浙江大学

ICP 航位推算作业报告

一 、解决思路

利用 ICP 进行航位推算，解决思路如下：

1．整体思路

（1）读入目的点云 ply_p（即为例程中的 laser_map）和待匹配点云 ply_pp，其中 ply_pp 是 i = 1：9 依次读入的帧，ply_p 会经过点云融合的更新过程

（2）通过上一次得出的机器人总的转移矩阵进行初始化匹配估计，如下图所示

```
%粗匹配
A = robot_tf{i}(1:3,1:3)* pp_points' + robot_tf{i}(1:3,4);
pp_points  =  A';
```

（3）对于待匹配点云 ply_pp，在 ply_p 中依次寻找最近的点，作为待匹配的目标 ply_p_temp

（4）将 ply_pp 和 ply_p_temp 输入 ICP 单帧单步匹配函数（见后），得出本次的转换矩阵，对 ply_pp 用得出的 tform_step 进行更新。若更新后的 ply_pp 和 ply_p_temp 误差小于规定值（设为 0.001）或迭代次数过大（>100），则停止迭代，输出 tform =$\prod_i$ tform_step{i}，作为本帧到 ply_p（即为例程中的 laser_map）的变换矩阵

```
tform_step = icpstep(p_points_temp ,pp_points);
tform = tform * tform_step;

%对输入做变换，以及计算二范数
A = tform_step(1:3,1:3)* pp_points' + tform_step(1:3,4);
pp_points  =  A';
norm_now = norm(pp_points - p_points_temp);
norm1 = abs(norm_before - norm_now);
norm_before = norm_now;

if  (norm1 < 0.001)||(step>100)
    break
end
```

（5）robot_tf{i+1} = robot_tf{i} * tform; 作为机器人累计的变换矩阵

（6）点云融合并更新ply_p：ply_p = pcmerge(ply_p, ply_pp, 0.001);

（7）完成i = 1：9的依次读入和运算融合后，进行输出，此时 ply_p 即为融合点云地图，读取robot_tf{i+1}中的t,即可得出每一帧小车的位置，通过robot_tf{9}的R，可以计算出最后一帧的位姿

2．ICP 单帧单步匹配函数实现

作业中设计了一个单独的函数 icpstep 来完成每一次迭代过程中 icp 的计算，具体步骤如下：

（1） 对 p 和 p′（程序中用 pp 表示）两个点集求质心，并计算去质心坐标

（2）计算 $\mathbf{W} = \sum_{i=1}^{n} \mathbf{q'}_i \mathbf{q}_i^T$

（3）对 W 进行 SVD 分解，[U,S,V] = svd(W);

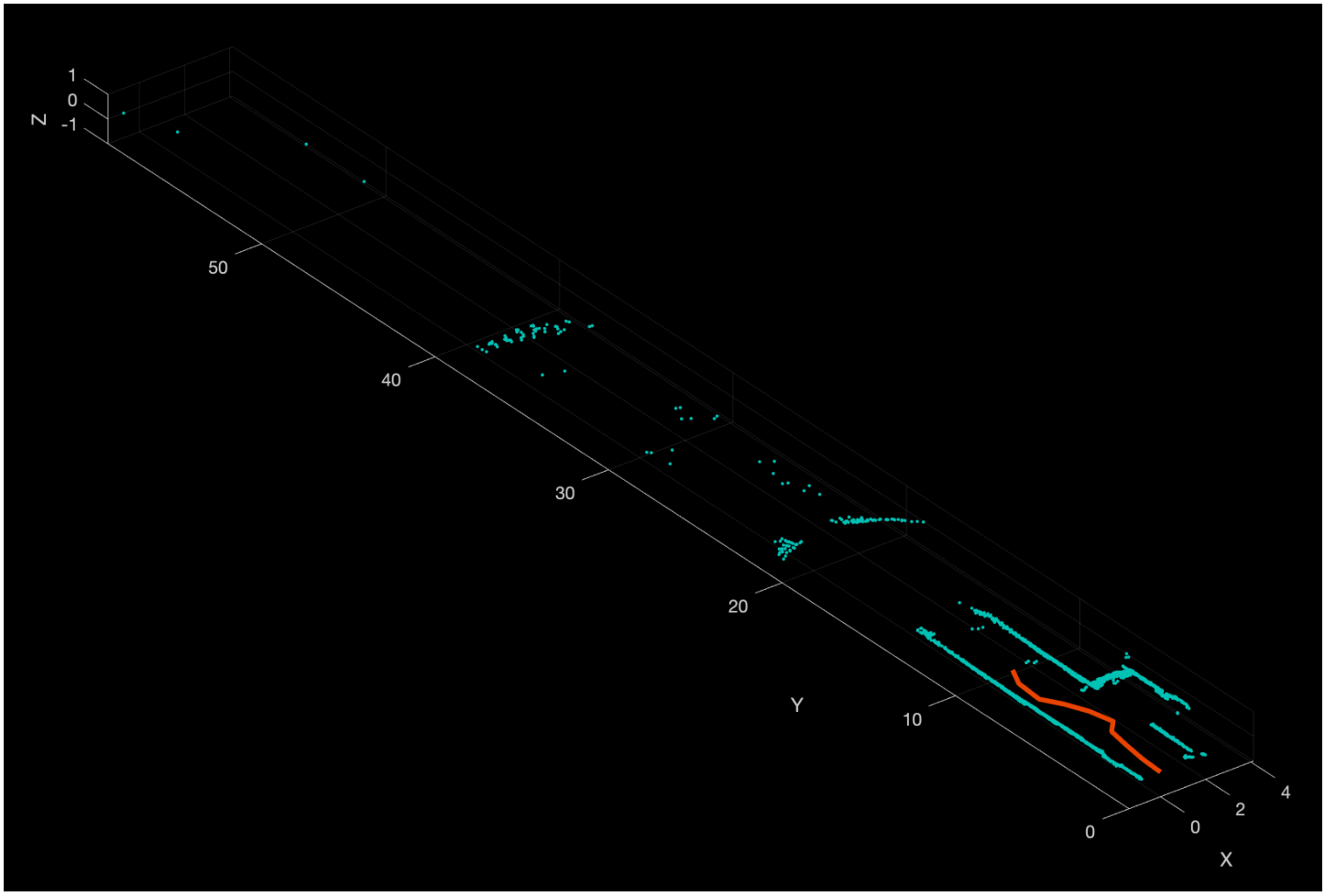
（4）
```
R = V'*U';
t =  p_center' - R * pp_center';
tform = [ R    t
          0 0 0  1];
```

（5）输出tform

二、实验结果

1. 融合点云地图与轨迹

输出结果如下图所示，图中蓝色部分为融合点云地图，红色为轨迹



2. 最后一帧位姿

运行结果截图如右图所示，
其中θ以角度制表示

```
>> myicp_final
最后一帧位姿为:

x_terminal =

    0.2431


y_terminal =

    8.8385


theta_terminal =

   -3.6442
```

三、实验分析

1.旋转矩阵和自带函数相比
以 1.ply 匹配 0.ply 为例，
自带函数输出的变换矩阵为：

```
 0.9995    0.0314         0    0.0500
-0.0314    0.9995         0    1.0925
      0         0    1.0000         0
      0         0         0    1.0000
```

作业中输出的变换矩阵为：

```
 0.9995    0.0316         0    0.0567
-0.0316    0.9995         0    1.1026
      0         0    1.0000         0
      0         0         0    1.0000
```

误差处于可接受范围内

2.最终位姿自带函数与作业中函数比较

```
真值

x_terminal =

    0.1687


y_terminal =

    8.8705


theta_terminal =

   -3.4943
```

```
>> myicp_final
最后一帧位姿为:

x_terminal =

    0.2431


y_terminal =

    8.8385


theta_terminal =

   -3.6442
```

误差：
Error_x = 0.0744 （由于 x 最终坐标数量级较小，故计算绝对误差）
Error_y = 0.36 %
Error_theta = 4.29%

由于 x 数量级较小，相对误差受影响较大，此处计绝对误差。可以看到在误差允许范围内，作业中的算法已实现较好的航位推算。

四、问题与改进

遇到的问题：刚开始代码运行速度较慢，要近 6~7 秒钟才能完成 ICP
解决办法：经过查阅资料，发现 MATLAB 擅长处理矩阵运算，不擅长处理 for 循环，将 for 循环改为矩阵运算将显著提高运算速度。使用 tic 和 toc 进行计时，对于耗时最长的一步：

在使用 for 循环时，一次运行就需要大约 0.06s，该段程序运行 100+次，所需时间将很长（本以为内置函数将较为节省时间，但应该是 findNearestNeighbors 函数内自带了 for 循环，导致 for 循环套 for 循环，严重减慢速度）

```
tic;
p_points_temp = zeros(180,3);
for j =1:180
index = findNearestNeighbors(ply_p,pp_points(j,:),1);
p_points_temp(j,:) = p_points(index,:);
end
toc;
```

```
历时 0.060551 秒。
历时 0.060706 秒。
历时 0.057044 秒。
历时 0.057611 秒。
历时 0.059219 秒。
历时 0.057922 秒。
历时 0.057549 秒。
```

将 for 循环重写为矩阵运算，一次运算不到 0.003 秒，大大加快了运算速度。

```
tic;
p_points_temp = zeros(180,3);
for j =1:180
dis=(p_points-pp_points(j,:)).^2;
distance =dis(:,1) + dis(:,2) + dis(:,3);
[M,index] = min(distance);
%index = findNearestNeighbors(ply_p,pp_points(j,:),1);
p_points_temp(j,:) = p_points(index,:);
end

toc;
```

```
历时 0.002591 秒。
历时 0.002729 秒。
历时 0.002780 秒。
历时 0.002673 秒。
历时 0.002806 秒。
历时 0.002842 秒。
历时 0.002524 秒。
```