Map My World

Udacity Robotics Software Engineering Project; Gene Foxwell 2018

ABSTRACT

Simultaneous Localization and Mapping (SLAM) is a common problem in the Robotics fields. In this project we looked at implementing a solution for this problem in simulation using the RTABMAP package. A simulated robot was equipped with a Microsoft Kinnect RGB-D camera and driven around two distinct environments with the goal of creating 2D and 3D maps of those environments. These maps where then evaluated using the rtabmap-databaseViewer to ensure that they provided a reasonable representation of their corresponding environments.

INTRODUCTION

In this project we looked at generating 3D maps of environments using an ROS SLAM (Simultaneous Localization and Mapping) package known as RTAB Map. (Real Time Appearance Based Mapping). To do this we used modified version of the robot that was used for the “Where am I” project that was previously done. (In this case the student created robot was used as opposed to the one that was provided by Udacity for that project). This robot was given a simulated version of a Microsoft Kinect RGB-D sensor and driven around each environment manually. Final maps where then extracted using the rtabmap-databaseViewer application.

Two environments where used to test the systems ability to map its environment. The first, provided by the Udacity course materials, was a model of a Kitchen / Dinning room. The second, created specifically for this project was an outdoor environment roughly simulating a school playground. Both environments where simulated with the Gazebo physics simulator provided with ROS. Top down images of both environments can be seen in Figure 1 and Figure 2 respectively.

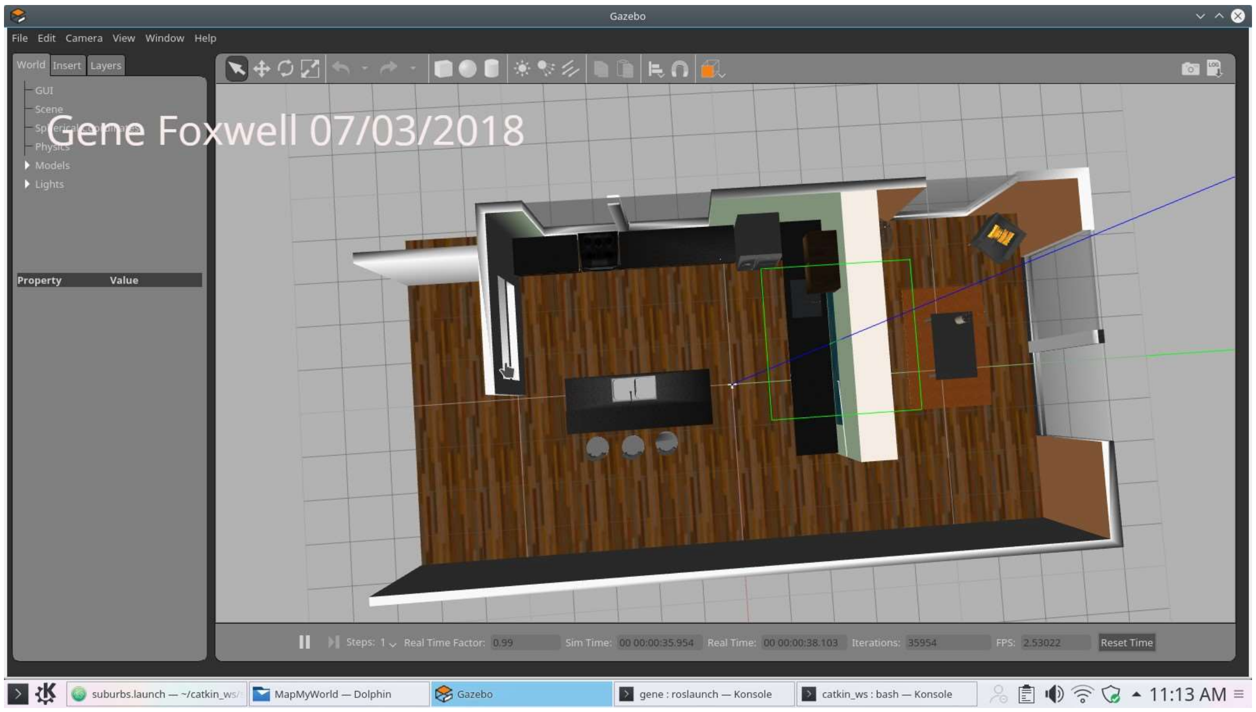
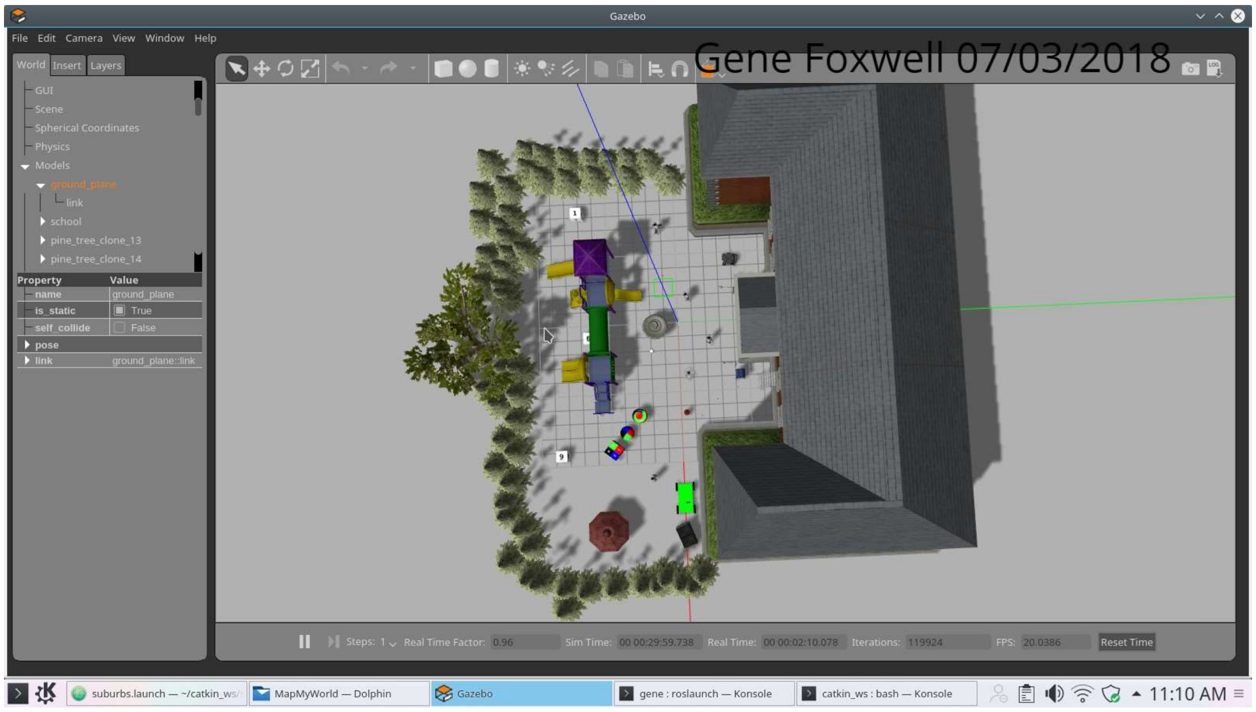

Gazebo
File Edit Camera View Window Help
World Insert Layers
GUI
Scene
Spherical Coordinates
Physics
Models
Lights
Property Value
Gene Foxwell 07/03/2018
Steps: 1 Real Time Factor: 0.99 Sim Time: 00 00:00:35.954 Real Time: 00 00:00:38.103 Iterations: 35954 FPS: 2.53022 Reset Time
suburbs.launch — ~/catkin_ws/
MapMyWorld — Dolphin
Gazebo
gene : roslaunch — Konsole
catkin_ws : bash — Konsole
11:13 AM

Figure 1 Kitchen / Dining Room Environment


Gazebo
File Edit Camera View Window Help
World Insert Layers
GUI
Scene
Spherical Coordinates
Physics
Models
ground_plane
link
school
pine_tree_clone_13
pine_tree_clone_14
Gene Foxwell 07/03/2018
Steps: 1 Real Time Factor: 0.96 Sim Time: 00 00:29:59.738 Real Time: 00 00:02:10.078 Iterations: 119924 FPS: 20.0386 Reset Time
suburbs.launch — ~/catkin_ws/
MapMyWorld — Dolphin
Gazebo
gene : roslaunch — Konsole
catkin_ws : bash — Konsole
11:10 AM

Property	Value
name	ground_plane
is_static	True
self_collide	False
pose	
link	ground_plane::link

Figure 2 Suburban School Map

As can be seen in Figure 1, the Kitchen Dining room environment is split into three sections. The robot starts in the kitchen area, which includes some standard features for a North American kitchen (Fridge, Stove, counter space) along with a small island counter "floating" in the middle. There is an empty room on one side of the kitchen and a simple dining room with a small dining room table in the middle.

Figure 2 shows the suburban school yard. As this was intended to simulate an outside environment (a choice made to provide some meaningful contrast to the system performance in an indoor environment as would bee seen in the kitchen/dinning world environment) it is far less structured. Toys such as cricket balls have been left laying around, there is a large play structure in the middle and people have been placed (standing stationary) at various points around the yard. Trees have been used to mark the boundaries of this world to further give this environment and outdoor feel.

Our goal for each environment was to create reasonably accurate 3D and 2D maps using the RTAB Map package in combination with our robot. Examples of the resulting maps can be found in the Results section of this report.

BACKGROUND

Our goal in this project was to create an ROS based solution for the SLAM problem. In the SLAM problem we have no prior knowledge of the underlying map of the environment and as such are required to build this map as we go along, while keeping track of the robot's location relative to the known map. This has numerous applications in the real world, for example generating maps of Caves / mines in areas where humans cannot readily (or safely) go, creating maps of other planets / asteroids with a rover or other robotic exploration vehicle, and generating 3D models of homes or cities for marketing and / or commercial purposes.

There are several different ways to approach the SLAM problem. We will examine two such approaches here. First, we will look at Grid-Based FastSLAM, then we will compare it to RTAB Map which is the package we have used for our robot.

Grid-Based Fast SLAM[1] combines the Monte-Carlo based localization techniques covered in the "Where am I?" project of this course with an Occupancy Grid. The idea here conceptually very similar to MCL – we generate many particles to represent possible locations for the robot (just like MCL). We then associate with each particle an "occupancy grid" meant to represent the robots current map of the space. As we move each robot we update the localization in a similar manner as with MCL, but with an additional term multiplied in for how probable the particles associated map is based on the current measurements.

An Occupancy Grid divides the potential map into a set of cells arranged in a grid. Each cell represents the probability that it is "occupied". A cell that is occupied with 100% probability would have a value of one, and a cell for which there is a 0% chance of being occupied would have a value of zero. Any cell which had not yet been encountered by the Robot would be marked as unknown.

An example of an occupancy grid can be seen in Figure 3. In this image the robot is currently occupying the cell marked in green. Grey cells represent grid cells which have not yet been sensed by the robot's

[1] http://www2.informatik.uni-freiburg.de/~stachnis/pdf/grisetti05icra.pdf

sensors, white cells represent areas that are believed to be unoccupied (or for which the sensors did not detect any objects) and the black cells are considered occupied.

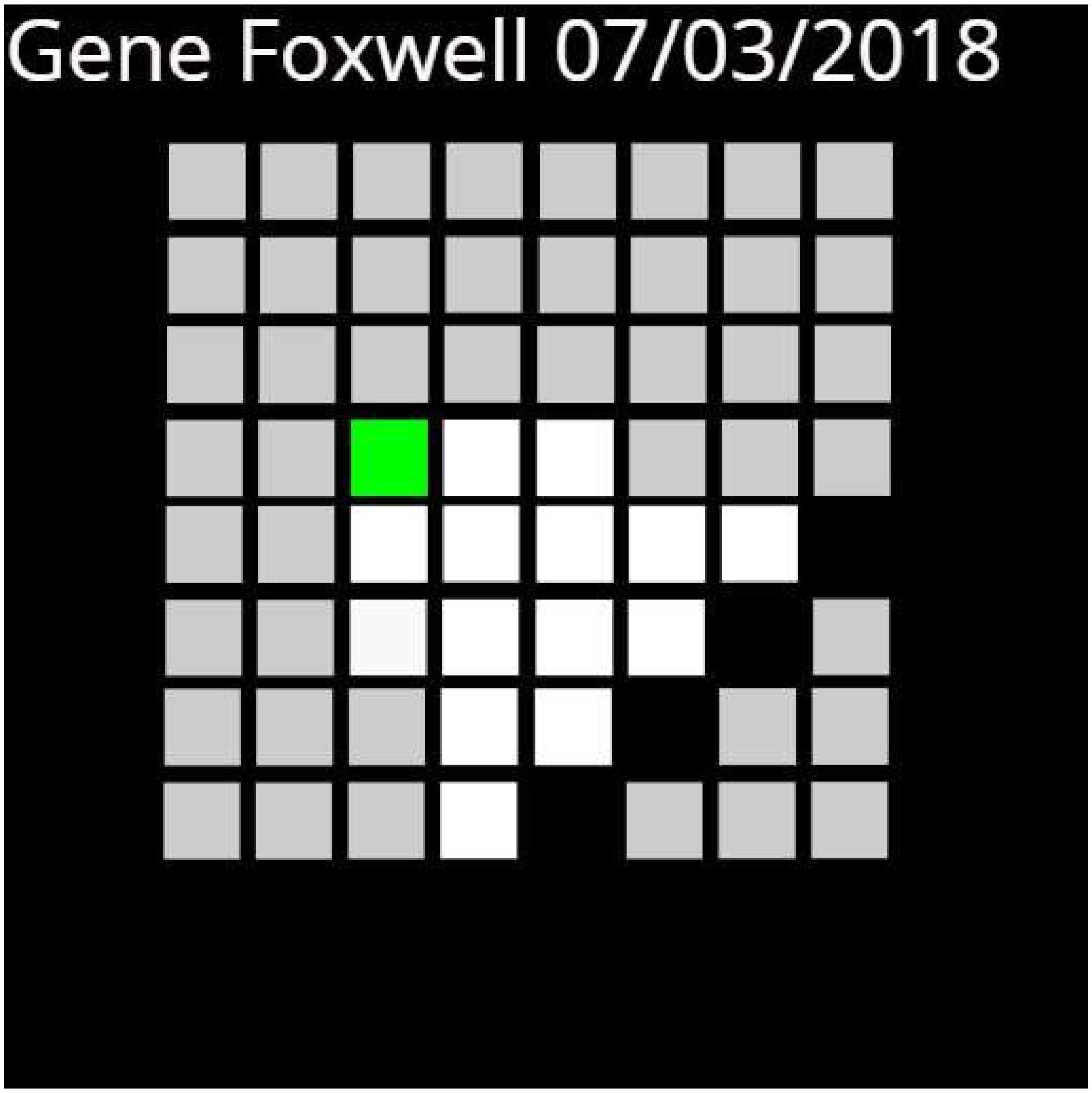


Figure 3 Grid Based Fast SLAM

A downside of this approach is that we need to track a lot of particles, each of which tracks its own copy of the map. Although adaptive sampling techniques can mitigate this issue, if we are required to map an especially large area then storing ten to a few hundred copies of the map could become severely limiting.

RTAB Map[2] is a variant of GraphSLAM which uses the incoming camera images to identify unique positions on the map. At a high level, RTAB Map works as follows:

[2] https://introlab.3it.usherbrooke.ca/mediawiki-introlab/images/b/bc/TRO2013.pdf

- Start with an initial link L0 representing the robots starting position.
- Robot moves forward until an image is collected.
- A new location object is created
- SURF[3] is extracted from the image and used to generate a signature.
- This signature is compared the signature of the previously created location.
- If the two locations are highly similar, then the new location is merged with the previous one and the process starts again.
- Otherwise a Bayesian filter is used to check for "loop closures" to see if this node has been visited in past.
- If such a location exists, a loop closure link is generated between the previous location and the new one.
- The graph generated by this procedure is then optimized and used to minimize the error between the generated occupancy maps and the real-world map.

Before continuing with this explanation, it remains to clarify the notion of "loop closures". When searching for loop closures we are looking to see if the current location has been visited in the past with high probability. If has been then this is a good indication that the robot has travelled in a "loop" or cycle through the environment and that we should associate the current location with the previously visited location.

If this was implemented naively, the number of nodes on the map could quickly spiral out of control. To resolve this issue RTAB Map uses a book keeping method that splits the locations between a "Short Term Memory" (STM), "Working Memory" (WM), and a "Long Term Memory" (LTM). Nodes in the STM are used to represent the robots local space with older nodes being transferred to the WM. These nodes are not used for checking loop closures. Nodes are moved from the STM into the WM when the size of the STM exceeds a pre-determined size based on the rate of new locations being created and the robot's velocity.

Nodes in the WM are used for evaluating loop closures. The size of WM is determined by how it takes to evaluate it for loop closures, when the time it takes to search the WM exceeds a given threshold the least visited nodes in WM are removed from the WM and placed into the LTM.

Nodes in the LTM are essentially kept in storage until a nearby node is moved into the WM. In these cases, nodes are pulled from the LTM and back into the WM to make loop detection easier.

Book keeping in this way allows RTAB Map to detect previously visited nodes without having to laboriously search through the entire history of the robot's movement. This method turns out to be a sufficient speed up to allow the robot to update its map in real time. (Hence the name Real-Time Appearance Based Mapping).

A diagram representing this process can be seen in Figure 4. In this diagram we start from location L1 and move along until we reach location L11 (likely merging many similar locations together in the process). At location L11 a loop closure is detected between L11 and L2 and as such a new link is created between those locations in the graph.

[3] https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_surf_intro/py_surf_intro.html

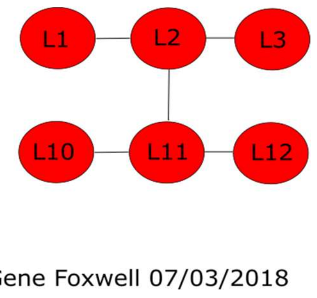


Figure 4 Graph SLAM

The two methods discussed here have their own strengths and weaknesses. Experimentation with RTAB Map has shown that it can sometimes be fooled by environments with a lot of similar features or with a lot of symmetry. In these cases, false positives may be detected that create loop closures where none should be. Environments with very little visual diversity may also cause problems for the RTAB Map method as there would be very little to differentiate one location from another. In these cases, a Grid Based method may perform better as here we are relying on a collection of particles each equipped with an occupancy grid. As these maps are based on the estimated location of the robot (and the probability of the particles corresponding maps) they are not susceptible to the failing mentioned above.

Grid Based Fast SLAM however does require a lot of particle each with its own occupancy grid attached. This can create issues with large maps as we would be required to store a lot of data in memory. This memory limitation limits the number of particles that we can track and thus limits the reliability of the method. RTAB Map on the other hand needs only store one copy of the graph and when combined with the book keeping methods outlined previously can update even a large map in real time. (According to the previously linked paper these maps can be up to 1000 km in size).

SCENE AND ROBOT CONFIGURATION

For this project we used a modified version of the robot that was created in the “Where am I?” project. The primary difference between the robot used in the “Where am I?” project and this one is the addition of an RGB-D camera. This camera was placed just above the counter weight at the front of the robot and rotated slightly to counter the incline of the robot. When the robot was first started up, it was found that the RGB-D camera was providing output with the depth information orthogonal to the direction of the map. To fix this a new link (camera_frame) was added which rotated the output of the camera so that the depth dimension could be correctly visualized. Marvin Bot’s joint configuration can be seen in Figure 5, while Figure 6 shows a close-up view of the Marvin Bot in action.

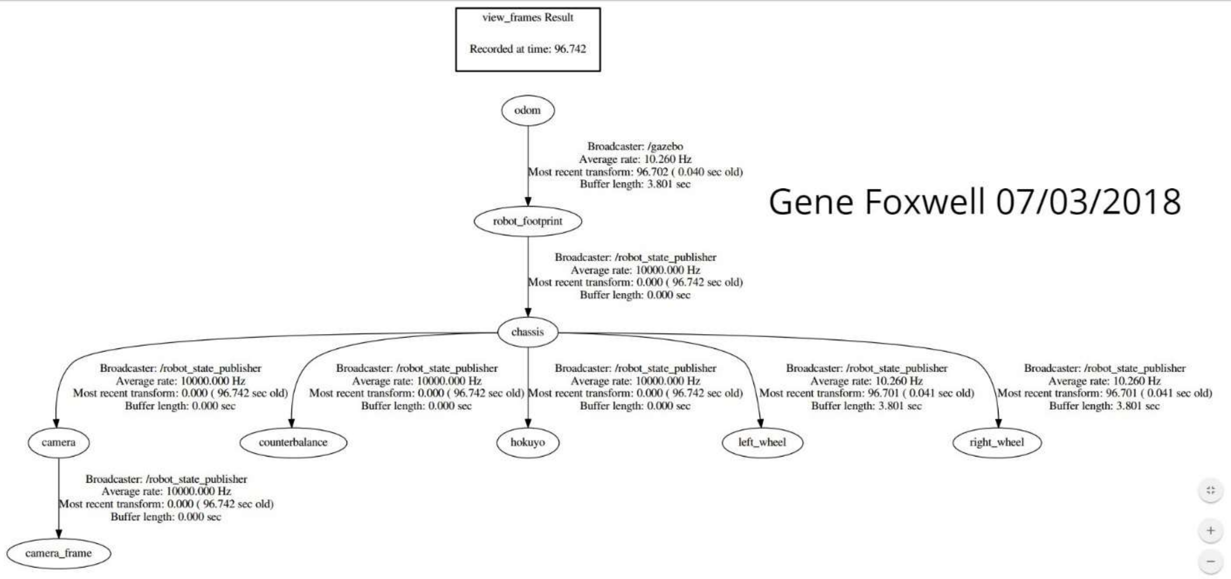

view_frames Result
Recorded at time: 96.742
odom
Broadcaster: /gazebo
Average rate: 10.260 Hz
Most recent transform: 96.702 (0.040 sec old)
Buffer length: 3.801 sec
Gene Foxwell 07/03/2018
robot_footprint
Broadcaster: /robot_state_publisher
Average rate: 10000.000 Hz
Most recent transform: 0.000 (96.742 sec old)
Buffer length: 0.000 sec
chassis
Broadcaster: /robot_state_publisher
Average rate: 10000.000 Hz
Most recent transform: 0.000 (96.742 sec old)
Buffer length: 0.000 sec
Broadcaster: /robot_state_publisher
Average rate: 10000.000 Hz
Most recent transform: 0.000 (96.742 sec old)
Buffer length: 0.000 sec
Broadcaster: /robot_state_publisher
Average rate: 10000.000 Hz
Most recent transform: 0.000 (96.742 sec old)
Buffer length: 0.000 sec
Broadcaster: /robot_state_publisher
Average rate: 10.260 Hz
Most recent transform: 96.701 (0.041 sec old)
Buffer length: 3.801 sec
Broadcaster: /robot_state_publisher
Average rate: 10.260 Hz
Most recent transform: 96.701 (0.041 sec old)
Buffer length: 3.801 sec
camera
counterbalance
hokuyo
left_wheel
right_wheel
Broadcaster: /robot_state_publisher
Average rate: 10000.000 Hz
Most recent transform: 0.000 (96.742 sec old)
Buffer length: 0.000 sec
camera_frame

Figure 5 Marvin Bot Configuration

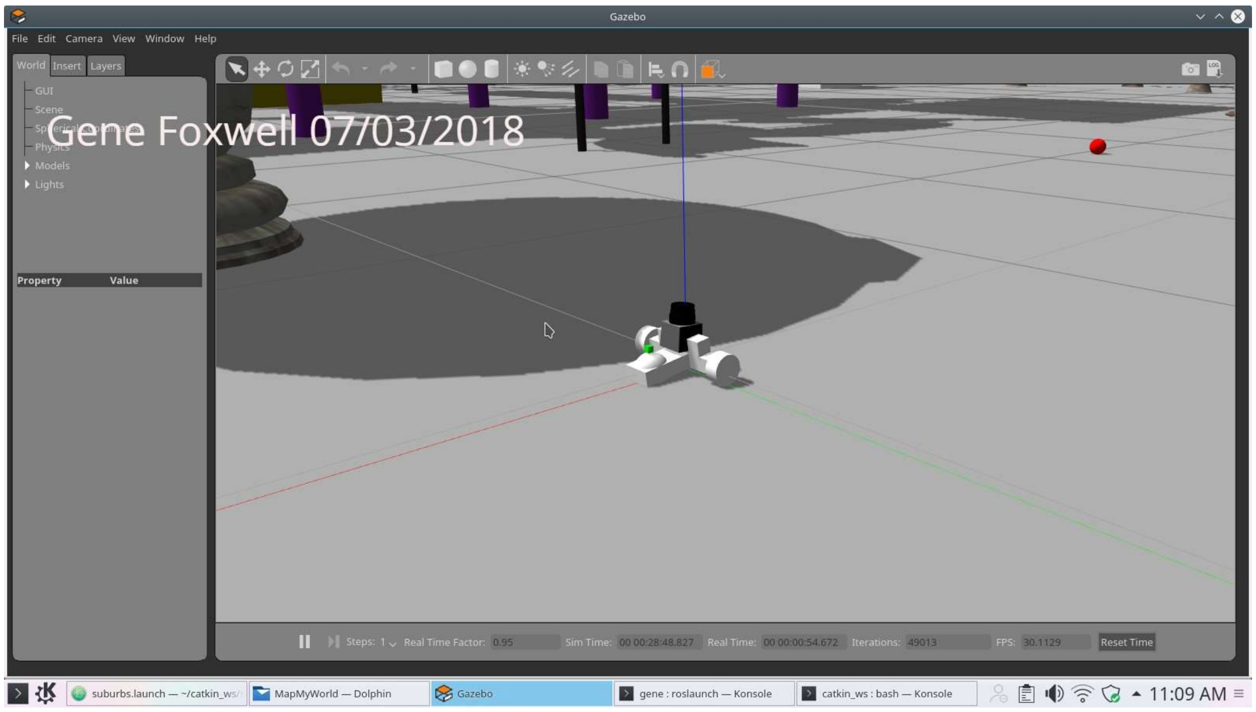

Gazebo
File Edit Camera View Window Help
World Insert Layers
GUI
Scene
Spherical Coordinates
Physics
Models
Lights
Gene Foxwell 07/03/2018
Property Value
Steps: 1 Real Time Factor: 0.95 Sim Time: 00 00:28:48.827 Real Time: 00 00:00:54.672 Iterations: 49013 FPS: 30.1129 Reset Time
suburbs.launch — ~/catkin_ws/
MapMyWorld — Dolphin
Gazebo
gene : roslaunch — Konsole
catkin_ws : bash — Konsole
11:09 AM

Figure 6 Marvin Bot Portrait

The Marvin Bot was tested against two simulated environments, one indoor and one outdoor. For the indoor environment a model provided by the Udacity course materials of a Kitchen / Dining room was used. For the outdoor environment a student create environment by combining freely available models from the Gazebo repository inside the Gazebo simulation environment. This environment is intended to be a rough approximation of a school playground. It should be noted that this model was left with no floor texture as there did not seem to be an appropriate option for a grass playground in the Gazebo set.

(The VRC Driving Terrain was tried but caused serious performance issues). Screenshots of the Kitchen / Dining Room and Suburban School Map can be seen in Figure's 7 and 8 respectively.

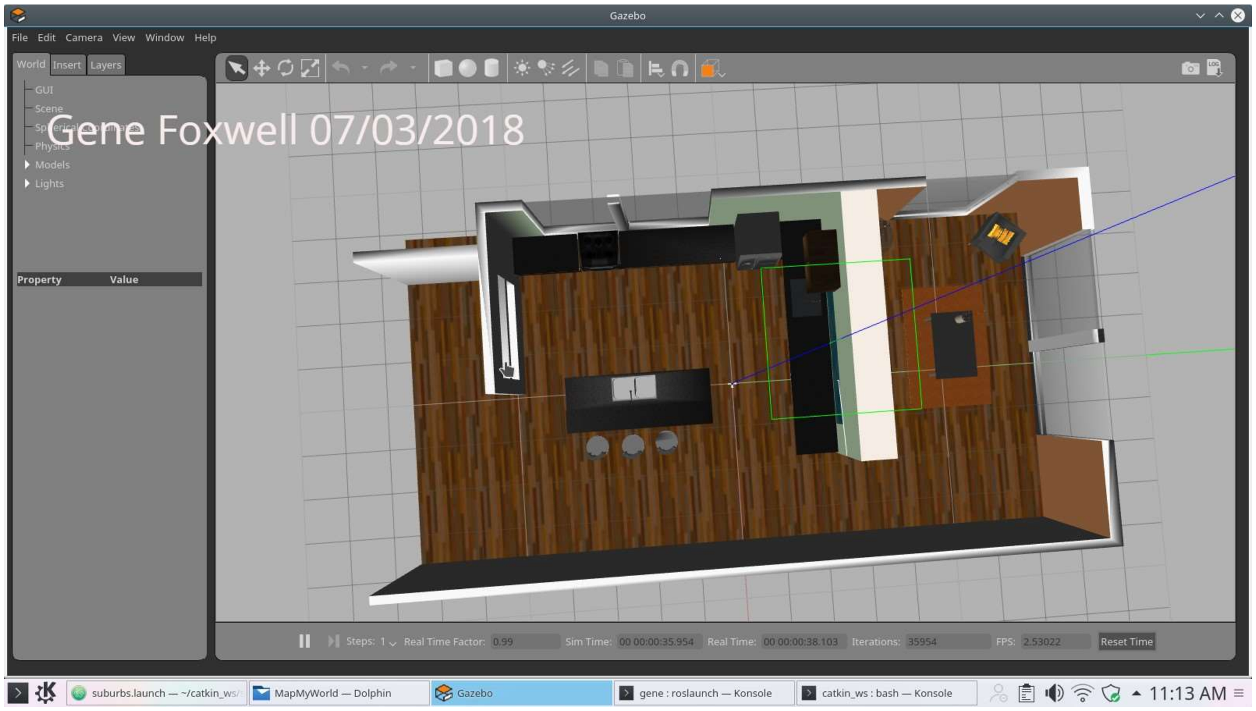
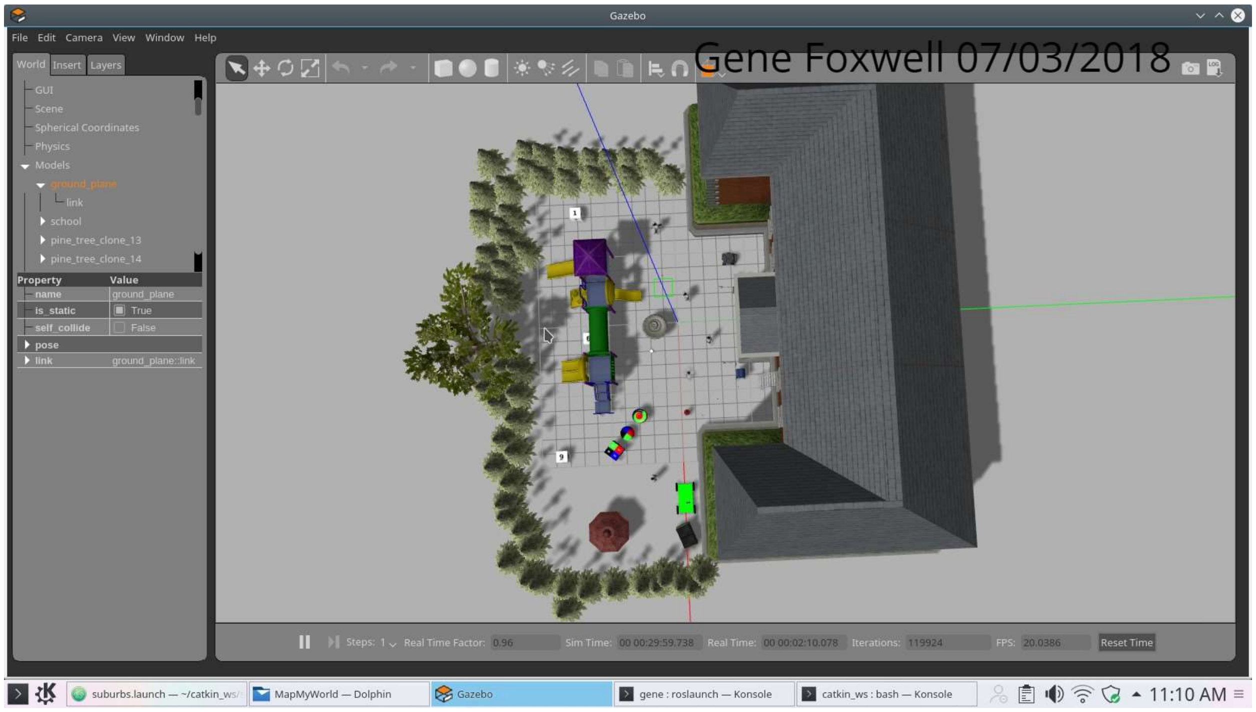

Gazebo
File Edit Camera View Window Help
World Insert Layers
GUI
Scene
Spherical Coordinates
Physics
Models
Lights
Property Value
Gene Foxwell 07/03/2018
Steps: 1 Real Time Factor: 0.99 Sim Time: 00 00:00:35.954 Real Time: 00 00:00:38.103 Iterations: 35954 FPS: 2.53022 Reset Time
suburbs.launch — ~/catkin_ws/
MapMyWorld — Dolphin
Gazebo
gene : roslaunch — Konsole
catkin_ws : bash — Konsole
11:13 AM

Figure 7 Top Down View of the Kitchen / Dining Room



Property	Value
name	ground_plane
is_static	☑ True
self_collide	☐ False
pose	
link	ground_plane::link

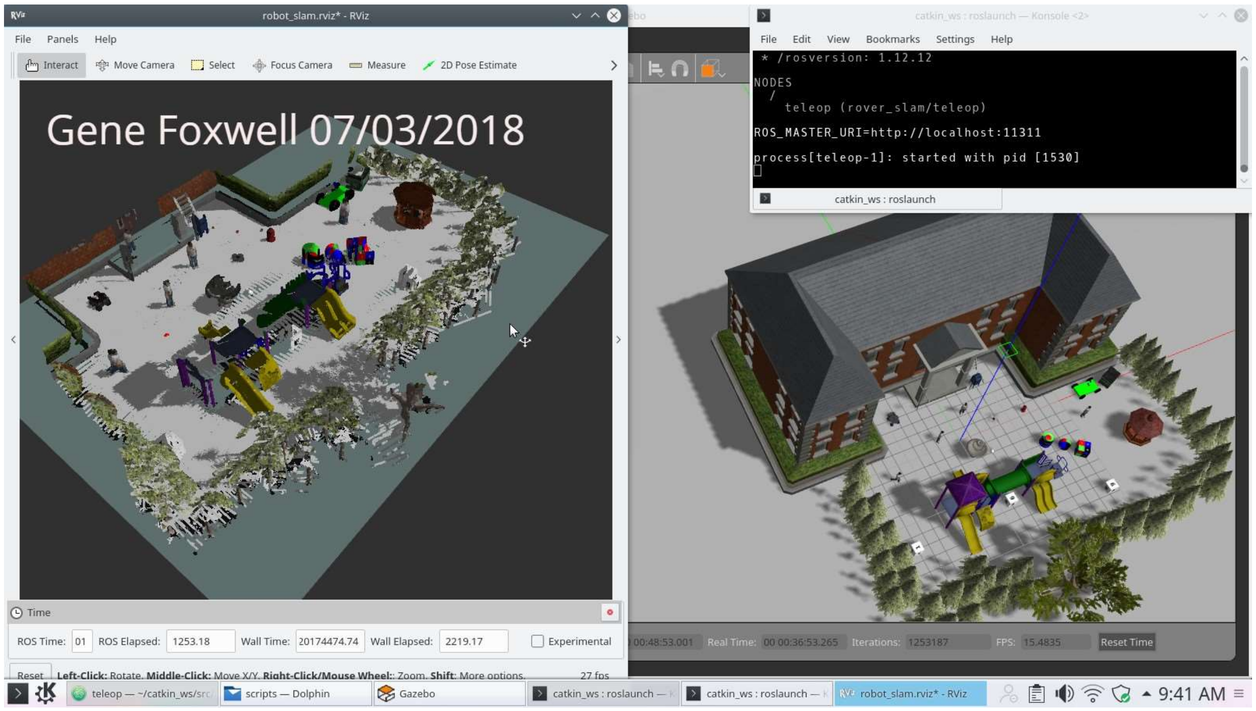
```
Steps: 1  Real Time Factor: 0.96  Sim Time: 00 00:29:59.738  Real Time: 00 00:02:10.078  Iterations: 119924  FPS: 20.0386
```

Figure 8 Top Down View of the School Playground

To facilitate this project an ROS package was created (this is attached with the project materials) which has been named “rover_slam”. The rover_slam package contains all environment files, gazebo files, and urdf files required to run the simulation. Please note however this package does depend on the rtabmap_ros package, which itself depends on the rtabmap software to be installed. The table below describes each component of the rover_slam package and its purpose. Figure 9 shows the rover_slam package in action.

File	Purpose
world/kitchen_dining.world	The gazebo world file for the Kitchen / Dining Room environment.
world/suburbs.world	The gazebo world file for the surburban school environment.
urdf/marvin_bot.xacro	Marvin Bots robot description file.
urdf/marvin_bot.gazebo	Marvin Bots sensor and actuator plugins, such as the differential drive system, the laser scanner, and its Microsoft Kinect RGB-D sensor.
scripts/teleop.py	Python based ROS node for controlling the robot. (Note: this script has been modified from the original version to allow the robot to work without having to hold down the keys continuously. This was done to reduce strain on the user’s hand).

meshes/hokuyo.dae	Mesh file for the hokuyo laser range finder.
launch/kitchen.launch	Launch file for starting all nodes required for mapping the kitchen/dining room environment.
launch/mapping.launch	This opens all nodes required for the rtabmap system to function correctly. (Environment, teleop, and rviz are provided by separate launch files).
launch/marvin_bot_description.launch	This loads the urdf file for the marvin_bot.
launch/rviz.launch	This loads Rviz using the pre-defined settings file provided.
launch/suburbs.laucnh	Loads all nodes required to map the Suburban School environment.
launch/teleop.launch	Loads the teleop python node.
launch/world-kitchen.launch	Loads gazebo with the Kitchen/Dining World and Marvin Bot inside.
launch/world-suburbs.launch	Loads gazebo with the School and Marvin Bot inside.
config/robot_slam.rviz	The RVIZ configuration file used for this project.


robot_slam.rviz* - RViz
File Panels Help
Interact Move Camera Select Focus Camera Measure 2D Pose Estimate
Gene Foxwell 07/03/2018
Time
ROS Time: 01 ROS Elapsed: 1253.18 Wall Time: 20174474.74 Wall Elapsed: 2219.17 Experimental
catkin_ws : roslaunch — Konsole <2>
File Edit View Bookmarks Settings Help
catkin_ws : roslaunch
Real Time: 00 00:36:53.265 Iterations: 1253187 FPS: 15.4835 Reset Time
teleop — ~/catkin_ws/src
scripts — Dolphin
Gazebo
robot_slam.rviz* - RViz
9:41 AM

```
 * /rosversion: 1.12.12

NODES
  /
    teleop (rover_slam/teleop)

ROS_MASTER_URI=http://localhost:11311

process[teleop-1]: started with pid [1530]
```

Figure 9 rover_slam

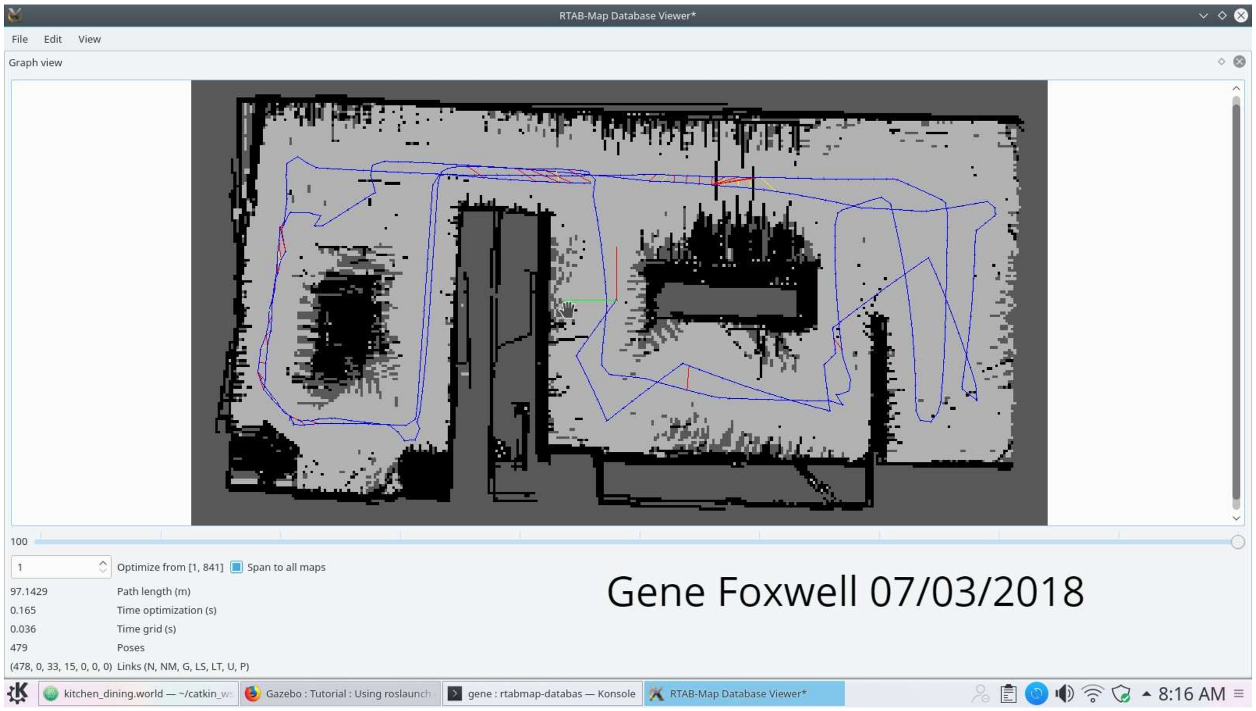
The rover_slam package is intended to map one of two environments. As requiring the user to enter in the full location of the world file each time the project started seemed like a poor user experience two launch files were provided to facilitate the experiment. To run mapping experiments on the kitchen dining room system run **roslaunch rover_slam kitchen.launch** ; to run mapping experiments on the School Yard run **roslaunch rover_slam suburbs.launch**.

RESULTS

Each map was generated by manually driving the robot throughout the world and having the RTAB Map system subscribe to the sensor data provided by the robot. When driving the robot throughout the world longer loops where preferred over small short loops. Multiple passes were taken of each room to get a complete map.

Once the mapping session was completed, results were extracted using the rtabmap-databaseViewer application provided by the rtabmap package. We extracted the 2D graph of the generated map along with the 3D Occupancy grid for each environment. The associated databases for these results have been included with the supporting files for this project. Depicted in each 2D map is a blue line representing the path that the robot took during the mapping process.

Results obtained for the Kitchen / Dining Room environment can be seen in Figure 10 and Figure 11. The maps for the Kitchen / Dining Room reasonably match up with the ground truth with some errors in the positioning of the dining room table and missing data from the tops of the fridge and kitchen counters. Mapping this area resulted in a database size of roughly 205MB.


RTAB-Map Database Viewer*
File
Edit
View
Graph view
100
kitchen_dining.world — ~/catkin_ws
Gazebo : Tutorial : Using roslaunch
gene : rtabmap-databas — Konsole
RTAB-Map Database Viewer*
8:16 AM

```
1                 Optimize from [1, 841]   Span to all maps
97.1429           Path length (m)
0.165             Time optimization (s)
0.036             Time grid (s)
479               Poses
(478, 0, 33, 15, 0, 0, 0) Links (N, NM, G, LS, LT, U, P)
```

Gene Foxwell 07/03/2018

Figure 10 2D Map of Kitchen

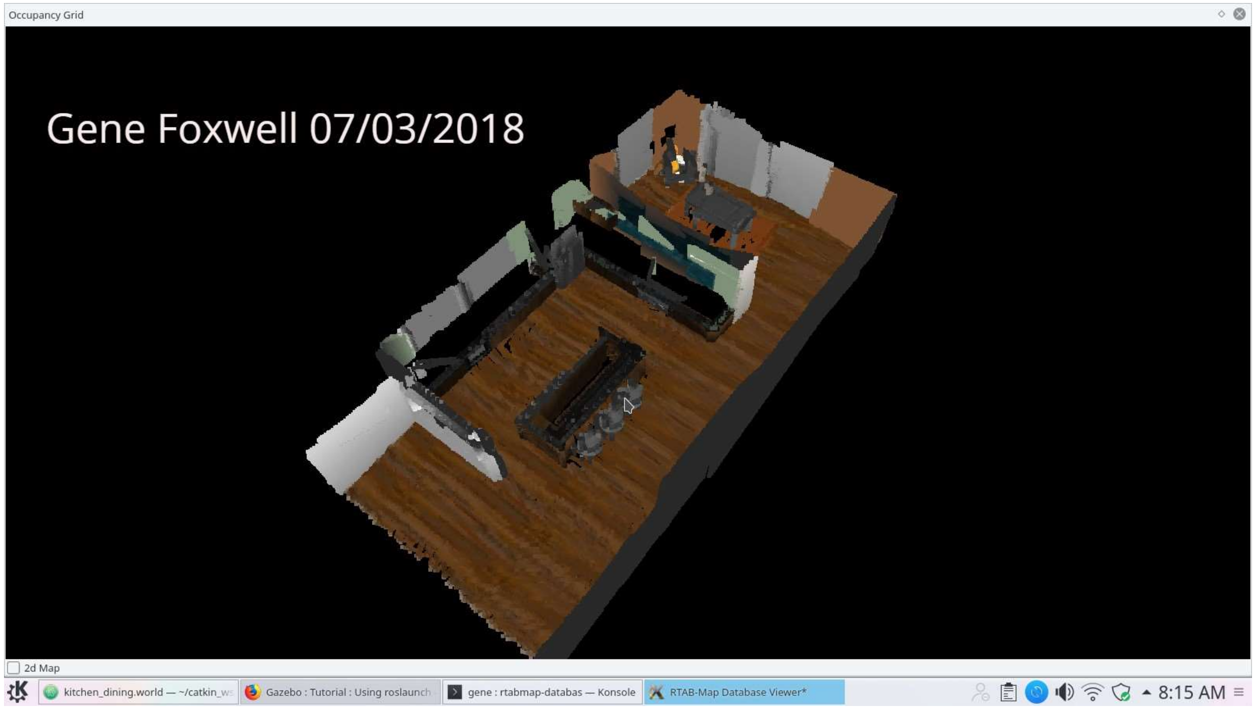
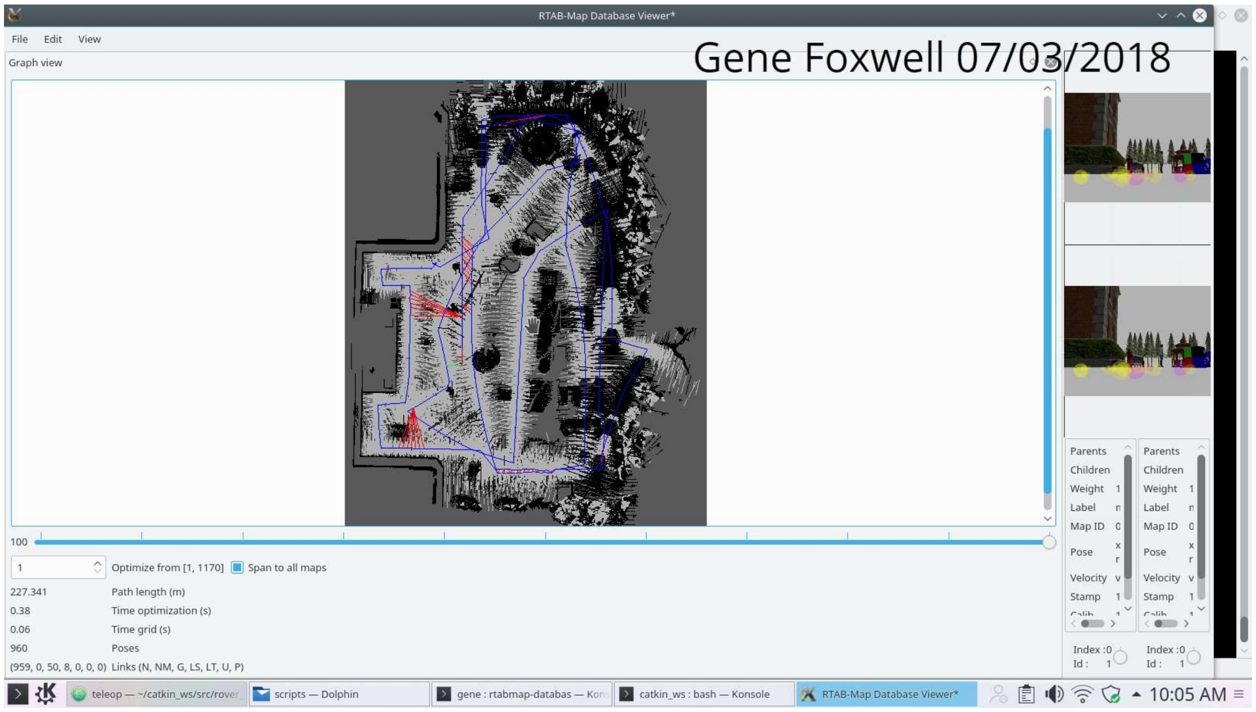

Occupancy Grid
Gene Foxwell 07/03/2018
2d Map
kitchen_dining.world — ~/catkin_w
Gazebo : Tutorial : Using roslaunch
gene : rtabmap-databas — Konsole
RTAB-Map Database Viewer*
8:15 AM

Figure 11 3D Map of Kitchen

Results obtained for the Suburban School Yard can be seen in Figure 12 and Figure 13. Like the Kitchen Dining Room environment there are blank spaces where the robot's sensors could not detect to its size. All major features of the environment are present however. Mapping this area resulting in a database of roughly 500MB.


RTAB-Map Database Viewer*
File
Edit
View
Graph view
Gene Foxwell 07/03/2018
100
Parents
Children
Weight
Label
Map ID
Pose
Velocity
Stamp
Index :0
Id :
teleop — ~/catkin_ws/src/rover
scripts — Dolphin
gene : rtabmap-databas — Kon
catkin_ws : bash — Konsole
RTAB-Map Database Viewer*
10:05 AM

1 Optimize from [1, 1170] Span to all maps

```
227.341                  Path length (m)
0.38                     Time optimization (s)
0.06                     Time grid (s)
960                      Poses
(959, 0, 50, 8, 0, 0, 0) Links (N, NM, G, LS, LT, U, P)
```

Figure 12 Suburban School 2D Map

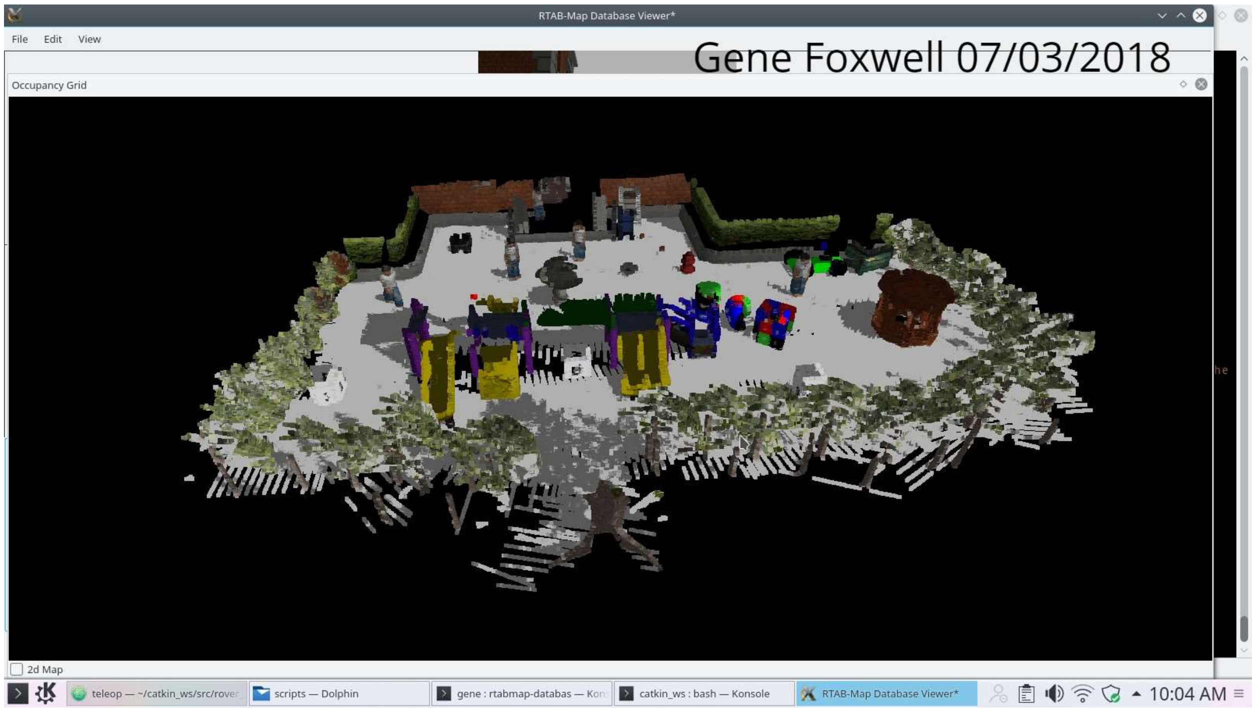

RTAB-Map Database Viewer*
File
Edit
View
Gene Foxwell 07/03/2018
Occupancy Grid
2d Map
teleop — ~/catkin_ws/src/rover
scripts — Dolphin
gene : rtabmap-databas — Kon
catkin_ws : bash — Konsole
RTAB-Map Database Viewer*
10:04 AM

Figure 13 Suburban School 3D Map

DISCUSSION

For both the indoor and outdoor environments the system was able to generate maps that clearly resembled the environment that they were exposed to. The resulting maps do appear to be low resolution however, appearing more as if built by a thick set of crayons than a photographic quality. There were also a few features that seem to have been mildly “duplicated” on the map. Specifically, the dining room table appears to have two instances both located in the same place but rotated slightly relative to each other. A similar defect occurred with the numbered cubes in the School Yard environments.

While both environments appear to have gotten comparable results, subjectively speaking it seemed like the structured nature of the indoor environment made it more straightforward to map. Attempting to map outdoor environments seemed a bit prone to creating inaccurate maps if the robot was moved around too much during the mapping process. Furthermore, without the clear structure that the indoor environment provided it wasn’t as clear what path should be taken to get good results.

Other difficulties not seen in this project also came up when attempting to map the outdoor environments. First there was the relative size of the map itself compared to the robot – this not only slowed down the mapping process but placed limitations on what could be mapped accurately. Early attempts to design and map a more elaborate outdoor environment using gazebo caused issues when good features weren’t also in range of the sensors. Attempts to circumvent this problem like driving the robot in a wave like fashion near the tree line decreased the accuracy of the map and caused the map to become quite unstable.

Symmetry also appeared to cause some problems when mapping an outdoor environment. One early attempt at designing a school yard was highly symmetrical (at least from the point of view of the robot) with an Oak tree at either end of the map. This seem to occasionally confuse the system however and would cause the map to seemingly “collapse” into chaos. It is possible that this is caused by premature loop closures pushing wrong results back up into the graph. It may be the case that further map exploration would fix this issue, but it was found to be more efficient to simply restart the mapping process when such errors occurred.

Excessive maneuvering also seemed to trip up the mapping algorithm in some cases. Like other issues mentioned, this showed more in the outdoor environment than in the indoor. While mapping the outdoor environment it was very easy to accidently miss a turn or find the robot in an area that required a bit more backing up and turning around than would normally be needed. Such events would occasionally throw off the mapping algorithm and reduce the quality of the map considerably.

Despite the limitations encountered with this system, it was still able to perform its task within the desired tolerances. A robot equipped with RTAB Map should have few problems mapping most commonly found indoor environments (provided there are no humans or pets in the way), and if there is sufficient room to drive it should be able to map outdoor environments that are feature rich with respect to the robot’s scale.

Future Work

There is plenty of room for improvement on this system. Its performance in outdoor environments suggests that further research into matching visual images for loop closures may be helpful. A system that can identify the same graph node from multiple points of view (rather than the single point of view that seems to be supported by the Visual Bag of Words built from SURF) would allow for building more accurate maps faster. This may allow the system to deal with symmetry in the environment better since it could use the current point of view to resolve ambiguous features in the image.

Unfortunately, the resources for building an actual robot where not available to the experimenter currently however there are several possible uses this system could have once the parts can be obtained. For example:

- A small robot equipped with this system could be used to build 3D maps of residential (or commercial) real-estate. This could be used by estate agents to provide virtual tours to their potential customers either before they decide on which house to view, or if they are forced to purchase from out of down, allow them to tour the property virtually without having to be physically present. Such maps once generated could also be used for other visualization purposes.
- This system could also be deployed on robots sent in to investigate industrial accidents where the environment would be too dangerous to risk human life. By generating a 3D map, workers can asses the situation and work out the feasibility of different recovery strategies.
- Used in conjunction with autonomous code to handle the behavior, this system could be used as a key component for a "support robot" for the disabled. By building and maintaining maps of the environments it finds itself in, it could be used to guide its owner safely through the world, or guide help towards its owner in the event such help was needed.

Finally, it seems that the morphology of the robot played a role in what it was able to map effectively. It would be interesting to investigate the effect of changing the size, speed, and shape of the robot has on its ability to effectively map different environments.