

SLAM ALGORITHM IMPLEMENTATION ON A BAG FILE

CPE-521 Final Report

Anishkumar Dhamelia

Stevens Institute of Technology
Hoboken, NJ



Contents

1 Introduction

2 SLAM algorithm

3 Dataset

4 Graphical User Interface

5 Implementation

6 Results

7 Further discussions

8 References

Introduction

The aim of this project is to demonstrate the ability of SLAM algorithms to generate a map of the path taken by a robot. These algorithms use the robot's Odometry and Laser-scan values to produce a map of the environment and the robot's location at any given moment.

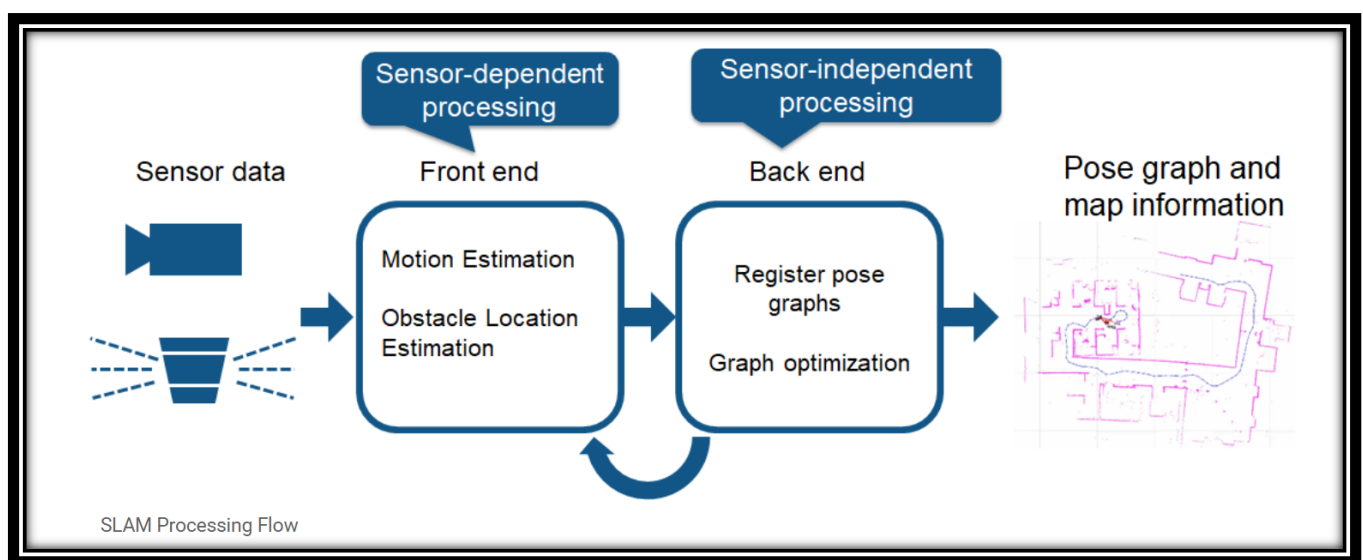
We will discuss SLAM algorithms and visualize their results on a single robot data. This will allow us to fairly compare different implementations to a ground truth.

This project is implemented using ROS (Robot operating system) on Ubuntu (18.04), and Rviz for visualization of the results.

SLAM Algorithm

SLAM (simultaneous localization and mapping) is a method used for autonomous vehicles that lets you build a map and localize your vehicle in that map at the same time.

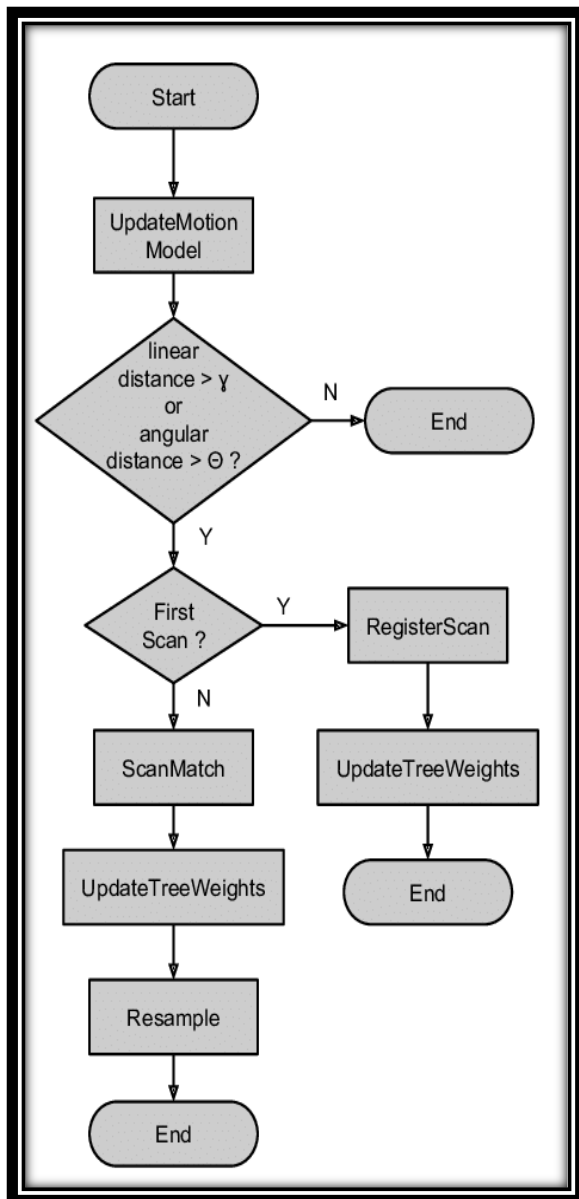
Broadly speaking, there are two types of technology components used to achieve SLAM. The first type is sensor signal processing, including the front-end processing, which is largely dependent on the sensors used. The second type is pose-graph optimization, including the back-end processing, which is sensor-agnostic [1].



Gmapping:

It is probably the most used SLAM algorithm.

This approach uses a particle filter in which each particle carries an individual map of the environment.



We propose an approach to compute an accurate proposal distribution taking into account not only the movement of the robot but also the most recent observation. This drastically decrease the uncertainty about the robot's pose in the prediction step of the filter. Furthermore, we apply an approach to selectively carry out re-sampling operations which seriously reduces the problem of particle depletion. [2].

Dataset

For our project we will take in raw input data from a robot in the form of a csv file. We then, convert this into a bag file using Python.

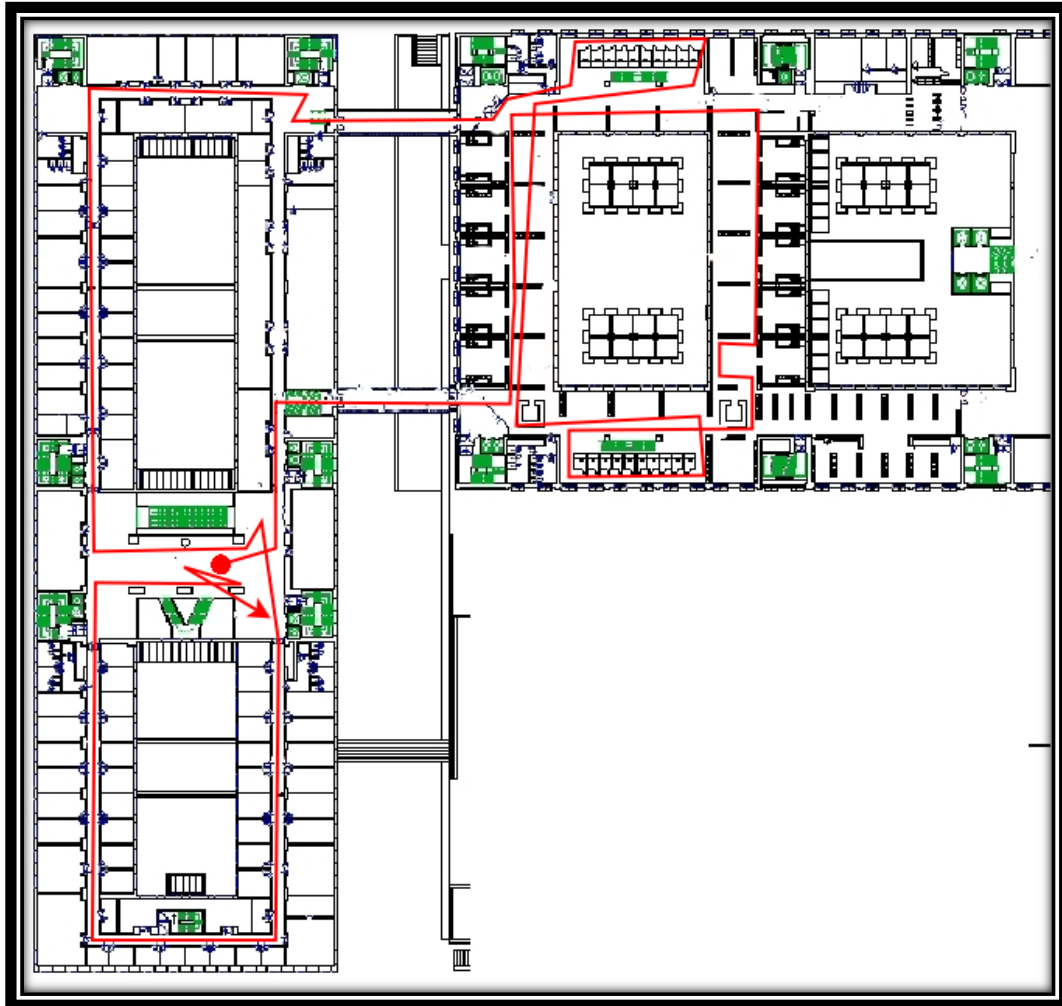
This Python file has the ability to convert Odometry and laser scan data into a bag file by performing the necessary TF transformation. This builds a TF tree where the 2 laser scanners are connected to base_link, and the base_link is connected to the odom.

This robot data is provided by “The Rawseeds project”.

The location is set into a pair of building belonging to the Università di Milano-Bicocca, in Milan (Italy). The part of the location explored by the robot during the gathering of the datasets includes one floor of an office building and the corresponding floor of a nearby building. The two buildings are connected by roofed, glass-walled bridges. They sport a wide range of different of architectural features, which are summarized below [3].

Ground Truth:

The given dataset “Bicocca_2009-02-25b” has the following map [4]:



The red line is the path taken by the robot and the black line represent the walls.

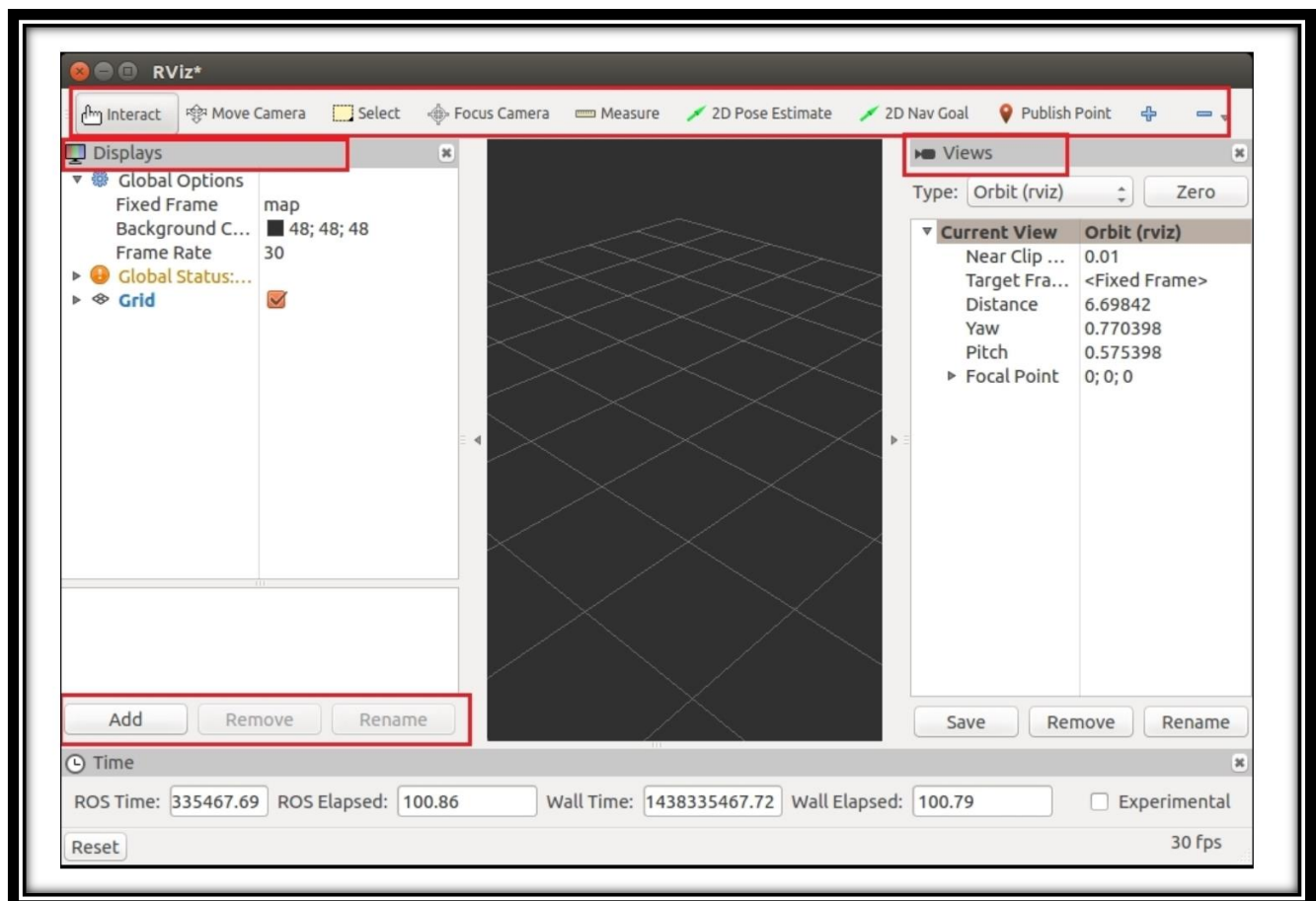
This map will be used to compare the results of the SLAM algorithm in order to judge accuracy of the map generated by us.

GUI

A graphical user interface is a graphics-based operating system interface that uses icons, menus and a mouse (to click on the icon or pull down the menus) to manage interaction with the system.

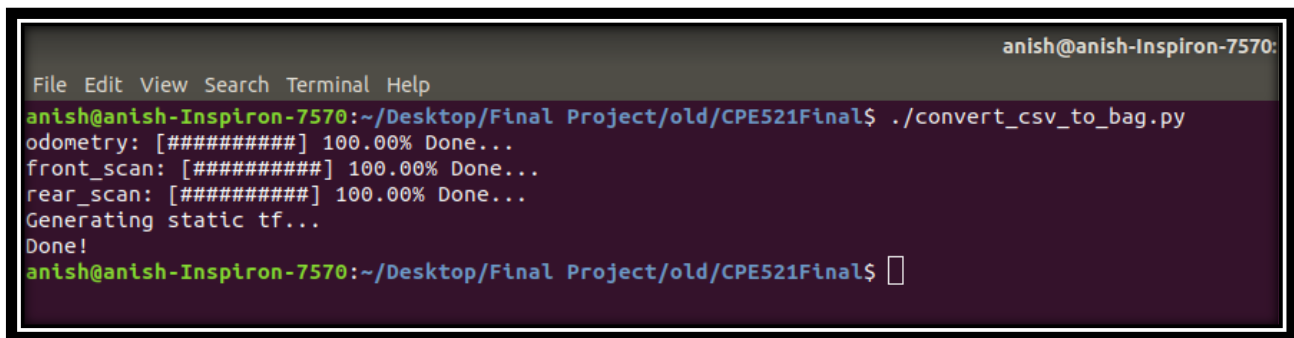
For our purposes, in order to visualize the map generation process in real time we use Rviz.

RVIZ [5]:



Implementation

Here, we start by converting the CSV files to bag file:

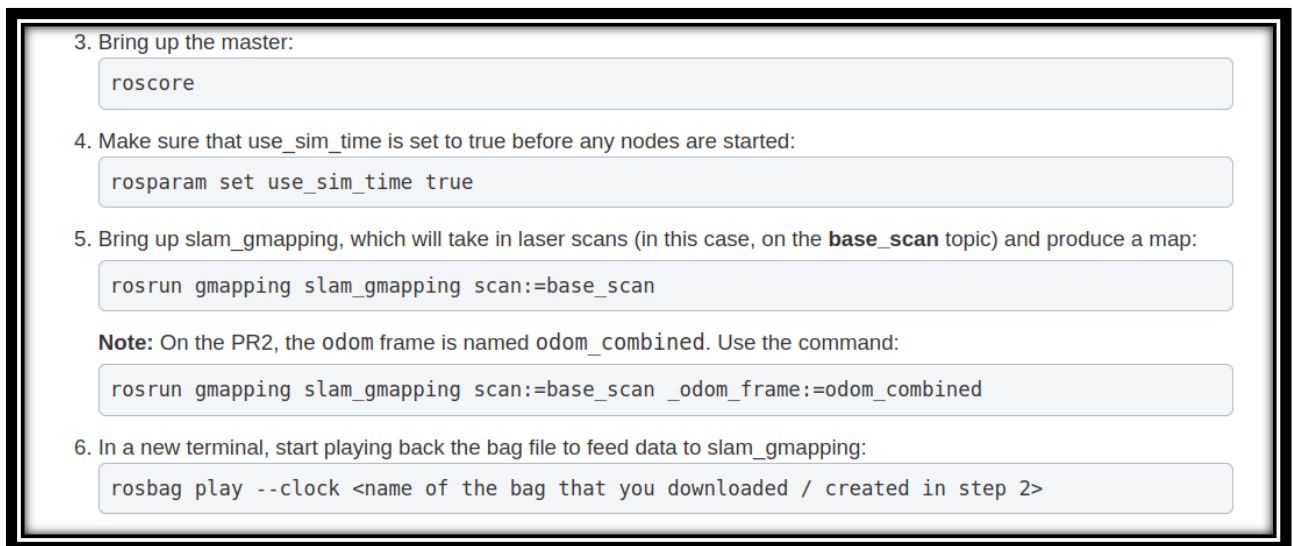


```

anish@anish-Inspiron-7570:~/Desktop/Final Project/old/CPE521Final$ ./convert_csv_to_bag.py
odometry: [#####] 100.00% Done...
front_scan: [#####] 100.00% Done...
rear_scan: [#####] 100.00% Done...
Generating static tf...
Done!
anish@anish-Inspiron-7570:~/Desktop/Final Project/old/CPE521Final$ 
  
```

This will create a “output.bag” file for us to use

Then, we use the following commands [6]:



3. Bring up the master:
`roscore`
4. Make sure that `use_sim_time` is set to true before any nodes are started:
`rosparam set use_sim_time true`
5. Bring up `slam_gmapping`, which will take in laser scans (in this case, on the **base_scan** topic) and produce a map:
`roslaunch gmapping slam_gmapping scan:=base_scan`

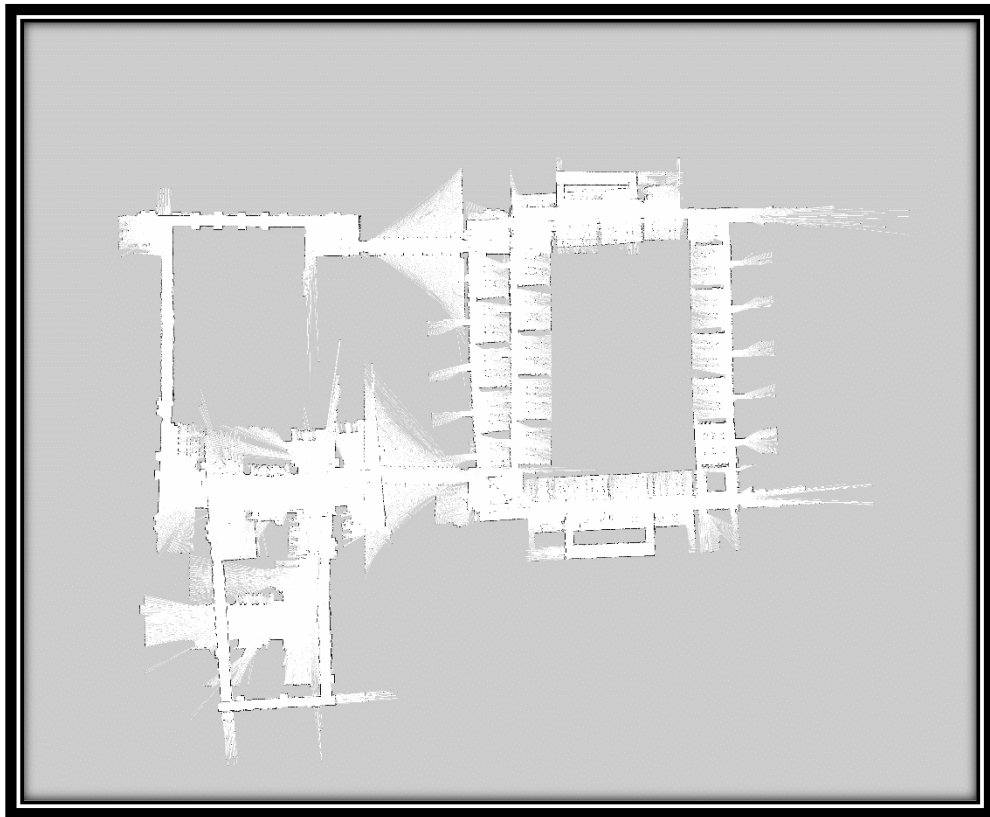
Note: On the PR2, the odom frame is named `odom_combined`. Use the command:
`roslaunch gmapping slam_gmapping scan:=base_scan _odom_frame:=odom_combined`
6. In a new terminal, start playing back the bag file to feed data to `slam_gmapping`:
`roslaunch gmapping slam_gmapping scan:=base_scan _odom_frame:=odom_combined`

Here there is a branch after command 5, we can use ‘front_scan’ or the ‘rear_scan’ to produce the map.

Results

rear_scan:

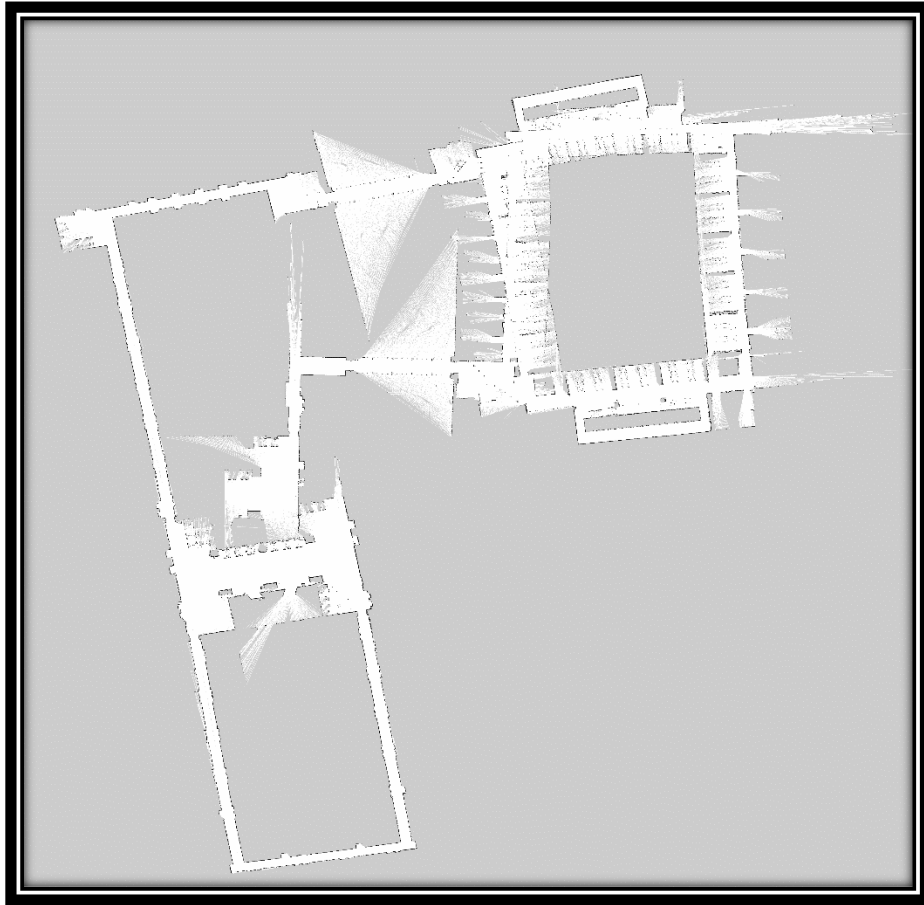
if we use “`roslaunch gmapping slam_gmapping scan:=rear_scan`” we get the following map:



Though far from perfect, our rear scan gmapping has some varied success. I assume that due to the path taken by the robot, the rear scanner hasn't been exposed to the area properly causing uneven mapping on the left side.

front_scan:

if we use “`roslaunch gmapping slam_gmapping scan:=front_scan`” we get the following map:

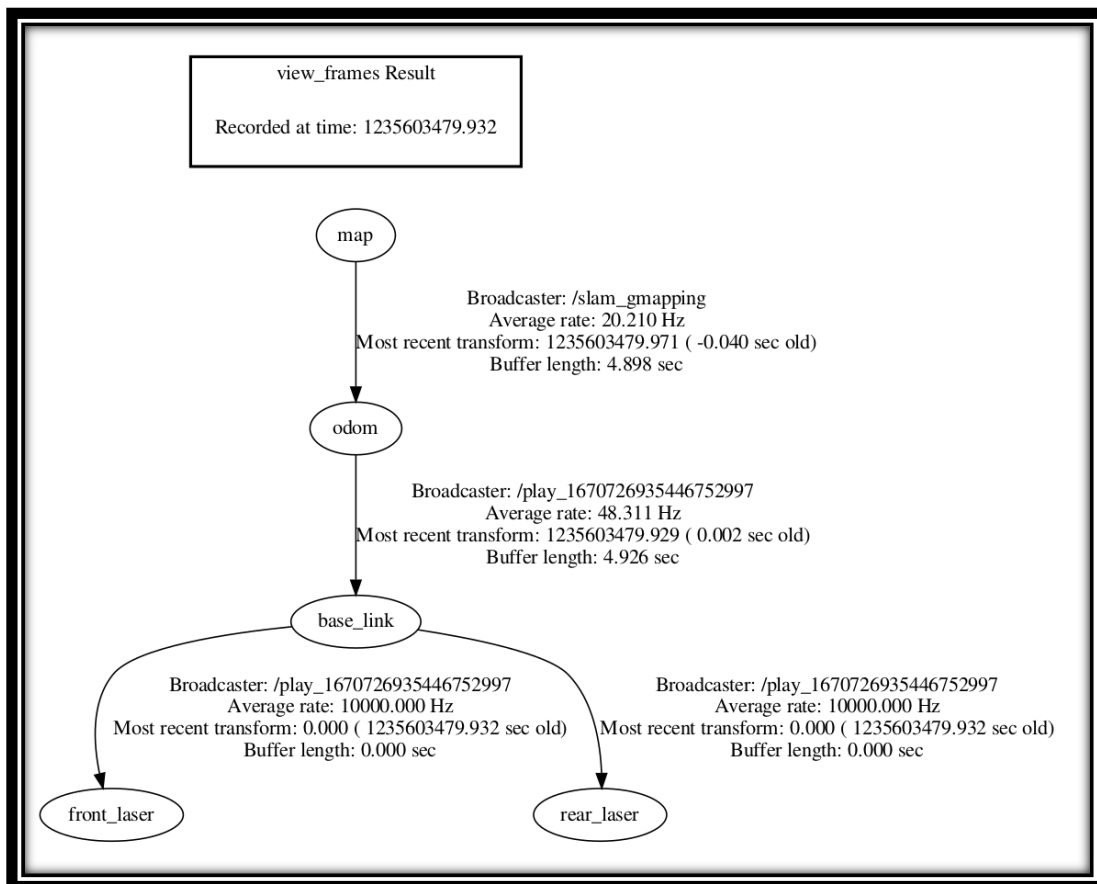


Front scan on the other hand has a far better accuracy. This I assume is due to the fact that the robot is always moving forward, and thus the front scanner has been exposed to the area completely. Though there are still some breaks in the map (especially on the left side) and the wide arches created due to the windows.

Overall:

Though our gmapping algorithm has provided with some decent results it is nowhere close to the ground truth. But, seeing the improvement of the map during the loop, leads me to believe that this result can be improved if the robot moved around the same path multiple times.

The front scan mapping led to a far better map generation and thus created the following TF tree:



Further discussions

- 1) We should try to add more robot data for instance; IMU, Infrared, GPS etc. In order to increase accuracy.
- 2) We should have the robot move forwards as well as backwards through the same map to avoid discrepancy in the front_scan and the rear_scan as noticed above.
- 3) The use of other SLAM algorithms like Extended Kalman filter, FastSLAM and Covariance intersection and Graph based SLAM, in order to compare their accuracy.
- 4) The use of outdoor robot data with uneven roads provides an extra level of difficulty and truly tests the algorithm's real-world use.

References

[1] <https://www.mathworks.com/>

[2] <https://openslam-org.github.io/gmapping.html>, Giorgio Grisetti; Cyrill Stachniss; Wolfram Burgard;

[3] <http://www.rawseeds.org/rs/datasets/view//6>

[4] http://www.rawseeds.org/rs/capture_sessions/view/5

[5] <https://www.oreilly.com/library/view/mastering-ros-for/9781783551798/ch06s04.html>

[6] http://wiki.ros.org/slam_gmapping/Tutorials/MappingFromLoggedData