VERSION [1.0]

MARCH 22, 2019

RoboND-SLAM-Project

ABSTRACT

In this project, Graph-SLAM which is an efficient algorithm for simultaneous mapping and localization problems is implemented and tested.

Robot model used in this project is built upon previous model that was used in RoboND-Localization project, Briefly, it consists of simple base with 2 differential wheels and 2 caster wheels to balance it in addition to RGB camera and Laser range sensor.

Testing process is performed in gazebo simulated environment which provides access to variety of models and worlds which gives us the advantage to observe the efficiency in different fields.

Through this process, input data is provided through RGBD camera and Laser Range Finder sensor to specific ROS package which is responsible for performing mapping and localization calculations and giving the results.

Final result consists of 2D and 3D maps of the environment and database of the completed process.

INTRODUCTION

When it comes to Robotics area, Simultaneous mapping and localization problems isn't that travail task that can be handled perfectly.

There has been a lot of research and development in this area due to the severe importance of stable, efficient and Realtime algorithms for mapping & localization in many aspects of robotics.

Nowadays, there is many algorithms and solution to solve SLAM problems, the most popular ones are:

- Fast-SLAM
- Grid-based Fast-SLAM
- Graph-SLAM

Each of these algorithms has its Pros and Cons, so choosing the suitable one for your project mainly depend on:

- Process power available.
- Problem Field.
- Circumstances needed to be handled.
- Required accuracy.

In this project, SLAM problem is handled using Graph-SLAM algorithm because of its accuracy over EKF and particle filters-based algorithms.

RTAB-Map which is a ROS package is used in this project because it provides the implementation for Graph-SLAM algorithm.

RTAB-Map package generates both 2D occupancy grid map and 3D point clouds of the environment.

BACKGROUND /FORMULATION

This is high level knowledge of important concepts that is addressed in this project.

FAST SLAM

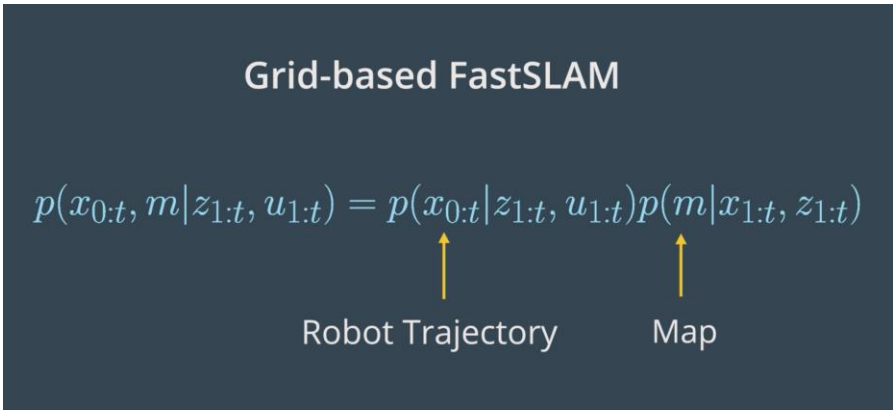
it is an algorithm used to solve Full-SLAM with known correspondences.

This algorithm uses:

- Custom particle filter to estimate trajectory of the robot, this will give an advantage to SLAM to solve the problem of mapping with known poses.
- Low dimension Extended Kalman Filter to solve independent features of the map which are modeled with local Gaussian.
- sample inefficiency is generated if particle filter is in high level.
- Can't solve arbitrary environment problems.

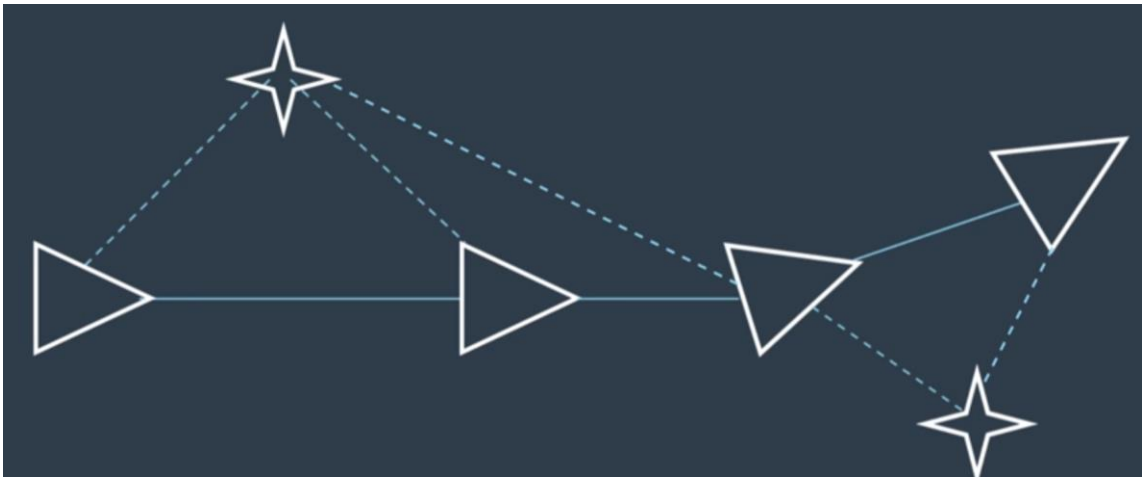
GRID-BASED FAST-SLAM

- Can be considered as extension of Fast-SLAM
- This algorithm adapts FastSLAM to grid maps.
- environment can be modeled using grid maps without predefining any landmark position.
- Uses MCL to get possible pose of the robot.
- can solve the SLAM problem in an arbitrary environment.
- Sampling motion, Map Estimation and Importance Weight techniques are used to implement Grid-mapping.



GRAPH-SLAM

- Solves Full SLAM Problem.
- Graph-SLAM concept is based on covering the entire path and map instead of just most recent pose and map.
- Dependencies exist between current and previous poses.
- Uses constraints:
    - Motion constraints: ties each successive poses to each other giving us result of likelihood between poses.
    - Measurement constraint: ties each measurement reading to specific poses giving us probability of the poses to have this feature.



- Parts:
  - Front-End:
    - Constructs graph using odometry and, sensors.
    - Creates nodes, edges then add measurements to it.
    - Depends on the application.
    - Solves data association problem.
  - Back-End:
    - Takes complete graph and all constraints as input.
    - Outputs the most probable configuration of robot poses. and map features.
    - Can be classified as optimization process.
    - Consistent across applications.
- Advantages:
    - Reduces process power needed
    - Has higher accuracy than Fast-SLAM

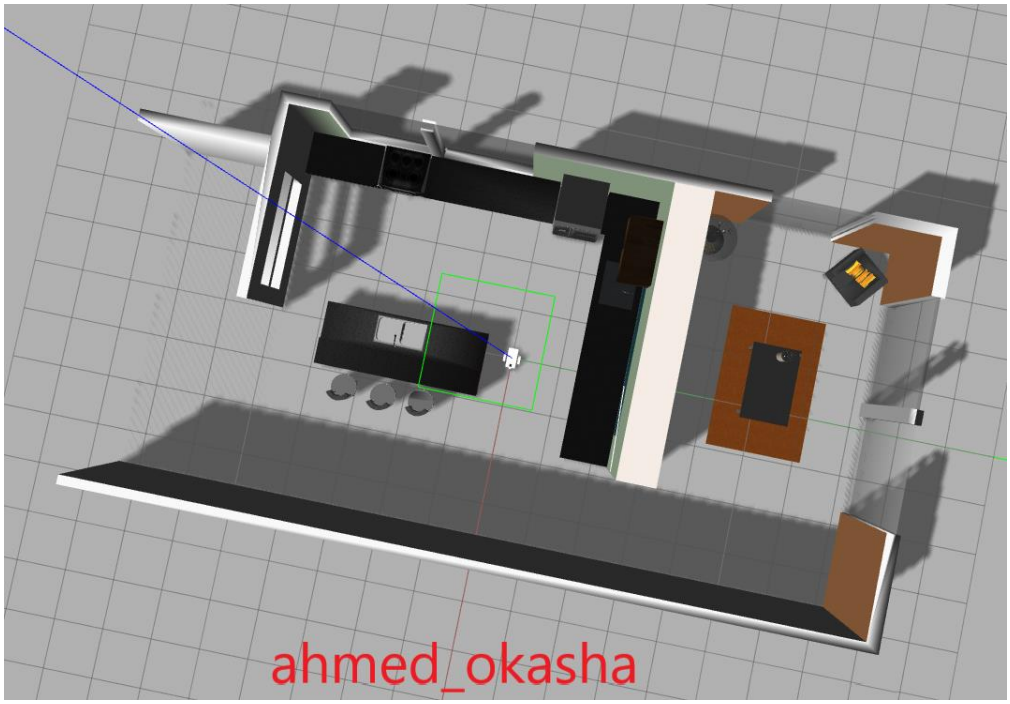
SIMULATION GAZEBO WORLDS

KITCHEN DINNING

This world is provided with the project materials.

Consists of:

- Kitchen room.
- Dining room.
- Windows and doors.



CREATED WORLD

This world is created with gazebo with models that is available in gazebo database.

Consists of:

- Plane.
- Helicopter parking area.
- Fast-Food restaurant.

- Tree.
- Beach Buggy.
- Fountain.
- Gracey barrier.
- Mars-Rover model.
- Cube with number 3.

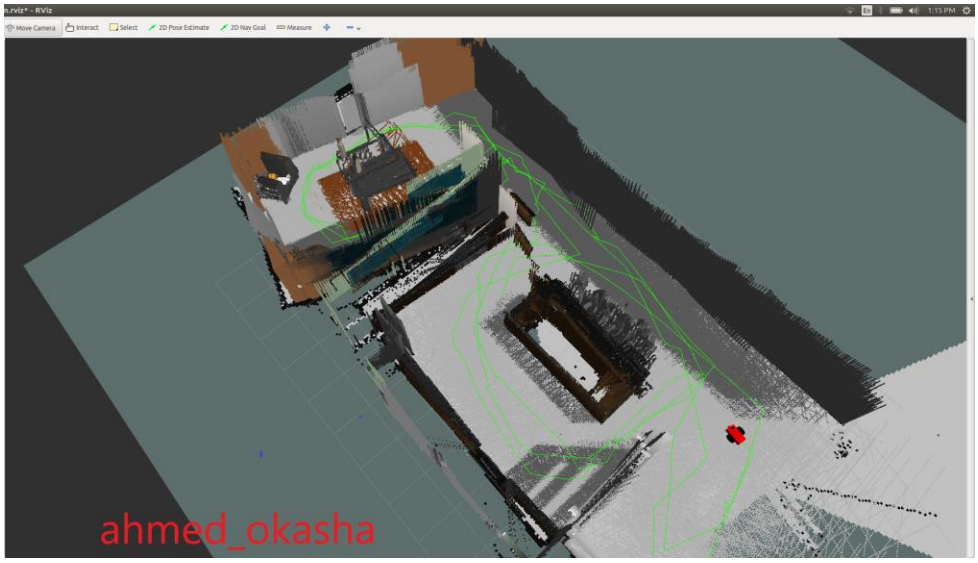
RESULTS

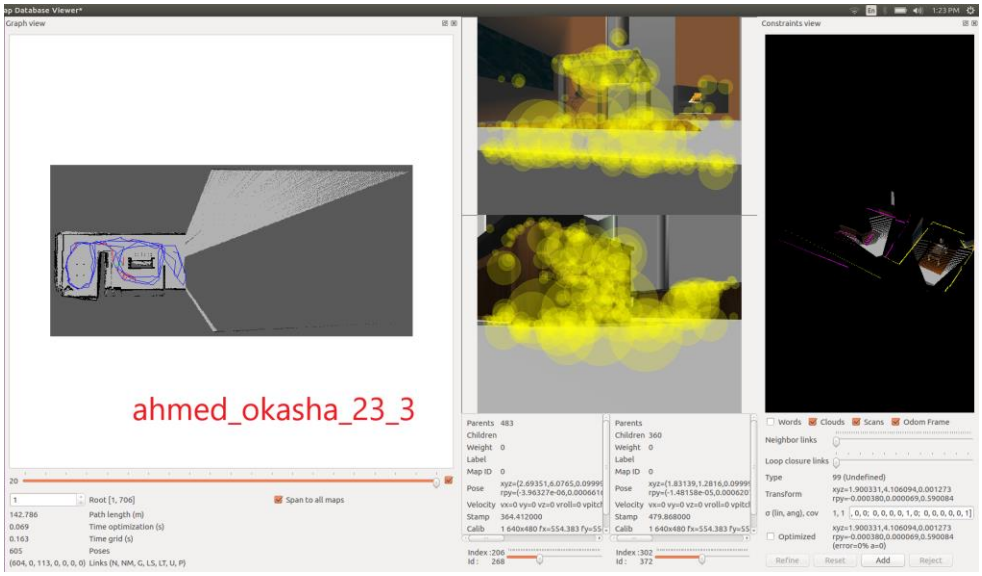
SUPPLIED WORLD

PARAMETERS

Detection rate: 1Hz

Minimum inliers: 15



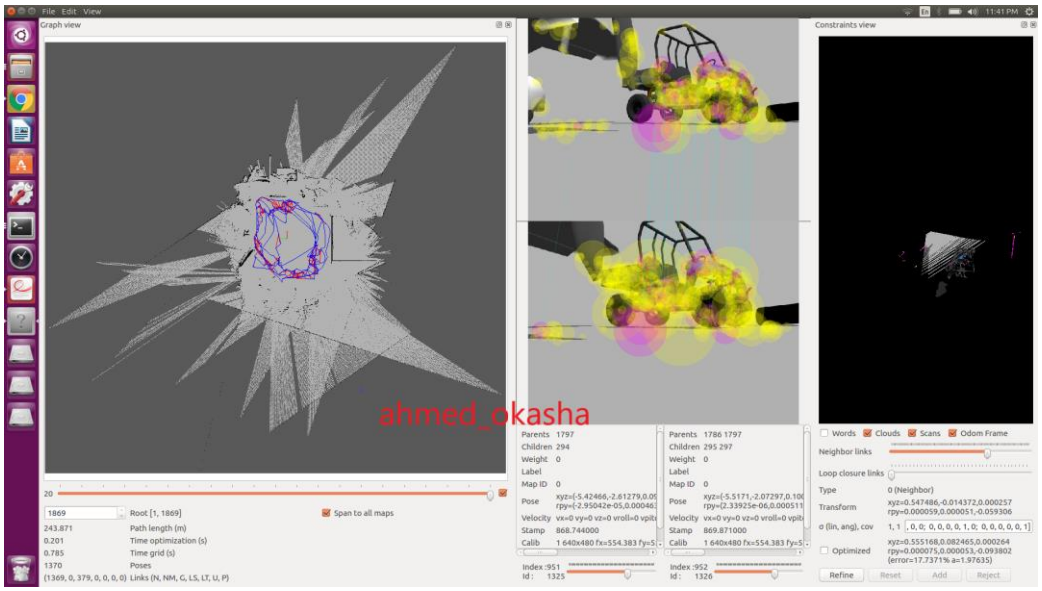


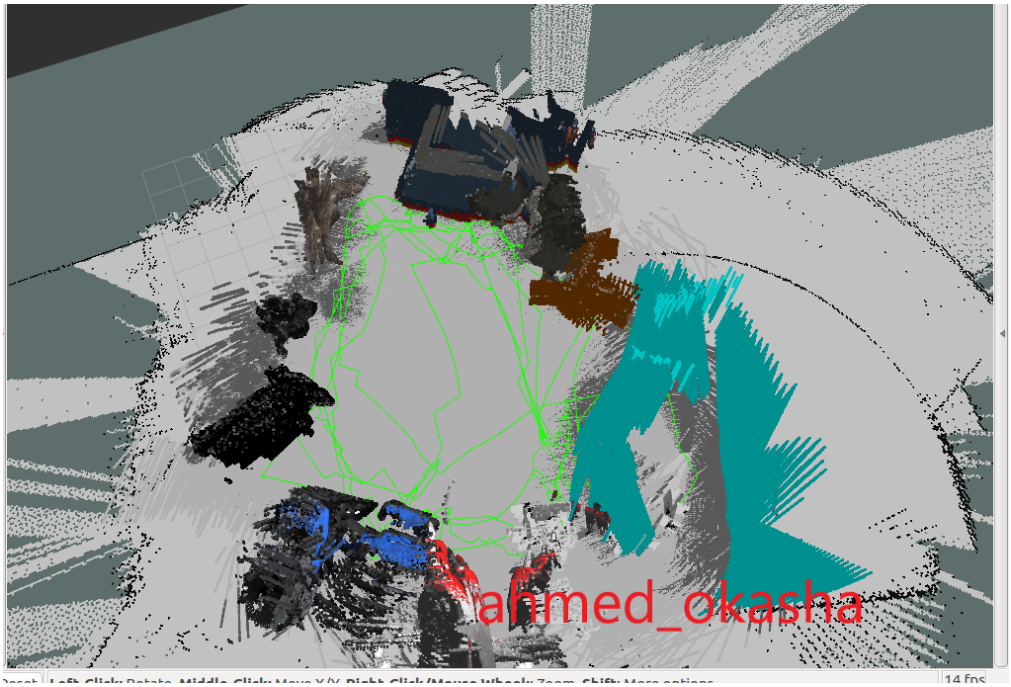
NEW WORLD

PARAMETERS

Detection rate: 4Hz

Minimum inliers: 10





PROJECT WALKTHROUGH

LOCALIZATION PROJECT PACKAGE

This project is built upon localization project package.

PACKAGE DETAILS:

- Robot model base has 2 differential wheels and caster wheels to stabilize the robot.
- Robot is mounted with RGB-camera and range finder laser sensor.
- Necessary launch files for gazebo world and RViz is provided.
- localization files (amcl, navigation_goal.cpp, amcl.launch) aren't necessary and could be deleted.

UPDATING PACKAGE FOR GRAPH-SLAM

CONFIGURING RGP-D CAMERA

- Old RGB Camera is replaced with Kinect camera which is an RGB-D camera.
- To complete this modification camera link in URDF file is replaced with the code provided.
- Required linkages are added to URDF files.
- Mis-named Topics are edited.

BUILDING LAUNCH FILES

GAZEBO WORLD

- Gazebo-World launch file is obtained from **"udacity _world.launch"** after deleting unnecessary launch headers.
- Launch File is modified with the new directory of the supplied gazebo world
- 2 environments are used in simulation, one is **"kitchen_dining"** and the second one is made using gazebo.

BUILDING TELEOP NODE

- Teleop node goal is to enable robot movement using keyboard during Mapping and localization process.
- Node script is attached with the supplied project materials.
- launch file is created for this node to:
    - make debugging easier.
    - Map necessary topics.

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

BUILDING RVIZ LAUNH FILE

- Rviz is used for visualizing robot obtained map.
- Configuration file is supplied with project materials.
- Path of configuration file is added to enable launching the same configuration every time.

BUILDING MAPPING LAUNCH FILE

- This file acts as the main node that interfaces with all the required parts to be able to perform SLAM with RTAB-Map.
- File template is provided in project materials.
- Database file directory is modified to be in **“/.ros/ “**.
- Frame id value is changed from **“base_footprint”** to **“robot_footprint” to** suit old package.
- Visualization part is added to enable Real time Visualization with **rtabmapviz** when needed.

```
 <!-- Arguments for launch file with defaults provided -->
<arg name="database_path"     default="~/.ros/rtabmap.db"/>
```

```
<param name="frame_id"          type="string" value="robot_footprint"/>
```

```
<!-- visualization with rtabmapviz -->
   <node 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"/>

    <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"/>
   </node>
```

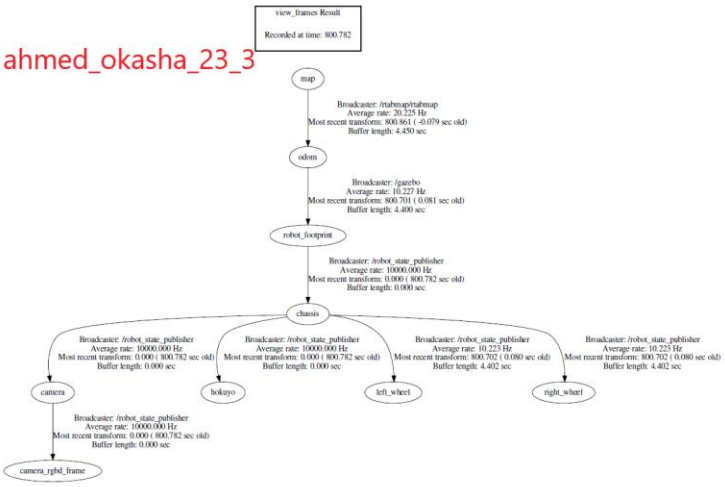
DEBUGGING

These tools were used for debugging in this project.

TRANSFORM FRAMES

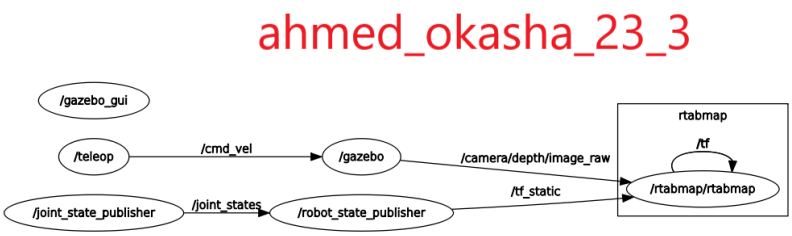
- The tf library helps keep track of all these different coordinate frames over time, and defines their relation with one another.
- The library allows you to debug your coordinate frames (or the tf tree) in several ways. **view_frames** will create a graphical representation of that tree, providing you with a broad view of how different frames (or links) in your setup connect to each other.

- Results:



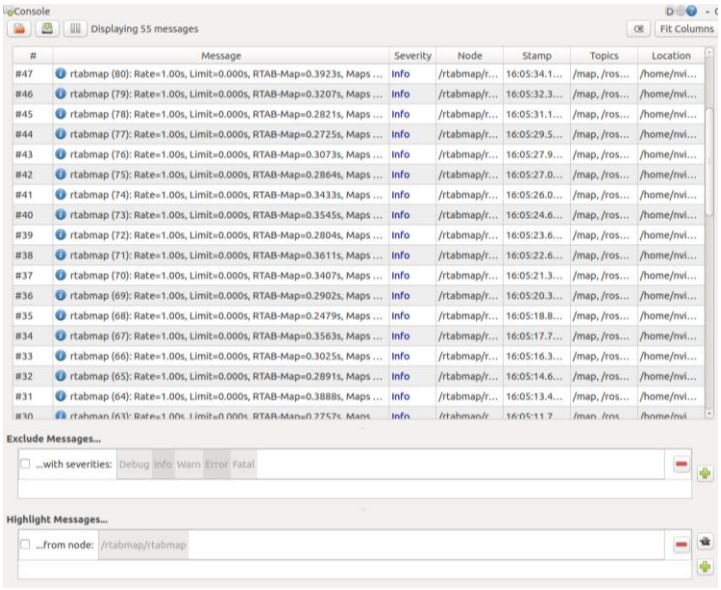
RQT_GRAPH

- Used to visualize the ROS graph corresponding to that package.
- Output: image
- **Roswtf** can be used to examine and analyze the setup or the graph above including any running nodes and environment variables, and warn you about any potential issues or errors.
- Output:



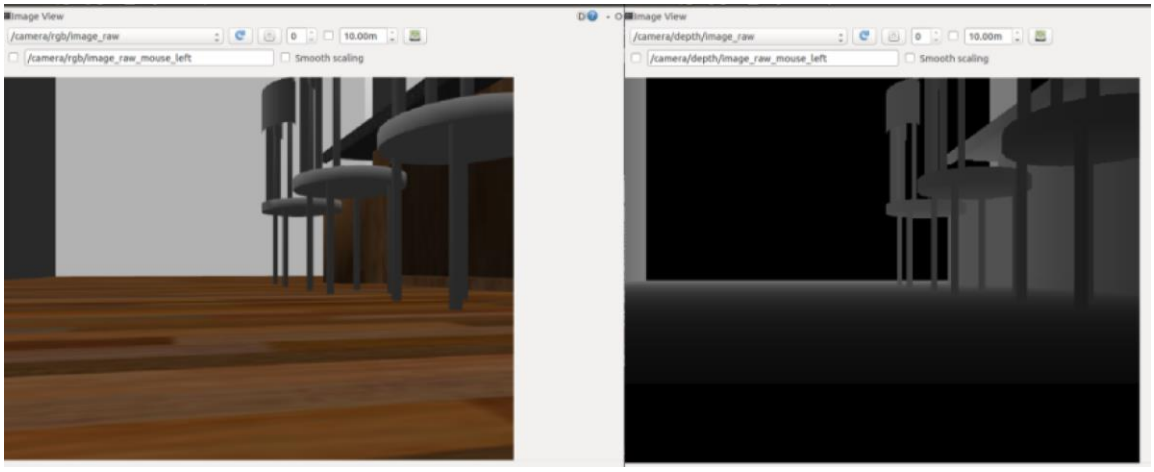
RQT_CONSOLE

- It aggregates all of the log messages of existing errors and allows you to sort them.
- Per documentation each message includes:
    - Message: The message specified by the user
    - Severity: The severity level of the message, e.g. Debug, Info, etc.
    - Node: The name of the node which broadcast the message
    - Time: The time at which the message was broadcast
    - Topics: The topics advertised by the node broadcasting the message
    - Location: Combines the file, function and line using colons
- Should run before launching any ROS code.



RQT_IMAGE_VIEW

This is a great tool to visualize images streaming from camera and depth maps from sensors. Use it to make sure you are properly receiving the data you want on its specific topic.



PERFORMING SLAM PROCESS

- SLAM process is performed with 3 loop closures.
- Date base which has details about:
    - Features and neighboring details
    - Robot path during SLAM process
    - Obtained 2D Map
    - Neighbor Merged
    - Global loop closure
    - Local loop closure by space
    - Local loop closure by time
    - User loop closure
- Database can be viewed by **rtabmap-databaseViewer** which is a great tool for exploring your database when you are done generating it.

DISCUSSION /FUTURE WORK

ERRORS

TELEOP NODE PARAMETERS MAPPING:

- “teleop/cmd_vel” needed to be mapped to “/cmd_vel” to suit old package topics.

MAPPING NODE ERRORS

When I was launching mapping.launch file at the beginning, I had an error related to the URDF model ,but after sometime I figured out that I forgot to add RGBD camera links in URDF files.

MAP DISTORTION

3D map gets distorted when:

- There is a lot of identical parts in the environment
- High rotational movement at beginning can causes errors in loop closure
- open environment with huge or symmetric models produces conflict in features identification.

- the less features in the environment the less obtained accuracy will be.
- The less frames per second token by mapping node the less neighboring features and total accuracy will be.

DISCUSSION

GAZEBO WORLDS DISCUSSION

1- Kitchen_dining

Results seams more accurate than created world, although created world Detection frame used =4Hz and this can be result of:

- High features density of world objects.
- Environments has low similarity between objects.
- Environment isn’t wide (has only 2 rooms)

The most confusing part of the environment to the robot is the long wall and this perhaps led to distortion in the wall between the two rooms.

2- Created World

Results seams less accurate than “kitchen_dining” world despite the high detection rate and this can be resulted from:

- Minimum inliers low value which can result wrong detected closure loops.
- Environment low feature density in each frame.
- Environment has similarity between objects and symmetric models.
- Environment is wide and have huge objects.

The most distorted part is the fast-food restaurant and this can be due to its symmetric shape.

CONCLUSION

- Graph-SLAM performs better in closed environment because its size is mostly smaller than open ones.
- Graph-SLAM performs better in features-rich environment.
- Environment with identical shapes or symmetric shapes can produce in accuracy in model
- It is better for total accuracy to divide environment map into smaller loops and then extend them slowly.
- Neighboring features can be increased by increasing total frame per second

FUTURE WORK

- Localization node can be added in the future.
- This part can be added to the drone assignment to obtain map of the environment and following the specific person with map and obstacle information available.
- RTAP-Map obstacle detection features can be added to static arm mover projects to avoid collision without much increase in process power because environment is limited to workspace.