运动规划第六章作业 ROS 版

1. 首先确定优化的目标函数，因为是轨迹跟踪问题，所以还是以实际的轨迹与目标轨迹的差值作为优化目标，

J = (xT-xrT)*Q*(x-xr)

状态 x 的表达式课件已经给出，如下式，带入进去展开即可，

x = AAx0+BBu+gg

但是最终的表达式要写成下面的形式

J = 0.5 x^T P_ x + q_^T x

在给出的代码中已经给出了 P_和 q_的表达式

Eigen::SparseMatrix<double> BBT_sparse = BB_sparse.transpose();

P_ = BBT_sparse * Qx_ * BB_sparse;

q_ = BBT_sparse * Qx_.transpose() * (AA_sparse * x0_sparse + gg_sparse) + BBT_sparse * qx;

比较完整的表达式，最后会得出 qx 的表达式的值为下式，

qx = -QT*xr

xr 即为包含 x,y 位置和 phi 值的状态矩阵，Q 是一个对角阵，可以直接乘出来计算，如果直接用两个矩阵去乘，编程的时候会报错，暂时没有找出原因

```
qx.coeffRef(i * n + 0, 0) = -1.0 * xy.x();
qx.coeffRef(i * n+ 1, 0) = -1.0 * xy.y();
qx.coeffRef(i * n + 2, 0) = -1.0 * rho_ * phi;
qx.coeffRef(i * n + 3, 0 ) = 0.0;
```

2. 在做无 delay 编程的时候，需要将 delay 相关的代码都屏蔽掉，不然可能会出错，比如下方的代码，另外 car_simulator 和 mpc_car 的 configure 文件中 delay 的值都需要改为 0，如果 car_simulator 里面的没有更改，达不到预期效果。

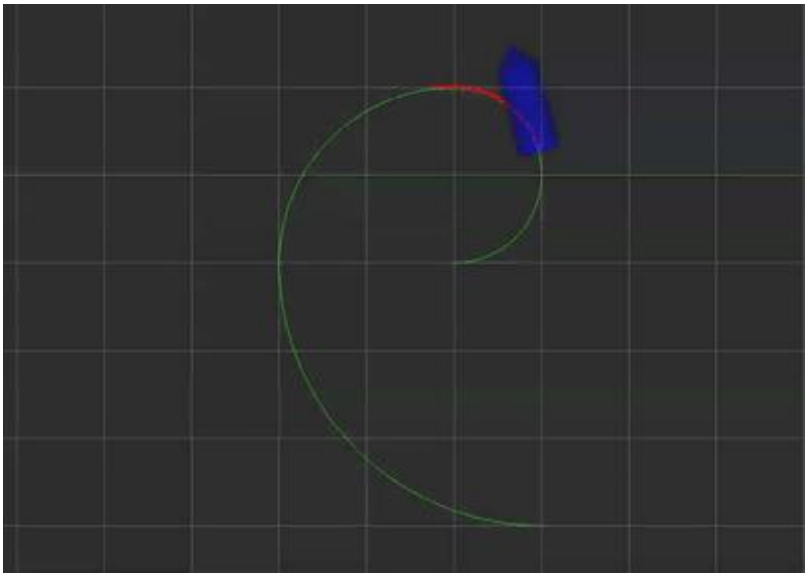
其他的按照代码里面给出的提示以及课件的资料填写即可

```
// VectorX x0_delay = x0_observe_;
// double dt = 0.001;
// 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);
//   p.pose.position.x = x0_delay(0);
//   p.pose.position.y = x0_delay(1);
//   p.pose.position.z = 0.0;
//   msg.poses.push_back(p);
// }
// traj_delay_pub_.publish(msg);
}
```

3. 有 delay 的代码主要是填写 compensateDelay 函数,这里面的逻辑和下方发消息的编写类似

```
// 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);
```

编写完成之后, 按照 reademe 里面的步骤运行代码, 有的电脑里面没有 zsh, 可以换成 bash。另外需要在 Ubuntu18.04 版本运行, Ubuntu16.04 运行会出错, 应该是对于 Eigen 库和 C++17 部分特性不支持
最终显示的效果如下图的 gif 所示，小车可以沿着预设的轨迹行驶



MPCC 代码改动量较大，有兴趣的可以参考下面这篇论文和对应代码，
Optimization-Based Autonomous Racing of 1:43 Scale RC Cars， Alexander Liniger, Alexander Domahidi and Manfred Morari