Developing a Hybrid-system, ROS algorithm to manage a behavior-based Robot to navigate and avoid static and dynamic obstacles

Sami Salama Hajjaj[1, a], and Keyvan Jafari[2,b]

[1, 2] Centre for Advanced Mechatronics and Robotics (CAMARO),

College of Engineering, UNITEN. Jalan IKRAM-UNITEN, 43000, Selangor, Malaysia

[a] ssalama@uniten.edu.my, [b]k1.jafari71@gmail.com

Keywords: ROS (Robot Operation System), Autonomous Mobile Robots, rviz visualization tools, Gmapping, Simultaneous Localization and Mapping (SLAM), sensor fusion, Pioneer 3-AT

Abstract. One of the new exciting ways to control mobile robots in both indoor and outdoor environments which allow mobile robots to autonomously map their environment, and localize relative to it in real time, while at the same time avoid static and dynamic obstacles is by developing and implementing a hybrid-robotic system that allows a mobile robot to navigate in an unknown environment. Such a design was achieved by utilizing ROS's Rao-Blackwellized Particle Filter (RBPF) algorithm. Other libraries from ROS, such as Gmapping and RVIZ, were also used for localization and visualization.

Introduction

As an important technology in mobile robotics, autonomous navigation enables the robot to avoid collision, reach the goal autonomously and carry out many tasks such as surveillance and exploration. One of the suggested autonomous system *Robot Operation system* (ROS).

ROS is an open-source, meta-operation system for robots [1]. It provides the services expected and received from an operating system. These include *hardware abstraction*, *control of lower level devices*, and others. It also provides tools and libraries for obtaining, building, writing and executing code on multiple computers running on different operating systems and platforms [2,3].ROS implements several types of communication, including communication of synchronous connection, asynchronous streaming of data and storing data on the server parameters [2,3]. ROS contains the stacks, packages, nodes, messages, topics and services necessary for this work. *Stacks* are a collection of packages that provide the cumulative functionality. P*ackages* are the main unit of coding, where actual programming is written. *Nodes* are the connection points between packages. Figure 1 as sample shows the navigation stack used for this work.

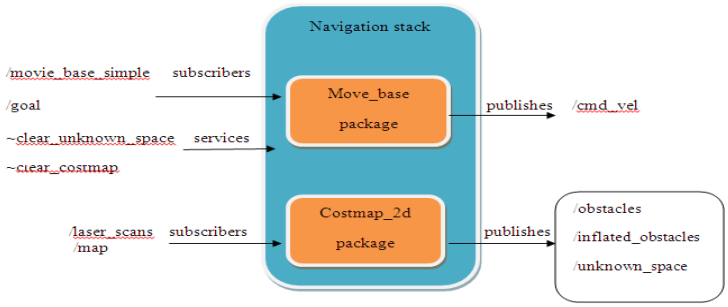


Figure 1: The ROS navigation stack used for this work.

Literature Review

Doriya, R. et al utilized Robot Operating System (ROS) to assist low cost heterogeneous robots in a large environment through the cloud. In their design, they introduced the cloud controller, ROS

master node, storage unit, Map-reduce computing cluster and other robotic services [4] Furthermore, every robot is facilitated with ROS capability to provide abstraction over hardware, as well as communication over TCP/IP. Finally, all the robots communicate with a master node present at the cloud controller to avail communication with other robots and to request services from the cloud [5].

Methodology

ROS version. Among the different versions of ROS, *groovy* was chosen and installed on the computer for the Pioneer3-AT robot, P2OS was chosen, in favor of ROSARIA as deriver. P2OS was chosen because it proved less problematic [6]. P2OS Stack contains the five packages. A summary of these packages and their functionalities are shown in table 1 below.

Table 1: Primary functions of packages in the P2OS stack

Package	Function
P2OS_dashboard	Provides a GUI for debugging and control low-level state, like displaying battery status and breaker states
P2OS_driver	Provides a driver to allow connection to the mobile robot in the form of a ROS node
P2OS_launch	Contains preset launch files and configuration files
P2OS_teleop	Allows the mobile robot to be controlled by a joystick or computer keyboards
P2OS_URDF	Provides Unified Robot Description Format (URDF) which is an XML format representing a 3D robot model

The first *topic* that has been subscribed to is the */cmd_vel* topic which contains messages of velocity commands that controls the robot's motion.

The significant topics published by the P2OS_driver are the *motor_state*, *pose* (combination of position and orientation), and the *tf*. The *tf* topic transforms the */Odom* frame to the */base_link*, as shown in figure 2.

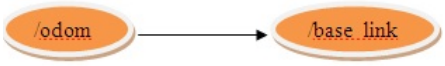


Figure 2: How the *tf* topic transform the frames of P2OS

This transformation is significant as it allows the algorithm to capture all odometry data by the robot and transform it into a very useful set of data that the hybrid algorithm uses to switch between behaviors. Figure 3 shows the details of the stack and its support topics and nodes.

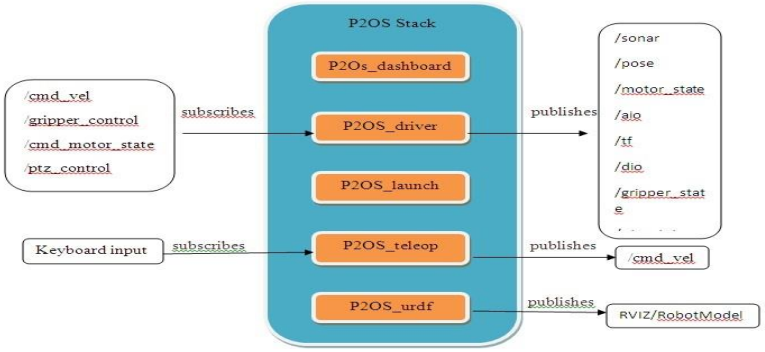


Figure 3: a detailed look at the packages in P2OS stack

The Pioneer 3-AT Robot is the selected mobile robot for this work. It is supported by ROS, as many libraries are written specifically for it also, this robot hosts many dead-reckoning sensors that is useful for the estimation of position and velocity of the robot [7]. Refer to Table 2.

Table 2: Specification of the Pioneer 3-AT used for this work.

Robot model	Pioneer3-AT
Maximum forward/backward speed	0.7 [m/s]
Rotation speed	140[°/s]
Run time	2 to 4 hour with 3 batteries
Batteries	Support up to 3 7.2Ah (each) at a time
Weight	12 [g]

Establishing connection with ROS cloud. After connecting the laptop to the robot by using the USB port the below command should be run in the terminal.

```
$ sudo chmod a+rw /dev/ttyUSB0
```

The *P2OS_driver* node and *P2OS_urdf* nodes are launched in order to control the robot manually using the *P2OS_teleop* node.

```
$ rosrun p2os_driver p2os_driver
$ roslaunch p2os_launch teleop_keyboard.lanuch
$ roslaunch p2os_urdf pioneer3at_urdf.launch
```

The Hokuyo laser range finder: To further enhance the performance of the Pioneer 3-AT, the Hokuyo URG-04LX-UG01 laser range finder was added to it. It is used for scanning the environment and detecting any static or dynamic obstacles in the robot's path. Furthermore, it is used for mapping and localization. all needed to be done after all cable connection is to connect it to ROS through the cloud:

```
$ Rosrun hokuyo_node hokuyo_node
$ rosrun dynamic_reconfigure reconfigure_gui
```

Discussion

The first step in SLAM is building the initial map. The Gmapping package provides a highly efficient Rao-Blackwellized particle filter (RBPF) to create occupancy grid maps data obtained from the laser range finder and the robot's own pose. Obtaining data from different sources and fusing them to do the mapping is performed by the RBPF package. In order to perform mapping, the RBPF mapping packed is called through the cloud from ROS which can be done with different method, either the package is run as it is, or the user can specify odometry noise level.

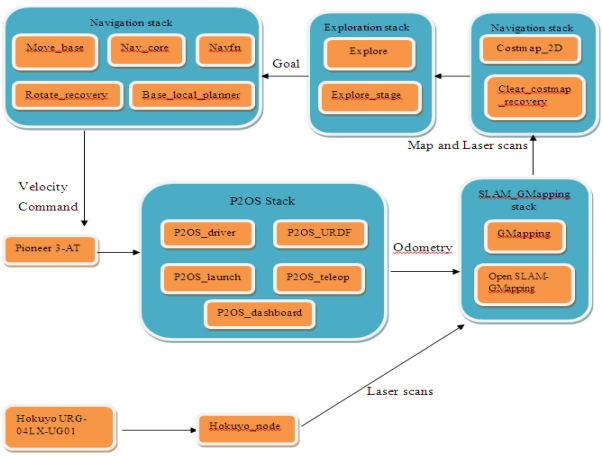
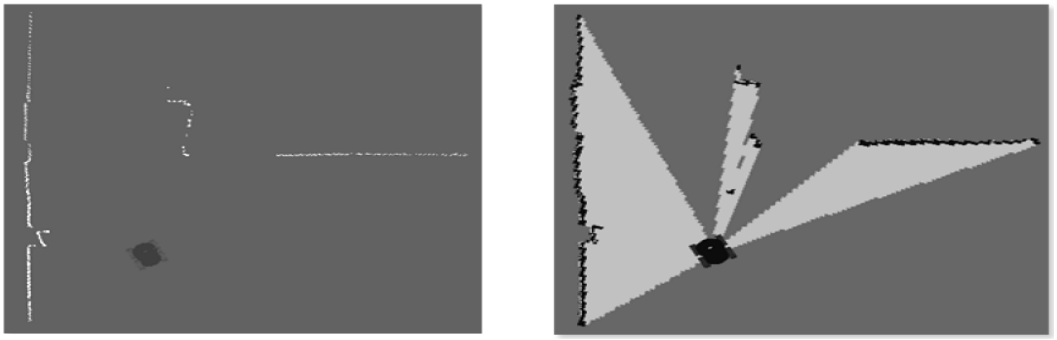


Figure 4: shows all the components of the ROS algorithm used in this work**.**

Results

The first step to visualize the outcome and view the performance of the robot was done by using the RVIZ (ROS Visualization program) package. Figure 5.a shows objects captured by the laser scanner as it detected the environment.



a **b**

Figure 5: (a) The Hyoko laser range finder's captured map 9, and (b) the results

In order to avoid obstacles, data from the laser range finder is incorporated into the *gmapping* node. Figure 5.b shows the outcome of that. The dark area represents the obstacles (static) and the white colour present the free space.

Conclusions

The main goal of this work was to developing a ROS-based hybrid algorithm to control a behaviral based mobile robot in order to navigate through its environment and avoid static or dynamic obstacles which has been achieved. This algorithm is defined and created by controlling ROS through the cloud and downloading the various components needed for hardware abstraction, sensor data processing, sensor fusion, and even visualization and simulation. The algorithm that was used is the *Rao-Blackwellized Particle Filter* (RBPF), which allowed for SLAM to be performed. This algorithm also works in collaboration with data captured from the Hukyuo laser range finder. This algorithm combines the many features of the Kalman Filter, as well as its extended version; the Extended Kalman Filter (EKF).

References

[1] Morgan Quigley, Brian Gerkey, Ken Conley, Tully Foote, Jeremy Leibs. An open-source Robot Operating System, Computer Science Department, Stanford University, Stanford, CA. Willow Garage, Menlo Park, CA. Computer Science Department, University of Southern California. ICRA2009.

[2] Morgan Q. et al. (2012), *ROS: an open-source Robot Operating System*. Retrieved 3 April 2010.

[3] ROSint - Integration of a mobile robot in ROS architecture By André Gonçalves Araújo

[4] Doriya, R. et al. (2012), 'Robot-Cloud': A framework to assist heterogeneous low cost robots, Int. Conf. on Communication, Information and Computing Technology, pp.1,5

[5] Do, H.M. et al. (2013), An open platform telepresence robot with natural human interface, IEEE 3rd Annual Int. Conf. on Cyber Technology in Automation, Control and Intelligent Systems, pp.81,86

[6] H. Wang, Y.Yu, and Q.Yuan, (2011), *Application of dijkstra algorithm in robot path-planning*, 2nd Int. Conf. on Mechanic Automation and Control Engineering (MACE), pp. 1067-1069.

[7] J.-H.Zhou and H.-Y.Lin, (2011), *A self-localization and path planning technique for mobile robot navigation*, 9th World Cong. in Intelligent Control and Automation, pp. 694-699