Map My World Project

Taariq Hassan

Abstract—The Simultaneous Localisation and Mapping (SLAM) project required using the Real-Time Appearance-Based Mapping (RTAB-Map) Graph SLAM approach to enable a mobile robot to map an unknown environment while localising itself with the environment. Two environments were mapped by the robot, a simulated environment provided for the project as well as a custom environment built in Gazebo. For each of these environments, RTAB-Map (using the *rtabmap_ros* package) was used to generate both a 2D and 3D map of the environment using the robots on-board sensors.

Index Terms—SLAM, RTAB-Map, ROS, Udacity, Robotics.

1 Introduction

The goal of the SLAM project was to use the *rtabmap_ros* package to implement the RTAB-Map SLAM algorithm in order for a simulated robot to map an unknown environment. SLAM enables a robot to construct a map of its environment and estimate its pose within the map, using only its own motion and sensor data. There are a number of algorithms which solve the SLAM problem, some of which will be discussed later. For this project however, the RTAB-Map algorithm was used. The algorithm was tested on a mobile robot in two different simulated environments using ROS and Gazebo. The robot configuration as well as the environments will be discussed in more detail later.

2 Background

SLAM is a fundamental problem in robotics. It is not always possible to provide the robot with a map of the environment, and for a robot to only work in known environments would be very limiting. Therefore, if a robot is to navigate in an unknown or changing environment, it needs to be able to construct a map and localise itself relative to it. This is a challenging task since both the robot's motion and sensor measurements are noisy, introducing uncertainties in the map and pose estimate.

SLAM is a combination of two problems in robotics, namely localisation and mapping. To solve the localisation problem, the robot needs a map of the environment. Conversely, to map its environment, the robot's poses need to be known. SLAM is essential in the case where neither a map nor the robot's poses are known. It is therefore found in a large number of robotics applications such as self-driving cars or autonomous underwater vehicles to name a few.

There are many different SLAM algorithms and implementations, two of which will be discussed in further detail below.

2.1 FastSLAM

The FastSLAM algorithm uses a modified particle filter combined with a low dimensional EKF to solve the SLAM problem. The algorithm is able to solve the full SLAM problem with known correspondences. FastSLAM makes use of the conditional independence property of SLAM, decomposing the problem into that of estimating the robot's path or trajectory and that of estimating the landmarks or the map. The posterior over the robot path is estimated using the modified particle filter. Each particle contains a set of Kalman filters estimating the landmarks, conditioned on the estimate of the path the particle represents. However, a drawback of this approach is that the algorithm must assume known landmark positions, and therefore is not able to model arbitrary environments.

An extension of FastSLAM, called Grid-based FastSLAM, solves this problem by introducing occupancy grid maps. The MCL and Occupancy Grid Mapping algorithms are combined to form the Grid-based FastSLAM algorithm. The MCL algorithm is used to estimate the pose of each particle as well as the likelihood of the measurement. A map estimate is then generated for each particle using the Occupancy Grid Mapping algorithm. The particles are then re-sampled as per the standard MCL algorithm, with surviving particles, and their associated map estimate, being added to the system belief for the next iteration.

2.2 GraphSLAM

GraphSLAM is another SLAM algorithm that solves the full SLAM problem and provides a number of advantages, including improved accuracy compared to FastSLAM. It works by creating a graph to represent the robot's poses as well as features and landmarks in the environment. The graph also represents constraints. Motion constraints join successive robot poses and measurement constraints join a feature and a pose. These constraints are soft since they include any uncertainties in the motion or measurements. As the robot moves, more poses and features are added to the graph with their associated constraints. GraphSLAM has the advantage of being able to handle a large number of features which is essential since the graph can grow very quickly. Ultimately, the goal of the algorithm is to create a graph off all the poses and features, and then optimise the constraints by finding the node configuration which represents the robot's most likely path and map.

For this project, a specific Graph-based SLAM approach was used called Real-Time Appearance-Based Mapping

(RTAB-Map). RTAB-Map relies on the use of an RGB-D camera to perform localisation and mapping. Images are used to generate a 3D map of the environment, with features being extracted from the images in order to perform localisation. Each new image from the camera is compared to previous images to determine whether the robot has been to a specific location before. This process is called loop closure. RTAB-Map is optimised for real-time loop closure detection.

3 Scene and Robot Configuration

3.1 Worlds

As mentioned previously, the RTAB-Map algorithm was tested on a mobile robot in two different simulated Gazebo environments. The first environment, found in *worlds/kitchen_dining.world*, was provided for the project. The environment can be seen in figure 1.

Fig. 1. Provided *kitchen_dining* world

The second environment, found in *worlds/my_world.world*, was created using Gazebo's built-in Building Editor. A simple environment was created with three rooms. A number of objects from the Gazebo Model Database were then placed in each room. The environment can be seen in figure 2.

3.2 Robot Configuration

The robot model used was a modified version of *udacity_bot* from the "Where Am I?" project. *udacity_bot* is a rectangular robot with two driven wheels on either side. There are also two caster wheels underneath the robot in the front and back in order to provide stability. The robot was modified to include a kinect RGB-D camera. Since the kinect is a 3D camera, it was mounted on top of the robot to maximize the height, making use of the vertical field of view. The LIDAR was moved to the front of the robot since it only scans in a 2D plane and therefore its output is less affected by height. The robot configuration is defined in the *urdf/udacity_bot.xacro* and *urdf/udacity_bot.gazebo* files of the package. A picture of the robot can be seen in figure 3.

The transform tree describing the robot configuration can be seen in figure 4. This was used to ensure that the robot was configured correctly in the urdf files. It should be noted that the links and frames found under the *camera_rgb_frame* were essential to orient the kinect camera's axes correctly. These link and frame definitions were adapted from the *turtlebot_description* ROS package.

Fig. 2. Custom world *my_world*

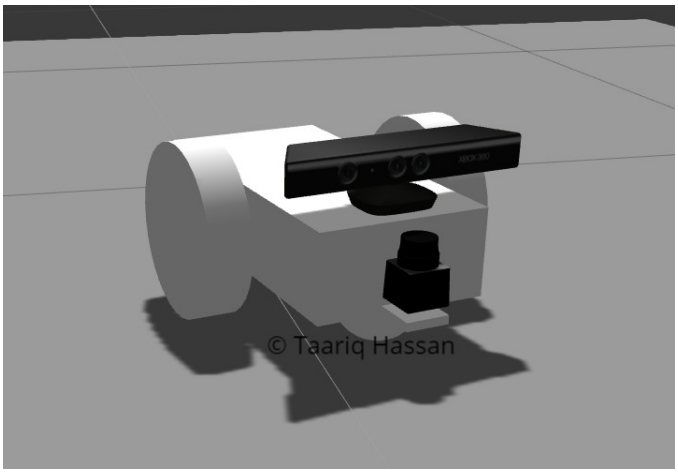

Fig. 3. Closeup of *udacity_bot*

3.3 Package

All project files were contained in a single ROS package called *slam_project*. The package contained all the necessary files to run the simulation in both worlds. The robot definition and world files are located in the *urdf* and *worlds* directories as mentioned previously. The package also contains a number of launch files in the *launch* directory, which are used to launch the various nodes required for the project. Two separate launch files exist for the two different worlds. *world.launch* launches the *kitchen_dining* world while *my_world.launch* launches the custom *my_world*. In addition,

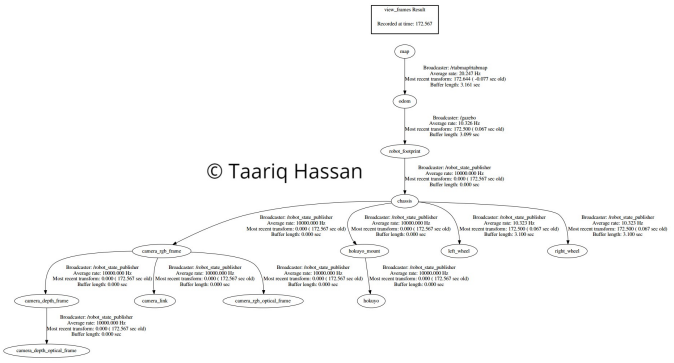

Fig. 4. Transform Tree for *udacity_bot*

the *teleop.launch*, *mapping.launch* and *rviz.launch* files launch the *teleop* node to control the robot, the *rtabmap_ros* node to implement the RTAB-Map algorithm, and the *rviz* node to launch Rviz, respectively. Once all nodes were launched, *rqt_graph* was used to ensure that all nodes were publishing and subscribing to the correct topics. The output of *rqt_graph* can be seen in figure 5.

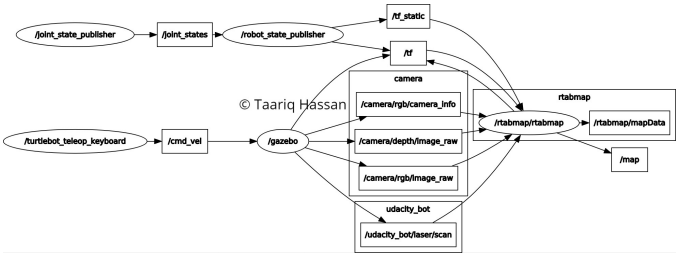

Fig. 5. Output of *rqt_graph* for the project package

4 Results

Using the package, the RTAB-Map algorithm was tested by driving the simulated *udacity_bot* around each of the worlds to generate a map. The sections below present the results for each world.

4.1 *kitchen_dining* world

A map of the *kitchen_dining* world was successfully created using RTAB-Map. Figure 6 shows the 3D point cloud generated by RTAB-Map.

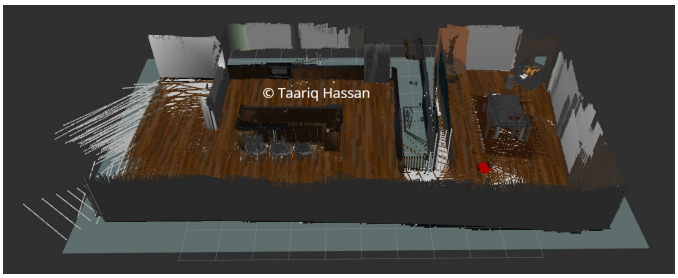

Fig. 6. *kitchen_dining* world 3D Map

Using *rtabmap-databaseViewer*, it was found that the map had 26 loop closures, as can be seen in figure 7. The 2D map of the world showing the path followed by the robot can also be seen, as well as some features and constraints.

![alt text][image7]

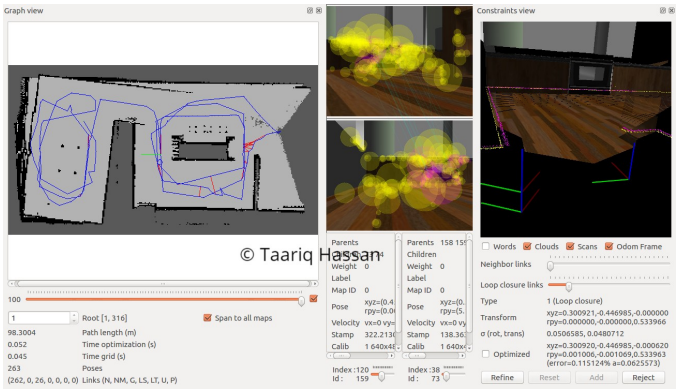

Fig. 7. *kitchen_dining* world database view

4.2 *my_world*

A map of the *my_world* environment was also created successfully using RTAB-Map and *udacity_bot*. The point cloud can be seen in figure 8.

Fig. 8. *my_world* 3D Map

rtabmap-databaseViewer reported 53 loop closures for this world which can be seen in figure 9, along with the 2D map showing the path followed by the robot.

5 Discussion

udacity_bot was able to map both environments reasonably well using RTAB-Map. There were however some problems that can be observed. From the generated map of the *kitchen_dining* world shown previously (Fig6), we can see that the left side of the world, the kitchen area, was mapped reasonably well, while the dining area on the right has more repeated, choppy artifacts. These artifacts are usually due to no loop closures being detected and therefore the graph not

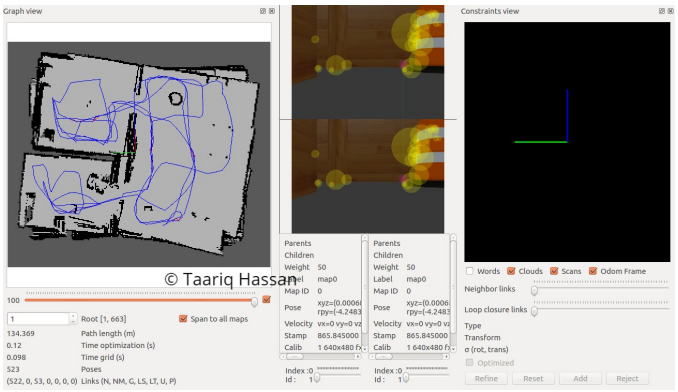

Fig. 9. *my_world* database view

being optimised. The map had 26 loop closures, however, it can be seen that most of the loop closures occurred in the kitchen area with only a few in the dining area. This would explain the better quality map in the kitchen area as opposed to the dining area. The same phenomenon can be seen in the 3D map of the *my_world* environment (Fig8). This may be solved by changing some of the RTAB-Map parameters such as the feature detector strategy or loop closure parameters however this was not explored in this project.

Another observation was that tall objects were not fully mapped as can be seen in the point clouds. This was due to the fact that the RGB-D sensor is relatively low down, close to the ground since the robot is small. In order to overcome this, the sensor can be mounted at an angle such that it is looking slightly up. This will allow more of the environment to be mapped and may increase the number of loop closures, since more features will be detected.

6 Future Work

This project tackled one of the fundamental building blocks of an autonomous robots, simultaneous localisation and mapping. However, for this project, the robot was driven manually. An obvious extension to this project would be to use the results from RTAB-Map to localise a robot within an unknown environment and to detect and avoid obstacles when doing path planning and navigating autonomously to a goal position.

The improvements and suggestions discussed in the previous section should also be implemented, such as increasing the angle of the kinect sensor.

Finally, the algorithm could be implemented in hardware and tested in a real world environment using a mobile robot platform with a kinect sensor.