1. 补充代码，实现两帧间的 IMLS-ICP 激光匹配；（6 分）

代码：

计算法向量:

Eigen::Vector2d IMLSICPMatcher::ComputeNormal(std::vector<Eigen::Vector2d>&*nearPoints*)

{

    Eigen::Vector2d normal;

*//TODO*

*//根据周围的激光点计算法向量，参考ppt中NICP计算法向量的方法*

    Eigen::Vector2d center(0, 0);

    for (size\_t i = 0; i < nearPoints.size(); i++) {

        center = center + nearPoints[i];

    }

    center = center / nearPoints.size();

    Eigen::Matrix2d m;

    for (size\_t i = 0; i < nearPoints.size(); i++) {

        m = m + (nearPoints[i] - center) \* (nearPoints[i] - center).transpose();

    }

    m = m / nearPoints.size();

*// A = V \* D \* VT.*

    Eigen::EigenSolver<Eigen::Matrix2d> es(m);

    Eigen::Matrix2d D = es.pseudoEigenvalueMatrix();

    Eigen::Matrix2d V = es.pseudoEigenvectors();

    if (D(0, 0) > D(1, 1)) {

        normal = V.col(1);

    } else {

        normal = V.col(0);

    }

*//end of TODO*

    return normal;

}

计算height：

*//TODO*

*//根据函数进行投影．计算height，即ppt中的I(x)*

    Eigen::Vector2d v;

    double w, w\_sum = 0, sum = 0;

    for (size\_t i = 0; i < nearPoints.size(); i++) {

        v = x - nearPoints[i];

        w = exp(-((v[0] \* v[0] + v[1] \* v[1])) / (m\_h \* m\_h));

        w\_sum += w;

        sum += w \* v.dot(nearNormals[i]);

    }

    height = sum / w\_sum;

*//end of TODO*

计算yi

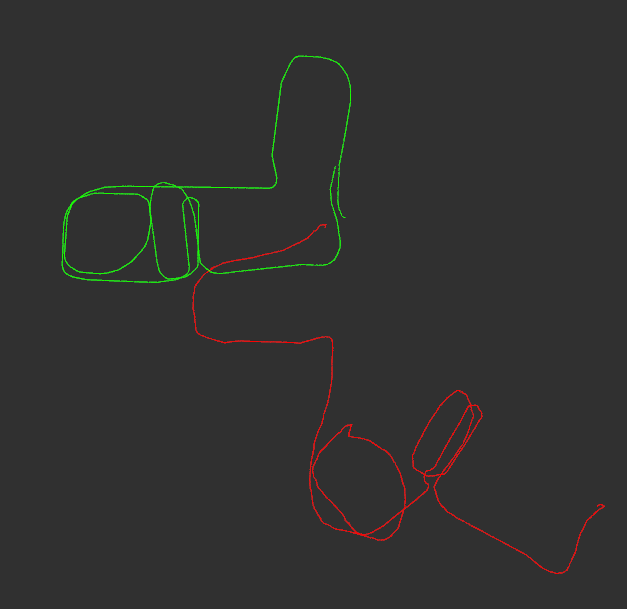
*//TODO*

*//计算yi．*

        yi = xi - height \* nearNormal;

*//end of TODO*

输出结果：



2. 将第一题 IMLS-ICP 匹配的接口换成第二次作业中 CSM 库的 ICP 匹配接口，并生成激光匹配的轨迹；

**[**代码修改部分]：

在main.cpp文件里面修改

（1）首先将champion\_nav\_msgs消息类型转化为csm的数据类型LDP，修改原来的数据转换函数如下所示：

*//把激光雷达数据 转换为PI-ICP需要的数据*

    void ConvertChampionLaserScanToLDP(const champion\_nav\_msgs::ChampionNavLaserScanConstPtr&*msg*,

        LDP&*ldp*)

    {

        int nPts = msg->ranges.size();

*// int nPts = pScan->intensities.size();*

        ldp = ld\_alloc\_new(nPts);

        for (int i = 0; i < nPts; i++) {

            double dist = msg->ranges[i];

            if (dist > msg->range\_min && dist < msg->range\_max) {

                ldp->valid[i] = 1;

                ldp->readings[i] = dist;

            } else {

                ldp->valid[i] = 0;

                ldp->readings[i] = -1;

            }

            ldp->theta[i] = msg->angles[i];

        }

        ldp->min\_theta = msg->angle\_min;

        ldp->max\_theta = msg->angle\_max;

        ldp->odometry[0] = 0.0;

        ldp->odometry[1] = 0.0;

        ldp->odometry[2] = 0.0;

        ldp->true\_pose[0] = 0.0;

        ldp->true\_pose[1] = 0.0;

        ldp->true\_pose[2] = 0.0;

    }

（2）在调用csm的接口函数之前还要设置一些基础的参数，在构造函数里面调用这个函数：

*//设置PI-ICP的参数*

    void SetPIICPParams()

    {

*//设置激光的范围*

        m\_PIICPParams.min\_reading = 0.1;

        m\_PIICPParams.max\_reading = 20;

*//设置位姿最大的变化范围*

        m\_PIICPParams.max\_angular\_correction\_deg = 20.0;

        m\_PIICPParams.max\_linear\_correction = 1;

*//设置迭代停止的条件*

        m\_PIICPParams.max\_iterations = 50;

        m\_PIICPParams.epsilon\_xy = 0.000001;

        m\_PIICPParams.epsilon\_theta = 0.0000001;

*//设置correspondence相关参数*

        m\_PIICPParams.max\_correspondence\_dist = 1;

        m\_PIICPParams.sigma = 0.01;

        m\_PIICPParams.use\_corr\_tricks = 1;

*//设置restart过程，因为不需要restart所以可以不管*

        m\_PIICPParams.restart = 0;

        m\_PIICPParams.restart\_threshold\_mean\_error = 0.01;

        m\_PIICPParams.restart\_dt = 1.0;

        m\_PIICPParams.restart\_dtheta = 0.1;

*//设置聚类参数*

        m\_PIICPParams.clustering\_threshold = 0.2;

*//用最近的10个点来估计方向*

        m\_PIICPParams.orientation\_neighbourhood = 10;

*//设置使用PI-ICP*

        m\_PIICPParams.use\_point\_to\_line\_distance = 1;

*//不进行alpha\_test*

        m\_PIICPParams.do\_alpha\_test = 0;

        m\_PIICPParams.do\_alpha\_test\_thresholdDeg = 5;

*//设置trimmed参数 用来进行outlier remove*

        m\_PIICPParams.outliers\_maxPerc = 0.9;

        m\_PIICPParams.outliers\_adaptive\_order = 0.7;

        m\_PIICPParams.outliers\_adaptive\_mult = 2.0;

*//进行visibility\_test 和 remove double*

        m\_PIICPParams.do\_visibility\_test = 1;

        m\_PIICPParams.outliers\_remove\_doubles = 1;

        m\_PIICPParams.do\_compute\_covariance = 0;

        m\_PIICPParams.debug\_verify\_tricks = 0;

        m\_PIICPParams.use\_ml\_weights = 0;

        m\_PIICPParams.use\_sigma\_weights = 0;

    }

（3）在激光雷达回调函数里面使用，使用csm的接口函数，代替原来的imls接口函数

 rPose\_csm = PIICPBetweenTwoFrames(currentLDP);

然后这里得到的结果是新的一帧在上一帧的位置，转换为矩阵的形式

rPose << cos(rPose\_csm(2)), -sin(rPose\_csm(2)), rPose\_csm(0),

            sin(rPose\_csm(2)), cos(rPose\_csm(2)), rPose\_csm(1),

            0, 0, 1;

就与原来的代码就连接起来了。

如下所示：

 void championLaserScanCallback(const champion\_nav\_msgs::ChampionNavLaserScanConstPtr&*msg*)

    {

        if (m\_isFirstFrame == true) {

            std::cout << "First Frame" << std::endl;

            m\_isFirstFrame = false;

            m\_prevLaserPose = Eigen::Vector3d(0, 0, 0);

            pubPath(m\_prevLaserPose, m\_imlsPath, m\_imlsPathPub);

*// ConvertChampionLaserScanToEigenPointCloud(msg, m\_prevPointCloud);*

            ConvertChampionLaserScanToLDP(msg, m\_prevLDP);

            return;

        }

*// std::vector<Eigen::Vector2d> nowPts;*

*// ConvertChampionLaserScanToEigenPointCloud(msg, nowPts);*

        LDP currentLDP;

        ConvertChampionLaserScanToLDP(msg, currentLDP);

*//调用imls进行icp匹配，并输出结果．*

*// m\_imlsMatcher.setSourcePointCloud(nowPts);*

*// m\_imlsMatcher.setTargetPointCloud(m\_prevPointCloud);*

        Eigen::Matrix3d rPose, rCovariance;

*// csm*

        rPose\_csm = PIICPBetweenTwoFrames(currentLDP);

*// std::cout << "---2---" << std::endl;*

*// std::cout << "csm Match Successful:" << rPose\_csm(0) << "," << rPose\_csm(1) << "," << rPose\_csm(2) << std::endl;*

        Eigen::Matrix3d lastPose;

        lastPose << cos(m\_prevLaserPose(2)), -sin(m\_prevLaserPose(2)), m\_prevLaserPose(0),

            sin(m\_prevLaserPose(2)), cos(m\_prevLaserPose(2)), m\_prevLaserPose(1),

            0, 0, 1;

        rPose << cos(rPose\_csm(2)), -sin(rPose\_csm(2)), rPose\_csm(0),

            sin(rPose\_csm(2)), cos(rPose\_csm(2)), rPose\_csm(1),

            0, 0, 1;

        Eigen::Matrix3d nowPose = lastPose \* rPose;

        m\_prevLaserPose << nowPose(0, 2), nowPose(1, 2), atan2(nowPose(1, 0), nowPose(0, 0));

        pubPath(m\_prevLaserPose, m\_imlsPath, m\_imlsPathPub);

*// m\_prevPointCloud = nowPts;*

    }

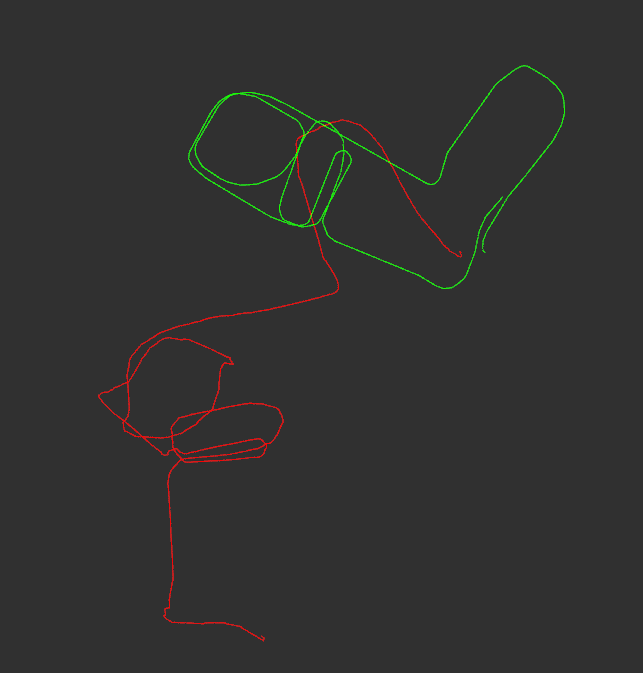
其中接口函数里面需要设置较好的初始值，设置的初始值是第一题目里面前面某一时刻的输出结果，如下所示：

        tmprPose[0] = -0.0474715;

        tmprPose[1] = 0.0464215;

        tmprPose[2] = 0.0791398 / 180 \* M\_PI;

输出结果：



1. 阅读 ICP 相关论文，总结课上所学的几种 ICP 及其相关变型并简述其异同(ICP，PL-ICL，NICP, IMLS-ICP)；

ICP：点对点进行匹配，但是点对点进行匹配是实际情况下的对面进行建模差距比较大，所以效果比较差。目标函数是距离最临近点的距离。

相同点：

都进行迭代计算

不同点：

PL-ICP：点对线进行匹配，更好的对实际进行建模，用分段线性的方法来对实际曲面进行模拟。目标函数是最临近直线的距离。收敛速度相比于ICP更快，是二阶收敛。但是对于初始值更加敏感，不单独使用，与里程计，CSM等一起使用。

NICP：将曲率和法向量考虑进去，筛选掉不符合匹配条件的点。相比与ICP，误差项除了包括点对点的欧式距离之外，还包括对应点法向量的角度差。

IMLS-ICP：选取具有代表性的点（结构化的点），同时保证选取具有客观性，分布均衡。计算点到点云代表的曲面的距离，计算一个对应的点来进行匹配。目标函数是当前点与计算出来的对应点的法向量上的距离。相比前面的方法，对实际场景进行建模更加贴切，但是计算量也更大。

1. 简答题，开放性答案：现在你已经了解了多种 ICP 算法，你是否也能提出一种改进的 ICP 算法，或能提升 ICP 总体匹配精度或速度的技巧？请简述你的改进策略。（2 分）

答：ICP算法对于初始值的比较敏感，这部分可以考虑进行改进以提高匹配的精度和速度，因为一个好的初始值可以加快进行收敛。对出值进行改进可以采用融合的方式，加入IMU或者里成计的数据，作为初始值。

另外，激光雷达后面准确度下降一部分是因为累计误差的原因，可以进行重定位，而激光点云进行重定位又比较麻烦，可以考虑进行视觉重定位进行融合，或者设计训练一个深度神经网络，判断两帧点云是否是同一地点，进行回环检测。