浙江大学实验报告

课程名称：智能移动技术　　指导老师：　熊蓉，王越　　成绩：
实验名称：ICP 点云匹配　　姓名：潘力豪

一、实验内容

读取 10 个 ply 格式点云文件，包含机器人的二维激光点云数据。理解点云数据的格式和表征含义。自己实现 ICP 算法，达到和 matlab 中 ICP 匹配库相近的效果，实现相邻点云之间的匹配。得到每两帧之间的相对位姿关系（提示：除了使用连续两帧的匹配，也可以尝试使用跨 n 帧的匹配来减少累计误差）【ICP 算法（点到点 ICP、基于特征的 ICP 等）不限、编程语言(C/C++、python、matlab)不限】基于 ICP 匹配的结果，生成机器人这 10 帧的定位轨迹和局部点云地图。轨迹包含机器人在每个时间步的位置和姿态信息，局部点云地图为将 10 帧点云地图按匹配结果进行坐标变换后叠加而成的点云。

二、问题解决思路

语言：python
调用库：numpy(用于矩阵运算)，matplotlib.pyplot（用于绘制点云图和机器人轨迹）

①算法设计思路：

该实验需要解决的问题有三个：点云文件的读取，两帧点云之间点与点的匹配，ICP 计算最优 R 和 t（转换矩阵 T）对于这三个问题我分别对我的代码原理进行阐释：

Ⅰ 对于点云文件的读取，我将 ply 文件中 180 个点的的坐标信息转化为 180*3 的矩阵进行处理。

Ⅱ 对于两帧点云之间点与点的匹配，我采用了最近邻匹配算法，对于两个点集 P 和 Q（Q 是被变换的对象点集），我对 Q 中每一个点 qi,在 P 中寻找与之距离最近的一点 pj，并记录 j 的下标，以此来实现点与点之间一一对应。

Ⅲ 对于 ICP 计算优 R 和 t，出于计算的简便性，我通过 4*4 的转换矩阵 T 进行计算（T 的左上角 3*3 为 R，最右边一列为 t 和 1）。

ICP 算法的基本原理是：分别在带匹配的目标点云 P 和源点云 Q 中，按照一定的约束条件，找到最邻近点（pi，qi），然后计算出最优匹配参数 R 和 t，使得误差函数最小。误差函数为 E（R，t）为：

$$E(R,t)=\frac{1}{n}\sum_{i=1}^{n}||q_i-(Rp_i+t)||^2$$

其中 n 为最邻近点对的个数，pi 为目标点云 P 中的一点，qi 为源点云 Q 中与 pi 对应的最近点，R 为旋转矩阵，t 为平移向量。

而 ICP 算法的基本步骤为循环迭代：

（1）在目标点云 P 中取点集 pi∈P；
（2）找出源点云 Q 中的对应点集 qi∈Q，使得||qi-pi||=min；
（3）计算旋转矩阵 R 和平移矩阵 t，使得误差函数最小；
（4）对 pi 使用上一步求得的旋转矩阵 R 和平移矩阵 t 进行旋转和平移变换，的到新的对应点集 pi’
（5）计算 pi’与对应点集 qi 的平均距离（欧式距离）；

（6）如果 d 小于某一给定的阈值或者大于预设的最大迭代次数，则停止迭代计算。否则返回第 2 步，直到满足收敛条件为止。

而根据点云匹配计算最优 R 与 t（T）则采用了 SVD 方法，通过质心与奇异值分解算出最优 R 和 T：

$$\mathbf{R}^* = \mathbf{U}\mathbf{V}^T$$
$$\mathbf{t}^* = \boldsymbol{\mu}_q - \mathbf{R}\boldsymbol{\mu}_p$$

Ⅳ 由于相邻两帧点云中存在一些无法匹配的点，这些点过于极端导致计算最优 R 与 t 时会因此出现误差。因此我增加了滤波环节，将点云匹配中难以匹配的点滤去。基本思想就是设置一个阈值，在匹配环节中，如果 Q 中的一点 qi 在 P 中的最近点之间的欧氏距离大于这个阈值，则认为不存在其匹配对应点，将其滤去。

②代码详细解释

对于点云文件的读取，我定义了函数 read_ply_file（file_path），用于读取 ply 格式点云文件中的点坐标信息并转化成 N*m 的矩阵输出（N 为点的信息个数）

对于两帧点云之间点与点的匹配，我定义了 nearest_neighbor（source, distination）函数，获取两个点云数据 source 与 distination 中 distination 的每个点距离 source 最近点的距离和下标，并输出，完成了点与点之间的最近点匹配。

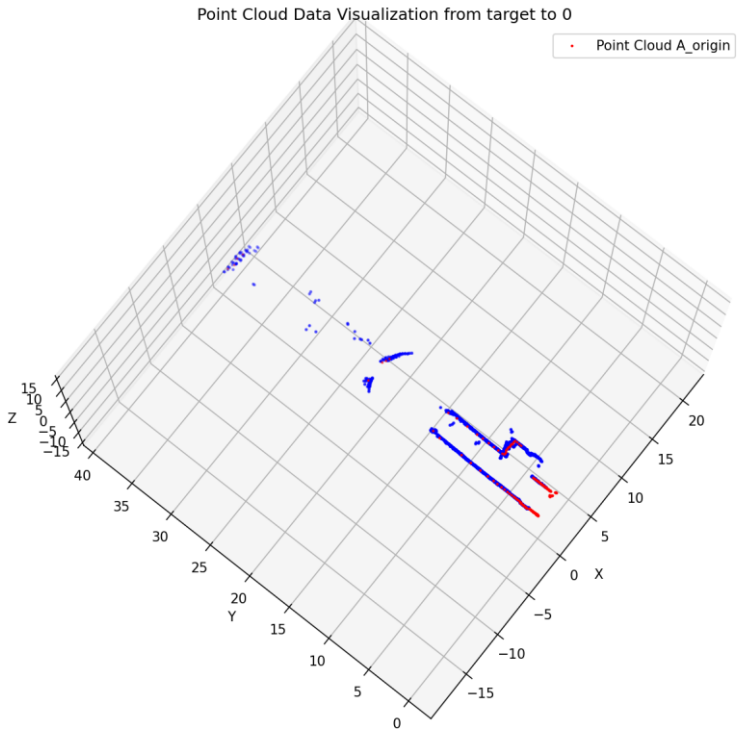
对于 ICP 计算优 R 和 t，我先定义了 best_transform_T(P, Q)函数，用于对两个匹配好的点集，采用 SVD 方法计算出最优转换矩阵 T 并输出。

之后，定义 ICP(P,Q,max_iteration_epoch = 50,tolerance = 0.01,step = 0.0001)函数用于进行 ICP 迭代，在循环中调用 best_transform_T 函数和 nearest_neighbor 函数进行最优 R 与 t 的计算，loss 为点集中平均每个点到最近点的欧氏距离的平均值。max_iteration_epoch = 50,tolerance = 0.01,step = 0.0001 分别表示最大循环次数，循环停止阈值，相邻两次循环差值阈值。迭代循环停止的条件为：平均误差 loss 小于阈值 tolerance 或者两次相邻循环的 loss 差值小于阈值 step。

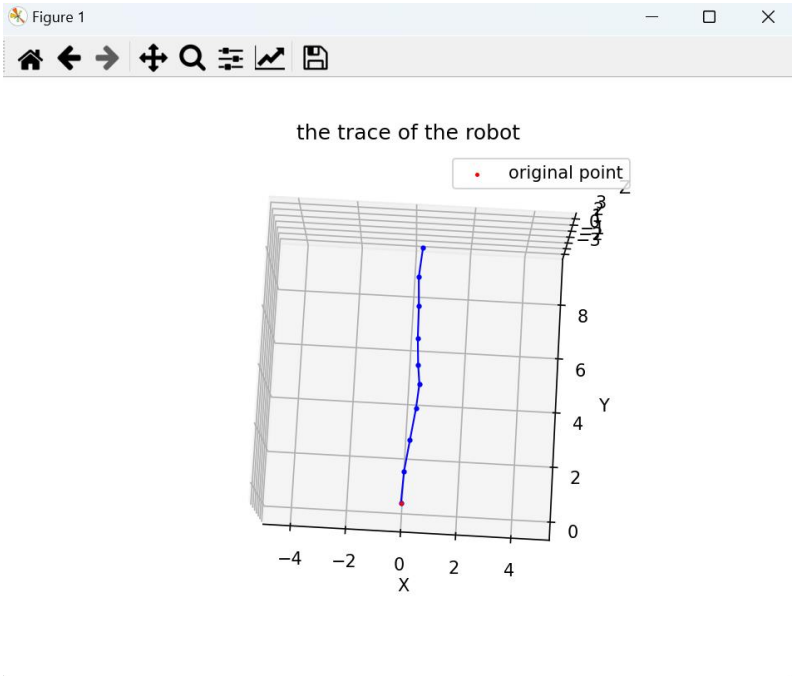
而滤波算法则是定义了 filtering（P,Q，max_dis = 5.0）函数，对 Q 点集进行极端值滤波。函数里调用了相邻点匹配函数 nearest_neighbor 的结果 distances，如果距离大于 max_dis 则将该点滤去。但是在滤波过程中需要保持两个点集之间点的个数一致（不然 SVD 分解会数据格式不匹配），因此对于极端点，我将其数据变为其点云中的第一个点的数据来计算 R 与 t（因为据观察，ply 文件中的第一个点都是能够匹配上的点），以减小计算的误差。当然，滤波的数据仅用于计算更优的 R 与 t，在绘制点云图时使用的是原始数据而不是滤波后的数据。

三、实验结果

十帧点云合成图（合成到 0.ply 的坐标系下，其中红色的是 0.ply 的点云，蓝色的是其余 9 帧的点云）：



机器人轨迹图（位置信息，第 0 帧定义为原点（x=0，y=0，z=0）红色表示）：



以下为具体的坐标：

```
第 0 帧时机器人的位置(x,y,z)为:[ 0 , 0 , 0 ]
第 1 帧时机器人的位置(x,y,z)为:[ 0.04591119627609119 , 1.1081719298019919 , 0.0 ]
第 2 帧时机器人的位置(x,y,z)为:[ 0.18816601725323756 , 2.1991495671544734 , 0.0 ]
第 3 帧时机器人的位置(x,y,z)为:[ 0.33860323246523627 , 3.3397292366913183 , 0.0 ]
第 4 帧时机器人的位置(x,y,z)为:[ 0.4085558205862974 , 4.1943668144624215 , 0.0 ]
第 5 帧时机器人的位置(x,y,z)为:[ 0.326329002809405 , 4.873685328210589 , 0.0 ]
第 6 帧时机器人的位置(x,y,z)为:[ 0.26149475895113616 , 5.817381146785306 , 0.0 ]
第 7 帧时机器人的位置(x,y,z)为:[ 0.2362003799797106 , 6.9492607111198215 , 0.0 ]
第 8 帧时机器人的位置(x,y,z)为:[ 0.17500813092556916 , 8.003565261875217 , 0.0 ]
第 9 帧时机器人的位置(x,y,z)为:[ 0.26568029391623055 , 9.059512206264486 , 0.0 ]
```

```
从第 0 帧到第 1 帧的转换矩阵为：
 [[ 0.99950168 -0.03156556  0.         -0.01365764]
 [ 0.03156556  0.99950168  0.         -1.09867103]
 [ 0.          0.          1.          0.        ]
 [ 0.          0.          0.          1.        ]]
从第 1 帧到第 2 帧的转换矩阵为：
 [[ 0.99738654 -0.07225025  0.         -0.03574304]
 [ 0.07225025  0.99738654  0.         -1.04616084]
 [ 0.          0.          1.          0.        ]
 [ 0.          0.          0.          1.        ]]
从第 2 帧到第 3 帧的转换矩阵为：
 [[ 0.99887517 -0.04741716  0.          0.02169042]
 [ 0.04741716  0.99887517  0.         -1.12623188]
 [ 0.          0.          1.          0.        ]
 [ 0.          0.          0.          1.        ]]
从第 3 帧到第 4 帧的转换矩阵为：
 [[ 0.99186635  0.12728372  0.         -0.05234219]
 [-0.12728372  0.99186635  0.         -0.86440188]
 [ 0.          0.          1.          0.        ]
 [ 0.          0.          0.          1.        ]]
从第 4 帧到第 5 帧的转换矩阵为：
 [[ 0.99812959  0.06113368  0.          0.10173764]
 [-0.06113368  0.99812959  0.         -0.6494814 ]
 [ 0.          0.          1.          0.        ]
 [ 0.          0.          0.          1.        ]]
从第 5 帧到第 6 帧的转换矩阵为：
 [[ 0.99985394 -0.01709082  0.          0.03141739]
 [ 0.01709082  0.99985394  0.         -0.90208041]
 [ 0.          0.          1.          0.        ]
 [ 0.          0.          0.          1.        ]]
从第 6 帧到第 7 帧的转换矩阵为：
 [[ 0.99979869 -0.02006461  0.          0.00619069]
 [ 0.02006461  0.99979869  0.         -1.13214523]
 [ 0.          0.          1.          0.        ]
 [ 0.          0.          0.          1.        ]]
从第 7 帧到第 8 帧的转换矩阵为：
 [[ 0.99415591 -0.10795383  0.          0.15658955]
 [ 0.10795383  0.99415591  0.         -1.03509124]
 [ 0.          0.          0.          1.        ]]
从第 8 帧到第 9 帧的转换矩阵为：
 [[ 0.99938386  0.03509858  0.         -0.03094516]
 [-0.03509858  0.99938386  0.         -1.05605701]
 [ 0.          0.          1.          0.        ]
 [ 0.          0.          0.          1.        ]]
```

四、结果分析与心得

实验结果非常合理，通过帧与帧之间的对应关系算出 9 个转换的矩阵 T，再通过矩阵变换计算出机械单位的位姿与轨迹。因为用的是 python，因此调用 matplot 库对点云与轨迹进行绘制，并尽力使得其效果与给的 matlab 例程相似。

这次实验对于我是一次非常有益的尝试，在调试过程中，对矩阵之间的关系花费了我较长的时间，也非常考验我的代码能力与理解能力。最终我成功完成了这次代码编写，在一次次的 debug 和学习中，我也对 ICP 的原理有了更加深入的了解。