**Your Virtual Machine**

A screenshot of a computer

Description automatically generated

To follow along with the exercises in these lessons, you will need to be running Ubuntu Linux with ROS installed. So that everyone can follow along in the same environment we are providing you with a virtual machine (VM) that is all setup with everything you need. The only problem is that it's a rather large download (4+GB!). So, find yourself a good internet connection and download the [**compressed VM disk image**](https://s3-us-west-1.amazonaws.com/udacity-selfdrivingcar/Udacity_VM_Base_V1.0.0.zip).

It is good idea to keep a backup of this VM image to avoid downloading again if you need it. But just in case you do not do that there is a backup snapshot built into the VM that will allow you to restore it to its original state at anytime!

To extract the compressed image in Windows:

1. Download the appropriate version of 7-zip from [**here**](http://www.7-zip.org/download.html)
2. Right-click and select "7-zip" > "Extract Here"

To extract at the command line in macOS or Linux:

1. Open Terminal and navigate to directory containing the compressed image
2. execute $ unzip Udacity\_VM\_Base\_V1.0.0.zip

In the zipped archive in addition to the VM disk image, there are two other small configuration files that you may or may not need.

We recommend using [**VirtualBox**](https://www.virtualbox.org/wiki/Downloads) to run the VM. This will enable you to easily use port forwarding, so you can run the final project simulator in your native operating system (the host) while using the VM for running ROS code.

Whichever operating system you're on, download and install the appropriate platform package for your operating system (the host) and then follow the instructions below:

**Import your VM image to VirtualBox**

1. Download and install VirtualBox.
2. Download the image from supplied link.
3. Unzip the image.
4. Open VirtualBox Application.
5. Click File > Import Appliance..
6. Click the folder icon on the right and navigate to your unzipped image (the **.ovf** file).
7. Follow the prompts to import the image.
8. From the VirtualBox Manager, select your VM and press start.

Before getting your VM up and running, you will need to make sure that you allocate enough resources to allow it to operate efficiently. With your VM shut down, navigate to the VM settings by clicking on the icon in the VirtualBox Manager. In the "System" cateogory Look for tabs labeled "Motherboard" and "Processor"; this is where you will change the amount of RAM and number of cores that you allocate from your own machine to the VM. If you have the resources on your machine, we recommend allocating at least 2 processor cores and 4 GB of RAM. If you are able to allocate more resources, feel free to do so. The more you allocate the better performance you will get! You're now all set to boot up your VM!

Upon your first boot you will be prompted to choose a keyboard layout of your choice. Once you select a keyboard you will not be prompted for this again. If you would like to change your option you can reset this feature by entering udacity\_reset in a terminal of your choice and restarting the VM.

Once you are up and running, you might be asked to input a password to enter the VM. **The password for the VM is udacity-nd**

To open a terminal in your VM press ctrl-Alt-t (or ctrl-option-t on a Mac). You should get a terminal window that looks like this:

Text

Description automatically generated

You're now ready to follow along in your VM with the the ROS essentials examples!

**Troubleshooting Tips**

* **Keyboard Mappings:** Use of certain keyboards can create issues unless the corresponding keyboard has been set in the VM. This is due to keyboard mappings. A frequent issue is special characters in passwords not being entered correctly when logging in. An example useage for VirtualBox is setting up an Italian keyboard. To do this, execute the following in a terminal localectl set-keymap it; localectl set-x11-keymap it.
* **roscore ip:** If the host network interface has multiple addresses (ex: ipv6 enabled) roscore will fail since hostname -I returns multiple ip, resulting into a invalid URL. One solution to this is to replace this line in .bashrc, export ROS\_IP=`echo $( hostname -I)' , with this export ROS\_IP=$( hostname -I | awk '{print $1}').

**Performing a Native Install of Linux and ROS (Unsupported)**

Some of you may choose not to use the virtual machine for a variety of reasons. If you choose to install ROS (Robot Operating System) on your own, it is recommended that you use [**Ubuntu 16.04 LTS**](http://releases.ubuntu.com/16.04/) as your operating system. To install ROS, please visit: [**ROS documentation**](http://wiki.ros.org/)

This method is **not** supported by Udacity. If you have trouble performing a native install of ROS, please visit [**ROS answers**](http://answers.ros.org/questions/) or you can try troubleshooting your install with your fellow students in the Student Hub, or post to [**Knowledge**](https://knowledge.udacity.com/).

# More on the history of Turtles in Robotics

As we mentioned in the video, William Grey Walter's influence is still felt today. He referred to his robots as 'turtles' and, as you will see, the moniker stuck.

The image below is William Walter's Elsie (the robot mentioned in the video) without her protective covering.

A picture containing projector

Description automatically generated

Elsie without her protective cover.

Long after William Walter’s work with Elmer and Elsie, Dr. Seymour Papert, a professor at MIT began to use turtle robots for education. One of the characteristics of Papert's robots was their ability to draw on paper.

In addition to being involved with the creation and development of MIT’s turtle robots, Dr. Papert is also known as the creator and evangelist for the educational programming language LOGO.

Despite being a general-purpose language, LOGO is known for its use of “[**turtle graphics**](https://en.wikipedia.org/wiki/Turtle_graphics)”, a system which allows users to draw by sending simple commands to a robotic turtle. The robotic turtle mentioned here could be either a real turtle robot, or a virtual on-screen cursor within the LOGO programming environment.

The image below shows an example of Valiant Technology’s Turtle robot drawing on a sheet of paper.

A picture containing businesscard, envelope

Description automatically generated

Robot using turtle graphics.

While turtle graphics is a relatively simple concept, people have been able to use it to create a wide variety of interesting drawings and art.

The image below depicts some drawings of three dimensional spheres creating using turtle graphics. The website [**turtleart.org**](http://turtleart.org/) hosts a gallery containing some even more outstanding examples!

Diagram

Description automatically generated

Three dimensional Turtle Graphic

If you'd like to have some fun with turtle graphics, feel free to try out the [**Turtle Graphics 1.01**](https://scratch.mit.edu/projects/1250518/) project, which allows you to experiment with turtle graphics through MIT’s graphical programming environment scratch.

Below is the result of a program I wrote using Turtle Graphics. Looking at this image, answer the quiz question about it.

Diagram

Description automatically generated

A screenshot of Turtle Graphics 1.01.  
What commands were necessary to create the line shown?

This tradition of turtles in robotics is alive and well today. In fact, each recent version of ROS has been named after some sort of turtle. In addition to this turtle-centric naming convention, the OSRF also adds a new turtle to turtle\_sim with each release.

Graphical user interface, application

Description automatically generated

# Environment Setup

Before we begin using ROS in a terminal, we must first ensure that all of the environment variables are present. To do this, we must source the setup script provided by ROS:

Text

Description automatically generated with medium confidence

Caveat:

Make sure you use the bash command source rather than ./. There’s a subtle distinction between the two commands, in that source executes the script in the current session, while ./ will start a new session, containing a copy of the current environment. When a script executed via ./ is exited, all environment variables set by it will be lost. We don’t want this. For more information on environment variables and terminal sessions, please see [**here**](https://help.ubuntu.com/community/EnvironmentVariables).

### Inspecting the Environment

You can inspect the changes that it has made to our environment variables by using the Unix command env.

$ env

Depending on your system configuration, there may be quite a few environment variables. One quick way to determine what changes were made to the existing environment is to run the following commands in a new terminal (where /opt/ros/kinetic/setup.bash has not already been sourced).

Text

Description automatically generated

The Unix diff command compares two files line by line, indicating any differences between them. The lines beginning with a ‘+’ indicate lines that were either added, or changed in newenv.txt. In our case, we can see that the following changes have been made to the environment by setup.bash

ROS\_ROOT=/opt/ros/kinetic/share/ros

ROS\_PACKAGE\_PATH=/opt/ros/kinetic/share

ROS\_MASTER\_URI=http:*//localhost:11311*

LD\_LIBRARY\_PATH=/opt/ros/kinetic/lib

PATH=/opt/ros/kinetic/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games

ROSLISP\_PACKAGE\_DIRECTORIES=

ROS\_DISTRO=kinetic

PYTHONPATH=/opt/ros/kinetic/lib/python2.7/dist-packages

PKG\_CONFIG\_PATH=/opt/ros/kinetic/lib/pkgconfig

CMAKE\_PREFIX\_PATH=/opt/ros/kinetic

ROS\_ETC\_DIR=/opt/ros/kinetic/etc/ros

While we won’t be discussing all of the environment variables that have been added or changed, there are a few worth mentioning.

ROS\_ROOT=/opt/ros/kinetic/share/ros The path where core ros packages are stored

PATH=/opt/ros/kinetic/bin:... The path to the ROS binaries, which we will be using throughout this lesson.

ROS\_DISTRO=kinetic Which distribution of ROS is being used. In this case, we are using kinetic

PYTHONPATH=/opt/ros/kinetic/lib/python2.7/dist-packages The path to the ROS python packages, which we will be using next lesson.

More information about the environment variables mentioned above, the ones not mentioned here, and others can be found [**here**](http://wiki.ros.org/ROS/EnvironmentVariables).

## Automatically Configuring the Environment

Setting up the ROS environment every time you open a new terminal window can be painful and tedious. To avoid the tedium, we can simply add the command to source the workspace to our ~/.bashrc file. This will cause our environment to be sourced any time a new terminal session is created.

Please note that the provided VM already has this configured so there is no need to execute this step!

Text

Description automatically generated

Now that you’ve added ROS’ environment variables to your terminal session, you can run the turtlesim package!

**Starting the Master process**

Before any ROS nodes can be run, the Master process must be started.

The Master process is responsible for the following (and more)

* Providing naming and registration services to other running nodes
* Tracking all publishers and subscribers
* Aggregating log messages generated by the nodes
* Facilitating connections between nodes

To run the master process, you must execute the command roscore. If all goes well (and it should), you will see an output similar to following:

Text

Description automatically generated

This indicates that ROS Master is running. To terminate the ROS Master process, simply send the SIGINT signal to the process by typing ctrl-c in the roscore terminal window.

We are now free to begin launching nodes!

**Running Turtlesim Nodes**

Now that the ROS master is running, we can run our first two ROS nodes.

To do so, we will execute the rosrun command in a new terminal window, passing as parameters the name of the package we wish to run, and the name of the actual node.

Note:

Tab completion is your friend. Each ROS distribution comes with a staggering number of packages, and an even more staggering number of nodes. In the bash shell, a single-tap of the tab key will cause the command on the command-line to be completed, if there is a single match. A double-tap of the tab key will result in a list of all possible matches, in the case that a single match cannot be found.

First we will start the turtlesim\_node, in the turtlesim package using the following command.

$ rosrun turtlesim turtlesim\_node

Next, we will start the turtle\_teleop\_key node, also from the turtlesim package.

$ rosrun turtlesim turtle\_teleop\_key

By using the arrow keys with the turtle\_teleop\_key node’s console selected, we are able to move the turtle in turtlesim!

Graphical user interface, text

Description automatically generated

**Turtlesim Comms: List All Active Nodes**

In the following concepts, we will investigate Turtlesimm Comms, and will cover the following steps:

* Listing all active nodes
* Listing all topics
* Getting information about topics
* Showing message information
* Echoing messages in real-time

Now that we’ve launched turtlesim\_node, and played around with sending commands via the turtle\_teleop\_key node, let’s dig deeper, to see what’s actually happening underneath the surface...

**Listing all Active Nodes**

To get a list of all nodes that are active and have been registered with the ROS Master, we can use the command rosnode list. Let’s do so now:

Text

Description automatically generated

Listing active nodes

We can see that there are three active nodes that have been registered with the ROS Master, /rosout, /teleop\_turtle, and /turtlesim.

* /rosout This node is launched by roscore. It subscribes to the standard /rosout topic, the topic to which all nodes send log messages.
* /teleop\_turtle This is our keyboard teleop node. Notice that it’s not named turtle\_teleop\_key. There’s no requirement that a node’s broadcasted name is the same as the name of it’s associated executable.
* /turtlesim The node name associated with the turtlebot\_sim node

**Listing All Topics**

In a similar fashion, we are able to query the ROS Master for a list of all topics. To do so, we use the command rostopic list.

Text

Description automatically generated

* /rosout\_agg Aggregated feed of messages published to /rosout.
* /turtle1/cmd\_vel Topic upon which velocity commands are sent/received. Publishing a velocity message to this topic will command turtle1 to move.
* /turtle1/color\_sensor Each turtle in turtlesim is equipped with a color sensor, and readings from the sensor are published to this topic.
* /turtle1/pose The position and orientation of turtle1 are published to this topic.

## Get Information About a Specific Topic

If we wish to get information about a specific topic, who is publishing to it, subscribed to it, or the type of message associated with it, we can use the command $rostopic info. Let’s check into the /turtle1/cmd\_vel topic:

Text

Description automatically generated

As would be expected, there are two nodes registered on this topic. Our publisher, the teleop\_turtle node, and our subscriber, the turtlesim node. Additionally, we can see that the type of message used on this topic is geometry\_msgs/Twist.

# Turtlesim Comms: Show Message Information

Let’s get some more information about the geometry\_msgs/Twist message on the /turtle1/cmd\_vel topic, to do so, we will use the rosmsg info command.

Text

Description automatically generated

We can see that a Twist message consists nothing more than two Vector3 messages. One for linear velocity, and another for angular velocities, with each velocity component being represented by a float64.

**Note**: Sometimes, the message definition won’t provide an ample amount of detail about a message type. For example, in the example above, how can we be sure that linear and angular vectors above refer to velocities, and not positions? One way to get more detail would be to look at the comments in the message’s definition file. To do so, we can issue the following command: rosed geometry\_msgs Twist.msg.

**Note 2:** More information about rosed, including how to select which editor is used by default can be found [**here**](http://wiki.ros.org/ROS/Tutorials/UsingRosEd).

# Turtlesim Comms: Echo Messages on a Topic

Sometimes it may be useful to look at a topic’s published messages in real time. To do so, we can use the command rostopic echo. Let’s take a look at the /turtle1/cmd\_vel topic.

$ rostopic echo /turtle1/cmd\_vel

If we then command the turtle to move from the turtle\_teleop\_key window, we will be able to see the output message in real-time.

Text

Description automatically generated