Map My World

Jeremy Olsen

Abstract—A review for the third project for Term 2 of the Robotics Software Engineer Nano-Degree. The lessons cover the topics of mapping and Simultaneous Localization and Mapping (SLAM). The topic of mapping covers the Occupancy Grid Map algorithm, and FastSLAM and GraphSLAM as flavors of SLAM. The development portion of the project utilizes the ROS and Gazebo environment from the last project and a provided world is mapped using the *rtabmap* library that has to be debugged before it will work. In addition, a custom world is created and also mapped. The results of the project are favorable as all mapping requirements were met, however there is much more to learn about *rtabmap*.

Index Terms—Robot, IEEEtran, Udacity, LaTeX, SLAM, Mapping, RTAB-Map.

1 Introduction

The ability for a robot to be able to localize itself, inside of an environment, with a map it has created using it's sensor data, is known as SLAM (Simultaneous Localization and Mapping). In the last project, the focus was on learning localization where the robot was provided a map of it's environment and will determine it's pose from that information. In this project, the material covers how a robot should obtain that map and then moving into different flavors of SLAM. In addition, further discussion into when and how to optimize for the different memory and processing challenges that will be encountered at each step of the process.

1.1 Mapping Problem

Maps of the surrounding environment can be extrapolated from sensor data, like cameras and range finders. Mapping can otherwise be though of as an estimate of an environment using an assumed pose. Challenges to mapping, such as known issues like noise and size are ever present. New challenges, such as perpetual ambiguity, add additional complexity and computational demands.

1.2 SLAM Problem

SLAM essentially is combining the results of the two prior problems of localization and mapping simultaneously. This is considered a bit of a chicken and the egg problem as neither the pose or the map are provided. This estimation processing has many different flavors and incorporates previously learned algorithms that address these challenges such as EKF (Extended Kalman Filter), and particle filters such as MCL (Monte Carlo Localization. [1]

The development part of the project then implements the *rtabmap* ROS package within a simulated Gazebo environment which makes use of the mapping techniques that were just introduced.

2 Background

The localization problem is solved by using a provided map which will estimate the pose of the robot within that map. Maps may not always be available and there are significant advantages of obtaining the map at the point of localization. Most environments are constantly changing and static environments are rarely encountered and the need to perform instantaneous mapping is required even if a priori map is available.

Mapping has many difficulties and challenges that are inherent due to the structure of the map. Some items to include are:

- **Unknown Map and Poses:** needs to be obtained to complete the task of localization.
- **Continuous and huge hypothesis space:** can contain an infinite number of variables to describe it. This space can be highly dimensional since maps are defined over a continuous space.
- **Sensor noise:** either inherent noise from the sensor or from the surrounding environment.
- **Perceptual Ambiguity:** cases where the environment contains similar geometry and repetitiveness.
- **Cycles:** the task of revisiting at different points in time and remapping the environments to correlate location.

2.1 Mapping with Known Poses

Posterior probability is the revised probability of an event occurring after taking into consideration new information. [2]

The problem of generating a map under the assumption that the robot poses are known and non-noisy is referred to as mapping with known poses. The problem can be represented with $p(m|z_{1:t}, x_{1:t})$ function and compute the posterior over the map given all the measurements up to time t and all the poses up to time t represented by the robot trajectory. [3]

This is in contrast this to the SLAM problem, which changes the problem to mapping with unknown poses. When determining the output of mapping with known poses, SLAM is run first and it's output poses are filtered out to be used for the known poses.

2.2 Occupancy Grid Algorithm

The mapping problem can be addressed by the Occupancy Grid Mapping Algorithm. The basic idea of the Occupancy Grid is to represent a map of the environment as an evenly spaced field of binary random variables each representing the presence of an obstacle at that location in the environment. [4]

The goal of an occupancy mapping algorithm is to estimate the posterior probability over maps using the $p(m|z_{1:t}, x_{1:t})$ function, which is representing mapping with known poses. The controls and optometry data play no part in the occupancy grid mapping algorithm since the path is assumed known. [5]

$$
\begin{aligned}
&Algorithm\ Occupancy\_Grid\_Mapping(\{l_{t-1,i}\}, x_t, z_t):\\
&\ for\ all\ cells\ m_i\ do\\
&\ \ if\ m_i\ in\ perceptual\ field\ of\ z_t\ then\\
&\ \ \ \ l_{t,i} = l_{t-1,i} + Inverse\_Sensor\_Model(m_i, x_t, z_t) - l_0\\
&\ \ else\\
&\ \ \ \ l_{t,i} = l_{t-1,i}\\
&\ \ end\ if\\
&\ endfor
\end{aligned}
$$

The Occupancy Grid Algorithm, as shown above, implements a Binary Bayes filter to estimate the occupancy value of each cell. It takes in the previous occupancy values of the cell, poses and measurements as input parameters into the function and starts to loop through each of the grid cells. For each cell, a test to find out if the cells are currently being sensed by a range finder sensor and computed their belief using the Binary Bayes filter algorithm. This is computed by adding the previous belief to the inverse sensor model. The inverse sensor model represents the log odds ratio of the probability of the posterior map, giving the measurements and the poses, and subtracting the initial belief, which is the initial state of the map in it's log odds form. The cells that are not being measured, their occupancy value remains unchanged. In ending, the function returns the occupancy values of the cells and another cycle starts. [6]

2.3 SLAM

As was discussed earlier, when solving the problem of SLAM, the poses and the map are not provided and must be extrapolated from the sensor information using knowledge from what was learned from mapping and localization. There are two key features that account for the solution. First feature is the 'forms' that the SLAM problem take and the second, the 'nature' of the SLAM problem. The forms are divided into the Online SLAM problem and the Full SLAM problem.

- **Online SLAM:** computes a posterior over the current pose along with the map.
- **Full SLAM:** computes a posterior over the entire path along with the map. [7]

Next feature is the nature of the SLAM problem. It is composed of two elements, which are the continuous and the discrete elements.

- **Continuous:** Robot continuously senses its pose and the location of the objects.
- **Discrete:** Robot has to identify if a relation exists between any newly detected and previously detected objects. It essentially is always answering the question, "Have I been here before?". [7] [8]

The discrete element leads us to the concept of correspondence and is explicitly included in the estimation problem of the posterior. The correspondence problem refers to the problem of ascertaining which parts of one image correspond to which parts of another image. The advantage of adding the correspondence to the calculation of the posterior is to help the robot better understand where it is located by creating a relationship between objects. These correspondence values are highly dimensional and can increase exponentially over time since the robot will be continuously sensing the environment. Because of the large number of outcomes, it is considered infeasible to use unknown correspondence to calculate the posterior and SLAM algorithm must rely on approximation for computational efficiency. [9] [10]

2.3.1 FastSLAM

FastSLAM solves the Full SLAM unknown correspondence problem by using known correspondences. By using a particle filter approach, solving through the Monte Carlo Localization (MCL) algorithm, it is able to estimate a trajectory. Next it is able to estimate the map by using a Extended Kalman Filter (EKF) to solve independent features that are of a Gaussian nature. The custom approach of representing the posterior with particle filter and Gaussian is known by the Rao-Blackwellized particle filter approach. [11]

FastSLAM is capable of solving both the Online SLAM and Full SLAM problems as it solves both the full robot path and estimates instantaneous poses. Over time, the FastSLAM algorithm has evolved from versions 1.0 & 2.0 to Grid-based FastSLAM which adapts FastSLAM to grid maps and the previously discussed Occupancy Grid Map algorithm. As can be seen in the included image, the Occupancy Grid Mapping is a component of the MCL algorithm, which is used in localization problems.

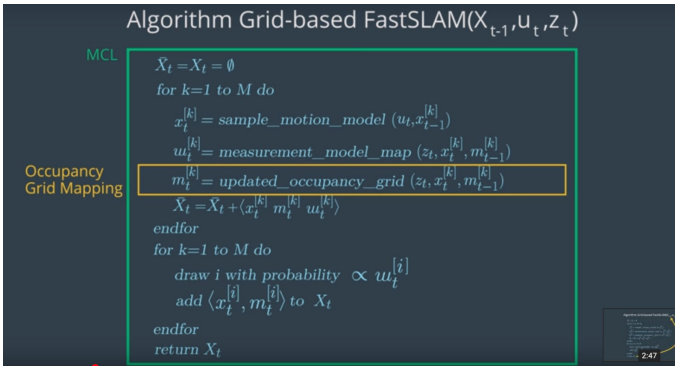

Fig. 1. FastSLAM Algorithm [12]

2.3.2 GraphSLAM

In robotics, GraphSLAM is a simultaneous localization and mapping algorithm which uses sparse information matrices

produced by generating a factor graph of observation inter-dependencies (two observations are related if they contain data about the same landmark). [4]

GraphSLAM solves the Full SLAM problem which will recover the entire path and map versus the most recent pose and map. This allows the consideration of dependencies between the current and previous poses.

Graphs

The algorithm uses graphs represent the robot's poses it's environment. The following descriptions are based upon the included image graph example.

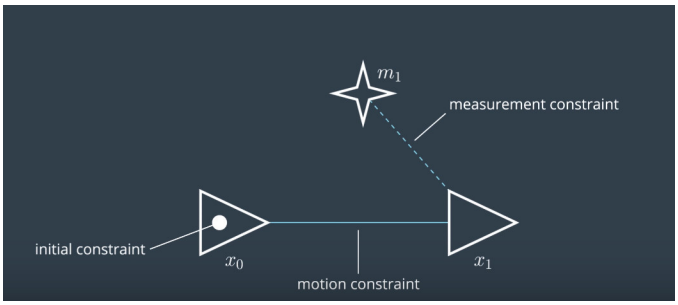

Fig. 2. GraphSLAM graph example [13]

Poses are represented by triangles and contain it's position and orientation and are otherwise known as a node. The first node is at time-step zero and is arbitrarily constrained to zero. The position at time-step 1 is represented by a second node and would be connected to the first by a motion constraint that is also called an edge or arc, which is shown by a solid line. Features that are in the surrounding environment are represented by stars and are tied together by measurement constraints which are represented by a dashed line. [13]

Constraints

Soft constraints, in which soft is defined by the fact there is some amount of error involved, it can be compared to the example of a rubber band or a spring. When no external forces being acted upon them, they will bring the system into a particular resting configuration. As more nodes and constraints are added to the graph, they will pull the system closer to that constraint's desired state. The over-all goal is to find the node configuration that minimizes the over all error present. [14]

Front-End vs. Back-End

- **Front-End:** The front-end of GraphSLAM looks at how to construct the graph using the odometry and sensory measurements collected by the robot.
- **Back-End:** The back-end of GraphSLAM 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. [15]

Maximum Likelihood Estimation (MLE)

Graph optimization is the process of minimizing the error present in all of the constraints in the graph using MLE. Likelihood tries to estimate the parameters that best explain the outcome and when applied to SLAM it tries to estimate the most likely configuration of state and feature locations given the observations of motion and measurement. [16]

Information Matrix and Vector

To compute a map posterior, GraphSLAM linearizes the set of constraints. The result of linearization is a sparse information matrix and an information vector. The sparseness of this matrix enables GraphSLAM to apply the variable elimination algorithm, thereby transforming the graph into a much smaller one only defined over robot poses. The path posterior map is then calculated using standard inference techniques. [17]

Inference

The path and map can be recovered by running the following operation after the matrix and vector has been completed:

$$\mu = \Omega^{-1}\xi$$

Of course, neither the graph representation nor the information matrix representation gives us what we want: the map and the path. In GraphSLAM, the map and the path are obtained from the linearized information matrix and the information vector ξ , via the equations $= -1$ and $\mu = \xi$. This operation requires us to solve a system of linear equations. [17] [18]

Non-linear Constraints

Its a fact that a robot and it's surrounding environment does not always produce Gaussian distribution results, as the idea would be quite limiting, it becomes important to be able handle non-linear data and models. The Jacobian way of linearizing data that was covered when learning localization is applicable in these situations. A Taylor Series approximates a function using the sum of an infinite number of terms. A linear approximation can be computed by using only the first two terms and ignoring all higher order terms.

Non-linear constraints can be linearized using Taylor Series, but this inevitably introduces some error. To reduce this error, the linearization of every constraint must occur as close as possible to the true location of the pose or measurement relating to the constraint. To accomplish this, an iterative solution is used, where the point of linearization is improved with every iteration. After several iterations, the result, μ, becomes a much more reasonable estimate for the true locations of all robot poses and features. [19]

2.3.3 RTAB-Map

Real-Time Appearance-Based Mapping is otherwise known as RTAB-Map is used in the development portion of the project as the basis to mapping the Gazebo environments. Appearance-Based SLAM uses data collected from vision sensors (depth cameras) to perform SLAM with. It's RGB-D Graph SLAM approach is based on a global Bayesian loop closure detector, which is the ability to determine if the robot has visited a location previously. RTAB-Map is optimized to perform loop closure in real-time, which is the ability to obtain the result before the next camera image is captured. [20]

Loop closure is computed using a bag-of-words approach in RTAB-Map. A feature is a very specific characteristic of a image. Example would be a building corner, or a patch with a texture. Default method for extracting features from images is Speeded Up Robust Features (SURF). SURF descriptors have been used to locate and recognize objects, people or faces, to reconstruct 3D scenes, to track objects and to extract points of interest. [21]

The main idea bag-of-words is to use SURF to extract feature and create feature descriptors by splitting the images up to categorize the sub-regions of the image. The similar features, or synonyms, are then clustered together and then represent a vocabulary. The feature descriptors are mapped to the vocabulary to perform quantization and can be referred as a visual word. The image becomes a bag-of-words when all of the features have been quantized.

Each word keeps track of which image it has been associated with by keeping a score, which is called an Inverted Index. If a word has been seen in a particular image, that score will increase. A Bayesian filter is used to evaluate the scores and a predetermined threshold is reached, a loop closure has been determined to have been detected. [22]

Memory management a key feature of RTAB-Map and allows for loop closure to be done in real time. The overall strategy is to keep the most recent and frequently observed locations in the robots Working Memory (WM), and transfer the others into Long-Term Memory (LTM). [23]

2.4 Comparison / Contrast

There is a lot of overlap between the various algorithms and how they are used to solve the problems associated with mapping and localization. One of those overlapping features is that all the algorithms are not limited by dimensionality, which means that we can use them in 2D or 3D data environments.

The top level differences between the algorithms that were discussed in the lessons are structural and are specific to the localization and mapping problem of SLAM. For example, the Occupancy Grid Algorithm is solely focused upon mapping, while FastSLAM, and GraphSLAM cover both mapping and localization problems.

From here the algorithms can be further dissected into the two covered SLAM approaches and their process differences. Both solve the problems of obtaining a map and pose of the robot. The FastSLAM approach is combination of using MCL for solving the localization problem and the Occupancy Grid Algorithm as the mapping solution. GraphSLAM on the other hand uses the concept of graphs, which are essentially features and poses that are bound by constraint and ultimately represented by a sparse information matrix and vector.

Further comparing both SLAM approaches, we know that FastSLAM uses particles to determine it's location, at any point in time. Since it only looking at one point in time, it's possible that there isn't a particle in the most likely location. GraphSLAM solves the full SLAM problem, it uses it entire trajectory as it's state and can use all of the data to determine the best possible solution.

3 Simulation Configuration

Much like the last project, the focus is built around using a simulated environment. A lot of the structure that was built for the last project is carried over to this project. The custom robot is also reused and is performing the task of mapping this new world.

Fig. 3. Custom 4-wheel robot in custom Gazebo world

Much like last project, the development portion was expanding ROS and Gazebo skills into the areas of debugging and creating a custom environment for testing. The model was also fitted out with a Kinect RGB-D camera to capture depth data. The custom model already contained a laser depth sensor. It was optional to either use that or the camera's depth sensor during the project and the laser sensor was chosen. The teleop package was utilized to drive the model around the maps since this was not a localization and path planning project, which was optional.

Due to the performance requirements of this project, it was suggest in the Slack channels to use the Jetson TX2. There were a couple of extra steps involved as ROS and some of it's dependencies required some additional setup and compilation for the ARM processor.

The project provided a basic mapping launch file which implemented *rtabmap*, and a couple of key changes were omitted for the sake of learning debugging techniques. The first on used was 'roswtf', which gives an overview of all of the topics that are in use and not in use. The *rqt* package contains many different tools. The 'rqt_graph' will produce a graphical overview of the topics and services connections. And 'rqt_console' will capture and show messages and logs from the nodes that are in use. Lastly debugging with transform frames produces a graph that gives an overview of the URDF model linkages in a tree view. The output of the 'tf view_frames' is provided as a part of the project files named *frames.pdf*.

The first set of changes that was performed was on the URDF model for swapping for the Kinect camera and getting the model structure right to map to *rtabmap*. In addition, the camera needed to have an extra link and rotation applied so it was facing the correct direction.

The next set of changes were to the create the launch files for mapping and RViz. When these were launched, there were errors that were occurring in the console that needed to be followed. The first change needed to increase

the 'queue_size' and gave better performance. This helped eliminate all of the 'noise' to be able to tackle the remaining connection problems.

The additional issues all centered around the mapping of the topics inside *mapping.launch*. The following additions and changes were made to get the project fully working.

```
<!-- Change to match URDF -->
<param name="frame_id"
type="string" value="robot_footprint"/>

<!-- Missing mappings from provided launch file
<remap from="odom" to="/odom"/>
<remap from="scan"
to="/udacity_bot/laser/scan"/>
```

Once the project was working, navigating around using the keyboard to capture the map and learning how to drive the rover was the biggest challenge left to overcome.

3.1 Three Rooms World

The provided world consisted of a simple three room layout. The middle room was a kitchen, the side room to the right when facing the refrigerator resembled a sitting room, and the room to to the left was empty.

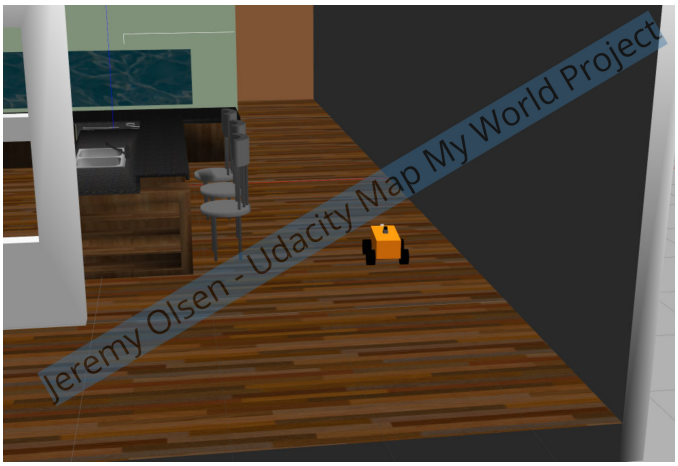

Fig. 4. Custom robot in Three Rooms Model

3.2 Custom World

The custom world is a bit larger then the provided world is laid out resembling a town square with concrete barriers as the edges. There are a few different type of structures that are in the middle of the square, including the mars moon rover a fountain, a post office with boxes, and various other buildings. The ground is all asphalt.

3.3 Results

Once *rtabmap* node was running, the mapping was fairly smooth on the TX2 once the high performance scripts were executed. The provided three room world was mapped first using the defaults provided by the *mapping.launch* file.

Driving with the keyboard required a fair about of care and learning. During the mapping process, if the robot would run to fast into an obstacle and get tipped around, the map would have a tendency to glitch and not show the environment correctly. This required trial and error approach to finding correct speeds and optimal paths to get a good looking map.

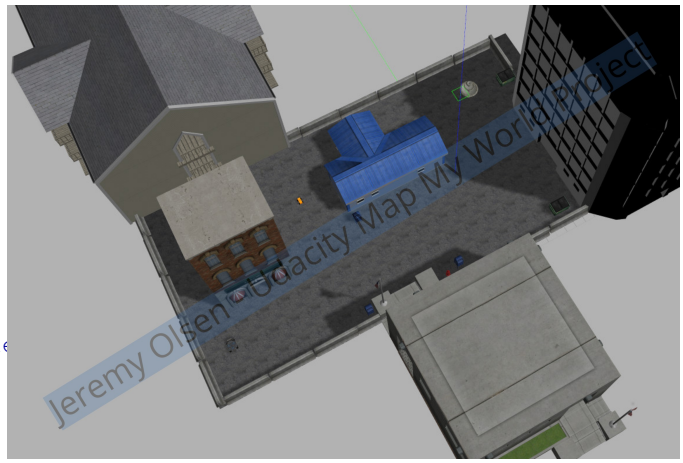

Fig. 5. Custom Gazebo world

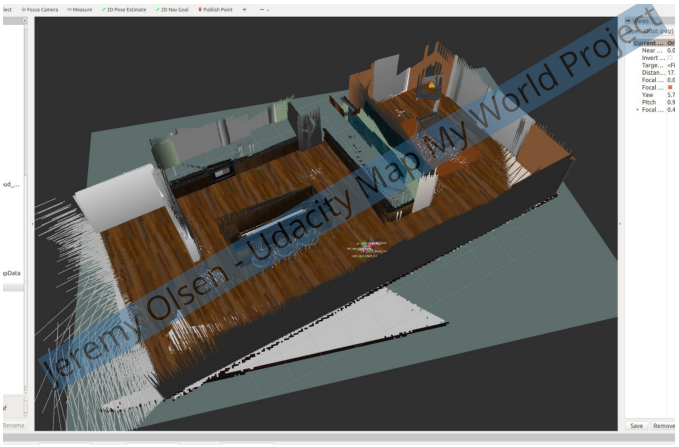

Fig. 6. Three Rooms Map using *rtabmap*

The custom world went through a lot of different iterations. Some of the models in the Gazebo database were unstable and would suddenly disappear and/or move after re-starting Gazebo. Once the driving and mapping started, the size of the rover started to be noticeably smaller compared to the rest of the environment and was scaled up to be larger so that it could cover more territory quicker. Also it was observed that mapping was more difficult in the larger space using the *rtabmap* defaults that were provided. Various changes to the rtabmap parameters were experimented with by researching outside tutorials. The best results were obtained by using the settings that are currently in the mapping.launch file.

4 Discussion

The depth of parameters that *rtabmap* supports is huge and is a black box that needs more research. The lessons covered most of the theory of how *rtabmap* works, but didn't dive

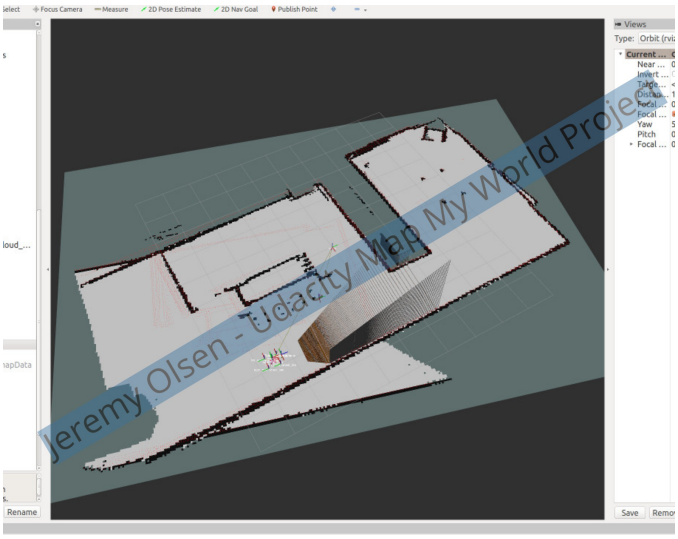

Fig. 7. Three rooms Occupancy Grid Map using *rtabmap*

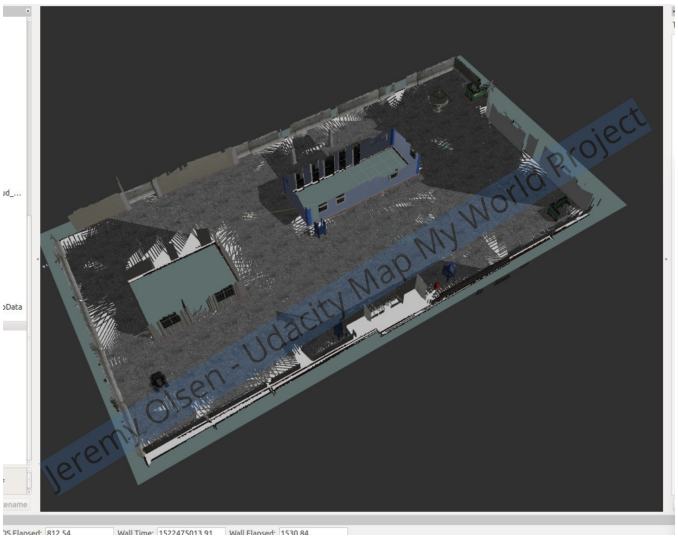

Fig. 8. Custom Model Mapped using *rtabmap*

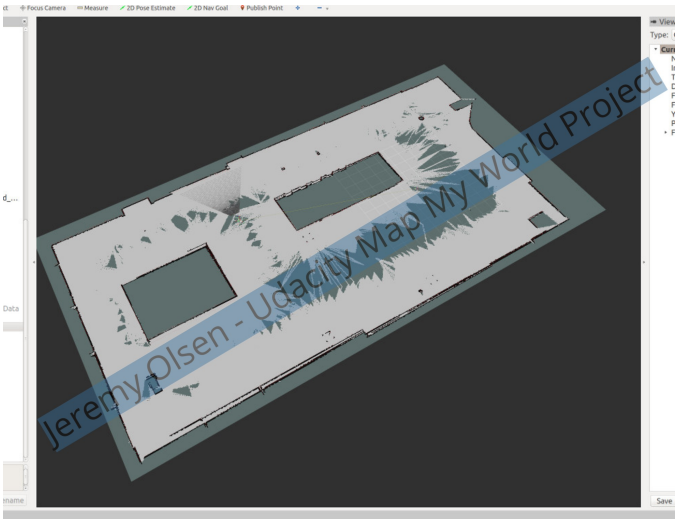

Fig. 9. Custom Model Occupancy Grid Map using *rtabmap*

into the inner working of the package, particularly with all of the possible configurations and best practices that would be encountered in the wild. The only changes that were experimented with was changing to a setup that was found in the ROS *rtabmap* tutorial which seemed to yielded much better results. The most significant improvement seemed to come from changing the strategy parameter, 'Reg/Strategy' from Visual to ICP (Iterative Closest Point).

In the Iterative Closest Point or, in some sources, the Iterative Corresponding Point, one point cloud (vertex cloud), the reference, or target, is kept fixed, while the other one, the source, is transformed to best match the reference. [24]

When reviewing the results of the database inside of *rtabmapviz*. The results from the default mapping file contained up to 30 global loop closures. When using the updated mapping method which is using ICP, the number of loop closures were reduced to 10. This may be a good thing as it seems that sometimes when the number of closures are too many, that will create undesirable results in the final map. According to a GitHub tutuorial, in some cases, enabling ICP will increase precision of the loop closure and the visual odometry. However for some other cases, it will add difficult detectable errors. [25]

This was the first Gazebo project that the TX2 was encouraged to be used for. This provided an opportunity to setup a full development environment and finally master all of the high performance tweaks that are available. ARM processors have their own challenges when it comes to usability from a development standpoint. For example, Java based IDE's like PyCharms are able to to be run after installing the correct JVM based for ARM processors, where as programs like Sublime Text do not support ARM builds. Finally, after spending months from moving up from a VM, to dual booting, the TX2 is now fully capable of running Gazebo / RViz and ROS simultaneously with high performance and smooth rendering.

5 Conclusion / Future work

rtabmap is a fully capable SLAM framework that creates beautiful maps when needing to visually inspects a map arises. Much more research is required to fully understand the solution and to optimize it for real-world scenarios. One of the final tasks that was optional for this project was to hook-up localization by using the output of the mapping into the localization project setup.

The ultimate goal would be to use *rtabmap* to navigate a real robot. The first steps to proving out a real-world scenario would be to convert the mapping layers to work with a live camera instead of the simulated one. A good example application that this would be useful for is for a competition where a robot, (in our case a 4-wheel drive RC car), needs to find and touch traffic cones that we know the position for as fast as it can.

The challenges with this particular competition is that the robot cannot enter the course before hand. Full SLAM would be the problem that needs to be solved, however SLAM requires some amount of time to be performed and performance may be a factor in trying to achieve the fastest speed. SLAM would then have to be utilized in conjunction of strategies to achieve the fastest time.

REFERENCES

[1] Udacity, "Lesson 15: Intro to mapping and slam: Section 1: Introduction: video," 2018.
[2] Investopedia, "Posterior probability," 2018.
[3] Udacity, "Lesson 16: Occupancy grid mapping: Section 5: Posterior probability," 2018.
[4] B. W. Thrun, S. and D. Fox, "Probabilistic robotics," *Cambridge, Mass: MIT Press*, 2005.
[5] Wikipedia, "Occupancy grid mapping," 2018.
[6] Udacity, "Lesson 16: Occupancy grid mapping: Section 5: Occupancy grid mapping algorithm: video," 2018.
[7] Udacity, "Lesson 17: Grid-based fastslam: Section 5: Correspondence," 2018.
[8] Udacity, "Lesson 17: Grid-based fastslam: Section 5: Nature of slam," 2018.
[9] Wikipedia, "Correspondence problem," 2018.
[10] Udacity, "Lesson 17: Grid-based fastslam: Sect 6. slam challenges," 2018.
[11] Udacity, "Lesson 17: Grid-based fastslam: Sect 8. introduction to fastslam," 2018.
[12] Udacity, "Lesson 17: Grid-based fastslam: Sect 12. the grid-based fastslam algorithm: video," 2018.
[13] Udacity, "Lesson 18: Graphslam: Sect 2. graphs: video," 2018.
[14] Udacity, "Lesson 18: Graphslam: Sect 3. constraints: video," 2018.
[15] Udacity, "Lesson 18: Graphslam: Sect 4. front-end vs back-end: video," 2018.
[16] Udacity, "Lesson 18: Graphslam: Sect 5. maximum likelihood estimation," 2018.
[17] M. M. Thrun, S., "The graphslam algorithm with applications to large-scale mapping of urban structures," *THE INTERNATIONAL JOURNAL OF ROBOTICS RESEARCH*, 2006.
[18] Udacity, "Lesson 18: Graphslam: Sect 11. inference," 2018.
[19] Udacity, "Lesson 18: Graphslam: Sect 12. nonlinear constraints," 2018.
[20] M. Labbe and F. Michaud, "Appearance-based loop closure detection for online large-scale and long-term operation," *IEEE Transactions on Robotics*, vol. 29, no. 3, pp. 734–745, 2013.
[21] Wikipedia, "Speeded up robust features," 2018.
[22] Udacity, "Lesson 18: Graphslam: Sect 16. visual bag-of-words: video," 2018.
[23] Udacity, "Lesson 18: Graphslam: Sect 17. rtab map memory management," 2018.
[24] Wikipedia, "Iterative closest point," 2018.
[25] G. matlabbe, "Icp [tutorial]," 2018.