Map My World Robot

Vladislav Artamonov

Abstract—In Robotics, localization and mapping are the key skills that enable robots to interact with their environment, by being able to locate itself relative to the map of previously visited locations. Mapping a location (e.g. an apartment room or an outdoor area) from sensor data while locating oneself in that consistently updating environment is referred to as Simultaneous Localization And Mapping (SLAM). In this project the SLAM implementation of Real-Time Appearance Based Mapping (RTABMap) is done through the rtabmap_ros package and implemented on a teleoperated robot to map 2 simulated worlds made in Gazebo. Both worlds were successfully mapped, which allows for a first look at some considerations to be made when attempting to map an environment using robots.

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

1 Introduction

The goal of this project is to map two environments using the RTABMap algorithm implemented on. Mapping in robotics enables a robot to keep track of the environment it is in.

More often than not, the architect blueprint of a building is inaccurate, and even in the event that it was perfectly accurate, the addition of furniture would alter the "map" of the building by the robot, since if it were to perform a task within that environment, it would need to know about the position of those furnitures. This problem is interesting because that means that every building needs to be mapped if a robot is to do anything within them, and this statement can be pushed further encompass any kind of environments. However ideally the robot should be adapted for the environment it is mapping, and therefore if one can make a robot able to navigate and sense in any environments, then every environment can be mapped.

Potential applications can go from mapping an office layout for automated cleaning, mapping a suburb for civil engineers to design upgrades to a city, to mapping a region of the moon for installing a base on the moon. The importance of mapping is similar to how important it is for us to remember places we've been to, and not remembering where anything is. It is a crucial part for any mobile robotic task, without which the robot would never be able to get an understanding of the layout of the environment it is mapping nor would it be able to locate itself within that environment.

2 Background

Mapping in Robotics is used to enable the robot to "visualize" the environment it is in. Locating the robot with respect to its environment requires a map of the environment, and mapping requires the robot's pose, which is obtained during the localization. This is a case of the "chicken and the egg" problem, where location requires a map of the environment and mapping requires the location of the robot.

2.1 SLAM

Simultaneous Localization and Mapping (SLAM) solves this problem by simultaneously and iteratively computing the map (m) and robot's pose (x_t) from sensory and odometric measurements (z_t), control actions (u_t), and correspondences (c_t) between features within the map. There are two different forms of SLAM:

- **Online SLAM**: it solves the SLAM problem by only looking at the current pose and local correspondences to estimate the robot's pose and update the map: $P(x_t, m, c_t|u_{1:t}, z_{1:t})$.
- **Full SLAM**: Unlike the Online SLAM, Full SLAM, also referred to Offline SLAM, takes into account all poses and map correspondences to estimate the pose and update the map: $P(x_{1:t}, m, c_{1:t}|u_{1:t}, z_{1:t})$.

There exist different implementations of the SLAM algorithm, some noteworthy ones are FastSLAM, Grid-Based SLAM and Graph SLAM.

FastSLAM estimates the pose of the robot using particles, like the Monte Carlo Localization algorithm and builds the map using an EKF to solve the independent features of the map as local Gaussians. The main disadvantage of this approach is that it always need to assume that there are known landmarks within the map, and therefore it is impossible to model an arbitrary environment.

Grid-Based SLAM is a variant of FastSLAM which updates the map using grids, like the Occupancy Grid Mapping algorithm, in order to fix the previously mentioned issues with the FastSLAM algorithm. Grid-Based SLAM, like FastSLAM uses a particle filter to estimates the pose.

GraphSLAM on the other hand aims create a graph of robot poses and map features and attempts to minimize the over error between all motion and measurement constraints.

2.2 RTABMap

Real-Time Appearance Based Mapping (RTABMap) or Appearance Based SLAM implements GraphSLAM by collecting data from visual sensors to locate the robot and map the environment. SLAM with RTABMap can be separated into

two parts: the front-end, which uses sensory and odometric data to create a graph with loop closure constraints; the second part, the back-end, handles the graph optimization and the 2D/3D mapping.

2.2.1 Loop closure and bag-of-words

RTABMap uses **loop closure** to determine whether a robot has seen a location before. RTABMap uses Speeded Up Robust Features (SURF) to extract features and describe them with a unique representations then it clusters similar or synonym features together and name them to create vocabulary. One of the feature description is then mapped to one the vocabulary, it is called quantization. Features are now linked to a word and can be referred to as a visual word. When all features in an image are quantified, the image is now a **bag-of-words**. To compare an image with previously seen images, a matching score is given to all images containing the same words using a Bayesian filter, and similar images can be found, this is called an inverted index. Finally if the current image contains similar words to that of a previously seen image (bag-of-words) it will have a higher score, and if the score reaches a threshold H, a loop closure is detected.

2.2.2 Memory management

RTABMap uses a memory management technique to limit the number of locations considered as candidates during loop-closure detection, which allows it to be done in Real-Time. The technique can be summarized as follows: it keeps the most frequently observed locations in the robot's **working memory** (WM) and transfers other images to the **long term memory** (LTM). When an image is first acquired a new node is created in the **short term memory** (STM), features are then extracted and a it is then compared to the vocabulary to find all of the words in the image, creating a bag-of-word for this node. Nodes are assigned a weight in the STM based on how long the robot spent in the ocation, the longer, the higher the weight. STM has a fixed size S, when the STM reaches S nodes the oldest node is transferred to the WM to be considered for loop closure detection. WM size depends on a fixed time T, and when the time required to process new data reaches T, then some nodes are transferred to the LTM. Finally if a loop closure is detected, neighbour nodes in the LTM are transferred to the WM.

3 Scene and Model Configuration

The robot used for this project was an updated version from the previous project. Beside some minor adjustments to the casters, the main modification was replacing the front camera with an RGB-D camera, a Kinect camera, to create a 3D map of the environment (Figure 1).

3.1 World Configuration

3.1.1 Kitchen and Dining - Udacity World

Mapping was done on two different worlds, first the 'kitchen and dining' world provided by Udacity. There are two small rooms, the kitchen with a table in the center and a second smaller room, as seen in Figure 2.

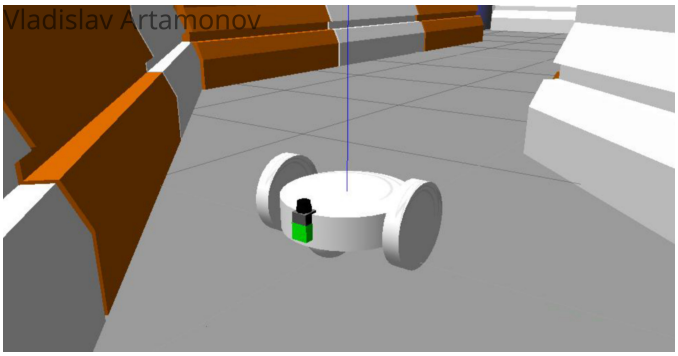

Fig. 1. Robot

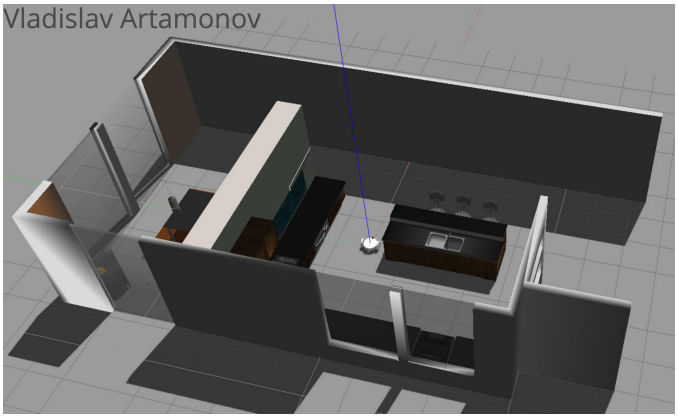

Fig. 2. Kitchen Dining - Udacity World

3.1.2 Playground - Custom World

A second custom world was made in Gazebo using the prefabs to create an outdoor playground to try mapping outdoor and larger locations (Figure 3. A large "empty" area in the middle was added to make it more challenging for the mapping algorithm to find correspondences and detect loop closures and assess how well the robot would manage under such environment.

Fig. 3. Playground - Custom World

4 Results

4.1 Mapping Kitchen and Dining

While mapping the kitchen and dining world, the robot was able to map out the whole area with ease compared to the second world, the smaller size of the environment and its ample amount of features allowed RTABMap to detect enough features after 2 cycles around the rooms to complete

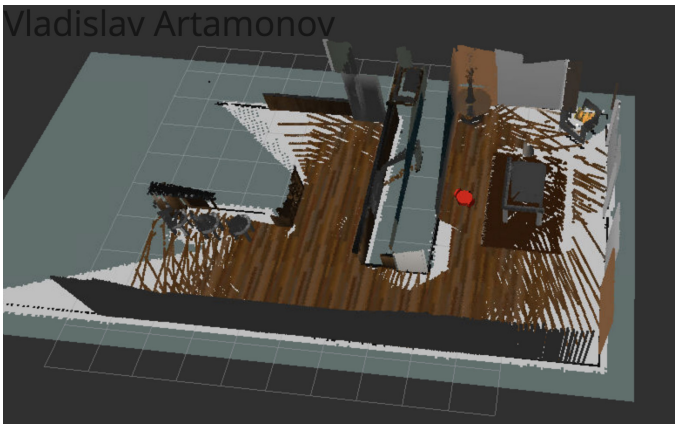

Fig. 4. Mapping the kitchen dining world in RViz

the map. Bumping into walls or doing to many maneuvers to unstuck the robot led to slight variations (objects not appearing where they should be) in the map, but after a few cycles the map corrected and realigned itself without issues.

A total of up to 86 global local closures were detected during the mapping which was enough to fully map out this environment (Figures 5 and 6).

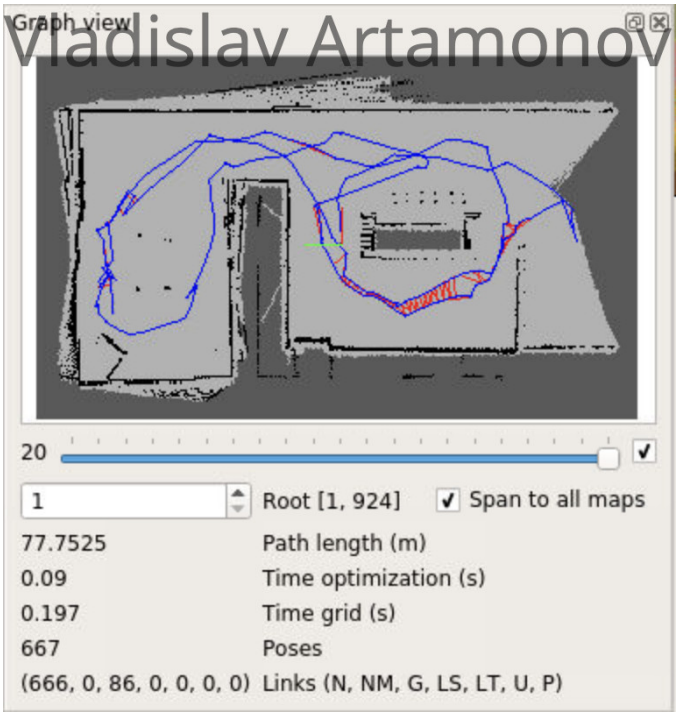

Fig. 5. Graph view of the kitchen dining world

4.2 Mapping the playground

The sheer size of the environment revealed that the robot used to do the mapping should be adapted to the terrain it needs to map. Some of the more obvious consequences one ca observe is that the range of the sensor, the size of the map and the robot's speed and odometric precision influence the time required to map an environment. While mapping the

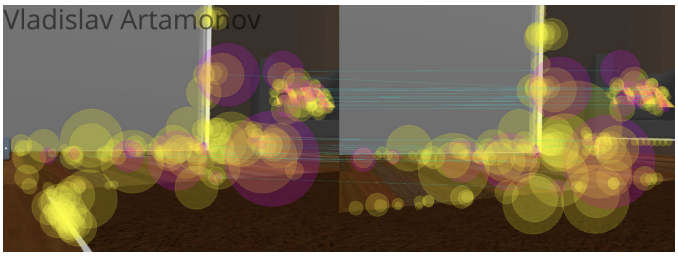

Fig. 6. RTABMap DB Viewer, viewing global loop closures in the kitchen and dining world

first world, some issues weren't as apparent as when the mapping was done in the playground environment, since the map was relatively small, the robot's speed wasn't an issue but when it came to mapping the second environment, it took several minutes for the robot to make a lap around the playground. As a result any noise and uncertainties in the robot's motion compounded over the length of the trip and when the robot finally completed a lap and ended in a part of the playground it had seen before, the robot's erroneous position influenced the mapping and led to some objects being mapped twice in the environment in places that were quite distant from the original one, as seen in the Figure 7.

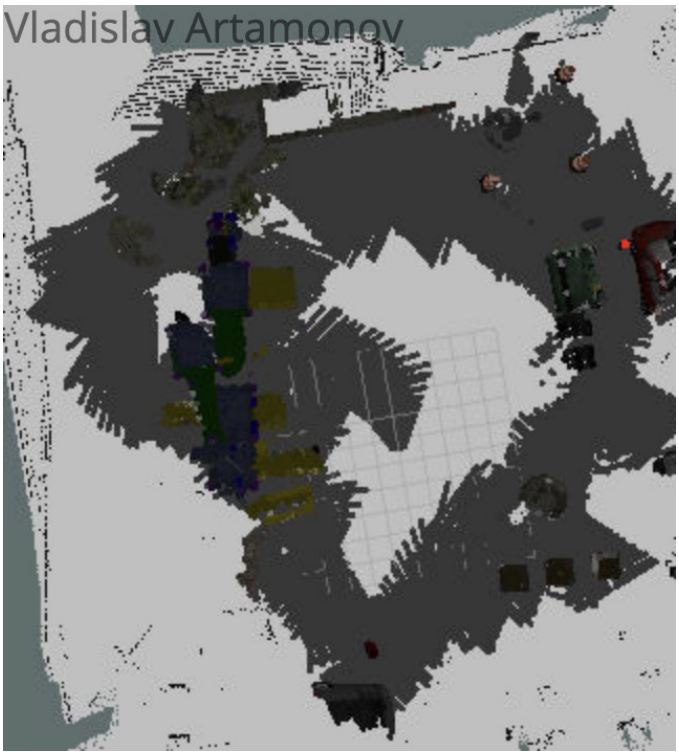

Fig. 7. Playground - errors while mapping

However like in the kitchen and dining world, after a few cycles the map corrected itself as intended (Figure 8) and the mapping process could be completed. This process was quite long and led to the discovery of some "tricks" to speed it up, such as standing still and doing a 360 turn on itself, the robot was able to capture more of the map on every passes over the same place, one would still need to

cycle multiple times however to be able to detect more loop closures that leads the map to correct itself overtime, but this trick helps capturing most features of the environment. One drawback to this method is that it induces more uncertainty to the robot's position and lead to having to do more laps around the map. An alternative to this would be to have the camera and the laser sensor mounted on a pivot, so that it would increase the sensors' field of view, the same way one turns its head to look around a new room or environment to get a more complete view rather than having your head fixed in the direction your body is heading, which narrows the amount of information one can perceive.

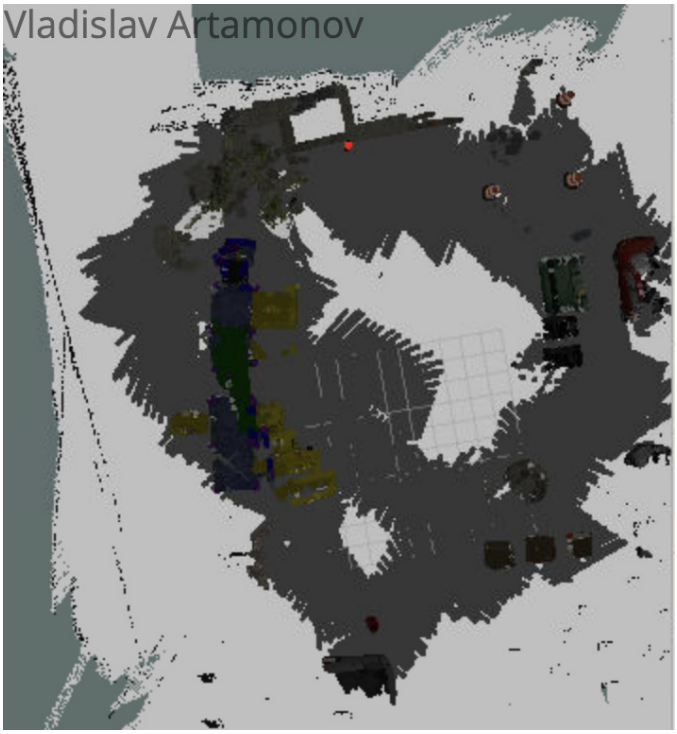

Fig. 8. Playground - 3rd cycle, fixed errors

The main issue encountered however was that the robot kept slightly turning right constantly, which drastically increased the amount of uncertainties in the robot's motion. While the issue most definitively comes from the design of the robot, one could compare it to the problems a robot will encounter while mapping environments, especially outdoors. The ground on which the robot is evolving indoor will have a different effect on the robot's motion than driving on sand, mud, or even grass which would lead to more inaccuracies in the robot's motion.

5 Discussion

Mapping the two worlds raised a number of issues one would have to consider when attempting to map environments. The first being that the robot used to do the mapping needs to be tailored to the environment, from its ability to navigate through the map, its speed and the sensors used. Considering that both worlds were mapped successfully, despite the issues with the robot, RTABMap is able to recover, and can therefore be used in mapping environments with less than ideal robots. However the size (of the environment that is to be mapped compared to the perceptual range of the robot), noise in perception and actuation, perceptual ambiguity (how similar different places are, the more similar different places can be, the higher the perceptual ambiguity), and cycles can affect the robots ability to map an environment without accumulating errors past a certain threshold where the map would be unusable.

The size, could potentially be solved by either increasing the number of robots or increasing the sensor's range, or both. For the noise in perception one could either improve the quality of the sensor or deepen the range of the perceived by adding new types of sensors, for the noise in actuation, then using rtabmap's ability to get visual odometry, which can be fused with the robot's odometry to improve the accuracy and therefore reduce the noise in actuation. Perceptual ambiguity and cycles depend on the map but their effects could be reduced by further tuning the rtabmap's parameters (Tutorial [1], list of parameters [2]).

6 Conclusion

Given the insightful results, RTABMap could be used on robots to map any environments, and while more experimentations need to be done with regards to the way it would interact with dynamic environments for instance, it is safe to say that it is possible to use RTABMap to map real environments using a laser scanner on a robot.

Future works would require upgrades on the robot, switching to a 4-wheel based configuration will help with the stability issues, and adding a pivot on the laser scanner would reduce the number of maneuvers needed to fully map out certain areas as stated previously.

References

[1] "RTABMap_ros advanced parameter tuning tutorial." http://wiki.ros.org/rtabmap_ros/Tutorials/Advanced%20Parameter%20Tuning
[2] "List of all RTABMap_ros parameters." https://github.com/introlab/rtabmap/blob/master /core-lib/include/rtabmap/core/Parameters.h.