Lab Number 10
Introduction to ROS

Introduction:

- ROS stands for "Robot Operating System", but it is not actually an operating system. ROS is actually a set of software libraries and tools made to ease the development of robotic applications. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management.

Ubuntu and ROS Version:

Ubuntu: 16.04
ROS : Kinetic

Book Reference:

A Gentle Introduction to ROS by Jason M. O'Kane

Objectives of the Lab:

- To learn ROS environment

WHY ROS:

- ROS is widely used in robotics companies, universities and robotics research institutes for designing, building, and simulating a robot model and interfacing it into real hardware. We almost have packages ready for every kind of Robots (manipulators, mobile robotics, quadrotor, etc). So we need not write every thing from scratch.

1. ROS is a **open source software**. ROS is free and you can use it for commercial purposes.
2. All the **Transforms** of the robot are tracked in ROS.
3. There are **IK slovers** to solve your Ik problems.
4. Struck with path planning and obstacle avoidance don't worry **Moveit** will take care.

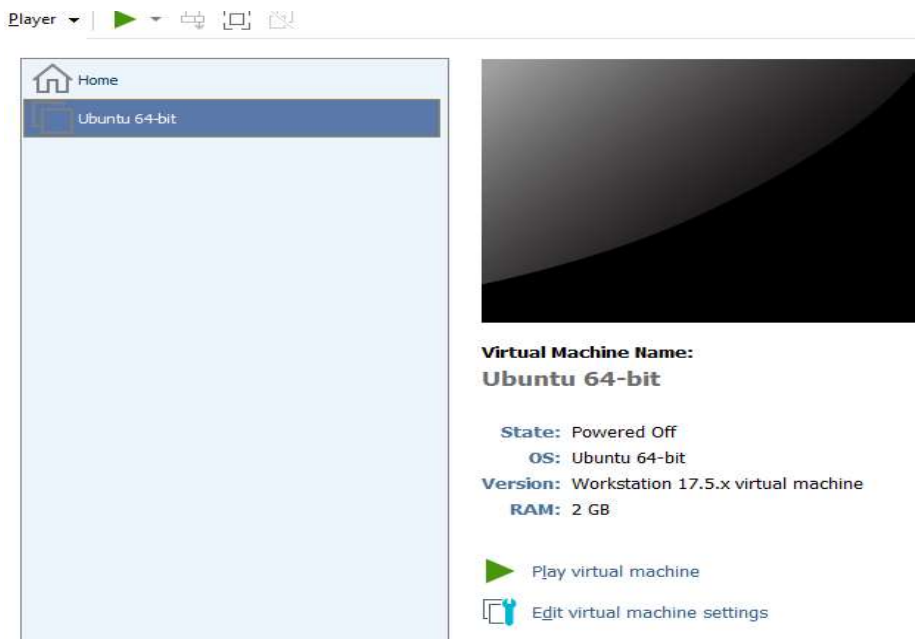
INSTALLATION OF ROS:

- ROS works with Linux operating system, so you need to have a PC with Linux operating system. You can use virtual machines (like Vmware or virtualbox) to run Linux operating system in you Windows PC.
- https://www.vmware.com/products/workstation-player/workstation-player-evaluation.html

INSTALLATION OF UBUNTU:

- https://releases.ubuntu.com/16.04/

After installing open ubuntu in VMWARE and start. But make sure that you have downloaded an image file of ubuntu. Image file should be open in your PC to turn ubuntu ON.

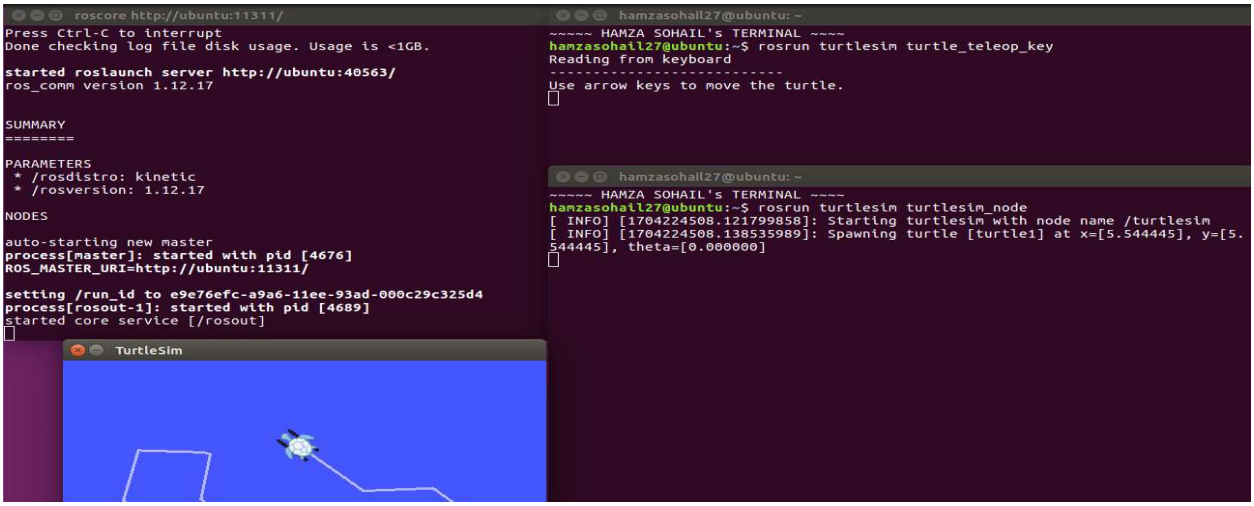


Open Ubuntu and then terminal.

AFTER INSTALLING ROS:

- **http://wiki.ros.org/kinetic/Installation/Ubuntu**
- Open three terminals and write the following commands in these separate terminals:

- roscore
- rosrun turtlesim turtlesim_node
- rosrun turtlesim turtle_teleop_key

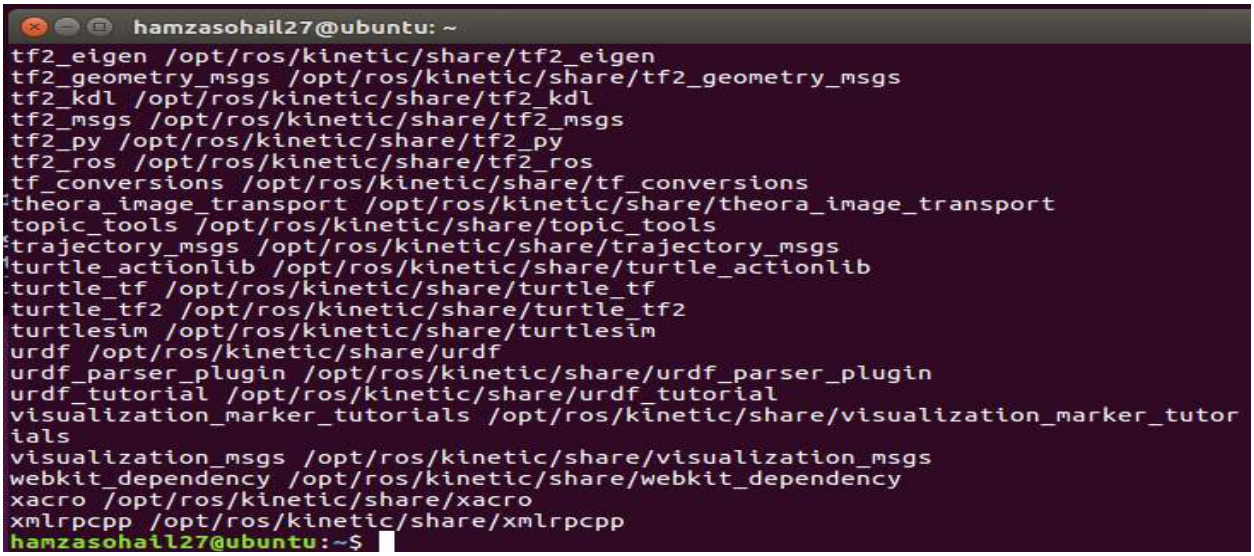


The separate terminals are intended to allow all three commands to execute simultaneously. If everything works correctly, you should see a graphical window. This window shows a simulated, turtle-shaped robot that lives in a square world. (The appearance of your turtle may differ. The simulator selects from a collection of "mascot" turtles for each of the historical distributions of ROS.) If you give your third terminal (the one executing the turtle_teleop_key command) the input focus and press the Up, Down, Left, or Right keys, the turtle will move in response to your commands, leaving a trail behind it.

PACKAGES:

- All ROS software is organized into packages. A ROS package is a coherent collection of files,
- generally including both executables and supporting files, that serves a specific purpose.
- In the example, we used two executables called turtlesim_node and turtle_teleop_key,
- both of which are members of the turtlesim package

LISTING AND LOCATION PACKAGES:
rospack list



FINDING A PACKAGE:
In general: rospack find package-name
OR
rospack find turtlesim
Then
rosls turtlesim
To check its directory.
roscd turtlesim
Now you are in folder of turtlesim directory.

ROS MASTER:
One of the basic goals of ROS is to enable roboticists to design software as a collection of small, mostly independent programs called nodes that all run at the same time. For this to work, those nodes must be able to communicate with one another. The part of ROS that facilitates this communication is called the ROS master. To start the master, use this command:
roscore
Ctrl + C to terminate any ingoing program

ROS NODES:

Once you've started roscore, you can run programs that use ROS. A running instance of a ROS program is called a node.
In the turtlesim example, we created two nodes. One node is an instance of an executable called **turtlesim_node.** This node is responsible for creating the turtlesim window and simulating the motion of the turtle. The second node is an instance of an executable called **turtle_teleop_key**. The abbreviation teleop is a shortened form of the word teleoperation, which refers to situations in a human controls a robot remotely by giving direct movement commands. This node waits for an arrow key to be pressed, converts that key press to a movement command, and sends that

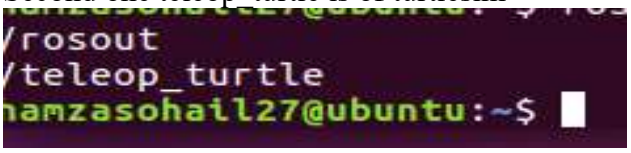
command to the turtlesim_node node.

RUNNING A NODE:

- rosrun package-name executable-name
- rosrun turtlesim turtlesim_node

Additional:

- ROS provides a few ways to get information about the nodes that are running at any particular time. To get a list of running nodes, try this command:
- rosnode list
- Rosout is the roscore node (master node)
- Second one teleop_turtle is of turtlesim



PARTICULAR NODE INFO:

- rosnode info node-name
- rosnode info /rosout

KILLING A NODE:

- rosnode kill node-name
- rosnode kill /rosout

RQT GRAPH:

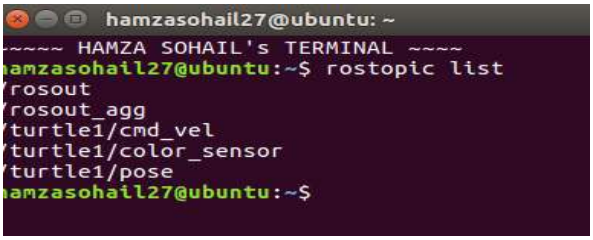
- rqt_graph

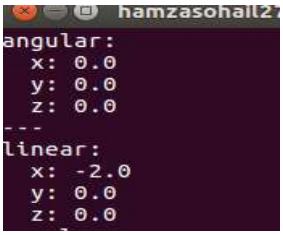
MESSAGING AND MESSAGE TYPES:

Listing topics To get a list of active topics, use this command:
rostopic list
In our example, this shows a list of five topics:
/rosout
/rosout_agg
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose

SELECTING ROS TOPIC:

rostopic echo topic-name
OR
rostopic echo /turtle1/cmd_vel
As long as I am moving turtle it is showing its movements





INSPECTING MESSAGE TYPE

- rosmsg show message-type-name
- rosmsg show turtlesim/Color
- The output is:
- uint8 r
- uint8 g
- uint8 b

PUBLISHING MESSAGES:

- **rostopic pub -r rate-in-hz topic-name message-type message-content**
- **rostopic pub -r 1 /turtle1/cmd_vel geometry_msgs/Twist '[2, 0, 0]' '[0, 0, 0]'**

Turtle will continously move in that direction

- rostopic pub -r 1 /turtle1/cmd_vel geometry_msgs/Twist '[0, 0, 0]' '[0, 0, 1]'

From this turtle will continously rotate

EXAMPLE:
First, stop any nodes that might be currently running. Start roscore if it's not already active. Then, in four separate terminals, run these four commands:

- **rosrun turtlesim turtlesim_node __name:=A**
- **rosrun turtlesim turtlesim_node __name:=B**
- **rosrun turtlesim turtle_teleop_key __name:=C**
- **rosrun turtlesim turtle_teleop_key __name:=D**