International Journal of Applied Engineering Research ISSN 0973-4562 Volume 13, Number 7 (2018) pp. 5348-5353

Programming a Mobile Robot to Navigate a Known Area Utilizing Resources from the Cloud

Muhammad Lutfi bin Hamdan, Sami Salama Hussen Hajjaj*
Centre for Advanced Mechantronics and Robotics (CAMaRo)
Universiti Tenaga Nasional, Malaysia.

Abstract

This paper investigates the implementation of Robot Operating System in mobile robot navigation. The developed algorithms and packages enhance the capabilities of the robot during navigation. The robot can navigate autonomously in a known area, reaching its navigational goals by planning its own path, executing its plan, while also avoiding obstacles and collisions. When a user sends a navigation goal at any given point in the map, the mobile robot performs an autonomous navigation process without human intervention. This process requires three components of the robot; mapping, localization, and planning. By executing all three of these component, the mobile robot simply become autonomous. In order to achieve that, the application of ROS was used. As a robust, open-source framework, ROS offers a flexible way for programming a mobile robot. ROS improves the capabilities of robots in term of efficiency, speed, and cost because when utilizing ROS, robots are connected to the cloud and its unlimited resources, significantly reducing computational burden.

Keywords: The Robot Operating System (ROS), Cloud-Robotics, Autonomous Navigation, Mobile Robots.

INTRODUCTION

As we march forward toward the 20th century, technologies around us keep growing at an exponential rate. Navigation has become one of the fast-growing robotics fields. Nowadays, robotics has been assimilated into our daily life. Every single aspect of our life revolves around robotics technology. Humankind is able to create a two-way communication between a robot and a user, making the robot to be able to give responses and feedbacks efficiently. By integrating robots with the cloud, it created something much more advanced. Instead of being limited to their architecture or hardware capability, the robots gain access to the cloud, which contains uncountable resources that can be exploited. By doing so, robots are able to break their limitations and no longer depend solely on their hardware as cloud helps the robots to retrieve information, and supply computational power, storage, and software optimization that greatly increase their abilities.

Or some parts of the navigation field, robots no longer need human intervention to accomplish their task. This act solely depends on their ability to communicate with their user. To have a two-way communication between users and robots that are able to execute a simple or complex task, a bridge that connects the two sides must be established. Typically, programmable robots were used in many complex situations. Capable of executing complex series of actions, this type of robots can be guided or designed to a specific task. This process requires the user to program the robot by means of robot software which contains coded commands or instructions that utilize the mechanical and electronic parts of the robot to perform an autonomous action. There are numerous ways that can be used to program a robot. Each of them offers different method, procedure, and architecture. Powerful hardware is essential for the robot to be able to execute a complex task. This is where the cost limitation begins. For this reason, Robot Operating System, simply known as ROS, offers a perfect solution. ROS enhances the robot capabilities by linking it to the cloud. This helps the robot to reduce dependency on hardware. Despite rapid development within the robotics community, this field of technology still remains out of reach for some. The main reason for this difficulty is simply the cost [1]. For students and researchers to own or develop robots that have complex hardware and software application, it consumes a lot of time and money. Most of the time, the robots created are less effective, big and heavy, expensive, and resources consuming [2]. Another challenge is, for one to design a robot system, absolute knowledge and programming know-how is very important to possess.

OBJECTIVES OF THIS WORK

A. Learn and Master the Linux OS

Learning and mastering Linux and ROS programming language is very important because it is the elementary aspect of this project. To successfully develop and design a program for the robot, extensive learning about Linux and ROS is number one priority in this project.

B. Increase Robot Efficiency and Abilities.

To heighten the efficiency level and capabilities of the robot in completing a given task. With improved abilities at a reduced cost, the robots fully utilize resources from the cloud and navigate with less sophisticated hardware. Navigation and mapping processes are done in a quick fashion as robots gather resources from the cloud in real time. Hence, reducing the time consumed for calculation and mapping processes.

C. Develop algorithms for mobile robot navigation

Developing and designing algorithm needed for an autonomous navigation of the mobile robot by using ROS. ROS provides the required algorithm and programs to control the mobile robots.

D. Implement ROS application to navigate a known area

The primary objective of this project is to program a mobile robot to navigate a known area by utilizing ROS packages such as navigation stack, Simultaneous Localization and Mapping, and simulation. It focuses on executing the navigation algorithms for the mobile robot by using ROS framework and applications: robot localization, mapping, and planning process. Robots share a common fundamental feature that makes them unique which is autonomous navigation. Mapping the environment and planning the path is an essential part of the mobile robot autonomous navigation. To perform this task, mobile robot utilizes a computational algorithm called Simultaneous Localization and Mapping, SLAM. This algorithm will optimize the hardware of the mobile robot to do the mapping and localization process. Applying and implementing SLAM application via ROS platform is the key component of this paper.

LITERATURE REVIEW

A. Mobile Robots and Cloud Robotics

A mobile robot is an automatic machine that is capable of movement in any given environment [3]. It is considered as a branch of robotics concerned with movable robot systems that are able to locomote within an environment or terrain, by navigating itself. Cloud Robotics is the application of the cloud computing concept to robots by means of the Internet or network communication to augment the robots' capabilities by providing services and demand. Cloud computing allows the user to store or access data and programs over the entire Internet rather than limited to the computer's hard drive [4]. This relationship enables robots to share capabilities, development tools, communication, storage and many more within the cloud.

B. How Robot Achieve Autonomous Navigation

An autonomous navigation is defined as the ability for a robot to plan its path and execute its plan without human intervention. Mobile robots with autonomous capabilities allow the robots to perform various tasks in structured or unstructured environments without continuous human intervention [5]. They have the ability to [6]:

1. Gain information about the environment.
2. Work for an extended period without human intervention.
3. Move either all or part of itself throughout its operating environment without human assistance.
4. Avoid dangerous situation such as collision, obstruction, and any harmful acts to human, itself, or the environment.

To navigate autonomously, robots have to deal with three working principles: mapping, localization, and planning.

Mapping is a process of collecting and connecting information from sensors into a map. Robots execute mapping algorithm to translate the surrounding environment into data that can be read, usually a 2D geometric representation.

Localization is where the robots use the sensors to estimate their position relative to the map. Robots try to locate their location within the map by means of the sensors by utilizing their sensors and laser scanners to receive geometric data and odometry data to move around the environment, determining their position relative to the map at the same time.

Once localization is established, the robot can start planning its path to achieve goal location. The map generated functions as an eye for the robot. It represents entities such as known space, obstacle, robot position, and unexplored space. This process is known as a global planner. Then, a local planner translates this path into movement of the robot.

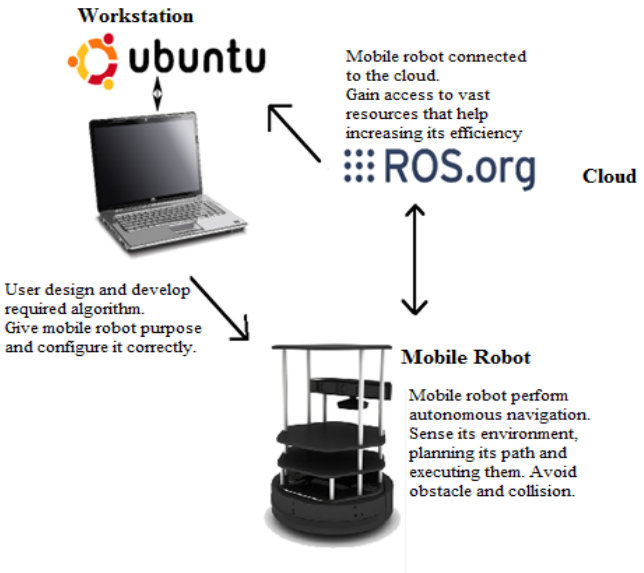

Figure 1: Components of Cloud-Robotic platform in this paper

C. Robot Operating System

Robot Operating System or simply called ROS is an open source, meta-operating system for robot [7]. Established in 2007 by Stanford Intelligence Laboratory of the Stanford AI Robot STAIR project, it was then introduced to the robotics community by Willow Garage in 2008 [8].

The main ROS client libraries consist of C++ and Python programming language. Functioned as a services provider, ROS do a lot of tasks including hardware abstraction, low-level device control, implementation of commonly-used functionality, and message passing between processes by

providing tools and libraries for obtaining, building, writing, and running code across multiple computers. In another way, ROS can be defined as a programming language for robots or robotics software platform.

ROS is essential in this research so that it can heavily reduce the workload for programming every single script and program for the mobile robot. The user can easily access the collection of tools, libraries, and packages inside ROS.

Figure **2**: The Turtlebot Robot Platform

D. The Turtlebot robot platform

The Turtlebot robot platform is an example of a mobile robot. It is a low-cost, personal robot kit with open source software using limited hardware and highly capable autonomous platform [9]. By utilizing an advanced autonomous navigation using Simultaneous Localization and Mapping (SLAM), the TurtleBot can be integrated into a mobile perception and navigation robot. Created at Willow Garage by Melonee Wise and Tully Foote in November 2010, it consists of a mobile base, a dual-core processor Asus 1215N laptop, the mounting hardware kit, and equipped with a 3D sensor, Microsoft Kinect [10].

METHODOLOGY

A mobile robot can be programmed to navigate any given environment. Numerous sources are available in the cloud that can be used for navigation purposes. In order to do that, an extensive and demanding effort must be considered as this task is very complex for researchers and engineers.

A. Learn ROS and setting up ROS Workspace

The autonomous navigation requires development from several parts. The beginning stage is started with a lot of learning and understanding the concept of ROS. This is the essential part of all. The basics were vital for creating a proper ROS workspace in order to implement the application for autonomous navigation. It covered a lot of practice and preparation to develop skills required to program, troubleshoot codes, understand the basics of programming language, handle ROS dependencies, navigate ROS structure, to create and build ROS package, and utilize ROS libraries.

B. Utilizing SLAM

Comprised of hundreds of useful applications in its libraries, ROS packages offer valuable usage for users. In relation to navigation, several packages can be implemented. The navigation package consists of several packages, namely navigation stack, SLAM, and simulator.

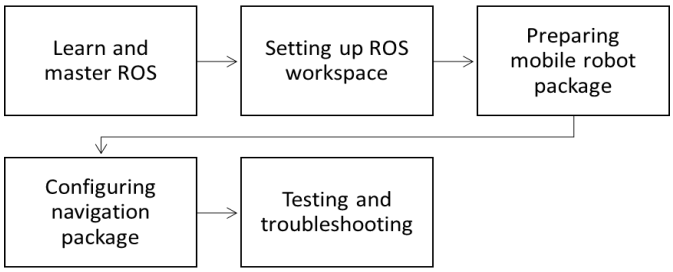

Figure **3**: The Process Workflow

SLAM offers a solution for mobile robot mapping and localization processes. SLAM allows the robot to perform both the processes at the same time by applying its algorithms. SLAM is applicable for both 3D and 2D motion and consists of multiple parts such as landmark extraction, data association, state estimation, and landmark update.

The gmapping package contains a gmapping node that exploits the robot sensors and laser scanners to create a 2D occupancy grid map from an unknown area. To create a map, it used a 2D occupancy grid method. By using the sensor particle, it scanned the surrounding area and obstacles on the map by scan matching and comparing scans from current to previous data. The gmapping node, slam_gmapping, subscribes to /tf topic and then creates and publishes the map to /map topic. It requires a transform from the sensor to base_link where it broadcast it from /map to /odom frames. Another node known as Adaptive Monte Carlo Localization shares the same working principle with its counterpart but it is used in a known area for localization purpose.

SLAM algorithms involve a number of parameters that dictate its functionality. These parameters were configured and adjusted until fine-tuned results were obtained. It is very important because the parameters setting directly affects the end results. Failure to do so may cause problems.

```
<launch>
  <arg name="scan_topic" default="scan" />

  <node pkg="gmapping" type="slam_gmapping" name="slam_gmapping" output="screen">
    <param name="base_frame" value="base_footprint"/>
    <param name="odom_frame" value="odom"/>
    <param name="map_update_interval" value="1"/>
    <param name="maxUrange" value="16.0"/>
    <param name="minimumScore" value="10000"/>
    <param name="maxRange" value="18.0"/>
    <param name="sigma" value="0.05"/>
    <param name="kernelSize" value="1"/>
    <param name="lstep" value="0.05"/>
    <param name="astep" value="0.05"/>
    <param name="iterations" value="5"/>
    <param name="lsigma" value="0.075"/>
    <param name="ogain" value="3.0"/>
    <param name="lskip" value="0"/>
    <param name="srr" value="0.01"/>
    <param name="srt" value="0.02"/>
    <param name="str" value="0.01"/>
    <param name="stt" value="0.02"/>
    <param name="linearUpdate" value="0.05"/>
    <param name="angularUpdate" value="0.1"/>
    <param name="temporalUpdate" value="-1.0"/>
    <param name="resampleThreshold" value="0.5"/>
    <param name="particles" value="25"/>

    <param name="xmin" value="-100.0"/>
    <param name="ymin" value="-100.0"/>
    <param name="xmax" value="100.0"/>
    <param name="ymax" value="100.0"/>

    <param name="delta" value="0.05"/>
    <param name="llsamplerange" value="0.01"/>
    <param name="llsamplestep" value="0.01"/>
    <param name="lasamplerange" value="0.005"/>
    <param name="lasamplestep" value="0.005"/>
    <remap from="scan" to="$(arg scan_topic)"/>
  </node>
</launch>
```

Figure 4: Gmapping parameters

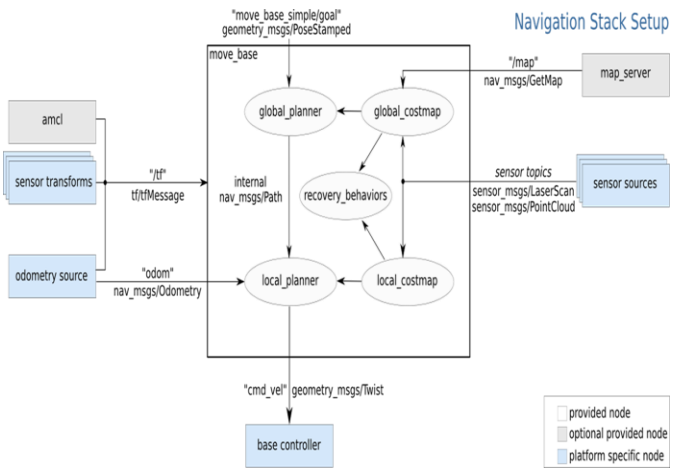

Figure 5: Components of navigation stack

C. *Configuring Navigation Stack*

The ROS Navigation Stack is a collection of packages that consist of numerous algorithms that use the sensors of the robot and the odometry to perform navigation process.

Transform configuration or simply called *tf* supplies the required information about the relationship between coordinates and frames of robots. tf also publishes odometry information. Sensors information provides data from scanners equipped with robots, i.e. laser scanner, which is useful for obstacle avoidance. Map_server allows the robot to subscribe to a map if it is already given.

It also enables robots to save a map acquired from the mapping process and can be called anytime. Finally, the base controller is needed for robot movement. It allows the user to control the behaviour of robots during navigation such as velocity and obstacle avoidance and translate into motor commands for the mobile base.

The *move_base* package consists of costmap, planner, and recovery behaviour. The costmap is a set of binary data that represents places which are safe for the robot, free spaces or places where a collision would happen.

Meanwhile, the global planner and local planner are responsible for computing a plan for the robot before it starts moving toward next destination by monitoring incoming sensor data and choosing the appropriate control of linear and angular movement.

In order to link all the nodes involved in navigation stack, move_base utilFor some parts of navigation field, robots no longer needs human intervention to accomplish its task. This act solely depend on its ability to communicate with its user.

To have a two way communication between uizes a configuration file in which YAML file is called. This configuration file consists of several parameters need to be configured.

```
<!--
    ROS navigation stack with velocity smoother and safety (reactive) controller
-->
<launch>
  <include file="$(find turtlebot_sim_2dnav)/launch/includes/velocity_smoother.launch.xml"/>
  <include file="$(find turtlebot_sim_2dnav)/launch/includes/safety_controller.launch.xml"/>

  <arg name="odom_topic" default="odom" />

  <node pkg="move_base" type="move_base" respawn="false" name="move_base" output="screen">
    <rosparam file="$(find turtlebot_sim_2dnav)/param/costmap_common_params.yaml" command="load" ns="global_costmap" />
    <rosparam file="$(find turtlebot_sim_2dnav)/param/costmap_common_params.yaml" command="load" ns="local_costmap" />
    <rosparam file="$(find turtlebot_sim_2dnav)/param/local_costmap_params.yaml" command="load" />
    <rosparam file="$(find turtlebot_sim_2dnav)/param/global_costmap_params.yaml" command="load" />
    <rosparam file="$(find turtlebot_sim_2dnav)/param/base_local_planner_params.yaml" command="load" />
    <rosparam file="$(find turtlebot_sim_2dnav)/param/move_base_params.yaml" command="load" />

    <remap from="cmd_vel" to="navigation_velocity_smoother/raw_cmd_vel"/>
    <remap from="odom" to="$(arg odom_topic)"/>
  </node>
</launch>
```

Figure **6**: The move_base launch file

D. *Turtlebot Mobile Robot Package*

The Turtlebot package was installed so that ROS can use its applications. It contains all applications of the robot such as motor movement, Kinect driver, speed controller, and teleoperation keys.

Aside from the basic drivers, the Turtlebot package also includes a simulator application, Gazebo. Gazebo simulator is a simulation package that can simulate the Turtlebot action or environment. Without depending on the actual robot, programs can be done on simulation instead and every application used in the simulation is transferrable to physical robots without major changes.

The development of the algorithms, troubleshooting and diagnostics were done in simulation. This was so that the dependency on the real robot can be minimized. To be able to virtually simulate the robot navigation, the environment for the robot must be created. Gazebo simulator, which is already pre-installed together with the Turtlebot package was specifically used for this task. The saved world can then be launched later while performing navigation in the simulation.

Rviz is used to visualize the process. As a 3D visualization tool, Rviz portrayed the data during the navigation process such as cameras, 3D laser, Kinect sensor and sending goal

location to the robot. In other words, the user can see how the mapping process took place, and visualize sensors data. From the Rviz interface, the user can see and control the robot movement.

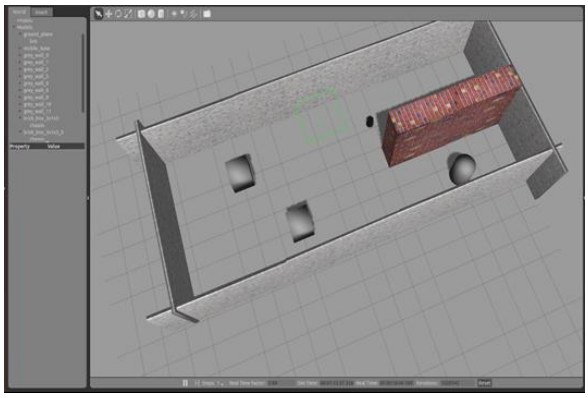

Figure 7: Turtlebot in Gazebo simulator. This is a created world for Turtlebot

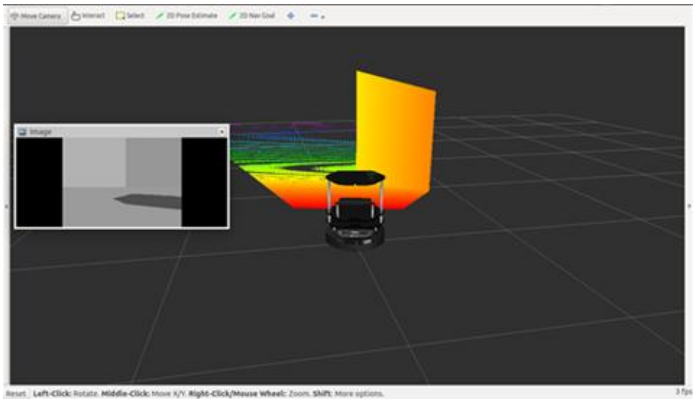

Figure 8: Rviz with its sensor and camera turned on

E. Testing and Troubleshooting

With all the components had been configured and constructed, a series of tests were carried out to troubleshoot any problems. This process was done several times to perfectly ensure that the codes written and built were free from errors. It also intended to optimize the parameters and algorithms.

FINDINGS AND DISCUSSION

The mapping process is where every node, package, and code that have been configured were tested. First, the created world earlier was called upon. Then, Rviz was launched. Finally, the mapping package which contains gmapping launch file and move_base launch file. Teleoperation also can be launched to control the Turtlebot but the move_base already contains planner which enables the user to give goals to the Turtlebot via Rviz.

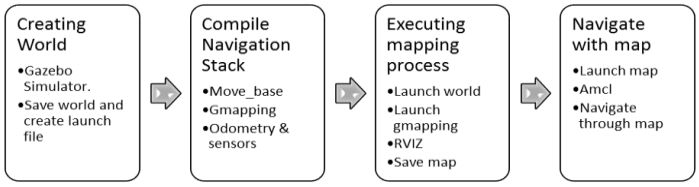

Figure 9: Autonomous navigation process

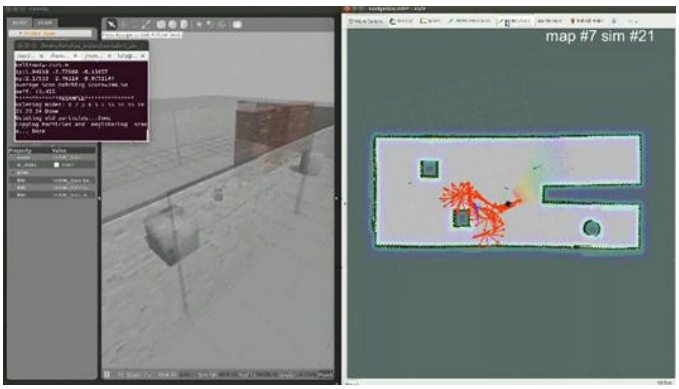

Figure 10: Finalize mapping process

The map generated from the process was saved. The Turtlebot now already knows the area via the mapping process done earlier. Gazebo was launched to simulate the environment. Then, rviz was launched to visualize the process and then, the map saved earlier was called upon. Navigation goals were sent to the Turtlebot via the rviz 2d Nav Goal button. The Turtlebot moved to the goal location, selected its path, and avoided obstacle independently.

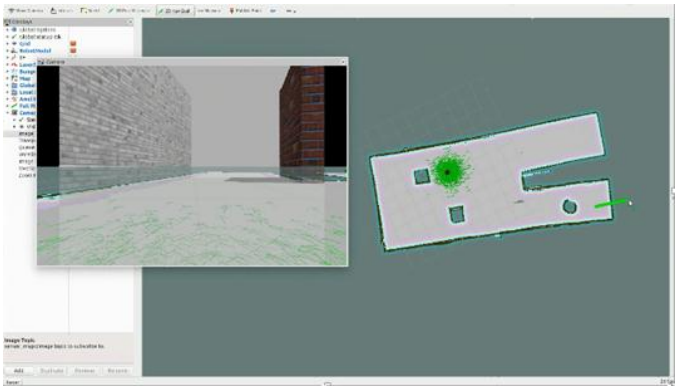

Figure **11**: Turtlebot navigate autonomously

CONCLUSION

In conclusion, the Turtlebot was programmed successfully to navigate a known area. It was able to perform autonomous navigation by the implementation of ROS packages and algorithms. The Turtlebot avoided obstacle and collision, planned its path and executed them without human intervention.

The developed algorithms and packages manage to apply ROS concept on the Turtlebot and all the parameters have been optimized. Although the simulation process consumes a lot of time due to its need for high graphic and processing power, the end results were enough to justify the success.

Hence, it is proven that cloud robotics enhance the capabilities of the robot by supplying information, reducing computational burdens, and providing massive resources.

REFERENCES

[1] S. H. Hajjaj, 2014, Cloud Robotics: Programming a Mobile Robot to Navigate Autonomously Through a Combined Indoor/Outdoor Path by Utilizing Software, Navigation, and Mapping Resources of the Cloud, Universiti Tenaga Nasional

[2] S. H. Hajjaj, 2014, Cloud Robotics: Programming a Mobile Robot to Navigate Autonomously Through a Combined Indoor/Outdoor Path by Utilizing Software, Navigation, and Mapping Resources of the Cloud, Universiti Tenaga Nasional.

[3] Meghan Meckstroth, 2009, "Mobile Robotics: Moving Robots Forward", National Instrument

[4] Guoqiang Huarticle._IoTgate.tex, WP Tay, Y Wen 2012, "Cloud Robotic: Architecture, Challenges, and application", IEEE Network

[5] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," *IEEE Translated J. Magn. Japan*, vol. 2, pp. 740-741, August 1987 [*Digest 9th Annual Conf. Magnetics Japan*, p. 301, 1982].

[6] Chatterjee, A., Rakshit, A., & Singh, N. N. (2013). Vision based autonomous robot navigation: algorithms and implementations

[7] D Wonnacott, M Karhumaa, J Walker, 2012, Autonomous Navigation Planning With ROS, Michigan Technology University

[8] Buniyamin N, Wan Ngah WAJ, Sariff N, Mohamad Z, 2011, A Simple Local Path Planning Algorithm for Autonomous Mobile Robots, International Journal of System Application, Engineering and Development

[9] Open Source Robotics Foundation, retrievied 12 December 2016, from wiki.ros.org/Robots/navigation

[10] Arbnor, P., Petrit, A., 2014. SLAM – Map Building and Navigation via ROS. Intelligent Systems and Application in Engineering. 2(4): 71-74