

## Implementation of Robot Operating System in Beaglebone Black based Mobile Robot for Obstacle Avoidance Application

Mohamad Fauzi Zakaria<sup>#</sup>, Joo Chin Shing<sup>#</sup>, Mohd Razali Md Tomari<sup>#</sup>

<sup>#</sup>Advanced Mechatronics Research Group (AdMiRe) Faculty of Electrical and Electronic Engineering Universiti Tun Hussein Onn Malaysia (UTHM), 86400 Batu Pahat, Johor, Malaysia  
E-mail: mfauzi@uthm.edu.my, franciscs125@gmail.com, mdrazali@uthm.edu.my

**Abstract**— The Robot Operating System (ROS) is a collection of tools, libraries, and conventions that focus on simplifying the task of creating a complex and advanced robotics system. Its standard framework can be shared with another robotics system that has a similar platform and suitable for being introduced as an educational tool in robotics. However, the problems found out in the current robot platform available in the market are expensive and encapsulated. The development of an open source robot platform is encouraged. Therefore, this research is carried out to design and develop an ROS based obstacle avoidance system for existing differential-wheeled mobile robot. The ROS was installed under Ubuntu 14.04 on a Beaglebone Black embedded computer system. Then, the ROS was implemented together with the obstacle avoidance system to establish the communication between program nodes. The mobile robot was then designed and developed to examine the obstacle avoidance application. The debugging process was carried out to check the obstacle avoidance system application based on the communication between nodes. This process is important in examining the message publishing and subscribing from all nodes. The obstacle avoidance mobile robot has been successfully tested where the communication between nodes was running without any problem.

**Keywords**— robot operating system; differential-wheeled mobile robot; beaglebone black; obstacle avoidance

### I. INTRODUCTION

Mobile robotics in education is a pedagogical tool that promoting active learning and involve multi-disciplinary knowledge [1], [2]. Educators in Science, Technology, Engineering and Math (STEM) frequently choose robotics system as a subject STEM-focused problem-based learning (PBL)[3]. Therefore, the ideal features of an educational robot for a group of students are a small physical size for minimum workspace area, low power consumption, high power computational and modular configuration.

Most mobile robotics systems have similar functionalities that can be shared and used in other robots when using the same standard software framework. The current framework widely used is Robot Operating System (ROS). ROS provides standard operating system facilities such as hardware abstraction, low-level device control, implementation of commonly used functionalities, message passing between processes and package management [4], [5]. It is based on graph architecture with a centralized topology where processing takes place in nodes that may receive or post, such as multiplex sensor, actuator, control, state, planning, and navigation [6], [7].

The current implementation of ROS in mobile robotics system is using a laptop computer integrated with an

embedded controller. Their dimension is more than 30 cm x 15 cm, consequently not suitable for academic purpose especially when the testing area is limited [8]–[13]. Therefore, there is a need a small size robot to be designed that uses an embedded Linux single-board computer. This embedded Linux board is low power consumption and open source in hardware and software capabilities which fulfil the ideal features of the educational robot.

The specific objectives of this paper are to implement the ROS on embedded Linux system and to design and develop an obstacle avoidance application for the differential-wheeled mobile robot. This robot would be a teaching tool for a course of Mobile Robotics at the Faculty of Electrical and Electronic Engineering, Universiti Tun Hussein Onn Malaysia. This course covers robot perception, localization, planning and navigation that would be relied on the successful projects done by ROS community.

### II. MATERIAL AND METHOD

The mobile robot is a differential wheeled configuration that developed based on 2WD miniQ robot chassis. The chassis diameter is 12.2 cm, and the wheel diameter is 4.2 cm. Two DC micro gear motors are used for the movement of the robot. An Infrared rangefinder sensor is used to detect the obstacle in front to stop the robot from moving. A servo

motor is attached to the infrared sensor to detect the obstacle in three angles. (45°, 90°, and 135°). The main controller board is a Beaglebone Black Revision C, and its base operating system is Ubuntu ARM 14.04 LTS. Under this operating system, the ROS Jade version is installed.

### A. System Architecture

The overall system architecture of this research is shown in Fig. 1. This system consists of three major applications that are obstacle avoidance system and motor control system. The obstacle avoidance system will continuously receive data from the infrared sensor and servo motor. These data will further be processed by the obstacle avoidance algorithm. After that, the processed information will be transferred to the motor control system to trigger the dc motors to turn to the desired direction. The motor control system is implemented to move the robot by trigger the DC motor in the forward direction or left, and right direction depends on the obstacle detected.

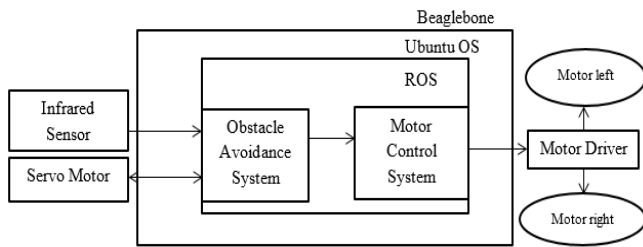


Fig. 1 Block diagram of system architecture

### B. Hardware

Fig. 2 shows the circuit connection of the electronics hardware interface. The infrared sensor that attached to RC Servo Motor is connected to an analog pin, AIN0 of Beaglebone Black. The position control of Servo Motor is based on pulse width modulation (PWM) signal from pin P9-P13. The Beaglebone Black control the speed and direction of two DC motors via an L293D motor driver. The power supply system has relied on a 5VDC power bank battery with maximum 2A current.

### C. ROS Installation on Beaglebone Black

The Beaglebone Black is communicated through Secure Shell (SSH) in a Linux system. The command to access SSH is:

```
$ ssh ubuntu@192.168.7.2
```

After that, the system will prompt the user to enter the username and password. The default username is ubuntu, and the default password is *temppwd*. In order to install ROS on Beaglebone Black, the network access is necessary. The network of main OS can be shared with the OS on Beaglebone Black with the following command:

On the Beaglebone Black:

```
$ sudo ifconfig usb0 192.168.7.2
$ sudo route add default gw 192.168.7.1
```

On the main OS:

#wlan0 is the main OS internet facing interface, eth1 is the Beaglebone USB connection.

```
$ sudo su
```

```
$ ifconfig eth1 192.168.7.1
$ iptables -table nat --append POSTROUTING --out-interface wlan0 -j MASQUERADE
$ iptables --append FORWARD --in-interface eth1 -j ACCEPT
$ echo 1 > /proc/sys/net/ipv4/ip_forward
```

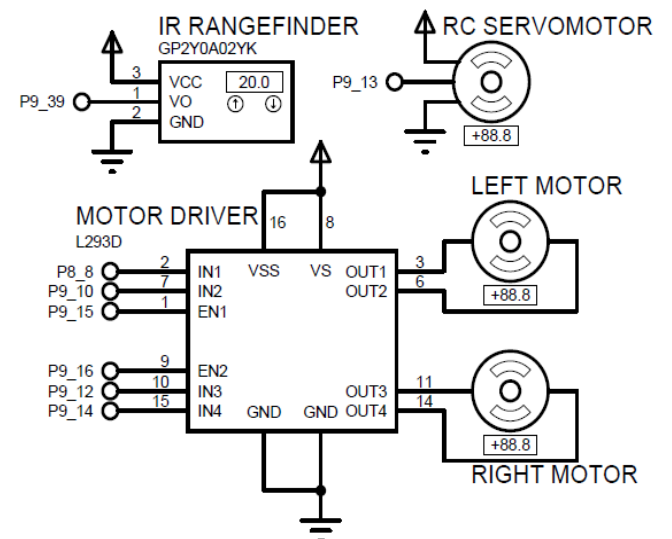
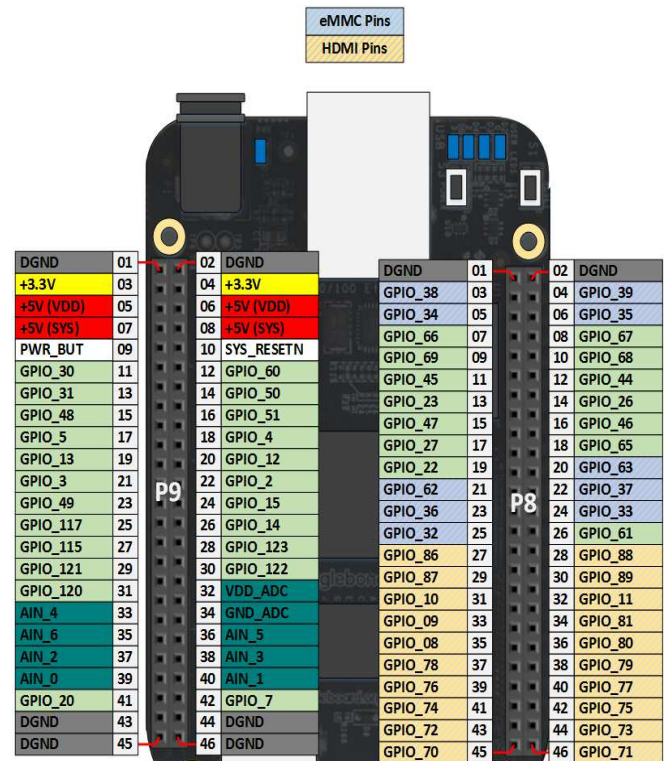


Fig. 2 Schematic diagram of electronics circuit

The following installation guideline is based on [6].  
 Step 1:  
 Before the installation of ROS, the system locale needs to be set with the command:



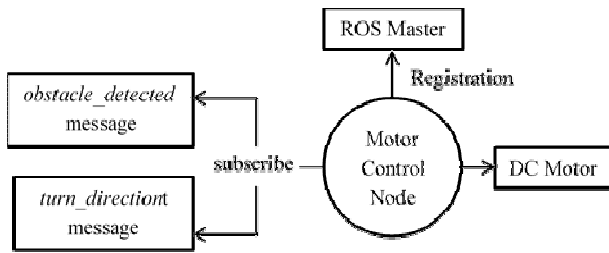


Fig. 4 Motor control system

The sensor node continuously receives data from the infrared sensor and the data received is converted into centimeter to indicate the distance detected in the current moment as shown in Fig. 5. The sensor node is always subscribed to the check message. When the check message is received, the sensor node will run the program to detect the distance obstacle in front. The distance detected is published as a *distance\_detected* message to the servo motor node. The *distance\_detected* message will be subscribed by servo motor node to receive current distance detected by the robot; this data will be further processed by servo motor node.

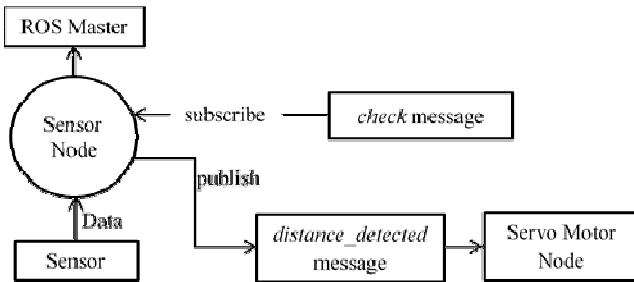


Fig. 5 Infra-red sensor block diagram

In Fig. 6, the servo motor node is responsible for control the motion of servo motor. This node will always subscribe to the *distance\_detected* topic. Due to the limitation of the infrared sensor that is unable to detect object not perpendicular to the sensor, thus the servo motor is used together with the sensor. The sensor will always check the distance in front from the *distance\_detected* topic in  $45^\circ$  to  $135^\circ$  with the use of servo motor; once an obstacle is detected, the sensor will check the distance of left and right. The distance detected in the direction of left and right is then published to compare distance node as a *distance\_right* message and *distance\_left* message. Then compare distance node will further process these two messages to decide which direction to be moved. Moreover, this node will also publish an *obstacle\_detected* message when an obstacle is detected in front.

The compare distance node is constructed to compare the distance at  $0^\circ$  and  $180^\circ$ . This node is continuously subscribed to two messages, *distance\_left* message and *distance\_right* message. Two of this message is sent by servo motor node when an obstacle is detected. The messages received are the distance detected at  $0^\circ$  and  $180^\circ$  which is an Int16 data type. The program of compare distance node will compare the distance received and publish a message called direction to DC motor node. If the distance at  $180^\circ$  is more than the distance at  $0^\circ$ , the

*turn\_right* message is published. Otherwise, a *turn\_left* message is sent. Fig. 7 shows the block diagram for compare distance node.

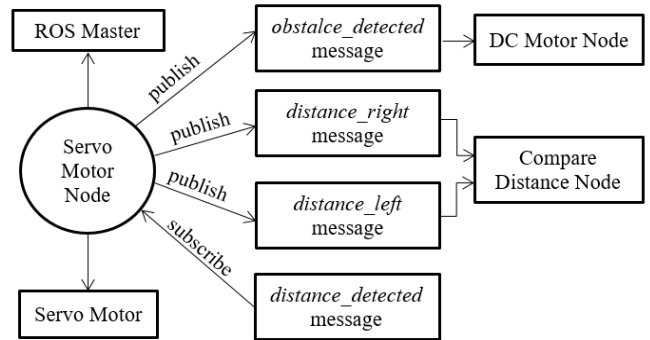


Fig. 6 Servomotor block diagram

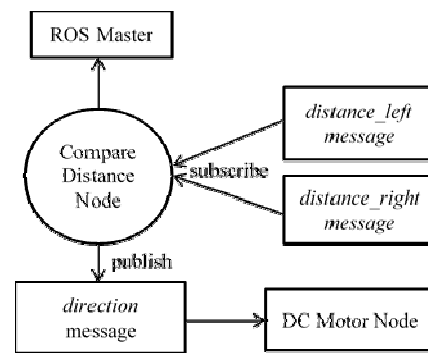


Fig. 7 Compare distance block diagram

### III. RESULT AND DISCUSSION

The implementation of ROS in obstacle avoidance system is debugged based on the communication between nodes. In a ROS system, many nodes are communicating with each other for an application. *roscore* is a collection of nodes and programs that are pre-requisites of a ROS system. To enable communication of all the nodes, *roscore* must be launched. Once the *roscore* is launched, it will start up: a ROS Master; a ROS Parameter Server; and a *rosout* logging node. All the systems above are required to start up the communication between nodes. It can be terminated with keyboard interrupt to shut down the ROS system.

#### A. ROS Graph

The ROS graph provides a visualization graph for the ROS computation graph. It is used to visualize the communication between different nodes [14]. Each of the nodes initializes a specific topic to communicate with the specific node. As a result, the sensor node initialized a topic called *distance\_detect*; this topic was subscribed by servo motor node by connecting to the same topic. This means that once the topic is subscribed by a node, communication can be stabilized by publishing and subscribing to a message.

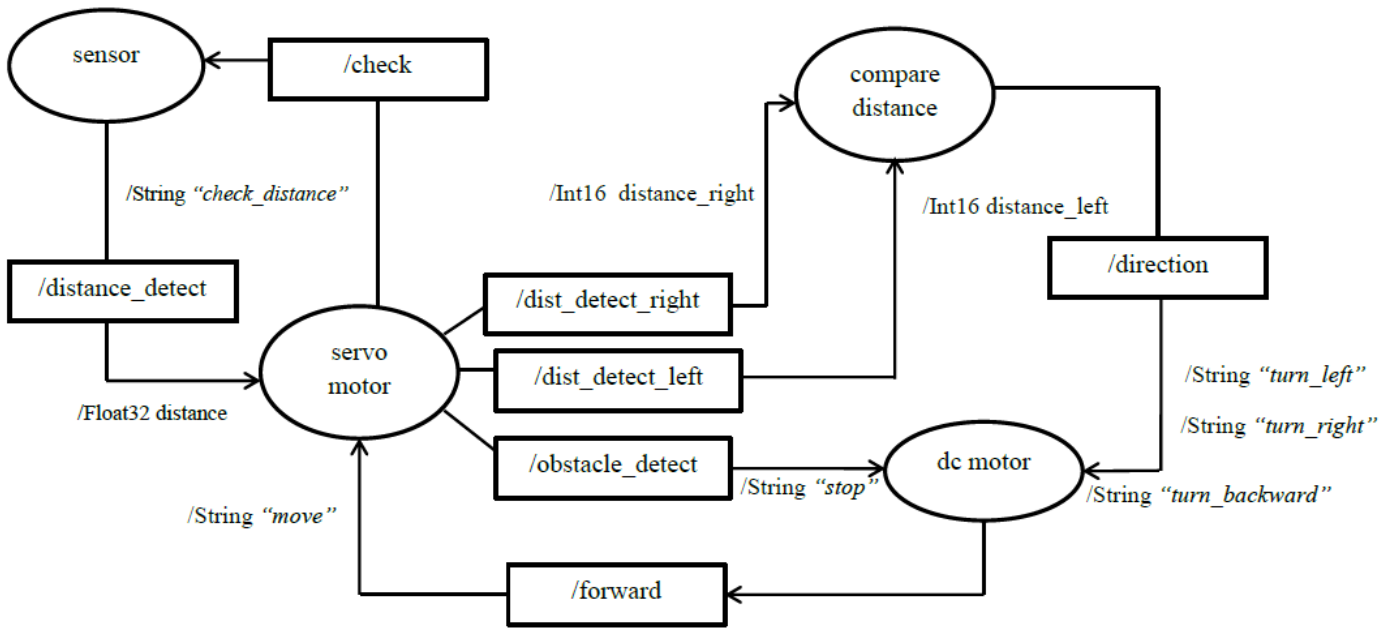


Fig. 8 ROS graph

Fig. 8 shows the ROS graph for communication between four nodes. The topic *distance\_detect* is initialized by sensor node and subscribed by servo motor node to detect the obstacle distance. Besides, the servo motor initializing three topics call *dist\_detect\_right*, *dist\_detect\_left*, and *dist\_detect*. The topics of *dist\_detect\_right* and *dist\_detect\_left* are subscribed by compare distance node to compare the distance of left and right, and then publish a message to DC motor by through *direction* topic. Whereas, the *dist\_detect* topic is subscribed by DC motor node to stop the robot from moving. The DC motor node also initializes a topic called *forward* to publish a *move* message to servo motor node.

### B. Execution of ROS Nodes

In a ROS system, many nodes are communicating with each other for an application. *roscore* is a collection of nodes and programs that are pre-requisites of a ROS system. To enable communication of all the nodes, *roscore* must be launched. Once the *roscore* is launched, it will start up:

- a ROS Master
- a ROS Parameter Server
- a *roscout* logging node

All the systems above are required to start up the communication between nodes. It can be terminated with keyboard interrupt to shut down the ROS system. Before running a *roscore*, the ROS must be source first to enable the access to the ROS commands and build into the workspace:

```

$ cd ~/catkin_workspace
$ source /opt/ros/jade/setup.bash
$ source devel/setup.bash

```

Fig. 9 and 10 show the *source* and *roscore* are run in the terminal.

```

root@arm:/home/ubuntu# cd catkin_ws/
root@arm:/home/ubuntu/catkin_ws# source /opt/ros/jade/setup.bash
root@arm:/home/ubuntu/catkin_ws# source devel/setup.bash

```

Fig. 9 Source of ROS setup bash file

```

root@arm:/home/ubuntu/catkin_ws# roscore
... logging to /root/.ros/log/b3cd4984-d13e-11e5-9226-84eb18e63450/roslaunch-arm-2352.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://arm:48830/
ros_comm version 1.11.16

SUMMARY
-----
PARAMETERS
 * /roscore: jade
 * /rosversion: 1.11.16

NODES

auto-starting new master
process[master]: started with pid [2365]
ROS_MASTER_URI=http://arm:11311/

setting /run_id to b3cd4984-d13e-11e5-9226-84eb18e63450
process[roscout-1]: started with pid [2378]
started core service [/roscout]

```

Fig. 10 Execution of *roscore*

### C. Execution of Application Nodes

After the *roscore* is launched, the application's program can be run by using *roscrun* command. In a ROS system, the program is called as a node. Most of the nodes in ROS system are required to communicate with another node to send and receive a message at that instance. Thus, all the nodes must be run so that all the features of an application can be executed. The command to run the program is:

In terminal 1:

```
$ roscrun ultra_class ultra_class.py
```

In terminal 2:

```
$ roscrun servo_class servo_class.py
```

In terminal 3:

```
$ roscrun compare_distance compare_distance.py
```

In terminal 4:

```
$roscrun dc_motor dc_motor.py
```

The sensor node and servo motor needed to be executed together to establish the communication between these two



thin object. Thus, a webcam is attached to the mobile robot so that the current image can be processed by a vision system.

#### ACKNOWLEDGMENT

The authors would like to thank the Office of Research, Innovation, Commercialization and Consultancy (ORICC), Universiti Tun Hussein Onn Malaysia (UTHM) for the funding of this paper publication.

#### REFERENCES

- [1] S. F. R. Alves, H. F. Filho, R. Pegoraro, M. A. C. Caldeira, J. M. Rosário, and W. M. Yonezawa, "Proposal of educational environments with mobile robots," in *2011 IEEE 5th International Conference on Robotics, Automation and Mechatronics (RAM)*, 2011, pp. 264–269.
- [2] M. Beschi, R. Adamini, A. Marini, and A. Visioli, "Using of the Robotic Operating System for PID control education," *IFAC-Pap.*, vol. 48, no. 29, pp. 87–92, Jan. 2015.
- [3] C. Vandeveld, F. Wyffels, M.-C. Ciocci, B. Vanderborght, and J. Saldien, "Design and evaluation of a DIY construction system for educational robot kits," *Int. J. Technol. Des. Educ.*, vol. 26, no. 4, pp. 521–540, Nov. 2016.
- [4] "About ROS." [Online]. Available: <http://www.ros.org/about-ros/>. [Accessed: 26-Mar-2017].
- [5] A. Koubaa, *Robot Operating System (ROS): The Complete Reference*. Springer, 2016.
- [6] Enrique Fernández, L. Sánchez Crespo, A. Mahtani, and A. Martínez, *Learning ROS for Robotics Programming*. Packt Publishing Ltd, 2015.
- [7] L. Garber, "Robot OS: A New Day for Robot Design," *Computer*, vol. 46, no. 12, pp. 16–20, Dec. 2013.
- [8] E. Ruiz, R. Acuña, N. Certad, A. Terrones, and M. E. Cabrera, "Development of a Control Platform for the Mobile Robot Roomba Using ROS and a Kinect Sensor," in *2013 Latin American Robotics Symposium and Competition*, 2013, pp. 55–60.
- [9] E. M. H. Zahugi, A. M. Shabani, and T. V. Prasad, "Libot: Design of a low cost mobile robot for outdoor swarm robotics," in *2012 IEEE International Conference on Cyber Technology in Automation, Control, and Intelligent Systems (CYBER)*, 2012, pp. 342–347.
- [10] V. Bayar, B. Akar, U. Yayan, H. S. Yavuz, and A. Yazici, "Fuzzy logic based design of classical behaviors for mobile robots in ROS middleware," in *2014 IEEE International Symposium on Innovations in Intelligent Systems and Applications (INISTA) Proceedings*, 2014, pp. 162–169.
- [11] A. Araújo, D. Portugal, M. S. Couceiro, and R. P. Rocha, "Integrating Arduino-Based Educational Mobile Robots in ROS," *J. Intell. Robot. Syst.*, pp. 1–18, Feb. 2014.
- [12] G. Fu and X. Zhang, "ROSBOT: A low-cost autonomous social robot," in *2015 IEEE International Conference on Advanced Intelligent Mechatronics (AIM)*, 2015, pp. 1789–1794.
- [13] H. I. M. A. Omara and K. S. M. Sahari, "Indoor mapping using kinect and ROS," in *2015 International Symposium on Agents, Multi-Agent Systems and Robotics (ISAMSR)*, 2015, pp. 110–116.
- [14] M. Quigley, B. Gerkey, and W. D. Smart, *Programming Robots with ROS: A Practical Introduction to the Robot Operating System*. O'Reilly Media, Inc., 2015.