wheeledrobot 13-16

杨沛山　谢俊　邵可乐　万晨阳

2023 年 7 月 11 日

1　ICP 算法

ICP(Iterative Closest Point), 迭代最近点算法.

本实验主要采用 Point-Point ICP, 该算法主要对于输入的两组激光点云集合, 不失一般性的记为 $P = \{p_1, p_2, ..., p_n\}$ 与 $P' = \{p'_1, p'_2, ..., p'_n\}$, 而我们的目标就是计算这两个点云集合之间的旋转平移量 R, t, 使两者得以匹配并距离误差最小.

其中, 第 i 个匹配点对之间的误差定义如下

$$e_i = p_i - (Rp'_i + t)$$

并构成最小二乘问题, 也就是将上述问题转化为以下的数学形式, 并求使之取到最小值的 R,t

$$\min \quad \frac{1}{2}\sum_{i=1}^{n} ||p_i - (Rp'_i + t)||^2\}$$

1.1　数学推导

1. 定义两组点集的质心位置分别为 p, p'

$$p = \frac{1}{n}\sum_{i=1}^{n} p_i \quad p' = \frac{1}{n}\sum_{i=1}^{n} p'_i$$

2. 计算每个点的去质心坐标

$$q_i = p_i - p \quad q'_i = p'_i - p'$$

3. 计算旋转矩阵

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

4. 根据获得的 R^* 计算 t

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

1.2 R,t 分离说明

不难发现上文中计算 R,t 的过程中, 将两者分开为两步计算, 而最开始的式子中应该是与两者的参数都有关的, 而我们能这样计算的原因主要是在将原式展开打开括号并进行变换之后, 我们可以将目标函数简化为

$$\min \quad \{\frac{1}{2}\sum_{i=1}^{n}||(p_i - p - R(p_i' - p'))||^2 + ||p - Rp' - t||^2\}$$

其中左侧只与旋转矩阵 R 有关, 而右侧既有 R 又有 t, 但是只与点集的质心有关, 所以求解时可以分解为两步.

1.3 求 R

以下的 q 为去质心坐标.

$$\frac{1}{2}\sum_{i=1}^{n}||(p_i - p - R(p_i' - p'))||^2 = \frac{1}{2}\sum_{i=1}^{n}||(q_i - Rq_i')||^2$$

$$= \frac{1}{2}\sum_{i=1}^{n}(q_i^T q_i + {q_i'}^T R^T R q_t' - 2q_i^T R q_i')$$

注意到虽然第二项与第三项中都包含 R, 但是第二项中 $R^T R = I$ 与 R 无关, 所以我们又可以将目标函数简化为

$$\max \quad \sum_{i=1}^{n} q_i^T R q_i'$$

$$\sum_{i=1}^{n} q_i^T R q_i' = \sum_{i=1}^{n} tr(q_i^T R q_i') = \sum_{i=1}^{n} tr(R q_i' q_i^T) = tr(R\sum_{i=1}^{n} q_i^T q_i')$$

其中 tr 表示矩阵的迹 (trace).

定义矩阵 $W = \sum_{i=1}^{n} q_i^T q_i'$

对 W 进行 SVD 分解可得 $W = USV^T$

其中 S 为奇异值组成的对角阵, 且对角线元素从大到小排列.

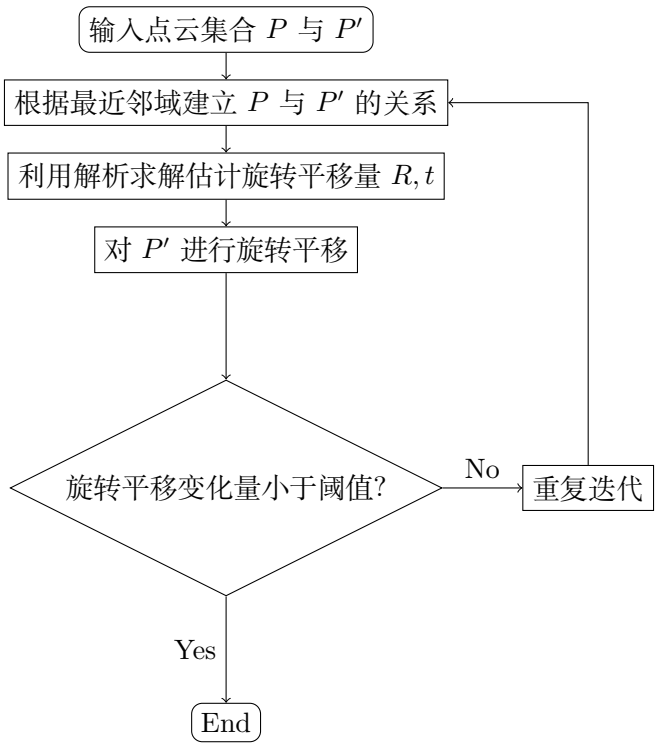
U V 均为酉矩阵, 即 $U^T U = I$

目标函数的上界 $tr(RUSV^T) = tr(SV^T RU) \sim tr(SH)$

$$tr(RUSV^T) = s_1 h_{11} + s_2 h_{22} + s_3 h_{33} \leq s_1 + s_2 + s_3$$

当上述等号成立时可以得到 $R = VU^T$, 并得到 $t = p - Rp'$

1.4 算法思路



2 数据处理

在开始进行实验的时候发现许多输出的值都是 NaN, 而这的原因是输入的激光点云集中本身就含有部分不合法数据, 其可能是激光雷达的信号丢失等问题, 为了解决这一问题, 我们在开始进行计算之前调用 $np.isnan$ 来排除这一因素, 实例如下:

```
tar_pc = tar_pc[:, ~np.isnan(tar_pc).any(axis=0)]

src_pc = src_pc[:, ~np.isnan(src_pc).any(axis=0)]
```

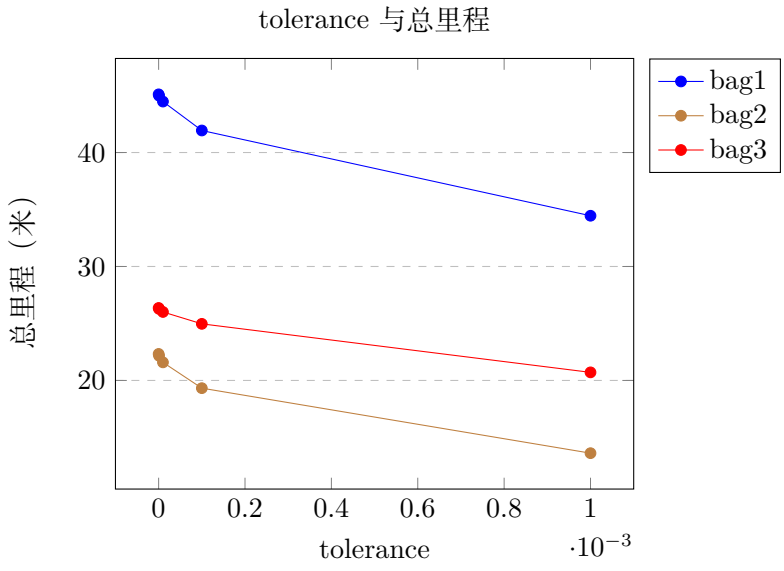
也就是通过该函数判断出非 NaN 数据 (使用 $\sim$ 来实现) 的下标, 并将其重新赋值.

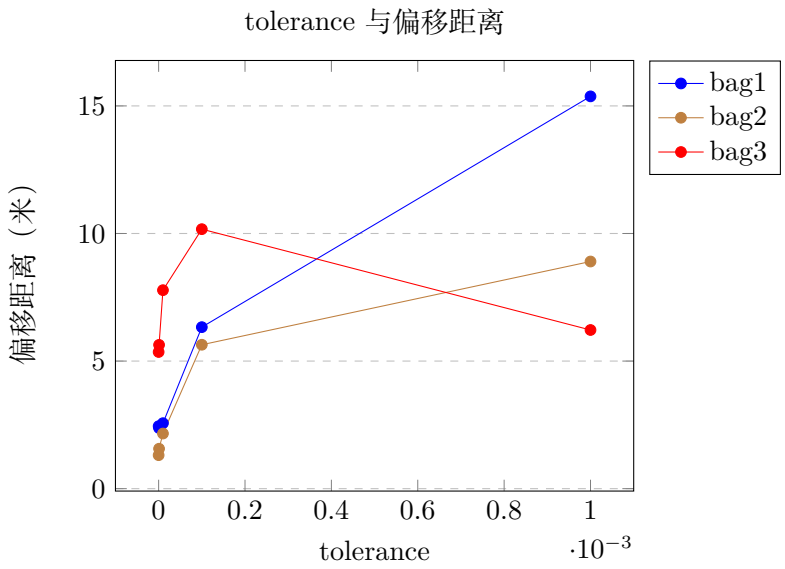
3 参数分析

3.1 tolerance

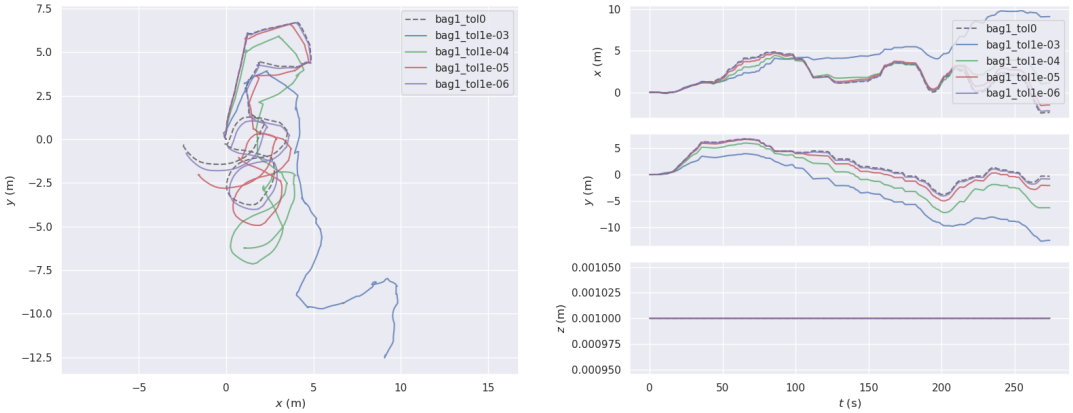
各 bag 总里程（米)、偏移距离和最大迭代次数如下:

tolerance		0	1e-6	1e-5	1e-4	1e-3
bag1	总里程	45.105	44.989	44.472	41.934	34.4529
	偏移距离	2.449	2.376	2.566	6.33	15.374
	最大迭代次数	50	44	36	19	8
bag2	总里程	22.327	22.166	21.587	19.323	13.618
	偏移距离	1.312	1.565	2.16	5.64	8.902
	最大迭代次数	50	50	29	17	7
bag3	总里程	26.36	26.274	26.009	24.966	20.709
	偏移距离	5.356	5.635	7.779	10.17	6.218
	最大迭代次数	50	40	33	18	7

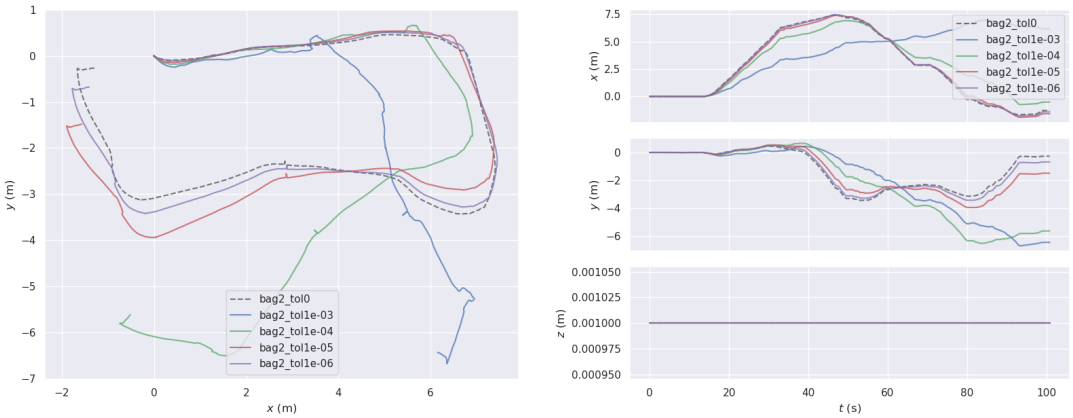




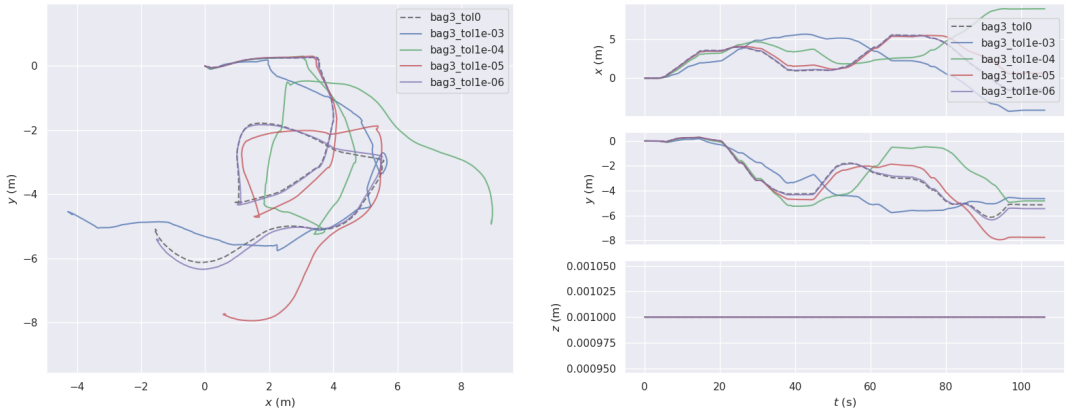
三种数据包的轨迹如下：



bag1 轨迹与 tolerance



bag2 轨迹与 tolerance



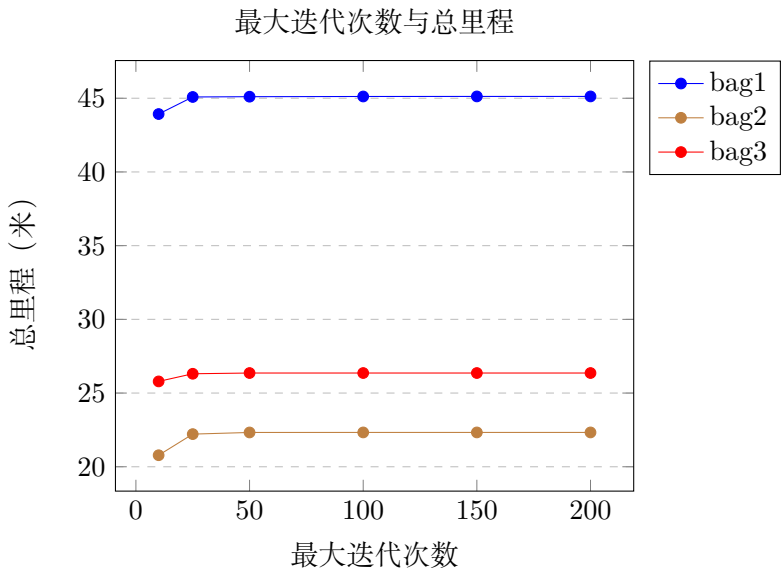
bag3 轨迹与 tolerance

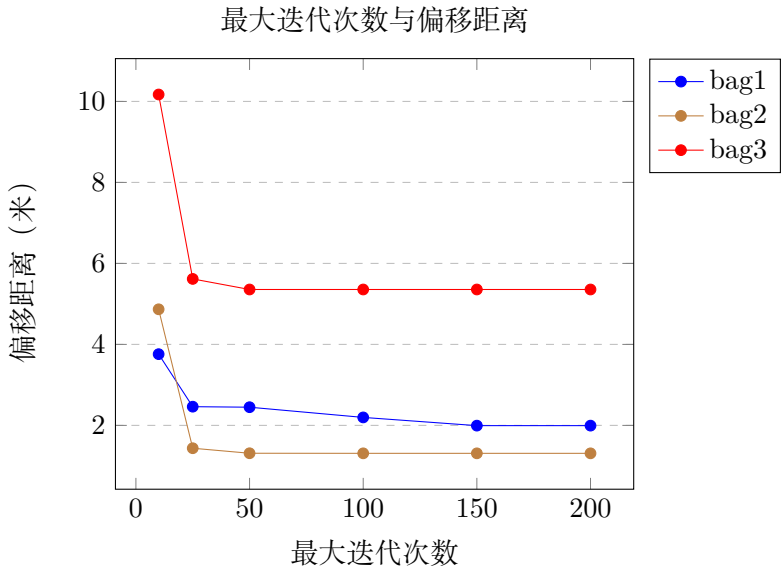
在调整 tolerance 参数的实验中，统一设置 $max\_iter = 50$，经实验测得，在 $tolerance \geq 1e-6$ 条件下，基本无法跑满 $max\_iter = 50$。从上述的实验结果可以看出，随着 $tolerance$ 的减小，max_iter 变大，算法精度越高，总里程与偏移距离逐渐收敛到一个值，也就是说每次求出的变换矩阵逐渐收敛。可以看到在收敛后，偏移距离仍然不为 0，这可能是 ICP 算法的缺陷以及传感器误差导致的。同样观察到当 tolerance 取 1e-3 时 bag3 的偏移距离反而变小了，合理猜测是偏移过度造成的巧合,bag3 轨迹与 tolerance 一图也证明了这一点.

3.2 最大迭代次数

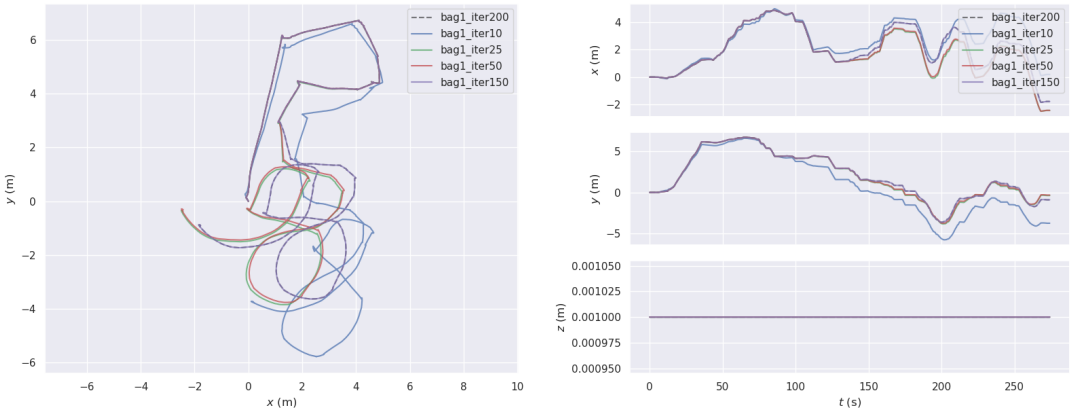
迭代次数与总里程（米）和偏移距离（米）的关系如下：

迭代次数		10	25	50	100	150	200
bag1	总里程	43.928	45.086	45.105	45.119	45.123	45.123
	偏移距离	3.759	2.462	2.449	2.197	1.994	1.994
bag2	总里程	20.782	22.215	22.327	22.329	22.329	22.329
	偏移距离	4.867	1.436	1.312	1.31	1.31	1.31
bag3	总里程	25.79	26.309	26.359	26.359	26.359	26.359
	偏移距离	10.171	5.617	5.356	5.356	5.356	5.356

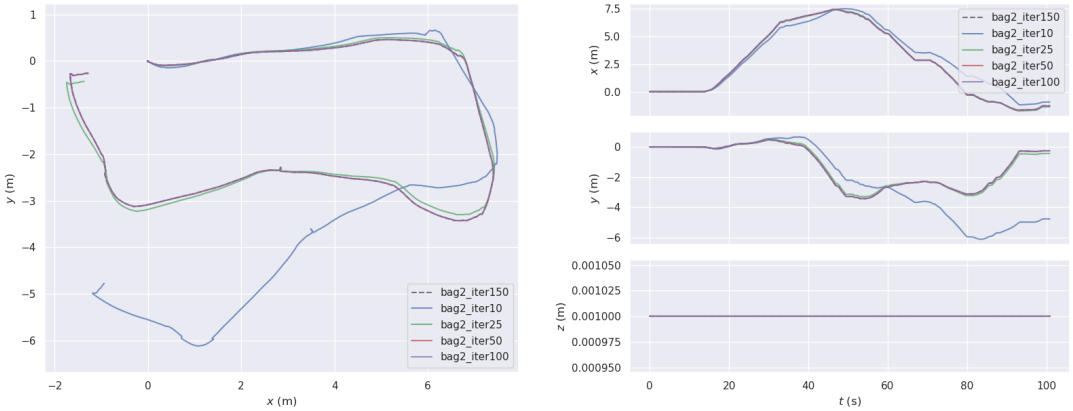

最大迭代次数与总里程
总里程（米）
45
40
35
30
25
20
0
50
100
150
200
最大迭代次数
bag1
bag2
bag3


最大迭代次数与偏移距离
偏移距离（米）
10
8
6
4
2
0
50
100
150
200
最大迭代次数
bag1
bag2
bag3

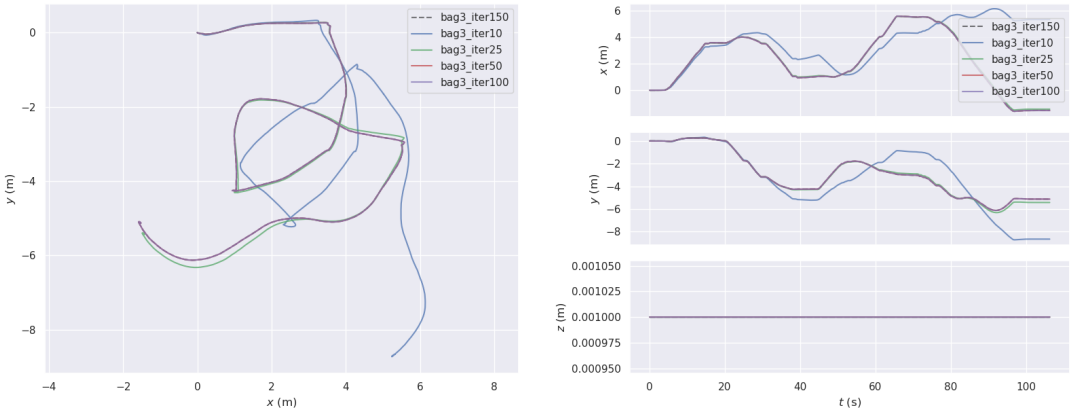
三种数据包的轨迹如下：



bag1 轨迹与迭代次数



bag2 轨迹与迭代次数



bag3 轨迹与迭代次数

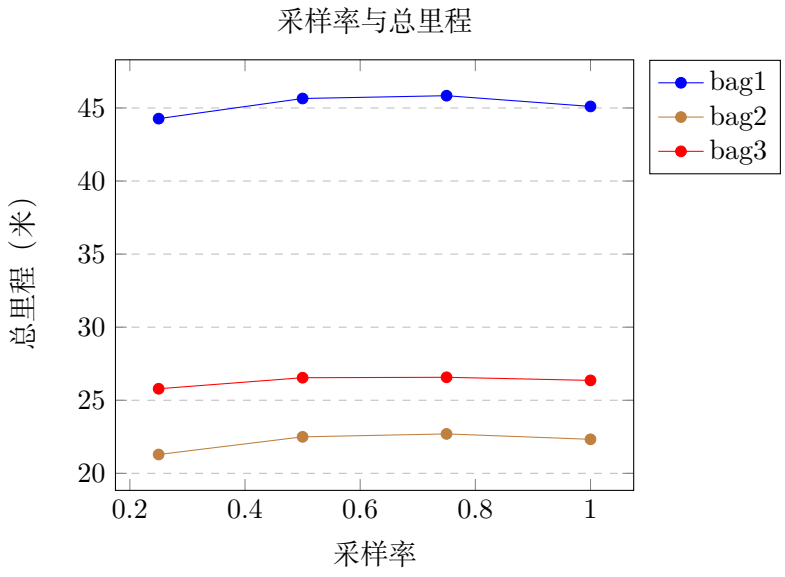
在迭代次数的实验中，统一设置 `tolerance=0`，因此实际迭代次数即为最大迭代次数。从上述的实验结果可以看出，随着最大迭代次数增大，偏移距离逐渐减小，总里程与偏移距离逐渐收敛到一个值，也就是说每次求出的变换矩阵逐渐收敛。对于 bag1，当迭代次数到达 150 时收敛。对于

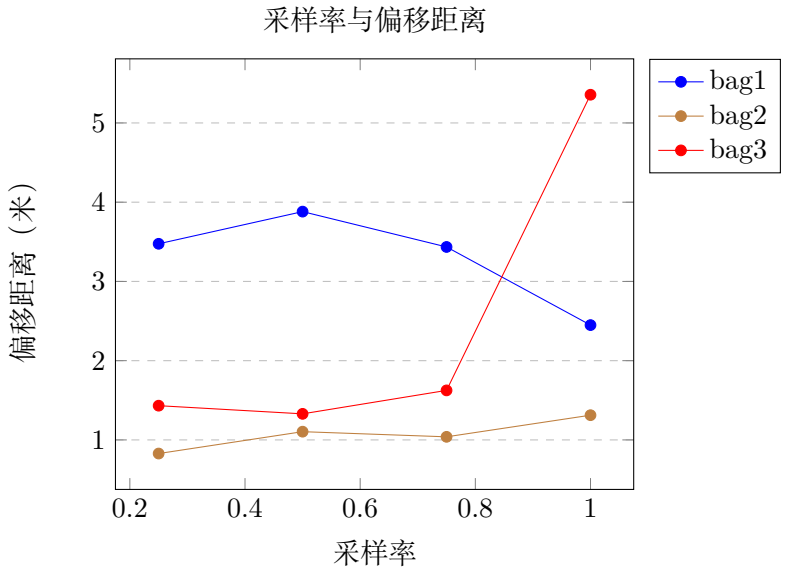
bag2、bag3，由于行程较为简单，因此在迭代 50 次后就已经收敛。可以看到在收敛后，偏移距离仍然不为 0，这可能是 ICP 算法的缺陷以及传感器误差导致的。

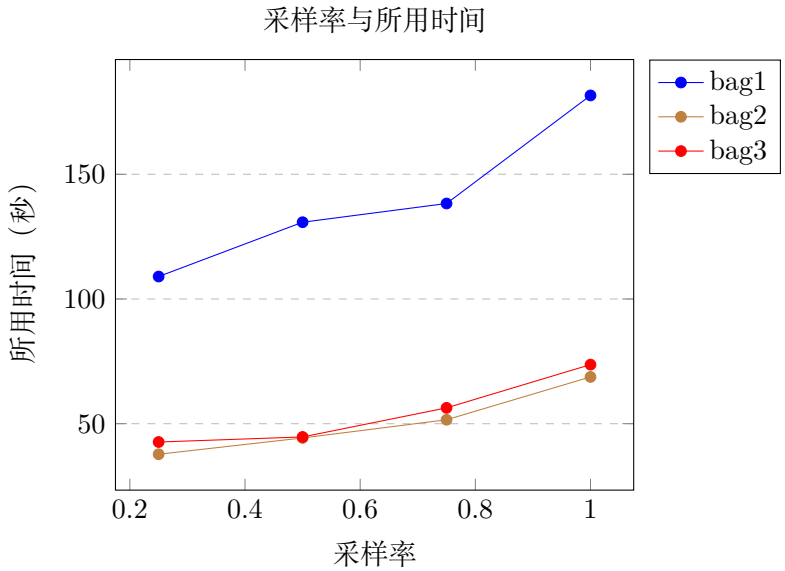
3.3 采样率

采样率与总里程（米）、偏移距离（米）、ICP 算法所用时间（秒）关系如下：

采样率		0.25	0.5	0.75	1
bag1	总里程	44.269	45.647	45.839	45.105
	偏移距离	3.475	3.881	3.435	2.449
	总时间	109.004	130.779	138.283	181.614
bag2	总里程	21.288	22.497	22.697	22.327
	偏移距离	0.828	1.104	1.039	1.312
	总时间	37.753	44.286	51.575	68.744
bag3	总里程	25.787	26.542	26.571	26.359
	偏移距离	1.432	1.329	1.625	5.356
	总时间	42.654	44.689	56.351	73.685







在这部分实验中，我们控制迭代次数为 50 次。采样率为每次迭代时选取的点的数量与点云总数的比值。从上述实验结果可以看出，随着采样率的增大，ICP 算法所用的时间逐渐增大，这是因为算法的复杂度与点的数量相关。然而，采样率与偏移距离、总里程没有显著的相关关系，且轨迹差距不大，我们推测出现这种情况的原因是随机采样可能会丢弃一些不完美的点云匹配，使得 ICP 算法在这种情况下选择了更好的点云匹配，从而导致了结果的不相关性。总的来说，适当地降低采样率，能够让我们的算法在具有更高实时性的情况下保持一定的精度。

3.4 ICP 算法实时性

我们发现，icp 算法中的热点在于 findNearest() 函数，我们主要通过使用 scipy.spatial 库中的 ckdtree 数据结构，来优化二重循环寻找最近点的过程. 通过在 icp.py 中设置与 icp.cpp 相同位置的计时点，即一次 process 前后，我们计算出了两者平均一次 process 的时间. 我们惊奇的发现，优化

后的 python 代码实时性远高于原 c++ 代码.

	py use ckdtree	c++
平均时间 (s)	0.0165	0.0394

4 实验心得

在之前的实验中, 我们主要注重于如何在一个现有的地图中使小车从一个地方前往一个目的地, 避开了对于环境本身勘察与定位的要求. 然而在现实的使用场景中, 我们对于地图可能是未知的, 即使是已知的地图, 也可能因为实物的部分偏差与误差积累, 使得当前位置出现较大的偏差.

由此, 我们需要通过诸如 ICP 这样的定位匹配方式, 基于已知的位置来推算出当前真实的位置, 并减少误差积累带来的影响.

而在实验过程中, 我们也接触到了许多问题, 其中无效数据 (即点云中的 NaN 数据) 对于我们的算法实践是比较新颖的, 而我们也用了特定的方式对其进行了处理 (可见上文中数据处理). 并且在使用代码进行实验的时候, 我们也关注到了许多参数 (诸如 Max_iter, 采样率和 tolerance 等) 对于代码效果的影响, 并通过控制变量的方法获取了较好的组合, 这样的探究过程给我们带来了许多实验上的助力.