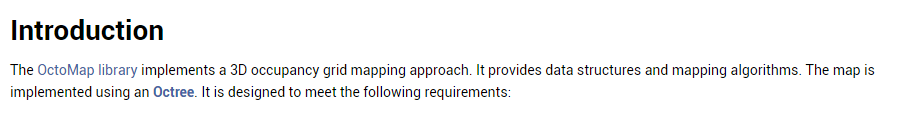
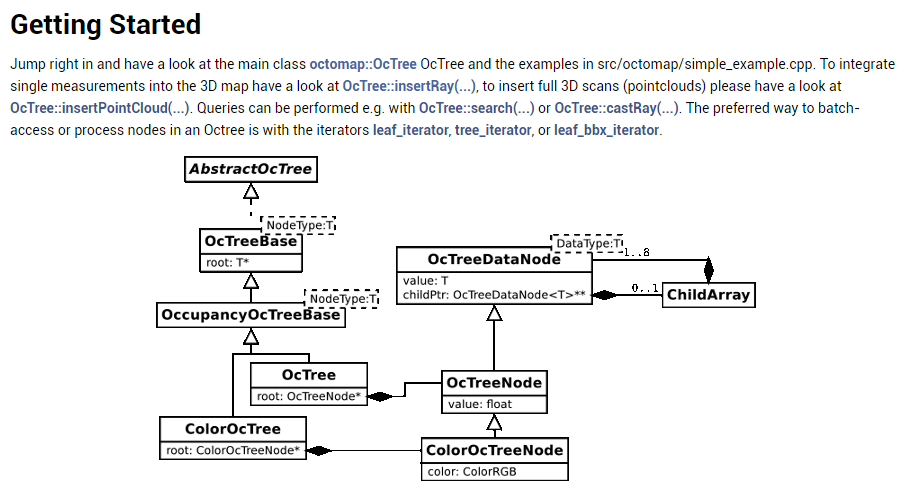
octomap库的一点总结

前些天突然想起高博《视觉SLAM十四讲》提到的octomap八叉树地图，在网上百度又只有高博写的那么一片博文，于是就想自己看看这个库，了解其几个基本的类和接口，以下是我这几天来的总结，如果有错误请及时指出，或者联系我，我也会及时改正。（[yuanliudongdong@163.com](mailto:yuanliudongdong@163.com)）

一，官方的 Introduction部分翻译(<http://octomap.github.io/octomap/doc/index.html>)



Octomap库实现了一种填充3D栅格的构图方式，并提供了相应的数据结构和构图算法。整个地图利用一个Octree的类来实现。



直接查看本库主要的类octomap::OcTree，以及其示例程序src/octomap/simple\_example.cpp。如果你想要将单组测量数据整合到3D地图中，请查看OcTree::insertRay(…)函数；如果你想要将整个3D扫描数据（点云）整合到地图中，请查看OcTree::insertPointCloud(…)函数。同时可以通过OcTree::search(...) 和OcTree::castRay(...)函数来进行地图空间占用的查询。推荐使用迭代器的方式对Octree的节点等进行查询，例如leaf\_iterator，tree\_iterator和leaf\_bbx\_iterator。

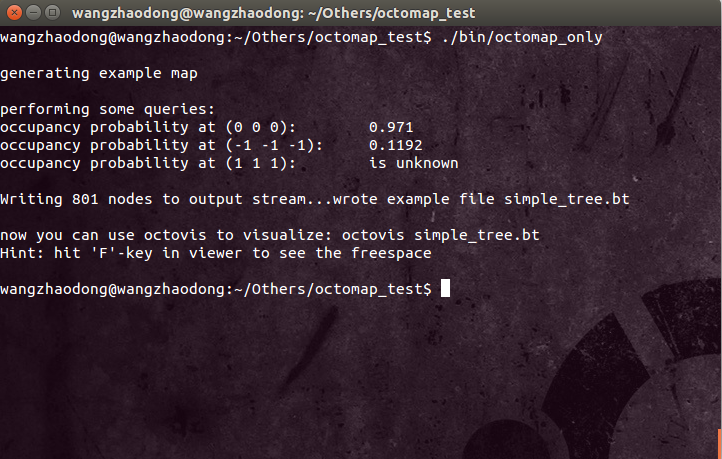
二，示例代码分析(Github源码地址：<https://github.com/OctoMap/octomap>)

在使用该库时，主要使用了基本的OcTree类和带有颜色信息的ColorOcTree类，所以只简单分析了这两个类的示例代码。

1，octomap/octomap/src/simple\_example.cpp

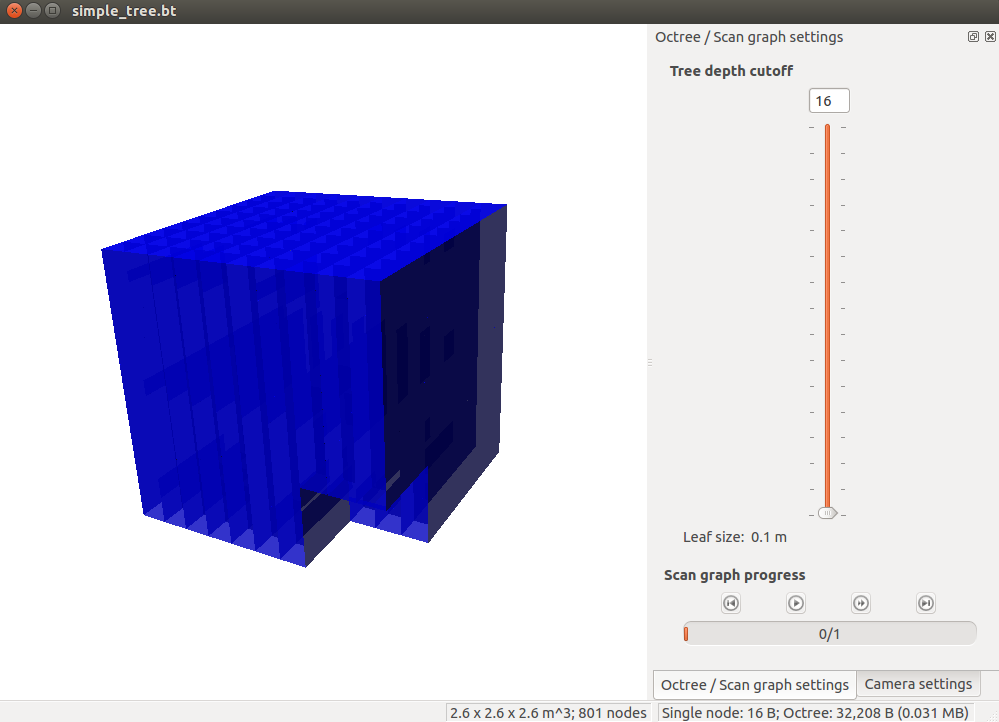
1. #include <octomap/octomap.h>
2. #include <octomap/OcTree.h>
3. using namespace std;
4. using namespace octomap;
5. //查询信息输出函数，输入为octomap命名空间下的3D点位置和用OcTree::search(...)得到的返回值
6. void print\_query\_info(point3d query, OcTreeNode\* node)
7. {
8. if (node != NULL)
9. {
10. cout << "occupancy probability at " << query << ":\t " << node->getOccupancy() << endl;
11. }
12. else
13. cout << "occupancy probability at " << query << ":\t is unknown" << endl;
14. }
15. // 主函数入口
16. int main(int argc, char\*\* argv)
17. {
18. cout << endl;
19. cout << "generating example map" << endl;
20. // 建立一个空的OcTree对象地图，分辨率为0.1
21. OcTree tree (0.1); // create empty tree with resolution 0.1
22. // insert some measurements of occupied cells
23. // 向地图中插入一些测量数据，这些数据点在地图中表现为已占用状态
24. for (int x=-20; x<20; x++)
25. {
26. for (int y=-20; y<20; y++)
27. {
28. for (int z=-20; z<20; z++)
29. {
30. //建立空间点对象，并向地图中更新节点
31. point3d endpoint ((float) x\*0.05f, (float) y\*0.05f, (float) z\*0.05f);
32. tree.updateNode(endpoint, true); // integrate 'occupied' measurement
33. }
34. }
35. }
36. // insert some measurements of free cells
37. // 向地图中插入一些测量数据，这些数据点在地图中表现为已未占用状态
38. for (int x=-30; x<30; x++)
39. {
40. for (int y=-30; y<30; y++)
41. {
42. for (int z=-30; z<30; z++)
43. {
44. //建立空间点对象，并向地图中更新节点
45. point3d endpoint ((float) x\*0.02f-1.0f, (float) y\*0.02f-1.0f, (float) z\*0.02f-1.0f);
46. tree.updateNode(endpoint, false); // integrate 'free' measurement
47. }
48. }
49. }
50. //上面已经完成了对地图的创建，之后的程序是对地图节点数据的访问
51. cout << endl;
52. cout << "performing some queries:" << endl;
53. // 节点（0,0,0）
54. point3d query (0., 0., 0.);
55. OcTreeNode\* result = tree.search (query);
56. print\_query\_info(query, result);
57. // 节点（-1,-1,-1）
58. query = point3d(-1.,-1.,-1.);
59. result = tree.search (query);
60. print\_query\_info(query, result);
61. // 节点（1,1,1）
62. query = point3d(1.,1.,1.);
63. result = tree.search (query);
64. print\_query\_info(query, result);
65. cout << endl;
66. // 将建好的地图保存
67. tree.writeBinary("simple\_tree.bt");
68. cout << "wrote example file simple\_tree.bt" << endl << endl;
69. cout << "now you can use octovis to visualize: octovis simple\_tree.bt" << endl;
70. cout << "Hint: hit 'F'-key in viewer to see the freespace" << endl << endl;
71. }

这是执行该代码后的运行结果：



图表 1

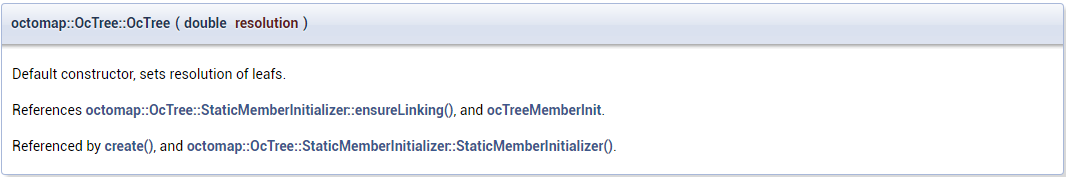
这是用octovis查看所生成的地图simple\_tree.bt：



图表 2

下面针对一些关键语句进行分析：

* OcTree tree(0.1);建立OcTree地图对象。这个函数的声明是这样的：

****

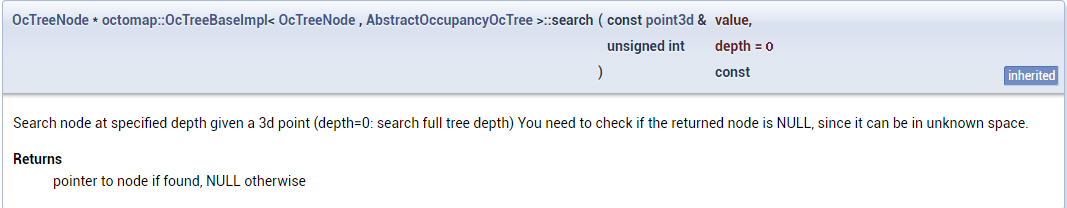
**r**esolution译为“分辨率”，即访问地图的最小单位；

* point3d endpoint( (float)x\*0.02f-1.0f, (float)y\*0.02f-1.0f, (float)z\*0.02f-1.0f );

tree.updateNode( endpoint, false );

建立一个三维空间点，并将其添加到地图当中；point3d构造函数的三个参数分别是三维点的x/y/z坐标；使用.updateNode()函数将空间点信息更新到地图当中，第二个参数表示该空间点对应的节点未被占用(false)

* OcTreeNode\* result = tree.search (query);节点信息查询函数；函数声明是这样的



传入参数为一个三维点对象，如果在这个地图tree中，这个三维点对应的位置有节点（不管占用与否），那么将返回该位置的节点指针，否则将返回一个空指针；

* cout << "occupancy probability at " << query << ":\t " << node->getOccupancy() << endl；如果查询到有该位置上节点的信息，则使用getOccupancy()函数输出该节点占用情况，那为什么这个Occupancy是个小数呢？这是因为Octomap在描述一个栅格是否被占用时，并不是单一的只描述为占用和被占用，而是用一个概率（Occupancy probability）来描述它，即这个栅格被占用的概率是多少，通过这个概率来确定这个栅格被占用的可能性。

2，Occupancy probability

关于这个占用概率值，高博在一篇博客中做出了和通俗易懂的解释，我就不再复制粘贴了，下面附上连接(<https://www.cnblogs.com/gaoxiang12/p/5041142.html>)。

我要说的是，对于一个最小单位的栅格，如何判断updateNode()函数对其做出了贡献或者说更新。就比如说

1. point3d endpoint( 4.05f, 4.05f, 4.05f );
2. tree.updateNode( endpoint, true );

这两行代码，由于分辨率是0.1，并不能精确到0.01，所以要把这个（4.05,4.05,4.05）归到（4.0,4.0,4.0）这个节点呢，还是（4.1,4.1,4.1）这个节点或者其他节点呢？经过我的多次测试，应当归到（4.0,4.0,4.0）节点当中，也就是说，对于节点（4.0,4.0,4.0）来说，凡是同时满足x=[4.0,4.1)，y=[4.0,4.1)，z=[4.0,4.1)的point，如果使用updateNode()函数将这个point更新到地图当中，那么必然会影响到节点（4.0,4.0,4.0）的Occupancy probability。

总结就是，对于一个点point(x,y,z)，使用updateNode()函数将其更新到地图当中，那么Occupancy probability受到影响的节点将是



即小于point坐标值的最大节点。同时当我们用search ()函数对point进行查询时，返回的信息也将是小于point坐标值的最大节点信息。

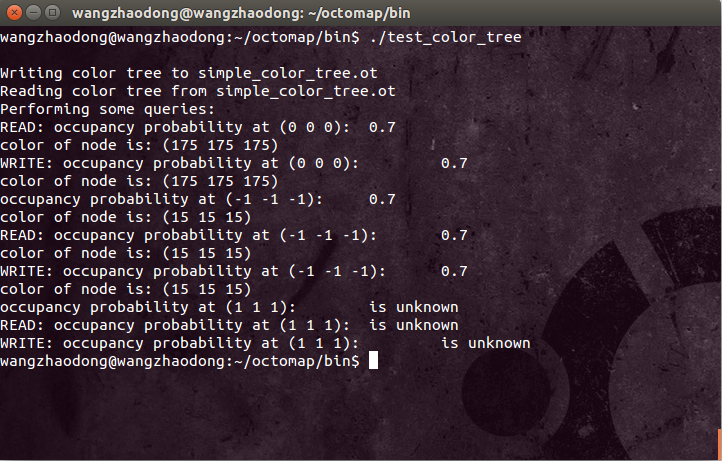
另外，在对某个节点进行不同次数的更新之后，发现Occupancy probability最大值为0.971，最小值为0.1192，这也验证了高博那篇博文中提到的最大值和最小值限制。

3，octomap/src/testing/test\_color\_tree.cpp

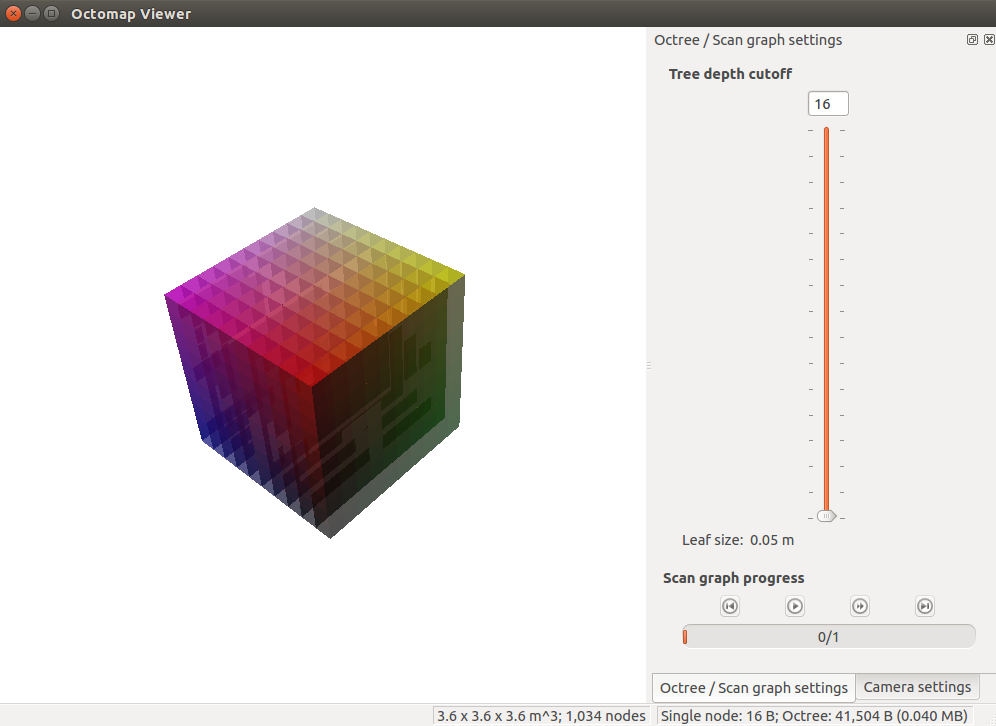
带有色彩的八叉树地图ColorOcTree与基本的OcTree类似，需要知道如何向节点当中添加颜色信息就好了。下面是test\_color\_tree.cpp的部分代码：

1. int main(int argc, char\*\* argv)
2. {
3. //分辨率
4. double res = 0.05; // create empty tree with resolution 0.05 (different from default 0.1 for test)
5. //建立彩色地图对象
6. ColorOcTree tree (res);
7. // insert some measurements of occupied cells
8. for (int x=-20; x<20; x++)
9. {
10. for (int y=-20; y<20; y++)
11. {
12. for (int z=-20; z<20; z++)
13. {
14. point3d endpoint ((float) x\*0.05f+0.01f, (float) y\*0.05f+0.01f, (float) z\*0.05f+0.01f);
15. ColorOcTreeNode\* n = tree.updateNode(endpoint, true);
16. //设置节点颜色信息的函数，每个地图节点差五个像素大小，渐变
17. n->setColor(z\*5+100,x\*5+100,y\*5+100);
18. }
19. }
20. }
21. // insert some measurements of free cells
22. for (int x=-30; x<30; x++)
23. {
24. for (int y=-30; y<30; y++)
25. {
26. for (int z=-30; z<30; z++)
27. {
28. point3d endpoint ((float) x\*0.02f+2.0f, (float) y\*0.02f+2.0f, (float) z\*0.02f+2.0f);
29. ColorOcTreeNode\* n = tree.updateNode(endpoint, false);
30. //不被占用的节点设置为黄色
31. n->setColor(255,255,0); // set color to yellow
32. }
33. }
34. }
35. // set inner node colors
36. tree.updateInnerOccupancy();
37. }

这是执行该代码后的运行结果：



这是用octovis查看所生成的地图simple\_color\_tree.ot：

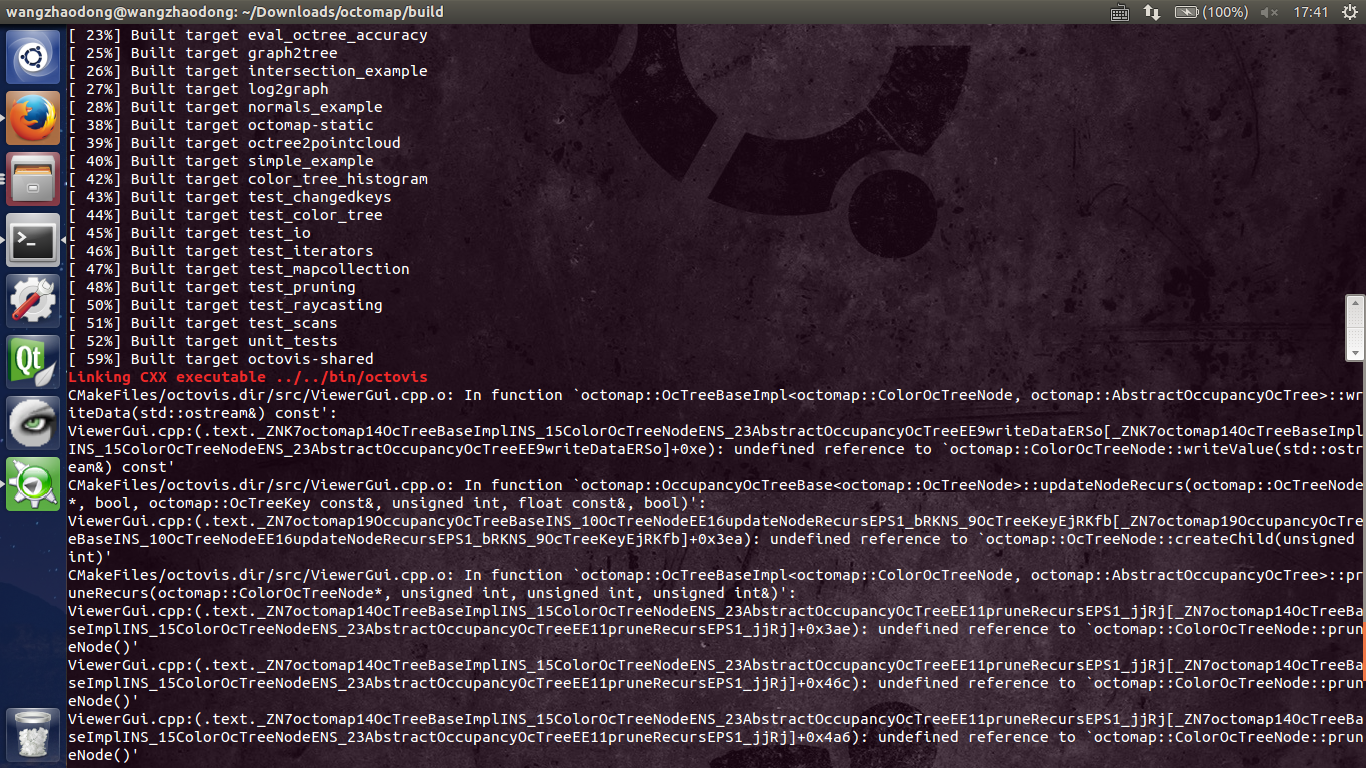


三，安装和编译octomap库时的一个小问题

第一次编译安装octomap库时，直接按照网上的教程进行的操作：

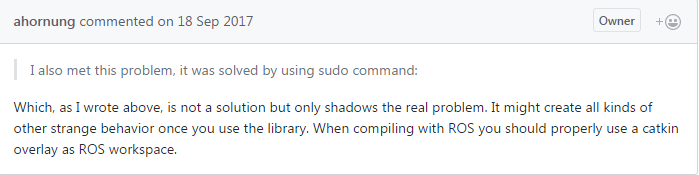
1. git clone https://github.com/OctoMap/octomap
2. cd octomap
3. mkdir build
4. cd build
5. cmake ..
6. make
7. sudo make install

可是在执行make指令时却出现了undefined reference to的错误

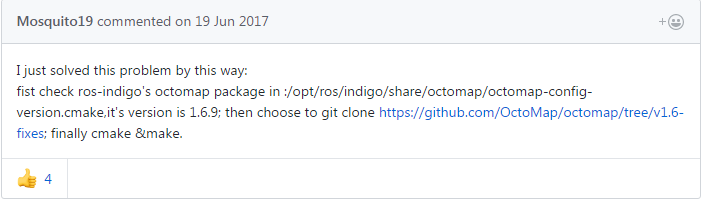


在网上百度了好久，找到了关于这个问题的帖子（<https://github.com/OctoMap/octomap/issues/171>）

1. 我首先用的方法是 sudo make，确实，编译通过，也安装成功了，但是我在编译自己写的octomap\_test程序时，仍然会出现类似的undefined reference to的错误，于是在编译时还是需要sudo；
2. 后来我觉得有一个对方法1的评价特别对：



于是我尝试了第二种做法：



打开查询了这个/opt/ros/indigo/share/octpmap/octomap-config-version文件，我的版本是1.6.9，在重新下载了octomap源码之后，校对了版本，具体指令如下：

1. git clone https://github.com/OctoMap/octomap
2. cd octomap
3. git tag //列出所有版本，查看是否有自己的版本
4. git checkout v1.6.9 //校验为自己的版本
5. mkdir build
6. cd build
7. cmake ..
8. make
9. sudo make install

感觉第二种做法才是真正的解决方案。