



LASER_SLAM_GMAPPING

FINAL PROJECT

STEVENS INSTITUTE OF TECHNOLOGY

SHABARIRAJ SIDDESWARAN

CPE_521-A

INSTRUCTOR: PROFESSOR YI GUO

12/20/2022

INTRODUCTION:

SLAM employs cameras and visible and invisible sensors to gather data, radar, sonar, and laser range finders to get basic positioning information (IMU). Together, these sensors collect information and create a map of the surroundings. The SLAM algorithm aids in providing the most accurate location and position estimates inside the surroundings. Another intriguing observation is that the device's location and the surrounding elements, such as the walls, flooring, furniture, and pillars, are relative. The iterative SLAM technique uses the updated positional data to refine the estimated position. The positional precision increases with the number of iterations. This required additional time for computing and specialized gear with GPU-based parallel processing. Both front-end data collecting, and back-end data processing are necessary for SLAM to function. Both Visual SLAM and LiDAR SLAM are used for SLAM's front-end data collecting.

A laser sensor is used in the LiDAR SLAM implementation, and lasers are more precise and accurate than Visual SLAM, which uses cameras. LiDAR sensors may be used in high-speed applications such as moving vehicles such as drones and self-driving automobiles because of their high data acquisition rate and increased accuracy. The output of a LiDAR sensor, also known as a point cloud, can be positioned in 2D (x, y) or 3D (x, y, z) space.

When creating maps using SLAM, the laser sensor point cloud offers highly accurate distance measurements. Typically, movement is calculated progressively by matching the point clouds. The computed movement (distance traveled) is utilized to localize the vehicle. Iterative closest point (ICP) and normal distributions transform (NDT) techniques are employed for LiDAR point cloud matching. Grid maps and voxel maps are two common ways to display 2D or 3D point cloud maps.

SLAM GMAPPING:

The gmapping package offers simultaneous Localization and Mapping (SLAM) using lasers as the ROS node Slam_gmapping. With the Slam_gmapping, laser and posture data gathered by a mobile robot may be used to build a 2-D occupancy grid map.

ROS AND HOW IT WORKS:

The open-source Robot Operating System (ROS) framework aids academics and developers in creating and reusing code for various robotics applications. A global open-source community of engineers, programmers, and enthusiasts known as ROS work to improve, expand the usability of, and make robots available to all.

Some of the leading brands in robotics have embraced ROS. Most businesses either use ROS, which everyone can run, or a ROS fork in some way. The use cases are also still expanding. In addition to vacuum cleaners and medical equipment, ROS is employed in many other industries, including agriculture and software-defined dynamic use cases. These are only a few businesses that publicly use ROS today; as time goes on, the list expands and becomes more diverse.

ROS INSTALLATION:

The RUFUS (3.20) package was initially obtained at <https://rufus.ie/en/>. Rufus is a utility that makes creating and formatting USB flash drives easier. The SD Card Formatter was next installed (5.0.2). Although it may need to be optimized for SD/SDHC/SDXC Cards and may lead to subpar performance, the SD card tool is used to format various storage media, including SD/SDHC/SDXC Cards.

In operating systems like Ubuntu, this utility is available. Installed the Ubuntu OS (20.04) that was downloaded from <https://releases.ubuntu.com>.

The storage for Ubuntu was then provided roughly 100GB, and the machine restarted to use the BOIS keys. After the reboot, the Ubuntu operating system starts, and the installation is finished. The ROS Noetic version was installed on the Ubuntu operating system using several command lines from the webpage [noetic/Installation/Ubuntu - ROS Wiki](http://wiki.ros.org/noetic/Installation/Ubuntu). On the PC, ROS was effectively installed.

The Linux package and Python3 were installed by adding the command line in Terminal “Sudo apt-get install libavcodec-dev libsdl1.2-dev xsltproc libbullet-dev libsdl1.2-dev libgoogle-glog-dev protobuf-compiler python3-wstool”.

ROS CSV FILE:

Data communication between various programs is frequently done using the CSV ("Comma Separated Value") file format. In a relatively straightforward form with one record on each line and each field inside that record separated by a comma, a CSV file is a specifically structured plain-text file that holds information similar to that found in spreadsheets or simple databases.

SLAM ALGORITHM:

The Dataset “**Laser SLAM - Bicocca_2009-02-25b**” from the Benchmark Problem in the “**17. Rawseeds: a large collection of benchmarked datasets for SLAM**” was used and downloaded from the link (<http://www.mobilerobots.ethz.ch/>) provided in the Canvas as shown in the Figure below.

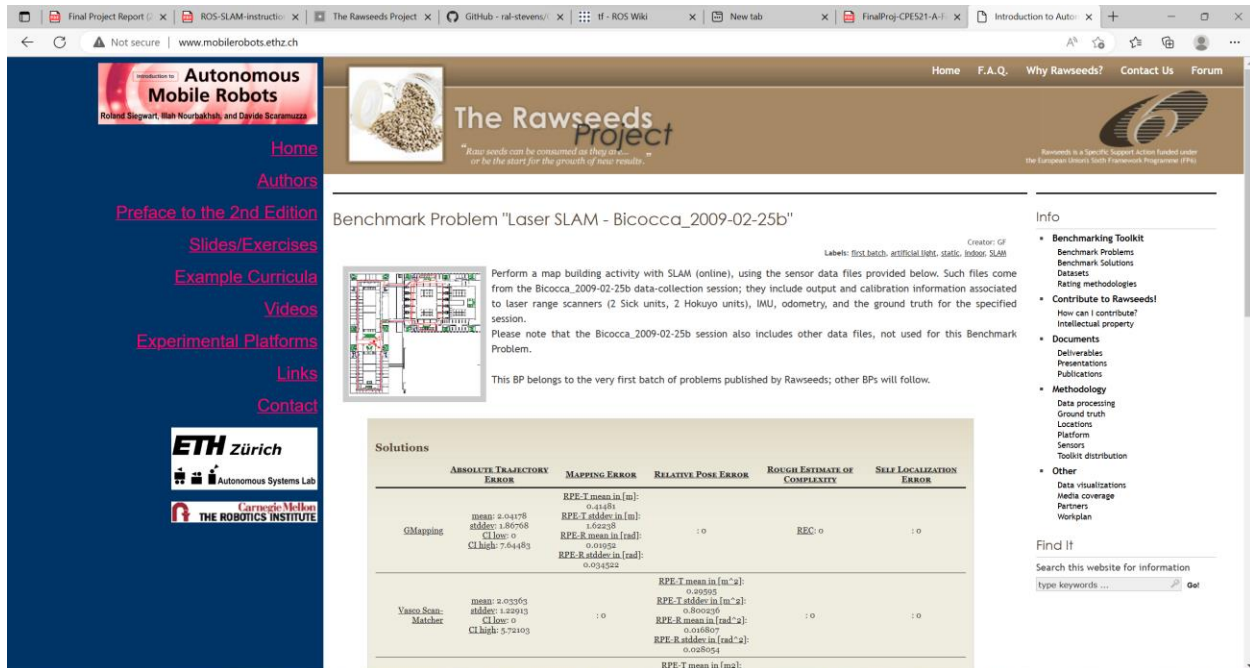


FIGURE 1 DATASET FROM THE RAWSEEDS

Using the GitHub Link (**GitHub - ral-stevens/CPE521Final: CPE521 Final project data**) in the Canvas, First, the Git LFS command ‘**Sudo apt install git-lfs**’ was executed in Terminal as shown in Figure 2, and the Git LFS has installed to Host the large CSV files from the GitHub link.

```
shabari@shabari@Blade-14-2022-RZ09-0427:~$ sudo apt install git-lfs
[sudo] password for shabari:
Reading package lists... Done
Building dependency tree
Reading state information... Done
git-lfs is already the newest version (2.9.2-1).
The following packages were automatically installed and are no longer required:
  libgraphviz-dev libgvc6-plugins-gtk libignition-cmake-dev libignition-common libignition-fuel-tools1-1
  libignition-fuel-tools1-dev libignition-math4 libignition-math4-dev libignition-msgs libignition-msgs-dev libignition-transport4
  libignition-transport4-dev libopenal-dev libqtpropertybrowser4 libstdformat6 libstdformat6-dev libxdot4 sdfmat-sdf
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 17 not upgraded.
shabari@shabari@Blade-14-2022-RZ09-0427:~$ git lfs install
Error: Failed to call git rev-parse --git-dir: exit status 128
Git LFS initialized.
shabari@shabari@Blade-14-2022-RZ09-0427:~$ git clone https://github.com/ral-stevens/CPE521Final.git
Cloning into 'CPE521Final'...
remote: Enumerating objects: 81, done.
remote: Counting objects: 100% (81/81), done.
remote: Compressing objects: 100% (59/59), done.
remote: Total 81 (delta 31), reused 10 (delta 2), pack-reused 0
Unpacking objects: 100% (81/81), 13.48 MiB | 1.55 MiB/s, done.
Filtering content: 100% (4/4), 376.51 MiB | 1.69 MiB/s, done.
shabari@shabari@Blade-14-2022-RZ09-0427:~$
```

FIGURE 2

After installing the Git LFS, the repository was cloned to the local hard drive by executing the command line “**git clone https://github.com/ral-stevens/CPE521Final.git**” in the Terminal, as shown in the figure below.

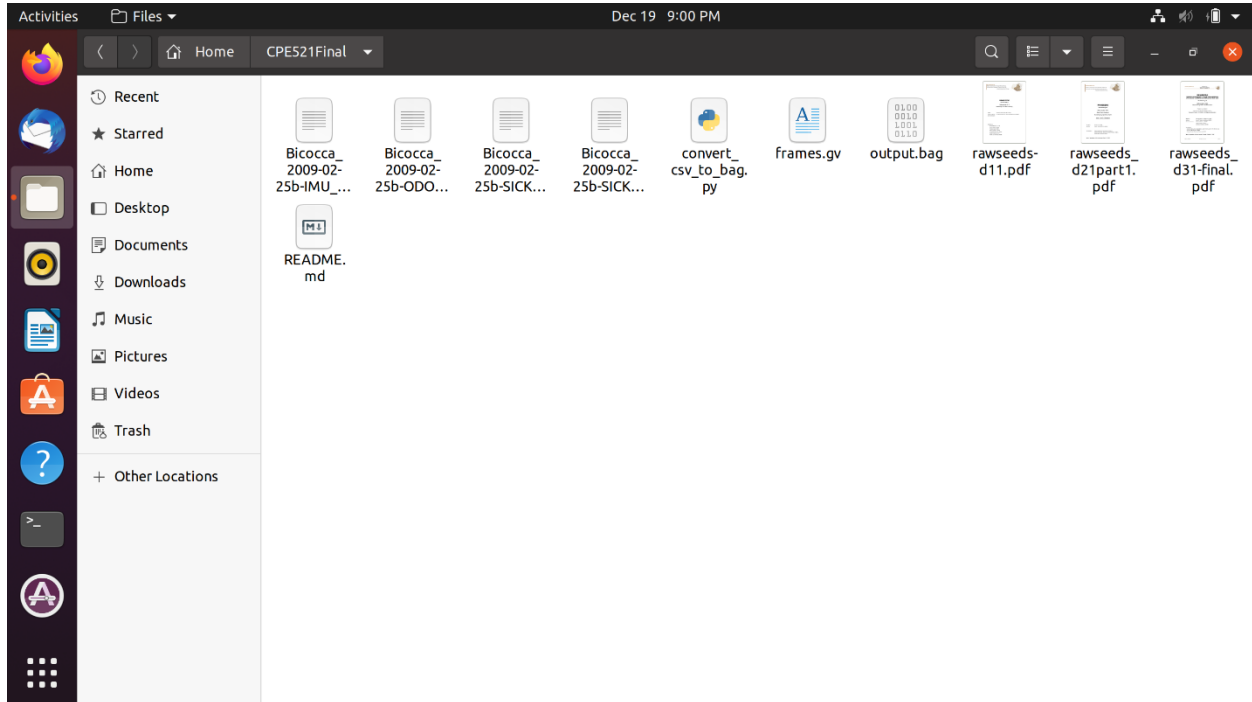


FIGURE 3 CPE521FINAL FILE

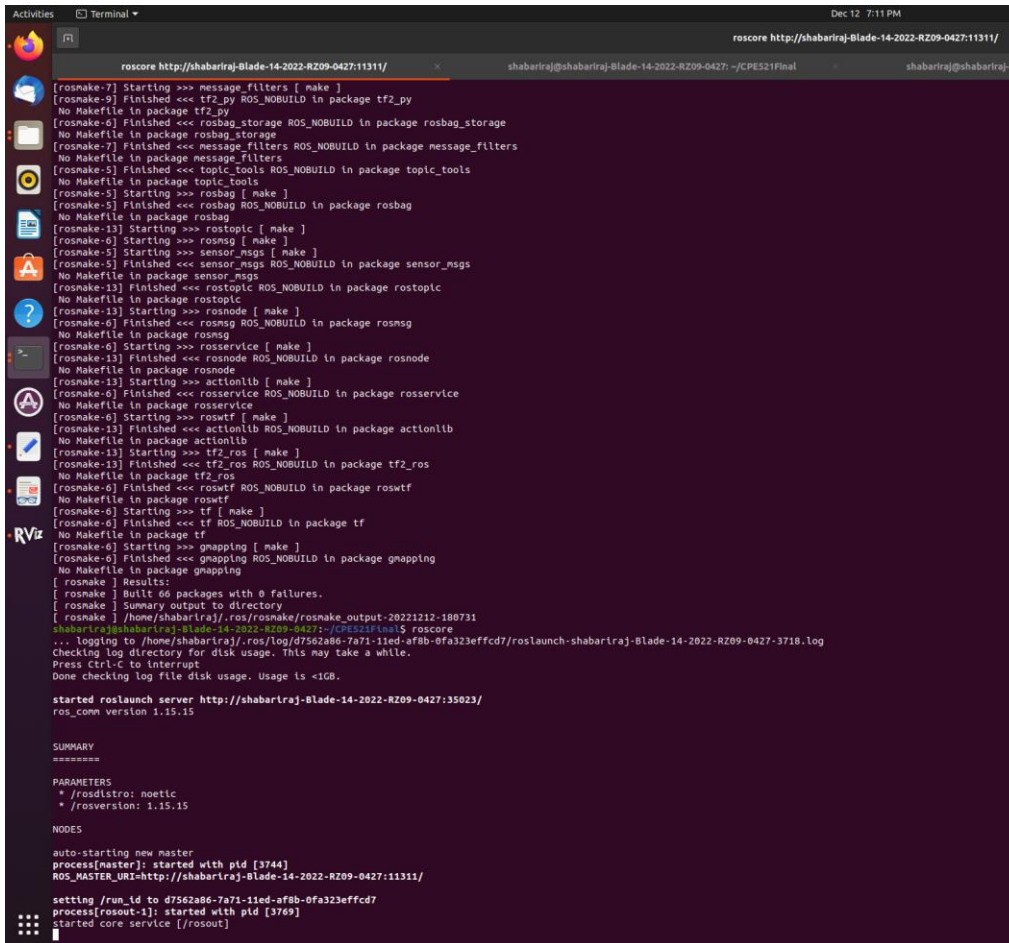
The dataset is cloned and saved in the Home file as “**CPE521Final**”. Then, the CSV files were converted to Bag files by performing the command line “**python3 ./convert_csv_to_bag.py**” in the CPE521Final Terminal as shown in the figure below.

The SLAM_gmapping was extracted by running two command lines, one is to install the gmapping and other is to install the map-server for Noetic ROS version as shown in the figure 4. The command lines are

- a. **sudo apt install ros-noetic-gmapping**
- b. **sudo apt install ros-noetic-map-server**



The next steps were followed from the ([slam_gmapping/Tutorials/MappingFromLoggedData - ROS Wiki](#)) to build the map using the logged data. Initially, the Map should be created by executing the command line “**rosmake gmapping**” in the CPE521Final Terminal. The “**roscore**” command line was performed to launch the ROS server in the computer, as shown in the figure below.



```
Activities Terminal Dec 12 7:11 PM
roscore http://shabariraj-Blade-14-2022-RZ09-0427:11311/
shabariraj@shabariraj-Blade-14-2022-RZ09-0427: ~/CPE521Final
shabariraj@shabariraj-Blade-14-2022-RZ09-0427: ~/CPE521Final

[rosnake-7] Starting >>> Message_filters [ make ]
[rosnake-9] Finished <<< tf2_py ROS_NOBUILD in package tf2_py
No Makefile in package tf2_py
[rosnake-6] Finished <<< rosbag_storage ROS_NOBUILD in package rosbag_storage
No Makefile in package rosbag_storage
[rosnake-7] Finished <<< message_filters ROS_NOBUILD in package message_filters
No Makefile in package message_filters
[rosnake-5] Finished <<< topic_tools ROS_NOBUILD in package topic_tools
No Makefile in package topic_tools
[rosnake-5] Starting >>> rosbag [ make ]
[rosnake-5] Finished <<< rosbag ROS_NOBUILD in package rosbag
No Makefile in package rosbag
[rosnake-13] Starting >>> rostopic [ make ]
[rosnake-6] Starting >>> rosmmsg [ make ]
[rosnake-5] Starting >>> sensor_msgs [ make ]
[rosnake-5] Finished <<< sensor_msgs ROS_NOBUILD in package sensor_msgs
No Makefile in package sensor_msgs
[rosnake-13] Finished <<< rostopic ROS_NOBUILD in package rostopic
No Makefile in package rostopic
[rosnake-13] Starting >>> rosmmsg [ make ]
[rosnake-6] Finished <<< rosmmsg ROS_NOBUILD in package rosmmsg
No Makefile in package rosmmsg
[rosnake-6] Starting >>> rosservice [ make ]
[rosnake-6] Finished <<< rosservice ROS_NOBUILD in package rosservice
No Makefile in package rosservice
[rosnake-13] Starting >>> actionlib [ make ]
[rosnake-6] Finished <<< rosservice ROS_NOBUILD in package rosservice
[rosnake-6] Starting >>> roswtf [ make ]
[rosnake-13] Finished <<< actionlib ROS_NOBUILD in package actionlib
No Makefile in package actionlib
[rosnake-13] Starting >>> tf2_ros [ make ]
[rosnake-13] Finished <<< tf2_ros ROS_NOBUILD in package tf2_ros
No Makefile in package tf2_ros
[rosnake-6] Finished <<< roswtf ROS_NOBUILD in package roswtf
No Makefile in package roswtf
[rosnake-6] Starting >>> tf [ make ]
[rosnake-6] Finished <<< tf ROS_NOBUILD in package tf
No Makefile in package tf
[rosnake-6] Starting >>> gnapping [ make ]
[rosnake-6] Finished <<< gnapping ROS_NOBUILD in package gnapping
No Makefile in package gnapping
[rosnake ] Results:
[rosnake ] Built 66 packages with 0 failures.
[rosnake ] Summary output to directory
[rosnake ] /home/shabariraj/.ros/rosnake/rosnake_output-20221212-180731
shabariraj@shabariraj-Blade-14-2022-RZ09-0427: ~/CPE521Final$ roscore
... logging to /home/shabariraj/.ros/log/d7562a86-7a71-11ed-af8b-0fa323effcd7
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://shabariraj-Blade-14-2022-RZ09-0427:35023/
ros_comm version 1.15.15

SUMMARY
*****

PARAMETERS
* /roslaunch: noetic
* /rosversion: 1.15.15

NODES

auto-starting new master
process[master]: started with pid [3744]
ROS_MASTER_URI=http://shabariraj-Blade-14-2022-RZ09-0427:11311/

setting /run_id to d7562a86-7a71-11ed-af8b-0fa323effcd7
process[roscout-1]: started with pid [3769]
started core service [/roscout]
```

FIGURE 5

To keep track of how the robot's various coordinate frames are positioned with one another, Tf messages are employed. The odometry Tf message, for instance, tracks the posture of the robot base if the robot comprises a base with a laser range finder attached to it. The robot's center and a global reference frame are the bases for these odometry measurements. Depending on the robot, this might vary. There will be a relative displacement between the base and the robot's laser range finder. It may, for example, be a specific x, y, and z distance from the robot base.

The Tf tree for the bag file we created is depicted in the figure below, demonstrating the relationships between each link.

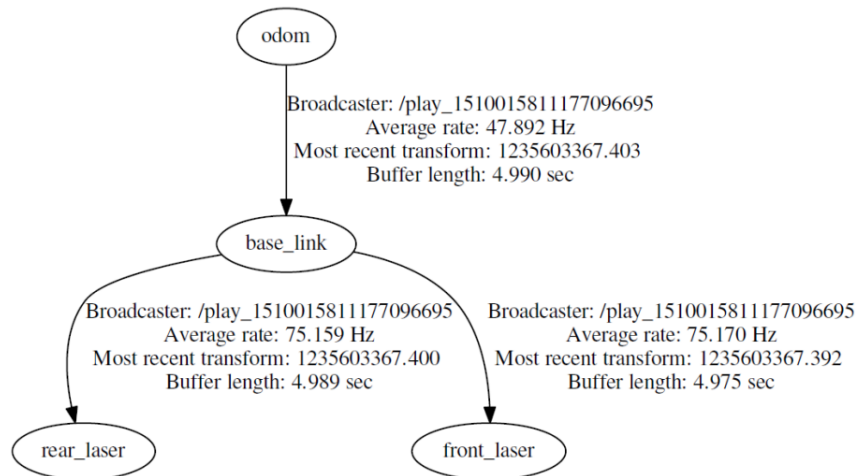


FIGURE 6 TF TREE FOR THE BAG FILE

To set the TF tree and the SLAM_gmapping for this dataset two command lines are used one is “**rosparam set use_sim_time true**” and other is “**roslaunch gmapping slam_gmapping scan:=front_scan**” as shown in the figure 7.

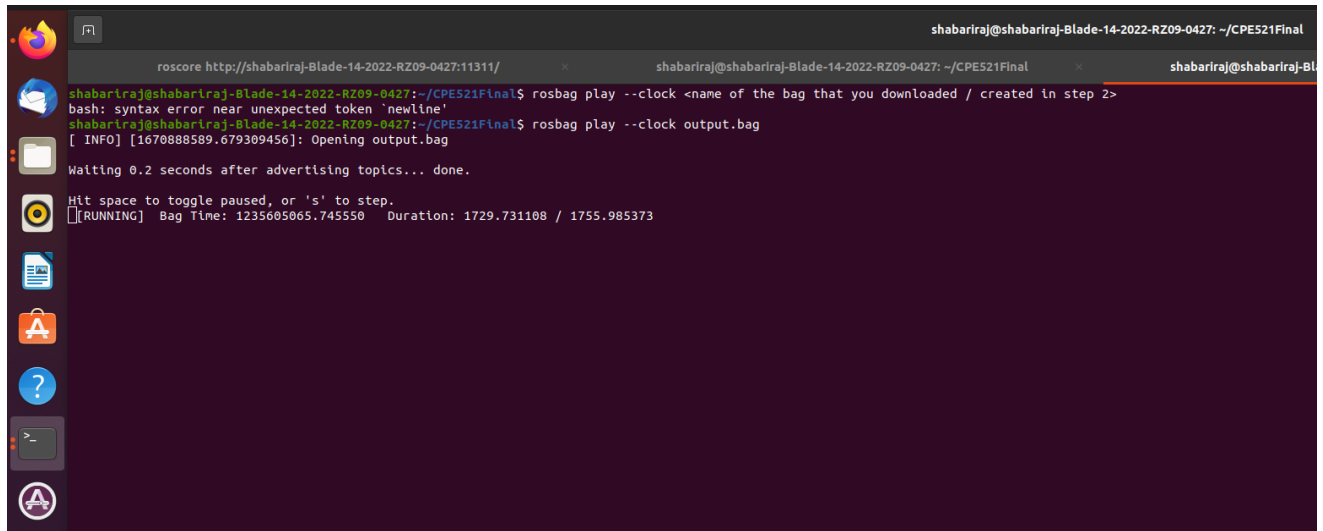
```

shabari@shabari-Blade-14-2022-RZ09-0427: ~/CPE521Final
roscore http://shabari-Blade-14-2022-RZ09-0427:~/.ros/roscore
shabari@shabari-Blade-14-2022-RZ09-0427:~/CPE521Final$ rosparam set use_sim_time true
shabari@shabari-Blade-14-2022-RZ09-0427:~/CPE521Final$ roslaunch gmapping slam_gmapping scan:=front_scan
Warning: TF_REPEATED_DATA ignoring data with redundant timestamp for frame odom at time 1235603336.115117 according to authority unknown_publisher
[ INFO] [1671498473.464237734, 1235603336.116074196]: Laser is mounted upwards.
-maxUrange 29.99 -maxUrange 29.99 -sigma 0.05 -kernelSize 1 -lstep 0.05 -lobsGain 3 -astep 0.05
-srr 0.1 -srt 0.2 -str 0.1 -stt 0.2
-linearUpdate 1 -angularUpdate 0.5 -resampleThreshold 0.5
-xmin -100 -xmax 100 -ymin -100 -ymax 100 -delta 0.05 -particles 30
[ INFO] [1671498473.478090219, 1235603336.126192568]: Initialization complete
update frame 0
update ld=0 ad=0
Laser Pose= 0.08 0 0
m_count 0
Registering First Scan
update frame 230
update ld=1.00432 ad=0.117571
Laser Pose= 1.0834 0.00811452 -0.0235707
m_count 1
Average Scan Matching Score=121.762
neff= 29.5128
Registering Scans:Done
update frame 348
update ld=1.00056 ad=0.0436925
Laser Pose= 2.08223 -0.0427452 -0.051
m_count 2
Average Scan Matching Score=145.935
neff= 28.8717
Registering Scans:Done
update frame 467
update ld=1.00029 ad=0.0669365
Laser Pose= 3.07997 -0.103841 -0.107143
m_count 3

```

FIGURE 7

A new tab is opened in the Terminal, and the “**rosbag play –clock output. bag**” command line was performed to start playing the bag file to feed data to Slam_gmapping and the simulation run time was around 30 mins as shown in the figure below.



```
shabariraj@shabariraj-Blade-14-2022-RZ09-0427: ~/CPE521Final
shabariraj@shabariraj-Blade-14-2022-RZ09-0427: ~/CPE521Final$ rosbag play --clock <name of the bag that you downloaded / created in step 2>
bash: syntax error near unexpected token 'newline'
shabariraj@shabariraj-Blade-14-2022-RZ09-0427: ~/CPE521Final$ rosbag play --clock output.bag
[ INFO] [1670888589.679309456]: Opening output.bag
Waiting 0.2 seconds after advertising topics... done.
Hit space to toggle paused, or 's' to step.
[ ] [RUNNING] Bag Time: 1235605065.745550 Duration: 1729.731108 / 1755.985373
```

FIGURE 8

In the meantime, the **frames. gv** was generated by executing the command “**roslaunch tf view_frames**” and Visualizing the Map building process during the simulation or after the “**roslaunch rviz rviz**” command line was executed, and the RVIZ tool was opened as shown in the figure 9.

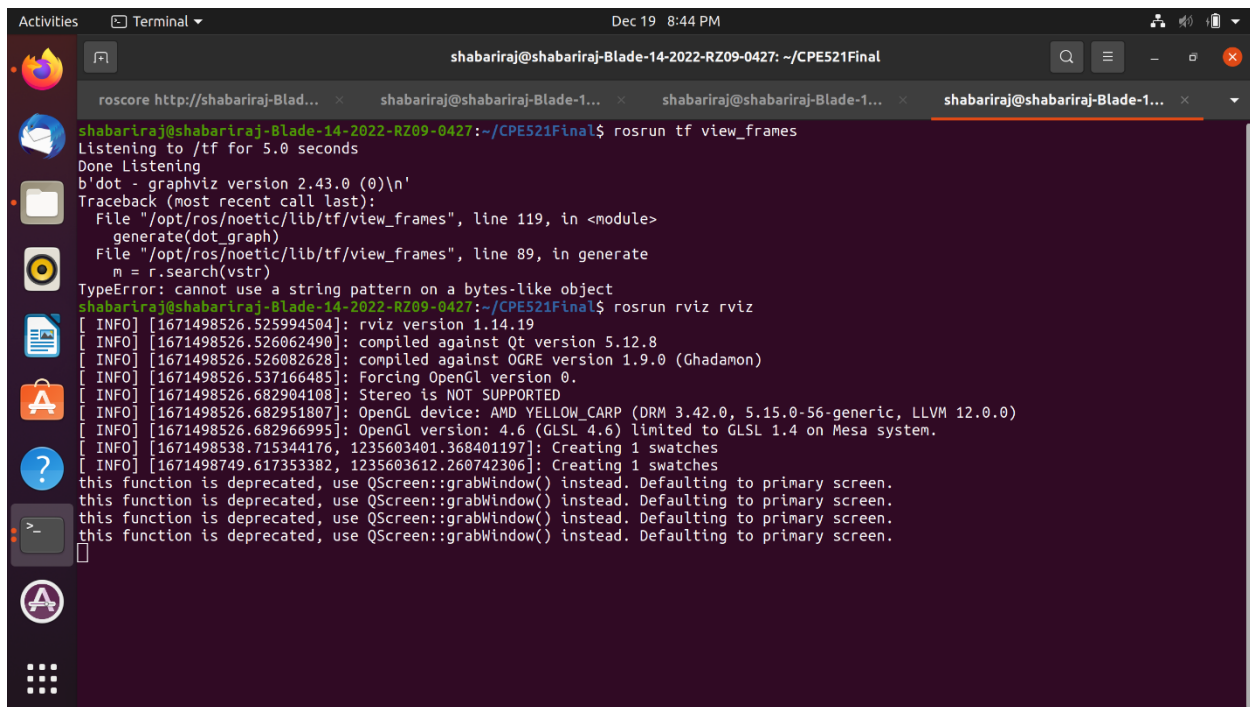


FIGURE 9

By adding the MAP, front laser, rear laser, and the Axes for the robot in the RVIZ tool to visualize the entire laser Map as shown in the figure 10.

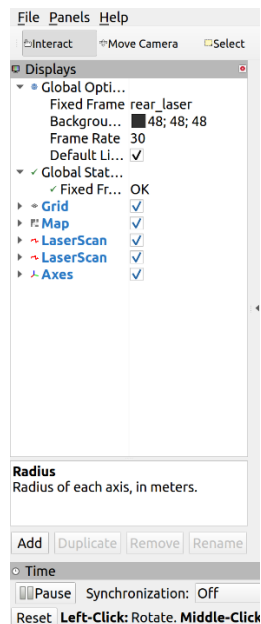


FIGURE 10 RVIZ TOOL

RESULT:

The figure below is the SLAM_LASER_Gmapping and the red color markings are the robot travelled during the Simulation.

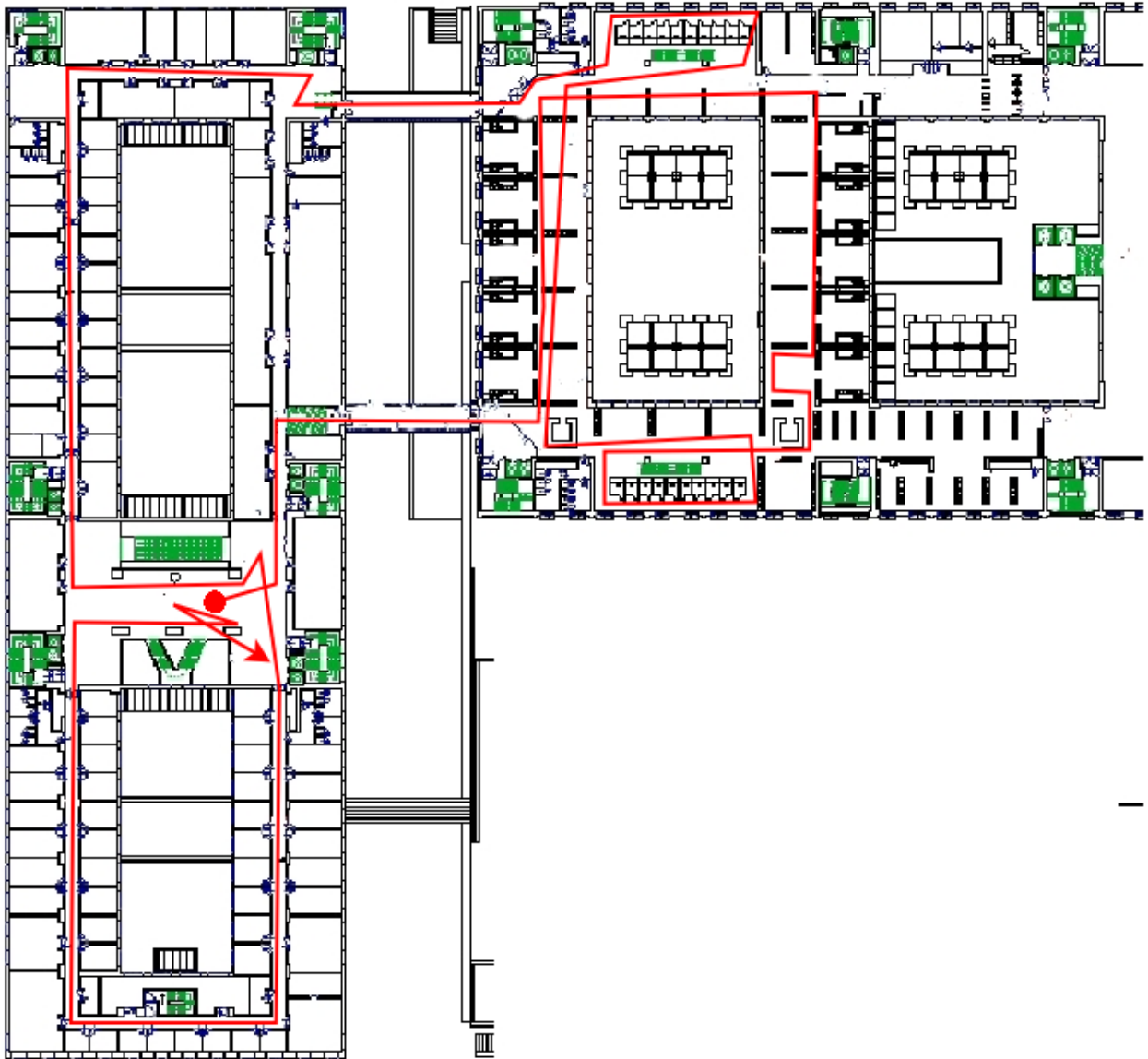


FIGURE 11 SLAM_GMAPPING_LASER_MAP

The obtained result from the gmapping server and the front_scan simulation is shown below

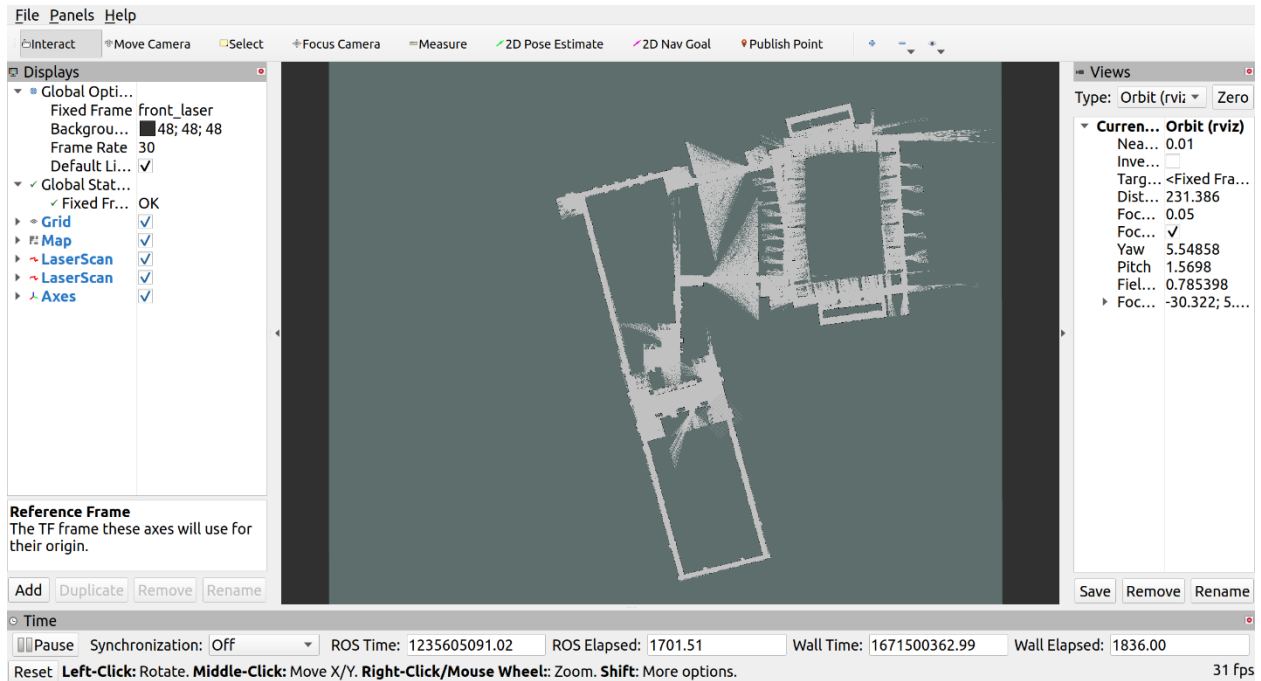


FIGURE 12 FRONT_SCAN OUTPUT MAP

The obtained result from the gmapping server and the rear_scan simulation is shown below.

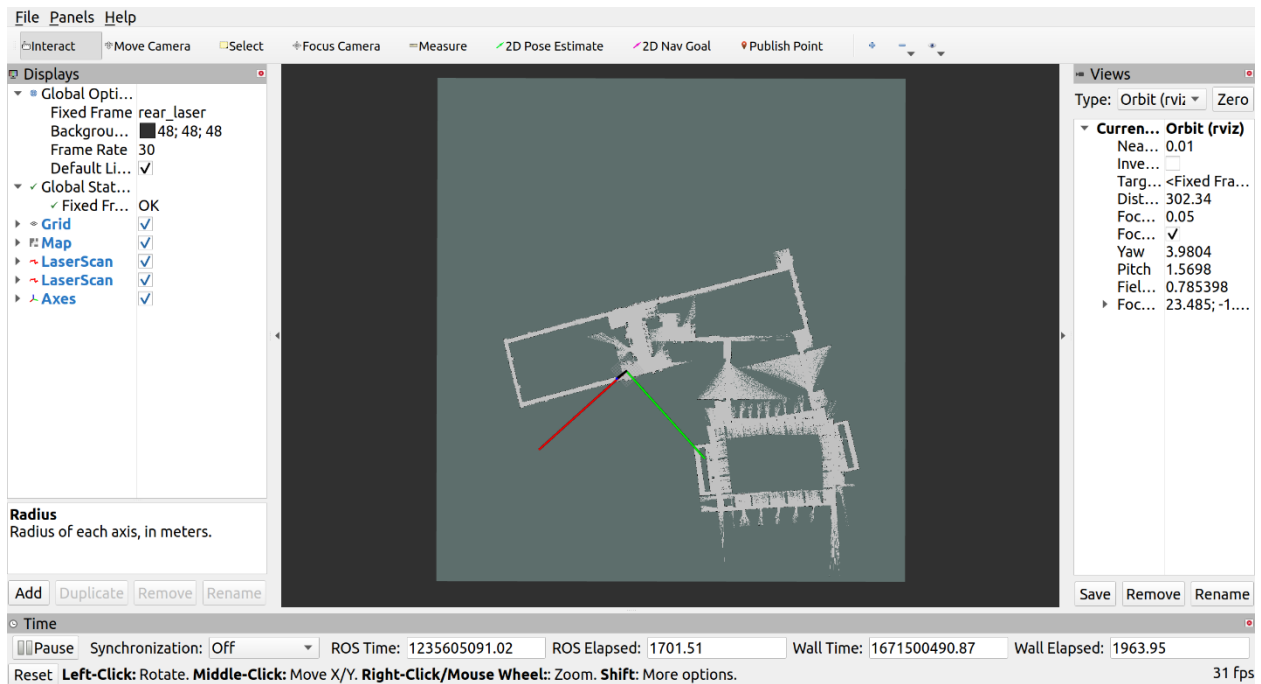


FIGURE 13 REAR_SCAN OUTPUT MAP

Discussion:

The Map is generated by performing various command lines, and the Gmapping is developed in the RVIZ for laser. The CSV files were converted into the bag, and the bag was simulated to visualize the Laser MAP for front_scan and Rear_scan, as shown in the above figures.