Real-Time 3D Appearance Based Simultaneous Localization and Mapping

Martin Bufi

Abstract—Simultaneous Localization and Mapping (SLAM), is the computational problem of constructing or updating a map of an unknown environment while simultaneously keeping track of an agent's location within it. Popular approximate solution methods include the particle filter, extended Kalman filter, and GraphSLAM. This report summarizes the steps in implementing FastGraph SLAM in the Robot Operating System (ROS) environment, followed by the results and future improvements.The goal of the software package is to generate navigatable robotic maps and perform loop detection in two specific worlds; a kitchen world, and a second created using gazebo.The kitchen world can be seen below:

Index Terms—Robot, IEEEtran, Udacity, LaTeX, Kalman filters, Particle Filters, Localization, Mapping, RTAB, FastSLAM.

1 Introduction

IMAGINE a mobile robot moving through an unknown, static environment. The robot executes controls and collects observations of features in the world using a suite of sensors and control feedback. The biggest issue here is that both the controls and the observations are corrupted by noise.

Simultaneous Localization and Mapping (SLAM) is the process of recovering a map of the environment and the path of the robot from a set of noisy controls and observations. 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 SLAM environments constructed in Gazebo were established and tested in simulation. Using the package that was constructed, the turtle bot was able to successfully locate itself and map the 3D world. The benchmark environment was in the kitchen dining world, while the second environment was called martinworld. The source software for this project can be found here: https://github.com/mbufi/slam_project.

Fig. 1. Kitchen Dining world.

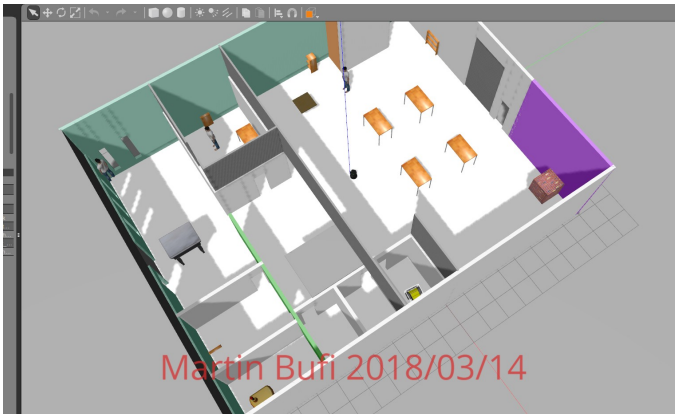

Fig. 2. Martin World

2 Background

SLAM is often called the chicken or the egg problem because the map is needed for localization, and the robots posed is needed for mapping, thus it is a challenging problem. There are two relevant techniques that may be considered in solving the simultaneous localization and mapping problem. The first is Extended Kalman Filters (EKF) and the second Adaptive Monte Carlo Simulation (AMCL).

2.1 Kalman Filters

The Kalman Filter is prevalent in control systems. The algorithm is good at taking in noisy measurements in real time and providing accurate predictions of actual variables such as position. However, the Kalman Filter algorithm is based on two assumptions:

- Motion and measurement models are linear.
- State space can be represented by a unimodal Gaussian distribution.

These assumptions limit the applicability of the Kalman Filter as most real robot scenarios do not meet these assumptions. The Extended Kalman filter (EKF) addresses these

limiting assumptions by linearizing a nonlinear motion or measurement function with multiple dimensions using multidimensional Taylor series.

While the EKF algorithm addresses the limitations of the Kalman Filter, the mathematics is relatively complex and the computation uses considerable CPU resources.

2.2 Particle Filters

Particle Filters operate by uniformly distributing particles throughout a map and then removing those particles that least likely represent the current position of the robot. The Monte Carlo Filter is a particle filter that uses Monte Carlo simulation on an even distribution of particles to determine the most likely position. Computationally it is much more efficient than the Kalman Filter. Also Monte Carlo localization is not subject to the limiting assumptions of Kalman Filters as outlined above.

2.3 SLAM

Unfortunately, neither of these approaches are adequate to solve the SLAM problem. One approach to solving SLAM is to add the robot pose weight and map to each particle. This approach is problematic because the map is modeled with many variables resulting in high dimensionality. The straight particle filter approach to SLAM in this form scales exponentially and is impractical.

2.3.1 Continuous Elements

SLAM has both continuous and discrete elements. The continuous component of SLAM relates to the fact that a robot continuously collects odometry to estimate the robot poses and continuously senses the environment to estimate the location of objects or landmarks. Both robot poses and object location are continuous aspects of the SLAM problem. **The continuous parameter space composed of the robot poses and the location of the objects is highly dimensional.**

2.3.2 Discrete Elements

While mapping the environment and performing localization, the robot will encounter many objects and have to keep track of each one of them. The number of variables increases with time, thus making the problem highly dimensional and making it challenging to compute. **The discrete component relates to sensing the environment to estimate the location of objects.** The SLAM algorithm must identify if a relation exists between any newly detected objects and previously detected ones in order to understand if it has been in this same location before and should not be remapped.

2.3.3 Correspondence

The discrete relation between objects is known as correspondence. The discrete parameter space is composed of the correspondence values. It is also highly dimensional due to the large number of correspondence variables. The correspondence values increase exponentially over time since the robot will keep sensing the environment and relating the newly detected objects to the previously detected ones. Because of the high dimensionality it is infeasible to compute the posterior under unknown correspondence. SLAM algorithms must rely on approximation to conserve CPU resources.

2.4 FastSLAM

The **FastSLAM** algorithm uses a particle filter. Each particle holds a guess of the robot trajectory so the SLAM problem is reduced to mapping with known poses. It solves the SLAM problem with known correspondences by breaking the problem into two phases:

1) **Estimating the Trajectory:** FastSLAM estimates a posterior over the trajectory using a particle filter.
2) **Estimating the Map:** FastSLAM uses a low dimensional EKF (Extended Kalman Filter) to solve independent features of the map which are modeled with local Gaussian.

The two-phase approach of representing the posterior probability with a particle filter and a Gaussian is called the **Rao-Blackwellized particle filter** approach. Early versions of FastSLAM had the disadvantage in that known landmark positions are assumed, so it is not possible to model arbitrary environments.

2.5 Grid-based FastSLAM

The Grid-based FastSLAM grid mapping algorithm models the environment using grid maps without predefining any landmark positions. It extends the FastSLAM algorithm to occupancy grid maps, to solve the SLAM problem in an arbitrary environment.

The basic steps of Grid-based FastSLAM are as follows:

1) Previous Belief
2) Sampling Motion
3) Importance Wait
4) Map Estimation
5) Re-sampling
6) New Belief

Each particle in the Grid-based FastSLAM holds a guess of the robot trajectory as in FastSLAM. In addition each particle has its own map. The grid-based FastSLAM algorithm updates each particle by solving the mapping with known poses problem using an occupancy grid mapping algorithm.

The following steps are used to adapt FastSLAM to **Occupancy Grid Mapping**:

1) **Sampling Motion** estimates the current pose x_t given the k^{-1} particle previous pose and the current controls u:
$$p(x_t|x_{t-1}^{[k]}, u_t)$$
2) **Map Estimation** estimates the current map m_t given the current measurements, the current k^{th} particle pose, and the k^{-1} particle map:
$$p(m_t|z_t; x_t^{[k]}, m_{t-1}^{[k]})$$
3) **Importance Weight** estimates the current likelihood of the measurement given the current kth particle pose and the current kth particle map:
$$p(z_t|x_t^{[k]}, m^{[k]})$$

2.6 GraphSLAM

GraphSLAM represents the SLAM problem with graphs composed of:

1) **Poses**.
2) **Features** from the environment (landmarks).
3) **Motion constraints** which tie together two poses.
4) **Measurement constraints** which tie together a feature and a pose.

GraphSLAM performs graph optimization minimizing the error in all the constraints in the graph using a principle known as **Maximum Likelihood Estimation (MLE)**.

Likelihood is complementary to probability. Probability estimates the outcome given the parameters while likelihood estimates the parameters that best explain the outcome. In the case of SLAM likelihood estimates the most likely state and feature locations given the motion and measurement observations.

MLE can be solved analytically, but is computationally very expensive. Though not as accurate as the analytical approach solving MLE numerically using gradient descent is much more computationally feasible. It is this algorithm that is used to tackle SLAM.

GraphSLAM can be viewed as a 5 step process:

1) Construct a graph.
2) Define the constraints.
3) Optimize to solve the system of equations.
4) Linearize using calculus, as most motion and measurement constraints are non-linear.
5) Iterate.

Graph-SLAM complexity is **linear with the number of nodes**, which increases with the size of the map. **Loop closure identifies when areas of the environment are revisited when mapping**. It identifies that images and locations have already been visited instead of identifying them as new locations. Without loop closure the map is not an accurate representation of the environment. With loop closure the map is significantly smoother and is an accurate representation of the environment. RTAB-Map is used in this project to find loop closures.

2.7 Real-Time Appearance Based Mapping (RTAB)

RTAB-Map (Real-Time Appearance-Based Mapping) is a RGB-D Graph-Based SLAM approach based on an incremental appearance-based loop closure detector. It is composed of Front-end and Back-end.

When a loop closure hypothesis is accepted a new constraint is added to the graph. A graph optimizer then minimizes the errors in the map. When a loop closure is detected, errors introduced by the odometry are propagated to all links thus correcting the map. **RTAB-Map uses only odometry constraints, not landmarks to optimize loop closure constraints.**

3 SLAM ROS Package Configuration

3.1 Robot configuration

The Robot chosen was the Willow Garage TurtleBot definition found at https://github.com/turtlebot/turtlebot.git.

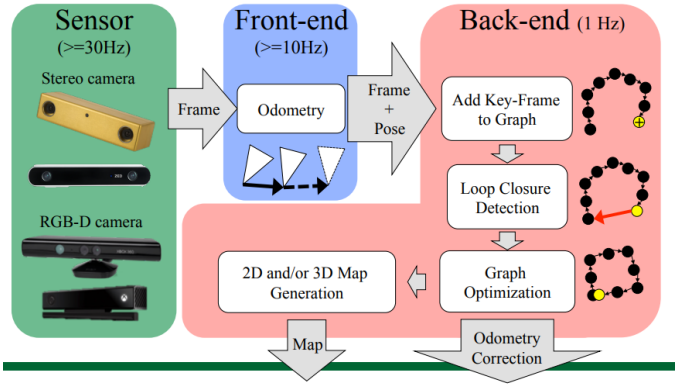

Fig. 3. RTAB Map Structure : http://introlab.github.io/rtabmap/

The Kinect RGBD camera was added to the robot definition from course materials to allow the use of depthimage to laser scan package. This robot was chosen due to having the most ideal sensor placement. It is placed high, above ground level to capture the most amount of feature in the world.

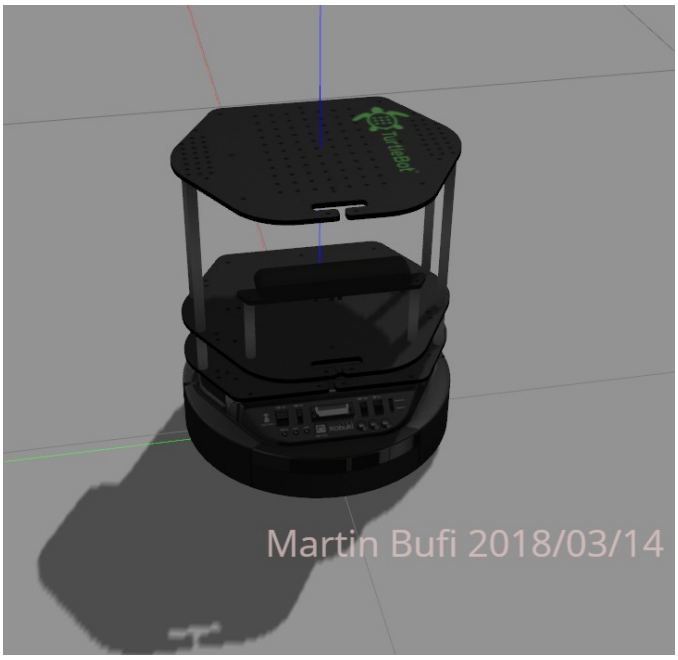

Fig. 4. TurtleBot in empty world

3.2 World Configuration

Both worlds (kitchen and Martinworld) were shown earlier in the report. The Martin world in particular was created with the OSRF Elevator layout. This map includes some desks, standing humans and a surprise Clearpath Husky robot.

3.3 Launch Files

1) **slam project world.launch**: includes the turtlebot robot description launch file, the world file to use, the Depth-Image-To-Laser-Scan Nodelet, interactive markers and spawns the robot in gazebo.

2) **mapping.launch**: launches RTAB-map (Real-Time Appearance-Based Mapping) to produce the map and find closures.
3) **rviz.launch**: launches RViz with the custom robot slam.rviz launch configuration provided as Student Materials.
4) **teleop.launch**: launches the teleop service for robot movement, as provided in Student Materials.

4 RESULTS

The simulations show the actual perfomance of the robots. All simulations were done in Gazebo.

4.1 Kitchen world

At the start of simulation, the robot will stand stationary (until controlled via teleop) while collecting information from the RGB-D camera based on what is directly in front of it.

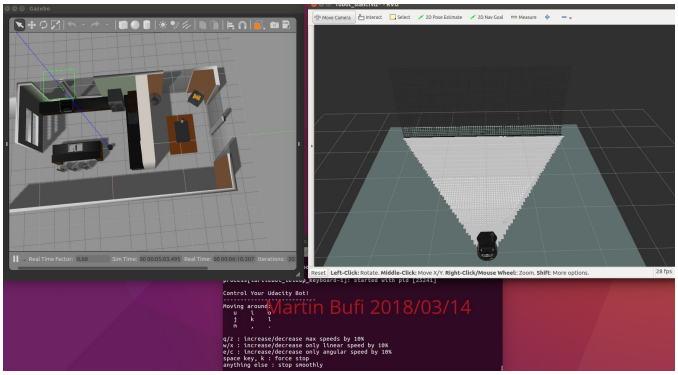

Fig. 5. TurtleBot initial scan

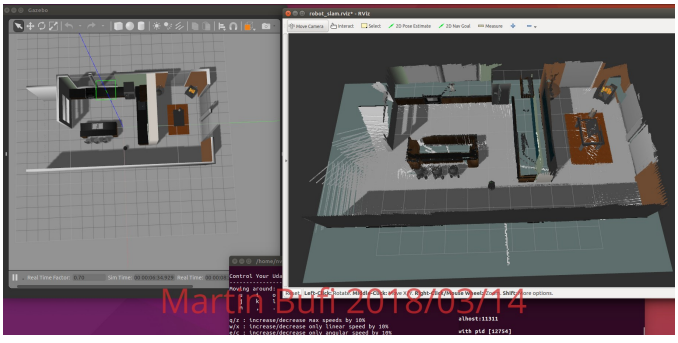

Fig. 6. TurtleBot final slam map

As the robot moves around the room, more data is obtained, filtered, and used to continue creating the 2D and 3D maps.After the robot walks several rounds, the final 2D/3D map.

4.2 Martin world

The robot performed equally well in the martin world. The world had many more features so it took longer to get an appropriate map of most of the environment.

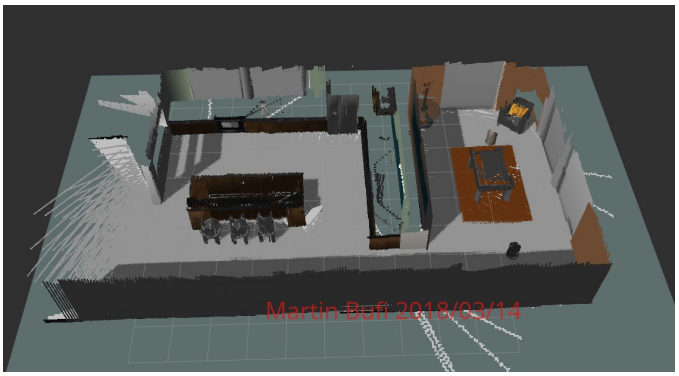

Fig. 7. Final map amplified

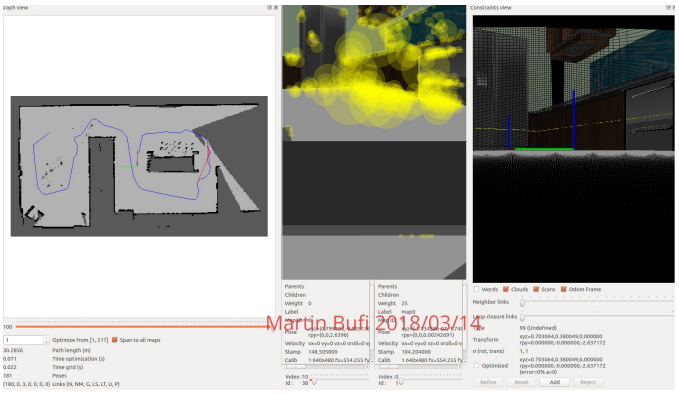

Fig. 8. Kitchen database with 2D map

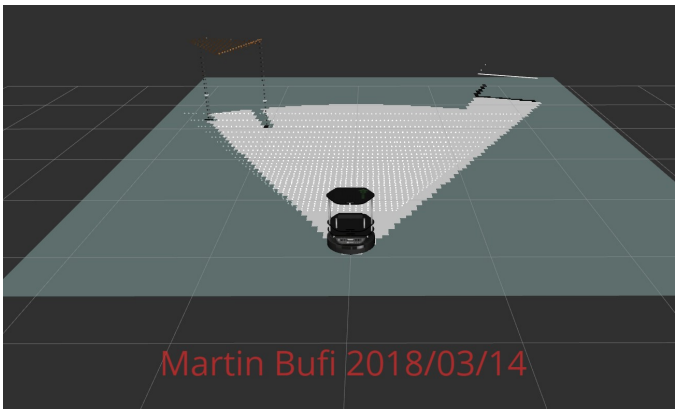

Fig. 9. TurtleBot initial scan

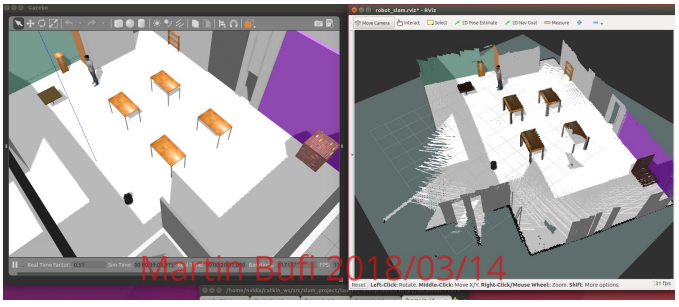

Fig. 10. TurtleBot mid mapping slam map

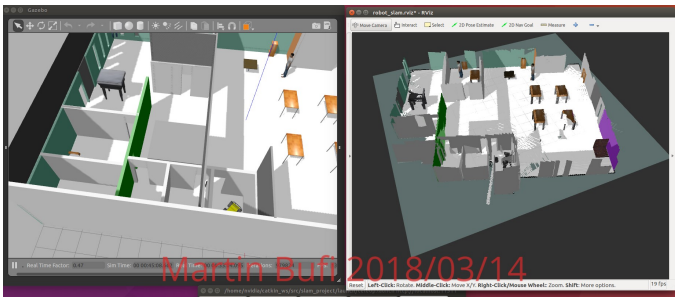

Fig. 11. Final RTAB map

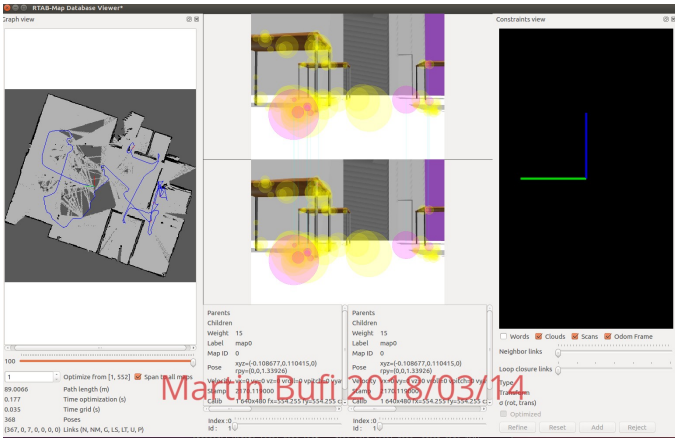

Fig. 12. Martin world database with 2D map

5 Discussion

The ROS package successfully meets all of the project requirements:

1) Creating a ROS package that is able to launch your robot.
2) Have it map the supplied environment with the models and launch files.
3) A custom Gazebo world is to be created and a 3D map for that environment.
4) At least 3 closures should be identified and recorded in the rtab.db database

While GraphSLAM does solve the problem of simultaneous Localization and Mapping, it doesnt work in environments that are very similar and have very little features such as a maze. It also does not work very well for dynamic environments. It only works very well for size constrained, static environments.

As well, the map database that is created/added to each run, tends to get large very quickly. This can be seen as an issue on some systems with memory constraints. This Real-time image mapping does not seem to scale very well for understanding objects that are very far vs. very close. This sometimes leads to distorted maps.

6 Conclusion / Future Work

The GraphSLAM algorithm used by RTAB Map can successfully perform localization and mapping quickly in the Jetson TX2 environment. While the FPS for big environments can be very low in Gazebo, RVIZ does not seem to suffer the same fate. SLAM is important for the typical case where a robot is placed in an environment where a map is not given at the start.

The RTAB-Map algorithm performs the important function of detecting closures - areas that the robot has already visited to ensure that mapping points that have already been mapped are not repeated. Mapping worked best in environments that were varied - meaning few hallways or rooms that had a similar appearance. The four desks in martin world had a similar appearance and pattern that made it difficult to get correct closures. Upon viewing it multiple times from different angles, the robot began to distort the desks in its map. Future work will include more localization tasks, and implementing object segmentation to allow it to store the locations of each of the important landmarks it sees.