巡逻机器人同步定位与地图导航研究

田雅夫

June 17, 2016

摘　要

SLAM(Simultaneous Localization And Mapping, 同步定位与地图构建) 是机器人实现自主移动的前提条件, 也是机器人领域的热点问题之一. SLAM 问题的难点在于长时间运行的累积误差导致定位结果发散, 同时随着运行时间的增加, 传感器数据量增加导致系统响应时间变慢.

本文面向服务型巡逻机器人, 利用激光传感器与里程计实现了简单的同步定位与地图构建算法. 在 SLAM 获取数据的基础上, 提出并实现了了构建栅格地图, 几何地图与拓扑地图的算法. 并设计实现了基于谱图理论的自适应环境分割算法, 有效减少了机器人运行时的累积误差. 在自主地图分割算法方面, 本文提出了四种不同的相似度度量方法以构建相似度矩阵. 并利用轮廓系数评价聚类结果的好坏, 实现了自适应簇数的谱聚类算法.

同时, 本文面向先锋机器人开发了相关软件平台进行机器人环境构建与导航. 并利用开发的 SLAM 平台进行模拟与实际环境测试实验. 实验结果表明本文提出的算法在计算量可接受的情况下有效抑制了累积误差, 地图建构效果良好. 最后, 本文分析了开发算法中遇到的难题, 并提出了进一步的发展方向.

关键词: SLAM, 环境构建, 自适应谱聚类, 地图分割, 机器人路径规划

目 录

1 绪论

1.1 课题的意义

1.1.1 机器人同步定位与地图构建 (SLAM) 问题的意义

SLAM(Simultaneous Localization And Mapping, 同步定位与地图构建), 指机器人在缺乏对外界环境先验信息与自身位置信息的情况下, 通过自身携带传感器所收集的信息, 同步估算自身在外界环境中的位置, 并进行地图构建的能力. SLAM 是机器人实现自主移动的前提保障, 被许多学者认为是真正实现全自主移动机器人的关键之一.[1, 2] 文献 [3] 表明, 机器人在已知环境信息, 或者环境中存在路标的情况下进行自我定位, 和机器人在知晓自身绝对位置 (通常通过地面站或无线信标) 的情况下构建所在环境的模型是容易的. 但在不存在环境先验信息与自身位置的情况下同步估算两者是十分困难的. 因为对环境中物体的测量依赖于机器人自身的位置, 而机器人要依靠环境中的障碍物与特征点进行定位. 难点在于:

- 在没有外部参考信息的情况下, 这种测量 -构建地图 -再测量的模式会导致测量误差迅速累积, 最终使得环境构建结果发散.
- 随着机器人运行时间的增加, 传感器收集的数据也在不断累积, 如何在有限的存储空间与固定的运算时间内处理这些数据也带来了一定的挑战.
- 机器人在室内空间作业时, 由于室内环境通常比较复杂, 缺乏可供测距的几何结构. 所以要求 SLAM 算法不存在对机器人工作空间的先期假设.[4]

在过去十几年中, 如何构建高效, 鲁棒的 SLAM 系统成为机器人行为规划与机器人导航领域最热门问题之一. 并取得了许多实用的研究成果. 在1.3节中介绍了近年来 SLAM 领域的研究成果.

1.1.2 自主地图分割问题的意义

针对机器人路径规划, 导航等领域中出现的, 在大尺寸地图中算法性能下降的问题. 本文提出一种基于谱图理论的地图分割方法. 该方法将大尺寸地图切分为多个较小尺寸的子图. 从而减少机器人导航算法搜索范围, 使导航算法层次化. 利用消减问题规模的方法, 减少机器人路径规划算法的计算时间与资源开销.

在机器人路径规划, 导航等领域有许多算法是 NP-Hard 的, 或者是对地图尺寸敏感的. 在全局地图扩展到一定范围的情况下, 搜索算法的响应时间面临指数式增长. 对该类问题的一种解决方法是将其分为多个子问题并分别进行求解. 对应机器人导航领域即将全局地图切分成若干子图, 各个子图间做到信息上的高内聚低耦合. 对于室内机器人的一般工作环境, 可以将每个房间的地图作为一个子图. 在导航过程中首先进行房间层次的路径规划, 然后进行各个房间内部的路径规划. 这样可以将 NP-Hard 问题的规模显著减小从而提高响应速度.

此外, 在机器人环境感知领域, 也要求将全局地图进行划分, 并将各个子图进行概念上的标注.[5] 例如在家庭环境中, 首先通过谱分割方式将住宅空间划分为各个区域, 然后通过机器学习, 模式识别方法将分割出来的区域标注为厨房, 卧室等. 从而实现计算机与人类认知上的统一协调.

1.2 巡逻机器人的导航问题

在巡逻机器人可能的应用场合中, 期望客户给出机器人所在环境的精确地图是不符合实际的. 并且机器人所在环境可能会经常性的发生变动. 为提高机器人的环境适应性,

巡逻机器人需要在不依靠绝对定位信标 (例如地面站, 红外标签等) 的情况下进行环境构建. 在这种情况下, 同步定位与地图构建能力, 与机器人自主导航能力就尤为重要. 机器人需要自主探索给定环境并在一定精度范围内构建出环境地图. 同时, 考虑到机器人可能在相当大的范围内进行工作, 机器人的地图表示应当尽可能紧凑, 以节省存储空间, 缩短导航算法运行时间. 另外, 在大范围内进行 SLAM 算法同样面临累积误差与结果发散的问题, 在环境范围扩大的情况下保持 SLAM 结果不发散带来了一定的挑战.

1.3 机器人自主地图构建问题的研究现状与发展趋势

已有的研究中对 SLAM 问题的解决方案可分为两类:

- 一类利用自身携带的多种内部传感器 (包括里程仪、罗盘、加速度计、陀螺仪等, 通过多种传感信息的融合减小定位的误差 [6, 7, 2] , 使用的融合算法多为基于卡尔曼滤波的方法. 在传感器信息融合方面, 文献 [8, 3] 独特地利用灰色系统理论进行地图的描述与传感器信息的处理. 该算法具有相当强的鲁棒性与准确度.

- 另一类方法在依靠内部传感器估计自身运动的同时, 使用外部传感器 (如激光测距仪、视觉, 分布式 RSSI 测距等) 感知环境, 并对获得的信息进行分析抽取环境特征并保存, 在下一步通过对环境特征的比较对自身位置进行校正 [9, 10, 11]

目前, 在 SALM 问题的计算复杂度与空间占用问题上已经获得了一定的研究成果. 文献 [12] 利用机器人工作空间分解方法提出了线性时间复杂度的, 与最优解相近的 SLAM 方法 (得到的并非最优解, 而是最优解的近似). 该方法在一定范围内得到了验证. 但长时间, 大范围的环境构建始终是一个难题.

目前机器人自主地图构建的研究方向有以下几个:

- 鲁棒性: 真实世界中的 SLAM 问题面对的是经常变化的环境 (例如人群, 经常进出货物的仓库), 而一般的, 基于帧间匹配的 SLAM 算法在原有环境改变的情况下可能导致匹配失败. SLAM 算法如何适应变动的环境, 如何在变动的环境中寻找不变的特征, 以及减少外部环境的干扰是当前研究的热点之一.

- 实时性: 大范围 SLAM 算法的实时性要求算法的时间复杂度不能太高, 同时对数据的处理需要尽可能快速. 这基于要求地图的数据存储形式要尽可能紧凑, 并能够剔除冗余的数据. 同时, 算法的时间复杂度不能高于对数复杂度, 否则随着数据量的增长算法的响应时间会增长至不可接受的范围.

- 闭环检测: 闭环检测的思路在于, 当机器人得到一个观测结果时, 判断该结果是否在全局地图中出现过. 若该观测结果在全局地图中出现过, 则可以利用全局地图矫正机器人传感器得到的累计误差. 在计算机视觉辅助定位问题的解决中, 文献 [13] 提出一种基于主动环形闭合约束的 SLAM 问题解决方案. 给方案假定机器人工作区域为闭合的, 通过检测环修正机器人位姿, 减少机器人定位的不确定性. 大大减少了储存与计算的开销并使导航路径更快速平滑. 闭环检测的实现思路有以下两种:

  - 一种实现思路是检查机器人的位置, 若该机器人的位姿与全局地图记录过的机器人位姿相似, 则认为该状态下机器人的观测结果与全局地图中的对应观测结果相似
  - 另一种实现思路是基于模式识别的闭环检测. 通过检测机器人观测结果间相似度, 或观测结果与全局地图关系判断是否出现闭环. 该方法的缺陷在于, 该算法的时间复杂度通常不小于 $O(n^2)$. 在观测结果数量不断增长的情况下, 该

算法的计算量会增长至不可接受的大小. 在文献 [14] 中, 让当前观测结果与历史观测结果中的随机 k 个结果进行匹配. 若能匹配说明出现闭环. 该方法在应用中被认为是高效且准确的.

另外, 对于该问题的研究思路包括使用自然语言处理领域的" 词袋模型"(Bag of Words) 进行闭环处理 [15, 16], 或者将观测结果中相近的结果进行聚类. 通过这些方法也可以有效降低算法的时间复杂度.

- 粒子滤波器的改进: 针对粒子滤波器维数较高, 计算量较大, 粒子退化的问题. 文献 [11] 提出将遗传算法与粒子滤波结合的 Rao-Blackwellized 算法, 减少算法迭代次数并提高精度. 文献 [2] 提出了利用无线传感网络辅助同步定位的方法引入额外信息进行附加定位. 文献 [1] 利用平方根无迹卡尔曼滤波进行机器人状态估计并与红外标签观测模型结合提高定位精度与算法稳定性.

- 基于语义学的 SLAM: 在人机交互领域, 通常要求机器人对环境的" 认知" 方式与人类对环境的认知相同或相似. 人类对室内环境的认知方式是基于障碍物进行划分的. 例如走廊与房间会被门划分为两部分, 并且每一部分在人类的认知中有特殊的意义. 而机器人对环境的认知是整体的, 基于区域间联通关系的. 目前在机器人导航领域常用的环境分割算法是基于 Voronoi 图的分割算法, 分割结果与人类的认知有差异. 如何协调人类与机器人对环境的认知涉及一系列不同领域的算法.

- 多机器人协同 SLAM: 由于单个机器人行动能力与数据处理能力有限, 可以考虑使用多台机器人分别探索给定区域, 并将结果汇总为全局地图. 在这个过程中, 设计多个机器人互相定位, 多机器人数据融合, 机器人互相定位, 网络控制中时延, 丢包对机器人的影响等问题. 是 SLAM 领域非常重要与热门的问题.

1.4 文章结构

本文的主要研究内容包括:

1. 针对激光传感器 (LRF)-里程计混合定位机器人, 提出数据融合与 SLAM 方案, 搭建 SLAM 平台.

2. 针对激光传感器获取的数据, 设计并实现了栅格地图, 几何地图, 拓扑地图的构建算法. 实现了高效, 紧凑, 便于索引的数据结构以储存地图.

3. 设计并实现了基于谱理论的自主地图分割算法, 将全局地图分成各个子图并实现信息方面的高内聚低耦合. 本文对激光传感器创建的地图, 提出了多种相似度构造策略以生成相似度矩阵. 并提出了基于轮廓系数的聚类效果评价准则. 根据该准侧实现了自适应聚类算法进行地图分割.

4. 设计并实现了基于 A-star 算法的路径规划算法, 并在分层地图中实验性的使用了这一算法. 特别的, 提出并实现了一个基于有限状态自动机的机器人避障探索算法.

5. 使用自主开发的机器人平台进行以上算法的测试, 检验其有效性.

本文的内容安排如下:

- 本文第一章绪论介绍机器人同步定位与地图构建, 地图分割问题的意义与实际应用, 阐述了这些问题的研究现状与进一步的发展方向.

- 在第二章中, 本文介绍了机器人传感器数据获取, 处理与 SLAM 过程. 提出了基于线段匹配与角点匹配的全局地图匹配算法, 在一定程度上提供了解决闭环检测问题的思路.
- 第三章介绍了机器人工作空间的栅格地图, 几何地图, 拓扑地图表示, 并提出了高效构建地图的方式. 同时, 对地图数据的存储进行了研究, 提出了多种优化的数据结构用以存储地图.
- 第四章利用前文构建的地图进行自主地图分割, 提出了基于谱聚类的地图分割算法, 并阐述了多种相似度矩阵的构造准则. 在本章最后本文利用轮廓系数作为聚类效果的评价标准, 提出了自适应的地图分割算法.
- 第五章介绍了基于 Astar 算法的机器人导航算法, 特别地, 本文提出了分层导航算法以降低时间复杂度与存储空间开销.
- 在第六章, 本文介绍了本系统的设计与实现, 介绍了一些算法的实现细节与优化处理方式. 特别地, 本文提出了基于有限自动机的机器人避障探索算法.
- 第七章给出了该平台的实验结果, 并对实验结果进行分析.
- 最后, 在第八章总结全文, 并提出了机器人自主环境构建领域进一步的发展方向.

2 机器人自主环境构建

2.1 激光传感器数据获取

在 SLAM 领域常用的主要数据收集方法有机器视觉, 深度视觉, 超声传感器与 Kinect 等. 与机器视觉相比, 激光传感器对外界环境要求低, 对光照不敏感. 同时激光传感器单位时间获取的的信息量要远小于摄像头获取的, 在设备功耗, 算法响应时间, 计算资源占用方面有一定的优势.

激光传感器使用一束激光对给定区域进行扫描, 通过测量发射激光与障碍物反射光线的相位差来获取障碍物的距离信息. 如图:

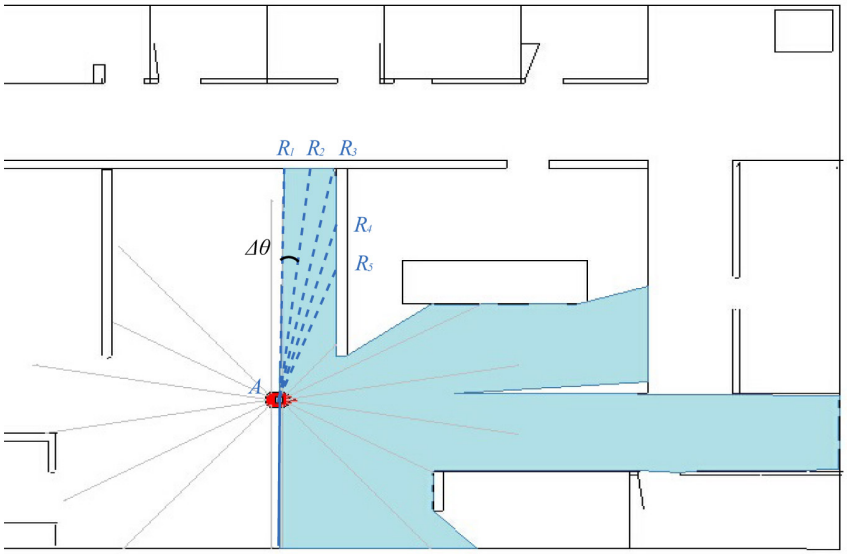


图 1: 激光传感器扫描示意图, 阴影部分为扫描区域

如图, 该机器人扫描范围为其正前方 180 度范围, 扫描精度为 1 度. 每次扫描获取 181 个数据点, 即每次扫描得到 181 个距离信息. 对于机器人的第 i 次扫描结果, 用 r_i 表示:

$$r_i = [r_{i1}, r_{i2}, r_{i3}, \cdots, r_{ij}, \cdots, r_{in}] \quad (0 < n \leqslant 181) \tag{1}$$

同时, 根据机器人内部储存的, 对应第 i 次扫描结果的位姿信息 (x_i, y_i, θ_i) 可以计算出第 i 次扫描结果中每个障碍物的坐标:

$$obstacle_i = \{(x_i + r_{ij}\cos\theta_i, y_i + r_{ij}\sin\theta_i) | j \in [1, 181]\} \tag{2}$$

为表示方便, 将扫描结果记为坐标点序列的形式:

$$o_i = \{(x_1, y_1), (x_2, y_2), \cdots, (x_i, y_i)\} \quad (0 < i \leqslant 181) \tag{3}$$

为了适配多种机器人, 同时增强机器人对多种环境的适应性, 将机器人获取的数据进行标准化, 并映射到给定区间中. 设机器人活动的最大范围为边长为 L_{max} 的矩形, 机器人内部地图为边长为 L 的矩形, 以矩形中心为坐标原点, 则做以下变换:

$$O_i = \{(\frac{Lx_i}{L_{max}}, \frac{Ly_i}{L_{max}}) | (x_i, y_i) \in o_i\} \tag{4}$$

机器人内部地图尺寸 L 根据要求的地图精度进行设置. 为了使机器人适配各种尺度的环境, 应该避免对外界环境的先验假设, 所以机器人活动的最大范围 L_{max} 在 SLAM 问题中是不可知的. 在机器人运动过程中根据积累的测量结果设置 L_{max} 的值, 同时要注意留出一部分余量.

2.2 机器人里程计数据获取

机器人里程计提供了机器人走过距离的累计值, 常见的里程计实现是在机器人的每一侧轮子上面安装光电码盘以测量轮子转的圈数. 在已知轮直径的情况下可以计算机器人该侧行进距离. 根据机器人两侧里程计之差值可获得机器人旋转运动量. 假设机器人在 $k-1$ 和 k 时刻的姿态为 S_{k-1} 与 S_k, 其中 $S_i = [X_i, Y_i, Z_i]^T$, 旋转运动量为 $\alpha_{k-1,k}$, 平移运动 $D_{k-1,k}$. 则根据里程计定位模型 [17] , 有:

$$\begin{bmatrix} X_k \\ Y_k \\ \theta_k \end{bmatrix} = f(\begin{bmatrix} X_{k-1} \\ Y_{k-1} \\ \theta_{k-1} \end{bmatrix}, \begin{bmatrix} D_{k-1,k} \\ \alpha k-1, k \end{bmatrix}) = \begin{bmatrix} X_{k-1} \\ Y_{k-1} \\ \theta_{k-1} \end{bmatrix} + \begin{bmatrix} D_{k-1,k}\cos(\theta_{k-1}+\alpha_{k-1,k}) \\ D_{k-1,k}\sin(\theta_{k-1}+\alpha_{k-1,k}) \\ \alpha_{k-1,k} \end{bmatrix} \tag{5}$$

里程计定位方式需要的定位信息较少, 实时性较高, 节省计算资源. 并且该导航方式几乎不受外部信息的干扰. 但影响里程计定位的误差因素较多, 包括轮子直径, 安装方式造成的系统误差与机械部件游隙, 打滑等因素造成的非系统误差. 并且在机器人运行过程中, 里程计定位的误差不断累积, 导致定位结果的误差增大, 可信度下降. 图2.2显示了机器人在运动过程中误差的累积. 故需要在机器人运行过程中利用外部信息实时校正里程计定位结果. 一种方案是在机器人运行过程中, 实时记录每次激光测距仪采集信息时的机器人位姿 (X, Y, θ), 在里程计定位的同时利用观测结果进行机器人位置的校正. 在2.3节中详细介绍了这种方法.

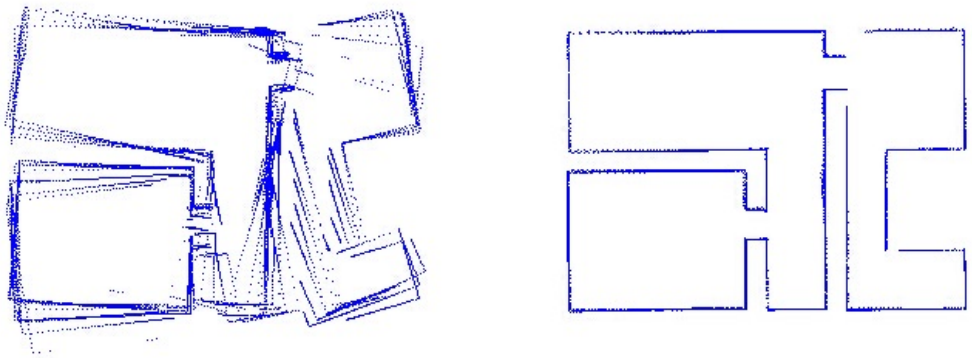


图 2: 左图显示了单纯依靠里程计进行 SLAM 带来的累计误差, 在该模式下, 不利用激光传感器进行里程计校正. 右图显示了利用激光传感器进行里程计误差矫正的结果, 与左图结果相比减少了累计误差

2.3 同步定位与地图构建

在传感器数据获取章节中提到了, 利用里程计进行机器人定位与姿态感知会产生累计误差, 导致定位精度下降. 并且里程计定位法对绑架问题 (即机器人在外力的作用下产生了没有记录的位移, 例如机器人被搬起来移动了一段距离) 没有很好地应对措施. 一种解决方案是利用激光传感器对机器人所在环境进行扫描, 利用扫描结果进行机器人定

位. 该方法不会产生累积误差, 并且精确度相当高. 但是时间复杂度相对较高, 并且激光传感器对环境的扫描结果有匹配失败的可能, 在匹配失败的情况下无法进行导航.

图 3中显示了两种可能导致激光扫描仪匹配失败的情景, 图中蓝色线段表示机器人行进轨迹, 阴影部分代表扫描结果. 左图中机器人进行了一个大角度的旋转, 导致扫描区域与原有地图完全不匹配, 在这种情况下产生的定位结果不可信. 右图中机器人所在环境被分割为相关性很小的两部分, 机器人在门两侧扫描的结果相关性不大, 会导致定位失败.

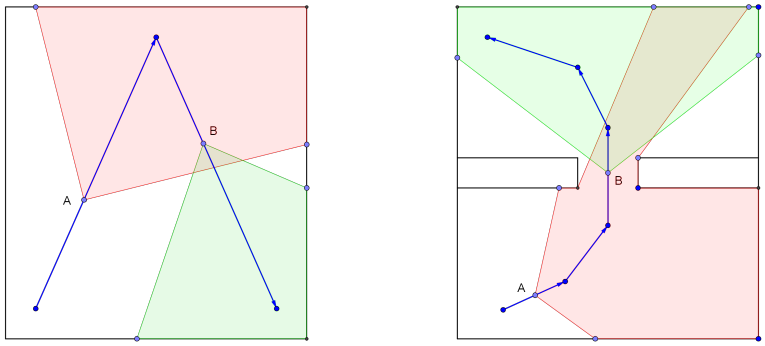


图 3: 激光传感器进行定位时可能导致失败的两种情况

在机器人自身位置估计方法中, 激光传感器获取的障碍物坐标点依赖于机器人自己的位姿. 是从第 k 次的扫描结果向第 $k+1$ 次扫描结果的一个递推. 所以用单纯利用激光传感器进行定位的方法同样面临累积误差. 为了削减累积误差, 需要让激光传感器每次的扫描结果与已知的所有结果进行匹配. 以减少逐帧匹配所带来的累积误差, 如图4所示. 但该方法带来的问题是随着扫描结果的增长, 匹配过程的计算量呈指数增长. 为解决此问题, 需要精简机器人工作空间的障碍物描述以消减问题规模. 同时利用高效的数据结构 (哈希表, 空间索引树等) 减少算法的时间复杂度, 达成" 空间换时间" 的效果. 机器人自适应定位的流程如图5所示, 在第6章中, 提供了该算法的详细实现思路.

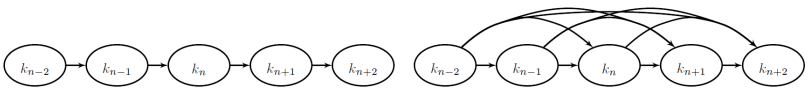


图 4: 利用逐帧匹配的方法削减累积误差

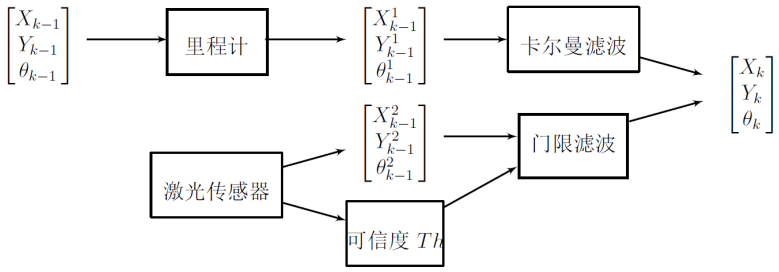


图 5: 机器人自身位置估计算法流程

在得到激光传感器扫描结果 o_i 的情况下, 可以通过扫描结果反推激光传感器的位置. 从扫描结果中任取三点 p_1, p_2, p_3, 这三点离机器人的距离为 r_1, r_2, r_3. 并设机器人坐标 $R(x, y)$ 如图6所示, 有以下关系:

$$\begin{aligned} (x_1 - x)^2 + (y_1 - y)^2 &= r_1^2 \\ (x_2 - x)^2 + (y_2 - y)^2 &= r_2^2 \\ (x_3 - x)^2 + (y_3 - y)^2 &= r_3^2 \end{aligned} \tag{6}$$

通过求解上述方程可以得出点 R 的坐标. 在实际应用中, 由于测量误差与噪声, 三个圆必回精确地交于点 R, 一种高效的替代方法为两两联立圆方程, 解出六个交点. 这六个焦点中有三个距离较近并远离另外三个, 对这三个交点取质心即为激光传感器所在点.

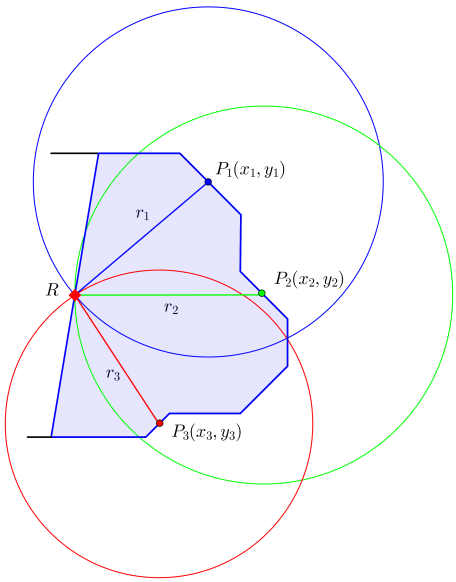


图 6: 激光传感器定位原理示意图

激光传感器可以得到机器人周边障碍物相对机器人的位置信息. 然而单凭该位置信息进行机器人自定位, 同样是开环系统, 也面临累积误差的问题. 在定位环节中, 需要通过反馈来抑制 SLAM 环节的累积误差. 一种反馈方式是将激光传感器 SLAM 结果与全局地图进行匹配, 通过匹配结果得到矫正信息. 在基于机器视觉的 SLAM 中, 可以通过 SIFT 特征点进行固定目标的匹配. 而在基于激光传感器的 SLAM 中可以通过如下方式进行匹配:

- 从激光传感器扫描结果中提取线段, 与基于线段表示的几何地图中障碍物进行匹配.
- 从激光传感器扫描结果中提取角点, 与全局地图中出现的角点进行匹配.

匹配的结果可能有如下三种情况:

- 激光传感器正确地将扫描结果与全局地图进行了匹配
- 激光传感器错误地匹配了扫描结果与全局地图
- 激光传感器不能匹配扫描结果与全局地图

第一种结果是我们希望看到的, 但在机器人运行过程中, 由于传感器误差, 外力作用, 特征不明显等原因也会时常发生误匹配现象. 当机器人突然扫描到了一个新的区域时 (例如突然通过一扇门, 或是急转弯的时候), 由于扫描结果不存在于全局地图中, 所以常常导致匹配失败与错乱匹配. 由于这些情况, 需要一个评价准则来对匹配结果进行评价. 当匹配结果正确时, 接纳该匹配结果, 当匹配结果不正确或无法匹配时, 应当能够有效过滤掉该匹配结果, 而依靠里程计进行继续定位. 同时, 当无法匹配情况发生时, 将扫描结果加入全局地图中. 虽然当激光传感器无法匹配全局地图时, 机器人处于开环 SLAM 状态, 但里程计定位在短时间内的精度是可以接受的, 当将未知区域加入到全局地图中, 无法匹配状态就会结束.

2.3.1 线段匹配

在实际运行环境中, 机器人工作空间内许多障碍物可以抽象成线段表示. 例如墙面, 台面等. 相对于海量的障碍物坐标点, 将障碍物进行抽象表示可以极大地减少计算量, 并使环境特征更为突出. 而线段匹配即将机器人当前扫描结果抽象成线段, 与全局地图中抽象为线段的障碍物相匹配. 若当前扫描到的障碍物为全局地图中障碍物的一部分 (由于累积误差原因, 可能进行了平移, 旋转变换), 则可以通过全局地图对当前机器人的位姿进行校正.

线段匹配问题可以用以下方式进行描述:

线段 l 用四元组表示, 即 $l = (p_{1x}, p_{1y}, p_{2x}, p_{2y})$ 表述, 其中 p_1, p_2 为线段的两个端点, 为方便比较, 规定 p_2 在 p_1 的第一象限方向, 即 $p_{2x} \leqslant p_{1x}, p_{2y} \leqslant p_{1y}$.

机器人工作环境的地图用集合 M 表示, 其中 $M = \{l_1, l_2 \cdots l_n\}$. 要求 M 中的线段长度要大于规定的最小长度 $length$, 以过滤环境中干扰与线段错误提取带来的干扰, 减小匹配误差. 机器人扫描结果用线段集合 Obv 表示, Obv 中一部分线段由 M 中一部分线段的一部分 (扫描结果中可能只包含全局地图中某线段的一部分) 经平移, 旋转变换而来. 要求:

- 从全局地图与机器人扫描结果中提取线段构造集合 M, Obv. 要求全局地图提取算法有良好的扩展性, 机器人扫描算法中的线段提取算法有较好的实时性. (全局地图中提取一次线段后就尽量不再次提取, 而是将机器人扫描结果中的线段加入到提取结果中, 减少计算量. 因为激光扫描仪扫描速度较快, 并且单位时间内匹配次数越多, SLAM 误差越小, 所以要求从扫描结果中提取线段的算法有良好的实时性)
- 找出 Obv 与 M 间最优匹配, 要求 Obv 中又尽量多的线段能够与 M 中线段进行匹配, 同时要求匹配误差尽可能小.
- 通过已匹配线段反推机器人位姿

全局地图的线段提取将栅格地图进行 Canny 边缘检测, 对检测出来的边缘使用 Hough 变换即可进行线段的提取. 在 OpenCV 中有这两个操作的高效实现. 但该方法计算时间相对较长, 不适合用在实时扫描算法中. 实时扫描算法中的线段提取是角点提取的" 副产品". 当前扫描结果为障碍物点序列 $O =< p_1, p_2, \cdots, p_n >$, 是定长, 按照扫描顺序顺时针排列的坐标点. 定义函数 $Angel(p_{i-1}, p_i, p_{i+1})$ 为线段 (p_{i-1}, p_i) 与 (p_i, p_{i+1}) 夹角. 将该函数 map 到序列 O 上可得到夹角序列 $A = a_1, a_2, \cdots a_{n-1}$. 对 A 求一阶导数, 并将相邻峰值进行合并. 其峰值位置为角点, 连续较长的接近 0 的序列为线段.

2.3.2 角点匹配

角点匹配通过记录全局地图中障碍物的特征点 (这里选择角点作为特征点) 以匹配观测结果与全局地图. 提取特征点的算法在4.2.1章节中进行了详尽的描述. 当机器人观测到一个全局地图中的特征点时, 通过该特征点到激光探头线段与激光传感器中轴线夹角, 与该特征点距机器人的距离可推测机器人的位姿. 通过一个点进行定位存在一定的误差, 更有效的方法是通过多个点进行定位, 如图6所示. 与基于线段的匹配方法相同, 若该机器人扫描结果中没有已知角点, 则无法完成定位.

使用角点定位的优点在于其大大的减少了计算量. 通常在地图中, 角点的数量要少于线段数量. 并且匹配角点的计算量要远少于匹配线段的计算量. 缺点是在实际运行中, 角点匹配匹配失败的概率要高于线段匹配的概率. 并且许多匹配结果是单点匹配, 即只观测到了一个曾经出现过的特征点. 在单点匹配情况下认为反推出的机器人位姿可信度 (confidence) 不高.

2.3.3 匹配结果的度量

为了实现不同匹配算法间结果融合, 需要实现一种度量匹配结果可信性的方案. 同时, 因为对全局地图的匹配有失败可能, SLAM 算法应当能在可信度过低 (匹配失败) 时丢弃匹配结果.

线段匹配可信度度量方法 再一次观测结果中, 若能够进行匹配的线段够多, 则认为匹配结果可信度更高. 即匹配结果的可信度用已匹配线段长度比总线段长度来定义. 设观测结果线段集合 Obv, 已匹配线段集合 $L_{matched} = \{(p_{11}, p_{12}), \cdots (p_{m1}, p_{m2})\}$, $L_{matched} \subseteq Obv$. 则可信度做如下定义:

$$confidence = \frac{\sum\limits_{p_i, p_j \in L_{matched}} \sqrt{(p_{ix} - p_{jx})^2 + (p_{iy} - p_{jy})^2}}{\sum\limits_{p_i, p_j \in Obv} \sqrt{(p_{ix} - p_{jx})^2 + (p_{iy} - p_{jy})^2}} \tag{7}$$

角点匹配可信度度量方法 度量角点匹配可信度的方法度量是在一次观测中匹配到的交点个数. 在实际应用中, 经测试, 可认为匹配到一个点的可信度为 0.33, 两个点 0.66, 三个点及以上取 1.

帧间匹配方法 在机器人运动过程中, 常常认为采集结果是连续的. 然而采集结果间相似度也有突变情况发生. 通常相似度突变的情况发生于探索新环境, 噪声干扰过大, 机器人环境突变等情况. 可以认为, 帧间结果连续是保证测量结果可信的依据. 在 4.2.1中, 描述了多种评价扫描结果相似度的算法.

2.4 小结

本章介绍了机器人机载里程计与激光传感器数据格式与处理, 给出了激光传感器与里程计定位模型, 并利用这两种传感器实现了了一个简单的 SLAM 算法.

SLAM 问题的核心在于削减连续工作导致的累计误差. 在 SLAM 部分, 本文提出了基于线段与角点的两种扫描结果匹配方式. 利用这两种匹配方式实现了单次扫描结果与全局扫描结果的高效匹配. 并且有效的减少了问题的规模, 节省了计算资源.

最后, 考虑到激光传感器扫描结果匹配存在失败与误匹配可能, 本文提出了线段匹配与角点匹配的可信度度量方法. 并提出帧间匹配的方法对匹配结果进行评价.

3 机器人工作空间的存储方式

机器人地图存储方式通常有拓扑地图, 栅格地图, 几何地图与混合地图四种.

- 栅格地图: 栅格地图将机器人的工作空间进行离散化, 并用一组基本单元进行表示. 通常采用正方形, 正六边形或正三角形进行环境的分解. 栅格地图在建立, 维护方面相对较方便, 空间中物体的表示是精确的. 栅格地图可以处理由于传感器精度与环境噪声带来的不确定性. 缺点是栅格地图占用空间较大, 遍历操作较复杂.
- 拓扑地图: 拓扑地图抽象了机器人工作空间的连通性与可达性. 拓扑地图用拓扑图表示机器人的工作空间, 图的节点表示一部分环境特征相似的空间, 图的边表示这些空间的可达性. 拓扑地图有助于机器人对环境进行符号化与推理. 同时拓扑地图也有占用存储空间较小, 便于规划算法的运行等优点. 但拓扑地图缺乏对环境细节的表示与机器人定位信息, 很难单独使用.
- 几何地图: 拓扑地图抽象了机器人工作空间的环境信息, 将机器人工作空间的障碍物用相近的几何模型描述 (例如二维空间内的线段, 矩形, 圆形, 三维空间内的立方体与圆柱). 相对于栅格地图, 几何地图可以大大减小用于描述环境所用的信息 (即地图占用存储空间), 同时缩短空间遍历, 导航, SLAM 的运行时间. 但在构建几何地图的过程中拟合环节会带来一定的误差. 不同的拟合算法在时间复杂度与拟合效果方面也有着不同的 ‘表现
- 混合地图: 混合地图同时包含栅格地图, 拓扑地图, 几何地图中的两种或多种信息, 根据不同算法的需求使用不同层次的信息进行计算.

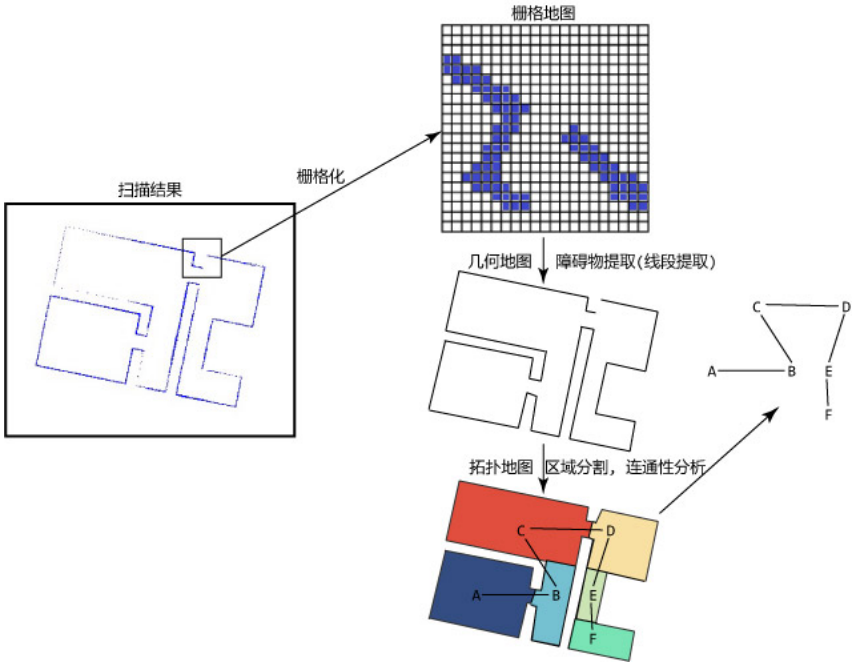


图 7: 机器人不同环境表示与构建方式示意图

3.1 栅格地图

在机器人运行过程中, 激光传感器不断地高速采集数据, 而用于容纳地图信息的存储器空间是一定的. 为了解决传感器数据增长与存储器容量之间的矛盾关系, 需要对采集

的数据进行归一化整理并丢弃原始采集数据. 即用固定的空间来容纳不断增长的数据. 栅格化可以在一定精度范围内解决这一问题.

如图8, 一般的栅格化过程步骤为:

1. 放缩变换
2. 二值化
3. 坐标变换

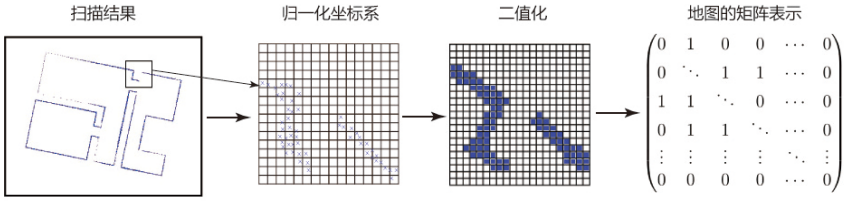


图 8: 栅格地图构建流程示意图

由于机器人的种类, 搭载传感器, 运行的空间尺度不同, 需要将机器人的工作空间划归至一个标准坐标系中. 在机器人工作空间用正方形栅格表示的情况下, 设机器人工作空间的尺度为 $[(0,0),(x_{max},y_{max})]$, 标准坐标系空间尺度为 $[(0,0),(columns\cdot\Delta S, rows\cdot\Delta S)]$, 其中 $rows, columns$ 为标准空间内的行数与列数, ΔS 为每个正方形单元的边长. 设点 (x,y) 为机器人工作空间内观测到的障碍物点, (x',y') 为标准坐标系中对应点的坐标, 则有:

$$\begin{cases} x' = x\cdot\dfrac{columns\cdot\Delta S}{x_{max}} \\ y' = y\cdot\dfrac{rows\cdot\Delta S}{y_{max}} \end{cases} \tag{8}$$

机器人内部通常用矩阵进行地图的存储, 为了尽可能减少存储空间, 使用 1 个比特位存储一个正方形栅格的信息. 即一个栅格取值为 0(没有障碍物) 或 1(有障碍物). 这样一个字节可以存储八个栅格的信息, 大大缩小了地图的存储空间. 然而由于传感器精度与累计误差的因素影响, 这种存储方式在机器人长时间运行时会造成障碍物的堆积, 后面基于概率的栅格地图中用 1 个 float 型数据存储一个栅格, 虽然占用空间有所增加, 但可以对机器人移动过程中存在的不确定信息进行处理.

假设地图存储空间 M 为 x 行 y 列的 $0-1$ 矩阵, 则:

$$M = \begin{pmatrix} m_{11} & m_{12} & \cdots & m_{1x} \\ m_{21} & m_{22} & \cdots & m_{2x} \\ \vdots & \vdots & \ddots & \vdots \\ m_{y1} & m_{y2} & \cdots & m_{yx} \end{pmatrix} \tag{9}$$

对应落到每个单元内的观测点集合 set_{ij} 为:

$$set_{ij} = \{(x,y) \mid j\cdot\Delta S \leqslant x < (j+1)\cdot\Delta S\ ,\ i\cdot\Delta S \leqslant y < (i+1)\cdot\Delta S\ ,\ (x,y)\in O\} \tag{10}$$

则有:

$$m_(ij) = \begin{cases} 0 & card(set_{ij}) = 0 \\ 1 & card(set_{ij}) \neq 0 \end{cases} \tag{11}$$

3.1.1 栅格地图的稀疏矩阵存储

稀疏矩阵是数值为零的元素远多于非零元素的矩阵. 通常要求稀疏矩阵中非零元素的比例小于 5% 在机器人工作环境很大而且环境中障碍物相对较少的情况下, 可以采用稀疏矩阵进行地图的存储. 稀疏矩阵的存储使用三元表存储形式, 如下表:

序号	行	列	值
0	5	7	0
1	2	3	1
⋮	⋮	⋮	⋮

图 9: 稀疏矩阵的三元表存储

图 10: 如图, 地图大小为 600×600, 但在全部 360000 个栅格中只有 2647 个栅格内为非零元素, 占全部元素数量的 0.735%, 远小于 5%. 可见传统的矩阵存储方式存在着巨大的浪费

稀疏矩阵存储可以极大地缩减地图的占用空间. 在地图索引频率较高的情况下, 也可以采用哈希表进行地图的存储.

3.1.2 栅格地图的空间索引树存储

栅格地图可以以任意给定精度 (不大于传感器精度) 记录地图细节, 在对地图精度要求高的场合 (例如 SLAM 过程), 栅格地图可以提供相对更加准确的信息. 但栅格地图也存在占用空间大, 不便于遍历等缺点. 在实际地图构建中, 经常出现大部分栅格被浪费的情况. 与二八定律类似地, 大部分栅格 (多于 90%) 中并没有包含算法所感兴趣的信息, 而少量存储障碍物的栅格由于存储空间限制而不能提供很高的精度. 针对该问题, 有着一些较好的解决方案.

针对栅格地图中栅格浪费的情况, 一个直观的解决方案是对栅格地图进行压缩存储, 即合并大片空白区域以减少栅格地图空间占用, 同时提供索引接口使这一压缩过程对外部算法透明化. 更进一步地, 可以使用空间索引树代替矩阵进行栅格地图的压缩存储.

四叉树是空间索引树的一种实现方式, 树的每个节点代表地图中的一块区域. 该节点的子节点将这块区域划分为四部分, 储存于四个子节点中, 如图所示:

如图, 四叉树将没有障碍物点的区域" 合并" 到了一起 (确切来说, 对于没有障碍物的区域, 不生成四叉树的子节点), 大大减少了栅格的生成量. 对层数为 n 的四叉树, 其精度为 $\frac{MapSize}{2^n}$, 即与栅格数量为 2^{2n} 的栅格地图精度相当. 如图中例子所示, 对于一个 10 层的四叉树, 与其精度相当的栅格地图需要包含 $2^{10} \times 2^{10} = 1048576$ 个栅格, 而四叉树表示一共花费了 2685 个节点用于四叉树的存储, 后者仅为前者存储空间的 0.26%, 可以说是极大地节省了存储空间.

四叉树的每个节点包含以下信息:

1. 节点层数, 用非负整数表示. 节点层数不应大于要求的最大层数
2. 该节点所包含区域, 用四元组 $(top, left, bottom, right)$ 表示, 若要进一步节省空间, 该信息可以由节点在树中位置计算得出
3. 节点所包含的观测结果, 该信息可以做如下表示:
    - 若该节点非根节点, 观测结果为空

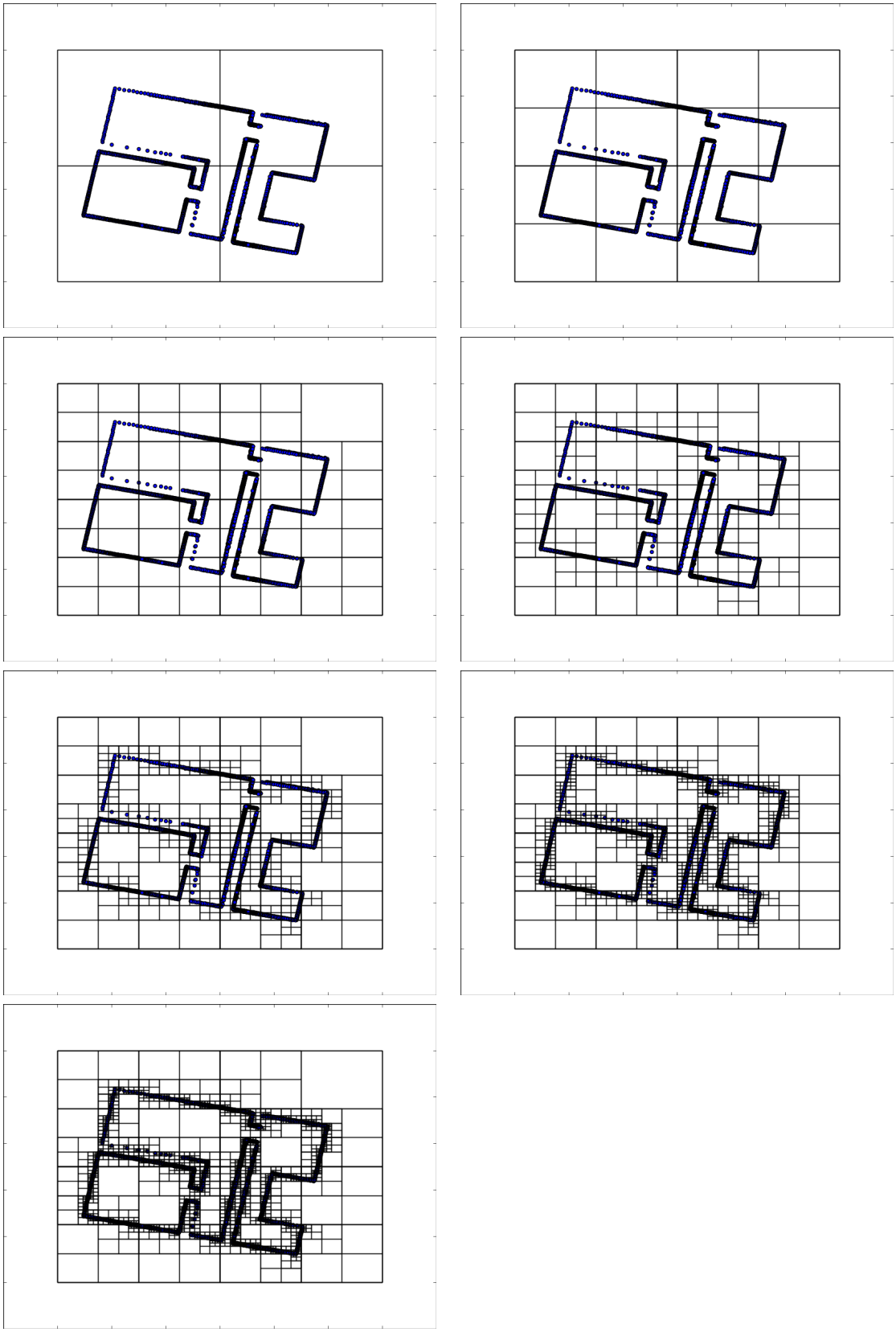


图 11: 对于给定地图的四叉树表示, 图中分别给出了四叉树深度为 1,2,3,4,5,6,7 时的四叉树

- 若该节点为根节点, 且该节点代表区域中不存在障碍物点, 观测结果为空
- 若该节点为根节点, 且该节点代表区域中存在障碍物点, 观测结果中包含这些障碍物点. 若要进一步节省存储空间, 可以直接存储布尔变量 $Truse$ 代替这些障碍物点.

4. 该节点的四个子节点. 若该节点非叶子节点, 则该节点的四个叶子节点将该节点代表区域分为四部分.

四叉树的构建算法如下, 通过递归方式生成子节点, 并将观测结果分配至子节点. 递归的出口条件有两个, 即不再有观测结果需要分配与节点层数到达最大层数:

```
class TreeNode:
    def __init__(self, layer, dataSet, region, max_layer):
        self.dataSet = dataSet
        self.region = region ##(top, left, bottom, right)

        self.top = region[0]
        self.left = region[1]
        self.bottom = region[2]
        self.right = region[3]
        self.xMiddle = (self.left + self.right) / 2
        self.yMiddle = (self.top + self.bottom) / 2

        self.layer = layer
        self.childlst = None
        self.Empty = True
        if(len(dataSet) == 0):
            self.childlst = None
        elif(self.layer <= max_layer):
            self.childlst = []
            self.Empty = False
            dataSet1 = []
            dataSet2 = []
            dataSet3 = []
            dataSet4 = []
            for i in dataSet:
                if(i[0] > self.xMiddle and i[1] > self.yMiddle):
                    dataSet1.append(i)
                elif(i[0] >= self.xMiddle):
                    dataSet4.append(i)
                elif(i[1] > self.yMiddle):
                    dataSet2.append(i)
                else:
                    dataSet3.append(i)
            self.childlst.append(TreeNode(self.layer+1, dataSet1, (self.top,
                self.xMiddle, self.yMiddle, self.right), max_layer))
            self.childlst.append(TreeNode(self.layer+1, dataSet2, (self.top,
                self.left, self.yMiddle, self.xMiddle), max_layer))
            self.childlst.append(TreeNode(self.layer+1, dataSet3, (self.yMiddle,
                 self.left, self.bottom, self.xMiddle), max_layer))
            self.childlst.append(TreeNode(self.layer+1, dataSet4, (self.yMiddle,
                 self.xMiddle, self.bottom, self.right), max_layer))
    else:
    self.dataSet = dataSet
    self.Empty = False
```

在四叉树中索引一个节点是十分便利的, 对于给定坐标 (x, y), 查找该坐标代表位置是否有障碍物, 就从根节点开始, 根据该坐标在四叉树中位置依次向下, 直到叶子节点为止. 查询结果有如下三种可能:

1. 叶子节点层数小于树的最大层数, 这代表该坐标附近没有障碍物.
2. 叶子节点等于树的最大层数, 该节点的观测结果为空, 这代表该坐标附近没有障碍物. 但附近四个栅格大小的范围内内存在障碍物点.
3. 叶子节点等于树的最大层数, 该节点的观测结果为" 存在障碍物".

对于深度为 n 的四叉树, 索引一个节点的计算次数不大于 n. 同时, 基于矩阵的栅格地图索引一个节点的计算次数恒为 2 次 (前提是矩阵使用定长数组存储). 通常使用的四叉树深度不大于 12, 即可以认为四叉树的索引操作与矩阵的索引操作时间复杂度相同, 均为 $O(1)$.

四叉树的插入, 删除操作与索引操作类似. 要注意的是插入操作要一直插入叶子节点直到树的最大深度. 删除节点时若该节点的兄弟节点均为空, 则同时删除该节点的兄弟节点.

进一步地, 对于空间搜索树, 还有更加高效的实现方案. 例如 Ball-Tree, KD-Tree(对于本文所述应用场合, 即为 2D-Tree) 等. Ball-Tree 将机器人工作空间映射到一个三维球面上, 利用球面的一些性质简化运算. 而 KD-Tree 关心障碍物点分布的概率, 利用障碍物分布的方差优化栅格分布.

3.2 几何地图

几何地图是机器人工作空间中障碍物的抽象描述. 由于激光扫描仪本身工作原理的限制, 机器人对空间中障碍物的扫描结果是孤立的点, 相当于对障碍物进行抽样. 构建几何地图的过程即将这些孤立点还原为障碍物空间形态的过程.

3.2.1 几何地图的构建

构建几何地图的过程可以看做对空间中离散数据点进行拟合的过程. 本文中利用线段作为障碍物的几何特征. 利用 Hough 变换进行障碍物的提取.

机器人工作空间的栅格地图 M 可以看做二值化后的图像, 则可以利用该图像进行障碍物的提取. 具体经过以下步骤:

1. 对图像 M 进行高斯模糊
2. 对图像 M 进行 Canny 检测以提取障碍物边缘
3. 对图像 M 进行 Hough 检测提取障碍物
4. 对检测出来的障碍物进行合并
5. 以适当的形式存储障碍物

3.2.2 几何地图的存储方式

在空间中描述一线段可以用其起始点与终点表示, 所以几何地图可以存储成定长点列:

$$M_{geometry} = \begin{pmatrix} (p1_x, p1_y) & (p2_x, p2_y) \\ (p3_x, p3_y) & (p4_x, p4_y) \\ \vdots & \vdots \\ (p(m-1)_x, p(m-1)_y) & (pm_x, pm_y) \end{pmatrix} \tag{12}$$

在实际应用中, 障碍物大多不是孤立的直线, 而是多条折线. 则式12的存储方式会导致浪费. 为节约存储空间, 同时更好的突出障碍物间的关系, 可以用指针链表的方式存储几何地图, 如图:

3.2.3 几何地图的特点

几何地图是对机器人工作空间中障碍物的抽象描述, 与栅格地图相比, 几何地图极大的减小了占用的存储空间, 但是会稍微破坏原有地图的细节. 同时, 栅格地图对环境中的障碍物有更好的描述. 在机器人导航, 探索与路径规划的过程中, 经常要判断路径, 机器人位姿与障碍物的关系. 在 SLAM 问题中也需要利用障碍物进行机器人位姿的矫正. 因几何地图抽象的描述了障碍物, 使用几何地图能大大减小这些计算的时间复杂度.

3.3 拓扑地图

3.3.1 拓扑地图的构建

拓扑地图是机器人工作空间连通性与可达性的抽象. 拓扑图的每个节点对应机器人工作空间内一块区域. 为尽可能缩减拓扑图的规模并使其清晰直观, 不同节点所对应的区域应做到信息层面的高内聚低耦合. 即不同区域间" 共享" 的地图特征信息最少. 另外构建的拓扑地图其节点数量应尽可能少. 拓扑地图构建的标准如图12所示. 构建拓扑地图的过程即对全局地图进行切分的过程. 环境切分算法的优劣决定了拓扑地图构建效果的好坏. 并且一个优秀的地图分割算法应当在任何环境下都有相对优秀的表现.

在常见的室内环境中, 以房间, 走廊作为拓扑地图的节点可以将机器人内部地图信息与人类对机器人所在空间的感知相统一. 同时, 由于门窗等障碍物的存在, 不同区域之间处于相对独立并相互连接的状态. 在第4章中介绍了基于谱聚类 (Spectral Clustering) 的地图切割算法. 该算法利用机器学习领域的相关知识解决地图切分问题, 切分效果较好.

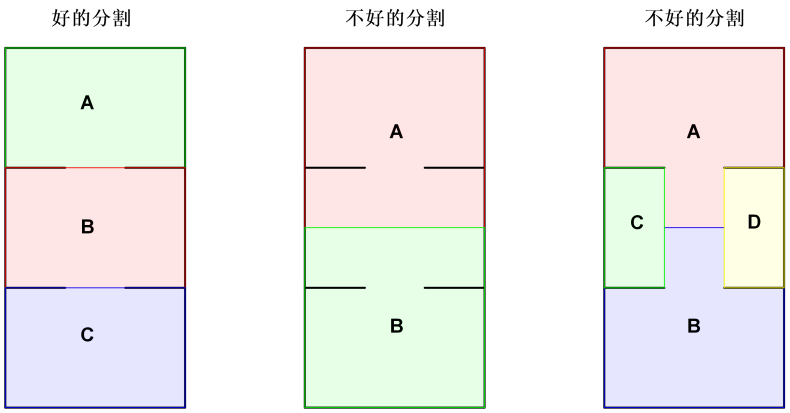


图 12: 图中显示了对同一地图的三种分割方式, 左面是最好的一种分割方式. 中间的分割方式在不同地图间共享了太多信息, 右面的分割方式产生了太多的区域, 增加了拓扑地图的复杂度

3.3.2 拓扑地图构建结果的评价

如图12所示, 拓扑地图构建的标准如下:

1. 不同区域内共享的信息量最少
2. 不同区域内的联通关系明确
3. 由算法划分的各个区域应该有一些语义上 (Semantic) 的联系, 以使人类对环境的感知与机器人对环境的感知相协调.
4. 划分的子区域应尽可能少, 以便于计算机处理

以上标准是对拓扑地图构建结果相对主观的评价标准, 为了自动化评价拓扑地图的构建结果, 应当建立一个评价函数 $Rank(map, segmentationResult)$ 对评价结果进行量化打分.

3.3.3 拓扑地图的存储

拓扑地图本质就是拓扑图, 占用存储空间较少, 不同的存储方式对存储空间影响都在可接受范围之内. 但拓扑地图索引, 查找子节点等操作相对较频繁, 所以设计拓扑地图存储方式的要点是要便于进行索引操作. 拓扑地图的存储内容包括:

1. 各个区域的中心点坐标
2. 各个区域间连接关系

区域的中心点为一个位于该区域内的, 代表该区域的点. 需要注意的是拓扑地图中不直接存储一个区域的形状. 而是在栅格地图中存储每个栅格的归属区域. 为便于索引操作, 在存储结构的实现中, 使用哈希表代替传统列表, 便于索引实现. 哈希表使用键值对的方式, 通过键的元素计算获得数据存放地址, 将列表的查找时间复杂度由 $O(n)$ 减小到 $O(1)$. 在 Python 中, 哈希表使用开放定址法处理冲突, 可以实现高效的索引.

3.3.4 拓扑地图的特点

拓扑地图中不包含机器人工作环境的细节, 这使得拓扑地图占用的空间极小. 但这同时导致拓扑地图无法独立承担机器人导航任务. 事实上, 可以讲拓扑地图理解为机器人工作环境的" 骨架". 通过拓扑地图将不规则的机器人工作空间划分为便于处理的子区域, 可以极大的减小机器人路径规划算法的工作量.

在巡逻机器人的应用场合中, 用户通常不关心机器人运动状态与周围环境的具体细节. 此时拓扑地图提供了一个用户友好的方式展现机器人工作状态, 便于用户理解与交互. 比起" 让机器人移动到坐标 (x, y)" 这样的指令, " 让机器人移动到 A 区域" 是更好的表达方式. 同时, 一个构建良好的拓扑地图其每个子区域也包含一定的语义学信息. 例如一个家用机器人会将工作区域划分为厨房 -客厅 -书房 -卧室 -卫生间等区域. 虽然机器人无法得到这些区域的语义学信息, 但这种划分方式与人类对环境的感知方式相同, 为客户提供了极大的便利. 进一步地, 通过摄像头与云物体识别数据库, 机器人能感知这些区域的语义学信息. 例如在区域 A 中识别出了床这个物体, 根据贝叶斯规则, 基本可以判断该物体为卧室. 水槽与马桶同时出现的区域通常为卫生间. 这种方式进一步减小了客户的人工工作量, 也使得机器人有了基本的" 智能".

3.4 小结

本章介绍了常用的机器人工作空间描述方式, 包括栅格地图, 几何地图与拓扑地图.

在栅格地图存储方式中, 提出了利用稀疏矩阵与空间搜索树进行存储的方式, 有效减少了栅格地图的存储资源占用且不引入过高的时间复杂度.

针对栅格地图障碍物描述不紧凑, 占用空间过大的缺点, 本文介绍了基于线段的几何地图存储与构建方式, 介绍了利用 openCV 函数库进行高效几何地图构建的方法, 并介绍了几何地图的特点.

拓扑地图, 作为机器人工作空间的" 骨架", 可以大大减少路径规划与导航算法的时间复杂度与问题规模. 在本章中给出了拓扑地图构建的大致方法, 拓扑地图的存储方式, 并提出了对构建结果的评价准则. 在本节最后介绍了拓扑地图的特点与应用场合. 在4中对地图分割方法进行了详细的介绍.

4 基于谱聚类的地图分割方法

为了减少机器人路径规划, 导航算法的时间复杂度与问题规模. 需要将全局地图分割为多个关联度较低的子图, 在拓扑地图构建章节描述了一个良好的地图分割所应当具有的性质, 如图12所示. 在常见的机器人工作环境中, 以房间 -走廊为代表的环境模型是十分常见的. 这种地图分割方式也与人类对环境的认知相符.

本文所讨论的地图分割问题可以做如下表述:

机器人工作空间内的障碍物由栅格或线段表示, 全局地图 Map 为障碍物集合, 对于栅格地图, 全局地图中包括所有栅格的中心点 (x, y). 对于几何地图, 全局地图为线段集合 $\{(x_i, y_i)\}$. 在机器人工作空间中存在一系列观测点, 表示为集合 $Obv = \{p_1, p_2, \cdots, p_n\}$. 求取集合 Obv 的一个划分:

$$Participation = \{Obv_1, Obv_2, \cdots, Obv_n\},\ Obv_1 \cdots Obv_n \subseteq Obv,\ Obv_i \cap Obv_j = \varnothing \quad (13)$$

在该划分中, 每个子集中的观测点应尽量属于同一房间. 在一些情况下, 对地图中”房间” 的概念很难做出明确的定义. 在这种情况下对地图分割算法的要求为:

- 生成的各个子图所包含的观测点的凸包彼此不相交 (或尽可能不相交)
- 每个观测点对应唯一的子图
- 子图间的关联尽可能小, 对子集 Obv_i, obv_j 如式14, 15所示, 并使15最小:

$$\begin{cases} d(p_i, p_j) = 0 & CanBeObserved(p_i, p_j) = True \\ d(p_i, p_j) = 1 & CanBeObserved(p_i, p_j) = True \end{cases} \quad (14)$$

$$\sum_i \sum_j d(p_i, p_j) \quad r_i \in obv_i, r_j \in Obv_j \quad (15)$$

对于地图内任意点, 可以通过该点与观测点的关系判断该点归属. 对于区域归属的判定, 可以利用栅格分割或 voronoi 分割对地图进行划分, 并用地图划分中各个子区域中心的归属代替整个区域的归属.

4.1 谱聚类

谱聚类是最热门的聚类算法之一. 相对于传统的聚类算法, 例如 KMeans,Clara 算法, 谱聚类算法在局部特征识别, 脏数据敏感性与计算资源占用方面有着重大的优势. 谱聚类的基本方法是将每次观测数据作为无向带权图 G 上的一个节点, 观测结果之间的相似度作为图 G 中边的权重. 从而将聚类问题转化为图的划分问题.

谱分割算法的步骤如下:

1. 构造观测结果相似度函数, 并通过该函数计算相似度矩阵
2. 通过相似度矩阵构造无向带权图 G
3. 根据相似度矩阵构造归一化拉普拉斯矩阵
4. 从归一化拉普拉斯矩阵求解其最小的 K 个特征值与对应的特征向量
5. 如果 $k = 2$, 则通过对应的特征向量正负性将原样本划分为两类
6. 如果 $k > 2$, 则将对应的特征向量代替原观测结果进行聚类, 将特征向量的聚类结果作为原样本的聚类结果.

4.2 基于谱聚类的地图分割方法

4.2.1 构造相似度矩阵

如图13所示, 度量两次扫描结果的相似度, 本质上就是衡量两次扫描结果间的重合程度. 然而, 对于扫描结果这一复杂的多边形, 计算其重合面积是困难的, 在时间复杂度上也是不可接受的. 一个好的相似度度量函数应当权衡时间复杂度与精确性, 并且考虑到一系列特殊情况. 评价不同观测结果间的相似度是构造相似度矩阵的前提条件, 观测结果的评价方法可以利用以下几种方法实现:

1. Hausdroff 距离
2. 基于谱聚类的方法
3. 基于特征点的评价方法

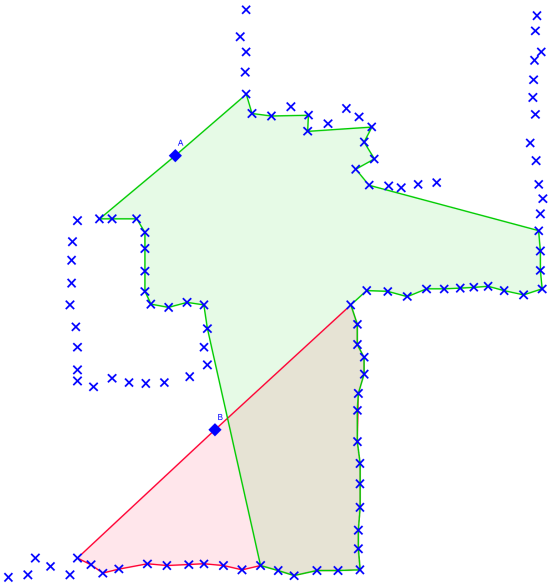


图 13: 在全局地图中的两次扫描结果, 这两次扫描结果有一部分是" 重叠" 的. 图中阴影部分为扫描结果, 叉号标记的是扫描到的障碍物坐标点, 菱形代表该扫描结果取得时机器人的位置

相似度评价函数的要求 为了在后面的计算中应用该函数, 并且使计算尽可能简单, 定义在观测结果集合 O 上的相似度评价函数 $S(O_i, O_j))$ 应当满足以下条件:

- $S(O_i, O_j)$ 满足自反性, 即 $\forall O_i \in O,\ S(O_i, O_i) = 1$, 由此定义相似度的最大可能性发生在两次观测结果完全相等的情况下, 此时函数值为 1
- $S(O_i, O_j)$ 满足对称性, 即 $\forall O_i, O_j \in O,\ S(O_i, O_j) = S(O_j, O_i)$. 在构造相似度矩阵时通过该性质可以节省一半的计算量, 所以相似度评价函数应当尽量具备对称性

在观测结果集合上的相似度评价函数 $S(O_i, O_j))$ 通常不要求满足传递性. 假设机器人在一个障碍物分布均匀的环境内减速直线前进且机器人观测频率固定, 对于每相邻两次观测结果而言 $S(O_i, O_{i+1}))$ 与时间 t 呈递增趋势. 但是第一次观测结果与最后一次观测结果之间的相似度 $S(O_1, O_n), n = card(O)$ 与时间 t 呈递减趋势.

由于激光扫描仪每次扫描的结果数量是稳定的, 即 $card(O_i)$ 为一个不太大的常值 (在本机器人应用情况下, 每次扫描结果为 181 个点). 对于扫描结果大小 $n = card(O_i)$, $O(n^2)$ 的时间复杂度是可以接受的. 但由于相似度评价函数在程序中被大量调用, 所以还是要尽量减少时间复杂度. 例如基于链表的点集配准算法 (ICP 问题) 复杂度通常为 $O(n^2)$, 但是通过改进数据结构, 使用二叉搜索树或者 Kd-Tree 树可以将时间复杂度减少至 $n\log(n)$ 以下.

基于 Hausdroff 距离的相似度计算方法 度量两个点集" 相似度" 的常用算法还有 Hausdorff 距离. 其基本思想是度量两个点集的最大不匹配程度. 该距离定义如下:

$$\begin{cases} H(r_a, r_b) = \max(d(O_i, O_j), d(O_i, O_j)) \\ d(r_a, r_b) = \max(p_{im} \in O_i) \min(p_{in} \in O_j) \|p_{im} - p_{in}\| \\ d(r_b, r_a) = \max(p_{im} \in O_j) \min(p_{in} \in O_i) \|p_{im} - p_{in}\| \end{cases} \tag{16}$$

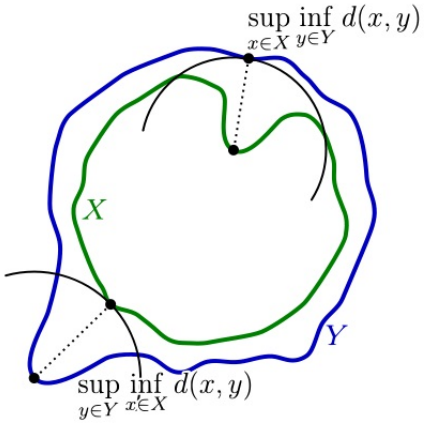


图 14: HausDroff 距离的意义如图, 图中两个 $\sup\inf d(x,y)$ 代表单边 Hausdroff 距离

上式中的 $h(r_a, r_b), h(r_b, r_a)$ 被称为单边 Hausdorff 距离, $H(r_a, r_b)$ 为两单边距离中较大者. 因单边距离不符合对称性原理, 取两单边距离中最大值使其具有对称性的运算规律. 除此之外针对 Hausdorff 距离量度计算量较大的问题还提出了改进的 Hausdorff 距离 (Rucklidge 算法等). Hausdorff 距离量度的优点在于其形式较简单, 其算法形式便于优化, 经过优化后速度较快.

根据以上算法可以得到空间中两个点集的" 相似度" 描述. 本文假定这种" 相似度" 具有对称性, 可以减少一半的计算量. 通过计算所有测量结果之间的相似度, 可以得到相似度矩阵 S.

基于 NearestNeighbor 算法的相似度计算方法 由激光传感器的特性可知, 对观测结果相似性的度量可以用两次观测中共同出现的障碍物个数代替. 令观测结果 O_i, O_j 的相似度为 $S(O_i, O_j)$, 则有:

$$S(O_i, O_j) = card(\{p1 \mid p1 \in O_i \; and \; p1 \in O_j\}) \tag{17}$$

但在实际操作中, 由于激光传感器的精度与环境干扰, 激光传感器几乎不可能两次扫描到同一个点. 如图15 故此需要算法进行两次扫描结果间的配准. 即对观测结果 O_i 中任一点 p 寻找观测结果 O_j, 或全局地图中中最邻近点 O_j. 通过度量配准结果间每组点的距离之和可得到两次扫描结果间相似度.

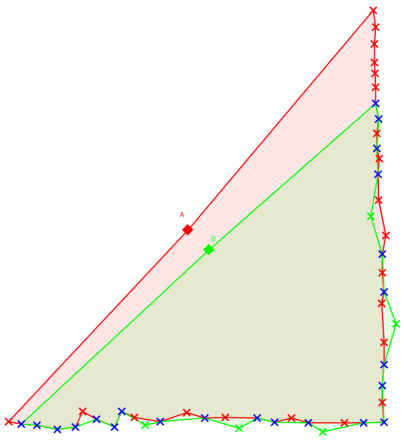


图 15: 激光传感器扫描中产生的交错现象, 红色的障碍物点表示仅被扫描结果 A 包含的点, 绿色点表明该点仅被扫描结果 B 包含, 蓝色点表明该点同时被 A, B 两次扫描结果包含

一个简单的计算方法是对观测结果 O_i 中的每一个点 p_i, 寻找在观测结果 O_j 中与 P_i 距离最近的点 p_j, 如果这两点距离 $\|p_i - p_j\|$ 小于给定值, 则用 P_j 代替 P_i. 如果 $\|p_i - p_j\|$ 大于给定阈值, 则将其最小值加入到相似度评价函数中.

伪代码如下:

Algorithm 1 求解观测结果间相似度的简单方法 (时间复杂度高)

Require: 观测结果 O_i, O_j, 给定阈值 Th
Ensure: 观测结果 O_i, O_j 的相似度 $S(O_i, O_j)$

```
function SimilarityAlgorithm_Easy(O_i, O_j, Th)
    result ← 0
    for each point p_i in O_i do
        minimum ← Inf
        for each point p_j in O_j do
            if ||p_i − p_j|| < Th then
                result ← result + 0
                Break
            else
                if |p_i − p_j|| < minimum then
                    minimum ← |p_i − p_j||
                end if
            end if
        end for
        result ← result + minimum
    end for
    return 1/result
end function
```

该算法相对简单也便于理解, 但是时间复杂度非常高, 达到了 $O(n^2)$. 分析上述算法可知大部分计算发生于对给定点 p_i, 寻找最近点 p_j 的计算中. 在使用链表作为数据结构存储点列的情况下, 这种计算是难以避免的. 所以考虑使用空间搜索树进行点集的存储,

利用额外的空间换取时间. 事实上, 相对于对响应时间的需求, 满足算法对内存的需求是相对简单的.

二维空间中空间搜索树的常用实现有四叉树, K-d 树 (在二维空间中为 2-d 树) 与 Ball-Tree 实现. 其中四叉树的实现相对简易, 但四叉树不是平衡树, 在四叉树中搜索一个给定点的算法其计算量是不稳定的. 在极端情况下 (所有数据点紧密分布在样本空间内一个小角落中) 其时间复杂度与链表存储方式相当.

K-NearestNeighbor 算法的思想是, 对需要配准的点集 O_i, O_j, 将 O_j 中元素进行聚类. 对 O_i 中任意元素 P_i, 查找与其最邻近的 K 个 O_j 中数据点的归属. 用这 K 个数据点中出现最多的簇作为 P_i 的归属. 该方法通过聚类有效减少了算法中需要比较的次数. 与上文给出算法相比, 该方法更加精确且效率更高. 在 Python 的 scikit-learn 机器学习裤中包含一个 K-NearestNeighbor 算法的高效实现.

基于特征点的相似度计算方法 基于特征点的相似度计算方法利用扫描结果中相对有代表性的点代替整个扫描结果. 利用不同观测结果之间共享的特征点数目计算观测结果的相似度. 特征点可以使用现实存在的点, 也可以使用虚拟的特征点进行计算.

在地形匹配与目标识别中常用的特征点有角点, SIFT 特征点, SURF 特征点等等. 从响应时间考虑采用角点作为特征点. SIFT, SURF 算法在时间复杂度方面不太理想. 同时, 我们希望选择的特征带有一定的空间语义学特性, 即选择的特征点在人眼中也是有一定" 意义" 的点. 角点恰好符合这一标准.

为了减少构造相似度矩阵的计算量, 可以将让提取特征点的过程与机器人扫描过程同时进行. 即在每一次扫描的过程中记录特征点的位置. 但由于扫描精度与噪声的原因, 该策略会产生一定的误差, 使不同观测结果间不能匹配, 如图16所示. 解决方法为在扫描过程中维护一公共特征点序列 (或特征点搜索树). 算法伪代码如下

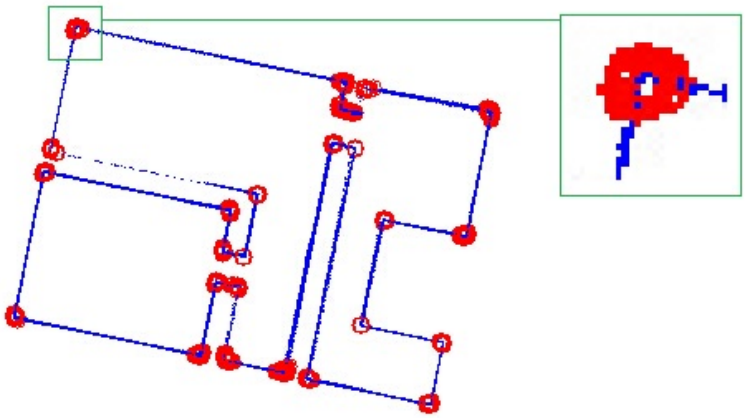


图 16: 在全局地图中的特征点识别结果, 方框部分表明在不同的扫描结果中特征点有一定的漂移现象

Algorithm 2 求解观测结果间相似度的简单方法 (时间复杂度高)

Require: 观测结果 O_i, 全局特征点序列 $FeaturePointList$, 给定阈值 Th
Ensure: 更新的全局特征点序列 $FeaturePointList$

```
function UpdateFeaturePoint(O_i, FeaturePointList, Th)
    result ← 0
    for each point p_i in O_i do
        if p_i is a feature point then
            for each feature point p_j in FeaturePointList do
                if ||p_i − p_j|| < Th then
                    Break
                else
                    FeaturePointList append p_i
                end if
            end for
        end if
    end for
    return NULL
end function
```

由于激光传感器的特点, 可以利用各个障碍物点到激光传感器的距离进行角点的提取. 首先获取各个障碍物点到激光传感器的距离 r_i, 如式1所示. 对数据进行平滑以去除毛刺. 然后对 r_i 求取一阶导数 r_i' 以获取其变化率. r_i' 导数的局部极大值与极小值即为特征点在观测结果中的索引值, 如图17所示:

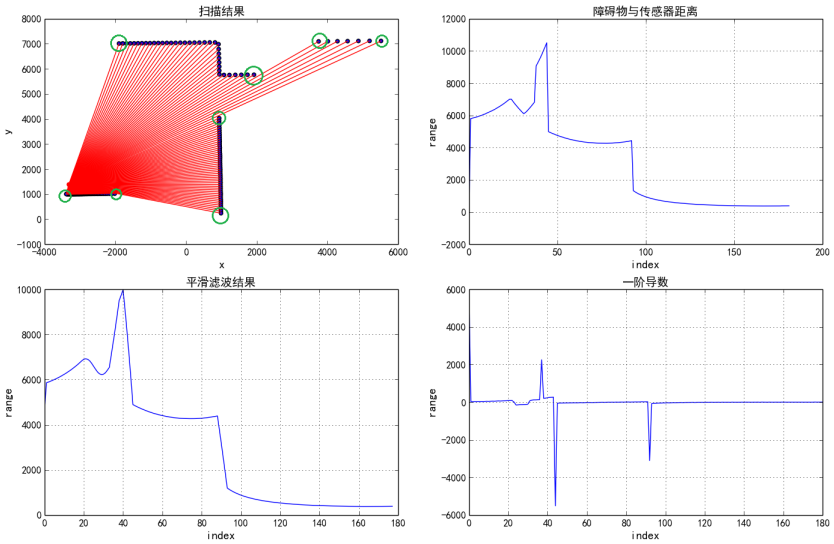


图 17: 子图 1 显示了机器人行进过程中的一次扫描结果 (机器人位置用红点表示), 子图 2 显示了各个障碍物点与机器人的距离, 子图 3 显示平均值滤波后的图 2 结果, 子图 4 显示了图 3 结果的一阶导数

在图17中, 一个需要额外注意的现象是由于障碍物的不连续性导致特征点的误判. 如图17中子图 1 绿色圆圈部分所示: 在相似度计算函数中这些点不影响相似度的计算. 但

因为这些点没有语义学特性, 而且随着机器人的移动而消失, 所以在观测结果中将这些点去除. 可以采用以下策略:

- 如果特征点在观测结果的前 2 个或最后两个中, 则不接纳此结果
- 如果该特征点附近伴随着距离突变, 则不接纳此结果
- 如果某个特征点值在只在一次观测中出现, 在其相邻的观测结果中均未发现此特征点, 则去除该特征点

基于特征点的观测结果相似度计算本质是计算两次观测结果间共同出现的特征点数量. 如下式所示:

$$S(O_i, O_j) = card(\{p \,|p \in featurePoint(O_i)\ and\ p \in featurePoint(O_j)\}) \tag{18}$$

算法伪代码如下:

Algorithm 3 基于特征点的观测结果相似度度量方法

Require: 观测结果 O_i, O_j 的特征点列 $featurePoint(O_i), featurePoint(O_j)$
Ensure: 观测结果 O_i, O_j 的相似度 $S(O_i, O_j)$

```
function SIMILARITY_FEATUREPOINT(featurePoint(O_i), featurePoint(O_j))
    result ← 0
    for each point p_i in featurePoint(O_i) do
        for each point p_j in featurePoint(O_j) do
            if p_i == p_j then
                result ← result + 1
            end if
        end for
    end for
    return result
end function
```

需要注意的是, 该算法产生的相似度并非落在 $(0, 1]$ 区间内, 需要进行进一步标准化. 该算法时间复杂度仍然为 $O(n^2)$, 然而因为特征点的数量要远小于障碍物点数量, 所以算法的实际运行时间也大大减少了. 与 KNN 算法类似的, 也可以通过使用空间搜索树代替链表以减少时间复杂度.

基于虚拟特征点的相似度度量 角点在机器人环境识别中是一类非常重要的点, 但事实上, 用于建构相似度判别准则的观测点 (特征点) 可以是任意可测量的点. 由观测模型图中可知, 空间中任意点都可以作为特征点进行相似度的计算. 将机器人可能运动的空间划分为 $m \times n$ 个网格, 令网格的中心店为虚拟的特征点 (实际在这一点并没有障碍物), 利用这些点进行相似度的计算. 相对于使用障碍物角点作为特征点, 该方法有如下优势:

- 真实激光扫描器扫描角度通常在 100 度到 180 度之间, 全向激光扫描器的价格过于昂贵. 而利用虚拟特征点的方式可以避免图所示的, 因扫描角度导致的过分割现象
- 该方法可以避免由特征点分布不均匀导致区域分割结果变动的情况. 图分割算法在处理相似度矩阵时, 考虑的是全局的相似度. 但在空间中角点的分布通常是不均匀的. 由于障碍物形状的原因, 在一个区域内角点可能特别集中. 这样的区域就使得地图的其他区域在图中的" 权重" 下降, 进而使得这些区域出现过分割现象.

- 该方法将实际的观测结果抽象为" 虚拟机器人位置" 与" 虚拟特征点", 实现了 SLAM 过程与地图分割过程的解耦. 使得该算法可以应用在更广阔的领域中.

该方法要求机器人能够知晓一个虚拟特征点是否是可以被观测到的. 该问题可以被分为两个子问题, 即:

1. 机器人能够知晓该虚拟特征点的位置
2. 该虚拟特征点是否在机器人的扫描范围内

问题 (1) 对于一个构建良好的地图来说是容易解决的. 如果 SLAM 过程中位置精度没有出现很大的飘移, 则每个虚拟特征点在地图中的位置应当是唯一确定的.

问题 (2) 由激光扫描仪的工作方式可知, 一个虚拟特征点 (x_{vp}, y_{vp}) 能被观测当且仅当该特征点与机器人位置 (x, y) 构成的线段附近没有障碍物. 判定线段附近是否有障碍物的方法有以下两种:

1. 在栅格地图中, 可以利用 Bresenham 算法判定一条给定线段上是否有障碍物, 但在栅格层面上只判断一条线段可能会出现交叉情况, 比较好的方法是判断紧邻的多条线段上是否存在观测到的障碍物点. Bresenham 算法提供了一种高效的计算离散化线段的方式. 也是计算机内部描绘线段的方式. 但在 CPU 中, 由于没有专门硬件处理该算法, 导致算法计算时间稍高.
2. 在基于线段的几何地图中, 一个虚拟特征点 (x_{vp}, y_{vp}) 能被观测当且仅当该特征点与机器人位置 (x, y) 构成的线段与任意一条障碍物线段间没有交点. 在实际计算中不用求取线段的交点, 只需要判断线段间是否彼此跨立. 进一步减少了计算的复杂度. 该方法时间复杂度比基于栅格地图的实现要小, 所以在本文中使用该方法进行地图的分割.

该算法步骤如下:

1. 由机器人观测结果生成几何地图
2. 生成用于地图分割的观测点集 $WayPoints = \{(x_1, y_1), (x_2, y_2), \cdots)\}$
3. 生成空相似度矩阵 M, 大小为 $card(WayPoints)$
4. 定义布尔函数 $CanbeObserved(p_1, p_2)$, 参数为两坐标, 若这两坐标所在位置可以互相观测到, 函数取值为 $True$, 否则为 $False$. 特别的, 定义 $CanbeObserved(p_1, p_1) = True$
5. 对任意观测点 i, 记录其能观测到的其他观测点集合 Obv_i:

$$Obv_i = \{p \mid p \in WayPoints\ if\ CanBeObserved(i, p)\} \tag{19}$$

6. 对于任意两个观测点, 记这两观测点序号为第 i 个与第 j 个:
   (a) 若 $i < j$, 则 $M_{ij} = M_{ji} = card((x, y) \mid (x, y) \in Obv_i\ and\ (x, y) \in Obv_j)$
   (b) 若 $i = j$, 则 $M_{ij} = 0$, 即使 M 主对角线元素为零
   (c) 若 $i > j$ 不进行任何操作, 由于该相似度度量满足对称性, 所以可以减少一半计算量.
7. 归一化矩阵 M, 利用该相似度矩阵进行地图分割.

一个巧妙的, 检测两线段彼此跨立的方法 ($CanBeObserved$ 函数实现思路) 如下, 该算法通过检查每四个顶点中的三个顶点排列顺序为顺时针排列或逆时针排列. 两线段 AB,CD 彼此跨立当且仅当 ACD, BCD, ABC, ABD 两组顶点每组旋转方向相反, 伪代码如下:

Algorithm 4 线段跨立检测算法

Require: 两线段对应的四个顶点坐标, 即线段 1:$(A_x, A_y), (B_x, B_y)$, 线段 2:$(C_x, C_y), (C_x, C_y)$

Ensure: 路径点集合 $Path$

1: **function** CCW($(A_x, A_y, B_x, B_y, C_x, C_y)$)
2: **Return** (Cy-Ay)*(Bx-Ax) > (By-Ay)*(Cx-Ax)
3: **end function**
4: **function** INTERSECT($(A_x, A_y, B_x, B_y, C_x, C_y)$)
5: **Return** ccw(Ax, Ay, Cx, Cy, Dx, Dy) != ccw(Bx, By, Cx, Cy, Dx, Dy) and ccw(Ax, Ay, Bx, By, Cx, Cy) != ccw(Ax, Ay, Bx, By, Dx, Dy)
6: **end function**

4.2.2 计算归一化拉普拉斯矩阵

基于上文所述的各种相似度计算算法, 可以构造相似度矩阵 S, 定义如下:

$$S = \begin{bmatrix} S_{11} & S_{12} & \cdots & S_{1n} \\ S_{21} & S_{22} & \cdots & S_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ S_{n1} & S_{n2} & \cdots & S_{nn} \end{bmatrix} \tag{20}$$

通过相似度矩阵 S 可以建立无向带权图 G, 即为所有测量结果对应的无向图, 如图18所示. 为了进一步简化计算, 定义接纳比例 d_i, 令矩阵 S 中最大元素为 S_{max}, 最小元素为 S_{min}. 做以下变换:

$$switch = S_{min} + d_i(S_{max} - S_{min}) \tag{21}$$

$$S_{ij} = \begin{cases} 0 & s_{ij} \leqslant switch \\ 1 & s_{ij} > switch \end{cases} \tag{22}$$

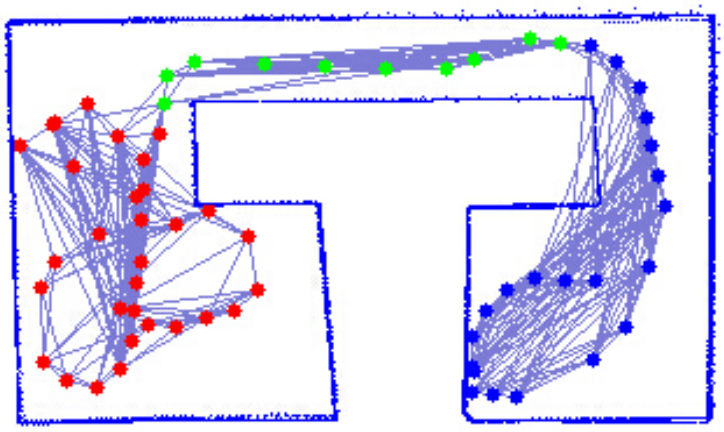


图 18: 在简单环境下的相似度拓扑图, 图的节点表示观测结果, 节点之间边代表这两个节点间相似度大于给定阈值 (图中接纳比例为 0.95, 即只取相似度前 5% 的边). 节点的颜色代表其被划分的区域

可以从 S 矩阵计算拉普拉斯矩阵 L. L 矩阵定义如下:

$$L(G) = D(G) - S(G) \tag{23}$$

即对顶点数为 n 的图 G, $L(G)$ 有以下定义:

$$L_{ij} = \begin{cases} deg(v_i) & i = j \\ -S_{ij} & S_{ij} \neq 0 \; and \; i \neq j \\ 0 & other \end{cases} \tag{24}$$

上式中 $D(G)$ 是图 G 的度矩阵, 形式如下, 式中 $d_1, d_2 \cdots d_n$ 为图 G 中各个节点的度数:

$$D(G) = \begin{bmatrix} d_1 & 0 & 0 & 0 \\ 0 & d_2 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & d_n \end{bmatrix} \tag{25}$$

根据文献 [18], 归一化拉普拉斯矩阵有如下两种定义, 两种定义非常相似且在应用中不作区别. 为计算简便, 本文中使用 L_{rw}:

$$L_{sym} = D^{-\frac{1}{2}} L D^{-\frac{1}{2}} = 1 - D^{-\frac{1}{2}} S D^{-\frac{1}{2}} \tag{26}$$

$$L_{rw} = D^{-1} L = I - D^{-1} S \tag{27}$$

在计算 L_{rw} 时需要注意矩阵 D 可能出现奇异矩阵使得求逆操作失败, 解决方法有两个:

- 在构造相似度矩阵时注意主对角线元素不要设为 0(设为 1 是一个比较保险的做法)
- 如果还是存在奇异矩阵, 将该矩阵加上一个大小相同的对角矩阵, 对角矩阵中元素取一个比较小的值 (本算法中取 0.01)

4.2.3 谱聚类算法

谱聚类算法如下所示:

- 通过样本间相似度矩阵 S 计算拉普拉斯矩阵 L.
- 计算归一化拉普拉斯矩阵 L_{rw}.
- 设原图 G 被分割为 k 个子图, 则寻找 L_{rw} 的前 k 小的特征向量, 记为 $u_1, u_2, \cdots, u_n$.
- 构造矩阵 $U \in \mathbf{R}^{n\times k}$, 将向量 $u_1, u_2, \cdots, u_n$ 作为该矩阵的每一列 (排列方式不限)
- 构造矩阵 $F \in \mathbf{R}^{n\times k}$, F 中每一行为 U 中对应行的归一化结果. 即:

$$F_{ij} = \frac{1}{\sum_{p=1}^{k} \|F_{ip}\|} \tag{28}$$

- 将矩阵 T 的每一行作为一个样本, 进行聚类分析, 常用 $K-Means$ 算法进行聚类. 第 k 行的聚类结果即为第 k 个元素的归属.
- 根据聚类结果将观测结果划分至各个族中.

4.2.4 聚类效果的评判准则

为评价聚类算法在给定样本上的效果好坏, 需要一种评判准则对聚类结果进行评价. 轮廓系数 (Silhouette Coefficient), 是聚类效果好坏的一种评价方式, 它结合内聚度和分离度两种因素. 可以用来在相同原始数据的基础上用来评价不同算法, 或同意算法的不同参数对聚类结果所产生的影响.

定义样本空间 X 的聚类结果为样本空间 X 的一个划分, 即:

$$cluster(X) = \{X_1, X_2, \cdots, X_n\} \tag{29}$$

要求 $cluster(X)$ 满足:

$$\begin{cases} X_1, X_2, \cdots, X_n \subset X \\ X_1 \cup X_2 \cup X_3 \cdots \cup X_m = S \\ X_i \cap X_j = \phi \qquad \forall i \neq j \end{cases} \tag{30}$$

对样本空间中任意元素 $x_i \in X_j$, 定义凝聚度 a_i 和分离度 b_i, 如式31, 32:

$$a_i = \frac{\sum_{m=1}^{|X_j|} d(x_i, x_m)}{|X_j|} \tag{31}$$

$$b_i = \min(\frac{\sum_{m=1}^{|X_j|} d(x_i, x_m)}{|X_j|}) \quad j = 1, 2, \cdots |X|, j \neq i \tag{32}$$

对于第 i 个元素 x_i, 计算 x_i 与其同一个簇内的所有其他元素距离的平均值为凝聚度, 选取 x_i 外的一个簇 b, 计算 x_i 与 b 中所有点的平均距离, 遍历所有其他簇, 找到最近的这个平均距离, 记作 b_i, 用于量化簇之间分离度.

对于元素 x_i, 轮廓系数 s_i 为:

$$s_i = \frac{b_i - a_i}{max(a_i, b_i)} \tag{33}$$

计算所有 x 的轮廓系数, 求出平均值即为当前聚类的整体轮廓系数.

不难发现若 s_i 小于 0, 说明 x_i 与其簇内元素的平均距离小于最近的其他簇, 则聚类效果不好, 如果 a_i 趋于 0, 或 b_i 足够大, 则 s_i 趋近于 1，说明聚类效果比较好.

4.2.5 自适应聚类方法

在谱聚类算法中, 通常的做法是给定聚类的簇数, 即 k 的值. 然后通过 K-Means 算法对 Laplace 矩阵的前 k 小特征值对应的特征向量进行聚类. 一个改进的方法是给定聚类簇数的范围, 在这个范围内对特征向量进行多次聚类, 并选取轮廓系数最大的一次聚类为聚类结果.

针对 K-Means 聚类结果不稳定, 结果依赖初始值严重的问题. 也可以在簇数一定的情况下用随机初始值多次聚类, 选取轮廓系数较大的一次作为聚类结果. 因为聚类算法的样本, 即 Laplace 矩阵前 k 小的特征值大小有限, 特别是样本的维度很小. 聚类的时间开销相对较小. 在7章的图40中展示了轮廓系数与聚类效果的关系.

轮廓系数作为一个评价聚类效果的指标可以度量聚类结果的优劣, 但轮廓系数的也有一定的不足之处. 轮廓系数的评价准则" 偏爱" 呈圆形的聚类, 而机器人工作空间中经常出现走廊, 管道等长条状的环境. 对于 K-Means 分割而言, 这种" 偏好" 是不可接受的.(因为 K-Means 聚类本身也倾向于圆形的聚类). 但在谱分割中, 由于谱分割中聚类的内聚度远大于蔟间的分离度, 所以能够正确的对环境进行聚类, 如图19所示, 自适应聚类算法成功将测试环境分为三部分, 分别用红绿蓝三种颜色表示.

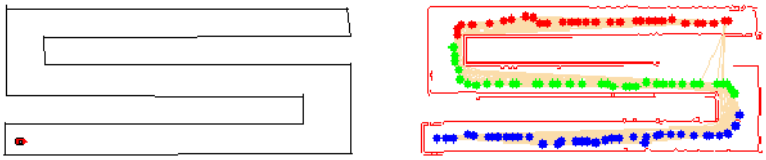


图 19: 自适应聚类对长条形环境测试

4.3 小结

本章介绍了基于谱聚类的地图分割方法, 用于对全局地图进行划分, 并生成各个子区域间的拓扑结构, 最终生成用户友好的, 信息层面上高内聚低耦合的拓扑地图. 谱聚类作为一种热门的, 有效的聚类方法相对 K-Means 聚类算法有着独特的优势. 本章对谱聚类算法的流程与其在地图分割方面的应用进行介绍.

在地图分割算法运行过程中, 其性能瓶颈与制约地图分割好坏的因素在于相似度矩阵的构建. 构建良好的, 符合一定条件的相似度矩阵可以产生很好地地图分割结果. 构建不当的相似度矩阵会产生错误的分割结果, 或产生奇异矩阵使得算法失败. 本文对相似度函数的性质进行了一定的讨论, 并提出了四种相似度评价准则, 分别为:

- 基于 Hausdroff 距离的相似度评价方法
- 基于 NearestNeighbor 算法的相似度计算方法
- 基于特征点的相似度计算方法
- 基于虚拟特征点的可视性相似度计算方法

针对传统谱聚类需要外界给定聚类簇数的缺点, 本文提出了自适应谱聚类算法, 该方法通过轮廓系数这一评价准则对聚类结果进行评价, 并选取较好的结果输出.

5 机器人导航算法

在路径规划与导航领域, A-star 算法是一种比较好的启发式最短路径搜索算法. 与广度优先搜索算法相比, 该算法利用启发函数规划搜索方向, 减少算法运行时间以及节点占用存储资源. 在启发函数满足可接纳性的条件下, 该算法得到的结果是全局最优的. 在地图中障碍物较少, 最优路径比较平直的情况下, 该算法可以直接找到最优路径. 该算法时间复杂度与空间复杂度均近似 $O(n)$ 数量级 (n 为最优解的路径长度)

5.1 路径点生成

Astar 算法是基于路径点的图搜索算法, 要求在地图中产生一系列合适的导航路径点. Astar 算法启发式的搜索这些路径点, 并返回最优路线中包含的路径点. 由于路径点选取的过程中不可能所有点都恰好在全局最优路线上, 所以该算法得到的最短路径与最优路径间具有一定偏差. 在路径点充足的情况下, 该误差是可以接受的. 路径点越密集该误差越小, 但路径点密集会带来存储空间与运算时间的更高开销. 在空间中用图 $G < V, E >$ 存储路径点与路径点间的连接关系, 其中 $V = \{p_1, p_2, \cdots, p_n\}, p_i = (x_i, y_i)$ 代表空间中的路径点. $E = \{(p_i, p_j) \mid p_i, p_j \in V\}$ 表示路径点间的邻接关系. 算法返回路径点集合 $Path \subseteq V$, 表示最优路径所经过的路径点.

为尽量使路径点分布均匀, 使用基于栅格的路径点分布算法进行路径点配置. 设地图大小为 $Rect(W, H)$, 网格大小 d, 给定任一点 $P(x, y)$, 及最邻近路径点坐标为:

$$\begin{cases} i = \lfloor \dfrac{x}{d} \rfloor & (x < W) \\ j = \lfloor \dfrac{y}{d} \rfloor & (y < H) \end{cases} \tag{34}$$

每个路径点与其他路径点的可达性关系有如下三种配置策略:

1. 每个路径点与其上下左右四个路径点 (4-邻域) 有可达关系 (如果没有障碍物的话)
2. 每个路径点与其周围八个路径点 (8-邻域) 有可达关系 (如果没有障碍物的话)
3. 利用4.2.1小节中定义的可视关系, 定义每个路径点的可达路径点为其可视路径点.

方法 1,2 计算量较低, 但需要额外的算法迭代次数. 方法 3 计算量稍大, 但可以明显的减少迭代次数. 在比较极端的例子中, 如果起始点与终止点之间有直线路径, 方法 3 只需要一步遍历就可以得到结果, 而方法 1, 2 需要遍历最短路径上的每个路径点. 但在地图较大的情况下, 计算可视关系所花费的计算量是难以接受的. 一个妥协的方法是在地图足够大的情况下, 计算每个路径点周围 n 步长路径点的可视关系.

5.2 启发函数

在启发式算法中, 每一个路径点 p 包含一个估价函数值 $f(p)$. 估价函数 $f(p)$ 表明算法将该节点加入结果所需要付出的代价. $f(p)$ 的形式为:

$$f(n) = g(n) + h(n) \tag{35}$$

上式中 $g(n)$ 表示扩展到该节点已经付出的代价, $h(n)$ 为启发函数, 表示该节点到目标所需代价的估计值. 启发函数的选取对算法表现的影响非常大. 通常, 启发函数需要满足下列条件:

- 可接纳性 (Admissibility):

$$h(n) \leqslant h^*(n) \tag{36}$$

可接纳性意味着一个节点到终点的估计代价不能大于实际代价, 该性质保证最短路径不会因为错误的代价估计而被放弃. 启发函数只有满足该性质, 搜索结果才能保证为最短路径.

- 单调性 (Monotone Condition)

$$h(n_i) \leqslant h(n_j) + cost(n_i, n_j) \tag{37}$$

单调性保证最优路径上估价函数值是单调递减的, 这使得启发函数能正确的指引搜索方向. 一个单调, 并且在搜索路径中均匀变化的估价函数可以有效的减少算法的运行时间. 非单调的启发函数会导致额外的时间开销, 但不会生成错误的路径.

一个良好的启发函数可以大大减少搜索的步数与搜索的节点数. 从信息论的角度来讲, 距离函数包含了一定的信息, 这些信息使得 Astar 算法与广度优先算法相区别并赋予其相对" 智能" 的导航能力. 然而, 一个良好的 (也就是包含的信息比较多) 的启发函数通常具有较大的计算成本. 因为启发函数本身包含的信息是算法的主要决策部分. 当启发函数为常值 (不包含任何信息) 时, 该算法就退化为广度优先搜索算法. 当启发函数精确的等于实际距离时, 该算法就变成 BFS(Best-First) 算法. 在一般导航算法中, 欧氏距离因满足可接纳性与单调性而被广泛接受. 其形式简单, 便于理解. 但缺点是欧氏距离运算过程中出现了开方运算, 在计算机中需要花费额外的运算时间. 另一个常用的启发函数是汉明距离, n 维空间中汉明距离由下式表示:

$$h_{Hamming}(X_i, X_j) = \sum_{i=1}^{n}(|X_{im} - X_{jm}|) \tag{38}$$

另一个比较有趣的启发函数形式是基于角度的启发函数. 基于角度的启发函数常用在矢量空间求最短路径中. n 维空间中其启发函数形式如下:

$$h_{hamming} = cos(X_i, X_j) = \frac{X_i \cdot X_j}{\|X_i\|\|X_j\|} = \frac{\sum_{p=1}^{n} X_{ip} \times X_{jp}}{\sqrt{\sum_{p=1}^{n}(X_{ip}^2)} \times \sqrt{\sum_{p=1}^{n}(X_{jp}^2)}} \tag{39}$$

该函数的有趣之处在于寻路算法会将方向作为启发因子的一部分, 在扩展节点的过程中会优先扩展与目标节点方向相同的节点. 在地图中障碍物较少的情况下会更快的寻找到目标点而且大大的减少抖动情况. 但是该函数会引入额外的计算代价. 一个取巧的办法是将角度离散化, 离散成 30 或 15 度的倍数, 这样就简化了三角函数操作.

在实际应用中, 需要对启发函数的精确性与计算成本做出平衡. 例如, 一个高实时性的系统通常无法容忍一个计算时间极长的算法, 这相当与在系统的主回路中串联一个大的积分环节, 会极大的影响系统的频率响应性能. 甚至在一些系统中, 要求算法的性能要保持在非常狭窄的时间窗口中. 即对于任意输入, 算法的输出时间应当不大于规定上上界也不小于规定下界.

5.3 Astar 算法

Astar 算法利用两个链表 (在优化的算法中, 可以使用诸如哈希表或堆来进行优化)$OPEN$ 与 $CLOSE$ 来表示搜索的区域, 其中 $OPEN$ 表表示即将搜索的区域,

$CLOSE$ 表表示已搜索区域. 对 $CLOSE$ 表中任意节点, 可以认为已经得到了开始点到该点的最优路径. 在地图上, $CLOSE$ 表中节点为已搜索区域的" 内部", $OPEN$ 表表示已搜索区域的边界. 在 AStar 算法中, 不断地计算 $OPEN$ 表中行进代价最小的节点所可能扩展的节点的到达代价. 最终选取一条路径其经过的路径点到达代价最小.

Astar 算法流程表示如代码5.3所示:

Algorithm 5 Astar 启发式最短路径搜索算法

Require: 路径点网络 G, 起始路径点 p_{start}, 结束路径点 p_{end}
Ensure: 路径点集合 $Path$

```
function ASTAR(V, E)
    初始化列表 OPEN, CLOSE
    将 p_start 插入 OPEN 表
    OPEN.value ← f(p_start)
    while OPEN 表非空 do
        路径点 n ← OPEN 表中估价值最小的节点
        if n = p_end then
            while n 不是 p_start do
                将 n 插入 Path 中
                n ← n 的父节点
            end while
            Break 搜索结束
        end if
        for 当前节点 n 的每一个子节点 x do
            x.value ← x 的估价值 f(x)
            if x ∈ OPEN then
                if x.value < OPEN.value then
                    n 为 x 的父节点
                    OPEN.value = x.value
                end if
            end if
            if x ∈ CLOSE then
                continue
            else
                n 为 x 的父节点
                将 x 插入 OPEN
            end if
        end for
        将 n 节点插入 CLOSE 表
    end while
end function
```

在算法的最后, 需要通过算法中给出的节点关系, 回溯出整条路径. 在算法运行过程中, 对路径上的节点赋予了一些父子关系. 如图20所示. 每次计算一个节点时, 会将其可达的, 代价最小的点作为其子节点. 因为在算法过程中向不同方向进行探索, 所以一个节点可能有多个子节点. 但一个子节点的父节点是一定的 (每个节点到起始点的最短路径是唯一的, 因为算法控制当存在两条等长路径时, 第二条不会被计算) 在算法找寻到子节点时, 通过子节点向上追溯父节点, 直到追溯到起始点, 该路径即为最短路径. 在导航测试章节, 演示了 Astar 算法是如何让寻找到最短路径的.

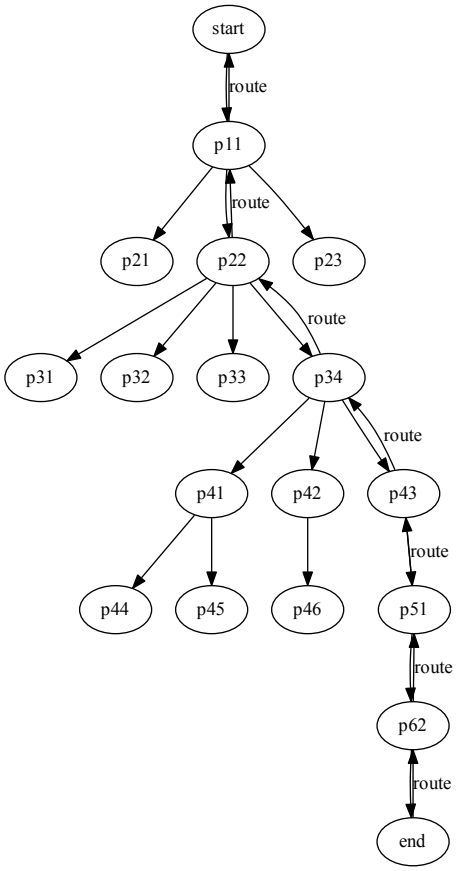


图 20: Astar 回溯路径示意图

5.4 分层 Astar 算法

随着地图的增大, Astar 用以进行导航的路径点数量平方倍增加, 导致额外的存储空间开销. 同时, 更大的地图意味着更长的最优路径, 更多的迭代次数, 更多的障碍物. 并且在一个大地图中可能搜索的无效路径与无效区域也相应的增加了. 一个优化的解决方案是将全局地图分割成一系列区域, 首先寻找最优路径可能经过的区域, 而后进行区域层次的最优路径规划. 该方法有以下优势:

1. 由于路径点数量与地图尺寸的平方倍成正比, 将全局地图分割成一系列局部地图会大大减少用于导航的路径点数量.

2. 一个良好的地图分割算法其分割的每一个小区域内只有很少的障碍物, 即对于每一个小区域, 搜索的时间开销都是很小的

3. 分层的 Astar 算法可以避免搜索一些最优路径不会经过的区域. 如果在区域层面上已经规划了最优路径, 则分层 Astar 算法只会搜索最优路径上的区域, 大大降低了算法的运行时间与存储资源开销.

4. 分层次导航算法便于进行并行化处理, 可以充分利用大型机器人计算资源优势. 对于本实验使用的 peopleBot 机器人, 其内置了 intel-i7 八核心处理器并且能够进行扩展, 只使用单核心进行运算造成了大量的资源浪费

5.5 小结

本节介绍了 Astar 算法在机器人路径规划领域中的应用. Astar 算法作为一种启发式算法可以高效率的寻找解空间内的最优解. 本文介绍了 Astar 算法的基本过程, 并对 Astar 算法中中启发函数的选取进行了讨论. 针对 Astar 算法在机器人导航领域中的应用, 本文给出了基于栅格的八连通导航路径点生成方案. 在本章最后, 对基于拓扑地图的分层 Astar 算法进行了介绍.

6 系统的设计与实现

6.1 平台搭建

本文中使用的机器人为 Aria Mobile Robot 公司的 PeopleBot-sh 机器人. 仿真实验使用 RoboSim 机器人仿真平台进行. 编程环境使用 Python 2.7.6 作为主要编程语言. 本文中搭建的实验平台使用了如下外部库.

1. Aria.py, Aria.dll:Aria Mobile Robot 公司提供的 API, 用于进行该公司生产机器人的运动控制
2. OpenCV: 机器人图像处理, 矩阵运算, 轮廓与边缘处理, 目标提取
3. OpenGL:3D 场景构建, 机器人地图显示
4. numPy, sciPy: Python 科学计算库, 提供部分数学运算函数, 并提供了求取矩阵特征值, 特征向量的函数
5. matplotlib: Python 数学绘图库, 用于绘制图表
6. scikit-learn: Python 机器学习库, 提供了高效的机器学习算法实现 (本平台中使用了 KMeans, NearestNeighbor 算法)
7. NetWorkX: Python 网络分析库, 提供了拓扑网络的支持与基本操作函数

以上外部库均根据 GPL 协议开源并可以免费获得, 这些库函数同时也是各自领域的行业标准. 搭建实验环境步骤如下:

- 安装 Python2.7.6 版本 (Python2.x 版本与 Python3.x 版本相互不兼容, 部分库在 Python3.x 上有兼容性问题)
- 将 Python 安装目录加入环境变量
- 安装 Python 包管理软件 pip, 该软件可以从 Python 官方网站上下载
- 安装 Python 科学计算库 NumPy, SCIPy, 绘图库 MatPlotLib, 可以通过二进制安装包安装, 也可以使用 pip 进行安装. 二进制安装包可以在http://www.lfd.uci.edu/~gohlke/pythonlibs/获取并安装, 也可以在 shell 下用以下指令进行安装:

```
1 pip install numpy
2 pip install scipy
3 pip install matplotlib
```

- 安装 Python 机器学习库 SCIKit-learn, 在 shell 下利用 pip 进行安装
- 安装 Python 网络与图处理包 NetWorkX, 在 shell 下利用 pip 进行安装
- 安装 openCV 并配置环境变量
- 安装 ARIA, MapperBasic, MobileSim 软件, 通过二进制安装包进行安装.
- 配置 ARIA 库, 将../ARIA2.9.0-1(32-bit)/bin目录添加进环境变量

- 将 ARIA 对应版本, 对应操作系统的 Python 接口 (在../ARIA2.9.0-1(32-bit)/Python) 放到工程文件夹中
- 启动 MobileSim, 载入任意地图并设置机器人初始位姿
- 运行 main.py, 在 gui 界面中可以控制机器人运动并记录传感器数值
- 运行 generateRobotPicture.py 生成地图分割结果

工作目录下主要文件:

- Aria.py　Aria 机器人控制程序的 Python 接口库
- main.py　控制机器人方向与实时运行的主程序
- generateRobotPicture.py　该程序从机器人的观测数据生成地图分割结果, 并进行显示
- robotTools.py　包含了一系列常用的函数
- GUI.py　提供了一部分界面显示的代码
- *.DLL　由 Aria 提供的动态链接库, 包括 Aria 核心代码及封装, Python 接口调用, 对应 VC 各个版本的函数库.
- tempFile 文件夹　临时文件存放位置
  - RobotPicture02.pkl　机器人运行过程中激光传感器读取数据记录于此文件中, 被用来生成地图分割结果. 激光传感器数据以追加方式写入文件中, 切换地图时删除该文件.
    * 读取: generateRobotPicture.py
    * 写入: main.py
  - linkedMat.pkl　聚类生成过程中存储的中间文件 (Laplace 矩阵), 为了调整参数方便将其记录下来避免重复计算. 在该文件存在情况下 generateRobotPicture.py 脚本不会重复计算拉普拉斯矩阵. 所以在需要时 (激光传感器数据有变化时) 将其删除.
    * 读取:generateRobotPicture.py
    * 写入:generateRobotPicture.py

系统内各个模块作用, 相互调用关系与组织结构如图:

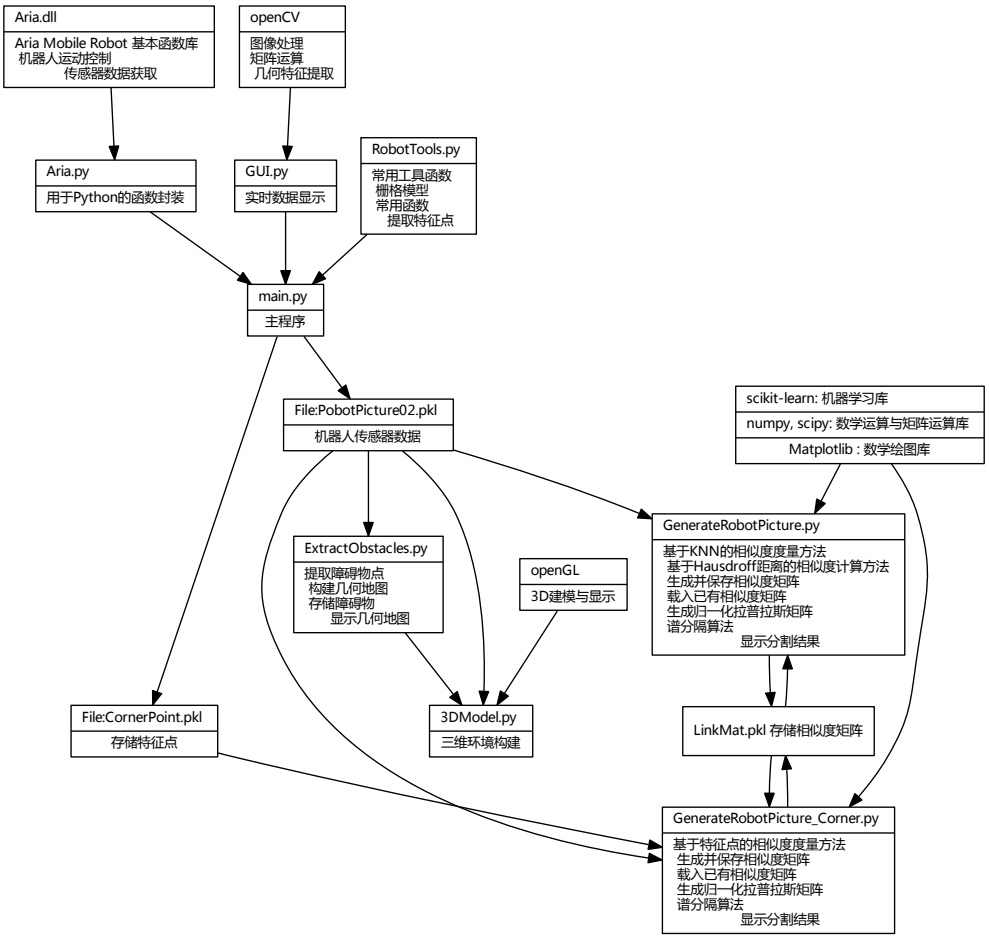


图 21: 系统内各模块作用示意图

6.2 数据采集与处理

通过 Aria Mobile Robot 公司提供的接口可以获取激光传感器数据, 如式3所示. 同时根据机器人内部里程计可以获取机器人位姿的估计 (x, y, θ).

机器人采集环境的过程可以手动操作, 也可以自动进行. 通过机器人面板上的按钮可以实现左转, 右转, 前进, 倒车与掉头功能. 同时机器人也可以在自动避障模式进行探索. 通过自己编写代码也可以实现自主避障探索策略, 在第6.3章中纤细叙述了如何实现一个机器人臂章探索策略.

6.3 机器人避障算法

本文所描述平台中实现了一个简单的避障探索算法. 该算法基于有限状态自动机描述, 具有效率高容易实现, 代码清晰, 容易修改的优点. 该算法将机器人最近一次的扫描结果分为四个区域, 即正前方, 前方, 前方周边, 周边四个相互重叠的区域, 如图所示:

同时, 将机器人可能进行的动作分为以下几种:

动作	对动作的说明
直行	机器人匀速前进
找最远可达方向	在机器人扫描结果中选择离机器人最远的障碍物点方向, 机器人一边前进一边尝试转至该方向
掉头	边转弯变前进, 是高效的转弯方式, 但转弯半径大
倒车	机器人停止, 原地旋转 180 度再前进, 效率不高但转弯半径几乎为 0
卡死状态报警	机器人处于不能动的状态, 并依次尝试各种动作, 如果始终不产生位移, 则发出卡死警报

图 22: 机器人行动模式

通过一般避障的要求与实验, 得到状态转移表:

序号	卡死	正前方 (30 度, 远)	前方 (60 度, 比较远)	前方周边 (120 度, 中等距离)	前方 (180 度, 近)	进行动作
0	0	0	0	0	0	直行
1	0	0	0	0	1	直行
2	0	0	0	1	0	找最远可达方向
3	0	0	0	1	1	直行
4	0	0	1	0	0	直行
5	0	0	1	0	1	找最远可达方向
6	0	0	1	1	0	找最远可达方向
7	0	0	1	1	1	直行
8	0	1	0	0	0	找最远可达方向
9	0	1	0	0	1	找最远可达方向
10	0	1	0	1	0	找最远可达方向
11	0	1	0	1	1	直行
12	0	1	1	0	0	找最远可达方向
13	0	1	1	0	1	直行
14	0	1	1	1	0	掉头
15	0	1	1	1	1	倒车
16-31	1	-	-	-	-	卡死警报

图 23: 机器人行动状态转移表

通过状态转移表, 构建转移条件簇:

簇序号	动作	条件
A	直行	0, 1, 3, 4, 7, 13
B	找最远可达方向	2, 5, 6, 8, 9, 10, 12
C	掉头	14
D	倒车	15
E	卡死	16 31

图 24: 机器人状态转移条件簇

根据状态转移条件簇, 画出有限自动机:

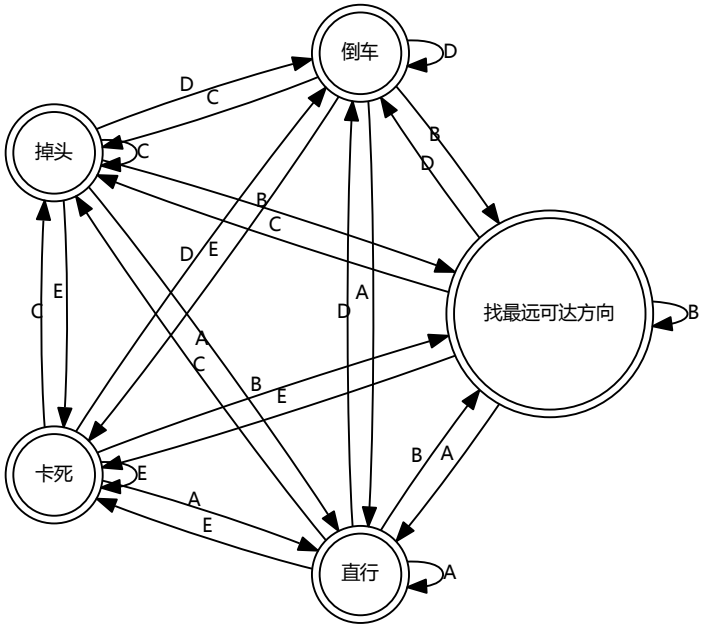


图 25: 机器人避障算法的有限状态自动机表示

在机器人探索地图的过程中, 需要检查是否有未探索区域. 由激光传感器的特性, 有简易的未探索区域标注算法. 如图26所示, 在激光传感器工作过程中, 检查每次扫描结果中相邻两障碍物点的距离. 若两障碍物点的距离大于给定值, 则可以确定在这两点之间有未探索区域, 将该区域用这两点的中点表示. 若在以后的探索中该点被观测到, 则从未探索点的列表中去除该点.

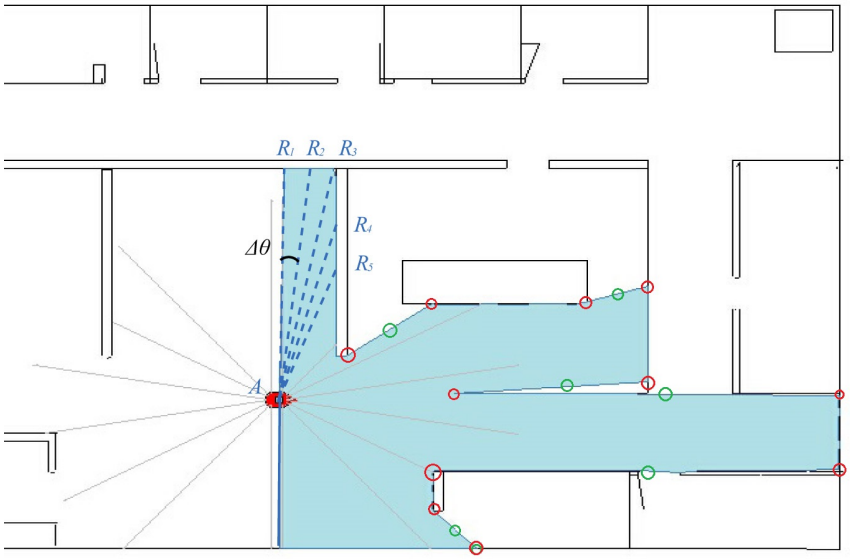


图 26: 未探索点标定算法示意图, 图中红圈部分表示激光传感器扫描结果中距离较大的相邻障碍物点, 绿圈表示这些障碍物点的中点. 如图, 绿圈部分基本上代表了机器人的未探索区域

利用自动避障技术, 机器人可以做到在无人控制的情况下进行地图的探索. 减轻了人工操作的压力.

6.4 地图构建

通过采集到的数据可以进行地图构建, 步骤如下:

1. 建立栅格地图
2. 扫描数据放缩变换
3. 栅格地图数据更新
4. 通过栅格地图提取障碍物
5. 建立几何地图
6. 将历次采集数据进行谱分割
7. 建立拓扑地图

机器人 SLAM 系统代码组织框架如图:

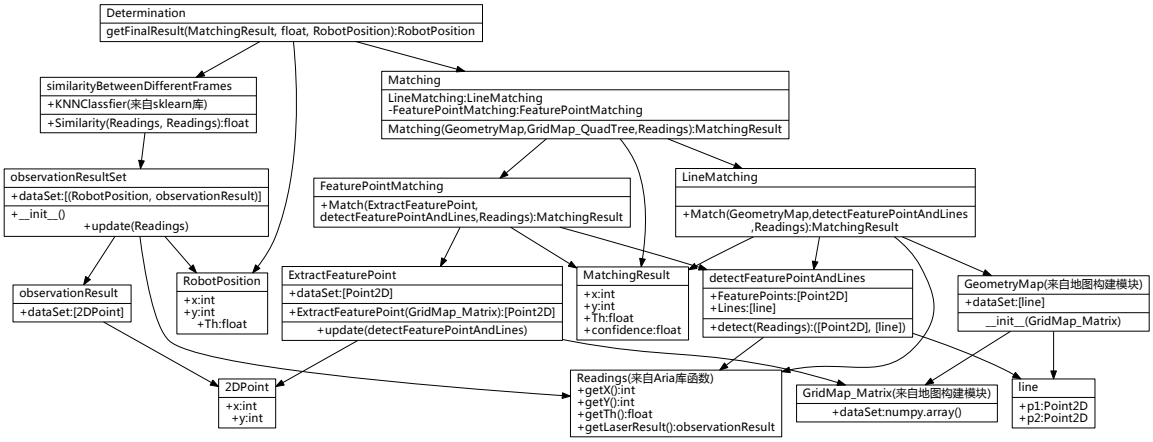


图 27: 机器人 SLAM 系统类图

主要模块意义与实现思路如下:

- Readings: 该类为来源于 Aria Mobile Robot 的 SDK 类库, 通过 Aria.py 中建立的激光传感器数据接口可调用该类的实例. 在机器人运动过程中, Readings 实时存储最近一次的激光传感器读取结果与读取时的机器人位姿 x, y, θ.
- GridMap_Matrix: 该类来自地图构建模块, 是栅格地图基于矩阵的存储实现.
- GridMap_QuadTree: 该类来自地图构建模块, 是栅格地图基于四叉树的存储实现.
  - ___init___(int,GridMap):GridMap_QuadTree: 四叉树地图类的构造函数, 利用已经构建的, 基于矩阵的栅格地图生成基于四叉树的栅格地图.
- GeometryMap: 该类来自地图构建模块, 存储了几何地图信息. 其构造函数利用基于矩阵的栅格地图构造几何地图.
- MatchingResult: 匹配结果类, 各个算法通过匹配扫描结果与全局地图, 反推改机器人的位姿, 并用 MatchingResult 类进行记录.
  - x,y,Th: 机器人位姿, 对应 x, y, θ
  - confidence: 根据各个算法本身的匹配结果计算此次算法的可信度 (与 similarityBetweenDifferentFrames 不同, similarityBetweenDifferentFrames 是根据连续两次观测结果间差异度计算可信度) 对于不同算法, 其实现方法不同. 取值均在 $(0, 1)$ 区间内.
- ExtractFeaturePoint 维护一个全局角点列表. 由于扫描精度原因, 同一个角点每次提取的结果间存在漂移现象. 所以需要将每次扫描结果中提取的角点与全局角点进行对比以确定归属.
- detectFeaturePointAndLines: 从每次扫描结果中实时提取线段特征与角点特征, 用一个 tuple 返回检测结果, tuple 中包含两个链表 FeaturePoints 与 Lines. 分别存储提取出来的角点与线段.
- observationResultSet: 存储过去一定时间内的机器人扫描结果与机器人位姿, 并将扫描结果与机器人位姿关联起来.

- similarityBetweenDifferentFrames: 利用 KNN 算法, 比较连续两次扫描结果的相似度. 在机器人运行过程中, 该算法实时计算当前一次扫描结果与前一次扫描结果的相似度, 若连续两次扫描结果不相似. 可能是扫描结果有差错, 也可能是机器人进入了一个未知领域. 根据该扫描结果决定机器人是否利用扫描结果矫正里程计累计误差.
- FeaturePointMatching: 特征点匹配算法.
- LineMatching: 基于线段的匹配算法.
- Matching: 综合 ICPMatching, LineMatching, FeaturePointMatching 三种匹配算法的结果. 现阶段单纯的将三种结果的各个分量取平均值. 一种更好地做法是将这三种结果分别计算协方差, 然后利用卡尔曼方法求取最佳估计值.
- Determination: 通过观测结果与连续两次结果间相似度决定是否采纳激光传感器矫正结果.
  - 如果连续两次扫描结果间相似度过低, 则返回里程计定位数据 RobotPosition(即激光扫描器扫描结果被丢弃了)
  - 如果连续两次扫描结果间相似度可接受, 匹配结果可信度低, 同样返回里程计定位数据 RobotPosition(即激光扫描器扫描结果被丢弃了)
  - 如果如果连续两次扫描结果间相似度可接受, 匹配结果可信度可接受, 则返回激光扫描器定位数据 (成功利用地图匹配矫正里程计误差)

6.5 地图分割

地图分割模块代码框架如图:

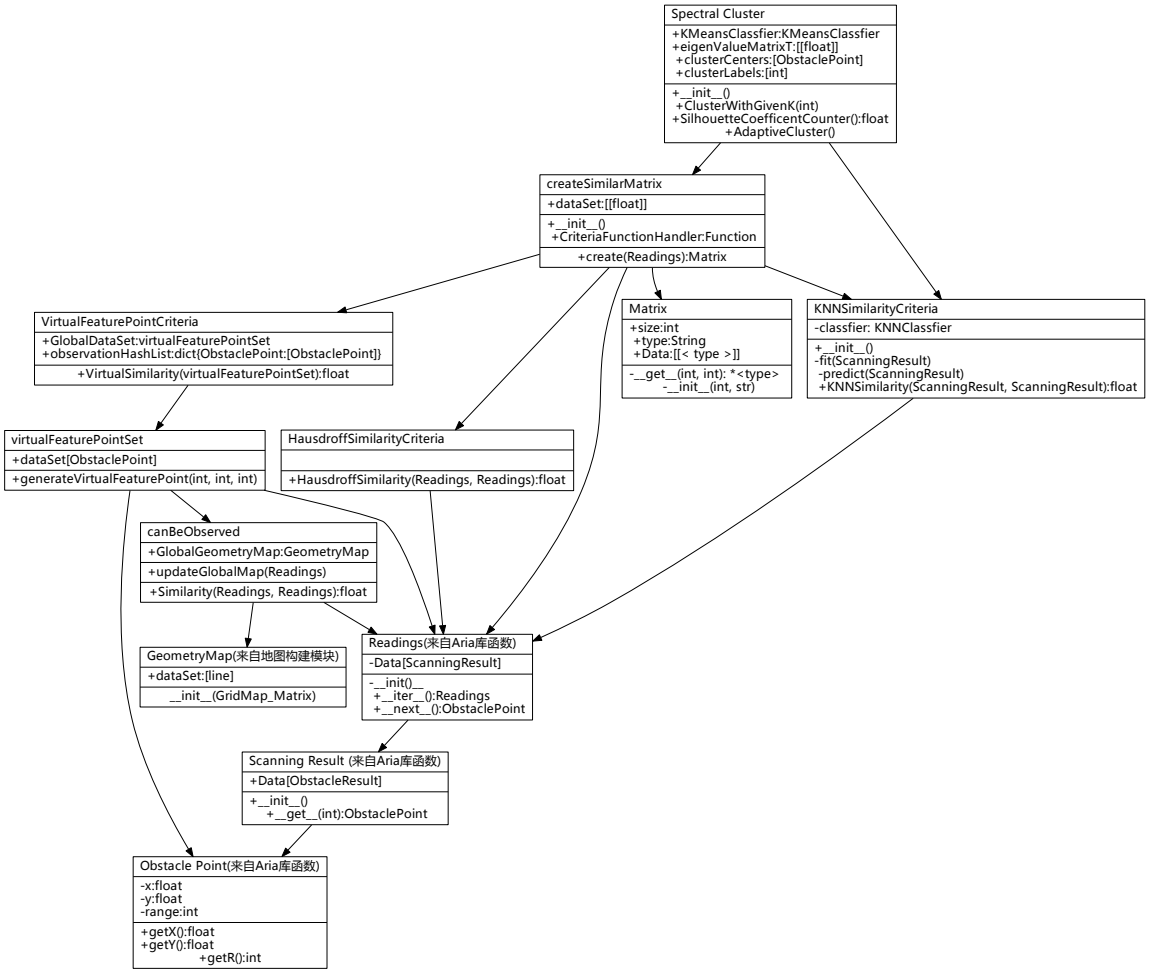


图 28: 在线地图分割算法类图

主要模块意义与实现思路如下:

- ObstaclePoint: 存储二维坐标点, 通常用于存储障碍物点, 也被复用做聚类中心, 线段端点等.
- Readings: 机器人运动状态在 Aria Mobile Robot 机器人控制平台中的存储形式, 包含当前机器人激光传感器扫描结果与机器人当时位姿信息
- GeometryMap: 地图构建环节产生的几何地图
- canBeObserved: 可视性检查函数, 在虚拟特征点相似度构造准则章节进行了详细介绍
- virtualFeaturePointSet: 虚拟特征点集合, 通过虚拟特征点的观测结果进行地图分割
- VirtualFeaturePointCriteria: 虚拟特征点相似度构造准则, 核心函数 +VirtualSimilarity(Readings,Readings):float 度量两次观测结果 (两个虚拟特征点) 间相似度
- HausdroffSimilarityCriteria: 用于度量两次观测结果之间的相似度, Hausdroff 距离除给定的观测结果之外不借助任何外部信息, 具有很好的独立性

- KNNSimilarityCriteria:KNN 相似度度量准则, 核心利用 scikit-learn 提供的 NearestNeighbor 算法实现, 用于计算两给定点集的相似度
- Matrix: 基本矩阵类
- CreateSimilarMatrix: 用于创建相似度矩阵, 类内部包含一个函数接口以决定调用何种相似度函数构造准则.
- SpectralCluster: 自适应谱聚类算法, 内部包含拉普拉斯矩阵计算, 提取特征向量, KMeans 聚类等算法. 同时, 利用轮廓系数, 挑选聚类结果中表现最好的一个进行输出. 该类提供 clusterCenters:[ObstaclePoint] 与 clusterLabels:[int] 两个成员变量供外部调用. clusterCenters 提供每个聚类中心坐标 (在 K-Means 计算过程中迭代得出) 与每个观测点所归属的聚类序号. 根据序号到 clusterCenters 中可以找到聚类中心点.

6.6 地图的表示

6.6.1 栅格地图的二维表示

图像的生成方式 栅格地图的二维表示使用 openCV 实现. 栅格地图的存储如式9所示, 而 openCV 中图像的存储形式如下:

$$Graphic = \begin{pmatrix} p_{11} & p_{12} & \cdots & p_{1x} \\ p_{21} & p_{22} & \cdots & p_{2x} \\ \vdots & \vdots & \ddots & \vdots \\ p_{y1} & p_{y2} & \cdots & p_{yx} \end{pmatrix} \tag{40}$$

其中 $p = (green, blue, red)$ 为颜色的三元组, 其排列顺序与常见的 RGB 排列不同. 其中每一个分量取值范围为 $[0, 255]$. 则由下式将栅格矩阵 M 扩展为图像 $Graphic$:

$$p_{ij} = \begin{cases} [255, 255, 255] & M_{ij} = 1 \\ [0, 255, 0] & M_{ij} = 0 \end{cases} \tag{41}$$

这样就将栅格矩阵 M 变换为适合 openCV 显示的矩阵, 其背景色为白色, 障碍物用蓝色显示.

基于四叉树的栅格地图类图如图29:

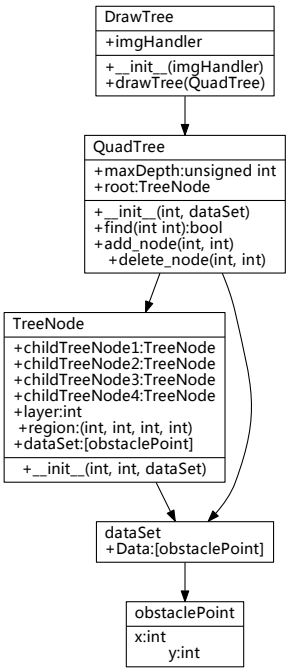


图 29: 栅格地图四叉树存储实现的类图

各个模块意义即实现思路如下:

- dataSet: 初始用于生成四叉树的数据集合. 在机器人避障探索过程中, 通常在探索一定范围后生成四叉树地图.
- TreeNode: 四叉树的主要数据结构, 每个 TreeNode 表示四叉树中一个节点. 四叉树中节点分为根节点与叶子节点两种. 其中根节点存储有指向子节点的指针 childTreeNode1 至 childTreeNode4. 且根节点 dataSet 为空. 叶节点不存在指向子节点的指针, dataSet 为该节点对应区域内的障碍物点. 这种存储方式可以保证每个障碍物点被存储且仅存储一次, 避免了空间浪费. 每个四叉树节点存储有深度与该节点对应区域坐标 (四个 int 型证书, 代表对应区域左上与右下坐标)
- QuadTree: 四叉树的外部接口类, 封装了一系列常用操作, 包括增删改查等操作的递归实现. 通过重载索引操作符, 使得四叉树的外部表现与基于矩阵的实现相同. 四叉树类包括一下成员与方法:
  - maxDepth: 四叉树的最大深度, 通常深度为 10 到 12 的四叉树就可以保证相当的精度.
  - root: 四叉树的根节点, 四叉树上的各种操作通过根节点作为入口递归实现.
  - ___init___(int, dataSet), 接受一个 int 型最大深度, 与一个障碍物点集 dataSet. 以 dataSet 中障碍物点分布的最大范围加 200% 为区域范围, 进行四叉树的构造.
  - find(int, int):bool: 该函数接受一个坐标, 返回一个布尔型变量. 若该坐标所在节点对应区域内存在障碍物点, 则返回 True, 表明该坐标附近一个精度范围内存在障碍物. 若不存在障碍物, 返回 False.

  - add_node(int, int), delete_node(int, int): 四叉树中动态增加, 删除节点. 由于在机器人行进过程中, 传感器不断地收集信息, 所以需要动态的进行四叉树节点的增加 (删除节点很少用得到). 需要注意的是增加节点只能在四叉树原有范围内增加. 若观察到的新障碍物点超出四叉树所能表示的最大范围, 则需要以更大的范围重新构建四叉树.

- DrawTree: 调用 Python 的 matplotlib 库进行绘图工作, 方便在程序中进行可视化 debug. 也可以在程序运行中动态观察四叉树的工作情况. 不建议对四叉树的全部节点进行绘制, 一方面因为底层区域范围太小看不清, 另一方面绘制四叉树需要大量的计算. 在本实现中仅绘制四叉树的前六层. 通过 imgHandler 返回生成图片的指针, 通过 drawTree 进行重绘. 在 GUI 中, 可以在键盘, 鼠标事件的回调函数中调用 drawTree 函数, 并将 imgHandler 指向的图片进行重绘.

利用 openCV 显示图像 在 openCV 中显示该地图的步骤如下:

1. 载入 openCV 图像处理库
2. 初始化窗口
3. 初始化给定大小的图像 M, 初始化画面中各个控件
4. 由式 40 , 41将矩阵 M 中的相应像素设置为对应颜色
5. 设置鼠标, 按键的回调函数以便于在程序中进行交互
6. 设置程序主循环, 事件触发函数, 并在窗口中更新图像

程序代码如下:

```
import cv2
import numpy as np

def show(obstaclePointList):
    cv2.namedWindow("ImageShow")
    ## 初始化窗口
    def mouseCallBack(event, x, y, flags, param):
        ## 鼠标事件回调函数设置
    cv2.setMouseCallback("ImageShow", mouseCallBack)
    ## 注册鼠标事件回调函数
    img = np.zeros((1024, 1024, 3), np.uint8)
    ## 初始化空图像矩阵
    cv2.rectangle(img, (0, 0), (1024, 1024), (255, 255, 255), -1)
    ## \color{mygreen}画一个与窗口等大的矩形作为背景
    for point in obstaclePointList:
        img[point[0]][img[point[1]]] = [0, 255, 0]
        ## 生成图像
    cv2.imshow("ImageShow", self.globalPic)
    ## 显示图像

if(__name__ == "__main__"):
    show()
```

二维显示模块代码结构如图:

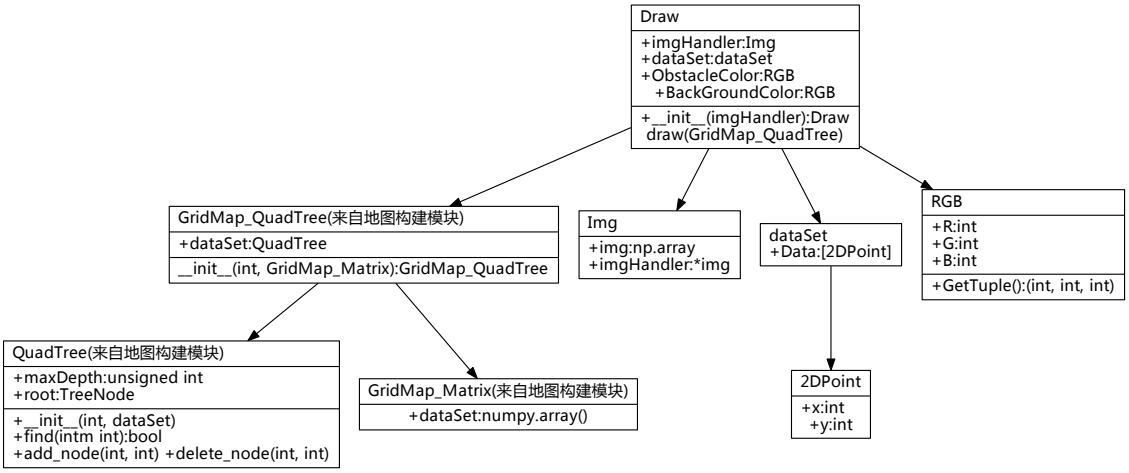


图 30: 二维显示模块类图

主要模块意义与实现思路如下:

- QuadTree(来自地图构建模块): 四叉树的实现. 该类中封装了四叉树节点, 最大深度与四叉树的一系列操作 (增删改查等)
- GridMap_QuadTree(来自地图构建模块): 基于四叉树的栅格地图实现, 在四叉树的基础上, 封装了与地图相关的操作, 包括地图生成, 地图更新等操作.
- GridMap_Matrix(来自地图构建模块): 基于矩阵的栅格地图实现
- Img: openCV 中图片的存储类, 内部通过一个 numpy.array 数组进行存储
- RGB: 颜色类, 存储一个颜色的 RGB 分量, 同时提供正常程序的 RGB 顺序输出与 openCV 的 GBR 顺序输出.
- Draw: 会图行为类, 提取栅格地图中障碍物, 并将其绘制在给定图片上.

6.6.2 几何地图的二维表示

几何地图与二维地图表示方式相似, 如式12所示, 每个障碍物包括两个顶点与一条边. openCV 中画线函数为:

```
1 cv2.line(img, p1, p2, RGB, width)
```

各参数意义如下:

- img: 图片指针, 指向给定图片
- p1,p2: 线段的两个端点, 传入 cvPoint 型变量或一个 Tuple:(float, float)
- RGB: 线段颜色, 传入一个三元组 (green, blue, red). 注意在 openCV 中颜色的顺序与常见顺序不同.
- width: 线宽, 通常设为 1

6.6.3 几何地图的三维表示

在三维空间中表现地图可以使客户对机器人工作空间产生更直观的理解. 本文使用 openGL 绘图库进行三维空间图形绘制. 与 openCV 相比, openGL 将大量运算放在 GPU 中使用硬件完成, 充分发挥了现代计算机的显卡优势. 同时也减少了 CPU 的负载, 可以将 CPU 的运算能力投入到 SLAM, 地图分割, 网络通信等更重要的领域. 与此同时, openGL 也提供了高效的绘图支持函数, 包括光照, 深度缓存, 图元装配等操作.

在本平台中, 三维几何地图表示实现了如下功能:

- 网格地面显示, 提供一个标尺以度量环境大小
- 机器人工作空间障碍物显示
- 可以自由变换的视角, 利用鼠标与键盘操作使视角变换, 以从各个方向观察地图

利用 openGL 绘图的关键步骤如下:

- 初始化 openGL 绘图参数
- 创建顶点序列
- 通过顶点序列创建顶点索引数组
- 将顶点序列与顶点索引数组发送至 GPU, 并完成绑定
- 设置绘图方式, 在缓冲区域完成图像绘制
- 设置观察视角, 创建可视变换
- 创建窗口, 在窗口中显示图像
- 绑定鼠标事件, 键盘事件的回调函数
- 创建程序主循环, 运行程序
- 释放无用内存区域

初始化绘图参数包含以下操作:

```
1    glutInit() ## 初始化openGL 绘图环境
2    glutInitDisplayMode(GLUT_RGBA | GLUT_SINGLE) 将##绘图模式设置
         为openGL RGBA 四通道
3    glutInitWindowSize(800, 800)
4    glutCreateWindow("Hello")
5    glutDisplayFunc(draw_3d)
6    glClearColor(0.0, 0.0, 0.0, 0.0)
7    gluOrtho2D(-1, 1, -1, 1)
8    glutMainLoop()
```

创建顶点序列的步骤可以通过几何地图中数据来完成. 二位几何地图中的线段 $(p_1, p_2) = (p_{1x}, p_{1y}, p_{2x}, p_{2y})$ 在三维地图中表示为一堵高度为 h 的墙

$$\begin{cases} p_1 = (p_{1x}, p_{1y}, 0) \\ p_2 = (p_{2x}, p_{2y}, 0) \\ p_3 = (p_{2x}, p_{2y}, h) \\ p_4 = (p_{1x}, p_{1y}, h) \\ VertexIndex =< p_1, p_2, p_3, p_4 > \end{cases} \tag{42}$$

上式中 $VertexIndex$ 为该障碍物的顶点索引序列, 在 openGL 的 GL_QUADS 绘图模式下会绘制顶点 p_1, p_2, p_3, p_4, 与 p_1, p_4, p_2, p_3 构成的四边形. 由于 openGL 绘制的平面其法向量遵循右手定则, 所以顶点序列顺序不能更改.

顶点序列与顶点索引序列被转换成 numpy 的定长 Array 并发送到 GPU 中完成绑定, 命令如下:

```
        vbo = vbo.VBO(np.array(vdata, f))
        ebo = vbo.VBO(np.array(vindex, H), target = GL_ELEMENT_ARRAY_BUFFER)
        eboLength = len(vindex)
        vbo.bind()
        glInterleavedArrays(GL_N3F_V3F, 0, None)
        ebo.bind()
```

绘制物体命令如下:

```
glDrawElements(GL_QUADS, eboLength, GL_UNSIGNED_SHORT, None) ## 绘制图形
## 绘制模式 GL_QUADS，在该模式下 ebo 中每四个坐标作为两个三角形，省略共享边进行绘制（绘制为一个四边形，法线方向根据右手定则）
glFlush() ## 将绘制结果发送至显示缓冲区
```

同样地, 创建网格地面也需要生成顶点序列与顶点索引序列, 对于栅格数量为 $row \times column$, 栅格大小 $gridsize$ 的网格地面, 符合以下公式:

$$\begin{cases} 0 \leqslant i \leq (row-1) \\ 0 \leqslant j \leq (column-1) \\ p_1 = (gridsize \times (i+1), gridsize \times j, 0) \\ p_2 = (gridsize \times (i+1), gridsize \times (j+1), 0) \\ p_3 = (gridsize \times i, gridsize \times (j+1), 0) \\ VertexIndex =< p1, p2, p3 > \end{cases} \tag{43}$$

对于矩形网格地面, 可以只绘制每个网格的两条边 (上式中选择上边与右边) 进行绘制, 减少图形重绘次数. 可以在绘制结束后补充场地外边缘矩形, 使得网格最外圈封闭.

三维显示模块主要类与行为如下:

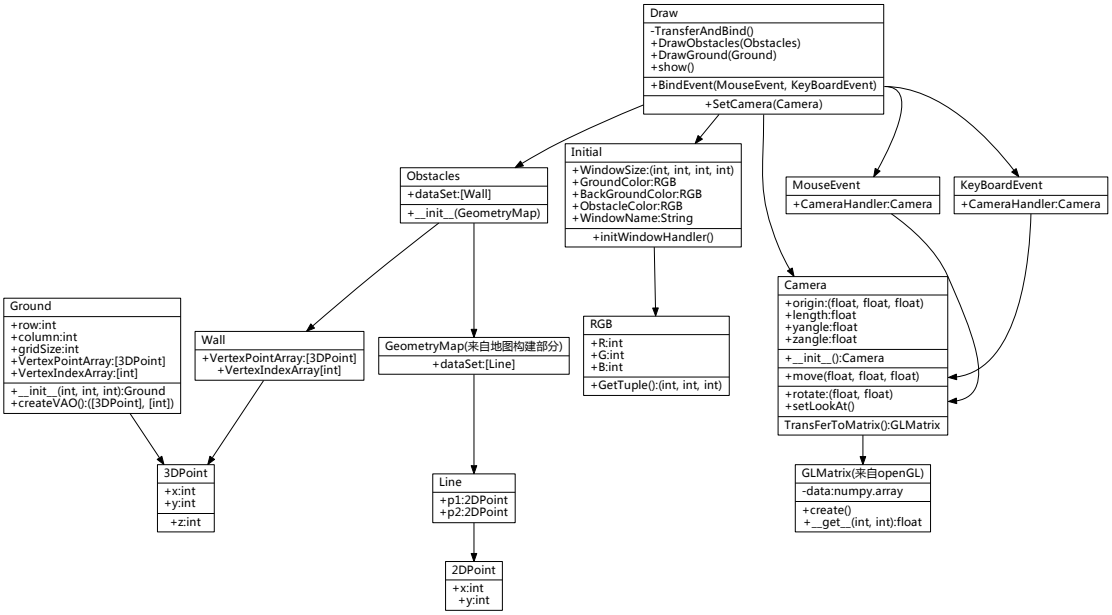


图 31: 三维显示模块类图

主要模块意义与实现思路如下:

- Ground: 栅格地面类, 用于创建栅格地面, 设置参数并生成顶点序列与顶点索引序列.
- Wall: 单个障碍物, 包括顶点序列与顶点索引序列. 在单个障碍物中顶点序列与顶点索引序列都是定长的.
- Obstacles: 全体障碍物集合, 其构造函数 +___init___(GeometryMap):Obstacles 接受几何地图作为参数, 产生障碍物序列.
- Initial: 初始化类, 负责 openGL 绘图模式设置, 窗口注册, 颜色设置等操作.
- Camera: 管理 openGL 中视点, 负责设置视点初始位置, 并提供函数修改视点参数. 并产生可视变换矩阵, 使三维空间内障碍物能够正确投影至窗口中.
- MouseEvent, KeyBoardEvent: 键盘, 鼠标事件回调函数. 通过调用 Camera 中 rotate, move 函数旋转, 平移视角.
- Draw: 绘图函数, Draw 函数遍历 Obstacles 中各个障碍物 Wall, 将其顶点序列与顶点索引序列传并绑定至 GPU 中. 同时, Draw 函数中注册了鼠标事件与键盘事件的回调函数 MouseEvent, KeyBoardEvent, 使得用户可以通过键盘鼠标直观地操作模型.

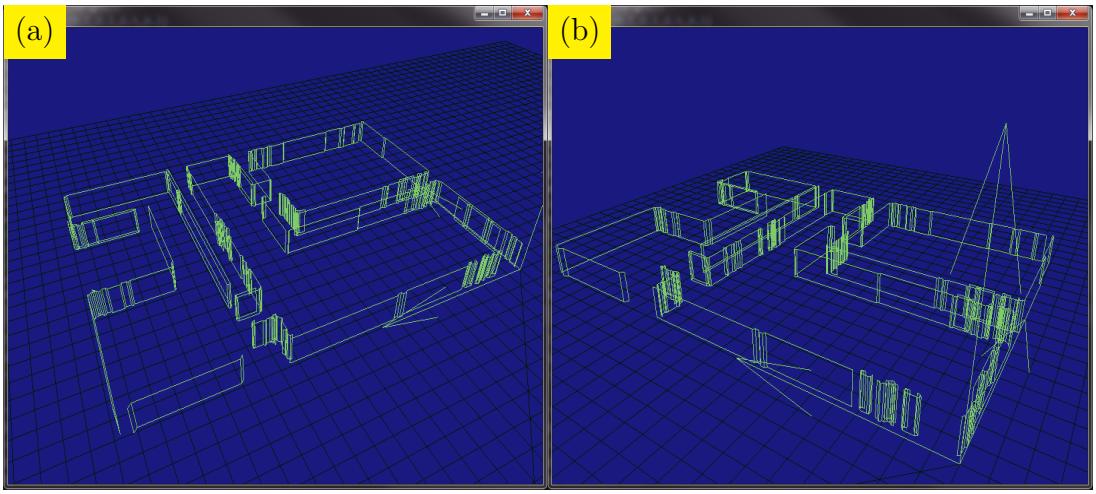


图 32: 3D 表示的几何地图示意图, 两张图为同一地图在不同视角下的显示

6.6.4 拓扑地图的存储与表示

拓扑地图主要模块结构关系图如下:

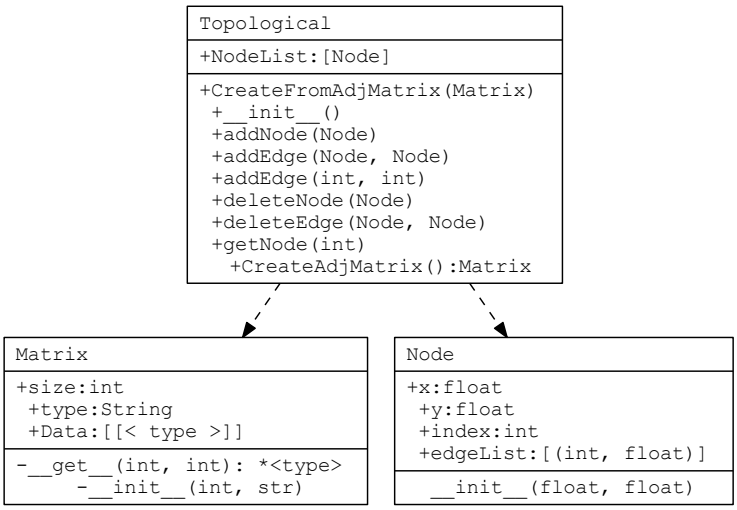


图 33: 拓扑图存储结构的类图

主要模块意义与实现思路如下:

- Node: 拓扑图基本组成元素, 每个 Node 代表拓扑图中一个节点. Node 中存储以下信息:
  - x, y: 每个节点在机器人工作空间中对应坐标. 事实上, 拓扑图中的一个节点对应的是机器人工作空间中的一个区域. 因为该区域是由聚类获取的, 可以用聚类中心代替该区域.
  - index: 每个节点的索引. 为了方便遍历, 用一个 Node 型单链表存储拓扑图中所有节点. index 即链表中每个节点的序号.

  - edgeList:[(int,float)]: 每个节点存储其相关联的其他节点. 对每个节点所连接的其他节点, 用 (节点序号, 距版节点距离) 来存储.

- Matrix: 在拓扑图中, 矩阵类可用来表示关联矩阵, 度矩阵, 拉普拉斯矩阵等信息. 矩阵类包含大小, 数据类型等信息.

- Topological: 拓扑图对外接口, 封装了拓扑图与图相关的操作. Topological 类使用一个 Node 型链表存储拓扑图中所有节点. 并实现了包括索引节点, 添加节点, 添加边, 删除节点, 删除边在内的操作另外, Topological 类还实现了导出关联矩阵的操作.

6.7 小结

本章介绍了机器人同步定位与地图构建与地图分割平台的构建与实现思路. 介绍了本平台的搭建与使用方法, 工程中各个文件, 类, 模块的意义, 实现思路与调用关系. 对于比较复杂的模块, 给出了模块的类图.

在机器人避障探索一节, 给出了该问题的一个基于有限自动机的实现. 该实现方案实现起来更为简易, 代码清晰容易修改.

在机器人工作空间表示一节, 给出了栅格地图, 几何地图的显示方案. 特别地, 本文利用 openGL 进行三维地图显示, 使地图构建结果更为直观.

7 实验结果与分析

7.1 机器人工作空间环境构建

7.1.1 机器人工作空间的四叉树存储测试

本实验测试机器人基于四叉树的地图生成效果, 并评价基于四叉树的地图与基于矩阵的地图性能上的比较关系.

在机器人工作空间表示环节, 本文使用四叉树作为机器人工作空间的基本表示方式. 四叉树的构建结果如图. 在本实验中, 采用的四叉树最大深度为 10 层. 在大幅度减少存储空间的基础上, 提供了不亚于矩阵存储的精确度.

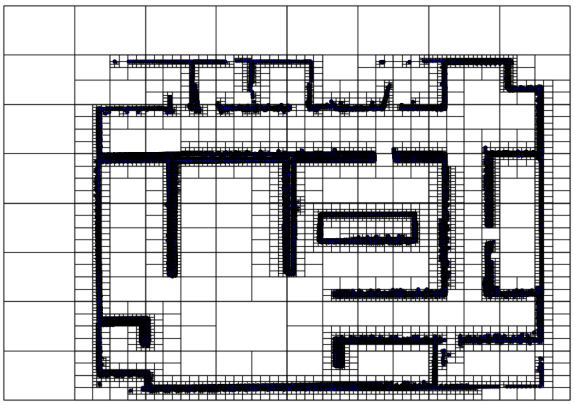


图 34: 全局地图的四叉树构建结果, 显示到第六层

在机器人探索环节, 经常出现噪声点. 噪声的产生可能来源于传感器测量误差, 也可能是传感器探索到了开放空间. 例如一扇没关紧的, 通往外界的门, 或落地窗. 这些区域不会反射激光, 在机器人传感器中, 就等效为一个非常远 (取决于传感器内部电路实现方式与所能探测到的最大距离) 的障碍物点, 如图:

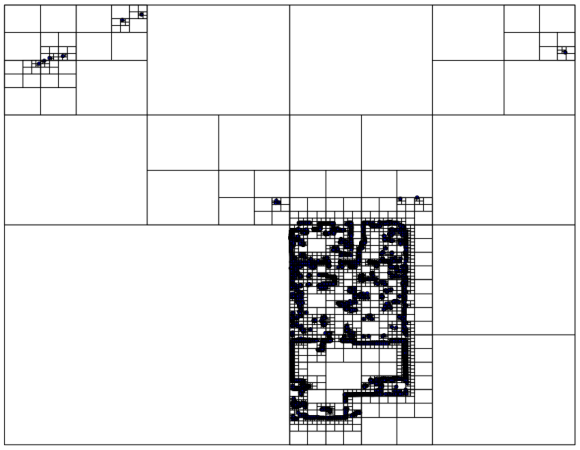


图 35: 四叉树算法对地图中的噪声点也能够很好地处理

在基于矩阵存储的栅格地图中, 这样的噪声点只能加以丢弃, 或是增加矩阵大小以容纳该点. 在四叉树中, 通过增加叶子节点的方式可以不花费很多额外代价的处理这些噪声点.

7.1.2 SLAM 算法测试

本实验测试机器人同步定位与地图导航模块工作状况, 并对地图构建效果进行评价. 本实验在 mobileSim 机器人仿真平台与实际环境下进行测试. 同步定位与地图构建模块用户界面如图36所示. 界面中前后左右四个按钮可以控制机器人前进, 左右转, 掉头后退 (走 U 形, 有转弯半径). back 按钮控制机器人倒退 (没有转弯半径, 在倒退过程中激光传感器无法扫描后方情况所以不建议使用). REC 键记录当前机器人扫描结果并导出至硬盘中. 界面中红色区域反映了地图中的障碍物, 红圈代表当前扫描结果中的角点.

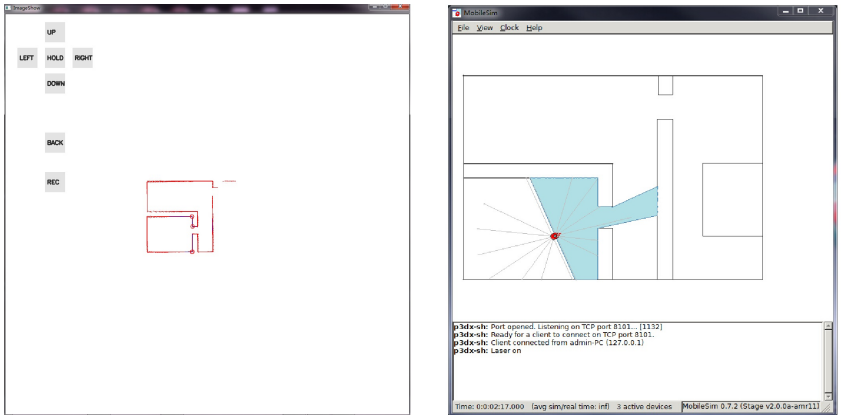


图 36: 左图为机器人 SLAM 平台的用户界面, 右图为 mobileSim 仿真平台界面

在机器人地图构建中, 利用上文所述 SLAM 算法, 有效的抑制了由累计误差带来的结果漂移现象, 如图7.1.2所示, 图中蓝色点代表测量到有障碍物的栅格. 左图显示了没有抑制累计误差的情况, 右图显示了包含里程计误差校正环节的 SLAM 过程. 同时在图41中, 显示了包含回环的 SLAM 过程. 在该实验中, 机器人成功匹配了全局地图中已出现的位置, 没有带来很大的累计误差.

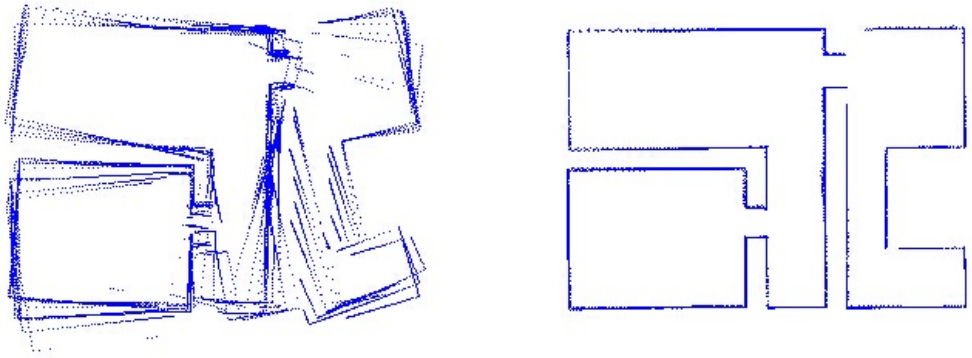


图 37: 左图显示了单纯依靠里程计进行 SLAM 带来的累计误差, 在该模式下, 不利用激光传感器进行里程计校正. 右图显示了利用激光传感器进行里程计误差矫正的结果, 与左图结果相比减少了累计误差

在地图构建过程中, 使用激光传感器进行里程计结果矫正. 并使用 KNN 算法 (在4.2.1中详细介绍) 进行可信度判别, 如图38所示. 在图中, 机器人分别在室内与室外探索采样, 帧间差异度峰值点对应机器人穿越门口瞬间. 可以看到, 在室内与室外机器人帧间相似度在同一水平上波动, 而穿越障碍物时出现了较大的波动. 此时利用激光传感器矫正采样结果的方法是不可用的.

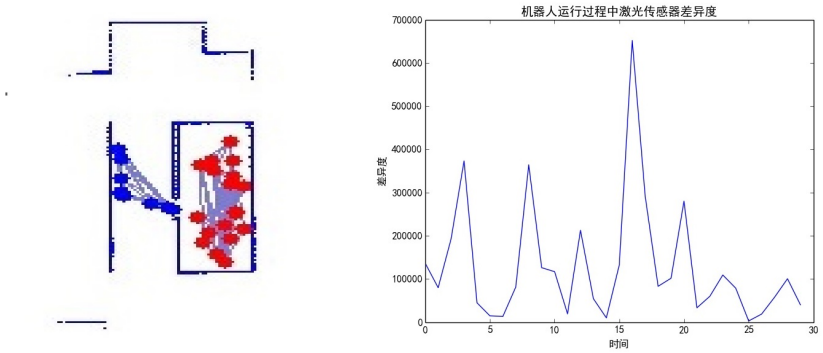


图 38: 图中显示了机器人在室内与走廊探索时相邻扫描结果间相似度的变化, 相似度使用 KNN 相似度评价准则

图7.1.2为机器人自主构建的, 三维显示的几何地图, 在三维显示模式下, 键盘 W, A, S, D 键可令视角在 xy 平面上移动, 键盘 R, F 键令视角在 Z 轴方向上下移动. 鼠标拖动可使视角转向相应方向. 图中障碍物, 地面, 坐标轴均正常显示.

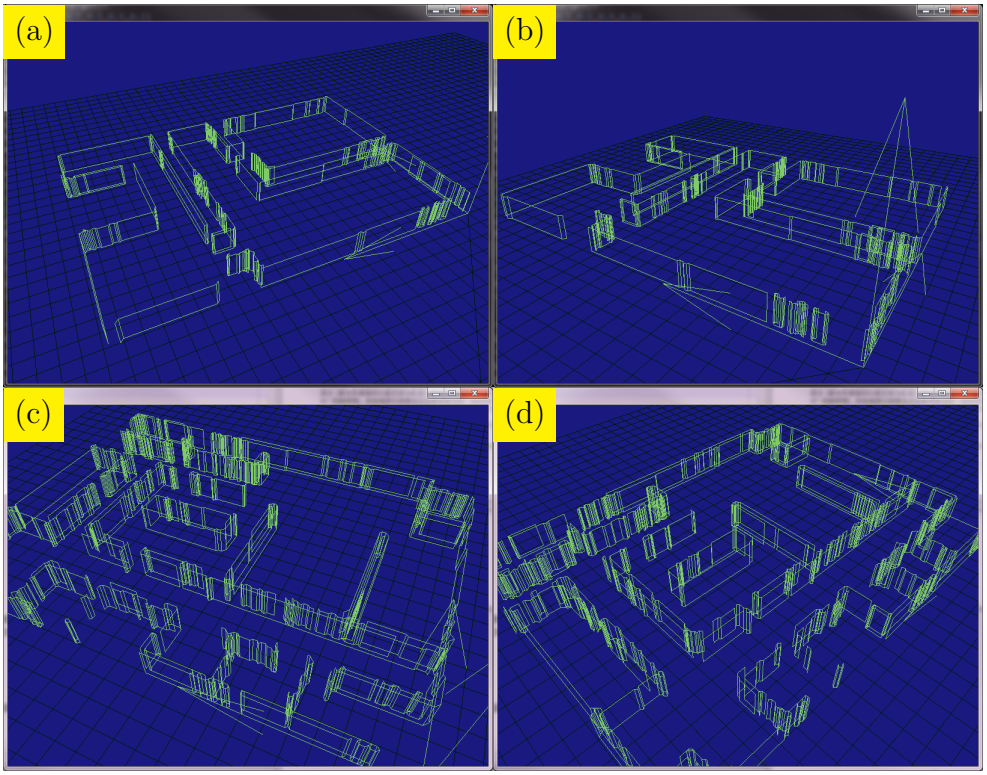


图 39: 基于几何地图的三维地图显示效果图

7.2 自主地图分割算法测试

本实验利用 mobileSim 仿真平台与实际环境进行机器人地图分割算法的测试, 具体测试内容为:

- 在简单地图中测试自主分割算法是否能够将给定地图分割为期望的结果.
- 在该算法对应的特殊情况下 (地图带有回环与长条形地图) 验证该算法是否正常工作.

- 在复杂地图中测试自主分割算法是否正常工作, 效果如何.
- 在不同类型地图 (几何地图, 栅格地图) 中, 对不同相似度构造准则 (角点, Hausdroff 距离, KNN 算法, 虚拟特征点) 进行试验并进行比较
- 在实际环境中, 利用移动机器人平台进行地图分割算法的测试.

使用简单地图, 对机器人 SLAM 平台与地图分割算法进行测试, 在不同聚类簇数下测试结果如图 (根据自适应聚类方法聚类簇数为 2):

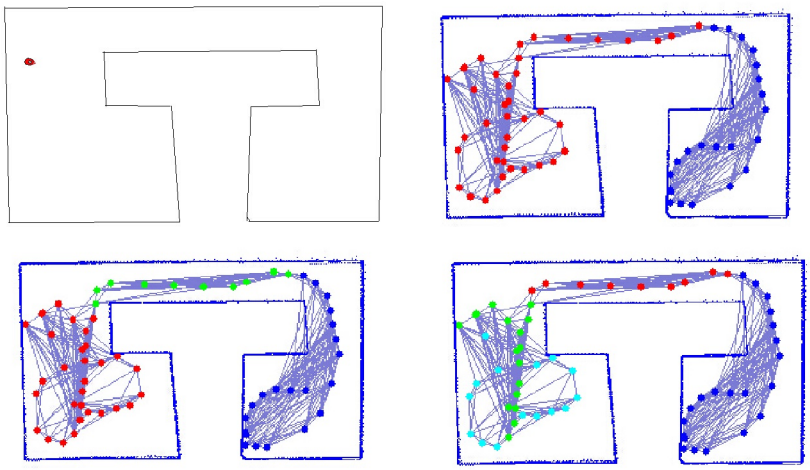
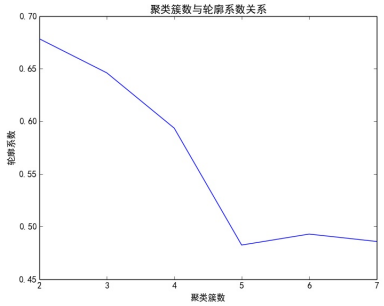


图 40: 地图分割算法对简单环境的测试结果. 图中圆点代表机器人在该点进行了观测, 节点之间的连线表示这些节点代表的观测结果相似度高于给定阈值. 图中分别展示了聚类簇数为 2, 3, 4 时的聚类情况.

图中每个节点表示机器人在该位置的一次观测 (此时机器人的朝向在地图中没有表现), 节点的颜色表示此次观测所对应的子图 (即聚类的结果), 节点间的边表示两节点之间的相似度大于一定阈值. 如图可见, 聚类簇数为 2, 3 时地图分割效果很好, 在簇数为 2 时稍显欠分割, 簇数为 4 时出现了过分割现象. 这是因为在获取数据的过程中数据量过小, 导致聚类结果类内凝聚度不足. 在实际应用中激光传感器读取数据频率可以达到每分钟上百次, 可以在一定程度上优化聚类结果.

聚类簇数与轮廓系数关系如下图:



K	轮廓系数
2	0.67825
3	0.64591
4	0.59339
5	0.48234
6	0.49276
7	0.48578

表 1: 聚类簇数与轮廓系数关系

通常情况下, 聚类簇数增长到一定阶段后, 轮廓系数会突然下降而后保持稳定. 这是因为随着聚类簇数的增长, 簇内凝聚性保持不变. 因为样本总体半径有限, 簇间分离度在

下降到一定阶段后就保持稳定. 一个可行的算法是利用手肘准则, 找出轮廓系数突然下降的阶段并以该阶段的一半作为聚类簇数. 在本文所描述算法中仅选择轮廓系数最大值所对应聚类簇数:

在图41所示的测试环境中, 机器人从左下角开始, 沿逆时针方向绕房间旋转一圈. 在此过程中生成地图并进行地图分割. 在地图的生成过程中, 累计误差被有效的抑制, 保持了相当的精度.

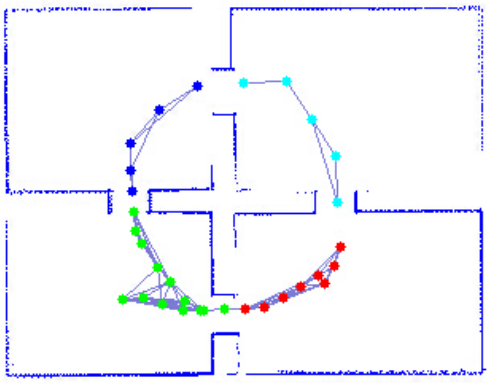


图 41: 机器人在环形房间中检测, 以观测其误差发散现象

几何地图构建与测试结果如图42, 43所示. 可以看出, 抽象的几何地图基本保持了原地图的细节 (在机器人扫描的过程中一些区域没有扫描到, 导致了一些细节的丢失). 但有一些连续的障碍物没有进行合并. 在后续的工作中, 应当对离散的障碍物进行合并:

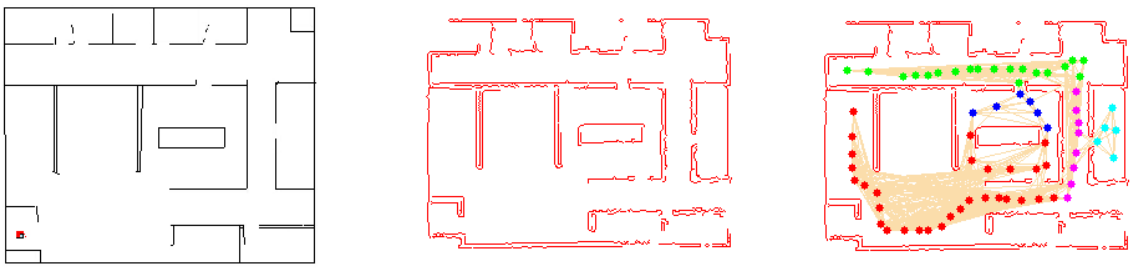


图 42: 利用几何地图进行地图分割测试, 左图为机器人仿真环境, 中图为几何地图构建结果, 右图为地图分割结果

利用几何地图进行自适应聚类的测试, 在极端环境下 (细长管道), 自适应聚类也能够将正确分割全局地图.

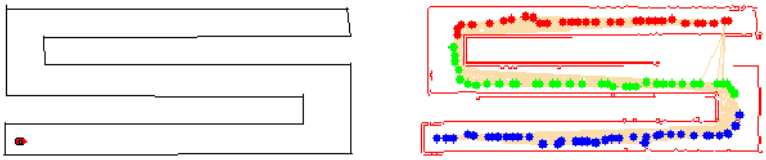


图 43: 自适应聚类对长条形环境测试

此外, 地图分割效果好坏也与地图中信息采集充分程度有关. 如图:

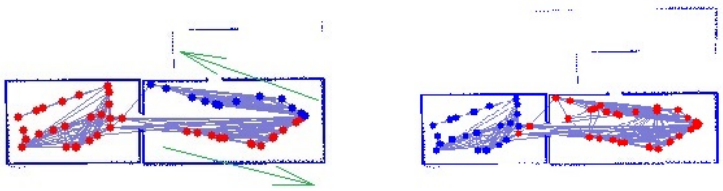


图 44: 不完全的信息采集导致了误分割

左图中, 机器人以逆时针方向沿左右两房间的边缘行进一周, 绿色箭头指示机器人在右侧房间的行进轨迹. 左图表明在地图分割过程中, 出现了比较明显的误分割现象. 出现该状况的原因在于对地图分割而言, 单纯的使扫描结果覆盖每个房间不足以产生合适的分割结果. 扫描结果间的覆盖率 (即各次扫描结果间的重合率) 对于地图分割的正确性更为重要. 在左图的右边房间扫描路径中, 上下两部分上下两侧观测结果几乎没有重合部分 (激光传感器的扫描范围为机器人朝向左右各 90 度范围, 即无法扫描到机器人" 身后" 的障碍物). 因而地图分割算法无法将其视为同一房间内的扫描点.

右图在左图观测结果的基础上补充了 3 个逆时针方向的观测点. 补充了观测点间的关联信息. 从而纠正了算法的误分割错误.

为避免图44所示的情况, 可以利用虚拟特征点替代真实扫描结果进行地图分割. 分割算法效果与右图相同. 虚拟特征点根据特征点间可视性进行相似度的构造, 相当于给机器人装上了全向的激光扫描仪. 事实上, 该方法与装有全向激光扫描仪的及其地图分割结果相同.

对于复杂地图的测试结果如图46所示, 该地图由激光传感器实际探测结果生成, 是对室内机器人工作空间一个比较直观的描述. 测试结果如图, 地图分割算法成功识别了地图中各个区域.

不同相似度计算策略对比 在4.2.1节中, 本文介绍了多种相似度计算策略. 在实际应用中, 我们需要将不同的相似度判别结果进行归一化, 以均衡其影响. 进一步的, 通过分析不同相似度策略间差异, 可以得到不同策略的使用场合, 为不同策略间结果融合提供一句. 图45给出了在不同相似度计算策略下, 根据同一次机器人 SLAM 结果生成的相似度矩阵. 图中观测结果按照机器人行进中时间顺序排列, 由于机器人连续的观测结果间彼此相似度较高, 使得图中主对角线附近区域相似度相对较高. 图中次对角线两端相似度较高, 因为机器人绕地图一周回到了第一个房间, 并且机器人成功从全局地图中识别出了该区域. 图中主对角线附近矩阵近似分块排列, 就对应各个房间内部的观测结果 (同一个房间内观测结果相似度较高). 从图中可看出, 基于 Hausdroff 的相似度计算策略数据双峰性质较好, " 相似" 与" 不相似" 差异较大. 基于特征点的相似度评价准则取值相对离散, 数据分布较大. 这是由于空间中特征点分布不尽均匀导致的. 在采用该策略时不仅要注意全局相似度的分布, 还要注意数据分布的局部性. 最近邻算法数据分布相对较好, 然而该算法倾向于赋予较高的相似度. 在数据融合时, 要适当降低最近邻算法的权重, 增加基于特征点算法的权重以获取一个相对" 公平" 的结果.

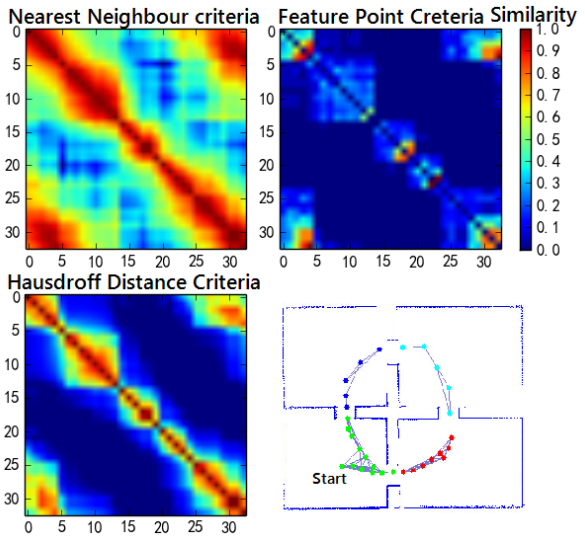


图 45: 不同相似度计算策略下相似度矩阵结果对比图

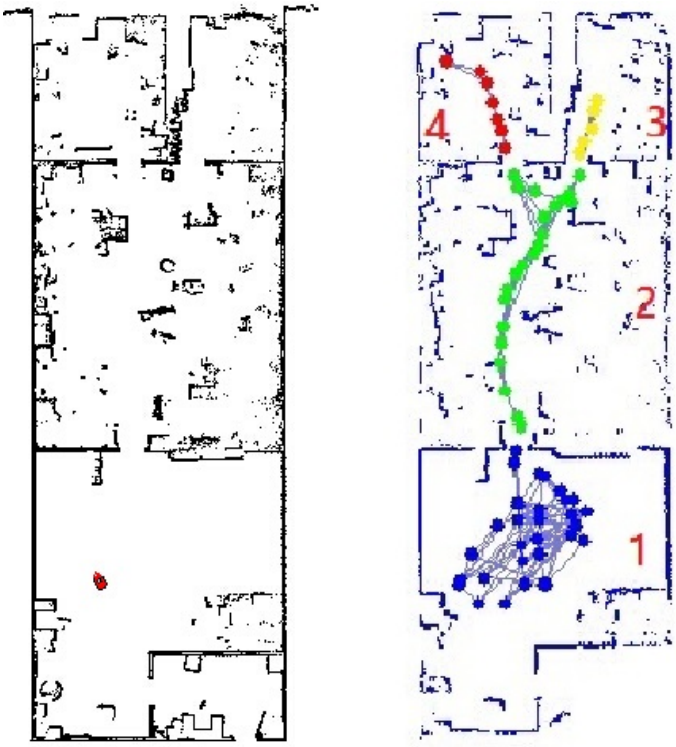


图 46: 左图为测试地图, 右图显示了对该地图的 SLAM 结果与在线地图分割结果. 可以看到, 本文提出的算法成功将区分了地图中的各个房间.

7.3 机器人导航算法

本实验对机器人导航算法进行测试, 验证 Astar 算法是否能够正确地, 以较少代价需找最优路径.

图47为 Astar 导航算法实验图. 由左到右, 由上到下分别为 1-9 号子图. 子图 1 显示在导航开始前对地图的预处理, 通过添加导航网格的方式生成可供导航的拓扑图结构. 节点之间连接的边表示节点之间的可达性. 用蓝色叉号表示的点为起点与终点 (左边起

点, 右边终点). 子图 2,3,4,5,6,7 显示了算法搜索整个解空间以寻找最优解的过程. 图中绿色叉号标记节点为在 OPEN 表中的节点. 红色叉号标记的是 CLOSE 表中的节点. 在此过程中, 算法在启发函数的指引下依次寻找" 代价" 最小的点并进行扩展, 最终找到终点. 子图 8,9 显示寻找到终点后, 通过父节点关系反推起始点的过程 (用黑色叉号表示最终生成的路径).

需要说明的是, 在最后一张图标记出的最优路径中, 在红色方框标记的区域路径出现了弯折, 而非直线前进. 这是因为在算法实现过程中, 为节约计算量, 将斜向前进与直线前进赋予了相同的权重. 在路径后处理环节中可以矫正这些错误.

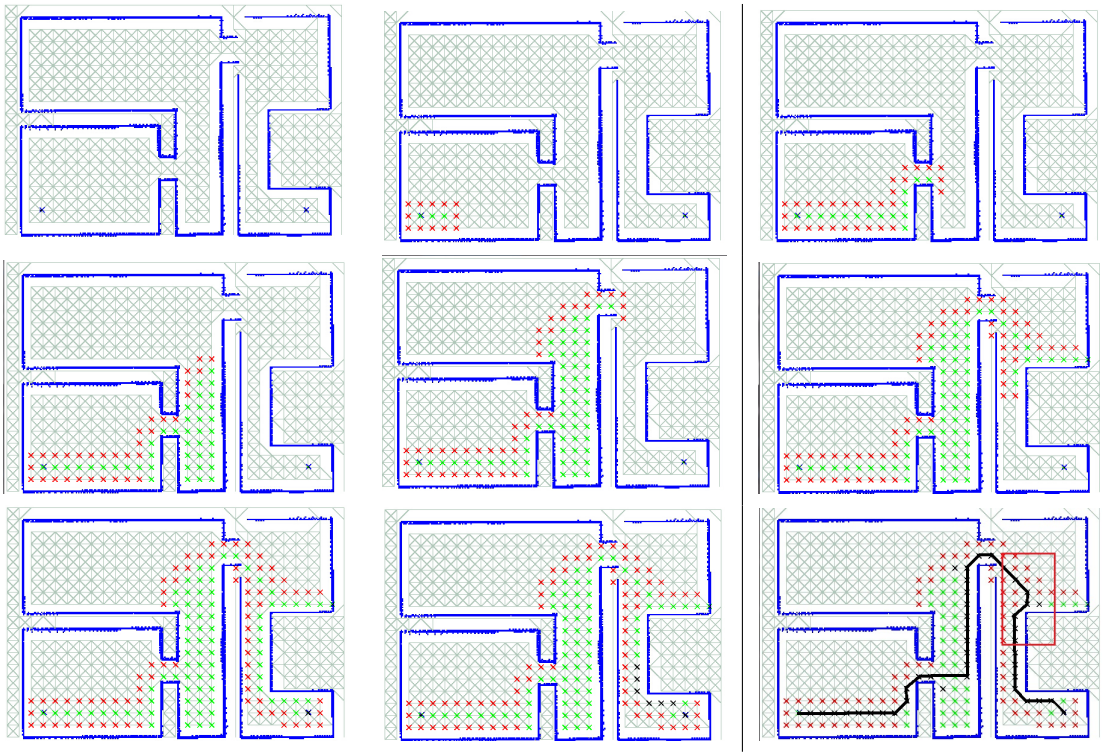


图 47: 基于 Astar 的导航算法测试图

7.4 实际测试

实际测试中, 本文使用 PeopleBot-sh 移动机器人平台 (图50) 进行机器人工作环境地图构建, 测试场地为哈尔滨工业大学机器人研究所三楼智能服务机器人研究室. 场地中包括走廊, 室内, 格子间, 小房间等场景. 环境内桌椅, 杂物较多, 同时包含行人等移动目标, 给激光传感器 SLAM 带来了一定的挑战.

本次测试使用里程计为机器人双轮上的光电码盘, 通过 Aria 模块读取包含惯性定位误差的机器人位姿. 激光传感器使用第三方公司 HOKUYO 生产的 UST-10LX 激光传感器 (图51). 传感器供电从机器人电池组引出. 机器人与传感器数据通过以太网发送至上位机进行处理.

UST-10LX 传感器参数如下, 该传感器被架设在机器人底部, 与机器人用电工胶布连接:

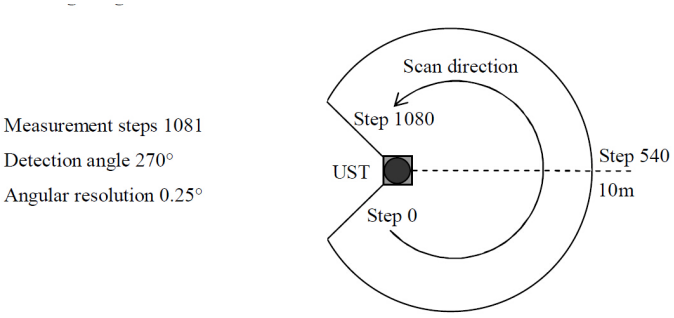


图 48: UST-10LX 传感器参数图

Product name	Scanning Laser Range Finder
Supply voltage	12VDC/24VDC
Supply current	150mA or less (during start up 450mA is necessary.)
Detection range	0.06m to 10m (white Kent sheet)
Max. detection distance	30m
Accuracy	±40mm (*1)
Dimensions(W×D×H)	50×50×70mm(sensor only)
Weight	130g (Excluding cable)

图 49: UST-10LX 基本参数

上位机选择神舟 K660E-I7 笔记本电脑 (图50中笔记本电脑). 通过综合传感器扫描结果与机器人里程计信息构建全局地图, 构建结果如图52所示.

图 50: PeopleBot-sh 机器人与机载电脑外形图

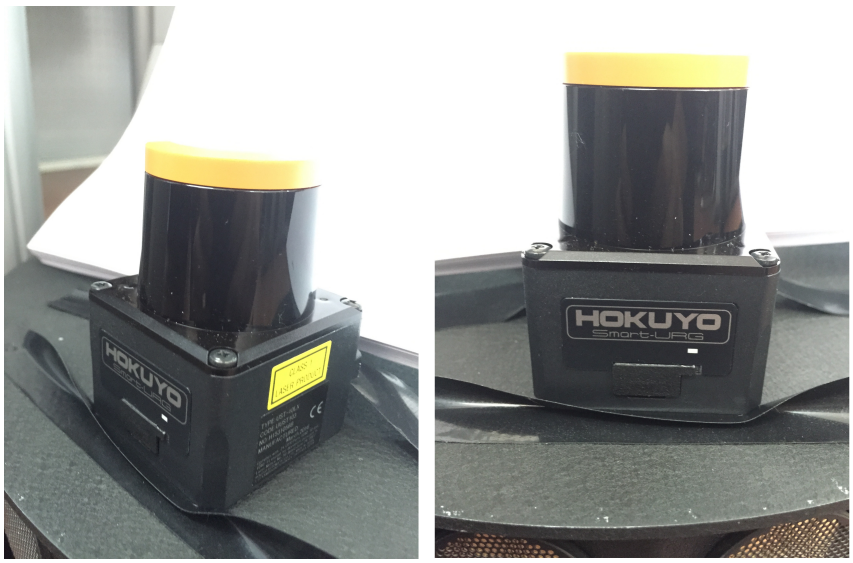


图 51: HOKUYO UST-10LX 外形图

图52显示了激光传感器实际扫描结果. 与仿真环境相比, 机器人实际工作空间要复杂得多. 实际工作情况下产生的干扰因素有以下几点:

1. 激光传感器外壳有污渍, 划痕导致激光传感器在该点读取数据为极小值 (污渍挡住了光束)

2. 机器人侧壁挡住了激光传感器扫描范围的一部分, 导致该区域读数无效 (实际上激光传感器有一个固有的最小读数半径, 机器人侧壁正好落在这个半径内, 如图50所示)

3. 机器人工作空间内物体反射率不同, 导致对不同材质的表面, 激光传感器工作效率不同, 误差范围也不一致. 一般而言, 对于浅色材质表面 (墙), 激光传感器表现要优于深色材质表面 (桌椅). 特别的, 激光传感器对金属材质的隔断处理能力弱 (激光被反射了很大一部分). 在本次试验中铝合金隔断带来了一些障碍. 通过增加激光传感器功率, 或对这些特殊材质表面采取插值的处理方式可以在一定程度上解决这一问题.

4. 实际情况下激光传感器与里程计的误差要高于模拟平台下的误差, 对于里程计而言, 机器人转动带来的误差要远高于平动时的误差.

5. 机器人工作空间内包含的移动物体 (人) 给地图构建带来了一定的困难.

图 52: 在实际环境下进行激光扫描仪数据收集示意图

图53显示了机器人地图分割算法对全局地图的分割结果. 红框部分代表自主地图分割算法对环境的分割结果. 可以看出, 地图分割算法将全局地图分为了七个各自关联度较低的区域, 这七个区域分别为大厅, 过道, 三条格子间中间的过道与侧面的小房间, 该分割结果与人类对环境的认识相似, 分割结果较好.

图 53: 地图分割结果

7.5 小结

本章对机器人地图构建, 地图分割算法进行了测试. 测试包括模拟平台测试 (MobileSim 仿真平台) 与实际测试. 测试结果表明本文提出算法能够正确处理机器人工作空间并构建地图模型. 地图分割算法能够自适应地将全局地图分割为各个子图. 在实际测试环境中, 机器人工作空间要远比仿真平台中复杂, 在这种条件下, 机器人地图构建与地图切分算法依然工作良好.

结论与展望

结论

本文实现了基于激光传感器的中小范围 SLAM 算法. 并根据 SLAM 环节收集数据, 进行了多种地图形式的构造. 包括:

- 基于矩阵存储的栅格地图
- 基于四叉树存储的栅格地图
- 基于线段的几何地图
- 拓扑地图

实验结果表明, 本文实现的地图构造形式, 能够在误差可接受的范围内构建机器人工作空间模型.

同时, 本文实现了机器人自主地图分割算法. 该算法可以在不借助人力与外界环境先验假设的情况下, 将全局地图分为多个高内聚低耦合的子图. 并且该分割算法分割结果与人类对环境的认知方式相似, 是一种用户友好的分割方式. 在地图分割领域, 本文创新性的提出了多种构造相似度矩阵的方式, 包括:

- 基于 Hausdroff 距离的相似度度量方式
- 基于 KNN 算法的相似度度量方式
- 基于特征点的度量方式
- 基于虚拟特征点与可视性的相似度度量方式

本文实现了这些相似度度量方式, 并对其结果进行了横向比较.

在构建地图的基础上, 本文还实现了地图的实施绘制与更新. 实施绘制地图可使用户更清晰的观测机器人工作状态. 地图更新技术使得地图重新生成频率降低, 解放了更多的计算资源.

在谱聚类部分, 本文还提出了基于轮廓系数的聚类结果度量方法. 并给予该度量方法实现了自适应簇数聚类.

此外, 本文还实现了基于 Astar 算法的机器人导航算法, 可实现地图内任意点对点的最优路径规划.

在本文的最后, 介绍了本平台的搭建方法, 内部设计思路与部分实现细节. 在该部分中本文提出一个相对高效的, 基于有限自动机的机器人避障探索方法, 节约了实验的人力. 在实验结果分析部分, 实验结果表明本平台能够在一定精度条件下完成给定任务.

展望

本文所实现的机器人 SLAM 平台尽管能够完成给定任务, 但部分环节仍有改进与提高的空间. 同时, 在机器人 SLAM 领域, 优秀的结果也在不断涌现. 一些新方法, 新思路可以极大的解决制约 SLAM 算法性能的瓶颈. 立足于本平台基础上, 可以做如下几点改进:

- 激光测距仪获取的数据被局限在其安装平面内, 数据缺乏直观性. 可以结合深度视觉, kinect 等传感器, 并将三维环境模型与摄像头采集视频数据进行融合. 产生三维全景信息, 更具有直观性.

- 在地图分割试验中, 尽管使用多种方法减少时间复杂度与问题规模, 相似度矩阵的建立依然是制约算法响应时间的瓶颈. 在今后的研究中应继续作出改进.
- 在线地图分割是自主地图分割的一个重要发展方向, 分割算法在线化意味着机器人可以在移动, 探索过程中建立全局地图. 可以有效利用机器人探索时的空余计算资源. 本文对在线地图分割已有一部分研究, 在今后的研究工作中在线地图分割依然是主要方向.
- 由于单个机器人行动能力与数据处理能力有限, 可以考虑使用多台机器人分别探索给定区域, 并将结果汇总为全局地图. 多机器人, 分布式的导航算法是当前学术界研究热点问题之一, 且能够极大地提高机器人工作空间探索效率. 在这个过程中, 涉及多个机器人互相定位, 多机器人数据融合, 机器人互相定位, 网络控制中时延, 丢包对机器人的影响等问题. 是 SLAM 领域非常重要与热门的问题.
- 当前 SLAM 算法的精度依然有限, 目前仅能控制误差不过于发散, 在进一步的研究中, 应当抑制累积误差到一个比较低的水平.

致谢

本课题选题与研究过程中得到了哈尔滨工程大学与哈尔滨工业大学许多老师的帮助与支持. 在此对各位给予我帮助的老师致以真诚谢意.

- 在选题即研究阶段, 哈尔滨工业大学机电学院机器人研究所的王珂老师, 李瑞峰老师对我的研究提供了极大的支持. 在选题方面, 两位老师为我的选题进行了把关. 王珂老师提出了机器人在线地图分割这一方向, 并指导我用先锋机器人进行仿真与测试. 在聚类分析方面, 王珂老师提出了谱分割这一数学工具, 使得聚类的准确度 dedaole 得到了极大地提升. 在他们的帮助下, 本文的谱分割部分得以单独成文并在 ICMA 会议上刊发.
- 在选题与研究阶段, 哈尔滨工程大学自动化学院的周卫东老师为我进行了指导与把关, 作为毕设指导老师详细的指出了我研究过程中存在的不足及改进方法.
- 在选题与研究阶段, 哈尔滨工业大学计算机学院的臧天仪老师提供了宝贵的研究思路与发展方向.
- 在四月份到五月份间, 哈尔滨工程大学的曾博文老师提供了宝贵的工作地点及网络环境.

向以上帮助了我的老师, 与大学四年间教育我的老师们致以最诚挚的谢意! 向所有关心并支持我的人们表示衷心的感谢!

参考文献

[1] 赵立军, 孙立宁, 李瑞峰, and 葛连正. 室内环境下同步定位与地图创建改进算法. 机器人, 31(5):438–444, 2009.

[2] 李阳铭, 孟庆虎, 梁华为, 李帅, 陈万明, et al. 基于粒子滤波的无线传感器网络辅助同步定位与地图创建方法研究. 机器人, 30(5):421–427, 2008.

[3] 王卫华, 陈卫东, and 席裕庚. 基于不确定信息的移动机器人地图创建研究进展 ξ. 机器人, 06:563–568, 2001.

[4] 王珂, 赵立军, and 李瑞峰. 基于贯序 mb-icp 融合的机器人复杂室内地图构建. 中国自动化学会控制理论专业委员会 *B* 卷, 6, 2011.

[5] C. Nieto-Granda, J. G. Rogers, A. J. B. Trevor, and H. I. Christensen. Semantic map partitioning in indoor environments using regional analysis. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 1451–1456, Oct 2010.

[6] 李瑞峰 and 李伟招. 基于多传感器信息融合的移动机器人路径规划. 机电一体化, 8(4):20–23, 2002.

[7] Von der Hardt, Didier Wolf, René Husson, et al. The dead reckoning localization system of the wheeled mobile robot romane. In *Multisensor Fusion and Integration for Intelligent Systems, 1996. IEEE/SICE/RSJ International Conference on*, pages 603–610. IEEE, 1996.

[8] 王卫华, 陈卫东, and 席裕庚. 移动机器人地图创建中的不确定传感信息处理. 自动化学报, 29(2):267–274, 2003.

[9] Jose A Castellanos, JM Martinez, Jose Neira, and Juan D Tardós. Simultaneous map building and localization for mobile robots: A multisensor fusion approach. In *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, volume 2, pages 1244–1249. IEEE, 1998.

[10] Dieter Fox, Wolfram Burgard, Hannes Kruppa, and Sebastian Thrun. A probabilistic approach to collaborative multi-robot localization. *Autonomous robots*, 8(3):325–344, 2000.

[11] 厉茂海, 洪炳熔, and 罗荣华. 用改进的 rao-blackwellized 粒子滤波器实现移动机器人同时定位和地图创建. 吉林大学学报: 工学版, 37(2):401–406, 2007.

[12] Liang Zhao, Shoudong Huang, and Gamini Dissanayake. Linear slam: A linear solution to the feature-based and pose graph slam based on submap joining. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 24–30. IEEE, 2013.

[13] 黄庆成, 洪炳熔, 厉茂海, and 罗荣华. 基于主动环形闭合约束的移动机器人分层同时定位和地图创建. 计算机研究与发展, 44(4):636–642, 2007.

[14] Felix Endres, Jurgen Hess, Jurgen Sturm, Daniel Cremers, and Wolfram Burgard. 3-d mapping with an rgb-d camera. *Robotics, IEEE Transactions on*, 30(1):177–187, 2014.

[15] David Filliat. A visual bag of words method for interactive qualitative localization and mapping. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 3921–3926. IEEE, 2007.

[16] Ethan Eade and Tom Drummond. Unified loop closing and recovery for real time monocular slam. In *BMVC*, volume 13, page 136. Citeseer, 2008.

[17] Kok Seng Chong and L. Kleeman. Accurate odometry and error modelling for a mobile robot. In *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*, volume 4, pages 2783–2788 vol.4, Apr 1997.

[18] Ulrike Von Luxburg. A tutorial on spectral clustering. max planck institute for biological cybernetics. Technical report, Tech. Rep, 2006.