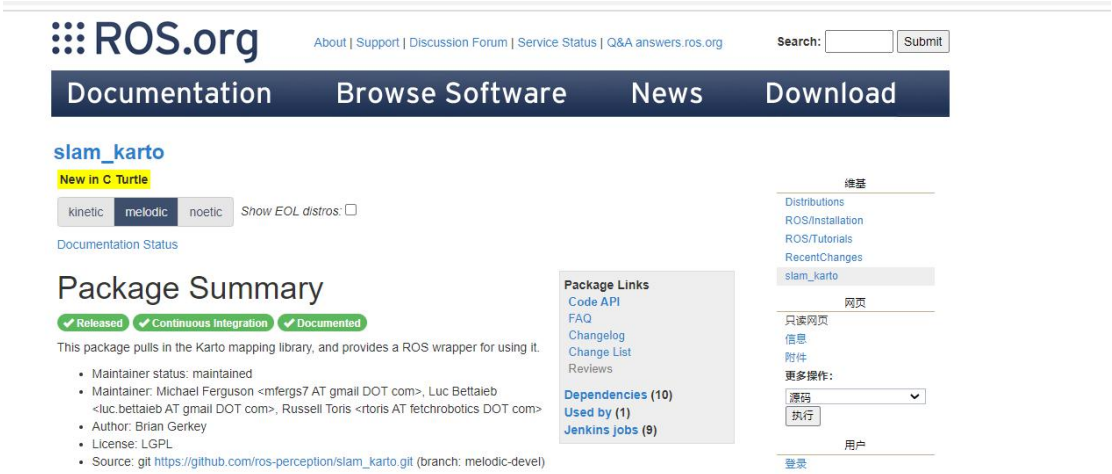
Karto SLAM 计算图

AI 航团队

1. Karto SLAM

Karto SLAM 和 Gmapping SLAM 在工作方式上非常类似，如下图所示

具体详细代码可以参见 ros.wiki.org 官网



Gmapping 算法是目前基于激光雷达和里程计方案里面比较可靠和成熟的一个算法，它基于粒子滤波，采用 RBPF 的方法效果稳定，许多基于 ROS 的机器人都跑的是 gmapping_slam。这个软件包位于 ros-perception 组织中的 slam_gmapping 仓库中。 其中的 slam_gmapping 是一个 metapackage，它依赖了 gmapping，而算法具体实现都在 gmapping 软件包中，该软件包中的 slam_gmapping 程序就是我们在 ROS 中运行的 SLAM 节点。如果你感兴趣，可以阅读一下 gmapping 的源代码。

如果你的 ROS 安装的是 desktop-full 版本，应该默认会带 gmapping。你可以用以下命令来检测 gmapping 是否安装。

命令如下：

```
apt-cache search ros-$ROS_DISTRO-gmapping
```

如果提示没有，可以直接用 apt 安装

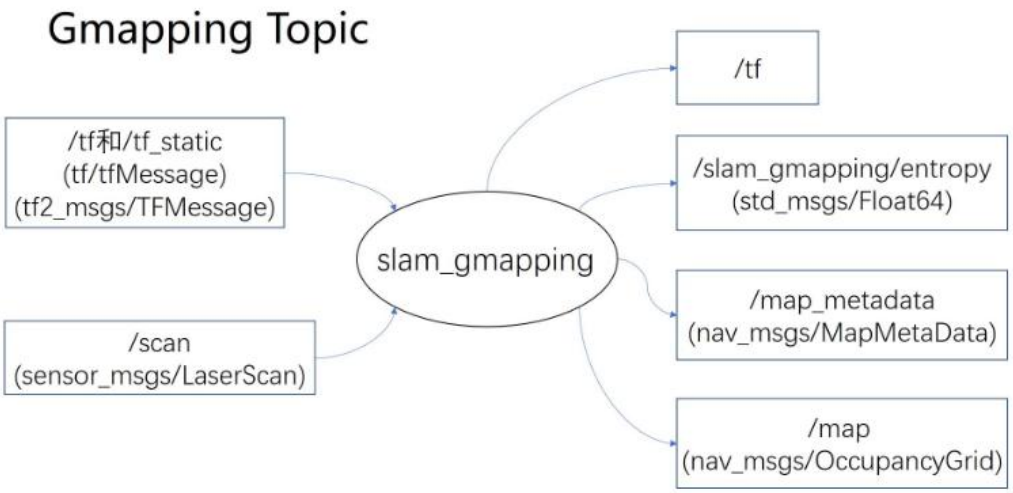
```
sudo apt-get install ros-$ROS_DISTRO-gmapping
```

gmapping 在 ROS 上运行的方法很简单

```
rosrun gmapping slam_gmapping
```

但由于 gmapping 算法中需要设置的参数很多，这种启动单个节点的效率很低。所以往往我们会把 gmapping 的启动写到 launch 文件中，同时把 gmapping 需要的一些参数也提前设置好，写进 launch 文件或 yaml 文件。 具体可参考教学软包中的 slam_sim_demo 中的 gmapping_demo.launch 和 robot_gmapping.launch.xml 文件。

gmapping 的作用是根据激光雷达和里程计（Odometry）的信息，对环境地图进行构建，并且对自身状态进行估计。因此它得输入应当包括激光雷达和里程计的数据，而输出应当有自身位置和地图。下面我们从计算图（消息的流向）的角度来看看 gmapping 算法的实际运行中的结构：



位于中心的是我们运行的 slam_gmapping 节点，这个节点负责整个 gmapping SLAM 的工作。它的**订阅**需要有两个：

/tf 以及/tf_static： 坐标变换，类型为第一代的 tf/tfMessage 或第二代的 tf2_msgs/TFMessage 其中一定得提供的有两个 tf，一个是 base_frame 与 laser_frame 之间的 tf，即机器人底盘和激光雷达之间的变换；一个是 base_frame 与 odom_frame 之间的 tf，即底盘和里程计原点之间的坐标变换。odom_frame 可以理解为里程计原点所在的坐标系。

/scan :激光雷达数据，类型为 sensor_msgs/LaserScan

/scan 很好理解，Gmapping SLAM 所必须的激光雷达数据，而/tf 是一个比较容易忽视的细节。尽管/tf 这个 Topic 听起来很简单，但它维护了整个 ROS 三维世界里的转换关系，而 slam_gmapping 要从中读取的数据是 base_frame 与 laser_frame 之间的 tf，只有这样才能

够把周围障碍物变换到机器人坐标系下，更重要的是 base_frame 与 odom_frame 之间的 tf，这个 tf 反映了里程计（电机的光电码盘、视觉里程计、IMU）的监测数据，也就是机器人里程计测得走了多少距离，它会把这段变换发布到 odom_frame 和 laser_frame 之间。

因此 slam_gmapping 会从/tf 中获得机器人里程计的数据。

需要**发布**的消息有：

/tf： 主要是发布 map_frame 和 odom_frame 之间的变换

/slam_gmapping/entropy： std_msgs/Float64 类型，反映了机器人位姿估计的分散程度

/map： slam_gmapping 建立的地图

/map_metadata： 地图的相关信息

发布的/tf 里又一个很重要的信息，就是 map_frame 和 odom_frame 之间的变换，这其实就是对机器人的定位。通过连通 map_frame 和 odom_frame，这样 map_frame 与 base_frame 甚至与 laser_frame 都连通了。这样便实现了机器人在地图上的定位。

同时，输出的 Topic 里还有/map，在上一节我们介绍了地图的类型，在 SLAM 场景中，地图是作为 SLAM 的结果被不断地更新和发布。

2. 里程计误差及修正

目前 ROS 中常用的里程计广义上包括车轮上的光电码盘、惯性导航元件（IMU）、视觉里程计，你可以只用其中的一个作为 odom，也可以选择多个进行数据融合，融合结果作为 odom。通常来说，实际 ROS 项目中的里程计会发布两个 Topic：

/odom: 类型为 nav_msgs/Odometry，反映里程计估测的机器人位置、方向、线速度、角速度信息。

/tf: 主要是输出 odom_frame 和 base_frame 之间的 tf。这段 tf 反映了机器人的位置和方向变换，数值与/odom 中的相同。

由于以上三种里程计都是对机器人的位姿进行估计，存在着累计误差，因此当运动时间较长时，odom_frame 和 base_frame 之间变换的真实值与估计值的误差会越来越大。你可能会想，能否用激光雷达数据来修正 odom_frame 和 base_frame 的 tf。事实上 gmapping 不是这么做的，里程计估计的是多少，odom_frame 和 base_frame 的 tf 就显示多少，永远不会去修正这段 tf。gmapping 的做法是把里程计误差的修正发布到 map_frame 和 odom_frame 之间的 tf 上，也就是把误差补偿在了地图坐标系和里程计原点坐标系之间。通过这种方式

来修正定位。

这样 map_frame 和 base_frame，甚至和 laser_frame 之间就连通了，实现了机器人在地图上的定位。

3. 服务

slam_gmapping 也提供了一个服务：

/dyn 该 srv 定义如下： nav_msgs/GetMap.srv

Get the map as a nav_msgs/OccupancyGrid

---

nav_msgs/OccupancyGrid map

amic_map： 其 srv 类型为 nav_msgs/GetMap，用于获取当前的地图。

可见该服务的请求为空，即不需要传入参数，它会直接反馈当前地图。

4. 参数

slam_gmapping 需要的参数很多，这里以 slam_sim_demo 教学包中的 gmapping_demo 的参数为例，注释了一些比较重要的参数，具体请查看 ROS-Academy-for-Beginners/slam_sim_demo/launch/include/robot_gmapping.launch.xml

```
<node pkg="gmapping" type="slam_gmapping" name="slam_gmapping" output="screen">
<param name="base_frame" value="$(arg base_frame)"/> <!--底盘坐标系-->
<param name="odom_frame" value="$(arg odom_frame)"/> <!--里程计坐标系-->
<param name="map_update_interval" value="1.0"/> <!--更新时间(s)，每多久更新一次地图，不是频率-->
<param name="maxUrange" value="20.0"/> <!--激光雷达最大可用距离，在此之外的数据截断不用-->
<param name="maxRange" value="25.0"/> <!--激光雷达最大距离-->
<param name="sigma" value="0.05"/>
<param name="kernelSize" value="1"/>
<param name="lstep" value="0.05"/>
<param name="astep" value="0.05"/>
<param name="iterations" value="5"/>
<param name="lsigma" value="0.075"/>
<param name="ogain" value="3.0"/>
<param name="lskip" value="0"/>
<param name="minimumScore" value="200"/>
<param name="srr" value="0.01"/>
<param name="srt" value="0.02"/>
<param name="str" value="0.01"/>
<param name="stt" value="0.02"/>
<param name="linearUpdate" value="0.5"/>
<param name="angularUpdate" value="0.436"/>
<param name="temporalUpdate" value="-1.0"/>
<param name="resampleThreshold" value="0.5"/>
<param name="particles" value="80"/>
<param name="xmin" value="-25.0"/>
<param name="ymin" value="-25.0"/>
<param name="xmax" value="25.0"/>
<param name="ymax" value="25.0"/>
<param name="delta" value="0.05"/>
<param name="llsamplerange" value="0.01"/>
<param name="llsamplestep" value="0.01"/>
<param name="lasamplerange" value="0.005"/>
<param name="lasamplestep" value="0.005"/>
<remap from="scan" to="$(arg scan_topic)"/>
</node>
```