TurtleBot Technical Manual for MIE 443H1S

Prepared by:
Daniel Dworakowski
Christina Moro
Christopher Thompson
Farzad Niroui
Sharaf Mohamed
Kaicheng Zhang
Siles Alves
Pooya Poolad
Goldie Nejat

January 2023

Table of Contents

1 Introducing the TurtleBot 2

1.1 What is a TurtleBot?

A TurtleBot 2, Figure 1, is a robot prototyping kit built on open-hardware and open-source software. It is supported by the Open Source Robotics Foundation. TurtleBots are designed to help you quickly build a robot capable of navigating autonomously, sensing in 3D, and being easily customized for different applications.

Figure 1: The TurtleBot 2 platform

1.2 What will you do with the TurtleBot?

In this course, you will learn how to control the TurtleBot 2 using the Robot Operating System (ROS). You will also learn to develop sensory-based intelligence applications for the robot through contests.

1.3 TurtleBot Etiquette

1.3.1 Carrying your TurtleBot

The TurtleBot is a fragile robot that must be treated with the utmost care. To transport your robot, always hold it from the base with two hands, as shown in Figure 2. The laptop should be carried separately by another member of your team.

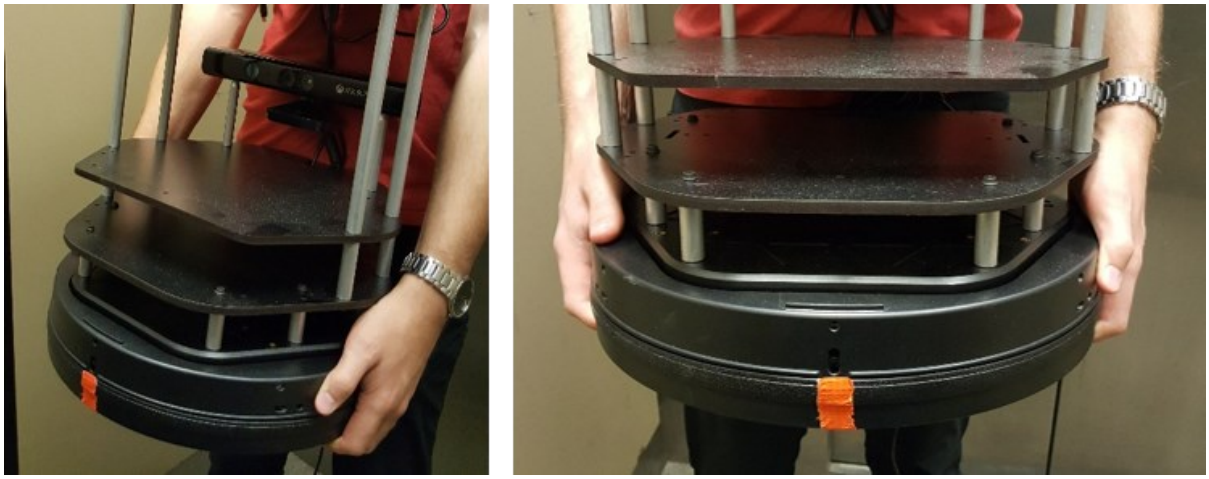


Figure 2 Demonstration of how to carry the TurtleBot 2 platform

1.3.2 Charging your TurtleBot

Your team is responsible for proper maintenance and caretaking of your robot. This includes ensuring no damage is done to the hardware and that your laptop is maintained in good condition.

Each team is responsible for charging both their TurtleBot and laptop.

1.3.3 Testing on your TurtleBot

All algorithms developed for contests and labs should first be tested in simulation before implementation on the TurtleBot. The simulator used in this course is Gazebo and can be used to test algorithms prior to integration onto the physical robot, which will be detailed in Section 5.5.

2 TurtleBot 2 Technical Specifications

2.1 Hardware Components Overview

The core components of the TurtleBot 2 robot are shown in Figure 3. The TurtleBot 2 used in this course has an iClebo Kobuki mobile base, on which 3 supporting plates are stacked as well as a Kinect for Xbox 360, all controlled via a Lenovo 11e laptop.

Note: As the TurtleBot project is open-hardware, different distributors may interchange components, such as the type of depth sensor and laptops used. Some may also use the iRobot iCreate platform rather than the Kobuki base. Be aware of these distinctions while searching for technical information.

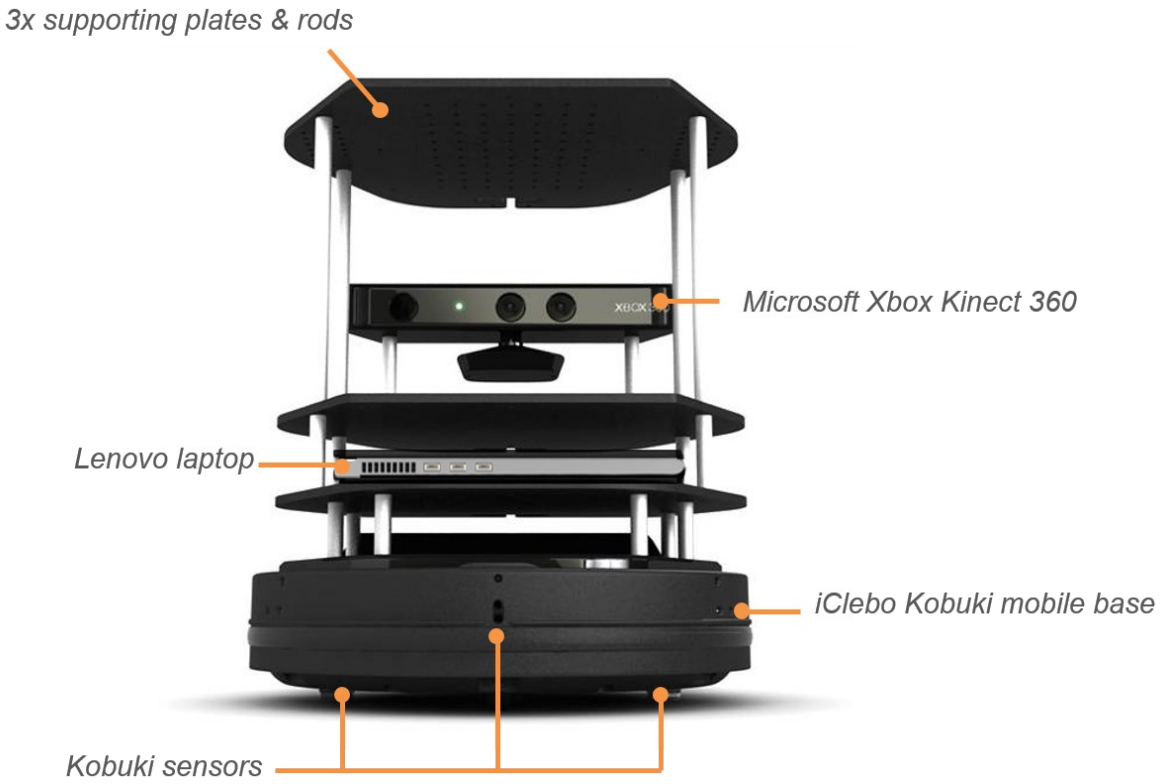


Figure 3 TurtleBot 2 hardware components overview

2.2 iClebo Kobuki Base Technical Specifications

2.2.1 Control Panel

The TurtleBot provides several accessible ports for supplying power to external sensors (e.g. Kinect sensor) and controlling the base with the laptop. The iClebo Kobuki provides power ports for the laptop, Kinect sensor, an external arm, and a general-purpose 5V supply. The LED lights can be programmed to turn on or

flash according to a predefined state. The Status LED turns green when the Kobuki is on, orange when charge is low, and flashes during charging. See Figure 4 for port labels and details.

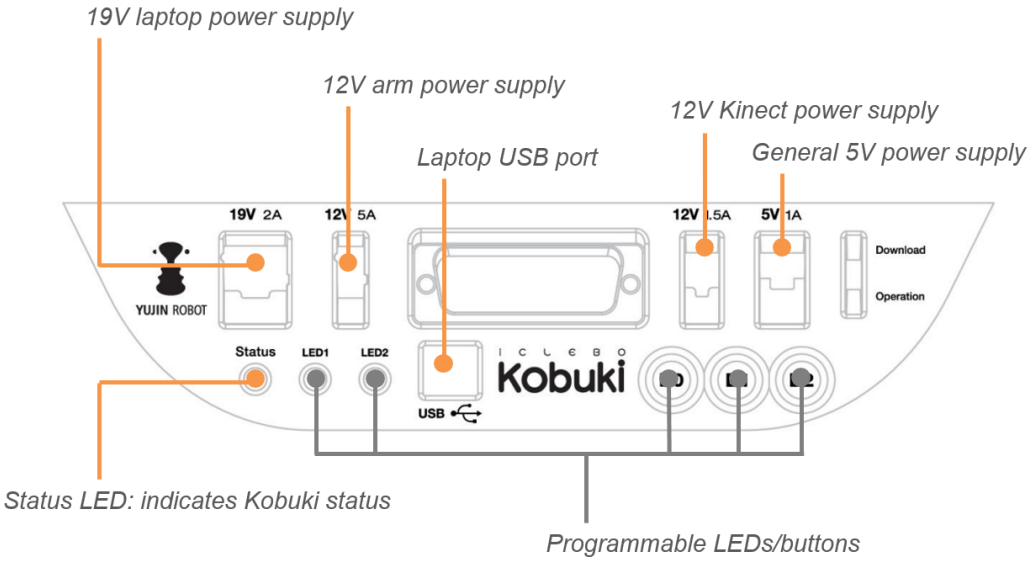


Figure 4 Kobuki Base control panel inputs and power supplies

2.2.2 Sensors

The iClebo Kobuki is equipped with the following sensors: 3x cliff sensors, 2x wheel drop sensors, and 3x front bumpers. The base is also equipped with a gyro and wheel encoders. A diagram of the base sensors on the iClebo Kobuki is shown in Figure 5. Power requirements are given in Table 1, performance characteristics are provided in Table 2, and sensor characteristics are given in Table 3, respectively.

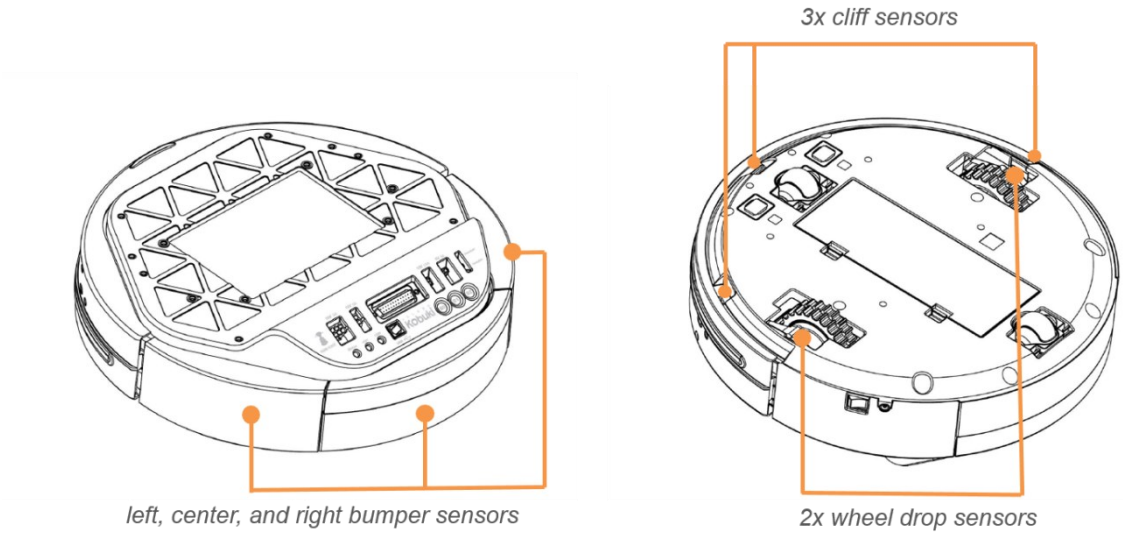


Figure 5 Sensor locations on the iClebo Kobuki base

Table 1 iClebo Kobuki power specifications

Small lithium-ion battery:	14.8 V, 2200 mAh (4S1P) 3 hours operating time 1.5 hours charging time
Large lithium-ion battery	14.8 V, 4400 mAh (4S2P) 7 hours operating time 2.6 hours charging time
Recharging adapter:	Input: 100-240 V AC, 50/60 Hz, 1.5 A max Output: 19 V DC
Laptop charging connector *(only enabled when robot is charging):*	19 V/2.1 A DC

Table 2 iClebo Kobuki base performance characteristics

Max translational velocity:	65 cm/s
Max rotational velocity:	180 deg/s
Max payload (hard floor):	5 kg
Max payload (carpet):	4 kg
Max height Kobuki can climb	12 mm thick (e.g. carpet)

Table 3 iClebo Kobuki sensor specifications

Gyro	1 axis (110 deg/s) 52 ticks/enc rev
Encoders	2578.33 ticks/wheel rev 11.7 ticks/mm
Cliff sensor depth threshold	max 5 cm deep

Further documentation on the iClebo Kobuki base can be found on the website at the following link: http://kobuki.yujinrobot.com/documentation/

To test the base sensors, please see Section 7.1 Testing the TurtleBot.

2.3 Lenovo Laptop Specifications

The laptop you will be using to control the TurtleBot is the Lenovo 11e. Computing specifications for this laptop are provided in Table 4.

Each turtlebot has a laptop assigned to it. That laptop will be the main device for your group to develop the software for the contests.

Table 4 Lenovo 11e Specifications

Processor	Intel Celeron, Intel Pentium, or AMD A4
Graphics card	Intel HD or AMD Mullins
RAM	8 GB
Hard drive	120, 128 or 500 GB

2.4 Microsoft Kinect 360 specifications

The Microsoft Kinect is composed of three main parts: (1) a RGB camera; (2) a depth sensor; and (3) a microphone array [1]. The sensor layout is shown in Figure 6.

RGB camera: The RGB camera consists of red, green, and blue (RGB) channels. These colours are known as the additive primary colours. The brightness of the red, green, and blue pixels is determined separately, each image pixel will consist of three RGB numeric components (a value of 0-255 for red, blue and green). The pixel's overall colour will be a combination of these. The final RGB colour image consists of these pixels. In this course, we will use the Kinect's RGB camera to perform object detection and recognition.

Depth sensor: The depth sensor is composed of an infrared projector coupled to a monochrome CMOS sensor. The projector is an infrared (IR) laser emitter, which projects a speckle pattern of IR beams onto objects in the environment, and an IR depth sensor, which detects the IR beams reflected back to the depth sensor. The sensor uses a structured light technique, where the pattern is projected onto an object or environment, which deforms the pattern based on its geometric shape. The distortion of the original pattern by the object is used to determine depth information. The *x,y,z* coordinates of each point in the observed pattern can be computed, resulting in a 3D point cloud.

Microphone array: The Kinect contains an array of four microphones that can be used to identify the location of a sound source and detect the direction of the sound wave. The array can be used to isolate a specific sound in noisy environments. One example application is the detection of voice commands from a person in front of the robot in the presence of background noise from robot motors and other environmental sounds.

Table 5 Microsoft Kinect 360 technical specifications

Pixel resolution *(at 30 FPS, for both RGB and depth streams)*	640 x 480 **Note that the RGB camera hardware allows for resolutions up to 1280 x 960 at lower frame rates.*
Frame rate (depth and colour streams)	30 FPS
Viewing angle	43° vertical x 57° horizontal
Vertical tilt angle	± 27°

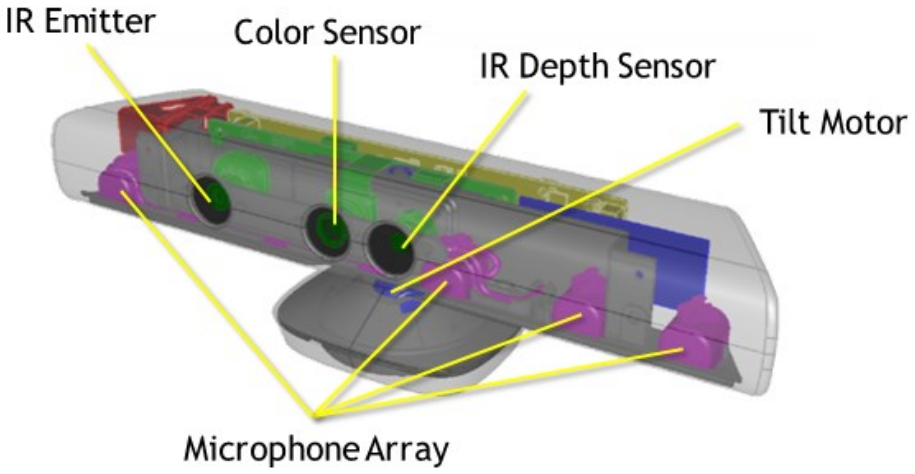


Figure 6 Microsoft Kinect 360 sensors [1]

3 Software & ROS overview

3.1 Introducing ROS

The TurtleBot SDK (Software Development Kit) is based on the Robot Operating System (ROS). The ROS website describes itself in this way:

"The Robot Operating System (ROS) is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behaviour across a wide variety of robotic platforms. [...] It provides hardware abstraction, device drivers, libraries, visualizers, message-passing, package management, and more."

ROS is designed by developers around the world, each contributing to developing a wide variety of robot functionality, from mapping an environment to manipulating objects in 3D space.

The TurtleBot project has its own set of ROS packages. These packages allow you to control the base, use the Kinect sensor to build a map of the environment, and autonomously navigate to a specified location, amongst other things. The material in this course builds on some of these existing packages. Contests and labs will teach robot navigation, map building, object identification, emotion recognition and human interaction.

3.2 Software Requirements

ROS is designed for the Ubuntu operating system. To install Ubuntu GNU/Linux 16.04 and ROS on your personal computer please see the following appendix: Appendix II: Installing Ubuntu 16.04 on your personal computer. The remaining sections will assume that you have Ubuntu installed.

ROS is compatible with both C++ and Python. We will provide you with contest packages for your team to complete. If you need to, please refresh your C++ and Python skills. For additional resources, please refer to Appendix III: Additional Resources.

4 Introduction to Ubuntu 16.04

To get started using ROS, you first need to be familiar with Ubuntu. Ubuntu is a GNU/Linux-based, open-source operating system. GNU is an Operating System that was initially developed by Richard Stallman in the 1980s, while Linux, the kernel, was originally developed by Linus Torvalds in the 1990s. GNU/Linux became a major success, and eventually was made available under different

distributions, e.g., Ubuntu, Fedora, Debian, Arch Linux, etc. We will be using Ubuntu 16.04 (Xenial Xerus), and ROS Kinetic.

Ubuntu is designed to be user-friendly and provides a large database (repository) of free software, making it perfect for developing robot applications.

ROS is used with GNU/Linux, so most of your development will be via the terminal command line. The following description is meant to help you become comfortable with using Ubuntu.

4.1 The Ubuntu Layout

When powering on the laptop, you will see the Ubuntu login screen shown in Figure 7. It will provide your username, and prompt you for a password.

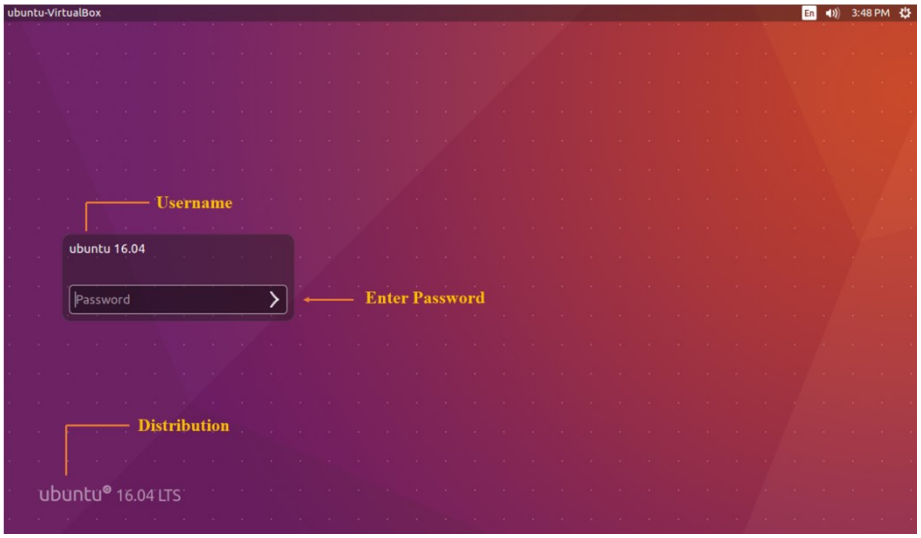


Figure 7 Ubuntu login screen showing username and password tabs

After logging in, you will see the Unity (Ubuntu's GUI) display shown in Figure 8. Take a moment to familiarize yourself with the launcher, terminal icon, and status menu.

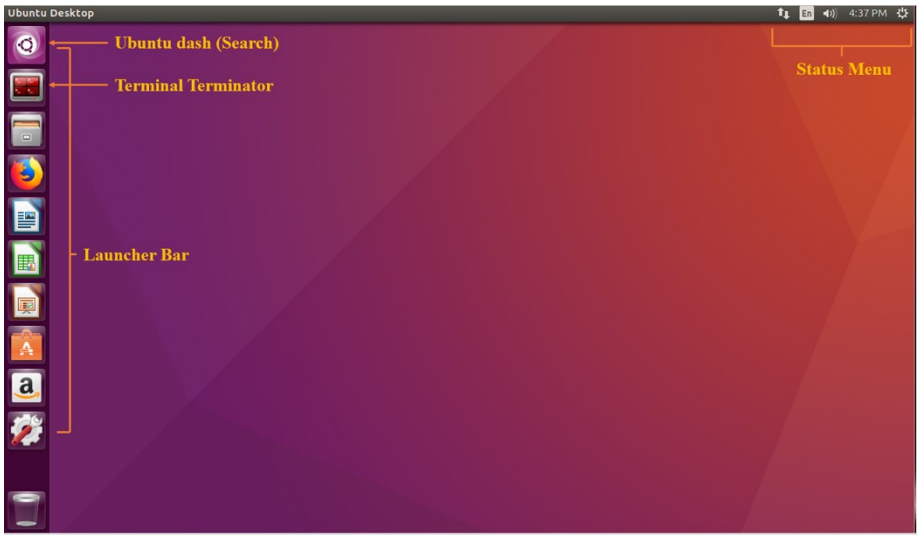


Figure 8 Ubuntu's Unity desktop layout showing terminal icons in launch bar

4.1.1 Using the Terminal

While Ubuntu comes with its own default terminal, we suggest you use Terminator. It is an efficient tool for working with many open terminal windows.

To open a terminal, click on the red terminator icon in the launcher as shown in Figure 9, or open the Ubuntu search and type in 'terminator' as shown in Figure 10.

Table 6 provides a list of useful terminal commands for you to become comfortable with.

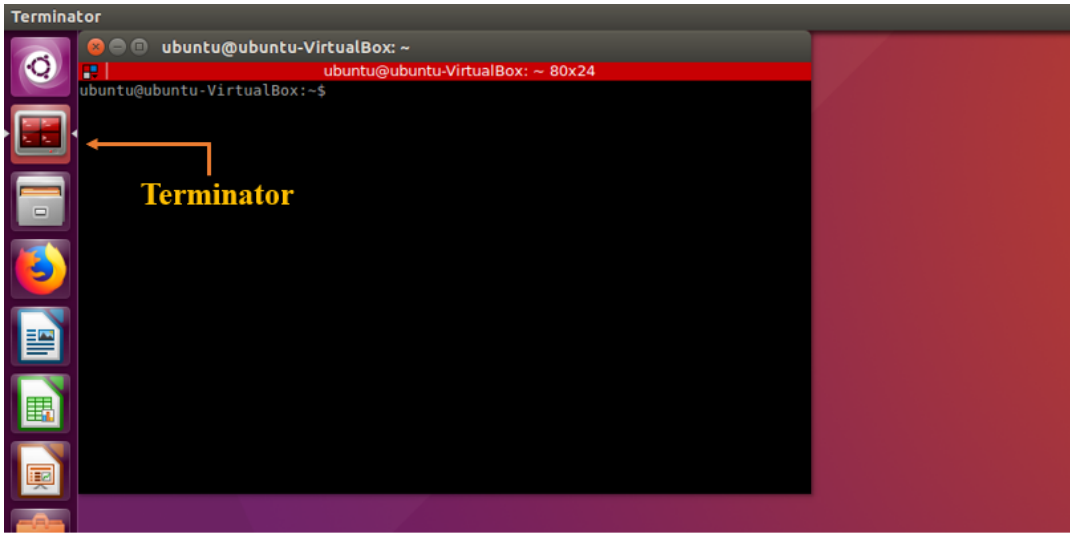


Figure 9 Terminator open via the shortcut in launch bar

Figure 10 Terminator found through search in Ubuntu Dash

Table 6 List of essential terminal commands

Terminal command	Action
ls	List all files in the current directory
cd	Go to your 'Home' directory
cd ..	Go back one step to a previous directory
cd /path/to/file/	Move to the directory specified
mkdir	Creates a new directory (e.g. ROS package)
rm	Delete a file or folder
mv	Moves a file or folder to another location
ifconfig *or* ip addr	Shows you the state of your network connections and provides corresponding ip addresses (important for ssh)
echo $HOSTNAME	Displays your laptop's hostname (this should be something like turtlebot-Thinkpad-11e)
pwd	Displays the path to the working directory
CTRL + C	Hitting the CTRL and C keys will terminate a running script

Exercise 1: Using the commands above, try the following exercise. Note that the final output is shown in Figure 11.

Open a new terminal using Terminator. List the files in your current (Home) directory.
Note: You will notice a folder called "catkin_ws". This is your ROS workspace.

```
ls
```

Navigate to the Desktop using the "cd" commands.

```
cd Desktop/
```

Create a folder called "temp" using the "mkdir" command.

```
mkdir temp
```

Navigate inside your temp folder and ensure it is empty.

```
cd temp/
```

Go back to your Desktop and delete your temp directory using the “rm” command.

```
cd ..
ls
rm -r temp/
ls
```

At the last “ls” command, you should see that the /temp folder is no longer listed. After implementing the above commands, you should see what is shown in Figure 11.

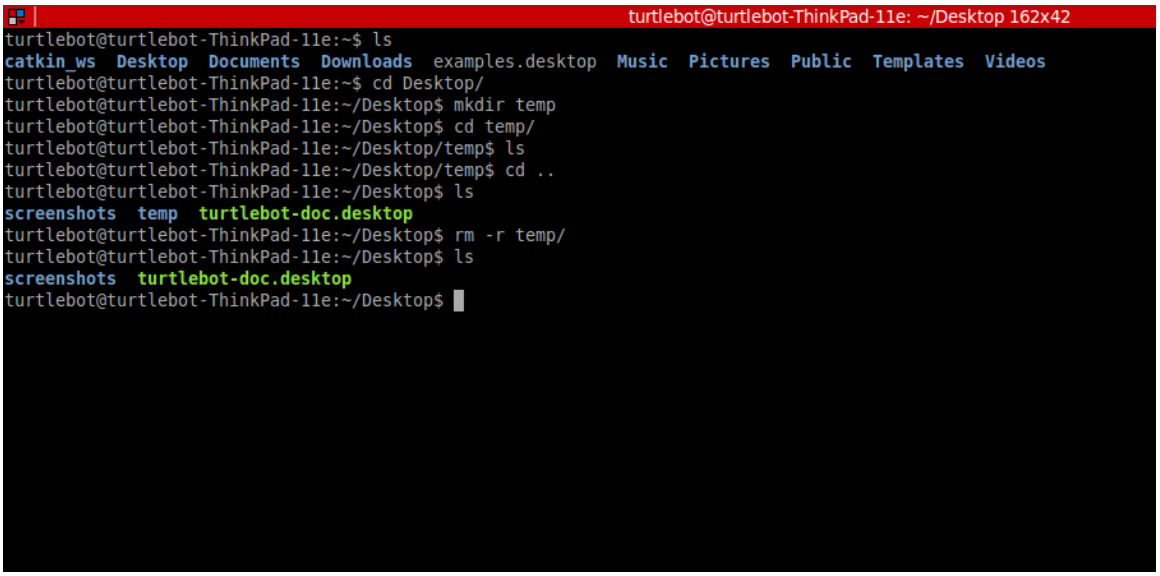


Figure 11 Example output of Exercise 1

4.1.2 Using the TAB-Complete feature

Rather than typing out an entire folder or file name, Ubuntu can automatically complete the line for you. Open a new terminal and start typing the name of an executable, file or folder in your current directory. Rather than write out the entire name, hit the **TAB** key on your keyboard. Ubuntu will either complete the line for you, or list the options available.

For example, to launch Firefox from the terminal, type in fir or fire, and press **TAB**. If Firefox is the only program starting with those letters, Ubuntu will complete the line for you. Then press **ENTER**.

This is also very useful to use for ROS. If you were to type in “rosrun turtlebot_” and press **TAB**, you should see all the different TurtleBot ROS packages currently available as shown in Figure 12. Please note that these ROS terms will be explained in the following sections.

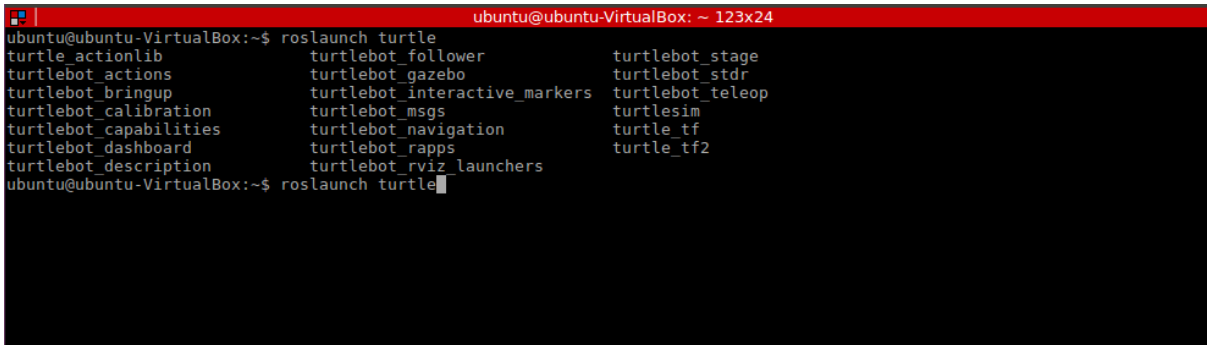


Figure 12 TAB-complete demonstration for running a TurtleBot node

For more tips and tutorials on using Ubuntu's command line, please refer to Tutorial 1 and Tutorial 2 from the University of Surrey Electrical Engineering Unix Tutorials [2].

****IMPORTANT: Ubuntu is designed to separate regular activities from "restricted-access" ones. The latter are called sudo commands. In this course, you will not be using any sudo commands. All necessary applications, packages and configuration files have been installed and configured for you. DO NOT download any software programs or packages, install updates, or modify any files that are not in your ROS workspace.**

4.1.3 A note on Terminator

Terminator is an advanced terminal which allows you to split your current tab into several sub-screens. It will be quite useful when you start running multiple ROS nodes.

To split your screen, use the following commands on your keyboard:
CTRL + O : Splits your current screen horizontally
CTRL + E : Splits your current screen vertically

This allows you to create custom setups for your needs. An example of a multiple-terminal setup for ROS using both types of commands is shown in Figure 13.

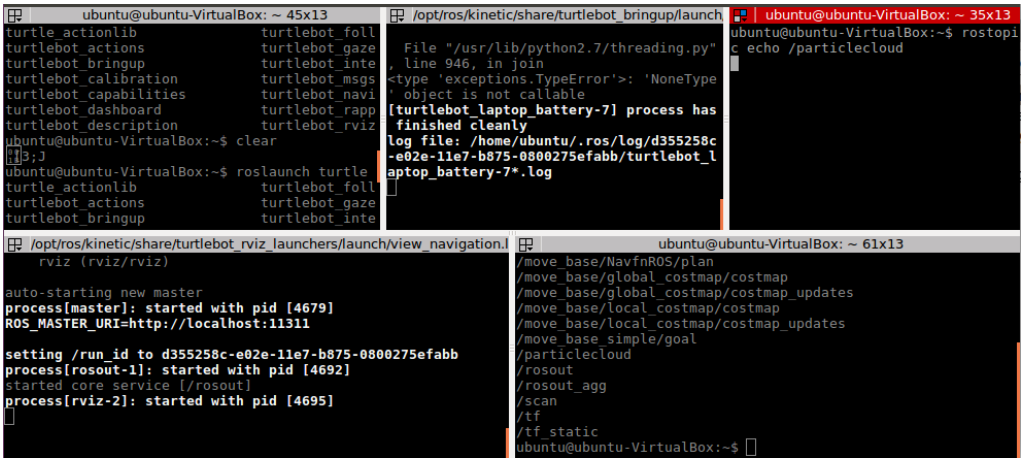


Figure 13 Terminator split terminal screens running ROS nodes and launch files

5 Introduction to ROS

5.1 What is ROS?

ROS acts like an operating system that controls a robot's functionality. It is a resource that provides common services for device control and communication, message-passing between processes, basic robot functionalities, and package management.

5.2 Why ROS?

Advantages of using ROS are:

- Free, open-source packages and algorithms readily available for implementation on the TurtleBot (e.g. navigation, mapping, sensor data processing, text-to-speech, etc.).
- Highly modular: no need to have all nodes running on the same machine. You may have a Raspberry Pi running your motors, a laptop sending text-to-speech requests to the Cloud, and an Android phone controlling the robot motions. They can all communicate together!
- Inter-platform operability: one node package may be written in C++, the other in Python, or Java, and they all can communicate with one another seamlessly.
- Continuous updates and maintenance made by developers around the world.
- Good hardware support – there are a lot of drivers and integrated packages to support different kinds of sensors and hardware.
- Easy-to-use development tools and resources.
- Great community and documentation.

5.3 How does ROS work?

ROS begins with the ROS Master, launched with the command line "roscore" typed into a terminal. Launching *roscore* should always look similar to what is represented in Figure 14. The ROS Master allows nodes to: (1) find each other; and (2) talk to each other. Implicit in (1) is its ability to find nodes being broadcast across many devices (e.g. a Raspberry Pi communicating with a laptop).

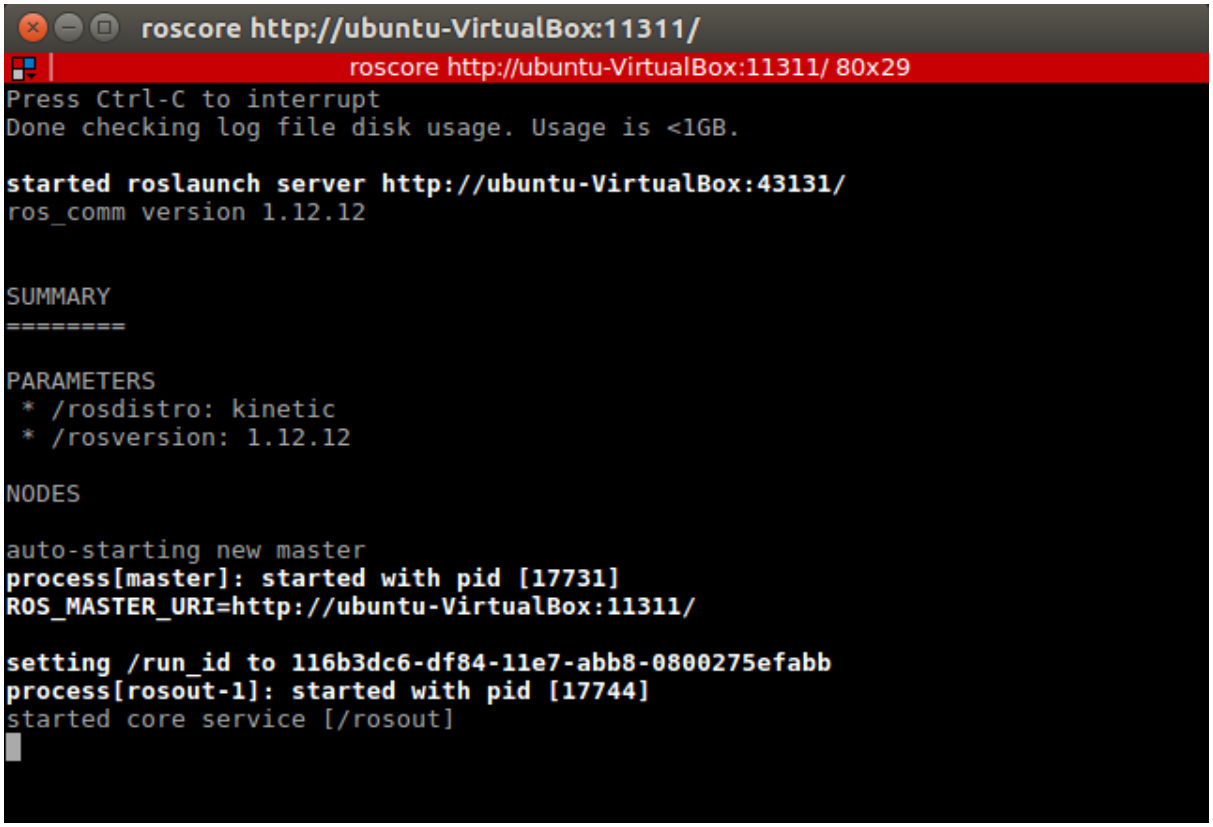


Figure 14 Example output after initiating roscore

5.3.1 Nodes

ROS uses independent "nodes" to activate robot functions, and offers a platform for these nodes to talk to each other. ROS defines a node as a "process that performs computation" for a very specific task [3]. In other words, a node is an executable that performs a single function. An example of a node could be the camera node for the simulated Kinect sensor, which broadcasts the RGB pixel data through a stream of messages. This stream of messages can be accessed by other nodes, for example, a node responsible for mapping or one for robot path-planning. This makes ROS a great robot communication tool.

5.3.2 Messages

Messages define the input/output data type for a node, and allow nodes to communicate by exchanging information. For example, if a robot's goal is to detect the presence of a door handle (i.e. the "Door Handle node"), it can take in images from a camera (input), and provide the coordinates of the door handle (output). In this case, one message containing pixel data for each image in the video stream is passed from the Camera node to the Door Handle node. The content of that message (the pixel data) is processed by the Door handle node. Then, a second message containing the door handle features is sent

from the Door Handle node to what is called a *topic*, where other nodes will be able to use the information.

5.3.3 Topics

A topic is the medium used by nodes to exchanges messages. In the previous example, the Door Handle node received messages from the Camera node. The Camera node is in charge of transferring the camera's video stream to the ROS framework. It sends a message containing pixel data (the message content) to a *topic* called /image_view.

In general, a node sending data to a topic is called a **publisher**, and a node obtaining information from a topic is called a **subscriber**. The publisher makes data (the message) available to any node interested in using it.

5.3.4 Publishers & Subscribers

At some point, you may want the TurtleBot to display the simulated Kinect RGB camera's 2D video stream. Perhaps you will also want to see the video to be displayed in both RGB colour and grayscale. This involves 3 nodes: a camera node, an image-processing node, and an image-display node. The camera node establishes communication with the simulated Kinect sensor. The image-processing node transforms the RGB video images into grayscale by converting pixel values. The image-display node outputs video data into a window. Each node is independent, performs a single function, and registers itself with the ROS Master, as shown by the grey connector lines in Figure 15.

The orange lines in the figure represent the direction of messages passed between nodes and topics. In this example, the camera node is a publisher, the image-display node is a subscriber, and the image-processing node is both a publisher and subscriber. The camera node *publishes* a *message* containing RGB pixel values to the *topic* /raw_image_data. To access those RGB pixels, the image-processing and image-display nodes will both *subscribe* to the /raw_image_data topic. This means both nodes can read the pixel data. The image-display node will use it to display a video, while the image-processing node will use it to convert images to grayscale.

The image-processing node obtains the RGB pixel values from the topic /raw_image_data, and converts them to grayscale. It then *publishes* the grayscale pixel values to a **new** *topic* called /processed_image_data. The image-display node subscribes to two topics: /raw_image_data and /processed_image_data, which means that it can display both RGB and grayscale video. Please carefully review Figure 15 to see which nodes communicate with each other, and how information is being passed.

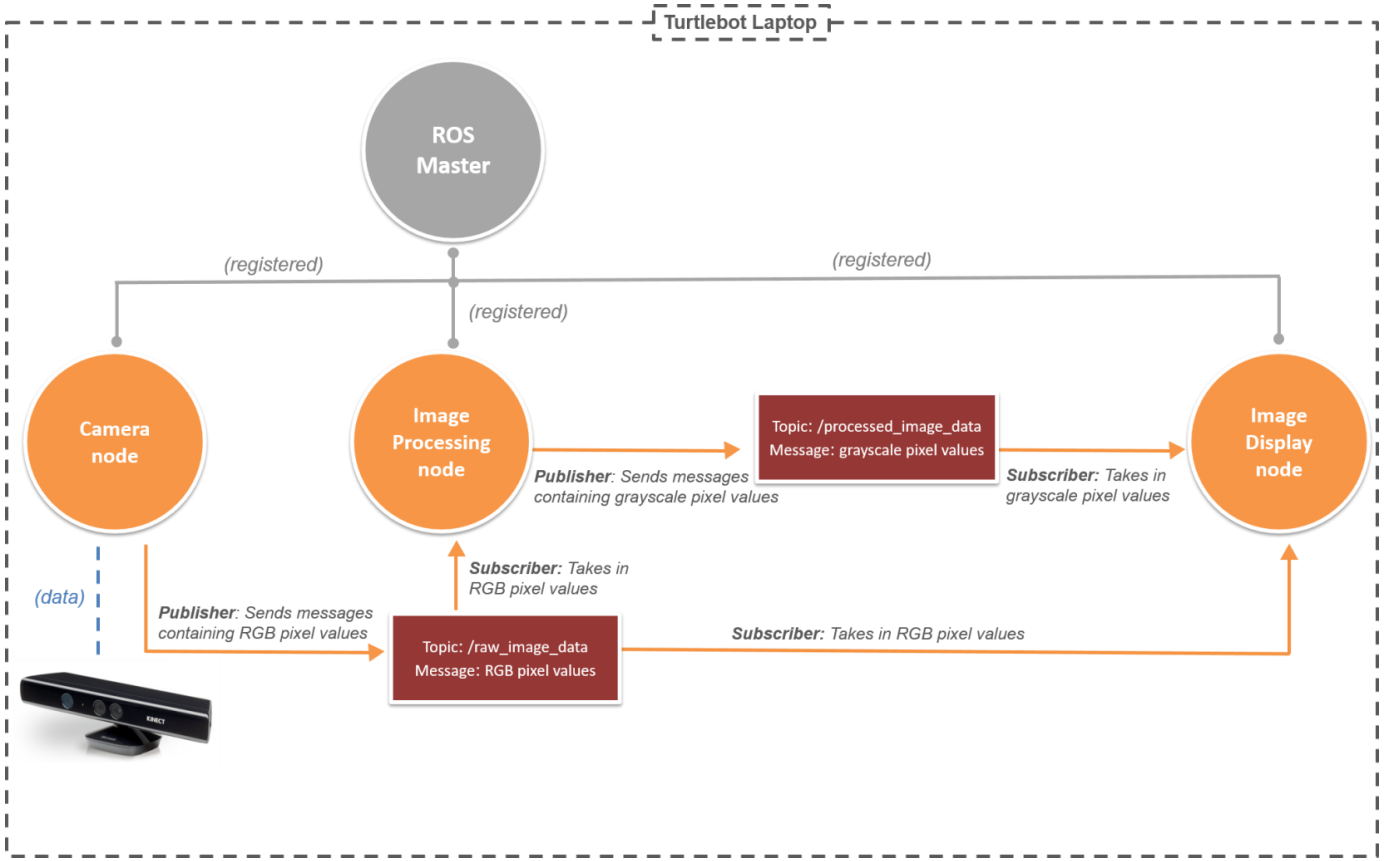


Figure 15 Example ROS publisher-subscriber for displaying and processing video data from a simulated Kinect sensor

5.3.5 Services

Publisher-subscribers are great to use for continuous data, for example, from motors driving the robot or a video feed from a simulated Kinect sensor. Sometimes, however, you may want information to be sent only *once*, and at a *specific instance of time*. In this case, you will use a ROS service. An example of a service would be a node that returns an answer to the question: "Is the ball in this image red?" The service will return a yes or no answer.

Services will not be covered in this course. If you would like to know more details about them, please refer to the corresponding ROS Wiki tutorials.

5.3.6 Packages & Launch files

ROS nodes, messages, publishers/subscribers, and services are grouped together within a package. Packages can also contain configuration files and datasets. A package is essentially a module. Each ROS package serves a specific function, and only contains the files needed to fulfil that function.

ROS packages may contain the following:

- CMake.txt file: CMake build file (for compiling your package)
- package.xml file: package manifest
- src/package-name/: source files
- include/package-name: header files
- msg/: folder containing message type (for publisher-subscriber)
- srv/: folder containing service type (for services)
- scripts/: folder containing executable scripts

Some ROS packages may also contain launch files. Roslaunch is used in the command line as follows:

```
roslaunch turtlebot_bringup minimal.launch
```

A launch file is an XML file which tells ROS to launch several nodes simultaneously, along with node configurations and parameters. For example, a launch files would allow for the launching of 10 nodes in one line, rather than opening 10 terminal windows to run each one individually. When launching nodes using the roslaunch command *it is not necessary* to already have a roscore running because the roslaunch structure automatically handles roscore. If interested, you can find more detailed information about launch files on the ROS roslaunch wiki. In this course, we will provide you with the necessary launch files. For more information refer to Section 7 Getting started on the TurtleBot 2.

Packages can be stand-alone or grouped into a *stack* if they pertain to a larger global function. Consider the image_pipeline stack (http://wiki.ros.org/image_pipeline). This stack is used to process raw images from a camera into a format useful for computer vision algorithms. Processing raw camera images may involve calibration, manipulation of both RGB and depth images, or simply visualizing the output. For this purpose, the image_pipeline stack contains a number of packages, as follows: camera_calibration, depth_image_proc, image_proc, image_rotate, image_view, and stereo_image_proc. Each package serves a specific function related to processing "raw images from a camera into a format useful for computer vision algorithms". Each package may contain its own set of nodes, and publish/subscribe to one or many topics. The package-stack relationship for the image_pipeline example is depicted in Figure 16.

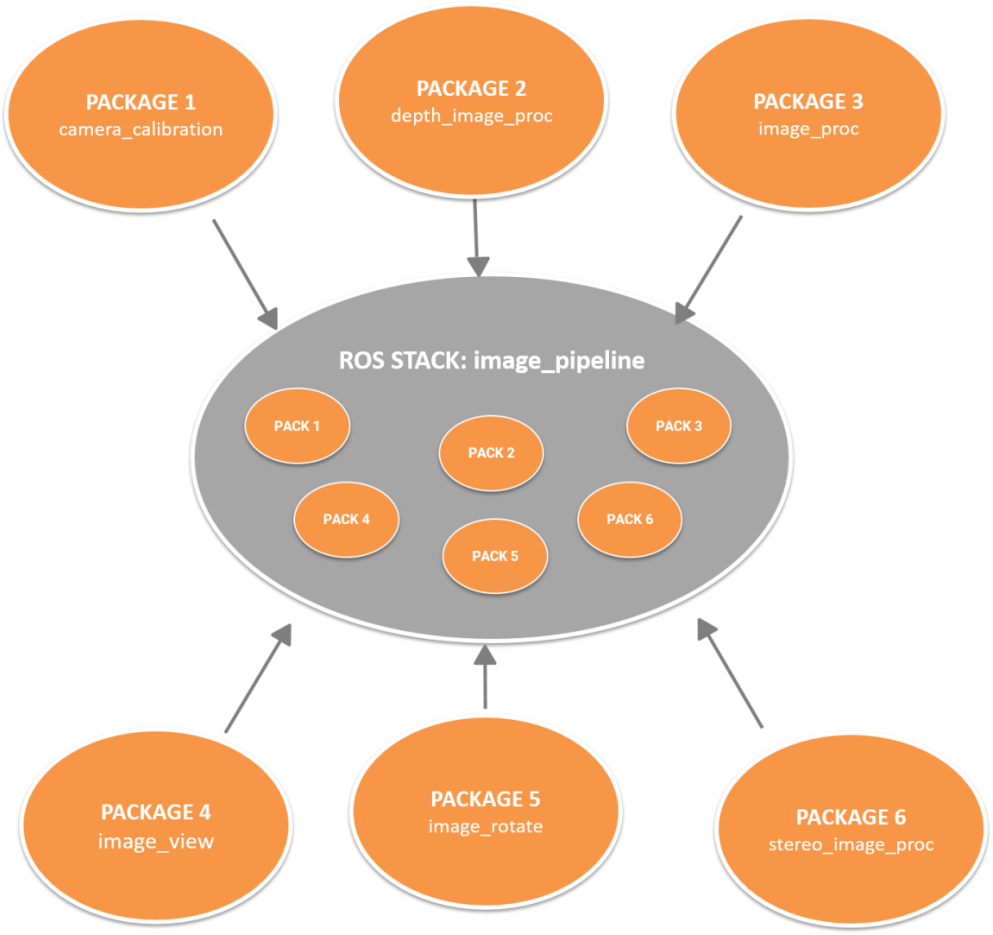


Figure 16 ROS Package Stack

5.4 Getting started with ROS

Now that you have a good overview of the core concepts behind ROS, it is time to start using ROS. The ROS wiki has great tutorials for you to review on your own.

For this course, it is strongly recommended that you go through the following ROS tutorials, found on the ROS wiki [4]:

- ☐ Creating a ROS Package
- ☐ Building a ROS Package
- ☐ Understanding ROS Nodes
- ☐ Understanding ROS Topics
- ☐ Understanding ROS Services and Parameters
- ☐ Using rqt_console and roslaunch
- ☐ Creating a ROS msg and srv
- ☐ Writing a Simple Publisher and Subscriber (C++)
- ☐ Examining the Simple Publisher and Subscriber
- ☐ Writing a Simple Service and Client (C++)
- ☐ Examining the Simple Service and Client
- ☐ Getting started with roswtf

Note: When navigating through the wiki website,the first several ROS tutorials give directions on how to set up your ROS environment and create a workspace. Environment variables have already been set, and a workspace has been created for you on your laptop (~/catkin_ws/) if you followed the steps in Appendix II. Therefore, you should follow the tutorials given in the list above, and only follow the first tutorials listed on the ROS wiki website.

5.5 An Introduction to Gazebo

ROS has a simulator called Gazebo which can be used to test algorithms and controllers, design robot modules, and implement various environments. Gazebo is a free 3D simulator which allows you (the developer) to create environments with physical properties to test your simulated robots in. For this course, Gazebo will be started through a file prepared for you. The TurtleBot Gazebo tutorial will be presented to you in a later chapter, though we encourage you to become familiar with the Gazebo GUI and functions through the following tutorial [5]:

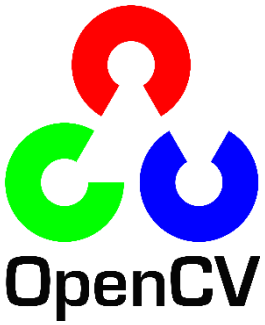
- ☐ Understanding the Gazebo GUI

5.6 An Introduction to Turtlebot Stage (Alternative to Gazebo)

Turtlebot Stage is a free light-weight 2D simulator that provides all the functionality needed to test 2D navigation capabilities. The Turtlebot Stage tutorial can be found on Quercus under >Files>Manual>Turtlebot_Stage_Tutorial 2020.pdf.

6 Introduction to OpenCV

6.1 What is OpenCV?



OpenCV is a free, open-source C++ library for image-processing and computer vision. It provides hundreds of machine-learning algorithms that can interface with both live video streams and static images. OpenCV was originally developed to accelerate the use of machine perception for commercial applications. The OpenCV person- and object-detection libraries are very commonly used, which you are encouraged to explore.

6.2 Why use OpenCV?

The main advantages of OpenCV are: (1) it is the fastest and most efficient image-processing library; (2) it is commonly used in industry; and (3) it contains a large database of algorithms ready for implementation.

6.3 Getting started with OpenCV

OpenCV is designed using modules, which can be included in your code according to your needs. The most common/important modules are summarized as follows:

core

This module is the core of OpenCV and provides a foundation for the other modules to build on. It contains data structures and some basic image processing functions. The first tutorial below will cover the Mat data structure used by OpenCV to store information about an image. The Mat structure is composed of two parts: (1) a matrix header, containing information about the matrix size, storage method, and address where the matrix is stored; and (2) a pointer to a matrix containing pixel data, which can vary by orders of magnitude depending on the image and data type.

highgui

Highgui controls everything related to user interfaces, for example, tracking mouse clicks and regulating the image window size. It handles low-level functions, such as loading an image, displaying video from a camera, and saving a modified image to a new file.

imgproc

This module provides algorithms for everything related to manipulating the visual aspects of an image. For example, it has libraries related to image filtering, converting an image from colour to grayscale, applying Gaussian blur, etc.

video

This module provides features specific to video streams. For instance, you can track an object across frames, subtract portions of a background, etc.

objdetect

This library includes algorithms related to object detection and recognition. For example, Objdetect provides the tools needed to do face detection and smile recognition using algorithms that apply machine learning techniques (e.g. Cascade Classifiers or Latent Support Vector Machines (SVM)).

OpenCV provides a detailed set of tutorials to help you understand its core concepts. The following tutorials are highly recommended [6]:

OpenCV core functionality

- ☐ Mat - The Basic Image Container
- ☐ How to scan images, lookup tables and time measurement with OpenCV
- ☐ Mask operations on matrices
- ☐ Adding (blending) two images using OpenCV
- ☐ Changing the contrast and brightness of an image
- ☐ Basic Drawing

Image-processing

- ☐ Smoothing Images
- ☐ Eroding and Dilating
- ☐ More Morphology Transformations
- ☐ Image Pyramids
- ☐ Basic Thresholding Operations
- ☐ Back Projection
- ☐ Finding contours in your image
- ☐ Creating Bounding boxes and circles for contours

2D Features Framework

- ☐ Feature Detection
- ☐ Feature Description
- ☐ Features2D + Homography to find a known object

6.4 **Interfacing ROS with OpenCV**

ROS has its own package for interfacing OpenCV with the TurtleBot applications, called cv-bridge. The ROS wiki provides a concise tutorial showing how to convert ROS images into OpenCV images, and vice versa. Please go step-by-step through the tutorial, ensuring you understand each line of code.

- ☐ ROS Tutorial: Converting between ROS images and OpenCV images (C++)

7 Getting started on the TurtleBot 2

7.1 Testing the TurtleBot

As a first step, the sensors and basic functionality of the TurtleBot should be tested. This will also help you become familiar with how ROS interfaces with the robot's hardware, detects state changes, and demonstrates some of the topics available.

Start the TurtleBot

Ensure the Kinect sensor and iClebo Kobuki base are both connected to the laptop via two USB ports and powered on. The first step is then to activate all the core functionalities of the TurtleBot and actuate the external sensors. To do so, open a Terminator terminal. Launch the minimal.launch node from the turtlebot_bringup package:

The turtlebot_bringup package launches the publishers and subscribers associated with the Kobuki sensors, Microsoft Kinect sensor, and TurtleBot navigation. It essentially sets up the robot to be "ready to go". Now that the TurtleBot publishers have been started, ROS can show you what information is being published to topics.

To understand how ROS interfaces with the TurtleBot's hardware components, the first part of this tutorial will show you how the content of topic messages changes when a sensor is activated. It will cover cliff sensors, bumpers, and the Kinect sensor.

Test the cliff sensors

The TurtleBot has three cliff sensors: one on the right, one in the front-centre, and one on the left. Cliff sensors are responsible for detecting altitude or height changes, for instance if the Kobuki encounters a staircase.

Information about the cliff sensors is published to the topic /mobile_base/events/cliff. To see information associated with a topic, the *rostopic* command can be used. Rostopic can display a list of active topics, the publishers/subscribers associated with a specified topic, the topic publishing rate, the topic bandwidth, and the messages published to it. To see the message being published to it in real time, we will use *rostopic echo.*

Open a third terminal using CTRL+SHIFT+T and enter the following command in the terminal:

```
rostopic echo /mobile_base/events/cliff
```

You should see the following output:

```
state: 0
sensor: 0
---------
state: 0
sensor: 1
---------
state: 0
sensor: 2
```

Now, lift the TurtleBot up from the floor and notice how the state values change:

```
state: 1
sensor: 0
---------
state: 1
sensor: 1
---------
state: 1
sensor: 2
```

The *state* will have a value of 0 if the Kobuki platform is on the floor, or 1 if it encounters a cliff. The sensor is either 0 (left), 1 (centre), or 2 (right). This is the message sent from the cliff sensor publisher to the /mobile_base/events/cliff topic. When writing your scripts for the course contests, you can subscribe to this topic and have the TurtleBot take some action when the state is 1.

End the rostopic echo event by typing in CTRL+C in its terminal window.

Test the Bumpers

Similar to the cliff sensors, the Kobuki has three bumpers: right, left, and centre. This time, run the rostopic echo command by reading messages sent to the /mobile_base/events/bumper topic in the terminal you previously used for the cliff sensors:

```
rostopic echo /mobile_base/events/bumper
```

You should see the following output:

```
state: 0
bumper: 0
---------
state: 0
bumper: 1
---------
state: 0
bumper: 2
```

Now, push on the centre bumper and notice how the value changes.

```
state: 0
bumper: 0
---------
state: 1
bumper: 1
---------
state: 0
bumper: 2
```

The state is given a value of 0 (released) or 1 (pressed), and the bumpers are numbered 0 (left), 1 (centre), and 2 (right). Test each bumper independently and then simultaneously to get a sense of how values change. Similar to the cliff sensors, you can subscribe to the /mobile_base/events/bumper topic to have the TurtleBot react to a change in the bumper state. Understanding the bumper events may be useful for the course contests.

End the rostopic echo event using CTRL+C in its terminal.

Using ROS to interface with the Kinect sensor

Now that you understand how the Kobuki interacts physically with its environment through the cliff and bumper sensors, it is time to move onto the Kinect sensor. The first step is to see the RGB and depth video output by the Kinect sensor.

Ensure your turtlebot_bringup minimal.launch node is still up and running. If not, launch it again as previously shown.

Open a new terminal and launch the TurtleBot 3dsensor nodes:

```
roslaunch turtlebot_bringup 3dsensor.launch
```

This launch file starts all the nodes for interfacing with the Kinect sensor, processing the video stream, and displaying the video data.

Once you see on the terminal that a device is connected, everything is running well. You can now see the Kinect camera's RGB video feed by running:

```
rosrun image_view image_view image:=/camera/rgb/image_color
```

The output is shown in Figure 17. The RGB camera's image data will be useful for recognizing objects using OpenCV. You will be asked to implement these features for Contests 2 and 3.

End the feed using CTRL+C.

The Kinect's depth sensor video feed can be visualized through grayscale images. The intensity of light at each pixel (where black is the lowest intensity and white is the highest intensity) represents its distance from the sensor. The darkest pixels represent objects furthest from the sensor, and the white pixels are objects closest to the sensor. There are 256 levels of intensity (i.e. shades of gray), with pixel intensity values ranging between [0, 255]. A representation of this is given in the right image of Figure 17.

To see the Kinect sensor's depth stream, type in a new terminal:

```
rosrun image_view image_view image:=/camera/depth_registered/image
```

Move your hand in front of the Kinect sensor and see how the pixel colour intensity changes. Notice the two topics used here: /camera/rgb/image_color and /camera/depth/image. You should be able to subscribe to both of these for the contests.

End the node using CTRL+C.

Rather than specifying the image_view topic directly in the command line, you can use RQT to visualize and select the image_view topics from a drop-down menu. RQT is a ROS GUI that can be used to visualize and control the output of various ROS packages. To do so, use the following command:

```
rqt_image_view
```

You will be able to select the desired image_view topics from the drop-down menu, and visualize the video feeds as in Figure 17. In this figure, the Kinect camera's RGB video stream is shown on the left, and its depth video stream is shown on the right.

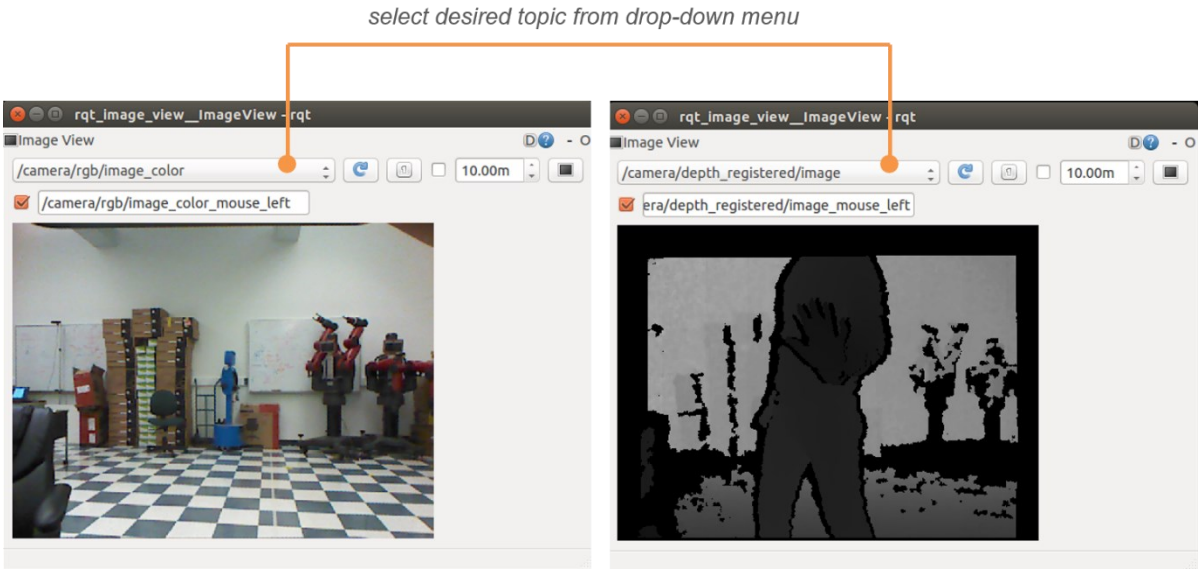


Figure 17 RQT GUI showing the video outputs from /camera/rgb/image_color (left) and /camera/depth_registered/image (right) topics selected from the drop-down menu.

7.2 Learning the ROS TurtleBot package

The ROS TurtleBot Wiki [7] provides a great step-by-step learning guide. You are encouraged to go through the tutorials in Sections 4.1 to 4.3 in the Wiki. This will teach you how to control your robot using the keyboard, create a map of your environment, and navigate autonomously to a location on the map.

ROS TurtleBot Tutorials

**All following tutorials were originally developed for ROS Indigo and have not been updated to reflect Kinetic. However, if you replace the word "indigo" with the word "kinetic" where they appear in the tutorials, they function identically.

- ☐ A First Interaction
- ☐ Visualisation
- ☐ 3D Visualisation
- ☐ Teleop
- ☐ SLAM map building with TurtleBot
- ☐ Autonomous Navigation of a Known Map with TurtleBot

The TurtleBot Simulator (Gazebo)

Gazebo is a simulator created in partnership with ROS. When launched, the simulator will subscribe to all the topics published by your TurtleBot nodes. This concept is best illustrated by way of an example.

Open a new terminal window using CTRL+SHIFT+T and start the TurtleBot Gazebo simulator:

```
roslaunch turtlebot_gazebo turtlebot_world.launch
```

At first the simulator may appear blank: it can take a few minutes for the Gazebo simulator to load. Once it has been loaded, you should see the simulated TurtleBot in an environment containing different objects made from different materials, such as shown in Figure 18. The screen view may be different (e.g., top view). Pressing the SHIFT key allows you to rotate the view field in 3D.

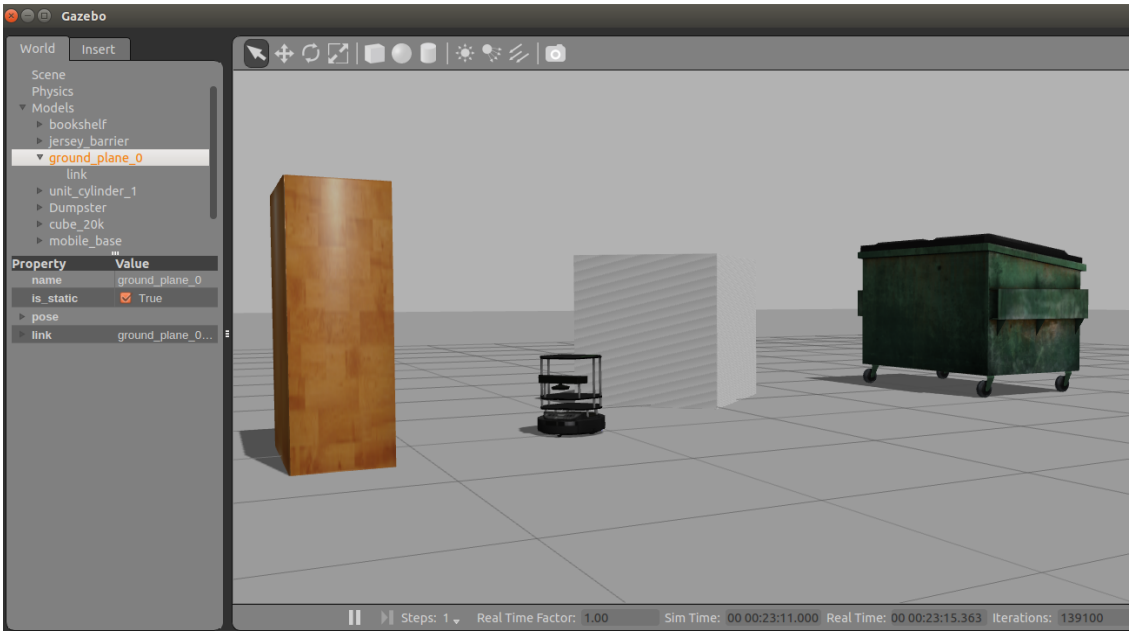


Figure 18 TurtleBot Gazebo Simulator on Initialization

At this point, the TurtleBot simply exist within the simulator, and is waiting for input. Start the keyboard teleop node using the command below and as shown in Figure 19.

```
roslaunch turtlebot_teleop keyboard_teleop.launch
```

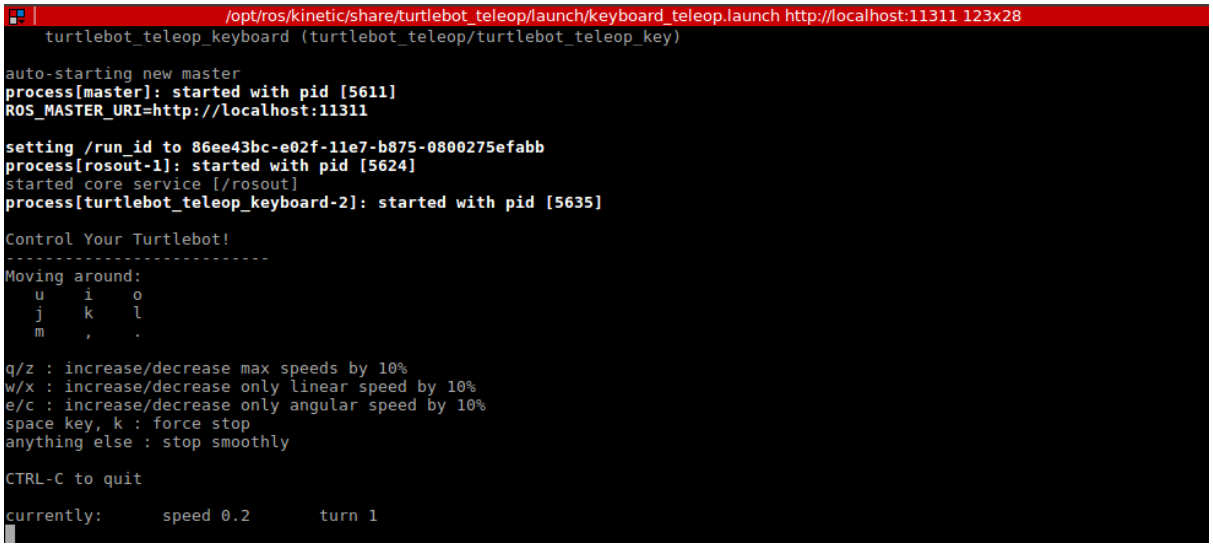


Figure 19 Terminal output after launching the turtlebot_teleop package

With a setup similar to Figure 20, click into the turtlebot_teleop terminal window such that the window is now active. The ***u i o j k l m , .*** keyboard keys are used to move your TurtleBot in the Gazebo simulator. For example, the '**i**', '**k**', and '**,**' keys are for moving forward, stopping, and moving backwards, respectively. The other keys are for turning. Type the desired commands into the turtlebot_teleop terminal window to move your simulated TurtleBot. In the background, you will see the TurtleBot and its environment displayed from an aerial perspective, navigating around according to the keyboard commands given.

The keyboard_teleop.launch file launches a number of nodes publishing to topics for controlling the TurtleBot with the keyboard. The Gazebo simulator subscribes to all those topics, and moves the TurtleBot the same way.

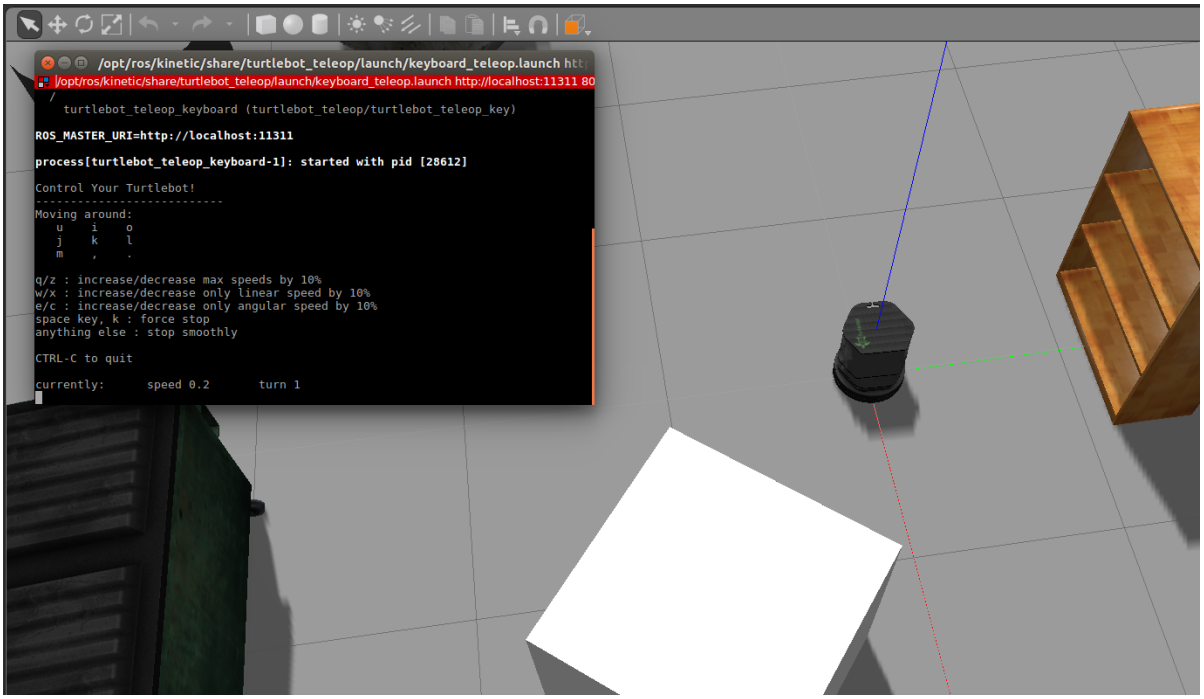


Figure 20 Setup for teleoperation of the TurtleBot Gazebo simulator with the terminal window active

Test the Bumpers

The simulated TurtleBot has three bumpers: right, left, and centre. In a new terminal run the rostopic echo command by reading messages sent to the /mobile_base/events/bumper topic to see the bumper's state:

```
rostopic echo /mobile_base/events/bumper
```

Now, using the teleop terminal window, navigate the simulated TurtleBot into an object to push on the front centre bumper and notice how the value changes.

```
state: 0
bumper: 0
---------
state: 1
bumper: 1
---------
state: 0
bumper: 2
```

The state is given a value of 0 (released) or 1 (pressed), and the bumpers are numbered 0 (left), 1 (centre), and 2 (right). Test each bumper independently and then simultaneously to get a sense of how values change. You can subscribe to the /mobile_base/events/bumper topic to have the TurtleBot react to a change in the bumper state. Understanding the bumper events may be useful for the course contests.

End the rostopic echo event using CTRL+C in its terminal.

Test the Odometry

The simulated TurtleBot has odometry estimates the simulated robot's relative position and orientation (pose). The relative pose is the robot's estimated location in space relative to its initial position at the begininning of the simulation, where the world frame is defined. In a new terminal run the echo command by reading messages sent to the /odom topic to see the state of odometry:

```
rostopic echo /odom
```

The terminal will immediately begin reporting the current odometry as shown below:
Now, using the teleop terminal window, navigate the simulated TurtleBot throughout the environment to see how the odometry changes with the robot's pose.

```
header:
  seq: 6145182
  stamp:
    secs: 172818
    nsecs: 263000000
  frame_id: "odom"
child_frame_id: "base_footprint"
pose:
  pose:
    position:
      x: 1.84865241489
      y: 0.838540891619
      z: 0.0
    orientation:
      x: 0.0
      y: 0.0
      z: -0.74986047553
      w: 0.661596000016
  covariance: [0.1, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.1, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1000000.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 1000000.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1000000.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.05]
twist:
  twist:
    linear:
      x: 0.000264665710346
      y: 0.0
      z: 0.0
    angular:
      x: 0.0
      y: 0.0
      z: 0.000516075974325
  covariance: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
```

End the rostopic echo event using CTRL+C in its terminal.

Viewing images from the simulated Kinect Camera

Now that you understand how the Kobuki interacts physically with the simulated environment through the bumper sensors and odometry, it is time to move onto the simulated Kinect sensor. The first step is to see the RGB and depth video output by the Kinect sensor.

Ensure that the Gazebo simulator node is still up and running. If not, launch it again as previously shown.

With Gazebo open, you can now see the Kinect camera's RGB video feed by running the following in a new terminal window:

```
rosrun image_view image_view image:=/camera/rgb/image_color
```

The output is shown in Figure 21. The RGB camera's image data will be useful for recognizing objects using OpenCV. You will be asked to implement these features for Contest 2.

End the feed using CTRL+C.

The Kinect's depth sensor video feed can be visualized through grayscale images. The intensity of light at each pixel (where black is the lowest intensity and white is the highest intensity) represents its distance from the sensor. The darkest pixels represent objects furthest from the sensor, and the white pixels are objects closest to the sensor. There are 256 levels of intensity, with pixel intensity values ranging between [0, 255]. A representation of this is given in the right image of Figure 21.

To see the Kinect sensor's depth stream, type in a new terminal:

```
rosrun image_view image_view image:=/camera/depth_registered/image
```

Use the keyboard teleop to change the view of the scene infront of the sensor sensor and see how the pixel colour intensity changes. Notice the two topics used here: /camera/rgb/image_color and /camera/depth/image. You should be able to subscribe to both of these for the contests.

End the node using CTRL+C.

Rather than specifying the image_view topic directly in the command line, you can use RQT to visualize and select the image_view topics from a drop-down menu. RQT is a ROS GUI that can be used to visualize and control the output of various ROS packages. To do so, use the following command:

```
rqt_image_view
```

You will be able to select the desired image_view topics from the drop-down menu, and visualize the video feeds as in Figure 21. In this figure, the simulated Kinect camera's RGB video stream is shown on the left, and its depth video stream is shown on the right.

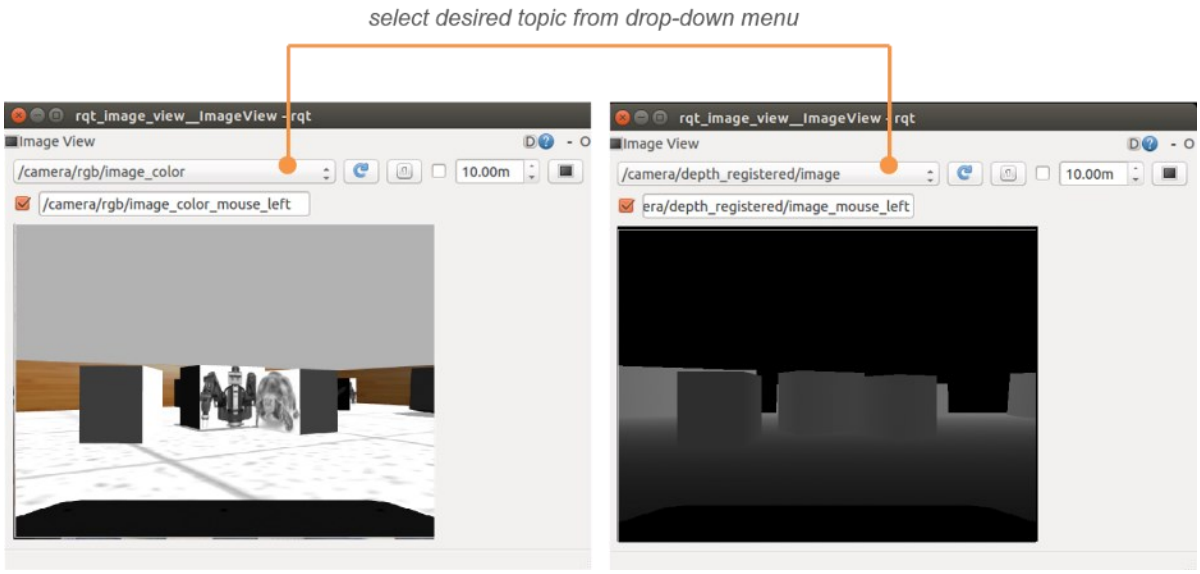


Figure 21 RQT GUI showing the video outputs from /camera/rgb/image_color (left) and /camera/depth_registered/image (right) topics selected from the drop-down menu.

Creating a Map of the Environment

The next step in learning to use the simulator is to create a map of the environment, and have the TurtleBot navigate autonomously within the environment. With the keyboard teleop node still running, start a new terminal by pressing CTRL+SHIFT+T.

In order for the robot to autonomously navigate in an environment, it makes use of three functions: (1) mapping; (2) localization; and (3) planning. A map is a robot's representation of the environment. The TurtleBot uses depth data from the Kinect sensor to construct a map of the environment, as it navigates around, using the GMapping package provided by ROS. GMapping is a 2D SLAM (Simultaneous Localization and Mapping) package that uses the odometry to determine the robot's relative pose and depth data from the Kinect sensor to create a 2D map of the environment by storing distances of walls and obstacles from this pose. The final 2D map is composed of an image showing a blueprint of the

environment, and a configuration file (yaml) that gives meta information about the map (i.e. absolute origin of the map).

To construct a map of the simulated environment, open a new terminal. Ensure your teleop node is still running. Launch the gmapping_demo node using the following command:

```
roslaunch turtlebot_gazebo gmapping_demo.launch
```

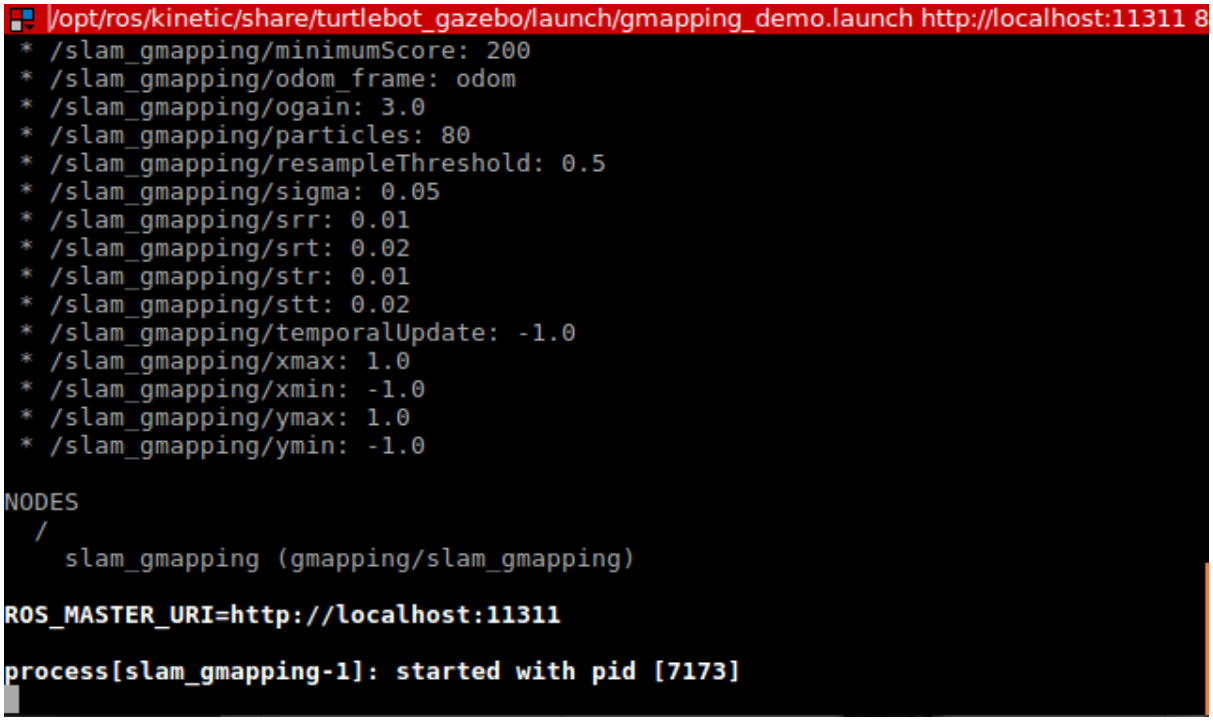


Figure 22 Terminal output after launching gmapping_demo

In the terminal, ensure you see the last line shown in Figure 22, “Registering First Scan”. At this point, your TurtleBot is recording depth data for its map. You will want to visually see the map as it is being recorded, in order to ensure you have covered all areas of the environment. RViz (ROS visualization) is a great tool for this purpose. It displays sensor data and state information from ROS.

You can use RViz to visualize the mapping process:

```
roslaunch turtlebot_rviz_launchers view_navigation.launch
```

You should see a gray grid. Divide your laptop screen in such a way that you can clearly see both the Gazebo and RViz windows, and open the turtlebot_teleop terminal window so it is active. A suggested setup is shown in Figure 23. Initially, you should see something similar to RViz in the top image. Then,

use the keyboard to navigate your simulated TurtleBot around the Gazebo environment as you did previously. As you move around the environment, you will notice the map being created in RViz (bottom image of Figure 23).

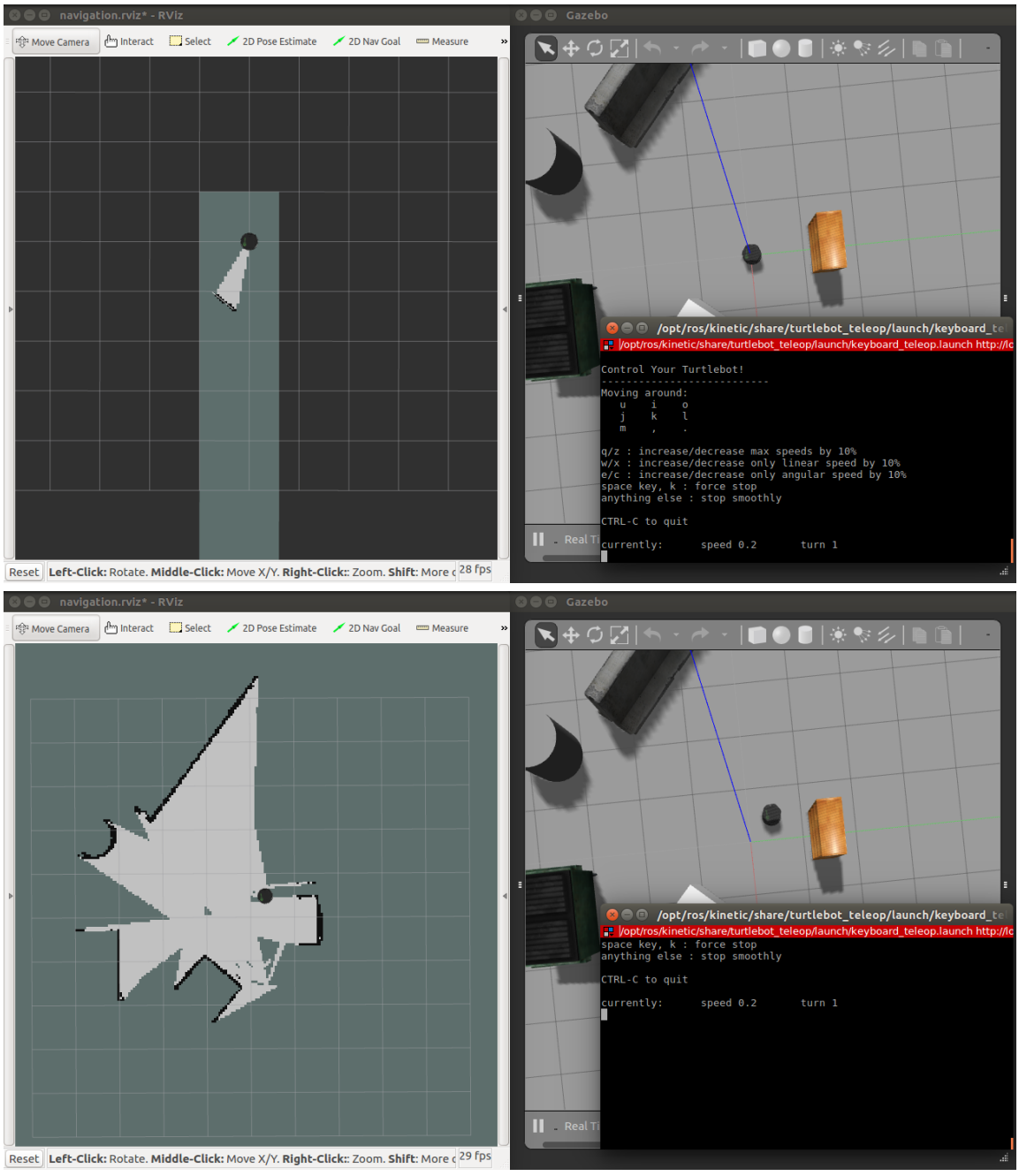


Figure 23 Simulated mapping process showing RViz (left), Gazebo (right), and Teleoperation terminal open (bottom right). Top: Initial gmapping. Bottom: Finalized map (note the changes in the map).

Once the TurtleBot has explored the whole environment, save your generated map using the following command. Replace <your_map_name> with a name for your map, e.g. my_map.

```
rosrun map_server map_saver -f <your_map_name>
```

This command will save a jpg image of your map along with the corresponding yaml data file. The map is correctly saved when you see the output in Figure 24. To ensure everything works as expected, kill all currently running nodes using CTRL+C. Ensure both Gazebo and RViz are also closed.

Now, repeat the steps that have the orange exclamation mark icon on the left hand side (). Instead of running the gmapping node, load your map and run the ROS amcl node by typing the following into a terminal:

```
roslaunch turtlebot_gazebo amcl_demo.launch map_file:=<path to YAML file>
```

Note that <path to YAML file> should be replaced by the path to your own map (e.g. /home/turtlebot/my_map.yaml).

Amcl (Adaptive Monte Carlo Localization) is an algorithm that allows the robot to estimate its location within a map in real time. RViz will show you where the robot estimates it is, along with a group of green arrows showing its orientation. To ensure that the initial position and orientation (pose) of the TurtleBot is correct, use the 2D Pose Estimate button on the top taskbar and click on the map to give the robot pose. Figure 25 shows where the robot initially estimated it was and Figure 26 shows the corrected version based on where the TurtleBot actually was in the Gazebo simulator after the above steps.

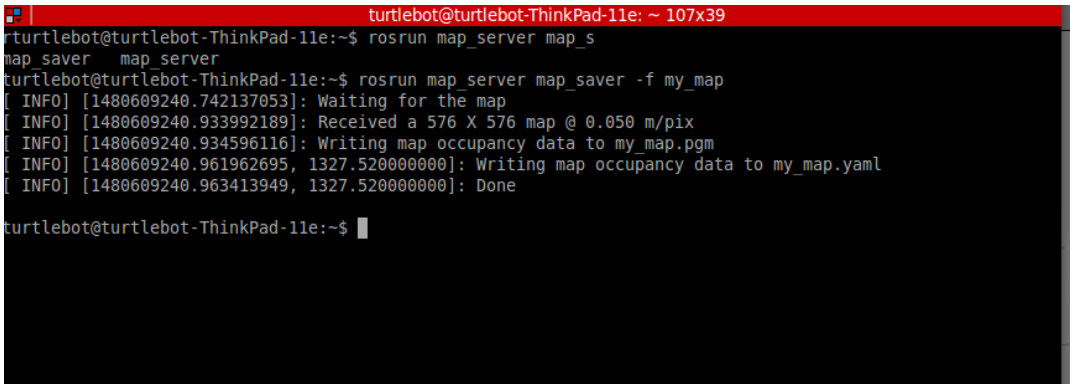


Figure 24 Save your map to the directory you are in. In the case shown in this figure it is the home directory (/home/turtlebot)

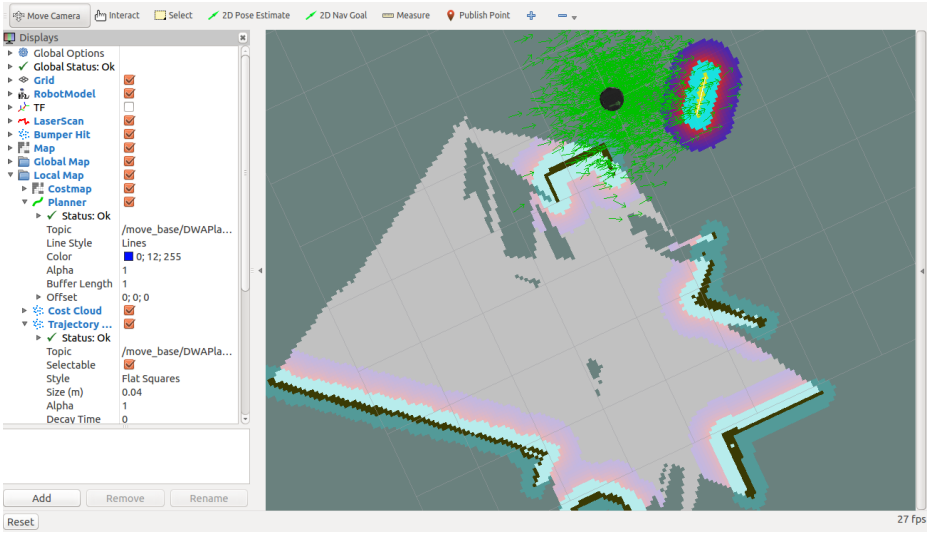


Figure 25 Original (incorrect) estimated position in RViz

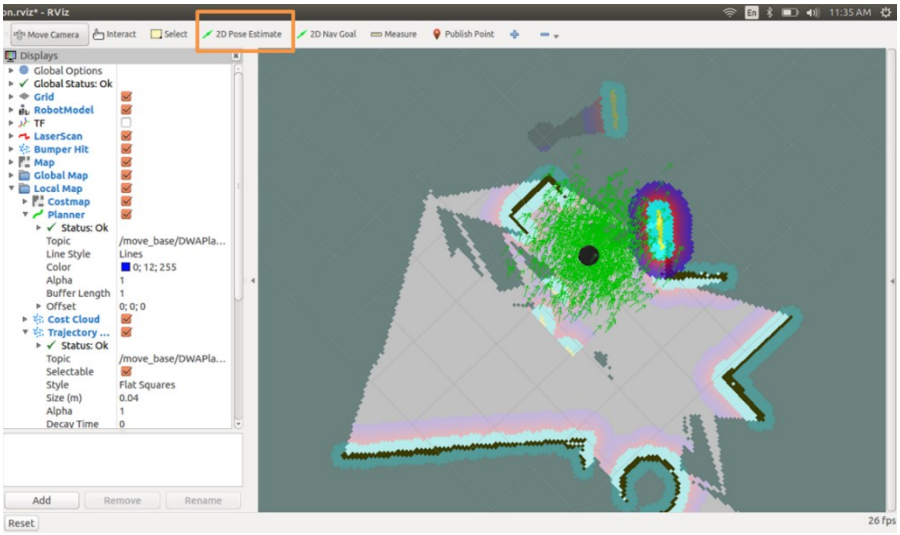


Figure 26 Corrected TurtleBot initial position and orientation in RViz using 2D Pose Estimate

Now, you are ready to send the simulated TurtleBot to any location in the map. In RViz, click on the "2D Nav Goal" button on the top taskbar. Click on the map where you want the TurtleBot to drive to and drag in the direction the TurtleBot should be oriented in. When it is ready to go, the TurtleBot will compute an optimal path, shown by the (discrete) red line in Figure 27. Watch it navigate there autonomously!

Ensure you understand each step above – it will be important for the MIE443 contests.

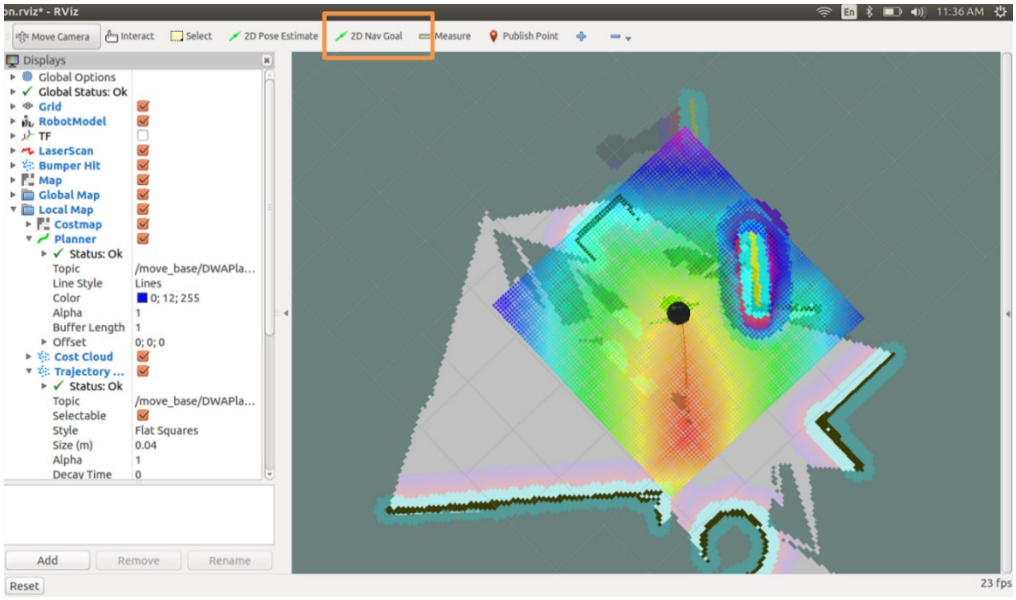


Figure 27 TurtleBot navigating autonomously in RViz. The red line shows its predicted path.

For more information and tutorials, please see the ROS TurtleBot Gazebo tutorials:

- ☐ Gazebo Bringup Guide
- ☐ Explore the Gazebo world
- ☐ Make a map and navigate with it

A. Appendix I: Tutorial Checklist

A.1.1 ROS Tutorials

ROS core concepts and exercises

- ☐ Creating a ROS Package
- ☐ Building a ROS Package
- ☐ Understanding ROS Nodes
- ☐ Understanding ROS Topics
- ☐ Understanding ROS Services and Parameters
- ☐ Using rqt_console and roslaunch
- ☐ Creating a ROS msg and srv
- ☐ Writing a Simple Publisher and Subscriber (C++)
- ☐ Examining the Simple Publisher and Subscriber
- ☐ Writing a Simple Service and Client (C++)
- ☐ Examining the Simple Service and Client
- ☐ Getting started with roswtf

Introduction to Gazebo Simulator

- ☐ Understanding the Gazebo GUI

A.1.2 OpenCV Tutorials

OpenCV core functionality

- ☐ Mat - The Basic Image Container
- ☐ How to scan images, lookup tables and time measurement with OpenCV
- ☐ Mask operations on matrices
- ☐ Adding (blending) two images using OpenCV
- ☐ Changing the contrast and brightness of an image
- ☐ Basic Drawing

Image-processing

- ☐ Smoothing Images
- ☐ Eroding and Dilating
- ☐ More Morphology Transformations
- ☐ Image Pyramids
- ☐ Basic Thresholding Operations
- ☐ Back Projection
- ☐ Finding contours in your image
- ☐ Creating Bounding boxes and circles for contours

2D Features Framework

- ☐ Feature Detection
- ☐ Feature Description
- ☐ Features2D + Homography to find a known object

Integrating ROS & OpenCV

- ☐ ROS Tutorial: Converting between ROS images and OpenCV images (C++)

A.1.3 TurtleBot Tutorials

**Please note that all tutorials in this section were originally developed for ROS Indigo. However, if you replace the word “indigo” with the word “kinetic” where they appear in the tutorials, they function exactly the same.

- ☐ Gazebo Bringup Guide
- ☐ Explore the Gazebo world
- ☐ Make a map and navigate with it

A.1.4 PyTorch Tutorials

Deep Learning Information

- ☐ The deep learning book

PyTorch Tutorials

- ☐ General tutorials page
- ☐ Intro to PyTorch
- ☐ Pytorch through examples
- ☐ What is PyTorch
- ☐ PyTorch Neural Networks
- ☐ Pytorch nn module
- ☐ Visualization
- ☐ Transfer learning

B. Appendix II: Installing Ubuntu 16.04 on your personal computer

To install Ubuntu on your own personal computer, you may consider the following options: (1) partition your drive; or (2) use a virtual machine. **Please note that you are doing this at your own risk. Do not attempt to "Dual-Boot" your computer with Ubuntu if your computer is a Microsoft surface laptop, or a macbook, both models have well known issues preventing them from working with Ubuntu directly.**

B.1.1 Partitioning your drive

The instructions for partitioning your drive can be found online. While partitioning is relatively common and easy process, **it can also be very risky if done incorrectly**. It is possible to lose everything on your computer, including your entire OS. Should you consider to do so, we recommend reading through several tutorials before starting and ensuring they provide you with the necessary information to create a solid backup of your OS and files in case they are lost in the process.

B.1.2 Installing a Virtual Machine on your personal computer

This step-by-step guide assumes you are using Windows 10. Should you be running another version of Windows, ensure you download the correct packages for your operating system. Steps are similar.

To follow this guide, you will need the following packages:

- Oracle VirtualBox
- Ubuntu
- VirtualBox guest additions

Important! The directions given below are suggestive and meant to be used as a guideline. Not all steps have been fully tested, and are referenced from various online sources [9], [10]**. Use your own judgement if you encounter an issue not explicitly discussed here.**

Step 1: Determine your system's type (Windows)

You must know whether your computer is a 32-bit, or 64-bit. If you are unsure, go to System and click on the About section, as shown in Figure 28. This tab will show you technical specifications about your computer's system. In this case, the computer is 64-bit. Take note of which system type you are using.

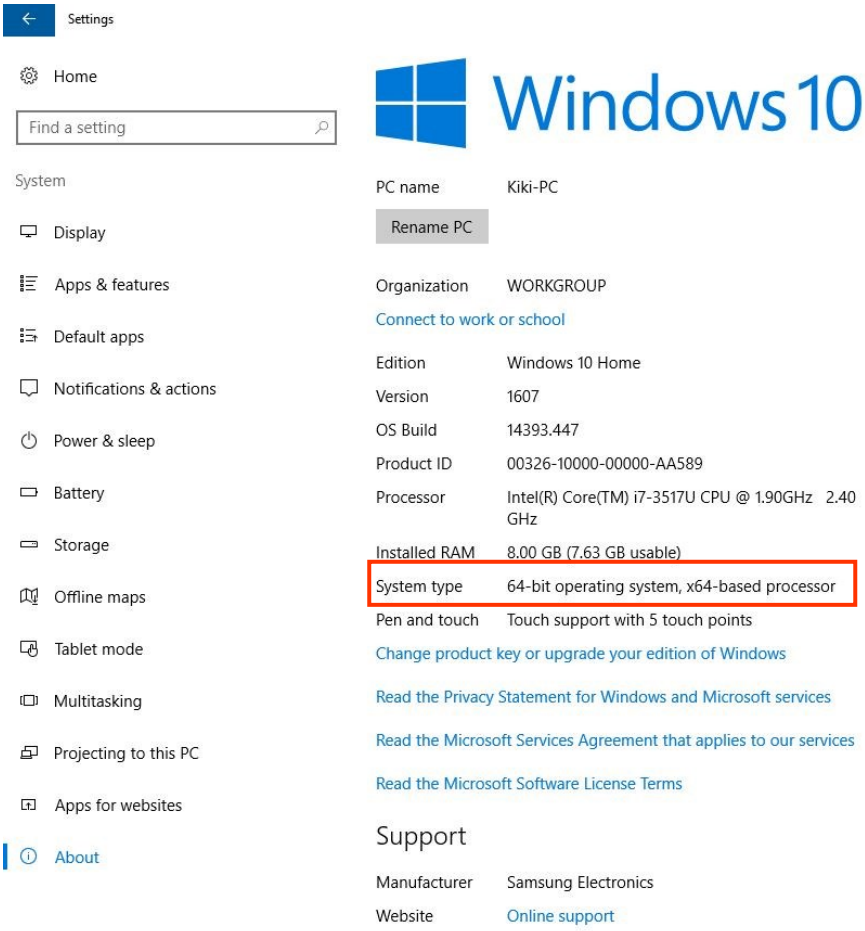


Figure 28 Computer system specifications showing 64-bit OS

Step 2. Download Ubuntu

Next, download the Ubuntu torrent. Ensure you are downloading Ubuntu 16.04 LTS edition. **DO NOT INSTALL ANYTHING**.

You can get the Ubuntu 32-bit ISO here or the Ubuntu 64-bit ISO here, according to your system specifications.

Step 3: Install Oracle VirtualBox

Oracle VirtualBox is like a second (virtual) computer. Rather than partitioning your drive, you can use VirtualBox to run a virtual computer operating under Ubuntu.

Go to www.virtualbox.org [8] and click on the large centre button (see Figure 29). The link will bring you to a page listing the different downloadable files.

Figure 29 virtualbox.org download page

For Windows users:

You want the Windows hosts (get the executable here), under the first bullet point for **VirtualBox 7.0** platform packages as shown in Figure 30.

For Mac OSX users:

Download the OS X hosts package, which is the second bullet point under *VirtualBox 7.0.4 platform packages* for intel platforms. The executable is directly provided to you at this link. Note that the directions given below apply for both Windows and Mac users, though the UI for the VirtualBox interface will look slightly different on the Mac. All buttons should be labelled identically, however. There is not an stable release yet for Apple Silicon Macbooks (M1 and M2) but you can use the developer preview at your own risk from the same web page or at this link.

Figure 30 Downloadable VirtualBox 7.0.4 binaries for Windows

Step 4. Download the VirtualBox Guest Additions

Guest Additions improve fullscreen resolution, amongst other things. In particular, this will be important for you while running simulations. You can get the VirtualBox Guest Additions version 7.0.4 from the link here, or by going to http://download.virtualbox.org/virtualbox/7.0.4/ . Download the file named "VBoxGuestAdditions_7.0.4.iso", highlighted in Figure 31.

Step 5. Install VirtualBox

Navigate to the directory where your VirtualBox (VB) executable ".exe" file has been downloaded (default: /Downloads folder). Double-click on it to launch the VB Installation Wizard, shown in Figure 32.

Click "Next", and choose the default installation settings. Decide whether you want a shortcut. Finally, you will be given the option to Install VirtualBox, and halfway through the installation, you will be asked whether you want to install the Oracle Universal Serial Bus software. Install both.

Step 6. Create a Ubuntu Virtual Machine

6.1 Start VirtualBox

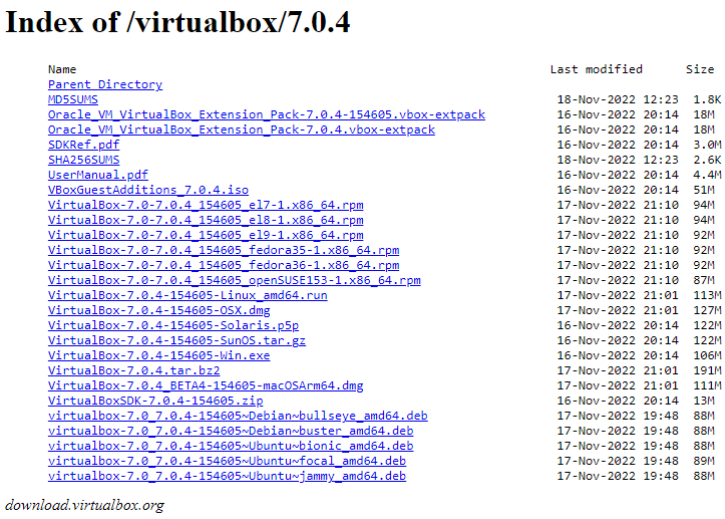
Start VirtualBox either by checking the "Start Oracle VM VirtualBox after installation" or from your launcher.

6.2 Create a new virtual machine

When the VirtualBox window opens, click on the blue "New" button on the top left corner (Figure 33). This will allow you to create a new virtual machine.

VB will prompt you to name and choose your operating system (Figure 34). Enter/choose the following options:

Name: Ubuntu 16.04
Type: Linux
Version: Ubuntu (32-bit or 64-bit, according to your system specs in Step 1).



Index of /virtualbox/7.0.4

Name	Last modified	Size
Parent Directory		
MD5SUMS	18-Nov-2022 12:23	1.8K
Oracle_VM_VirtualBox_Extension_Pack-7.0.4-154605.vbox-extpack	16-Nov-2022 20:14	18M
Oracle_VM_VirtualBox_Extension_Pack-7.0.4.vbox-extpack	16-Nov-2022 20:14	18M
SDKRef.pdf	16-Nov-2022 20:14	3.0M
SHA256SUMS	18-Nov-2022 12:23	2.6K
UserManual.pdf	16-Nov-2022 20:14	4.4M
VBoxGuestAdditions_7.0.4.iso	16-Nov-2022 20:14	51M
VirtualBox-7.0-7.0.4_154605_el7-1.x86_64.rpm	17-Nov-2022 21:10	94M
VirtualBox-7.0-7.0.4_154605_el8-1.x86_64.rpm	17-Nov-2022 21:10	94M
VirtualBox-7.0-7.0.4_154605_el9-1.x86_64.rpm	17-Nov-2022 21:10	92M
VirtualBox-7.0-7.0.4_154605_fedora35-1.x86_64.rpm	17-Nov-2022 21:10	92M
VirtualBox-7.0-7.0.4_154605_fedora36-1.x86_64.rpm	17-Nov-2022 21:10	92M
VirtualBox-7.0-7.0.4_154605_openSUSE153-1.x86_64.rpm	17-Nov-2022 21:10	87M
VirtualBox-7.0.4-154605-Linux_amd64.run	17-Nov-2022 21:01	113M
VirtualBox-7.0.4-154605-OSX.dmg	17-Nov-2022 21:01	127M
VirtualBox-7.0.4-154605-Solaris.p5p	16-Nov-2022 20:14	122M
VirtualBox-7.0.4-154605-SunOS.tar.gz	16-Nov-2022 20:14	122M
VirtualBox-7.0.4-154605-Win.exe	16-Nov-2022 20:14	106M
VirtualBox-7.0.4.tar.bz2	17-Nov-2022 21:01	191M
VirtualBox-7.0.4_BETA4-154605-macOSArm64.dmg	17-Nov-2022 21:01	111M
VirtualBoxSDK-7.0.4-154605.zip	16-Nov-2022 20:14	13M
virtualbox-7.0_7.0.4-154605~Debian~bullseye_amd64.deb	17-Nov-2022 19:48	88M
virtualbox-7.0_7.0.4-154605~Debian~buster_amd64.deb	17-Nov-2022 19:48	88M
virtualbox-7.0_7.0.4-154605~Ubuntu~bionic_amd64.deb	17-Nov-2022 19:48	88M
virtualbox-7.0_7.0.4-154605~Ubuntu~focal_amd64.deb	17-Nov-2022 19:48	89M
virtualbox-7.0_7.0.4-154605~Ubuntu~jammy_amd64.deb	17-Nov-2022 19:48	88M

download.virtualbox.org

Figure 31 Download the VBoxGuestAdditions 7.0.4 ISO file

Figure 32 Launch the VirtualBox Installation Wizard

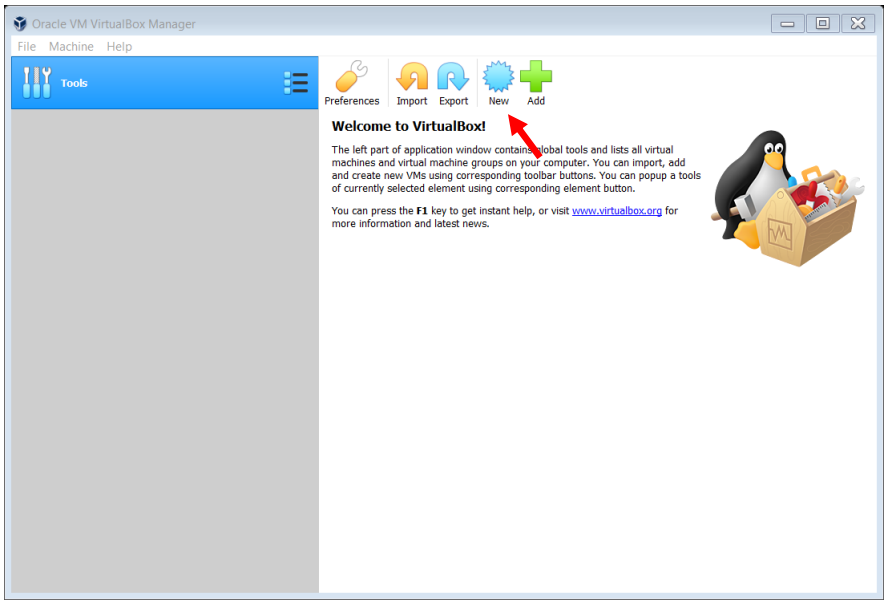


Figure 33 Launch VirtualBox and create a new Virtual Machine

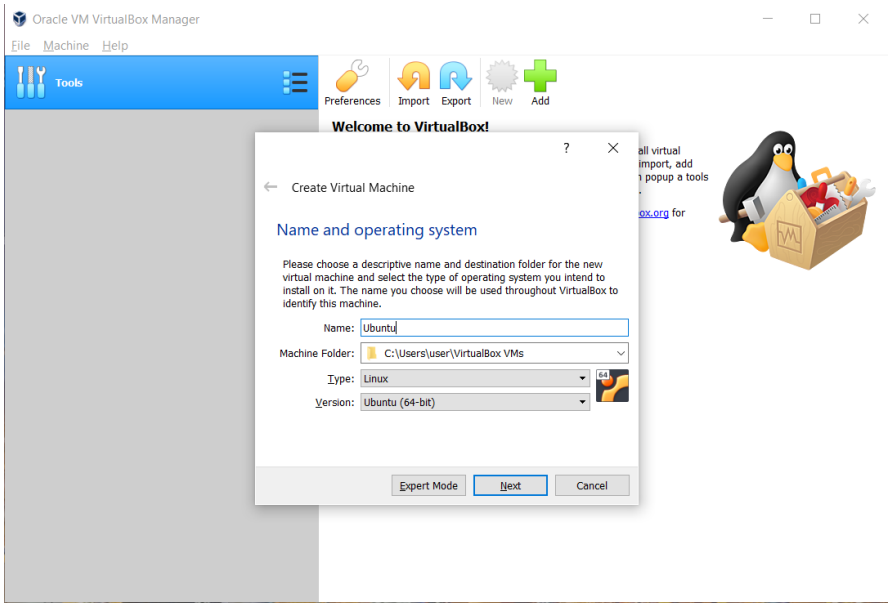


Figure 34 Name and select the operating system for your Virtual Machine

6.3 Allocate memory to your virtual machine

Next, you will be prompted to select the amount of RAM you want to allocate to the VM. This is only used when the VM is running. The choice here is entirely up to you, but keep in mind that you will be required to run computationally-intensive simulators, sometimes two at once. Consider at least 4GB of RAM (Figure 35).

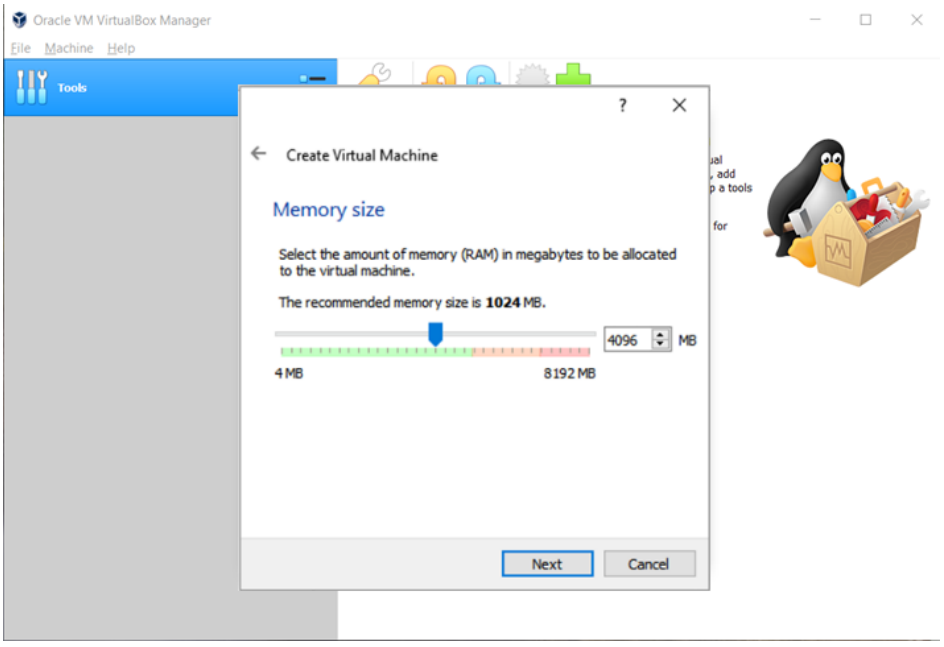


Figure 35 Allocate at least 4 GB of RAM to your Virtual Machine

6.4 Create a VM hard disk

On the next step, select “Create a virtual hard disk now” and click “Create”.

Choose the “VDI” hard drive option (Figure 36), and “Fixed size” for hard disk space. (Setting a predefined fixed amount of space will increase performance). For the amount of space to allocate, that depends on you. We recommend 20 GB due to the number of packages you will have to install, but this is up to you and what is available on your computer.

Click “Create”.

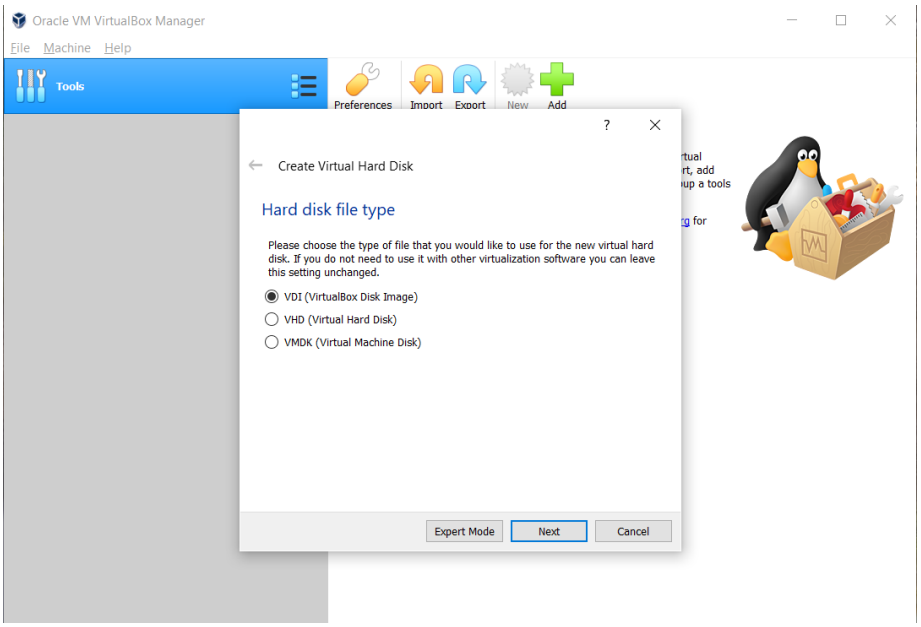


Figure 36 Select “VirtualBox Disk Image” as your Virtual Machine OS

Step 7. Install Ubuntu on your Virtual Machine

At this point, your Virtual Machine has been created, but it has no operating system. You have to boot into Ubuntu using the .iso file you downloaded earlier.

Under the window “Select start-up disk”, click on the folder icon beside the Host Disk drop-down (Figure 37), navigate to the directory where your Ubuntu 16.04 .iso file was downloaded, and click “Open”.

Next, click on the “Start” button (green arrow on top VB bar). You will see the Ubuntu Installer come up. It will give you the option of either trying out Ubuntu, or installing it (Figure 38). Choose the install option.

Ensure your computer meets all the requirements, namely:

- Minimum 6.6. GB dedicated disk space
- Laptop is charged (plug it in)
- Internet connection

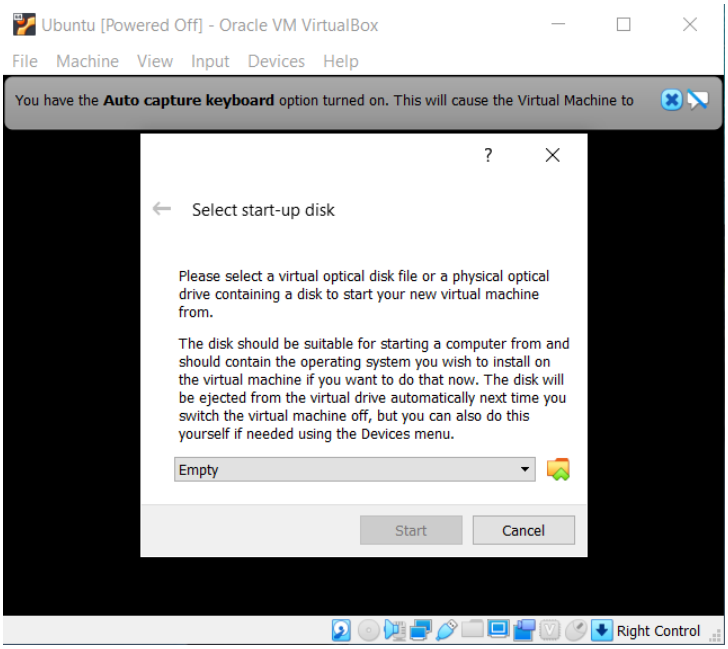


Figure 37 Select your Ubuntu ISO by clicking on the folder

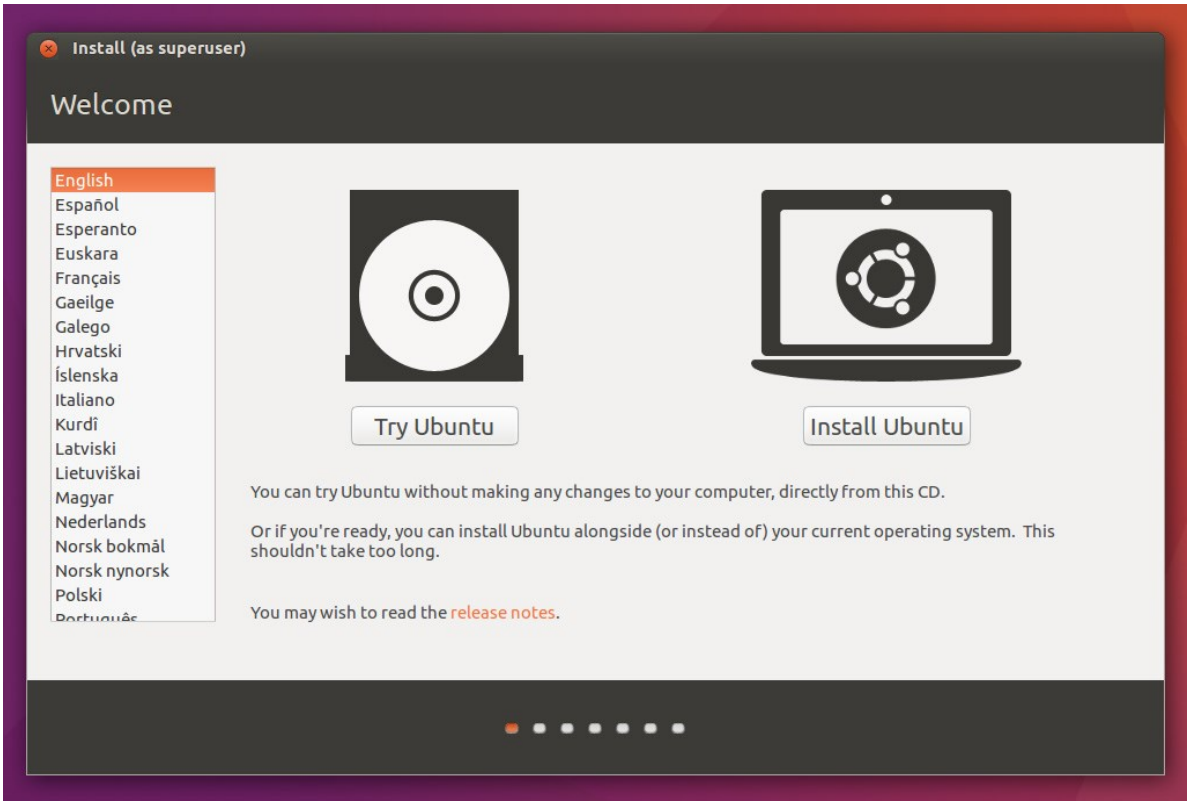


Figure 38 Options to try or install Ubuntu on start-up

If your internet connection is poor, uncheck the option to install updates simultaneously. You will be prompted to install them at a later time.

Next, you will be asked to choose an installation type. Since you are using this in a Virtual Machine, choose “Erase disk and Install Ubuntu”. This will not erase your computer. Only what is in the particular Virtual Machine you just created (which actually has no operating system currently).

Follow the steps, creating a username, selecting your time zone and location, etc.

Step 8. Install Guest Additions

You should be up and running with Ubuntu on a virtual machine now! Well done.

You will see a Ubuntu virtual machine appear in the Virtual Box.

Now, click on the orange Settings icon in the Virtual Box toolbar. Choose: Storage option >> IDE >> (+) symbol in a green circle, which signifies “Add new optical drive”. You will be prompted to choose a disk. Navigate to where your VBGuestAdditions .iso file was downloaded, and select it. Click “Open”.

Next, choose “Ok” to close the settings window.

Step 9. Open the Guest Additions in Ubuntu

Boot Ubuntu for the first time. You won’t be able to open it in full screen – that is because you have not yet installed the Guest Additions. But you will do that now.

In the launch bar, click on the CD icon on the bottom left. Check to ensure the files for VirtualBox Guest Additions are there. Right click on an empty space in the corresponding folder and select “Open in terminal”.

Now, copy the following into a terminal using CTRL+SHIFT+V.

sudo sh ./VBoxLinuxAdditions.run

When the script is done, restart the virtual machine by clicking on the little cog () in the top right toolbar and choose “Shutdown”, then “Restart” (Figure 39).

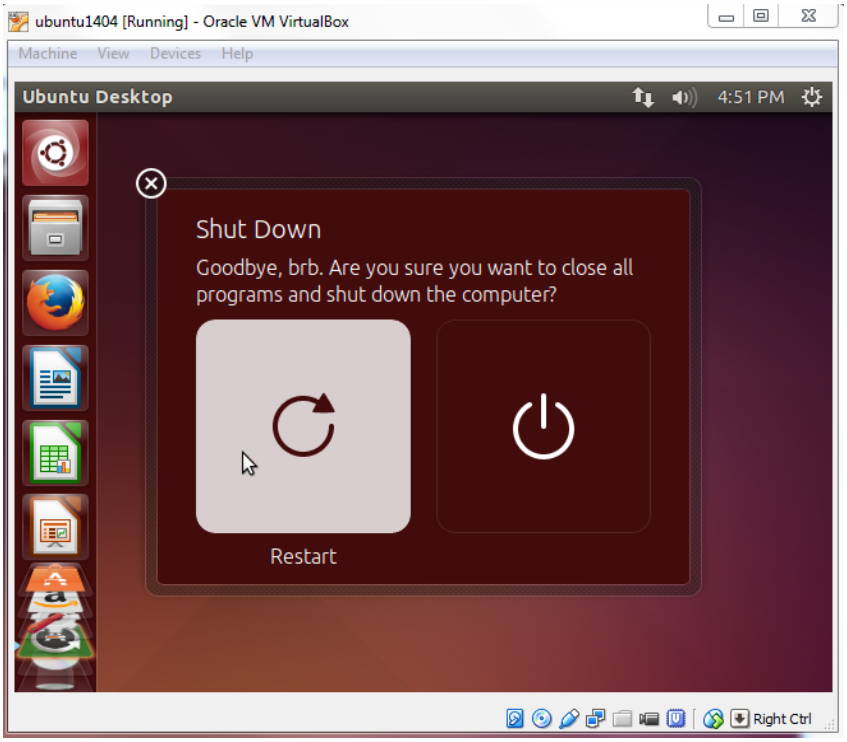


Figure 39 Restart the Ubuntu machine before installing the Guest Additions

When the VM restarts, click on the "View" menu and select "Full Screen Mode". You will be told that you can toggle between full screen and minimized by holding CTRL+F. Click "Switch" to continue.

Now, you are ready to go!

B.2.1 Putting ROS and TurtleBot packages on your VM

Using the script to install ROS and required packages

To simplify the installation process a script has been developed that, when run in terminal using the bash command, installs not only the required distribution of ROS but also downloads and installs all of the required supporting packages and sets some important environment variables. To run the script follow these steps:

- Access the blackboard content for this course and download the "turtlebot_script.sh"
- Open an instance of terminal and navigate to the directory where you have downloaded the script
- **Ensure that you are connected to the internet by either Wifi or Ethernet**
- Enter the following line into the terminal *bash turtlebot_script.sh*
    - Note the script is located at: Tutorials/Ubuntu Setup/turtlebot_script.sh on Quercus.

This bash script will now run in within the terminal and set up your system with the necessary requirements. It will run for approximately 20 mins depending on your system and internet connection.

C. Appendix III: Additional Resources

C.1 Additional ROS tutorials

- ROS Tutorial Video Demos at ANU
- NooTriX Step-by-Step ROS Tutorials
- Jonathan Bohren's ROS Tutorials
- Clearpath Robotics' knowledge base
- Erle Robotics - Learning ROS
- ROS-Industrial Training Class Curriculum

C.2 Library-specific ROS tutorials:

- Robot Model
- Visualization
- actionlib
- pluginlib
- nodelets
- navigation
- ROS-Industrial Tutorials

C.3 TurtleBot Resources

Additional TurtleBot tutorials are provided by the following universities and research groups.

- Learn TurtleBot and ROS
- Gaitech Getting Started Guide

C.4 C++ Resources

- http://www.cplusplus.com/doc/tutorial/
- http://www.learncpp.com/

D. Appendix IV: References

[1] Microsoft, “Kinect Sensor Components and Specifications.” [Online]. Available: https://msdn.microsoft.com/en-us/library/jj131033.aspx.

[2] University of Surrey, “UNIX / Linux Tutorial for Beginners.” [Online]. Available: http://www.ee.surrey.ac.uk/Teaching/Unix/.

[3] ROS, “ROS Concepts - ROS Wiki.” [Online]. Available: http://wiki.ros.org/ROS/Concepts.

[4] Robot Operating System, “ROS/Tutorials - ROS Wiki.” [Online]. Available: http://wiki.ros.org/ROS/Tutorials.

[5] Gazebo, “Gazebo Tutorial : Beginner: GUI.” [Online]. Available: http://gazebosim.org/tutorials?cat=guided_b&tut=guided_b2.

[6] OpenCV, “The Core Functionality — OpenCV 2.4.13.1 documentation.” [Online]. Available: http://docs.opencv.org/2.4/doc/tutorials/core/table_of_content_core/table_of_content_core.html.

[7] ROS, “TurtleBot - ROS Wiki.” [Online]. Available: http://wiki.ros.org/Robots/TurtleBot.

[8] Oracle, “Oracle VM VirtualBox.” [Online]. Available: https://www.virtualbox.org/wiki/Downloads.

[9] R. McKillop, “How to Install Ubuntu on Your Mac Using VirtualBox - Simple Help.” [Online]. Available: http://www.simplehelp.net/2015/06/09/how-to-install-ubuntu-on-your-mac/.

[10] G. Newell, “How To Install Ubuntu Linux On Windows 10.” [Online]. Available: https://www.lifewire.com/install-ubuntu-linux-windows-10-steps-2202108.