智能移动技术二维ICP点云匹配

苏浩 学号：3210102806

1.作业要求

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。

2.解决思路

2.1数据处理

PLY（Polygon File Format）是一种常用的用于描述三维物体几何形状的文件格式。它可以包含点云、多边形、边界、曲线等各种几何元素。PLY文件格式的表征如下：

- `format`：指定文件格式，常见的有`ascii`和`binary_little_endian`等。
- `element`：描述文件中包含的元素类型及数量。
- `property`：描述每个元素的属性，如位置坐标、颜色等。
- `end_header`：表明头部信息结束，接下来的内容为实际数据。

在给定的PLY文件示例中：

- `format ascii 1.0` 表示文件格式为ASCII格式。
- `element vertex 180` 表示文件包含180个顶点。
- `property double x`、`property double y`、`property double z` 定义了每个顶点的坐标属性，分别是x、y、z坐标。
- `end_header` 表明头部信息结束。

接下来是180个顶点的坐标信息，每一行对应一个顶点的坐标。

通过plyfile库读取和处理点云数据：

```
 #读取点云文件函数
def load_point_cloud(filename):
    """Load point cloud data from a PLY file."""
    plydata = PlyData.read(filename)
    points = np.vstack([plydata['vertex']['x'], plydata['vertex']['y'],
plydata['vertex']['z']]).T
    return points
```

2.2ICP函数

采用点对点匹配的方式，主要的流程伪代码：

POINT-POINT ICP

input : Two pointclouds: $A = \{a_i\}, B = \{b_i\}$
An initial transformation: T_0
output: The correct transformation, T, which aligns A and B

$T \leftarrow T_0$;
while *not converged* **do**
  for $i \leftarrow 1$ **to** N **do**
    $m_i \leftarrow \texttt{FindClosestPointInA}(T \cdot b_i)$;
    if $||m_i - T \cdot b_i|| \leq d_{max}$ **then**
      $w_i \leftarrow 1$;
    else
      $w_i \leftarrow 0$;
    end
  end
  $T \leftarrow \underset{T}{\operatorname{argmin}} \{\sum_i w_i ||T \cdot b_i - m_i||^2\}$;
end

采用连续两次变换矩阵的误差作为收敛判定条件：

```
#判断是否收敛
        error = np.linalg.norm(transformation - prev_transformation)
        if error < error_tolerance:
            print('iterate finished')
            break
```

2.3最近邻查找

以计算欧氏距离（Euclidean distance）来查找最近邻，并使用`KDTree`数据结构来构建目标点云的索引，以加速最近邻搜索过程。

```
#寻找最近邻函数
def find_correspondences(source_points, target_points):
    """Find corresponding points in target cloud for source points."""
    valid_indices = np.isfinite(source_points).all(axis=1)
    cleaned_source_points = source_points[valid_indices]
    cleaned_target_points = target_points[valid_indices]
    #构造KDTree，进行最近邻查找
    kdtree = KDTree(cleaned_target_points)
    _, indices = kdtree.query(cleaned_source_points)
    correspondences = target_points[indices]

    return correspondences
```

2.4最小二乘问题

采用线性代数方法解决最小二乘问题：

线性代数求解方法

1. 定义两组点集合的质心位置$\mathbf{a}, \mathbf{b}$

$$\mathbf{a} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{a_i} \qquad b = \frac{1}{n}\sum_{i=1}^{n} b_i$$

2. 计算每个点的去质心坐标

$$q_i = \mathbf{a_i} - \mathbf{a} \qquad q'_i = b_i - \mathbf{b}$$

3. 根据以下优化问题计算旋转矩阵 $\mathbf{R}^* = \operatorname{argmin} \frac{1}{2}\sum_{i=1}^{n} \|\mathbf{q_i} - \mathbf{R}\mathbf{q'_i}\|^2$

4. 根据$\mathbf{R}$计算$\mathbf{t}$ $\mathbf{t}^* = \mathbf{a} - \mathbf{R}^*\mathbf{b}$

```
#最小二乘函数
def solve_least_squares(source_points_nearest, target_points_nearest):
    """Solve least squares problem for point cloud registration."""
    source_center = np.mean(source_points_nearest, axis=0)
    target_center = np.mean(target_points_nearest, axis=0)
    #求取点云质心并进行归一化
    source_relative = source_points_nearest - source_center
    target_relative = target_points_nearest - target_center
    #进行SVD分解
    W = np.dot(source_relative.T, target_relative)
    U, _, Vt = np.linalg.svd(W)
    rotation = np.dot(Vt.T, U.T)
    #求取平移量t
    translation = target_center - np.dot(rotation, source_center)
    transformation = np.identity(4)
    transformation[:3, :3] = rotation
    transformation[:3, 3] = translation.T
    return transformation
```

2.5其他函数

其他封装了保存点云、保存轨迹路径等函数，增加可读性和便于进行调试

2.6可视化

为了表征误差和可视化结果，通过计算ate表征轨迹误差

```
error = np.mean(np.linalg.norm(Correct_points - predict_points, axis=1))
```

并且写了一个脚本可视化轨迹。（时间所限，就直接复制了robo_trajectory.txt下的生成轨迹手动赋值）

3.实验结果与分析

3.1定位匹配误差

计算得到相邻两帧间的变换矩阵为：

```
Frame 0:
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]

Frame 1:
[[ 0.999475  0.032402  0.        0.044232]
 [-0.032402  0.999475  0.        1.085925]
 [ 0.        0.        1.        0.      ]
 [ 0.        0.        0.        1.      ]]

Frame 2:
[[ 0.994699  0.102826  0.        0.19022 ]
 [-0.102826  0.994699  0.        2.181384]
 [ 0.        0.        1.        0.      ]
 [ 0.        0.        0.        1.      ]]

Frame 3:
[[ 0.988222  0.153027  0.        0.335301]
 [-0.153027  0.988222  0.        3.389808]
 [ 0.        0.        1.        0.      ]
 [ 0.        0.        0.        1.      ]]

Frame 4:
[[ 0.999648  0.02654   0.        0.409749]
 [-0.02654   0.999648  0.        4.32209 ]
 [ 0.        0.        1.        0.      ]
 [ 0.        0.        0.        1.      ]]

Frame 5:
[[ 0.99818  -0.060303  0.        0.343737]
 [ 0.060303  0.99818   0.        5.192588]
 [ 0.        0.        1.        0.      ]
 [ 0.        0.        0.        1.      ]]

Frame 6:
[[ 0.999069 -0.043141  0.        0.273041]
 [ 0.043141  0.999069  0.        6.135863]
 [ 0.        0.        1.        0.      ]
 [ 0.        0.        0.        1.      ]]

Frame 7:
[[ 0.999718 -0.023767  0.        0.239555]
 [ 0.023767  0.999718  0.        7.053531]
 [ 0.        0.        1.        0.      ]
 [ 0.        0.        0.        1.      ]]

Frame 8:
```

```
[[ 0.997749  0.067057  0.        0.196093]
 [-0.067057  0.997749  0.        8.137574]
 [ 0.        0.        1.        0.      ]
 [ 0.        0.        0.        1.      ]]

Frame 9:
[[ 0.999487  0.032017  0.        0.259797]
 [-0.032017  0.999487  0.        9.170733]
 [ 0.        0.        1.        0.      ]
 [ 0.        0.        0.        1.      ]]
```

选取平移量上的轨迹，与matlab的pcregistericp函数计算真值对比，计算ate误差。

```
 Correct_points = [[0,0,0],
                [0.0459,1.1082,0],
                [0.1924,2.2068,0],
                [0.3375,3.3133,0],
                [0.3970,4.01,0],
                [0.3145,5.0264,0],
                [0.2742,5.9423,0],
                [0.1316,6.8498,9],
                [0.1663,8.1005,0],
                [0.1687,9.0731,0]
                ]
    predict_points= [[0,0,0],
                [0.044232,1.085925,0],
                [0.19022,2.181384,0],
                [ 0.335301, 3.389808,0],
                [0.409749, 4.32209,0],
                [0.343737, 5.192588,0],
                [ 0.273041,6.135863,0],
                [0.239555,7.053531,9],
                [0.196093,8.137574,0],
                [0.259797,9.170733,0]
                ]
    # print(Correct_points)
    # 将列表转换为 NumPy 数组
    Correct_points = np.array(Correct_points)
    predict_points = np.array(predict_points)
# 计算 ATE 误差
    error = np.mean(np.linalg.norm(Correct_points - predict_points, axis=1))
```

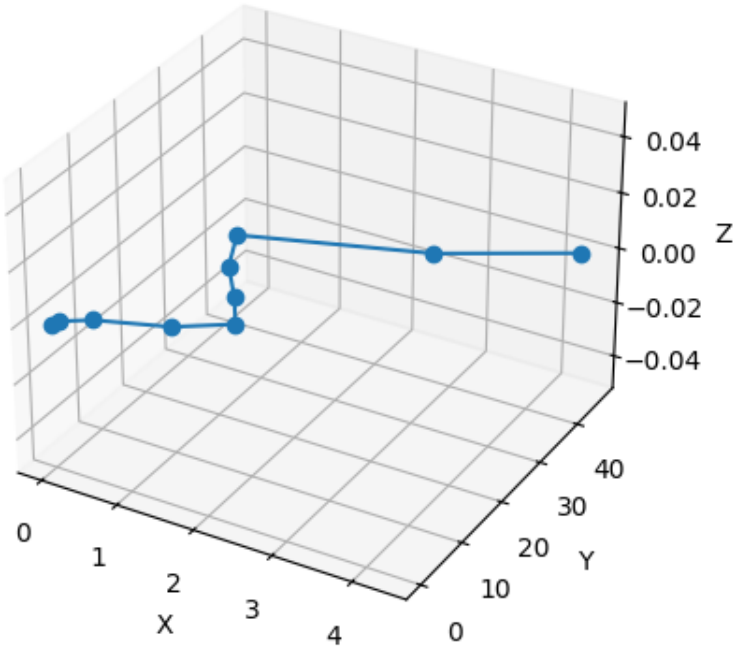
得到误差结果：

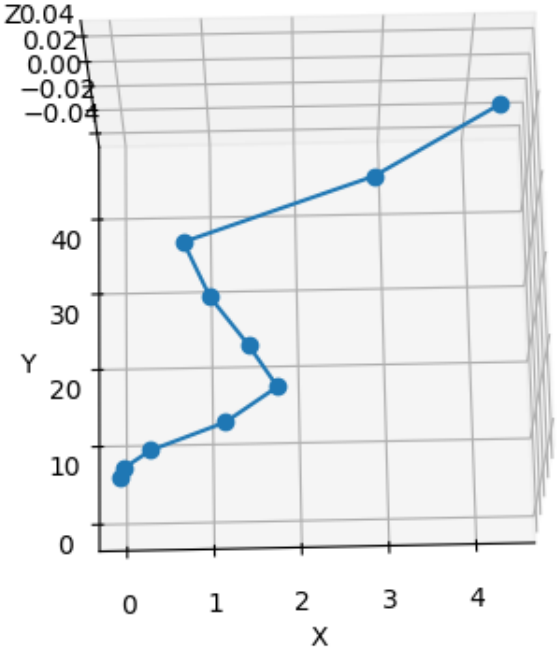
```
ATE 误差: 0.12107029198427252
```

误差已符合要求，定位较为准确。

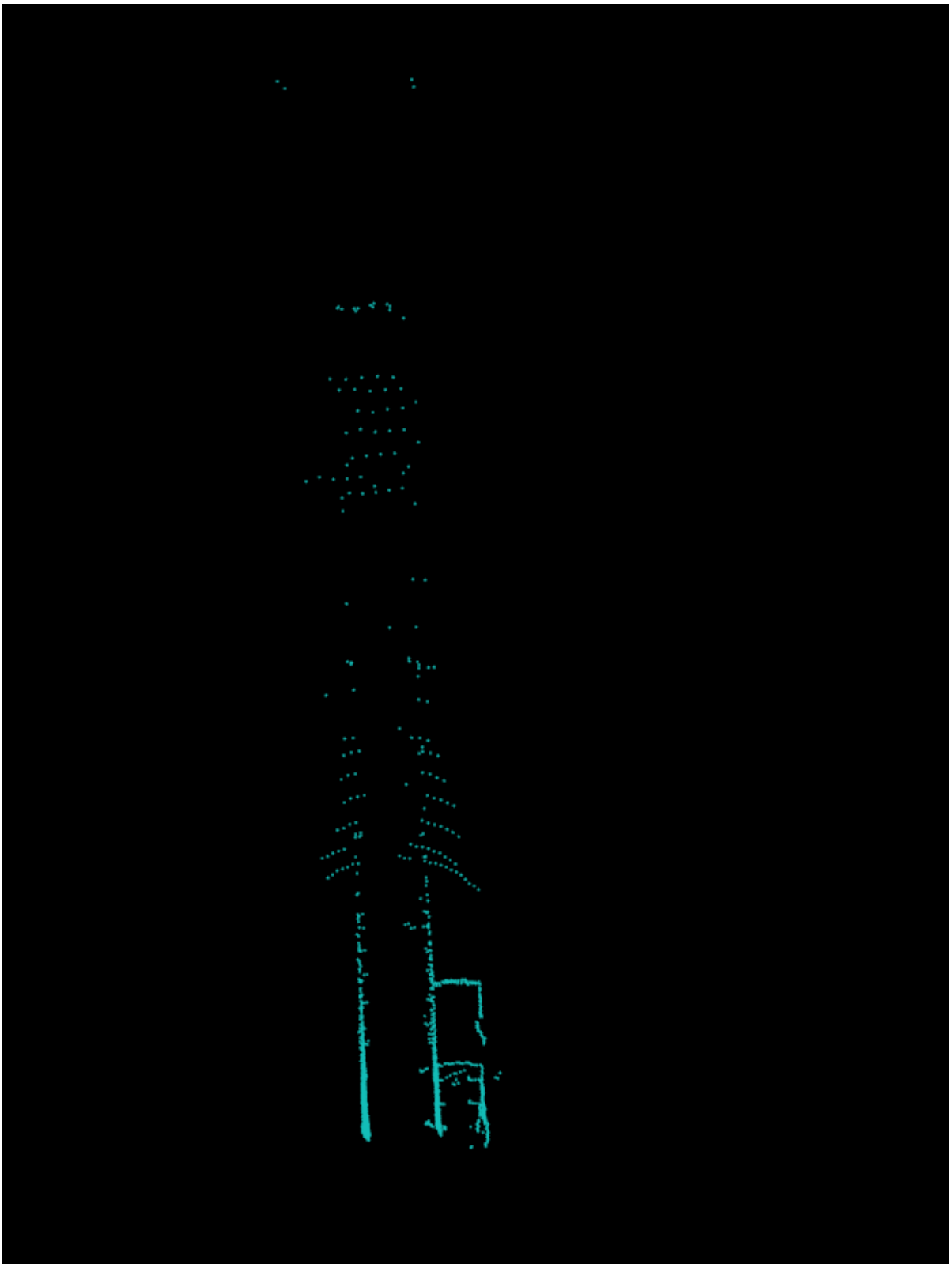
3.2轨迹与点云可视化

对于定位轨迹可视化，使用trajectory_visualization.py脚本可视化结果如下：





进行点云拼接：以0.ply点云作为基坐标系，计算其他帧的转换矩阵，并将目标点云变换后保存。使用matlab读取点云。



3.3误差分析

对于定位误差，其来源可能是：

- 在最近邻匹配时存在误匹配的现象，以及遗漏匹配，这可能与dmax参数的选择也有关，需要进行参数调优。
- 计算误差，尤其是最小二乘使用SVD分解等矩阵计算的误差。
- 相邻帧变换矩阵计算的累计误差
- 可能会存在迭代中的局部最优解的情况

对于建图误差，除上述定位误差的来源外，可能在合并点云数据时存在误差。

解决方法：

- 跨帧匹配，减小累计误差
- 使用特征匹配先进行粗匹配
- 使用牛顿-高斯法求解最小二乘问题等。

由于时间和精力所限，尝试了解决方法但没有很好地实现。