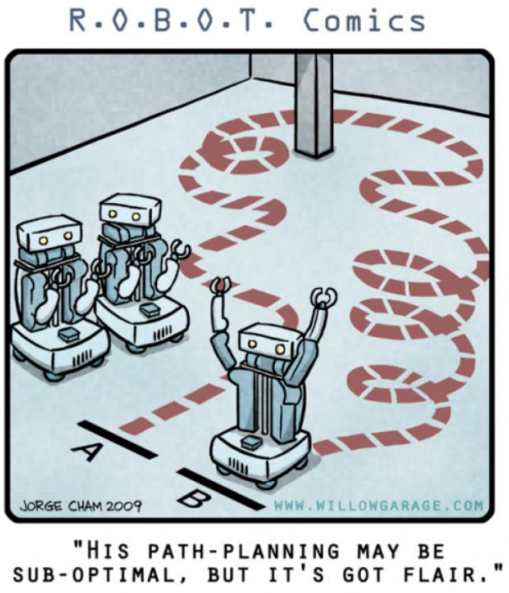
轻舟机器人 navigation 导航栈介绍

AI 航团队

上一篇教程已经将轻舟机器人的自主导航基本原理介绍给大家，大家一定有一个疑问，这么多复杂的算法，该如何下手，又该重点学习那些部分知识呢？如何将这些算法与实际硬件相结合呢？其实 ROS 机器人操作系统已经为我们提供了整套开源的示例算法，帮助我们快速实现 ROS 机器人的自主导航功能，在此基础上进一步进行各类算法的研究。

1. ros navigation stack 概述

ros navigation stack 是 ros 经典的导航包集合，在通过 gmapping 等方法得到地图后可以实现定位、全局路径规划、动态局部规划等一系列导航核心功能。ros 官网有丰富的代码和配套的学习资料 http://wiki.ros.org/navigation?distro=melodic



我们可以到 ROS 官网下载 navigation 导航栈，其下边包含 16 个 package，每个 package 完成特定的功能，共同为导航服务。

将 navigation 栈下载后，如下图所示：

名称	修改日期	类型	大小
amcl	2020/2/13 星期...	文件夹	
base_local_planner	2020/2/13 星期...	文件夹	
carrot_planner	2020/2/13 星期...	文件夹	
clear_costmap_recovery	2020/2/13 星期...	文件夹	
costmap_2d	2020/2/13 星期...	文件夹	
dwa_local_planner	2020/2/13 星期...	文件夹	
fake_localization	2020/2/13 星期...	文件夹	
global_planner	2020/2/13 星期...	文件夹	
map_server	2020/2/13 星期...	文件夹	
move_base	2020/2/13 星期...	文件夹	
move_slow_and_clear	2020/2/13 星期...	文件夹	
nav_core	2020/2/13 星期...	文件夹	
navfn	2020/2/13 星期...	文件夹	
navigation	2020/2/13 星期...	文件夹	
rotate_recovery	2020/2/13 星期...	文件夹	
voxel_grid	2020/2/13 星期...	文件夹	
.gitignore	2020/2/13 星期...	GITIGNORE 文件	1 KB
README.md	2020/2/13 星期...	MD 文件	1 KB

2. 硬件需求

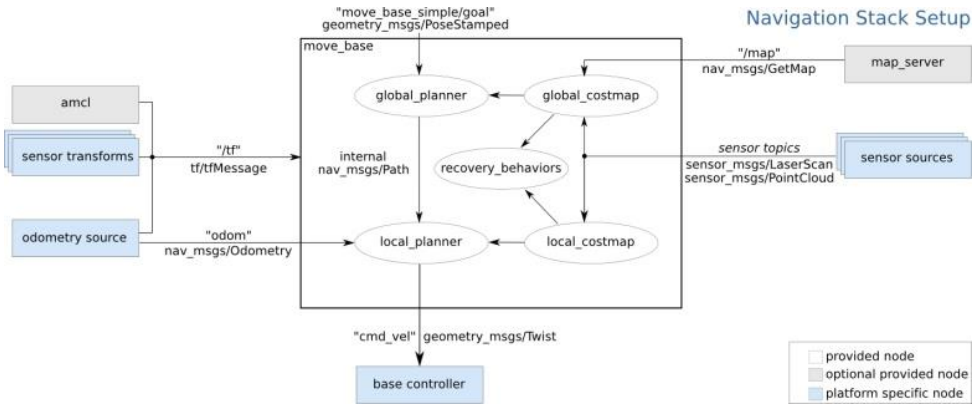
虽然导航功能包集被设计成尽可能的通用，在使用时仍然有三个主要的硬件限制：

（1）它是为差分驱动的完全约束的轮式机器人设计的。它假设移动基站受到理想的运动命令的控制并可实现预期的结果，命令的格式为：x 速度分量，y 速度分量，角速度(theta)分量。

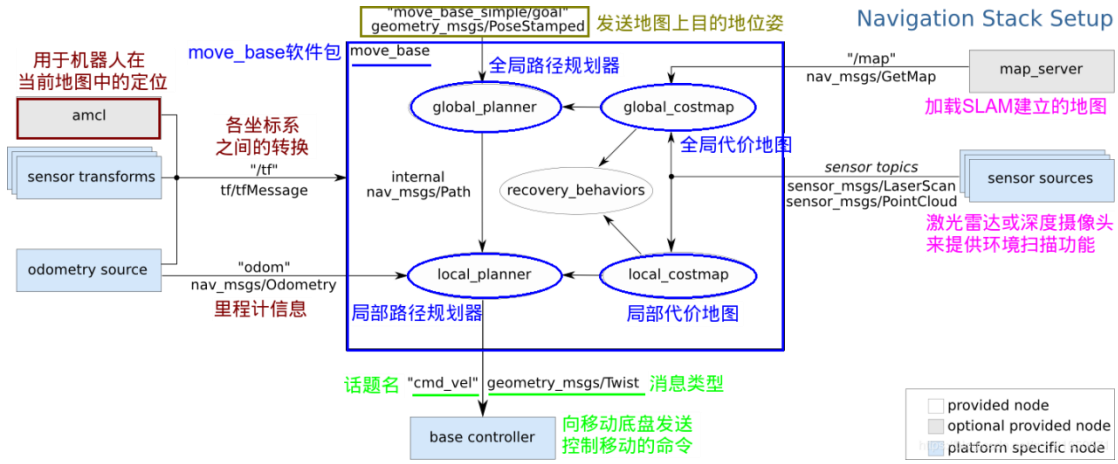
（2）它需要在移动基站上安装一个平面二维激光。这个激光用于构建地图和定位。

（3）导航功能包集是为正方形的机器人开发的，所以方形或圆形的机器人将是性能最好的。 它也可以工作在任意形状和大小的机器人上，但是较大的矩形机器人将很难通过狭窄的空间，例如门道。

3.导航框架



上图对应的各模块功能如下：



由此可以得出，导航共有三个核心功能：

（1） 定位，amcl（adaptive Monte Carlo localization）实现，简单来说就是放很多个 pose(x, y, z)，然后估计下哪个 pose 从地图中看到的跟雷达的数据最符合。

（2） 全局规划，意思就根据目标位置规划出全局路径。

（3）局部规划，根据实时测的数据规划，尽量沿着全局规划走，但可以实现动态避障。

navigation 栈是 2D 的导航功能包集，通过接收里程计数据、tf 坐标变换树以及传感器数据，为移动机器人输出目标位置以及安全速度。概念层面上讲，导航功能包集是相当简单的。它从里程计和传感器数据流获取信息，并将速度命令发送给移动基站（比如你的机器人）。但是，想要在任意机器人上使用导航功能包集可能有点复杂。使用导航功能包集的先决条件是，机器人必须运行 ROS，有一个 tf 变换树，使用正确的 ROS Message types 发布传感器数据。而且，我们需要在高层为一个具有一定形状和动力学特点的机器人配置导航功能包集。

4. navigation 栈各个功能包的作用

acml：是一个针对在二维移动的机器人的基于概率定位系统。它实现了自适应蒙特卡罗滤波的定位方法，并使用粒子滤波器去跟踪在已知地图中机器人的位置。

base_local_planner： 完成局部窗口内的路径规划任务，机器人行动速度的具体生成在此包当中完成。目前有两种局部路径规划算法实现，一是航迹推算法（TrajectoryROS），一是动态窗口法(DWA)，该包内部的默认实现是航迹推算法，但是留出了 DWA 的定义接口，DWA 的实现在 dwa_local_planner 中。

carrot_planner：这个规划器是一个简单的全局规划器，可以通过 nav_core::BaseGlobalPlanner 来进行调用，并且被 move_base 节点用作一个全局规划的插件。这个规划器从用户处采集到一个目标点，之后检查用户指定的目标点是否是障碍物，如果是的话沿着 robot 与目标点构成的向量向后退，直到找到一点没有障碍物位置。之后它会将此目标点作为目标发送给局部规划器和控制器。这个规划器允许机器人尽可能到达离用户指定的目标点最近的位置。

clear_costmap_recovery：为导航包提供了一种自救行为，试图通过将代价地图还原成已知区域外的静态地图从而清除出空间。

costmap_2d：通过激光或点云的数据，投影到 2D 平面上，创建代价地图，并可以设置膨胀半径。以层的概念来组织图层，默认的层有 static_layer（通过订阅 map_server 的/map 主题）来生成， obstacle_layer 根据传感器获得的反馈来生成障碍物图层， inflation_layer 则是将前两个图层的信息综合进行缓冲区扩展。

dwa_local_planner：局部规划器，提供动态窗口方法（ Dynamic Window Approach）在平面上局部导航。与 base_local_planner 类似。

fake_localization：提供了一个简单节点 fake_localization node，可以代替一个定位系统，并提供了 acml 包的 ROS API 的子集。由于较低的计算量，这个节点非常频繁的用于在仿真环境中提供完美的定位。这个节点将里程计数据转换为位置、粒子云，并以 acml 发布的数据格式发布。

global_planner：全局路径规划节点。global_planner 和后边的 navfn 的功能是一样的，实现目标点与当前点之间的全局路径规划，内部都有 Dijkstra 算法和 A*导航算法的实现，ROS 系统默认采用的是 navfn。

map_server：主要功能是读取 pgm 和 yaml 配套的地图文件，并将之转换到/map 话题发送出来（比如 rviz 中显示的地图就是订阅了该话题），将地图作为 ROS Service 发布，提供了 map_saver 节点，可以通过命令行存储地图。

move_base： 维护一张全局地图（基本上是不会更新的，一般是静态 costmap 类型），维护一张局部地图（实时更新，costmap 类型），维护一个全局路径规划器 global_planner 完成全局路径规划的任务， 维护一个局部路径规划器 base_local_planner 完成局部路径规划的任务。然后提供一个对外的服务，负责监听 nav_msgs::goal 类型的消息，然后调动全局规划器规划全局路径，再将全局路径送进局部规划器，局部规划器结合周围障碍信息（从其维护的 costmap 中查询）、全局路径信息、目标点信息采样速度并评分获得最高得分的轨迹（即是采样的最佳速度），然后返回速度值，由 move_base 发送 Twist 类型的 cmd_vel 消息上，从而控制机器人移动，完成导航任务。

move_base_msgs： 通过 MoveBase.action 文件定义产生的消息文件，用于 actionlib 与 move_base 的通信。

move_slow_and_clear：为 robot 提供一种自救行为，即清除代价地图的信息并限制机器人速度，但这不绝对安全，robot 可能会撞到某些障碍。但这是唯一一种可以与允许最大速度动态设置的局部规划器兼容的自救行为。

nav_core：该包定义了整个导航系统关键包的接口函数，包括 base_global_planner, base_local_planner 以及 recovery_behavior 的接口。里面的函数全是虚函数，所以该包只是起到规范接口的作用。

navfn：全局规划器，提供了一个快速插值的函数，可以在起始点到目标点之间快速插值，并找到代价最小的一条路径。

robot_pose_ekf：这个包用于估计 robot 的三维位置，利用扩展卡尔曼滤波的方法，建立了一个六维模型，联合了轮子里程计、IMU、视觉里程计的数据。

rotate_recovery：提供了一种自救行为，通过旋转 360 度来清除空间。rotate_recovery、clear_costmap_recovery 这两个包都继承自 nav_core 中定义的 recovery_behavior 类，具体实现是：当导航发现无路可走的时候，机器人在原地旋转，并更新周围障碍物信息看是否有动态障碍物运动开，如果能够找到路就继续走。

voxel_grid：提供一个有效的三维体素网格的实现。

以上这些功能包并非所有包都必须用到，我们可根据实际需求安装，进而可以减小程序包的体积和加快编译速度，其中比较重要的的包有 acml、base_local_planner、clear_costmap_recovery、costmap_2d、dwa_local_planner、global_planner、map_server、move_base、nav_core、navfn 等。