2019 Spring Intelligent Robot Lab Undergraduate Special Project

Visual Search and Navigation on Semantic SLAM

Author: Szu-Yu Mo
Department: Electronic Engineering, National Taiwan University
Student ID: B04901164
Advisor: Professor Li-Chen Fu
Senior: Shao-Hung Chan
Project date: Feburary 2019 - June 2019
DEMO video (SLAM): https://reurl.cc/xMM0z
DEMO video (map visualization): https://reurl.cc/jZZqm
DEMO video (navigation): https://reurl.cc/lrrRd

Content

Abstract

In this semester, I integrated 2D grid mapping as well as navigation function on to my system done last semester, which is 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 dataset, a semantic 3D octomap was constructed. The octomap was projected into 2D grid map for navigation. The navigation function was appended to the system. The target coordinate was generated by my self-defined localization algorithm, and the path planner is conducted by move_base. Although the system requires improvement for accuracy, it is now a complete visual search engine with SLAM and navigation.

Motivation

In last semester, I constructed a visual search system with SLAM and object localization functions. 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 navigation functions. Thus, I tried to implement the navigation methods into the current system.

Work From Last Semester

I. System Structure (Partial of the Final System)

The partial system structure is designed as fig. 1. As you can see, it does not include any map projection or navigation. 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 meantime, 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.

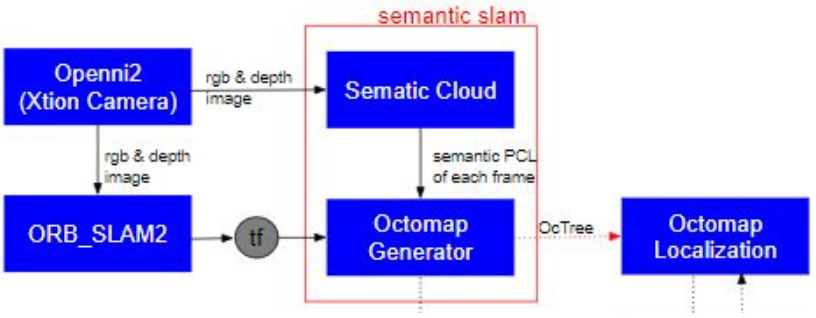

Fig. 1Partial System Structure

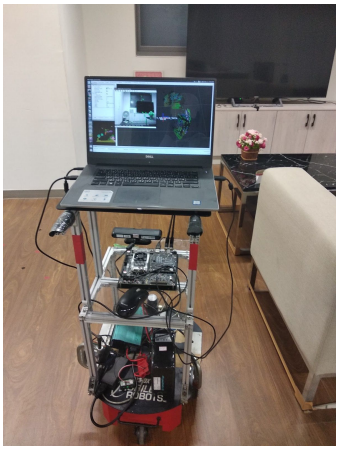

Fig. 2 Devices on Pioneer

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 portable. Thus, the devices are placed on a mobile robot Pioneer with a 21V-battery and a DC-AC converter (fig. 2). Instead of the typical method of handheld camera, I push around the Pioneer to let Xtion camera sees every corner of the room.

II. Algorithms

A. 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. 3), and then fine-tuned on SUNRGBD dataset that results in 32 different classes (fig. 4). 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. 5~7, which respectively represents window & sofa, TV & cabinet, and pillow & bed.

Fig. 3 ADE20K labels

Fig. 4 SUNRGBD labels

Fig. 5 wondow & sofa

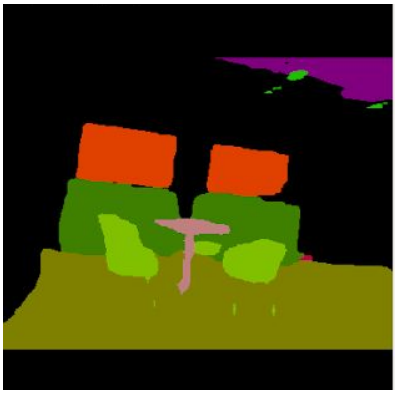

Fig. 6 TV & cabinet

Fig. 7 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 field includes position values (xyz), rgb values, semantic color, and confidence.

B. Merging Semantic PCL into an Octomap

To talk about the construction of an octomap, the original octomap data structure (fig. 8) has to be illustrated in advance. The arrows represent the class inheritance relations. In the Github code I cloned down, the author defined another class SemanticsOcTree inheriting OccupancyOcTreeBase and a class SemanticsOcTreeNode inheriting 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.

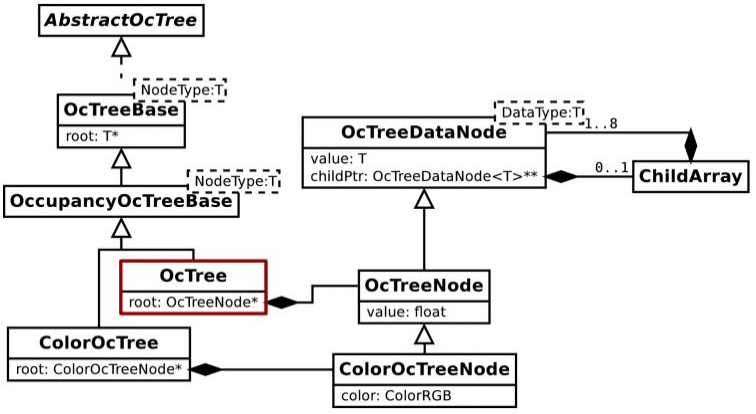

Fig. 8 Octomap data structure

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.

C. 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.

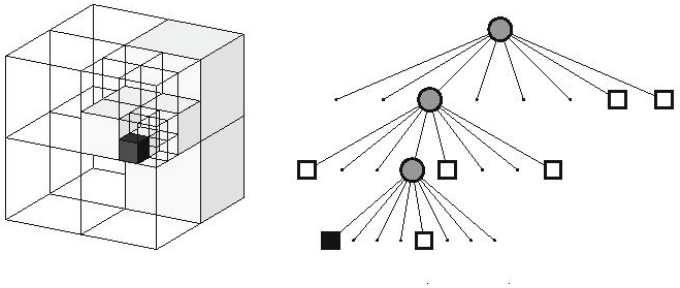

Fig. 9 Octree data structure

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 edge 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. 10 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 meantime, 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. 11).

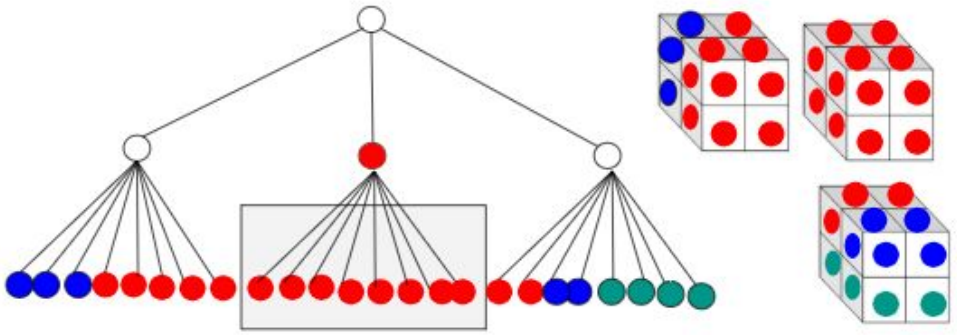

Fig. 10 A scenario of an octree

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

Fig. 11 Octomap localzation algorithm

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. Developing octomap_localization

$ catkin_create_pkg octomap_localization roscpp rospy std_msgs

write my code in the “octomap_localization” directory

$ cd ~/catkin_ws
$ catkin_make

V. Installing octomap_server

It should be a default package for ROS. However, if the package is missing,
$ cd ~/catkin_ws
$ git clone https://github.com/OctoMap/octomap_mapping.git
$ rosrun tf tf_echo camera_rgb_optical_frame camera_rgb_frame
add a node to transform original fixed frame_id /world to /map with previous found tf
$ rosrun tf static_transform_publisher 0 0 0 0.5 -0.5 0.5 0.5 /world /map 100
Modify some parameters in the octomap_mapping.launch:

- frame_id => /world
- colored_map => True
- latch => True
- ground_filter => True

Also, remap some topics:

- from cloud_in to /semantic_pcl/semantic_pcl
- from octomap_full to /duplicate_octomap_full
- from octomap_binary to /duplicate_octomap_binary

VI. Installing LzRobot

$ cd catkin_ws/src
$ git clone https://github.com/r06921017/lzrobot.git
$ cd ~/catkin_ws
$ catkin_make

VII. Write pub and sub for target coordinate

Modify the code in octomap_localization.cpp so that it will publish the target coordinate in the msg type: float64, float64, and float64. Of course you will have to create the.msg file. Next, write the listner in the lzrobot package.
$ cd ~/catkin_ws
$ catkin_make

VIII. Running the System

To run the five threads simultaneously, we have to run 8 commands in 7 terminals:
$ roslaunch semantic_slam camera.launch
$ roslaunch semantic_slam slam.launch
$ roslaunch semantic_slam semantic_mapping.launch
$ roslaunch octomap_server octomap_mapping.launch
$ roslaunch octomap_localization octomap_localization.launch

first connect pioneer with laptop usb, secondly power-on pioneer, and then launch a new terminal. Run the next 2 steps in the same tab.

```
$sudo chmod 777 /dev/ttyUSB0
$rosrun rosaria RosAria
$roslaunch lzrobot pioneer_simple_move_base.launch
```

If one need to control Pioneer manually, run the following command and you can control Pioneer with “uiojklm,.” keys.

```
$rosrun teleop_twist_keyboard teleop_twist_keyboard.py
/cmd_vel:=/RosAria/cmd_vel
```

Complete System Structure

The complete system structure is shown as fig. 12, containing three main parts: SLAM, visual search, and navigation. After gaining the 3D octomap as well as the coordinate of target object, we run navigation based on the 2D projected map. The navigation here is only based on odometry after path planning with the projected map. Move_base would send message to RosAria through the topic /cmd_vel, and RosAria would then communicate with Pioneer to move according with the value in /cmd_vel.

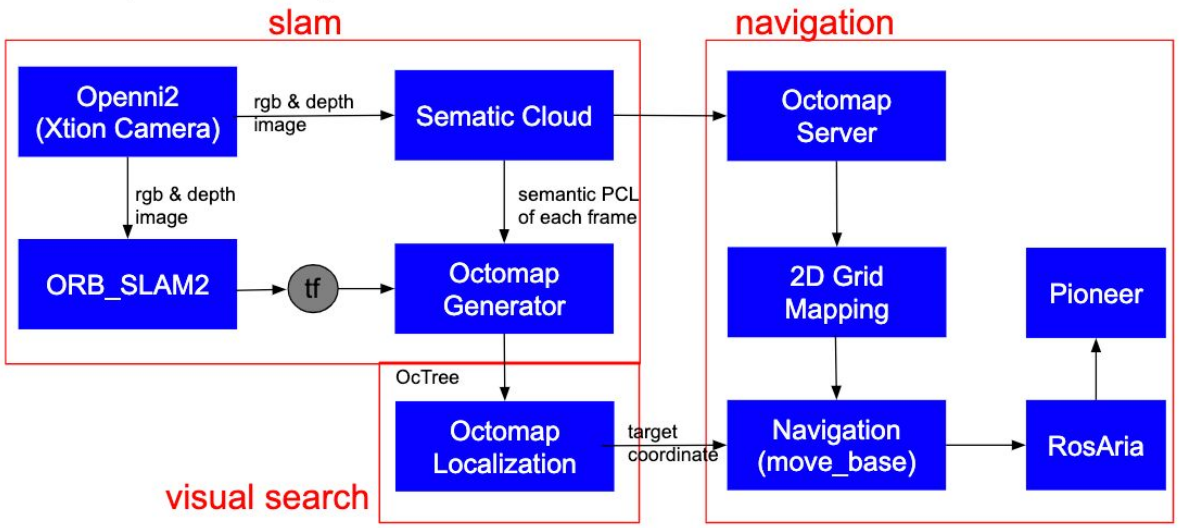

Fig. 12 Complete system structure

Challenge and Solution

I. Multiple topics with the same name publishing

Originally, /octomap_full and /octomap_binary are the two topics that both octomap_server and octomap_generator publish. Thus, the two packages are colliding with each other.

My workaround is that instead of stopping one of them form publishing, I simply remap the topics of one package to other names. To illustrate the strategy more clearly, I changed the following:

A. from cloud_in to /semantic_pcl/semantic_pcl
B. from octomap_full to /duplicate_octomap_full
C. from octomap_binary to /duplicate_octomap_binary

II. Projected map not aligned with 3D octomap

At first, the frame_id of octomap_server is set to be /world, which is the same frame_id set to octomap_generator, so I expect that both maps should align with each other. However, they did not (as shown fig. 13).

Solution: generate a fixed frame suitable for octomap_server

A. Step one: find the transform between 2 frames causing the tf mismatching
   $ rosrun tf tf_echo camera_rgb_optical_frame camera_rgb_frame (fig. 14)
B. Step two: add a node to transform original fixed frame_id /world to /map with previous found tf (fig. 15 ~ fig. 16)
   $ rosrun tf static_transform_publisher 0 0 0 0.5 -0.5 0.5 0.5 /world /map 100

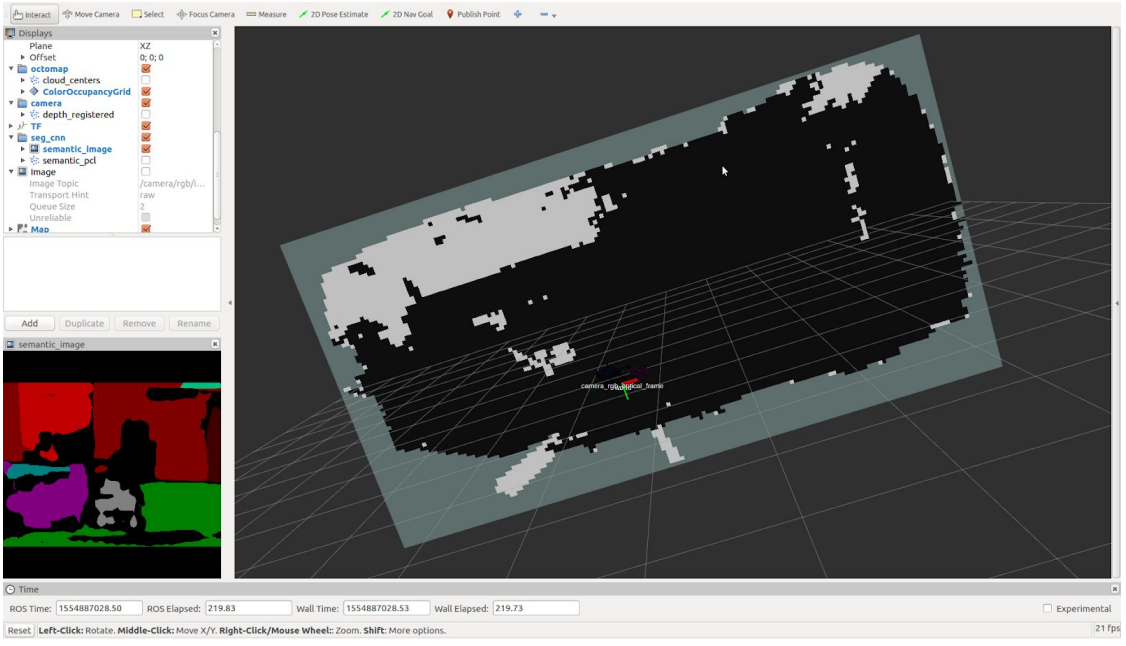

Fig. 13 Two maps not aligned with each other

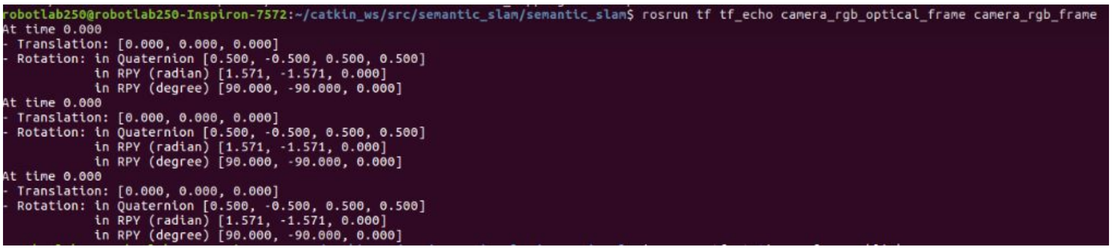

Fig. 14 Step one: find the transform between 2 frames causing the tf mismatching

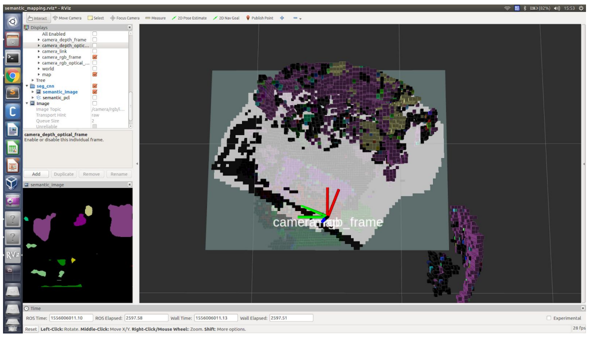

Fig. 15 top view of projected map

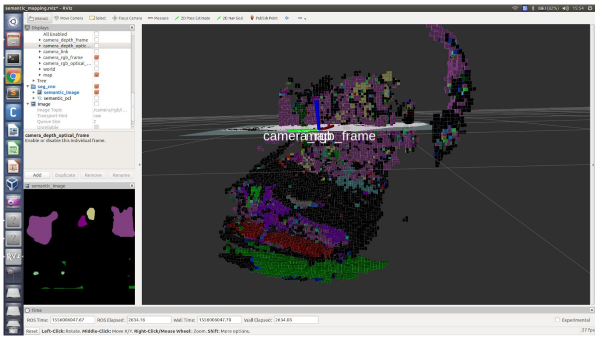

Fig. 16 side view of projected map

III. Transformation between /world and /odom missing

Originally, the transformation trees look like the graph in fig. 17, where the tf between /world and /odom in missing. /world is defined under semantic_slam, while /odom created by RosAria. In this way, Pioneer would not know where it is in the map, and thus impossible to navigate.

To solve this problem, I try to make /odom the same as the transformation between /world and /camera_link at the moment Pioneer is open. Therefore, the initial robot pose is provided by ORB_slam2 instead of RosAria odometry. The schematic diagram is shown at fig. 18. After looking up the transform between /world and /camera_link, it will then feed to /odom, so finally the whole graph is connected (fig. 19). By further checking camera_link is the same as odom, we are all set.

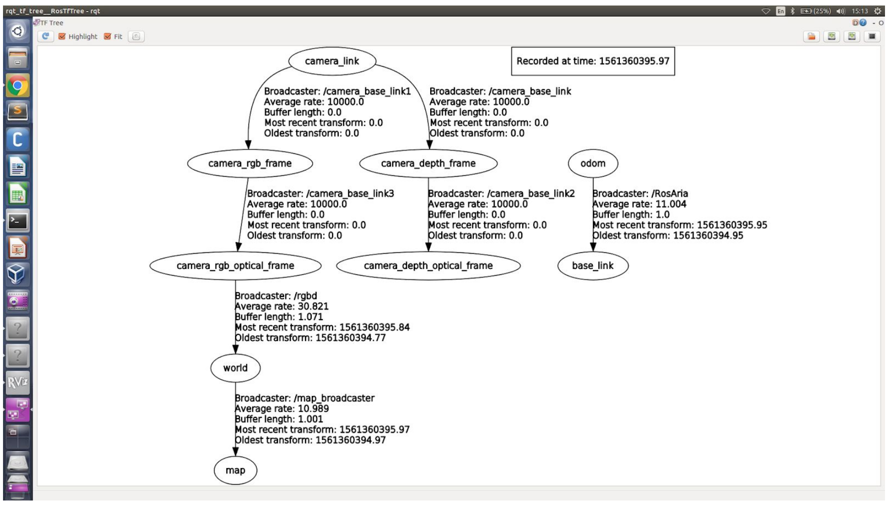

Fig. 17 orginal rqt tree

Fig. 18 schematic diagram

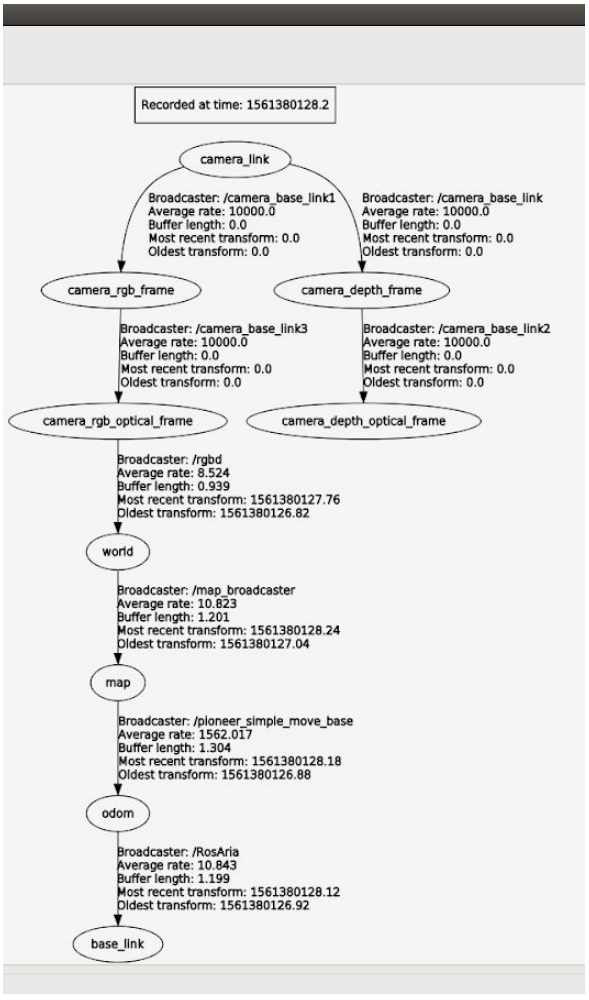

Fig. 19 connected graph

IV. Passing target coordinate to move_base

The target coordinate of aimed object is generated by my self-defined algorithm, and thus not published among nodes. In order to let the move_base package access target coordinate data, I have to first create a message type under

catkin_ws/src/octomap_localization/msg and modify cmakelists.txt as well as package.xml. Next, I modified some code in octomap_localization to publish the topic /target_coordinate, and at the move_base package, add a python script to subscribe the topic. In this way, the coordinate of the target object (x, y, z) is correctly transferred. (fig. 20)

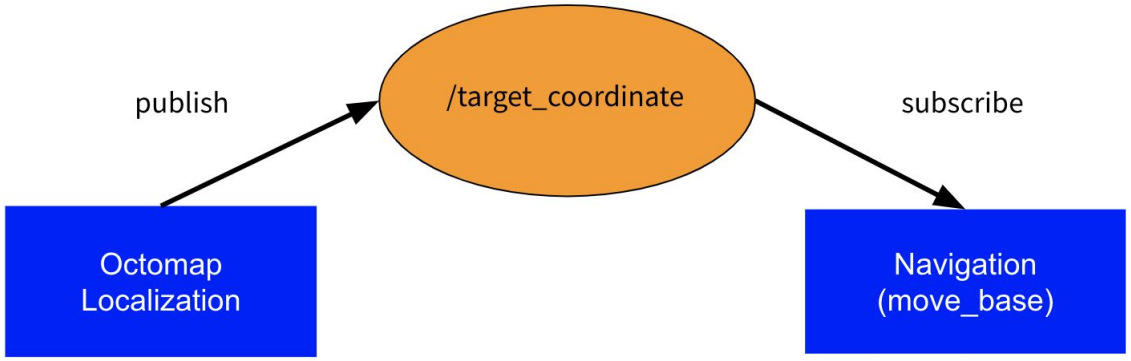

Fig. 20 /target_coordinate pub and sub

Achievement

In this project, I successfully implemented octomap localization on the semantic slam system, which is a visual search application of slam. An octomap localization algorithm is designed originally by me. It is proved the algorithm effective through testing. In addition, the 3D semantic map is down projected to a 2D grid map, and fed in to move_base navigation package. To sum up, this is a complete visual search system with SLAM and navigation functions.

Future Work

I. Improve accuracy of constructing 3D map

Currently the 3D map only preserves the relative positions of objects. However, the shape of the object is not preserved. For example, a table should be rectangular, but it is more like an oval in the current map. Moreover, the map is like a circle instead of the real shape of the room. Calibration of camera is required every time before building a map. Other algorithms may be implemented to the system as well.

II. Improve navigation method with depth-camera simulated laser

Right now the navigation method does not come with object avoidance due to the fact that I did not integrate the laser sensor with the system. Nevertheless, in order to make the system only dependent of visual sensors, I should simulate lasers with the depth data of Xtion camera. In that way, during navigation, Pioneer can avoid bumping into objects as well as calibrating its path real-time.

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
[9] Octomap, octomap_mapping, (2012), octomap_server, https://github.com/OctoMap/octomap_mapping
[10] Shao-Hung Chan, HackMD, (2019), Notes for Gmapping, https://hackmd.io/@shaohungchan/BkNvytnT-?type=view