Map My World Robot
Mohamed Ewis

Abstract—This paper implements Simultaneous Localization and Mapping (SLAM) technique to construct a map of a given environment. A Real Time Appearance Based Mapping (RTAB-Map) approach was taken for accomplishing this task. Initially, a 2d occupancy grid and 3d octomap was created from a provided simulated environment. Next, a personal simulated environment was created for mapping as well. In this appearance based method, a process called Loop Closure is used to determine whether a robot has seen a location before or not.

1 Introduction:
In SLAM (Simultaneous Localization and Mapping), a robot must construct a map of the environment, while simultaneously localizing itself relative to this map. This problem is more challenging than localization or mapping, since neither the map nor the robot poses are provided. With noise in the robot's motion and measurements, the map and robot's pose will be uncertain, and the errors in the robot's pose estimates and map will be correlated. The accuracy of the map depends on the accuracy of the localization and vice versa. Given a series of sensor observations over discrete time steps , the SLAM problem is to compute an estimate of the agents location and a map of the environment. In this paper, two simulation environments were provided where SLAM was performed. The robot was successfully able to localize itself and map the 3d world.
The benchmark environment is called kitchen-dining (Figure 1) and the second environment is that of a cafeteria called ewis_cafe (Figure 2).

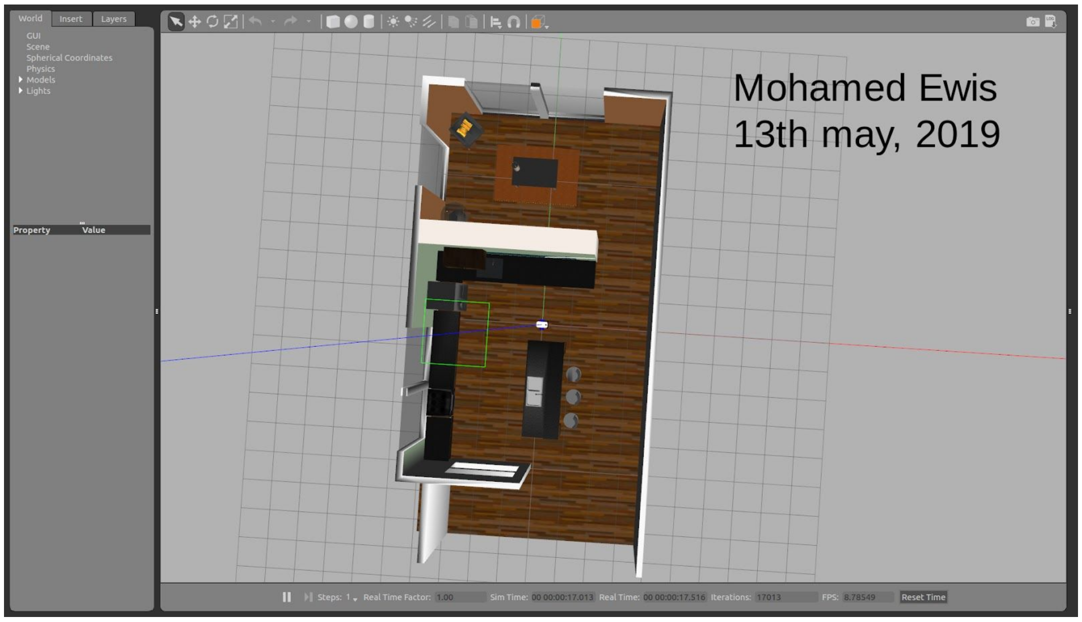


Fig. 1. Kitchen-Dining World

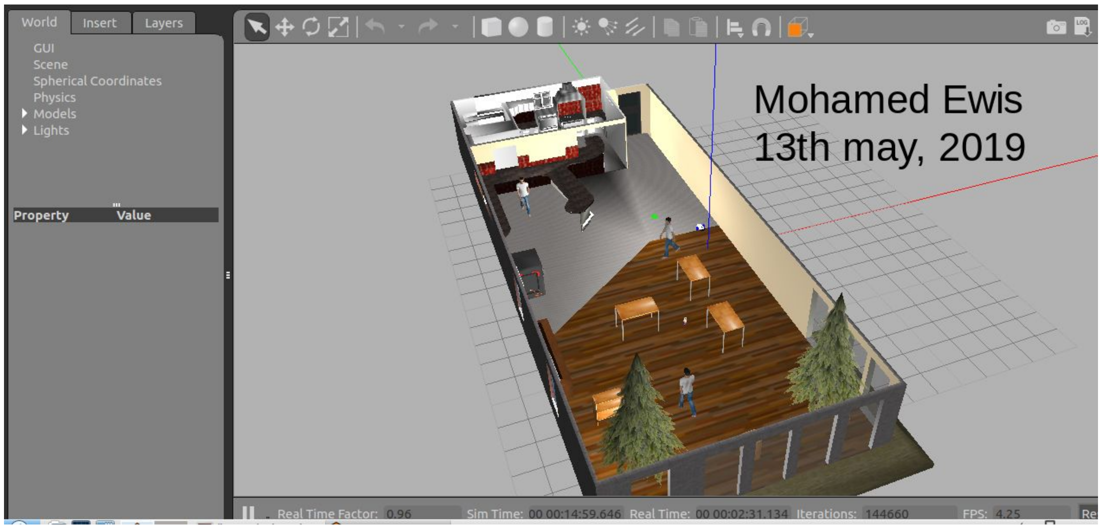


Fig. 2. ewis_cafe World

2 Background:

SLAM algorithms generally fall into 5 categories:

1. Extended Kalman Filter SLAM (EKF)
2. Sparse Extended Information Filter (SEIF)
3. Extended Information Form (EIF)
4. FastSLAM
5. GraphSLAM

The two most useful approaches to SLAM are Grid based FastSLAM and GraphSLAM.

2.1 Grid based FastSLAM:

The FastSLAM algorithm uses a custom particle filter approach to solve the full SLAM problem with known correspondence. Using particles, FastSLAM estimates a posterior over the robot's path along with the map. Each of these particles hold the robot's trajectory which gives an advantage to SLAM to solve the problem of mapping with known poses. In addition to the trajectory, each particle holds a map and each feature of the map is represented by a local Gaussian.
With the FastSLAM algorithm, the problem is now divided into two separate independent problems, each of which aims to solve the problem of estimating features of the map. To solve these independent mini-problems, FastSLAM will use the low dimensional Extended Kalman Filter. While map features are treated independently, dependency only exists between robot pose uncertainty. This custom approach of representing the posterior with particle filter and Gaussian is known by Rao-Blackwellized particle filter approach. The Grid based FastSLAM is an extension of FastSLAM and it adapts FastSLAM to grid maps.
With grid mapping algorithm, the environment can be modeled using grid maps without predefining any landmark position. So by extending the FastSLAM algorithm to occupancy grid maps, the SLAM problem can now be solved in an arbitrary environment. While mapping the real world environment, mobile robots equipped with range sensors can be used and the FastSLAM algorithm can be extended to solve the SLAM problem in terms of grid maps.

2.1.1 Grid based FastSLAM techniques:
To adapt FastSLAM to grid mapping, we need three different techniques:

1. **Sampling Motion-** $p(xt \mid xt-1[k], ut)$**:** Estimates the current pose given the k-th particle previous pose and the current controls u.
2. **Map Estimation-** $p(mt \mid zt, xt[k], mt-1[k])$**:** Estimates the current map given the current measurements, the current k-th particle pose, and the previous k-th particle map.
3. **Importance Weight-** $p(zt \mid xt[k], m[k])$**:** Estimates the current likelihood of the measurement given the current k-th particle pose and the current k-th particle map.

The sampling motion, map estimation and importance weight techniques are the essence of the grid based FastSLAM algorithm. Grid based FastSLAM implements them to estimate both the map and the robot's trajectory, given the measurements and the control.

2.2 Graph SLAM:
Graph SLAM is a SLAM algorithm that solves the full SLAM problem. This means that the algorithm recovers the entire path and map, instead of just the recent pose and map. This difference allows it to consider dependencies between current and previous poses. One of the benefits of graph SLAM is the reduced need for significant on-board processing capability. Another is graph SLAM's increased accuracy over FastSLAM. FastSLAM uses particles to estimate the robot's most likely pose. However, at any point in time, it is possible that there is not a particle in the most likely location. In fact, chances are slim to none especially in large environments. Since graph SLAM solves the full SLAM problem, this means that it can work with all of the data at once to find the optimal solution.

2.2.1 Front End vs Back End:
The front end of graph SLAM looks at how to construct the graph, using the odometry and sensory measurements collected by the robot. This includes interpreting sensory data, creating the graph and continuing to add nodes and edges to it as the robot traverses the environment. The front end of graph SLAM also has the challenge of solving the data association problem. In simpler terms, this means accurately identifying whether features in the environment have been previously seen.
The back end of graph SLAM is where the magic happens. The input to the back end is the completed graph with all of the constraints and the output is the most probable configuration of robot poses and map features. The back end is an optimization process that takes all of the constraints and find the system configuration that produces the smallest error.
The front end and the back end can be completed in succession or can be performed iteratively, with a back end feeding an updated graph to the front end for further processing.

2.2.2 Using RTAB-Map for 3D Graph SLAM:

RTAB-Map (Real Time Appearance Based Mapping) is a graph based SLAM approach. Appearance based SLAM means that the algorithm uses data collected from vision sensors to localize the robot and map the environment. In appearance based methods, a process called Loop Closure is used to determine whether the robot has seen a location before. As the robot travels to new areas in its environment, the map is expanded and the number of images that each new image must be compared to increases. This causes the loop closure to take longer with the complexity increasing linearly. RTAB-Map is optimized for large scale and long term SLAM by using multiple strategies to allow for loop closure to be done in real time.
Figure 3 shows the block diagram of the front end and the back end.

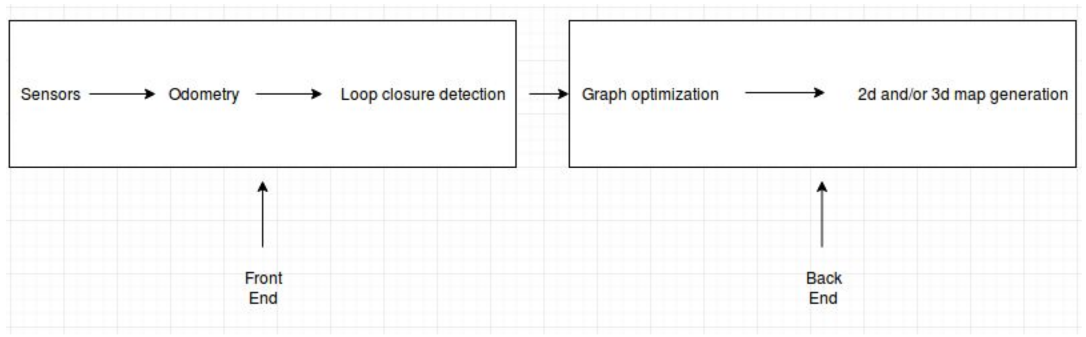


Fig. 3. RTAB-Map Front end and Back End block diagram

- The front end of RTAB-Map focuses on sensor data used to obtain the constraints that are used for feature optimization approaches. Only odometry constraints and loop closure constraints are considered here. The front end also involves graph management, which includes node creation and loop closure detection using Visual Bag of Words.

- The back end of RTAB-Map includes graph optimization and assembly of an occupancy grid from the data of the graph. Loop closure detection is the process of finding a match between the current and previously visited locations in SLAM.

  There are two types of loop closure detection:

1. Local Loop closure detection: in this approach, matches are found between a new observation and a limited map region. The size and location of this limited map region is determined by the uncertainty associated with the robot's position. This type of approach fails if the estimated position is incorrect.
2. Global Loop closure detection: in this approach, a new location is compared with the previously visited locations. If no match is found, the new location is added to memory. As the map grows and more locations are added to the memory, the amount of time to check whether the location has been previously seen increases linearly. If the time it takes to search and compare new images to the one stored in memory becomes larger than the acquisition time, the map becomes ineffective.

3 Scene and Robot Configurations:

In order for the robot to perform SLAM, it must have a World to perform it in. For this reason, two environments will be used to test and simulate a robot performing SLAM and to generate 2D and 3D maps from them. The first environment is an environment provided by Udacity and is named Kitchen-Dining. The second environment is a custom made environment of a cafeteria named ewis_cafe.

3.1 Robot model:

The robot model used was based on the one created in the previous project as the student robot model (which had a square base with two actuators for the left and right wheels). The camera was removed and replaced with a kinect leveraging the openni_camera ros package with the gazebo controller Openni Kinect.

No changes were made to the hokuyo laser range finder.

An additional joint was added to rotate the kinect data 180%. It was positioned on the front of the robot so as to not interfere with the laser range finder.

The transform tree associated with the robot is shown in Figure 4.

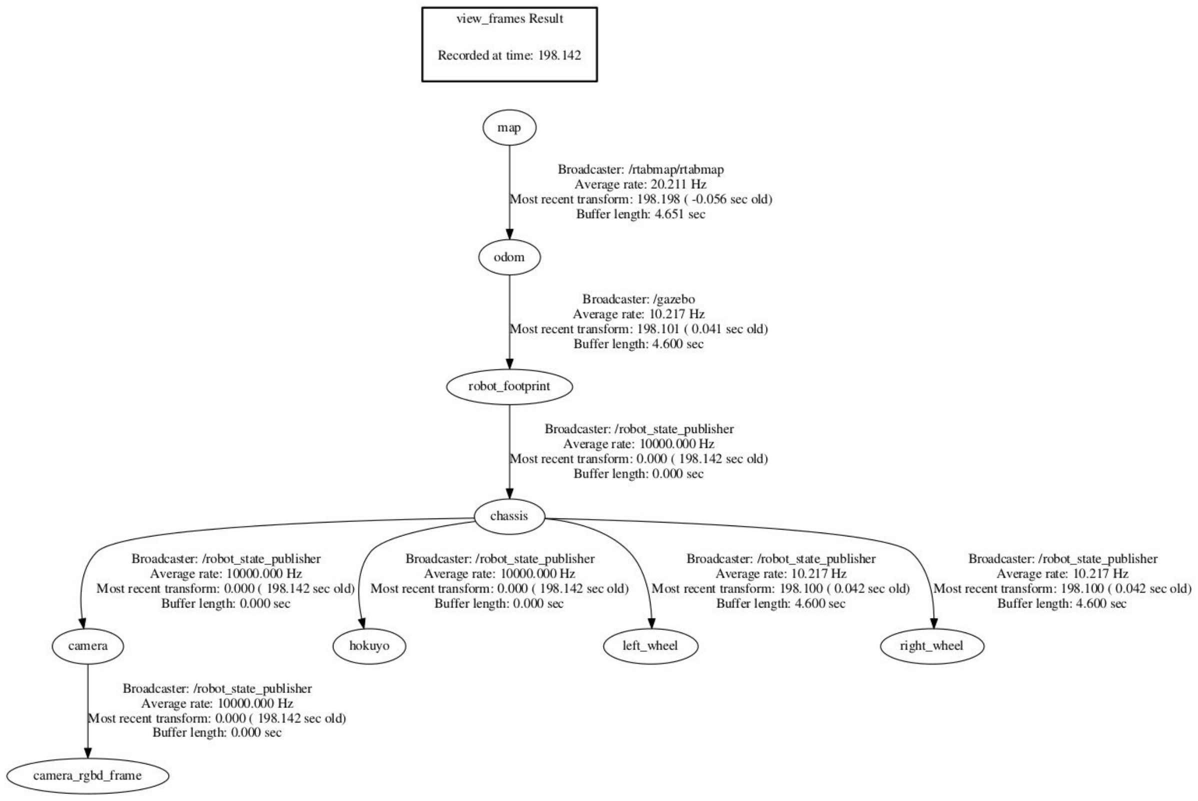

view_frames Result
Recorded at time: 198.142
map
Broadcaster: /rtabmap/rtabmap
Average rate: 20.211 Hz
Most recent transform: 198.198 (-0.056 sec old)
Buffer length: 4.651 sec
odom
Broadcaster: /gazebo
Average rate: 10.217 Hz
Most recent transform: 198.101 (0.041 sec old)
Buffer length: 4.600 sec
robot_footprint
Broadcaster: /robot_state_publisher
Average rate: 10000.000 Hz
Most recent transform: 0.000 (198.142 sec old)
Buffer length: 0.000 sec
chassis
Broadcaster: /robot_state_publisher
Average rate: 10000.000 Hz
Most recent transform: 0.000 (198.142 sec old)
Buffer length: 0.000 sec
Broadcaster: /robot_state_publisher
Average rate: 10000.000 Hz
Most recent transform: 0.000 (198.142 sec old)
Buffer length: 0.000 sec
Broadcaster: /robot_state_publisher
Average rate: 10.217 Hz
Most recent transform: 198.100 (0.042 sec old)
Buffer length: 4.600 sec
Broadcaster: /robot_state_publisher
Average rate: 10.217 Hz
Most recent transform: 198.100 (0.042 sec old)
Buffer length: 4.600 sec
camera
hokuyo
left_wheel
right_wheel
Broadcaster: /robot_state_publisher
Average rate: 10000.000 Hz
Most recent transform: 0.000 (198.142 sec old)
Buffer length: 0.000 sec
camera_rgbd_frame

Fig. 4. Transform Tree of the Robot - ewis_bot

3.2 Design of the World:
The custom world is based on the cafe model inside Gazebo database. The base model is customized with different objects like tables, beer can, people, trees etc. These objects serve as distinctive elements in the base world for the robot to distinguish and map. In this world, the kitchen cannot be entered by the robot. A top view of this world is provided in Figure 5.

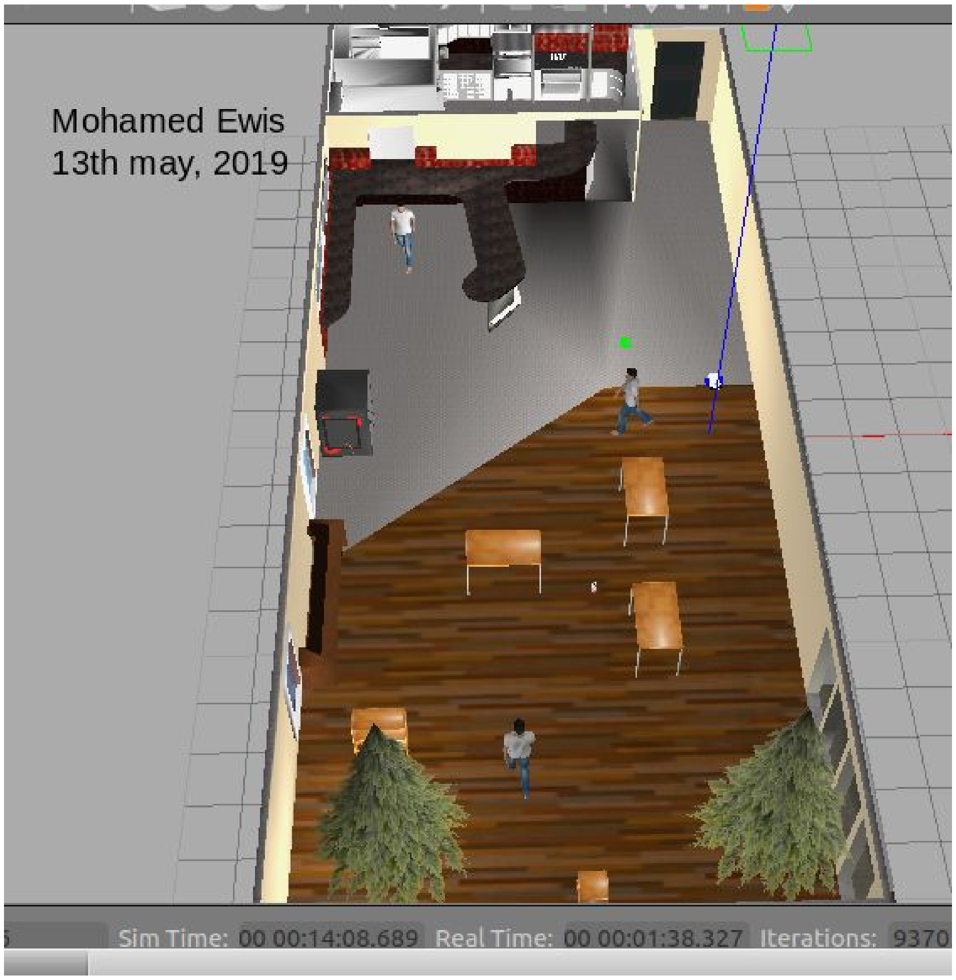


Fig. 5. top view of ewis_cafe.

3.3 Launch file configuration:
Four launch files are required for a successful mapping of the environments in simulation:

1. The gazebo simulation environment is (kitchen-dining or cafe) specified in the world.launch file.
2. The teleop.launch file launches the teleop keyboard which is required for moving the robot in the simulation world.

3. The mapping.launch file is used to start the RTAB-Map node. This node is used for loop closure detection using the ORB-SLAM algorithm. ORB-SLAM is a versatile and accurate Monocular SLAM solution able to compute in real-time the camera trajectory and a sparse 3D reconstruction of the scene in a wide variety of environments, ranging from small hand-held sequences to a car driven around several city blocks. It is able to close large loops and perform global re-localization in real-time and from wide baselines.
4. Finally, the rviz.launch file starts visualization of the rover, sensor data, as well as map and camera topics in RViz. Figure 6 depicts RViz view of the world at the starting point.

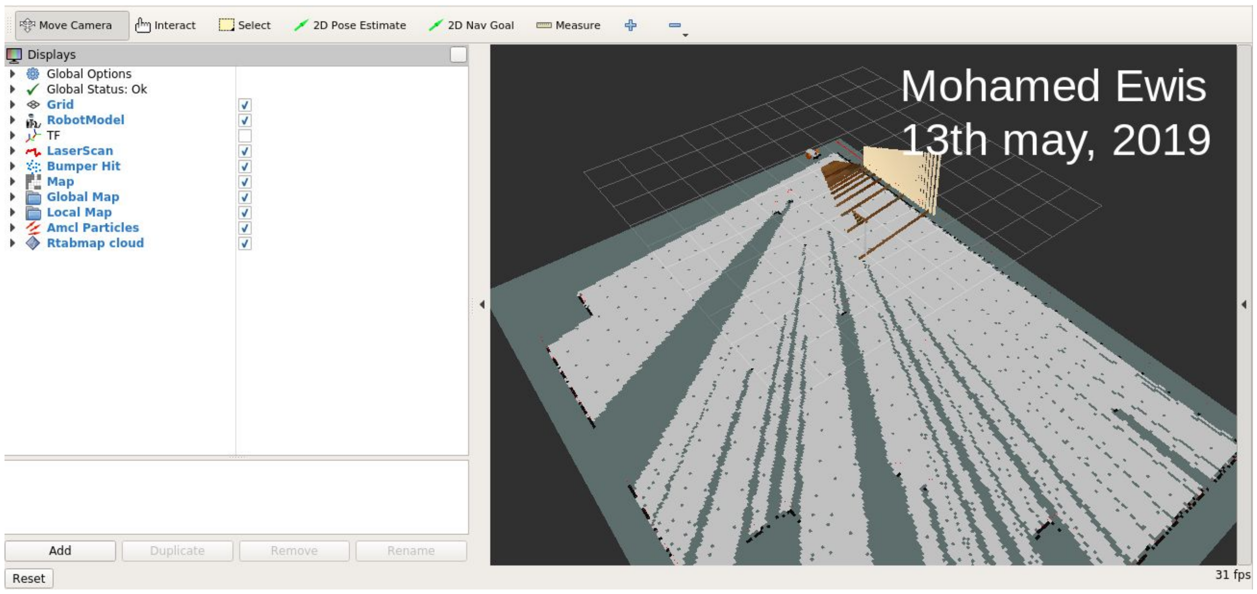


Fig. 6. Starting point of the robot as seen in RViz

During the mapping of the environment, the mapping data is saved in the rtabmap.db database. The localization. launch file can be started in order to localize the robot during the run.

4 Results:

The mapping was done by the robot controlled by the teleop keyboard. In order to be able to have more than 3 loop closure detection, which was the project's benchmark, the robot was navigated through the full environment of both the worlds so that it could collect more images.

4.1 Kitchen-Dining World:

The mapping run in the provided world ended with 66 global loop closures. This file has a size of 256 MB and is named as rtabmap_kitchen_dining.db. Figure 7 shows the robot's trajectory as well as the 2d occupancy grid map of the kitchen dining world.

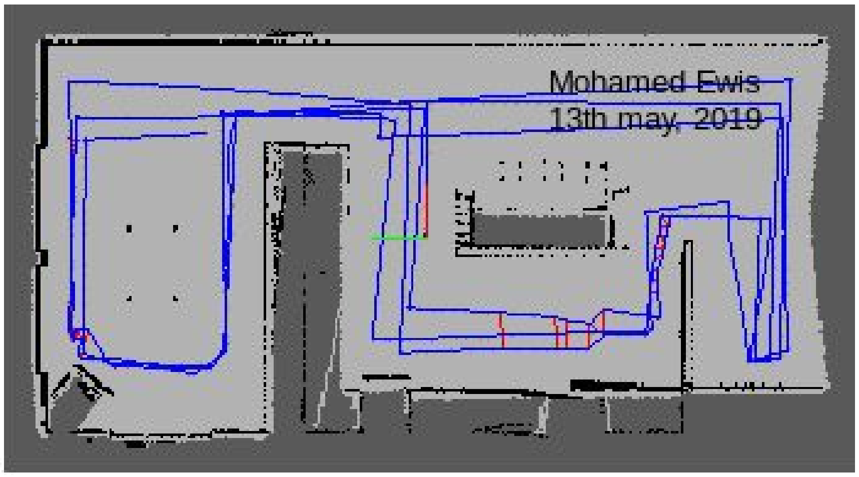


Fig. 7. Robot's trajectory and 2d occupancy grid map of the Kitchen-Dining world

At the end of the multiple passes, a well structured 3d point cloud map was created by using the Export 3d Clouds functionality. Figure 8 depicts the reconstructed point cloud data. It can be seen that most features in the world like the chairs and tables are reconstructed properly and are distinctive. Figure 9 shows the RViz result of the same world after the end of the mapping task.

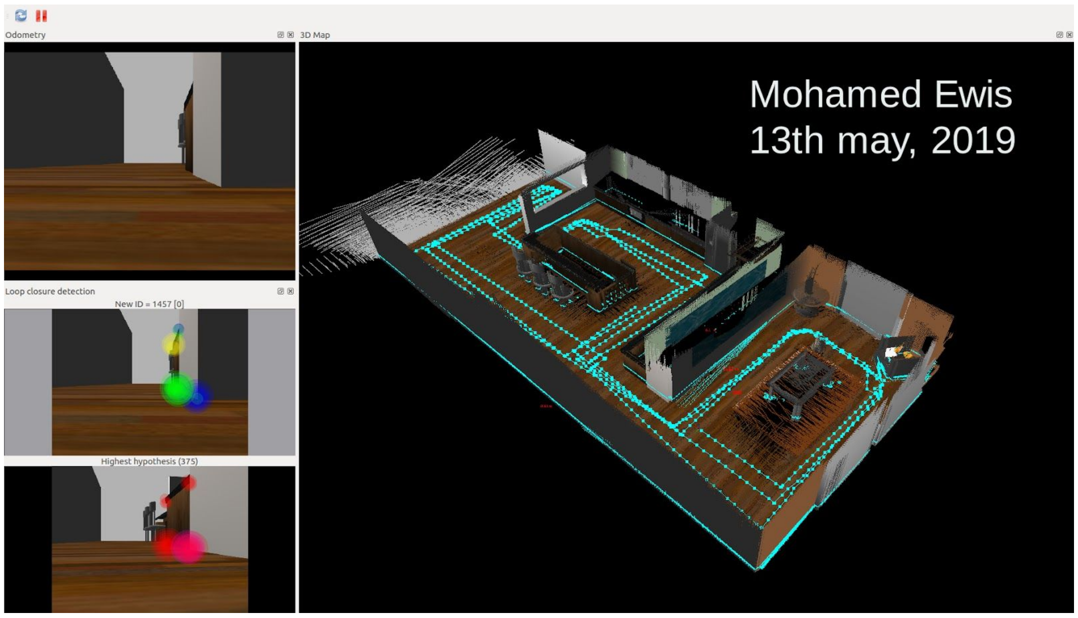
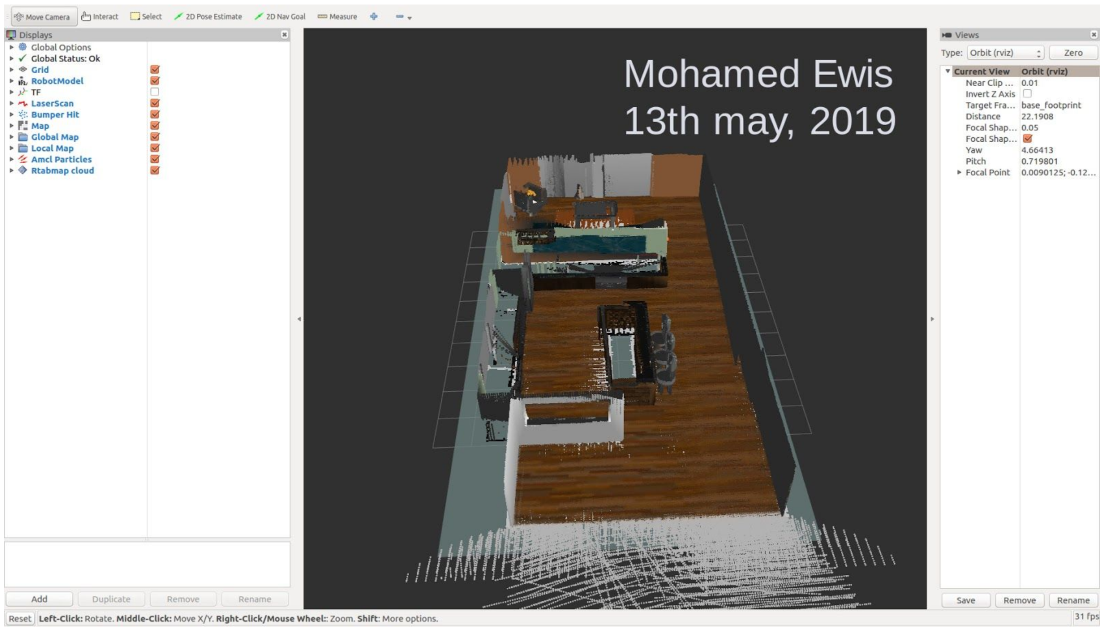


Fig. 8. Reconstructed Point cloud data in RTAB-Map viewer of the Kitchen-Dining world



Displays	
Global Options	
✓ Global Status: Ok	
Grid	✓
RobotModel	✓
TF	☐
LaserScan	✓
Bumper Hit	✓
Map	✓
Global Map	✓
Local Map	✓
Amcl Particles	✓
Rtabmap cloud	✓

Views

Type: Orbit (rviz) Zero

Current View	Orbit (rviz)
Near Clip ...	0.01
Invert Z Axis	☐
Target Fra...	base_footprint
Distance	22.1908
Focal Shap...	0.05
Focal Shap...	✓
Yaw	4.66413
Pitch	0.719801
Focal Point	0.0090125; -0.12...

Fig. 9. Rviz view of the Kitchen-dining world

At the end of the map, the loop closures can be seen in the rtabmap kitchen dining.db. Figure 10 shows one of them.

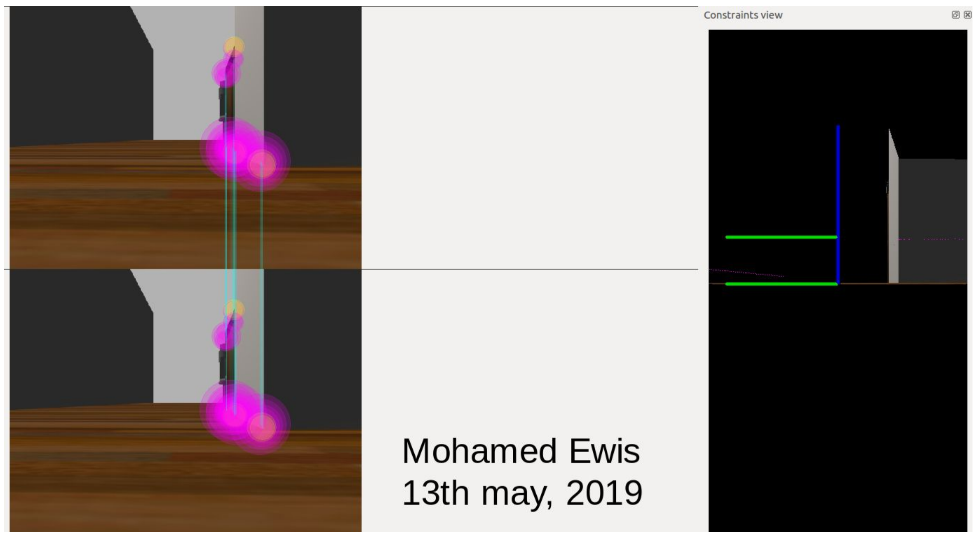


Fig. 10. Loop Closure detection

4.2 ewis_Cafe World:
In the custom made world, the robot performed well. As the robot is very short, some of the taller objects like the people, trees are not fully mapped. The kitchen also could not be traveled by the robot.This file has a size of 118 MB and is named as rtabmap.db .
Figure 11 and 12 depicts the RTAB-map view and the RViz view of this world respectively at the end of the mapping.

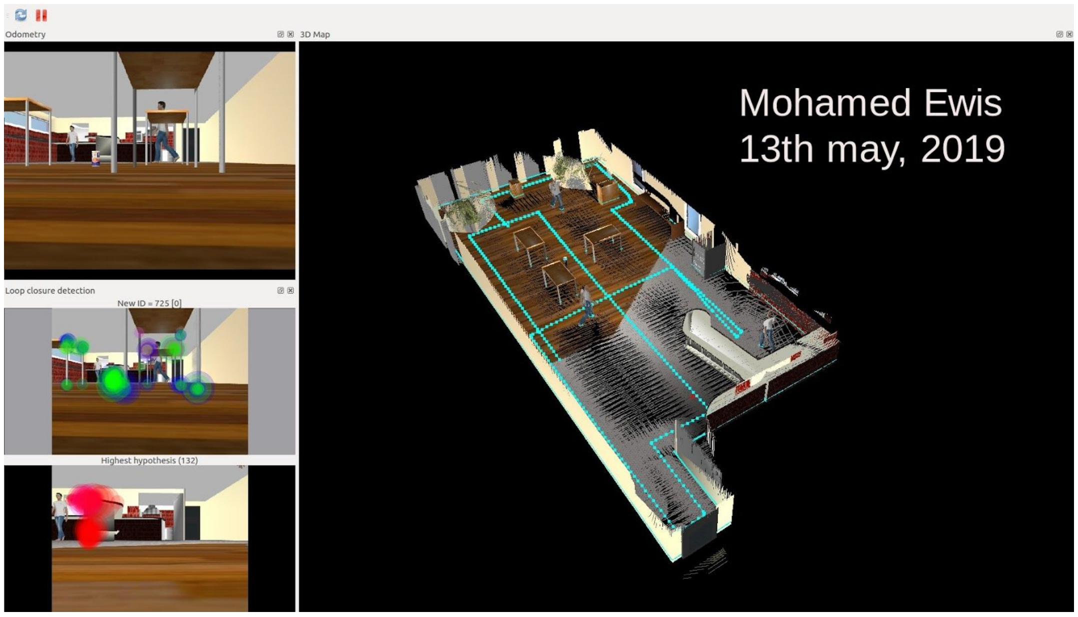
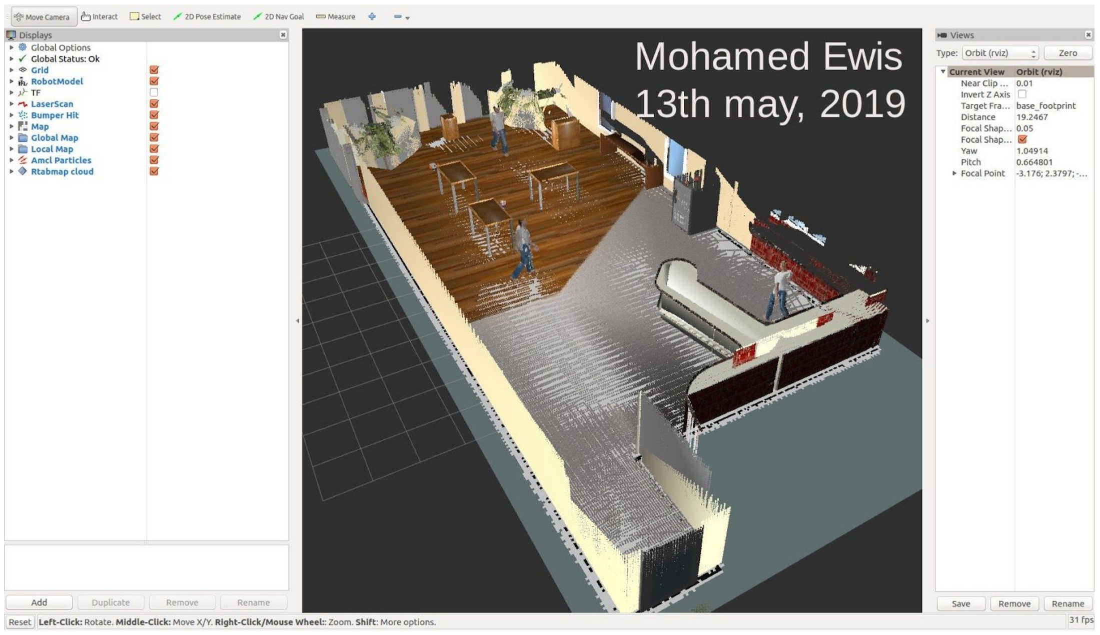


Fig. 11. Reconstructed Point cloud data in RTAB-Map viewer in the ewis_cafe world



Displays

Global Options	
Global Status: Ok	
Grid	✓
RobotModel	✓
TF	☐
LaserScan	✓
Bumper Hit	✓
Map	✓
Global Map	✓
Local Map	✓
Amcl Particles	✓
Rtabmap cloud	✓

Views

Type: Orbit (rviz)

Current View	Orbit (rviz)
Near Clip ...	0.01
Invert Z Axis	☐
Target Fra...	base_footprint
Distance	19.2467
Focal Shap...	0.05
Focal Shap...	✓
Yaw	1.04914
Pitch	0.664801
Focal Point	-3.176; 2.3797; -...

Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel:: Zoom. Shift: More options.

31 fps

Fig. 12. RViz view of the ewis_cafe world

5 Discussion:

In both the environments, successful mapping was performed in order to identify the ground truth and the distinctive features of the environment. In the ewis_cafe world, the robot could not enter the kitchen area, which can be seen in right lower corner of Figure 11 and Figure 12. One of the possible explanations might be that there is a height difference of the floor between the living room and the kitchen. Another possible explanation might be that there is a transparent door separating those two rooms and hence the robot couldn't pass through. The generated 2d and 3d maps can be improved by doing more mapping runs which cover the environment in a more complete manner and by optimizing the loop closure detection further.

6 Conclusion/Future Work:

An interesting future work would be to explore the RTABMap package's visualization section in more details. The obstacle detection feature can be deployed in order to extracts obstacles and the ground from your point cloud. With this information in hand, these obstacles can be avoided when executing a desired path.