International Journal of
Intelligent Systems and Applications in Engineering

ISSN:2147-6799 www.ijisae.org Original Research Paper

SLAM – Map Building and Navigation via ROS#

Arbnor Pajaziti*[1], Petrit Avdullahu[2]

Received 05th September 2014, Accepted 18th December 2014

Abstract: The presented work describes a ROS based control system of a Turtlebot robot for mapping and navigation in indoor environments. It presents the navigation of Turtlebot in self-created environment. The mapping process is done by using the GMapping algorithm, which is an open source algorithm and the localization process is done by using the AMCL pack. There are ROS built functions used in order to perform navigation of Turtlebot. The SLAM method implemented in ROS has proven a way for robots to do localization and mapping autonomously. The aim defined in paper to fulfill mapping, localization and navigation of Turtlebot in new and unknown environment is achieved

Keywords: *Map Building, Navigation, ROS, Simulation, Turtlebot.*

1. Introduction

Human beings get the information about their surrounding through vision, hearing the voice through ears, smelling through nose and they do feel the strength of objects through touching. In general human beings get information about reality through senses. A robot cannot explore an unknown environment unless it is provided with some sensing sources to get information about the environment. There are different kinds of sensors used to make a robot capable of sensing a wide range of environments [1]: Odometers, Laser range finders, Global Position System (GPS), Inertia Measurement Units (IMU), Sound Navigation and Ranging (Sonar) and cameras.

The map of environment is a basic need for a robot to perform actions like moving room to room, picking an object from one place and taking it to another one. To perform such actions, the robot should not only know about the environment but while it is moving it should also be aware of its own location in that environment.

The robot simulated is Turtlebot, and in absence of real robot the simulator provided by ROS is used. The aim to achieve is the navigation of the robot in new and unknown environment by using the ROS. As it will be presented, the robot builds the map, localizes itself on the map and performs navigation.

Turtlebot as ROS compatible robot to demonstrate the ROS power has been selected. Robot operating system (ROS) is designed to promote code sharing and enable the development of open-source robotics commons. Sharing code will help the robotics community to progress faster by letting the researchers in the community replicate and extend the results of other research groups. ROS makes it easy to find the software and integrate it into robot systems [2-5]. The complete TurtleBot includes a Kobuki base, Microsoft XBOX Kinect, ROS compatible net book, and turtlebot structure. General description on turtlebot website is "TurtleBot is a low-cost, personal robot kit with open-source software. With TurtleBot, you'll be able to build a robot that can drive around your house, see in 3D, and have enough horsepower to create exciting applications". There are numerous abilities that Turtlebot offers, while there are given part list, assembly instructions and links to suppliers. There is also a possibility to add other components, or to substitute similar components.

1.1. Outline

Section 2 starts with an overview of robotics definition and with an introduction to ROS. There is described a history of ROS followed by ROS compatible robots and ROS concepts. Section 2 is a description of ROS in general. Section 3 describes the SLAM concept and as a solution to SLAM problem there is a review of an algorithm called SLAM with extended Kalman filters. Section 4 starts with an overview of Turtlebot simulation on ROS, and at the end of section 4 a step-by-step simple solution to navigate the Turtlebot is brought.

The main reference for section 3 is the book on [6], and the main reference for section 2 and section 4 is the ROS website referenced on [7].

2. Introduction to ROS

ROS is meta-operating system for a robot. It provides services that one would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers. It is named as meta-operating system because it is something between an operating system and middleware. It provides not only standard operating system services (like hardware abstraction), but also high-level functionalities like asynchronous and synchronous calls, centralized database, a robot configuration system, etc. ROS can be interpreted also as software framework for robot software development, providing operating system [7].

2.1. The general organization of ROS

ROS is a thin, message-based, tool-based system designed for mobile manipulators. The system is composed of reusable libraries

[1] *IDEA Teknoloji Çözümleri, Sun plaza BBDO Blok Dereboyu Cd. Bilim Sk. No:5, 34398, Maslak /İstanbul / Turkey*
** Corresponding Author:email:unsal.gokdag@gmail.com*
 # This paper has been presented at the International Conference on Advanced Technology&Sciences (ICAT'14) held in Antalya (Turkey), August 12-15, 2014.

that are designed to work independently. The libraries are wrapped with a thin message-passing layer that enables them to be used by and make use of other ROS nodes. Messages are passed peer to peer and are not based on a specific programming language; nodes can be written in C++, Python, C, LISP, Octave, or any other language for which someone has written a ROS wrapper. ROS is based on a Unix-like philosophy of building many small tools that are designed to work together (more on that in a bit). ROS grows out of collaboration between industry and academia and is a novel blend of professional software development practices and the latest research results [8].

2.2. ROS-compatible robots

There is quite a big list of compatible robots. A few of them are:
1. Aldebaran Nao
2. Willow Garage PR2
3. TurtleBot
4. AscTec Quadrotor
5. Lego NXT
6. Stanford Racing's Junior

The list of ROS-compatible robots grows constantly.

Turtlebot - TurtleBot combines popular off-the-shelf robot components like the iRobot Create, Yujin Robot's Kobuki and Microsoft's Kinect into an integrated development platform for ROS applications. TurtleBot is a low-cost, personal robot kit with open-source software.

Figure 1. Turtlebot [7].

2.3. Programming with ROS

ROS is language-independent. At this time, three main libraries have been defined for ROS, making it possible to program ROS in C++, Python or Lisp. In addition to these three libraries, two experimental libraries are offered, making it possible to program ROS in Java or Lua.

There is a project which is still under development named Rosjava. Rosjava is a pure Java implementation of ROS created and maintained by Google and Willow Garage. Under Rosjava, the Rosjava project totally rewrote the ROS core in Java. Google's objective is to have a version of ROS that is fully compatible with Android. Rosjava can be used to control robots for which the operating system is not Linux, but Android.

ROS software is organized into packages. Before writing any programs, the first step is to create a workspace that holds the packages, and then create the package itself. The groovy distribution of ROS contains some major changes to the way software is compiled. Older versions used a built system called rosbuild, but groovy has begun to replace rosbuild with a new build system called catkin. This is important to mention. An example of creating a package is:

catkin_create_pkg example1,

which will create package named example1. Created packages should live in a workspace directory.

3. SLAM – Map Building and Navigation

SLAM is concerned with the problem of building a map of an unknown environment by a mobile robot while at the same time navigating the environment using the map. The term SLAM is an acronym for Simultaneous Localization and Mapping. It was originally developed by Hugh Durrant-Whyte and John J. Leonard. SLAM consists of multiple parts; Landmark extraction, data association, state estimation, state update and landmark update [9]. SLAM is more like a concept than a single algorithm. There are many steps involved in SLAM and these different steps can be implemented using a number of different algorithms. SLAM is applicable for both 2D and 3D motion [9].

3.1. The hardware

The hardware of the robot is quite important. To do SLAM there is a need for a mobile robot and a range measurement device. The mobile robots considered are wheeled indoor robots. Some basic measurement devices commonly used for SLAM on mobile robots are presented here.

Important parameters to consider are ease of use, odometry performance and price. The odometry performance measures how well the robot can estimate its own position, just from the rotation of the wheels. The robot should not have an error of more than 2 cm per meter moved and 2° per 45° degrees turned. Typical robot drivers allow the robot to report its (x, y) position in some Cartesian coordinate system and also to report the robots current bearing/heading [9].

3.2. The description of SLAM

The SLAM problem is defined as a concurrent localization and mapping problem, in which a robot seeks to acquire a map of the environment while simultaneously seeking to localize itself relative to this map.

From a probabilistic perspective, there are two main forms of the SLAM problem, which are both of equal practical importance [6]:

1. Online SLAM problem:

$$p(x_t, m \mid z_t, u_t) \tag{1}$$

2.Full SLAM problem:

$$p(x_{1:t}, m \mid z_{1:t}, u_{1:t}) \tag{2}$$

where,

X_t is the pose at time t,
m is the map,
Z_t is the measurements, and
U_t is the controls.

The graphical model of online SLAM problem is shown in figure 2. The goal of online SLAM is to estimate a posterior over the current robot pose along with the map.

3.3. SLAM with Extended Kalman Filters

The earliest SLAM algorithm is based on the extended Kalman filter. In a nutshell, the extended Kalman filter SLAM algorithm applies the EKF (Extended Kalman Filter) to online SLAM using maximum likelihood data association. In EKF maps are feature-based; they are composed of point landmarks. For computational reason, the number of landmarks is usually smaller than 1000. The extended Kalman filter tends to work well the less ambiguous the landmarks are. For this reason EKF SLAM requires significant engineering of feature detectors [2].
The EKF SLAM algorithm can only process positive sightings of landmarks. It cannot process negative information that arises from the absence of landmarks in sensor measurements.

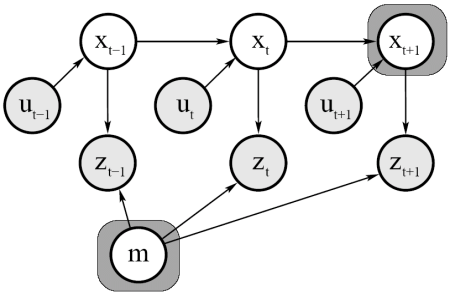

Figure 2. Graphical model of the online SLAM problem [6].

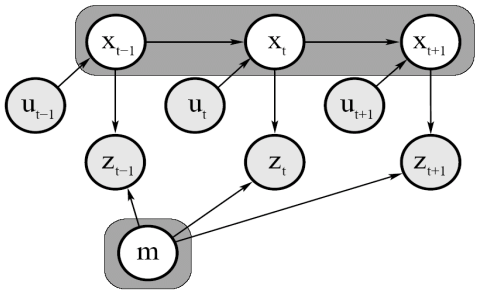

Figure 3. Graphical model of the full SLAM problem [6].

The SLAM algorithm for the case with known correspondence addresses the continuous portion of the SLAM problem only. In addition to estimating the robot pose, the EKF algorithm also estimates the coordinates of all landmarks encountered along the way. The combined state vector is given by

$$y_t = \begin{pmatrix} x_t \\ m \end{pmatrix} = (x \quad y \quad \theta \quad m_{1,x} \quad m_{1,y} \quad S_1 \quad \ldots \quad m_{N,x} \quad m_{N,y} \quad S_N)^T \quad (3)$$

where x, y, and θ denotes the robots coordinates at time t
$m_{1,x}$, $m_{1,y}$ are the coordinates of the i-th landmark, for i=1… N, and
S_i is its signature.
The dimension of this state vector is 3N+3, where N denotes the number of landmarks in the map.
Figure 4 illustrates that the EKF SLAM algorithm for an artificial example.

4. Results and Discussion

Turtlebot in Gazebo with a simulated laser build a map of an environment with the GMapping algorithm has been setup. It was assumed that the workspace environment has been created. When the workspace is created, new packages need to be put in a path that is in the variable $ROS_PACKAGE_PATH. Echo this variable and confirm if the package path has been set.

Instructions for ROS Fuerte have been used. All commands used in this paper are stated on documentation status for ROS Fuerte. As mentioned earlier, there is a difference between ROS versions.

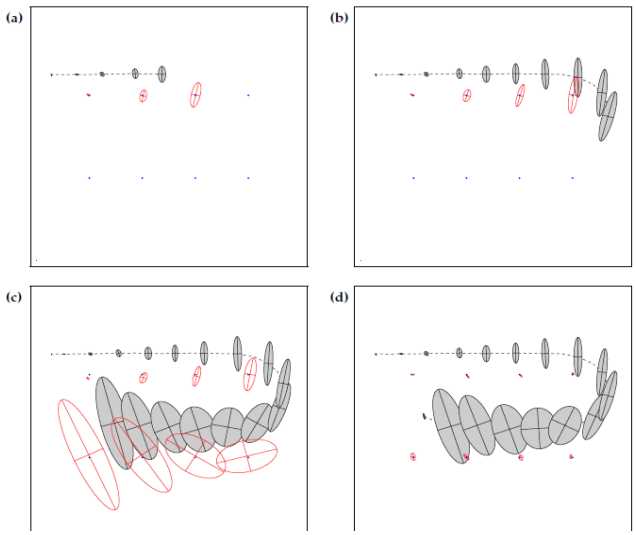

Figure 4. EKF applied to the online SLAM problem [6].

ROS package called ‘ros’ that depends on these packages has been created:

- ***turtlebot_gazebo*** – this package contains launchers, maps and world descriptions for the TurtleBot simulation with Gazebo.
- ***turtlebot_teleop*** – this package provides launch files for teleoperation with different input devices.
- ***gmapping*** – this package provides laser-based SLAM, as a ROS node called slam_gmapping.
- ***amcl*** – this package provides probabilistic localization system for a robot moving on 2D. It implements the adaptive Monte Carlo localization approach, which uses a particle filter to track the pose of a robot against a known map.
- ***move_base*** – this package provides an implementation of an action.

The roscreate-pkg command-line tool creates a new ROS package.

```
# roscreate-pkg ros turtlebot_gazebo turtlebot_teleop gmapping amcl move_base
```

By executing this command, these files will be created:

- manifest.xml,
- CMakeLists.txt,
- mainpage.dox, and
- Makefile.

In order to simulate a turtlebot in an appointed environment a launch file has to be created. Simple office space environment where the turtlebot can move around and navigate are to be added. Launch file ‘*turtlebot_office.launch*’ can be created within the folder *launch*.

```
# mkdir launch
# gedit launch/turtlebot_office.launch
```

While gedit command-line is used to create or edit files, in our case it is used to create turtlebot_office.launch file, and write the following code into the created file:

```
<launch>
<param name=”/user_sim_time” value=”true” />
<node name="gazebo" pkg="gazebo" type="gazebo" args="-u
$(find gazebo_worlds)/worlds/simple_office.world" />
<node name="gazebo_gui" pkg="gazebo" type="gui" />
<include file="$(find turtlebot_gazebo)/launch/robot.launch" />
<include file="$(find turtlebot_teleop)/keyboard_teleop.launch"
/>
</launch>
```

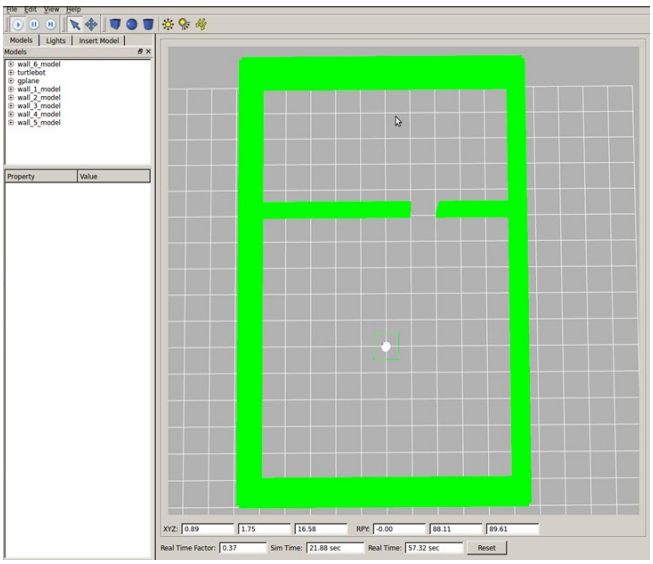

Figure 5. The simulated world with turtlebot in gazebo.

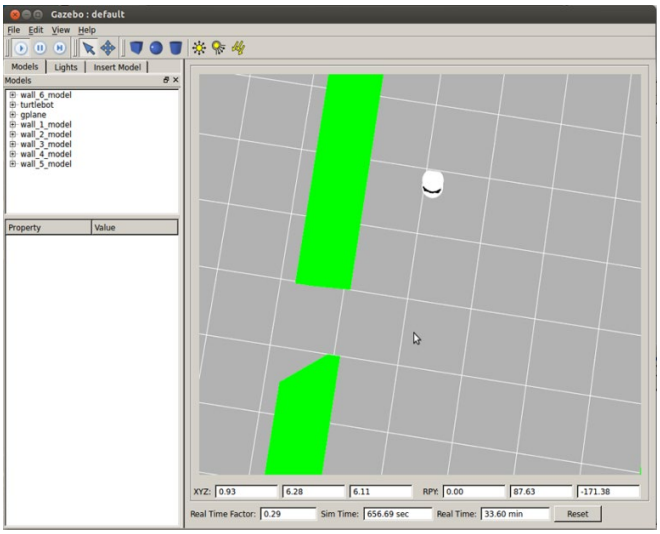

Figure 6. Turtlebot while operating with joystick or keyboard.

This file is used by roslaunch command.
The roslaunch package contains the roslaunch tools which read the xml format files. The standard format of the roslaunch command is

roslaunch *package-name launch-file-name.*

After executing the roslaunch command the simulated robot and environment appears.
Also there is the turtlebot teleoperating pack (turtlebot_teleop) included, which means that the turtlebot will move, if joystick or keyboard is used.
The navigation on given map is depicted in figure 7. First we have to be sure that Gazebo, map server, amcl and rviz are running [10]. By executing this command in new terminal the turtlebot will be ready to perform navigation on the given map.

```
# roslaunch ros move_base_turtlebot.launch
```

From Rviz tool the navigation goal can be set, by clicking the ‘2D Nav Goal’ button, setting it somewhere in the map, then the robot will start moving towards the destination [11].

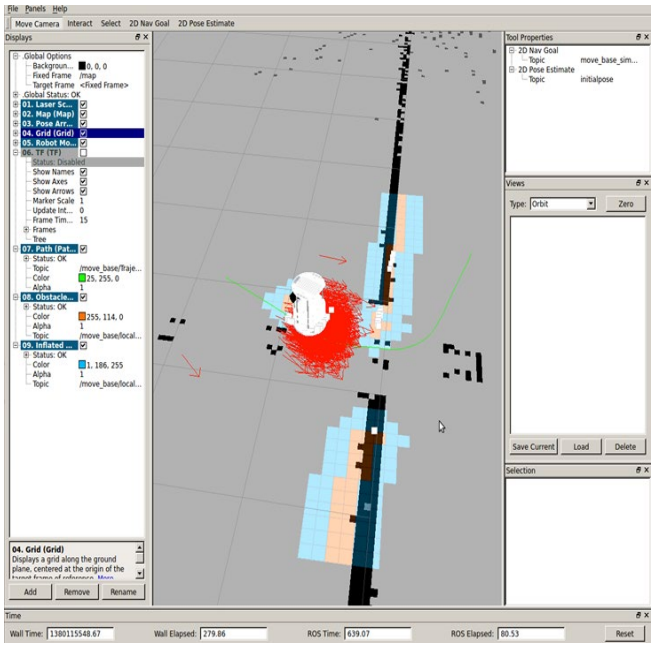

Figure 7. Navigation of Turtlebot.

5. Conclusion

ROS is a very powerful control system. There is an increased tendency of supporting a larger variety of robots by ROS. ROS provides Gazebo simulator, and it is used for simulating the robot and the environment together. For visualizing the data published from Turtlebot the RVIZ tool is used. The combination of Gazebo and RVIZ brings the feel of real demonstrations.
Once mapping and localization are successfully done then navigation can be easily achieved. The ROS navigation stack uses plugins for local and global planning which makes ROS very useful and very simple to navigate in undefined environments. In this paper the move_base package has been used for implementing path planning, which accomplished autonomous navigation.

References

[1] “The free dictionary”. [Online]. Available at http://www.thefreedictionary.com

[2] Steve Cousins, Brian Gerkey, Ken Conley, and Willow Garage: Sharing Software with ROS, IEEE Robotics & Automation Magazine, June 2010.

[3] Steve Cousins: Is ROS Good for Robotics? IEEE Robotics & Automation Magazine, June 2012.

[4] Steve Cousins: Welcome to ROS Topics: EEE Robotics &

Automation Magazine, March 2012.
[5] http://www.willowgarage.com/turtlebot
[6] S. Thrun, B. Wolfram and D. Fox, “Probabilistic robotics”, Cambridge, MIT Press, 2006.
[7] "ROS.org". [Online]. Available: http://wiki.ros.org/. [Accessed 19.09.2013].
[8] S. Cousins, "Welcome to ROS Topics", IEEE ROBOTICS & AUTOMATION MAGAZINE, March 2010.
[9] S. Riisgaard, M. Rufus Blas, “SLAM for Dummies, A Tutorial Approach to Simultaneous Localization and Mapping, Massachusetts Institute of Technology”, [Online], Available: http://ocw.mit.edu/courses/aeronautics-and-astronautics/16-412j-cognitive-robotics-spring-2005/projects/1aslam_blas_repo.pdf
[10] E. Marder-Eppstein, E. Berger, T. Foote, B. P. Gerkey, and K. Konolige, “The Office Marathon: Robust Navigation in an Indoor Office Environment”, In International Conference on Robotics and Automation (ICRA), Anchorage, AK, USA, 2010.
[11] P. Avdullahu, “Turtlebot control, mapping and localization via ROS”, Master Thesis, University of Prishtina, Kosovo, 2014.