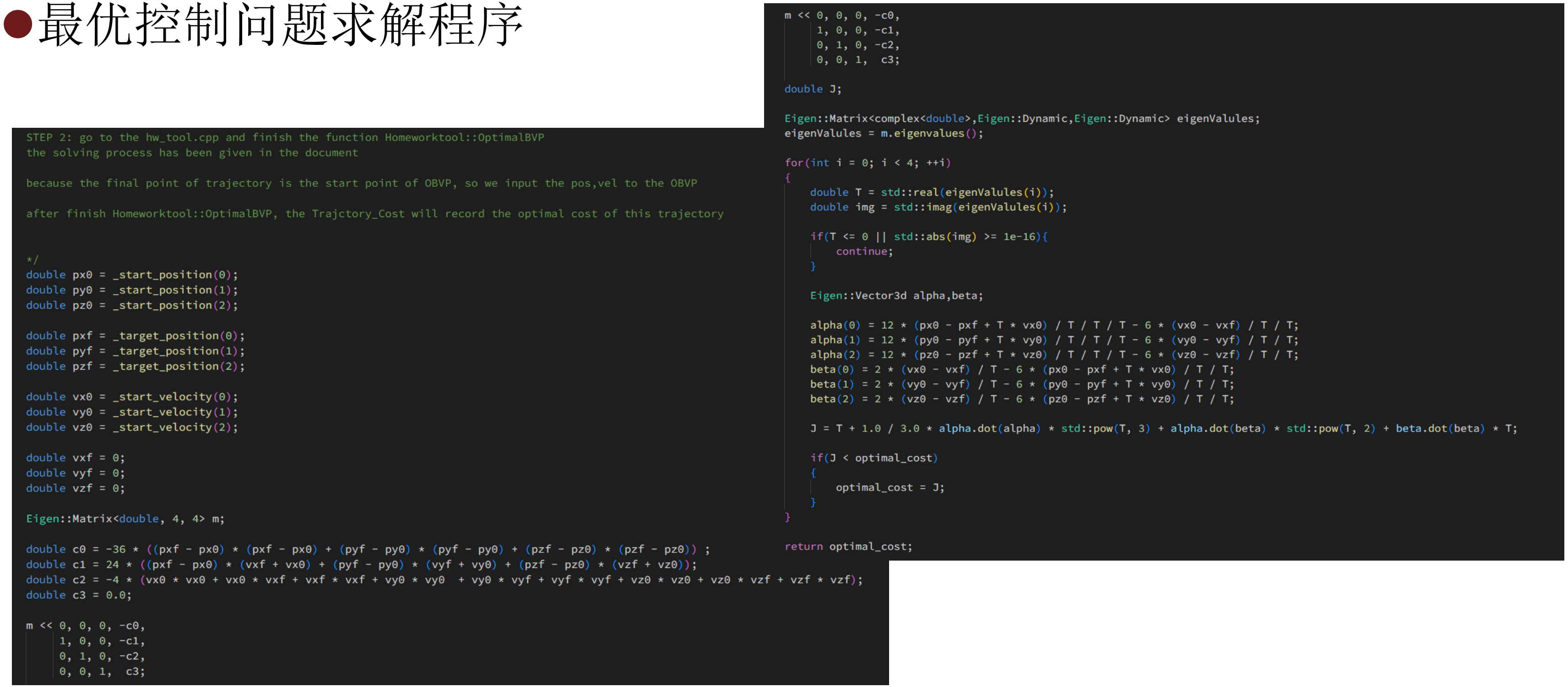
- 机器人简单动力学程序



```
    /*




    STEP 1: finish the forward integration, the modelling has been given in the document
    the parameter of forward integration: _max_input_acc|_discretize_step|_time_interval|_time_step   all have been
    given
    use the pos and vel to recored the steps in the trakectory



    */
    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;
    }
}
/*
```

仿真实现

- 最优控制问题求解程序



```
STEP 2: go to the hw_tool.cpp and finish the function Homeworktool::OptimalBVP
the solving process has been given in the document

because the final point of trajectory is the start point of OBVP, so we input the pos,vel to the OBVP

after finish Homeworktool::OptimalBVP, the Trajctory_Cost will record the optimal cost of this trajectory


*/
double px0 = _start_position(0);
double py0 = _start_position(1);
double pz0 = _start_position(2);

double pxf = _target_position(0);
double pyf = _target_position(1);
double pzf = _target_position(2);

double vx0 = _start_velocity(0);
double vy0 = _start_velocity(1);
double vz0 = _start_velocity(2);

double vxf = 0;
double vyf = 0;
double 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;
```

```
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;
```

作业要求

- 作业内容：ROS下完成仿真演示。
- 评价标准：
  - 及格：补全机器人动力学部分代码；
  - 良好：补全机器人动力学部分代码和最优控制求解部分代码；
  - 优秀：在良好的基础上，撰写说明文档，对本次课程内容的公式进行推导，对作业的运行结果进行展示，也可以记录自己的心得。

在线问答

Q&A

深蓝学院
shenlanxueyuan.com

感谢各位聆听！

Thanks for Listening