Cartographer 2D SLAM 算法研究

吴成鼎 姚剑敏 胡海龙 福州大学物理与信息工程学院

摘要：本文主要研究 Cartographer 激光雷达 SLAM 算法的工作原理以及建图效果。首先，主要分析了 Cartographer 的算法原理并对 Hector 和 Gmapping 两种传统算法做了简要分析。之后，在 ROS 系统下使用 RVIZ 软件通过 ROSbag 数据重放对 Cartographer 和 Hector、Gmapping SLAM 算法做建图仿真。并在此基础上，深入地探讨和分析三种 SLAM 算法建图效果的区别和硬件资源消耗。

关键词：SLAM Cartogrpher ROS 激光雷达

随着人工智能技术的逐步发展与成熟，室内自主服务型机器人成为众多企业的研究热点，具有广泛的应用空间和广阔的发展前景。近年来，众多初创公司纷纷展出了自己的室内服务机器人产品，但具有室内自主导航功能的产品只占一小部分。完成机器人的室内即时定位与地图构建（Simultaneous Localization and Mapping，SLAM），使得机器人在陌生环境中能够“看到”自己所处的位置，并准确地移动到目标地点，是提升机器人智能化程度的重要指标。

目前，室内服务型机器人的自主定位与导航系统通常选用的 SLAM 方法分为两种，一种是使用视觉摄像头，另一种则是利用激光雷达。视觉 SLAM 利用搭载的单目或者双目摄像头扫描周围的环境，利用数学运算和几何、三角法测绘房间的地图。由于图像内包含的信息量太大，运算起来比较消耗 CPU 和内存，很难做到实时显示，目前很难达到实用的效果。应用激光雷达进行建图是向目标发射激光探测信号，然后将接收到的从目标反射回来的信号与发射信号进行比较，作适当处理后，就可获得目标的有关信息，如目标距离、方位、高度、速度、姿态、甚至形状等参数。激光的指向性特别好，是目前技术最可靠的定位技术，而且也最为实用。

随着低成本激光雷达的研制和逐步推出，激光 SLAM 的产品化可能性也得到了极大的提高。本研究着重于验证目前开源的激光雷达 SLAM 算法进行室内建图与定位的可能性，为后期室内机器人产品的研发提供技术支持和储备。

1 Cartographer 及其他 SLAM 算法原理

1.1 Cartographer 算法原理

Google 的 Cartographer SLAM 算法使用一个安装有激光雷达的背包，提供了一个实时的室内建图方案，能够在测试人员背着背包在室内走动的过程中实时生成一个分辨率为 r=5cm 的 2D 网格地图，激光雷达的扫描帧能够在短时间内高效准确地插入到子地图帧中，为了得到好的地图表现，Cartographer 没有用传统的粒子滤波，而是用了姿态优化，激光雷达的数据帧只和子地图进行匹配，只与最近的扫描数据相关。当子地图创建停止，没有新的扫描被插入，进行局部闭环检测。所有子地图和扫描帧都自动地参与闭环检测。如果基于当前估计的姿态，在所有子地图中，有与当前帧足够接近的子地图，那么当前扫描帧就在该子地图中寻找匹配扫描帧。如果两个扫描帧位姿足够接近，就可以添加闭环优化。每隔几秒中进行一次这样的优化，当一个位置到达后新的数据帧被添加进来，同时闭环可以立刻完成，保证系统的实时性。

整个系统包括两部分，分别用局部优化和全局优化的方法对机器人位姿（$\varepsilon_x, \varepsilon_y, \varepsilon_\theta$）进行优化，前两个参数表示平移，后面一个参数表示旋转。局部优化部分是激光雷达扫描帧与子地图的匹配，得到局部优化的子地图。全局优化部分是在找到闭环扫描帧后，根据扫描帧间的位姿关系进行全局的地图优化。

1.1.1 局部优化

局部优化是将激光雷达扫描帧与子地图进行匹配的过程，迭代对准激光雷达扫描帧和子地图参考帧构建子地图。激光雷达扫描图之后被称为帧，初始扫描帧在原点 (0,0)，扫描点被描述为 H={h_k}，k=1,2,3…k。式（1）中，扫描帧到子地图的姿态变换被描述为 T_ξ，它可以严格地将扫描点从扫描帧中映射到子地图中。

$$T_\xi p = \underbrace{\begin{pmatrix} \cos\xi_\theta & -\sin\xi_\theta \\ \sin\xi_\theta & \cos\xi_\theta \end{pmatrix}}_{R_\xi} p + \underbrace{\begin{pmatrix} \xi_x \\ \xi_y \end{pmatrix}}_{t_\xi} \quad (1)$$

几个迭代的扫描帧建立一个子地图，这些子地图采取分辨率为 r（例如

5cm) 的概率网格形式。对每个网格点，我们定义一个相应的像素，每当一个新的扫描图被插入进概率网格，一组 hit 网格点和不相交的 miss 网格点概率就被计算。对于 hit 的点，我们将相邻的网格点插入 hit 集合，对于 miss 的点，除了 hit 集内的点外，将扫描中心和扫描点连接射线上的所有相关点添加到 miss 集合中来。对每个之前没有观测到的点分配 p_{hit} 和 p_{miss}，如果之前已经观测过这一点，那么按照式（2）和式（3）更新该点概率。

$$odds(p)=\frac{p}{1-p} \quad (2)$$

$$M_{new}(x)=(odds^{-1}(odds(M_{old}(x))\cdot odds(p_{hit}))) \quad (3)$$

插入扫描图到子地图之前，扫描帧姿态跟当前的子地图应用 ceres 进行优化，通过非线性最小二乘法优化式（4），对 k 个扫描点的映射情况进行叠加，一共 k 个扫描点 hit 点，变换位姿后与 submap 中的概率值进行配对，应该对应的每个地方都显示是大概率被 hit，那么匹配的比较好。

$$\arg\min_{\xi}\sum_{i=1}^{K}(1-M(S_i(\xi)))^2 \quad (4)$$

因为最小二乘问题是一个局部最优问题，故一个好的初值（位姿初值）对求解有很大影响。因此 IMU 能被用来提供位姿初值的扫描匹配的旋转变量。在缺乏 IMU 的时候，就需要提高扫描匹配的频率或匹配精度。

1.1.2 全局优化

全局优化是通过闭环检测来实现的。由于每个激光雷达扫描帧只和一个包含一些近期扫描帧的子地图进行匹配，所以会缓慢的累积建图误差，为消除累积误差，需要进行全局优化。激光雷达扫描帧被插入到子地图时的姿态被存储到内存中，当一个子地图创建完成后，有所对应的扫描帧和子地图都被考虑进来进行闭环检测。所有的扫描匹配都在后端进行，一旦一个好的闭环匹配被发现就加入到全局优化中来。

1.2 Hector SLAM 算法原理

HectorSLAM 的运行需要使用一个更新频率高、测量噪声小的激光雷达，同时不需要使用里程计就可以完成实时地图的创建和定位，这给空中飞行器和在颠簸路面上运行的机器人实现 SLAM 提供了可行性方案。利用已经获得的地图对激光束点阵进行优化，估计激光点在地图的表示和占据网格的概率。其中扫描匹配利用的是高斯牛顿的方法进行求解。找到激光点集映射到已有地图的刚体转换（x,y,theta）。为避免局部最小而非全局最优的（类似于多峰值模型的局部梯度最小了，但非全局最优）出现，地图采用多分辨率的形式。导航中的状态估计可以加入惯性测量，进行 EKF 滤波。

1.3 Gmapping 算法原理

Gmapping 是目前激光 2D SLAM 用得最广的方法，Gmapping 采用的是 RBPF（利用统计特性描述物理表达式下的结果）的方法。粒子滤波的方法一般需要大量的粒子来获取好的结果，但这必会引入计算的复杂度。粒子是一个依据过程的观测逐渐更新权重与收敛的过程，这种重采样的过程必然会代入粒子耗散问题，大权重粒子显著，小权重粒子会消失（有可能正确的粒子模拟可能在中间的阶段表现权重小而消失）。自适应重采样技术引入减少了粒子耗散问题，计算粒子分布的时候不单单仅依靠机器人的运动（里程计），同时将当前观测考虑进去，减少了机器人位置在粒子滤波步骤中的不确定性。

2 数据测试实验结果

针对 3 种激光雷达 SLAM 算法，分别使用两个地图的激光雷达 ROSbag 在机器人操作系统（Robot Operating System,ROS）的 RVIZ 软件中进行重放，首先修改 cartographer_ros 包中 launch 文件夹下的 demo_revo_lds.launch 文件，将参数 <param name="/use_sim_time" 设置为 true，末尾添加 <node name="playbag" pkg="rosbag" type="play" args="--clock $(arg bag_filename)"/>，将 rosparam use_sim_time 设置为 true 是为了配置 ROS 启用重放数据中的时间而非本机时间，这会影响整个系统所有时间 API 的输出结果。在默认情况下，ROS 使用 Ubuntu 系统的时间，也就是墙上时钟时间（Wall Clock）。但我们重放一个历史记录文件时，里面记录的是历史时间，所以需要告诉 ROS 从现在起开始启用模拟时间。然后修改 cartographer_ros 包中 configuration_files 文件夹下面的 revo_lds.lua 文件，将参数 map_frame 设置为 map，tracking_frame 和 published_frame 设置为激光雷达的 frame 名称 laser，use_horizontal_laser 设置为 true，use_horizontal_multi_echo_laser 设置为 false，TRAJECTORY_BUILDER_2D.use_imu_data 设置为 false，修改完以上两个文件后对整个 Cartographer 进行重新编译，最后运行 roslaunch cartographer_ros demo_revo_lds.launchbag_filename:=${HOME}/[数据包名称]，该命令会自动启动 roscore 和 rviz，实际使用中应将上面命令中 bag_filename:= 后面修改为自己 Rosbag 所在的详细地址和包名。

图 1 是分别使用 Cartographer、HectorSLAM 和 Gmapping 这三种算法

重放名为 Team_Hector_MappingBox_RoboCupGermanOpen2011_Arena 的 ROSbag 数据包生成的结果图，可以看出整体上 HectorSLAM 算法对外部轮廓和室内细节都刻画得很好，Cartographer 算法在外部轮廓上出现部分重叠，不过室内细节和未重叠部分描绘得更为清晰和准确，而 Gmapping 算法在外部轮廓上描绘得比较准确，不过在室内细节方面则较为模糊且有些许错位。

如图 2 所示是重放名为 Team_Hector_MappingBox_L101_Building 的 ROSbag 数据包生成的结果图。从图中可以看出 Cartographer 算法在地图外部轮廓和室内细节描绘上都十分清晰和准确，而 HectorSLAM 由于没有闭环检测功能，在图中出现了闭环错误而导致的前后两次建图重叠，Gmapping 算法在外部轮廓描绘上比较准确，只有少量的重叠，不过内部细节描绘模糊，无法准确描绘出细节。

表 1 为三种算法的运行内存和 CPU 占用率，在算法运行时使用 top 命令输出，计算机配置为 Intel Coretm2 Quad CPU Q8300 @ 2.50GHz x 4，内存 1.9Gib。从表中可以看出 Cartographer 算法在内存占用率和 CPU 占用率上都是最高的，Hector 算法两项都较低，而 Gmapping 算法在内存占用上较低，在 CPU 占用上也较高。

3 结论

目前的开源 SLAM 算法包括 Gmapping、HectorSLAM 和 Google 开源的 Cartographer 等系统。Gmapping 与 HectorSLAM 计算量小，对 CPU 和内存的要求较低，但不包含闭环检测功能。Google 的 Cartographer 目前在 ROS 环境中使用 RVIZ 软件仿真，计算量较高，CPU 占用率高，内存损耗也较大，但是如果能够脱离 ROS 环境运行并且根据具体硬件环境和使用场景进行一些优化，应该可以节省大量的 CPU 和内存。目前对 Gmapping、HectorSLAM 和 Cartographer 都进行了测试，结果显示，Cartographer 对场景考虑更全面，更容易建图成功。

图 1 Team_Hector_MappingBox_RoboCupGermanOpen2011_Arena 数据包测试结果图

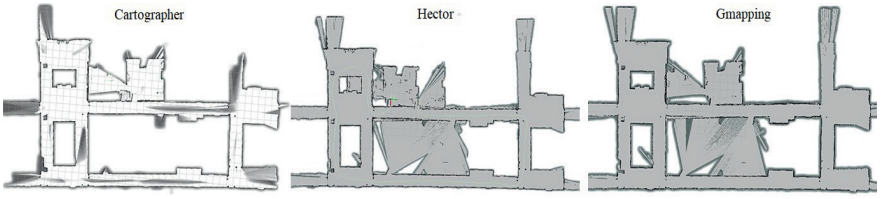

图 2 Team_Hector_MappingBox_L101_Building 数据包测试结果图

表 1 算法运行内存和 CPU 占用率

	Cartographer	Hector	Gmapping
内存占用 /%	17.2	5.3	1.5
CPU 占用 /%	306	5.3	103

参考文献

[1] 徐曙 . 基于 SLAM 的移动机器人导航系统研究 [D]. 华中科技大学 , 2014.

[2] 李磊 , 叶涛 , 谭民 , 等 . 移动机器人技术研究现状与未来 [J]. 机器人 , 2002, 24(5):475–480.

[3] 杨明 , 王宏 , 何克忠 , 等 . 基于激光雷达的移动机器人环境建模与避障 [J]. 清华大学学报 (自然科学版), 2000, 40(7):112–116.

[4] Hess W, Kohler D, Rapp H, et al. Real–time loop closure in 2D LIDAR SLAM[C]. IEEE International Conference on Robotics and Automation. IEEE, 2016:1271–1278.

[5] Olson E B. Real–time correlative scan matching[C]. IEEE International Conference on Robotics and Automation. IEEE Press, 2009:1233–1239.

[6] Santos J M, Portugal D, Rui P R. An evaluation of 2D SLAM techniques available in Robot Operating System[C]. IEEE International Symposium on Safety, Security, and Rescue Robotics. IEEE, 2014:1–6.

[7] Kohlbrecher S, Stryk O V, Meyer J, et al. A flexible and scalable SLAM system with full 3D motion estimation[C]. IEEE International Symposium on Safety, Security, and Rescue Robotics. IEEE, 2011:155–160.

[8] Grisetti G, Stachniss C, Burgard W. Improved Techniques for Grid Mapping With Rao–Blackwellized Particle Filters[J]. IEEE Transactions on Robotics, 2007, 23(1):34–46.

[9] Grisettiyz G, Stachniss C, Burgard W. Improving Grid–based SLAM with Rao–Blackwellized Particle Filters by Adaptive Proposals and Selective Resampling[C]. IEEE International Conference on Robotics and Automation. IEEE, 2005:2432–2437.

[10] Quigley M, Conley K, Gerkey B P, et al. ROS: an open–source Robot Operating System[C]. ICRA Workshop on Open Source Software. 2009.

CATV