Robotics Software Engineer Nanodegree Program: Term 2: Project 3: SLAM Project - Map My World

Li Feng,
acumenta@yahoo.com

ABSTRACT

This paper presents the implementation of SLAM (Simultaneous Localization And Mapping) in ROS, Gazebo, Rviz, and RTAB-Map (Real-Time Appearance-Based Mapping) environment and created 2D occupancy grid map and 3D octomap using two different world surroundings (i.e. one supplied - kitchen_dinning, the other student created - my_world). The results show that RTAB-Map approach can be an excellent solution for SLAM in the robots development.

1 Introduction

SLAM in robotics is to determine the location of the robot given uncertainties of the noisy sensors and imperfect actuators, and simultaneously mapping the unknown environment with no prior map. In this project, the student used the project 2's robot which consisted of laser scanner, IMU/Wheel Encoder, and camera. However, the camera was upgraded to a kinect RGB-D camera. Several launch files were created and teleop was used to move robot around the environment or worlds. RTABMap technique was used to localize the robot's position and generate maps with loop detections in two worlds. The 1st world is the kitchen_dining.world and a second world is my_world.world (created by student using gazebo).

2 Background

For robots to be useful to society, they must be able to move in environment that they've never seen before. A robot must construct a map of the environment, while simultaneously localizing itself relative to this map. Therefore, solving the SLAM problem is more challenging than localization and mapping alone, since neither the map nor the robots poses are provided. Currently, there are two common algorithms (i.e. FastSLAM and GraphSLAM) for solving SLAM. In this module, the student learned the Grid based FastSLAM and RTAB-Map which is based on Graph-SLAM.

2.1 Grid-based FastSLAM

The FastSLAM algorithm uses particle lter. Each particle holds robot trajectory which reduces the problem to mapping with known poses. It estimates a posterior over the trajectory using particle lter and uses a low dimensional EKF (Extended Kalman Filter) to solve independent features of the map which are modeled with local Gaussian. FastSLAM had the disadvantage in that known landmark positions are assumed, so it is not possible to model arbitrary environments.

The Grid-based FastSLAM algorithm extends FastSLAM to occupancy grid maps without predefining any landmark positions. Each particle holds a guess of the robot trajectory and its own map. To adapt FastSLAM to grid mapping, three different techniques were applied: (1) Sampling Motion: Estimates the current pose given the k-th particle previous pose and the current controls u. (2) Map Estimation: Estimates the current map given the current measurements, the current k-th particle pose, and the previous k-th particle map (3) Importance Weight: Estimates the current likelihood of the measurement given the current k-th particle pose and the current k-th particle map. The steps in the Grid-based FastSLAM algorithm are: Previous Belief, Sampling Motion, Importance Weight, Map Estimation, Resampling, and New Belief.

2.2 RTAB-Map - a GraphSLAM

GraphSLAM uses graph to represent SLAM problem. It constructs a graph whose nodes represent robot poses or features/landmarks and the edge between two nodes encodes a sensor measurement, like motion constraints or measurement constraints. After constructing a graph and defining the constraints, it then optimizes and solves sytem of equations and linearizes the non-linear motion and measurement constraints. GraphSLAM performs graph optimization minimizing the error in all the constraints in the graph using a principle known as Maximum Likelihood Estimation (MLE). Graph-SLAM complexity is linear with the number of nodes, which increases with the size of the map.

RTAB-Map (Real-Time Appearance-Based Mapping) implements GraphSLAM and this project uses RTAB-Map. It uses loop closure with Visual Bag-of-Words for optimization. Loop closure is the process of finding a match between current and previous visited image and location in SLAM. There are two types of loop closure detection: local and global. For local loop closure, matches are found in the limited map region with uncertainty associated with the robot's position. Whereas in global loop closure, a new location is compared with previously viewed locations. If no match found, the new location is added to memory. The amount of time to check can grow as the map grows. For this, RTAB-Map used memory management techniques to ensure that the loop closure process happens in real time. RTAB-Map uses only odometry constraints, no landmarks constraints. The possible outputs of RTAB-Map are 2D occupancy grid map, 3D octomap or a 3D point cloud.

3 Scene and Robot Configuration

A "slam_project" package was created inside the catkin workspace, src folder. The following subfolders (launch, materials, meshes, scripts, urdf, worlds) were created along with the appropriate files, and catkin_make was used to build the slam_project package.

Here is a list of files in the slam_project package:

./slam_project/CMakeLists.txt
./slam_project/package.xml
./slam_project/launch/localization.launch
./slam_project/launch/mapping.launch
./slam_project/launch/my_world.launch
./slam_project/launch/robot_description.launch
./slam_project/launch/rviz.launch
./slam_project/launch/teleop.launch
./slam_project/launch/world.launch
./slam_project/launch/config/robot_slam.rviz
./slam_project/materials/textures/kinect.png
./slam_project/meshes/hokuyo.dae
./slam_project/meshes/kinect.dae
./slam_project/scripts/rtab_run*
./slam_project/scripts/teleop*
./slam_project/urdf/li_bot.gazebo
./slam_project/urdf/li_bot.xacro
./slam_project/worlds/kitchen_dining.world
./slam_project/worlds/my_world.world

Here is a list of supporting files included in this project:

./images/frames-1.png
./images/KitchenDiningDBviewer.jpg
./images/KitchenDiningRtabmapViz.jpg
./images/li_bot.jpg
./images/li_bot_in_rviz.png
./images/li_bot_in_kitchen_dining_world.jpg
./images/my_world_dbviewer.jpg
./images/my_world_li_bot.jpg
./images/my_world_rtabmapviz.jpg
./misc/rtabmap_kitchen_dining.db
./misc/topic_list

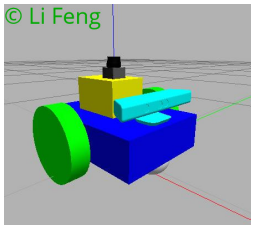
3.1 Robot - li_bot - Configuration

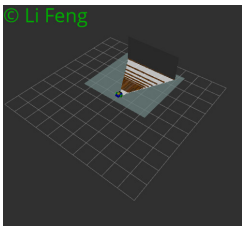
The robot model used was based on the li_bot created in the previous project. The camera was upgraded to a RGB-D kinect camera. The chassis's width was widened to support the width of the RGB-D kinect camera. The two skinny cylinders were removed and the top cube and hokuyo laser range finder were moved to the back of the chassis.

This upgraded li_bot is composed of a hokuyo laser range finder sitting on top of a cube which is sitting on the back of the main chassis, a RGB-D kinect camera sitting on the front of the chassis, two differential wheels (left and right), two passive wheels (front and back).

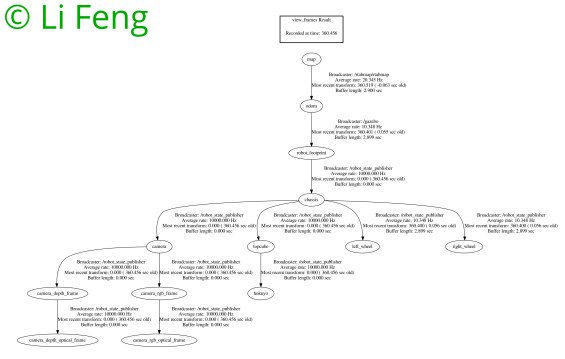
The urdf folder contains li_bot.xacro and li_bot.gazebo files. The li_bot.xacro provides the robot shape description in xacro format. The li_bot.gazebo provides definition of the differential drive controller for wheels, openni kinect camera controller for RGB-D camera, and the hokuyo controller for hokuyo laser scanner in the Gazebo environment. (Reference: ./slam_project/urdf/li_bot.xacro, ./slam_project/urdf/li_bot.gazebo)

Below, picture on the left is a standalone li_bot in Gazebo, picture in the center is a li_bot in the kitchen-dining world environment, and picture on the right is a li_bot in the rviz environment. (Reference: ./images/li_bot.jpg, ./images/li_bot_in_kitchen_dining_world.jpg, ./images/li_bot_in_rviz.png)





Visulization of the li_bot's frames are as following:



Reference ./misc/topic_list file for a list of topics published.

3.2 Launch files

There are seven .launch files in the ./slam_project/launch/ folder: (mapping.launch, my_world.launch, robot_description.launch, rviz.launch, teleop.launch, world.launch, localization.launch)

In the mapping.launch file below, three args (rviz, rtabmapviz, localization) are created, so that they can be set flexibly in the command line (ex: roslaunch slam_project mapping.launch rviz:="false" rtabmapviz="true" localization="false")
It also created four args (database_path, rgb_topic, depth_topic, and camera_info_topic) and set their defaut path according to li_bot and environment, so that they can be used or remapped in the rtabmap node and rtabmapviz node.
In the rtabmap node, it set five param appropriately and remapped the four args from above. It remapped grid_map to /map/. Other params were set appropriately.
In the rtabmap node, it set the loop closure detection to SURF(Speeded Up Robust Features).
In the rtabmap node, when the localization arg is true, it sets Mem/IncrementalMemory to false and Mem/InitWMWithAllNodes to true.

```
<?xml version="1.0" encoding="UTF-8"?>

<launch>

  <!--arg name="rviz" default="true"/-->
  <arg name="rtabmapviz" default="true"/>
  <arg name="localization" default="false"/>

  <!-- Arguments for launch file with defaults provided -->
  <arg name="database_path"     default="~/.ros/rtabmap.db"/>
  <arg name="rgb_topic"   default="/camera/rgb/image_raw"/>
  <arg name="depth_topic" default="/camera/depth/image_raw"/>
  <arg name="camera_info_topic" default="/camera/rgb/camera_info"/>

  <!-- Mapping Node -->
  <group ns="rtabmap">
    <node name="rtabmap" pkg="rtabmap_ros" type="rtabmap" output="screen" args="--delete_db_on_start">
      <!-- Basic RTAB-Map Parameters -->
      <param name="database_path"       type="string" value="$(arg database_path)"/>
      <param name="frame_id"            type="string" value="robot_footprint"/>
      <param name="odom_frame_id"       type="string" value="odom"/>
      <param name="subscribe_depth"     type="bool"   value="true"/>
      <param name="subscribe_scan"      type="bool"   value="true"/>

      <!-- RTAB-Map Inputs -->
      <remap from="scan"            to="/scan"/>
      <remap from="rgb/image"       to="$(arg rgb_topic)"/>
      <remap from="depth/image"     to="$(arg depth_topic)"/>
      <remap from="rgb/camera_info" to="$(arg camera_info_topic)"/>

      <!-- RTAB-Map Output -->
      <remap from="grid_map" to="/map"/>

      <!-- Rate (Hz) at which new nodes are added to map -->
      <param name="Rtabmap/DetectionRate" type="string" value="1"/>

      <!-- 2D SLAM -->
      <param name="Reg/Force3DoF" type="string" value="true"/>

      <!-- Loop Closure Detection -->
      <!-- 0=SURF 1=SIFT 2=ORB 3=FAST/FREAK 4=FAST/BRIEF 5=GFTT/FREAK 6=GFTT/BRIEF 7=BRISK 8=GFTT/ORB
9=KAZE -->
      <param name="Kp/DetectorStrategy" type="string" value="0"/>

      <!-- Maximum visual words per image (bag-of-words) -->
      <param name="Kp/MaxFeatures" type="string" value="400"/>

      <!-- Used to extract more or less SURF features -->
      <param name="SURF/HessianThreshold" type="string" value="100"/>

      <!-- Loop Closure Constraint -->
      <!-- 0=Visual, 1=ICP (1 requires scan)-->
      <param name="Reg/Strategy" type="string" value="0"/>

      <!-- Minimum visual inliers to accept loop closure -->
      <param name="Vis/MinInliers" type="string" value="15"/>

      <!-- Set to false to avoid saving data when robot is not moving -->
      <param name="Mem/NotLinkedNodesKept" type="string" value="false"/>

      <param if="$(arg localization)" name="Mem/IncrementalMemory" type="string" value="false"/>
      <param if="$(arg localization)" name="Mem/InitWMWithAllNodes" type="string" value="true"/>

    </node>

    <!-- visualization with rtabmapviz -->
      <node if="$(arg rtabmapviz)" pkg="rtabmap_ros" type="rtabmapviz" name="rtabmapviz" args="-d $(find
rtabmap_ros)/launch/config/rgbd_gui.ini" output="screen">
      <param name="subscribe_depth"             type="bool" value="true"/>
      <param name="subscribe_scan"              type="bool" value="true"/>
      <param name="frame_id"                    type="string" value="robot_footprint"/>
      <param name="queue_size"                  type="int" value="20"/>

      <remap from="rgb/image"       to="$(arg rgb_topic)"/>
      <remap from="depth/image"     to="$(arg depth_topic)"/>
      <remap from="rgb/camera_info" to="$(arg camera_info_topic)"/>
      <remap from="scan"            to="/scan"/>
      <remap from="odom"            to="/odom"/>
    </node>

  </group>

  <!-- Visualisation -->
      <!-- node if="$(arg rviz)" pkg="rviz" type="rviz" name="rviz" args="-d $(find
rtabmap_ros)/launch/config/demo_robot_mapping.rviz" output="screen"/-->
   <!--node if="$(arg rviz)" pkg="rviz" type="rviz" name="rviz" args="-d $(find slam_project)/launch/config/robot_slam.rviz"
output="screen"/-->
</launch>
```

The localization.launch file is exactly the same as mapping.launch file except it set the localization arg to true instead of false.

In the world.launch file below, it includes: robot_description, the kitchen_dining.world, and the DepthImageToLaserScanNodelet. It then spawns the robot in the gazebo environment.

```
<?xml version="1.0" encoding="UTF-8"?>

<launch>
  <include file="$(find slam_project)/launch/robot_description.launch"/>
  <arg name="world" default="empty"/>
  <arg name="paused" default="false"/>
  <arg name="use_sim_time" default="true"/>
  <arg name="gui" default="true"/>
  <arg name="headless" default="false"/>
  <arg name="debug" default="false"/>
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" value="$(find slam_project)/worlds/kitchen_dining.world"/>
    <arg name="paused" value="$(arg paused)"/>
    <arg name="use_sim_time" value="$(arg use_sim_time)"/>
    <arg name="gui" value="$(arg gui)"/>
    <arg name="headless" value="$(arg headless)"/>
    <arg name="debug" value="$(arg debug)"/>
  </include>
  <node name="urdf_spawner" pkg="gazebo_ros" type="spawn_model" respawn="false"
      output="screen" args="-urdf -param robot_description -model li_bot"/>
  <node pkg="nodelet" type="nodelet" name="laserscan_nodelet_manager" args="manager"/>
  <node pkg="nodelet" type="nodelet" name="depthimage_to_laserscan"
      args="load depthimage_to_laserscan/DepthImageToLaserScanNodelet laserscan_nodelet_manager">
    <param name="scan_height" value="10"/>
    <param name="output_frame_id" value="/camera_depth_frame"/>
    <param name="range_min" value="0.45"/>
    <remap from="image" to="/camera/depth/image_raw"/>
    <remap from="scan" to="/scan"/>
  </node>
</launch>
```

The my_world.launch file includes my_world.world instead of kitchen_dining.world. Otherwise, it is exactly the same as world.launch file.

teleop.launch: launches the teleop service for robot movement.

```
<?xml version="1.0" encoding="UTF-8"?>
<launch>
  <node name="teleop" pkg="slam_project" type="teleop" output="screen">
    <remap from="teleop/cmd_vel" to="/cmd_vel"/>
  </node>
</launch>
```

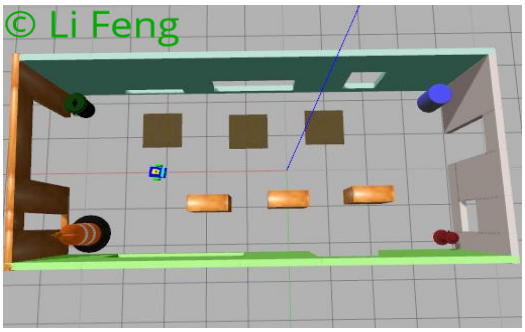
rviz.launch: can be launched independetly as below. (vs in the mapping.launch)

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- For visualising your robots map bulding in RVIZ -->

<launch>
  <node pkg="rviz" type="rviz" name="rviz" args="-d $(find slam_project)/launch/config/robot_slam.rviz" output="screen"/>
</launch>
```

3.2 World (i.e. my_world.world) Creation

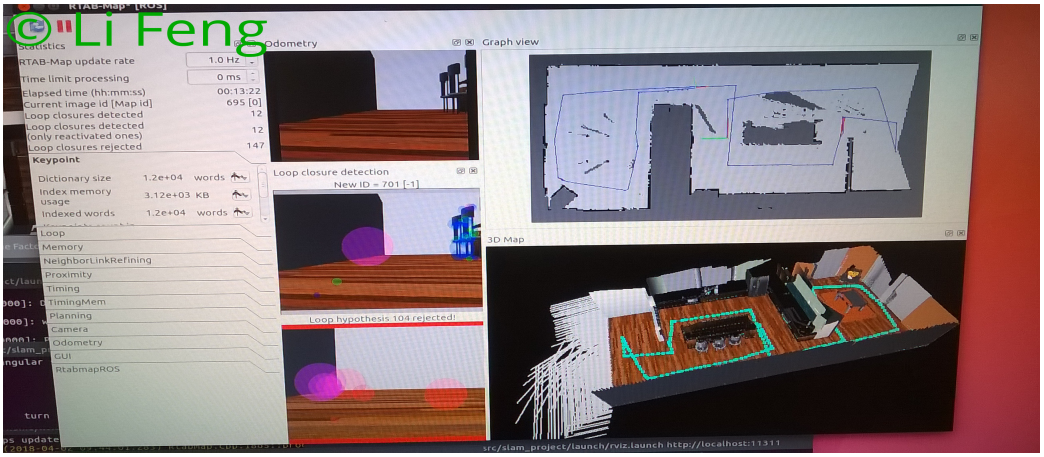
The world file "my_world.world" was created in gazebo through following steps:
(1) started "gazebo" in the command line window
(2) clicked "Edit", then "Building Editor"
(3) created 4 walls, windows, doors, and colored the walls
(4) saved it as a model (ie li model) in the gazebo's building editor folder
(5) exited the "Building Editor"
(6) clicked "Insert" tab, and choose the above created model (li model)
(7) deposited followings into the scene: (3 cafe tables, 3 bookshelves, 1 construction barrier, 1 fire hydrant, 1 trash can, 1 blue cylinder)
(8) clicked "File"->"Save World As" and saved the file as : ./slam_project/worlds/my_world.world.

Below is a screenshot of li_bot in the my_world.world Gazebo environment. (reference: ./images/my_world_li_bot.jpg)

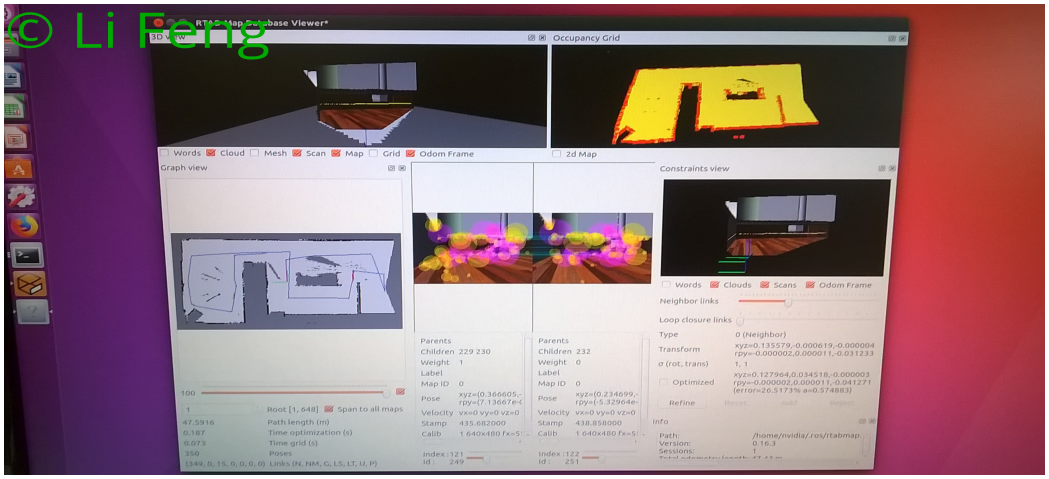


4 Results

Below is a snapshot of running the rtabmapviz, using teleop keys, traversed the kitchen_dining.world. It had 2D GraphView and 3D Map and stated 12 loop closures detected in the Statistics pannel. (Reference: ./images/KitchenDiningRtabmapViz.jpg)

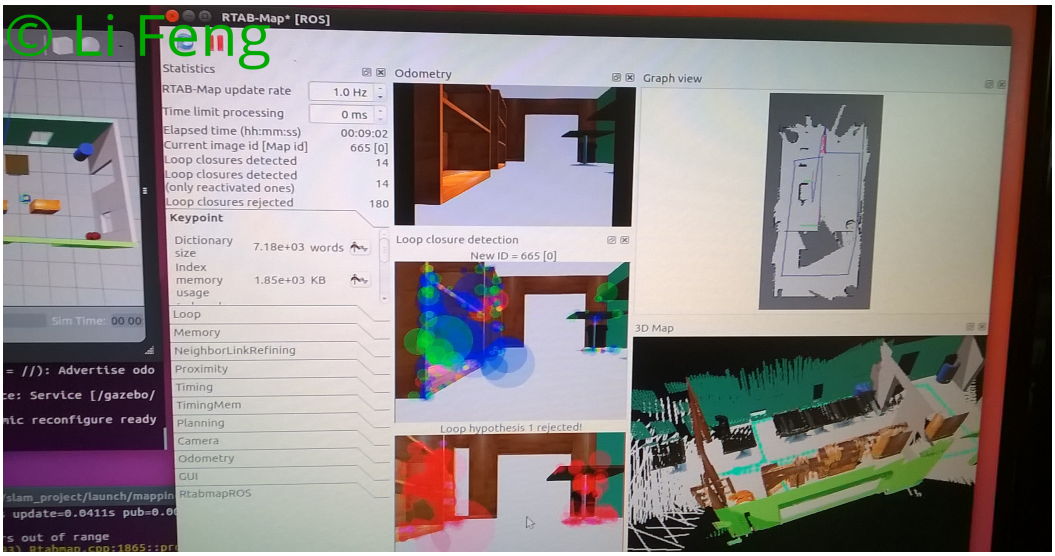


Below is a snapshot of running rtabmap-database-viewer on the rtabmap.db of the kitchen_dining.world. It contained 3D View, Occupancy Grid, GraphView, OdomFrame, Constraintsview, and it displayed 15 Global Loop Closures detected.(Reference: ./images/KitchenDiningDBviewer.jpg)

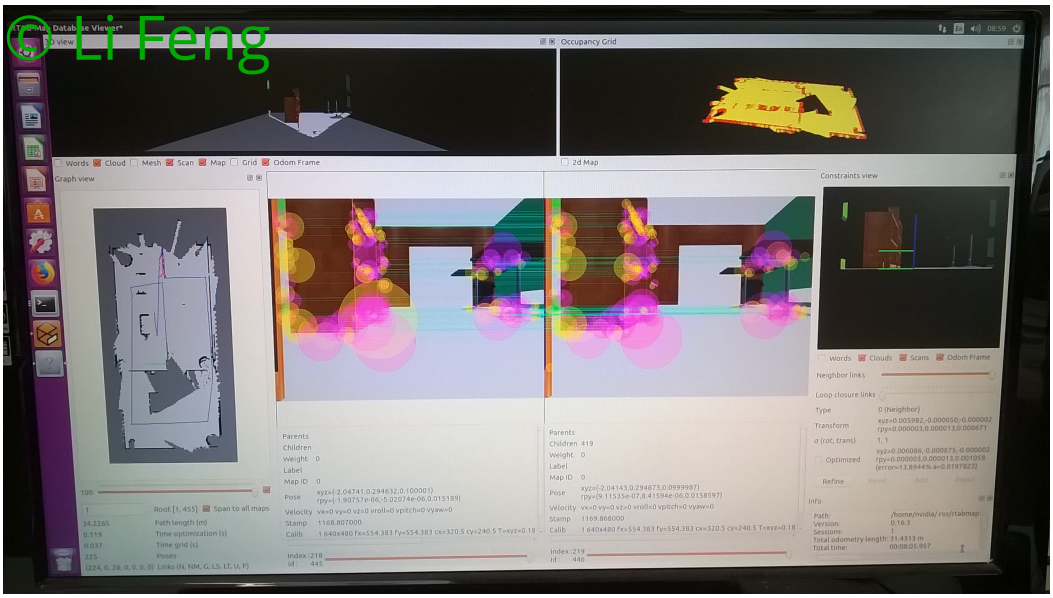


4 Results - continued

Below is a snapshot of running the rtabmapviz using teleop keys and traversed my_world.world. It had 2D GraphView and 3D Map and stated 14 loop closures detected in the Statistics pannel (Reference: ./images/my_world_rtabmapviz.jpg).



Below is a snapshot of running rtabmap-database-viewer on the rtabmap.db of the my_world.world. It contained 3D View, Occupancy Grid, GraphView, OdomFrame, Constraintsview, and it displayed 28 Global Loop Closures detected. (Reference: ./images/my_world_dbviewer.jpg)



5. Discussion

The robot and model configurations performed very well in both kitch-dining.world and my_world.world. The student had successfully met the course requirements of creating a ROS package, several launch files, a student created world, and launched the robot in two worlds, moved the robot using teleop keys in the ROS, Gazebo, Rviz, rtabmapviz, and rtabmap-databaseViewer environment. More than 3 loop closures were identified and recorded in the rtabmap.db files.

For the kitchen-dining.world, sometimes the robot stuck in the rug or corner and could not be moved easily. The database files grew very large (ex: ~172 MB for kitchen-dining.world).

For the my_world.world, there was 14 loop closures in the rtabmapviz, but there was 28 loop closures in the rtabmap-databaseViewer. It's probably due to the double images generated at the end of traversing on both side of the building and that resulted loop closures to be counted twice.

The student had created and tested many other my_world.world files earlier (ex: cafe, OSRF buildings, etc), and they did not work well. The contributing factors could be that the buildings were large in size and objects were repeated often. The student noticed when loop closure happened, it slowed down a little, and might impose images incorrectly to satisfy the balance of all constraints.

Although GraphSLAM solved the SLAM problem, it does not work well in similar environment, like maze, or similar colors objects. It does not handle a large open space well. It may have loop closure, but the images could be sliced or staggered or not precise. These problem maybe resoved by adding reenforcement learning algorithm or path planning.

6. Future Work

RTAB-Map is a top algorithm for solving SLAM problem. However, based on experience from this project, the RTAB-Map would not work well for care-provider robots, because it does not seem to be able to reliablly recognize small objects in crowded spaces. So, there is room for improvement here for future work.

The student would like to automate the teleop operation. For example, it records the sequence of keystrokes the first time around and plays back the sequences thereafter.

The student would like to spend more time playing around with rtabmapviz and rtabmap-databaseViewer application programs. There are many buttons or options in the pull-down menu and there are many parameters settings in the .ini files.

References

[1] A Tutorial on Graph-Based SLAM, G. Grisetti, R. Kummerle, C. Stachniss, and W. Burgard.
[2] The graphSLAM algorithm with applications to large-scale mapping of urban structures, S. Thrun and M. Montemerlo.
[3] https://introlab.3it.usherbrooke.ca/mediawiki-introlab/images/3/31/Labbe2015ULaval.pdf
[4] https://github.com/LCAS/lincoln_youbot/blob/master/picksie_descriptor/urdf/sensors/kinect_camera.urdf.xacro
[5] http://wiki.ros.org/depthimage_to_laserscan
[6] https://github.com/introlab/rtabmap_ros
[7] https://github.com/introlab/rtabmap_ros/tree/master/launch/demo
[8] https://github.com/introlab/rtabmap_ros/blob/master/launch/demo/demo_turtlebot_mapping.launch
[9] https://www.youtube.com/watch?v=gJz-MWn7jhE