ROS 作业：

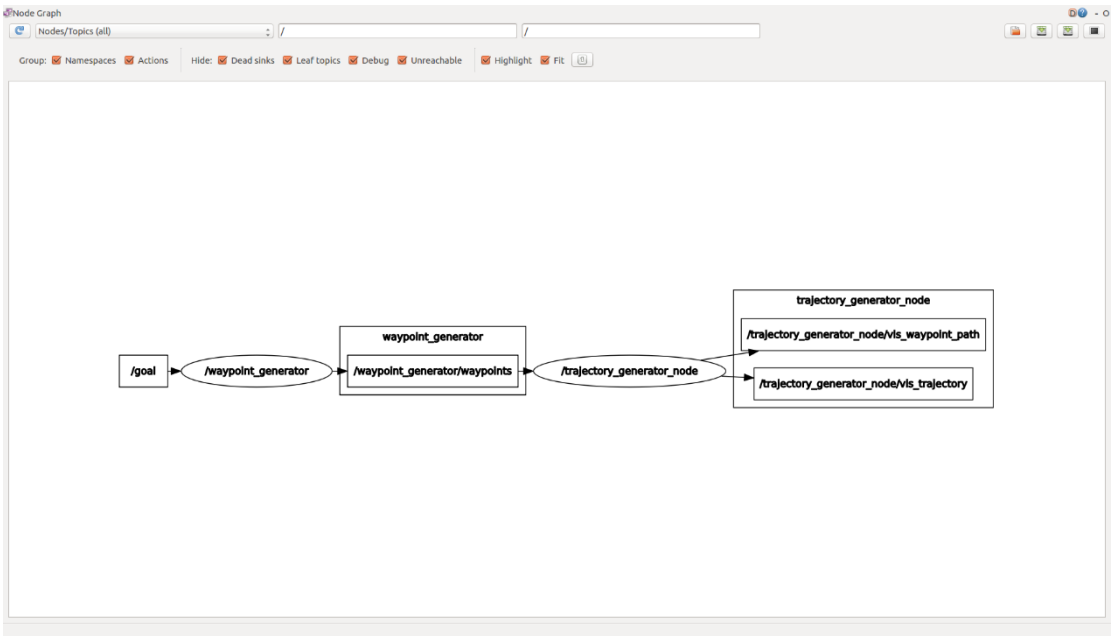
Homework 2.1: In C++/ROS, use the OOQP solver, write down a minimum snap trajectory generator

Homework 2.2: In C++/ROS, use Eigen, generate minimum snap trajectory based on the closed form solution

Hw_5_ros 作业包仅提供了 Homework 2.2 的代码框架，如果想要使用 OOQP solver 进行问题求解，请自行按照 Homework2.2 的代码框架修改即可。

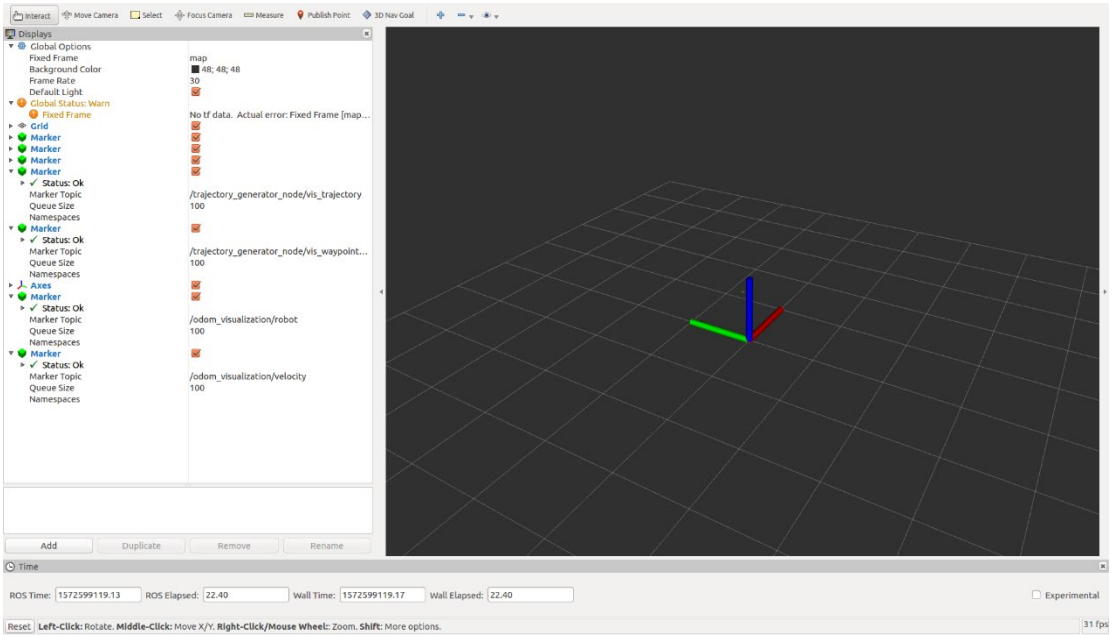
提醒：本次作业 ros 部分难度极大，建议课程学习比较吃力的同学选择做 matlab 部分的作业，因为此次 c++编程的难度主要集中在大规模矩阵的使用上，使用 matlab 完成作业难度会下降很多。建议首先选择 matlab 作业，把课程知识通过编程巩固，之后学有余力再选择 ros 作业作为自己的锻炼。

0. 编译
   步骤与之类教程包类似。
1. 功能包关系



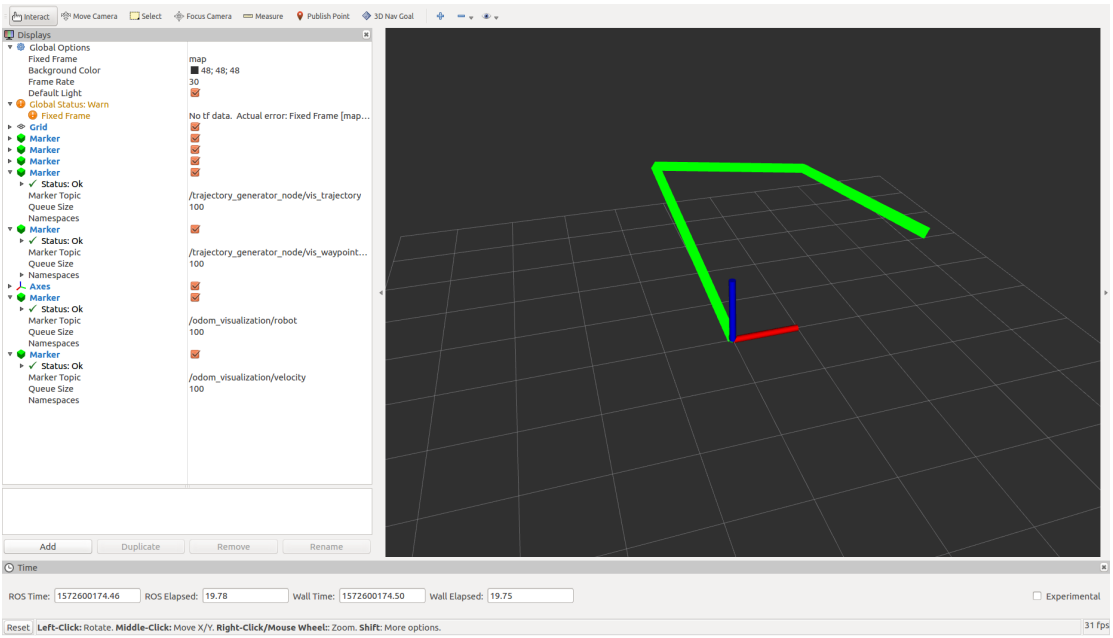
2. 运行

```
source devel/setup.bash
roslaunch waypoint_trajectory_generator test.launch
```



使用方法：

每按一次快捷键“G”都可以重复添加 waypoint，waypoint 选择方法和之前一致：鼠标点击确认 x/y 坐标，按住左键旋转确认 yaw 角，同时按住右键上下拖动确认 z 坐标。多次给定 waypoint 后即出现待优化的轨迹 path。（结束 path 给定，开始 trajectory 生成的标志是给定一个 z 轴小于 0 的 waypoint）



3. 作业部分

首先完成功能包 waypoint_trajectory_generator 下 trajectory_generator_node.cpp 里的 VectorXd timeAllocation(MatrixXd Path)

```cpp
VectorXd timeAllocation( MatrixXd Path)
{
    VectorXd time(Path.rows() - 1);

    /*

    STEP 1: Learn the "trapezoidal velocity" of "TIme Allocation" in L5, then finish this timeAllocation function

    variable declaration: _Vel, _Acc: _Vel = 1.0, _Acc = 1.0 in this homework, you can change these in the test.launch

    You need to return a variable "time" contains time allocation, which's type is VectorXd

    The time allocation is many relative timeline but not one common timeline

    */

    return time;
```

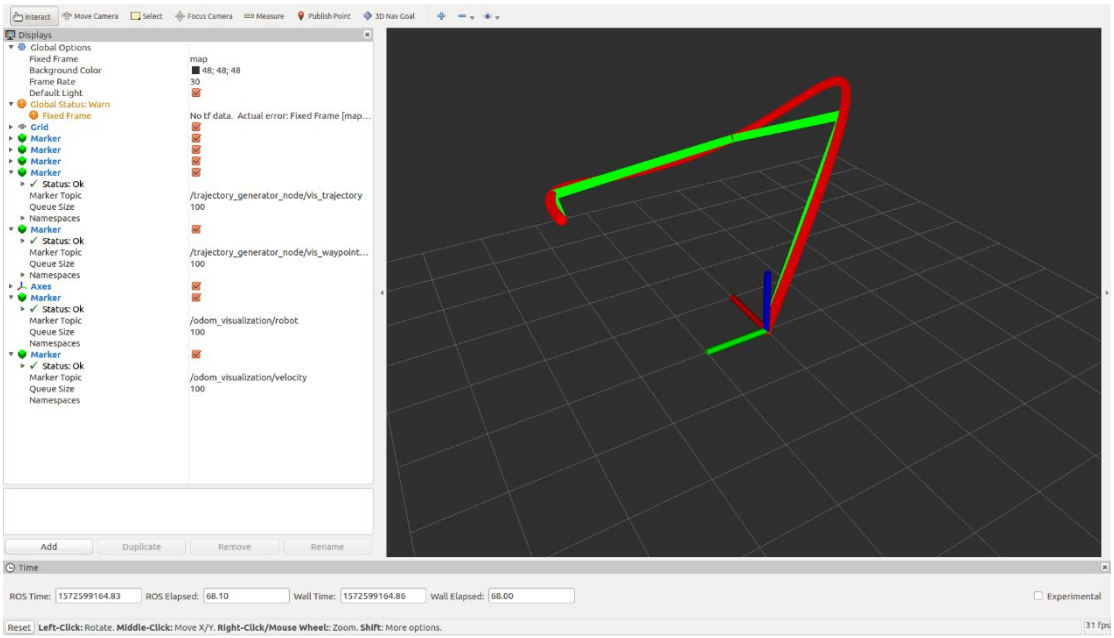
之后完成功能包 waypoint_trajectory_generator 下 trajectory_generator_waypoint.cpp 里的 Eigen::MatrixXd TrajectoryGeneratorWaypoint::PolyQPGeneration(…)

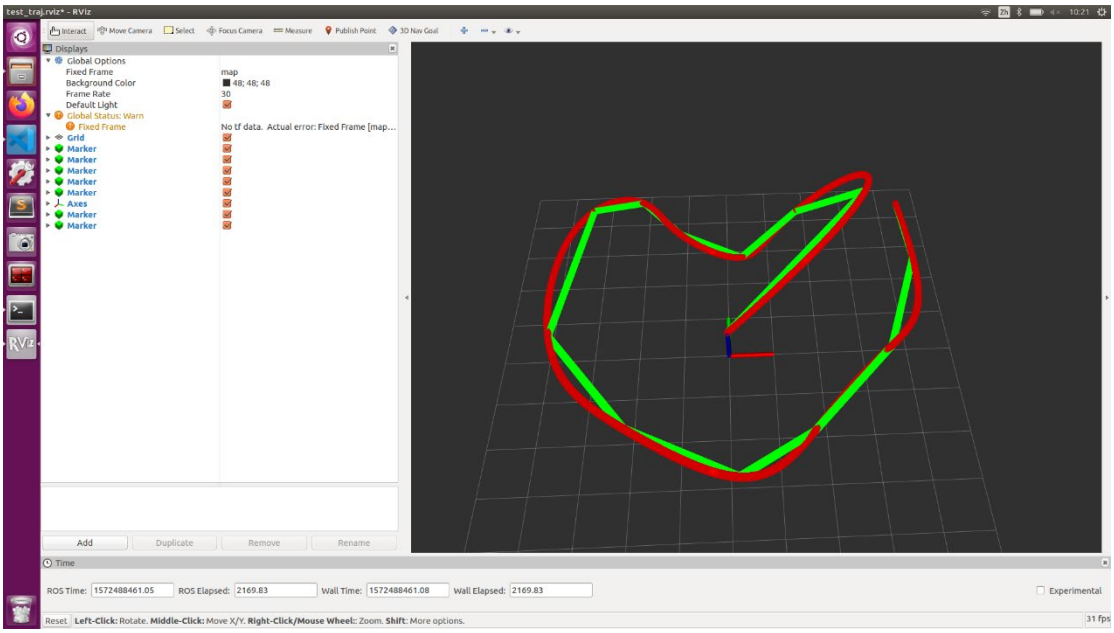
```cpp
    STEP 2: Learn the "Closed-form solution to minimum snap" in L5, then finish this PolyQP

    variable declaration: input       const int d_order,                    // the order of
                                      const Eigen::MatrixXd &Path,          // waypoints co
                                      const Eigen::MatrixXd &Vel,           // boundary vel
                                      const Eigen::MatrixXd &Acc,           // boundary acc
                                      const Eigen::VectorXd &Time)          // time allocat
                          output      MatrixXd PolyCoeff(m, 3 * p_num1d);   // position(x,y

*/

Eigen::MatrixXd TrajectoryGeneratorWaypoint::PolyQPGeneration(
            const int d_order,                    // the order of derivative
            const Eigen::MatrixXd &Path,          // waypoints coordinates (3d)
            const Eigen::MatrixXd &Vel,           // boundary velocity
            const Eigen::MatrixXd &Acc,           // boundary acceleration
            const Eigen::VectorXd &Time)          // time allocation in each segment
{
    // enforce initial and final velocity and accleration, for higher order derivatives, ju
    int p_order   = 2 * d_order - 1;              // the order of polynomial
    int p_num1d   = p_order + 1;                  // the number of variables in each segmen

    int m = Time.size();                          // the number of segments
    MatrixXd PolyCoeff = MatrixXd::Zero(m, 3 * p_num1d);           // position(x,y,z), so w
    VectorXd Px(p_num1d * m), Py(p_num1d * m), Pz(p_num1d * m);

    /*   Produce Mapping Matrix A to the entire trajectory, A is a mapping matrix that maps
```

详情内容请仔细阅读代码框架里的注释，并结合课程 ppt 教授的内容一起消化。

4. 最终效果

   绿色为手动给定改的 path，红色为优化后的 trajectory





学习机会不易，且行且珍惜。