Exploring SLAM

Y. Khaky

Abstract: The purpose of this paper is to introduce and explain a specific kind of a SLAM algorithm called RTAB-Map. Simultaneous Localization and Mapping or SLAM for short is the problem at which the robot attempts to construct a map of the environment while simultaneously localizing itself within that map. Neither the map nor the location is known, however, each one of them is needed to define the other. SLAM is an important problem to solve as it is a problem that shows up very frequently in many applications, range from autonomous vacuum cleaning to surveilling construction sites.

***Index Terms*— SLAM, Graph SLAM, RTAB-Map, mapping**

I. Introduction

There are several algorithms to implement SLAM, namely: Extended Kalman filter, Sparse Extended Information Filter, Extended information form, FAST Slam and Graph Slam. The one implemented in this lab is a version of the Graph slam referred to as RTAB-Map.

II. Background

A. The Theory

Real-Time appearance Based Mapping (RTAB-Map) implies that the data is collected in the form of images, these steam of images from the camera on the robot will be used to perform graph SLAM, localizing the robot and mapping the environment at the same time.

There are two sections to graph mapping, the front and the back end. The front end is concerned about sensor data. The image and odometry data are collected. They are used in calculating loop closure. Loop closure is the process of classifying the current location as – previously visited. This is done by comparing the current image with the previously captured images. The back end is concerned about graph optimization and outputting the map. In a perfect world, if the robot visits the same position again, the images from that place would perfectly overlap with the previously captured images, however due to the error accumulated from the noisy robot motion and the noisy sensors, this would never happen, thus the usage of the Bag of Words algorithm. When an image looks similar enough (a certain threshold is reached) to a previously detected image, the robot concludes it is at the same position as the earlier image. When a positive match happens, and two images are linked, a loop closure takes place. When a loop closure occurs, the back end is given sufficient input to reduce the error in the graph and the output map. This results in a non-choppy image as the image isn't recorded as a new image in an unvisited new location. The figure below shows the effect of incorporation loop closure detection algorithm to the output map produced.

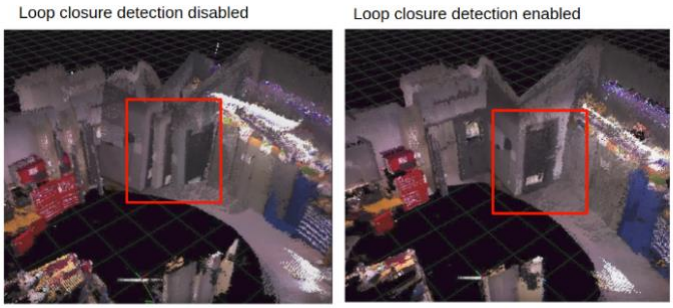

Fig. 1 Showing the difference between two maps one done with loop closure and one without

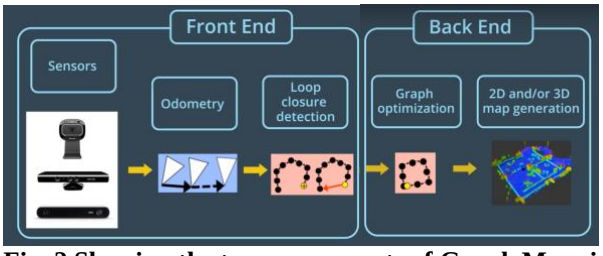

Fig. 2 Showing the two components of Graph Mapping

III. Robot Configuration

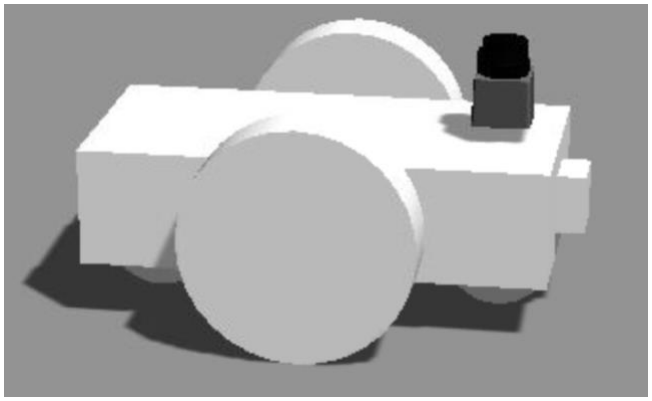

Fig. 3 Showing a picture of the robot

The robot has two motorized wheels and two caster wheels. The robot has two sensors, a LIDAR and Kinect sensor. The Kinect captures images that are going to be used in the Bag of Words algorithm. The resolution of the camera is set to 640X480. The wheels of the robot have encoders that are used for odometry. The reason for choosing this robot model was due to simplicity and ease of creation.

IV. World Creation

There are 3 maps used in this project, 2 of them produced a successful mapping, but the last one didn't result in any loop closure. That observation will be discussed under the "Failed Experiments" section.

The first world used is the Kitchen world which is available as an item to be inserted as an item from Gazebo if the entire objects list is downloaded into the computer. The world has two different rooms with several features that would help in loop closure. The figure below shows the kitchen map.

Fig. 4 Showing the kitchen map

The second world was created using the building editor in Gazebo. A square is divided into 4 triangular compartments by constructing two walls across the two diagonals. Random small items were placed in the environment to increase the number of features thus facilitating loop closure. The figure below shows the "4 compartments" map.

Fig. 5 Showing the "4 compartments" map

Further discussion on the 3rd (failed map) will be done in one of the following sections.

V. Procedure Summary

Gazebo was launched simulating both the environment and the robot using "`roslaunch mapping udacity_world.launch`". Then the mapping node was run using "`roslaunch mapping mapping.launch`". Then the robot was navigated around the environment using a teleop node which was launched using the teleop node launched by running "`roslaunch mapping teleop.launch`". The robot was circulated around the environment more than once to enable loop closure detection.

VI. Results

The two mapping jobs had good results. Unfortunately, a method to evaluate the accuracy of a map after a mapping procedure was not used (more about that in the future works section) so there is no way to know the accuracy of a map and compare it to another map other than visually. A metric that we can use is the number of loop closures, this doesn't tell us directly if the map result is correct, however the more loop closures there are the more likely a map is accurate because with every loop closure we get an added constraint to the graph that needs to be optimized for, so in theory, the more loop closures/ graph constraints, the more the chances the robot had to correct for errors in odometry.

The kitchen map resulted in 139 loop closures. The figure below shows the map obtained from mapping. The two figures below show the output 2D-Map and 3D-Map and the number of loop closures. These results are viewed using the RTAB-Map database viewer. After every mapping job, a .db file is generated, to see the database of nodes, the

following command is ran "`rtabmap-databaseViewer <generated file.db>`",

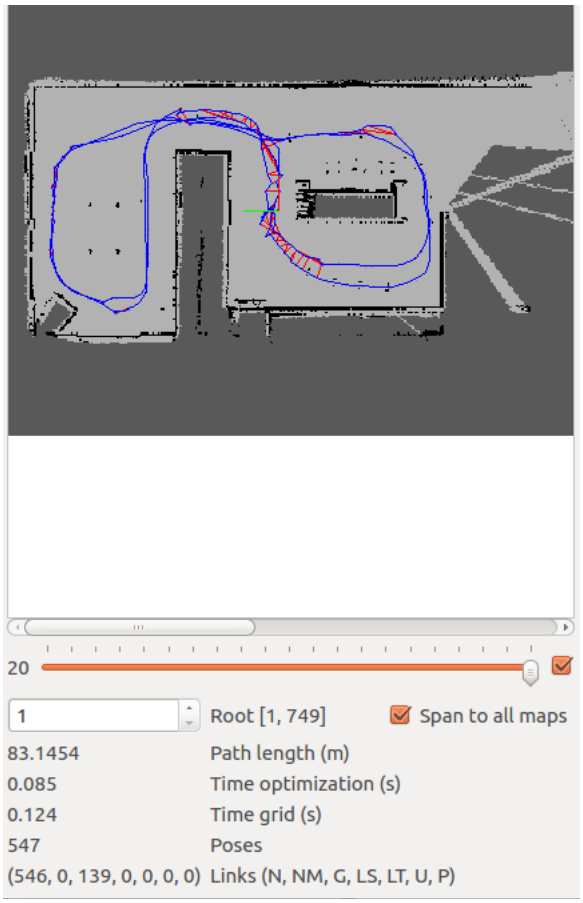

Fig. 6 Kitchen map showing 139 loop closures

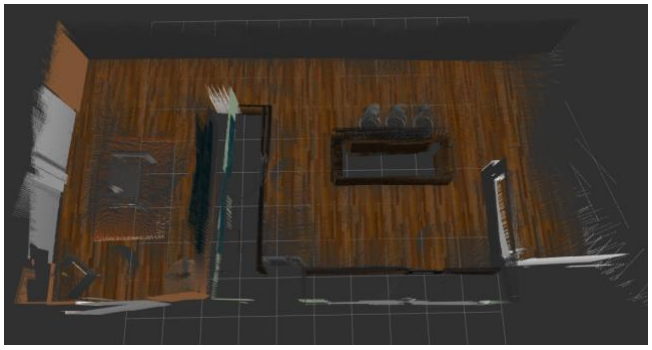

Fig. 7 3D-Map of the Kitchen map

The mapping of the second map resulted in 71 loop closures. Initially they were 73 with 2 nodes wrongly mapped which resulted in an inaccurate map, after the removal of these two wrong nodes the 2D map looked much better. The following two figures show the 2D-Map and 3D-Map and the number of loop closures.

Fig. 8 The 4-compartments map showing 71 loop closures

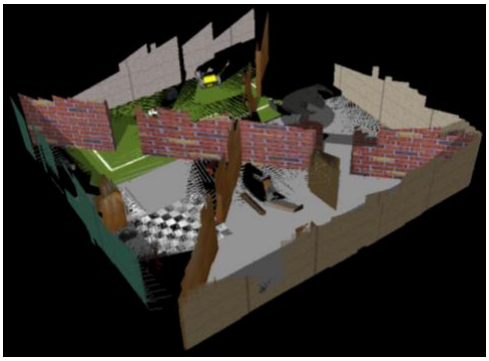

Fig. 9 3D-Map of the 4-compartments map

VII. Discussion

The 2D and 3D mapping of the two maps looks fairly accurate, however, because there is no way to numerically measure the accuracy of the map at this point in time, the correct accuracy is not known. Visual inspection is what is used to gauge the accuracy of the map.

There are several problems with the maps generated, first one being the high amount of noise. The figure below shows the chairs from the first environment. The high noise is most visible on the right most chair. It is almost no longer distinguishable as a chair. If this point cloud data of the chair were to be passed into a classifier, the classifier would probably not be able to classify the object.

Fig. 10 noisy 3D mapping with the right one being the noisiest and no longer distinguishable as a chair

The second problem noticed is that sometimes during mapping, the same wall is detected as two walls each with a different angle. The figure below shows an instance where that happened for one mapping instance.

Fig. 11 showing multiple walls detected in the kitchen area of the first map

Fig. 12 showing multiple walls detected in the living room area of the first map

The third problem noticed is that the walls of the first map (in the 2D view) should be represented as a solid line all the way around the environment, however there are some gaps in the wall. The top left corner of the map has few gaps, which later may be assumed by a much smaller robot as a valid path that can be taken. The figure below shows the gaps in the wall.

Fig. 13 gaps in the wall of the first map

The possible reason for this problem may be due to a low resolution of a LIDAR sensor. As seen from the trajectory of the robot, the robot never approached the top left corner of the map. The further the robot is from a wall the more the spaces between the LIDAR laser lines when they hit a surface as the laser rays are diverging from the source.

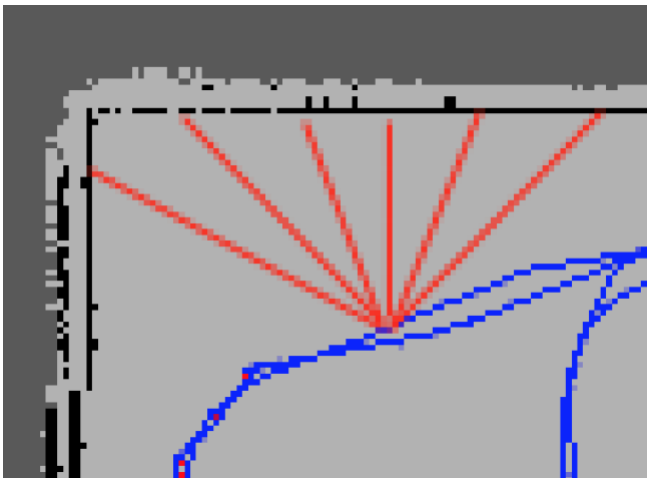

Fig. 14 The further away from the wall the robot is, the more spread out the LIDAR rays intersection point with the wall. The rays are denser when the wall is closer

Mapping the second map was considerably more difficult than the first one. The mapping was attempted at least 30 times before the current map was generated. The main problem is faced is false loop closure detection. The map has a rotational symmetry to it, so the loop detections provided several wrong loop closures that had to be manually rejected. Sometimes, the number of false loop closures was extremely high such that it was very hard to reject every false loop closure detected in the RTAB database viewer especially because the software doesn't have the feature of going to the loop closured images by clicking on the node in question from the 2D-map. When that would happen, the easier thing is to map again from the

scratch. The figure below shows an example of a map with a false loop closure.

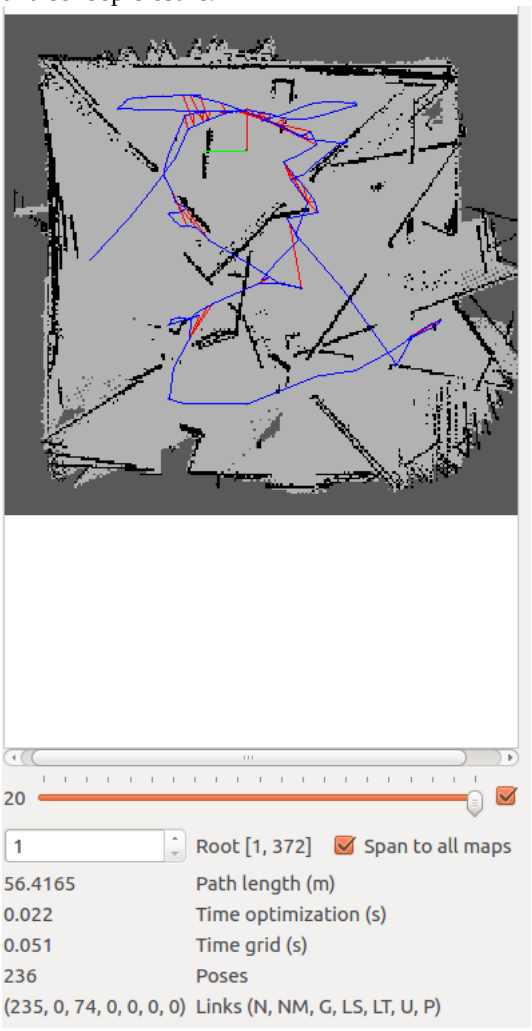

Fig. 15 showing a map that was often generated as a result of wrong loop closures

When this happens, moving the slider all the way to the left fixes the map and would show where the mapping went wrong. The slider in the image above is at 20. When it is moved to 0, this map is generated. No information was found regarding the functionality of this slider and what it means.

The figure below shows what happens when the slider is moved all the way to the left.

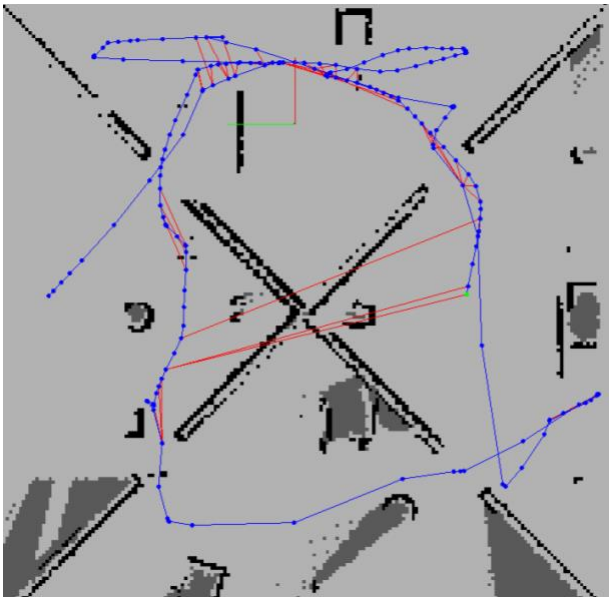

Fig. 16 graph after slider moves to 0

The above graph shows us where the mapping went wrong. A node on the right thought it detected a similar image as a node in the left, which is impossible because these two nodes are separated by a wall and the distance between both of them is very significant. That causes the messy map generated in the previous figure.

Luckily these are only 3 false loop closures and it can be seen that they happened at the end of the mapping session. Therefore, by removing all loop closures that occurred at the last 3 nodes we fix the map.

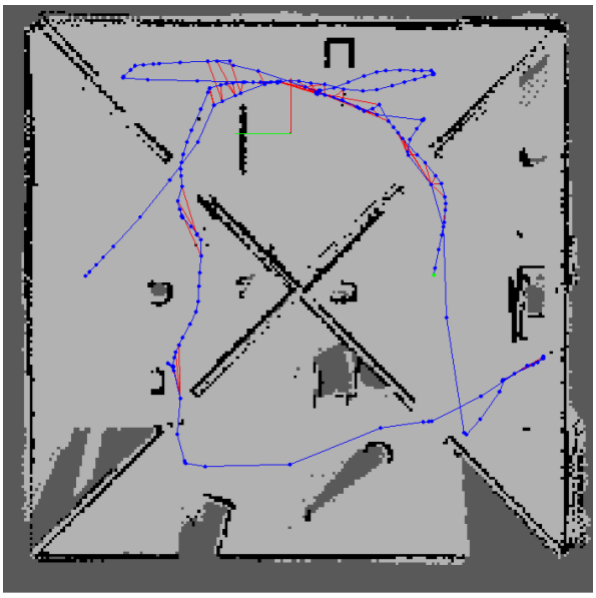

Fig. 17 Graph after rejecting few false loop closures

Another problem that was faced in this map is that the algorithm was detecting many very small features that seem to replicate across the map, such as the brick wall pattern. That would cause the algorithm to detect a lot of false loop closures.

Several RTAB parameters were used in attempt to stop the detection of these small features, however it doesn't seem that these parameters are working.

Here are some parameters that were changed:
Vis/MinInliers: 15

The initial value was much higher than this, the reason it was reduced was to reduce the number of features that the robot would detect such that it would focus on the big features.

Icp/MaxTranslation: 0.1

The MaxTranslation was not set initially, it seems that the default value is 0.2m. The value was set reduced in hopes that it would reject the loop closures that are far away from each other. However, the parameter doesn't seem to have the thought of purpose and didn't seem to have an effect on the mapping.

VIII. Failed Experiments

It is also worth noting that several other big maps were tried, however mapping them was not successful. The main issue found was that the capture rate of the camera was not frequent enough for these big maps. Therefore, the chances of two pictures having similar features is significantly reduced especially if the map is big. With this slow capture rate, the mapping had to be kept to small maps to increase the chances of loop closure. Since no lop closures were detected, errors in mapping were never corrected. The probable cause of the bad mapping is due to hitting the walls which may lead the wheels to slip and results in the wheel encoder counting wheel rotation while in fact is has not moves from its place.

IX. Future Work

There are a couple of problems discussed in the previous sections. Here is a list:

1. No metrics to compare the map output to actual environment
2. Noisy point cloud
3. Same wall detected at different angles
4. Gaps in a wall in the 2D-Map
5. False loop closures
6. Detection of many small features that should be ignored
7. No loop closure when map is huge

There are multiple solutions for the first problem. The first one is to mesh the point cloud data somehow and generate the objects again from the mesh. Afterwards generated map would be intersected with the actual map. A metric that can be used is IOU (intersection over union). The higher this ratio is, the more similar the generated mesh is to the real map. The generated 2D map can also be compared to a ground truth map.

For the second problem regarding the noisy point cloud data, there is not much that can be done, this is a hardware problem. Perhaps moving extremely slowly and having a very stable base that doesn't change its pitch angle upon braking or accelerating would help.

For the third problem regarding the walls at multiple angles. It seems that the reason for this kind of behavior is due to a problem in the odometry. Because there are two distinct walls visible and not a continuous number of walls, that would suggest that an error in the odometry was introduced abruptly and suddenly and is not some kind of small error

runaway. This suggests that the reason this problem occurs is due to the robot hitting the wall, accelerating or decelerating suddenly or turning too fast from rest or vise versa (high angular acceleration). These things would result in the odometry loosing track of the robot's pose and error would be introduced.

The fourth problem can be solved in multiple of ways, the resolution of the LIDAR can be increased by spinning it faster or using a higher resolution LIDAR. Another way to solve this is to make sure that the robot moves close to a wall so the divergence between two LIDAR laser lays is minimal. A third way to solve this is to process the map and filter out those apparent gaps in the wall and replace them by a black dot. The filter can be made such that it fills in the gaps in straight lines only and not curving surfaces.

The fifth and sixth problem require more parameter tuning. There are over 100 parameters a more in-depth look into the theory would help in knowing which parameters are going to most likely have an effect in reducing the effect of these two problems.

The seventh problem is very problematic if we use this algorithm for big spaces. As discussed in the "FAILED EXPERIMENTS" section, the reason for this problem is the low capturing frequency of the camera. The "DetectionRate" is an argument used in the mapping node to set the capturing frequency rate. For these experiments that variable was set to 1Hz. The higher this value is better because the chances to skip an important feature is reduced, however there is a tradeoff. The time between the two captures is the maximum time allowed for comparing bags of images with each other inside the Working Memory. If we increase the capturing frequency we reduce the time period thus reducing the size of our working memory. The reduction in working memory would result in reduction in loop closures because less features are being compared with one another. There is an optimal amount of image capturing and that was not looked into in the experiments explained in this paper. In the future, a study can be made to formulate the optimal frequency to result in most accurate loop closures.

X. Appendix

A. *Memory Allocation*

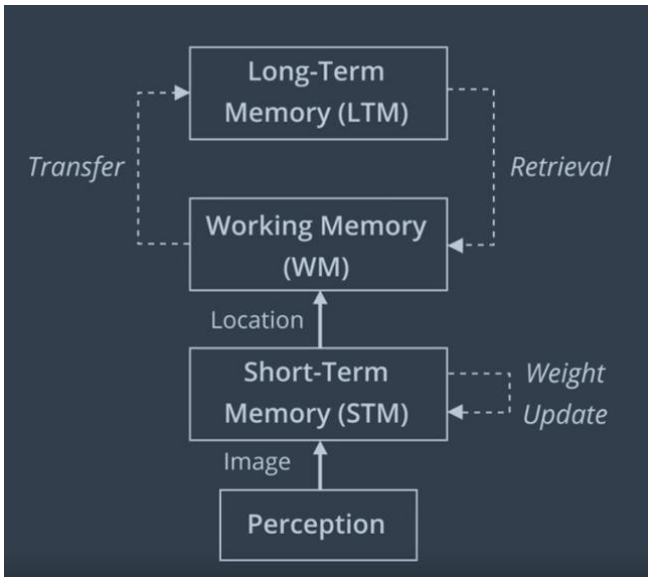

Fig. 18 Showing memory allocation flowchart

There are three stages/ locations an image can go to. The Short-Term memory, Working Memory and Long-Term Memory. Working memory is where all the images that will be compared to the incoming image will be stored. As mentioned earlier, we need to keep the computation period be lower than the capturing period. To achieve a constant computation time, we limit the number of images in the working memory. Only the images in the working memory are considered for loop closure algorithm. Images have weights in the WM, these weights depend on the amount of time the robot spends around that location of the image. An image would be weighted highly if the robot spends more time around that location. Highly weighted images are more likely to remain inside the working memory because the ones with low weight are transferred out to the Long-Term Memory.

The Short-Term memory is where incoming images get stored. The STM collects new images, the weight of these images is what decides if this image should get transferred to the Working Memory. After every new image capture, the weights are updated, the images with the highest weights are moved to the WM.

In the event of a loop closure, all the images in the LTM around the location where the loop closure happened will be called into the WM from the LTM thus increasing the chances of a additional loop closures in the near future because the robot is currently at a location where it has visited before, therefore previous images from around this location have high chance of loop closure.