Turtlebot3 SLAM与导航虚拟仿真实验-拓展实验二

- 本次实验课适用ROS系统版本为`melodic`和`kinetic`。如果有同学使用其他版本ROS系统而在实验过程中出现问题，恐怕无法给予帮助。
- 仿真实验和实际环境实验有较大不同，本实验仿真运行效果也许不错，但实际环境中要差很多。
- 从PDF文件复制终端命令的时候会将行末的回车换行符一并复制，可先将复制内容粘贴到一文本文件中，删除行尾换行符后再复制、粘贴到终端中执行。

目录

三、拓展实验二：三维建图

基础实验部分中我们建立了室内环境的二维地图，这能够满足一部分机器人如扫地机器人的需求；但是对于无人机、带有机械臂的机器人、个头比较高的送餐机器人来说，二维地图提供信息较少，大多数应用要求机器人了解三维空间结构，需要构建三维地图。例如在本次实验构建的二维地图中，三张桌子底下均有足够的空间可供机器人穿行，但机器人的高度再高一些，机器人从桌子底下穿行可能会发生碰撞。用于三维建图的多线激光雷达价格昂贵，在精度要求不高的情况下通常可使用RGB-D相机构建环境的三维地图。拓展实验二中，我们将使用turtlebot3机器人的RGB-D相机拍摄的深度图像计算空间点云数据，进而构建房间的简单的三维点云地图和八叉树地图。

3.1 源代码文件介绍

进行实验之前，请大家将功能包`octo_mapping`的源代码放在自己的工作空间下完成编译。功能包中的文件及其作用如下：

```
.
├── cmake                                  存放cmake模块，同学们不需要了解
|   └── FindOpenCV.cmake                   强制使用ROS自带的OpenCV，避免链接时错误
```

```
├── CMakeLists.txt                          告知cmake如何处理源文件
├── include                                 头文件文件夹
|   ├── CameraIntrinsic.hpp                 相机内参对象定义
|   └── ExperNodeBase.hpp                   ROS节点对象定义
├── launch                                  启动文件文件夹
|   ├── image_base.launch                   一“键”启动image_base节点
|   └── point_cloud_map.launch              一“键”启动point_cloud_map节点
├── package.xml                             ROS功能包定义
├── rviz                                    rviz显示配置文件
|   └── point_cloud_mapping.rviz
└── src                                     源代码文件
    ├── ImageBase                           image_base 基础图像实验
    |   └── ImageBase.cc
    ├── OctoMap                             octo_map 八叉树建图实验
    |   └── OctoMap.cc
    └── PointsCloudMap                      point_cloud_map 点云建图实验
        └── PointsCloudMap.cc
```

进行实验之前请大家先阅读一下相关代码的注释，对代码比较好的理解有助于快速完成实验。

另外这部分的实验涉及到图像和点云处理，有一定的算力需求，同学们在进行实验的过程中可以根据自己需要灵活调节参数服务器中的参数。

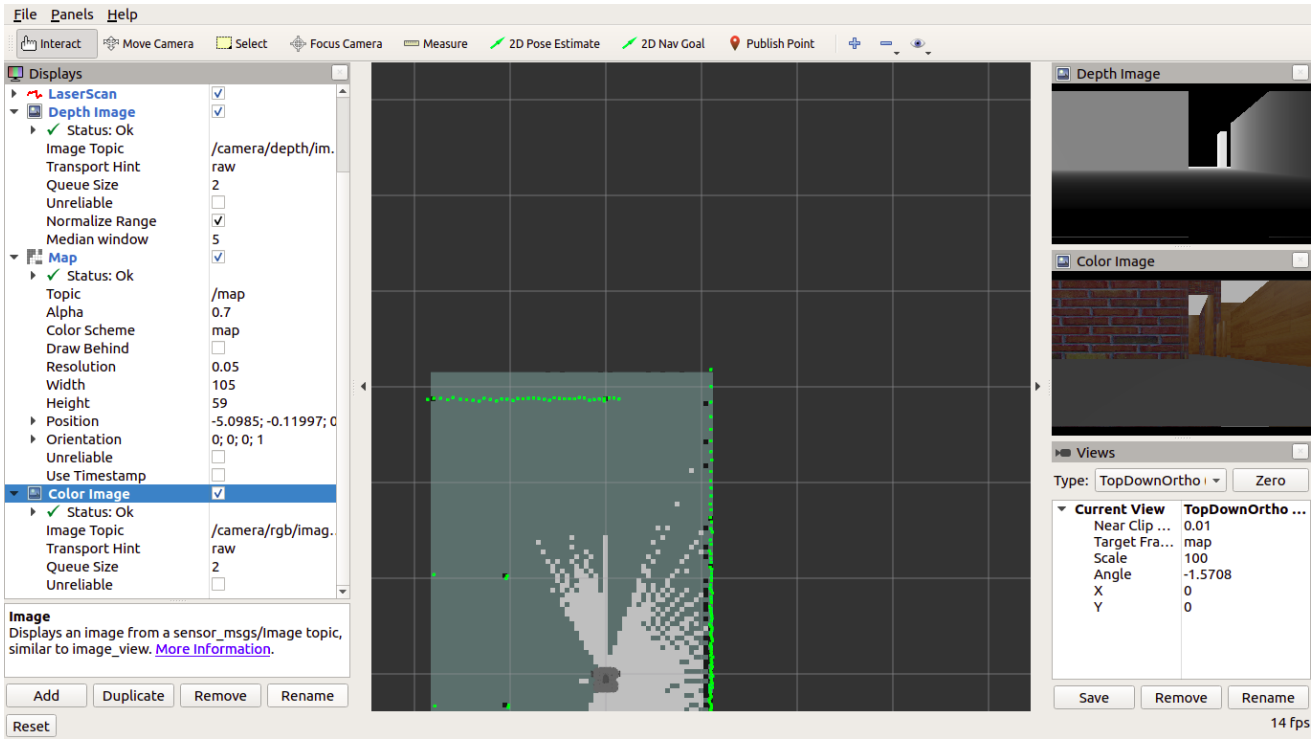
3.2 基础图像实验

在进行地图构建之前，首先我们需要掌握在ROS中接收和发布图像的技能。在本次实验中，我们使用`image_transport`接管图像的发布和订阅。运行建图实验中的两个launch文件：

```
$ roslaunch turtlebot3_gazebo turtlebot3_house.launch
$ roslaunch turtlebot3_slam turtlebot3_slam.launch slam_methods:=karto
```

> 进行拓展实验二时，均需要先运行上述两个文件。

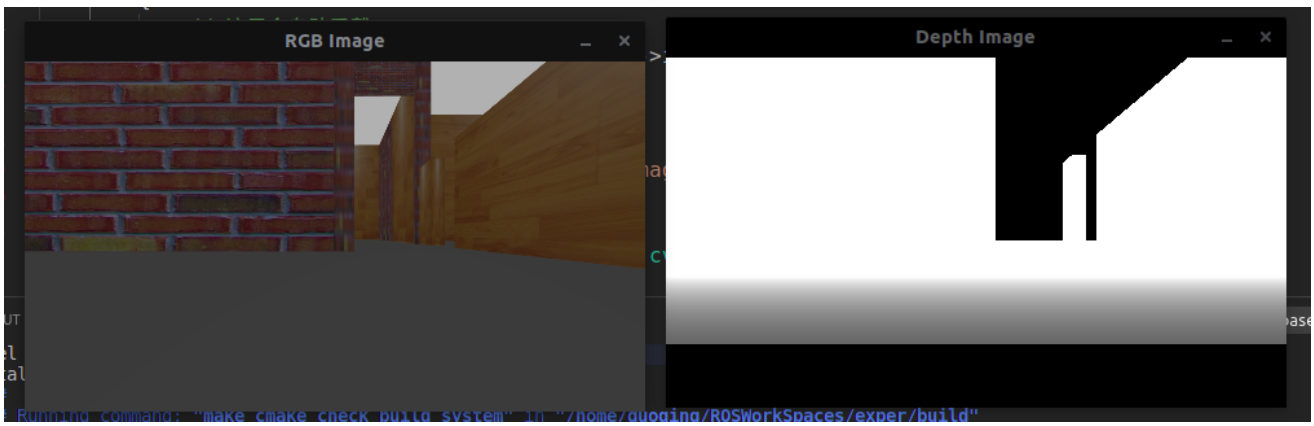
在rviz中，打开左侧的Image显示模块，`TransportHint`选`raw`，订阅`/camera/rgb/image_raw`或`/camera/depth/image_raw`就可以实时查看Turtlebot3机器人的RGB-D相机拍摄的彩色图像和深度图像了：



其中彩色图像和普通相机拍摄的图像并没有什么不同；深度图像中的每一个像素值保存的是该像素对应的空间点到相机成像平面的距离。本次实验对应的源代码文件为`$(find octo_mapping)/src/ImageBase/ImageBase.cc`，主要有四个实验阶段：订阅深度图像并显示、伪彩色化深度图像并发布、获取相机内参数据和创建launch文件。

3.2.1 第一阶段：获取深度图像

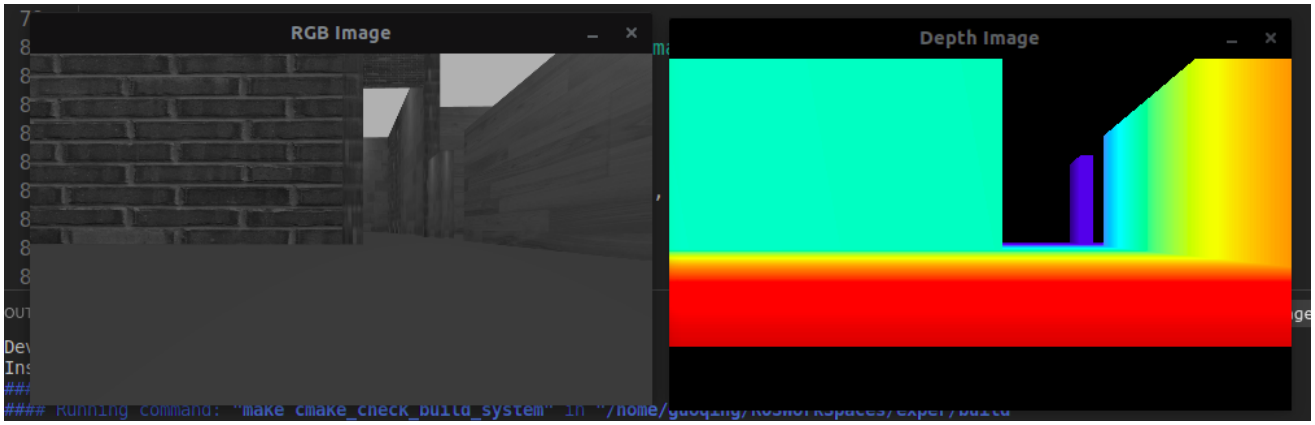
在本次实验中，图像数据均通过OpenCV库进行处理，考虑到大部分同学对OpenCV不熟悉，代码中已经给出了彩色图像的获取过程，并给出了常用函数和数据结构的介绍。彩色图像处理的基本流程是，通过`image_transport::Subscriber`获取相应topic下的图像消息，接着由`cv_bridge`将图像消息转换为OpenCV可以直接处理的数据格式，最后使用`cv::imshow()`函数显示出来。请同学们参考彩色图像的处理方式，完成对深度图像的订阅和显示，效果如下：



注意上述效果图中深度图像的显示效果和rviz中看到的效果并不相同，这是由于rviz和OpenCV显示图像的不同规则导致的，本次实验中深度图像的深度范围在0.4~5之间，但对于像素类型为浮点数的图像，OpenCV只会将0~1的部分按照灰度显示，超过1的部分直接显示成为白色。感兴趣的同学可以对修改程序，尝试将0.4~5的数值映射到0~1之间后显示；可能需要处理非数nan的问题，请参考`ImageBaseNode::ColoredDepth()`的处理方式。

3.2.2 第二阶段：处理图像并发布

搜索相关资料，对于彩色图像使用OpenCV提供的函数将彩色图像灰度化；对于深度图像，利用代码包中的`ImageBaseNode::ColoredDepth()`函数对深度图像进行伪彩色化处理；最后将二者使用`image_transport`发布出去，可以在rviz中查看。最终处理图像得到的效果如下图所示：



3.2.3 第三阶段：获取相机的内参数据

为了根据深度图计算点云，我们需要获取深度相机的内参数据。请同学们参考程序中彩色相机内参数据的获取过程，订阅`/camera/depth/camera_info`，编写深度相机内参获取的程序。要求内参数据仅获取一次即可，一旦完成获取，调用`ros::Subscriber::shutdown()`停止订阅。

示例输出：

```
guoqing@GQLU18:~/ROSWorkSpaces/exper$ rosrun octo_mapping image_base
[ INFO] [1591525464.021249413, 3963.970000000]: Initialized, scaled factor inverse = 0.2500
[ INFO] [1591525464.451913179, 3964.292000000]: RGB camera intrinsic: fx = 301.722, fy = 301.722, cx = 240.125, cy = 135.125
[ INFO] [1591525464.453773549, 3964.292000000]: Depth camera intrinsic: fx = 301.722, fy = 301.722, cx = 240.125, cy = 135.125
```

3.2.4 第四阶段：编写launch文件

为了减少对电脑的计算资源消耗，程序中默认将图像缩小至原来的四分之一后进行处理，这个缩小的倍数放在参数服务器下的`/image_base/scale_factor_inv`中。除了拓展实验一中的命令行传参，通过参数服务器传参也是ROS程序通常使用的方式，请同学们查阅资料，补充`$(find octo_mapping)/launch/image_base.launch`，实现启动`image_base`节点时自动设置参数服务器的作用，并且验证通过修改launch文件中的参数值，可以直接修改图像的缩放倍数。

> 这里需要关注一下launch文件中写的参数名称；如果你不确定这样写是否正确，可以在launch文件运行后使用`rosparam list`检查

3.3 点云建图实验

在本实验中，程序将从深度图像计算当前帧点云，并且通过不断累积组成描述整个环境的全局点云地图，对应的源代码文件为`$(find octo_mapping)/src/PointsCloudMap/PointsCloudMap.cc`整个实验分为获取图像和内参、计算单帧局部点云、获取相机位姿、全局点云地图构建和launch文件编写五个阶段。

3.3.1 第一阶段：获取图像和内参

这一段要做的内容和上一节`image_base`的实验非常相似，请同学们根据上一节实验中的内容，获取彩色图像和深度图像并进行缩放处理，同时获取一次深度相机的内参备用。注意本实验中为了节省计算资源，不需要对彩色图像和深度图像进行进一步的处理了，上个实验中的处理只是作为练习。

3.3.2 第二阶段：计算单帧局部点云

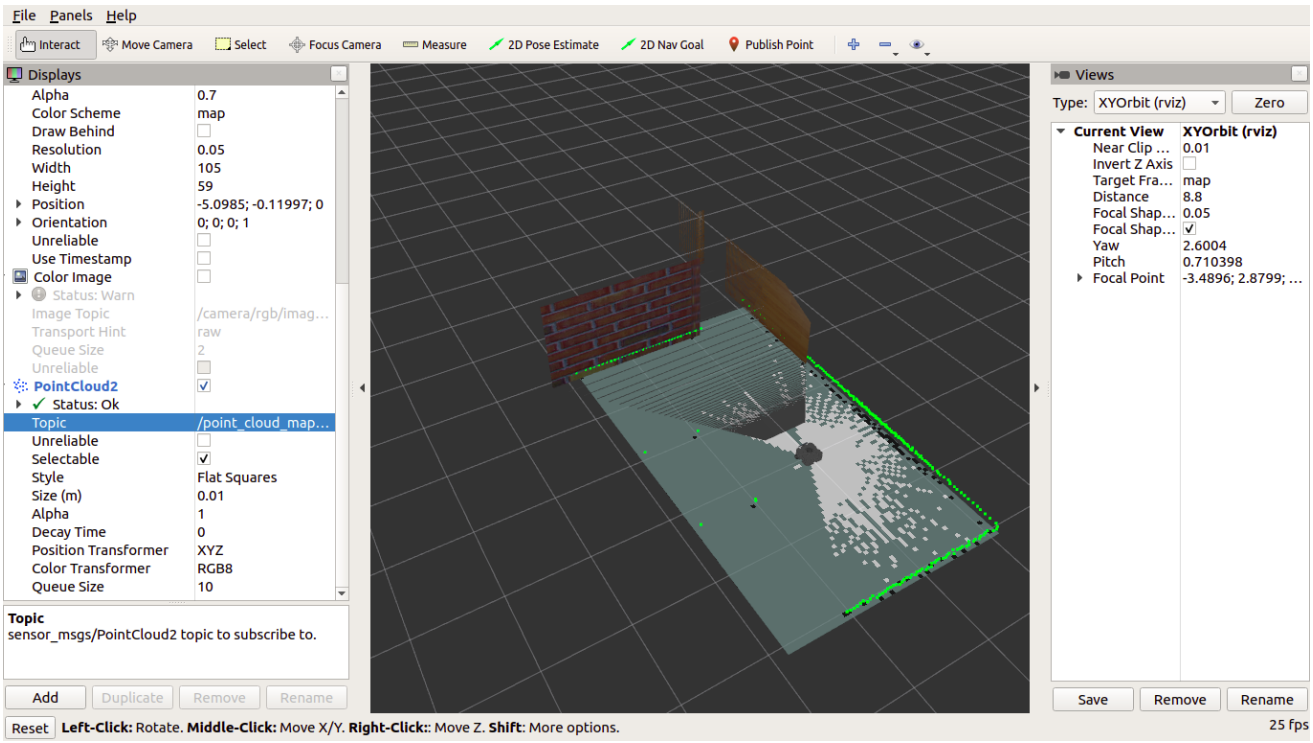
对于图像上一像素点$p(u,v)$和对应的空间点$P(x,y,z)$，根据针孔相机模型满足如下关系：

$$u = \frac{1}{z}(xf_x + zc_x)$$
$$v = \frac{1}{z}(yf_y + zc_y)$$

其中f_x、f_y、c_x、c_y、为上一阶段中获得的深度相机内参。关于针孔相机模型的推导，感兴趣的同学可以查找相关文献。实验中将直接使用该结论，请同学们完善程序，参考课本第六章点云处理的相关里程，计算并生成一帧点云信息；同时使用彩色图像中对应像素点的彩色信息为点云着色，最后发布到`/point_cloud_map/local_map`话题中。

> 实验中的仿真环境过于理想，通过查看TF变换树我们会发现彩色相机的光心和深度相机的光心是重合的，所以在实验中我们可以简单认为彩色图像和深度图像中的像素点是一一对应的；但现实中不会出现这样的情况，彩色图像和深度图像中的像素并不是一一对应，处理起来要麻烦一些。
>
> 发布点云消息时注意选择正确的坐标系。

如果一切正确，在rviz中可以添加一个PointCloud2显示模块查看`/point_cloud_map/local_map`中的局部点云，如下图所示：



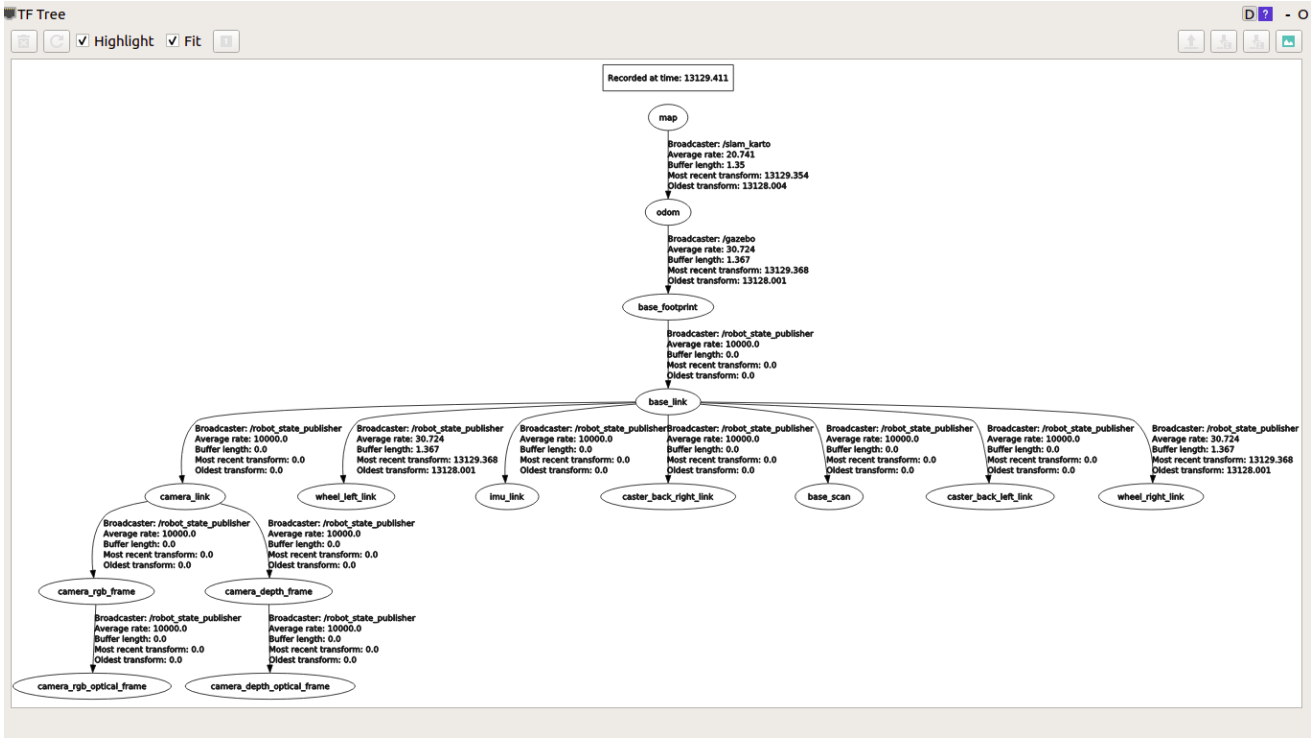
> 提示：rviz一直是俯视状态？点击右侧View窗口中的Type，可以切换不同的视图方式
>
> 有时因为点云数量较少等原因，rviz上可能无法看清。尝试调节`PointCloud2`下的`Style`、`Size`等属性，看看有什么效果？
>
> 有时点云看不清、看不到不一定是点云消息没有发布出来，不妨调节这些可视化选项看一看。

3.3.3 第三阶段 获取深度相机位姿

在获取相机位姿之前，请同学们执行（保持gazebo和rviz均处于运行状态）：

```
$ rosrun rqt_tf_tree rqt_tf_tree
```

我们可以直观看到当前的TF坐标变换树状态：

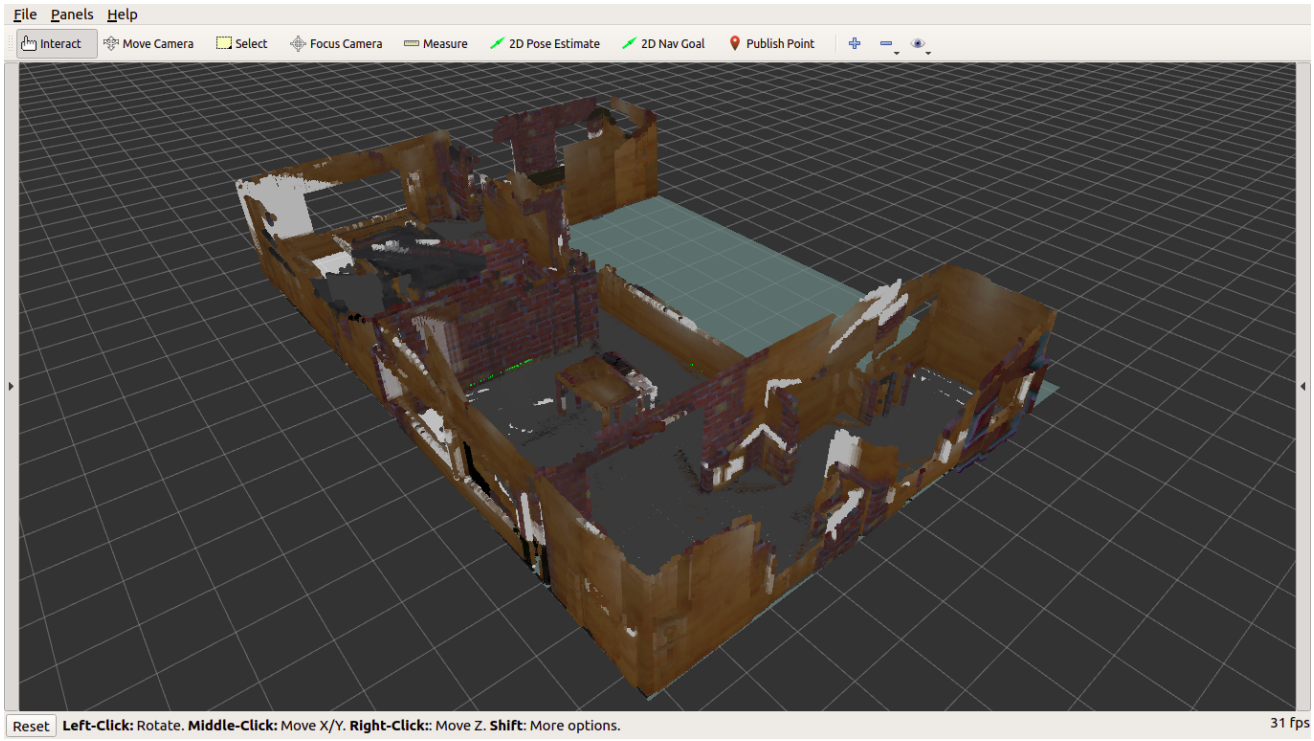


由于上一步中的点云均是基于深度相机光心坐标系计算的，在将其合并到全局点云之前，我们需要先将其坐标转换到地图坐标系`map`下。请同学们补充程序，通过TF获取`camera_depth_optical_frame`和`map`之间的坐标系变换，并将上一步的点云坐标从相机坐标系下的表示转换到`map`坐标系下的表示；考虑到同学们对三维刚体间的变换并不熟悉，代码注释中均给出了相关提示。为验证转换结果是否正确，请大家将转换后的点云发送到`/point_cloud_map/global_map`中，注意坐标系设置为`map`，通过rviz订阅该话题，显示点云。如果程序正确，则和上一阶段显示的局部点云数据一致。

3.3.4 全局点云地图构建

接下来我们准备构建全局点云地图。为了节省计算机的计算资源，程序中`PCMapNode::NeedUpdateGlobalMap()`将会基于运动大小的判断，决定现在是否可以将局部点云数据融合到全局点云中。如果可以，程序将会调用`PCMapNode::UpdateGlobalPointCloudMap()`完成全局地图的更新，为了减少计算量，这个过程中局部点云将会被降采样、滤波，然后加入到全局点云中，最后全局点云会再进行一次降采样，得到最终的融合结果。请同学们根据第六章的相关点云例程边写程序，完成全局点云地图的创建。

如果程序正确，则可以控制机器人在环境中运动，构建环境的点云地图：



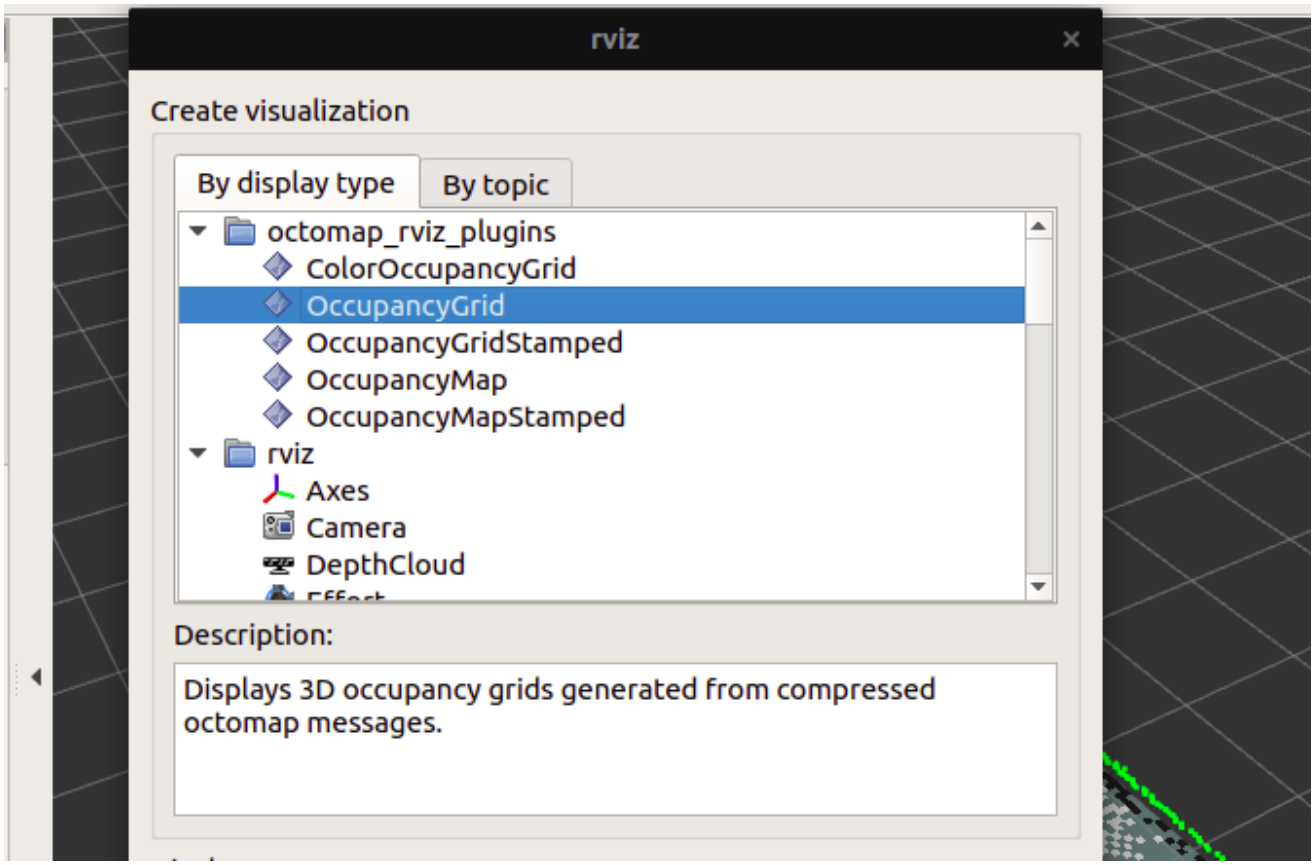
3.3.5 launch文件编写

在前面的实验过程中，同学们会发现每次关闭rviz时都会有对话框弹出，询问是否保存当前rviz配置；同时每次进行实验时都需要添加点云的可视化组件、调整rviz的视图，很不方便。

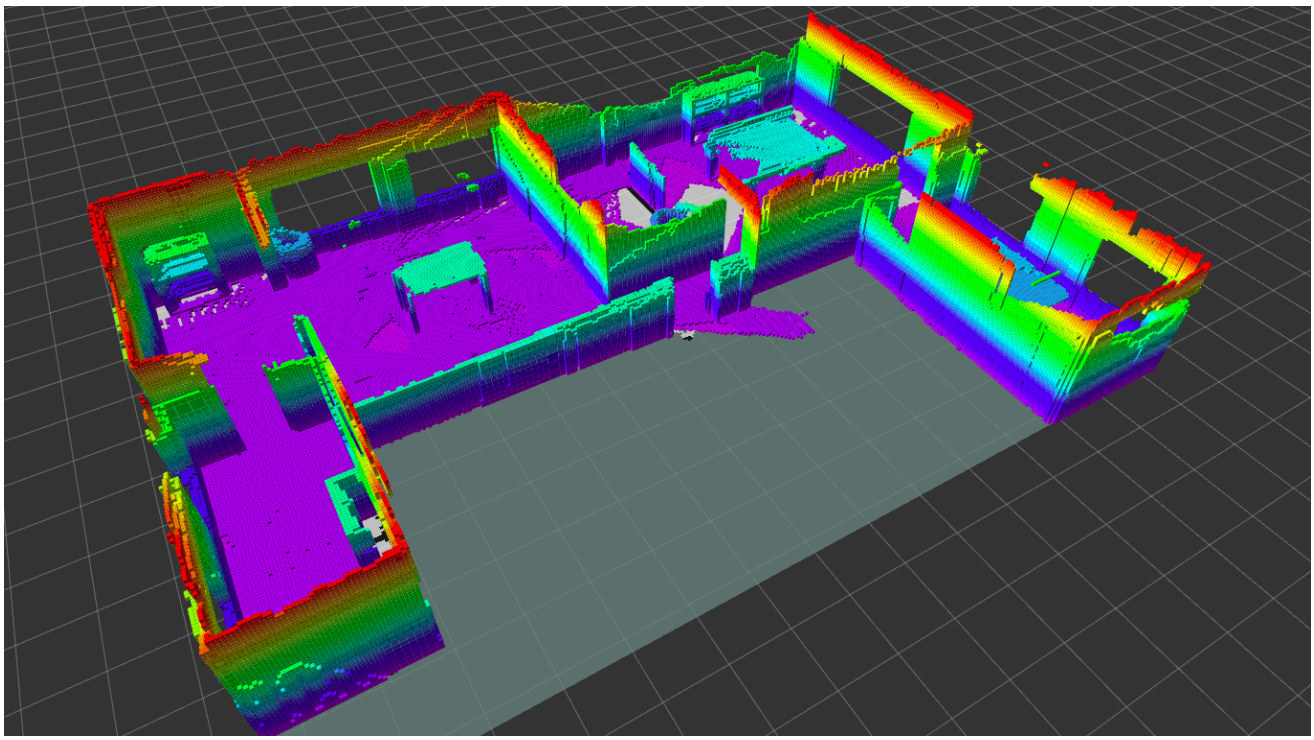
我们可以在设置好rviz的可视化状态后，将配置文件保存在`$(find octo_mapping)/rviz/point_cloud_mapping.rviz`文件中，然后编写launch文件启动时可以给rviz指定该配置文件。请参考`turtlebot3_navigation.launch`或`turtlebot3_slam.launch`文件，在现有的`$(find octo_mapping)/launch/point_cloud_map.launch`的基础上，补全rviz的启动部分，同时添加启动`point_cloud_map`节点的标签，并指定相关的参数服务器参数。

3.4 八叉树地图构建

获得了全局点云地图后，可以直接将其转换成为八叉树地图。点云地图只能表示某个地方有个点，但无法表示某个区域是否有物体被占据、机器人是否可以通过，而八叉树可以提供环境的占有信息，可供机器人在三维环境中的导航使用。同学们只需完善`$(find octo_mapping)/src/OctoMap/OctoMap.cc`中获取全局点云数据的部分，运行后先开启`point_cloud_map`节点，然后在rviz中添加八叉树地图的显示模块：



然后选择对应的话题`/octomap`，即可看到构建的八叉树地图：



> 八叉树地图的节点只有在接收到全局点云地图之后才会进行转换，如果启动后没有收到全局点云地图信息，是不会显示信息的。所以一开始需要先使机器人运动一段距离或者转动一段角度，触发`point_cloud_map`节点更新全局地图后，才能够在rviz中看到转换后的八叉树地图。

3.5 如果你还有时间……

目前进行的三维建图实验还比较粗糙。注意到目前生成的全局点云地图和八叉树地图均没有保存地图的机制，如果你还有时间，可以尝试分别写个节点，用于保存地图为pcd文件（点云地图）或bt文件（八叉树地图）。相应地，你还可以再编写节点，读取已经保存的地图文件，将地图数据发布在某个topic中，使用rviz查看等等。这些内容比较简单，因此本次实验不要求大家完成，若你在实验课上尚有时间，可以考虑做一做。

刘国庆

2020.06.07