ICP 航位推算

刘玉彬

9 November 2020

Contents

1 问题介绍

给定 10 帧点云，利用 ICP 的方法估计出机器人在这 10 帧的轨迹，并实现局部点云地图的生成。实现计算机语言不限（推荐 MATLAB），ICP 核心算法必须自己实现。要求最终提交包含关键步骤注释的源代码和实验报告，实验报告中需要包含解决思路、实验结果、实验分析。数据中包含 10 个 PLY 点云文件，文件名为帧的 ID，可自行转换点云格式。

机器　起点坐标为 $(x, y, \theta) = [0, 0, 0]$，及初始帧 $0.ply$ 已给定，等同于机器　起始坐标与世界坐标系原点重合。用 ICP 计算位姿变换，并累计得到这 10 帧位姿轨迹。轨迹的真值可以参考 MATLAB 中 *pcregistericp* 函数的结果，测试脚本已包含在数据文件中。得到机器人每帧位姿后，将点云对应到各自位姿，得到所有点云融合后的局部点云地图。报告中要求画出机器人轨迹曲线，给出最后一帧机器人的位姿。

考虑到工具的熟练程度，本次实验利用 *python* 语言实现，并最终利用 *matplotlib* 库完成了最终点云图与机器人轨迹的绘制。[1]报告简述了 matlab 中 *pcregistericp* 函数的计算流程并与自己的算法做了对比，提出了改进的想法，但由于时间限制，未付诸实践。

[1]注意到本次实验中 $z = 0$，因此在程序中计算均在二维欧式空间下完成。但应当指出，很容易将这一情况推广到三维状态。

2 ICP 算法简介

ICP(Iterative Closest Point) 算法用于估计两个给定集合点的位姿关系。即给定两组 3D 点：

$$P = \{p_1, p_2, ..., p_n\} \quad P' = \{p'_1, p'_2, ..., p'_n\}$$

现在，希望寻找一个欧式变换，对匹配好的点对，有：

$$\forall i, p_i = Rp'_i + t \tag{1}$$

我们可以使用 SVD 或非线性优化的方法实现求解，算法如 Algorithm1所示，这里简述 SVD 方法的推导过程。

Algorithm 1: ICP Algorithm

Input: Pointsets: P, P'

Output: Rotate matrix R and translate vector t

R.init(); t.init();

while *True* **do**

  $p = \text{Center}(P)$; $p' = \text{Center}(P')$;

  Match(P, P')

  Optimize:

$$R^* = arg\min_R \frac{1}{2}\sum_{i=1}^{n} ||q_i - Rq'_i||^2$$

  $t^* = p - R^*p'$;

  Renew(P', R, t);

  if *Distance(p, p') < request* **then**

    break;

如算法中所示，对点集做去质心运算后，我们只需先计算旋转量再计算平移量即可。展开关于 R 的误差项，我们有：

$$\frac{1}{2}\sum_{i=1}^{n} ||q_i - Rq'_i||^2 = \frac{1}{2}\sum_{i=1}^{n}(q_i^T q_i + q_i'^T R^T R q'_i - 2q_i^T R q'_i) \tag{2}$$

注意到 $R^T R = 1$，我们去除与 R 的无关项，优化目标函数变为：

$$\sum_{i=1}^{n} -q_i^T R q'_i = \sum_{i=1}^{n} -tr(Rq'_i q_i^T) = -tr(R\sum_{i=1}^{n} q'_i q_i^T) \tag{3}$$

因此，我们定义矩阵 W 并对其做 SVD 分解，有：

$$W = \sum_{i=1}^{n} q_i q_i'^T = USV^T \tag{4}$$

此时，可以解得:

$$R = UV^T \tag{5}$$

将 R 的值代入算法，可以解得 t。之后不断重复迭代算法至质心距离小于设定的距离即可。

3 程序介绍

本次实验的程序主要包括两个部分：function.py 和 experiment.py。后者为实现的主函数，第一个文件为 $Frame$ 类的定义与 ICP 算法各函数的实现，包括 $match$，$special\_value\_eliminate$,$svd\_optimize$ 和实现的函数 $transform$。最后利用 $plot_result.mlx$ 绘制结果的图像并计算机器人的路径。

其中，$match$ 函数使用暴力搜索进行匹配，寻找与另一个集合之间距离 d 最小值的点。其中 d 为欧式距离：

$$d_i = \sqrt{((q_i(x) - q_i'(x))^2 + ((q_i(y) - q_i'(y))^2} \tag{6}$$

根据 matlab 文档中介绍的内容[2]显示，函数采用 $Kd\ Tree$ 方式对搜索算法实现优化，但并没有自己实现成功。在 $special\_value\_eliminate$ 函数中，我们设置了一个阈值来消除匹配后距离过大的误差点。经过对比，这一操作可以显著的提高最终结果的精确性。SVD 分解利用 $numpy$ 内置的函数实现。在最终的函数中，我们使用齐次矩阵 T：

$$T = \begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix} \tag{7}$$

实现更方便地利用矩阵乘法进行迭代。实验结果存于 $data.mat$ 文件，包括每一帧修正后的点云数据与每一帧对应的旋转矩阵和平移向量。

4 实验结果与分析

自己的结果与使用 matlab 内置函数 $pcregistericp$ 的结果分别示于图1与图2。可以看出，实现的效果比较成功。机器人的行走路径由图3给出，图中可见两种计算过程的对比结果。

[2] $https://ww2.mathworks.cn/help/vision/ref/pcregistericp.html?s\_tid=srchtitle$

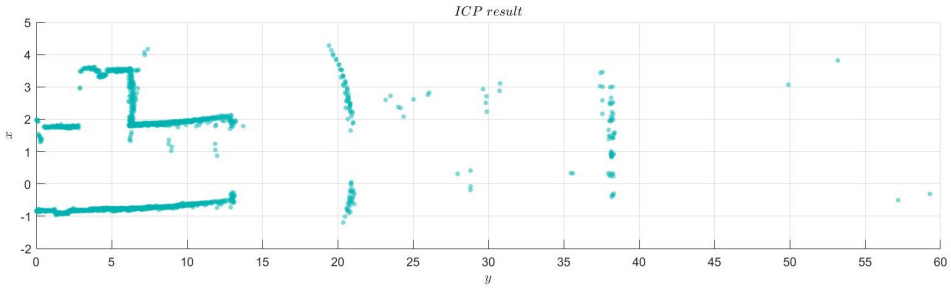

Figure 1: ICP 实验结果

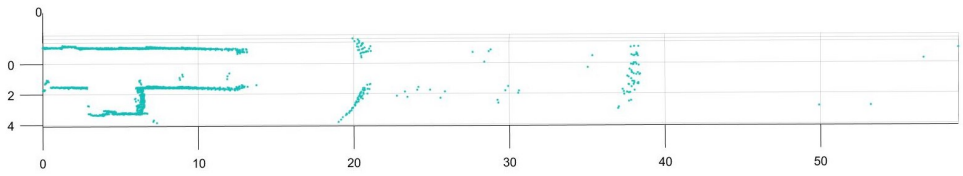

Figure 2: ICP 标准结果（使用 matlab 内置函数）

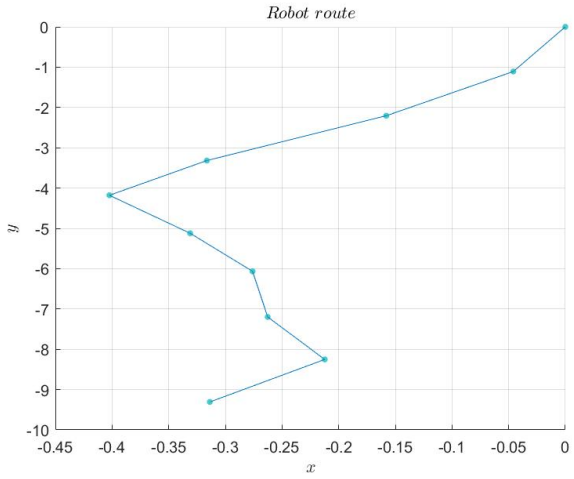

Figure 3: 机器人路径