Robot Simultaneous Localization and Mapping

Jordan Croom

Abstract—In this paper, a simulation is presented leveraging GraphSLAM to map the simulated environment using a wheeled robotic vehicle equipped with an RGBD sensor and 2D lidar. Results are evaluated for two test environments.

Index Terms—Robot, IEEEtran, Udacity, LaTeX, Mapping.

1 Introduction

Simultaneous localization and mapping is a well-studied problem with numerous real-world applications. Robotics companies such as Neato and iRobot utilize SLAM to continuously monitor and update indoor maps to guide domestic robots. Disaster recovery robots have leveraged SLAM techniques for tasks such as nuclear disaster cleanup and earthquake recovery. Researchers have also demonstrated that SLAM techniques may be a more effective approach to autonomous vehicles than detailed, pre-compiled maps [1]. This paper attempts to address a similar but more limited problem in simulation. Given two simulated indoor environments, a wheeled robot equipped with an RGBD camera, 2D lidar, and wheel odometry sensors is used to create detailed interior maps (both 2D and 3D) while simultaneously updating its estimated pose within the map. The sensor data is fused to build the map using the RTAB-map ROS package, which uses modified GraphSLAM to simultaneously map the environment and localize the robot.

1.1 The Gazebo Platform

The Gazebo multiphysics simulation engine paired with the RViz visualization platform greatly decreases the complexity of simulating and testing complex physical systems to enable efficient, low-risk algorithm development. Using ROS (the robot operating system), Gazebo is able to emulate arbitrary robot architectures with a number of typical sensors, actuators, and simulated objects with which to interact. ROS, Gazebo, and RViz were leveraged in this case to demonstrate the performance of GraphSLAM (RTAB-map) by effectively mapping the simulated environment and localizing the robot within it.

2 Background

In the field of mobile robotics, the robot typically needs three pieces of information to perform a given task:

1) **Pose**: Its pose within some frame of reference
2) **Map**: Some representation of the environment in which it is operating.
3) **State**: Some knowledge of the state of various systems comprising the robot (e.g. gripper open or closed, etc.)

For many practical systems, neither an accurate map or pose are available. For example, consider an autonomous submarine attempting to navigate toward some waypoint beacon. It's likely that an accurate map of the environment is not available ahead of time. In this case, the robot must both build a map of its environment and estimate its pose with respect to the map as it moves toward the beacon. In order to do so, it may need to continuously fuse multiple sources of information about its environment (e.g. depth sensors, cameras, sonar, etc.). This class of problem is the motivation for SLAM. In this case, pose estimates and an estimate of the map must be built and continuously updated as new information is gathered.

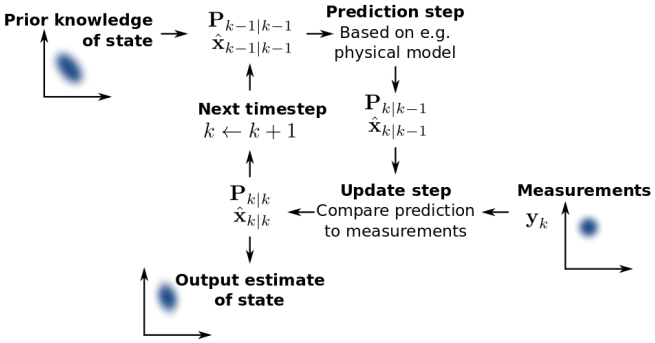

Fig. 1: Visual Outline of the Kalman Filter Algorithm

The presented simulation requires the fusion of multiple sources of position and orientation information: wheel odometry, a 2D laser range sensor (lidar), and an RGBD camera. The information from these sensors are used to build a map of the environment and to continuously estimate the pose of the simulated robot within this map. Numerous SLAM algorithms are available to solve this class of problems, but two in particular were considered for this work: Occupancy Grid FastSLAM [2] and GraphSLAM [3].

2.1 Occupancy Grid FastSLAM

The FastSLAM algorithm is based on a particle filter approach. In this algorithm, virtual elements (particles) each store not only a potential robot pose but also a potential map. In the particular implementation considered, this map is represented as an occupancy grid. Particles are initialized randomly in the first step, the distribution of which across the unknown map is dependent on the particulars of the specific problem. Each subsequent iteration is made up of three processes:

1) **Motion Update**: Each particle's state is adjusted according to the expected (commanded) change in state since the last step (e.g. change in position commanded by the wheel controller)

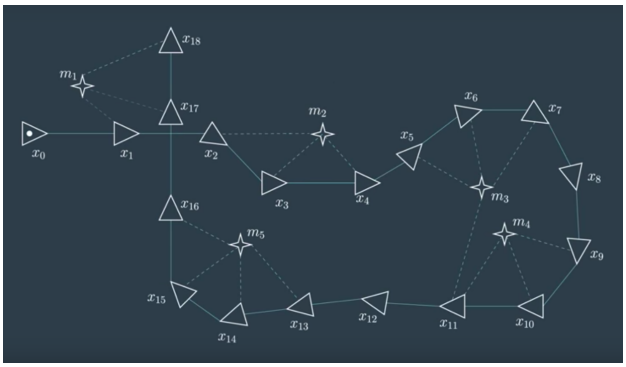

Fig. 2: Graphical Representation of GraphSLAM Algorithm

.

2) **Measurement Update**: The current sensor measurements are used to estimate the probability each particle represents the robot's (or other system's) current state. This probability is translated to a weight value for each particle.
3) **Map Update**: Each particle's stored map is updated assuming the estimate of the particle's pose is accurate. This step is effectively reduced to the problem of mapping with known poses.
4) **Resampling**: N particles are selected from the current batch to be passed to the next iteration. Particles with higher weight are more likely to be propagated to the next iteration than those with lower weight.

The collection of particles is used to form a statistical distribution of the possible states of the system and configurations of the map. This procedure requires no assumption of the nature of process or sensor noise (e.g. Gaussian) and has the additional benefit of being adjustable based on the processing capabilities of a given system simply by adjusting the number of particles used. This can even be optimized each iteration based on system load.

2.2 GraphSLAM

The GraphSLAM algorithm represents the robot's pose and measurements of its motion or the environment as constraints. Consider the graphical representation of the GraphSLAM algorithm depicted in Figure **??**. Let estimates of each robot pose be denoted x_i. These poses are represented as triangular nodes in the figure. At each node, the robot may take a measurement of a landmark (e.g. distance) denoted m_i. The measurement in this case is represented as an edge (dotted line) in the graph which functions as a constraint on the graph between the landmark m_i and the node x_i. In addition, the robot may also keep track of its motion between pose nodes. This motion estimate results in another constraint, i.e. the distance between subsequent pose nodes x_i and x_{i+1}.

As each new constraint is added to the system, the GraphSLAM algorithm modifies the graph by adjusting motion constraints and measurement constraints between each node to produce the overall system graph which produces the smallest cumulative error. In practice, this is done with iterative optimization.

In summary, each iteration of the GraphSLAM algorithm is composed of the following primary phases:

1) **Add Node**: A new node representing the current state of the robot is added to the graph.
2) **Add Constraints**: The node is connected to previous nodes using motion constraints and to environment landmarks using measurement constraints.
3) **Optimize Graph**: Given the updated graph, an optimization scheme (method varies by implementation) modifies the constraints (or graph edges) between all nodes such that the overall error from the previous state is minimized.

GraphSLAM conveniently keeps track of not only the overall map with landmark nodes in the graph but also the path of the robot (or generalized system) since the inception of the graph. By simply connecting the robot pose nodes in the graph, the robot's path can be represented and updated with each iteration.

2.3 GraphSLAM vs. FastSLAM

In general, GraphSLAM is a more robust approach to FastSLAM. FastSLAM can only consider the current state of each particle. While each particle implicitly contains information about previous states that resulted in its current pose and map estimate, the error in previous iterations is not corrected in subsequent iterations.

In contrast, GraphSLAM maintains the graph of all previous measurements and estimated poses. In each iteration, GraphSLAM can consider adjustments to all previous constraints simultaneously. As a result, the algorithm is more robust to momentary lapses in sensor accuracy (e.g. drop-outs or false detections) and has the benefit of incorporating its entire path into its map estimate explicitly.

2.4 RTAB-Map

The simulations presented in this paper leveraged RTAB-Map, an appearance-based SLAM implementation in ROS. RTAB map incorporated the basic GraphSLAM characterisitics previously explained with the additional of loop closure. Loop closure identifies features within gathered images that correspond to one another and matches corresponding points to create new constraints between additional graph nodes. This allows for reduced map error. RTAB-Map also implements memory management techniques to allow for real-time SLAM in large environments or over long periods of data collection.

3 Scene and Robot Configuration

3.1 Robot Configuration

A summary of the robot links can be seen in Figure 3.

TABLE 1: Ultra Udacity Bot Summary

Sensors	+/-180 Degree HFOV Lidar Front-facing RGBD Camera
Total Mass	18kg
Track Width	0.35m
Wheelbase	0.25m
Configuration	2 Fore-Mounted Wheels 1 Heavy Aft-Mounted Caster
Approximate Robot Envelope Dimensions	0.40m x 0.40m

Ultra Udacity Bot builds on the Udacity Bot design provided in the robot localization project with a few modifications. Most

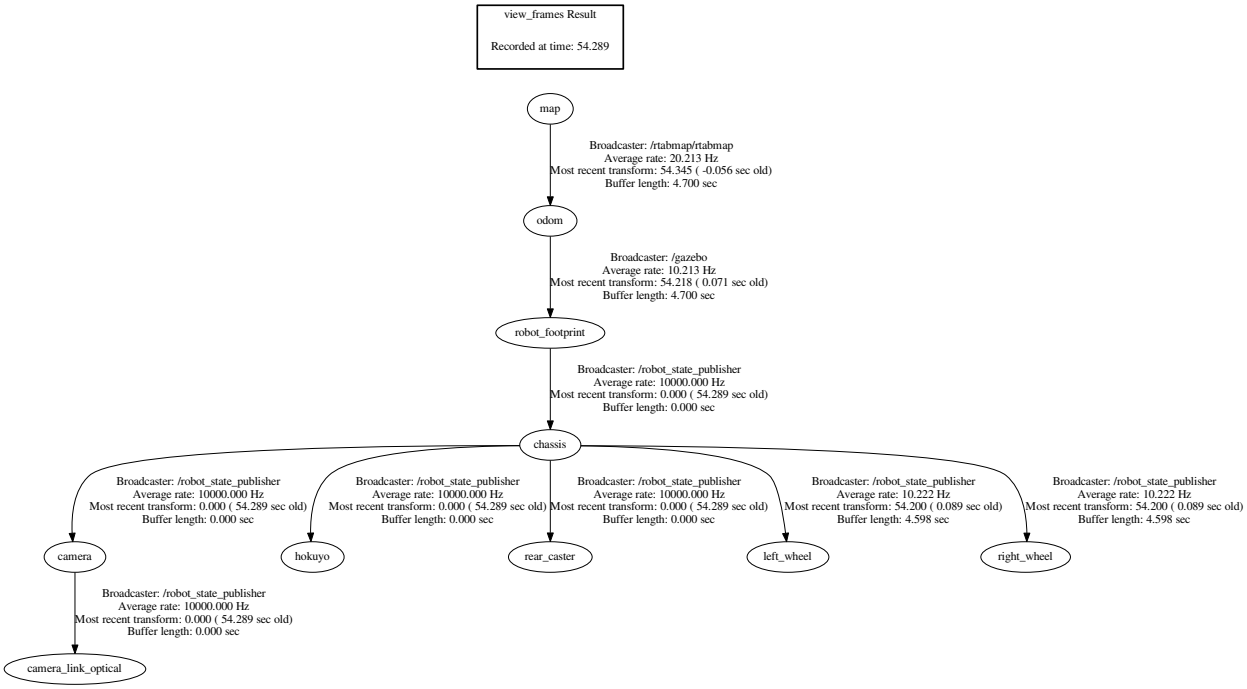

Fig. 3: Robot Link Summary

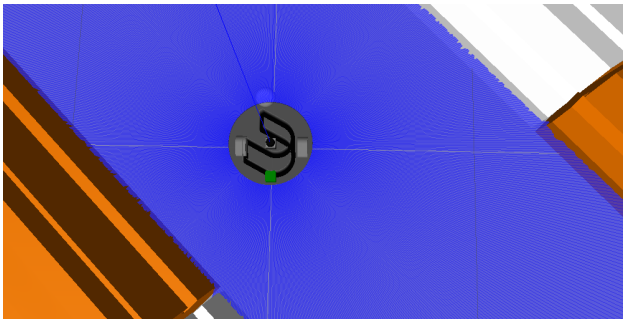

Fig. 4: Top View of Ultra Udacity Bot with Blue Lines Representing Lidar Distance Measurements

.

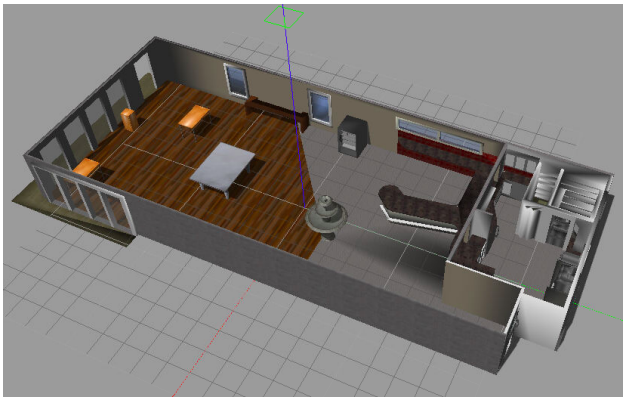

Fig. 5: View of the Cafe Scene

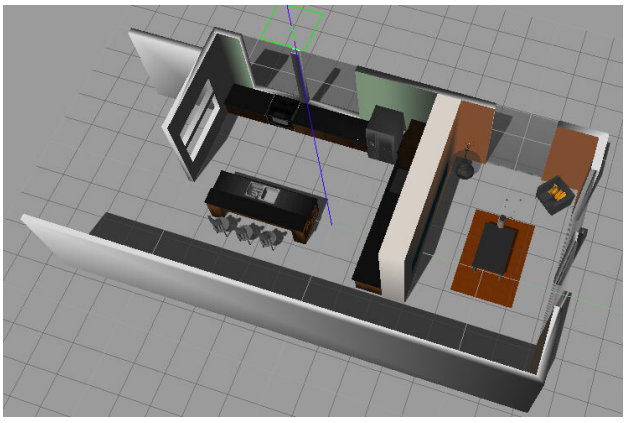

Fig. 6: View of the Kitchen and Dining Room Scene

importantly, the chassis styling has been upgraded. Secondly, the lidar unit has been modified to return the same number of point measurements as the original Udacity Bot but in a +/-180 degree HFOV rather than the original +/-90 degree HFOV. The intention of this modification was to allow for more certainty in robot position enabling the robot to sample a wider range of features. In order to support full +/-180 degree HFOV, the wheel radius is reduced by 50%.

3.2 Scene Configuration

Simulations were performed in two different indoor scenes: a kitchen and dining room scene (Figure 6) and a commercial cafe scene (Figure 5).

The commercial cafe scene is constructed with a bare cafe scene augmented with various pieces of furniture and an exquisitely calming water feature. While this was not simulated,

one imagines that such an environment would delight potential customers.

4 Results

Videos of successful mapping runs can be found at the following locations:

- Cafe Mapping Run
- Kitchen Mapping Run

For each run, a map database was gathered and saved. The maps for each environment can be downloaded at the links below:

- Cafe Map Database
- Kitchen Map Database

4.1 Kitchen Map

See Figures 7, 8, and 9.

4.2 Cafe Map

See Figures 10, 11, and 12.

5 Discussion

Each scene contained a number of disparate objects to aid identification of shared landmarks between different RGBD camera images. Earlier implementations of the cafe scene without sufficient scene variety resulted in false-positive loop closures, which led to completely inaccurate maps. A video of a failed cafe mapping run in which too many loop closures were detected can be found here.

In addition, some experimentation was required with the RTAB-Map parameters to get reliable performance. In particular, the SURF Hessian threshold was set to 500 (in fact, the default RTAB-Map value) rather than the initial Udacity-provided value (< 100). Increasing this number reduces the number of points of interested used in the SURF algorithm to detect corresponding image points for use in loop closure. Prior to modifying this value, too many key points were falsely identified as corresponding. This resulted in inappropriate loop closures, which jumbled the maps irrecoverably.

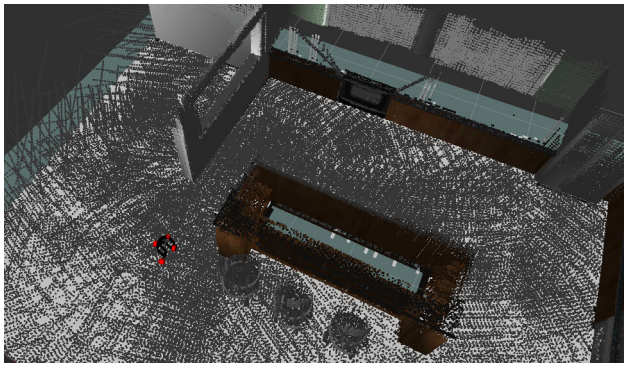

Fig. 7: Robot in the Process of Mapping the Kitchen & Dining Room

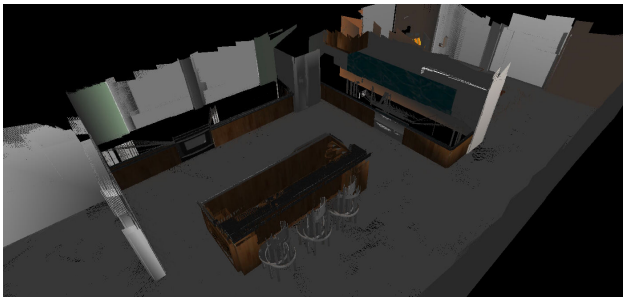

Fig. 8: View of the Generated 3D Kitchen Map

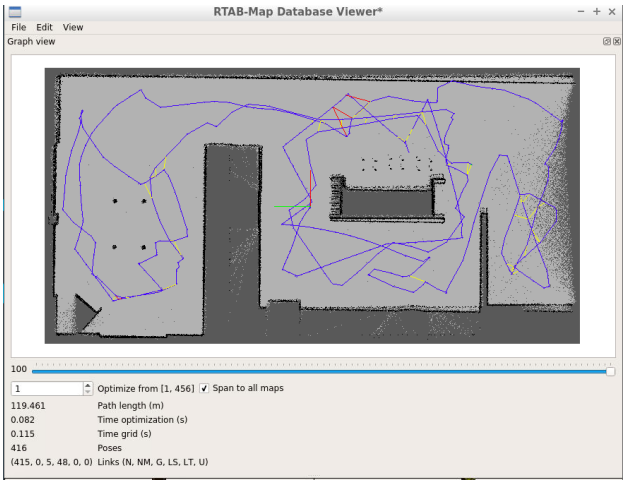

Fig. 9: View of the Kitchen Mapping Path and Occupancy Grid Map

Fig. 10: Robot in the Process of Mapping the Cafe

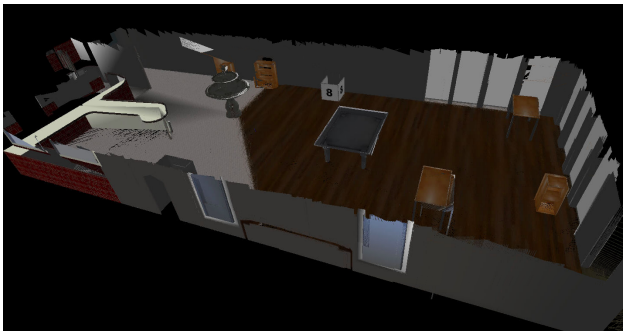

Fig. 11: View of the Generated 3D Cafe Map

Fig. 12: View of the Cafe Mapping Path and Occupancy Grid Map

6 Future Work

This paper has successfully demonstrated the use of appearance based GraphSLAM implemented with RTAB-Map in a selection of simulated environments. It is noted that performance may be improved by adding an additional RGBD camera in another orientation (e.g. rear-facing). At this time, RTABMap allows for up to two RGBD cameras.

It should also be possible to extend the results to accomplish a number of tasks of interest. For example, RTABMap can be extended to recognize a target object and navigate toward it. This is a classic mobile robotics task that would be applicable in a number of practical scenarios. Furthermore, it should also be possible to implement the demonstrated software stack in hardware (e.g. an actual mobile robot). This would be an interesting demonstration of GraphSLAM in the real world.

References

[1] T. Ort, L. Paull, and D. Rus, “Autonomous vehicle navigation in rural environments without detailed prior maps,”

[2] S. Thrun, M. Montemerlo, D. Koller, B. Wegbreit, J. Nieto, and E. Nebot, “Fastslam: An efficient solution to the simultaneous localization and mapping problem with unknown data,” vol. 4, 05 2004.

[3] S. Thrun and M. Montemerlo, “The GraphSLAM algorithm with applications to large-scale mapping of urban structures,” *International Journal on Robotics Research*, vol. 25, no. 5/6, pp. 403–430, 2005.