实验四 ICP感知 实验报告

第九组 章帆 黄永全 陈浩 米博宇

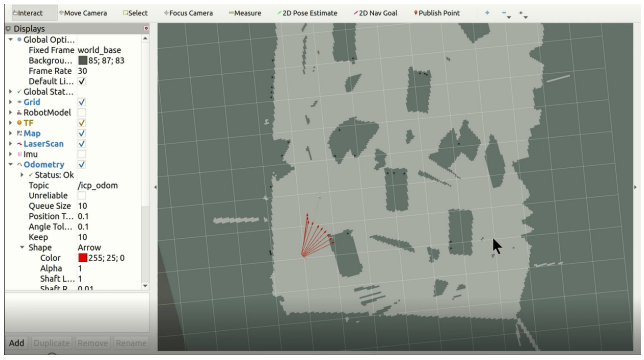
Level 1：基本实验结果

实验流程： 完成`icp.py`中缺少的计算最近点和生成变换矩阵的函数，运行`icp.launch`文件，在另一个命令行窗口中发送数据包文件，rviz窗口下观察定位和移动效果。
遇到的问题和解决方式：

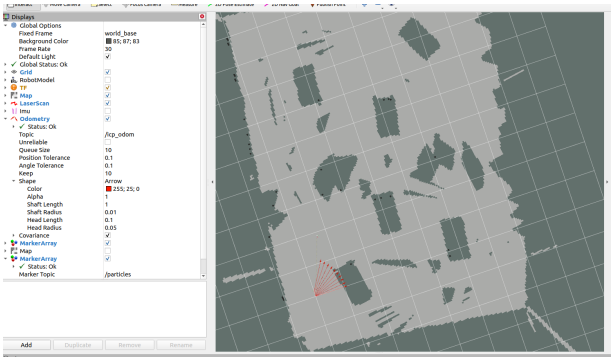
1. 数据包中，机器人周围环境与机器人的距离依靠激光测算得到。而若机器人与墙体距离太远，没有在一定时间内接收到反射的激光信号，则该位置的坐标将被记录为`NaN`，后续计算中将引发各种运算异常，从而中止程序。
   解决方案：在计算变换矩阵前，先对输入的source和destination矩阵进行预处理，调用`numpy`的`isnan`函数探测矩阵中的`NaN`元素并将其修改为0，即可正常进行后续运算。
2. c++代码中最近点对的计算方法为二重循环，而由于距离采样点过多，python中二重循环的计算速度太慢，这是程序无法快速运行，达到实时定位的效果的主要原因。
   解决方案：最初采用每10个点采样一次的方式提高速度，在`Level 4`部分调用了`sklearn`包中的计算近邻点的函数以达到接近于c++的运算速度。
3. 最初仿真过程中无法看见红色箭头，且roslaunch界面没有报错，无法确认全过程中哪一部分出错。
   解决方案：在试运行c++代码后确认了程序的其他部分没有问题，最后发现为最简单的icp算法运行时间过长，未能完成箭头产生所需的计算，而后在程序的迭代部分加入了print的显示部分，证明了是程序运行速度的问题。

实验结果：实验当天通过验收，与标准c++代码运行效果对比如下。

标准结果



运行结果



Level 2：程序框架流程

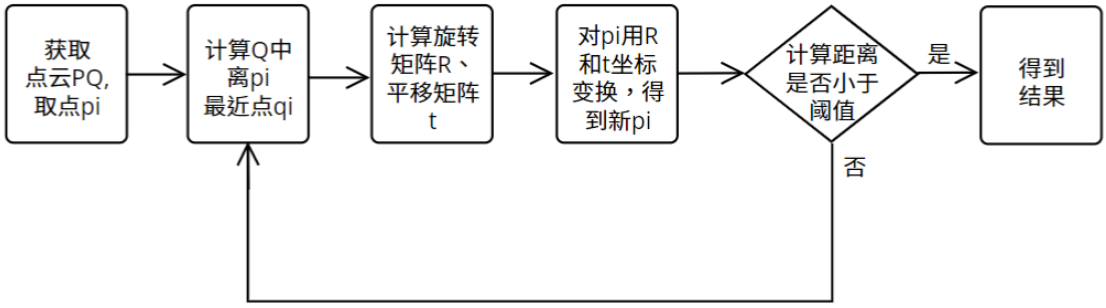
数学推导部分

1. 点云配准过程就是根据输入的点云P和Q，计算两组数据之间的旋转量和平移量R,t，使得P中点经过$RP+t$的刚性变换后与已有的Q形成最佳匹配，即两组数据的距离误差最小。在此，第i个匹配对点的误差为$e_i=p_i-(Rq_i+t)$，因而问题转化为求R,t，使得$min\frac{1}{2}\sum_{i=1}^{n}||e_i||^2$。
2. 利用线性代数的求解方法，我们对两个点云进行去质心处理（质心定义为p,q），最终将目标函数简化为$min\frac{1}{2}\sum_{i=1}^{n}||p_i-p-R(qi-q))||^2+||p-Rq-t||$。由于上式中两个项无关，因而后续需要计算的内容转变为$R^*=argmin\frac{1}{2}\sum_{i=1}^{n}||(p_i-p)-R(q_i-q)||$，计算得到$R$后，可得$t$计算表达式$t^*=p=R^*q$。

3. 而后首先求解R，优化目标函数为$max \sum_{i=1}^{n}(p_i-p)^T R(q_i-q)$，此处定义矩阵$W=\sum_{i=1}^{n}(q_i-q)(p_i-p)^T$，对$W$进行SVD分解，可得$W=USV^T$。
4. 当$tr(RUSV^T)=s_1+s_2+s_3$时，$R=VU^T$，$t=p-Rq$，该值即目标解，因而迭代计算求得最值即可。

算法执行流程

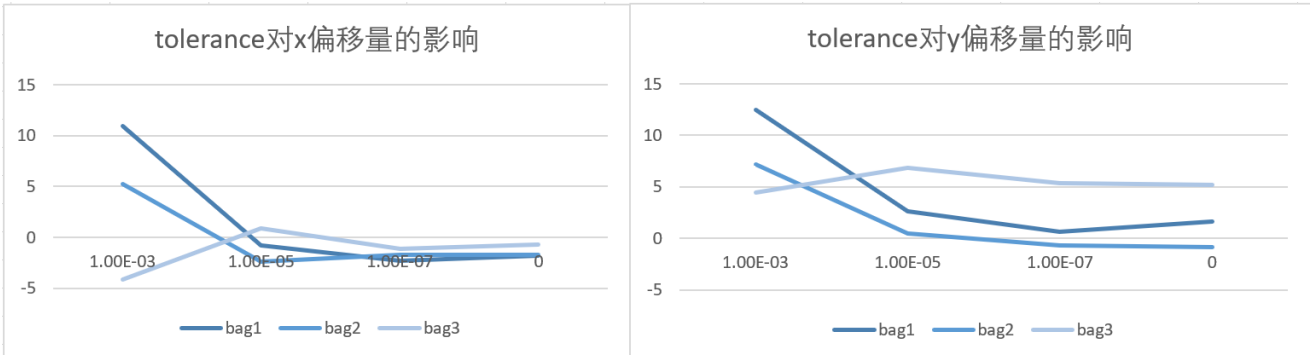
1. 在P中去一点p_i，找到Q中的最近点q_i，p_i和q_i间构成一个点对，两者间存在欧式变换关系$q_i=Rp_i+t$。
2. 根据1中的对应点匹配，我们取n对点对，计算得到当前的R和t，并计算当前情况下的误差。
3. 我们得到了一个新的R与t，导致了一些点转换后的位置发生变化，一些最邻近点对也相应的发生了变化。因此，我们又回到了1中的寻找最邻近点。第1 2步骤不停迭代进行，直到误差小于阈值这一条件后离开。



Level 3：分析ICP算法参数的影响

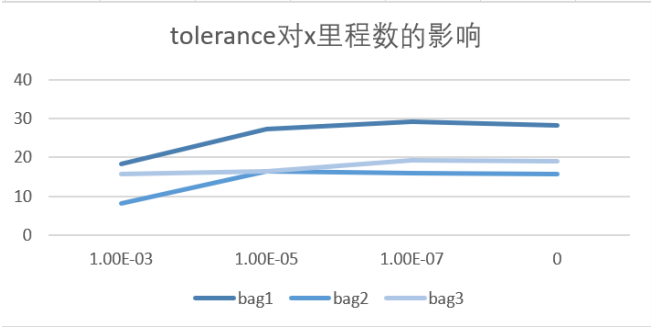
在icp.cpp中有tolerance参数，其意义是用于判断误差是否足够小以至可以退出迭代的阈值，下面我们分析了tolerance参数对程序运行结果的影响（为了保证结果的准确性，我们运行了icp.cpp进行分析）。
在这一过程中，我们比较算法精确度的指标有两个，偏移量为rosbag起始点与终点间的距离偏差；总里程数为完成仿真后记录得到的小车行驶里程。仿真结果如下。

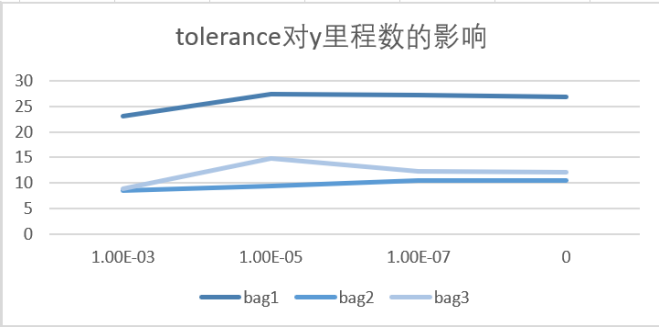
tolerance	**1e-3**	**1e-3**	**1e-5**	**1e-5**	**1e-7**	**1e-7**	**0**	**0**
偏移量	x	y	x	y	x	y	x	y
bag1	10.914	12.498	-0.756	2.597	-2.282	0.621	-1.806	1.655
bag2	5.269	7.169	-2.396	0.465	-1.716	-0.675	-1.742	-0.828
bag3	-4.102	4.456	0.8716	6.819	-1.094	5.354	-0.672	5.230



tolerance	**1e-3**	**1e-3**	**1e-5**	**1e-5**	**1e-7**	**1e-7**	**0**	**0**
总里程数	x	y	x	y	x	y	x	y

tolerance	1e-3	1e-3	1e-5	1e-5	1e-7	1e-7	0	0
bag1	18.259	23.007	27.335	27.341	29.241	27.107	28.313	26.776
bag2	8.225	8.582	16.335	9.4133	15.864	10.491	15.751	10.548
bag3	15.762	8.971	16.335	14.895	19.197	12.241	18.928	12.193

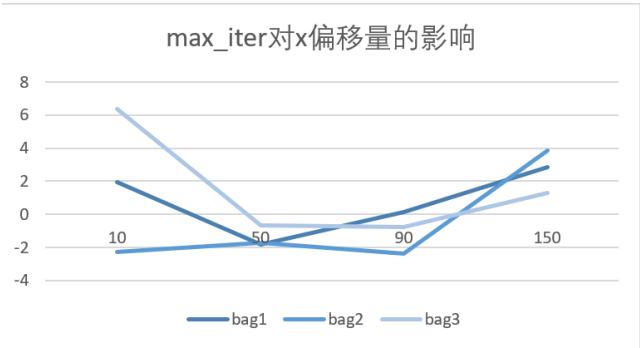


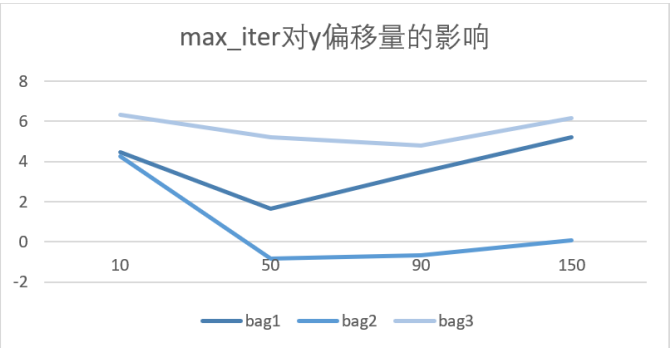


由以上结果可得，随着tolerance的降低，算法精度要求的增高，运算结果的偏移量降低，说明计算结果更加符合真实情况；总里程数逐渐上升，说明计算的里程数逐渐从折线接近真实的曲线情况。

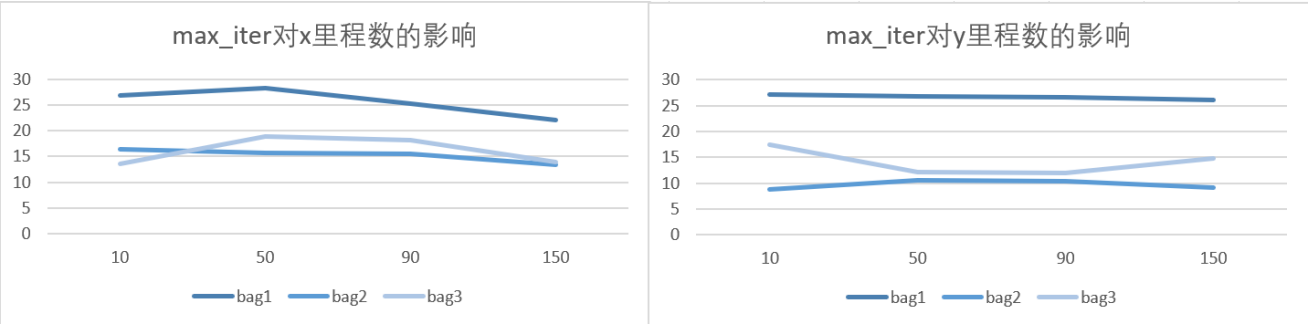
此外，`icp.cpp`中的`max_iter`参数为进行迭代的最大次数，我们同样更改了该参数进行分析（为了保证结果的准确性，我们运行了`icp.cpp`进行分析，比较的指标量同上）。

max_iter	10	10	50	50	90	90	150	150
偏移量	x	y	x	y	x	y	x	y
bag1	1.924	4.478	-1.806	1.655	0.137	3.485	2.843	5.193
bag2	-2.263	4.250	-1.742	-0.828	-2.386	-0.660	3.865	0.091
bag3	6.374	6.330	-0.672	5.230	-0.759	4.787	1.280	6.161





max_iter	10	10	50	50	90	90	150	150
总里程数	x	y	x	y	x	y	x	y
bag1	26.788	27.074	28.313	26.776	25.302	26.574	22.075	26.156
bag2	16.490	8.742	15.751	10.548	15.441	10.377	13.384	9.214
bag3	13.605	17.366	18.928	12.193	18.177	12.047	13.839	14.796

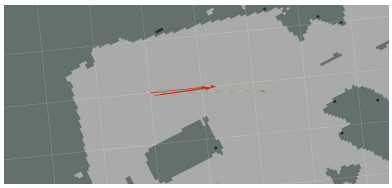
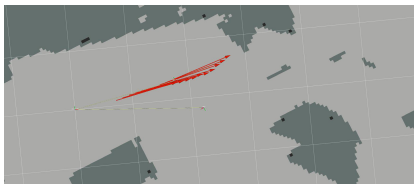


由以上结果可得，若以max_iter=50为基准，max_iter的增加或降低都会使偏移量和里程数发生偏移，迭代次数过少会导致过少的尝试无法得到满足要求的结果，迭代次数过多会导致每次求解时间增加，进而发生偏移，所以迭代次数应更根据精确度和计算速度综合选取最优的大小。

Level 4：ICP的一个算法改进点和数据包结果

在Level 1中，我们发现icp算法中二重循环查找最近点的速度过慢，这是导致算法的速度降低，失去实时性的主要原因。而为此减少采样点的方式有可能导致算法误差增加，为了在保证采样量的同时提高速度，我们采用了sklearn库，利用了neigh.kneighbors函数以快速得到distances和indices两个矩阵，结果表明运行速度大提升快，下面是利用二重循环、使用neigh.kneighbors加速和c++在相同仿真条件下运行的结果。

```
def findNearest(self,src, dst):
    Neigh = NeighBor()
    assert src.shape == dst.shape
    neigh = NearestNeighbors(n_neighbors=1)
    neigh.fit(dst)
    distances, indices = neigh.kneighbors(src, return_distance=True)
    dis = distances.ravel().tolist()
    ind = indices.ravel()
    for i in range(len(dis)):
        if dis[i] < self.dis_th:
            Neigh.distances.append(dis[i])
            Neigh.src_indices.append(i)
            Neigh.tar_indices.append(ind[i])
    return Neigh
```

python二重循环	neigh.kneighbors加速	c++
		

可见相同情况下，使用neigh.kneighbors进行加速的python和c++有相近的运行速度，而若不进行加速会有非常明显的误差。

心得感想：在上一次课中，我们已经有了关于如何让机器人跑起来，并且如何在程序中指定机器人的运动轨迹的经验，但是如何得知机器人现在处于场景中的位置成了一个至关重要的问题。此次实验编写并运行了ICP算法，成功的得知了机器人现在的位置。在这一过程中，我们体会到了调试算法的不易与艰辛，但是同时我们也收获了一次宝贵的经验。

在这次实验中，我们了解了点对点的ICP算法的流程和具体实现，对该算法有了深入的认识，也体会了采用rosbag进行仿真的实验流程。此外，对于算法速度的优化、学习新的库以及解决程序运行中出现的各种问题也让我们的学习和实践能力有了显著的提升。