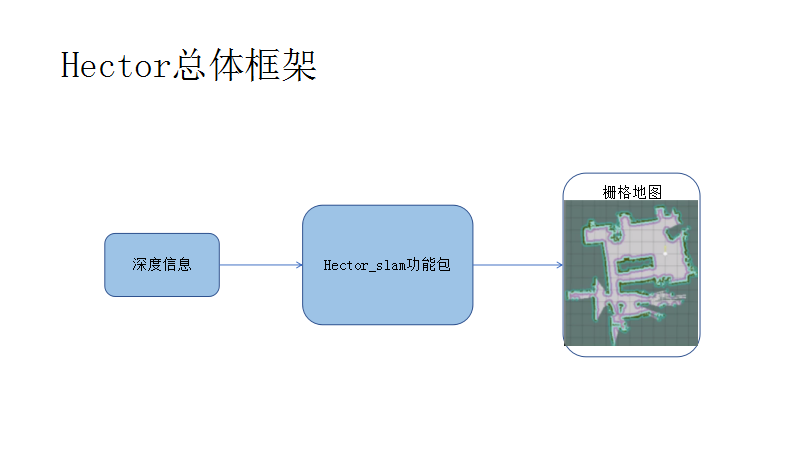
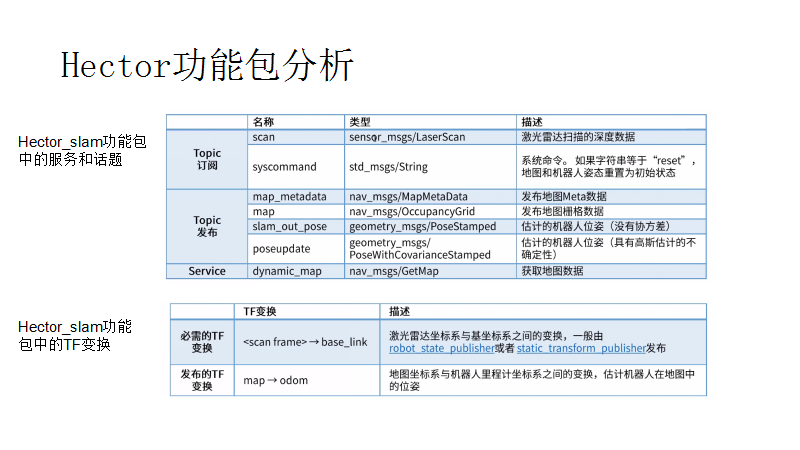


接下来我们介绍第二个比较常用的功能包，这个功能包的名字叫做hector\_slam，这个功能包也是基于激光雷达去做一个二维的栅格构建，它使用的也是另外一种高斯牛顿的方法来做地图构建的，但是它和我们之前介绍的gmapping算法最大的不同是hector功能包时不需要机器人发布里程计数据的。



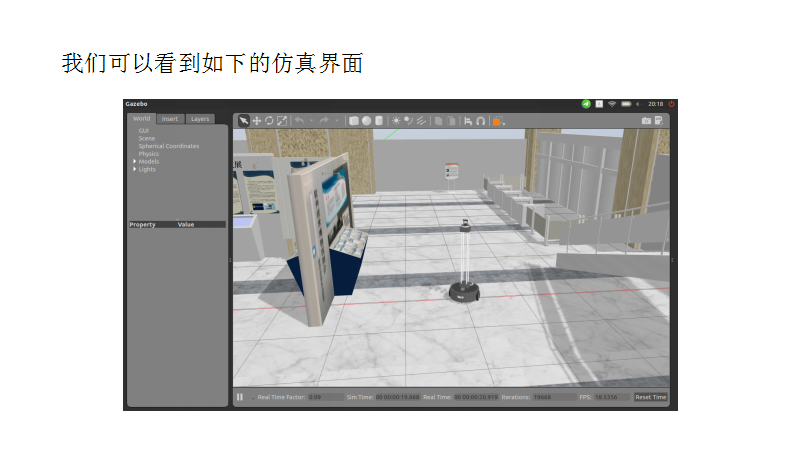
大家可以看这个框架图，可以看到中间是hector这样一个节点，它的整个的输入只需要我们传入深度信息，也就是激光雷达信息或者电源的信息就可以了，通过这种深度信息可以实现里程计的估算，最终可以构建出和gmapping一样的栅格地图。所以它和gmapping最大的不同就是不需要里程计信息。如果大家机器人上面没有安装里程计信息的话，可以用hector功能包。但是因为hector功能包只有一个深度信息的输入，所以对深度信息的依赖比较强，如果大家的激光雷达的精度非常高，并且机器人运行速度不是很快的话，建图精度还是可以的。如果在建图过程中，运行速度较快，可能产生定位不准确，地图发生偏移的情况。



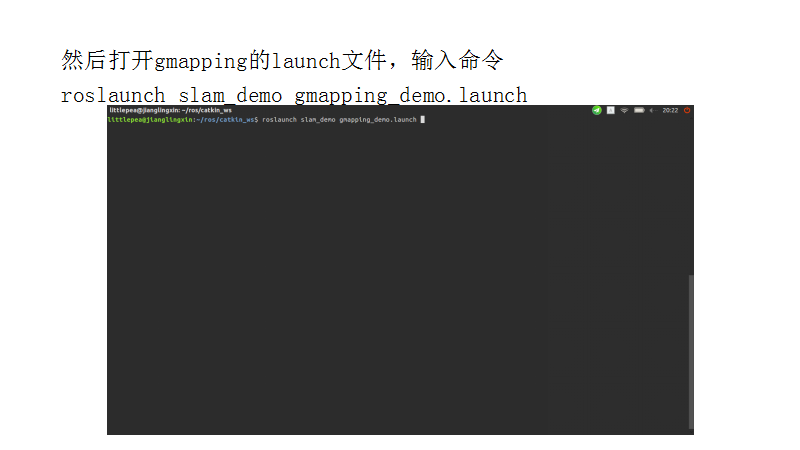
关于hector功能包提供的话题以及服务接口，在订阅的话题中我们可以看到它是不需要里程计信息的，只需要scan激光雷达信息就可以了，而它最终发布的话题同样是一个二维的栅格地图信息，同时它也提供了一个获取地图信息的这样一个服务。它同时也提供了一系列的tf变换，因为这里不需要提供一些里程计信息，因此这里里程计与baselink的变换是不需要的，最终发布的变换也是map和坐标系之间的变换。



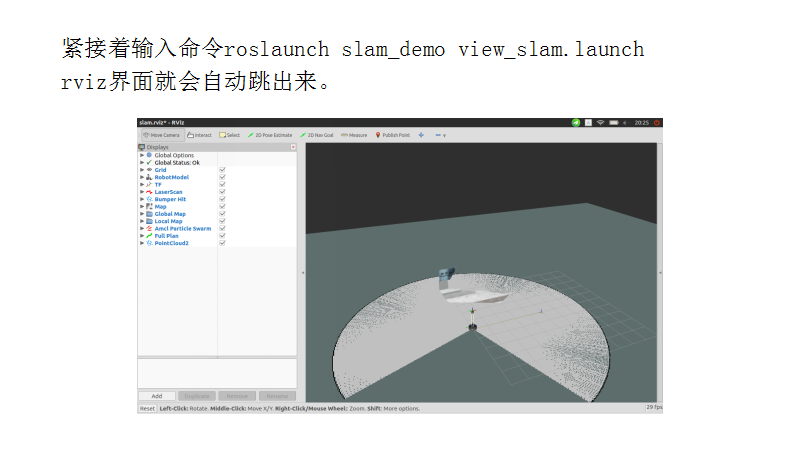
下面我们就在自己建立的仿真环境中运行gmapping功能包。gmapping功能包可以在GitHub网站下载或者课程资源中拷贝到当前的工作目录，编译成功后。首先打开之前建立的gazebo仿真环境，输入命令roslaunch robot\_sim\_demo robot\_spawn.lacunch



就可以打开如图所示的界面，我们可以看到xbot以及中科院软件所的环境。



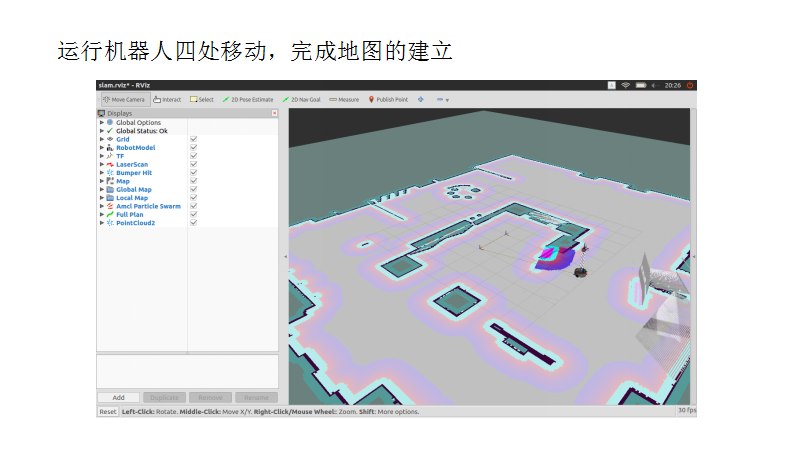
接着就是打开gmapping功能包的launch文件，输入命令roslaunch slam\_demo hector\_slam.launch。



接着输入命令roslaunch slam\_demo view\_slam.launch这样一个rviz的界面会自动弹出来，我们可以看到xbot的模型就在其中，另外浅灰色区域就是雷达已经探测出来的，而深灰色的区域则是还未探测的区域。



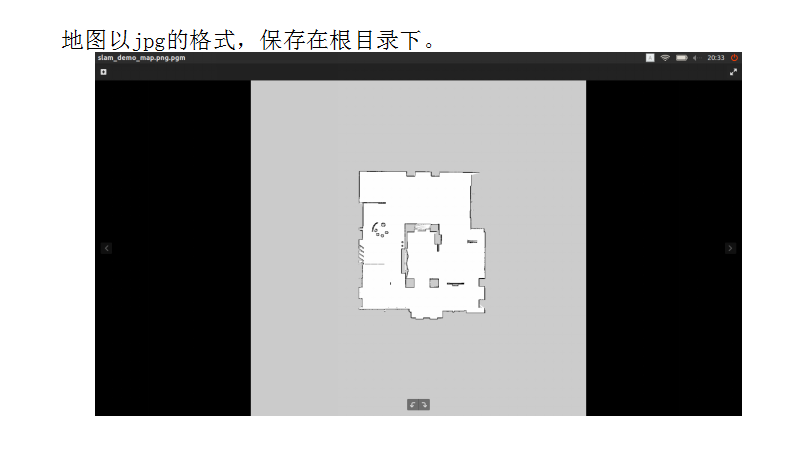
为了使机器人获得更多更加完整的地图信息，需要机器人在仿真环境中四处走动。因此打开键盘控制文件，即输入命令rosrun robot\_sim\_demo robot\_keyboard\_teleop.py。我们可以看到gazebo中的模型小车在我们的控制下运动，而rviz界面中的小车也随着控制在运动，并且浅灰色的范围越来越大。



在控制小车跑完整个地图之后，我们可以看到仿真的环境基本上出现在rviz中，基本和真实的环境相差不大，到这里基本slam的图已经建立完成了。



然后使用命令rosrun map\_server map\_server map.yaml，rosrun map\_server map\_saver -f slam\_demo.png将构建的地图保存下来。



地图一般会直接保存到根目录下面，以jpg的形式，可以找到并打开看一下，我们可以对比一下建立的地图和真实的仿真环境，还是比较吻合的。



hector功能包节点的启动其实和gmapping差不多的，一样是要去启动这样一个launch文件，启动的时候也是要求配置一系列的参数，大家同样要去看一下参数里面hector节点订阅的激光雷达的话题是不是scan，最终建立的地图是在map坐标系下面的。同时通过hector节点根据激光雷达的信息估算出里程计信息，从而把位置发布到里程计坐标系当中去，所以这里的odom坐标系是hector提供的。其他的很多参数和gmapping是差不多的，是跟算法直接相关的。如果大家对算法比较了解的话可以去调试这些参数，不是很了解的话也可以参考许多其他机器人的参数