A Robotic Framework for Analyzing Simultaneous Localization and Mapping Algorithms using Robotic Operating System

W.A.D.K.Gunawardhana, C.J. Basnayakege*, L.S.K. Udugama


*Corresponding Author: email: cjbas@ou.ac.lk, Tele: +94714819196

Abstract-This paper deals with the task of developing a robotic framework for analyzing Simultaneous Localization and Landmark Mapping (SLAM) algorithms using the Robotic Operating System (ROS). The robot capability to navigate in a mapped or unmapped indoor environment according to the identified landmarks. The Beagle bone single board computer (SBC) which runs the ROS platform reads the inputs using the Light Detection and Ranging (LiDAR) sensor, process the data and feeds the outputs signals to the motor controller of the autonomous mobile robot. The LiDAR sensor sends the data to the SBC as a binary data stream. The main purpose of the robot development is to the analysis of Simultaneous Localization and Mapping (SLAM) algorithms and identify their performance to apply to an autonomous outdoor grass cutting robot.

For this research, a probabilistic robotics approach is used. It represents describing uncertainty using the calculus of probability theory. These probabilistic robotics algorithms represent information by probability distributions over a whole space of possible hypothesis. These probability algorithms enable to accommodate all the sources of uncertainty. SLAM algorithms are probabilistic algorithms. In this research, FastSLAM, and tinySLAM algorithms are used and the aim is to test the accuracy of these algorithms when identifying landmarks of an unknown environment using ROS.

For the localization and mapping of the robot, ROS Navigation Stack is used because the developed autonomous robot has a differential wheel and Navigation Stack is specially designed for these types of robots. It holds all the information of the sensors attached to the robot using Transform Frames (TF) software library. This software library manages the transform tree which has information about each sensor attached to the robot. Also, this robot has a 3D model in which real-time shows the motion and the scanned map of the environment. Real-time visualization is envisaged by using Robot Visualization (RViz) tool in the ROS. After scanning the area, the map of the scanned environment can be plotted and test the accuracy of the maps. Usually, the data that comes from LiDAR sensor is saved in ROS bag files in binary format.

Keywords: ROS, SBC, LiDAR, SLAM, RViz, Robot, Robot Visualization, LRF

Nomenclature

s - Pose of the robot
z - Sensor observations
u - Robot control

Greek Letters
θ - Position of the landmarks
Θ- Set of all landmark positions
1 INTRODUCTION

Simultaneous Localization and Mapping (SLAM) is an active research area in robotics. The first mention of SLAM problem occurred at the 1986 IEEE Robotics and Automation Conference held in California. This was a time when the probabilistic methods were starting to be introduced into both robotics and Artificial Intelligence (AI) research areas and applications. The goal of the SLAM problem is to create a consistent map of an unknown environment using sensors attached to a mobile robotic platform. When the mobile robotic platform to be placed at an unknown location in an environment, the robot incrementally builds a consistent map of this environment while simultaneously determining its location within this map. With the expansion of the robotics research, the SLAM task becomes more important. There are several possible applications in agriculture, industry, entertainment and security areas (Petr & Miroslav, 2016).

SLAM has been formulated and solved as a theoretical problem in several different forms. SLAM has also been implemented in several different domains from indoor robots, to outdoor, underwater and airborne systems. At a theoretical and conceptual level, SLAM can now be considered a solved problem. However, substantial problems remain in practically realizing more general SLAM solutions and notably in building and using perceptually rich maps and robotic framework as part of a SLAM algorithm (Durrant-Whyte & Bailey, 2006).

This research involves the design and development of a robotic framework for analyzing simultaneous localization and mapping algorithms. The main purpose of this framework to provide a platform to analysis e SLAM algorithms and use those algorithms to autonomously navigate a robot according to input data grabbed from a Laser Range Finder (LRF). This framework consists of a software platform and a robot. The software platform works as an extension library to Robotic Operating System (ROS), and it may help to process data quickly in the Beagle Bone Black single board computer (SBC) and output that processed data to actuators as signals to intelligently move a prototype mobile robot in an indoor environment.

2 LITERATURE SURVEY

2.1 Similar Systems

There were some similar software platforms by different companies that can be used to develop and test SLAM algorithms. They were,

- CARMEN
- Mobile Robotic Programming Toolkit (www.mrpt.org)
- Microsoft Robotics Developer Studio (MRDS)

Some of these algorithms were proprietary and some do not have a good community around it.

2.1.1 CARMEN

Carnegie Mellon Robot Navigation Toolkit (CARMEN) is an open source mobile robot programming toolkit that supports localization, navigation, mapping, obstacle avoidance
and sensor control developed by a Carnegie Mellon University Research Team. As mentioned this framework supports sensors such as LRF, Sonar sensors, Hokuyo Infrared (IR) sensors, etc. However, the problem was, it has not community similar to ROS, and there are no development releases after the year 2008 (Michael, et al., 2017).

2.1.2 Mobile Robotic Programming Kit

Mobile Robotic Programming Kit (MRPT) is an open source cross-platform C++ robotics programming toolkit that helps the robotics researchers to do test experiments about SLAM algorithms, but it is user specific. It has not community same as ROS as reported. It supports Extended Kalman Filtering (EKF), SLAM algorithms and able to grab datasets from robotic sensors (Jose-Luis, et al., 2017).

2.1.3 Microsoft Robotic Developer Studio (MRDS)

Microsoft Robotics Developer Studio (MRDS) is suitable for simulation and testing different robotic algorithms but it is proprietary and only runs on expensive hardware such as desktop computers. It does not support Beagle bone SBC (Microsoft, 2017).

By considering the above systems and their community support, the selected framework for the research is ROS because it has larger growing community and has many libraries to research on robotics. Also, it is distributed as an opensource software which can be freely accessible.

2.2 Theoretical background

2.2.1 SLAM algorithm

In this research, SLAM on ROS and runs on Beagle bone hardware. Here a mobile robot that placed in an unknown environment and building up a map of the environment and localizing in that map according to landmarks. Here the expectation is to identify the landmarks in a two-dimensional (2D) map.

In SLAM algorithm it takes a series of readings over discrete time frames. It computed an estimate of the robot’s location and a map of the environment. Here all the quantities were usually in probabilistic logic. Then applied a Bayes’ theorem to this probability function rule then it expressed as below format.

The position of the mobile robot at time \( t \) denoted as \( s^t \). This means the position of the robot in a 2D plane. The position of the robot at every time step can be written as the following equations where the symbols depicted in Table 1 below.

\[
s^t = \{s_1, s_2, s_3, ..., s_t\} \quad (1)
\]

The set of N landmark locations can be written as \( \{\theta_1, \theta_2, \theta_3, ..., \theta_N\} \) and it is equal to,

\[
\Theta = \{\theta_1, \theta_2, \theta_3, ..., \theta_N\} \quad (2)
\]

The control of the robot at time \( t \) can be written as \( u^t \). So then the set of all controls can be written as,

\[
u^t = \{u_1, u_2, u_3, ..., u_t\} \quad (3)
\]
The observations that the robot get on the navigation can be written as \( z_t \). So the set of all the observations can be written as,

\[
z^t = \{ z_1, z_2, z_3, ..., z_t \}
\]  

(4)

After applied the identity of the landmark corresponding to the observations \( z_t \) as \( n^t \) where \( n \in \{1, ..., N\} \) then the set of all data identities were expressed as,

\[
n^t = \{ n_1, n_2, n_3, ..., n_t \}
\]  

(5)

Then according to the probability was expressed as,

\[
P ( s^t, \theta | z_t, u_t, n_t )
\]  

(6)

### Table 1: Symbols of the equations

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( s_t )</td>
<td>The pose of the robot at time ( t )</td>
</tr>
<tr>
<td>( s^t )</td>
<td>Set of all positions</td>
</tr>
<tr>
<td>( \theta_N )</td>
<td>The position of the ( N )-th landmark</td>
</tr>
<tr>
<td>( \Theta )</td>
<td>Set of all ( N ) landmark positions</td>
</tr>
<tr>
<td>( z_t )</td>
<td>Sensor observation at time ( t )</td>
</tr>
<tr>
<td>( z^t )</td>
<td>Set of all observations</td>
</tr>
<tr>
<td>( u_t )</td>
<td>Robot control at time ( t )</td>
</tr>
<tr>
<td>( u^t )</td>
<td>Set of all controls</td>
</tr>
<tr>
<td>( n_t )</td>
<td>Data associations of observation at time ( t )</td>
</tr>
</tbody>
</table>

If SLAM represented as dynamic Bayes’ network, it could be shown in Figure 1 below.

![Figure 1: SLAM as a Dynamic Bayes Network](image-url)
Above mentioned SLAM algorithms have been used as one mathematical model in the Robotic Operating System (ROS)(Thrun, 2011).

There are many SLAM algorithms developed in the world. From those algorithms, the SLAM algorithms listed in below have been chosen for this research.

1. Fast SLAM
2. DP SLAM
3. Tiny SLAM (Core SLAM)

2.2.2 Fast SLAM

FastSLAM is a good SLAM algorithm and the supported by the Robot Operating System with g-mapping. It is using Rao-Blackwellized particle filtering based on landmarks. In this SLAM algorithm, each landmark is represented by a 2x2 Extended Kalman Filter (EKF). So, each particle has to maintain M EKFs (Here M number of map features). The complexity of Fast SLAM is stated as follows,(Serrata, 2017),

- Constant time per particle $O(N)$
- Log Time per particle $O(N \times \log(M))$

M= Number of Map Features

N=Number of particles

2.2.3 DP-SLAM

In this algorithm, it works by maintaining joint distribution over robot positions and maps using a particle filter. Each and every particle corresponds to a distinct hypothesis reference to the map and the robot’s position and orientation within the map. Here the map consists of M map features and particle filter maintains P particles. Considering the complexity of the algorithm, implementation of particle filter that maintains distribution would require $O(\text{AP})$ work per iteration. DP-SLAM exploits the redundancy between the different maps to avoid copying maps in the resampling phase of the particle filters. It can perform a semantically equivalent computation and may copy with less effort. It maintains the ancestry tree of all the particles. Ancestry tree contains all of the current particles as leaves and parent of a given node represents the particle of the previous iteration from which that particle was resampled. DP-SLAM maintains only a single grid map. The efficiency of the localization step is directly affected by the size of the ancestry tree. Any node in the tree can absorb at most A from iteration i and ancestry tree is minimal has P leaves and at most P interior nodes. No interior node can absorb observations from two different particles from iteration i, the total number of observation merges resulting from the observation made at iteration i is bounded by AP. Here each observation merge costs $O(\log P)$. So in this algorithm, the cost for amortized analysis is $O(\text{AP} \times \log P)$. DP-SLAM 2.0 algorithm is used in this research(Stachniss, et al., 2017).

2.2.4 Tiny SLAM (Core SLAM)

This algorithm is the smallest SLAM algorithm. Unlike DP-SLAM tiny SLAM stores single position and single map at a time. So it shows that tiny SLAM needs less memory than DP-SLAM. Because ofDP-SLAM needs more gigabyte data to process and graph a map.
Using the map and new odometry, tiny SLAM guesses the best new position for the robot and updates the map accordingly. This algorithm contains less than 200 lines of C code. It is the simplicity of this algorithm.

By considering the above algorithms, tiny SLAM is the smallest algorithm. DP-SLAM is the more accurate algorithm because it can process larger maps. DP-SLAM similar to the Fast SLAM but the difference is each particle has a history of positions and maps. Fast SLAM is used as an alternative to EKF (Extended Kalman Filter) SLAM algorithm and has efficient time complexity (Stachniss, et al., 2017).

2.2.5 Robotic Operating System (ROS)

Robotic Operating System (ROS) is an open-source software library or a meta operating system that runs on mobile Linux operating systems. Meta Operating System means it runs on the main operating system that works as a separate operating system to do a specific task which the main operating system cannot achieve. It uses host operating systems’ services to get the power of its tasks. Robotic Operating System implements the Adaptive Monte Carlo Localization (AMCL) to grab the data from a Laser Range Finder (LRF) or Laser Illuminated Detection and Ranging (LiDAR) device. AMCL is probabilistic localization method for a robot moving in a 2D environment. It uses a particle filter to track the position of a mobile robot against a known map. On startup, AMCL initializes its particle filter system according to the parameters provided. AMCL has many configuration options that will affect the performance of the localization. AMCL also can be extended to work with other ranging sensors such as sonar or Infrared Sensors (IR) (Foundation, 2017).

3 DESIGNS OF HARDWARE AND SOFTWARE

3.1 Design of hardware

3.1.1 Block diagram of the robotic framework
Figure 2 depicts the block diagram of the robotic framework. There are two general purpose microcontrollers which are used separately from the main SBC for controlling the motors. SRF10 ultrasonic sensors used for detecting obstacles in the field to facilitate autonomous navigation. Each SLAM algorithm can be configured using a desktop/laptop computer via SSH.

3.2 Design of software

3.2.1 Solution Algorithm

The main flow of the robotic framework is shown in Figure 3. LiDAR scanner collects the environment data as a distance from the scanner (robot) to each object. Based on this distance information, localization and mapping can be computed using each SLAM algorithm.

![Flow chart: SLAM robot](image)

Figure 3: Main flow of the robotic framework
3.2.2 Customizations of the ROS navigation stack

The detail description of customization of the ROS navigation stack is shown in Figure 4.

Figure 4: Customized Navigation Stack of the ROS

In this research, the software platform is the main actor. Here the “ROS Navigation Stack” was selected because it facilitates for moving differential robots. The navigation stack needs to know the position of the wheels, joints, and the sensors attached to it. The Transform Frames (TF) library which manages a transform tree of the ROS was used to define it. This library is capable of computed complex mathematical calculations. The user can customize this library when the user wanted to add more sensors to the robotic framework. The TF library builds all the relation of each sensor attached to the robotic framework.

As an example, to add a new sensor to the robotic framework, it also needs to add a new frame to the transform tree with the dimensions. After adding a new frame, it helps to easily identify the position of the sensor according to the wheels of the base. Calling the TF library gives the transformations of that sensor.

Specially created launch file was used which can run all the custom created ROS nodes concurrently. It was beneficial when running the robot as autonomously. Individual restart needed for each algorithm to runs the autonomous robot. Because of one algorithm can be responsible for process data when reads from the laser range finder and pose the robot in
a known environment. The basic file structure of SLAM robot package was implemented as follows.

Here “catkin_create_pkg slambot” command used to create the slambot package. Modular “catkin_make” build method used to create the slambot package(s) because it is better than oldest “rosbuild cmake” package build methods. The directory structure of the current configuration as shown in Figure 5.

![Figure 5: Directory structures of the source codes and launch files of the robotic framework](image)

It defined the robot’s movement environment in a *.yaml file. It is namedas “custom_map.yaml.” It has the virtual environment and virtual objects. After the robot is started in a real environment, it moves through the area. While scan, the “rosbag record” command will store the data in a bag file. Then in the next round when it navigates using with the help of “RViz” graphical tool, it will able to identify the object which is not defined in the map file. This is the basic concept work in this robotic framework.

The “CMakeLists.txt” has the description about the all C++ source files and ability compile all *.cpp files concurrently.

The “slamrobot_test.launch” file has the description about laser node, base_controller node, broadcaster and listener nodes and cost map files and description about map and bag files and when it launches through the Linux terminal, the robot can work autonomously.

ATmega32A microcontroller is used to communicate with Beagle Bone Black SBC that sends the odometry data from sensors and actuators via the RS232 protocol. Custom wrote C++ header file which follows the rules of “rosserial” library developed for ATmega32A to convert odometry data into ROS messages. The received data will be used in the ROS navigation stack to avoid the obstacles in a known recorded map environment.

Separate ATmega8A microcontroller used to control the rotation mechanism of continuous rotation servo motor. This helps to start or stop the servo motor to control the RpiLiDAR.
4 FABRICATION OF HARDWARE

Figure 6: Upper deck of the robot

Figure 6 depicts the upper deck of the robot. It consists of rotating LiDAR Lite Rangefinder, continuous rotation servo motor, 3.3v to the 5v level shifter, Inter-Integrated Circuit (I2C) hub for LiDAR Lite Rangefinder, Servo motor signal controller and two of LM2596 switching mode regulators.

Figure 7: Side view of the robot
Figure 7 depicts the side view of the robotic framework. The battery packs were included between the upper and lower decks of the robotic framework’s body with all the other components.

5 EXPERIMENTAL SETUP

This system worked as a Simultaneous Localization and Mapping robotic framework. After the system powered on, logged in to robotic framework’s Linux terminal via secure shell (SSH) protocol and started the “roscore” to power on the ROS meta operating system. Three algorithms “FastSLAM”, “DP-SLAM” and “tinySLAM” were selected here. Each algorithm runs with the robot with the individual restart. To launch each algorithm, separately used specific launch files for each algorithm. When the robot moves through the unknown area, it scanned and recorded the objects around it. Custom wrote ROS package and rosnodes were used for laser scanning. The “rosbag” file responsible for recording that information after the first run of the robot. When playing the “rosbag” file for each and every algorithm, it was able to generate maps according to the laser and odometry data included in the rosbag file. For the indoor autonomous robot navigation, the “RViz” graphical tool visualizes the autonomous navigation when the robot starts to move according to the generated map with the help of configured SLAM algorithm.

Figure 8 depicts the experimental setup of the research. There are four obstacles around the robot and the robotic framework configured with each SLAM algorithm to analyse the behavior of each algorithm in the same environment.

Figure 8: Coverage area of the autonomous robot
6 RESULTS AND DISCUSSION

6.1 Results

Figure 9 shows the customization steps of the LiDAR lite sensor on the ROS node.

![Figure 9: Customization of lidar lite on the ROS node](image)

Figure 10 shows the list of ROS topics that are running on the ROS.

![Figure 10: ROS topics that are currently running which includes the laser node](image)

Figure 11 shows the RViz window that shows the laser topic and the odom_combined topic as transfer frames (TF) for TinySLAM algorithm.
Figure 11: RViz window that shows the laser topic and the odom_combined topic as transfer frames (tf) for tiny SLAM

Table 2 depicts the comparison of the actual distance and the LiDAR sensor reading for one object in the different distance on the experiment setup. There is an average 2.5 cm error in the sensor reading when compared with the actual distance. According to the datasheet of the LiDAR sensor, there is ±2.5 cm error found as specified by the sensor manufacturer.

Table 2: LiDAR Sensor readings compared with actual measured distances

<table>
<thead>
<tr>
<th>Actual Distance (cm)</th>
<th>LiDAR sensor readings (cm)</th>
<th>Error in readings (cm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>150</td>
<td>148</td>
<td>2</td>
</tr>
<tr>
<td>100</td>
<td>97</td>
<td>3</td>
</tr>
<tr>
<td>90</td>
<td>87</td>
<td>3</td>
</tr>
<tr>
<td>80</td>
<td>78</td>
<td>2</td>
</tr>
<tr>
<td>70</td>
<td>67</td>
<td>3</td>
</tr>
<tr>
<td>60</td>
<td>57</td>
<td>3</td>
</tr>
<tr>
<td>50</td>
<td>48</td>
<td>2</td>
</tr>
<tr>
<td>40</td>
<td>39</td>
<td>1</td>
</tr>
<tr>
<td>30</td>
<td>27</td>
<td>3</td>
</tr>
<tr>
<td>20</td>
<td>17</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>8</td>
<td>2</td>
</tr>
</tbody>
</table>
Without correcting the LiDAR sensor reading, the same sensor readings are applied to the selected SLAM algorithms named FastSLAM and TinySLAM. Table 3 shows the distance, the time consumed and reading error for the FastSLAM algorithm.

Table 3: Distance, time, and accuracy for FastSLAM algorithm

<table>
<thead>
<tr>
<th>LiDAR sensor readings (cm)</th>
<th>Distance computed FastSLAM (cm)</th>
<th>Error in readings (cm)</th>
<th>Time Taken (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>148</td>
<td>147</td>
<td>1</td>
<td>500</td>
</tr>
<tr>
<td>97</td>
<td>96</td>
<td>1</td>
<td>490</td>
</tr>
<tr>
<td>87</td>
<td>87</td>
<td>0</td>
<td>500</td>
</tr>
<tr>
<td>78</td>
<td>76</td>
<td>2</td>
<td>480</td>
</tr>
<tr>
<td>67</td>
<td>65</td>
<td>2</td>
<td>490</td>
</tr>
<tr>
<td>57</td>
<td>56</td>
<td>1</td>
<td>500</td>
</tr>
<tr>
<td>48</td>
<td>48</td>
<td>0</td>
<td>480</td>
</tr>
<tr>
<td>39</td>
<td>39</td>
<td>0</td>
<td>500</td>
</tr>
<tr>
<td>27</td>
<td>26</td>
<td>1</td>
<td>500</td>
</tr>
<tr>
<td>17</td>
<td>16</td>
<td>1</td>
<td>500</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>0</td>
<td>490</td>
</tr>
</tbody>
</table>

There is an average less than a 1cm error in the FastSLAM algorithm according to the LiDAR sensor readings and average 495ms time taken to complete the task for the FastSLAM algorithm.

Table 4 shows the distance, the time consumed and reading error for the TinySLAM algorithm.

Table 4: Distance, time, and accuracy for TinySLAM algorithm

<table>
<thead>
<tr>
<th>LiDAR sensor readings (cm)</th>
<th>Distance computed TinySLAM (cm)</th>
<th>Error in readings (cm)</th>
<th>Time Taken (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>148</td>
<td>145</td>
<td>3</td>
<td>320</td>
</tr>
<tr>
<td>97</td>
<td>94</td>
<td>3</td>
<td>310</td>
</tr>
<tr>
<td>87</td>
<td>86</td>
<td>1</td>
<td>290</td>
</tr>
<tr>
<td>78</td>
<td>76</td>
<td>2</td>
<td>310</td>
</tr>
<tr>
<td>67</td>
<td>65</td>
<td>2</td>
<td>320</td>
</tr>
<tr>
<td>57</td>
<td>55</td>
<td>2</td>
<td>280</td>
</tr>
<tr>
<td>48</td>
<td>46</td>
<td>2</td>
<td>310</td>
</tr>
<tr>
<td>39</td>
<td>37</td>
<td>2</td>
<td>300</td>
</tr>
<tr>
<td>27</td>
<td>25</td>
<td>2</td>
<td>320</td>
</tr>
<tr>
<td>17</td>
<td>16</td>
<td>1</td>
<td>290</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>1</td>
<td>300</td>
</tr>
</tbody>
</table>
According to the Table 4, there is an average 2 cm error in the TinySLAM algorithm according to the LiDAR sensor readings and average 300 ms time is taken to complete the task for the TinySLAM algorithm.

6.2 Discussion

Numerous investigations about existing computers and processing boards were done before doing this research which runs the ROS. Therefore, desktop, laptop computers, Raspberry Pi and the beagle bone black SBC were selected. Since, the requirement to make a portable robotic framework, the desktop and laptop computers were not considered. Thus, Raspberry Pi and BeagleBoneBlack processing boards were chosen. Next, the Raspberry Pi processing board was eliminated due to a limited number of GPIO pins, and it was based on ARM version 6 processor. Then, Beagle Bone Black processing board was selected because it has ARM version 7 Texas Instruments processor and 46 of GPIO pins. After that, the Ubuntu 14.04 armhf version was installed for beagle bone black processing board. Finally, ROS indigo meta operating was fixed for Ubuntu 14.04.

Also, three SLAM algorithms: Fast SLAM, tiny SLAM and DP-SLAM were taken to compare the accuracy of maps which were generated through the data collected from LiDAR sensor. However, due to resource limitation of the experiment setup, the DP-SLAM algorithm was not implemented. Hence, those generated maps were used for autonomously navigate a robot through a known environment. Here, Monte Carlo localization was used for autonomous navigation inside ROS.

When comparing selected SLAM algorithms, the map generated by the FastSLAM algorithm is sharper than the map generated by the TinySLAM algorithm. However, FastSLAM took considerable time to process the map generation process with compared to the TinySLAM algorithm. Even though, TinySLAM showed significant distance error with compared to the FastSLAM algorithm.

To configure the PWM devices such as servo motors for rotating the LiDAR sensor, special separated microcontroller was used. And also, a specific “device tree” overlay file was used to get the encoder readings from IR sensors because beagle bone black processing board has no Basic Input Output System (BIOS) to identify the devices attached to the system when the system power on. Addition to that, the facility to supervise the robot’s each and every sensor remotely was added and scanned the laser data using the Robot Visualizer (RViz) tool.

7 CONCLUSION

In this research, the main aim was to develop a Robotic Framework for Analyzing Simultaneous Localization and Mapping Algorithms using the Robotic Operating System. The selected components and the fabrication of the robotic framework were successful. The selected Beagle Bone Black SBC that act as ROS master node and succeeded with two of SLAM algorithms out of three that process the laser data from LiDAR sensor. The 3D rendering of Beagle Bone Black SBC was not at a satisfying rate and the “RViz” graphical tool freezes well when generating maps or sending 2D pose commands for navigation. The DP-SLAM algorithm was unable to run from this processing board because it needs more processing power and considerable memory. Also, the custom coded “rosserial” C++ wrapper unable to communicate with the serial port of the Beagle Bone Black processing board.
The concept used in the robotic framework can be applied to build a robot that can move through an outdoor area used to cut grass after some modifications. The rotational LiDAR unit can be used without any modifications for the outdoor robot. Intel NUC processing board can be used as the main processing board instead of Beagle Bone Black processing board. Intel NUC has a good processor such as Intel Core i5 and will be well suited to execute the SLAM algorithms similar to DP-SLAM.

REFERENCES


