2018 Fall Intelligent Robot Lab Undergraduate Special Project

Visual Search with ORB Slam2 & Semantic Segmentation

Author: Szu-Yu Mo
Department: Electronic Engineering, National Taiwan University
Student ID: B04901164
Advisor: Professor Li-Chen Fu
Senior: Shao-Hung Chan
Project date: September 2018 - January 2019
DEMO video (without localization): https://goo.gl/pcxhpR
DEMO video (with localization): https://goo.gl/DfUDRe

Content

- **Abstract**
- **Motivation**
- **Paper Survey**
- **Learning Robot Operating System (ROS)**
- **Environment setup**
- **System Structure**
- **Algorithms**
- **Achievement**
- **Future Work**
- **Reference**

Abstract

In this semester, I successfully built a real-time visual search system based on the octomap constructed by ORB slam2 and semantic segmentation. The whole system runs on Robot Operating System (ROS), using the Asus Xtion camera as the only sensor. Adapting the RGB-D version of ORB slam2, the system is able to estimate camera poses accurately. Using a PSPNet model first trained on ADE20K dataset and then fine-tuned on SUNRGBD data set, I was able to precisely predict the object classification for each pixel in the frame. Combining those two results in a semantic octomap, which I later made use of to develop the localization algorithm. In the end, I was able to locate the target object in the world coordinate system. For the next semester, I plan to append the navigation function to the current system and thus producing a complete visual search engine.

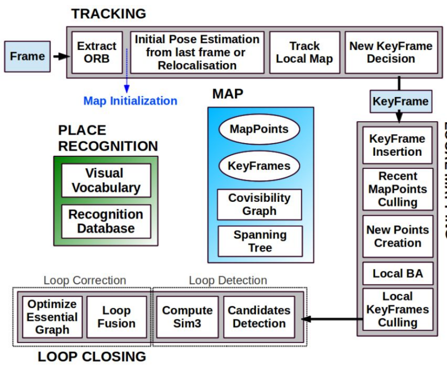
Motivation

During the last summer internship, I had built a feature-based monocular visual odometry system. This experience has sparked my interest in simultaneous localization and mapping (slam) algorithm. Since this lab owns several robots, I was thinking of applying the slam algorithm on one of them and make some improvements or applications. Asking Shao-Hung for advice, I decided to try to combine semantic segmentation with slam. Upon finding a Github repo about semantic mapping, I started to focus my work on the localization algorithm. Later on, I thought that a robot should not only know where it is, but also figure out the way it can go to a specific destination. That is the moment I picture my system having localization and navigation functions.

Paper Survey

I. ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras

This paper is written by Raul Mur-Artal, Juan D. Tardos, J. M. M. Montiel. They conducted an efficient slam system based on ORB features, currently available for monocular, stereo, RGB-D, and AR cameras. It computes the camera trajectory and a sparse 3D reconstruction in real time, with three threads running simultaneously: tracking, local mapping, and loop closing. The system structure can be seen in fig. 1.



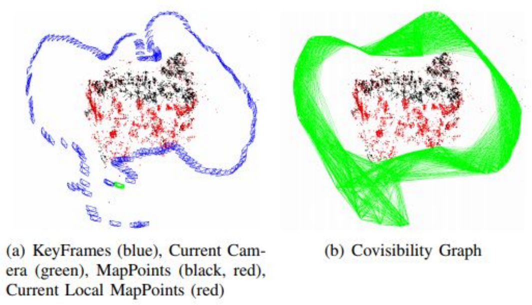
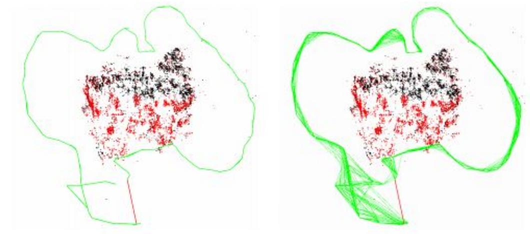


Fig. 1 ORB slam2 system structure   Fig. 2 KeyFrames, MapPoints, and Covisibility Graph

First of all, in the “tracking” thread, camera poses are estimated. ORB features are extracted for each frame to localize the camera, known as the camera pose initialization. Next, camera poses are optimized over the Covisibility Graph (fig. 2). This thread also decides the insertion of a new keyframe, since memory consumption is a big issue for a feature-based slam. Secondly, the “local mapping” thread would process newly inserted keyframes and perform local bundle adjustments (BA). In addition, it would triangulate new points with unmatched ORB descriptor in the new keyframe as well as get rid of redundant keyframes and outliers. Finally, the third thread “loop closing” search for loops with every new keyframe by DBoW2 (bags of words). If a loop is detected, drifts will be computed and both sides of the loop will be fused together. A pose-graph optimization is also conducted over the Essential Graph (fig. 3). In conclusion, this system is stable a wide variety of environments and can close large loops and perform global relocalization in real-time.



(c) Spanning Tree (green) and Loop Closure (red)   (d) Essential Graph

Fig. 3 Spanning Tree and Essential Graph

II. Bags of Binary Words for Fast Place Recognition in Image Sequences

This is the system used for loop detection and relocalization in ORB slam2. It utilizes bags of binary words for place recognition, where the words are BRIEF descriptors. The classification is done by kmeans++ clustering. The vocabulary tree is shown in fig. 4.

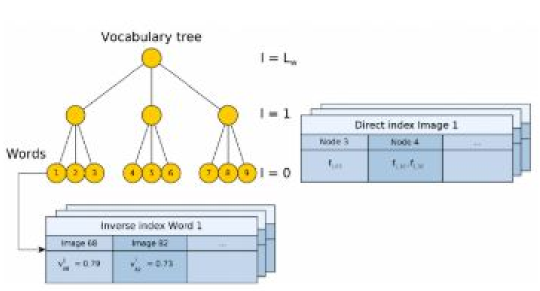


Fig. 4 Vocabulary tree of DBoW2

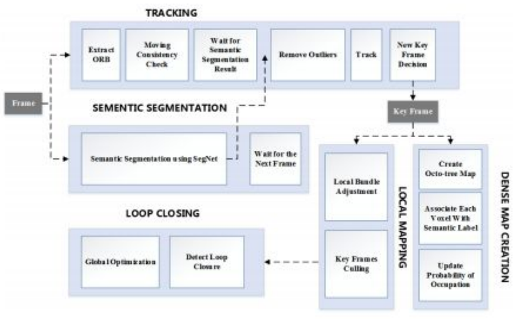


Fig. 5 DS-SLAM system structure

III. DS-SLAM: A Semantic Visual SLAM towards Dynamic Environments

The system is based on ORB slam2. However, the authors utilize SegNet to perform semantic segmentations on each frame and combined the results with poses estimated by ORB slam2 into a dense semantic octomap. DS-SLAM also combines semantic segmentation network with moving consistency check method to reduce the impact of dynamic objects, and thus the localization accuracy is highly improved in dynamic environments. The DS-SLAM system structure is shown in Fig. 5.

IV. SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation

Semantic segmentation is a combination of a combination of classification and object detection, giving us the understanding of an image at the pixel level. The differences in classification, object detection, and semantic segmentation are visualized in fig. 6. The general structure of the net consists of an encoder network followed by a decoder network, and so does SegNet. The encoder is usually a pre-trained classification network like VGG/ResNet, while the decoder is to semantically project the discriminative features (lower resolution) learned by the encoder onto the pixel space (higher resolution) to get a dense classification.

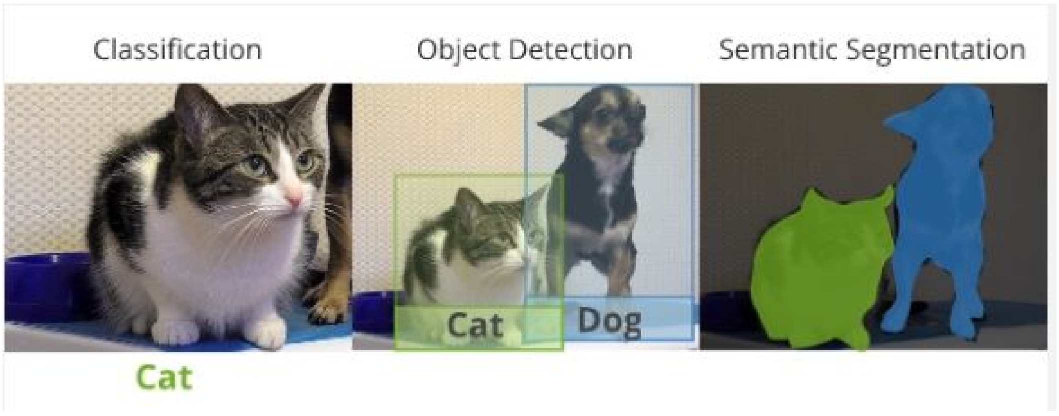


Fig. 6 Difference among classification, object detection, and semantic segmentation

Existing approaches include FCN, SegNet, Dilated Convolutions, DeepLab, RefineNet, PSPNet, and Large Kernel Matters. FCN architecture can be seen in fig. 7, and SegNet architecture is shown in fig. 8. Instead of copying the encoder features as in FCN, SegNet copied indices from the maxpooling layers, and thus became more memory efficient than FCN.

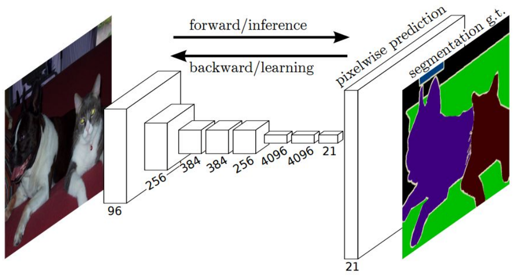


Fig. 7 FCN architecture

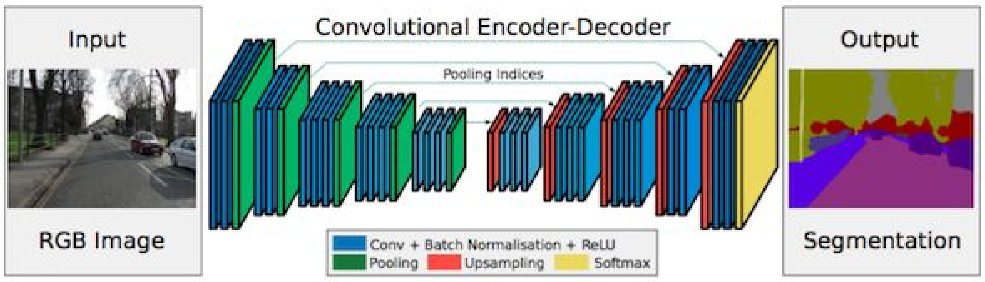


Fig. 8 SegNet architecture

V. Real-time Semantic Slam in ROS with a Handheld RGB-D Camera

This is actually a Github repository Shao-Hung found about semantic slam, which is exactly what I want to do. As a result, my system is based on this research. The system structure is shown in fig. 9. The author constructed a voxel-based 3D semantic map with a handheld RGB-D camera in real time. He combined the state-of-the-art feature based ORB-SLAM, a Convolutional Neural Network (CNN) for semantic segmentation – PSPNet and an efficient voxel-based 3D map representation – Octomap to build a working system. Detailed elaboration would be mentioned in my system structure.

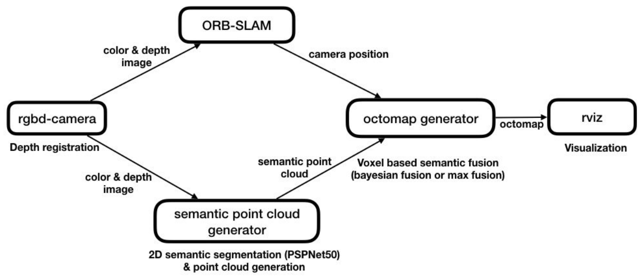
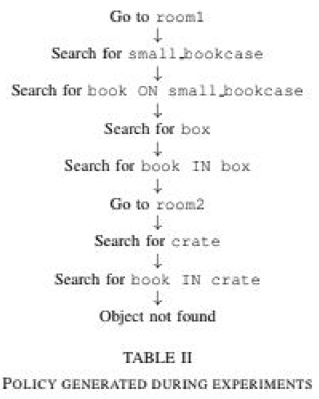


Fig. 9 Semantic Slam system



Go to `room1`
↓
Search for `small_bookcase`
↓
Search for `book ON small_bookcase`
↓
Search for `box`
↓
Search for `book IN box`
↓
Go to `room2`
↓
Search for `crate`
↓
Search for `book IN crate`
↓
Object not found

TABLE II
POLICY GENERATED DURING EXPERIMENTS

Fig. 10 Search strategies

VI. Search in the Real World: Active Visual Object Search Based on Spatial Relations

This paper helped me with the design of octomap localization. The visual search here focused on spatial relations among objects. Take looking for a book for example. It is highly possible to be in a study room, so we should look in it first instead of other rooms. Inside the room, it is highly possible that it is on a table or in a bookshelf, so we can first look for those rather large objects. The search strategies can be observed through fig. 10.

Learning Robot Operating System (ROS)

Since my system depends on data of cameras and several data communications among processes, it would be extremely convenient to use ROS as a platform to transfer information. Lacking experience of it, I went through the official beginners' tutorials on ROS Wiki. The following is a summary about those lessons.

I. Installing and configuring my ROS environment
II. Navagating the ROS filesystem
III. Creating and building a ROS package
IV. Understanding ROS nodes, topics, and sevices
V. Using roslaunch
VI. Creating a ROS msg and srv
VII. Writing a simple publisher and subscriber
VIII. Writing a simple service and client

Environment Setup

The system requires an Asus Xtion camera and a pc/laptop with GPU running on Ubuntu 16.04. As for the software packages, there are several dependencies necessary. They will be illustrated in the following paragraphs.

I. Installing ROS Kinetic

Kinetic is the ROS version compatible with Ubuntu 16.04.

```
$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" >
/etc/apt/sources.list.d/ros-latest.list'
$ sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key
421C365BD9FF1F717815A3895523BAEEB01FA116
$ sudo apt-get update
$ sudo apt-get install ros-kinetic-desktop-full
$ sudo rosdep init
$ rosdep update
$ echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc
$ source ~/.bashrc
$ sudo apt install python-rosinstall python-rosinstall-generator python-wstool
build-essential
```

II. Installing Openni2 Driver for Xtion Camera

```
$ sudo apt-get install ros-kinetic-rgbd-launch
$ sudo apt-get install ros-kinetic-openni2-camera
$ sudo apt-get install ros-kinetic-openni2-launch
```

III. Installing Semantic_slam

```
$ roscd
$ git clone https://github.com/floatlazer/semantic_slam.git
$ pip3 install torch torchvision  #pytorch 0.4.0
```

Install Pangolin following instructions at https://github.com/stevenlovegrove/Pangolin
Install OpenCV required version at leas 2.4.3
Install Eigen3 following instructions at http://eigen.tuxfamily.org

```
$ cd semantic_slam/ORB_SLAM2
$ ./build.sh
$ rosdep install semantic_slam
$ cd ~/catkin_ws
$ catkin_make
```

IV. Compiling and Running the System

```
$ catkin_create_pkg octomap_localization roscpp rospy std_msgs
```

write my code in the “octomap_localization” directory

```
$ cd ~/catkin_ws
$ catkin_make
```
To run the five threads simultaneously, we have to run 4 commands in 4 terminals:
```
$ roslaunch semantic_slam camera.launch
$ roslaunch semamtic_slam slam.launch
$ roslaunch semantics_slam semantic_mapping.launch
$ rosrun octomap_localization octomap_localization
```

System Structure

The system structure is designed as fig. 11. Openni2 is used as the camera driver as usual, publishing two important image topics, /camera/rgb/image_raw and /camera/depth_registered/image_raw. The ORB slam2 thread would estimate camera poses of each frame in real-time and publish the transformation to the octomap generator thread. In the mean time, the semantic cloud thread takes in the two topics of camera and predict the semantic segmentation of each frame, generating a semantic point cloud and publishing it to the octomap generator thread. Therefore, the octomap generator now gets both the camera poses and the semantic information, which can lead to the insertion of the octomap. The octree of the octomap will then publish to the octomap localization thread. The localization would find out the target object coordinate in world frame according to the octree. The afornenetioned threads are all done this semester. Next semester, I wish to extract 2D grid mapping from the octomap and come up with the navigation algorithm.

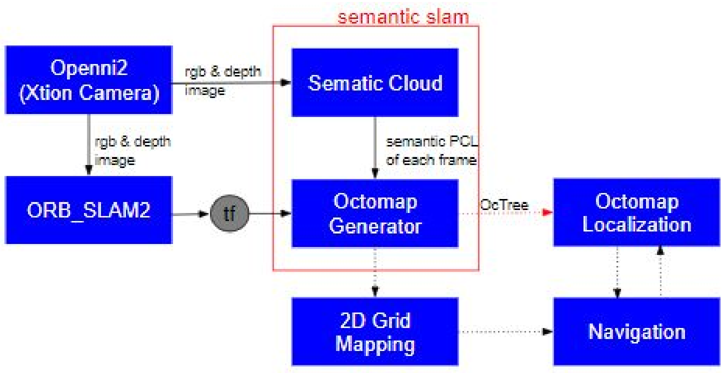


Fig. 11 System Structure

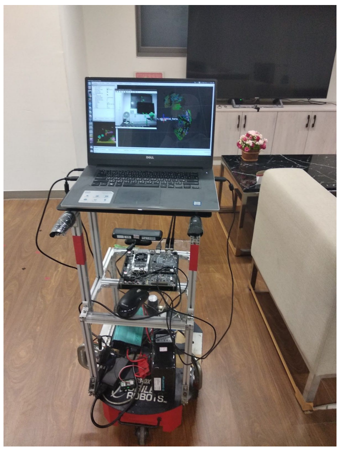


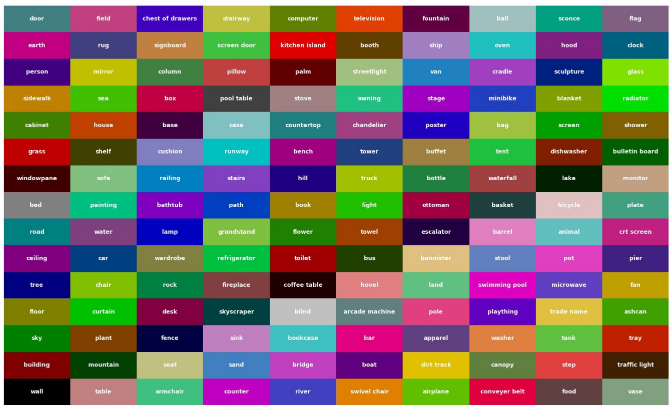
Fig. 12 Devices on Pioneer

When running the system, the Xtion camera has to be connected with a notebook. Since the computation requires much power, the power supply has to be connected with the notebook as well. However, in order to move the camera around for constructing a precise map, the power supply should be protable. Thus, the devices are placed on a mobile robot Pioneer with a 21V-battery and a DC-AC converter (fig. 12). Instead of the typical method of handhelding the camera, I push around the Pioneer to let Xtion camera sees every couner of the room.

Algorithms

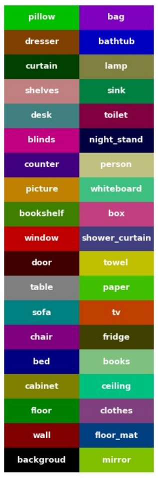
I. Generating Semantic Point Cloud (PCL)

The first step is to perform semantic segmentation on each frame. The CNN PSPNet model used is first trained on AED20K dataset that results in 150 different classes (fig. 13), and then fine-tuned on SUNRGBD dataset that results in 32 different classes (fig. 14). The rgb values of the SUNRGBD-trained model labels are combinations of multiples of 64. For example, the rgb value of a table is (128, 128, 128) and that of a window is (192, 0, 0). Three sample results of the semantic segmentation are shown in fig. 15~17, which respectively represents window & sofa, TV & cabinet, and pillow & bed.



door	field	chest of drawers	stairway	computer	television	fountain	ball	sconce	flag
earth	rug	signboard	screen door	kitchen island	booth	ship	oven	hood	clock
person	mirror	column	pillow	palm	streetlight	van	cradle	sculpture	glass
sidewalk	sea	box	pool table	stove	awning	stage	minibike	blanket	radiator
cabinet	house	base	case	countertop	chandelier	poster	bag	screen	shower
grass	shelf	cushion	runway	bench	tower	buffet	tent	dishwasher	bulletin board
windowpane	sofa	railing	stairs	hill	truck	bottle	waterfall	lake	monitor
bed	painting	bathtub	path	book	light	ottoman	basket	bicycle	plate
road	water	lamp	grandstand	flower	towel	escalator	barrel	animal	crt screen
ceiling	car	wardrobe	refrigerator	toilet	bus	bannister	stool	pot	pier
tree	chair	rock	fireplace	coffee table	hovel	land	swimming pool	microwave	fan
floor	curtain	desk	skyscraper	blind	arcade machine	pole	plaything	trade name	ashcan
sky	plant	fence	sink	bookcase	bar	apparel	washer	tank	tray
building	mountain	seat	sand	bridge	boat	dirt track	canopy	step	traffic light
wall	table	armchair	counter	river	swivel chair	airplane	conveyer belt	food	vase

Fig. 13 ADE20K labels



pillow	bag
dresser	bathtub
curtain	lamp
shelves	sink
desk	toilet
blinds	night_stand
counter	person
picture	whiteboard
bookshelf	box
window	shower_curtain
door	towel
table	paper
sofa	tv
chair	fridge
bed	books
cabinet	ceiling
floor	clothes
wall	floor_mat
backgroud	mirror

Fig. 14 SUNRGBD labels

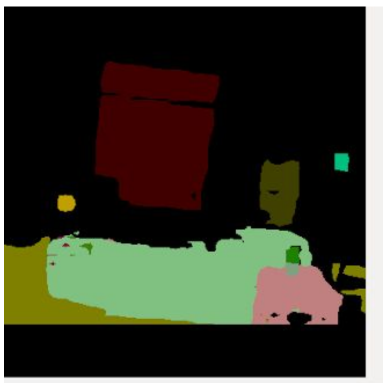


Fig. 15 wondow & sofa

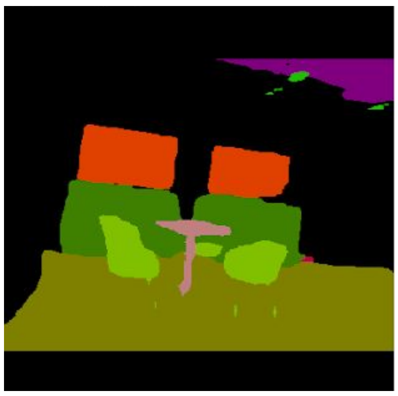


Fig. 16 TV & cabinet

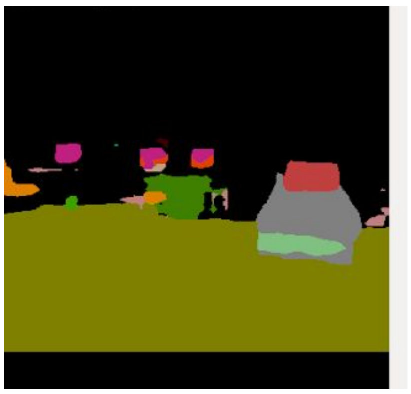


Fig. 17 pillow & bed

The semantic result of each frame would combine with the ORB feature points extracted by ORB slam2 and generate a point cloud with semantic color registered. Besides the semantic color that the pixel belongs to, the confidence of the prediction is also saved with the semantic point cloud for further application. Overall, a point cloud memorizes the frame id, the size of the cloud, and the fields of each pixel, where the filed includes position values (xyz), rgb values, semantic color, and cofidence.

II. Merging Semantic PCL into an Octomap

To talk about the construction of an octomap, the original octomap data structure (fig. 18) has to be illustrated in advance. The arrows represent the class inheritence relations. In the Github code I cloned down, the author defined another class SemanticsOcTree inheritting OccupancyOcTreeBase and a class SemanticsOcTreeNode inheritting ColorOcTreeNode. The newly defined classes own the member variables and functions that the original classes do, but with more data. They contain the semantic color and the confidence as well as some functions to support to insertion, computation, and access of those information.

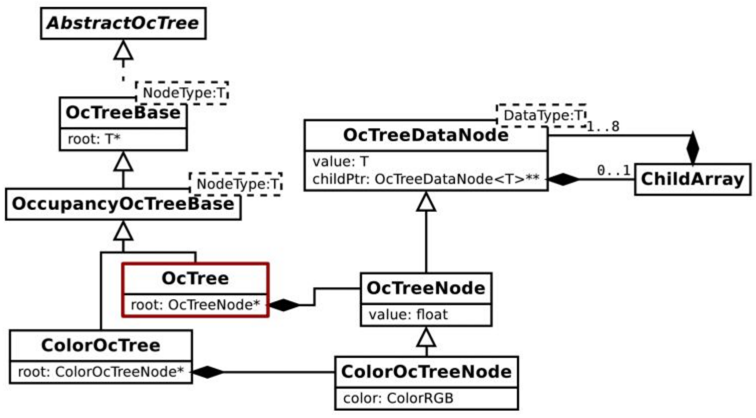


Fig. 18 Octomap data structure

When a semantic point cloud and the current camera pose are accessed by the octomap generator, the points will first go through a voxel filter to down sample the points. Then we insert these points into the octomap. After that we perform raycasting to clear free space in a certain range. Next, we update inner nodes of octomap, i.e. voxels with lower resolution. Finally we serialize the updated octomap for visualization.

III. Octomap Localization

Before explaining the localization algorithm, we should first pay attention to the data structure (fig. 19) of an octree. An octree recursively cut each cube into 8 voxels until it reaches its max depth. The size of the smallest voxel is defined by the resolution of the octree. This is an efficient way to store a map since it saves much memory compared to a point cloud.

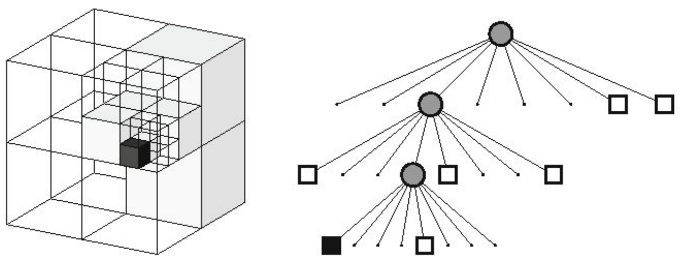


Fig. 19 Octree data structure

A more efficient aspect of an octree is that it would prune nodes that are not useful. For example, when a node is occupied by 8 children with the same semantic color, the children can be pruned. The children can also be pruned if their parent is unoccupied. We will later utilize this pruning algorithm as the spirit of the localization.

While the aforementioned threads are creating the up-to-date octomap of the environment, octomap localization subscribes to the topic /octomap_full and search for the coordinate of some particular target objects. Reading the message from the topic, we can convert it to a ColorOcTree by msgToMap method defined in octomap_msgs library. Since it does not support the convertion to a SemnaticsOcTree, we can only get the node color representing the rgb value of the semantic color. Mapping the rgb value to what object it stands for, we can start the searching.

Typically, when a node contains children with different semantic colors, it means that coordinate is the egde of an object. On the contrary, if a parent node is occupied with children having the same semantic color, it is more possible that the coordinate is in the center of the target object. Take the scenario in fig. 20 for example, where a color represent an object. On the left is the octree data structure, and on the right is the actual object distribution in space. In the second layer, the parent node on the left and the right have children with different semantic colors, and thus the coordinates are the edges of the red object. Nevertheless, the middle parent node contains 8 children all with the red color, so the children can be pruned. Therefore, the branch would be the shortest of all, and in the mean time, representing the coordinate of the center of the red object. Thus, the algorithm that I came up with this localization problem is to find the leaf node with the shortest depth having the target semantic color (fig. 21).

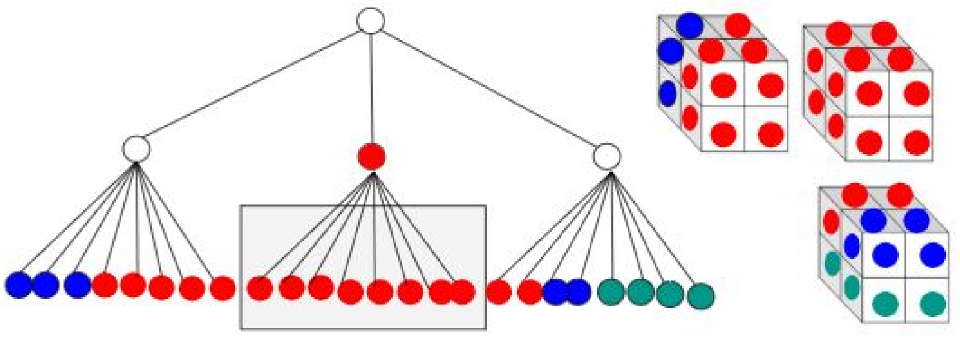


Fig. 20 A scenario of an octree

Greedy Search: Find the leaf with the least depth having the target semantic color.

Fig. 21 Octomap localzation algorithm

Achievement

In this project, I successfully implemented octomap localization on the semantic slam system, which is a visual serach application of slam. A demo video without the localization thread is at https://goo.gl/pcxhpR, and a demo video with the localization thread is at https://goo.gl/DfUDRe. An octomap localization algorithm is designed originally by me. It is proved the algorithm effective through testing.

Future Work

I. Search based on spatial relations

The current search algorithm simply goes through every leaf node to find the lead with minimum depth, which is not an efficient way. In the future, I shoud develop a search algorithm utilizing spatial relations among objects. Take fig. 22 for example. If we want to find the location of a book, we can first look in the room with the highest possibility that a book is in it, which is a studyroom. Afterwards, we might look on a table rather than a chair. In conclusion, we find bigger objects related to the target object and then smaller objects, instead of directly find the rather small target object.

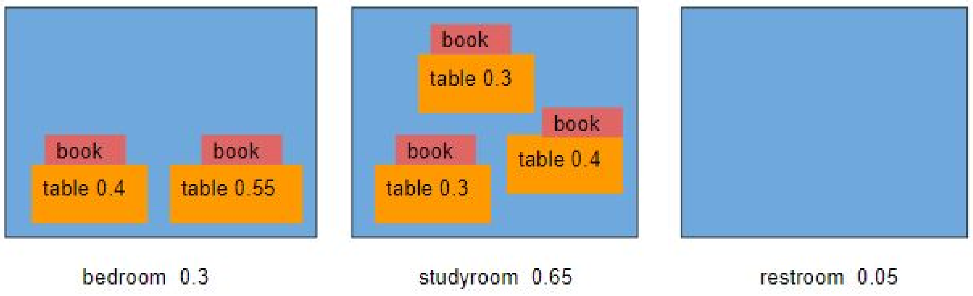


Fig. 22 A Scenario of spatial relations

II. 2D grid mapping and navigation

In order to establish a complete localization and navigation system, the navigation function has to be added. The plan is to transform the 3D octomap into a 2D grid map for path planning. The localization result will be sent to the navigation thread too. Applying the system on the mobile robot Pioneer, we look forward to making Pioneer walk to the destination according to the path planned by navigation.

Reference

[1] V. Badrinarayanan, A. Kendall, and R. Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *arXiv preprint arXiv:1511*.00561, 2015.
[2] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard. Octomap: An efficient probabilistic 3d mapping framework based on octrees. *Autonomous Robots*, 34(3):189–206, 2013.
[3] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015.
[4] R. Mur-Artal and J. D. Tardós. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, 2017.
[5] D. Galvez-L´ opez and J. D. Tard´ os. Bags of binary words for fast place recognition in image sequences. *IEEE Trans. Robot*., vol. 28, no. 5, pp.1188–1197, 2012.
[6] Chao Yu, Zuxin Liu, Xinjun Liu, Fugui Xie, Yi Yang, Qi Wei, Fei Qiao "DS-SLAM: A Semantic Visual SLAM towards Dynamic Environments." Published in the Proceedings of the 2018 *IEEE/RSJ International Conference on Intelligent Robots and Systems* (IROS 2018).
[7] A. Aydemir, K. Sjoo, J. Folkesson, A. Pronobis, " Search in the real world: Active visual object search based on spatial relations" , in *Proc. IEEE ICRA*, 2011, pp. 2818-2824
[8] Xuan Zhang, Semantic SLAM, (2018), semantic_slam, https://github.com/floatlazer/semantic_slam