COMPARATIVE STUDY OF ROS ON EMBEDDED SYSTEM FOR A MOBILE ROBOT

Submitted: 16th August 2018, accepted: 25th October 2018

Min Su Kim, Raimarius Delgado, Byoung Wook Choi

DOI: 10.14313/JAMRIS_3-2018/19

Abstract:
This paper presents a comparative study of Robot Operating System (ROS) packages for mobile robot navigation on an embedded system. ROS provides various libraries and tools in developing complex robot software. We discuss the process of porting ROS to an open embedded platform, which serves as the main controller for a mobile robot. In the case of driving the robot, ROS provides local path planners such as the base, elastic band, and timed elastic band. The local planners are compared and analyzed in terms of accuracy in tracking the global path conducted on a robot model using Gazebo, 3D simulation tool provided by ROS. We also discussed the difference in performance of deploying ROS packages on a personal computer and on the embedded environment. Experiments were performed by controlling two different mobile robots with results showing that tracking error is highly dependent on the goal tolerance. This study will serve as a promising metric in improving the performance of mobile robots using ROS navigation packages.

Keywords: *Robot Operation System, mobile robot, embedded system, navigation, SLAM, path planner*

1. Introduction

Mobile robots are widely used in various fields, especially in scientific, industrial, and governmental sectors. However, the combination of devices and software are getting more complex and difficult to develop. Robot Operating System (ROS), one of the most popular robotic framework, is an open source meta OS that provides control algorithms and supports different hardware devices for mobile robot operation [1]. ROS focuses on software modularization and easy redistribution. As a result, the development of robots has become easier and innovative software are conveniently shared within the community [2], [3].

Development time and expenses are very essential factors to consider in robot distribution. In the case of commercial robots, cheaper price has been proven to improve market stability and helped in increasing sales [2]. On the contrary, expensive robots have trouble in selling and are very hard to access. In this paper, we utilize a low cost and high efficient open embedded platform for the main controller of mobile robots using ROS [4]. This minimizes the development costs and enhances portability as embedded systems are cheaper and smaller in comparison to the widely used personal computers.

However, software development on an embedded environment is difficult because all the software must be compatible with each other and with the embedded platform itself. The availability of technical documents and support is very limited. This paper provides the detailed procedure of successfully porting ROS on a Raspberry Pi 3 (RPi3), one of the leading open embedded platform used in robotic applications [5].

Controlling a mobile robot requires considerable amount of computation. ROS provides a navigation package that contains global and local path planning algorithms [4]. The local path planners included in ROS are defined as base [6], elastic band (EBand) [7], [8], and timed elastic band (TEB) [9]. Simultaneous localization and mapping (SLAM) is made easier with ROS [10]–[12].To utilize the navigation package, distance sensors is attached to a mobile robot such as a laser rangefinder (LRF) to detect the obstacles within the environment and measure the distance between the detected obstacle and the robot to perform an avoidance scheme.

In this paper, the performance of the local planners is analyzed and compared in terms of tracking a given global path on a robot model designed using Gazebo, the 3D simulation tool included in ROS. We designed a robot model based on a commercial mobile robot using SolidWorks. As Gazebo requires high 3D graphics that could not be supported by the RPi3, simulations were performed on a desktop computer. Actual driving experiments were conducted on two commercial mobile robots. The RPi3 serves as the main controller, responsible for acquiring sensor data,measuring the position of the mobile robot within the environment, calculating the distance between the robot and the obstacles, and driving the mobile robots.

2. ROS Basic Concepts

ROS is often called a “meta” OS. Although ROS is not a traditional operating system, it provides a variety of services [1]. ROS processes are called nodes which are independent with each other managed by a master node. Message passing between nodes is classified into: a topic or a service. Unlimited number of nodes can either subscribe or publish on the same topic. Topics are

usually used for continuous data streams such as sensor data and robot status. On the other hand, a service only provides communication between a host and a client service node. It is recommended to use services for remote procedure calls that terminates quickly.

A robot software based on ROS is divided into hardware-independent and device-specific parts as shown in Fig. 1. The hardware-independent part is composed of ROS core, other ROS native software, and algorithms shared by different developers to the ecosystem. The device-specific part contains the local information of the robot, which includes the connected sensors, kinematics, and other necessary information to operate the robot [5]. The only task of a user is to create the device-specific node according to the specifications of the robot in hand. The hardware-independent part does not require any modification on the code itself, the user is only advised to change the configurations according to the required functionality.

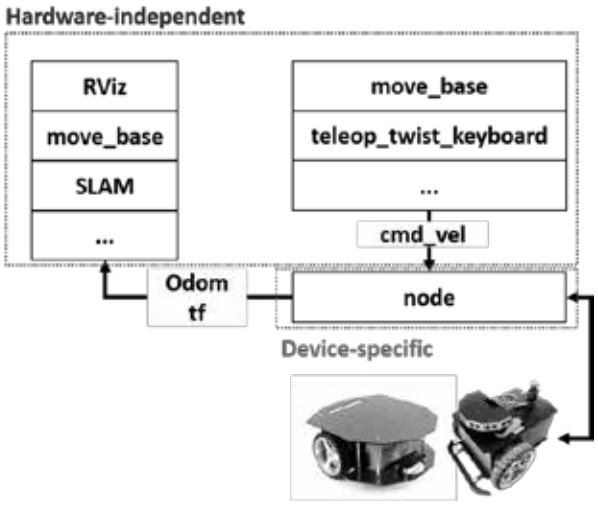


Fig. 1. An Example of using Hardware-Independent Software and Device-Specific Drivers

As an example, a device-specific robot driver can be used with a variety of hardware-independent ROS packages such as *teleop_twist_keyboard* and *move_base*. The *teleop_twist_keyboard* is a package for remote operation using keyboard, and the *move_base* is a package for navigation. Both packages publish messages which contain linear and angular velocities of mobile robot. The device-specific robot driver receives these messages and converts them to joint space velocities for actual operation of the mobile robot. The speed of the point space for robot operation varies depending on the robot's kinematics, so the related software should be changed accordingly, but in ROS, adding a small device-specific software can utilize various hardware-independent software without modification. Some robot manufacturers provide the device-specific node for easier handling of their users [5]. Otherwise, you have to create your own.

3. ROS Navigation Stack

The navigation stack is a hardware-independent software included in ROS to simplify navigation control of mobile robots. Adevice node should be created by the user, which supplies the odometry information, facilitates the sensor stream, and processes velocity commands sent to the physical components of the robot [4], [13]. The navigation stack should be configured in accordance to the physical characteristics and dynamics of the mobile robot to perform at a high level.

The software architecture of the navigation stack is shown in Fig. 2 [13]. The main component of the navigation stack is *move_base* package, which consists of software for path planning, map building, and recovery behaviors in case the robot gets stuck. The urg_node is a node created to acquire sensor data from an LRF and publishes it as a topic called '/scan'. The mobile_node is a device node for a specific mobile robot as mentioned earlier.

Fig. 2. Software architecture of the navigation stack

Navigation stack operates in several steps for the robot to reach a specific position within the environment. The move_base acquires data from the '/scan' topic of urg_node and creates a global and local costmap which calculates the position of the mobile robot and the obstacles. The global planner generates the shortest path available for the mobile robot to reach the target position and the local planner is responsible for tracking the global path. Velocity commands generated by the local planner is published as the '/cmd_vel' topic and is subscribed by the mobile_node. Feedback control is realized with the mobile_node calculating the position of the mobile robot using encoder data and publishing it as the '/tf' and '/odom' topics received by the move_base.

3.1. Global Planner

The navigation stack contains of both global and local path planners. The global planner calculates the shortest available path from the current position of the mobile robot to the specified target position.However, the actual path that the robot is that of the local planner. Thus, this paper focuses more on analyzing the tracking accuracy of the mobile robot with the different local planners available in ROS. The global planner is configured with the default parameters as explained in [4], [6], [14].

3.2. Local Planner

The local planner tracks the global path and performs feedback control considering the actual position and movement of the mobile robot within the environment. There are several types of availalble local planners, including the base local planner [4], [6], [15], elastic band (EBand) [7], [8], and timed elastic band (TEB) [9].

3.2.1. Base Local Planner

Fig. 3 shows the basic concept of the base local planner. Possible trajectories in the velocity space are generated discretely sampled. Forward simulation is performed for each sampled trajectory for a short period to predict the outcome. The simulation results are scored according to metrics that incorporates characteristics such as proximity to obstacles, proximity to the target position, proximity to the global path, and speed. Trajectories that fails to meet any of the metrics are discarded. The trajectory with the highest score is selected and is published as the velocity commands for the mobile robot [6]. These steps are repeated until the mobile robot reaches the target position.

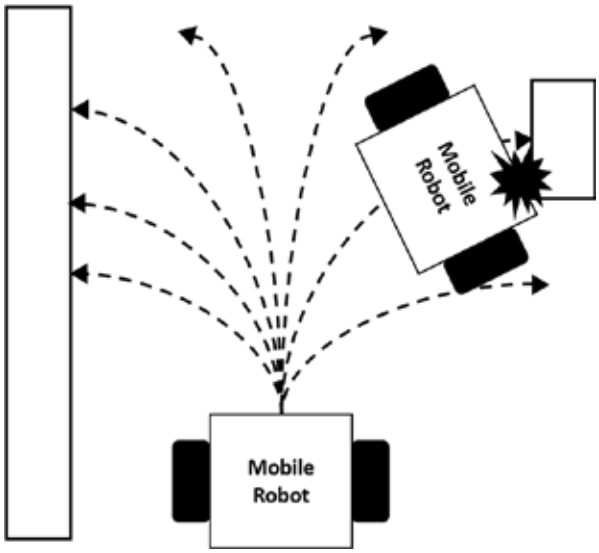


Fig. 3. Base local planner

3.2.2. EBand Local Planner

The basic concept of the EBand local planner is illustrated in Fig. 4. EBand searches for a path as it extends to both sides with external force, imitating the elastic behavior of a rubber band, and generates a path by shrinking the inside and reducing the search path by the pulling force. If additional, obstacles are encountered or detected, modify the path to the same principle, including obstacles [7], [8].

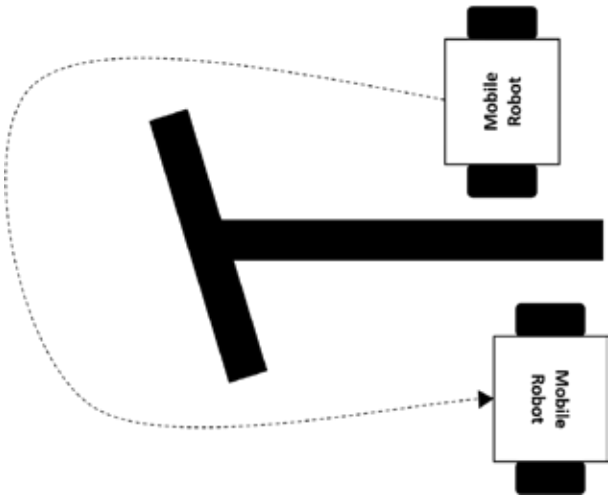


Fig. 4. Eband Local Planner

3.2.3. TEB Local Planner

Fig. 5 is a planner supplemented by adding temporal parameters to the Eband approach. As a whole, it follows the characteristics of EBand and optimizes every moment of trajectory deformation and minimizes the target cost function instead of generating and applying force [9].

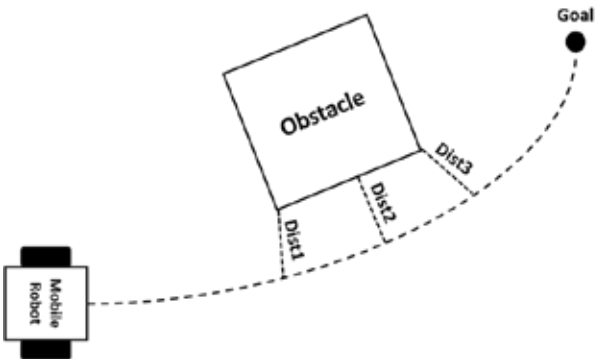


Fig. 5. TEB local planner

In this study, various global planners and local planners are applied to mobile robots, and the movement according to the type of planner is compared and analyzed, and the navigation path according to the parameters is experimented.

4. Simulation

Prior to the actual experiment using the robot, a simulation experiment was conducted to select an appropriate local planner. In the simulation, we observed the navigation of the robot using three different local planners in two situations: with an obstacle and without obstacles. The local planners in evaluation were base local planner, EBand, and TEB local planner. Among the parameters that can customize the behavior of each planner, only those related to the physical limits of the mobile robot were configured and the rest were set to default value.

4.1. Robot Model

In this study, various types of simulations were performed in a 3D environment using Gazebo, the built-in simulation tool available in ROS. All objects in the Gazebo environment are required to be defined in the Unified Robot Description Format (URDF), including the mobile robot, sensors, and obstacles. The mobile robot model is designed using the computer aided designing tool, SolidWorks, as it offers a URDF converter for easier integration to ROS and Gazebo. The 3D model of the mobile robot and the designed simulation environment in Gazebo is shown in Fig. 6.

The necessary nodes to utilize the ROS navigation package are created. We used an LRF model based on the Hokuyo URG-04LX-UG01LRF to detect obstacles and analyze the movements of the robot. As mentioned in section 2, ROS provides a node for the LRF called urg_node which acquires sensor data from the hardware and publishes them as the topic called '/ scan'. The data flow of the urg_node is shown

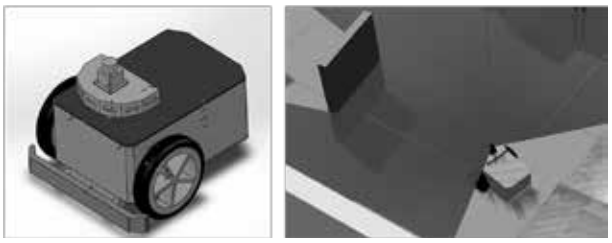


Fig. 6. Modeling in SolidWorks (Left) and simulation environment in Gazebo (Right)

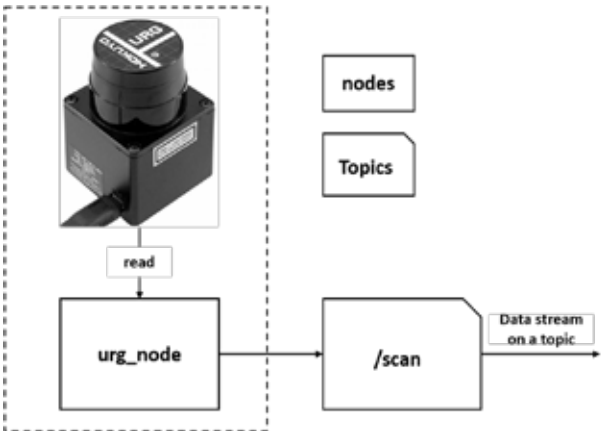


Fig. 7. LRF node flow

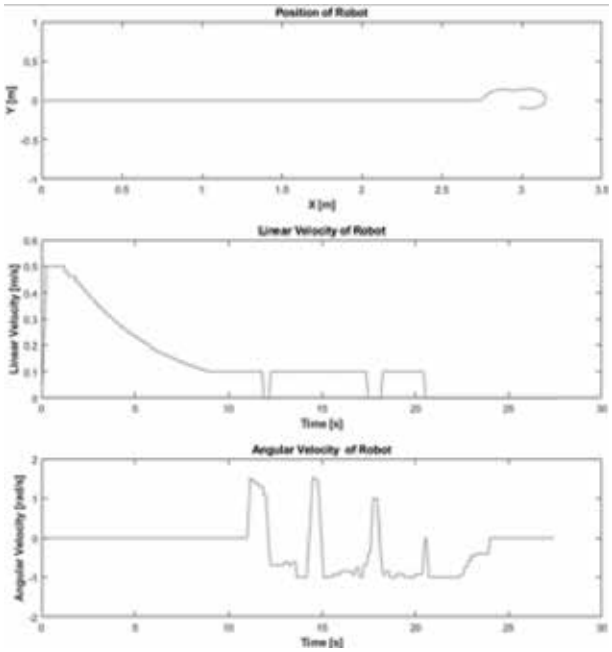
in Fig. 7 [10].The mobile_node that specifies the kinematics of the mobile robot, facilitates the sensor stream, and processes velocity commands is created for a two-wheel differential drive mobile robot. The the size is configured as 0.58 in width and 0.44 in length. The diameter of the wheels is 240, and the distance between the center of each wheel is 380.

4.2. Results of Simulation

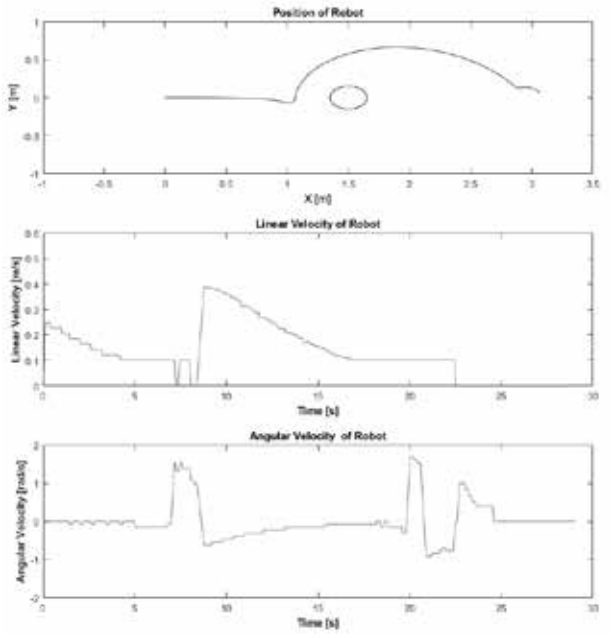
4.2.1. Base Local Planner

Fig. 8 a) shows the results of simulation using the base local planner in an environment without obstacles. It shows the changes in robot position, linear velocity and angular velocity after setting the goal position to x = 3 m, y = 0 m. After the navigation started, the linear velocity was accelerated to 0.5 m/s and the robot moved forward. The linear speed then deaccelerated to 0.1 m/s, which is the minimum operating speed of the robot.

After approximately 11 seconds from the start, the angular velocity changed, and the position of the robot in the y axis changed. Because of the changes in the linear velocity and the angular velocity, the mobile robot reached x = 2.98 m, y = 0.9 m near the target position of x = 3 m, y = 0 m, which ended the navigation procedure. Reduction in linear velocity after reaching 0.5 m/s and the changes in angular velocity after 10 seconds is observed. The linear velocity deceleration appears to reduce the velocity at which the robot reaches the target position. If the robot is actuated constantly at a high speed when it reaches the target, an oscillation may occur, violating the acceleration limit. The changes in angular velocity also prevent the oscillations. Since the small oscillation may occur even when the robot moves at the minimum speed of 0.1 m/s, the angular velocity was changed to reduce the x variation of the robot in the Cartesian space.



a) without obstacles



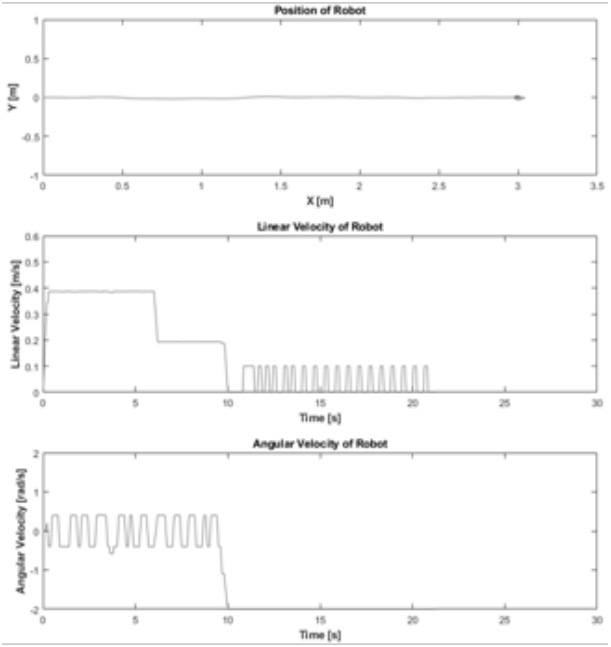
b) with an obstacle

Fig. 8. Base local planner

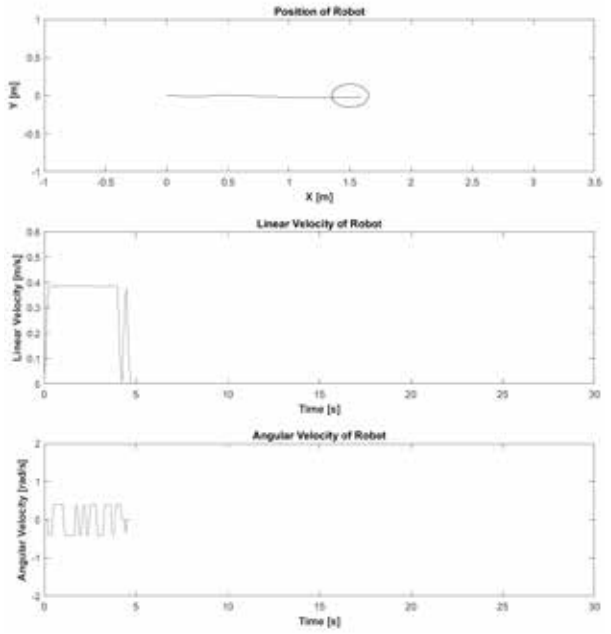
On the other hand, Fig. 8 b) shows the result of simulation with an obstacle. It shows changes in robot position, linear velocity and angular velocity for the same target position. After the navigation started, the robot moved forward to a point 0.5 m away from the obstacle. A deviation occurs to avoid the obstacle and reached the point x = 3.07 m, y = 0.06 m close to the target position without a collision. When approaching the obstacle, we can observe that a slight deceleration occurs when the robot approaches the obstacle to reduce the possibility of collision.

4.2.2. EBand Local Planner

Fig. 9 a) shows the results of simulation using the EBand local planner in the environment without obstacles. It shows changes in robot position, linear velocity and angular velocity after setting the goal position to x = 3 m, y = 0 m.



a) without obstacles



b) with an obstacle

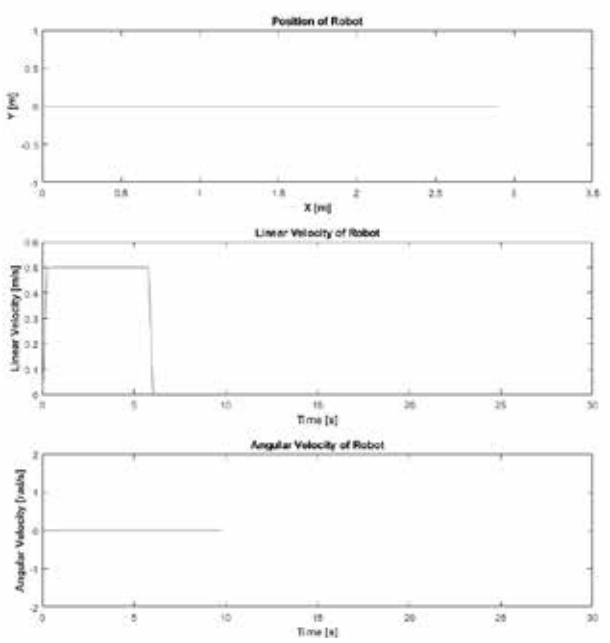
Fig. 9. EBand local planner

After the navigation started, the robot started moving forward while its heading angle was slightly shaking. After 6 seconds from the start, the linear velocity of the robot was decelerated and the decelerated velocity was maintained until the robot reached the target position. After reaching x = 3.03, y = 0.00, the robot stopped for a while. And then, an oscillation occured while adjusting the heading angle.

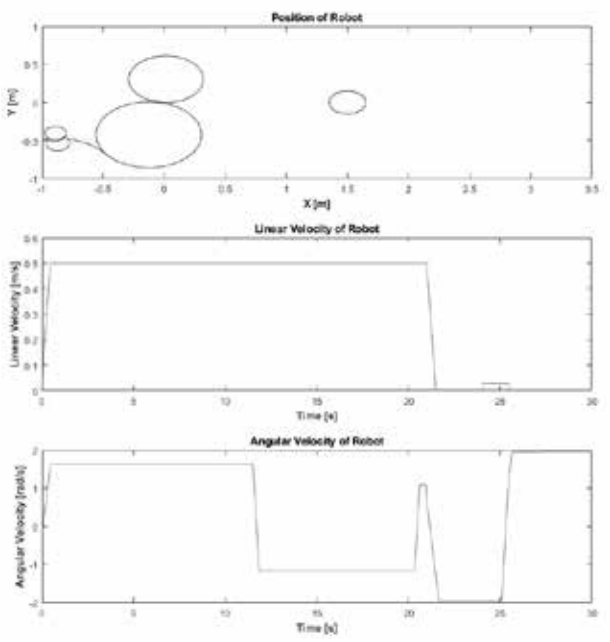
Fig. 9 b) shows the result of simulation using EBand in the environment with an obstacle. It shows changes in robot position, linear velocity and angular velocity after setting the goal position to x = 3 m, y = 0 m. The robot moved directly towards the obstacle resulting to a collision.

4.2.3. TEB Local Planner

Fig. 10 a) shows the results of simulation using TEB in the environment without obstacles. It shows changes in robot position, linear and angular velocity.



a) without obstacles



b) with an obstacle

Fig. 10. TEB local planner

The goal position is set to x = 3 m, y = 0 m. After the navigation started, the linear velocity of the robot accelerated to 0.5 m/s, and the robot moved forward. And then, it stopped at x = 2.89, y = 0. Fig. 10 b) shows the results of simulation using teb local planner in the environment with an obstacle. It shows changes in ro-

bot position, linear velocity and angular velocity after setting the goal position to x = 3 m, y = 0 m. In the environment with an obstacle, the TEB planner was not able to compute the velocity of robot on time due to high computation load. As a result, the robot failed to follow the path created by the global planner and could not reach the target.

The results of the simulation experiments show that the EBand and TEB local planners both produced exemplary results in tracking a target position in an environment in case of no obstacles. But, base local planner is the only local planner that can reach the goal in the existence of an obstacle, and others are not.

5. Experimental Specifications

The system structure of the experimental testbed including the hardware devices and mobile robots are shown in Fig. 11. We have selected an ARM-based a embedded platform, RPi3, for its portability and low cost in comparison to desktop computers. The latest available development environment is shown in the in Fig. 12. The RPi3 is installed with Ubuntu 16.04 with the Linux kernel version of 4.1.21-v7+. ROS Kinetic Kame was selected as it is the latest stable version available in the ROS repository. In this study, we only tackle the basic features of ROS without considering the strict scheduling deadlines required in robust control of robotic applications. ROS was implemented in a straight forward manner following the user guide in [16]. In more advanced control applications that requires real-time environment for an embedded platform, compatibility of ROS with the other software components is a huge issue that is complex owing to the limited availability of systematic documentations and technical support.

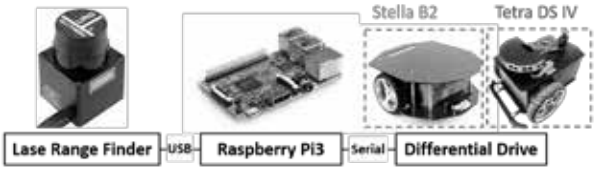


Fig. 11. System Structure to Control Two Mobile Robots

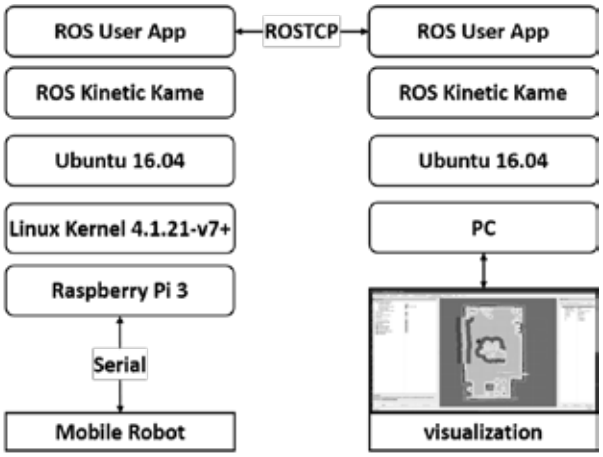


Fig. 12. Software architecture of the main controller

6. Experiment & Results

Base local planner in Section 4 is applied to the mobile robot in consideration of the various planners tested through simulation. The performance difference between two mobile robots is shown according to the presence of obstacles. Change the size of the variable of the *'xy_tolerance'* according to the parameter setting. The performance of the global planner is not significantly different. Therefore, only the local planner is applied to the experiment and compared and analyzed. The specifications of each robot are shown in the following Table 1.

Table. 1. Robot Specifications

Robot Name	Tetra DS IV	Stella B2
Wheel diameter	240 mm	150 mm
Distance between wheels	380 mm	289 mm
Width	0.58 m	0.41 m
Length	0.44 m	0.32 m

In the experiment, we used the navigation stack to move the mobile robot from one point to another. The distance between the starting point and the destination is 3, and the obstacle is installed at the starting point of 1.5 m. Fig. 13 shows two mobile robots and experimental environment. The left side of the figure is Tetra DS IV and the right side is Stella B2.

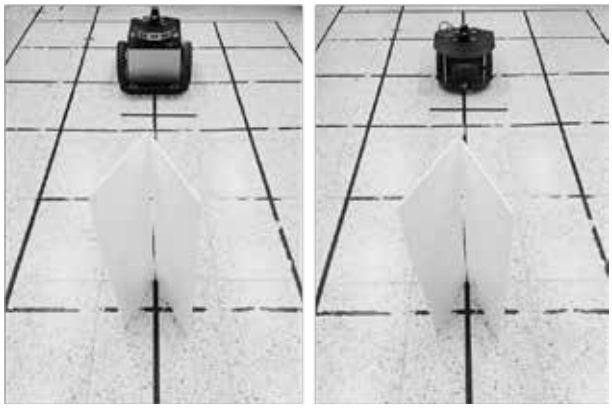


Fig. 13. The experimental environment

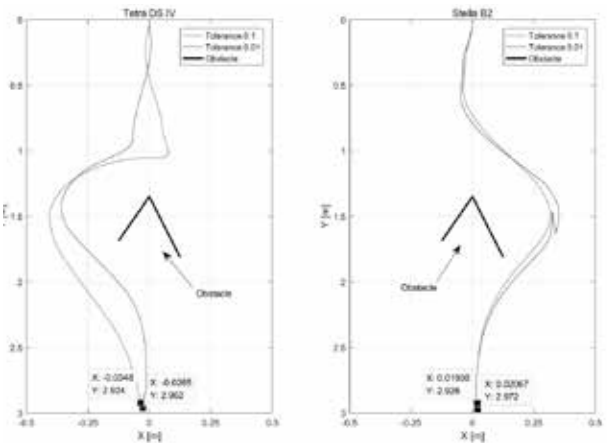


Fig. 14. Path of mobile robots

Fig. 14 shows the motion of the mobile robot. After setting the target position to , the mobile robot moved forward. And the difference value of xy_tolerance [15] parameter was changed.

The smaller the tolerance value is, the closer the target value is reached. Due to the nature of the local planner, the direction of detour is not constant because the calculation is changed every moment at the designated location. You can also see that the Stella B2 has a cleaner line out of the path than the Tetra DS IV. And the closer the shape of the mobile robot is to the circle, the better it follows the planned path.

7. Conclusion and Future Work

Robotics is an important and exciting area of research that requires the integration of various devices and complex software. Even for the embedded system, the various libraries and tools provided by ROS make it easier to access and integrate hardware and software. Moreover, various devices can be connected to ROS through ROS drivers and reduce the complexity of the development of the function by using various software components of ROS.

Local planners for SLAM and navigation functions of ROS packages were evaluated both on PC simulation and embedded system for various mobile robots. This study showed the usefulness of robot development through ROS and control of the mobile robot can be implemented more easily and quickly.

Finally, we conducted comparative study for the navigation of mobile robots according to planners when using ROS package. Results showed that developer should be more careful about using ROS packages and much effort is needed to get the desired results. Detailed research will be carried out later to obtain better results when using ROS packages.

ACKNOWLEDGEMENT

This study was financially supported by Seoul National University of Science and Technology.

AUTHOR

MinSu Kim – Dept. of Electrical and Information Engineering, Seoul National University of Science and Technology, Nowon-gu, 01811, Seoul, Republic of Korea. E-mail:min50190@seoultech.ac.kr

Raimarius Delgado – Dept. of Electrical and Information Engineering, Seoul National University of Science and Technology, Nowon-gu, 01811, Seoul, Republic of Korea. E-mail:raim223@seoultech.ac.kr

ByoungWook Choi* – Dept. of Electrical and Information Engineering, Seoul National University of Science and Technology, Nowon-gu, 01811, Seoul, Republic of Korea. E-mail:bwchoi@seoultech.ac.kr

*Corresponding author

REFERENCES

[1] "ROS.org | About ROS." http://www.ros.org/about-ros/.

[2] M. Quigley *et al.*, "ROS: an open-source Robot Operating System," *ICRA workshop on open source software*. Vol. 3. No. 3.2. 2009.

[3] Y. S. Pyo, *ROS Robot Programming*. RubyPaper Press, 2015.

[4] K. Zheng, "ROS Navigation Tuning Guide," *arXiv preprint arXiv:1706.09068*, 2017.

[5] S.-Y. Jeong *et al.*, "A Study on ROS Vulnerabilities and Countermeasure," *Proceedings of the Companion of the 2017 ACM/IEEE International Conference on Human-Robot Interaction*. ACM, 2017, pp. 147–148. DOI:10.1145/3029798.3038437.

[6] P. Marin-Plaza, A. Hussein, D. Martin, A. de la Escalera, "Global and Local Path Planning Study in a ROS-Based Research Platform for Autonomous Vehicles," *Journal of Advanced Transportation*, vol. 2018, 2018, pp. 1–10. DOI:10.1155/2018/6392697.

[7] S. K. Gehrig, F. J. Stein, "Elastic bands to enhance vehicle following,"*Intelligent Transportation Systems, 2001. Proceedings. 2001 IEEE*. IEEE, 2001, pp. 597–602. DOI:10.1109/ITSC.2001.948727.

[8] S. Quinlan, O. Khatib, "Elastic bands: connecting path planning and control," *Robotics and Automation, 1993. Proceedings, 1993 IEEE International Conference on.* IEEE, 1993, pp. 802–807. DOI:10.1109/ROBOT.1993.291936.

[9] M. Keller, F. Hoffmann, C. Hass, T. Bertram, and A. Seewald, "Planning of Optimal Collision Avoidance Trajectories with Timed Elastic Bands," *IFAC Proceedings Volumes*, vol. 47, no. 3, 2014, pp. 9822–9827. DOI:10.3182/20140824-6-ZA-1003.01143.

[10] M. G. Ocando, N. Certad, S. Alvarado, A. Terrones, "Autonomous 2D SLAM and 3D mapping of an environment using a single 2D LIDAR and ROS,"*Robotics Symposium (LARS) and 2017 Brazilian Symposium on Robotics (SBR), 2017 Latin American*. IEEE, 2017, pp. 1–6. DOI:10.1109/SBR-LARS-R.2017.8215333.

[11] R. Reid, A. Cann, C. Meiklejohn, L. Poli, A. Boeing, and T. Braunl, "Cooperative multi-robot navigation, exploration, mapping and object detection with ROS," *Intelligent Vehicles Symposium (IV), 2013 IEEE*. IEEE, 2013, pp. 1083–1088. DOI:10.1109/IVS.2013.6629610.

[12] J. M. Santos, D. Portugal, R. P. Rocha, "An evaluation of 2D SLAM techniques available in Robot Operating System,"*Safety, Security, and Rescue Robotics (SSRR), 2013 IEEE International Symposium on*. IEEE, 2013, pp. 1–6. DOI:10.1109/SSRR.2013.6719348.

[13] "navigation/Tutorials/RobotSetup – ROS Wiki." http://wiki.ros.org/navigation/Tutorials/RobotSetup.

[14] M. Lundgren, "Path Tracking for a Miniature Robot,"*Masters, Department of Computer Science, University of Umea* (2003). p. 9.

[15] "base_local_planner – ROS Wiki." http://wiki.ros.org/base_local_planner?distro=melodic.

[16] "kinetic/Installation –ROS Wiki." http://wiki.ros.org/kinetic/Installation.