

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NCP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;
}
```

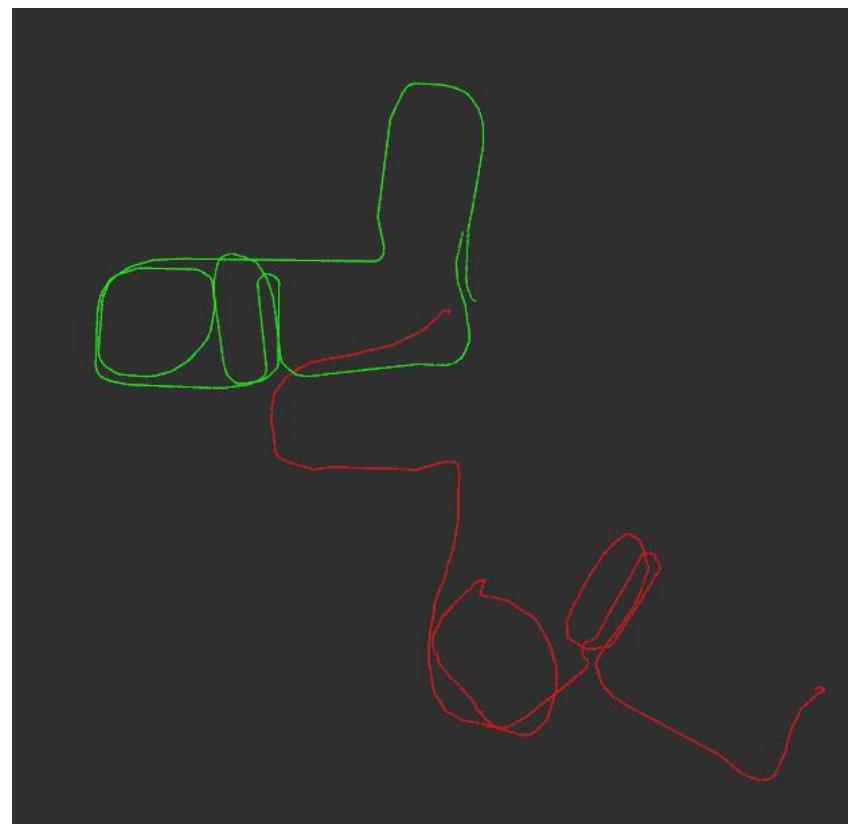
计算 height:

```
//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 SetPIICPPParams()
{
    //设置激光的范围
    m_PIICPPParams.min_reading = 0.1;
    m_PIICPPParams.max_reading = 20;

    //设置位姿最大的变化范围
    m_PIICPPParams.max_angular_correction_deg = 20.0;
    m_PIICPPParams.max_linear_correction = 1;
    //设置迭代停止的条件
    m_PIICPPParams.max_iterations = 50;
    m_PIICPPParams.epsilon_xy = 0.000001;
    m_PIICPPParams.epsilon_theta = 0.0000001;
    //设置 correspondence 相关参数
    m_PIICPPParams.max_correspondence_dist = 1;
    m_PIICPPParams.sigma = 0.01;
    m_PIICPPParams.use_corr_tricks = 1;
    //设置 restart 过程，因为不需要 restart 所以可以不管
    m_PIICPPParams.restart = 0;
    m_PIICPPParams.restart_threshold_mean_error = 0.01;
    m_PIICPPParams.restart_dt = 1.0;
    m_PIICPPParams.restart_dtheta = 0.1;
    //设置聚类参数
    m_PIICPPParams.clustering_threshold = 0.2;
    //用最近的 10 个点来估计方向
    m_PIICPPParams.orientation_neighbourhood = 10;
    //设置使用 PI-ICP
    m_PIICPPParams.use_point_to_line_distance = 1;
    //不进行 alpha_test
    m_PIICPPParams.do_alpha_test = 0;
    m_PIICPPParams.do_alpha_test_thresholdDeg = 5;
    //设置 trimmed 参数 用来进行 outlier remove
    m_PIICPPParams.outliers_maxPerc = 0.9;
    m_PIICPPParams.outliers_adaptive_order = 0.7;
    m_PIICPPParams.outliers_adaptive_mult = 2.0;
    //进行 visibility_test 和 remove double
    m_PIICPPParams.do_visibility_test = 1;
    m_PIICPPParams.outliers_remove_doubles = 1;
    m_PIICPPParams.do_compute_covariance = 0;
    m_PIICPPParams.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::ChampionNavLaserScanConst
Ptr& 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;
}

```

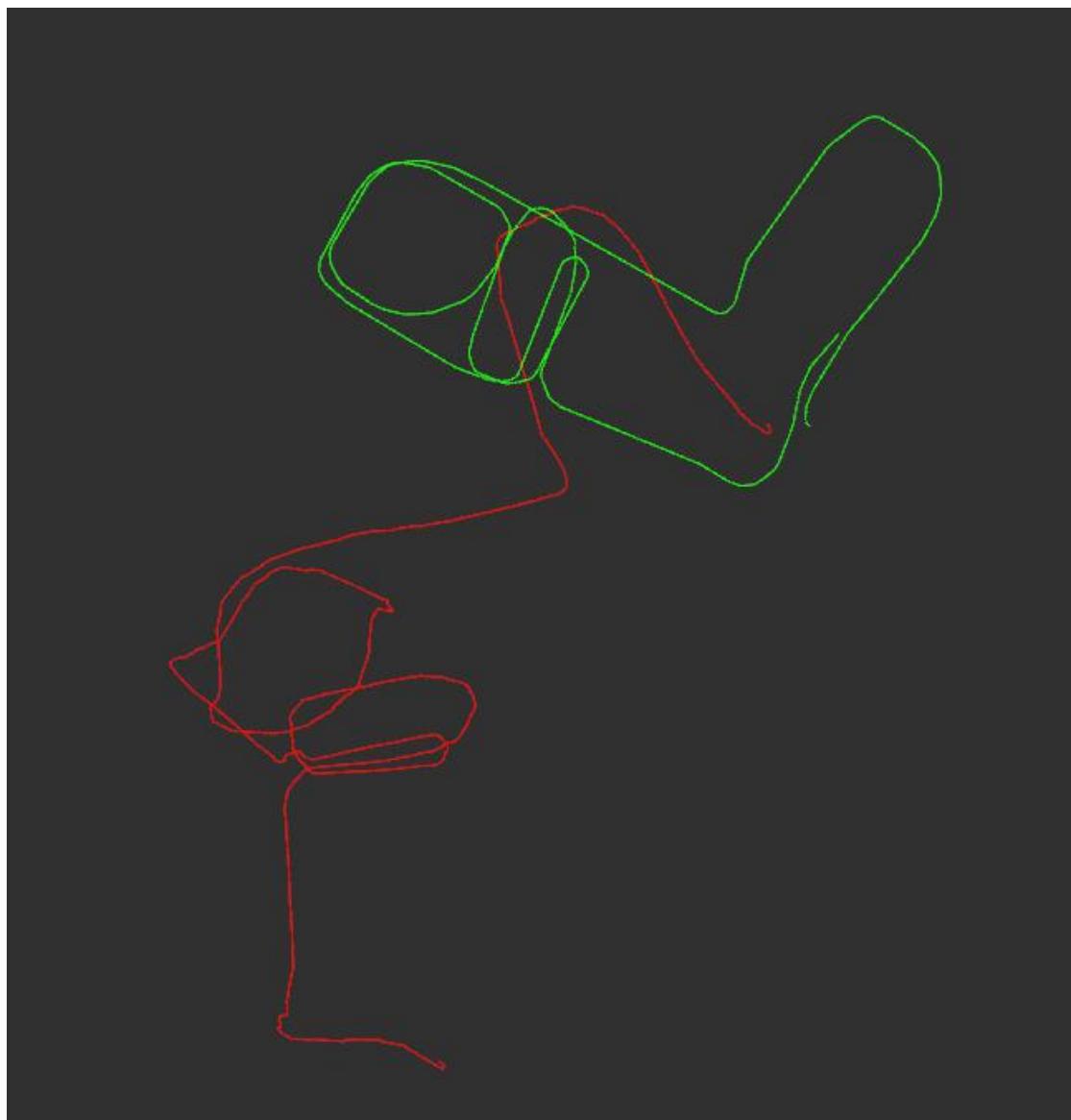
其中接口函数里面需要设置较好的初始值，设置的初始值是第一题目里面前面某一时刻的输出结果，如下所示：

```

tmpPose[0] = -0.0474715;
tmpPose[1] = 0.0464215;
tmpPose[2] = 0.0791398 / 180 * M_PI;

```

输出结果：



2. 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。

3. 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 或者里程计的数据，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。