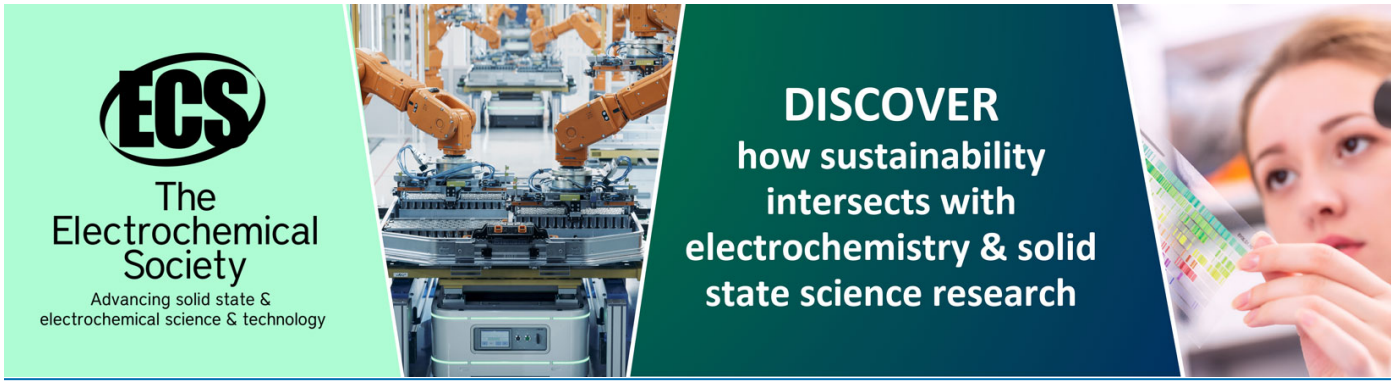
Setting up and Using ROS-Kinetic and Gazebo for Educational Robotic Projects and Learning



IOP Conf. Series: Journal of Physics: Conf. Series **1207** (2019) 012019 doi:10.1088/1742-6596/1207/1/012019

Setting up and Using ROS-Kinetic and Gazebo for Educational Robotic Projects and Learning

S Pietrzik[1, a] and B Chandrasekaran[1, b]

[1] Department of Computer Engineering, Florida Polytechnic University, 4700 Research Way, Lakeland, FL 33805-8531, USA

E-mail: [a]spietrzik2910@floridapoly.edu and [b]bchandrasekaran@floridapoly.edu

Abstract. This paper is to introduce readers to the Robot Operating System (ROS) and Gazebo, a robot simulator, for educational use. The paper covers setting up a proper environment for ROS Kinetic and the steps needed to properly install ROS into the setup environment. It includes an explanation of ROS and its inner workings, notably nodes and packages. This includes, but is not limited to, information on messages, topics, subscribers, and publishers. After a foundation of ROS understanding is established, Gazebo is introduced for the reader to test nodes without need of a physical robot. The Gazebo sections include an understanding of Gazebo, using a model, building a map/environment, running nodes, and sensors. Furthermore, this paper introduces the current project along with our team's current progress. The use of teleop and lidar within a project. Lastly, the paper covers future works and direction of the current project.

1. Introduction

Robotics is an important field in our modern society. Many technologies that we interact with today branch from the stem of robotics. Interacting with robotics can be confusing without guidance and overwhelming on how to start. Educationally, Robot Operating System (ROS) is the most used robotics tool, with open source and an active community, ROS has made robotics accessible going newcomers a place to start [1].

This paper's goal is to guide newcomers into the world of robots using ROS and Gazebo. This covers from setting a Linux environment, installing ROS, making a package, setting up Catkin, writing a node, and running a node. Along with explanations and examples to help the reader understand and follow along.

2. ROS Environment

The proper ROS environment is important because only certain versions of ROS run over certain versions of Linux. Additionally, there are different ways you can setup your environment which have their own benefits and downsides. For this paper, the ROS version that is being used is ROS-Kinetic and the Linux version Xenial (Ubuntu 16.04) [2].

2.1. Choosing the Linux version

For ROS-Kinetic only three versions of Linux can be used to run it, Wily (Ubuntu 15.10), Xenial (Ubuntu 16.04) and Jessie (Debian 8) [3]. Two versions are of ubuntu and the other is a version of Debian. The two choices are between Ubuntu and Debian, this is more of a preference choice. If you

are new to Linux, Ubuntu is a user-friendly distribution, that is easy to pick up and use. People who choose Debian, do so because they prefer it more than using ubuntu, however most of the community used ubuntu, so support is easier to find. The paper did not test the Debian build, so the effectiveness of Debian not measured here. However, between the two Ubuntu Distributions, Xenial (Ubuntu) was the easiest to work with and is recommend using as the operating system [3]. Xenial still have a working apt-get repository to install ROS, which Wily does not have. Finally, Xenial is more updated as well, making the interaction smoother.

2.2. Running the Operating System

After choosing the operating system that is going to be used, the next step is to figure out how the operating system is going be run. There are two options to choose from: a virtual machine or as the main operating system. *Do note, if you are planning to run ROS on a robot that you are using, this means you need to install the operating system directly onto the robot. Virtual machine are only used for testing and development*. Each have their own benefit and downside.

2.2.1. Virtual Machine. A virtual machine is an emulation of a computer, allowing a simulation of an operating system without needed to hard install it onto a computer as its main operating system [4]. This allows for the operating system to be run along with the current operating system that is being run. This method is more user friendly, due to simplicity of installations and the fact that the operating system can be contained to only be used when need. With the convince, comes one major downside, because the operating system is being emulated though the virtual machine, it does not full access of the computer's hardware. This means only a portion of the computer's hardware is being used to run the operating system. Thankfully ROS and Linux are relatively light and do not require heavy processing power, however with a more complex system and node structure this can become noticeable. For entry level code and systems using a VM should still be no problem. Lastly, this also is harder for Gazebo, since Gazebo renders and simulates the robot in a 3D space it requires more video proceeding, which is hard to properly give over a VM.

2.2.2. Dedicated Machine. Having a dedicated machine, means you install the operating system onto the computer. There are two configuration that can be done depending on how you want to use the operating system. The operating system can be set to be the main system on the computer or alongside the main operating system. By making it the sole operating system gives it full unrestricted access to the hardware of the system. With dual booting, both systems must share hard disk space, however this comes with the upside of having the original operating system with the Linux, meaning the computer is not completely converted.

Once decided on which method works best, instillation for both methods are similar other than a small variance for dual boot. For dual boot, make sure that a software is preinstalled to allow dual boot and that the BIOS is set to ask which operating system to launch from. There are different softwires that help for this process, chose the best for the system. For both methods the process are identical, with a USB with the Linux image enter the computers BIOS and install from there. The installation should guild the steps need for a clean installation onto the system. The main downside of this method is the lack of convince, but if that is not a problem, this is the recommended method to run the operating system on.

3. ROS

The Robot Operating System (ROS) is a flexible framework for writing robot software [5]. ROS has an assortment of tools and libraries that that work together to program any robot. The open software makes ROS accessible to millions of users across the globe, and the active community have help others and birth innovation. Before getting into ROS and its inner workings, the task of ROS installation needs to be done first.

3.1. ROS Installation

The installation of ROS is fairly easy and direct. Under the website in the Kinect section, there is a step by step installation guide [3]. The installation lists commands to add the responsory into the apt-get library so ROS can be installed through that. Most of the time the main repository does not work, and you have to choose from the mirrors, choose a close server or a working one. For the package to install, choose Desktop-Full, this includes all the tools that come with ROS which is helpful development, for a more focused build or idea, downloading the Base is optimal and any additional tools or library needed. Lastly, after the installation, make sure that ROS is sourced by running the code:

```
$ roscore (1)
```

It should output the same image from 'figure 1'. If nothing happens or an error pops up, this means that either ROS is not installed or not sourced. To source ROS type the command:

```
$ source /opt/ros/kinetic/setup.bash (2)
```

Run the code again from code line (1), and the output from 'figure 1' should be shown. If not, then ROS is not properly installed. Recommend actions for this, is to run through the installation guide once more or look at the forum to see if anyone else has the same issue and solution.

Once ROS is installed and have been verified to be working, another tool to use is catkin-tools [6]. A version comes with ROS, but it is more of a light version. Catkin is the build system that ROS uses to run and compile code. The next big library to install are the TurtleBot library. The TurtleBot library is a grate introductory resource that has basic files to build a foundation off [7, 8]. The installation packages are listed on the wiki, but under the indigo version of ROS not the Kinetic [8]. The version needs to be changed in order to get the correct packages, simply change every instance of indigo with kinetic and it should work fine. This should provide all the necessary packages for TurtleBot. Additional software that is Recommend installing is a text editor of choice and Terminator [9].

3.2. ROS and Nodes

ROS runs off a node system. A node performs a computation [10]. These nodes are meant to operate at a fine-grained scale; a robot control system will usually comprise many nodes [10]. Think of a node as a point in a network, each responsible of their own task and sending and reviving information to other nodes. There are three major components to a node; publishing, subscribing, and topics. ROS nodes needs to have the ROS master active to communicate and run correctly. The master needs to be running on its own terminal and can be invoked with the command (1) and should display what is seen in "figure 1".

3.2.1. Topics. Topics are named buses over which nodes exchange messages [11]. Topics are the data network of a node, they name the data and push or pull to other nodes. A node has at least its own topic plus which ever other topics that the node themselves publish. Topics are an important part of a node and having the right topics active are important to get the robot working properly. Some important topics include teleop (navigation), odem (location), and scan (lidar, visual input). These topics help navigate the robot around, teleop send values to the motor of the robot, moving it around, while odem and scan allow the robot to see the world around it. This is just a small taste of what a topic can do.

3.2.2. Publishing and Subscribing. While topics are the main identities all they act are as a location for the data to go. Publishing and subscribing are the transportation systems used to transport message from one topic to another. Publishing is when I node is constantly pushing the message from the topic to other nodes that are subscribed to the topic. Subscribing works in the reverse, a node that is subscribing listen from the chosen topic, waiting for a message to be pushed from the topic. This is how data is passed with a ROS node network.

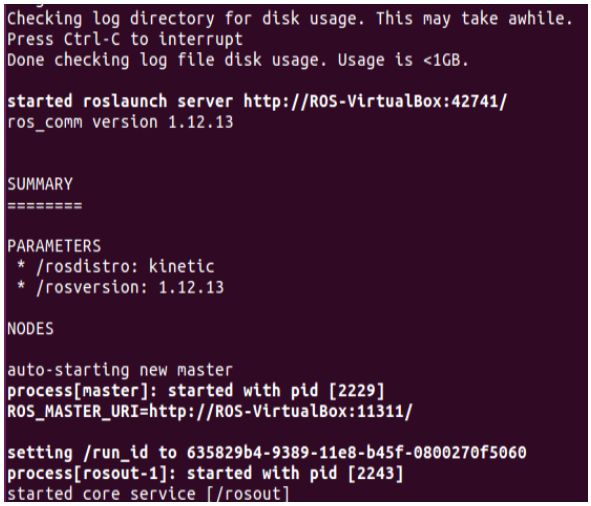


Figure 1. Desired output for the roscore command**.**

3.2.3. Example. Let's have three nodes teleop, lidar, and my_node. Teleop node has the topic teleop, the lidar node has the topic scan, and my_node has the topic teleop and scan. The my_node, is subscribed to the teleop and scan topic getting the information from both nodes. The my_node is also publishing to the teleop topic. The my_node topic pulls from the messages from teleop and scan. With that information, the nodes does whatever it is programmed to do. Then it publishes a message to the teleop moving the robot as intended. 'Figure 2' shows an example of a node graph.

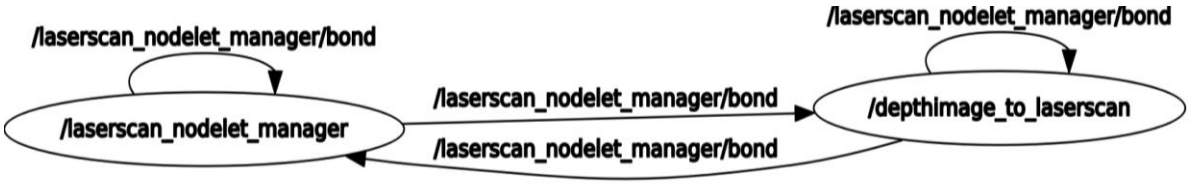


Figure 2. Node / Topic graph, arrows represent topics and direction show pub/sub, circles represent nodes

3.3. Packages

Being able to understand a node is nice but now it is time to make one. Packages is how ROS organizes nodes and this is where catkin comes into play. To create a node a package must first be made and to make a package a workspace is needed. To create a workspace, in terminal, in a desired location create a workspace folder with a 'src' folder within it. This can be done with the command:

$ mkdir -p {workspace name}/src (3)

$ cd {workspace name}/src (4)

By doing command (3) and then (4) should lead to the workspace inside the source folder (src) [12]. Next is to use catkin to make the workspace.

$ catkin_init_workspace (5)

Then go up a directory using the cd command and build using catkin

$ catkin_make (6)

Now in the {workspace name} there should be three folders build, devel, and src. This means the workspace has been properly created. Next is to source the workspace just like it was done with ROS.

$ source devel/setup.bash (7)

For command to work, it needs to be ran in the workspace folder. Next is packages, packages are held within the src folder workspace. Go to the src folder by cd into src, variant of command (4). Next is to make the package [13].

$ catkin_create_pkg {package name} [depend1] [depend2] [depend3] (8)

This creates a package with the given name alongside the chosen dependences (Ex. rospy, for python, or roscpp, for c++). Enter the package by cd into the package, within the package should be two files and two directories CMakeList.txt, Package.xml, src, and include. The package can be edited with the xml file, to add more dependencies, name, and other meta data. Within src of the package is where the node is held, the node tells which topics are being published to and subscribed to. This is explored in a future edition of the paper.

3.4. Making a node and future tools

Currently the paper does not cover the process of making a node within the package and more in-depth tools like rivs, rqt, and libraries like open cv. These are topics for future iterations of the paper, as the paper is developed farther.

4. Gazebo

Gazebo is a robot simulator, it allows for live testing and simulation of a ROS environment and node system without need to have the physical hardware of the robot. This allows for robotic development without the dedicated need for a hardware to properly test software. Gazebo incorporates itself into the ROS node network so that the nodes do not notice the lack of hardware or environment. This allow for much more innovation and development that can be stopped by not have the hardware.

4.1. Getting Accustomed with Gazebo

While Gazebo is a great tool, at first glance it can daunting. First, to launch Gazebo, open terminal and enter the command:

$ Gazebo (9)

This should launch Gazebo bring up a window like in ‘Figure 3’. The panel on the left deal with objects and the world, the top bar is a quick tool that deal with sizing, undo/redue, basic shapes, lighting, alignment, and camera angle. Lastly, is the centre, the building plain, this show the world and point space being simulated.

The left panel have three tabs, the world tab list all active elements within the simulated world, in addition to a property panel which reflects the selected element. The insert tab list different models that can be added into the world. The layers tabs to for model organization by adding different models to different layers to add changing environment.

On the top row, the first icon is the selection mode, this allows for selecting elements within the world and interacting with them. The second icon is translation mode, this allows movement of the element selected. The third icon is rotation mode which allows the rotation of the element. Lastly, is the scale mode which allows for the scaling of the element. Other than that, the next non-direct element is the are the lighting tools located next to the simple shapes, each adds a lighting source on to the world plane. The first icon, point light, adds a spherical light onto the world dispersing light around the sphere. The second icon, spot light, add a contained directional light source to the world. Lastly, the last icon, directional light, adds a global directional light source to the world.

For the bottom row, the pause and play button respectively pause and play the simulated world. The reset timer, reset the sim time and real time for any needed real time timing with in the simulation.

Lastly, the main panel is controlled by the mouse. By left clicking, this allows for directional movement within the world, middle mouse changes the angle of view at a fixed point, and the right click changes the zoom the camera is to the world. This should cover the main functions of the starting screen of Gazebo.

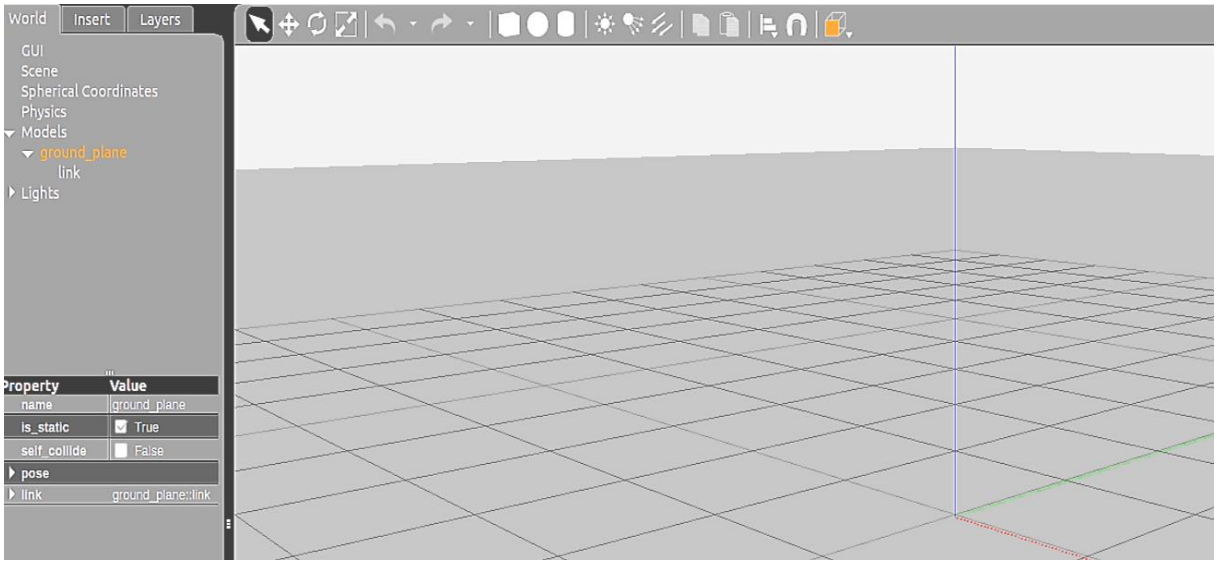


Figure 3. Start-up view of Gazebo

4.2. Making a world

After opening Gazebo, everything needed to make a world for the robot is there. The basic shapes on the top are useful for making generic obstacles and pathways, for more complex or detailed elements the insert tab on the left side had built in models and objects as well. Once the world is created, it can be saved by going to the top left on to the menu bar, go to file, and press 'Save World As'. Then save the world to a desired location. To open the world again type the command:

$ Gazebo {file location} (10)

'Figure 4' provides an example of a world created through gazebo.

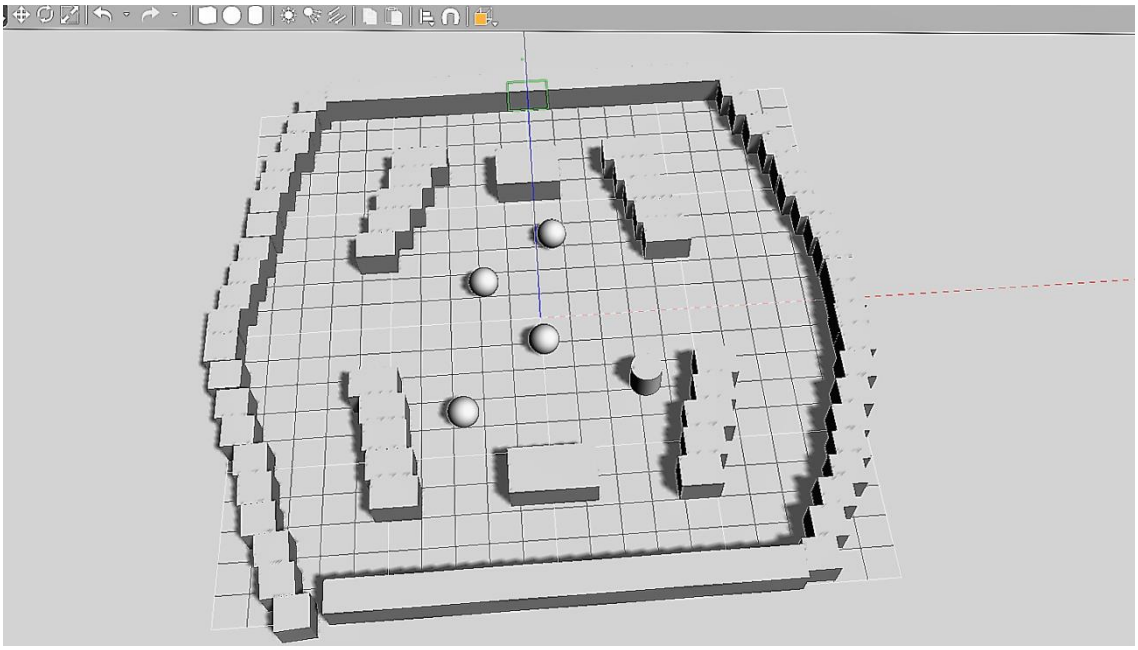


Figure 4. World space created in Gazebo

4.3. Running TurtleBot within Gazebo

The TurtleBot library proves a Gazebo launch file to emulate their robot with the Gazebo simulator. A launch file, is another ROS tool that allows for multiple nodes and configurations to be run within one simple command. To run a launch file, the command is *roslaunch* compared to the normal *rosrun*, which is needed to run a node. The command to launch the TurtleBot is [14]:

$ roslaunch turtlebot_gazebo turtlebot_world.launch (11)

This command should load up a TurtleBot into Gazebo surrounded by objects, seen with 'figure 5'. Having the TurtleBot loaded within Gazebo is the same as having physical version. Now that the robot is being simulated through Gazebo, next is to run a node that interacts with the robot and watch it move within Gazebo's simulated environment. Using *rosrun* or *rolaunch*, run the node that is being tested on the robot, if needed the example teleop node from TurtleBot can be ran with [14]:

$ roslaunch turtlebot_teleop keyboard_teleop.launch (12)

With the following command, the robot moves based on the keyboard command sent from the teleop node. The TurtleBot is a great model to work off and has a range of instrument and sensors to test an assortment of different nodes and software.

4.4. Advance functions with Gazebo

Gazebo allows for an arraignment of customizations to emulate any model or environment. This section will mention the functions and go over the basic understanding of it. Gazebo comes equipped with a model editor, the model editor allows for more controlled editing of objects, combining elements, and interactions between objects (axis and/or latches). With the model editor it is possible to make any robot model or world object to simulate within Gazebo. The next tool to go over is the plugins. Plugins are scripts that interact with the Gazebo environment, plugins are important for sensors and moving object within Gazebo. Without plugins, the ROS topics would not communicate with the active robot and the sensor data being retrieved would be null. With Plugins, movement of objects can be added, triggered events, and sensors can be implemented into the Gazebo world.

5. The Project

The project, is a TurtleBot 3 that will be able to navigate to a specific location using teleop and lidar. Currently the project is working with simulating the robot within Gazebo to ensure that the software works with the robot. Using lidar, the robot moves forward until an object is too close, then it enters user-controlled mode, giving the user, human, control of the robot. With this the user can move the robot away from the obstacle and return to autonomous mode, 'figure 5'. ROS theory and examples can be found in [15].

5.1. Future of the work

Currently, the robot need to have path planning added, to make it navigate from point a to goal point. With the path planning the robot will be up to the current goal. For the future, the robot will include sensor fusions of camera and lidar. This will make for better world interpretation for the robot and the object detection. An object avoidance is also needing to reduce the amount of human robot interaction needed. An arraignment of different path planning's is also going to be implemented to test the efficacy of the different algorithms.

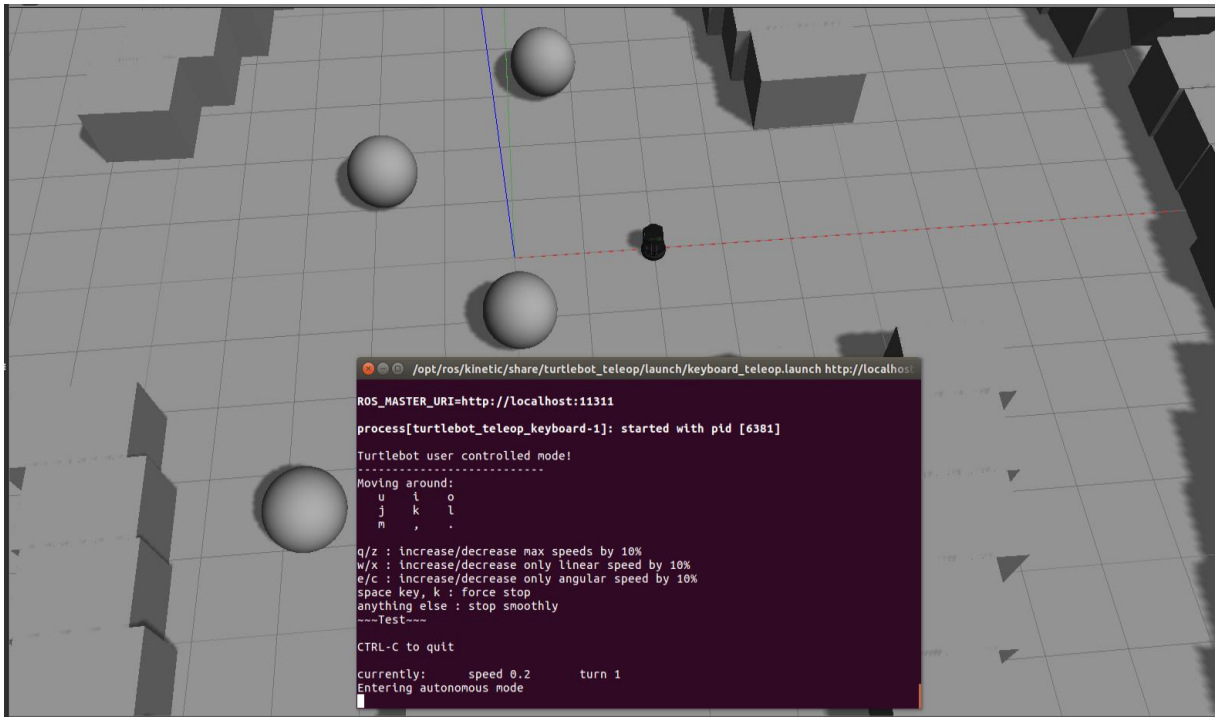


Figure 5. The robot running autonomous within Gazebo.

References

[1] Open Source Robotics Foundation *Is ROS For Me?* http://www.ros.org/is-ros-for-me/
[2] Ubintu *Ubuntu 16.04.5 LTS (Xenial Xerus)* http://www.releases.ubuntu.com/16.04/
[3] DHood *Ubuntu install of ROS Kinetic* Open Source Robotics Foundation http://wiki.ros.org/kinetic/Installation/Ubuntu
[4] Oracle *Wellcome to VirtualBox.org* https://www.virtualbox.org/
[5] Open Source Robotics Foundation *About ROS* http://www.ros.org/about-ros/
[6] Open Source Robotics Foundation *A Brief History of Catkin* Revision b5c54586 http://catkin-tools.readthedocs.io/en/latest/history.html
[7] Yoonseok Pyo *TurtleBot* Open Source Robotics Foundation http://wiki.ros.org/Robots/TurtleBot
[8] AnisKoubaa *turtleBot/ Tutorials/ indigo* Open Source Robotics Foundation http://wiki.ros.org/turtlebot/Tutorials/indigo
[9] Stephen Boddy *Introduction* https://gnometerminator.blogspot.com/p/introduction.html
[10] KenConley *Nodes* Open Source Robotics Foundation http://wiki.ros.org/Nodes
[11] DariushForouher *Topics* Open Source Robotics Foundation http://wiki.ros.org/Topics
[12] WillamWoodall *Creating a workspace for Catkin* Open Source Robotics Foundation http://wiki.ros.org/catkin/Tutorials/create_a_workspace
[13] davetcoleman *Creating a ROS Package* Open Source Robotics Foundation http://wiki.ros.org/ROS/Tutorials/CreatingPackage
[14] IsaacSaito *Explore the Gazebo world* Open Source Robotics Foundation http://wiki.ros.org/turtlebot_gazebo/Tutorials/indigo/Explore%20the%20Gazebo%20world
[15] Jason M O'Kane 2013 *A Gentle Introductiong to ROS* Independently published (978-1492143239)