ICP 定位

周炜 杨帅 章皓然 沈小康

**Part 1 程序框架**

**ICP（Iterative Closest Points）**算法包含两个方面：对应点搜索和位姿求解，目的是寻求点集之间的匹配关系，求解的结果是两点集之间的平移及旋转量。在实验中，需要先对点云数据集进行**预处理**，剔除**nan[[1]](#footnote-1)**类型的数据，将3维点云数据集转换为2维

实现**ICP算法**后使用**bag**的点云数据进行测试，再将代码部署至实物，使用激光传感器数据代替激光数据包，测试小车运动轨迹与**ICP**估计轨迹重合程度。在该程序中通过里程估计实现定位，通过接受激光数据，计算**ICP**获取小车定位并发布至TF

|  |  |
| --- | --- |
|  |  |

图1 程序运行流程和ICP算法

**Part 2 ICP算法优化**

在程序运行时，由于程序运行速度较慢，复杂度较高，导致无法对每一帧数据及时输入程序中进行处理，因此可以采用降低程序复杂度或减少输入数据量的方式改进算法

**采用kd tree 代替二重循环**

原算法最近邻匹配时采用**暴力搜索**的方式，可以改为**kd tree**的方式, 对点的匹配复杂度由降低到了。**kd tree**，对x和y两个维度(或者更高维度)做划分，在进行类似于二分查找一样的操作。一旦**kd tree**被建立，就可以利用其做最近邻查询，即查找距离给定点最近的k个节点

**点云关键点检测**

点云的关键点值能够代表点云关键信息的点，当待匹配的点云较大时，通常先提取它们的关键点再进行匹配，以减少匹配数量。我们小组使用了**ISS关键点算法**(调用了库**Open3d**中的**compute\_iss\_keypoints()**函数)，伪代码和采用该方法优化后得效果如图2[[2]](#footnote-2)

|  |  |
| --- | --- |
|  |  |

图2 ISS算法伪代码和使用ISS算法后运行时间的变化

**使用优化的计算函数来提高效率**

比如说**scipy.distance\_matrix(array x, array y)**就可以加快点云之间距离的计算

**Part 3 ICP 算法定量分析**

ICP算法的重要考核指标是**精度**与**实时性**。当算法比较慢的时候，可以考虑减少数据量（这里分析复杂度）、改进算法或者不要测量每一帧的位移，而是跳过几帧测量位移。但是总的来说，处理的频率应当不小于雷达的扫描频率

|  |  |
| --- | --- |
|  |  |

图3 尚未调参时的运行参数曲线图

**迭代误差条件**

|  |  |
| --- | --- |
| 当相邻两次迭代的误差小于**tolerance**时，迭代会停止；减小**tolerance**可以让定位的**精度上升**，同时由于每次迭代的耗时增长导致**算法延迟增加**。但是实际上，如下图，在迭代了约30次之后，**loss**梯度非常小，需要把tolerance取得非常小才会有作用，但是实际上经过测试，**tolerance**并没有起到十分大的作用，因此可以直接设置**tolerance = 0** | 图4 loss与iter关系图 |

**减小数据量**

可以采用运行跳帧的方法，对应**ICP.launch**中的参数**robot\_skip**，最后选择了2，即**mod2**， 这意味着只使用1/2的数据。减少数据会损失精度，但是能提高运行速度

还可以采用跳数据进行匹配的方法，对应**ICP.launch**中的参数**point\_skip**，跳过部分采样点可以小幅提升运行速度，但会损失大量精度，权衡后我们最后选择了1，即100%保留数据不跳过

|  |  |
| --- | --- |
|  |  |

图5 60次迭代下100%的数据量

|  |  |
| --- | --- |
|  |  |

图6 60次迭代下75%的数据量

通过对比图3、图5、图6，我们可以发现，使用75%数据量可以有效提升运行速度，提高实时性，代价是精度略微减小，但仍处于可接受范围。因此整体性能提升，与预期结果相符

**最大循环**

对应**ICP.launch**中的参数**max\_iter**，最后选择了70次

当**tolerance**无法达到时，若迭代次数达到了**max\_iter**，迭代会停止

增大**max\_iter**会导致：定位的精度上升；每次迭代的耗时增长，进而导致算法延迟较高，当是由于又存在实时性的要求，因此存在tradeoff

|  |  |
| --- | --- |
|  |  |

图7 70次迭代下100%的数据量，但跳帧mod2

|  |  |
| --- | --- |
|  |  |

图8 30次迭代下75%的数据量

**最近邻距离 dis\_th**

在launch文件参数中，**dis\_th**描述用于筛选匹配点的距离阈值。变换后前后两帧点误差不超过**dis\_th**即认为匹配成功。由于实测中前后两帧变换较小、算法准确性较高，**dis\_th**的值在一定范围内对算法影响较小。在上文的参数中，若未特别说明，均默认**dis\_th**取值为0.05。下图展示了**dis\_th**参数对算法精度与性能的影响

|  |  |
| --- | --- |
|  |  |

图9 30次迭代下，75%数据量，dis\_th=0.005

|  |  |
| --- | --- |
|  |  |

图10 30次迭代下，75%数据量，dis\_th=2.0

因为运行环境不一致，报告中指标与指标间的代码效果会与其他部分不同，但是我们在单一指标的测试中保证各个参数都在同一台机器上进行测试，排除了运行环境的干扰。由于算法整体运行速度较快，因此增大dis\_th只会略微改善算法的运行速度，但是减小**dis\_th**可以显著增加算法的精度。经过试验后选取0.05作为实验取值。

特别的，当**dis\_th**过小时（如小于0.0001），由于固有误差的存在会导致算法无法匹配前后两帧点，会导致程序一直停滞无法运行（相当于运行时间无穷大）。因此实际取值中也要考虑到该因素的影响，不能为了精度过分减小**dis\_th**。

**两帧点匹配误差 loss**

考虑用**loss**来表示本帧点和上一帧点在匹配后的误差，衡量ICP算法的精度。实验中**loss**等同于**mean\_error**，即ICP算法本帧点同上一帧点匹配后的总误差和，图像描述了**loss**随算法运行时间变化的过程。要求**loss**满足：1）数值尽可能小；2）浮动尽可能小。这两者分别可以用均值和标准差描述

**总计算时间 cost**

考虑用**cost**来衡量ICP算法的实时性。实验中**cost**等同于算法每次执行时间，通过调用计时器实现，图像描述了**cost**随算法不断运行变化的过程。要求**cost**满足：1）数值尽可能小；2）浮动尽可能小。这两者分别可以用均值和标准差描述

**整体分析**

可见，在我们的改进算法中，使用75%的单帧数据，可以在不损失精度的同时显著提升算法速度，同时将双层循环换为**k近邻算法**也可以提升算法速度。在我们平衡精度和计算时间的过程中，我们发现，即使小车不移动，两次最完美的匹配的结果也会有0.005的误差，在小车运动时，这个误差将会更大，那么只要我们观察到误差接近0.005时就不必进一步提升算法精度，此时可以认为算法已经收敛，在这个基础上，我们通过减少循环次数，减少单次匹配点的数量的方式在不过多损失精度的同时可以较大提升算法的计算速度，相比直接复现助教的原始代码，我们的算法在计算精度相近的同时将计算时间减小到原来的五分之一，从接近20ms降低到4ms，能与激光扫描频率相匹配，极大提升了算法的实时性

经过测试，得到最佳的代码运行参数为：30次迭代下**iter=30**，75%的数据量**point\_skip=0.75**，最近邻距离**dis\_th**为0.05。此时测得各数据包下运行时间为：

|  |  |  |
| --- | --- | --- |
| 001.bag | 002.bag | 003.bag |
| 2:10:00 | 1:39:00 | 1:48:00 |

表1 不同bag下总运行时间

**Part 4 踩坑记录**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

在**ubuntu18**上复现代码时，我们发现代码的计算速度较慢，同时结果也有很大的误差，但当相同代码移植到**ubuntu20**上使用**python3**运行时，代码的计算速度和精度极大提升了，通过对比我们发现，可能是由于**python2**的**sklearn**库中的k近邻算法在实现上有一定错误，在**python3**版本的**sklearn**库中，这个错误得到了修复

**Part 5 键盘控制实物小车**

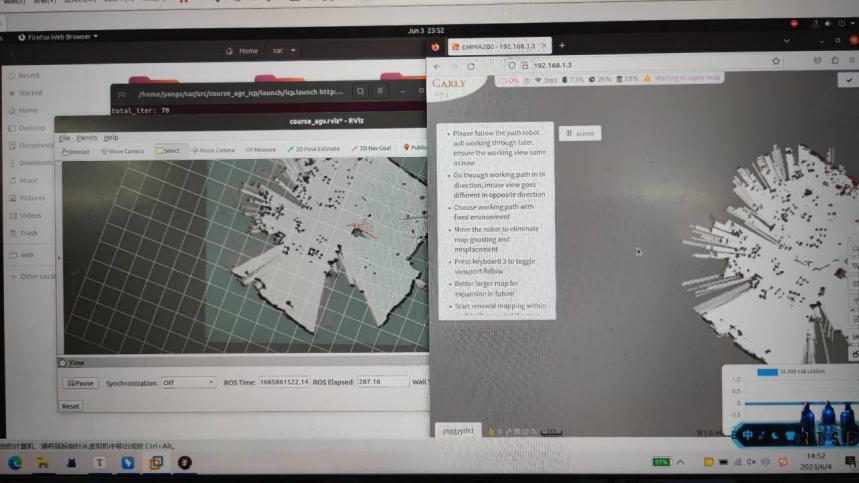


图11 使用键盘控制小车实物运行ICP

1. 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，则会产生缺失 [↑](#footnote-ref-1)
2. 这里采用的迭代次数为**70**次，迭代误差条件**tolerance = 0** [↑](#footnote-ref-2)